Flutter Curriculum



| [Watch our launch video](http://www.google.com) |
| --- |

# 

# 

## 

## **Table of Content**

[Table of Content](#_t7yblt7az47p) 2

[**Introduction to Programming with Dart**](#_ns0obfhdcyx0) **4**

[Variables and Data types](#_5nrkon90yl06) 4

[Decision Making and Loops](#_inzb692kj280) 5

[Functions](#_1mvw3psbj5j6) 6

[Object Oriented Programming](#_qgjh9lwzo7pt) 6

[**Flutter Development**](#_7npop86ydfdl) **8**

[Widget Build Visualization](#_xyr3fnm6lo6) 9

[Platform specific widgets](#_wmgobav4ugmv) 9

[Layout widgets](#_hxr5srbyfquj) 11

[State maintenance widgets](#_aeh27iuqjv8h) 11

[Platform independent / basic widgets](#_75luyticcxkt) 12

[Flutter Layouts](#_ju7xyessdnsu) 16

[Type of Layout Widgets](#_3gafbpmzrfrp) 16

[Single Child Widgets](#_bz9yk9g2xc2f) 17

[Multiple Child Widgets](#_ovu7qz46ztnb) 21

[Advanced Layout Application](#_x7cy9rix7073) 22

[**User Interface Development**](#_x8449fin1lf8) **32**

[Installing the required dependencies and adding Assets to Flutter](#_n299iny8gklt) 33

[Adding Dependencies](#_dxmgcywpwfek) 33

[Adding Asset to your project](#_ikj3un4zpt3a) 35

[The main.dart file](#_8pdqa8szlb52) 36

[The login.dart file](#_ev7i7rcxdzzg) 37

[Custom Widgets](#_a5yoigdc0djz) 40

[TxtField Custom Widget](#_jjfhufch3fpd) 40

[Btn Custom Widget](#_bsz02chq0faq) 42

[The forgot\_pass.dart file](#_kl341je3oas) 43

[The reset\_pass.dart file](#_u9mm4lyj9ipj) 45

[The signup.dart file](#_jdwtgze2v2bh) 47

[The view\_courses.dart file](#_jvlewoun167u) 51

[The profile.dart file](#_g7du3gn34nmr) 56

[The edit\_profile.dart file](#_j35ruhmhquv) 61

[The single\_course.dart file](#_xjq5pcvsh59n) 64

[The add\_edit\_course.dart file](#_s33uotnaxdkq) 67

[**Backend and State Management**](#_7lcmrw9zxxn2) **69**

[Setting up Backend](#_ogvebmxdpi4t) 69

[**EXTRA: Push Notifications**](#_krk1lcbp5n3) **88**

[**EXTRA: APIs with Flutter**](#_e6orxaxmiagc) **89**

[**EXTRA: Monetization with Pesapal**](#_l0phiiak5vn) **90**

[**What Next?**](#_oxrfua7zrlv1) **91**

# 

# Introduction to Programming with Dart

Dart is an open-source general-purpose programming language. It was originally developed by Google. Dart is an object-oriented language with C-style syntax. It supports programming concepts like interfaces, classes, unlike other programming languages Dart doesn’t support arrays. Dart collections can be used to replicate data structures such as arrays, generics, and optional typing.

The following code shows a simple Dart program −

| void main() {  print("Dart language is easy to learn"); } |
| --- |

### Variables and Data types

Variable is named storage location and Data types simply refers to the type and size of data associated with variables and functions.

Dart uses var keyword to declare the variable. The syntax of var is defined below,

| var name = 'Dart'; |
| --- |

The final and const keyword are used to declare constants. They are defined as below −

| void main() {  final a = 12;  const pi = 3.14;  print(a);  print(pi); } |
| --- |

Dart language supports the following data types −

* Numbers − It is used to represent numeric literals – Integer and Double.
* Strings − It represents a sequence of characters. String values are specified in either single or double quotes.
* Booleans − Dart uses the bool keyword to represent Boolean values – true and false.
* Lists and Maps − It is used to represent a collection of objects. A simple List can be defined as below −.

| void main() {  var list = [1,2,3,4,5];  print(list); } |
| --- |

The list shown above produces [1,2,3,4,5] list.

Map can be defined as shown here −

| void main() {  var mapping = {'id': 1,'name':'Dart'};  print(mapping); } |
| --- |

Dynamic − If the variable type is not defined, then its default type is dynamic. The following example illustrates the dynamic type variable −

| void main() {  dynamic name = "Dart";  print(name); } |
| --- |

### Decision Making and Loops

A decision making block evaluates a condition before the instructions are executed. Dart supports If, If..else and switch statements.

Loops are used to repeat a block of code until a specific condition is met. Dart supports for, for..in , while and do..while loops.

Let us understand a simple example about the usage of control statements and loops

| void main() {  for( var i = 1 ; i <= 10; i++ ) {  if(i%2==0) {  print(i);  }  } } |
| --- |

The above code prints the even numbers from 1 to 10.

### Functions

A function is a group of statements that together performs a specific task. Let us look into a simple function in Dart as shown here −

| void main() {  add(3,4); } void add(int a,int b) {  int c;  c = a+b;  print(c); } |
| --- |

The above function adds two values and produces 7 as the output.

### Object Oriented Programming

Dart is an object-oriented language. It supports object-oriented programming features like classes, interfaces, etc.

A class is a blueprint for creating objects. A class definition includes the following −

* Fields
* Getters and setters
* Constructors
* Functions

Now, let us create a simple class using the above definitions −

| class Employee {  String name;    //getter method  String get emp\_name {  return name;  }  //setter method  void set emp\_name(String name) {  this.name = name;  }  //function definition  void result() {  print(name);  } } void main() {  //object creation  Employee emp = new Employee();  emp.name = "employee1";  emp.result(); //function call } |
| --- |

# Flutter Development

Let us check the Hello World application’s *MyHomePage* widget. The code for this purpose is as given below −

| class MyHomePage extends StatelessWidget {   MyHomePage({Key key, this.title}) : super(key: key);     final String title;   @override   Widget build(BuildContext context) {  return Scaffold(   appBar: AppBar(title: Text(this.title), ),   body: Center(child: Text( 'Hello World',)),  );  } } |
| --- |

Here, we have created a new widget by extending *StatelessWidget*.

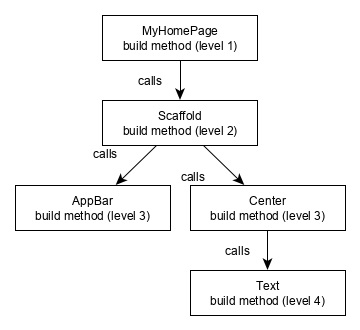
Note that the *StatelessWidget* only requires a single method build to be implemented in its derived class. The build method gets the context environment necessary to build the widgets through the *BuildContext* parameter and returns the widget it builds.

In the code, we have used title as one of the constructor arguments and also used Key as another argument. The title is used to display the title and Key is used to identify the widget in the build environment.

Here, the build method calls the build method of Scaffold, which in turn calls the build method of AppBar and Center to build its user interface.

Finally, *Center* build method *calls* Text build method.

For a better understanding, the visual representation of the same is given below −



## Widget Build Visualization

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

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

Let us discuss each of them in detail now.

### Platform specific widgets

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

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

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

Some of the most used material widgets are as follows −

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

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

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

### Layout widgets

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

Some of the popular layout widgets are as follows −

* Container − A rectangular box decorated using *BoxDecoration* widgets with background, border and shadow.
* Center − Center its child widget.
* Row − Arrange its children in the horizontal direction.
* Column − Arrange its children in the vertical direction.
* Stack − Arrange one above the other.

We will check the layout widgets in detail in the upcoming *Introduction to layout widgets* chapter.

### State maintenance widgets

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

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

We will learn the concept of *StatefulWidget* widgets in detail in the upcoming *State management chapter*.

### Platform independent / basic widgets

*Flutter* provides a large number of basic widgets to create simple as well as complex user interfaces in a platform independent manner. Let us see some of the basic widgets in this chapter.

Text

*Text* widget is used to display a piece of string. The style of the string can be set by using *style* property and *TextStyle* class. The sample code for this purpose is as follows −

| Text('Hello World!', style: TextStyle(fontWeight: FontWeight.bold)) |
| --- |

*Text* widget has a special constructor, *Text.rich*, which accepts the child of type *TextSpan* to specify the string with different style. *TextSpan* widget is recursive in nature and it accepts *TextSpan* as its children. The sample code for this purpose is as follows −

| Text.rich(   TextSpan(   children: <TextSpan>[   TextSpan(text: "Hello ", style:   TextStyle(fontStyle: FontStyle.italic)),   TextSpan(text: "World", style:   TextStyle(fontWeight: FontWeight.bold)),   ],   ),  ) |
| --- |

The most important properties of the *Text* widget are as follows −

* maxLines, int − Maximum number of lines to show
* overflow, TextOverFlow − Specify how visual overflow is handled using *TextOverFlow* class
* style, TextStyle − Specify the style of the string using *TextStyle* class
* textAlign, TextAlign − Alignment of the text like right, left, justify, etc., using *TextAlign* class
* textDirection, TextDirection − Direction of text to flow, either left-to-right or right-to-left

Image

*Image* widget is used to display an image in the application. *Image* widget provides different constructors to load images from multiple sources and they are as follows −

* Image − Generic image loader using *ImageProvider*
* Image.asset − Load image from flutter project’s assets
* Image.file − Load image from system folder
* Image.memory − Load image from memory
* Image.Network − Load image from network

The easiest option to load and display an image in *Flutter* is by including the image as assets of the application and loading it into the widget on demand.

* Create a folder, assets in the project folder and place the necessary images.
* Specify the assets in the pubspec.yaml as shown below −

| flutter:   assets:   - assets/smiley.png |
| --- |

* Now, load and display the image in the application.

| Image.asset('assets/smiley.png') |
| --- |

* The complete source code of *MyHomePage* widget of the hello world application and the result is as shown below −.

| class MyHomePage extends StatelessWidget {  MyHomePage({Key key, this.title}) : super(key: key);   final String title;    @override   Widget build(BuildContext context) {  return Scaffold(   appBar: AppBar( title: Text(this.title), ),   body: Center( child: Image.asset("assets/smiley.png")),  );   } } |
| --- |

The loaded image is as shown below −



The most important properties of the *Image* widget are as follows −

* image, ImageProvider − Actual image to load
* width, double − Width of the image
* height, double − Height of the image
* alignment, AlignmentGeometry − How to align the image within its bounds

Icon

*Icon* widget is used to display a glyph from a font described in *IconData* class. The code to load a simple email icon is as follows −

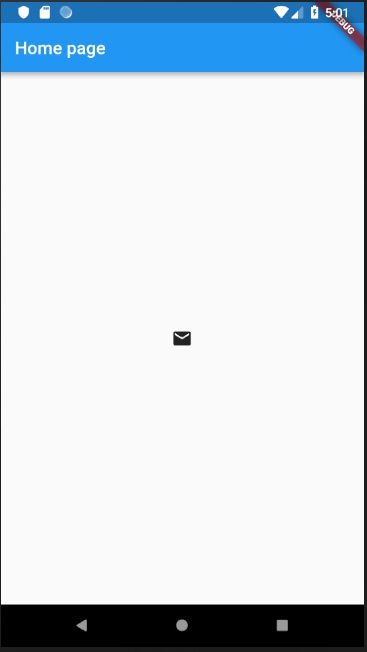
| Icon(Icons.email) |
| --- |

The complete source code to apply it in hello world application is as follows −

| class MyHomePage extends StatelessWidget {   MyHomePage({Key key, this.title}) : super(key: key);   final String title;    @override   Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(title: Text(this.title),),  body: Center( child: Icon(Icons.email)),  );  } |
| --- |

}

The loaded icon is as shown below −



## Flutter Layouts

Since the core concept of *Flutter* is *Everything* is *widget*, *Flutter* incorporates a user interface layout functionality into the widgets itself. *Flutter* provides quite a lot of specially designed widgets like *Container, Center, Align*, etc., only for the purpose of laying out the user interface. Widgets build by composing other widgets normally use layout widgets. Let use learn the *Flutter* layout concept in this chapter.

## Type of Layout Widgets

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

* Widget supporting a single child
* Widget supporting multiple child

Let us learn both type of widgets and its functionality in the upcoming sections.

## Single Child Widgets

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

For example, *Center* widget just centers it child widget with respect to its parent widget and *Container* widget provides complete flexibility to place it child at any given place inside it using different option like padding, decoration, etc.,

Single child widgets are great options to create high quality widget having single functionality such as button, label, etc.,

The code to create a simple button using *Container* widget is as follows −

| class MyButton extends StatelessWidget {  MyButton({Key key}) : super(key: key);    @override   Widget build(BuildContext context) {  return Container(  decoration: const BoxDecoration(  border: Border(  top: BorderSide(width: 1.0, color: Color(0xFFFFFFFFFF)),  left: BorderSide(width: 1.0, color: Color(0xFFFFFFFFFF)),  right: BorderSide(width: 1.0, color: Color(0xFFFF000000)),  bottom: BorderSide(width: 1.0, color: Color(0xFFFF000000)),  ),  ),  child: Container(  padding: const  EdgeInsets.symmetric(horizontal: 20.0, vertical: 2.0),  decoration: const BoxDecoration(  border: Border(  top: BorderSide(width: 1.0, color: Color(0xFFFFDFDFDF)),  left: BorderSide(width: 1.0, color: Color(0xFFFFDFDFDF)),  right: BorderSide(width: 1.0, color: Color(0xFFFF7F7F7F)),  bottom: BorderSide(width: 1.0, color: Color(0xFFFF7F7F7F)),  ),  color: Colors.grey,  ),  child: const Text(  'OK',textAlign: TextAlign.center, style: TextStyle(color: Colors.black)  ),   ),   );   } } |
| --- |

Here, we have used two widgets – a *Container* widget and a *Text* widget. The result of the widget is as a custom button as shown below −

OK

Let us check some of the most important single child layout widgets provided by *Flutter* −

* Padding − Used to arrange its child widget by the given padding. Here, padding can be provided by *EdgeInsets* class.
* Align − Align its child widget within itself using the value of *alignment* property. The value for *alignment* property can be provided by *FractionalOffset* class. The *FractionalOffset* class specifies the offsets in terms of a distance from the top left.

Some of the possible values of offsets are as follows −

* FractionalOffset(1.0, 0.0) represents the top right.
* FractionalOffset(0.0, 1.0) represents the bottom left.

A sample code about offsets is shown below −

| Center(  child: Container(  height: 100.0,   width: 100.0,   color: Colors.yellow, child: Align(  alignment: FractionalOffset(0.2, 0.6),  child: Container( height: 40.0, width:  40.0, color: Colors.red,  ),   ),   ),  ) |
| --- |

* FittedBox − It scales the child widget and then positions it according to the specified fit.
* AspectRatio − It attempts to size the child widget to the specified aspect ratio
* ConstrainedBox
* Baseline
* FractinallySizedBox
* IntrinsicHeight
* IntrinsicWidth
* LimitedBox
* OffStage
* OverflowBox
* SizedBox
* SizedOverflowBox
* Transform
* CustomSingleChildLayout

Our hello world application is using material based layout widgets to design the home page. Let us modify our hello world application to build the home page using basic layout widgets as specified below −

* Container − Generic, single child, box based container widget with alignment, padding, border and margin along with rich styling features.
* Center − Simple, Single child container widget, which centers its child widget.

The modified code of the *MyHomePage* and *MyApp* widget is as below −

| class MyApp extends StatelessWidget {  @override  Widget build(BuildContext context) {  return MyHomePage(title: "Hello World demo app");  } } class MyHomePage extends StatelessWidget {  MyHomePage({Key key, this.title}) : super(key: key);  final String title;  @override  Widget build(BuildContext context) {  return Container(  decoration: BoxDecoration(color: Colors.white,),  padding: EdgeInsets.all(25), child: Center(  child:Text(  'Hello World', style: TextStyle(  color: Colors.black, letterSpacing: 0.5, fontSize: 20,  ),  textDirection: TextDirection.ltr,  ),  )  );  } } |
| --- |

Here,

* *Container* widget is the top level or root widget. *Container* is configured using *decoration* and *padding* properties to layout its content.
* *BoxDecoration* has many properties like color, border, etc., to decorate the *Container* widget and here, *color* is used to set the color of the container.
* *padding* of the *Container* widget is set by using E*dgeInsets* class, which provides the option to specify the padding value.
* *Center* is the child widget of the *Container* widget. Again, *Text* is the child of the *Center* widget. *Text* is used to show message and *Center* is used to center the text message with respect to the parent widget, *Container*.

The final result of the code given above is a layout sample as shown below −



## Multiple Child Widgets

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

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

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

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

## Advanced Layout Application

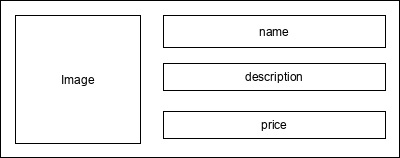
In this section, let us learn how to create a complex user interface of *product listing* with custom design using both single and multiple child layout widgets.

For this purpose, follow the sequence given below −

* Create a new *Flutter* application in Android studio, *product\_layout\_app*.
* Replace the *main.dart* code with following code −

| import 'package:flutter/material.dart';  void main() => runApp(MyApp());   class MyApp extends StatelessWidget {  // This widget is the root of your application.  @override   Widget build(BuildContext context) {  return MaterialApp(   title: 'Flutter Demo', theme: ThemeData(   primarySwatch: Colors.blue,),   home: MyHomePage(title: 'Product layout demo home page'),  );   }  }  class MyHomePage extends StatelessWidget {  MyHomePage({Key key, this.title}) : super(key: key);   final String title;     @override   Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(title: Text(this.title),),   body: Center(child: Text( 'Hello World', )),   );   } } |
| --- |

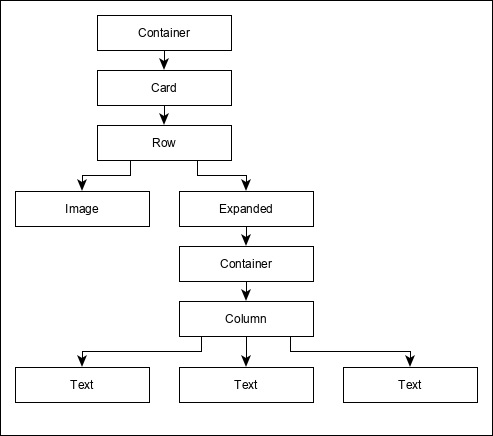
* Here,
* We have created *MyHomePage* widget by extending *StatelessWidget* instead of default *StatefulWidget* and then removed the relevant code.
* Now, create a new widget, *ProductBox* according to the specified design as shown below −



* The code for the *ProductBox* is as follows.

| class ProductBox extends StatelessWidget {  ProductBox({Key key, this.name, this.description, this.price, this.image})   : super(key: key);   final String name;   final String description;   final int price;   final String image;    Widget build(BuildContext context) {  return Container(  padding: EdgeInsets.all(2), height: 120, child: Card(   child: Row(  mainAxisAlignment: MainAxisAlignment.spaceEvenly, children: <Widget>[  Image.asset("assets/appimages/" +image), Expanded(  child: Container(  padding: EdgeInsets.all(5), child: Column(  mainAxisAlignment: MainAxisAlignment.spaceEvenly,   children: <Widget>[     Text(this.name, style: TextStyle(fontWeight:   FontWeight.bold)), Text(this.description),   Text("Price: " + this.price.toString()),   ],   )  )  )  ]  )  )  );  } } |
| --- |

* Please observe the following in the code −
* *ProductBox* has used four arguments as specified below −
  + name - Product name
  + description - Product description
  + price - Price of the product
  + image - Image of the product
* *ProductBox* uses seven build-in widgets as specified below −
  + Container
  + Expanded
  + Row
  + Column
  + Card
  + Text
  + Image
* *ProductBox* is designed using the above mentioned widget. The arrangement or hierarchy of the widget is specified in the diagram shown below −



* Now, place some dummy image (see below) for product information in the assets folder of the application and configure the assets folder in the pubspec.yaml file as shown below −

| assets:   - assets/appimages/floppy.png   - assets/appimages/iphone.png   - assets/appimages/laptop.png   - assets/appimages/pendrive.png   - assets/appimages/pixel.png   - assets/appimages/tablet.png |
| --- |

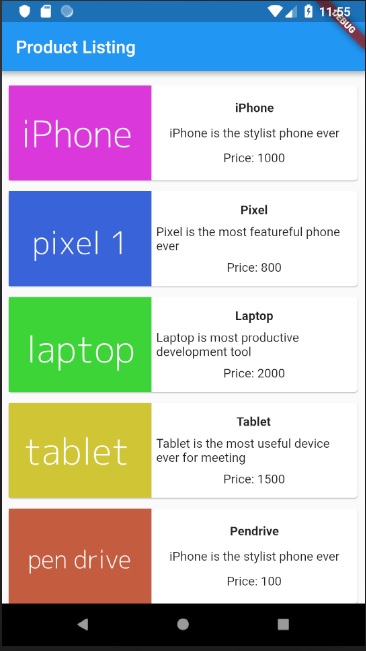
Finally, Use the *ProductBox* widget in the *MyHomePage* widget as specified below −

| class MyHomePage extends StatelessWidget {   MyHomePage({Key key, this.title}) : super(key: key);   final String title;    @override   Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(title:Text("Product Listing")),   body: ListView(  shrinkWrap: true, padding: const EdgeInsets.fromLTRB(2.0, 10.0, 2.0, 10.0),   children: <Widget> [  ProductBox(  name: "iPhone",   description: "iPhone is the stylist phone ever",   price: 1000,   image: "iphone.png"  ),   ProductBox(  name: "Pixel",   description: "Pixel is the most featureful phone ever",   price: 800,   image: "pixel.png"  ),   ProductBox(   name: "Laptop",   description: "Laptop is most productive development tool",   price: 2000,   image: "laptop.png"  ),   ProductBox(   name: "Tablet",   description: "Tablet is the most useful device ever for meeting",   price: 1500,   image: "tablet.png"  ),   ProductBox(  name: "Pendrive",   description: "Pendrive is useful storage medium",   price: 100,   image: "pendrive.png"  ),   ProductBox(  name: "Floppy Drive",   description: "Floppy drive is useful rescue storage medium",   price: 20,   image: "floppy.png"  ),   ],  )  );  } } |
| --- |

* Here,we have used *ProductBox* as children of *ListView* widget.
* The complete code *(main.dart)* of the product layout application *(product\_layout\_app)* is as follows −

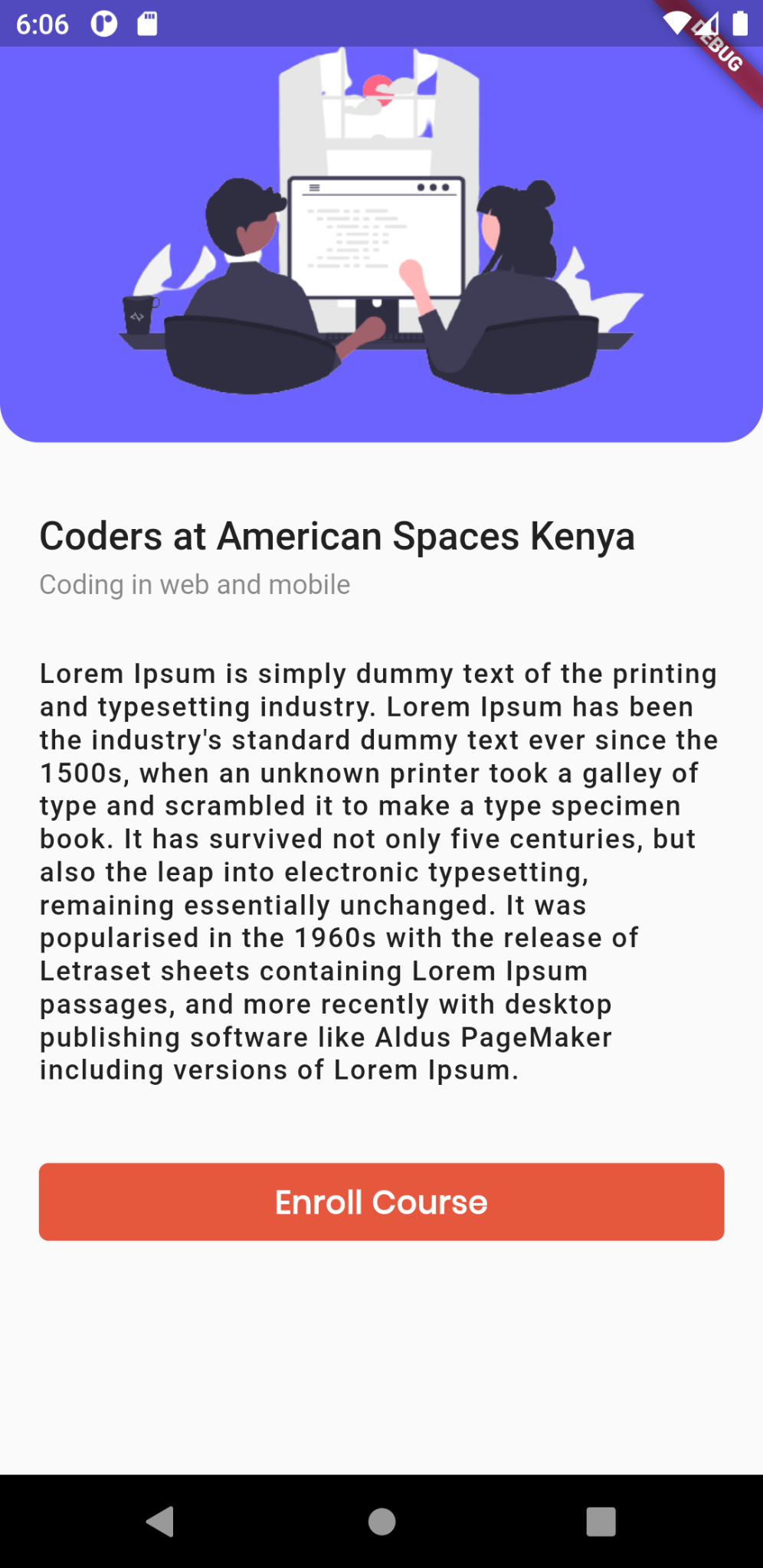
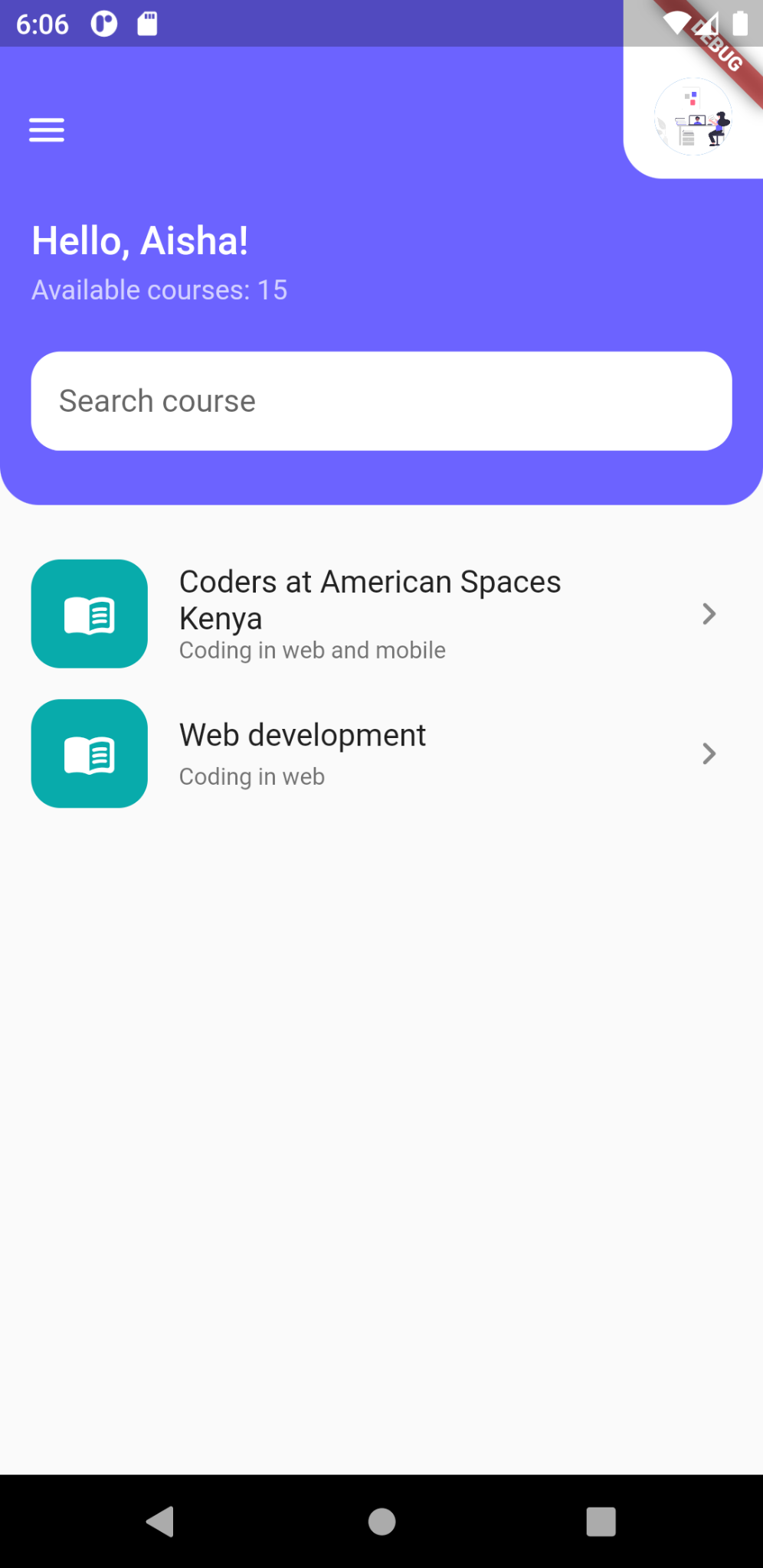
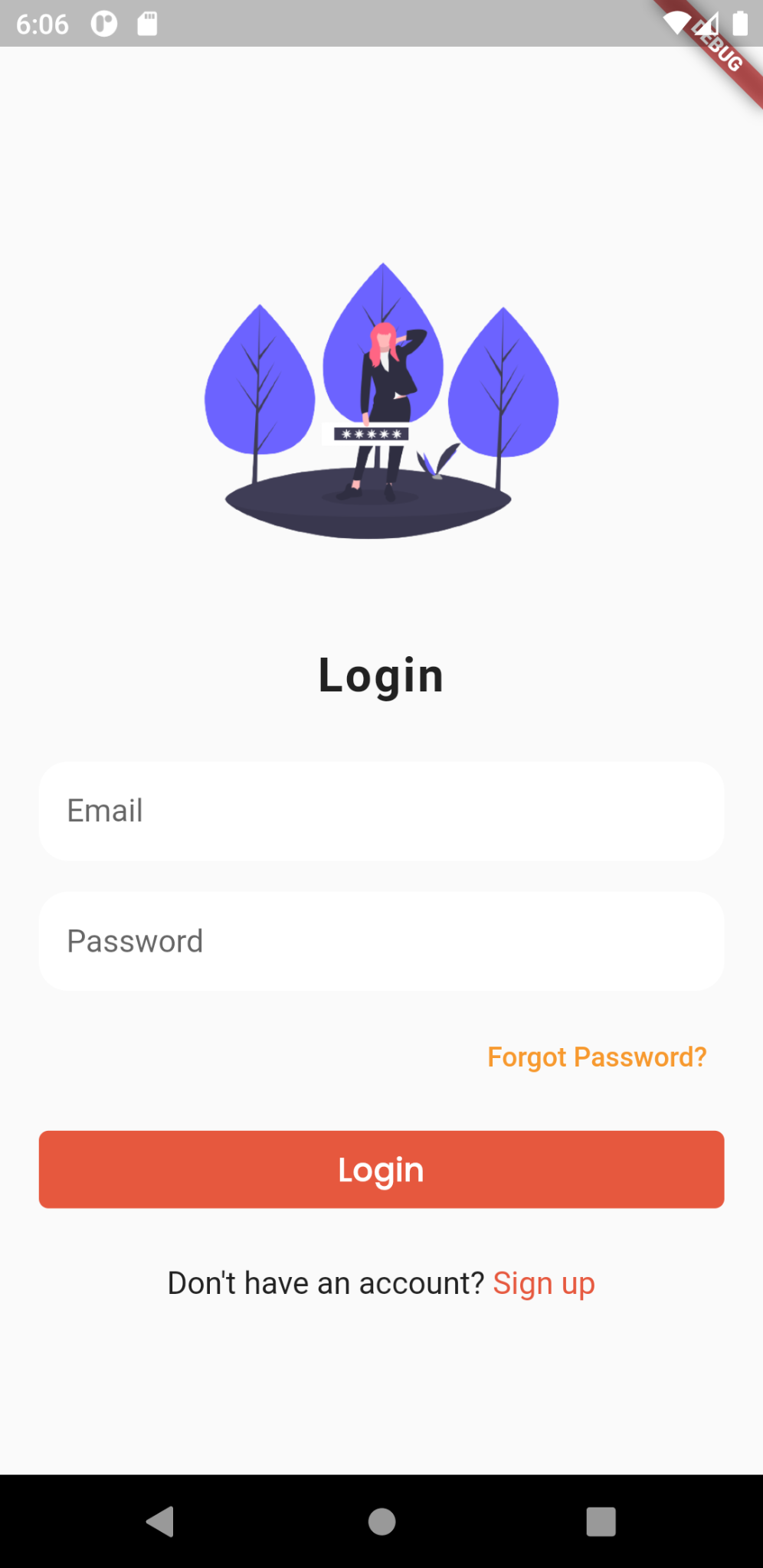
| import 'package:flutter/material.dart';  void main() => runApp(MyApp());   class MyApp extends StatelessWidget {   // This widget is the root of your application.   @override   Widget build(BuildContext context) {  return MaterialApp(  title: 'Flutter Demo', theme: ThemeData(  primarySwatch: Colors.blue,  ),   home: MyHomePage(title: 'Product layout demo home page'),   );  } } class MyHomePage extends StatelessWidget {   MyHomePage({Key key, this.title}) : super(key: key);   final String title;     @override   Widget build(BuildContext context) {   return Scaffold(   appBar: AppBar(title: Text("Product Listing")),   body: ListView(  shrinkWrap: true,   padding: const EdgeInsets.fromLTRB(2.0, 10.0, 2.0, 10.0),   children: <Widget>[   ProductBox(  name: "iPhone",   description: "iPhone is the stylist phone ever",   price: 1000,   image: "iphone.png"  ),   ProductBox(   name: "Pixel",   description: "Pixel is the most featureful phone ever",   price: 800,   image: "pixel.png"  ),   ProductBox(   name: "Laptop",   description: "Laptop is most productive development tool",   price: 2000,   image: "laptop.png"  ),   ProductBox(   name: "Tablet",   description: "Tablet is the most useful device ever for meeting",   price: 1500,   image: "tablet.png"  ),   ProductBox(   name: "Pendrive",   description: "Pendrive is useful storage medium",   price: 100,   image: "pendrive.png"  ),   ProductBox(  name: "Floppy Drive",   description: "Floppy drive is useful rescue storage medium",   price: 20,   image: "floppy.png"  ),   ],  )  );  } } class ProductBox extends StatelessWidget {  ProductBox({Key key, this.name, this.description, this.price, this.image}) :  super(key: key);   final String name;   final String description;   final int price;   final String image;     Widget build(BuildContext context) {  return Container(  padding: EdgeInsets.all(2),   height: 120,   child: Card(  child: Row(  mainAxisAlignment: MainAxisAlignment.spaceEvenly,   children: <Widget>[   Image.asset("assets/appimages/" + image),   Expanded(   child: Container(   padding: EdgeInsets.all(5),   child: Column(   mainAxisAlignment: MainAxisAlignment.spaceEvenly,   children: <Widget>[   Text(  this.name, style: TextStyle(  fontWeight: FontWeight.bold  )  ),  Text(this.description), Text(  "Price: " + this.price.toString()  ),   ],   )  )  )  ]  )  )  );  } } |
| --- |

The final output of the application is as follows −

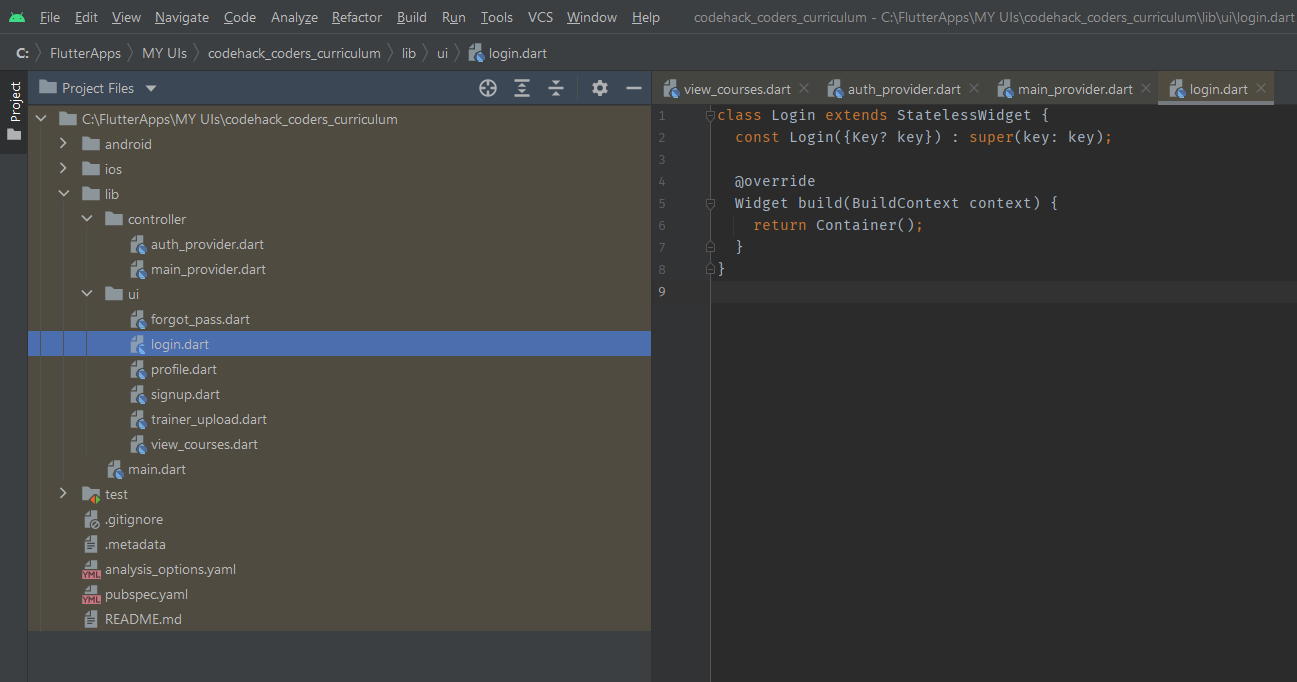


# User Interface Development

We are going to develop an app that manages courses for this course.



For authentication, the user shall login, sign up, reset password and have a profile. For the courses, the users shall see a dashboard with a list of courses. On clicking, they shall see details and further enroll in the course.



It is important to structure your user interfaces and backend appropriately. This is to ensure you separate concerns and arrange your work. For this task, you will do the following:

* In the lib folder, create 3 main folders. Name them ***controller*** and ***ui***. The controller will contain the backend, while ui will contain the frontend. Create files inside them and structure as follows:
  + The ***controller*** contains *auth\_provider.dart and main\_provider.dart*
  + The ***ui*** contains *forgot\_pass.dart, login.dart, profile.dart, signup.dart, forgot\_pass.dart, reset\_pass.dart, trainer\_upload.dart, add\_edit\_course.dart, edit\_profile.dart* and *view\_courses.dart*
  + Use the ***stless*** command to create the classes that extends Stateless widgets and name them appropriately

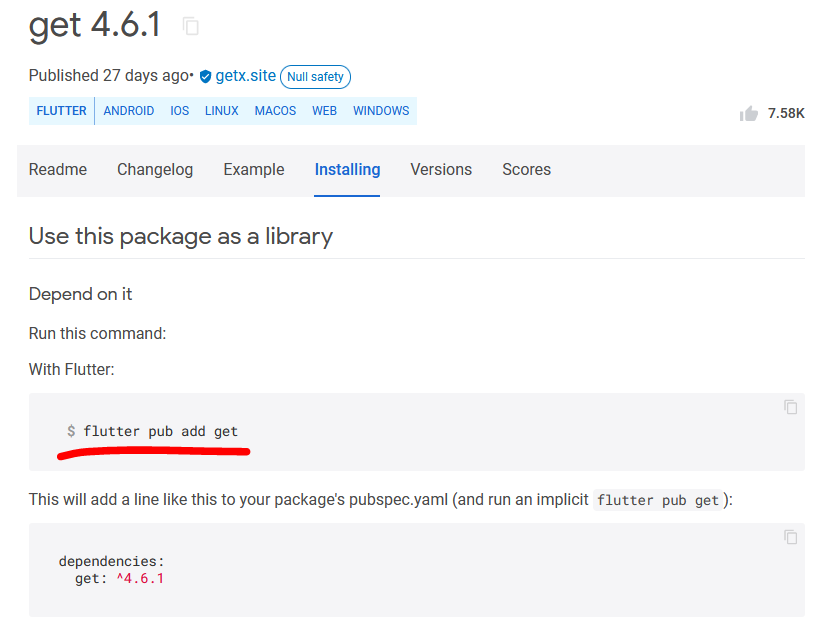
## Installing the required dependencies and adding Assets to Flutter

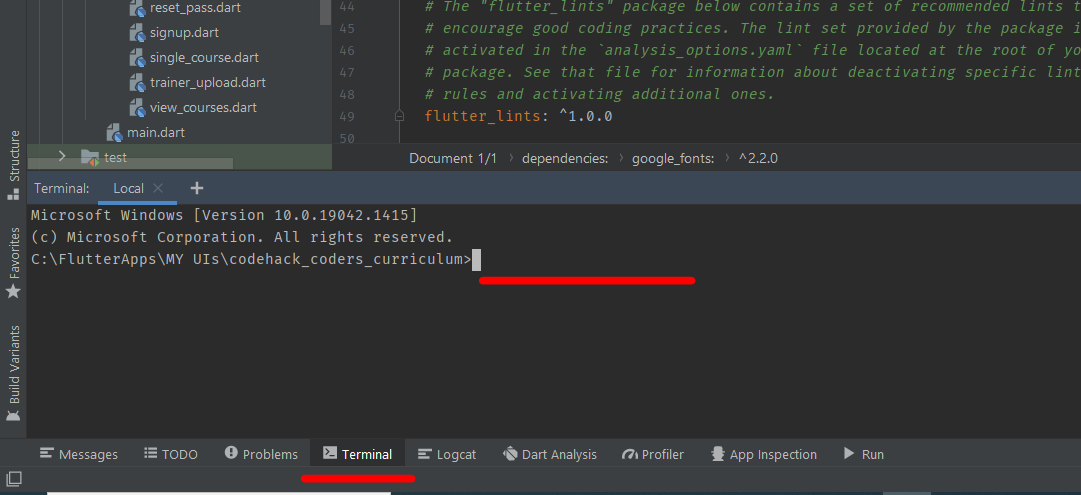
### Adding Dependencies

There are 2 ways in which you can install dependencies to Flutter.

1. Using command line and *running pub* command

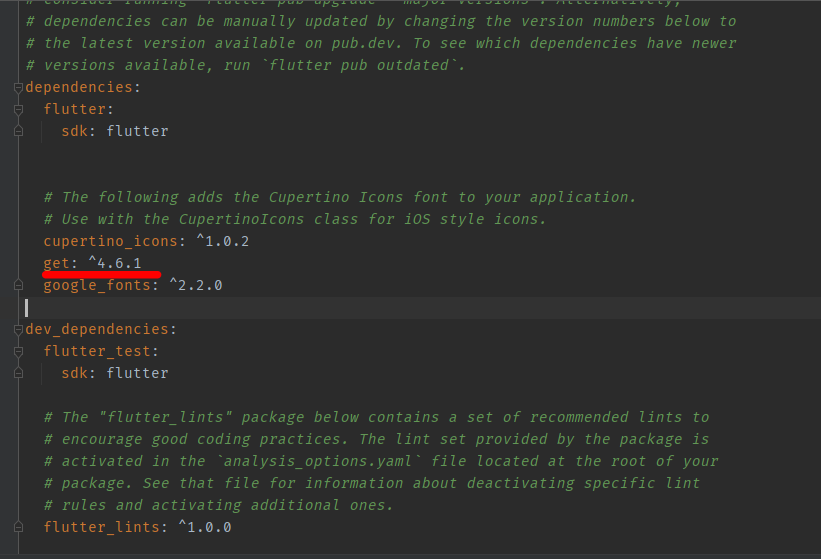
Go to Flutter’s pub.dev and search for the dependencies you need. We need to use GetX for state management (we shall explain further as we progress with the course).





Run the command in the terminal as instructed

1. Pasting in the *pubspec.yaml* file



Paste your Flutter dependencies after curpertino\_icons. Be careful to make sure you have indented properly.

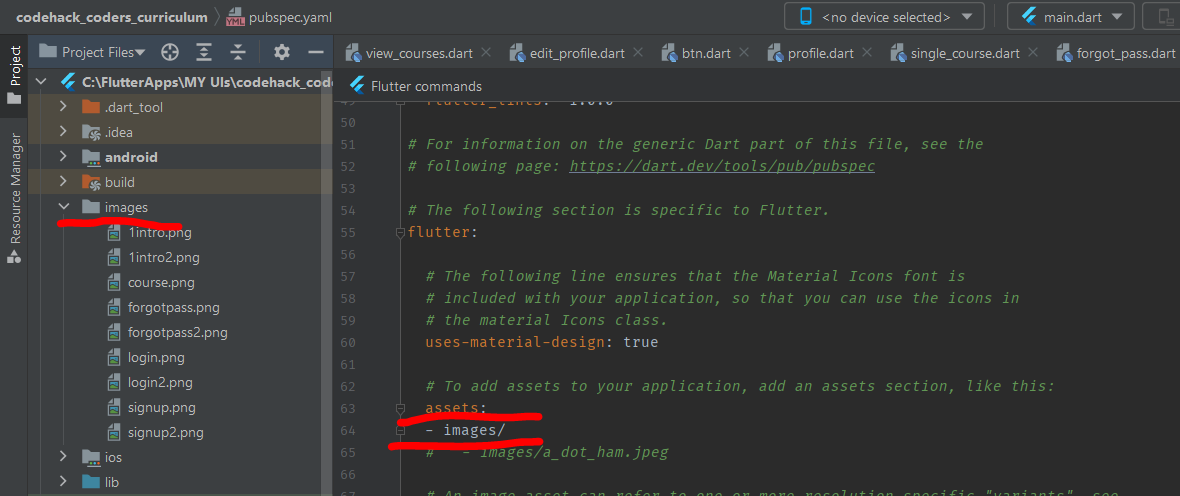
For this UI, we shall add 2 dependencies.

1. [GetX](https://pub.dev/packages/get/install)
2. [Google fonts](https://pub.dev/packages/google_fonts/install)

The rest, we shall add as we implement the backend.

### Adding Asset to your project

1. I have premade all the assets you will need for this project. [Download them here](https://drive.google.com/drive/folders/1QD4f81XSMW8EHCMLlrRmf6teoWeXt3Wr?usp=sharing).
2. Create a folder in the root of your project file. Call it **images**. Paste the images there
3. Now finally, in pubspec.yaml file, uncomment the following line by deleting the # and space before it. Be careful, yaml files are keen on indentation.



## The ***main.dart*** file

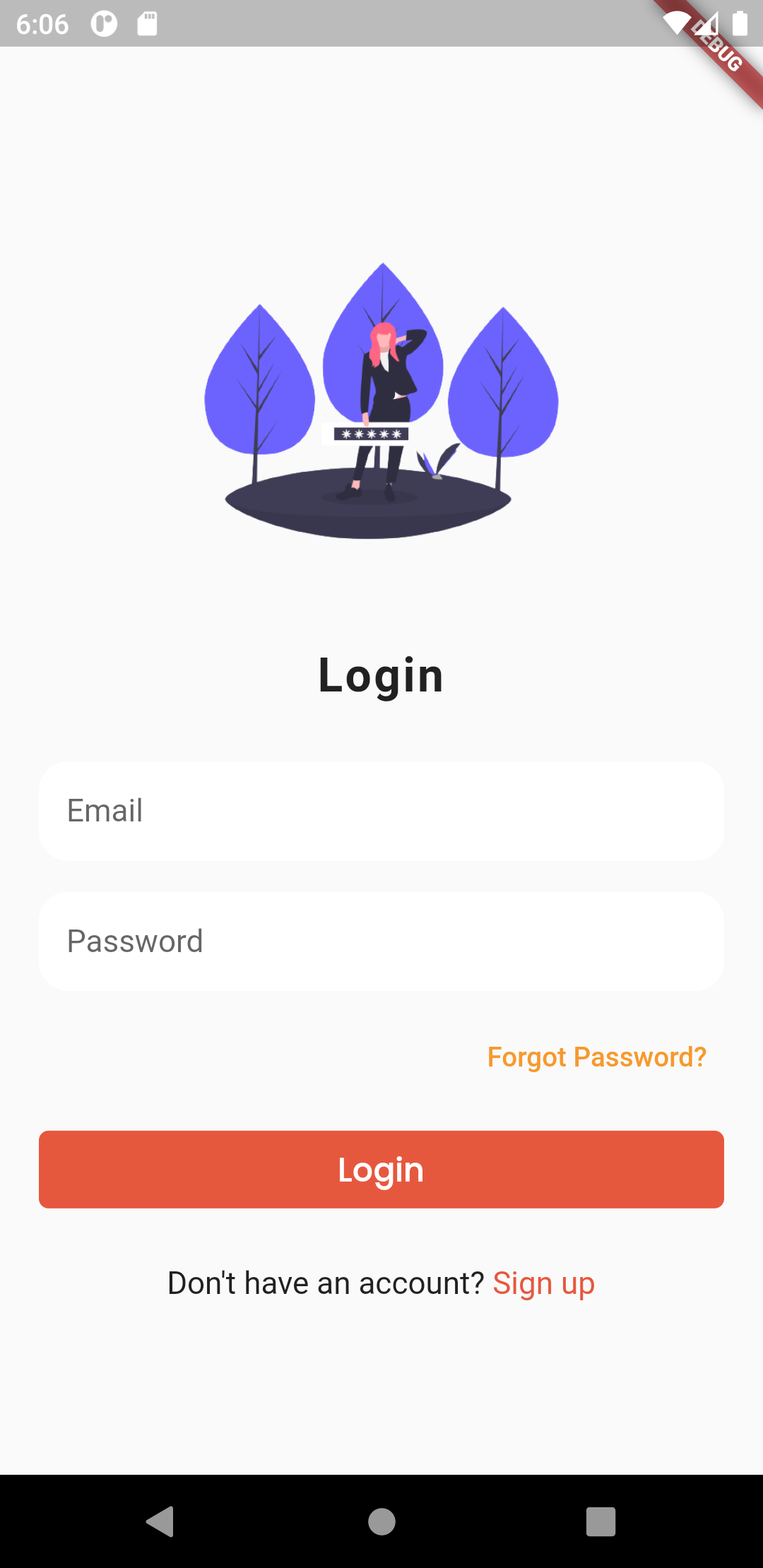
This is the entry point of the application. It contains major configurations that you need in the app, like state management, theme, routes, among others.

The initial Flutter app comes with a default app to counter and add the number of times you have clicked. Delete all that and add the following. Note that, we have changed **MaterialApp to GetMaterialApp**, which is the state management of GetX.

| import 'package:flutter/material.dart'; import 'package:get/get.dart';  import 'ui/login.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 GetMaterialApp(  title: 'Pwani Teknowgalz Courses',  theme: ThemeData(   primarySwatch: Colors.blue,  ),  home: const Login(),  );  } } |
| --- |

Login will be second in the hierarchy now.

## The ***login.dart*** file



To achieve this design, we shall use what we call Widgets, which are UI elements in Flutter. Flutter has a huge list of widgets, like Texts, Images, Icons, among others. We have widgets that can nest other widgets like Containers, Columns, Rows, etc. Scaffold is one root widget we shall use a lot, since it can host all other widgets well.

| import 'package:codehack\_coders\_curriculum/ui/forgot\_pass.dart'; import 'package:codehack\_coders\_curriculum/ui/signup.dart'; import 'package:codehack\_coders\_curriculum/ui/view\_courses.dart'; import 'package:codehack\_coders\_curriculum/ui/widgets/btn.dart'; import 'package:flutter/material.dart'; import 'package:get/get.dart';  import 'widgets/textfield.dart';  class Login extends StatelessWidget {  const Login({Key? key}) : super(key: key);   @override  Widget build(BuildContext context) {  return Scaffold(  body: Container(  width: MediaQuery.of(context).size.width,  height: MediaQuery.of(context).size.height,  padding: EdgeInsets.all(20),  child: SingleChildScrollView(  child: Column(  children: [  SizedBox(height: MediaQuery.of(context).size.height \* .09),  Container(  height: MediaQuery.of(context).size.height \* .3,  width: MediaQuery.of(context).size.height \* .3,  decoration: BoxDecoration(  image: DecorationImage(  image: AssetImage("images/login2.png"))),  ),  SizedBox(height: 18),  Text("Login",  style: Theme.of(context).textTheme.headline5!.copyWith(  fontWeight: FontWeight.bold, letterSpacing: 1.2)),  SizedBox(height: 30),  TxtField(  hintText: "Email",  inputType: TextInputType.emailAddress,  obscureText: false),  SizedBox(height: 16),  TxtField(  hintText: "Password",  inputType: TextInputType.text,  obscureText: true),  SizedBox(height: 10),  Row(  mainAxisAlignment: MainAxisAlignment.end,  children: [  TextButton(  onPressed: () {  Get.to(() => ForgotPass());  },  child: Text("Forgot Password?",  style: TextStyle(  color: Color.fromRGBO(247, 153, 46, 1)))),  ],  ),  SizedBox(height: 10),  Btn(  text: "Login",  onPressed: () {  Get.to(() => ViewCourses());  }),  SizedBox(height: 25),  Center(  child: InkWell(  onTap: () {  Get.to(() => Signup());  },  child: RichText(  text: TextSpan(  text: "Don\'t have an account?",  style: TextStyle(color: Colors.black87, fontSize: 16),  children: <TextSpan>[  TextSpan(  text: ' Sign up',  style: TextStyle(  color: Color.fromRGBO(230, 88, 62, 1),  fontSize: 16))  ]),  ),  ),  ),  SizedBox(height: 25),  ],  ),  ),  ),  );  } } |
| --- |

You will notice that there are errors, especially from custom widgets we have, namely **Btn** and **TxtField**.

## Custom Widgets

With Flutter, we create custom widgets so that we can reuse them. For example, a button is something we need in the entire app. Instead of creating it all the time, we can use a custom widget with Properties that suit the entire project.

### TxtField Custom Widget

| import 'package:flutter/material.dart';  class TxtField extends StatelessWidget {  const TxtField(  {Key? key,  required this.inputType,  required this.hintText,  this.obscureText = false,  this.textEditingController})  : super(key: key);  final TextInputType inputType;  final String hintText;  final bool obscureText;  final TextEditingController? textEditingController;   @override  Widget build(BuildContext context) {  return Theme(  data: Theme.of(context).copyWith(splashColor: Colors.transparent),  child: TextField(  controller: textEditingController,  autofocus: false,  style: TextStyle(fontSize: 16, color: Color(0xFFbdc6cf)),  decoration: InputDecoration(  filled: true,  fillColor: Colors.white,  hintText: hintText,  contentPadding:  const EdgeInsets.only(left: 14.0, bottom: 16.0, top: 16.0),  focusedBorder: OutlineInputBorder(  borderSide: BorderSide(color: Colors.white),  borderRadius: BorderRadius.circular(15),  ),  enabledBorder: UnderlineInputBorder(  borderSide: BorderSide(color: Colors.white),  borderRadius: BorderRadius.circular(15),  ),  ),  ),  );  } } |
| --- |

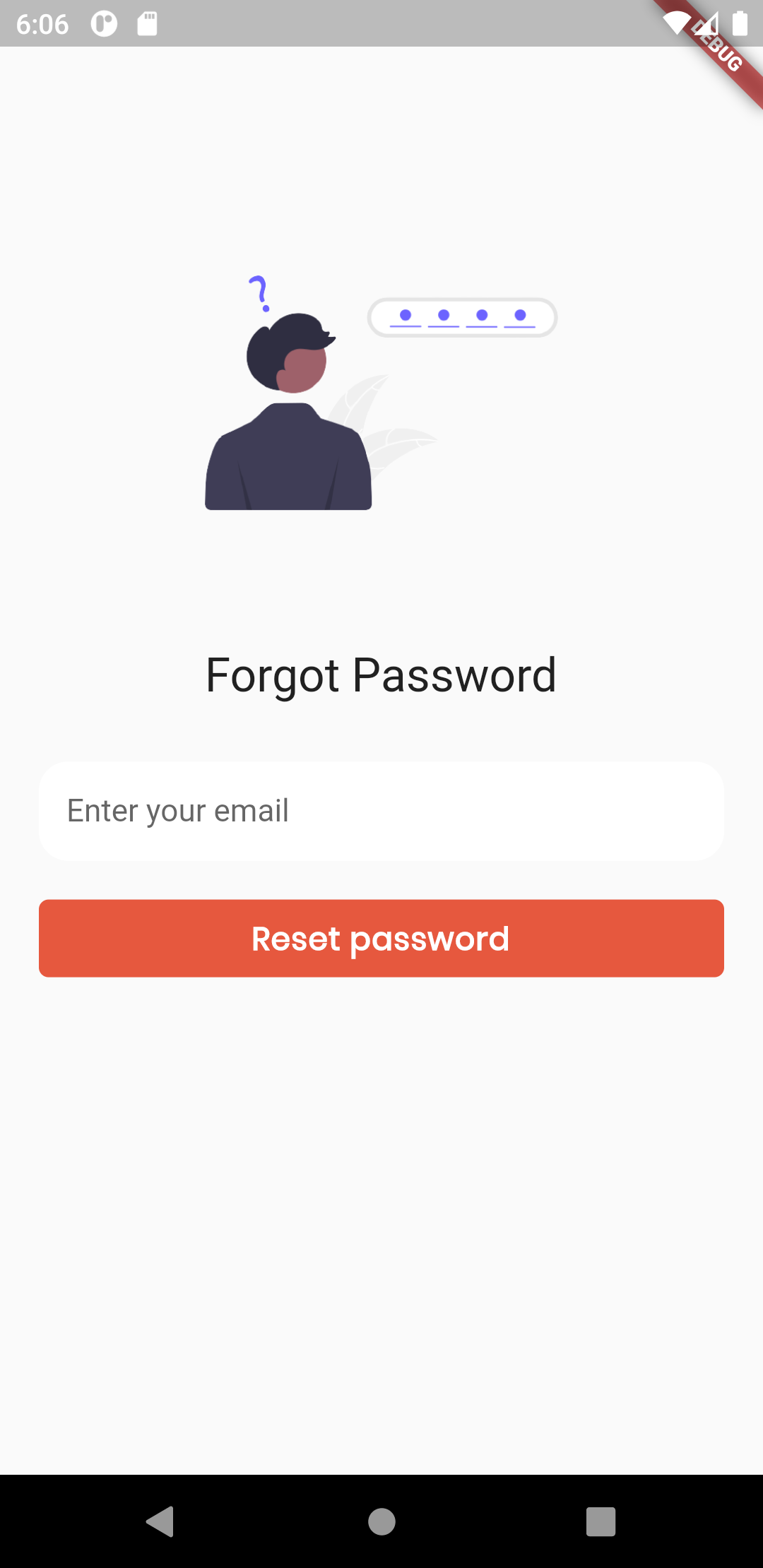
This custom widget has a TextField widget which contains fields that will be called in the other files that call this widget file.

### Btn Custom Widget

| import 'package:flutter/material.dart'; import 'package:google\_fonts/google\_fonts.dart';  class Btn extends StatelessWidget {  const Btn({Key? key, required this.text, required this.onPressed})  : super(key: key);  final String text;  final VoidCallback onPressed;   @override  Widget build(BuildContext context) {  return TextButton(  onPressed: onPressed,  child: Text(text,  style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold)),  style: TextButton.styleFrom(  minimumSize: Size(double.infinity, 40),  primary: Colors.white,  backgroundColor: Color.fromRGBO(230, 88, 62, 1),  textStyle: GoogleFonts.poppins()  .copyWith(fontWeight: FontWeight.w400, letterSpacing: .5),  padding: EdgeInsets.symmetric(horizontal: 16.0),  shape: const RoundedRectangleBorder(  borderRadius: BorderRadius.all(Radius.circular(5.0)),  ),  ));  } } |
| --- |

This widget too contains Button widget which has fields in the class, that will be called in the Classes that use this widget.

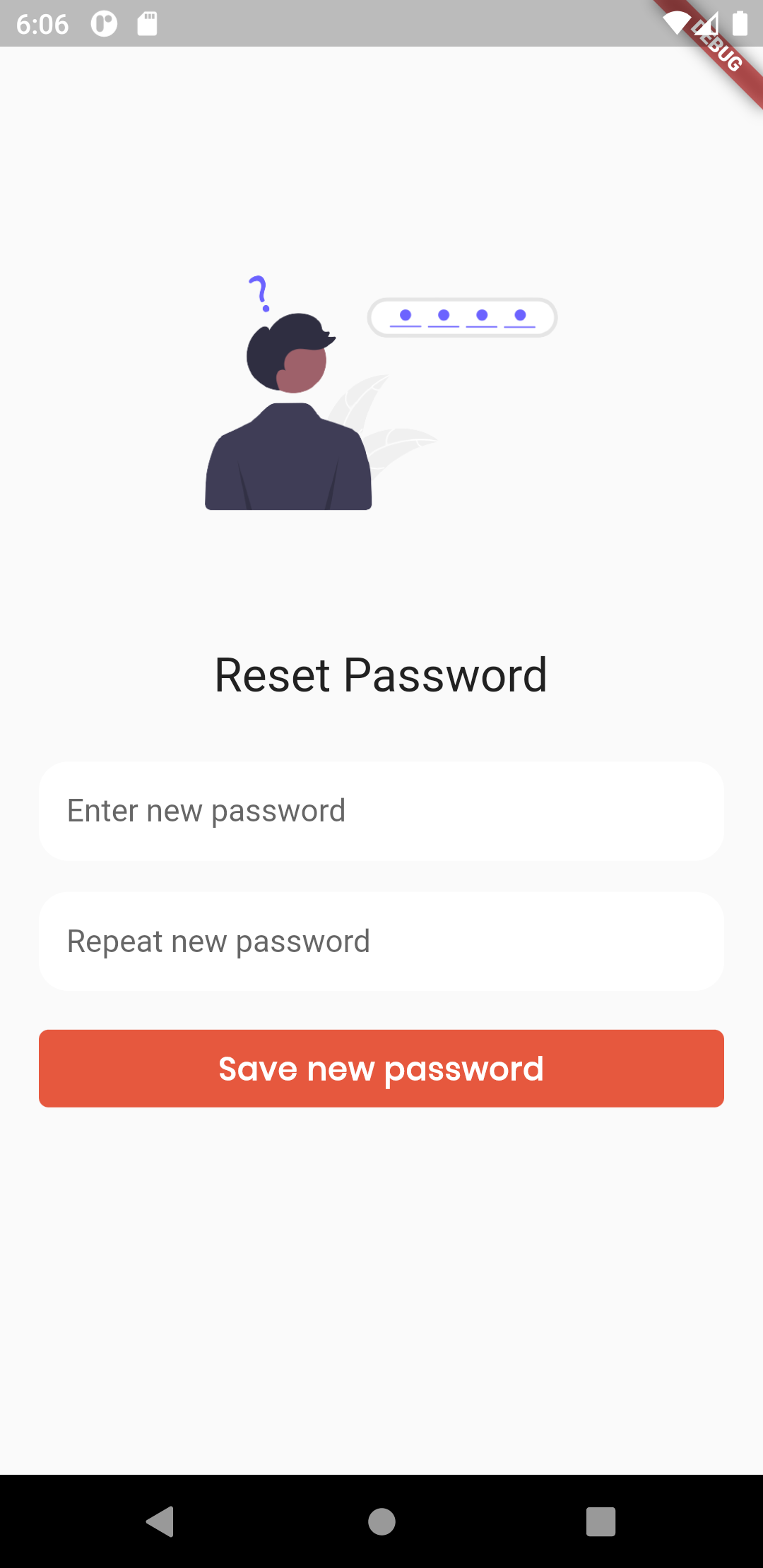
## The ***forgot\_pass.dart*** file



As you saw in the login.dart file, there’s a Get navigation to forgot password page. This is the code to achieve this

| import 'package:flutter/material.dart'; import 'package:get/get.dart';  import 'reset\_pass.dart'; import 'widgets/btn.dart'; import 'widgets/textfield.dart';  class ForgotPass extends StatelessWidget {  const ForgotPass({Key? key}) : super(key: key);   @override  Widget build(BuildContext context) {  return Scaffold(  body: Container(  width: MediaQuery.of(context).size.width,  height: MediaQuery.of(context).size.height,  padding: EdgeInsets.all(20),  child: SingleChildScrollView(  child: Column(  children: [  SizedBox(height: MediaQuery.of(context).size.height \* .09),  Container(  height: MediaQuery.of(context).size.height \* .3,  width: MediaQuery.of(context).size.height \* .3,  decoration: BoxDecoration(  image: DecorationImage(  image: AssetImage("images/forgotpass2.png"))),  ),  SizedBox(height: 18),  Text("Forgot Password",  style: Theme.of(context).textTheme.headline5!.copyWith()),  SizedBox(height: 30),  TxtField(  hintText: "Enter your email",  inputType: TextInputType.emailAddress,  obscureText: false),  SizedBox(height: 16),  Btn(  text: "Reset password",  onPressed: () {  Get.to(() => ResetPass());  }),  SizedBox(height: 25),  ],  ),  ),  ),  );  } } |
| --- |

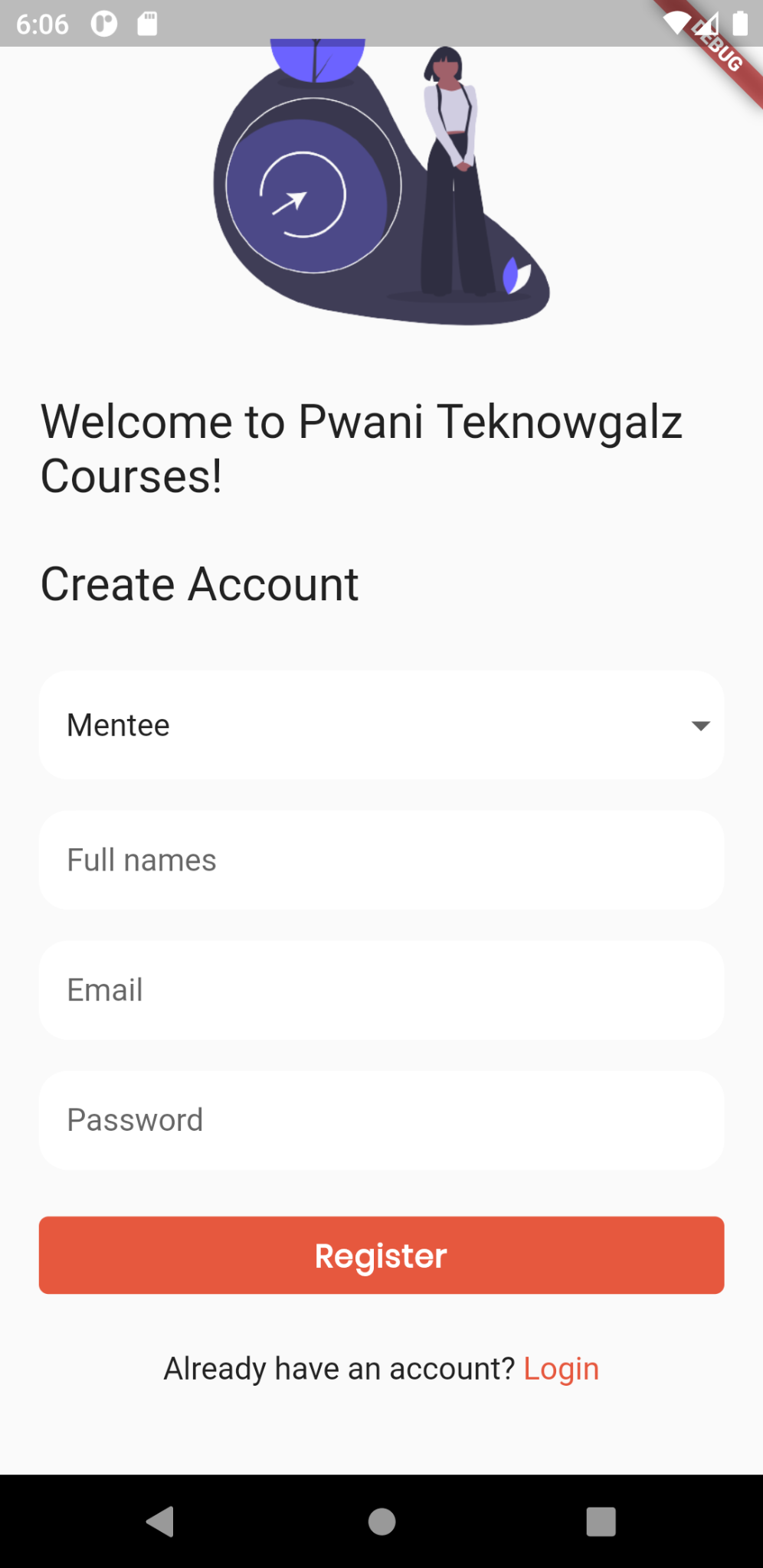
## The ***reset\_pass.dart*** file



The following screen is achieved by the following code

| import 'package:codehack\_coders\_curriculum/ui/view\_courses.dart'; import 'package:flutter/material.dart'; import 'package:get/get.dart';  import 'widgets/btn.dart'; import 'widgets/textfield.dart';  class ResetPass extends StatefulWidget {  const ResetPass({Key? key}) : super(key: key);   @override  \_ResetPassState createState() => \_ResetPassState(); }  class \_ResetPassState extends State<ResetPass> {  @override  Widget build(BuildContext context) {  return Scaffold(  body: Container(  width: MediaQuery.of(context).size.width,  height: MediaQuery.of(context).size.height,  padding: EdgeInsets.all(20),  child: SingleChildScrollView(  child: Column(  children: [  SizedBox(height: MediaQuery.of(context).size.height \* .09),  Container(  height: MediaQuery.of(context).size.height \* .3,  width: MediaQuery.of(context).size.height \* .3,  decoration: BoxDecoration(  image: DecorationImage(image: AssetImage("images/forgotpass2.png"))  ),  ),  SizedBox(height: 18),  Text("Reset Password",  style: Theme.of(context)  .textTheme  .headline5!  .copyWith(  )),  SizedBox(height: 30),  TxtField(hintText: "Enter new password", inputType: TextInputType.text,  obscureText: true),  SizedBox(height: 16),  TxtField(hintText: "Repeat new password", inputType: TextInputType.text,  obscureText: true),  SizedBox(height: 16),  Btn(text: "Save new password", onPressed: (){  Get.to(() => ViewCourses());  }),  SizedBox(height: 25),   ],  ),  ),  ),  );  } } |
| --- |

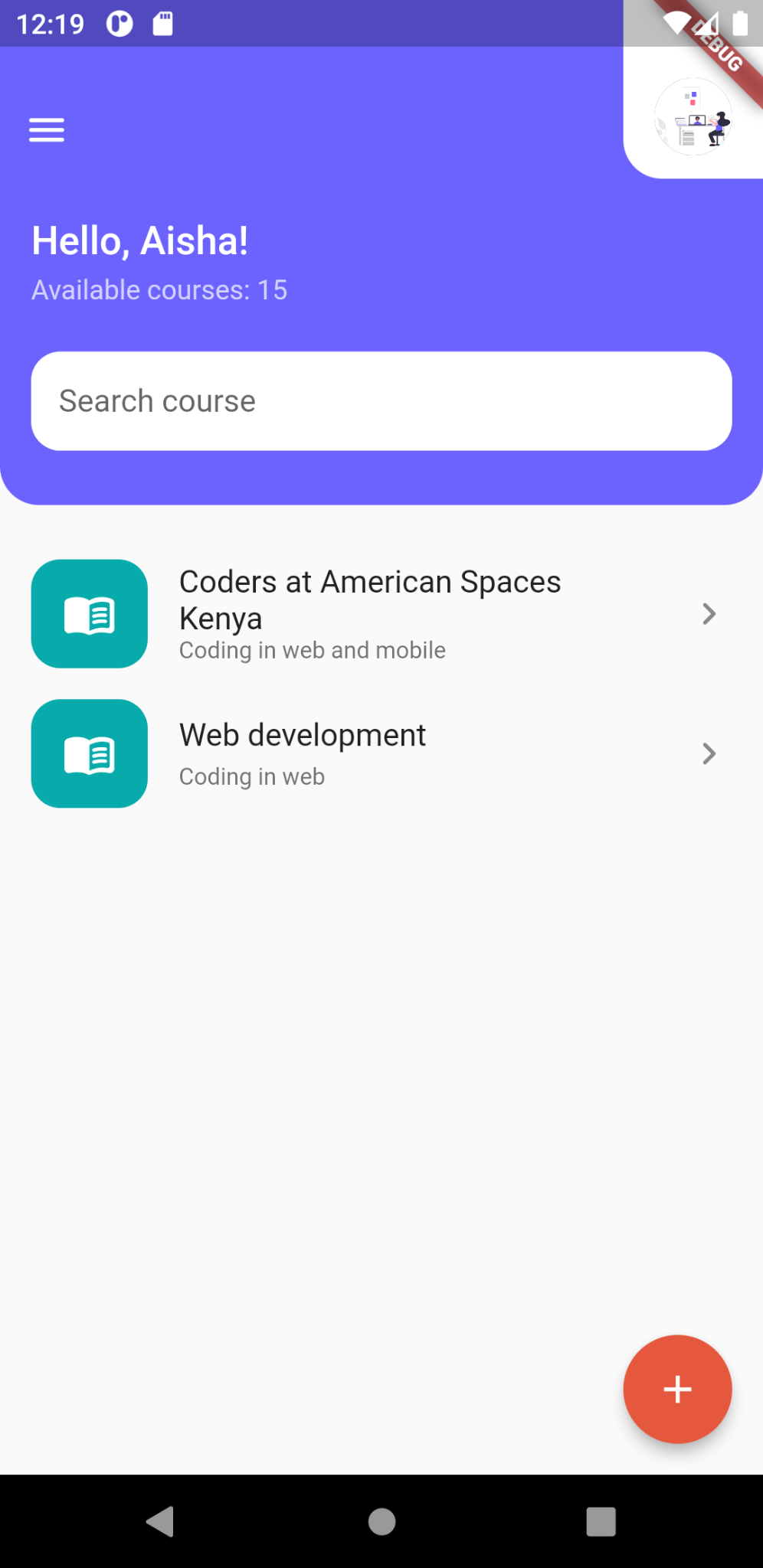
## The ***signup.dart*** file



The following code achieves the sign up screen shown above

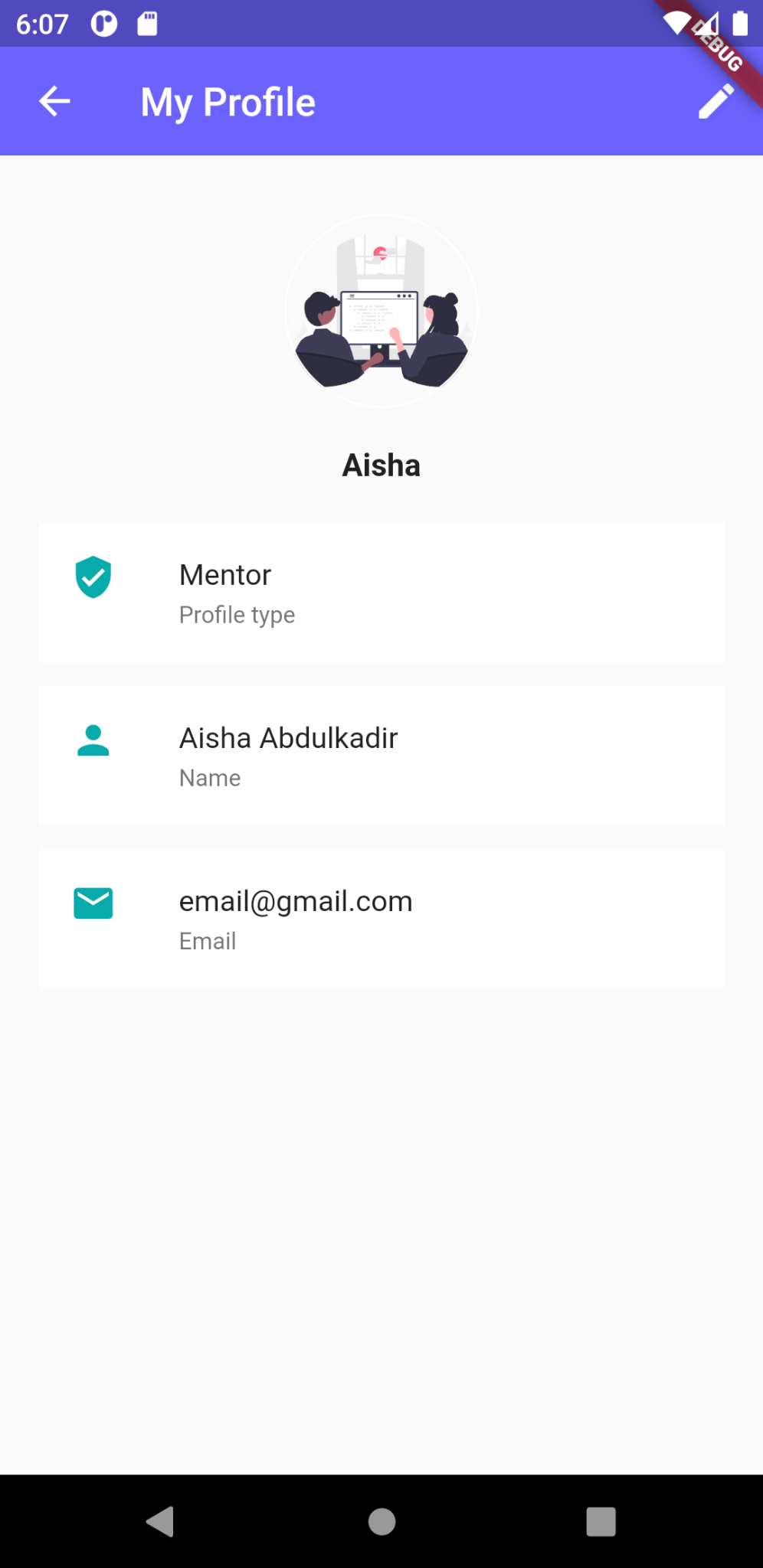
| import 'package:codehack\_coders\_curriculum/ui/login.dart'; import 'package:codehack\_coders\_curriculum/ui/view\_courses.dart'; import 'package:codehack\_coders\_curriculum/ui/widgets/btn.dart'; import 'package:codehack\_coders\_curriculum/ui/widgets/textfield.dart'; import 'package:flutter/material.dart'; import 'package:get/get.dart';  class Signup extends StatefulWidget {  const Signup({Key? key}) : super(key: key);   @override  State<Signup> createState() => \_SignupState(); }  class \_SignupState extends State<Signup> {  late String \_currentSelectedValue;   @override  void initState() {  \_currentSelectedValue = 'Mentee';  super.initState();  }   @override  Widget build(BuildContext context) {  var \_selectType = [  "Mentee",  "Mentor/Teacher",  ];   return Scaffold(  body: Container(  width: MediaQuery.of(context).size.width,  height: MediaQuery.of(context).size.height,  padding: EdgeInsets.all(20),  child: SingleChildScrollView(  child: Column(  children: [  SizedBox(height: MediaQuery.of(context).size.height \* .05),  Container(  height: MediaQuery.of(context).size.height \* .3,  width: MediaQuery.of(context).size.height \* .3,  decoration: BoxDecoration(  image: DecorationImage(  image: AssetImage("images/signup2.png"))),  ),  SizedBox(height: 18),  Text("Welcome to Pwani Teknowgalz Courses! \n\nCreate Account",  style: Theme.of(context).textTheme.headline5),  SizedBox(height: 30),  FormField<String>(  builder: (FormFieldState<String> state) {  return InputDecorator(  decoration: InputDecoration(  //labelStyle: textStyle,  errorStyle:  TextStyle(color: Colors.redAccent, fontSize: 16.0),  filled: true,  fillColor: Colors.white,  hintText: 'Mentor or mentee?',  contentPadding: const EdgeInsets.only(  left: 14.0, bottom: 16.0, top: 16.0),  focusedBorder: OutlineInputBorder(  borderSide: BorderSide(color: Colors.white),  borderRadius: BorderRadius.circular(15),  ),  enabledBorder: UnderlineInputBorder(  borderSide: BorderSide(color: Colors.white),  borderRadius: BorderRadius.circular(15),  ),  ),  //isEmpty: \_currentSelectedValue == '',  child: DropdownButtonHideUnderline(  child: DropdownButton<String>(  value: \_currentSelectedValue,  isDense: true,  onChanged: (newValue) {  setState(() {  \_currentSelectedValue = newValue!;  state.didChange(newValue);  });  },  items: \_selectType.map((String value) {  return DropdownMenuItem<String>(  value: value,  child: Text(value),  );  }).toList(),  ),  ),  );  },  ),  SizedBox(height: 16),  TxtField(  hintText: "Full names",  inputType: TextInputType.text,  obscureText: false),  SizedBox(height: 16),  TxtField(  hintText: "Email",  inputType: TextInputType.emailAddress,  obscureText: false),  SizedBox(height: 16),  TxtField(  hintText: "Password",  inputType: TextInputType.text,  obscureText: true),  SizedBox(height: 20),  Btn(  text: "Register",  onPressed: () {  Get.to(() => ViewCourses());  }),  SizedBox(height: 25),  Center(  child: InkWell(  onTap: () {  Get.to(() => Login());  },  child: RichText(  text: TextSpan(  text: "Already have an account?",  style: TextStyle(color: Colors.black87, fontSize: 16),  children: <TextSpan>[  TextSpan(  text: ' Login',  style: TextStyle(  color: Color.fromRGBO(230, 88, 62, 1),  fontSize: 16))  ]),  ),  ),  ),  SizedBox(height: 25),  ],  ),  ),  ),  );  } } |
| --- |

## The ***view\_courses.dart*** file



| import 'package:codehack\_coders\_curriculum/ui/add\_edit\_course.dart'; import 'package:codehack\_coders\_curriculum/ui/profile.dart'; import 'package:codehack\_coders\_curriculum/ui/single\_course.dart'; import 'package:flutter/material.dart'; import 'package:get/get.dart';  import 'widgets/textfield.dart';  class ViewCourses extends StatelessWidget {  const ViewCourses({Key? key}) : super(key: key);   @override  Widget build(BuildContext context) {  return Scaffold(  body: Container(  child: SingleChildScrollView(  child: Column(  children: [  //<Custom header that will scroll   //you can use AppBar if you want a sticky appbar  // >  Container(  decoration: BoxDecoration(  color: Color.fromRGBO(108, 99, 255, 1),  borderRadius: BorderRadius.only(  bottomLeft: Radius.circular(20),  bottomRight: Radius.circular(20))),  child: Column(  children: [  Row(  mainAxisAlignment: MainAxisAlignment.spaceBetween,  children: [  Container(  margin: EdgeInsets.only(top: 42),  child: IconButton(  onPressed: () {},  icon: Icon(  Icons.menu\_rounded,  color: Colors.white,  ))),  InkWell(  onTap: () {  Get.to(() => Profile());  },  child: Container(  decoration: BoxDecoration(  color: Colors.white,  borderRadius: BorderRadius.only(  bottomLeft: Radius.circular(20))),  padding: EdgeInsets.only(  top: 40, left: 16, right: 16, bottom: 12),  child: CircleAvatar(  backgroundImage: AssetImage('images/1intro.png')),  ),  ),  ],  ),  SizedBox(height: 20),  //<Title   //Container is there to wrap and add padding  // >  Container(  width: MediaQuery.of(context).size.width,  padding: EdgeInsets.only(left: 16),  child: Column(  crossAxisAlignment: CrossAxisAlignment.start,  children: [  Text("Hello, Aisha!",  style: Theme.of(context)  .textTheme  .headline6  ?.copyWith(color: Colors.white)),  SizedBox(height: 6),  Text("Available courses: 15",  style: Theme.of(context)  .textTheme  .bodyText2  ?.copyWith(color: Colors.white70)),  ],  )),  SizedBox(height: 8),  Container(  padding: EdgeInsets.all(16),  child: TxtField(  hintText: "Search course",  inputType: TextInputType.text,  obscureText: false),  ),  SizedBox(height: 12),  ],  ),  ),   Column(  children: [  SizedBox(height: 20),  ListTile(  leading: Container(  width: 60,  height: 70,  decoration: BoxDecoration(  color: Color.fromRGBO(8, 171, 171, 1),  borderRadius: BorderRadius.circular(15)),  child: Icon(Icons.menu\_book\_rounded,  color: Colors.white, size: 28),  ),  title: Text("Coders at American Spaces Kenya"),  subtitle: Text("Coding in web and mobile",  style: TextStyle(fontSize: 12)),  trailing: Icon(Icons.chevron\_right\_rounded),  onTap: () {  Get.to(() => SingleCourse());  },  ),  ListTile(  leading: Container(  width: 60,  height: 70,  decoration: BoxDecoration(  color: Color.fromRGBO(8, 171, 171, 1),  borderRadius: BorderRadius.circular(15)),  child: Icon(Icons.menu\_book\_rounded,  color: Colors.white, size: 28),  ),  title: Text("Web development"),  subtitle:  Text("Coding in web", style: TextStyle(fontSize: 12)),  trailing: Icon(Icons.chevron\_right\_rounded),  )  ],  ),  ],  )),  ),  floatingActionButton: FloatingActionButton(  onPressed: () {  Get.to(() => AddEditCourse());  },  backgroundColor: Color.fromRGBO(230, 88, 62, 1),  child: Icon(  Icons.add,  color: Colors.white,  )),  );  } } |
| --- |

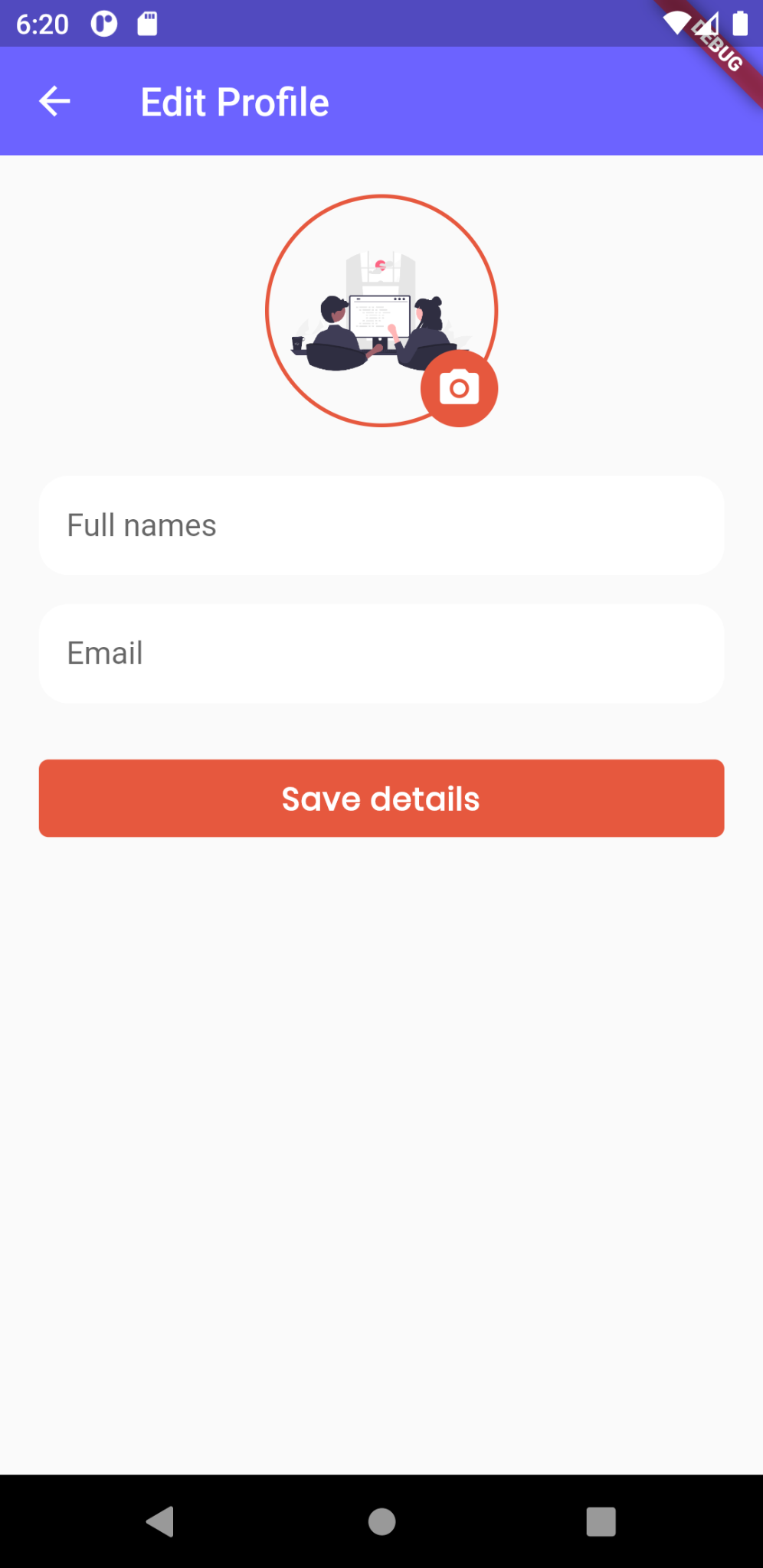
## The ***profile.dart*** file



| import 'package:flutter/material.dart'; import 'package:get/get.dart';  import 'edit\_profile.dart';  class Profile extends StatefulWidget {  @override  \_ProfileState createState() => \_ProfileState(); }  class \_ProfileState extends State<Profile> {  void initState() {  super.initState();  }   @override  Widget build(BuildContext context) {  return Scaffold(  backgroundColor: Colors.grey[50],  appBar: AppBar(  elevation: 0,  backgroundColor: Color.fromRGBO(108, 99, 255, 1),  automaticallyImplyLeading: false,  leading: IconButton(  icon: Icon(Icons.arrow\_back, color: Colors.white),  onPressed: () => Navigator.pop(context)),  title: Text(  "My Profile",  ),  actions: [  IconButton(  icon: Icon(Icons.edit, color: Colors.white),  onPressed: () {  Get.to(() => EditProfile());  },  ),  ],  ),  body: Container(  width: Get.size.width,  height: Get.size.height,  padding: EdgeInsets.all(20),  child: SingleChildScrollView(  child: Column(  children: [  SizedBox(height: 10),  Container(  width: 100,  height: 100,  decoration: BoxDecoration(  shape: BoxShape.circle,  border: Border.all(color: Colors.white, width: 1),  image: DecorationImage(  image: AssetImage("images/course.png"),  fit: BoxFit.cover)),  ),   SizedBox(height: 20),  Text(  "Aisha",  style: TextStyle(  color: Colors.black87,  fontSize: 16,  fontWeight: FontWeight.bold),  ),  SizedBox(height: 5),  // Text(  // "General public profile",  // style: TextStyle(color: Colors.black54),  // ),  SizedBox(height: 15),  Container(  color: Colors.white,  child: ListTile(  leading: Icon(Icons.verified\_user,  color: Color.fromRGBO(8, 171, 171, 1)),  title: Text(  "Mentor",  style: TextStyle(fontSize: 15),  ),  subtitle: Text(  "Profile type",  style: TextStyle(fontSize: 12, color: Colors.black54),  ),  )),  SizedBox(height: 12),  Container(  color: Colors.white,  child: ListTile(  leading: Icon(Icons.person,  color: Color.fromRGBO(8, 171, 171, 1)),  title: Text(  "Aisha Abdulkadir",  //"${ authController.userProfile['email'].toString().trim().isEmpty ? '[NOT SET]' : authController.userProfile['email']}",  style: TextStyle(fontSize: 15),  ),  subtitle: Text(  "Name",  style: TextStyle(fontSize: 12, color: Colors.black54),  ),  )),  SizedBox(height: 12),  Container(  color: Colors.white,  child: ListTile(  leading: Icon(Icons.email,  color: Color.fromRGBO(8, 171, 171, 1)),  title: Text(  "email@gmail.com",  style: TextStyle(fontSize: 15),  ),  subtitle: Text(  "Email",  style: TextStyle(fontSize: 12, color: Colors.black54),  ),  )),  SizedBox(height: 12),  ],  ),  )),  );  } } |
| --- |

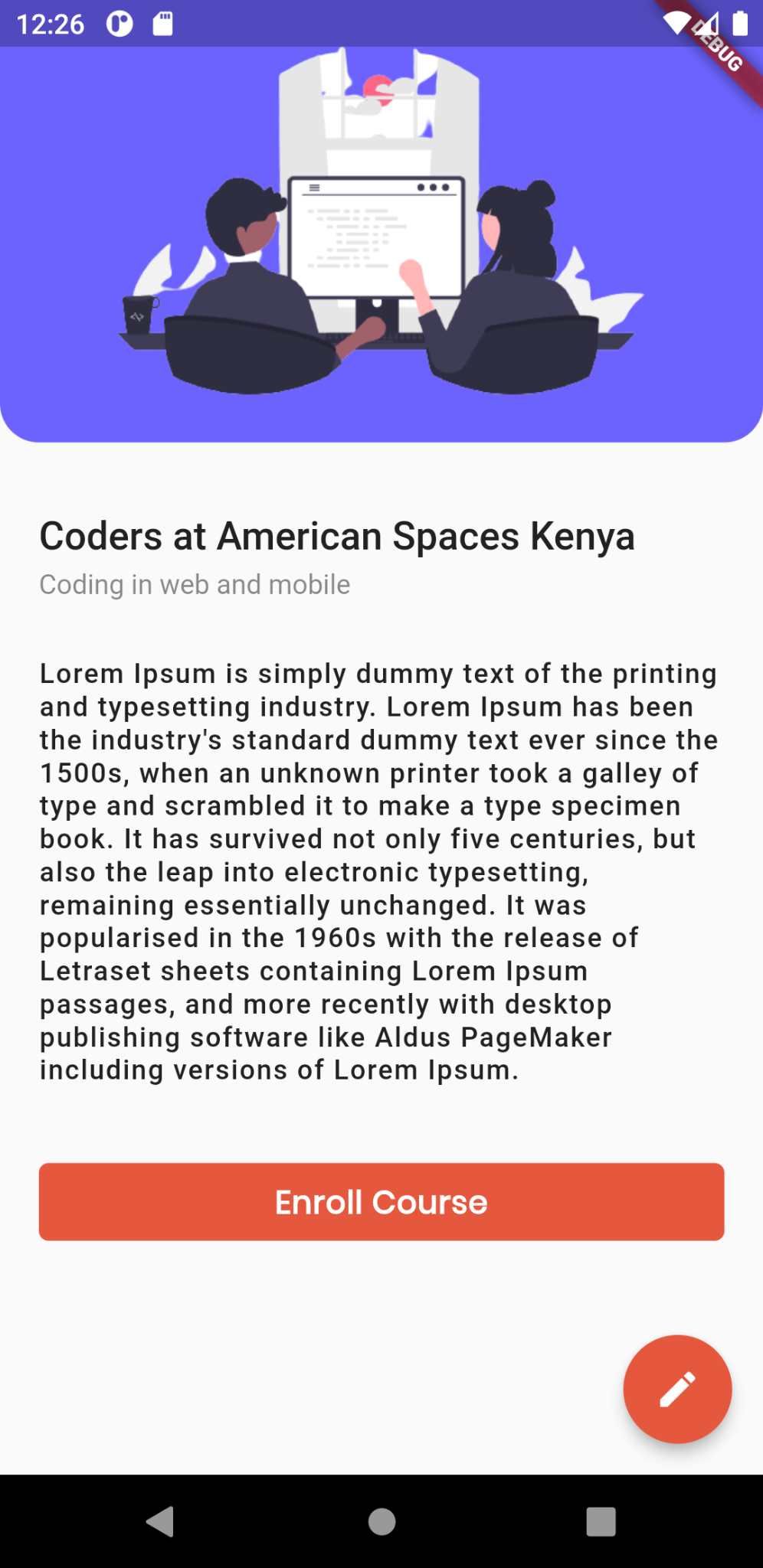
## 

## The ***edit\_profile.dart*** file



| import 'package:codehack\_coders\_curriculum/ui/profile.dart'; import 'package:flutter/material.dart'; import 'package:get/get.dart';  import 'widgets/btn.dart'; import 'widgets/textfield.dart';  class EditProfile extends StatefulWidget {  const EditProfile({Key? key}) : super(key: key);   @override  \_EditProfileState createState() => \_EditProfileState(); }  class \_EditProfileState extends State<EditProfile> {  @override  Widget build(BuildContext context) {  return Scaffold(  backgroundColor: Colors.grey[50],  appBar: AppBar(  elevation: 0,  backgroundColor: Color.fromRGBO(108, 99, 255, 1),  automaticallyImplyLeading: false,  leading: IconButton(  icon: Icon(Icons.arrow\_back, color: Colors.white),  onPressed: () => Navigator.pop(context)),  title: Text(  "Edit Profile",  ),  ),  body: Container(  width: Get.size.width,  height: Get.size.height,  padding: EdgeInsets.all(20),  child: SingleChildScrollView(  child: Column(  children: [  Container(  height: 120,  width: 120,  child: Stack(  children: [  InkWell(  onTap: () {},  child: Container(  height: 120,  width: 120,  padding: EdgeInsets.all(12),  decoration: BoxDecoration(  image: DecorationImage(  image: AssetImage("images/course.png")),  shape: BoxShape.circle,  border: Border.all(  color: Color.fromRGBO(230, 88, 62, 1), width: 2),  ),  ),  ),  Positioned(  bottom: 0,  right: 0,  child: Container(  decoration: BoxDecoration(  color: Color.fromRGBO(230, 88, 62, 1),  shape: BoxShape.circle),  width: 40,  height: 40,  alignment: Alignment.center,  child: Icon(  Icons.camera\_alt,  color: Colors.white,  size: 24,  )),  ),  ],  ),  ),  SizedBox(height: 25),  TxtField(hintText: "Full names", inputType: TextInputType.text),  SizedBox(height: 15),  TxtField(  hintText: "Email", inputType: TextInputType.emailAddress),  SizedBox(height: 25),  Btn(  text: "Save details",  onPressed: () {  Get.to(() => Profile());  }),  SizedBox(height: 25),  ],  ),  ),  ),  );  } } |
| --- |

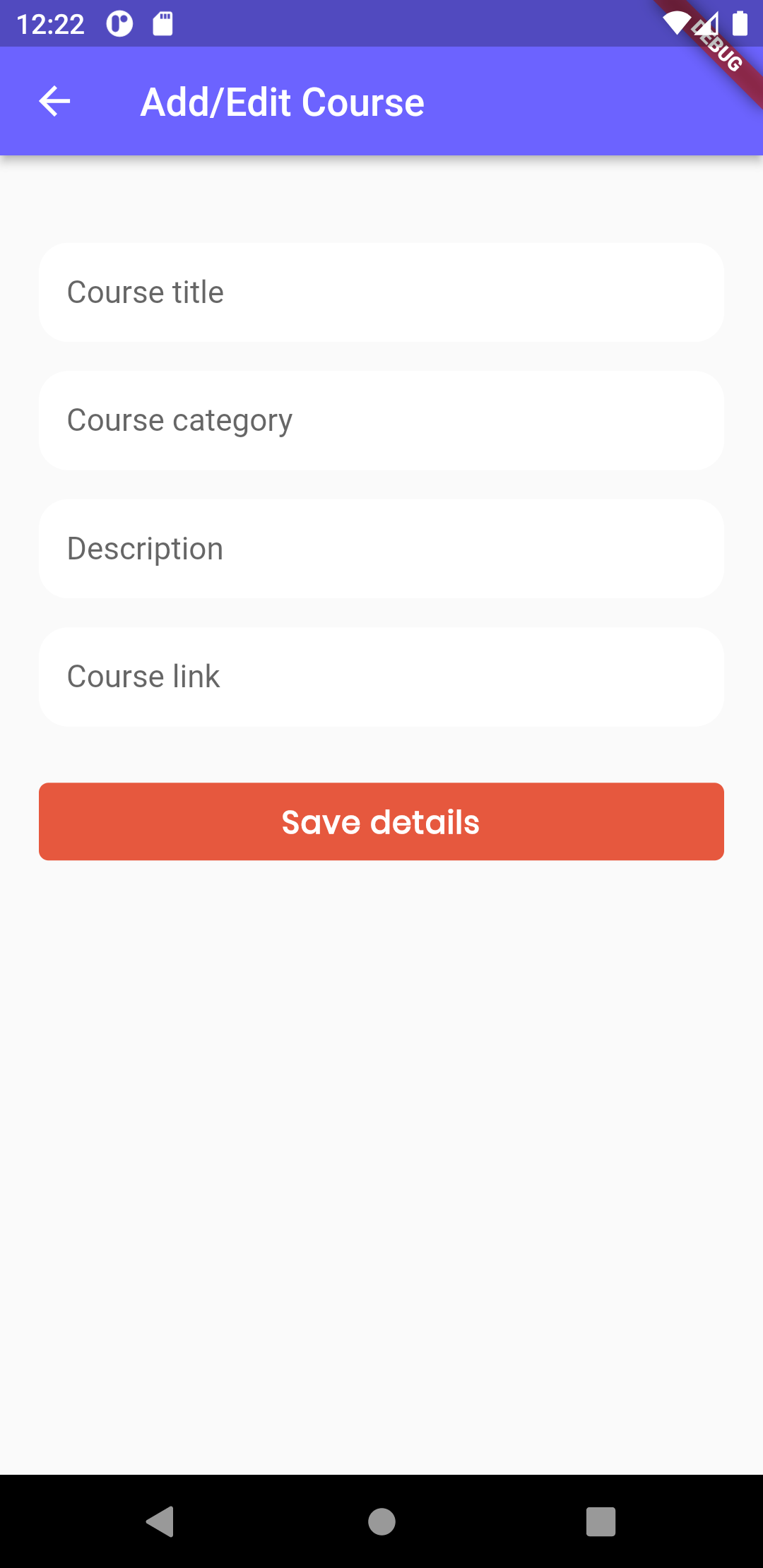
## The ***single\_course.dart*** file



| import 'package:flutter/material.dart'; import 'package:get/get.dart';  import 'add\_edit\_course.dart'; import 'widgets/btn.dart';  class SingleCourse extends StatefulWidget {  const SingleCourse({Key? key}) : super(key: key);   @override  \_SingleCourseState createState() => \_SingleCourseState(); }  class \_SingleCourseState extends State<SingleCourse> {  @override  Widget build(BuildContext context) {  return Scaffold(  body: SingleChildScrollView(  child: Column(  children: [  Container(  width: MediaQuery.of(context).size.width,  height: MediaQuery.of(context).size.height \* .3,  decoration: BoxDecoration(  color: Color.fromRGBO(108, 99, 255, 1),  borderRadius: BorderRadius.only(  bottomLeft: Radius.circular(20),  bottomRight: Radius.circular(20))),  foregroundDecoration: BoxDecoration(  image:  DecorationImage(image: AssetImage('images/course.png'))),  ),  Container(  padding: EdgeInsets.all(20),  child: Column(  crossAxisAlignment: CrossAxisAlignment.start,  mainAxisSize: MainAxisSize.min,  children: [  SizedBox(height: 16),  Text("Coders at American Spaces Kenya",  style: Theme.of(context).textTheme.headline6),  SizedBox(height: 6),  Text("Coding in web and mobile",  style: Theme.of(context)  .textTheme  .bodyText2  ?.copyWith(color: Colors.black45)),  SizedBox(height: 30),  Text(  "Lorem Ipsum is simply a dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularized in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.",  style: Theme.of(context).textTheme.bodyText1?.copyWith(  color: Colors.black87,  letterSpacing: .8,  height: 1.2)),  SizedBox(height: 35),  Btn(text: "Enroll Course", onPressed: () {}),  SizedBox(height: 25),  ],  ),  )  ],  ),  ),  floatingActionButton: FloatingActionButton(  onPressed: () {  Get.to(() => AddEditCourse());  },  backgroundColor: Color.fromRGBO(230, 88, 62, 1),  child: Icon(  Icons.edit,  color: Colors.white,  )),  );  } } |
| --- |

## 

## The ***add\_edit\_course.dart*** file



| import 'package:flutter/material.dart'; import 'package:get/get.dart';  import 'widgets/btn.dart'; import 'widgets/textfield.dart';  class AddEditCourse extends StatefulWidget {  const AddEditCourse({Key? key}) : super(key: key);   @override  \_AddEditCourseState createState() => \_AddEditCourseState(); }  class \_AddEditCourseState extends State<AddEditCourse> {  @override  Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(  title: Text("Add/Edit Course"),  backgroundColor: Color.fromRGBO(108, 99, 255, 1),  ),  body: Container(  width: Get.size.width,  height: Get.size.height,  padding: EdgeInsets.all(20),  child: SingleChildScrollView(  child: Column(  children: [  SizedBox(height: 25),  TxtField(hintText: "Course title", inputType: TextInputType.text),  SizedBox(height: 15),  TxtField(  hintText: "Course category", inputType: TextInputType.text),  SizedBox(height: 15),  TxtField(hintText: "Description", inputType: TextInputType.text),  SizedBox(height: 15),  TxtField(hintText: "Course link", inputType: TextInputType.url),  SizedBox(height: 25),  Btn(  text: "Save details",  onPressed: () {  Navigator.pop(context);  }),  SizedBox(height: 25),  ],  ),  ),  ),  );  } } |
| --- |

# Backend and State Management

## Setting up Backend

First, add the following libraries to your **pubspec.yaml** file. These are loader libraries, state manager and Parse SDK libraries you need for your app

# The following adds the Cupertino Icons font to your application.

# Use with the CupertinoIcons class for iOS style icons.

cupertino\_icons: ^1.0.2

get: ^4.6.1

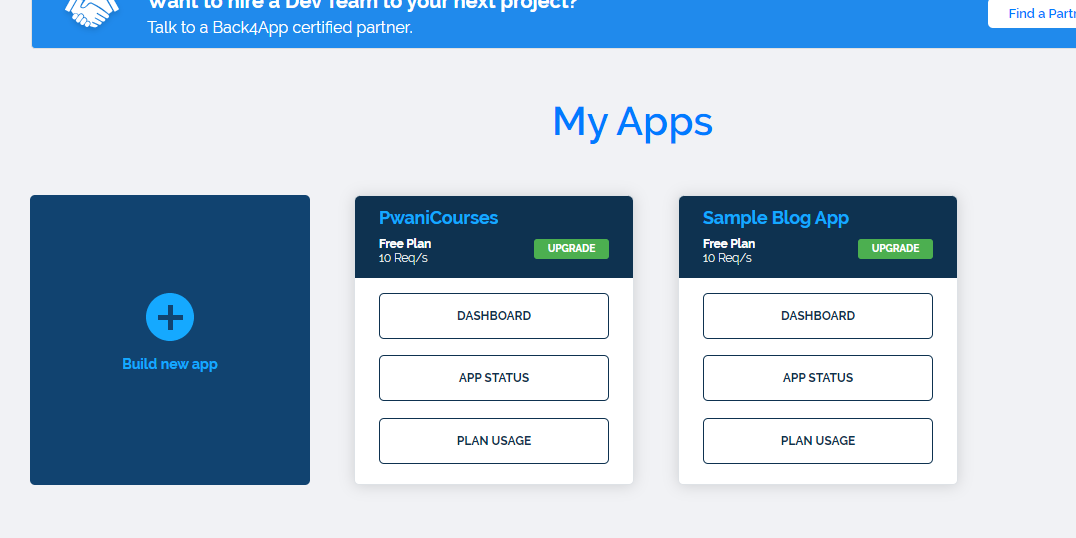
google\_fonts: ^2.2.0

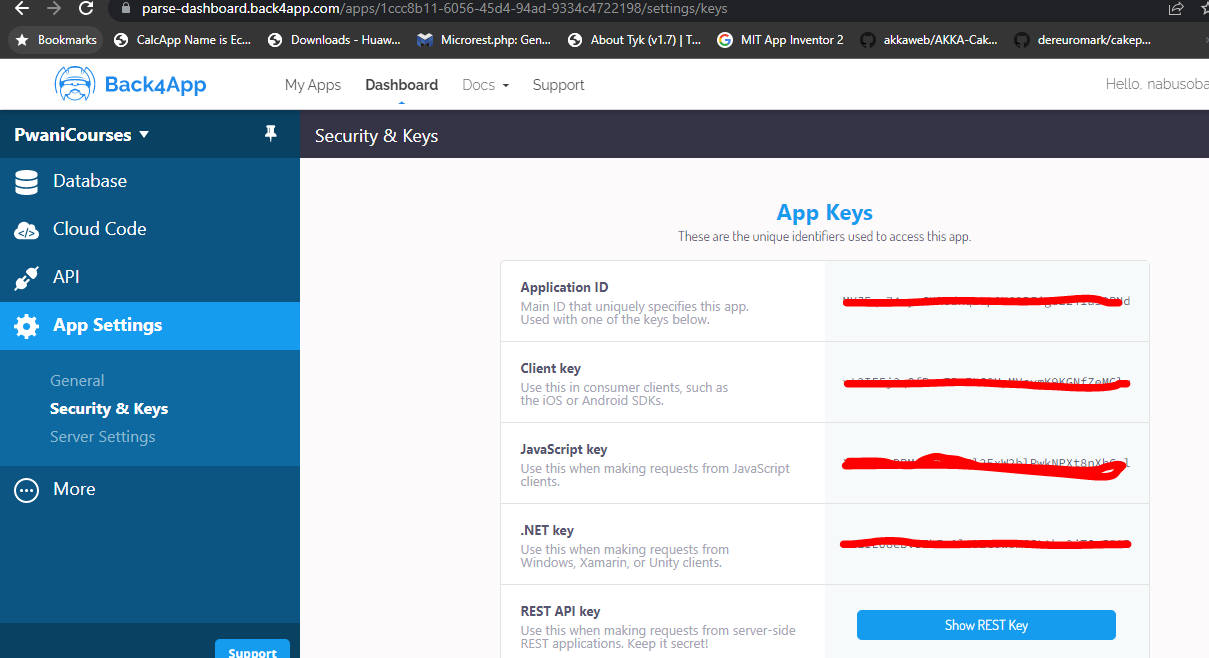
parse\_server\_sdk\_flutter: ^3.1.0

flutter\_smart\_dialog: ^3.4.0+1

flutter\_spinkit: ^5.1.0

Secondly, set up your Parse in the main.dart file. We do this by creating an account at <https://back4app.com/>, and create an app.



Go to App Settings and copy the Application ID and Client Key

At your main.dart file, here’s the code where you set up your parse (put it before runApp). Paste your application and client IDs

Future<void> main() async {

//--setting up parse initially at the main

//init widget binding for platform bridge

WidgetsFlutterBinding.ensureInitialized();

var keyParseApplicationId = "PASTE-YOUR-APPLICATION-ID";

var keyParseServerUrl = "https://parseapi.back4app.com";

var keyParseClientKey = "PASTE-YOUR-CLIENT-ID";

var store = await CoreStoreSembastImp.getInstance(password: "Pwani");

await Parse().initialize(

keyParseApplicationId,

keyParseServerUrl,

clientKey: keyParseClientKey,

autoSendSessionId: true,

coreStore: store);

//--

runApp(const MyApp());

At the controller folder where we have **auth\_provider.dart**, add the following:

import 'package:codehack\_coders\_curriculum/ui/login.dart';

import 'package:codehack\_coders\_curriculum/utils/apputils.dart';

import 'package:flutter\_smart\_dialog/flutter\_smart\_dialog.dart';

import 'package:get/get.dart';

import 'package:parse\_server\_sdk\_flutter/parse\_server\_sdk.dart';

class AuthProvider extends GetxController {

//create a map that will hold the user

var userDets = {}.obs;

//parse user

var parseUser = {}.obs;

///// You do not need that. I recommend using it just for ease of syntax.

// /// with static method: Controller.to.increment();

// /// with no static method: Get.find<Controller>().increment();

// /// There is no difference in performance, nor any side effect of using

// ///either syntax. Only one does not need the type, and the other the IDE will autocomplete it.

static AuthProvider get to => Get.find();

//When this controller is init, always check when the user has changed

@override

onReady() {

ever(parseUser, userHandler);

}

//---method to sign up user

signUpUser({username, password, emailAddress, usertype}) async {

//show loader

AppUtils.showLoading();

// use validate if empty method created to validate user inputs

if (validateIfEmpty("username", username) ||

validateIfEmpty("password", password) ||

validateIfEmpty("email", emailAddress) ||

validateIfEmpty("usertype", usertype)) {

//dismiss dialog

SmartDialog.dismiss();

// the function stops here, won't go down to create the user

return false;

}

//create a user using ParseUser, add ..set to add extra fields

var user = ParseUser(emailAddress, password, emailAddress)

..set('usertype', usertype)

..set("fullnames", username);

var response = await user.signUp();

//successful signup

try {

if (response.success) {

//get current logged in user

await getParseUser();

//stop loader

AppUtils.showSuccess("Success!");

//dismissing the loader

SmartDialog.dismiss();

return true;

}

} catch (e) {

AppUtils.showError("An error occurred, please try again later");

print(e.toString());

}

return false;

}

//---end of method to sign up user

//---method to login user

loginUser({email, password}) async {

//show loader

AppUtils.showLoading();

// use validate if empty method created to validate user inputs

if (validateIfEmpty("email", email) ||

validateIfEmpty("password", password)) {

//dismiss dialog

SmartDialog.dismiss();

// the function stops here, won't go down to create the user

return false;

}

try {

var user = ParseUser(email, password, email);

var response = await user.login();

//success login

if (response.success) {

//get current logged in user

await getParseUser();

//stop loader

AppUtils.showSuccess("Login successful!");

//dismissing the loader

SmartDialog.dismiss();

return true;

}

} catch (e) {

SmartDialog.dismiss();

AppUtils.showError("An error occurred, please try again later");

print(e.toString());

}

return false;

}

//---end of method to login user

//---forgot password

forgotPass({email}) async {

//show loader

AppUtils.showLoading();

// use validate if empty method created to validate user inputs

if (validateIfEmpty("email", email)) {

//dismiss dialog

SmartDialog.dismiss();

// the function stops here, won't go down to create the user

return false;

}

//creating user object

try {

var user = ParseUser(email, "", email);

var response = await user.requestPasswordReset();

if (response.success) {

AppUtils.showSuccess("Password reset! Check your email");

SmartDialog.dismiss();

return true;

}

} catch (e) {

AppUtils.showError("An error occurred, please try again later");

SmartDialog.dismiss();

print(e.toString());

}

return false;

}

//---end of forgot password

////function to get parse user

getParseUser() async {

var user = await ParseUser.currentUser();

parseUser.value.assignAll((user as ParseUser).toJson());

}

////Logout

logoutUser() async {

//get the user from parse user from parseUser

await getParseUser();

//logout if user in empty

if (parseUser.isNotEmpty) {

var user = await ParseUser.currentUser();

await (user as ParseUser).logout();

parseUser.value.clear();

update();

}

}

////Whenever the user is empty, go to login screen

userHandler(user) {

if (user.isEmpty) {

//removes all screen

Get.offAll(() => Login());

}

}

//----validate if empty

bool validateIfEmpty(fieldName, field) {

if (field.toString().isEmpty) {

AppUtils.showError("Field $fieldName should not be empty");

return true;

}

return false;

}

}

At the controller folder, add **main\_binding.dart** with the following code:

import 'package:codehack\_coders\_curriculum/controller/auth\_provider.dart';

import 'package:codehack\_coders\_curriculum/controller/main\_provider.dart';

import 'package:get/get.dart';

class MainBinding implements Bindings {

@override

void dependencies() {

Get.put<AuthProvider>(AuthProvider());

Get.put<MainProvider>(MainProvider());

}

}

Your AuthProvider should have the following:

| import 'package:codehack\_coders\_curriculum/ui/login.dart'; import 'package:codehack\_coders\_curriculum/utils/apputils.dart'; import 'package:flutter\_smart\_dialog/flutter\_smart\_dialog.dart'; import 'package:get/get.dart'; import 'package:parse\_server\_sdk\_flutter/parse\_server\_sdk.dart';  class AuthProvider extends GetxController {  //create a map that will hold the user  var userDets = {}.obs;  //parse user  var parseUser = {}.obs;   ///// You do not need that. I recommend using it just for ease of syntax.  // /// with static method: Controller.to.increment();  // /// with no static method: Get.find<Controller>().increment();  // /// There is no difference in performance, nor any side effect of using  // ///either syntax. Only one does not need the type, and the other the IDE will autocomplete it.  static AuthProvider get to => Get.find();   //When this controller is init, always check when the user has changed  @override  onReady() {  ever(parseUser, userHandler);  }   //---method to sign up user  signUpUser({username, password, emailAddress, usertype}) async {  //show loader  AppUtils.showLoading();   // use validate if empty method created to validate user inputs  if (validateIfEmpty("username", username) ||  validateIfEmpty("password", password) ||  validateIfEmpty("email", emailAddress) ||  validateIfEmpty("usertype", usertype)) {  //dismiss dialog  SmartDialog.dismiss();  // the function stops here, won't go down to create the user  return false;  }  //create a user using ParseUser, add ..set to add extra fields  var user = ParseUser(emailAddress, password, emailAddress)  ..set('usertype', usertype)  ..set("fullnames", username);   var response = await user.signUp();  //successful signup  try {  if (response.success) {  //get current logged in user  await getParseUser();  //stop loader  AppUtils.showSuccess("Success!");  //dismissing the loader  SmartDialog.dismiss();  return true;  }  } catch (e) {  AppUtils.showError("An error occurred, please try again later");  print(e.toString());  }  return false;  }  //---end of method to sign up user   //---method to login user  loginUser({email, password}) async {  //show loader  AppUtils.showLoading();  // use validate if empty method created to validate user inputs  if (validateIfEmpty("email", email) ||  validateIfEmpty("password", password)) {  //dismiss dialog  SmartDialog.dismiss();  // the function stops here, won't go down to create the user  return false;  }  try {  var user = ParseUser(email, password, email);  var response = await user.login();   //success login   if (response.success) {  //get current logged in user  await getParseUser();  //stop loader  AppUtils.showSuccess("Login successful!");  //dismissing the loader  SmartDialog.dismiss();  return true;  }  } catch (e) {  SmartDialog.dismiss();  AppUtils.showError("An error occurred, please try again later");  print(e.toString());  }  return false;  }  //---end of method to login user   //---forgot password  forgotPass({email}) async {  //show loader  AppUtils.showLoading();  // use validate if empty method created to validate user inputs  if (validateIfEmpty("email", email)) {  //dismiss dialog  SmartDialog.dismiss();  // the function stops here, won't go down to create the user  return false;  }   //creating user object  try {  var user = ParseUser(email, "", email);  var response = await user.requestPasswordReset();  if (response.success) {  AppUtils.showSuccess("Password reset! Check your email");  SmartDialog.dismiss();  return true;  }  } catch (e) {  AppUtils.showError("An error occurred, please try again later");  SmartDialog.dismiss();  print(e.toString());  }  return false;  }  //---end of forgot password   ////function to get parse user  getParseUser() async {  var user = await ParseUser.currentUser();  parseUser.value.assignAll((user as ParseUser).toJson());  }   ////Logout  logoutUser() async {  //get the user from parse user from parseUser  await getParseUser();  //logout if user in empty  if (parseUser.isNotEmpty) {  var user = await ParseUser.currentUser();  await (user as ParseUser).logout();  parseUser.value.clear();  update();  }  }   ////Whenever the user is empty, go to login screen  userHandler(user) {  if (user.isEmpty) {  //removes all screen  Get.offAll(() => Login());  }  }   //----validate if empty  bool validateIfEmpty(fieldName, field) {  if (field.toString().isEmpty) {  AppUtils.showError("Field $fieldName should not be empty");  return true;  }  return false;  } } |
| --- |

Finally, make sure your signup.dart code is as follows:

| import 'package:codehack\_coders\_curriculum/ui/login.dart'; import 'package:codehack\_coders\_curriculum/utils/apputils.dart'; import 'package:flutter\_smart\_dialog/flutter\_smart\_dialog.dart'; import 'package:get/get.dart'; import 'package:parse\_server\_sdk\_flutter/parse\_server\_sdk.dart';  class AuthProvider extends GetxController {  //create a map that will hold the user  var userDets = {}.obs;  //parse user  var parseUser = {}.obs;   ///// You do not need that. I recommend using it just for ease of syntax.  // /// with static method: Controller.to.increment();  // /// with no static method: Get.find<Controller>().increment();  // /// There is no difference in performance, nor any side effect of using  // ///either syntax. Only one does not need the type, and the other the IDE will autocomplete it.  static AuthProvider get to => Get.find();   //When this controller is init, always check when the user has changed  @override  onReady() {  ever(parseUser, userHandler);  }   //---method to sign up user  signUpUser({username, password, emailAddress, usertype}) async {  //show loader  AppUtils.showLoading();   // use validate if empty method created to validate user inputs  if (validateIfEmpty("username", username) ||  validateIfEmpty("password", password) ||  validateIfEmpty("email", emailAddress) ||  validateIfEmpty("usertype", usertype)) {  //dismiss dialog  SmartDialog.dismiss();  // the function stops here, won't go down to create the user  return false;  }  //create a user using ParseUser, add ..set to add extra fields  var user = ParseUser(emailAddress, password, emailAddress)  ..set('usertype', usertype)  ..set("fullnames", username);   var response = await user.signUp();  //successful signup  try {  if (response.success) {  //get current logged in user  await getParseUser();  //stop loader  AppUtils.showSuccess("Success!");  //dismissing the loader  SmartDialog.dismiss();  return true;  }  } catch (e) {  AppUtils.showError("An error occurred, please try again later");  print(e.toString());  }  return false;  }  //---end of method to sign up user   //---method to login user  loginUser({email, password}) async {  //show loader  AppUtils.showLoading();  // use validate if empty method created to validate user inputs  if (validateIfEmpty("email", email) ||  validateIfEmpty("password", password)) {  //dismiss dialog  SmartDialog.dismiss();  // the function stops here, won't go down to create the user  return false;  }  try {  var user = ParseUser(email, password, email);  var response = await user.login();   //success login   if (response.success) {  //get current logged in user  await getParseUser();  //stop loader  AppUtils.showSuccess("Login successful!");  //dismissing the loader  SmartDialog.dismiss();  return true;  }  } catch (e) {  SmartDialog.dismiss();  AppUtils.showError("An error occurred, please try again later");  print(e.toString());  }  return false;  }  //---end of method to login user   //---forgot password  forgotPass({email}) async {  //show loader  AppUtils.showLoading();  // use validate if empty method created to validate user inputs  if (validateIfEmpty("email", email)) {  //dismiss dialog  SmartDialog.dismiss();  // the function stops here, won't go down to create the user  return false;  }   //creating user object  try {  var user = ParseUser(email, "", email);  var response = await user.requestPasswordReset();  if (response.success) {  AppUtils.showSuccess("Password reset! Check your email");  SmartDialog.dismiss();  return true;  }  } catch (e) {  AppUtils.showError("An error occurred, please try again later");  SmartDialog.dismiss();  print(e.toString());  }  return false;  }  //---end of forgot password   ////function to get parse user  getParseUser() async {  var user = await ParseUser.currentUser();  parseUser.value.assignAll((user as ParseUser).toJson());  }   ////Logout  logoutUser() async {  //get the user from parse user from parseUser  await getParseUser();  //logout if user in empty  if (parseUser.isNotEmpty) {  var user = await ParseUser.currentUser();  await (user as ParseUser).logout();  parseUser.value.clear();  update();  }  }   ////Whenever the user is empty, go to login screen  userHandler(user) {  if (user.isEmpty) {  //removes all screen  Get.offAll(() => Login());  }  }   //----validate if empty  bool validateIfEmpty(fieldName, field) {  if (field.toString().isEmpty) {  AppUtils.showError("Field $fieldName should not be empty");  return true;  }  return false;  } } |
| --- |

# EXTRA: Push Notifications

# EXTRA: APIs with Flutter

# EXTRA: Monetization with Pesapal

# What Next?