The purpose of my application '**Show Up'** is to fulfill requirements of my client so that he can create his best portfolio for his photography, on his mobile. Further, dart-Flutter language is used to build this application which contains a total number of 11 classes that extends a custom widget (either stateful or stateless) pre-built in flutter. These widgets help programmer to maintain the state of classes when they are updated. Hence, every class for each screen extends this widget and are separately build in different dart file.

[ADD WHAT LANGUAGE YOU HAVE USED, IS IT CLASS BASED HOW MANY CLASSES WERE THERE ETC, SOME INTRO OF LANGUAGE COMBINATION]

The welcome screen of my app shows a logo (fig.1.1.1) with three main objectives i.e. innovate, create and captivate to help user understand the objectives of the application. The shades of blue are only used as per the demand of our client.

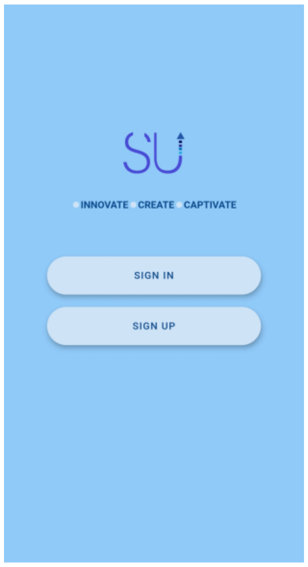
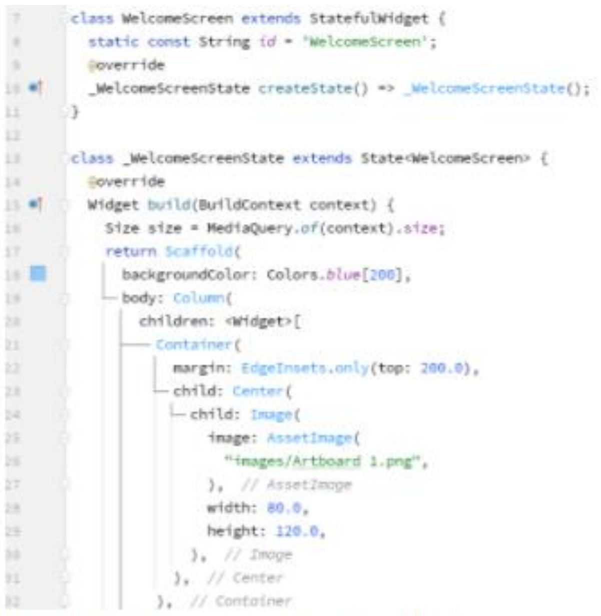


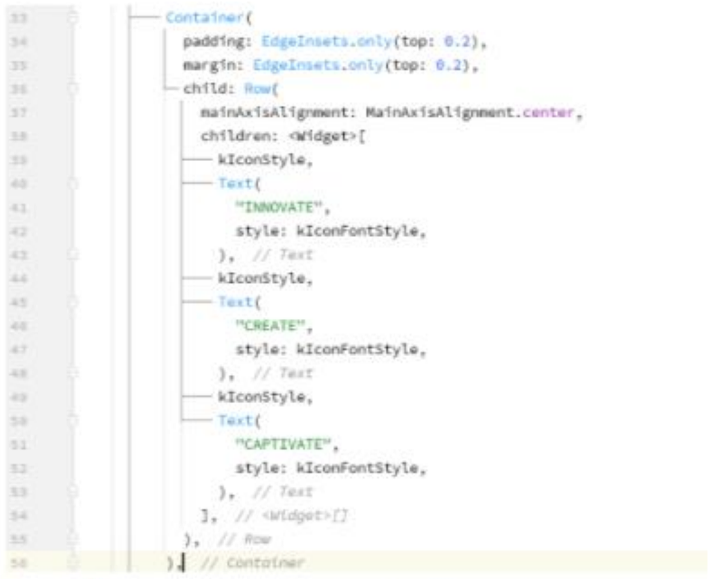
Fig.1.1.1. welcome screen

The coding in (fig.1.1.2 and fig1.1.3) shows how welcome screen is given a stateful widget so that the state of application can be updated for new changes (e.g. While navigating to a different screen when the user clicks the buttons). The application is wrapped in a scaffold to display the screen on the entire device. Here, one of scaffold properties is used to give a consistent background color on the entire screen. Then the body of the widget scaffold is given a column widget to show every widget in a horizontal direction. The image and button widgets are wrapped inside the container to give a balance padding. Inside the image widget container, Asset image is used to show images that are imported in the flutter images package.



```
class WelcomeScreen extends StatefulWidget {
  static const String id = 'WelcomeScreen';
  @override
  _WelcomeScreenState createState() => _WelcomeScreenState();
}

class _WelcomeScreenState extends State<WelcomeScreen> {
  @override
  Widget build(BuildContext context) {
    Size size = MediaQuery.of(context).size;
    return Scaffold(
      backgroundColor: Colors.blue[200],
      body: Column(
        children: <Widget>[
          Container(
            margin: EdgeInsets.only(top: 200.0),
            child: Center(
              child: Image(
                image: AssetImage(
                  "images/Artboard 1.png",
                ), // AssetImage
                width: 80.0,
                height: 120.0,
              ), // Image
            ), // Center
          ), // Container
```

Fig.1.1.2 code while uploading the logo



```
          Container(
            padding: EdgeInsets.only(top: 0.2),
            margin: EdgeInsets.only(top: 0.2),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                kIconStyle,
                Text(
                  "INNOVATE",
                  style: kIconFontStyle,
                ), // Text
                kIconStyle,
                Text(
                  "CREATE",
                  style: kIconFontStyle,
                ), // Text
                kIconStyle,
                Text(
                  "CAPTIVATE",
                  style: kIconFontStyle,
                ), // Text
              ], // <Widget>[]
            ), // Row
          ), // Container
```

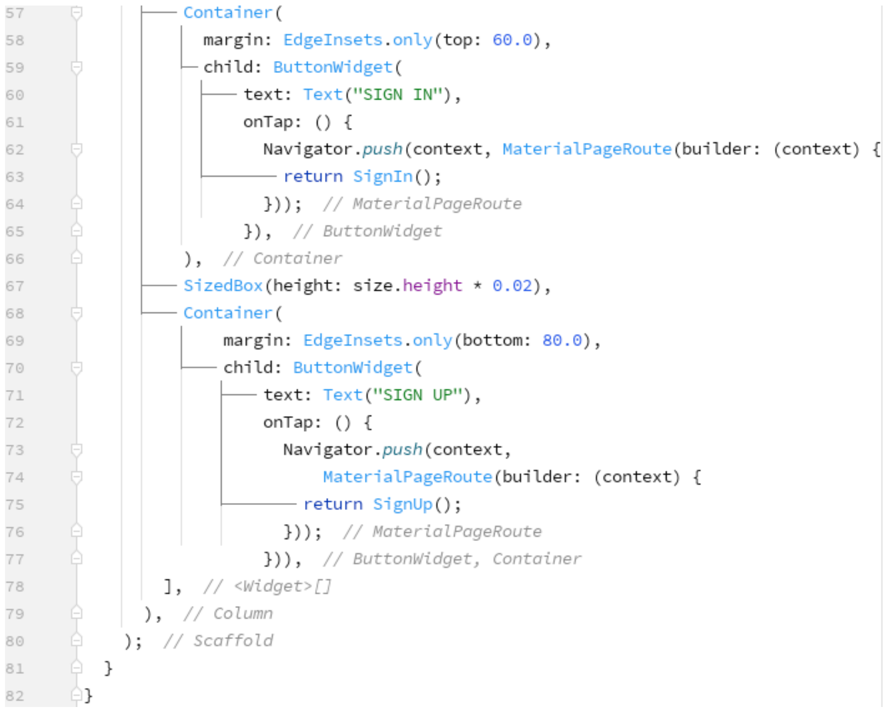
Fig.1.1.3 code for writing the logo objectives

Registration Screen:

This screen presents two buttons for users to join the application (Fig1.1.4 and 1.1.5). The sign in button will lead them to a different screen for joining in again if they have already joined the app. whereas, the sign up button will navigate users to a different screen to register themselves in the application.

Fig.1.1.4 Sign In Button

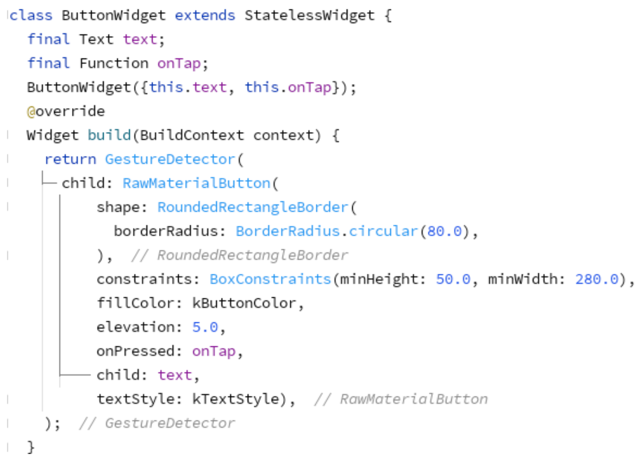
Fig.1.1.5. Sign Up Button



```
        Container(
          margin: EdgeInsets.only(top: 60.0),
          child: ButtonWidget(
            text: Text("SIGN IN"),
            onTap: () {
              Navigator.push(context, MaterialPageRoute(builder: (context) {
                return SignIn();
              })); // MaterialPageRoute
            }), // ButtonWidget
        ), // Container
        SizedBox(height: size.height * 0.02),
        Container(
            margin: EdgeInsets.only(bottom: 80.0),
            child: ButtonWidget(
              text: Text("SIGN UP"),
              onTap: () {
                Navigator.push(context,
                    MaterialPageRoute(builder: (context) {
                  return SignUp();
                })); // MaterialPageRoute
              })), // ButtonWidget, Container
      ], // <Widget>[]
    ), // Column
  ); // Scaffold
 }
}
```

Fig.1.1.6 coding for Sign-in and Sign up buttons; navigating it to their screens.

F.g.1.1.6 shows the code view of designing for both buttons. A custom button widget is made so that it can be followed throughout the application. Below fig.1.1.7 shows the coding of button widget that is created with balanced width, height and elevation. The button is wrapped in a gesture detector to detect user actions on button quickly. **[thora sa explain the codde like name is set and dimension etc]**



```
class ButtonWidget extends StatelessWidget {
  final Text text;
  final Function onTap;
  ButtonWidget({this.text, this.onTap});
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      child: RawMaterialButton(
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(80.0),
          ),  // RoundedRectangleBorder
          constraints: BoxConstraints(minHeight: 50.0, minWidth: 280.0),
          fillColor: kButtonColor,
          elevation: 5.0,
          onPressed: onTap,
          child: text,
          textStyle: kTextStyle),  // RawMaterialButton
    );  // GestureDetector
  }
```

Fig.1.1.7 Button Widget

The sign-in screen shows two text fields with the consistent logo on the top and same color in the background (fig.1.1.8). Fig.1.1.9 shows the coding of this screen. All the text fields are wrapped inside a form widget so that they can be validated and grouped using a single key. The body is also wrapped in a modal progress Hud package to show a circular icon while loading a class or a widget. Gesture detectors are used to detect user gestures in the text field widgets and buttons. Moreover, the screen is wrapped up with a child scroll view to create an ease for the user to scroll up to another text field while having their keyboard opened.

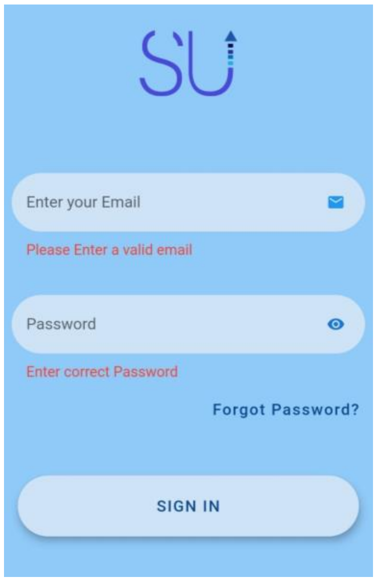
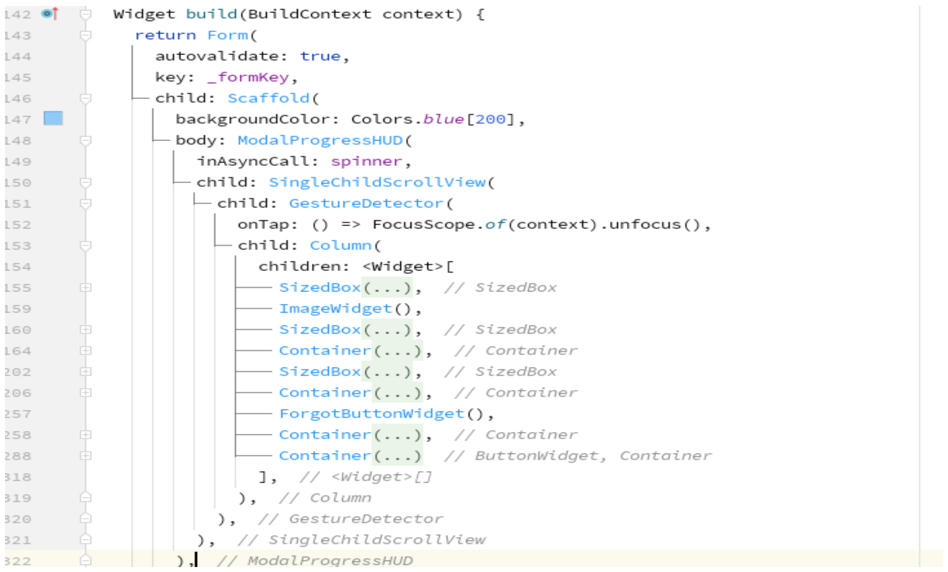


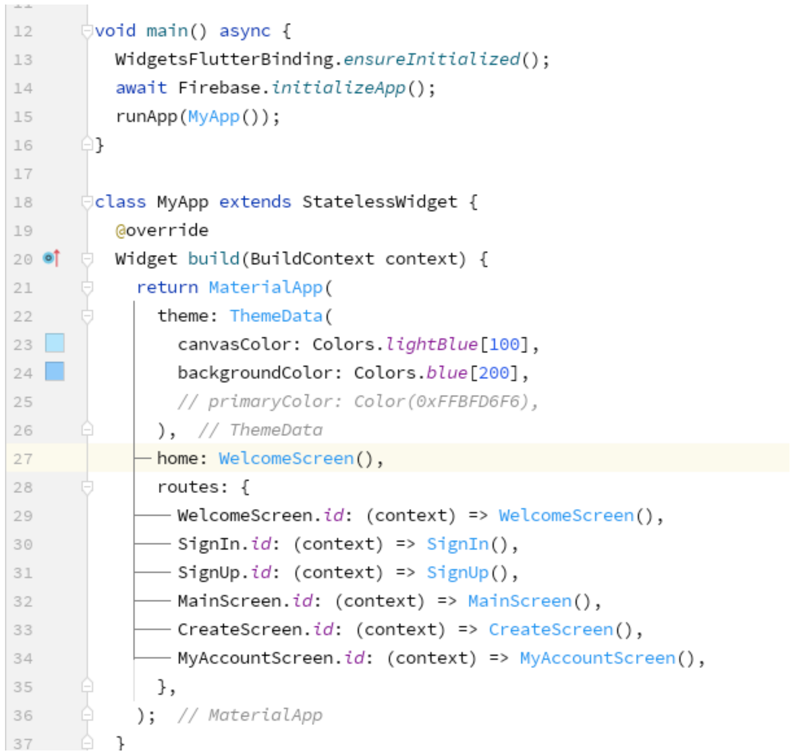
Fig.1.1.8 Sign-in Screen view



```
Widget build(BuildContext context) {
  return Form(
    autovalidate: true,
    key: _formKey,
    child: Scaffold(
      backgroundColor: Colors.blue[200],
      body: ModalProgressHUD(
        inAsyncCall: spinner,
        child: SingleChildScrollView(
          child: GestureDetector(
            onTap: () => FocusScope.of(context).unfocus(),
            child: Column(
              children: <Widget>[
                SizedBox(...),  // SizedBox
                ImageWidget(),
                SizedBox(...),  // SizedBox
                Container(...),  // Container
                SizedBox(...),  // SizedBox
                Container(...),  // Container
                ForgotButtonWidget(),
                Container(...),  // Container
                Container(...)  // ButtonWidget, Container
              ],  // <Widget>[]
            ),  // Column
          ),  // GestureDetector
        ),  // SingleChildScrollView
      ),  // ModalProgressHUD
```

Fig.1.1.9 code(summarized) of Sign- in Screen

In Addition, to maintain the same background color in every screen without having its repetitive coding; in the **main.dart** (fig.1.2.0) of application theme data is used, a custom class in material app widget provided by flutter that helps the user to set the background color which will be used throughout the application on every screen.



```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        canvasColor: Colors.lightBlue[100],
        backgroundColor: Colors.blue[200],
        // primaryColor: Color(0xFFBFD6F6),
      ),  // ThemeData
      home: WelcomeScreen(),
      routes: {
        WelcomeScreen.id: (context) => WelcomeScreen(),
        SignIn.id: (context) => SignIn(),
        SignUp.id: (context) => SignUp(),
        MainScreen.id: (context) => MainScreen(),
        CreateScreen.id: (context) => CreateScreen(),
        MyAccountScreen.id: (context) => MyAccountScreen(),
      },
    );  // MaterialApp
  }
```

Fig.1.2.0 main.dart

For the design of the text field, it is given a rounded corner with a balance radius and also has an email icon to make it look more structured and at the same time deliver the message towards the user to enter their email. The field is given a hint text for the user to enter the valid address they previously entered while signing up(fig1.2.1).Furthermore, while the user enters the data on the text field, the user's keyboard will change to show '@' to help them easily write their address.(fig1.2.2)

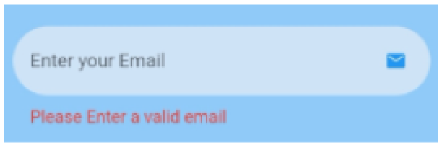


Fig.1.2.1 Email Text Field

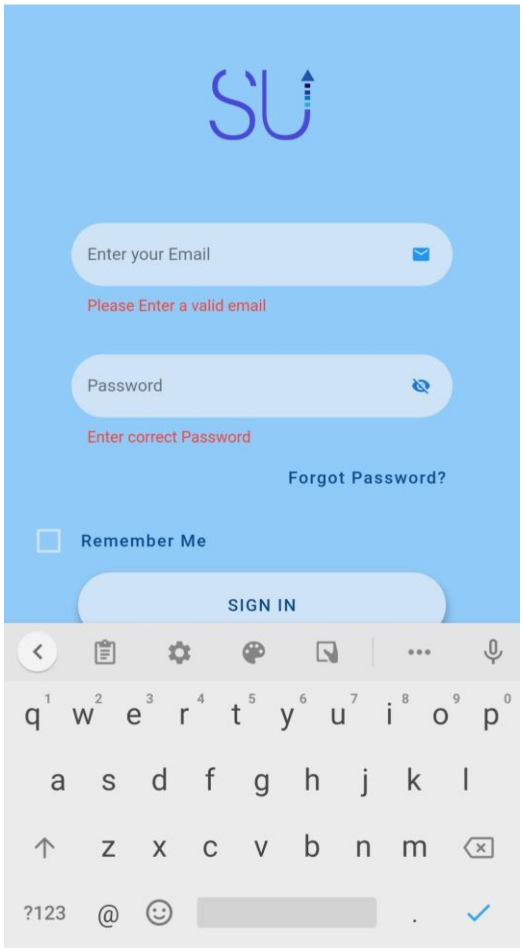
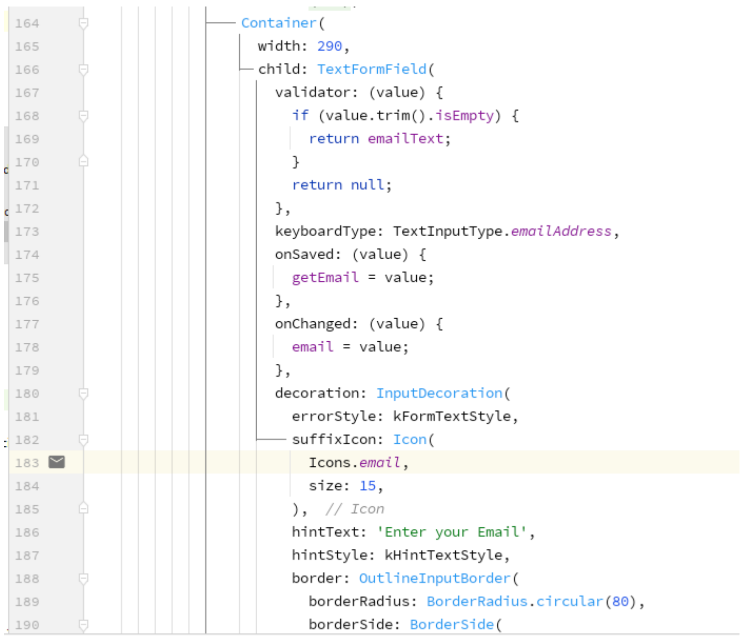


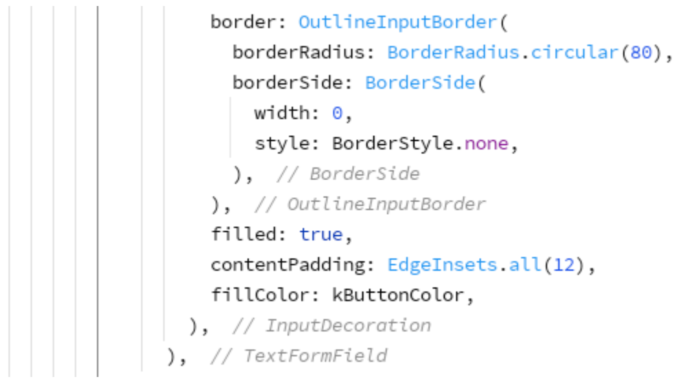
Fig.1.2.2 Showing '@' in the keyboard for only email text field

Fig. (1.2.3 and 1.2.4) shows how the user input value is saved in the text field.



```
Container(
  width: 290,
  child: TextFormField(
    validator: (value) {
      if (value.trim().isEmpty) {
        return emailText;
      }
      return null;
    },
    keyboardType: TextInputType.emailAddress,
    onSaved: (value) {
      getEmail = value;
    },
    onChanged: (value) {
      email = value;
    },
    decoration: InputDecoration(
      errorStyle: kFormTextStyle,
      suffixIcon: Icon(
        Icons.email,
        size: 15,
      ), // Icon
      hintText: 'Enter your Email',
      hintStyle: kHintTextStyle,
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(80),
        borderSide: BorderSide(
```

Fig.1.2.3 coding for email Text Field



```
border: OutlineInputBorder(
  borderRadius: BorderRadius.circular(80),
  borderSide: BorderSide(
    width: 0,
    style: BorderStyle.none,
  ), // BorderSide
), // OutlineInputBorder
filled: true,
contentPadding: EdgeInsets.all(12),
fillColor: kButtonColor,
), // InputDecoration
), // TextFormField
```

Fig.1.2.4. Coding for email text field

Below the email text field, there is another text field for users to enter their password(fig1.2.5). The hint text is written again to help users understand what they have to enter in this field. Then, the icon in the right end of the field shows an eye and while clicking on that icon it changes to another icon with '/ 'on the eye. This icon is created to help users either to show or hide their password. Moreover, (fig1.2.7) shows that clicking on this icon hides their password and the text field will also be converted into showing each alphabet with a dot. Furthermore, the text field is functionalized with the error text for the user to enter a valid password provided previously by the user. Also, (fig.1.2.6) shows how these fields contain an 'on saved' functionality so that the entered data by the user will be saved in the one field when they switch to another text field. Also, the on Changed functionality of the text field is used to handle changes in text fields.

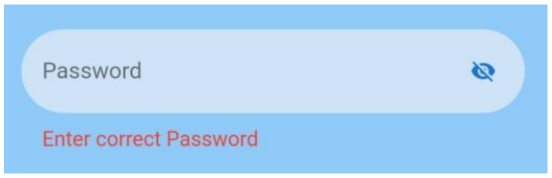
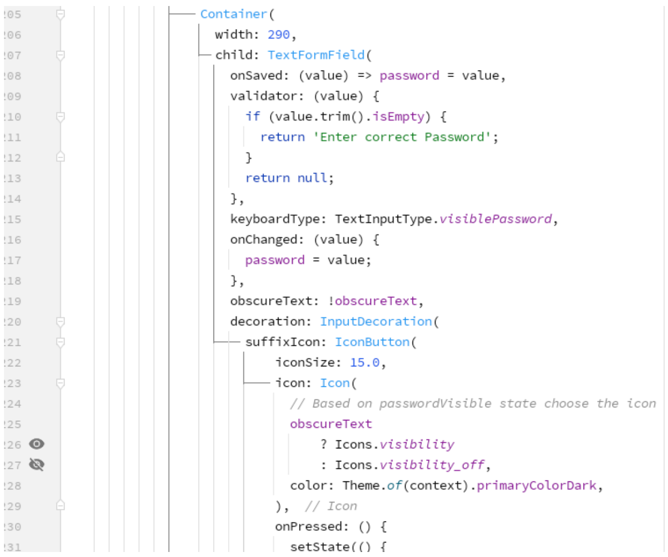
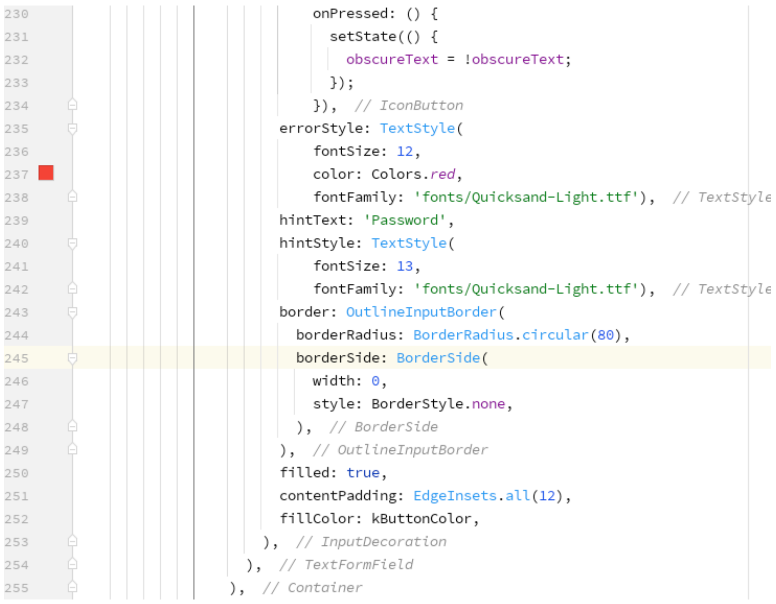


Fig.1.2.5 Password Text field



```
Container(
  width: 290,
  child: TextFormField(
    onSaved: (value) => password = value,
    validator: (value) {
      if (value.trim().isEmpty) {
        return 'Enter correct Password';
      }
      return null;
    },
    keyboardType: TextInputType.visiblePassword,
    onChanged: (value) {
      password = value;
    },
    obscureText: !obscureText,
    decoration: InputDecoration(
      suffixIcon: IconButton(
        iconSize: 15.0,
        icon: Icon(
          // Based on passwordVisible state choose the icon
          obscureText
              ? Icons.visibility
              : Icons.visibility_off,
          color: Theme.of(context).primaryColorDark,
        ), // Icon
        onPressed: () {
          setState(() {
```

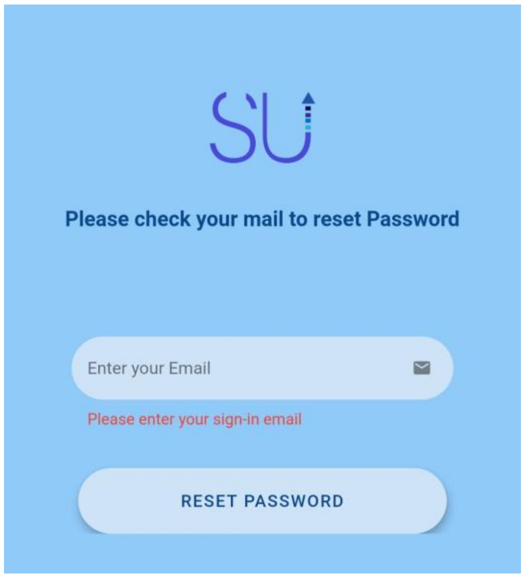
Fig.1.2.6 coding for Password Text Field



```
        onPressed: () {
          setState(() {
            obscureText = !obscureText;
          });
        }), // IconButton
      errorStyle: TextStyle(
          fontSize: 12,
          color: Colors.red,
          fontFamily: 'fonts/Quicksand-Light.ttf'), // TextStyle
      hintText: 'Password',
      hintStyle: TextStyle(
          fontSize: 13,
          fontFamily: 'fonts/Quicksand-Light.ttf'), // TextStyle
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(80),
        borderSide: BorderSide(
          width: 0,
          style: BorderStyle.none,
        ), // BorderSide
      ), // OutlineInputBorder
      filled: true,
      contentPadding: EdgeInsets.all(12),
      fillColor: kButtonColor,
    ), // InputDecoration
  ), // TextFormField
), // Container
```

Fig.1.2.7 coding for Password Text Field

Afterwards, below the text fields in the right corner of the screen, marginalized with the text fields, a text is written ‘forgot password?’. The user will click on the text if they didn't remember the password they entered lastly and this text will navigate them to another screen. The screen will again open up with the logo and same background color (fig.1.2.8). Moreover, in the forgot button screen there is a text field with a hint text to enter the sign-in email address. The text above the field enables the user to check their inbox; after pressing the reset password button. The below fig. (1.2.9) shows the no-reply email user will receive. It will ask the user to click on the provided link to reset their password. The fig. (1.3.0) shows the screen that will appear when the user clicks on the link.


Please check your mail to reset Password

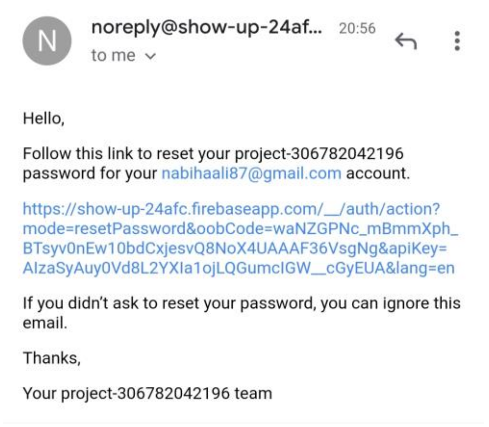
Enter your Email

Please enter your sign-in email

Fig.1.2.8 Forgot Password Screen

Fig.1.2.9 Email to reset Password

Fig.1.3.0 Reset- Screen when user clicks on link


noreply@show-up-24af... 20:56

to me

Hello,

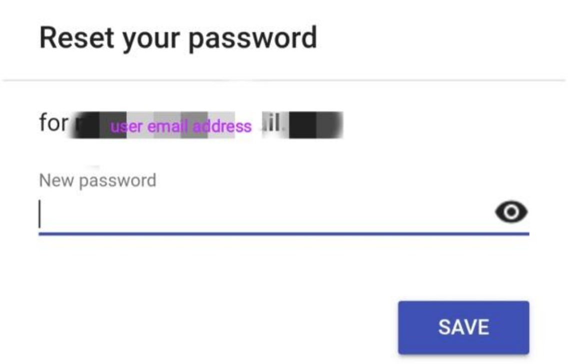
Follow this link to reset your project-306782042196 password for your nabihaali87@gmail.com account.

https://show-up-24afc.firebaseapp.com/__/auth/action?mode=resetPassword&oobCode=waNZGPNc_mBmmXph_BTsyv0nEw10bdCxjesvQ8NoX4UAAAF36VsgNg&apiKey=AIzaSyAuy0Vd8L2YXla1ojLQGumclGW__cGyEUA&lang=en

If you didn’t ask to reset your password, you can ignore this email.

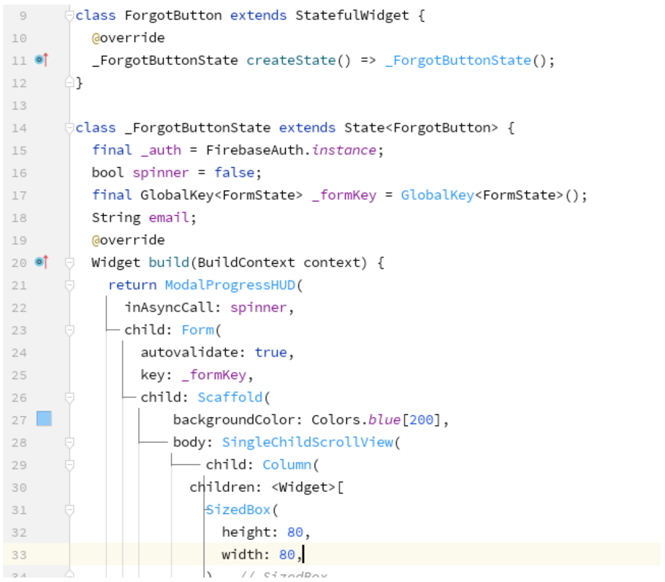
Thanks,

Your project-306782042196 team


Reset your password

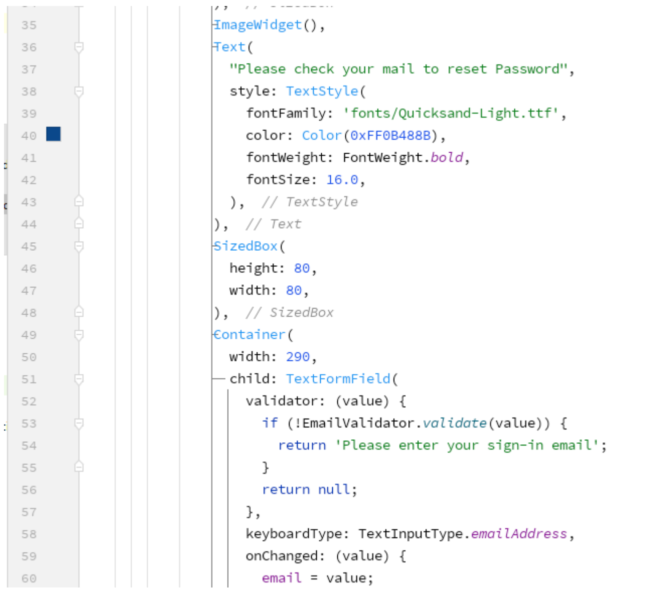
for [redacted] user email address [redacted]

New password



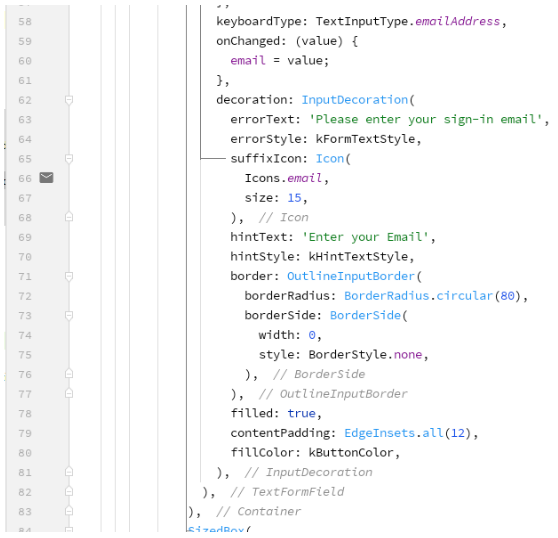
```
class ForgotButton extends StatefulWidget {
  @override
  _ForgotButtonState createState() => _ForgotButtonState();
}

class _ForgotButtonState extends State<ForgotButton> {
  final _auth = FirebaseAuth.instance;
  bool spinner = false;
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  String email;
  @override
  Widget build(BuildContext context) {
    return ModalProgressHUD(
      inAsyncCall: spinner,
      child: Form(
        autovalidate: true,
        key: _formKey,
        child: Scaffold(
            backgroundColor: Colors.blue[200],
            body: SingleChildScrollView(
                child: Column(
              children: <Widget>[
                SizedBox(
                  height: 80,
                  width: 80,
                ), // SizedBox
```

Fig.1.3.1 Coding for forgot Password button



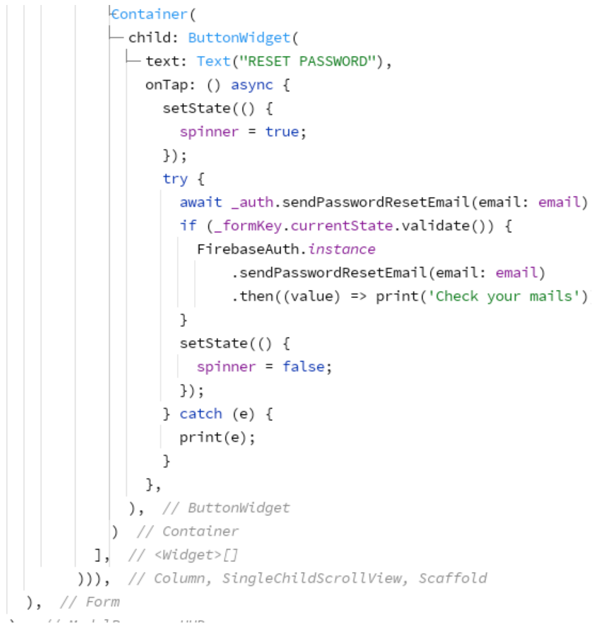
```
                ImageWidget(),
                Text(
                  "Please check your mail to reset Password",
                  style: TextStyle(
                    fontFamily: 'fonts/Quicksand-Light.ttf',
                    color: Color(0xFF0B488B),
                    fontWeight: FontWeight.bold,
                    fontSize: 16.0,
                  ), // TextStyle
                ), // Text
                SizedBox(
                  height: 80,
                  width: 80,
                ), // SizedBox
                Container(
                  width: 290,
                  child: TextFormField(
                    validator: (value) {
                      if (!EmailValidator.validate(value)) {
                        return 'Please enter your sign-in email';
                      }
                      return null;
                    },
                    keyboardType: TextInputType.emailAddress,
                    onChanged: (value) {
                      email = value;
```

Fig.1.3.2 Coding for forgot Password Button



```
    keyboardType: TextInputType.emailAddress,
    onChanged: (value) {
      email = value;
    },
    decoration: InputDecoration(
      errorText: 'Please enter your sign-in email',
      errorStyle: kFormTextStyle,
      suffixIcon: Icon(
        Icons.email,
        size: 15,
      ), // Icon
      hintText: 'Enter your Email',
      hintStyle: kHintTextStyle,
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(80),
        borderSide: BorderSide(
          width: 0,
          style: BorderStyle.none,
        ), // BorderSide
      ), // OutlineInputBorder
      filled: true,
      contentPadding: EdgeInsets.all(12),
      fillColor: kButtonColor,
    ), // InputDecoration
  ), // TextFormField
), // Container
SizedBox(
```

Fig.1.3.3 Coding for forgot Password Button



```
          Container(
            child: ButtonWidget(
              text: Text("RESET PASSWORD"),
              onTap: () async {
                setState(() {
                  spinner = true;
                });
                try {
                  await _auth.sendPasswordResetEmail(email: email)
                  if (_formKey.currentState.validate()) {
                    FirebaseAuth.instance
                        .sendPasswordResetEmail(email: email)
                        .then((value) => print('Check your mails'))
                  }
                  setState(() {
                    spinner = false;
                  });
                } catch (e) {
                  print(e);
                }
              },
            ), // ButtonWidget
          ) // Container
        ], // <Widget>[]
      ))), // Column, SingleChildScrollView, Scaffold
), // Form
```

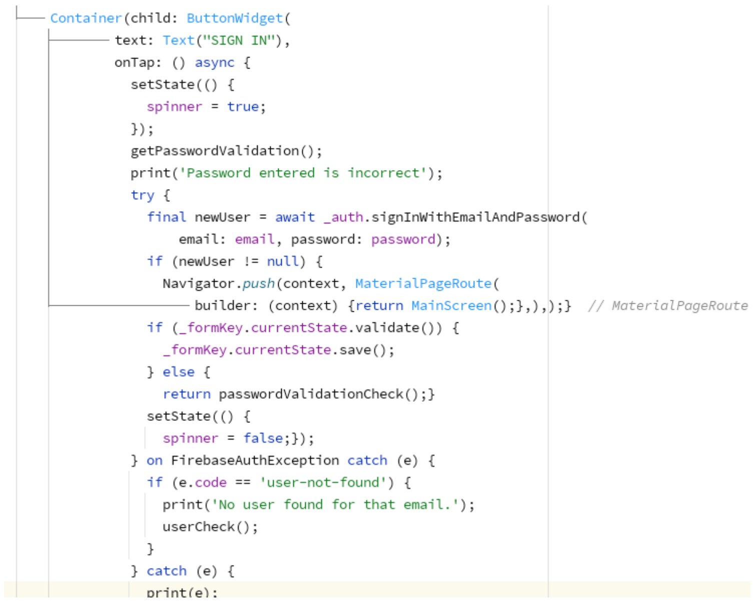
Fig.1.3.4 Coding for forgot Password Button

Fig.1.3.1 ,1.3.2 and 1.3.3 shows how the forgot screen is designed with a logo, a text and a text field. Fig 1.3.4 shows how the button is linked with firebase authentication which authorizes the user’s email address and then sends the email.

Lastly, the user will click on the sign-in button (Fig.1.3.5). This button will not only navigate the user to the home screen but it will also authenticate the entered data in the firebase console (firebase is used as a database storage for this flutter app) of the application.

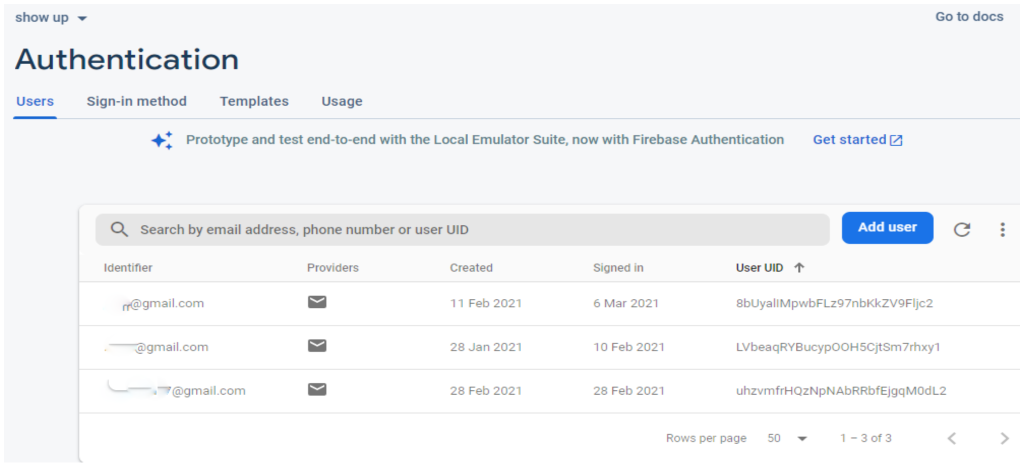
Fig.1.3.5 Sign In Button

In Fig.1.3.6, shows Sign In button coding when a user has entered required text in the fields and is ready to authenticate its data and join the application. Fig.1.3.7 shows when firebase authenticates user data by saving their email address.



```
Container(child: ButtonWidget(
        text: Text("SIGN IN"),
        onTap: () async {
          setState(() {
            spinner = true;
          });
          getPasswordValidation();
          print('Password entered is incorrect');
          try {
            final newUser = await _auth.signInWithEmailAndPassword(
                email: email, password: password);
            if (newUser != null) {
              Navigator.push(context, MaterialPageRoute(
                builder: (context) {return MainScreen();},),);} // MaterialPageRoute
            if (_formKey.currentState.validate()) {
              _formKey.currentState.save();
            } else {
              return passwordValidationCheck();}
            setState(() {
              spinner = false;});
          } on FirebaseAuthException catch (e) {
            if (e.code == 'user-not-found') {
              print('No user found for that email.');
              userCheck();
            }
          } catch (e) {
            print(e);
```

Fig.1.3.6 Sign In Button coding



Authentication

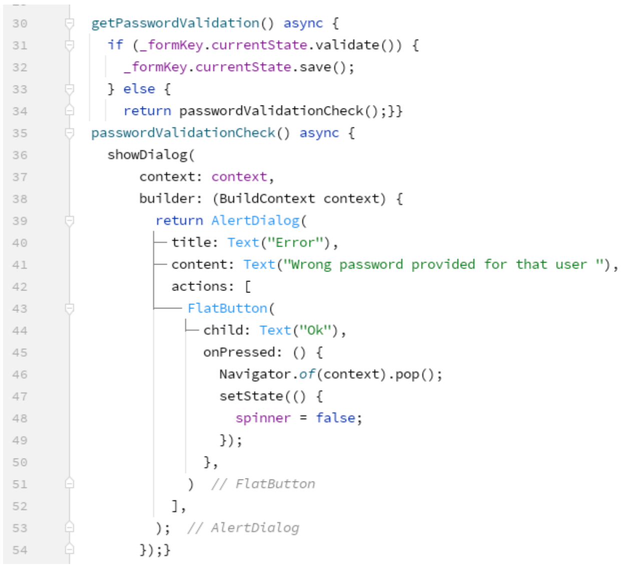
Prototype and test end-to-end with the Local Emulator Suite, now with Firebase Authentication Get started

Identifier	Providers	Created	Signed in	User UID
[redacted]m@gmail.com	✉	11 Feb 2021	6 Mar 2021	8bUyalIMpwbFLz97nbKkZV9Fljc2
[redacted]@gmail.com	✉	28 Jan 2021	10 Feb 2021	LVbeaqRYBucypOOH5CjtSm7rhxy1
[redacted]7@gmail.com	✉	28 Feb 2021	28 Feb 2021	uhzvmfrHQzNpNAbRRbfEjgqM0dL2

Rows per page 50 1 – 3 of 3

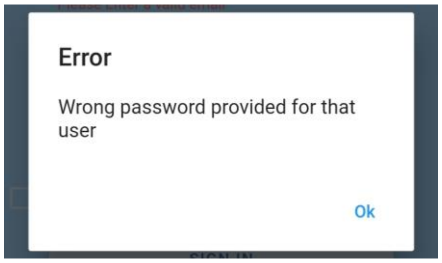
Fig.1.3.7 User- Signing Authentication stored in Firebase database authentication

Fig.1.3.8 shows the process of thinking recursively i.e. password is validated by checking the current state of the text field where password is entered correctly for that specific email or not. fig.1.3.9 shows an alert box if the user entered the wrong password for the provided email.



```
getPasswordValidation() async {
  if (_formKey.currentState.validate()) {
    _formKey.currentState.save();
  } else {
    return passwordValidationCheck();}}
passwordValidationCheck() async {
  showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: Text("Error"),
          content: Text("Wrong password provided for that user "),
          actions: [
            FlatButton(
              child: Text("Ok"),
              onPressed: () {
                Navigator.of(context).pop();
                setState(() {
                  spinner = false;
                });
              },
            )  // FlatButton
          ],
        );  // AlertDialog
      });}
```

Fig.1.3.8 coding for password Validation function



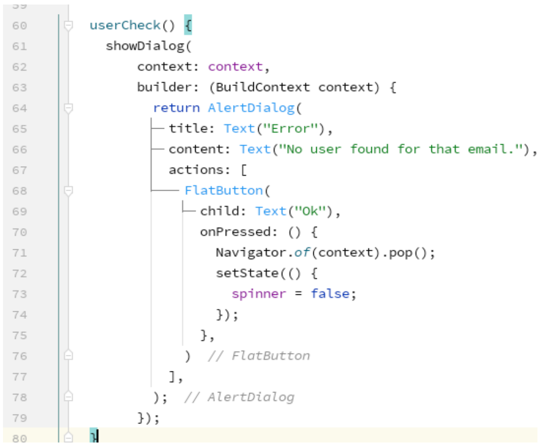
Error

Wrong password provided for that user

Ok

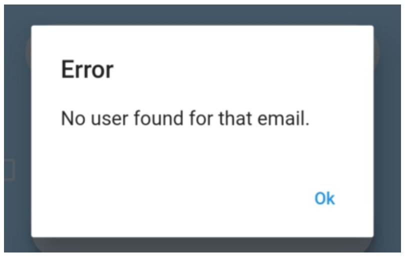
Fig.1.3.9 The error on screen

Fig.1.4.0 shows the function written in fig.1.4.1 coding where the user entered email is verified to show an error if it's correct or not.



```
userCheck() {
  showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: Text("Error"),
          content: Text("No user found for that email."),
          actions: [
            FlatButton(
              child: Text("Ok"),
              onPressed: () {
                Navigator.of(context).pop();
                setState(() {
                  spinner = false;
                });
              },
            )  // FlatButton
          ],
        );  // AlertDialog
      });
}
```

Fig.1.4.0 user email verification


Error

No user found for that email.

Fig.1.4.1 shows on screen error

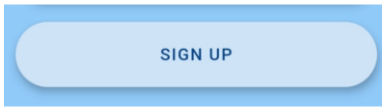
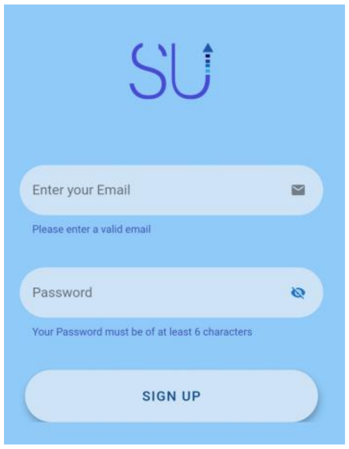


Fig.1.4.2 Sign Up Button

On the other hand, the user clicking on the sign-up button(fig.1.4.2) will navigate them to another screen with persistent design of logo, background color and text fields(Fig.1.4.3).



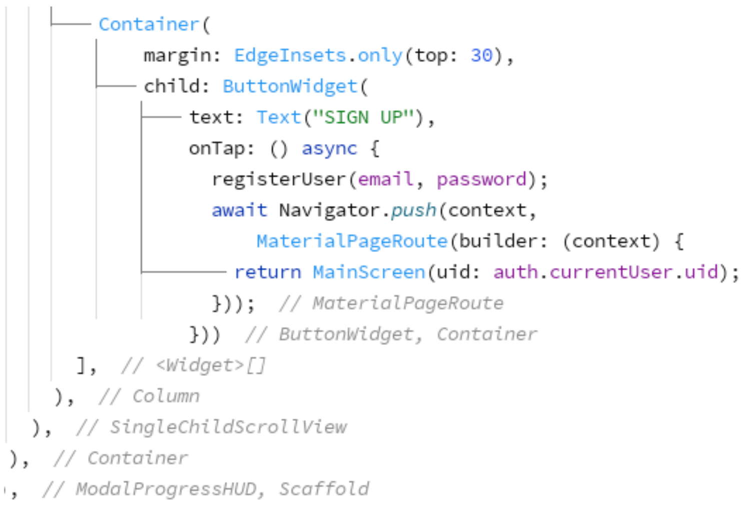
Please enter a valid email

Your Password must be of at least 6 characters

Fig.1.4.3 Sign Up Screen

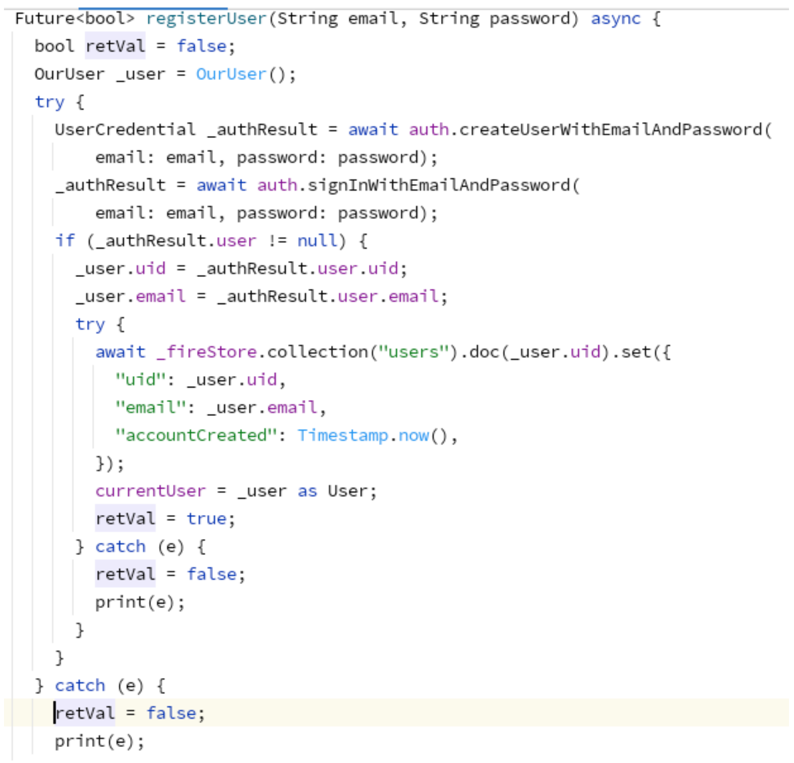
Here, the difference in the text fields is only with the hint text for the password to ask the user to enter the correct password (maximumly containing 6 or more characters). After entering the required data in the fields the user will click on the sign up button which will navigate them to the home screen of the application.

When the user clicks on the signup button, the data will be saved in firebase authentication (Fig.1.4.4,1.4.5,1.4.6). But in order to generate user.id and other user information it will be saved in a firebase collection (Fig 1.4.7) so that user details can be accessed in other classes while generating their separate portfolios.



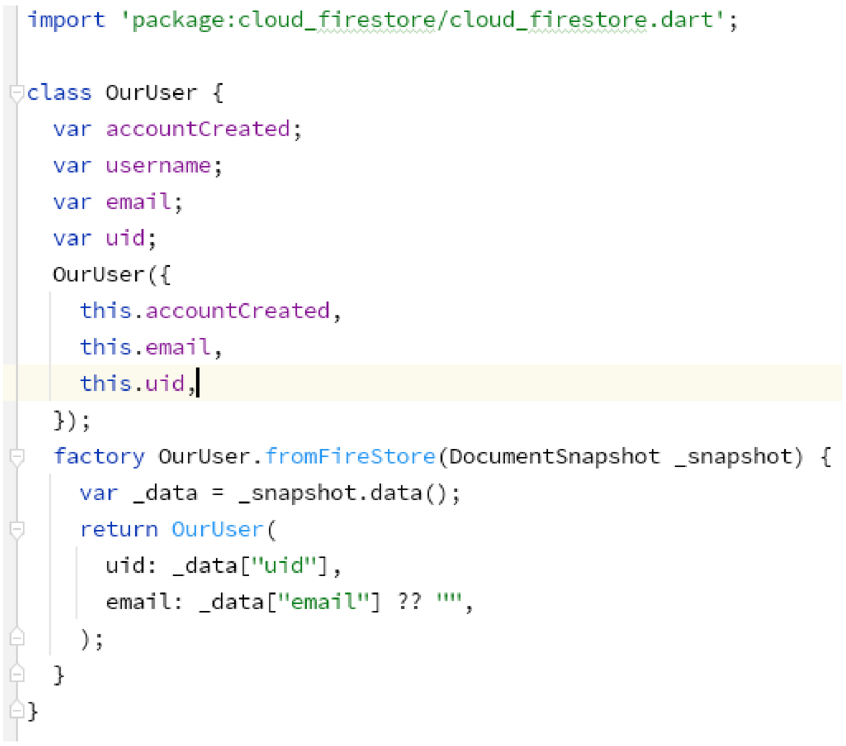
```
      Container(
          margin: EdgeInsets.only(top: 30),
          child: ButtonWidget(
              text: Text("SIGN UP"),
              onTap: () async {
                registerUser(email, password);
                await Navigator.push(context,
                    MaterialPageRoute(builder: (context) {
                  return MainScreen(uid: auth.currentUser.uid);
                })); // MaterialPageRoute
              })) // ButtonWidget, Container
    ], // <Widget>[]
  ), // Column
), // SingleChildScrollView
), // Container
, // ModalProgressHUD, Scaffold
```

Fig.1.4.4 Sign up button coding



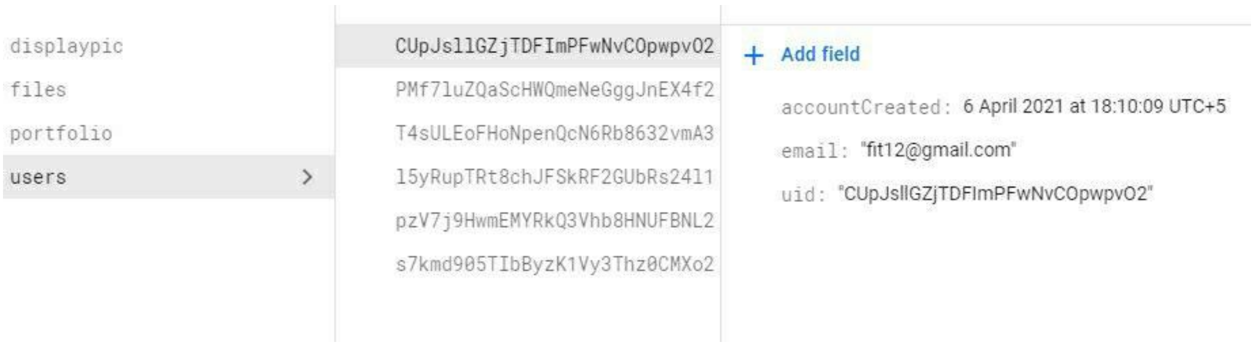
```
Future<bool> registerUser(String email, String password) async {
  bool retVal = false;
  OurUser _user = OurUser();
  try {
    UserCredential _authResult = await auth.createUserWithEmailAndPassword(
        email: email, password: password);
    _authResult = await auth.signInWithEmailAndPassword(
        email: email, password: password);
    if (_authResult.user != null) {
      _user.uid = _authResult.user.uid;
      _user.email = _authResult.user.email;
      try {
        await _fireStore.collection("users").doc(_user.uid).set({
          "uid": _user.uid,
          "email": _user.email,
          "accountCreated": Timestamp.now(),
        });
        currentUser = _user as User;
        retVal = true;
      } catch (e) {
        retVal = false;
        print(e);
      }
    }
  } catch (e) {
    retVal = false;
    print(e);
```

Fig.1.4.5 Sign up button coding



```
import 'package:cloud_firestore/cloud_firestore.dart';

class OurUser {
  var accountCreated;
  var username;
  var email;
  var uid;
  OurUser({
    this.accountCreated,
    this.email,
    this.uid,
  });
  factory OurUser.fromFireStore(DocumentSnapshot _snapshot) {
    var _data = _snapshot.data();
    return OurUser(
      uid: _data["uid"],
      email: _data["email"] ?? "",
    );
  }
}
```

Fig.1.4.6 Sign up button coding



```
displaypic      CUpJs11GZjTDFImPFwNvCOpwpv02    + Add field
files           PMf71uZQaScHWQmeNeGggJnEX4f2
portfolio       T4sULEoFHoNpenQcN6Rb8632vmA3    accountCreated: 6 April 2021 at 18:10:09 UTC+5
users       >   15yRupTRt8chJFSkRF2GUbRs2411    email: "fit12@gmail.com"
                pzV7j9HwmEMYRkQ3Vhb8HNUFBNL2    uid: "CUpJsllGZjTDFImPFwNvCOpwpvO2"
                s7kmd905TIbByzK1Vy3Thz0CMXo2
```

Fig.1.4.7 User information saved in collection

Home/Main Screen:

Fig (1.4.8) screen is designed with a dark-blue colored app bar with app name, a text in the front page and a side drawer.

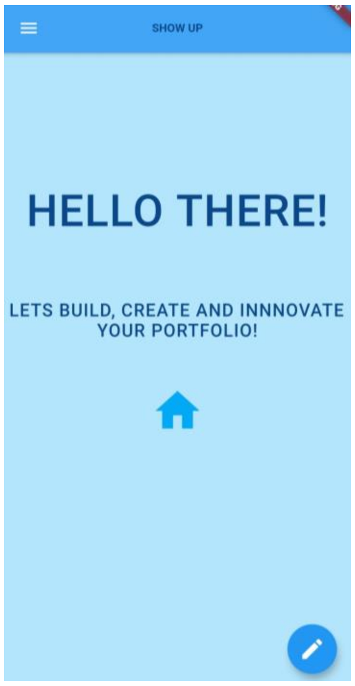


Fig.1.4.8 Home screen

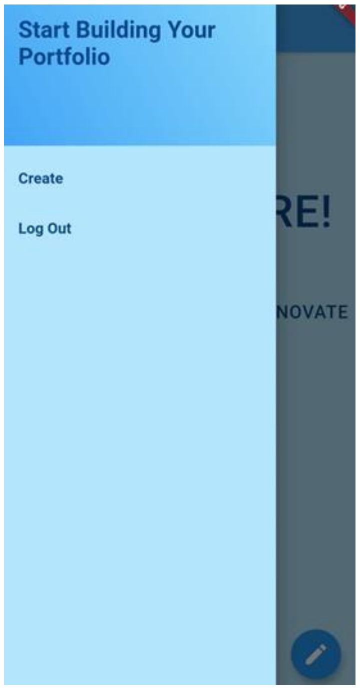
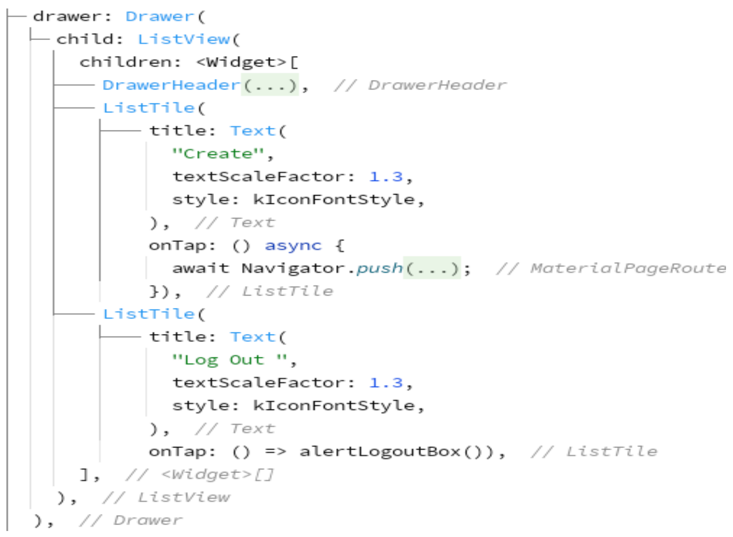


Fig.1.4.9. Home screen Drawer

(Fig. 1.4.9) shows a drawer inside the sidebar option.

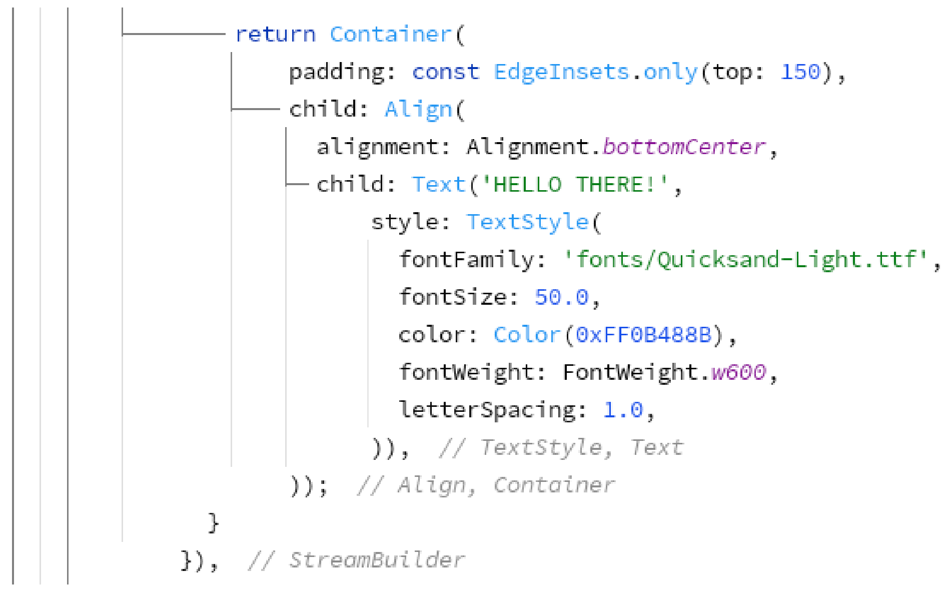


```
drawer: Drawer(
  child: ListView(
    children: <Widget>[
      DrawerHeader(...),  // DrawerHeader
      ListTile(
          title: Text(
            "Create",
            textScaleFactor: 1.3,
            style: kIconFontStyle,
          ),  // Text
          onTap: () async {
            await Navigator.push(...);  // MaterialPageRoute
          }),  // ListTile
      ListTile(
          title: Text(
            "Log Out ",
            textScaleFactor: 1.3,
            style: kIconFontStyle,
          ),  // Text
          onTap: () => alertLogoutBox()),  // ListTile
    ],  // <Widget>[]
  ),  // ListView
),  // Drawer
```

Fig.1.5.0. Home screen Drawer coding

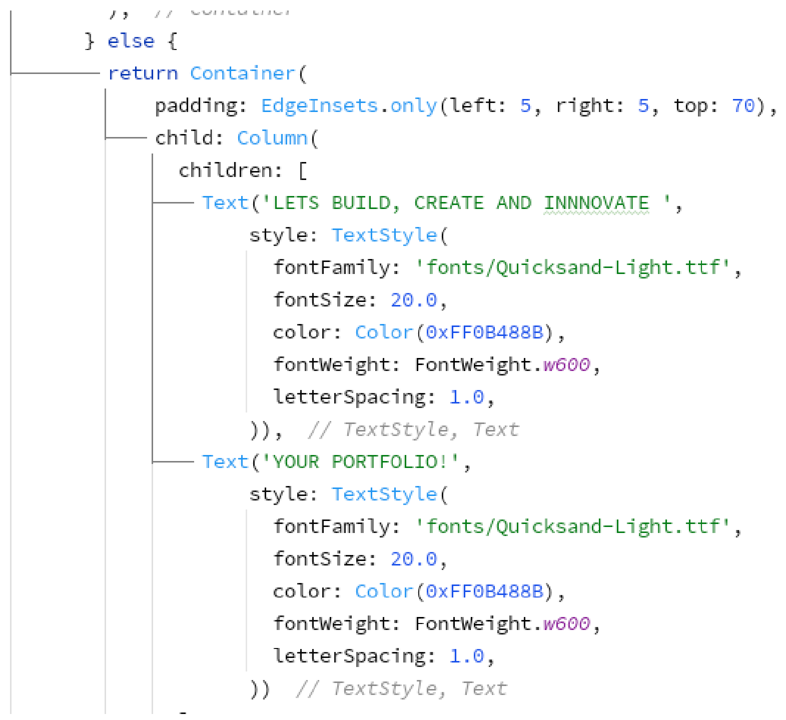
(Fig.1.5.0) This side drawer widget shows an icon on the appbar, and by clicking on the icon it opens up. This drawer contains a drawer header showing a gradient medium size font text i.e. ‘start building your portfolio’ for the user. Then, right below the header it contains a List Tile (a flutter custom widget to present the clickable text) of three different screens in the form of text Create and log out. These list Tiles are followed by design to be in a list hence they are wrapped in a list view widget inside the drawer.

(Fig.1.5.1, 1.5.2, 1.5.3) The home screen is designed in the stream builder of the portfolio which is designed with an exception that if the document in the database doesn’t exist for the given user. Hence, if that user does not yet started to make his portfolio or has just joined the application will see this screen as no data is yet entered in the portfolio.



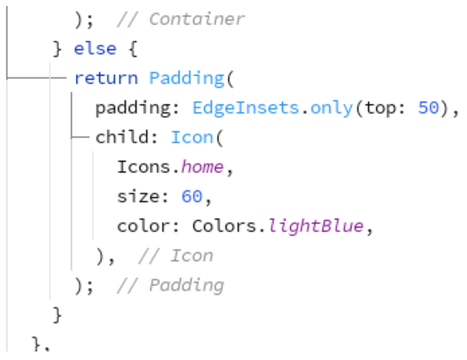
```
            return Container(
              padding: const EdgeInsets.only(top: 150),
              child: Align(
                alignment: Alignment.bottomCenter,
                child: Text('HELLO THERE!',
                    style: TextStyle(
                      fontFamily: 'fonts/Quicksand-Light.ttf',
                      fontSize: 50.0,
                      color: Color(0xFF0B488B),
                      fontWeight: FontWeight.w600,
                      letterSpacing: 1.0,
                    )),  // TextStyle, Text
              ));  // Align, Container
          }
        }),  // StreamBuilder
```

Fig.1.5.1. Home screen coding



```
      ), // Container
    } else {
    return Container(
        padding: EdgeInsets.only(left: 5, right: 5, top: 70),
        child: Column(
          children: [
          Text('LETS BUILD, CREATE AND INNNOVATE ',
              style: TextStyle(
                fontFamily: 'fonts/Quicksand-Light.ttf',
                fontSize: 20.0,
                color: Color(0xFF0B488B),
                fontWeight: FontWeight.w600,
                letterSpacing: 1.0,
              )),  // TextStyle, Text
          Text('YOUR PORTFOLIO!',
              style: TextStyle(
                fontFamily: 'fonts/Quicksand-Light.ttf',
                fontSize: 20.0,
                color: Color(0xFF0B488B),
                fontWeight: FontWeight.w600,
                letterSpacing: 1.0,
              ))  // TextStyle, Text
```

Fig.1.5.2. Home screen coding

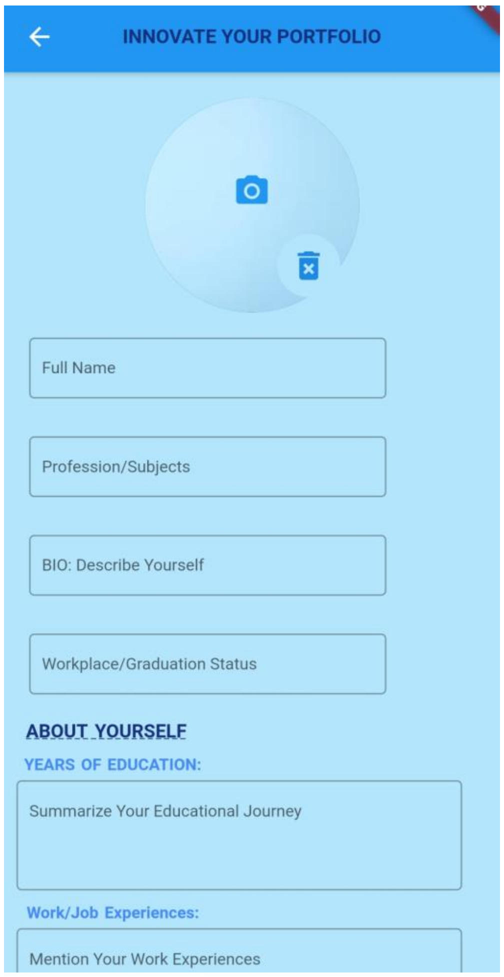


```
      );  // Container
    } else {
    return Padding(
      padding: EdgeInsets.only(top: 50),
      child: Icon(
        Icons.home,
        size: 60,
        color: Colors.lightBlue,
      ),  // Icon
    );  // Padding
    }
  },
```

Fig.1.5.3. Home screen coding

In the drawer, the users click on the ‘Create’ list tile will navigate them to a screen with persistent light blue background color and an app bar showing text ‘innovate your portfolio’. On the left the app bar also contains a back icon button which will send the user back to the drawer screen. (Fig.1.5.4 and 1.5.5)

Create Screen:



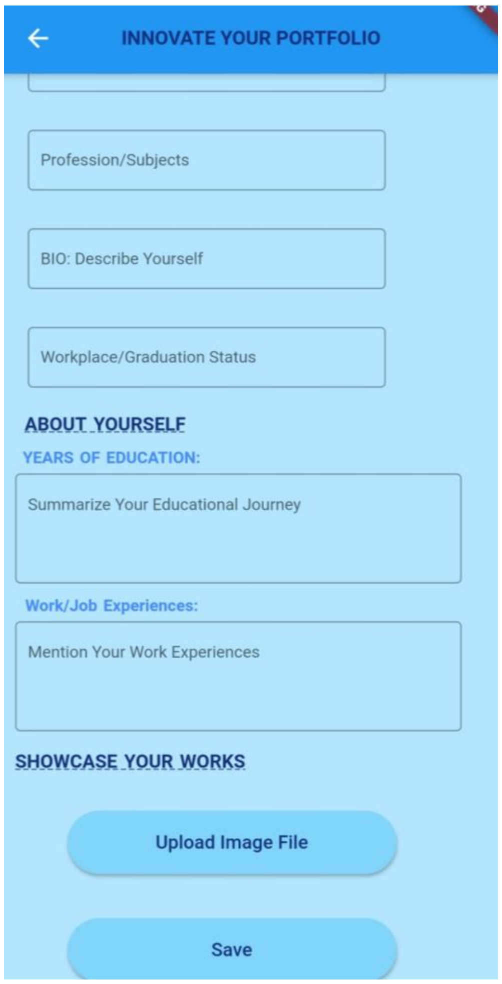
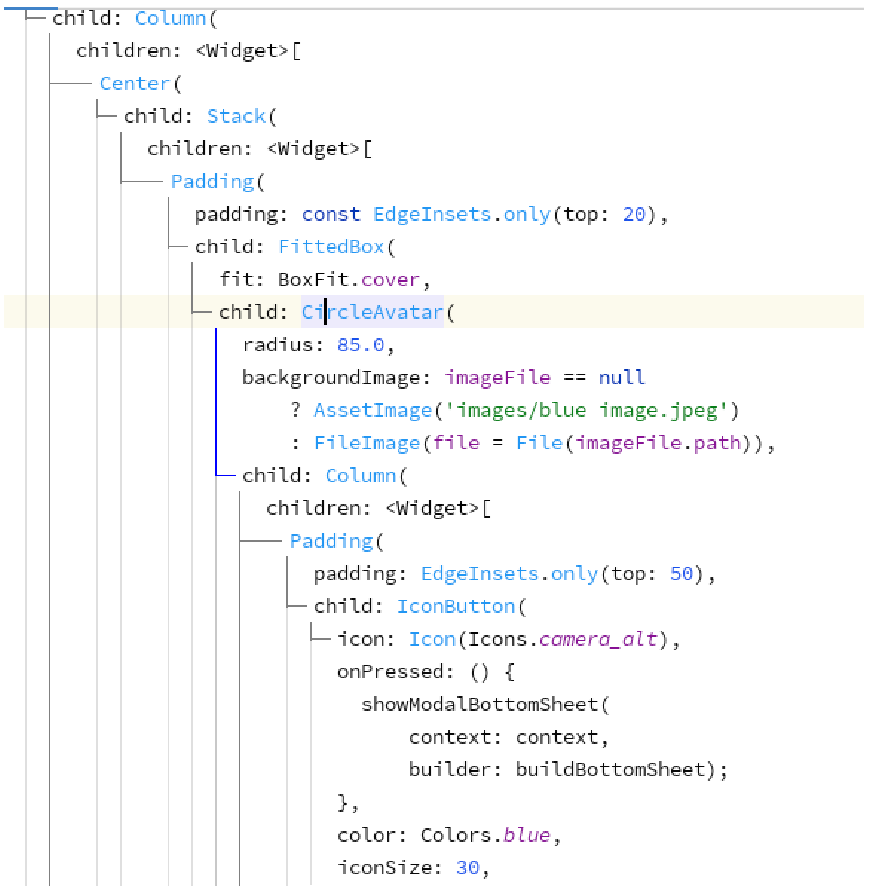


Fig.1.5.4. Create screen

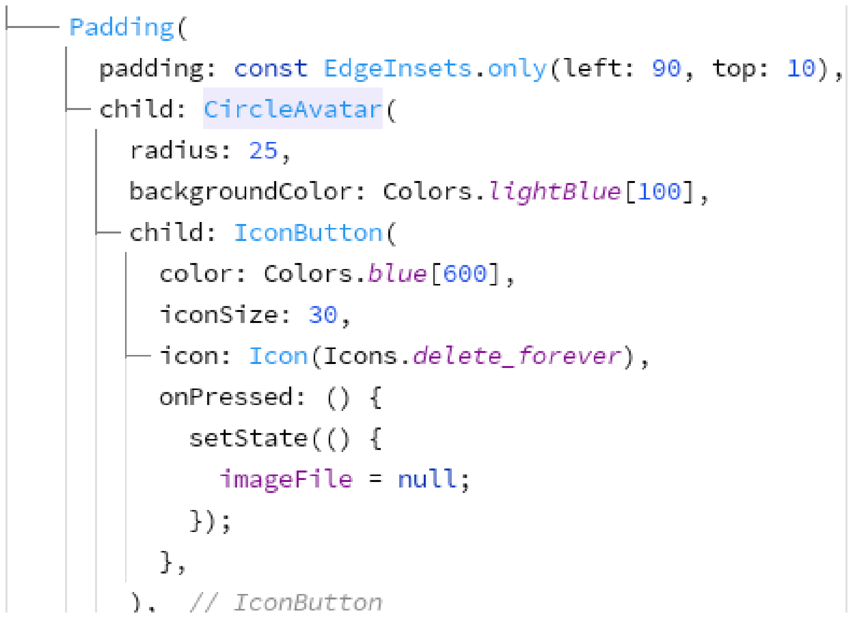
Fig.1.5.5 Create screen

(Fig.1.5.6 and 1.5.7) This screen starts with showing a circle Avatar (a custom flutter class that enables to show images in a circle shape on screen). This widget provides functionality for the users to upload their profile picture. Image picker package is used to make users pick images from camera or gallery. While clicking on the camera button, the user picked image from either camera or gallery will be stored in firebase collection and displayed in the circle avatar widget. The user then fills out the above shown details.



```
child: Column(
  children: <Widget>[
    Center(
      child: Stack(
        children: <Widget>[
          Padding(
            padding: const EdgeInsets.only(top: 20),
            child: FittedBox(
              fit: BoxFit.cover,
              child: CircleAvatar(
                radius: 85.0,
                backgroundImage: imageFile == null
                    ? AssetImage('images/blue image.jpeg')
                    : FileImage(file = File(imageFile.path)),
                child: Column(
                  children: <Widget>[
                    Padding(
                      padding: EdgeInsets.only(top: 50),
                      child: IconButton(
                        icon: Icon(Icons.camera_alt),
                        onPressed: () {
                          showModalBottomSheet(
                              context: context,
                              builder: buildBottomSheet);
                        },
                        color: Colors.blue,
                        iconSize: 30,
```

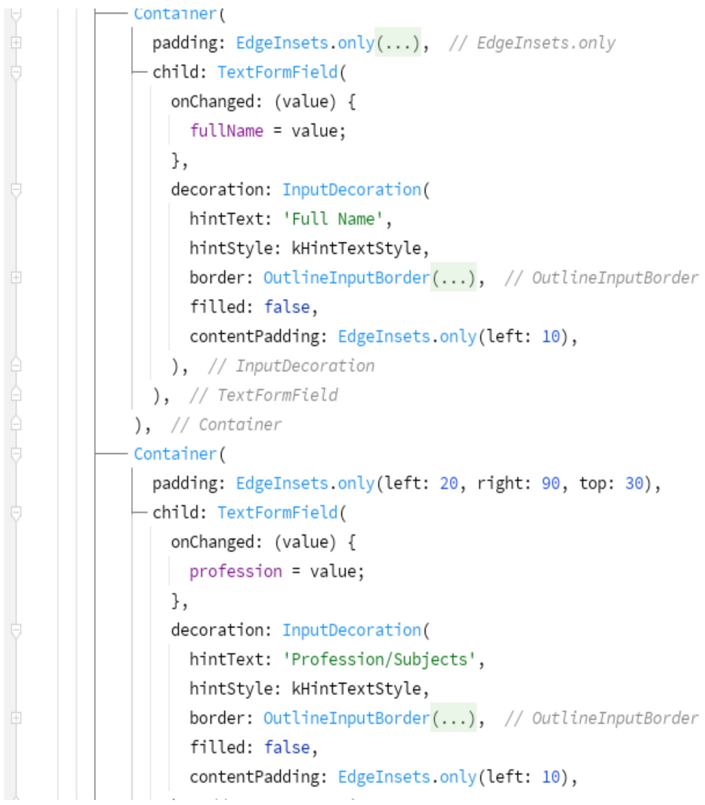
Fig.1.5.6 Create screen coding



```
Padding(
  padding: const EdgeInsets.only(left: 90, top: 10),
  child: CircleAvatar(
    radius: 25,
    backgroundColor: Colors.lightBlue[100],
    child: IconButton(
      color: Colors.blue[600],
      iconSize: 30,
      icon: Icon(Icons.delete_forever),
      onPressed: () {
        setState(() {
          imageFile = null;
        });
      },
    ), // IconButton
```

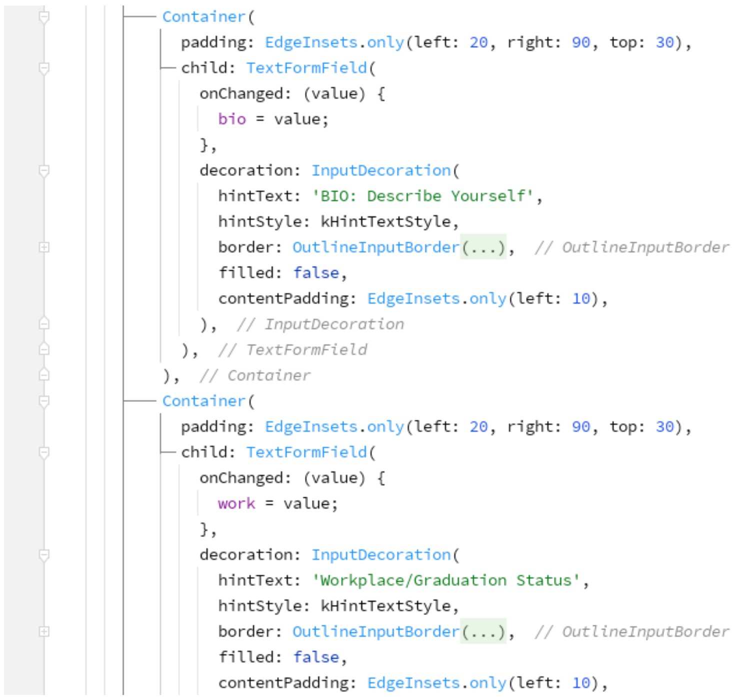
Fig.1.5.7 Create screen coding

(Fig.1.5.8, Fig.1.5.9, Fig.1.6.0, Fig.1.6.1, Fig.1.6.2) shows the designing of all the text fields and headings created in the screen. The screen is wrapped in a single child scroll view so that the user can scroll on the screen. The text fields values are set as string so that it can be saved in firestore collection as a string. Some of the text fields of experience and education (fig.1.6.0 and 1.6.1) with detail description were set as maximum lines so that user can enter text in given limits.



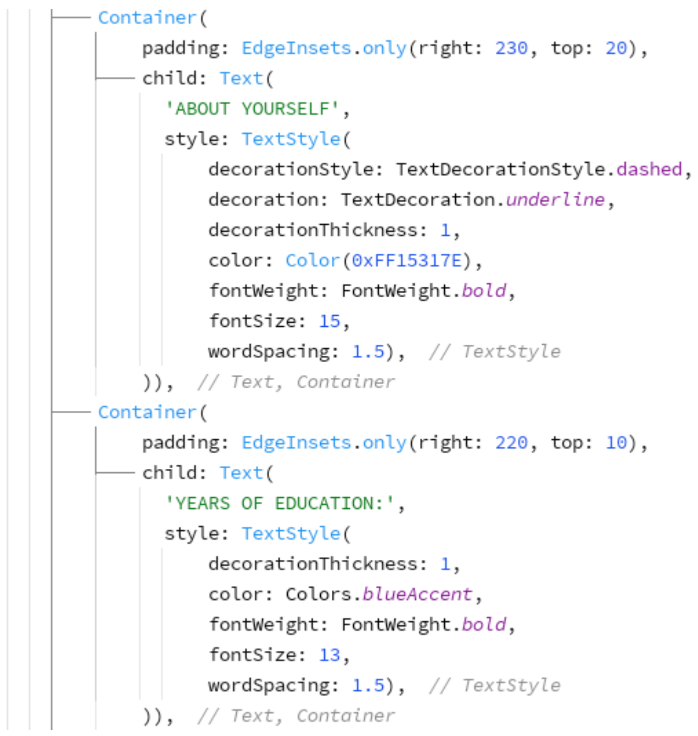
```
Container(
  padding: EdgeInsets.only(...),  // EdgeInsets.only
  child: TextFormField(
    onChanged: (value) {
      fullName = value;
    },
    decoration: InputDecoration(
      hintText: 'Full Name',
      hintStyle: kHintTextStyle,
      border: OutlineInputBorder(...),  // OutlineInputBorder
      filled: false,
      contentPadding: EdgeInsets.only(left: 10),
    ),  // InputDecoration
  ),  // TextFormField
),  // Container
Container(
  padding: EdgeInsets.only(left: 20, right: 90, top: 30),
  child: TextFormField(
    onChanged: (value) {
      profession = value;
    },
    decoration: InputDecoration(
      hintText: 'Profession/Subjects',
      hintStyle: kHintTextStyle,
      border: OutlineInputBorder(...),  // OutlineInputBorder
      filled: false,
      contentPadding: EdgeInsets.only(left: 10),
```

Fig.1.5.8 Create screen coding



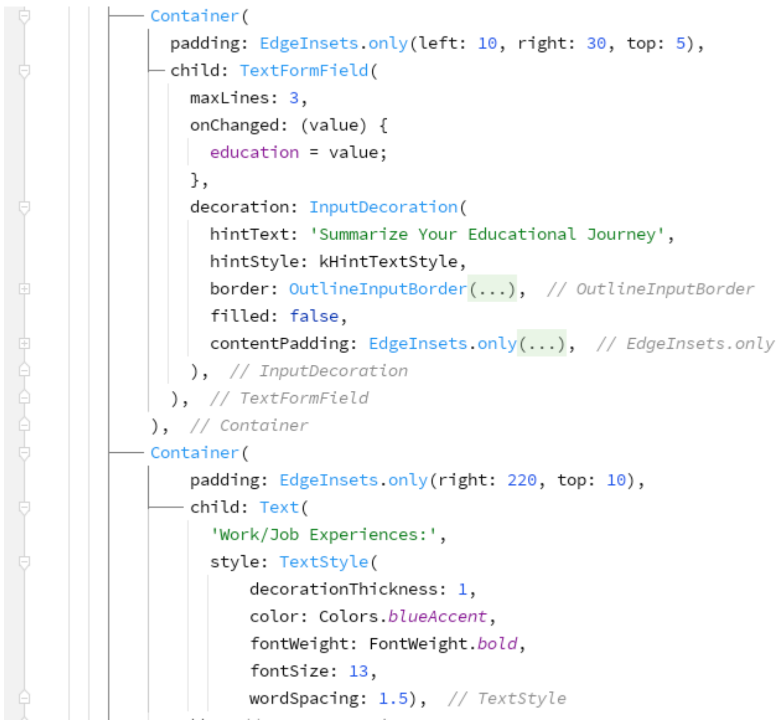
```
Container(
  padding: EdgeInsets.only(left: 20, right: 90, top: 30),
  child: TextFormField(
    onChanged: (value) {
      bio = value;
    },
    decoration: InputDecoration(
      hintText: 'BIO: Describe Yourself',
      hintStyle: kHintTextStyle,
      border: OutlineInputBorder(...),  // OutlineInputBorder
      filled: false,
      contentPadding: EdgeInsets.only(left: 10),
    ),  // InputDecoration
  ),  // TextFormField
),  // Container
Container(
  padding: EdgeInsets.only(left: 20, right: 90, top: 30),
  child: TextFormField(
    onChanged: (value) {
      work = value;
    },
    decoration: InputDecoration(
      hintText: 'Workplace/Graduation Status',
      hintStyle: kHintTextStyle,
      border: OutlineInputBorder(...),  // OutlineInputBorder
      filled: false,
      contentPadding: EdgeInsets.only(left: 10),
```

Fig.1.5.9 Create screen coding



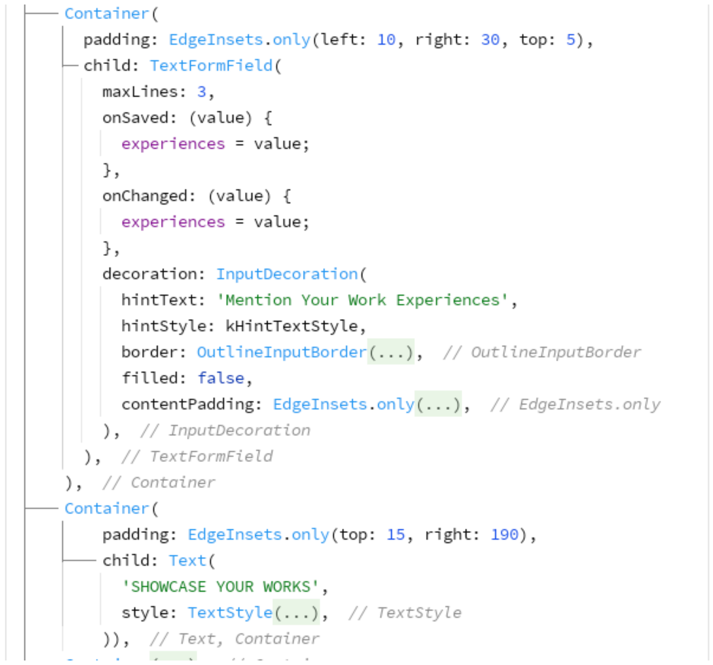
```
Container(
    padding: EdgeInsets.only(right: 230, top: 20),
    child: Text(
      'ABOUT YOURSELF',
      style: TextStyle(
          decorationStyle: TextDecorationStyle.dashed,
          decoration: TextDecoration.underline,
          decorationThickness: 1,
          color: Color(0xFF15317E),
          fontWeight: FontWeight.bold,
          fontSize: 15,
          wordSpacing: 1.5),  // TextStyle
    )),  // Text, Container
Container(
    padding: EdgeInsets.only(right: 220, top: 10),
    child: Text(
      'YEARS OF EDUCATION:',
      style: TextStyle(
          decorationThickness: 1,
          color: Colors.blueAccent,
          fontWeight: FontWeight.bold,
          fontSize: 13,
          wordSpacing: 1.5),  // TextStyle
    )),  // Text, Container
```

Fig.1.6.0 Create screen coding



```
Container(
  padding: EdgeInsets.only(left: 10, right: 30, top: 5),
  child: TextFormField(
    maxLines: 3,
    onChanged: (value) {
      education = value;
    },
    decoration: InputDecoration(
      hintText: 'Summarize Your Educational Journey',
      hintStyle: kHintTextStyle,
      border: OutlineInputBorder(...),  // OutlineInputBorder
      filled: false,
      contentPadding: EdgeInsets.only(...),  // EdgeInsets.only
    ),  // InputDecoration
  ),  // TextFormField
),  // Container
Container(
    padding: EdgeInsets.only(right: 220, top: 10),
    child: Text(
      'Work/Job Experiences:',
      style: TextStyle(
          decorationThickness: 1,
          color: Colors.blueAccent,
          fontWeight: FontWeight.bold,
          fontSize: 13,
          wordSpacing: 1.5),  // TextStyle
```

Fig.1.6.1 Create screen coding



```
Container(
  padding: EdgeInsets.only(left: 10, right: 30, top: 5),
  child: TextFormField(
    maxLines: 3,
    onSaved: (value) {
      experiences = value;
    },
    onChanged: (value) {
      experiences = value;
    },
    decoration: InputDecoration(
      hintText: 'Mention Your Work Experiences',
      hintStyle: kHintTextStyle,
      border: OutlineInputBorder(...),  // OutlineInputBorder
      filled: false,
      contentPadding: EdgeInsets.only(...),  // EdgeInsets.only
    ),  // InputDecoration
  ),  // TextFormField
),  // Container
Container(
    padding: EdgeInsets.only(top: 15, right: 190),
    child: Text(
      'SHOWCASE YOUR WORKS',
      style: TextStyle(...),  // TextStyle
    )),  // Text, Container
```

Fig.1.6.2 Create screen coding

When clicking on the upload image file button (fig.1.5.5) will lead the user to select an image from the gallery. (Fig.1.6.4 and 1.6.5) shows the same image picker is used here to select image from gallery. The coding also shows that how the image is stored as image url firebase collection under its specific user id. After selecting an image, the user will receive an alert box showing that their image is successfully uploaded (Fig.1.6.3).

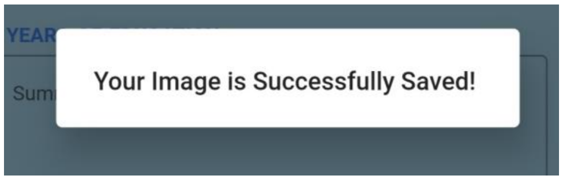

Your Image is Successfully Saved!

Fig.1.6.3 Create screen (upload ImageFile button)

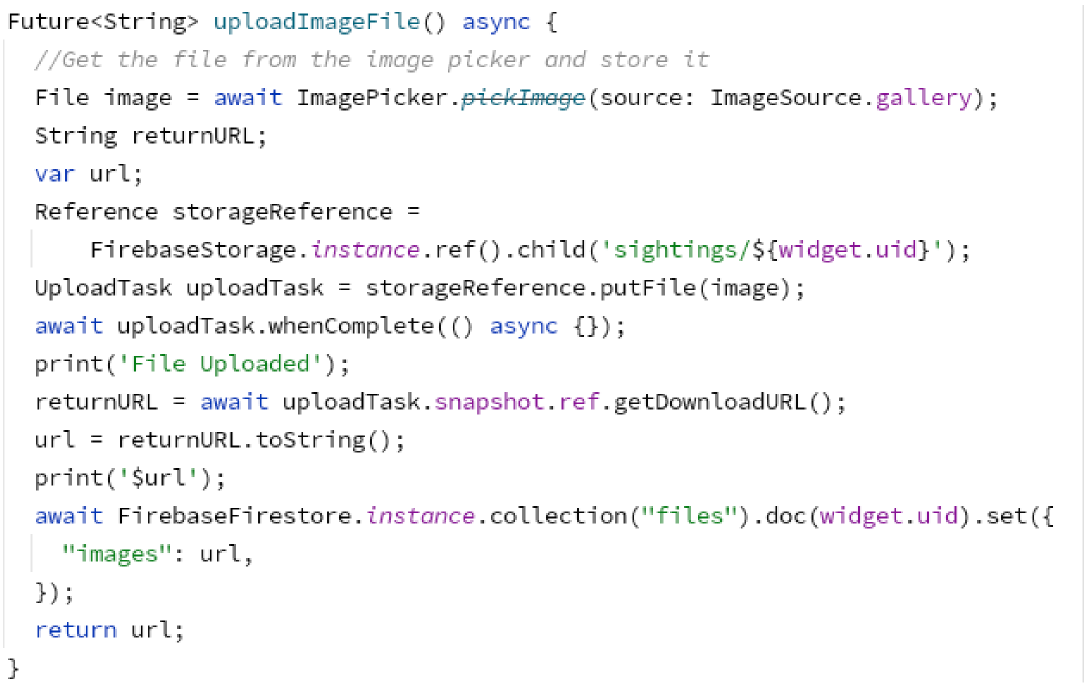
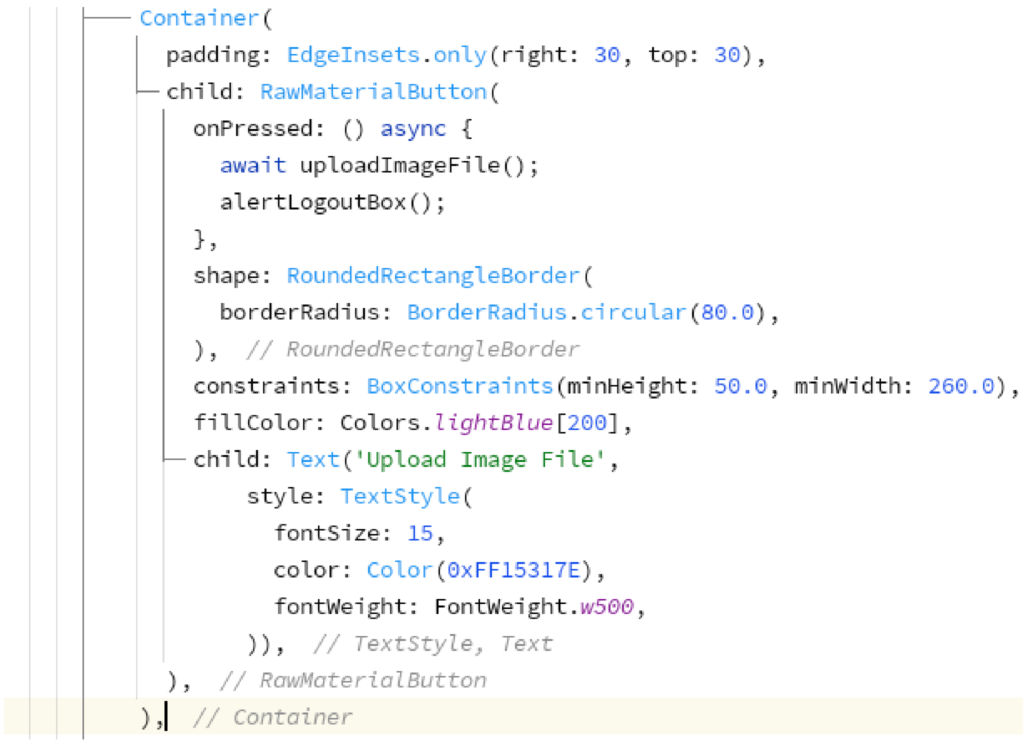

```
Future<String> uploadImageFile() async {
  //Get the file from the image picker and store it
  File image = await ImagePicker.~~pickImage~~(source: ImageSource.gallery);
  String returnURL;
  var url;
  Reference storageReference =
      FirebaseStorage.instance.ref().child('sightings/${widget.uid}');
  UploadTask uploadTask = storageReference.putFile(image);
  await uploadTask.whenComplete(() async {});
  print('File Uploaded');
  returnURL = await uploadTask.snapshot.ref.getDownloadURL();
  url = returnURL.toString();
  print('$url');
  await FirebaseFirestore.instance.collection("files").doc(widget.uid).set({
    "images": url,
  });
  return url;
}
```

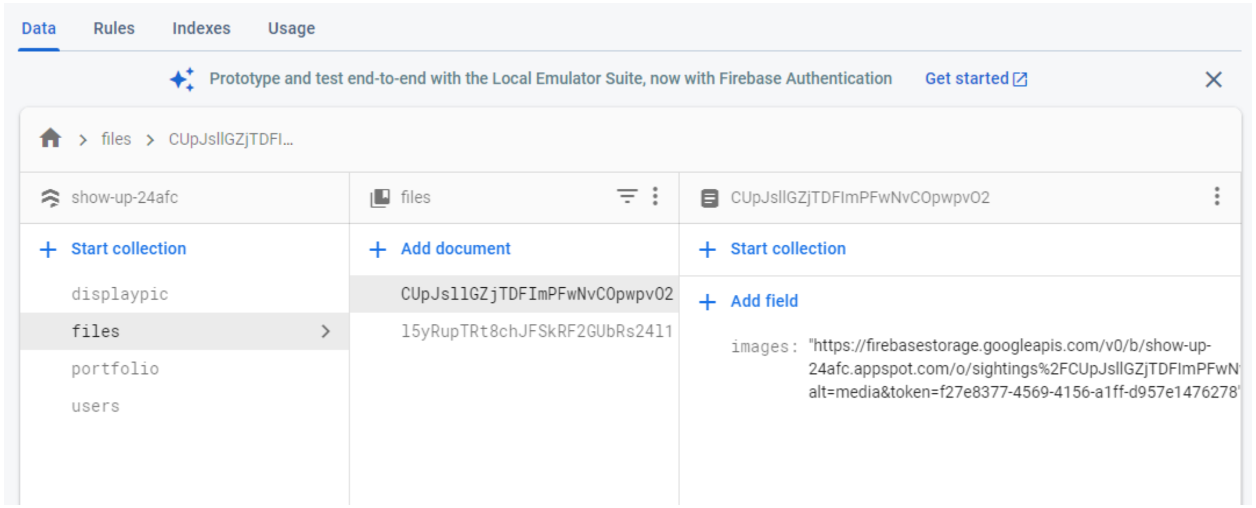
Fig.1.6.4 Create screen (upload ImageFile button coding)



```
Container(
  padding: EdgeInsets.only(right: 30, top: 30),
  child: RawMaterialButton(
    onPressed: () async {
      await uploadImageFile();
      alertLogoutBox();
    },
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(80.0),
    ), // RoundedRectangleBorder
    constraints: BoxConstraints(minHeight: 50.0, minWidth: 260.0),
    fillColor: Colors.lightBlue[200],
    child: Text('Upload Image File',
        style: TextStyle(
          fontSize: 15,
          color: Color(0xFF15317E),
          fontWeight: FontWeight.w500,
        )), // TextStyle, Text
  ), // RawMaterialButton
), // Container
```

Fig.1.6.5 Create screen (upload ImageFile button coding)

(Fig.1.6.6) The image selected will be saved as image url under firebase collection with that user specific id.


Data
Rules
Indexes
Usage
Prototype and test end-to-end with the Local Emulator Suite, now with Firebase Authentication
Get started
files > CUpJslIGZjTDFI...
show-up-24afc
files
CUpJslIGZjTDFImPFwNvCOpwpvO2
Start collection
Add document
Start collection
Add field

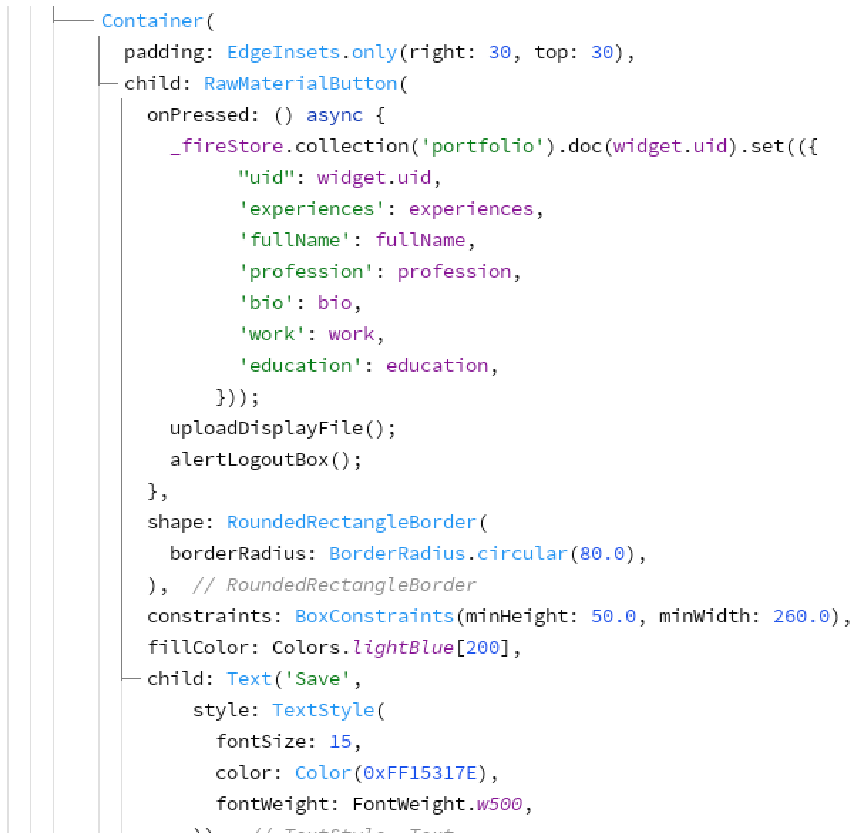
```
displaypic      CUpJs1lGZjTDFImPFwNvCOpwpvO2
files           15yRupTRt8chJFSkRF2GUbRs2411
portfolio
users
```

images: "https://firebasestorage.googleapis.com/v0/b/show-up-24afc.appspot.com/o/sightings%2FCUpJsIlGZjTDFImPFwN
alt=media&token=f27e8377-4569-4156-a1ff-d957e1476278

Fig.1.6.6 Image saved in firebase collection

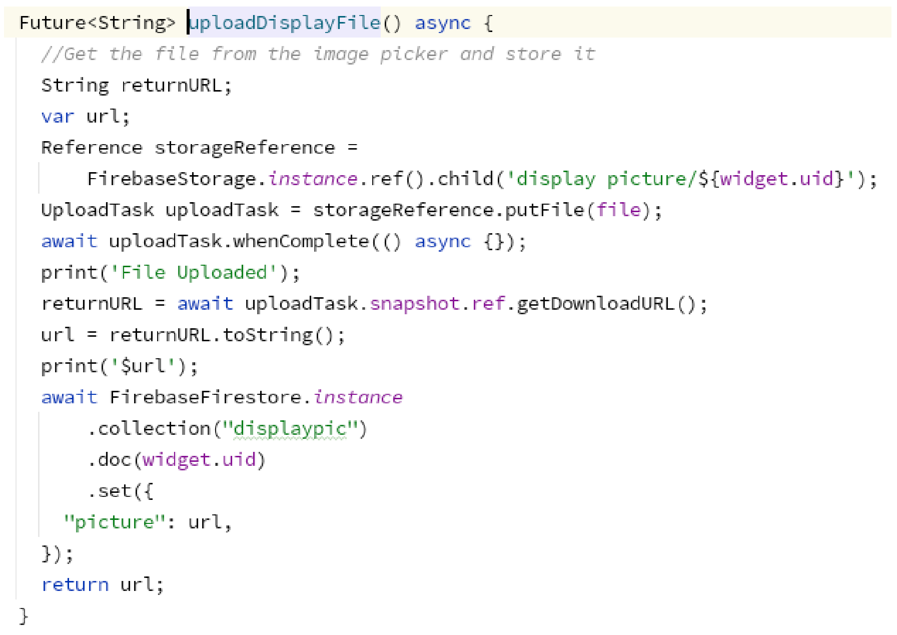
Then the save button in (Fig.1.5.5) will save all the portfolio details in the firebase collection with user specific id so that the portfolio will only be shown to the user who has created it. Fig.1.6.7

shows how when the user presses the button it will save the entered data in the firebase collection (fig.1.7.0) along with its user id. Then the function uploadDisplayFile (fig.1.6.8) shows that the how the user display picture will be stored as image url in a separate collection (fig 1.6.9) with same user id.



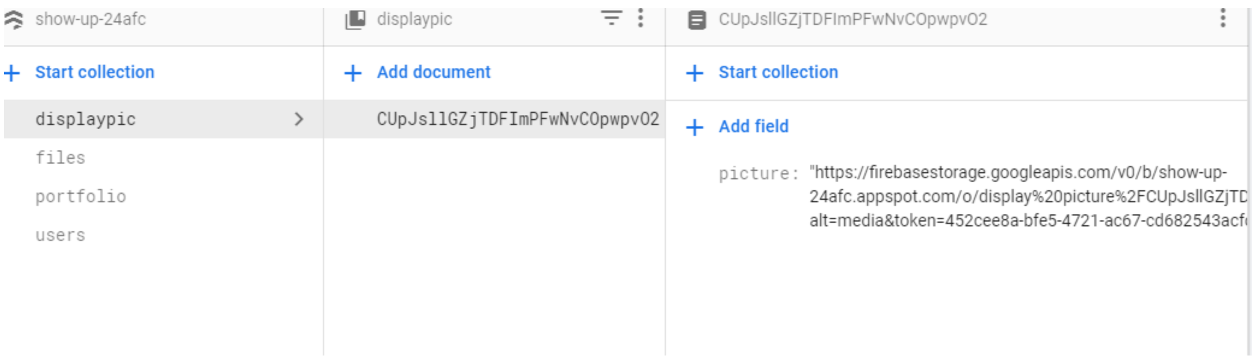
```
Container(
  padding: EdgeInsets.only(right: 30, top: 30),
  child: RawMaterialButton(
    onPressed: () async {
      _fireStore.collection('portfolio').doc(widget.uid).set(({
            "uid": widget.uid,
            'experiences': experiences,
            'fullName': fullName,
            'profession': profession,
            'bio': bio,
            'work': work,
            'education': education,
          }));
      uploadDisplayFile();
      alertLogoutBox();
    },
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(80.0),
    ),  // RoundedRectangleBorder
    constraints: BoxConstraints(minHeight: 50.0, minWidth: 260.0),
    fillColor: Colors.lightBlue[200],
    child: Text('Save',
        style: TextStyle(
          fontSize: 15,
          color: Color(0xFF15317E),
          fontWeight: FontWeight.w500,
```

Fig.1.6.7 Save button coding



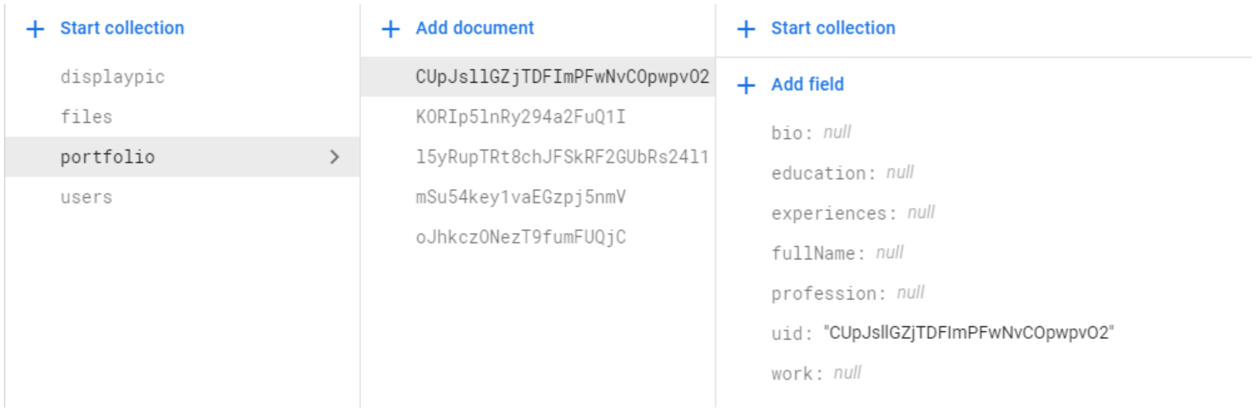
```
Future<String> uploadDisplayFile() async {
  //Get the file from the image picker and store it
  String returnURL;
  var url;
  Reference storageReference =
      FirebaseStorage.instance.ref().child('display picture/${widget.uid}');
  UploadTask uploadTask = storageReference.putFile(file);
  await uploadTask.whenComplete(() async {});
  print('File Uploaded');
  returnURL = await uploadTask.snapshot.ref.getDownloadURL();
  url = returnURL.toString();
  print('$url');
  await FirebaseFirestore.instance
      .collection("displaypic")
      .doc(widget.uid)
      .set({
    "picture": url,
  });
  return url;
}
```

Fig.1.6.8 Save button (user profile picture coding)



```
show-up-24afc               displaypic                     CUpJsllGZjTDFImPFwNvCOpwpvO2
+ Start collection          + Add document                 + Start collection
displaypic  >               CUpJsllGZjTDFImPFwNvCOpwpvO2   + Add field
files                                                      picture: "https://firebasestorage.googleapis.com/v0/b/show-up-
portfolio                                                  24afc.appspot.com/o/display%20picture%2FCUpJsllGZjTD
users                                                      alt=media&token=452cee8a-bfe5-4721-ac67-cd682543acf
```

Fig.1.6.9 In Save button user profile picture save as url in firebase collection



```
+ Start collection     + Add document                  + Start collection
displaypic             CUpJsllGZjTDFImPFwNvCOpwpvO2    + Add field
files                  KORIp5lnRy294a2FuQ1I            bio: null
portfolio  >           l5yRupTRt8chJFSkRF2GUbRs24l1    education: null
users                  mSu54key1vaEGzpj5nmV            experiences: null
                       oJhkczONezT9fumFUQjC            fullName: null
                                                       profession: null
                                                       uid: "CUpJsllGZjTDFImPFwNvCOpwpvO2"
                                                       work: null
```

Fig.1.7.0 In Save button user portfolio details stored as string in firebase collection with user id.

After clicking on the save button the user can see an alert dialogue box (Fig.1.7.1) which will show that their portfolio has been saved.

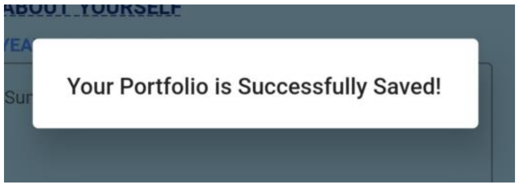
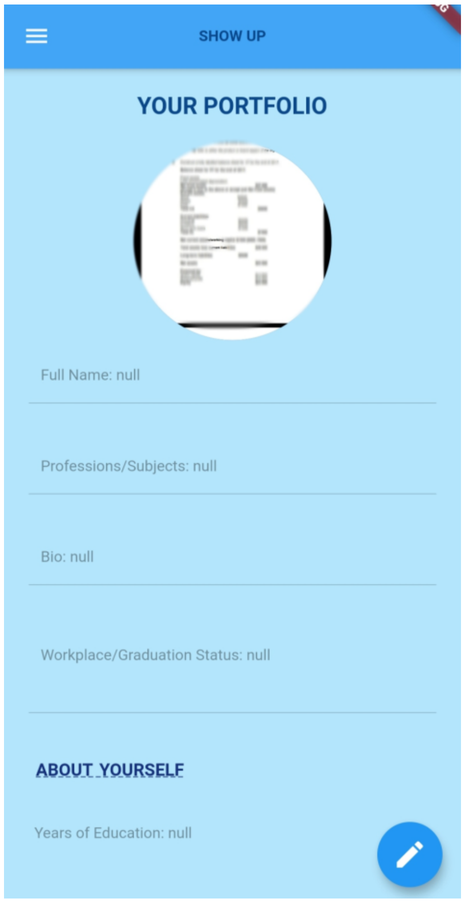

Your Portfolio is Successfully Saved!

Fig.1.7.1 alert box for save button

The saved portfolio will be shown in the home screen of the application. (Fig.1.7.2, 1.7.3)

Home Screen (showing Saved Portfolio):



YOUR PORTFOLIO

Full Name: null

Professions/Subjects: null

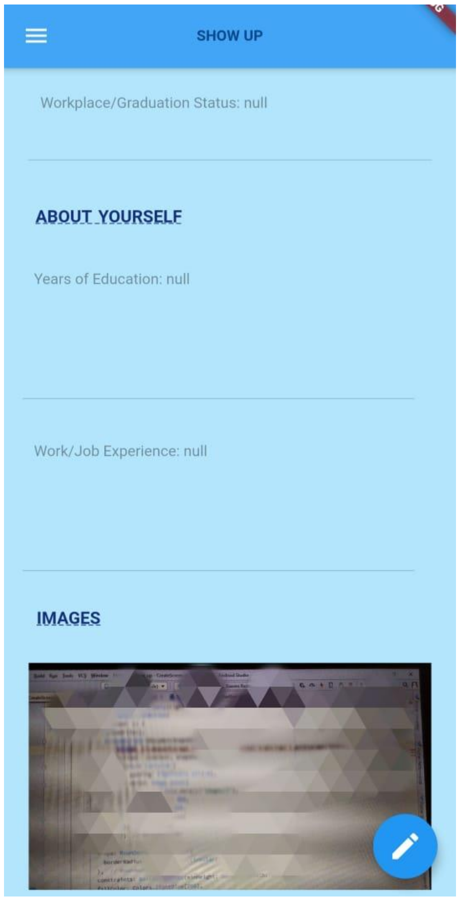
Bio: null

Workplace/Graduation Status: null

ABOUT YOURSELF

Years of Education: null

Fig.1.7.2 Home screen coding



Workplace/Graduation Status: null

ABOUT YOURSELF

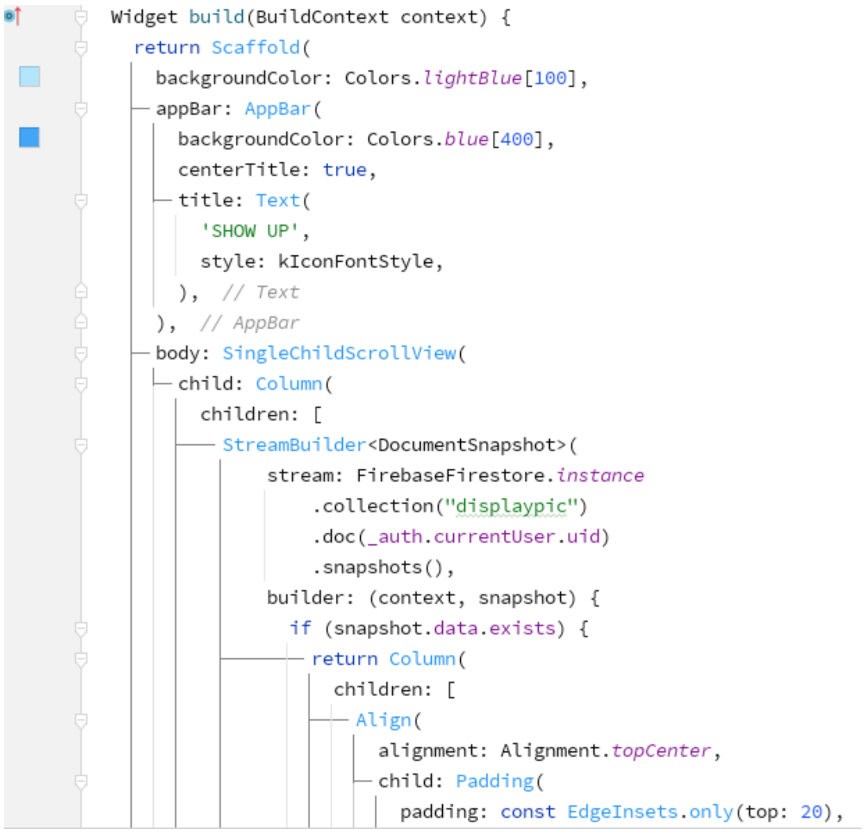
Years of Education: null

Work/Job Experience: null

IMAGES

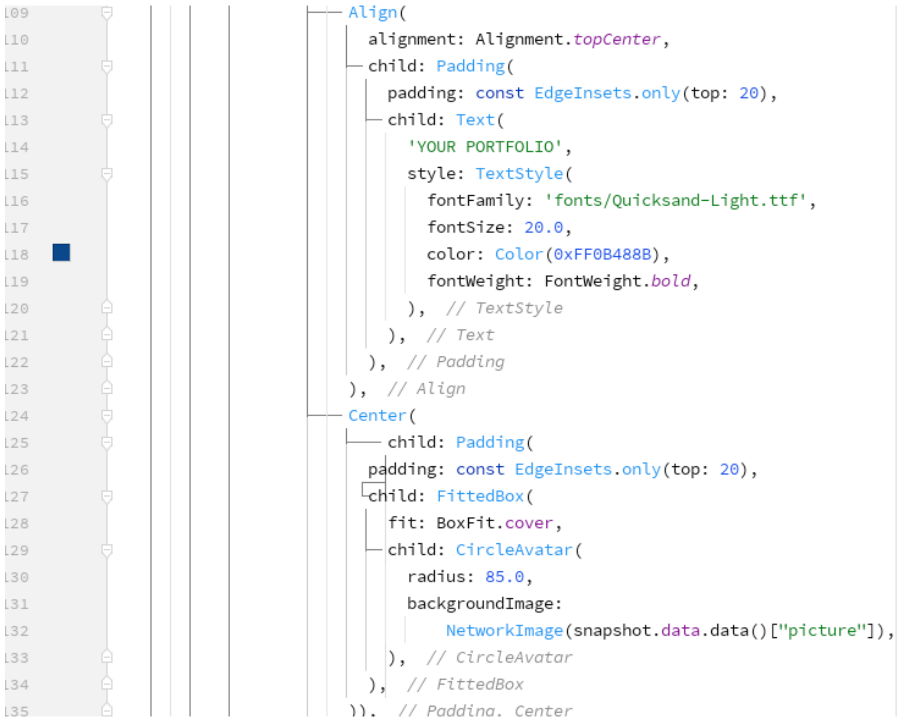
Fig.1.7.3 Home screen coding

The below coding (Fig.1.7.4, 1.7.5, 1.7.6, 1.7.8, 1.7.9, 1.8.0) shows how the details entered by the user in the text fields is retrieved from the firebase collection and are displayed in the home screen. In order to retrieve this data stream builder is used which is a pre-built class in flutter used to interact with other stream of data and display that data in a widget. Here in the stream builder, the stream is set to get the document snapshots (stream data) from the specified firebase collection. This data will be presented in a widget through a builder which is a widget inside the stream builder. This builder is set with thinking logically i.e. if the snapshots or data exists in firebase then it will create this column widget. Otherwise, it will return the home screen where the text is written (Fig.1.4.8) shown above.



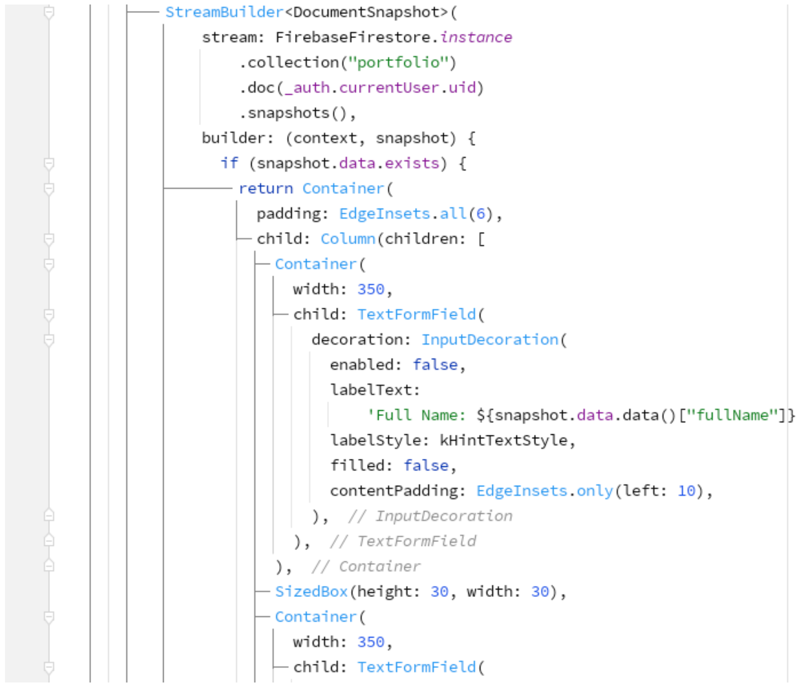
```
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.lightBlue[100],
    appBar: AppBar(
      backgroundColor: Colors.blue[400],
      centerTitle: true,
      title: Text(
        'SHOW UP',
        style: kIconFontStyle,
      ),  // Text
    ),  // AppBar
    body: SingleChildScrollView(
      child: Column(
        children: [
          StreamBuilder<DocumentSnapshot>(
            stream: FirebaseFirestore.instance
                .collection("displaypic")
                .doc(_auth.currentUser.uid)
                .snapshots(),
            builder: (context, snapshot) {
              if (snapshot.data.exists) {
                return Column(
                  children: [
                    Align(
                      alignment: Alignment.topCenter,
                      child: Padding(
                        padding: const EdgeInsets.only(top: 20),
```

Fig.1.7.4 Home screen coding



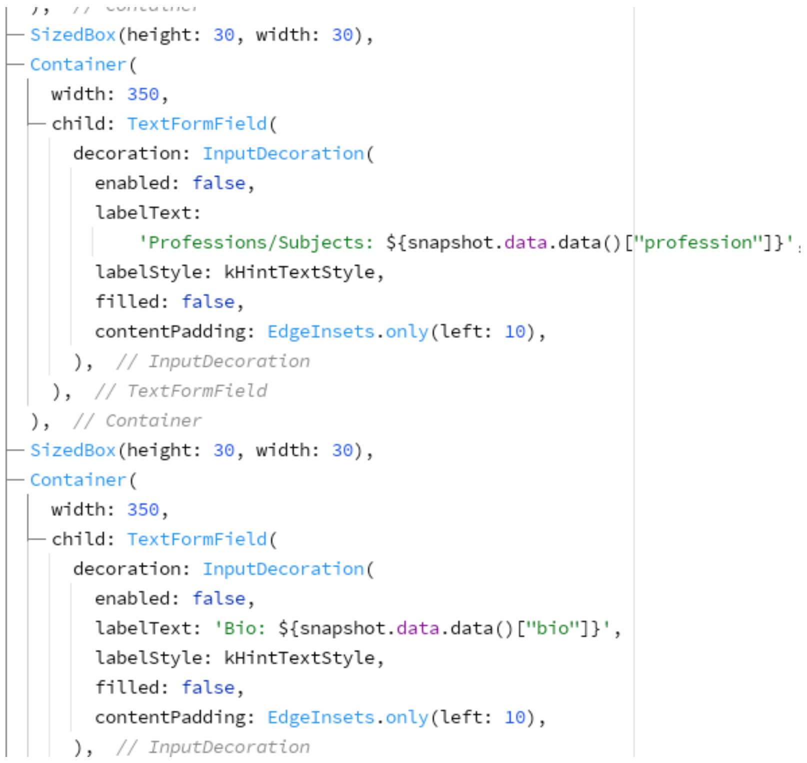
```
Align(
  alignment: Alignment.topCenter,
  child: Padding(
    padding: const EdgeInsets.only(top: 20),
    child: Text(
      'YOUR PORTFOLIO',
      style: TextStyle(
        fontFamily: 'fonts/Quicksand-Light.ttf',
        fontSize: 20.0,
        color: Color(0xFF0B488B),
        fontWeight: FontWeight.bold,
      ), // TextStyle
    ), // Text
  ), // Padding
), // Align
Center(
    child: Padding(
  padding: const EdgeInsets.only(top: 20),
  child: FittedBox(
    fit: BoxFit.cover,
    child: CircleAvatar(
      radius: 85.0,
      backgroundImage:
          NetworkImage(snapshot.data.data()["picture"]),
    ), // CircleAvatar
  ), // FittedBox
)), // Padding, Center
```

Fig.1.7.5 Home screen coding



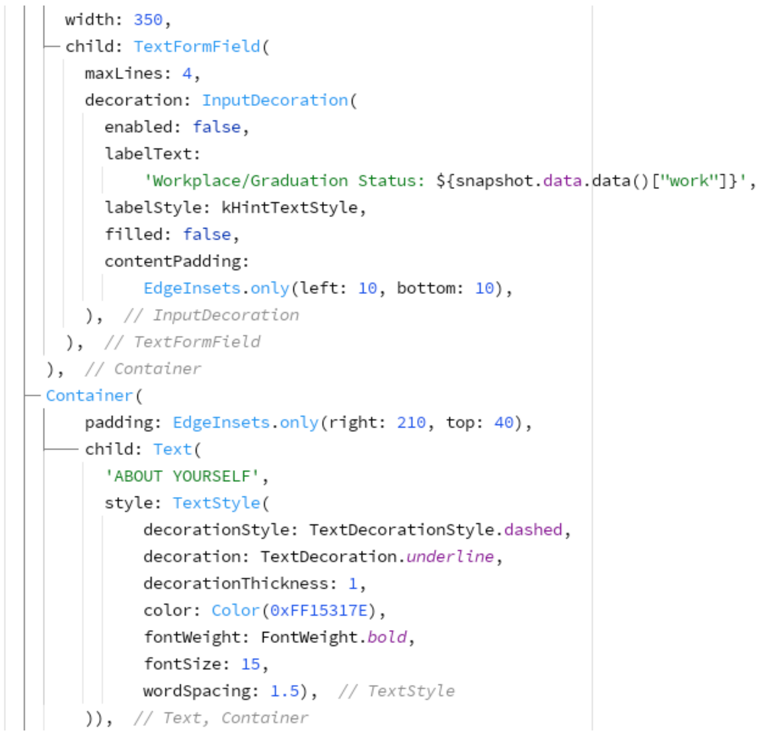
```
StreamBuilder<DocumentSnapshot>(
    stream: FirebaseFirestore.instance
        .collection("portfolio")
        .doc(_auth.currentUser.uid)
        .snapshots(),
    builder: (context, snapshot) {
      if (snapshot.data.exists) {
        return Container(
          padding: EdgeInsets.all(6),
          child: Column(children: [
            Container(
              width: 350,
              child: TextFormField(
                decoration: InputDecoration(
                  enabled: false,
                  labelText:
                      'Full Name: ${snapshot.data.data()["fullName"]}',
                  labelStyle: kHintTextStyle,
                  filled: false,
                  contentPadding: EdgeInsets.only(left: 10),
                ), // InputDecoration
              ), // TextFormField
            ), // Container
            SizedBox(height: 30, width: 30),
            Container(
              width: 350,
              child: TextFormField(
```

Fig.1.7.6 Home screen coding



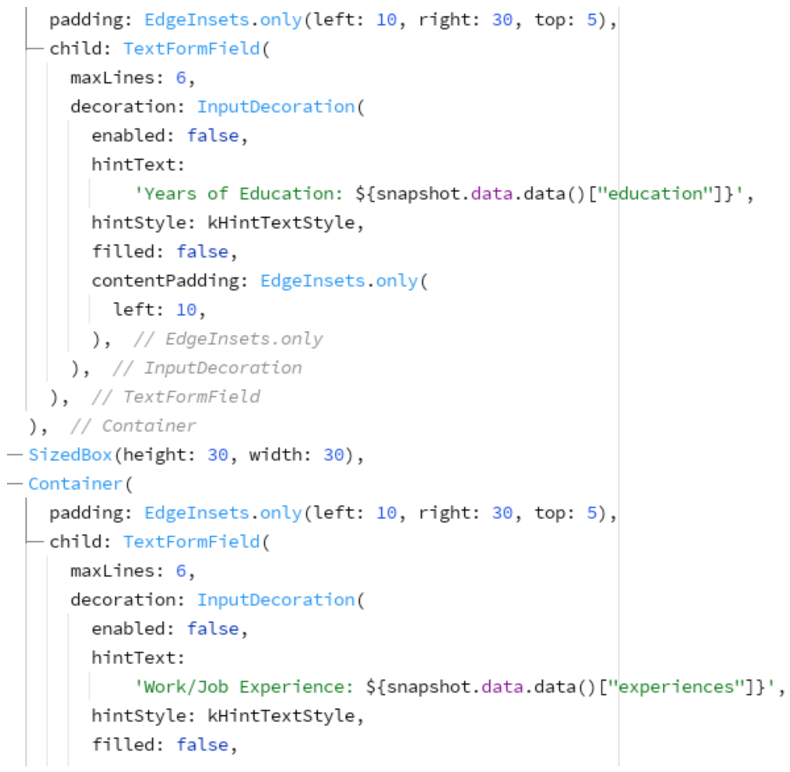
```
  ),  // Container
  SizedBox(height: 30, width: 30),
  Container(
    width: 350,
    child: TextFormField(
      decoration: InputDecoration(
        enabled: false,
        labelText:
            'Professions/Subjects: ${snapshot.data.data()["profession"]}',
        labelStyle: kHintTextStyle,
        filled: false,
        contentPadding: EdgeInsets.only(left: 10),
      ),  // InputDecoration
    ),  // TextFormField
  ),  // Container
  SizedBox(height: 30, width: 30),
  Container(
    width: 350,
    child: TextFormField(
      decoration: InputDecoration(
        enabled: false,
        labelText: 'Bio: ${snapshot.data.data()["bio"]}',
        labelStyle: kHintTextStyle,
        filled: false,
        contentPadding: EdgeInsets.only(left: 10),
      ),  // InputDecoration
```

Fig.1.7.7 Home screen coding



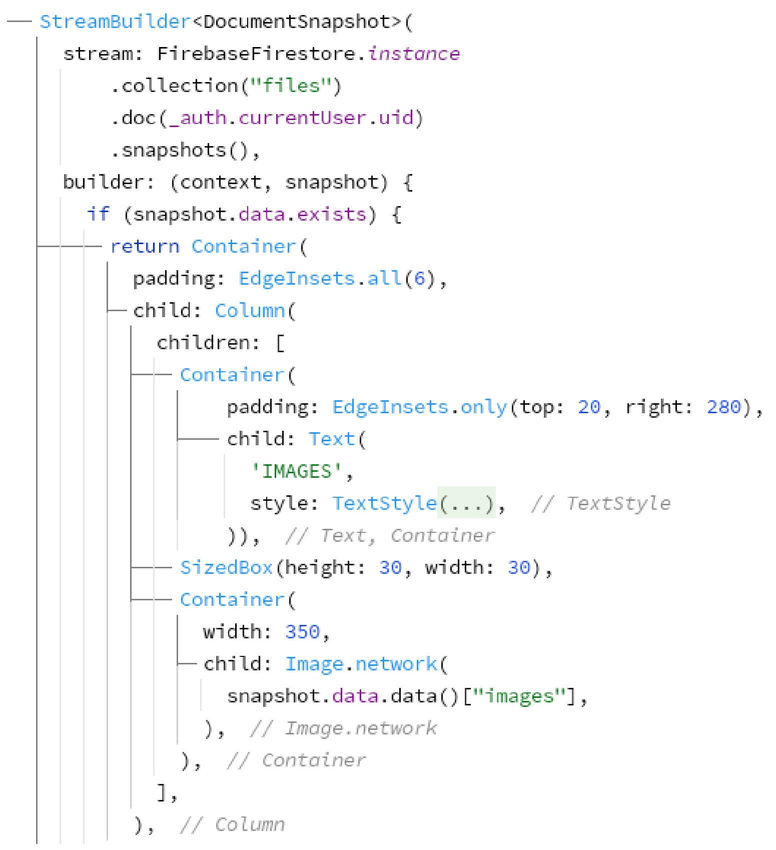
```
      width: 350,
      child: TextFormField(
        maxLines: 4,
        decoration: InputDecoration(
          enabled: false,
          labelText:
              'Workplace/Graduation Status: ${snapshot.data.data()["work"]}',
          labelStyle: kHintTextStyle,
          filled: false,
          contentPadding:
              EdgeInsets.only(left: 10, bottom: 10),
        ),  // InputDecoration
      ),  // TextFormField
    ),  // Container
    Container(
        padding: EdgeInsets.only(right: 210, top: 40),
        child: Text(
          'ABOUT YOURSELF',
          style: TextStyle(
              decorationStyle: TextDecorationStyle.dashed,
              decoration: TextDecoration.underline,
              decorationThickness: 1,
              color: Color(0xFF15317E),
              fontWeight: FontWeight.bold,
              fontSize: 15,
              wordSpacing: 1.5),  // TextStyle
        )),  // Text, Container
```

Fig.1.7.8 Home screen coding



```
  padding: EdgeInsets.only(left: 10, right: 30, top: 5),
  child: TextFormField(
    maxLines: 6,
    decoration: InputDecoration(
      enabled: false,
      hintText:
          'Years of Education: ${snapshot.data.data()["education"]}',
      hintStyle: kHintTextStyle,
      filled: false,
      contentPadding: EdgeInsets.only(
        left: 10,
      ), // EdgeInsets.only
    ), // InputDecoration
  ), // TextFormField
), // Container
SizedBox(height: 30, width: 30),
Container(
  padding: EdgeInsets.only(left: 10, right: 30, top: 5),
  child: TextFormField(
    maxLines: 6,
    decoration: InputDecoration(
      enabled: false,
      hintText:
          'Work/Job Experience: ${snapshot.data.data()["experiences"]}',
      hintStyle: kHintTextStyle,
      filled: false,
```

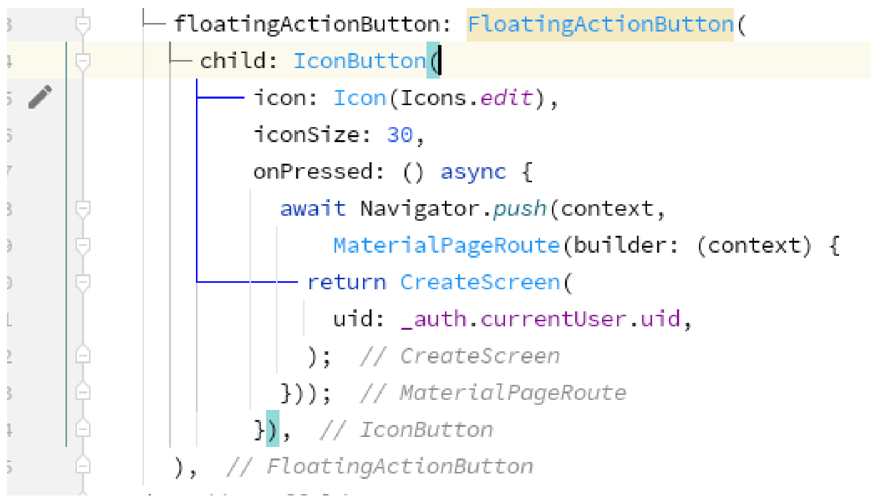
Fig.1.7.9 Home screen coding



```
StreamBuilder<DocumentSnapshot>(
  stream: FirebaseFirestore.instance
      .collection("files")
      .doc(_auth.currentUser.uid)
      .snapshots(),
  builder: (context, snapshot) {
    if (snapshot.data.exists) {
      return Container(
        padding: EdgeInsets.all(6),
        child: Column(
          children: [
            Container(
                padding: EdgeInsets.only(top: 20, right: 280),
                child: Text(
                  'IMAGES',
                  style: TextStyle(...),  // TextStyle
                )),  // Text, Container
            SizedBox(height: 30, width: 30),
            Container(
              width: 350,
              child: Image.network(
                snapshot.data.data()["images"],
              ),  // Image.network
            ),  // Container
          ],
        ),  // Column
```

Fig.1.8.0 Home screen coding

In the below corner of the home screen the edit icon (Fig.1.8.1) is provided for the user that will navigate them to create a screen where they can make changes to the field they want to edit in their portfolio. By clicking on save the update changes will be seen in the same portfolio uploaded in the home screen.



```
floatingActionButton: FloatingActionButton(
  child: IconButton(
      icon: Icon(Icons.edit),
      iconSize: 30,
      onPressed: () async {
        await Navigator.push(context,
            MaterialPageRoute(builder: (context) {
          return CreateScreen(
            uid: _auth.currentUser.uid,
          );  // CreateScreen
        }));  // MaterialPageRoute
      }),  // IconButton
),  // FloatingActionButton
```

Fig.1.8.1 Edit icon on home screen

Log Out:

The logout list tile in the drawer will make the user exit the app. When the user clicks on the log out button it will show them an alert box if they surely want to exit the application or not (Fig.1.8.2). If the user clicks on yes, the application will be finished and if the user clicks on no then the drawer will be closed and it will only show the home screen (Fig.1.8.3 and 1.8.4).

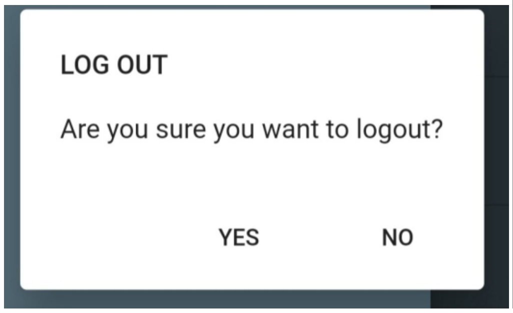
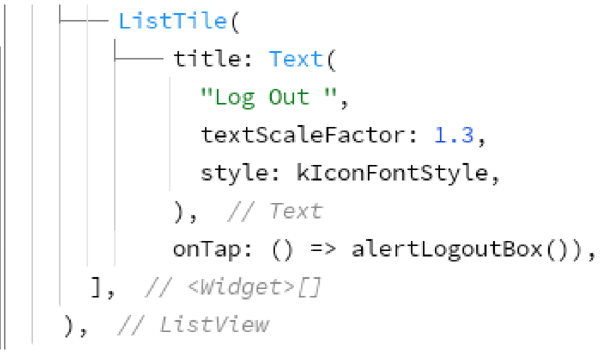
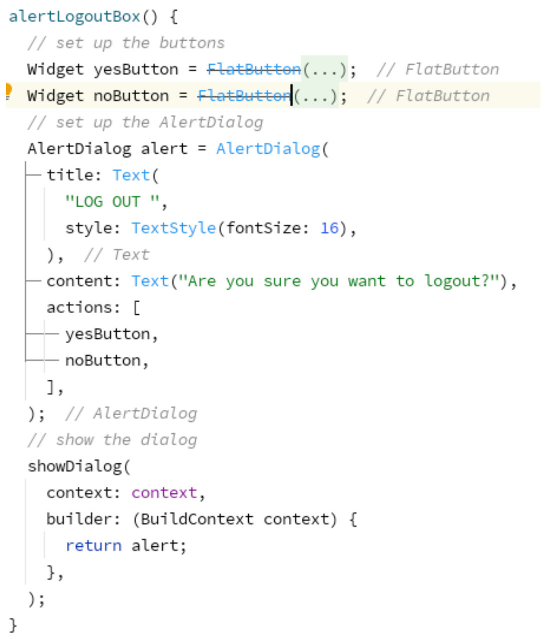


Fig.1.8.2 Logout alert box



```
├── ListTile(
      ├── title: Text(
            "Log Out ",
            textScaleFactor: 1.3,
            style: kIconFontStyle,
          ),  // Text
          onTap: () => alertLogoutBox()),
    ],  // <Widget>[]
  ),  // ListView
```

Fig.1.8.3 Logout button coding



```
alertLogoutBox() {
  // set up the buttons
  Widget yesButton = ~~FlatButton~~(...);  // FlatButton
  Widget noButton = ~~FlatButton~~(...);  // FlatButton
  // set up the AlertDialog
  AlertDialog alert = AlertDialog(
    title: Text(
      "LOG OUT ",
      style: TextStyle(fontSize: 16),
    ),  // Text
    content: Text("Are you sure you want to logout?"),
    actions: [
      yesButton,
      noButton,
    ],
  );  // AlertDialog
  // show the dialog
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return alert;
    },
  );
}
```

Fig.1.8.4 Logout alert box coding