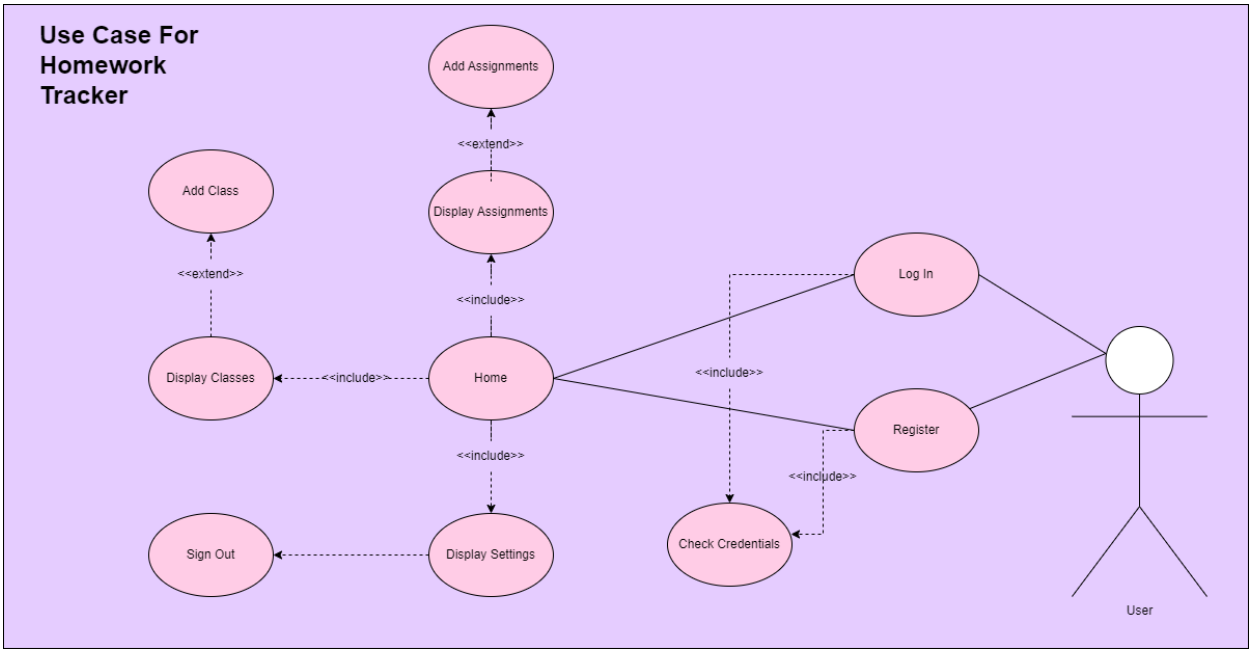
CRITERION B: DESIGN

USE CASE:

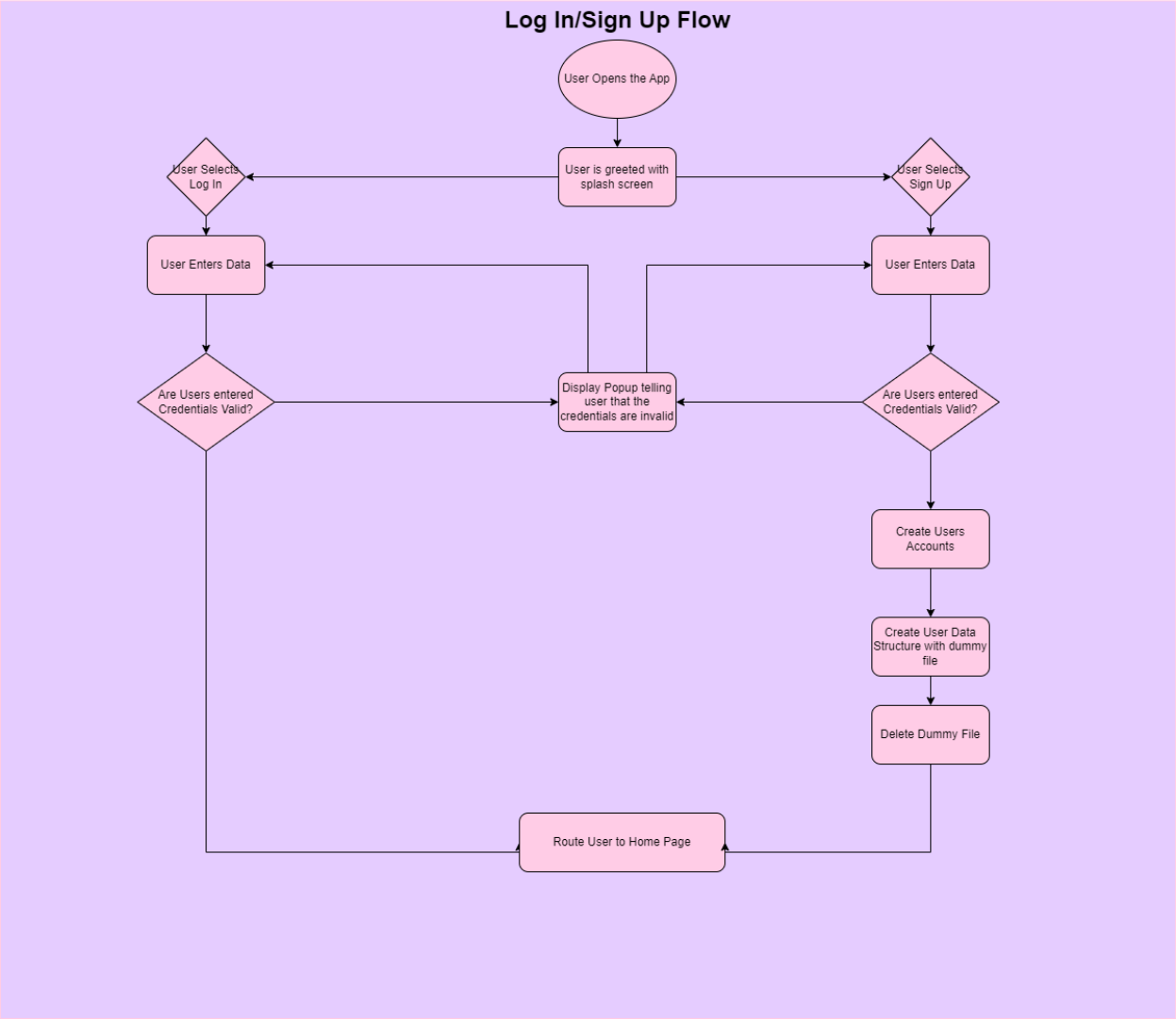
Use Case Diagram:



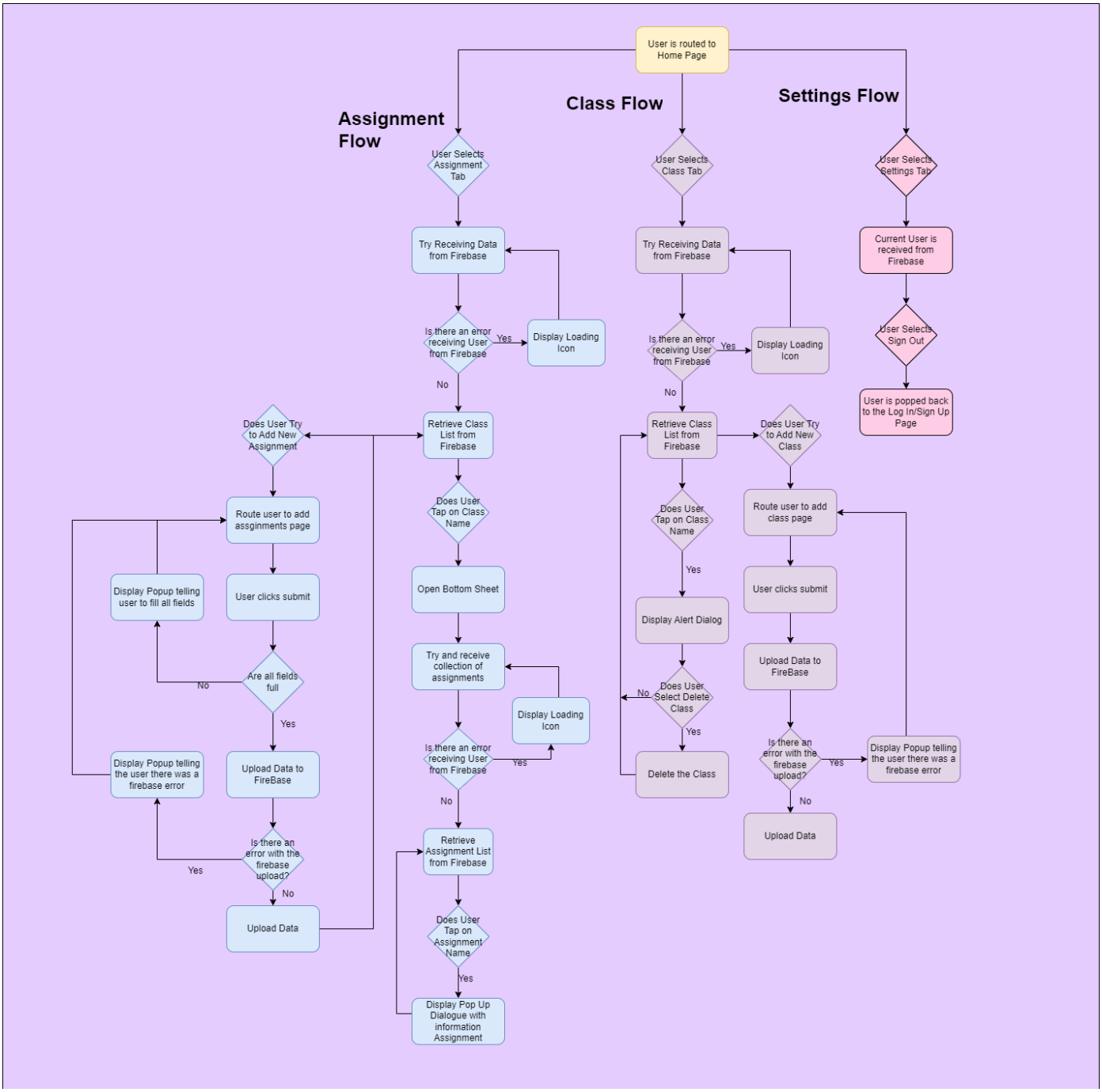
FLOW CHART(S):

The flow charts below represent the flow of using the application as a user navigates through. Proof of previous iterations can be seen in **Appendix D.1 – D.2**

Log In/Sign Up Flow:



App Flow:

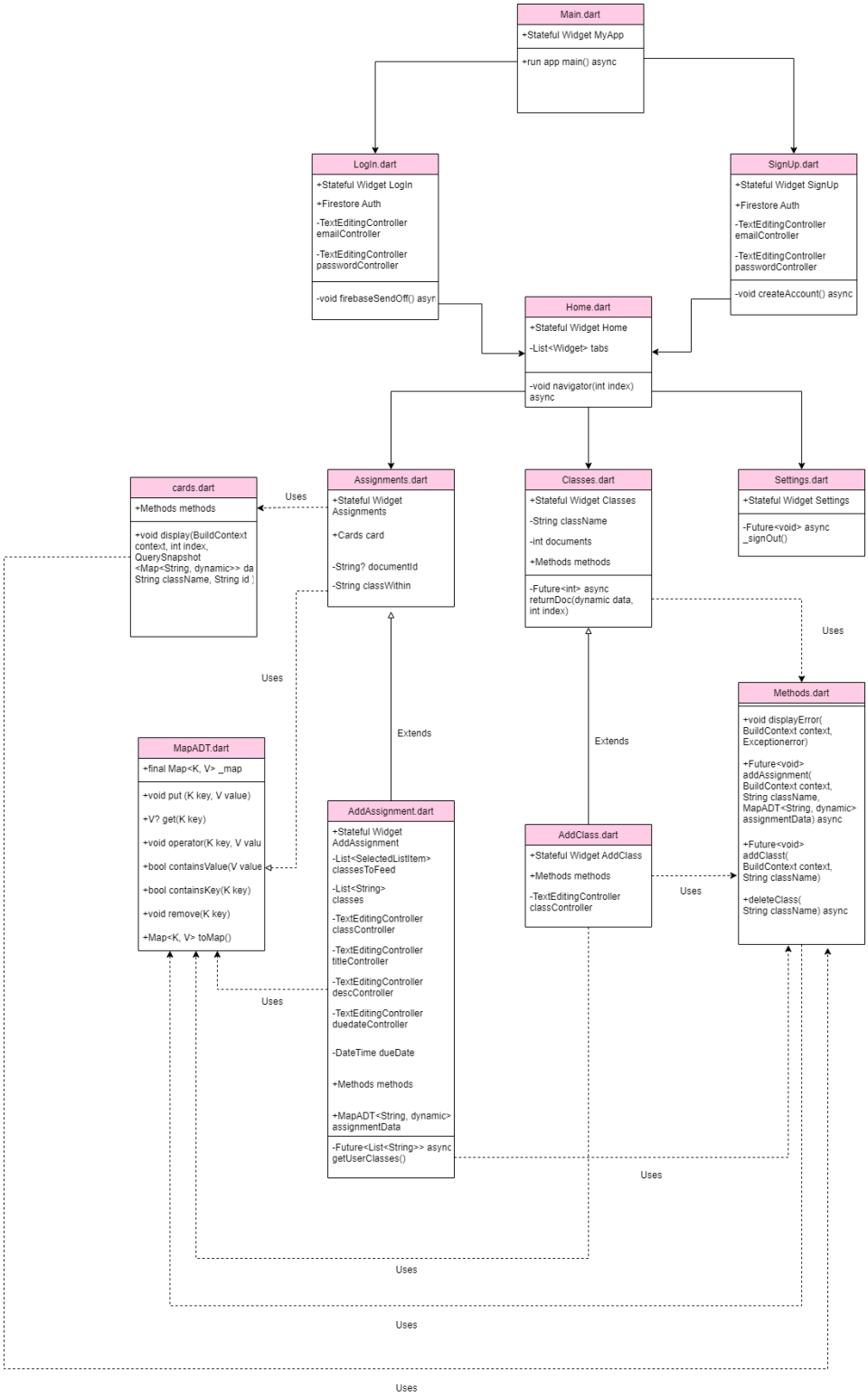


The flow chart is broken up into 3 different Flows to represent the overall apps flow:

- Blue is for the assignment screen flow
- Purple is for the class screen flow
- Pink is for the Settings Screen Flow

UML DIAGRAM:

Iterations of the UML Diagram can be seen in **Appendix D.3 – D.4**



ALGORTHIMS:

Throughout the application there are many algorithms used to control the flow of the app and the UI. The main two are the Stream Builder and the asynchronous methods of the Methods.dart class, used to control the displays of the app. In criterion C, I dive into detail of the usage of these methods. Below is a breakdown of Methods.dart.

```
  deleteClass(String className) async{
    return await FirebaseFirestore.instance.collection('users').doc(FirebaseAuth.instance.currentUser?.email.toString()).collecti
  }
}
```

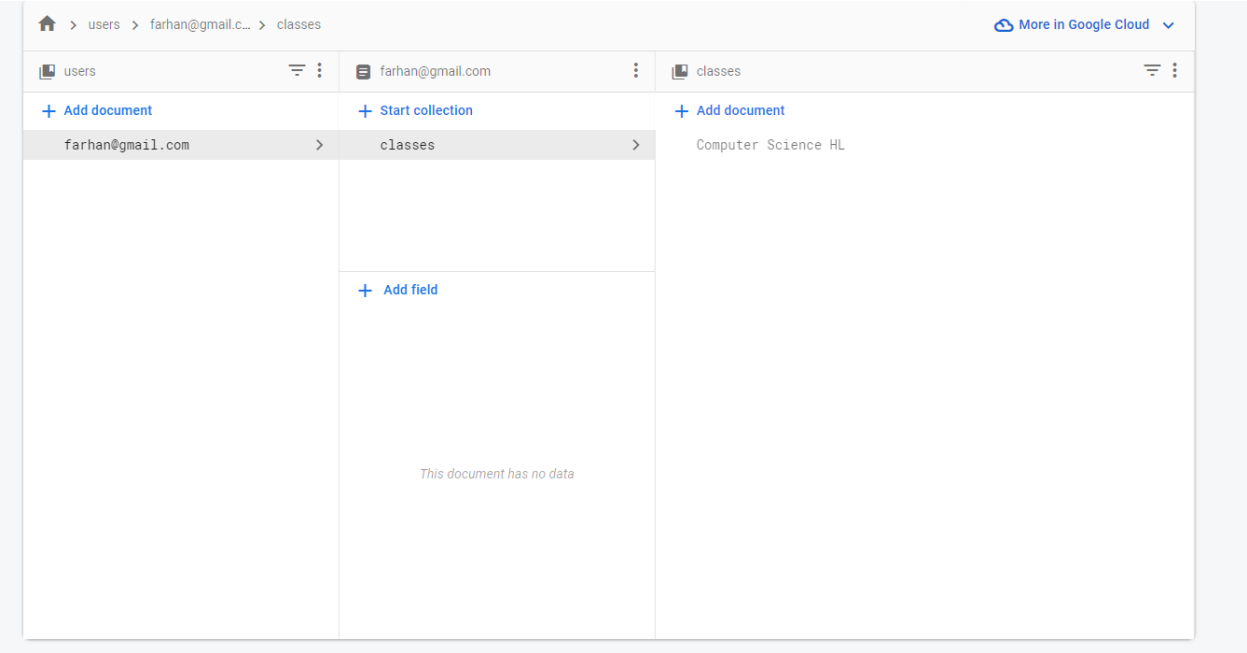
```
Future<void> addClass(BuildContext context, String className) async {
  try {
    await FirebaseFirestore.instance
        .collection('users')
        .doc(FirebaseAuth.instance.currentUser?.email.toString())
        .collection('classes').doc(className).set({});
    //Navigator.pop(context);
  } on Exception catch (error) {
    print(error);
  }
  try {
    await  FirebaseFirestore
        .instance
        .collection('users')
        .doc(
        FirebaseAuth.instance
            .currentUser?.email
            .toString())
        .collection('classes')
        .doc(className).collection('assignments').doc('delete').set({});
    Navigator.pop(context);
  } on Exception catch (error) {
    displayError(context, error);
  }
  await FirebaseFirestore.instance.collection('users').doc(FirebaseAuth.instance.currentUser?.email.toString()).collec
  //createSubcollection(context, className);
}
deleteClass(String className) async{
  return await FirebaseFirestore.instance.collection('users').doc(FirebaseAuth.instance.currentUser?.email.toString())
}

```

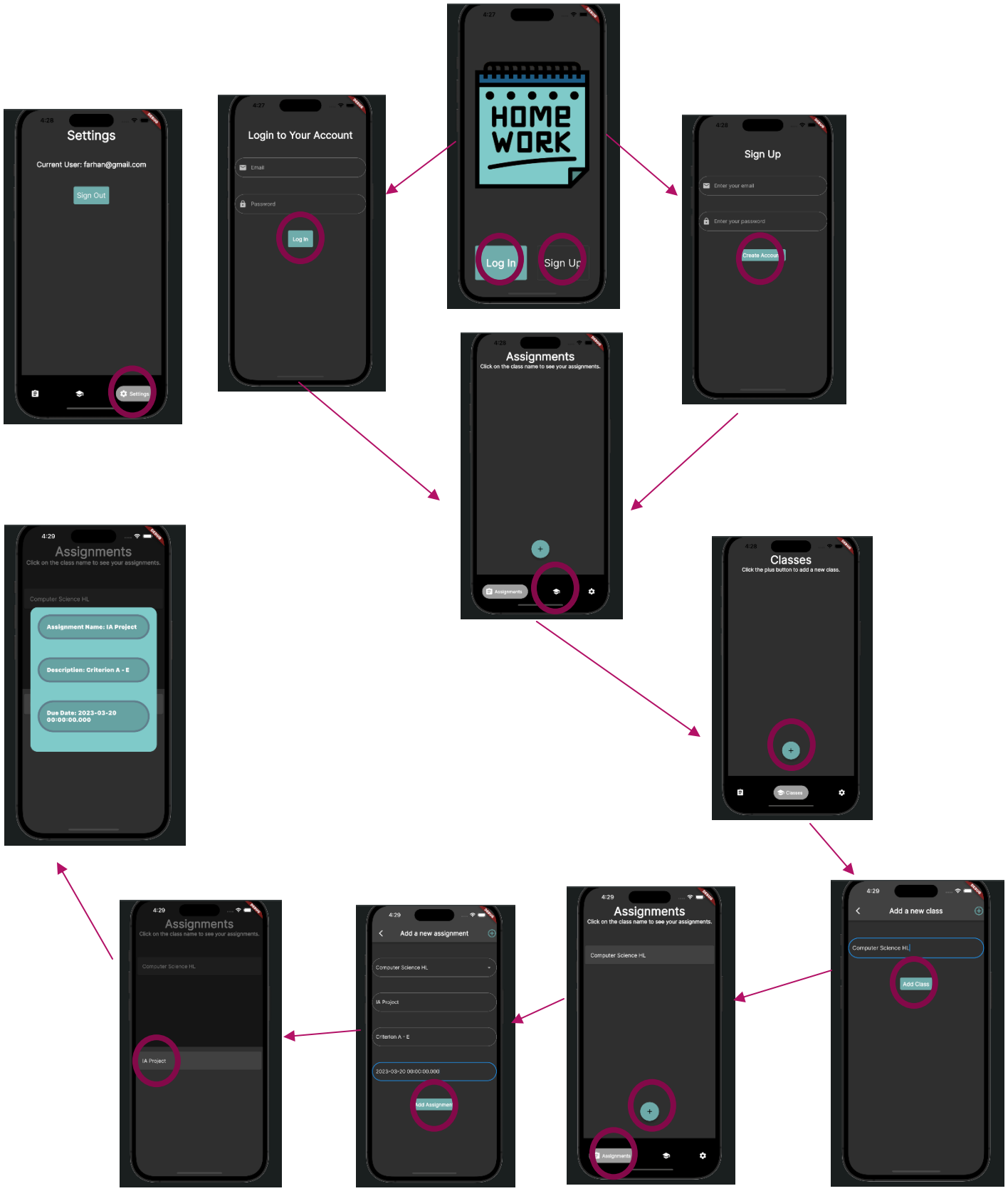
```
Future<void> addAssignment(BuildContext context, String className, MapADT<String, dynamic> assignmentData) async
  try {
    await FirebaseFirestore.instance
        .collection('users')
        .doc(FirebaseAuth.instance.currentUser?.email.toString())
        .collection('classes')
        .doc(className)
        .collection('assignments')
        .add(assignmentData.toMap());
    Navigator.pop(context);
  } on Exception catch (error) {
    displayError(context, error);
  }
}
```

DATA STRUCTURES:

While designing the app, I had to make many decisions regarding what was the most efficient way to store data. To accomplish this I ultimately decided on using my ADT MapsADT, and then transferring this data into separate fields in firestore. Once again this effectively broken down in Criterion C.



UI FLOW:


HOME WORK
Log In
Sign Up
Login to Your Account
Email
Password
Log In
Sign Up
Enter your email
Enter your password
Create Accou
Settings
Current User: farhan@gmail.com
Sign Out
Assignments
Click on the class name to see your assignments.
Computer Science HL
Assignment Name: IA Project
Description: Criterion A - E
Due Date: 2023-03-20 00:00:00.000
Classes
Click the plus button to add a new class.
Add a new class
Computer Science HL
Add Class
Add a new assignment
IA Project
Criterion A - E
2023-03-20 00:00:00.000
Add Assignment

Above is navigation of a first time user adding an assignment and it aims to provide a basic UI flow with example gestures of a user navigating the app. The circles, indicate the users input. Evidence development can be found in **Appendix C.1 – C.2**

OBJECTS:

Class	Description
Main.dart	• Runs App • Creates Routes • Holds the Log In and Sign Up Widgets
LogIn.dart	• Allows user to log in to account using email and password • Accesses Firebase Auth • Controls error handling through Firebase Auth
SignUp.dart.	• Allows user to Sign Up to account using email and password • Accesses Firebase Auth • Controls error handling through Firebase Auth • Creates Documents for user's data • Acesses Method.dart to delete dummy data used to create user collections
Home.dart	• Holds Google NavBar which allows user to access different Widgets
Assignments.dart	• Holds two StreamBuilders, one for assignments and one for classes • Allows users to add assginments by routing to AddAssignment.dart • Accesses card.dart to display data
Classes.dart	• Holds StreamBuilder, which allows user to view enrolled classes • Allows user to create new class through elevatedButton which routes to AddClass.dart Acesses Methods.dart to delete user created data
Settings.dart	• Uses firebase Auth to display current user • Button to signout view Firebase Auth Function
AddClass.dart	• Allows user to add classes, via use of Methods.dart • Acesses firebase to upload data • Handles errors via in class methods and Firebase calls

AddAssignment.dart	• Uses external libraries to ease convience of adding Data • Acesses MapADT for data storage, allows for easy use of user data
Methods.dart	• Uses various methods to create add user data and delete user data
cards.dart	• Uses Firestore to return database in an easy-to-read manner
MapADT.dart	• Custom ADT Map, with method to restore to normal map. • Allows for code expandability and flexibility.

TEST PLAN:

Success Criteria	Test Plan
1. User should be able to add assignments into app.	Add assignments into the app and effectively ensure that they are accessible in both the database and client.
2. App should be able to handle errors with empty fields and ensure user has entered all details of the assignment before continuing with the next step.	User should not be able to confirm any data if TextFields ae left empty. I will check each submit button and ensure that this is the case, and that users can only confirm data that is filled out appropriately.
3. UI should be easy to navigate to ensure that user does not have problems with finding or interacting with current and previous assignments.	Test each UI field and ensure there are no errors with empty data fields or the overall flow of app. All text should be readable.
4. Load time and fetching assignments should have minimal delay (> 2 seconds) to ensure that user experience isn't sacrificed.	I will run the app of launch and reloading the pages and ensure that data is effectively called without delay.
5. User should be able to create accounts with email and password. In addition, password must have strength requirements to ensure security and validity of data.	I will create an account and demonstrate there are adequate password strength requirements.
6. User can add different classes into the app and add assignments to those desired classes.	I will add multiple classes, from which I will create different assignments.
7. User can delete classes and their data.	I will demonstrate that user data can be deleted in the test client.

Word Count: 246