

Attendo Design Document

Team - 13

Angad Kambli, 19114041
Ayush Gupta, 191114017
Chirag Wadhwa, 19114023
Devanshu Chaudhari, 19114027
Shubham Bansode, 19114019

Abstract

Attendo is developed by us as an app to streamline marking attendance in classroom settings. It is developed using the **Flutter** framework and it makes use of **Network Service Discovery** and **Socket Programming** for the networking part and **Firestore** for backend. It aims at making the attendance process quick and easy for both the teachers and the students. The following document will give a thorough description of the methods used in the implementation of the app, the tech stack and the thinking behind selecting it, some design decisions and clever approaches applied by us and the UML diagrams. The code is hosted on GitHub and can be accessed [here](#).

Contents

1	Introduction	2
1.1	Problem Statement	2
1.2	The Solution	2
2	Division of labour	2
3	Use Case Diagram	3
4	Class Diagram	5
5	Sequence Diagram	6
5.1	Mark Attendance Use Case	6
5.2	Take Attendance Use Case	6
5.3	Login Use Case	7
5.4	Register Use Case	7
5.5	View Course Use Case	8
6	Tech stack	8
6.1	Flutter	8
6.2	Firebase	8
6.3	Bonsoir (Network Service Discovery)	9
6.4	Socket Programming	9
7	Architecture and the directory structure	9
8	Implementation of networking	9
9	The Database Schema	10
10	Course Display Image Solution	10
11	Snapshots	11
12	References and tools	17

1 Introduction

1.1 Problem Statement

Classical methods used in classrooms to take attendance are cumbersome and time taking. Also, calling out names also gives a rise to the issue of not being able to maintain the sanctity of records due to students marking proxies. Also, it is disadvantageous for both professors and students with respect to availability of data. For the professor, it is a pain to physically comb through the records to check whether a certain student fulfills the minimum criteria. For the student, he has to keep a record of how many classes he has attended and if he doesn't, there are chances he would never understand his attendance is dangerously low before it's too late. With the availability of mobile devices, it is high time that we start making use of it for taking attendance.

1.2 The Solution

With **Attendo**, we bring a solution to all these qualms. With a simple and well-designed UI, it can be easily introduced in classroom settings. It makes sure every device can mark attendance only once thus guaranteeing the sanctity of records. The professor can easily check the attendance percentage of his students and can also check every lecture's record which might prove useful in detecting patterns in attendance statistics. With the records easily accessible at any time, a student can correct his habits well in time if his attendance percentage is low.

2 Division of labour

While all the team members were involved in the brainstorming for deciding the app's flow and implementation, there was a clear division of labour with regards to the domains a certain person would be responsible for. Following are the contributions of various members :

1. **Angad Kambli, 19114041** : Initialization of project, implementation of socket programming and network service discovery, coding the login and registration screen UI and other UI fixes, maintaining the GitHub repository(reviewing PRs, solving merge conflicts and so on), the course image solution, designing the icon and other minor fixes.
2. **Ayush Gupta, 19114017** : Setting up Firebase in the app, setting up authentication logic for the user, writing database queries for read and writes, writing controller functions to clean data recieved from database and add minor functionality in registration pages.
3. **Chirag Wadhwa, 19114023** : Implementing the flow of taking and marking attendance, implementing the logic to prevent multiple attendance markings from the same device, Major final changes to bring all the implementation together and to remove all hard coded values and replace them with database query calls, final bug fixes and testing.
4. **Devanshu Chaudhari, 19114027** : Designing the UI for the App. Coding most of the Screens including the Dialogue Boxes showing successful and failed errors , designing hard-coded List Views and minor visual changes to the UI to make the design more attractive and comprehensible.
5. **Shubham Bansode, 19114019** : Writing database queries for read and writes, writing controller functions to clean data recieved from database, final screenshots.

3 Use Case Diagram

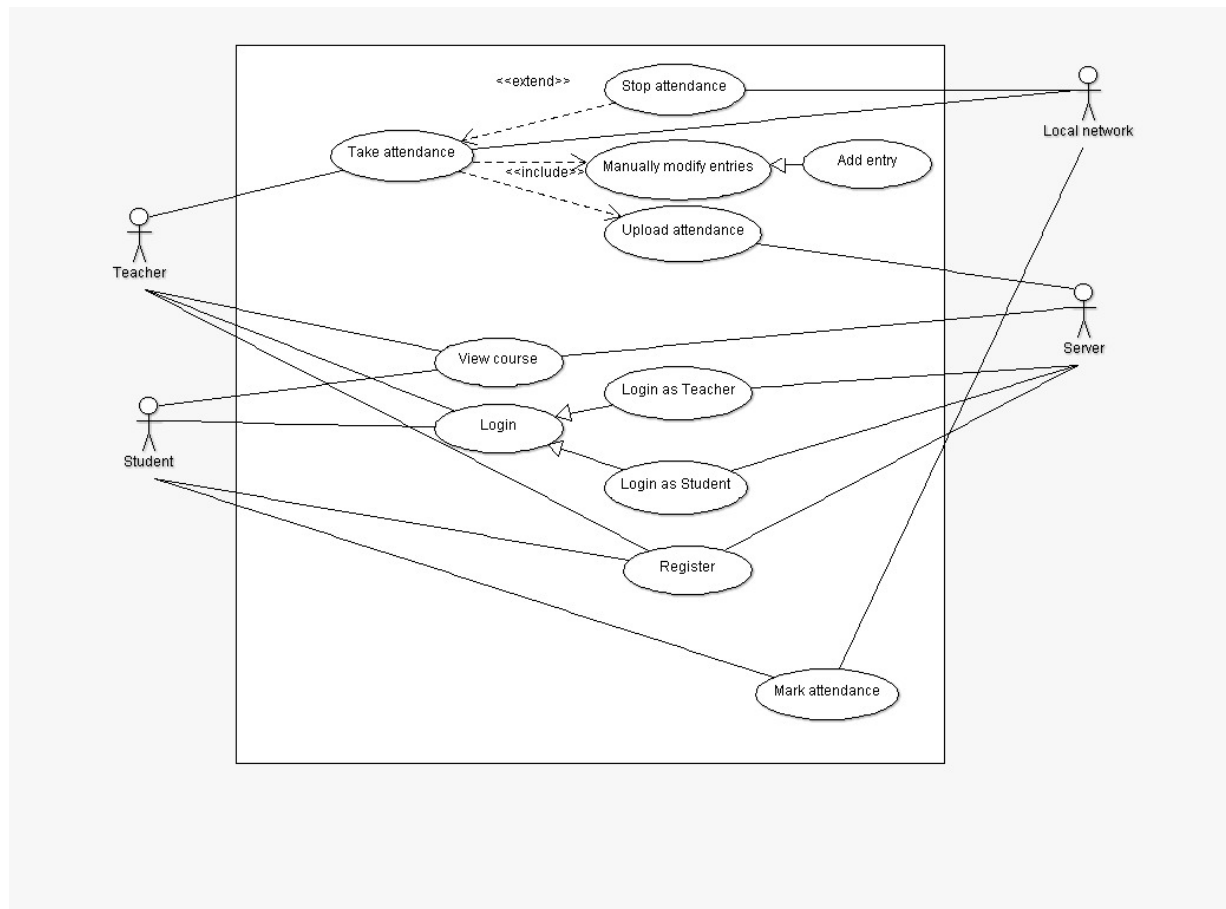


Figure 1: Use Case Diagram

Use Case 1 : Register

Actor : Student, Professor

Pre-conditions : Is not already registered

Basic Flow :

- Scenario 1 : Mainline Sequence
 1. Student/Professor : Enter credentials and tap on the register button.
 2. AuthService : Pass the credentials to the database for checking validity.
 3. Database: Check credentials for validity.
- Scenario 2: At Step 4 of Mainline Sequence
 4. Database: return the user id if credentials are valid.
 5. AuthService: display successful dialog box.
- Scenario 3: At Step 4 of Mainline Sequence
 4. Database: return null if credentials not valid
 5. AuthService: print error message.

Post-Conditions: Is registered

Use Case 2: Login

Actor : Student, Professor

Pre-conditions : Is a registered user

Basic Flow :

- Scenario 1: Mainline Sequence
 1. Student/Professor : This use case starts when a user wishes to log in into the System.
 2. System : Request user to enter Login Information i.e Email ID and Password.
 3. Student/Professor : Insert Email and Password.
 4. Controller: Validate the entered Email ID and password.
 5. Controller : Confirm Log in
- Scenario 2: At Step 5
 5. Controller : Show error

Post-Conditions: Is logged in

Use Case 3 : Mark Attendance

Actor : Student,Local Network

Pre-conditions : Is logged in as a student and is also registered in course

Basic Flow :

- Scenario 1 : Mainline Sequence
 1. Student : Click Mark Attendance Button
 2. Local Network: Discover service,get ip and port of server
 3. Local Network: Connect to server using ip and port
 4. Local Network: Communicate enrollment number to server
- Scenario 2 : At Step 5 of Mainline Sequence
 5. Local Network: Send “OK” message from server to Student
 6. Controller: Displays attendance success dialogue
- Scenario 3: At Step 5 of Mainline Sequence
 5. Local Network: Send “Error” message from server to Student
 6. Controller: Displays attendance failure dialogue

Post-Conditions: If the IP address is unique,attendance is marked for the selected course.

Use Case 4 : Take Attendance

Actor : Professor,Local Network

Preconditions : Is logged in as a professor and the instructor for the course

Basic Flow:

- Scenario 1: Mainline Sequence
 1. Professor: Click on take attendance button
 2. Local Network: Register network service
 3. Local Network: Start server
 4. Local Network: Await enrollment number from students
 5. System: Check for duplicate ip address
- Scenario 2: Step 6 of Mainline Sequence
 6. System: If unique ip address, add student enrollment number
 7. Local Network: Send “ok” message to student
- Scenario 3: Step 6 of Mainline Sequence
 6. System: If duplicate ip address, don't add student enrollment number
 7. Local Network: Send “Error” message to student

- Scenario 4: Step 8 of Mainline Sequence

8. Professor: Click on Manually add attendance button
9. System: Ask for enrollment number of student
10. Professor: Enter enrollment number
11. System: After maximum time(3 minutes),ask local network to close server and deregister service
12. Local Network: Close server
13. Local Network: Deregister network service
14. System: Upload attendance list to database

Post-Conditions: Attendance is taken and the records are uploaded to database

Use Case 5 : View Course

Actor : Student, Professor

Preconditions : Is logged in

Basic Flow:

- Scenario 1 : Mainline Sequence

1. Student/Professor : Select Course
2. System: Request Course Details
3. Database: Return Course Details
4. System : Format and Display details accordingly.

Post-Conditions: User can view course details for the selected course

4 Class Diagram

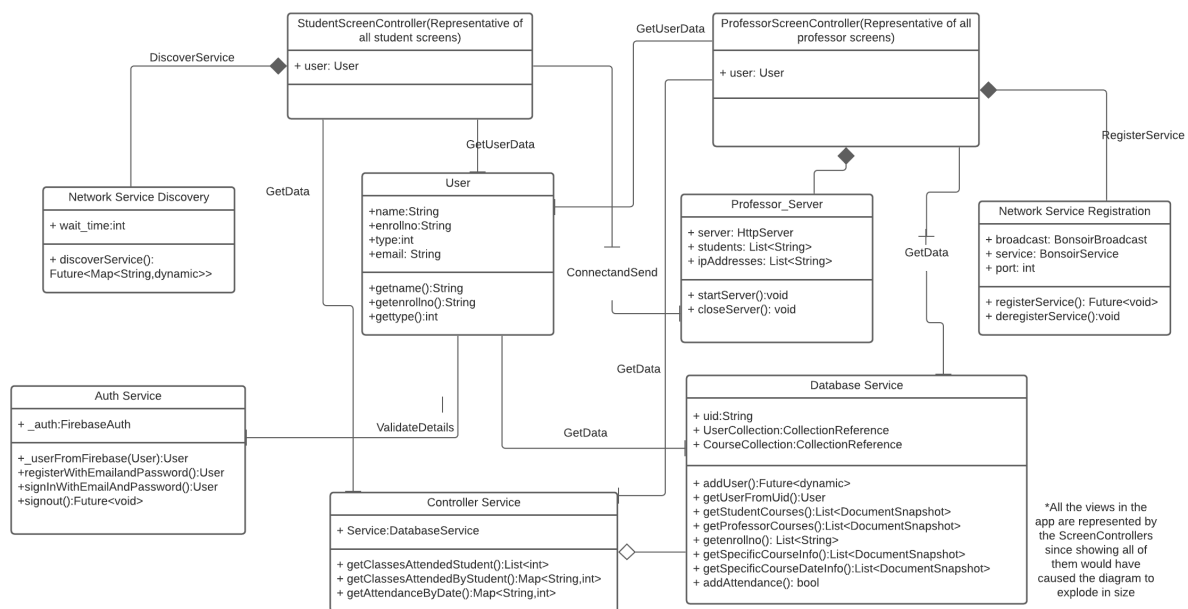


Figure 2: Class Diagram

5 Sequence Diagram

5.1 Mark Attendance Use Case

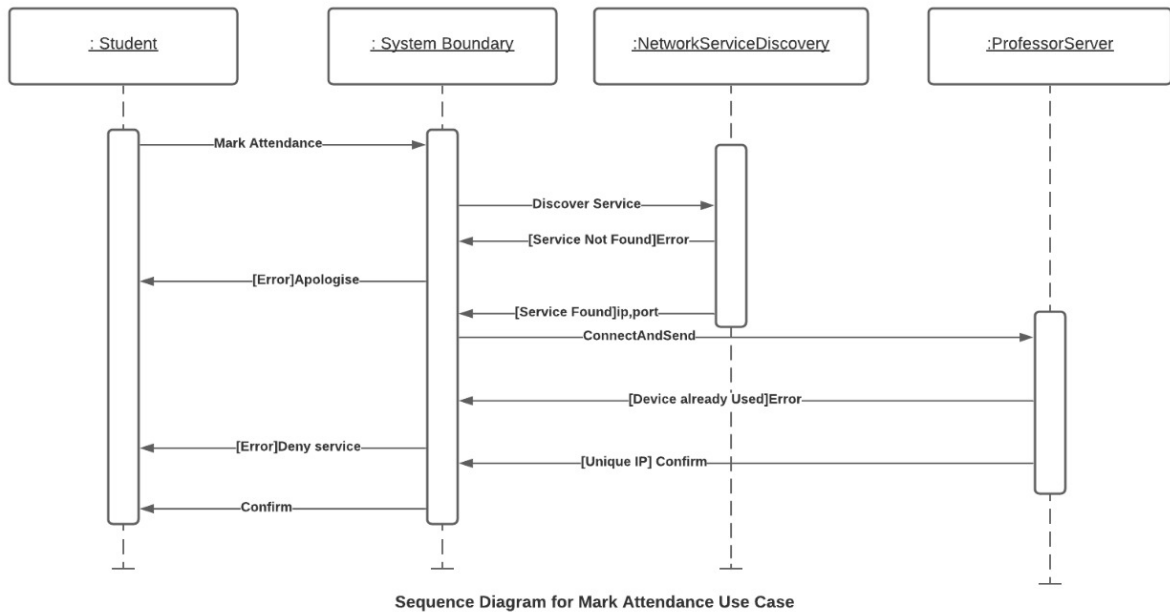


Figure 3:

5.2 Take Attendance Use Case

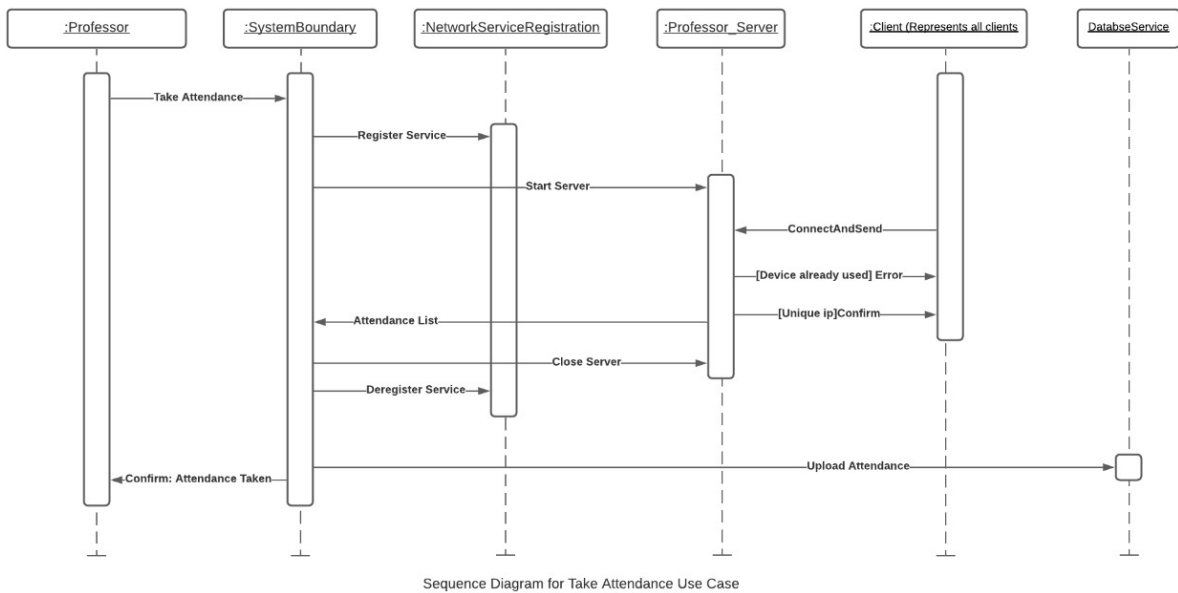
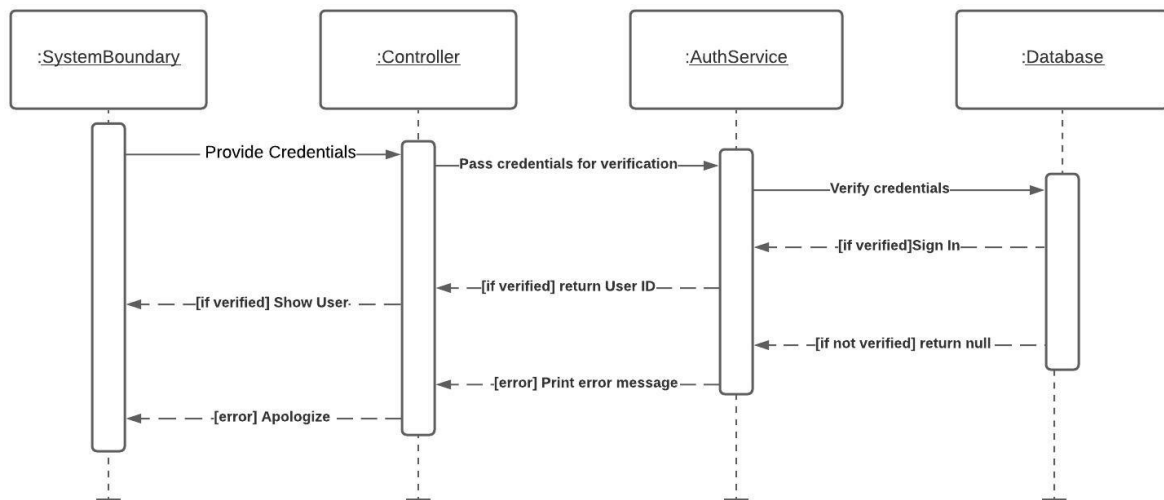


Figure 4:

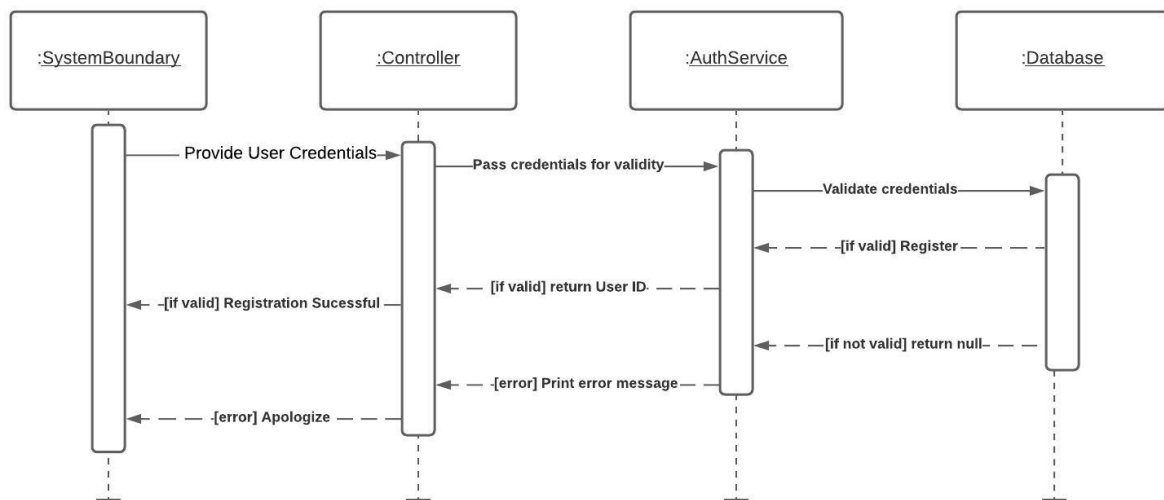
5.3 Login Use Case



Sequence Diagram for Login Use Case

Figure 5:

5.4 Register Use Case



Sequence Diagram for Register Use Case

Figure 6:

5.5 View Course Use Case

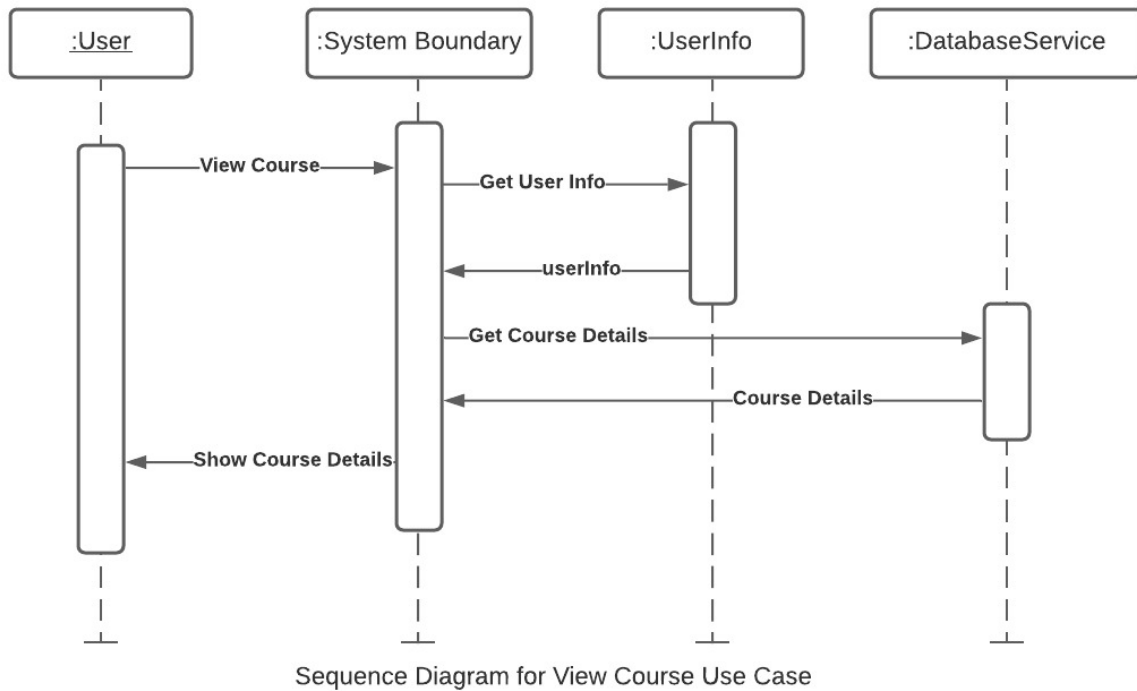


Figure 7:

6 Tech stack

The different frameworks, technologies and libraries used for the implementation will be discussed in the following subsections.

6.1 Flutter

Flutter is a framework developed by Google mainly for increasing the productivity of Mobile development workflow but has also been extended for web app development. Here, we had the choice of either going with Flutter or going the route of native Android development. The motivation behind selecting Flutter was some of the added benefits that Flutter gives which includes but are not limited to :

- Flutter can build for both iOS and Android with very less extra coding. This is in stark contrast to native Android or iOS development which requires a lot of redundant coding. However, for now, we have built the app only for Android but a build for iOS wouldn't be a huge undertaking in case the app needs to be actually used in classroom situations.
- Flutter provides the hot reload feature which really speeds up development. Native Android coding requires us to reinstall the app to check for new changes which can become a real pain once the app increases in size.
- Flutter unifies the UI design and the logic which is a huge relief from native Android development's requirement for writing XML files for each screen.
- Last but not the least, Flutter provides some really good widgets that require little or no customization before they look good.

6.2 Firebase

Firebase was used as the backend for the app. This was a no-brainer choice since the backend requirements of the app aren't very complex and Firebase provides an easy to setup interface. It was used by us for both Authen-

tication and Database handling. Firebase provides two types of databases. The one we used is Cloud Firestore. It is a noSQL (JSON based) database. The app includes an API key which the app uses to communicate with Firebase.

6.3 Bonsoir (Network Service Discovery)

Bonsoir is an open source Flutter library that we used for Network Service Discovery. Network Service Discovery is used for registering the attendance service over the network from the professor side and is then used from the student side to get the host and port of the professor's phone. In native Android development, this would have been an in-built feature but since Flutter is coded in dart, we have to use libraries that further get ported to both Android and iOS.

6.4 Socket Programming

Socket programming was done using the TCP protocol. On professor's side, a server is created which the student side communicates through socket programming.

7 Architecture and the directory structure

The architecture is loosely based on the **Model-View-Controller(MVC)**. We say loosely because the division of labour between the models, views and controllers that is usually expected wasn't followed as strictly by us. Some of controller workload is shared by the models and some by the views. The views are in the screens since its a convention to call the views as screens in Flutter development. The file *main.dart* is the entrypoint for the app and the flow of code starts from there itself. The directory structure is as follows :

```
lib
├── models
│   └── all the model files
├── controller
│   └── all the controller files
├── screens
│   └── all the views or screens
├── networking
│   ├── socket-programming.dart
│   └── network-service-discovery.dart
└── main.dart (entrypoint)
```

8 Implementation of networking

The actual act of taking attendance is divided into two parts on both the professor and the student sides. The two parts are concerned with network service discovery(NSD) and socket programming respectively.

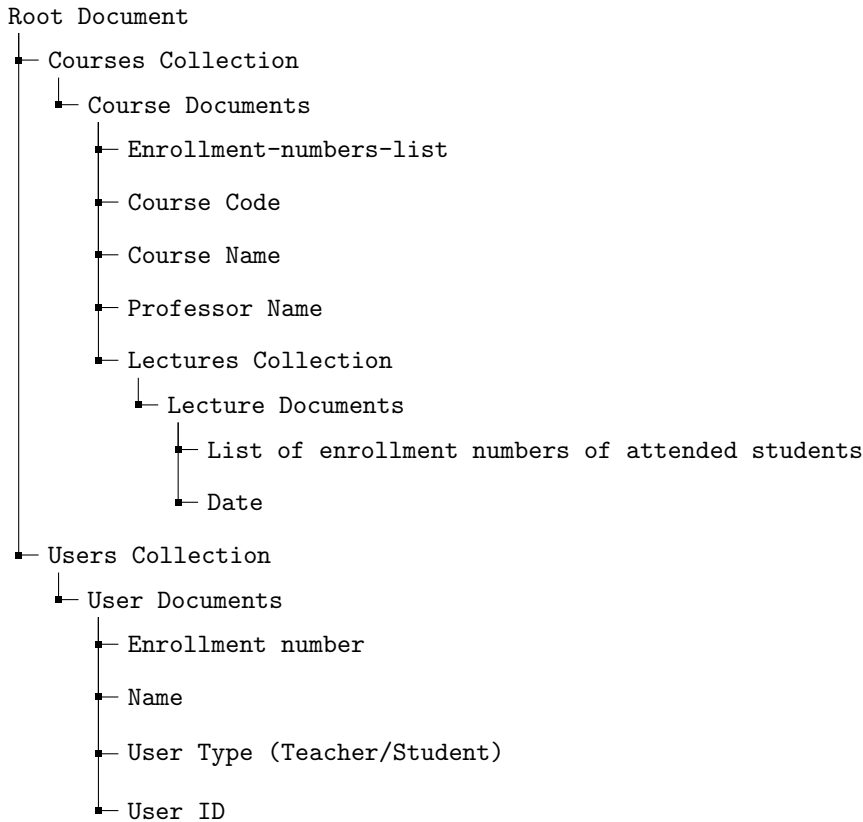
Let's delve into the details of the professor's flow. Once the professor taps on take attendance, first a service is registered on the network with the host and port and the service name using NSD. Next, a TCP server is started by the professor's phone and it starts listening for connections. Once a connection is recieved, it checks whether it is from an IP address that hasn't already marked attendance and adds the enrollment number to the attended list accordingly. Since this is happening on the local network, students can't fool the system by using a VPN. Also, by ensuring the same IP can't mark attendance twice, we make sure that the student can't mark attendance twice using the same device.

On the student side, the phone searches for the service using nsd and gets the required host and port. It then connects to the service, sends the enrollment number and disconnects.

The networking is done in TCP and NSD is implemented using a flutter library *Bonsoir*.

9 The Database Schema

Firebase includes includes alternating JSON based documents and their collections in its database. The following is the Database Schema :



10 Course Display Image Solution

We had hard-coded the course display image for most of our development period. However, it is important to have a recognizable and good looking image associated with every course to improve the user experience. Now, to give the option for users to customize the display pictures for every course would be a major undertaking since it would involve storing each and every image which wouldn't be desirable since such storage requirements would hurt the scalability of the project.



Figure 8: The Course Display icons

Keeping this in mind, we created ten abstract, simple and snappy icons for different courses. While making it seemingly random to the user, the image is still chosen deterministically to ensure that it remains constant for any particular course. We decided to take the modulo of the number formed by the last two digits of the course code as the index that would select one icon out of the ten we made.

This would make the home page much more vivid and also improve the User Experience since we tend to associate images and text and having that remembrance value always helps.

11 Snapshots

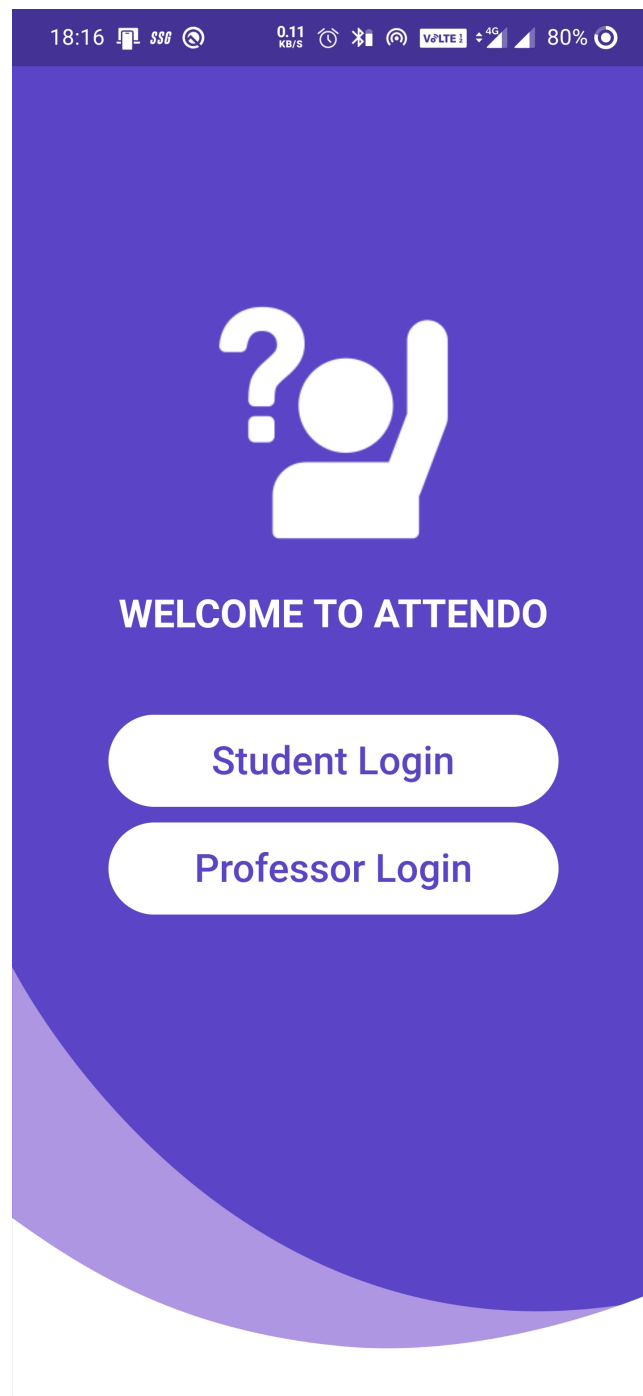









Figure 9: Auth Landing Screen

16:40



0.51 KB/S






VOLTE

4G

59%



So, you don't have an account?
Sign up below!

Student

Email

Full name

Unique ID






Password

Confirm password


I have an account!


Submit

16:40



9.48 KB/S






VOLTE

4G

59%



So, you don't have an account?
Sign up below!

Teacher

Email

Full name

Unique ID

Password

Confirm password

I have an account!

Submit

Figure 10: Student and Professor registration

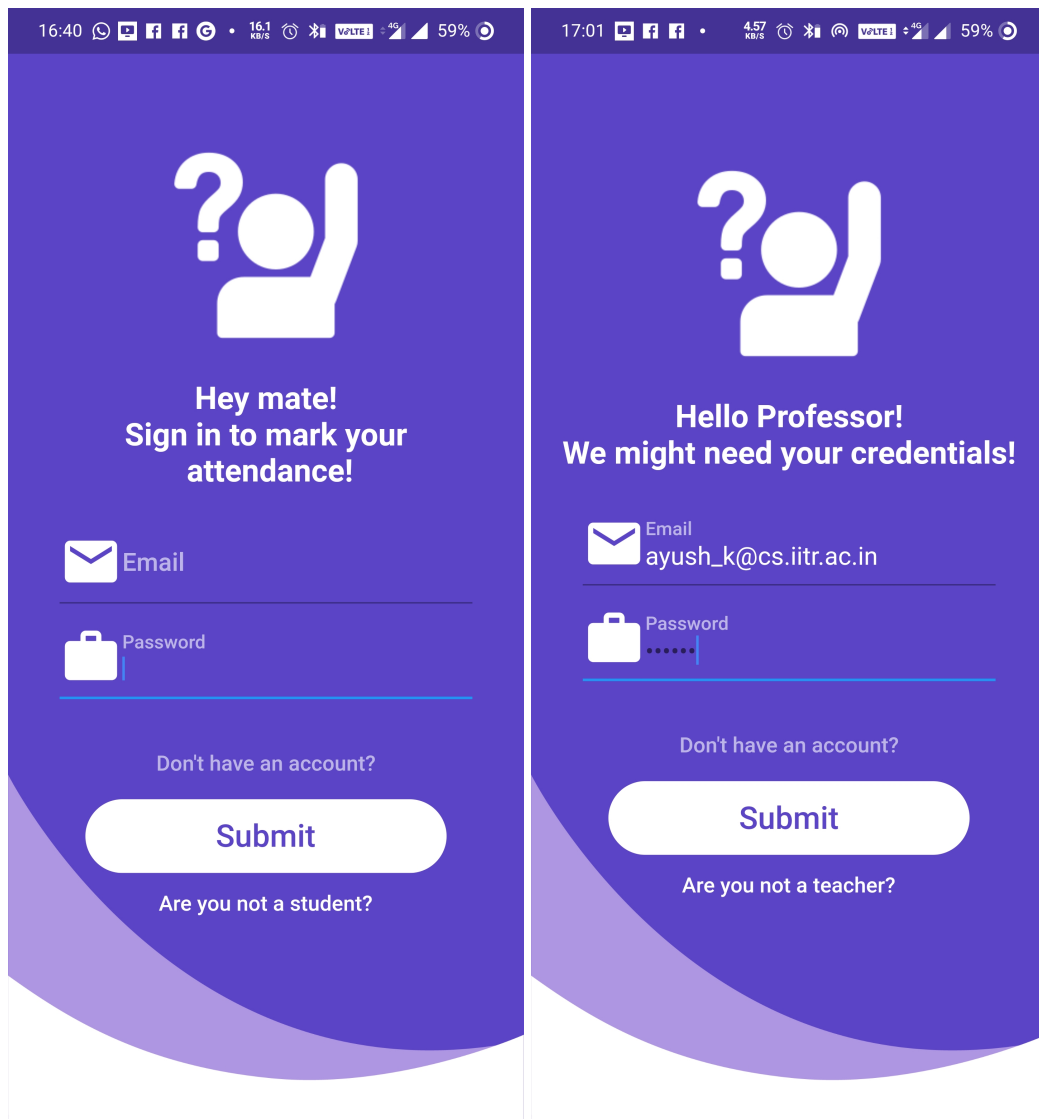


Figure 11: Student and Professor Login

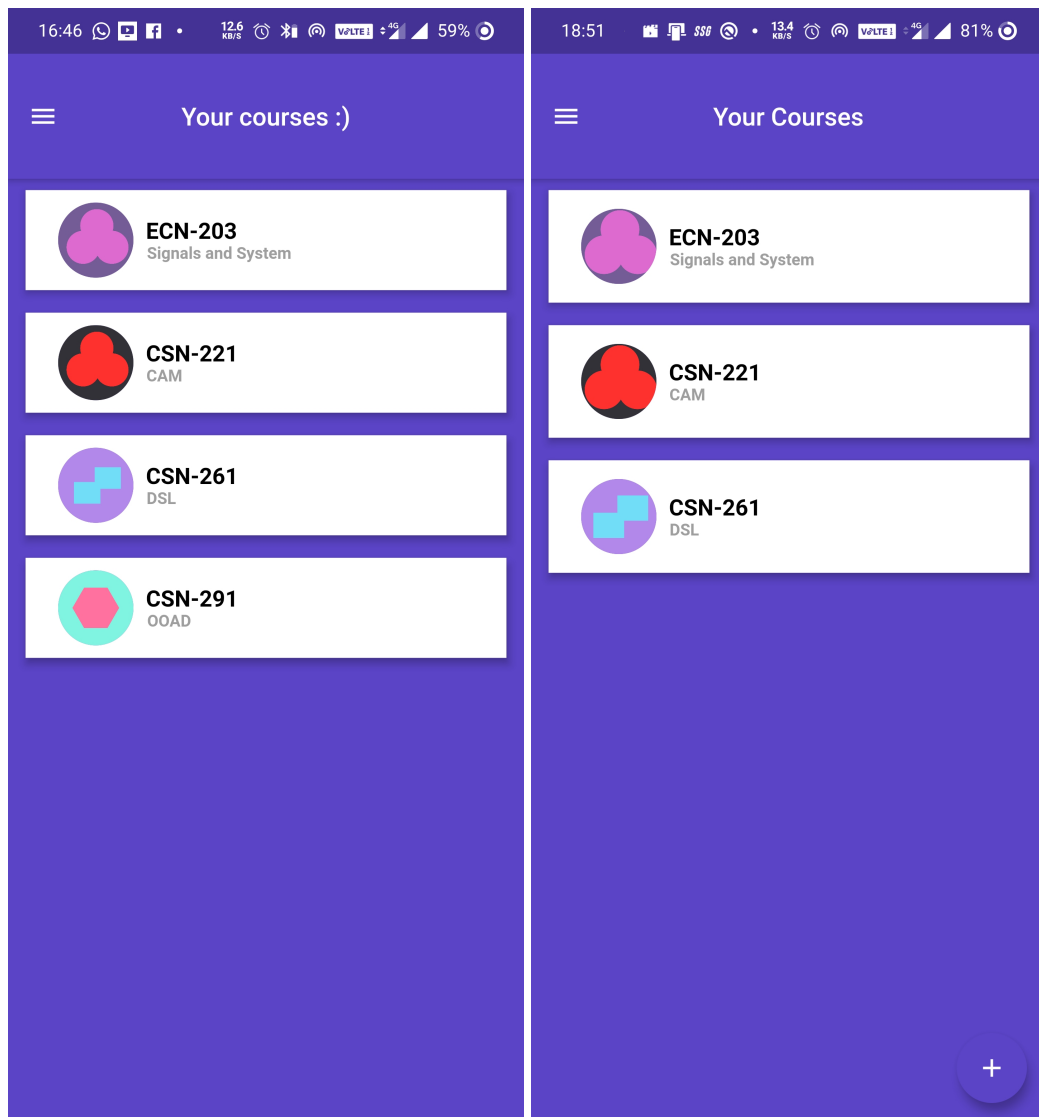


Figure 12: Student and Professor Home Page

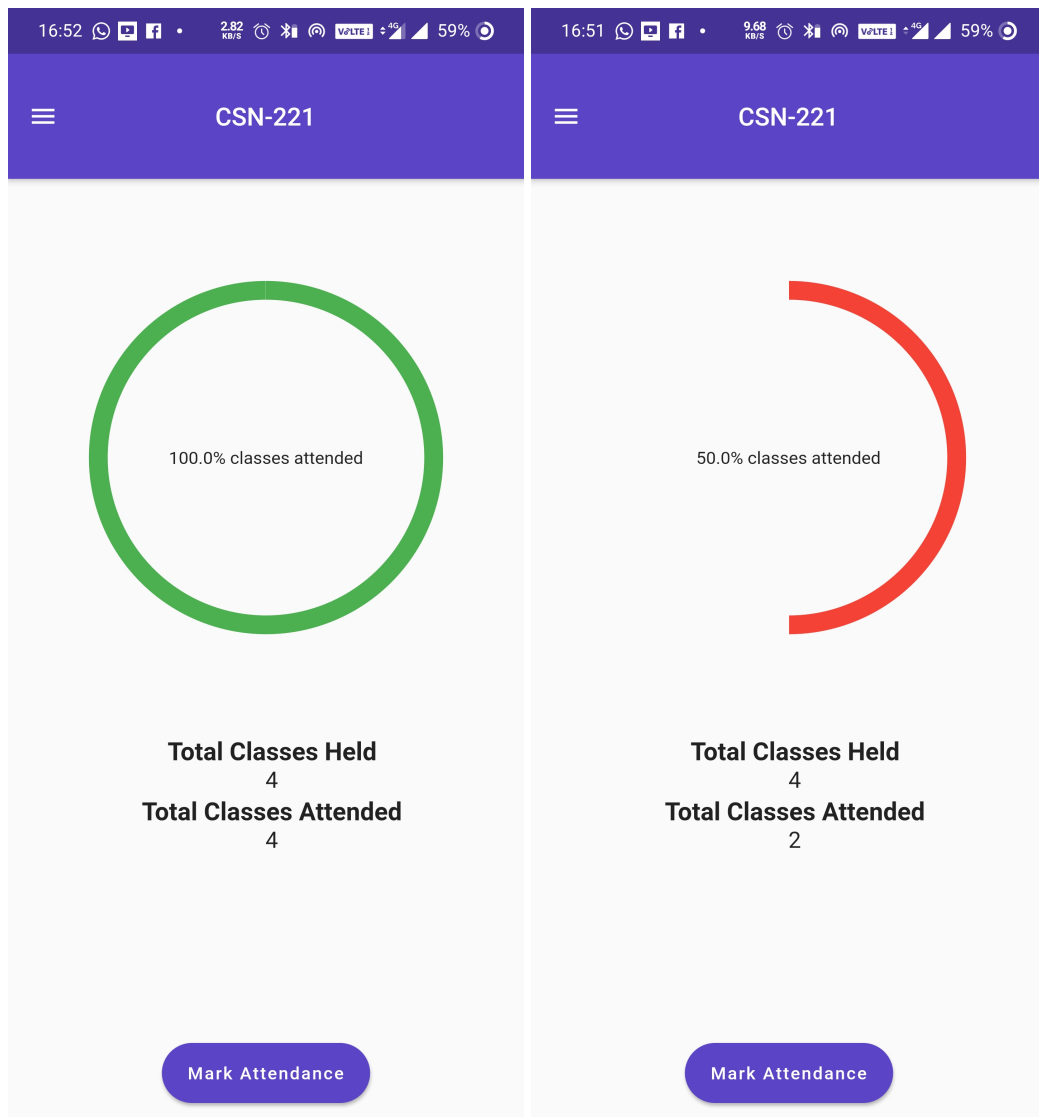


Figure 13: Attendance bar

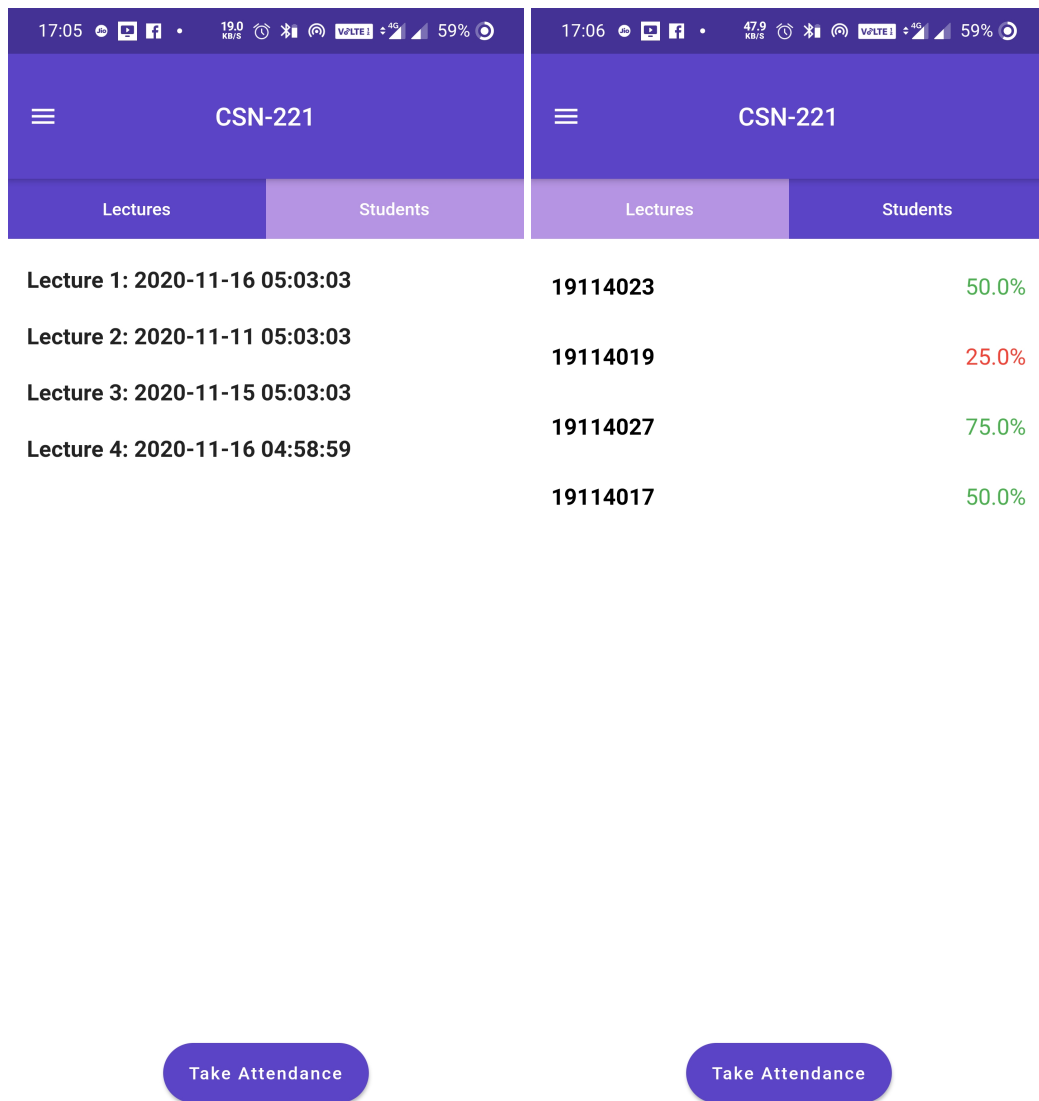


Figure 14: Lecture details and Student Stats

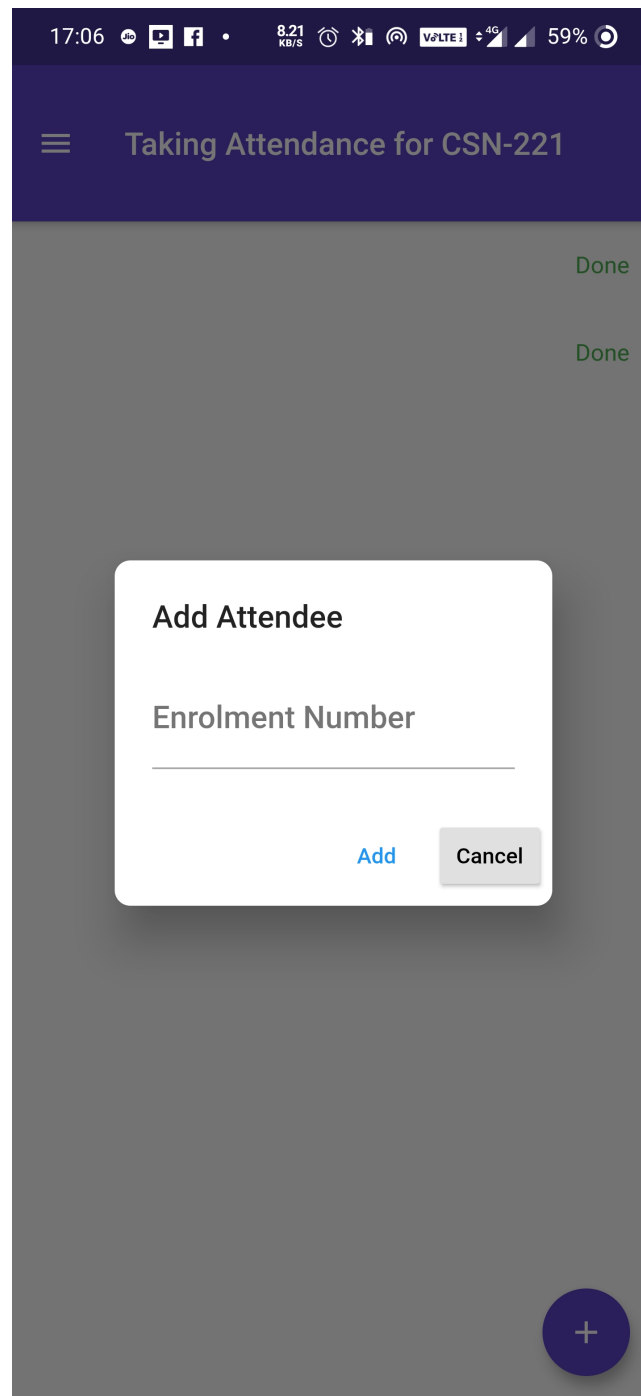


Figure 15: Add Student attendance manually

12 References and tools

Firebase related links :

1. <https://firebase.google.com/docs>
2. <https://www.youtube.com/watch?v=sfA3NWDBPZ4&list=PL4cUxeGkcC9j--TKIdkb3ISfRbJeJYQwC>
3. <https://www.youtube.com/playlist?list=PL1-K7zZEsYLluG5MCVEzXAQ7ACZBCuZgZ>
4. <https://medium.com/flutterdevs/using-firebase-firestore-in-flutter-b0ea2c62bc7>

Networking related links :

1. <https://jamesslocum.com/blog/post/67566023889>

2. <https://developer.android.com/training/connect-devices-wirelessly/nsd#kotlin>
3. <https://pub.dev/packages/bonsoir>

Flutter related links :

1. <https://flutter.dev/>
2. <https://api.flutter.dev/flutter/dart-core/dart-core-library.html>