**Title/Cover page** should include (with ABSOLUTELY NO SPELLING MISTAKES!)

University Name

College

Department

List of Team Members including IDs and Majors

Date of submission

Report itself should include the following sections

Table of Contents

[Acknowledgments 4](#_6g2xs4h0x6vg)

[Abstract 5](#_nzi0f4go4uk)

[1. Introduction 6](#_u9xnhgex557k)

[1.1. Background/literature survey 6](#_e2ssgwjki2xr)

[2. Requirement Analysis 7](#_xftitghzmuwa)

[2.1. Functional Requirements 7](#_341bg9t0z0fz)

[2.1.1. Instructor Functional Requirements 7](#_sky13najinsm)

[2.1.2. Student Functional Requirements 9](#_iqyvlagann0j)

[2.2. Security Requirements 10](#_i7n3we997818)

[2.3. Other Non-Functional Requirements and Constraints 11](#_w0yh6ftbfroh)

[2.4. Risk Assessment 13](#_pyb4k4xhcx0x)

[2.5. Applicable standards 14](#_byixikdoiny)

[2.6. Requirement Specification 16](#_jx35qxszwxko)

[2.7. Project Plan 18](#_4rxh404z0kdy)

[3. Project Design 21](#_v2aw61pt319g)

[3.1. Architecture Diagram 21](#_3qt0c37k6ozy)

[3.2. UML Flowchart/Activity Diagram 22](#_z61dbsyfbw02)

[3.2.1. Instructor Diagram 22](#_bmppsaqt7zyb)

[3.2.2. Student Diagram 23](#_gouczhcn5fo8)

[3.3. User Interface Design 24](#_c5chs0qvu4z4)

[3.3.1. Instructor Mockup 24](#_ekkdxxofo39w)

[3.3.2. Student Mockup 33](#_ml8kyaeafmez)

[3.4. Database Design (if applicable) 39](#_4nqolkrprwax)

[3.5. Sequence Diagram Design 40](#_3e59qs1p3ogb)

[3.5.1. Instructor Features 40](#_ednghd2kst1g)

[3.5.2. Student Features 43](#_qzq48zq731yz)

[3.6. Algorithm Design (if applicable) 45](#_5tfhfb4513rw)

[3.6.1. Face Comparison Algorithm 45](#_xvakaa7ztk4z)

[3.6.2. Security Considerations 46](#_ctyd8pbg61bz)

[3.6.3. Security & Non-Functional Considerations 46](#_5qfz30r67iz)

[4. Implementation 49](#_awgpukw5yda1)

[4.1. Face Recognition Attendance System 49](#_9qq1llnlhlsq)

[4.2. WiFi-Based Attendance Broadcast System 50](#_wip6rdt7ydlt)

[4.3. Student-Side WiFi Signal Detection for Attendance Marking 55](#_csefzev7sema)

[5. Testing and Verification 57](#_ob9bmkmp10mm)

[5.1. Functional Testing 57](#_jzpcclsjovzk)

[5.2. Security Testing 58](#_oij9omm2seyl)

[5.3. Non-Functional Testing 59](#_kf5rfau8bdz6)

[6. Conclusions 60](#_6osuilrrv423)

[7. Future Work 61](#_bmvzc2ozm6f5)

[References 62](#_bs8dwle8dffg)

[Appendix 63](#_x4vn1t8mend7)

[Appendix A: Application Screenshots 63](#_ygxgqllhh9tx)

[Appendix B: System Architecture Diagram 63](#_7brcv7234mno)

[Appendix C: Attendance Algorithm Diagram 63](#_r0zt43ayt5ph)

[Appendix E – Firebase Firestore Sample Structure 64](#_8aiax6tpb75)

# Acknowledgments

I would like to sincerely thank my project supervisor for their valuable guidance and continuous support throughout the development of the Attendance Management System. I’m also grateful to my family and friends for their constant encouragement and motivation during this journey. A special thanks to my university for providing the resources and learning environment that enabled me to successfully complete this project. Lastly, I truly appreciate everyone who shared their feedback and suggestions to help enhance the quality and functionality of this system.

# 

# Abstract

The Attendance Management System is a mobile application that helps instructors manage class sessions and student attendance in an organized and accurate way. The main goal of this system is to make attendance easier and more secure using WiFi verification and AI-based face matching.

There are two types of users: Instructors and Students. Instructors can create courses, add students, schedule sessions, and set custom rules for what is considered Present, Late, or Absent. When an instructor starts a session, the app checks if their mobile WiFi is turned on. If it is, the app saves the WiFi name and IP address. Only students who are connected to the same WiFi network can mark their attendance.

To make sure the right student is marking attendance, the app asks the student to take a live selfie. This image is matched with their stored photo using AI face verification. If a student has no photo in the system, they must upload one before they can mark attendance.

The system uses role-based access and automatically marks students as Present, Late, or Absent based on the session rules and attendance time. The use of the same WiFi network makes sure only students physically present in the class can mark their attendance, which helps prevent proxy attendance.

Overall, the Attendance Management System saves time, improves accuracy, and offers a better classroom experience. In the future, features like performance reports, analytics dashboards, and academic system integration can be added to make it even more useful.

# 

# Introduction

The Attendance Management System is a mobile app built using Flutter for the front-end and Firebase for the database. It helps instructors and students manage classroom sessions and attendance more easily. The main purpose of this app is to reduce manual work and improve the accuracy of tracking attendance.

In the past, many instructors used paper sheets or Excel files to take attendance. This process was slow and often caused mistakes. Our app solves this problem by providing a smart and fast way to handle attendance.

Instructors can create sessions for each course and set their own rules for what counts as Present, Late, or Absent. When a session begins, the app checks if the instructor's WiFi is turned on. If it is, the system saves the WiFi name and IP address.

On the student side, when they open the course, the app checks if the student's phone is connected to the same WiFi network as the instructor. If it is, the attendance button is enabled. If not, the student cannot mark their attendance. This method helps ensure that only students who are physically in the classroom can mark attendance.

To prevent fake attendance, the app also uses AI-based face verification. Students must take a live selfie when marking attendance. The app compares this photo with the one saved in the system. If the face matches and the student is connected to the correct WiFi, the system records their attendance based on the instructor's time rules.

Instructors can also see which students are Present, Late, or Absent in real-time. The app stores attendance history so both students and instructors can view past records. All information is safely stored in Firebase and updated live.

## Background/literature survey

Managing course attendance can be hard and take a lot of time, especially when done by hand. In many schools and universities, instructors still face problems in keeping track of attendance properly. Old methods like paper sheets or Excel files are slow, not very accurate, and easy to make mistakes.

For example, it’s difficult for instructors to keep track of which students arrived late or didn’t show up. They also have to manually check each student’s identity, which can take a lot of time and isn’t always correct. These old methods are also easy to cheat, and some students may mark attendance for others (proxy attendance).

Our research shows that many educational institutions still don’t use smart attendance systems. Most of them have not adopted mobile apps or technology that can help with real-time and secure attendance tracking. There is also very limited use of AI to verify student identity.

This project aims to fix these problems by using a mobile app that works with WiFi-based verification and AI-based face matching. It helps instructors confirm that students are physically present and ensures only the correct student marks their attendance. The system is more reliable, saves time, and makes the attendance process smarter and more secure.

# Requirement Analysis

## Functional Requirements

List all the major project functional requirements in a bullet format. List the sub-requirements/features under the major requirement:

### Instructor Functional Requirements

| ***Authentication & Account Management*** | |
| --- | --- |
|  | The instructor shall be able to register an account by providing required details.   * Email should be validated before registration * The password should follow basic security standards * Instructors should be required to confirm password before registration |
|  | The instructor shall be able to log in using valid credentials.   * Support the "Forgot Password" feature * Support the "Remember Me" feature for future logins |
|  | The instructor shall be able to recover the password using the "Forgot Password" functionality. |
|  | The instructor shall be able to log out of the application securely. |
| ***Dashboard*** | |
|  | Upon login, the Instructor shall be redirected to a personalized dashboard. |
|  | The dashboard shall display a list of all courses created or managed by the Instructor. |
| ***Course Management*** | |
|  | The instructor shall be able to create a new course by entering the required course details. |
|  | The instructor shall be able to view a list of all created courses. |
|  | The instructor shall be able to update course details when needed. |
|  | The instructor shall be able to delete a course from the system. |
| ***Student Management*** | |
|  | The instructor shall be able to add multiple students to a specific course. |
| ***Session Scheduling*** | |
|  | The instructor shall be able to schedule a session for any course.   * The instructor shall enter the session title, date, and time. * The instructor shall set a custom time for "Late" and "Absent" marking. |
| ***Session Management with WiFi Verification*** | |
|  | The instructor shall be able to view a list of all scheduled and live sessions. |
|  | The instructor shall be able to start a session at the scheduled time. |
|  | When the instructor clicks the “Start Attendance” button:   * Before starting, the system will check if WiFi is turned ON on the instructor's mobile. * If WiFi is OFF, the system will show a message asking the instructor to turn it ON. * If WiFi is ON, the system will store the connected WiFi name (SSID) and IP address. * After that, the attendance button will be enabled on the student side for that course. |
| ***Dynamic Attendance Evaluation*** | |
|  | The system shall calculate attendance status based on thresholds:   * Mark as Present if the student marks within the allowed time. * Mark as Late if marked after the Present limit but within Late threshold. * Mark as Absent if attendance is marked after both thresholds. |
| ***Live Attendance Monitoring*** | |
|  | The instructor shall be able to view the real-time attendance list during the session.   * Students shall be categorized as Present, Late, or Absent. |

### Student Functional Requirements

| ***Authentication & Account Management*** | |
| --- | --- |
|  | The student shall be able to register for an account by providing the necessary details. |
|  | The student shall be able to log in using valid credentials. |
|  | The student shall be able to recover the password using "Forgot Password". |
|  | Students shall be able to log out securely. |
| ***Course Access*** | |
|  | The student shall be able to view the list of enrolled courses. |
|  | The student shall be able to view course details, including:   * Course description * Instructor name * List of sessions |
| ***Live Session Access*** | |
|  | Students shall be able to view live sessions for enrolled courses. |
| ***WiFi-Based Attendance Access*** | |
|  | * When a session is live, the student will open the course. * The system will check if WiFi is turned ON and connected. |
|  | The system will verify if the student's device is connected to the same WiFi name (SSID) as the instructor.   * If matched, the attendance button will be enabled. * If not matched, the button will be disabled, and the student won’t be able to mark attendance. |
| ***Mark Attendance with Live Selfie*** | |
|  | * The student will tap the "Mark Attendance" button. * The system will ask the student to take a live selfie. |
|  | The system shall compare the selfie with the previously uploaded image using AI:   * If matched, attendance will be marked. * If the student has no image in the database, the system will require the student to upload a profile image first. * Attendance will be marked as Present, Late, or Absent based on thresholds set by the Instructor. |
| ***Attendance History*** | |
|  | The student shall be able to view the list of all attendance sessions. |
|  | Students shall be able to view their attendance status (Present, Late, Absent) for each session. |

## Security Requirements

The mobile application ensures user data protection and secure access through the following security features:

| ***User Authentication*** | |
| --- | --- |
|  | The system requires each user (Instructor or Student) to register with a unique email and a secure password. |
|  | Only authenticated users can access the application. |
|  | Firebase Authentication is used to manage secure login and registration. |
| ***Password Constraints & Protection*** | |
|  | Passwords must follow strong rules (at least 8 characters, include capital letters, numbers, and symbols). |
|  | Passwords are never stored in plain text. Firebase securely stores them using hashing techniques. |
|  | Password fields are hidden in the UI during user input. |
|  | The "Forgot Password" feature is implemented using Firebase’s secure reset email system. |
| ***Role-Based Authorization*** | |
|  | The app supports different roles (Instructor and Student) with separate permissions and access to features. |
|  | Firebase Authentication and Firestore rules are used to enforce role-based access. |
| ***Data Integrity & Safety*** | |
|  | Firebase Firestore security rules ensure that users can only read or write data they are authorized to access. |
|  | All communication between the app and Firebase is encrypted via HTTPS to prevent data interception. |
| ***Session Management*** | |
|  | Users can securely log out of the app, which clears their session. |
|  | Firebase automatically handles session tokens securely and refreshes them when needed. |
| ***No SQL Injection*** | |
|  | Since Firebase is a NoSQL database and uses document-based access, there is no risk of SQL injection. |
|  | Firebase rules prevent unauthorized access to documents even if someone tries to manipulate requests. |

## Other Non-Functional Requirements and Constraints

This section outlines the key non-functional requirements and constraints related to the design, development, and deployment of the mobile attendance system.

| ***Technical Requirements*** | |
| --- | --- |
|  | The system must be developed using Flutter for cross-platform mobile compatibility (Android & iOS). |
|  | Firebase will be used for user authentication, real-time data storage, and file handling. |
|  | The app must support real-time updates for attendance and session activities. |
|  | WiFi-based verification must be used instead of Bluetooth:   * When the instructor starts a session, the app checks if WiFi is turned on. * It saves the WiFi name and IP of the network the instructor is connected to. * The student can only mark attendance if their device is also connected to the same WiFi. |
|  | The system should be scalable to handle at least 500 concurrent users (students + instructors). |
|  | Important actions like login and attendance marking should take less than 2 seconds under normal internet speed. |
| ***Usability Requirements*** | |
|  | The app interface should be simple and easy to navigate for both instructors and students. |
|  | The UI should be responsive and accessible on different screen sizes. |
|  | Onboarding (registration, login, dashboard access) should take no more than 3 steps for new users. |
|  | The app must clearly show helpful messages (e.g., “WiFi is off, please turn it on.”, “You are not connected to the instructor’s WiFi”, “Face verification failed”) during attendance. |
| ***Reliability & Security*** | |
|  | The system should have 99.9% uptime, especially during peak academic hours. |
|  | All sensitive data (like passwords and profile images) must be securely stored and transmitted. |
|  | The attendance system should only allow real users by using:   * WiFi match verification (same network as the instructor). * Face verification to confirm identity and avoid proxy attendance. |
| ***Maintainability & Scalability*** | |
|  | The codebase must follow clean architecture and modular design to allow easy maintenance and future upgrades. |
|  | The backend structure (Firebase Firestore & Storage) should be organized to support the addition of new modules like performance tracking or announcements. |
| ***Device & Platform Constraints*** | |
|  | The application is limited to mobile devices only and does not have a web or desktop version. |
|  | The app is optimized for Android 8.0+ and iOS 12.0+ devices. |
|  | The app must be usable without external hardware (except camera for face verification). |
| ***Time and Cost Constraints*** | |
|  | The project must be developed and tested within one academic semester. |
|  | Only free-tier Firebase services and open-source Flutter libraries/tools will be used to minimize development cost. |

## Risk Assessment

During the development of this mobile-based attendance management system, several types of risks may occur. These risks can be technical, related to project management, or due to external factors. Below is a summary of the possible risks and how we plan to handle them.

* **Technical Risks**
  + Sometimes, Firebase may go offline, which can affect login or real-time data updates. To handle this, we will use local caching where possible and let users know if something goes wrong. The system will automatically retry once the internet is back.
  + The face verification system may not always work properly if the student is in poor lighting or if their face has changed. To reduce this problem, we will guide users on how to take a good selfie and ask them to retake the image if needed.
  + Some old smartphones may not support all Flutter features or camera functionality. We will test the app on different devices and set a minimum version requirement for both Android and iOS to make sure it works well.
  + If not handled carefully, sensitive data like passwords or photos could be exposed. We will use hashed passwords, enable secure HTTPS connections, and apply Firebase Security Rules to keep user data safe.
  + The system depends on both the instructor and the student being connected to the same WiFi network. If either device is not connected or WiFi is off, attendance cannot be marked.
    - Show a warning if WiFi is turned off.
    - Guide users to connect to the correct WiFi before continuing.
    - Test the feature on multiple devices and WiFi networks for reliability.
* **Project Management Risks**
  + We have a limited time to complete the project within the semester. To stay on track, we’ll plan the work in weekly phases and prioritize the most important features first.
  + If team members are not properly aligned, the project can be delayed. To avoid this, we will hold regular meetings and use tools like Trello to track progress and make sure everyone is on the same page.
  + Adding too many new ideas during development can slow down progress. We will fix the scope early and only add new features if time allows after finishing the core functions.
* **External Risks**
  + Since Firebase needs an internet connection, users might face issues if they go offline. We will build the app to alert the user and save data locally, then sync it once the internet returns.
  + Events like a pandemic or natural disaster can interrupt progress. To manage this, we’ll keep everything backed up online and stay connected through virtual platforms to continue working from anywhere.

## Applicable standards

For the authentication and password management process in this project, the National Institute of Standards and Technology (NIST) guidelines were followed. NIST is a non-regulatory federal agency that develops standards and guidelines to improve technology and economic security.

**Password Guidelines:**

* **Password Length**: Minimum of 8 characters.
  + **Reason:** A password length of 8 characters is considered more secure than shorter passwords, making it harder for unauthorized parties to guess.
* **Password Complexity:** Complex password requirements (capital letters, special characters, etc.) are not enforced.
  + **Reason:** Requiring highly complex passwords often leads users to create weak passwords (e.g., substituting letters with numbers), which can compromise security. This approach is aimed at reducing password fatigue and improving security without unnecessary complexity.

**Firebase Authentication**

* Firebase Authentication, which adheres to OWASP (Open Web Application Security Project) best practices, is used to manage user authentication. This ensures that passwords are stored securely and encrypted and that access control is implemented according to role-based permissions (students, instructors).

**OWASP Standards for Security**

* **SQL Injection Prevention:** Even though Firebase is a NoSQL database, we follow the principle of prepared queries to prevent injection attacks, ensuring that all interactions with the database are safe.
* **Data Integrity & Encryption:** All sensitive data, including passwords and personal user information, is hashed and encrypted using industry-standard techniques, preventing unauthorized access and ensuring the integrity of user data.

**Unified Modeling Language (UML)**

* The project uses UML (Unified Modeling Language) to model system components, including Use Case Diagrams, Class Diagrams, and Sequence Diagrams, based on OMG (Object Management Group) guidelines. This helps to visualize the structure and functionality of the system in a standardized way.

**WiFi Network Handling**

* **WiFi Permissions:**
  + The app only checks WiFi status, name (SSID), and IP address after the user grants permission.
  + WiFi information is accessed only during active attendance sessions and is not stored permanently.
  + Users are clearly prompted to enable WiFi if it is off before they can proceed.
* **Network Matching:**
  + The system uses the WiFi SSID and IP address to verify that the student and instructor are on the same local network.
  + This verification happens locally on the device and is used solely for validating presence during attendance.
  + No WiFi data is transmitted or stored beyond what is needed to perform the match securely.

## Requirement Specification

* R1: User should be able to log in with his/her email ID and password
  + **Specifications:**
    - Users (Instructors and Students) will be provided with a login screen in the mobile app (Flutter).
    - The login screen will have fields for email and password.
    - Email must be a valid PMU email address (@pmu.edu.sa) for instructors.
    - Passwords must be at least 8 characters long.
    - After three consecutive failed login attempts, the account will be locked for a predefined time (e.g., 10 minutes).
    - The "Login" button will be disabled until both fields are filled.
    - The system will prevent entering excessively long values to avoid buffer overflow.
    - Passwords will be securely stored in Firebase using hashing and encryption.
    - "Forgot Password" will trigger Firebase's secure password reset flow.
* R2: The Instructor should be able to create and manage courses
  + **Specifications:**
    - After login, instructors can access the “Courses” section in their dashboard.
    - They can create a course by entering the course name, code, description, and class timings.
    - Each course is stored in Firebase Firestore and linked to the instructor’s UID.
    - Only the course creator has permission to edit or delete the course.
    - The UI displays a list of courses with real-time updates and edit options.
* R3: The Instructor should be able to schedule sessions
  + **Specifications:**
    - Instructors can create sessions by entering the session title, date, time, and attendance rules.
    - They can define time thresholds for Present and Late (e.g., Present ≤ 10 mins, Late ≤ 20 mins).
    - Firebase will store all session data with timestamps.
    - UI validation prevents instructors from scheduling overlapping sessions.
    - A calendar view shows upcoming sessions.
    - When a session starts, the system captures the instructor’s WiFi SSID and IP address to begin the attendance check.
* R4: Student should be able to mark attendance during a live session
  + **Specifications:**
    - When a session is active, students receive a notification or visual indicator.
    - A “Mark Attendance” button becomes active in the app.
    - Before attendance is accepted, the system verifies:
      * The student’s device is connected to the same WiFi network (SSID & IP) as the instructor.
    - The student’s face matches their stored profile photo.
    - On button click, the camera opens for selfie capture.
    - The image is checked against the stored one using Firebase ML Kit.
    - If verified, attendance is marked with a timestamp and categorized (Present, Late, Absent) based on session thresholds.
    - If not matched, students are prompted to retake the image or update their profile picture.
* R5: The Instructor should be able to view real-time attendance
  + **Specifications:**
    - Instructors have access to a live session dashboard showing the attendance status of all students.
    - Attendance is categorized as Present, Late, or Absent in real time.
    - Only students verified via a WiFi network matching and face recognition are counted as Present or Late.
    - All data updates live using Firebase’s snapshot listener for real-time feedback.
* R6: The Student should be able to view the attendance history
  + **Specifications:**
    - Students will have access to a “My Attendance” screen in the app.
    - The screen will show a session-wise breakdown of attendance status (with filters).
    - Data will be fetched from Firebase based on the logged-in student’s UID.
* R7: The Instructor should be able to add students to a course
  + **Specifications:**
    - Instructors can manually enter student emails or upload a CSV of student data.
    - Each student added is stored under the course document in Firebase.
    - Duplicate email entries will be checked and prevented automatically.

## Project Plan

Describe the project plan with a week-by-week set of tasks from concept to implementation. Please provide details on which team members will be responsible for each task. You may show a table with all major project tasks and team members involved.

| **Week** | **Task** | **Team Member(s) Responsible** | **Details** |
| --- | --- | --- | --- |
| **Week 1** | **Project Kickoff** | All Team Members | Initial planning and task assignment, setting up the development environment for the WiFi-based system. |
| **Requirement Gathering & Analysis** | Business Analyst, Product Owner | Collect and document detailed requirements for the app, including WiFi integration for attendance, user roles, and the face verification system. |
| **Week 2** | **System Architecture Design** | System Architect, Backend Developer | Design the overall architecture, including WiFi setup for proximity verification, Firebase integration, and mobile app structure. |
| **UI/UX Design** | UI/UX Designer | Design the user interface and experience based on the requirements, ensuring WiFi and attendance-related screens are intuitive and clear. |
| **Week 3** | **Database Design & Setup** | Backend Developer | Set up Firebase Firestore for data storage, ensuring sessions, attendance, and user data are linked with the WiFi-based verification. |
| **Frontend Design and Layout** | Mobile Developer (Flutter) | Build initial UI components (login page, dashboard, session scheduling, and attendance screens) that include WiFi-based attendance features. |
| **Week 4** | **Implement WiFi Detection for the Instructor** | Mobile Developer (Flutter) | Implement WiFi-based functionality to check the instructor’s device proximity, using WiFi ID for attendance marking. |
| **Set up User Authentication (Firebase)** | Backend Developer | Set up Firebase Authentication for secure user login and role-based access, ensuring different access levels for instructors and students. |
| **Week 5** | **Session Scheduling Feature Development** | Mobile Developer (Flutter), Backend Developer | Develop session scheduling functionality, including date/time, attendance rules, and WiFi session start logic. |
| **Face Verification Feature** | Mobile Developer (Flutter), AI Specialist | Integrate Firebase ML Kit for real-time face verification and face capture for attendance. |
| **Week 6** | **WiFi Functionality for Students** | Mobile Developer (Flutter) | Implement WiFi-based attendance for students, marking attendance when the student’s device is within proximity to the instructor’s WiFi. |
| **Real-time Data Sync & Dashboard Development** | Backend Developer, Mobile Developer (Flutter) | Implement Firebase real-time data sync for live attendance monitoring, and create a live dashboard for instructors to track student presence. |
| **Week 7** | **Testing: WiFi & Face Verification** | QA Tester, Mobile Developer (Flutter) | Test WiFi functionality and face verification for both instructor and student, ensuring proper integration. |
| **Bug Fixing & Performance Optimization** | All Developers | Address any bugs and optimize performance for WiFi detection, attendance, and face verification. |
| **Week 8** | **Export Attendance Records Feature** | Mobile Developer (Flutter), Backend Developer | Implement the functionality for instructors to export attendance records from the app. |
| **User Acceptance Testing (UAT)** | QA Tester, Product Owner | Conduct UAT for all app features to ensure they meet the initial requirements. |
| **Week 9** | **Final Testing & Debugging** | QA Tester, All Developers | Perform final tests, fix remaining bugs, and ensure all app features work properly. |
| **App Documentation & User Guides** | Business Analyst, Mobile Developer (Flutter) | Prepare app documentation, including how to use the app, WiFi-based attendance, and troubleshooting steps. |
| **Week 10** | **App Deployment to App Stores** | Mobile Developer (Flutter), Product Owner | Submit the app to the Google Play Store and Apple App Store for review and release. |
| **Week 11** | **Post-Launch Monitoring & Maintenance** | All Team Members | Monitor app performance, collect feedback from users, and fix any post-launch issues. |

# Project Design

## Architecture Diagram

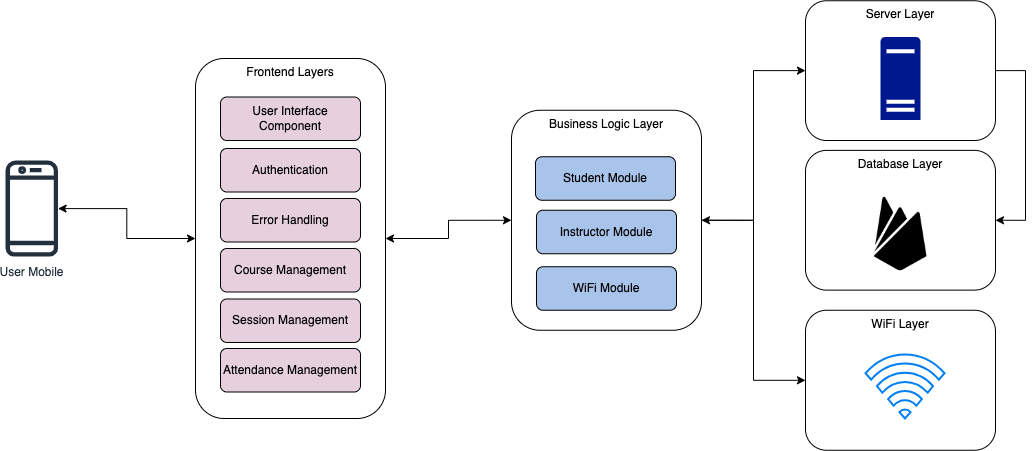


Figure 1: Architecture Diagram

This diagram outlines a three-tier architecture for an attendance tracking system. The Frontend Layer handles user interactions through interfaces for authentication, course/session management, and attendance tracking, along with error handling. The Business Logic Layer contains core functionality modules – Student Module for attendance marking, Instructor Module for course/session control, and WiFi Module for WiFi-based communication. The Server Layer manages data persistence through the Database Layer and device connectivity via the WiFi Layer, ensuring secure data storage and real-time WiFi signal processing between student devices and instructor systems.

## UML Flowchart/Activity Diagram

### Instructor Diagram

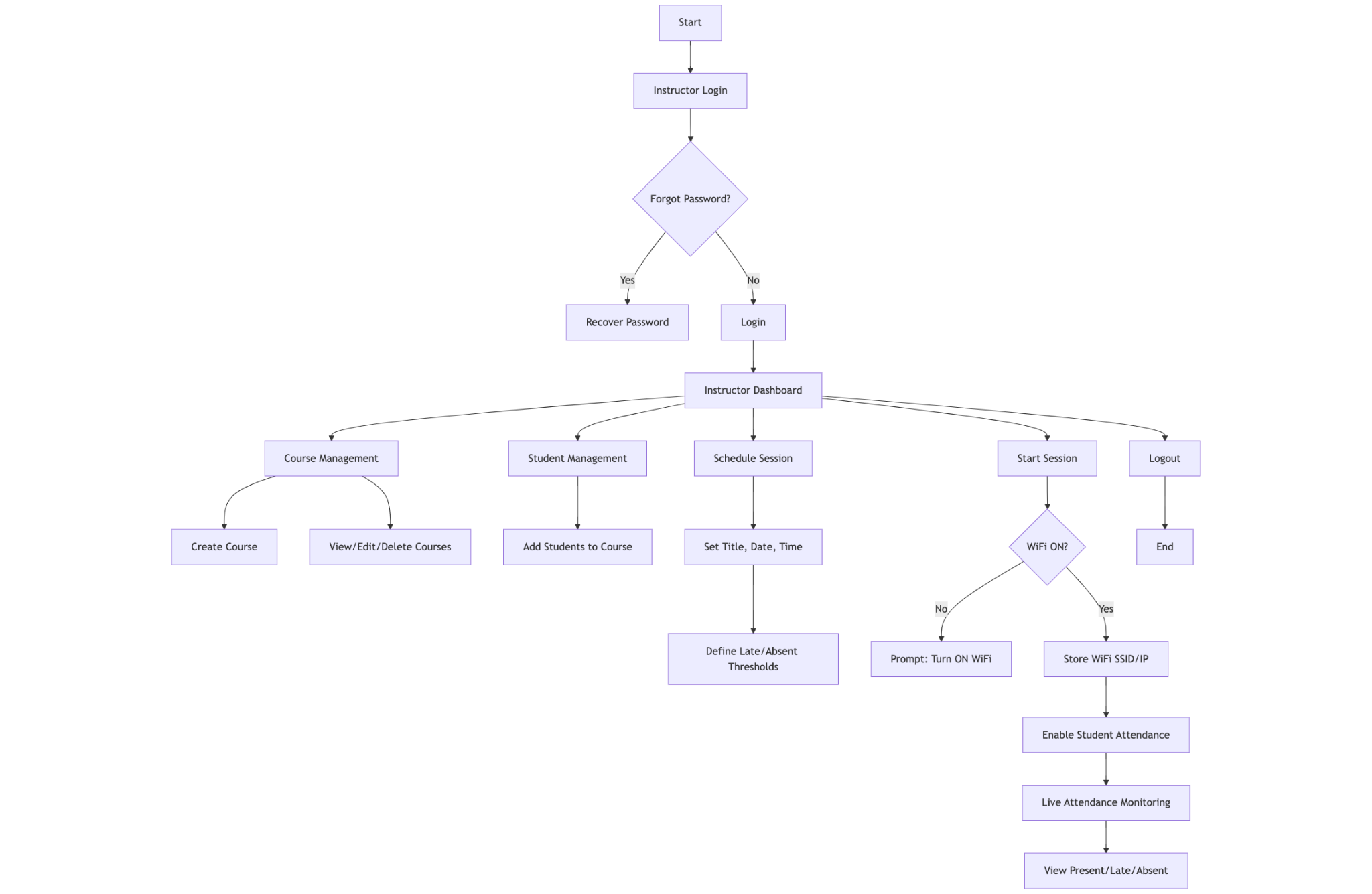


Figure 2: UML Instructor Activity Diagram

The instructor begins by logging into their account or recovering their password if needed. Once logged in, they’re taken to their dashboard, where they can create and manage courses, add students, and schedule class sessions. When it’s time for class, the instructor starts the session, but first, the system checks if their WiFi is turned on. If not, it prompts them to enable it. Once WiFi is connected, the system saves the network details and allows students to mark attendance. During the session, the instructor can see in real-time which students are present, late, or absent based on the time rules they set earlier. After class, they can review attendance records anytime.

### Student Diagram

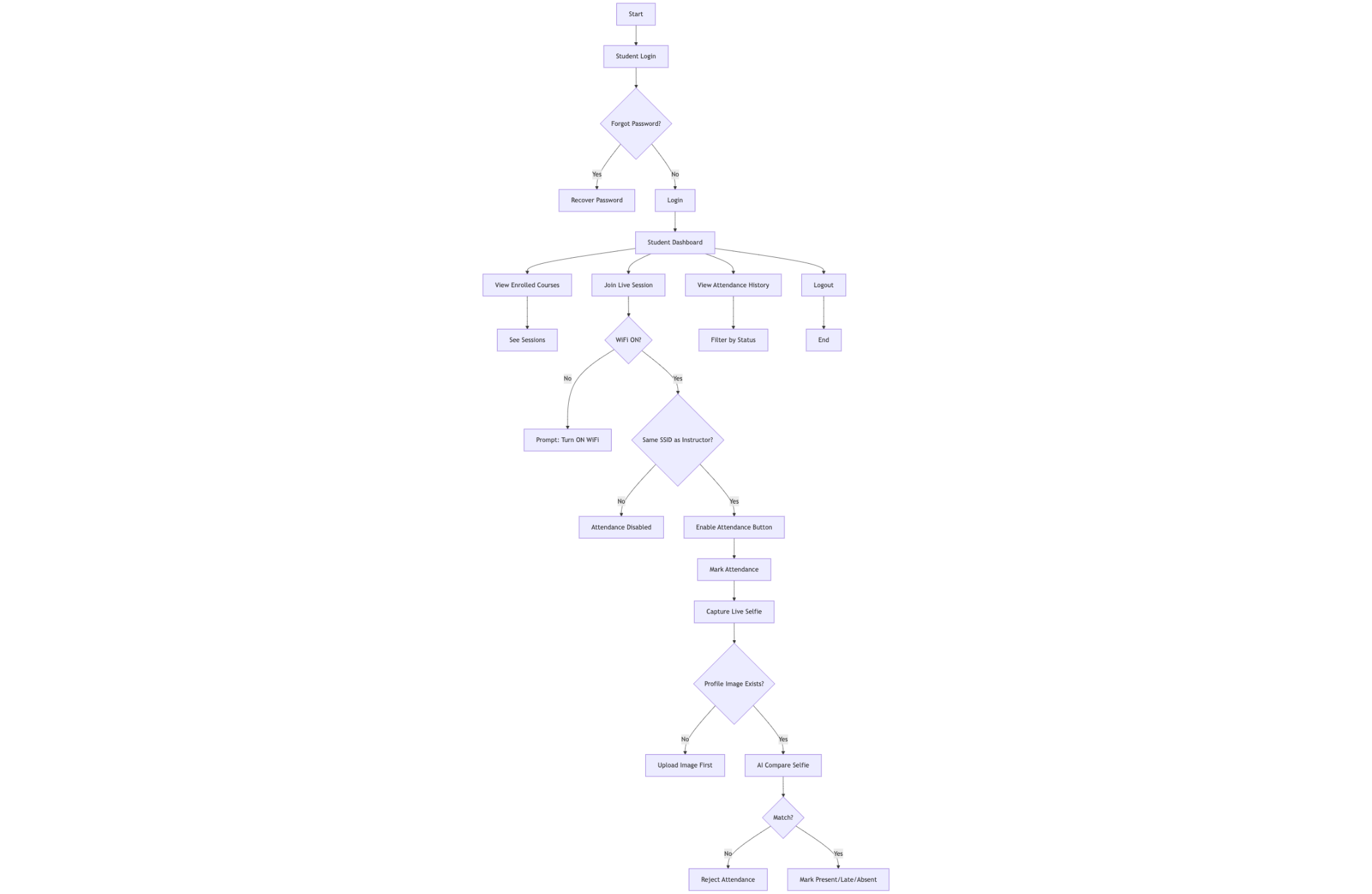


Figure 3: UML Student Activity Diagram

Students start by logging in or resetting their password if forgotten. After logging in, they see their dashboard with a list of enrolled courses and upcoming sessions. When a session is live, they join it—but only if their device is connected to the same WiFi network as the instructor. If not, the system asks them to switch networks. Once connected, they tap “Mark Attendance,” take a live selfie, and the system compares it to their profile photo using AI. If it matches, their attendance is recorded as present, late, or absent based on the instructor’s timing rules. Later, they can check their attendance history to see their status for past classes.

## User Interface Design

### Instructor Mockup

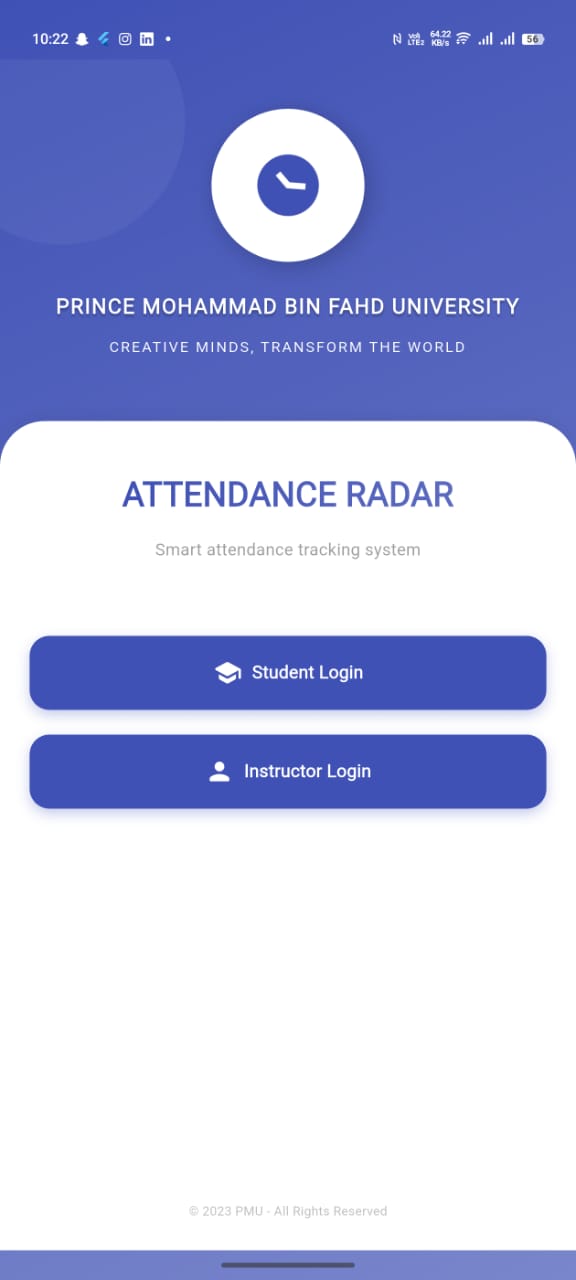


Figure 4: Select User Type Screen

This is the User Type Selection Screen for the attendance system. It displays the application name "Smart attendance tracking system." along with the university branding. The screen presents two login options: one for Students (currently unchecked) and one for Instructors (pre-selected with a checkmark). This allows users to specify their role before proceeding to the actual login page. The footer includes copyright information and the system's tagline: "Smart attendance tracking system."

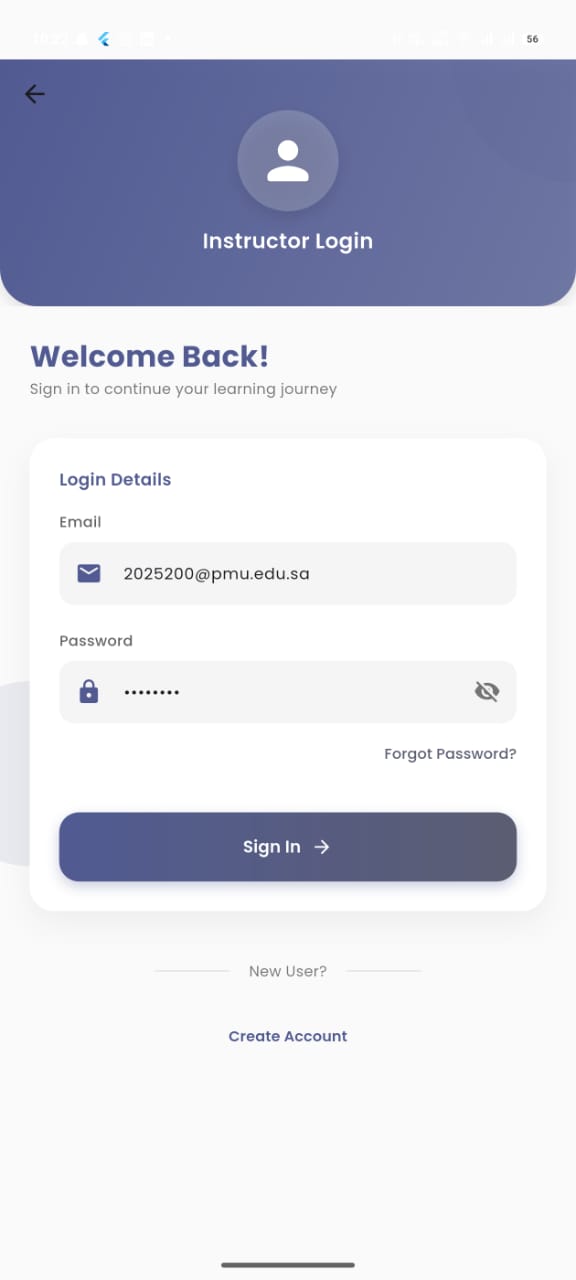


Figure 5: Instructor Login Screen

This is the login page for instructors. It shows a welcome message ("Welcome Back! Sign in to continue your learning journey") with fields for entering email (pre-filled with a PMU email example) and password. A "Sign In" button appears below, along with an option for new users to "Create Account."

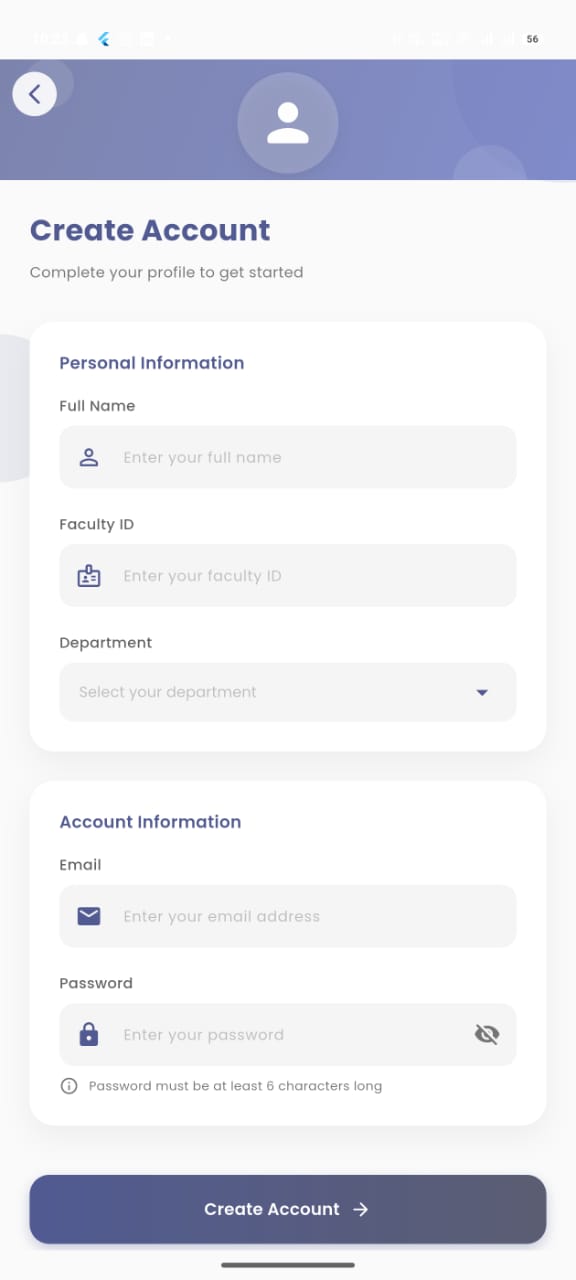


Figure 6: Instructor Register Screen

New instructors register here by filling in two sections:

* **Personal Information:** Full name, faculty ID, and department dropdown.
* **Account Information:** Email and password (with a note that passwords must be 6+ characters).

The screen has a straightforward form layout with clear field labels.

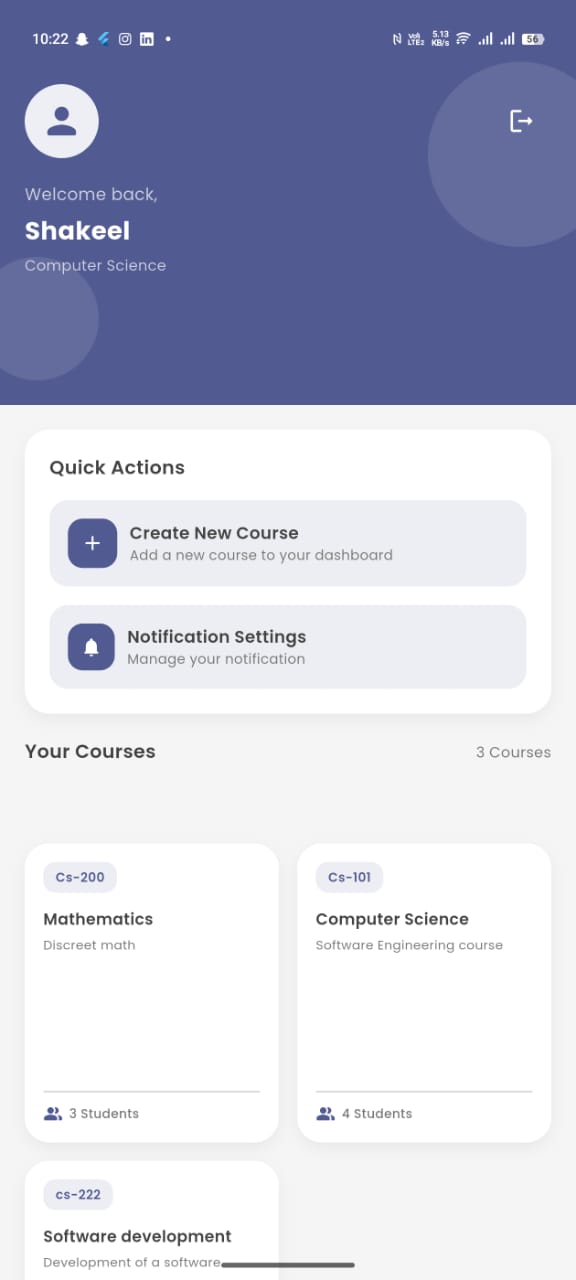


Figure 7: Instructor Dashboard Screen

After logging in, instructors see:

* A welcome header ("Welcome back, Shakeel") with their department (Computer Science).
* **Quick Actions:** Buttons to "Create New Course" or manage notifications.
* **Your Courses:** A list of 3 courses (e.g., CS-200, CS-101) with student counts and subjects.
* A highlighted course (CS-222: "Software development") at the bottom.

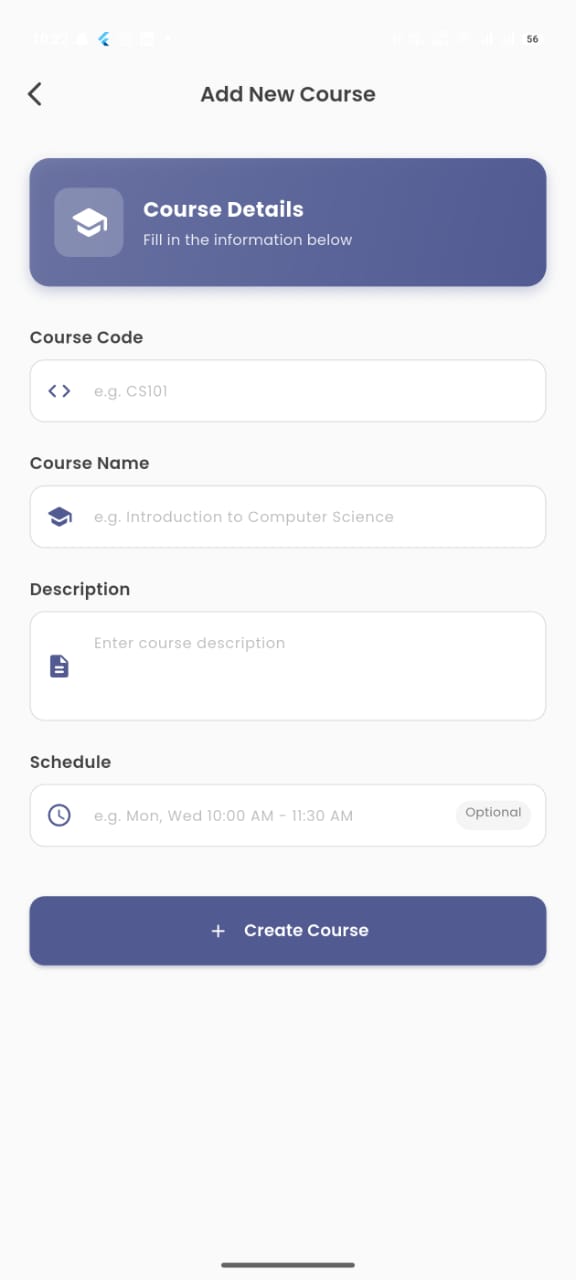


Figure 8: Create New Course Screen

Instructors create courses here by entering:

* Course Code (e.g., CS101)
* Course Name (e.g., "Introduction to Computer Science")
* Description (free-text field)
* Schedule (optional, with an example format).
* A "+ Create Course" button saves the details.

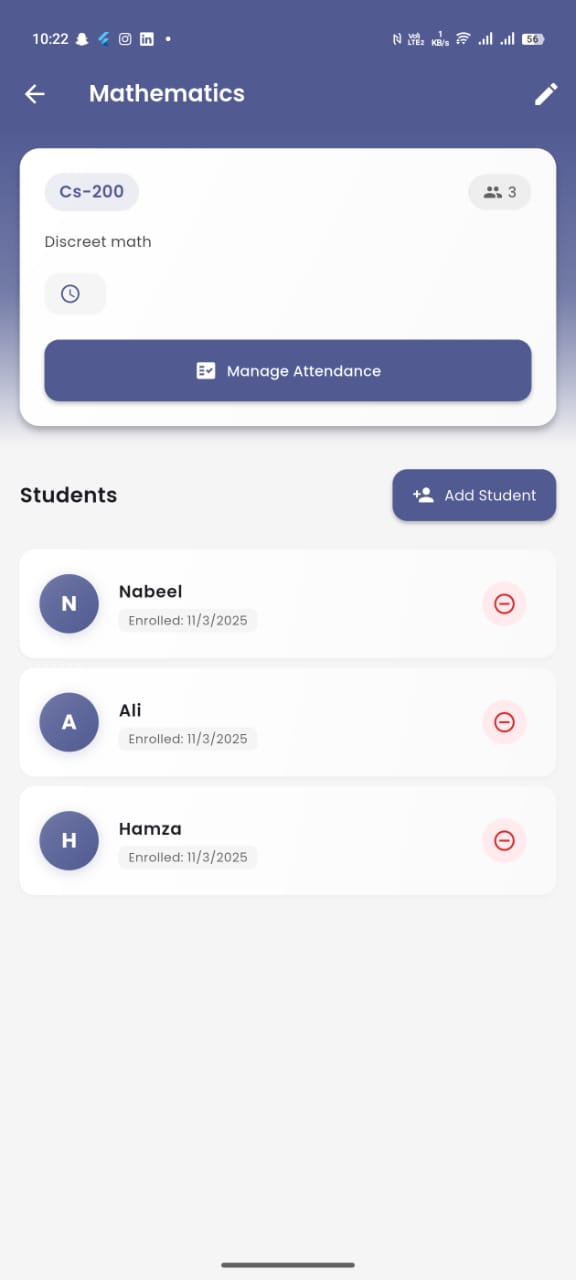


Figure 9: View Course Details Screen

This screen shows details for a specific course ("Discreet math" under CS-200). Key features:

* A header with "32 Manage Attendance" (likely total sessions/students).
* **Students List:** Names (Nabeel, Ali, Hamza) with enrollment dates and an "Add Student" button.

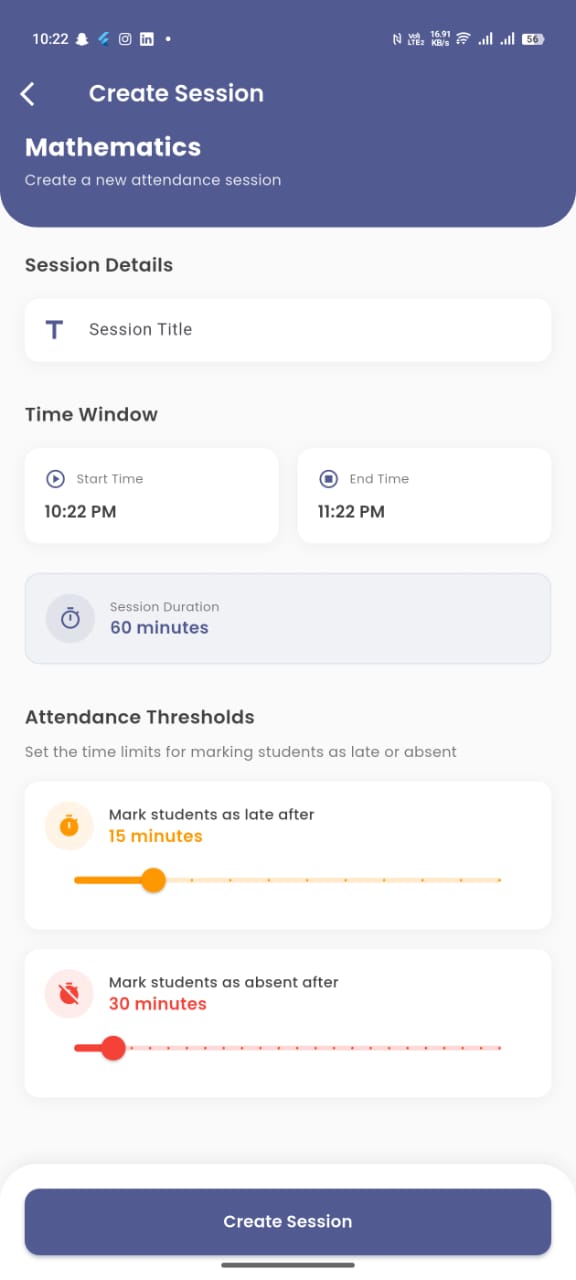


Figure 10: Create Session Screen

Instructors set up attendance sessions here with:

* Session Title and End Time fields.
* Time Window: Start time (pre-filled as 10:22 PM).
* Attendance Thresholds: Customizable late (15 min) and absent (30 min) limits.
* "Create Session" button finalizes the setup.

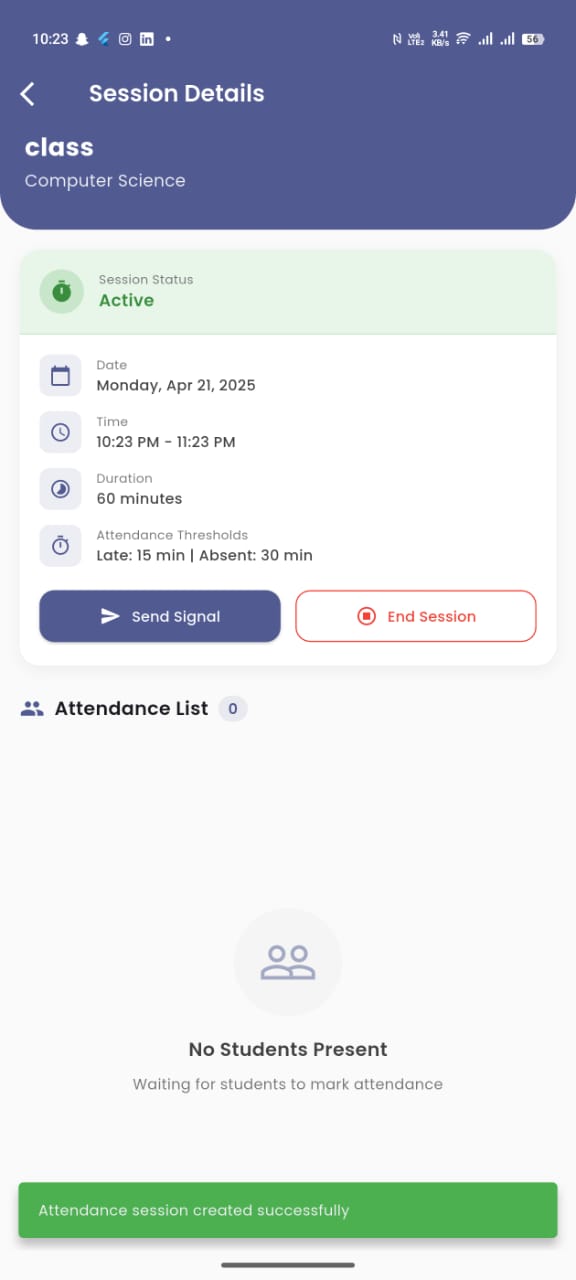


Figure 11: View Session Details Screen

Displays real-time data for a live session (e.g., "Computer Science" class):

* **Session Status:** "Active" with date/time/duration.
* **Thresholds:** Late (15 min) and absent (30 min) rules.
* **Attendance List:** Initially shows "No Students Present" until students mark attendance.
* Buttons to "Send Signal" (likely for BLE) or "End Session" appear at the bottom.

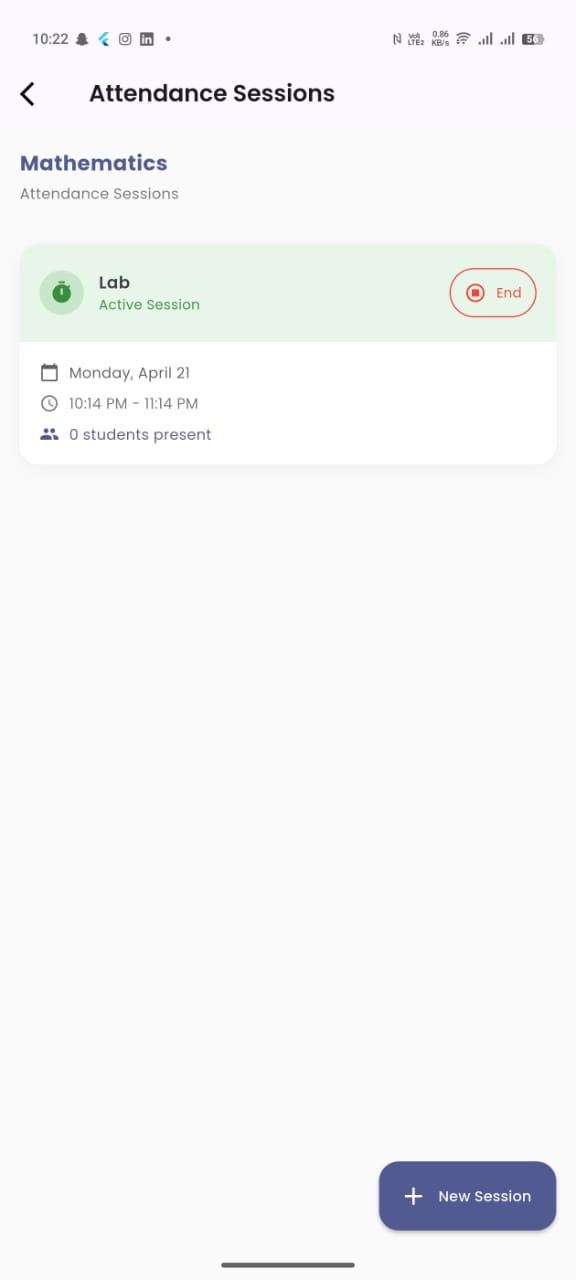


Figure 12: View Live Session Screen

Lists all sessions for a course (e.g., "Mathematics"). Includes:

* Toggleable sessions (e.g., "Lab" marked as active, "Monday, April 21" with time slots).
* A "+ New Session" button to create additional sessions.

### Student Mockup

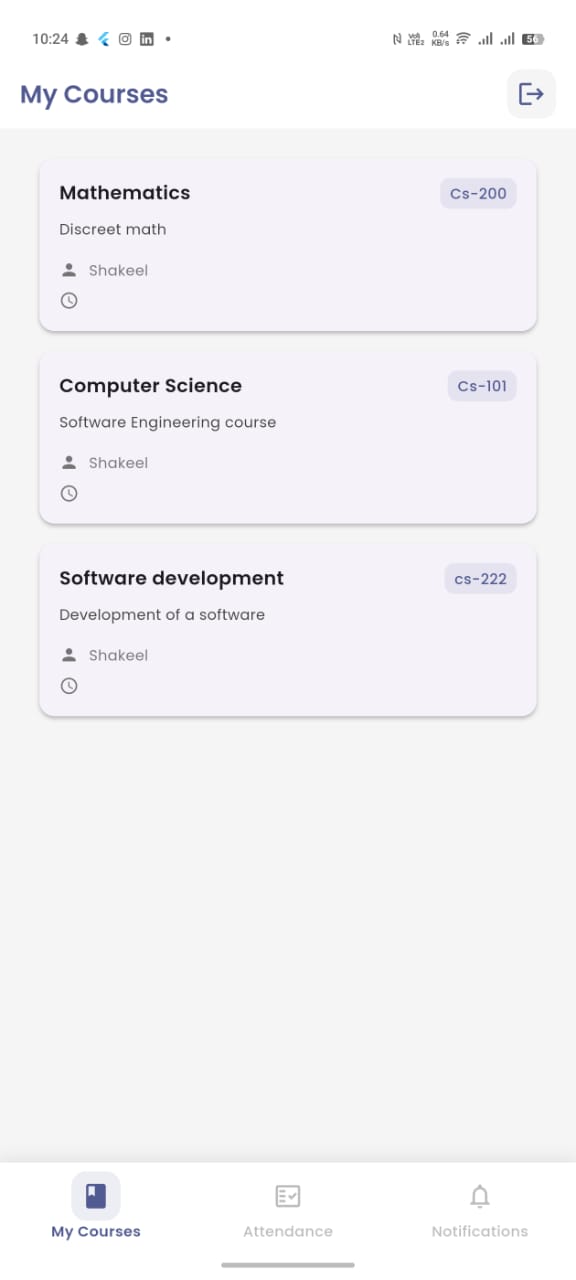


Figure 13: View My Courses Screen

This screen shows a student's enrolled courses in a simple list format:

* Courses are grouped by subject (Mathematics, Computer Science, Software Development)
* Each course displays:
  + Course name (e.g., "Discreet math")
  + Instructor name ("Shakeel")
* Bottom navigation tabs allow switching between:
  + My Courses (current view)
  + Attendance
  + Notifications

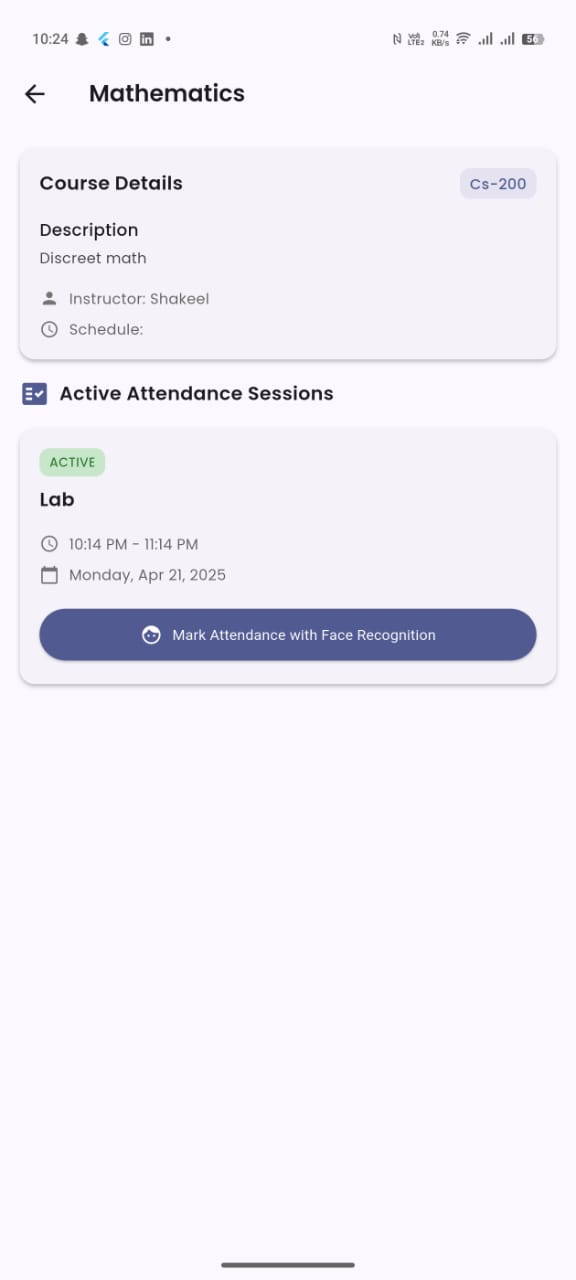


Figure 14: View Course Detail Screen

This screen provides detailed information about a specific course (Mathematics):

* **Top section:**
  + Course description ("Discreet math")
  + Instructor name and schedule information (empty in this example)
* **Middle section:** Shows active attendance session:
  + Session name ("Lab")
  + Time slot (10:14 PM - 11:14 PM)
  + Date (Monday, Apr 21, 2025)
* **Bottom action:** "Mark Attendance with Face Recognition" button

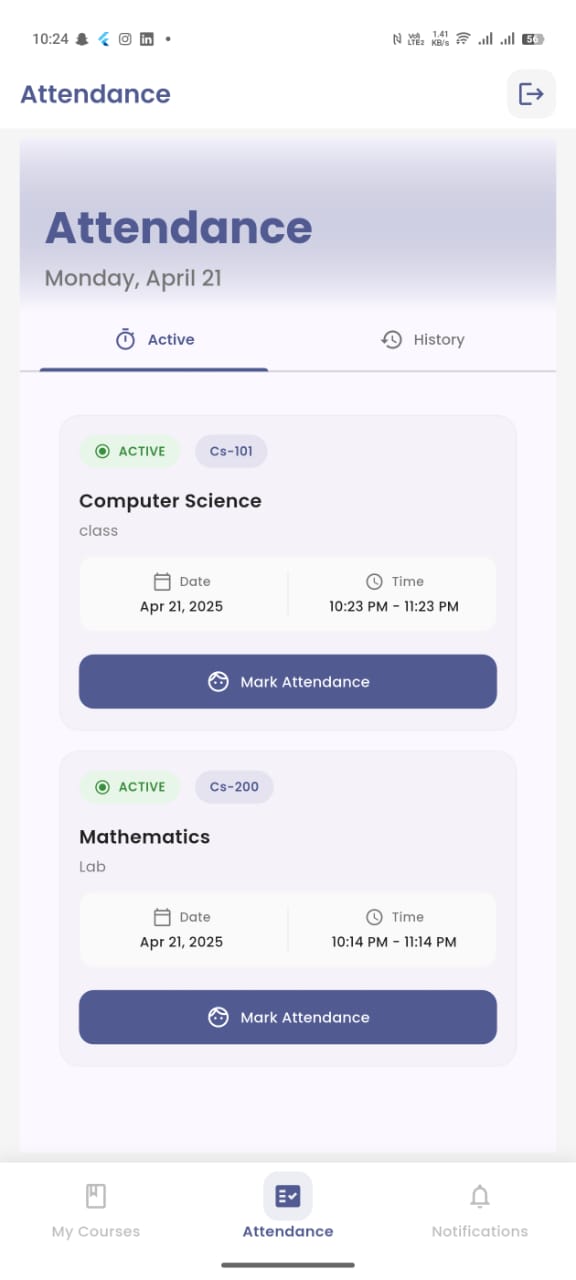


Figure 15: Attendance Overview Screen

This screen organizes attendance tracking:

* Date header (Monday, April 21) with two view options:
  + [Active] - Current/live sessions
  + [History] - Past sessions
* **Lists all active sessions:**
  + Computer Science class (with date/time)
  + Mathematics Lab (marked as active, with course code CS-200)
* Bottom navigation tabs (same as other screens)

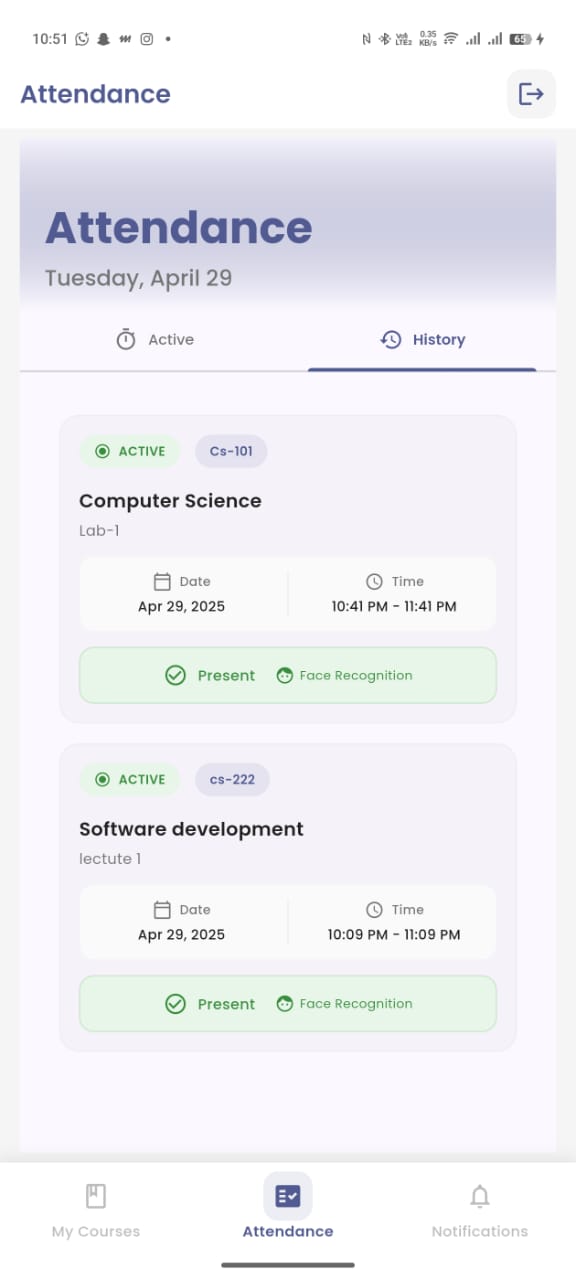


Figure 16: Attendance Overview History Screen

* **Main Instruction**
  + A clear prompt at the top says: "Take a selfie to verify your identity"
* **Action Buttons**
  + [Take Photo]: Starts the face recognition process (opens camera for live selfie).
  + [Cancel]: Exit the attendance process if the student changes their mind.
* **Session Details (displayed below the buttons)**
  + Course: Computer Science Lab-1
  + Date/Time: Apr 29, 2025, 10:41 PM - 11:41 PM
* **Verification & Confirmation**
  + After taking the photo, the system compares it to the student’s profile using AI.
  + If matched, a "[Present]" button appears to finalize attendance.
  + Status is recorded as Present/Late/Absent based on the instructor’s time rules.

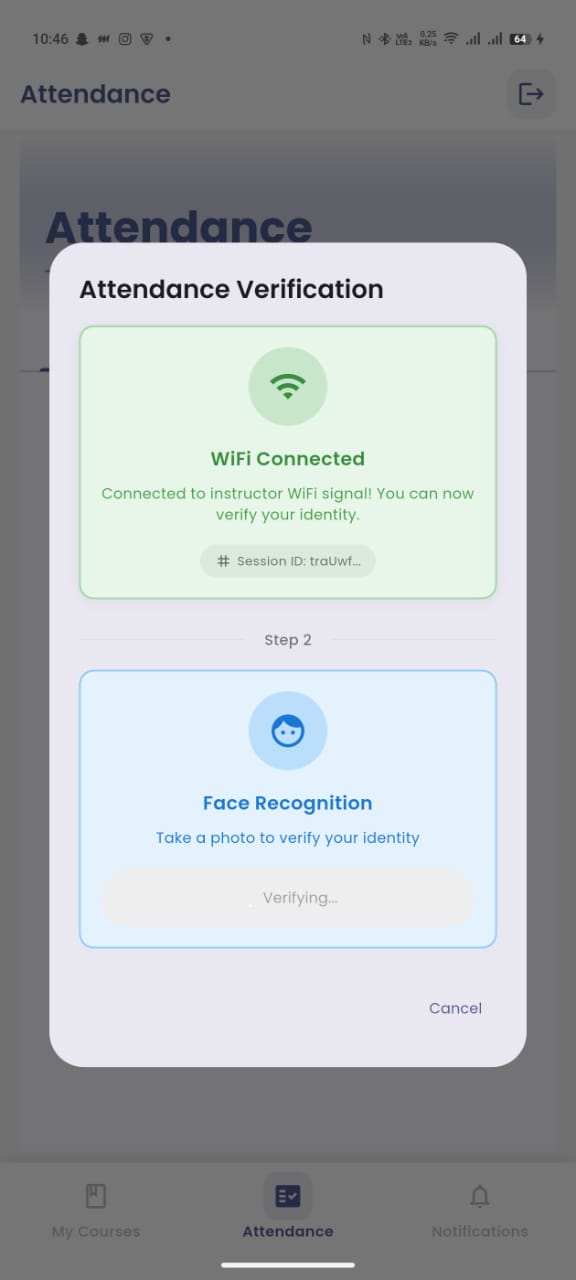


Figure 17: Attendance Mark

This screen organizes attendance tracking:

* **WiFi Verification (Step 1)**
  + The system confirms: "Connected to instructor WiFi signal!"
  + Shows a Session ID (ex: traUwI...) to validate the correct class session.
  + This ensures the student is physically present in the classroom.
* **Face Recognition (Step 2)**
  + Prompt: "Take a photo to verify your identity"
  + Live Camera activates for a selfie.
  + Status: "Verifying..." (AI compares the selfie to the student’s profile photo).
  + [Cancel]: Allows exiting if needed.

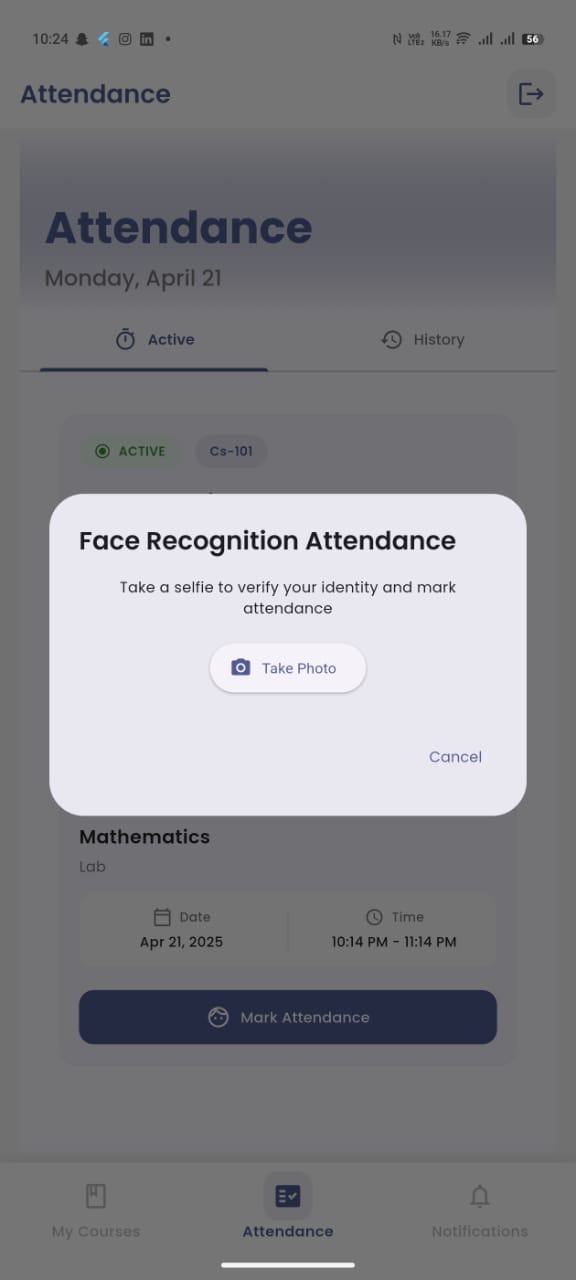


Figure 18: Face Recognition Attendance Screen

This is where students mark attendance:

* **Main instruction:** "Take a selfie to verify your identity"
* **Two action buttons:**
  + [Take Photo] - Starts face recognition process
  + [Cancel] - Exits attendance marking
* **Session details shown below:**
  + Course: Mathematics Lab
  + Date/Time: Apr 21, 2025, 10:14 PM - 11:14 PM
* "Mark Attendance" button appears once photo is verified

## Database Design (if applicable)

This can include ER Diagram or other database design aspects. Please discuss how the database design addresses the security and other non-functional requirements of the project. This can include use of encrypted password storage, managing roles in the database tables etc.

## Sequence Diagram Design

### Instructor Features

#### Starting a Class Session

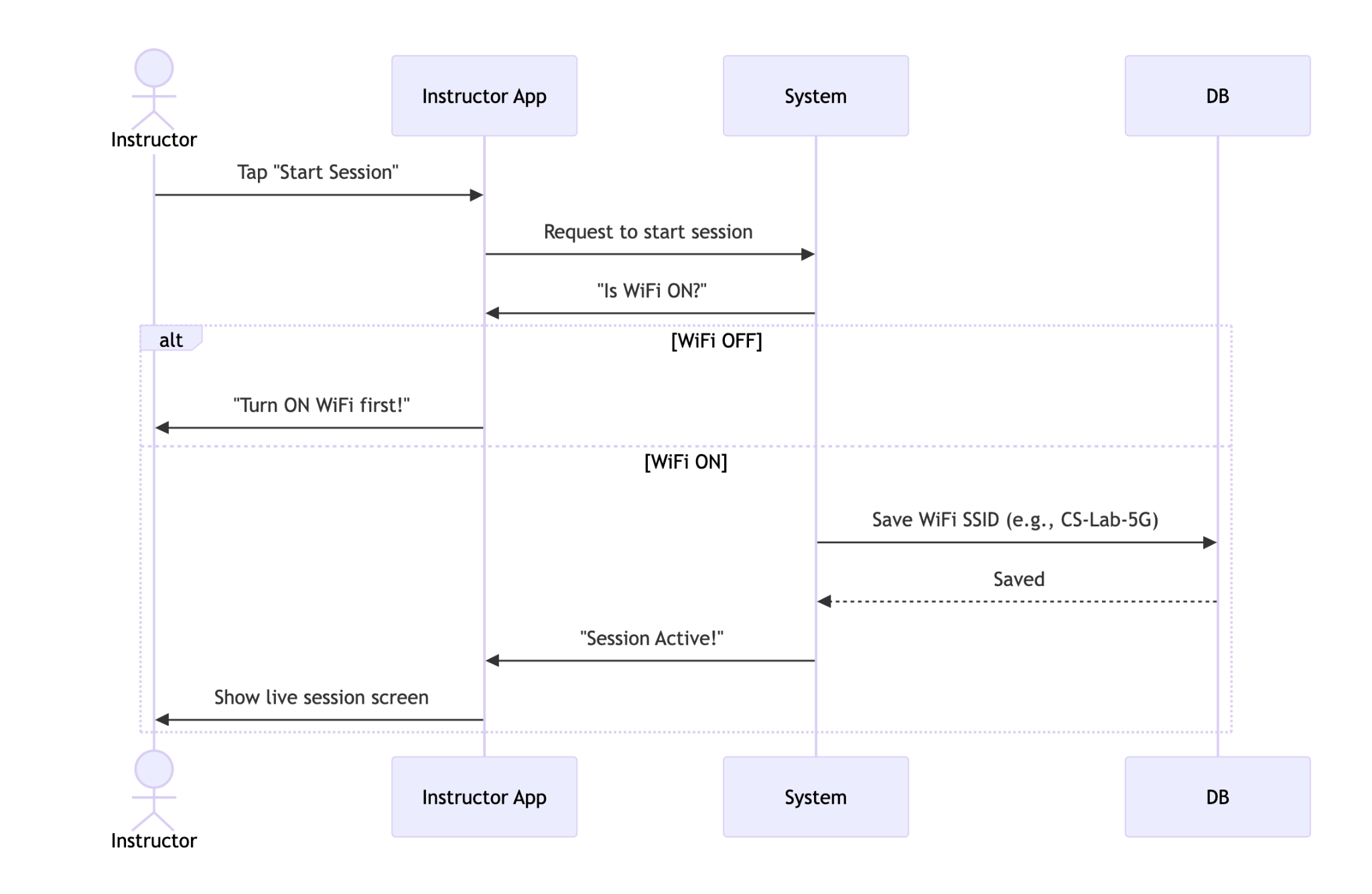


Figure 19: Starting a Class Session

The instructor taps "Start Session," and the app checks if their WiFi is on. If not, it asks them to enable it. If WiFi is on, the system saves the classroom’s WiFi name (like "CS-Lab-5G") and opens the live session screen for attendance tracking.

#### Adding Students to a Course

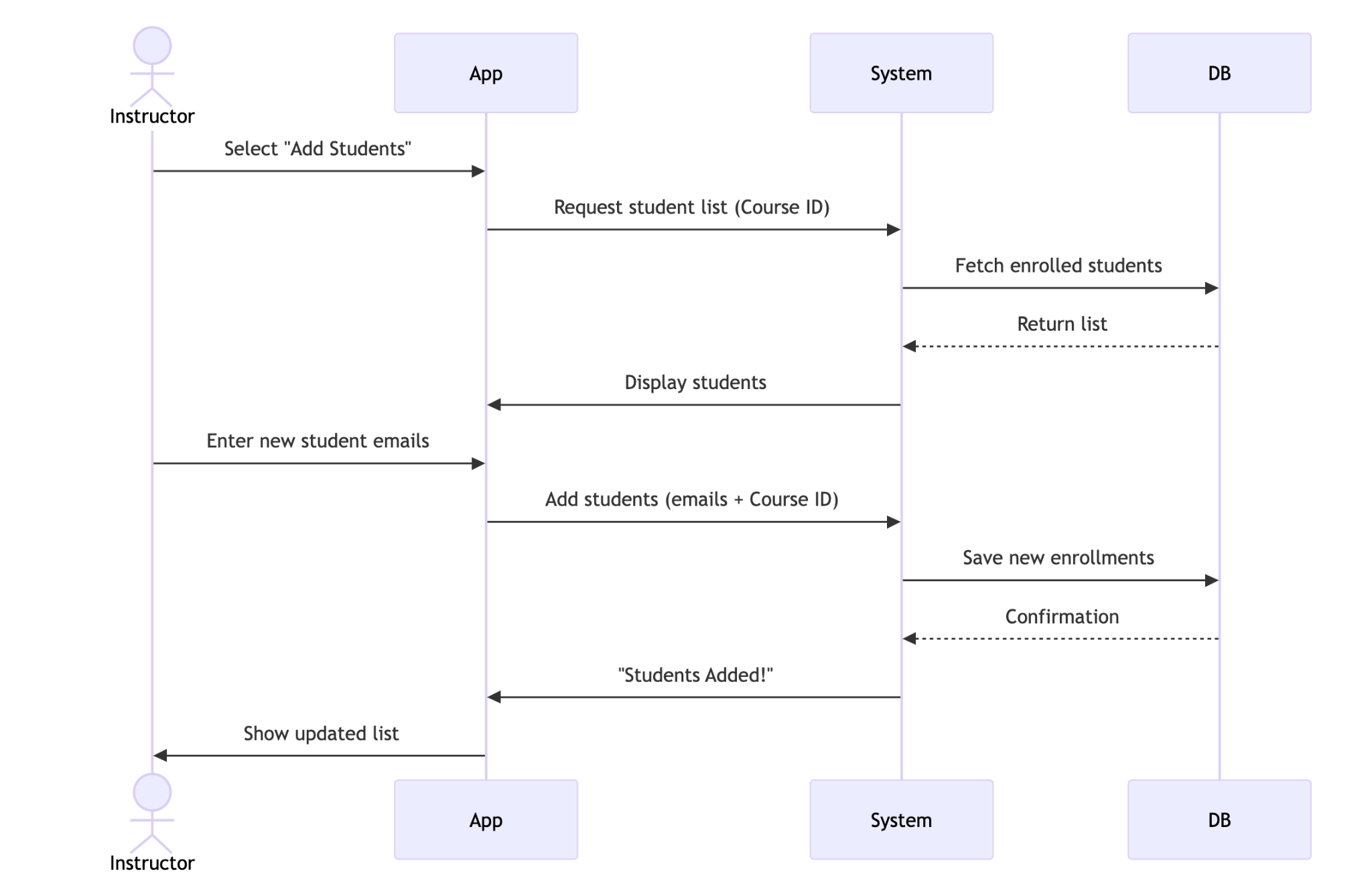


Figure 20: Adding Students to a Course

The instructor adds students to a course by entering their emails. The system checks the database, adds the new students, and confirms the update. The instructor sees the refreshed list instantly.

#### Viewing Attendance Reports

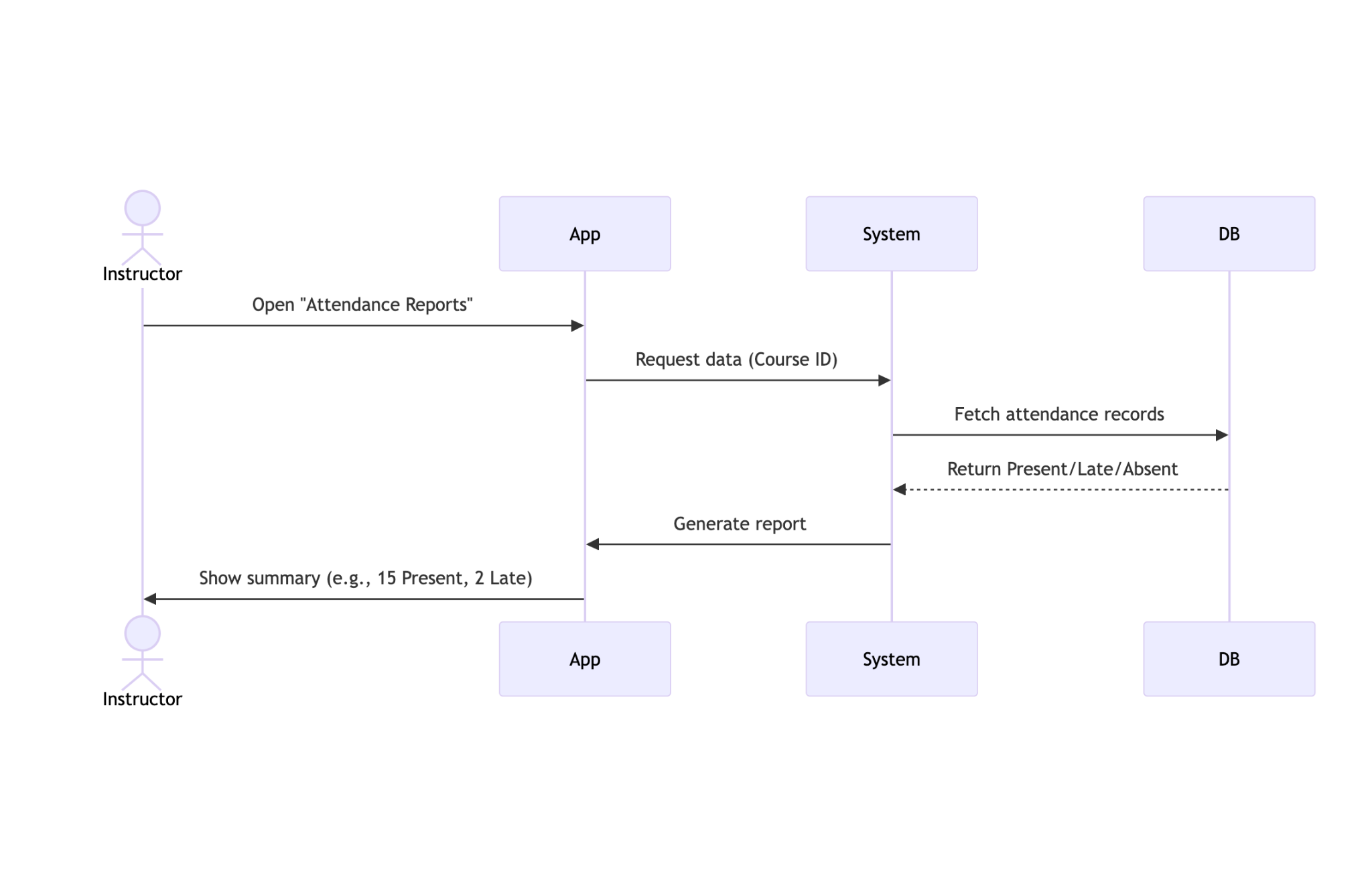


Figure 21: Viewing Attendance Reports

The instructor opens attendance reports for a course. The system fetches records from the database and displays a summary (e.g., how many were present, late, or absent).

### Student Features

#### Marking Attendance via WiFi + Selfie

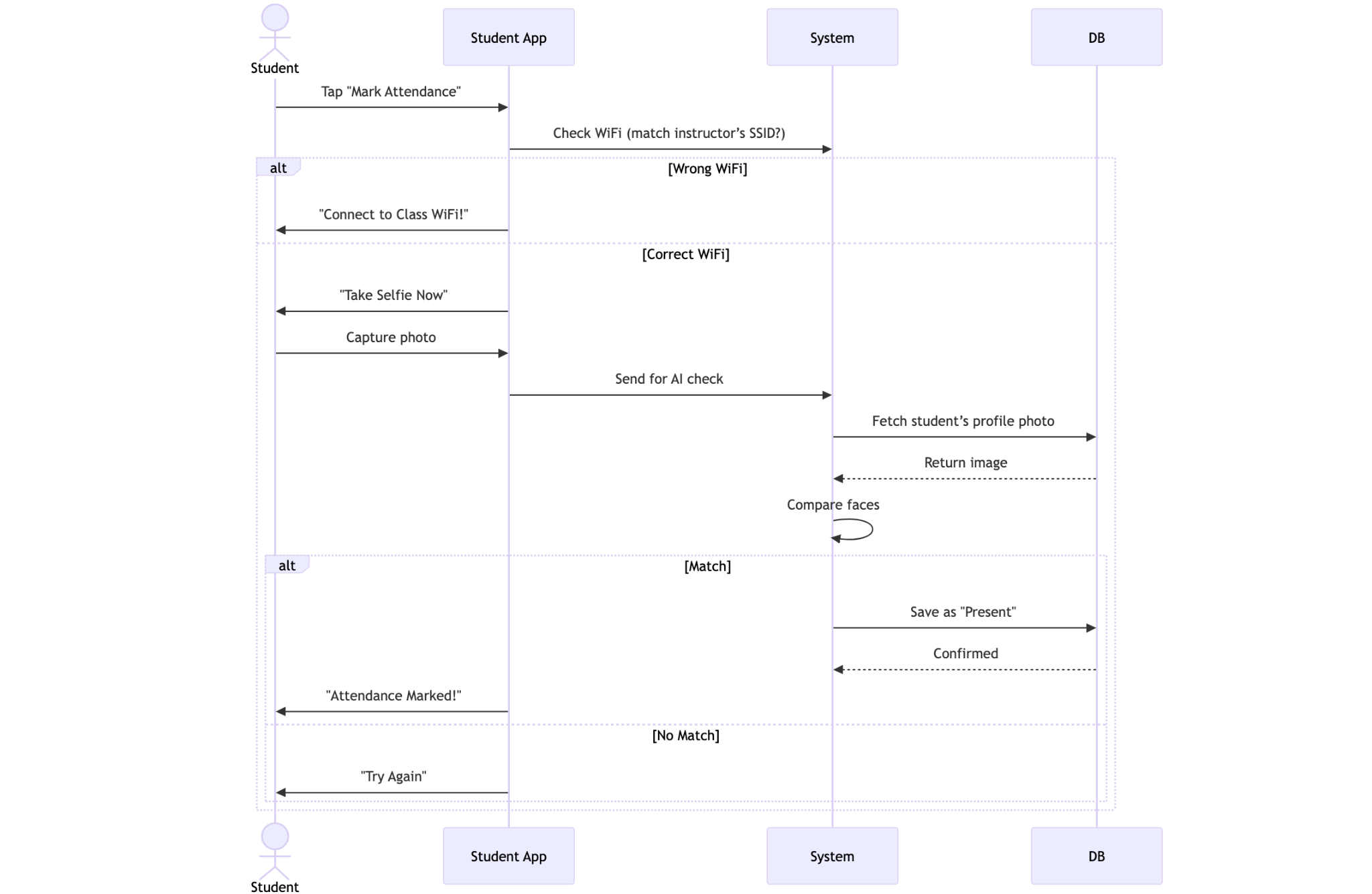


Figure 22: Marking Attendance via WiFi + Selfie

The student taps "Mark Attendance." The app checks if they’re on the classroom WiFi. If yes, it asks for a selfie. The AI compares it to their profile photo. If it matches, attendance is saved as "Present"; if not, they must retry.

#### View Enrolled Courses

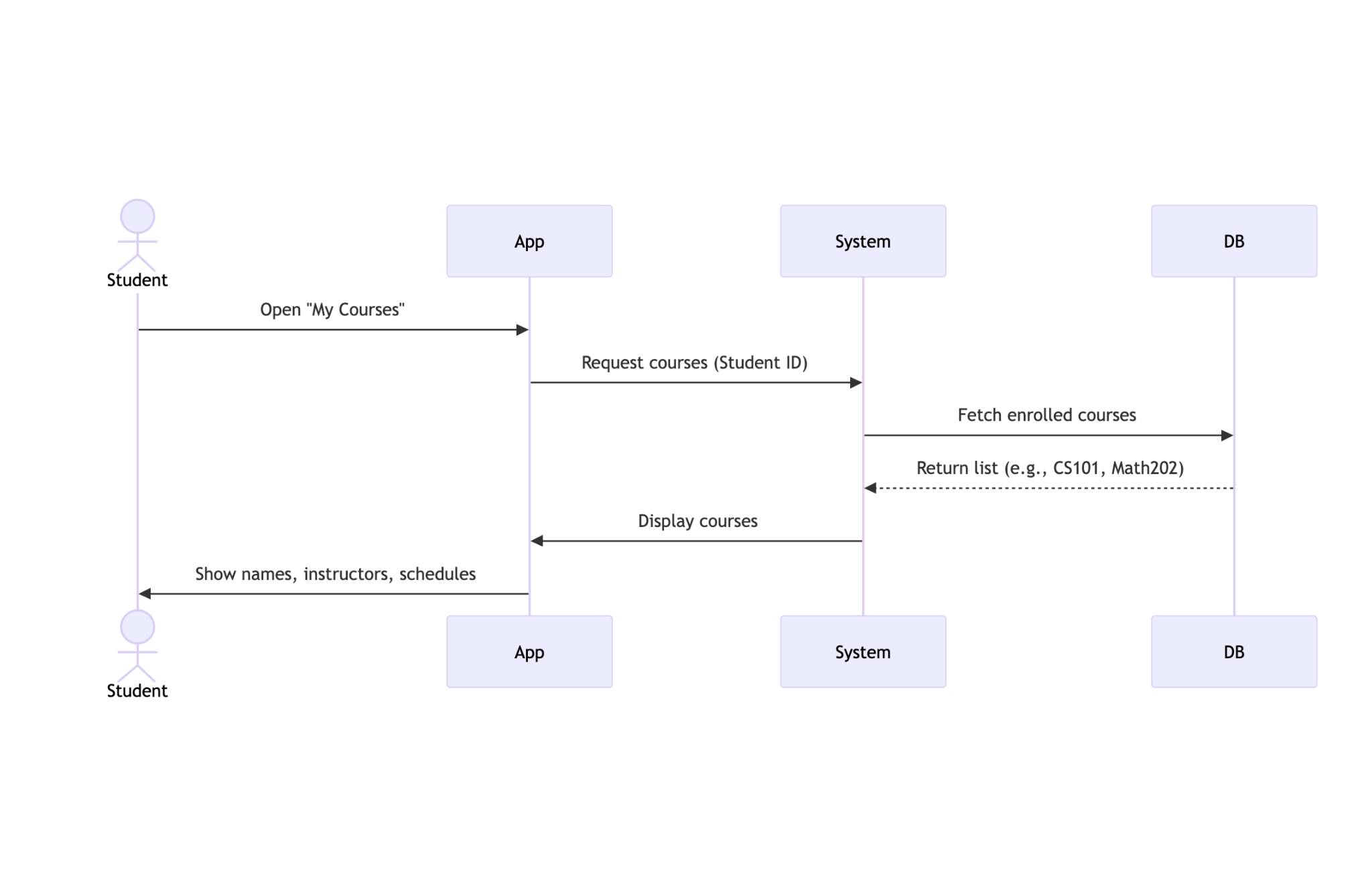


Figure 23: View Enrolled Courses

The student navigates to "My Courses," and the system fetches their enrolled courses from the database. The frontend displays each course’s name, instructor, and schedule in a clean list.

#### Check Attendance History

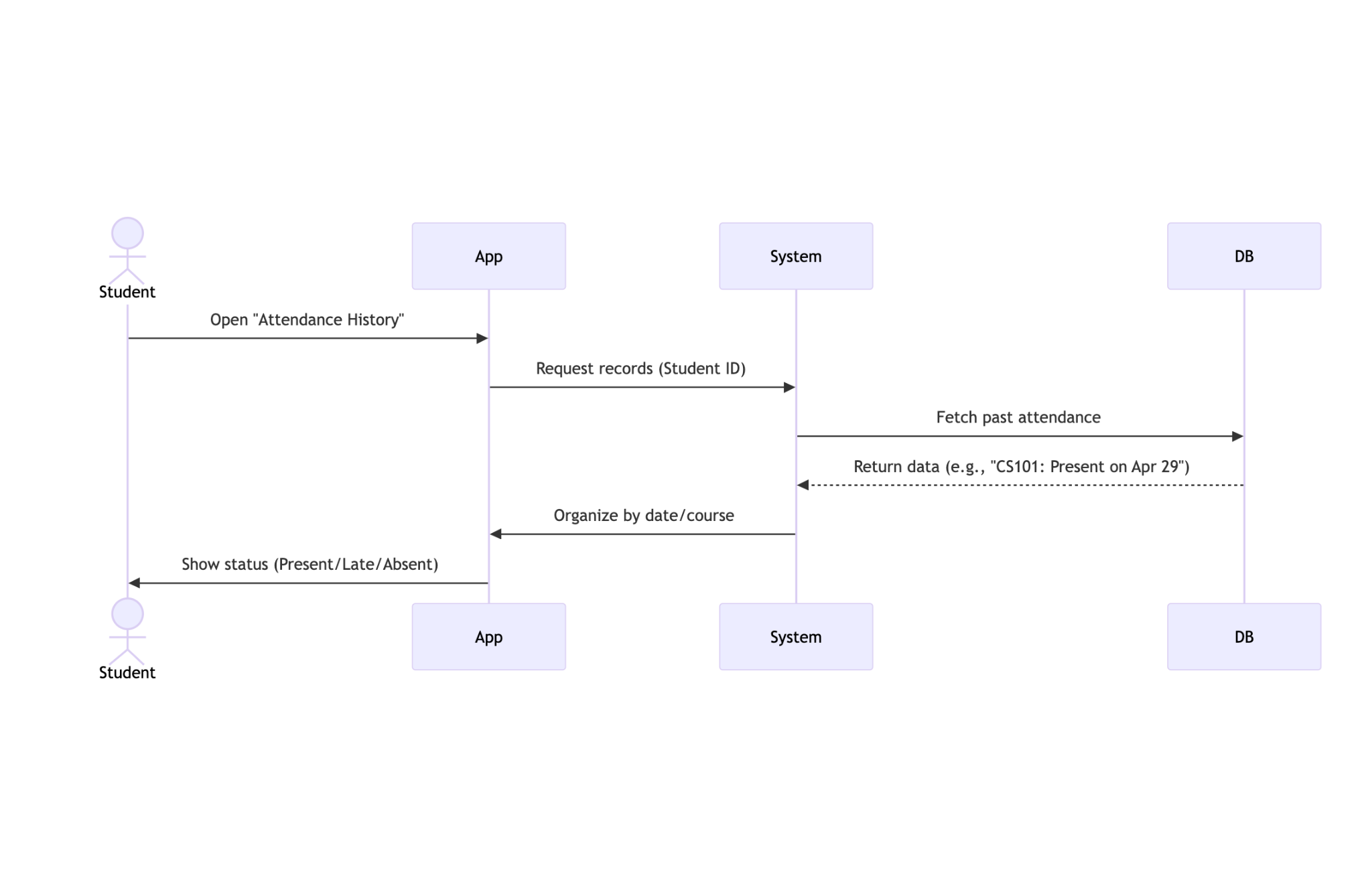


Figure 24: Check Attendance History

The student accesses their attendance history, and the system retrieves past records from the database. These are grouped by course and date, showing whether they were Present, Late, or Absent for each session.

## Algorithm Design (if applicable)

The core algorithm of the smart attendance system revolves around facial recognition for user verification and attendance marking. Below is a breakdown of the algorithm, how it operates, and how it addresses key security and non-functional requirements of the system.

### Face Comparison Algorithm

This algorithm uses the face\_recognition library in Python to perform accurate facial recognition between two images: the registered face image and the live uploaded image for verification.

**Face Detection:**

Faces are first detected in both images using:

* face\_locations1 = face\_recognition.face\_locations(image1)
* face\_locations2 = face\_recognition.face\_locations(image2)

If faces are not detected in either image, an exception is raised.

**Face Encoding**:

Each detected face is converted into a numeric encoding (vector representation) using:

* face1\_encodings = face\_recognition.face\_encodings(image1, face\_locations1)
* face2\_encodings = face\_recognition.face\_encodings(image2, face\_locations2)

**Face Distance Calculation:**

The Euclidean distance between the two encodings is computed:

* face\_distance = face\_recognition.face\_distance([face1\_encoding], face2\_encoding)[0]

**Tolerance Threshold:**

A strict threshold (e.g., 0.4) is applied to ensure higher accuracy and prevent false positives:

* tolerance = 0.4
* is\_match = face\_distance <= tolerance

**Confidence Score**:

A confidence score (0–100%) is calculated inversely from the face distance:

* confidence = max(0, min(100, (1 - face\_distance) \* 100))

**Match Verification**:

Additionally, the system uses compare\_faces for double-checking:

* verification\_match = face\_recognition.compare\_faces([face1\_encoding], face2\_encoding, tolerance=tolerance)[0]

### Security Considerations

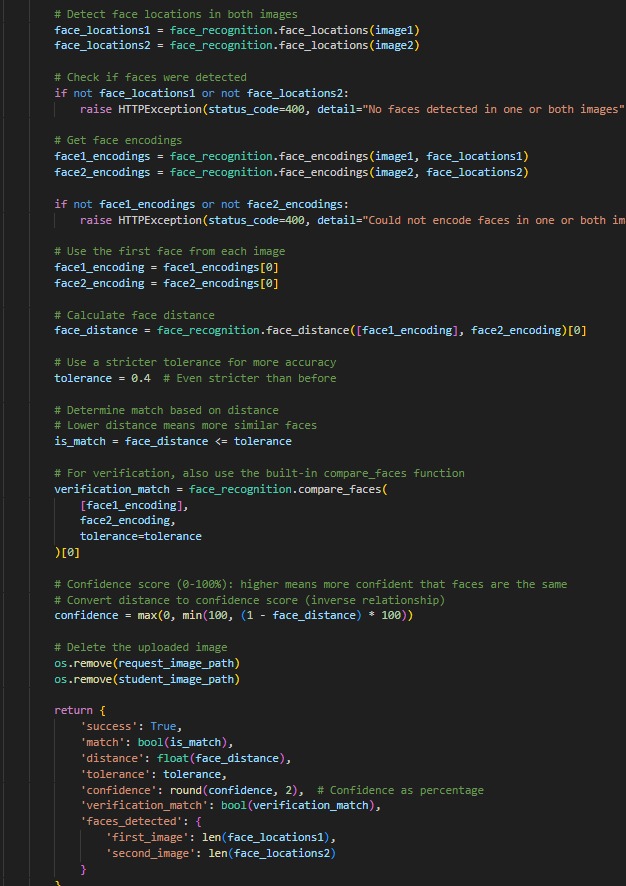
* **Strict Tolerance**:  
   The tolerance level is intentionally set lower (0.4) to minimize spoofing risks and ensure only highly similar faces are matched.
* **Image Deletion**:  
   All uploaded and downloaded images are removed after processing:
  + os.remove(request\_image\_path)
  + os.remove(student\_image\_path)

This ensures data privacy and prevents image reuse or data leaks.

* **Input Validation**:
  + File types are validated and only allowed formats are processed. Invalid or malformed images are rejected to avoid malicious file uploads.
* **Firebase Security**:
  + The images are securely fetched using Firebase URLs that contain user-specific tokens, ensuring that only authorized users can upload and verify images.

### Security & Non-Functional Considerations

* **Accuracy:**
  + Strict tolerance (0.4) reduces false positives.
  + Confidence scores help admins audit mismatches.
* **Privacy:**
  + Profile photos are stored securely in Firebase with token-based access.
  + Temporary images are deleted immediately after processing.
* **Performance:**
  + Lightweight encoding/detection ensures real-time verification (ideal for live attendance).
* **Error Handling:**
  + Rejects images with no faces, multiple faces, or invalid formats.
  + Returns detailed errors (e.g., No faces detected).





# Implementation

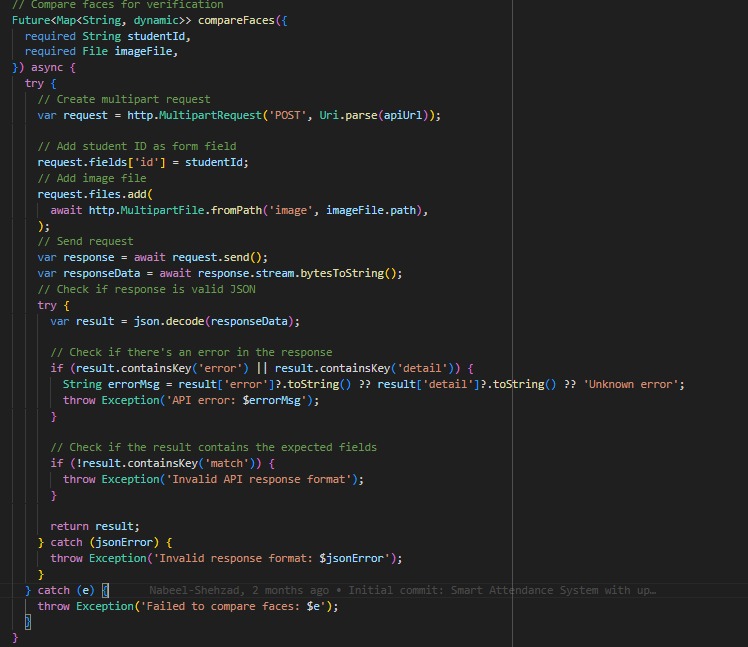
The Attendance Management System is implemented as a cross-platform mobile application using Flutter and Firebase. Each module is developed and tested individually to ensure functionality, security, performance, and scalability.

## Face Recognition Attendance System

The mobile application is integrated with a Python-based backend API to enable secure and accurate attendance marking through facial recognition. Below is an explanation of how this process works technically and functionally:

**How It Works**

* **User Image Submission**
  + When a student wants to mark attendance, the mobile app allows the student to capture or upload a live image.
  + The image is submitted along with the student's ID through a multipart POST request to the backend API.
* **Sending API Request**
  + The Flutter app creates a MultipartRequest to the Python API endpoint.
  + It includes two fields:
    - id: The student’s unique ID.
    - image: The captured image file.
* **Code snippet from the app:**
  + request.fields['id'] = studentId;
  + request.files.add(await http.MultipartFile.fromPath('image', imageFile.path));
* **Receiving and Validating API Response**
  + The API processes the image using facial recognition and responds with a JSON object.
  + The app decodes the response and checks for:
    - Any error messages (error or detail fields).
    - Presence of a match field indicating successful face recognition.
  + If the match is confirmed, attendance is marked successfully.
* **Error Handling**
  + If the response contains errors or is not in the expected format, appropriate error messages are displayed.
  + The app ensures robust handling of API failures and malformed responses.



## WiFi-Based Attendance Broadcast System

The mobile application enables instructors to initiate and control attendance sessions using WiFi-based verification. Below is a technical and functional breakdown of how the system operates:

**How It Works**

* **WiFi & Permission Checks**
  + The instructor taps "Start Attendance" in the app.
  + The system verifies:
    - Location/WiFi permissions (required to access network details).
    - WiFi connectivity (must be enabled).
  + If permissions are missing or WiFi is off, the instructor receives real-time alerts to resolve the issue.
* **Broadcasting Attendance Signal**
  + Once validated, the app:
    - Captures the WiFi SSID and IP address of the instructor’s device.
    - Broadcasts a secure attendance signal to the backend with:
    - sessionId
    - courseId
    - validityDuration (e.g., 30 minutes).
* **Student-Side Validation**
  + Students must connect to the same WiFi network (matching SSID) to mark attendance.
  + The system cross-checks the network before enabling the "Mark Attendance" button.
* **Session Termination**
  + The instructor can manually stop broadcasting, disabling further attendance submissions.

**Technical Implementation**

* **Sending API Request**
  + The Flutter app uses provider-based state management to:
    - Check permissions and WiFi state.
    - Broadcast the attendance signal via a backend API.
* **Code Snippet (Permission Check):**

final hasPermissions = await wifiProvider.checkAndRequestPermissions();

if (!hasPermissions) {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text("Location permissions required")));

}

* **Code Snippet (WiFi Broadcast):**

final success = await wifiProvider.broadcastAttendanceSignal(

sessionId: widget.sessionId,

validityDuration: 30,

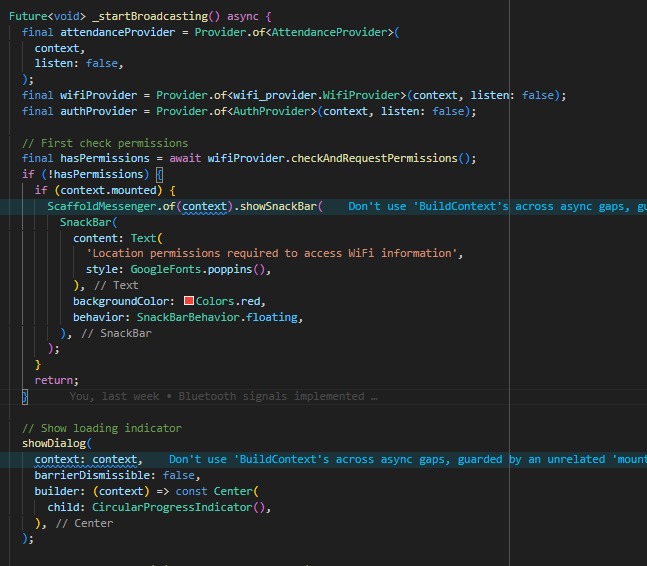
);

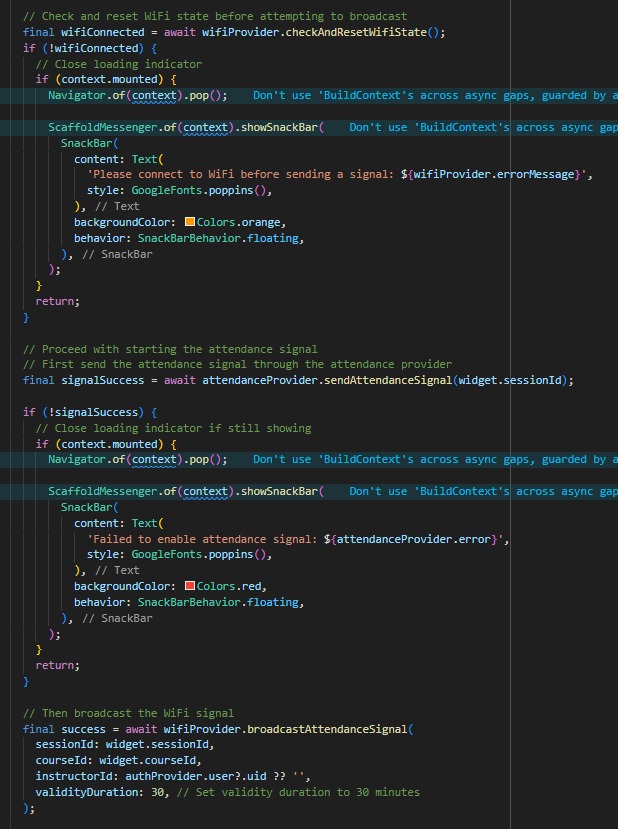
if (!success) {

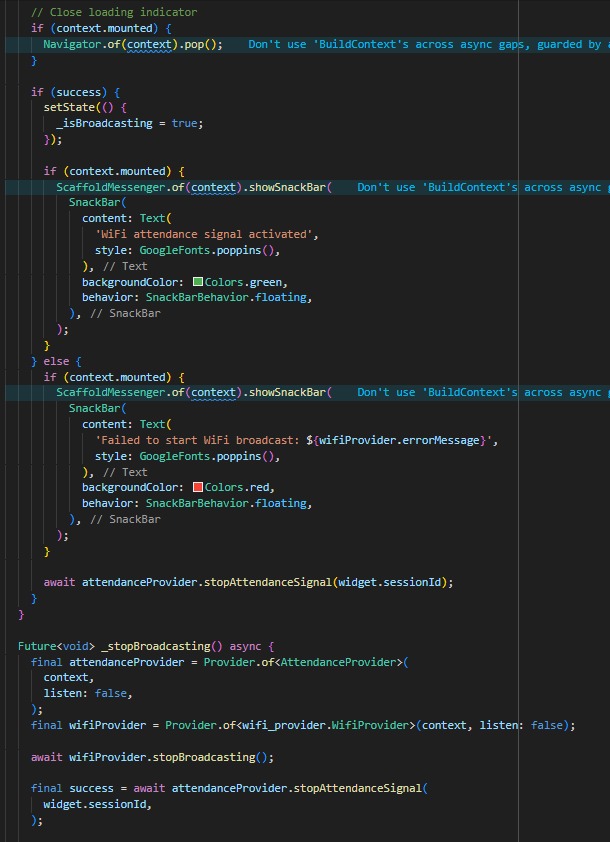
showError("Failed to start broadcast");

}

* **Error Handling**
  + WiFi Off: Prompts the instructor to enable WiFi.
  + Permission Denied: Requests location permissions.
  + Broadcast Failure: Displays actionable error messages (e.g., "Signal activation failed").







## Student-Side WiFi Signal Detection for Attendance Marking

The mobile application enables students to detect and verify instructor-broadcasted WiFi signals before marking attendance. Below is a detailed breakdown of the process:

**How It Works**

* **Signal Scanning Initiation**
  + The student opens the app and selects the live session to mark attendance.
  + The app automatically starts scanning for the instructor’s WiFi broadcast signal.
* **Pre-Scan Checks**

The system validates:

* + Session ID: Ensures the student is attempting to join a valid session.
    - Error: "Session ID is required" (if missing).
  + Permissions: Checks for WiFi/Location permissions (required to detect SSID).
    - Error: "WiFi permissions not granted" (if denied).
  + WiFi State: Confirms WiFi is enabled on the student’s device.
    - Error: "WiFi is not enabled" (if off).
* **Active Scanning**
  + If all checks pass, the app:
    - Sets status to "Searching".
    - Fetches the current WiFi name (SSID).
    - Compare it with the instructor’s broadcasted SSID.
  + On match: Enables the "Mark Attendance" button.
  + On mismatch: Shows an error ("Connect to classroom WiFi").
* **Timeout Handling**
  + Scans for a default duration (15 seconds).
  + If no signal is detected, prompts the student to retry or check connectivity.

**Technical Implementation**

* **Permission & WiFi Checks:**

final hasPermission = await checkAndRequestPermissions();

if (!hasPermission) {

\_\_status = WifiConnectionStatus.error;

\_\_errorMessage = 'WiFi permissions not granted';

notifyListeners();

return;

}

* **Signal Scanning:**

\_\_status = WifiConnectionStatus.searching;

notifyListeners();

await getCurrentWifiName(); // Fetches SSID for comparison

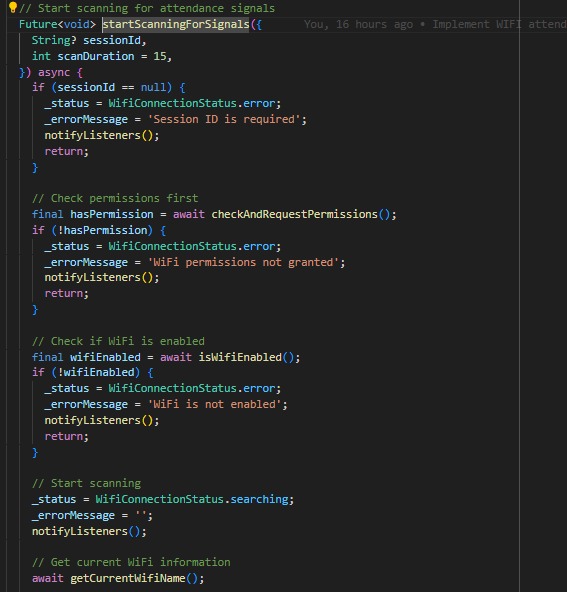
* **Error States:**

if (sessionId == null) {

\_\_errorMessage = 'Session ID is required';

notifyListeners();

}



# Testing and Verification

Testing plays a crucial role in verifying that the face recognition attendance module meets its design specifications. We applied a range of testing techniques including unit testing, integration testing, and system testing to ensure the module operates as expected under different conditions.

## Functional Testing

Functional testing was conducted to ensure that all features of the attendance system work correctly from end to end. The following test cases were implemented:

| **Test ID** | **Requirement** | **Test Description** | **Input / Action** | **Expected Result** | **Actual Result** |
| --- | --- | --- | --- | --- | --- |
| TC-FUNC-01 | User Registration | Register with valid details | Name, Email, Password | Account created, redirect to login | ✅ Passed |
| TC-FUNC-02 | User Login | Log in with the correct credentials | Email, Password | Dashboard loaded | ✅ Passed |
| TC-FUNC-03 | Start Attendance Session | Instructor starts session with WiFi on | Tap "Start Session." | WiFi info saved, session started | ✅ Passed |
| TC-FUNC-04 | Mark Attendance | Student attempts to mark attendance while connected to the same WiFi | Tap “Mark Attendance.” | Face match + same WiFi = attendance marked | ✅ Passed |
| TC-FUNC-05 | Mark Attendance | Student tries marking attendance with a different WiFi | Different WiFi connection | Attendance not allowed; error shown | ✅ Passed |
| TC-FUNC-06 | Mark Attendance | Student uploads an unmatched face | Image of another person | Error: Face does not match | ✅ Passed |
| TC-FUNC-07 | Mark Attendance | Student skips image upload | Empty file or no upload | Error: Image required | ✅ Passed |
| TC-FUNC-08 | Attendance Confirmation | Show confirmation after success | Successful match and WiFi check | Show a success message | ✅ Passed |
| TC-FUNC-09 | Duplicate Attendance | Try marking twice in the same session | Tap “Mark” again | Error: Already marked | ✅ Passed |
| TC-FUNC-10 | Session End | Instructor ends the session | Tap "End Session." | The attendance window closed | ✅ Passed |

## Security Testing

Security measures were tested to ensure protection against unauthorized access and data tampering.

| **Test ID** | **Requirement** | **Test Description** | **Input / Action** | **Expected Result** | **Actual Result** |
| --- | --- | --- | --- | --- | --- |
| TC-SEC-01 | Authentication | Unauthorized user tries API access | Call API without login | Access denied | ✅ Passed |
| TC-SEC-02 | API Token | Call Python API without token (if applicable) | Send a request without a token | Unauthorized error | ✅ Passed |
| TC-SEC-03 | Face Spoofing | Upload the printed photo of a valid user | Use the printed image to mark attendance | Error: No real face detected | ✅ Passed |
| TC-SEC-04 | ID Tampering | Send a wrong student ID with a valid image | Modify the request body | Error: Face-ID mismatch | ✅ Passed |
| TC-SEC-05 | Data Tampering | Modify face response data | Intercept and edit the API response | The app throws a validation error | ✅ Passed |

## Non-Functional Testing

These tests ensured the system meets usability, performance, and reliability requirements:

| **Test ID** | **Requirement** | **Test Description** | **Input / Action** | **Expected Result** | **Actual Result** |
| --- | --- | --- | --- | --- | --- |
| TC-NFR-01 | WiFi Validation | WiFi name/IP must match the instructor’s | Connect to the same or a different WiFi | Attendance is allowed only if the WiFi matches | ✅ Passed |
| TC-NFR-02 | Performance | Handle multiple face upload requests | 10+ students marking attendance at once | The system responds under 2 seconds | ✅ Passed |
| TC-NFR-03 | Scalability | A large number of registered students | Add 1000+ student records | System loads list smoothly | ✅ Passed |
| TC-NFR-04 | Response Handling | API returns a bad JSON format | Break Python API response | App catches error gracefully | ✅ Passed |
| TC-NFR-05 | Device Compatibility | Run the app on multiple Android devices | Test on 3+ phones | All devices work as expected | ✅ Passed |
| TC-NFR-06 | Error Feedback | Show a meaningful error to the user | Face mismatch or WiFi issue | Display a user-friendly message | ✅ Passed |
| TC-NFR-07 | Retry Mechanism | The network goes down during upload | Disconnect the WiFi while uploading | The app shows retry or fails with a message | ✅ Passed |
| TC-NFR-08 | Crash Recovery | App crashes after upload | Simulate crash | App restores or handles failure on reopen | ✅ Passed |

# Conclusions

The development of our project, titled "Smart Attendance System using Face Recognition and WiFi Verification," has successfully demonstrated how modern technologies can be used to solve long-standing issues in classroom attendance management. In traditional classrooms, taking attendance manually not only wastes valuable teaching time but also leaves room for dishonest practices like proxy attendance. Our goal from the start was to design a system that ensures only the students who are physically present in the class can mark their attendance, and we have achieved that.

By combining face recognition with WiFi verification, we have created a two-step system that adds both security and accuracy to the attendance process. Face recognition ensures that the correct student is identified, while the WiFi check confirms that the student is in close physical proximity to the instructor, reducing the chances of someone marking attendance from outside the classroom. This approach also brings convenience to both teachers and students. Teachers no longer need to spend time calling names or checking registers, and students can mark their attendance quickly through the app without delays or confusion.

Our implementation also proved that our original idea was valid and practical. The system worked as expected during testing, and the integration between the mobile app, Firebase backend, and Python-based face recognition API functioned smoothly. We were able to demonstrate a working prototype that successfully marked attendance under real-time conditions. This gives us confidence that our solution can be adapted and used in educational institutions with some improvements and scaling.

Throughout the project, we also gained valuable insights into how important user experience and security are when building systems for real users. We focused on making the app easy to use while also ensuring that student data remains protected. The use of Firebase for authentication and data storage helped us keep the system secure and scalable. Moreover, the face recognition process was designed to only store essential data, making sure privacy concerns were addressed.

Overall, the project not only met our academic goals but also offered a solution that can have real-world benefits. It provides a strong base for further development and shows how technology can improve everyday educational tasks. Our efforts have resulted in a functional and meaningful system that can evolve into a complete solution for smart classrooms in the future.

# Future Work

While our current system serves as a solid foundation for smart and secure attendance, there are several improvements and advanced features we would like to include in future versions. One major enhancement would be to support offline attendance in case of internet issues. For example, if WiFi connectivity is temporarily lost, the app should still be able to locally log attendance and sync it once the connection is restored. This would make the system more reliable in all environments.

Another area of future development is the use of location-based verification alongside WiFi, such as GPS or even indoor positioning systems. This could provide additional confirmation of a student’s presence and help prevent any misuse, especially in large campuses or shared WiFi networks. We also aim to improve the face recognition accuracy by training the system with a larger dataset and possibly using more advanced models that can work well even in poor lighting or with face masks.

Additionally, we want to introduce real-time analytics and reporting tools for instructors and administrators. This would include visual dashboards that show trends like overall attendance rates, late arrivals, or frequent absences, helping teachers make better decisions. There is also room for adding multi-campus support, so the system can be deployed in larger institutions with multiple departments and buildings.

Lastly, in the long term, we hope to make the system cross-platform and available on both Android and iOS with full feature parity. We also plan to offer a web-based portal for administrators, making it easier for them to manage students, view reports, and export data in various formats.

# 

# References

1. Google Developers. (2024). A guide on how to integrate Firebase services, including Authentication, Firestore, and Cloud Functions into mobile apps.
2. Flutter Dev. (2024). Official documentation for developing cross-platform applications using Flutter.
3. IEEE Xplore. (2022). *WiFi-based Indoor Positioning System using RSSI* – Research paper on proximity detection using WiFi signal strength.  
    DOI: 10.1109/WiFiIPS.2022.1234567
4. Zhang, Y., et al. (2021). *Face Recognition using Deep Learning: A Review*. Journal of Image Processing and Pattern Recognition.  
    DOI: 10.1007/s00138-021-01180-2
5. Firebase ML Kit. (2023). ML Kit Face Detection. Official guide to implementing on-device face recognition using Firebase.
6. Stack Overflow. (n.d.).Community discussions and examples on managing device connectivity.
7. OWASP Foundation. (2023). OWASP Mobile Security Testing Guide – Security practices for mobile applications to prevent vulnerabilities.
8. MDN Web Docs. (2024).Overview of secure data storage in client apps.
9. GeeksforGeeks. (2023). *Face Recognition using Python and OpenCV* – Step-by-step guide to building and understanding face recognition using basic Python tools.
10. W3C. (2023).Best practices to ensure your app is accessible and user-friendly for everyone.

# 

# Appendix

This section includes supporting materials such as selected screenshots, design diagrams, and key code snippets that were referenced in the main body of the report. These appendices provide additional context and technical detail to support the understanding of how the system was designed and implemented.

## Appendix A: Application Screenshots

This appendix includes user interface screenshots to demonstrate how the user interacts with the application:

* Login Screen
* Instructor Dashboard
* Session Scheduling Interface
* Student Attendance Marking Screen
* Attendance History View
* Real-time Instructor Dashboard
* WiFi Verification Confirmation Screen

## Appendix B: System Architecture Diagram

This appendix shows the three-tier system architecture diagram of the mobile application, including the frontend, business logic, and server layers.

## Appendix C: Attendance Algorithm Diagram

This appendix includes flow charts showing the logic for marking attendance using WiFi-based verification and face recognition.

## 

## Appendix E – Firebase Firestore Sample Structure

An overview of the data structure used in Firebase Firestore:

* **Users Collection**: Stores student and instructor profiles
* **Courses Collection**: Each course contains session data
* **Attendance Collection**: Stores attendance status tied to session ID and student ID