IIT Delhi Attendance System - Report

Harshit Mawandia   2020CS10348

December 8, 2023

1 Introduction

IIT Delhi has been using an outsourced app "TIMBLE" to monitor the student attendance. Not only does it costs the institute heavily, but also has some easily visible flaws and bugs, which make the system untrustworthy. So we aim to fix as many of the flaws as possible, and create a new system, which is both secure, and easy to use for the students, the faculty and the management.

2 Review

We will see what the strengths and weaknesses of the previous system was, and how we plan to bridge those gaps.

2.1 TIMBLE

2.1.1 Strengths

- Has a face detection supprot to mark attendance
- Has a proximity sensing system that creates a radius around each lecture hall for location identification.
- Has 2 ways to mark attendance - a. Personal Mobile App, b. Lecture Hall Tablets

2.2 Weaknesses

- Face Detection does not match a persons face, so anyone can mark anyone's attendance
- Proximity radius allows people to mark attendance from some spots outside the Lecture Halls
- Has a bug which allows people to modify device time and mark attendance in the past and future
- Doesn't share the database with IIT Delhi, so the data we get isn't always how we want it and in big CSV files
- Multiple Logins are allowed, so one person can mark the attendance of many students

2.3 Addressing the gaps

We will try to bridge as many of these gaps as possible by our new attendance system.

- We create an app that requires Student to scan the code from Lecture Hall tablet, ensuring proximity. We will add Bluetooth proximity later to provide more security.
- We use serverside timestamp, which is how most of the students mark proxy.
- We use QR scanning to mark attendances, and each QR will have a 15 second expiry which ensures that proxies would be very difficult to manage.
- We try to create a one to one mapping between studetns and devices to prevent multiple login.
- since this is an inhouse project, our data can be utilised and maintained as necessary.

3 System Architecture

3.1 Overview

The attendance system comprises a comprehensive architecture that seamlessly integrates mobile applications, a web portal, a Django-based backend server, and a PostgreSQL database. The entire system is designed to ensure efficiency, security, and cross-platform compatibility.

3.2 Components

3.2.1 Mobile Applications (Flutter)

The attendance system's mobile applications are developed using Flutter, offering a unified codebase for both Android and iOS platforms. `flutter_udid`, a dedicated Flutter plugin, is employed to uniquely identify each device using the UDID (Unique Device Identifier). This identifier plays a crucial role in preventing multiple logins from a single device.

3.2.2 Web Portal (React)

The web portal is built on React, providing a responsive and user-friendly interface for accessing attendance-related information. It shares data and functionalities with the mobile applications through API calls to the Django backend.

3.2.3 Django Backend

The core logic and functionality of the attendance system reside in the Django-based backend server. It acts as a bridge between the front-end applications and the database, handling user authentication, data retrieval, and updates. The backend server communicates with the Flutter applications and React web portal through RESTful APIs.

3.2.4 PostgreSQL Database

Data persistence is managed by a PostgreSQL database. It stores essential information such as user details, attendance records, and device-UDID mappings. The relational nature of PostgreSQL ensures efficient data retrieval and management.

3.3 Device Identification and Security

To ensure a single login per device, the `flutter_udid` plugin is utilized to map each device to a UDID. This Unique Device Identifier is a critical security measure, preventing unauthorized access and multiple logins. However, it is essential to acknowledge the assumption that the `flutter_udid` plugin is secure and cannot be easily spoofed.

4 Methodology

4.1 Workflow Overview

The methodology employed for the development and implementation of the attendance system involves a systematic process, utilizing various technologies and plugins. The workflow is designed to ensure a secure and efficient attendance tracking system.

4.2 Steps in the Workflow

4.2.1 Student Device Registration

- Students initiate the process by logging into the web portal for one-time device registration.
- Accessing the "Register Your Device" page, they obtain a QR code containing encrypted user information.

4.2.2 Device Registration via QR Code

- Using the student-side app, students scan the QR code acquired during device registration.
- The scanned information registers the device, logs in the student, and directs them to the homepage in the app, displaying their unique electronic ID (e-ID).

4.2.3 Attendance Marking Process

- To mark attendance during a class, the student accesses the QR page on the app to generate a QR code.
- The generated QR code is then scanned using the LHC Tablets during the class.

4.2.4 Data Transmission and Decryption

- The tablet-side app decrypts the QR data and sends it to the server through an API call.

4.2.5 Server-Side Processing

- The server performs a series of checks to validate attendance.
  - Matches the device ID of the student and the tablet to determine the active course in the LH.
  - Verifies the timestamp to check if the student has a registered course during that time-slot, matching the active course in the LH.
  - Checks if the timestamp is not expired.

4.2.6 Attendance Verification and Notification

- If the criteria are satisfied, attendance is marked.
- If not, the LH Tablet shows an error, notifying the user of the issue.

4.2.7 Attendance Records

- Students can check their attendance records through either the mobile app or the web portal.

4.3 Technologies and Tools Used

4.3.1 Mobile App Development (Flutter)

- Flutter is utilized for developing both the student-side and tablet-side applications, ensuring cross-platform compatibility.

4.3.2 Web Portal Development (React)

- React is employed for building the web portal, providing an interactive and responsive interface.

4.3.3 Backend Development (Django)

- Django serves as the backend framework, managing data processing, user authentication, and communication between the front-end and the database.

4.3.4 Database Management (PostgreSQL)

- PostgreSQL is chosen for its relational database management capabilities, storing user details, attendance records, and device-UDID mappings.

4.3.5 Testing (Postman)

- Postman is used for testing API endpoints, ensuring seamless communication between components.

This methodology ensures a robust and secure attendance system, integrating various technologies and tools to streamline the process for both students and administrators.

5 System Design

5.1 Architecture Overview

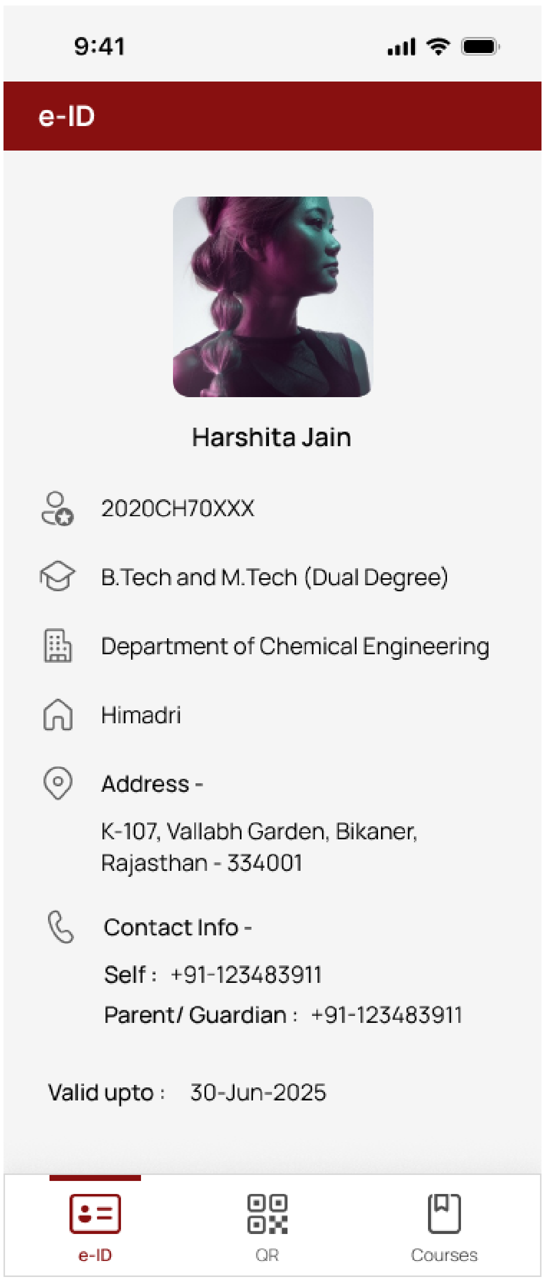
The system is designed as a client-server architecture with a focus on modularity and scalability. It comprises mobile applications built with Flutter for both students and tablets, a React-based web portal, and a Django backend server connected to a PostgreSQL database. The system ensures cross-platform compatibility for mobile applications.

5.2 Client-Side Design

5.2.1 Student Mobile App (Flutter)

- **Login and Device Registration:**
  - Students log in for one-time device registration via the web portal.
  - A QR code containing encrypted user info is generated during device registration.
  - The Flutter app scans this QR for device registration, utilizing the `flutter_udid` plugin for unique device identification.
- **Homepage:**

– After successful registration, students are directed to the homepage displaying their electronic ID (e-ID).


9:41

e-ID

Harshita Jain

2020CH70XXX

B.Tech and M.Tech (Dual Degree)

Department of Chemical Engineering

Himadri

Address -

K-107, Vallabh Garden, Bikaner, Rajasthan - 334001

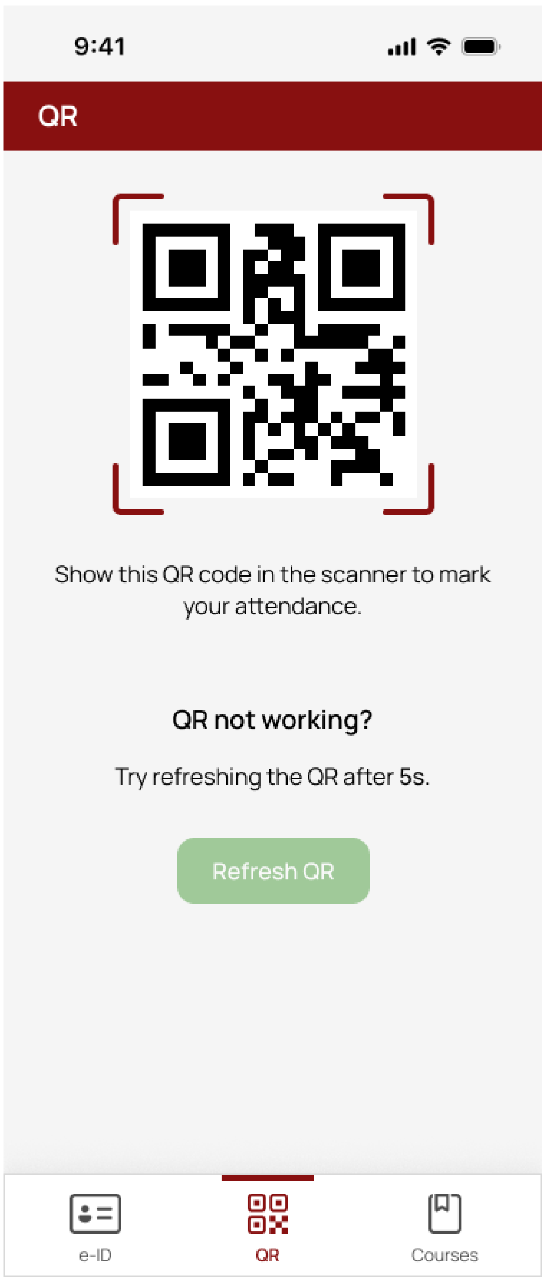
Contact Info -

Self : +91-123483911

Parent/ Guardian : +91-123483911

Valid upto : 30-Jun-2025

(a) eID card


9:41

QR

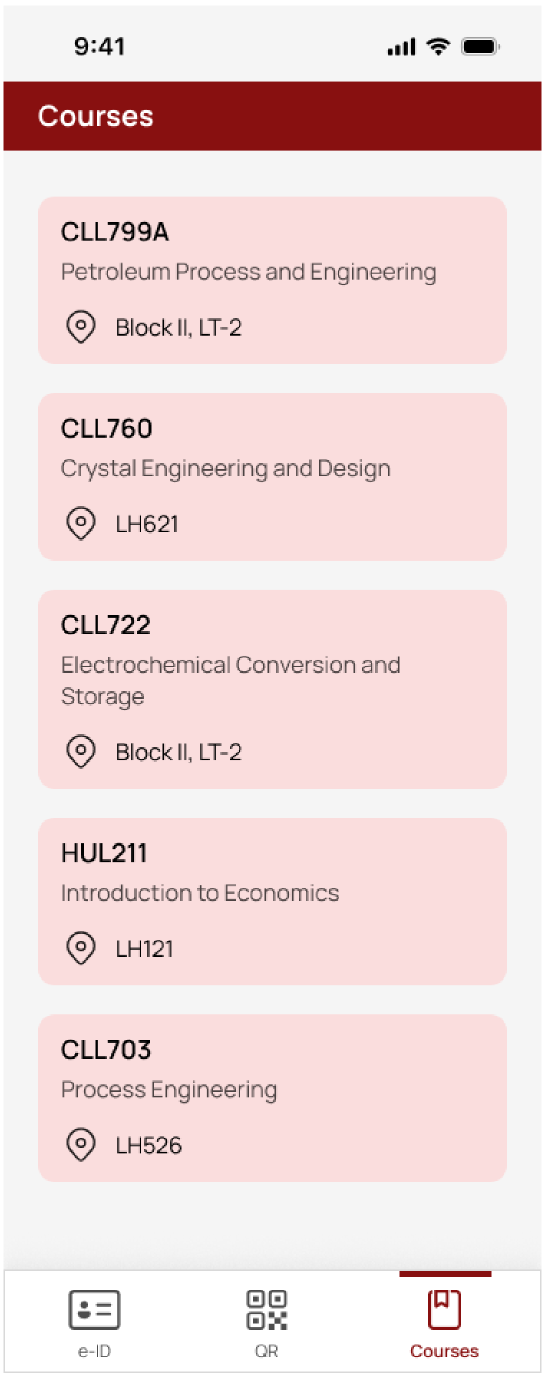
Show this QR code in the scanner to mark your attendance.

QR not working?

Try refreshing the QR after 5s.

(b) QR to mark attendance

- **Attendance Marking:**
  - Students generate a QR code on the app for attendance marking during class.
  - QR code is scanned by LH Tablets.
- **View Attendance:**
  - Students can view their attendance records through the app.



Courses

CLL799A
Petroleum Process and Engineering
Block II, LT-2

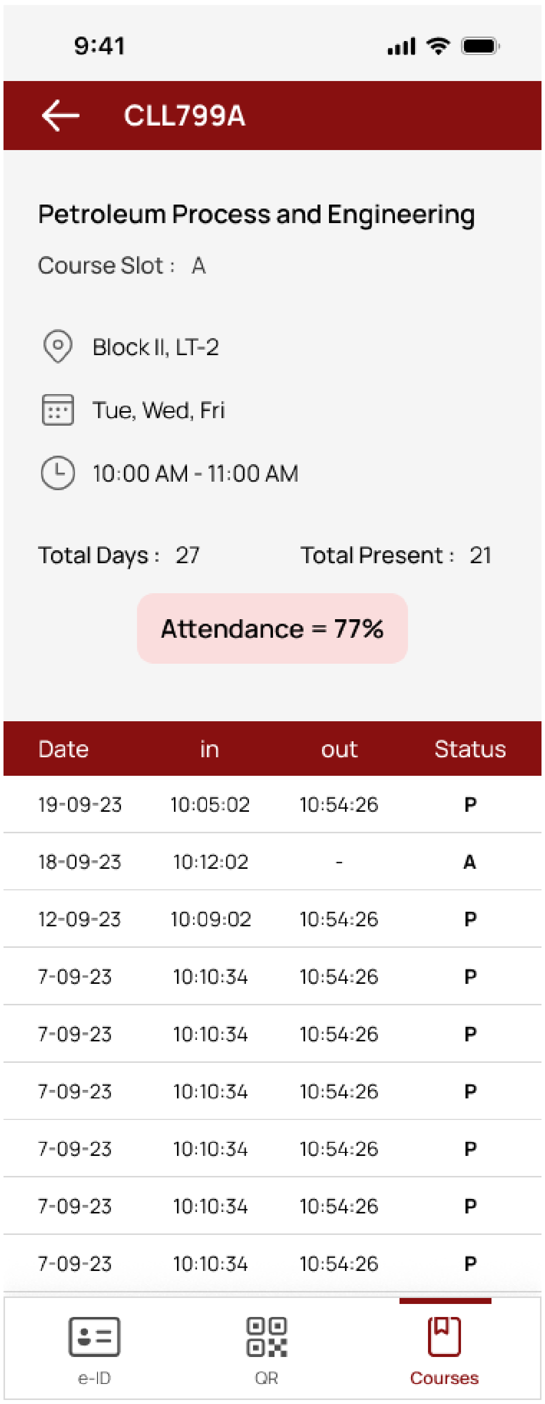
CLL760
Crystal Engineering and Design
LH621

CLL722
Electrochemical Conversion and Storage
Block II, LT-2

HUL211
Introduction to Economics
LH121

CLL703
Process Engineering
LH526

(a) Courses overview



CLL799A

Petroleum Process and Engineering
Course Slot : A

Block II, LT-2
Tue, Wed, Fri
10:00 AM - 11:00 AM

Total Days : 27
Total Present : 21

Attendance = 77%

Date	in	out	Status
19-09-23	10:05:02	10:54:26	P
18-09-23	10:12:02	-	A
12-09-23	10:09:02	10:54:26	P
7-09-23	10:10:34	10:54:26	P
7-09-23	10:10:34	10:54:26	P
7-09-23	10:10:34	10:54:26	P
7-09-23	10:10:34	10:54:26	P
7-09-23	10:10:34	10:54:26	P
7-09-23	10:10:34	10:54:26	P

(b) Attendance for a course

5.2.2 Tablet Mobile App (Flutter)

- **QR Decryption and Data Transmission:**
  - Tablet app decrypts the QR data and sends it to the server through an API call.

5.3 Server-Side Design

5.3.1 Django Backend

- **API Endpoints:**
  - Exposes RESTful APIs for communication between the front-end and the server.
  - Utilizes JWT tokens for session management, ensuring secure and authenticated access.
- **Data Processing:**
  - Validates and processes attendance data received from tablet apps.
  - Checks device ID, active course, timestamp, and other criteria for attendance marking.

- **Database Interaction:**
  - Interacts with the PostgreSQL database to store and retrieve user details, attendance records, and other relevant data.

5.4 Database Design

The system utilizes a PostgreSQL database to store and manage various entities related to course registration, attendance tracking, and user information. Below is the detailed description of the database schema:

5.4.1 Course (`iitd_schema_course`)

- **Attributes:**
  - `courseCode` (Primary Key, varchar(10)): Course code uniquely identifying each course.
  - `courseName` (varchar(100)): Name of the course.

5.4.2 CourseSlot (`iitd_schema_courseslot`)

- **Attributes:**
  - `id` (Primary Key, bigint): Auto-generated identifier for each course slot.
  - `numberOfClasses` (integer): Number of classes scheduled for the course slot.
- **Foreign Keys:**
  - `course_id` (References `iitd_schema_course(courseCode)`): Links to the corresponding course.
  - `lectureHall_id` (References `iitd_schema_lecturehall(lectureHallName)`): Links to the corresponding lecture hall.
  - `semester_id` (References `iitd_schema_semester(id)`): Links to the corresponding semester.
  - `slotGroup_id` (References `iitd_schema_slotgroup(slotGroupName)`): Links to the corresponding slot group.
- **Constraints:**
  - `unique_course_slot`: Ensures the uniqueness of the combination of course, slot group, and semester.

5.4.3 LectureHall (`iitd_schema_lecturehall`)

- **Attributes:**
  - `lectureHallName` (Primary Key, varchar(100)): Name of the lecture hall.
  - `lectureHallCapacity` (integer): Capacity of the lecture hall.

5.4.4 Semester (`iitd_schema_semester`)

- **Attributes:**
  - `id` (Primary Key, bigint): Auto-generated identifier for each semester.
  - `academicYear` (varchar(10)): Academic year of the semester.
  - `semesterNumber` (integer): Number indicating the semester.
  - `startDate` (date): Start date of the semester.
  - `endDate` (date): End date of the semester.

5.4.5 Slot (`iitd_schema_slot`)

- **Attributes:**
  - `id` (Primary Key, bigint): Auto-generated identifier for each slot.
  - `slotName` (varchar(10)): Name of the slot.
  - `slotStartTime` (time): Start time of the slot.
  - `slotEndTime` (time): End time of the slot.
  - `slotDay` (integer): Day of the week for the slot.

5.4.6 Student (`iitd_schema_student`)

- **Attributes:**
  - `kerberosID` (Primary Key, varchar(10)): Unique identifier for each student.
  - `entryNumber` (varchar(11)): Entry number of the student.
  - `name` (varchar(100)): Name of the student.
  - `department` (varchar(100)): Department of the student.
  - `deviceUUID` (varchar(100) UNIQUE): Unique identifier for the device used by the student.
  - `mail` (varchar(100)): Email address of the student.

5.4.7 StudentRegistration (`iitd_schema_studentregistration`)

- **Attributes:**
  - `id` (Primary Key, bigint): Auto-generated identifier for each student registration.
  - `numberOfClassesAttended` (integer): Number of classes attended by the student for the registered course slot.
- **Foreign Keys:**
  - `courseSlot_id` (References `iitd_schema_courseslot(id)`): Links to the corresponding course slot.
  - `student_id` (References `iitd_schema_student(kerberosID)`): Links to the corresponding student.
- **Constraints:**
  - `unique_student_registration`: Ensures the uniqueness of the combination of student and course slot.

5.4.8 SlotGroup (`iitd_schema_slotgroup`)

- **Attributes:**
  - `slotGroupName` (Primary Key, varchar(10)): Name of the slot group.

5.4.9 SlotGroup_Slots (`iitd_schema_slotgroup_slots`)

- **Attributes:**
  - `id` (Primary Key, integer): Auto-generated identifier for each entry.
- **Foreign Keys:**
  - `slotgroup_id` (References `iitd_schema_slotgroup(slotGroupName)`): Links to the corresponding slot group.
  - `slot_id` (References `iitd_schema_slot(id)`): Links to the corresponding slot.
- **Constraints:**
  - `unique_slotgroup_slots`: Ensures the uniqueness of the combination of slot group and slot.

5.4.10 LectureHallTablet (`iitd_schema_lecturehalltablet`)

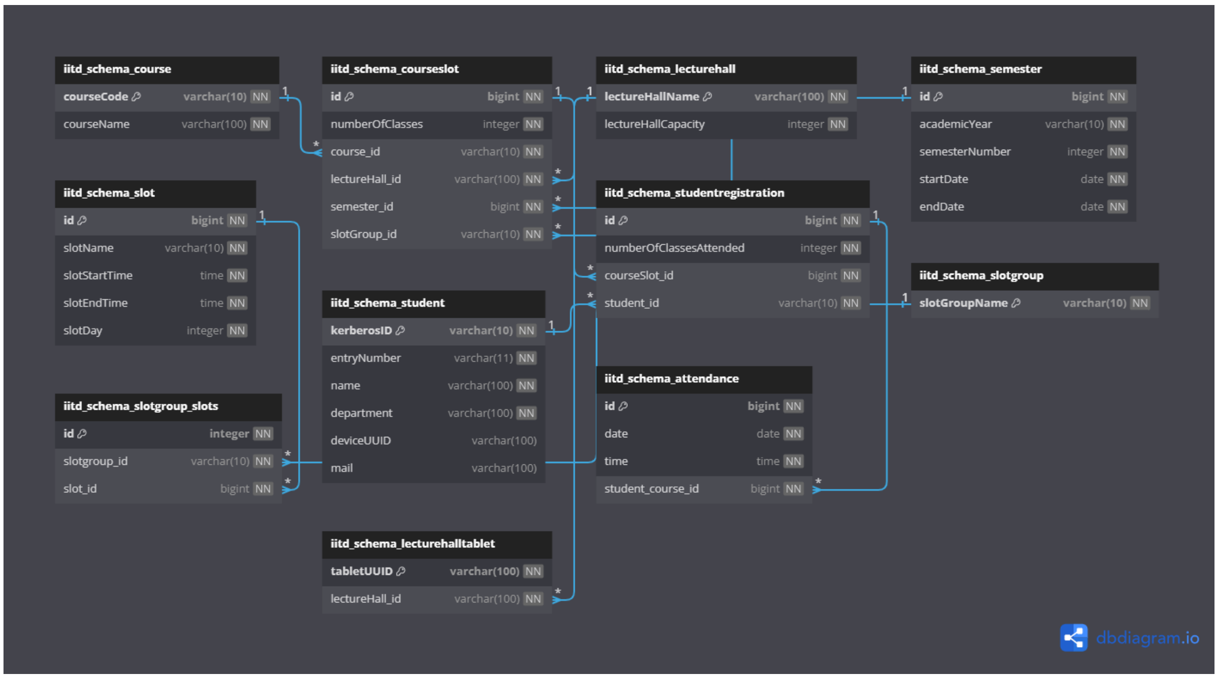
- **Attributes:**
  - `tabletUUID` (Primary Key, varchar(100)): Unique identifier for each tablet.
- **Foreign Keys:**
  - `lectureHall_id` (References `iitd_schema_lecturehall(lectureHallName)`): Links to the corresponding lecture hall.

5.4.11 Attendance (`iitd_schema_attendance`)

- **Attributes:**
  - `id` (Primary Key, bigint): Auto-generated identifier for each attendance record.
  - `date` (date): Date of the attendance record.
  - `time` (time): Time of the attendance record.
- **Foreign Keys:**
  - `student_course_id` (References `iitd_schema_studentregistration(id)`): Links to the corresponding student registration.
- **Constraints:**
  - `unique_attendance`: Ensures the uniqueness of the combination of student registration and date.

5.4.12 Conclusion

The database schema is designed to maintain a structured and normalized representation of the data, ensuring efficient data retrieval and integrity. Relationships between entities are established through foreign key constraints, and indexes are utilized to optimize query performance.



iitd_schema_course
courseCode varchar(10) NN
courseName varchar(100) NN
iitd_schema_courseslot
id bigint NN
numberOfClasses integer NN
course_id varchar(10) NN
lectureHall_id varchar(100) NN
semester_id bigint NN
slotGroup_id varchar(10) NN
iitd_schema_lecturehall
lectureHallName varchar(100) NN
lectureHallCapacity integer NN
iitd_schema_semester
id bigint NN
academicYear varchar(10) NN
semesterNumber integer NN
startDate date NN
endDate date NN
iitd_schema_slot
id bigint NN
slotName varchar(10) NN
slotStartTime time NN
slotEndTime time NN
slotDay integer NN
iitd_schema_studentregistration
id bigint NN
numberOfClassesAttended integer NN
courseSlot_id bigint NN
student_id varchar(10) NN
iitd_schema_slotgroup
slotGroupName varchar(10) NN
iitd_schema_student
kerberosID varchar(10) NN
entryNumber varchar(11) NN
name varchar(100) NN
department varchar(100) NN
deviceUUID varchar(100)
mail varchar(100)
iitd_schema_slotgroup_slots
id integer NN
slotgroup_id varchar(10) NN
slot_id bigint NN
iitd_schema_attendance
id bigint NN
date date NN
time time NN
student_course_id bigint NN
iitd_schema_lecturehalltablet
tabletUUID varchar(100) NN
lectureHall_id varchar(100) NN
dbdiagram.io

Figure 3: Placeholder for Database Schema Image

5.5 Security Measures

- **JWT Tokens for Session Management:**
  - JWT tokens are employed to manage user sessions securely.
  - Tokens contain information about the user's authentication status and permissions, reducing the risk of unauthorized access.
- **Device Identification (UDID):**
  - The `flutter_udid` plugin ensures a unique device ID for each registration, preventing multiple logins from a single device.
- **HTTPS Protocol:**
  - All communication between the client applications and the server is secured using HTTPS to encrypt data during transmission.
- **Input Validation:**
  - Robust input validation is implemented at both the client and server sides to prevent injection attacks and ensure data integrity.

5.6 Workflow and Security Integration

- **Device Registration:**
  - Unique device identification (UDID) prevents multiple logins.
  - JWT tokens manage the session securely.
- **Attendance Marking:**
  - Tablet apps decrypt QR data and securely transmit it to the server.
  - Server validates device ID, course details, and timestamp before marking attendance.
- **Data Transmission:**
  - All data transmission between clients and the server is encrypted using HTTPS.
- **Input Validation:**
  - Stringent input validation measures are in place to prevent security vulnerabilities.

5.7 Conclusion

The system design ensures a secure, scalable, and modular architecture. Integrating JWT tokens, device identification, and encryption measures, the attendance system maintains the confidentiality and integrity of user data while providing a seamless user experience. The implementation aligns with the outlined workflow, emphasizing security throughout the entire process.

5.8 Future Work

The current system, while ready for deployment, has opportunities for future enhancements and expansions. The following areas represent potential avenues for further development:

- **Integration with ID Card Data:** To fully enable the E-ID feature, there is a need to integrate with the ID card data from the Computer Services Centre (CSC).
- **Bluetooth Proximity Sensing:** Implementing Bluetooth proximity sensing can enhance proxy prevention measures, providing an additional layer of security for attendance tracking.
- **QR Code Scanning:** Enable tablet-side QR code scanning and user-side scanning to improve the efficiency and accuracy of data collection during attendance tracking.
- **Alpha Testing:** Prior to full-scale deployment, conduct alpha testing within a small group of students to gather feedback, identify potential issues, and refine the user experience.
- **Scalability Testing:** Conduct thorough testing to assess the system's scalability, ensuring its performance and reliability as the user base and data volume increase.
- **Staff-Side Portal:** Develop a portal for staff members to facilitate the entry of course registration data, streamlining administrative processes.
- **Extended E-ID Features:** Extend the E-ID feature to cover other services within IIT Delhi, such as library access, hospital services, and more, with the aim of fully replacing RFID-based systems.

While the system is poised for initial testing, these future steps will contribute to its continuous improvement and adaptation to the evolving needs of the academic environment.

5.9 Deployment and Testing Plan

In the immediate future, the following steps are planned for the deployment and testing of the system:

- **Data Gathering from CSC:** Initiate discussions with the CSC to gain access to course data and explore the possibility of setting up APIs for a permanent dynamic solution.
- **Alpha Testing with Small Group:** Conduct alpha testing within a small group of students to assess user experience, identify potential challenges, and make necessary refinements.
- **Initial Testing for Courses:** Begin testing the system with a limited number of courses during the next semester to validate its effectiveness and gather real-world feedback.
- **Security Verification:** Perform a thorough security review of the Flutter_UDID library to ensure its resistance against spoofing. If vulnerabilities are identified, take appropriate measures to enhance security and prevent potential exploits.

These steps will ensure a systematic and well-considered approach to the deployment, testing, and refinement of the system, paving the way for a successful and comprehensive implementation.