॥वसुधैव कुटुम्बकम्॥

SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE

Pune – 412115, Maharashtra State, India

https://www.sitpune.edu.in/

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**AY 2025-26**

**A PROJECT REPORT**

**ON**

# "College Event & Attendance Manager"

BACHELOR OF TECHNOLOGY COMPUTER SCIENCE & ENGINEERING

Submitted By:



| Name | PRN |
|---|---|
| Gargi Mittal | 22070122064 |
| Meet Golani | 22070122071 |
| Aryan Malkani | 23070122502 |
| Yogita Beniwal | 23070122506 |

## Problem Statement

In colleges, students often miss important event updates, and attendance tracking is still managed manually. This leads to inefficiency, loss of participation, and difficulty in maintaining accurate records. A digital solution is required to streamline event communication, automate attendance, and provide analytics.

## Key Features of the Project

1. **Event Management**
    - Create, update, and view upcoming events.
    - Event notifications to students.
2. **Attendance Management**
    - QR code generation for students.
    - Faculty scans QR to mark attendance.
3. **Analytics Dashboard**
    - Attendance % tracking for each student.
    - Reports for event participation.
4. **Feedback & Surveys Module**
    - Students can submit ratings and comments.
    - Faculty can view feedback reports.

**Additional Features:**

- Role-based login (Student, Faculty).
- Push/email notifications for event reminders.

## Proposed Technology Stack

1. Backend: Spring Boot (Java)
2. Frontend: React.js
3. Database: MySQL
4. Authentication: JWT-based Authentication
5. QR Code: ZXing library (Java) or third-party QR libraries

## Rest Endpoints

**Authentication Endpoints (/api/auth)**

- POST /register: Public endpoint for a new user (student) to register.
- POST /login: Public endpoint for any user to log in and receive a JWT.

**User Endpoints (/api/users)**

- GET /me: Retrieves the profile of the currently logged-in user. (All Roles)
- GET /my-qr-code: Generates and returns the QR code image for the logged-in student. (Student Only)

**Event Endpoints (/api/events)**

- POST /: Create a new event. (Faculty Only)
- GET /: Get a list of all upcoming events. (All Roles)
- GET /{eventId}: Get details of a specific event. (All Roles)
- PUT /{eventId}: Update an existing event. (Faculty Only - creator of the event)
- DELETE /{eventId}: Delete an event. (Faculty Only - creator of the event)
- POST /{eventId}/register: Register the logged-in student for an event. (Student Only)
- GET /{eventId}/registrations: Get a list of all students registered for an event. (Faculty Only)

**Attendance Endpoints (/api/attendance)**

- POST /mark: Marks a student's attendance via QR code scan. Request body: {"qrData": "...", "eventId": ...}. (Faculty Only)
- GET /event/{eventId}: Get a list of all students who attended a specific event. (Faculty Only)

**Analytics Endpoints (/api/analytics)**

- GET /student/{studentId}/attendance-percentage: Get the overall attendance percentage for a specific student. (Faculty, or the student themselves)
- GET /event/{eventId}/participation-report: Get a participation summary for an event (e.g., registered vs. attended count). (Faculty Only)

**Feedback Endpoints (/api/feedback)**

- POST /event/{eventId}: Submit feedback for an event the student attended. (Student Only)
- GET /event/{eventId}: View all feedback for a specific event, including average ratings. (Faculty Only)

## Technical Implementation (Spring Boot)

**1. Security & Authentication (Spring Security + JWT)**

- **Setup:** Use Spring Security to manage authentication and authorization.
- **Login Flow:**
    1. A user sends their email and password to a /api/auth/login endpoint.
    2. Spring Security's AuthenticationManager validates the credentials against the users table.
    3. If successful, your JwtTokenProvider service generates a JWT. This token will contain the user's ID, email, and role (e.g., 'STUDENT') in its payload.
    4. The JWT is sent back to the React client.
- **Authorization:**

1. The React client stores the JWT (in localStorage or sessionStorage) and includes it in the Authorization: Bearer <token> header of all subsequent API requests.
2. A custom JwtRequestFilter intercepts every request, validates the token, and sets the user's authentication context.
3. You can then secure endpoints using annotations like @PreAuthorize("hasRole('FACULTY')") on controller methods.

**2. QR Code Management (ZXing Library)**

- **Generation:**
    1. When a student user is created, generate a unique, hard-to-guess string (like a UUID.randomUUID().toString()) and save it in the qr_code_data column of the users table.
    2. Create an endpoint (e.g., GET /api/users/my-qr-code).
    3. This endpoint's service method will fetch the user's qr_code_data, use the ZXing library to encode this string into a QR code image (as a byte[]), and return it as a PNG or JPEG response. The React frontend can then display this image.
- **Scanning & Attendance Marking:**
    1. The Faculty user, using the React app on a mobile device, will use a frontend library (like react-qr-scanner) to access the camera.
    2. The library scans the student's QR code and decodes it back to the unique string (qr_code_data).
    3. The frontend then makes a POST request to an endpoint like POST /api/attendance/mark. The request body will contain the qr_code_data and the eventId.
    4. The backend service finds the user by their qr_code_data. If the user and event exist, it creates a new record in the attendance table.

**3. Analytics Dashboard Logic**

This involves creating custom queries in your Spring Data JPA repositories.

- **Student Attendance Percentage:**

- Create a method in AttendanceRepository or a dedicated AnalyticsService.
- The logic would be: (Number of events attended by student / Number of events registered for by student) * 100.
- This can be implemented with a custom @Query that performs COUNT operations on the attendance and event_registrations tables for a given user_id.

- **Event Participation Reports:**
  - For a given eventId, you'll need queries to get:
    - Total number of registrations (COUNT from event_registrations).
    - Total number of attendees (COUNT from attendance).
    - A list of all attendees.

**4. Notifications (Spring Mail / Push Notifications)**

- **Implementation:** Use the spring-boot-starter-mail dependency for sending emails.
- **Asynchronous Operations:** Wrap your notification logic in a method annotated with @Async. This prevents the user from waiting for an email to be sent before getting an API response.
- **Triggers:** Send notifications when:
  - A new event relevant to the student is created.
  - A reminder a day before the event.
  - Confirmation of event registration.

## Database Structure (MySQL)

```
-- Users and Roles

CREATE TABLE roles (

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(20) UNIQUE NOT NULL -- 'STUDENT', 'FACULTY'
```

```
);

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL, -- Hashed password
    role_id INT,
    qr_code_data VARCHAR(255) UNIQUE, -- A unique string for generating the QR code (e.g., a UUID)
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (role_id) REFERENCES roles(id)
);

-- Event Management
CREATE TABLE events (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(150) NOT NULL,
    description TEXT,
    event_date DATETIME NOT NULL,
    location VARCHAR(255),
    created_by INT, -- FK to the faculty user who created it
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (created_by) REFERENCES users(id)
);
```

```
-- Linking table for students registering for events (Many-to-Many)
CREATE TABLE event_registrations (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    event_id INT,
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (event_id) REFERENCES events(id),
    UNIQUE KEY (user_id, event_id) -- A student can only register once per event
);

-- Attendance Tracking
CREATE TABLE attendance (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    event_id INT,
    check_in_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    marked_by INT, -- FK to the faculty user who scanned the QR
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (event_id) REFERENCES events(id),
    FOREIGN KEY (marked_by) REFERENCES users(id),
    UNIQUE KEY (user_id, event_id) -- A student's attendance can only be marked once
);
```

```
-- Feedback Module
CREATE TABLE feedback (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    event_id INT,
    rating INT CHECK (rating >= 1 AND rating <= 5),
    comment TEXT,
    submitted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (event_id) REFERENCES events(id)
);
```