Implementation Document

for

TrackIT

Version 1.0

Group 5 **Group Name: Chill guys**

Aditya Gautam	220064	agautam22@iitk.ac.in
Aayush Singh	220024	aayushs22@iitk.ac.in
Dhruv Rai	220365	dhruvrai22@iitk.ac.in
Ved Prakash Vishwakarma	221180	vedprakash22@iitk.ac.in
Sharique Ahmad	221002	asharique22@iitk.ac.in
Dhruv Varshney	220366	vdhruv22@iitk.ac.in
Mayur Agrawal	220641	mayurag22@iitk.ac.in
Akash Verma	220097	akashv22@iitk.ac.in
Rahul Ahirwar	220856	rahula22@iitk.ac.in
Abhijeet Agarwal	210025	abhijeeta21@iitk.ac.in
Aryan Bansal	200198	aryanb20@iitk.ac.in

Course:	CS253
Mentor TA:	Jeswaanth Gogula
Date:	28 March 2025

Contents........ 2
Revisions........ 3

1 IMPLEMENTATION DETAILS........ 4

2 CODEBASE........ 7

3 COMPLETENESS........ 65

Appendix A - Group Log........ 70

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Aayush Singh Abhijeet Agarwal Aditya Gautam Akash Verma Dhruv Rai Dhruv Varshney Mayur Agrawal Ved Prakash Vishwakarma Rahul Ahirwar Sharique Ahmad Aryan Bansal	Final draft of implementation document	28/03/25

1 Implementation Details

1.1 Overview

TRACKit is a web-based college course information portal, designed to provide a centralized system for managing courses, users, and academic data. It supports three primary user types:

- **Admin**: Has the highest level of permissions. Can manage courses, users, and overall system settings.
- **Faculty**: Can create and manage course announcements, lectures, and handle student data.
- **Student**: Can view their enrolled courses, check announcements, schedules, grades, and more.

1.2 Tools, Languages, and Frameworks

Below is a brief description of the main tools and technologies used. We selected each for reasons of performance, familiarity, and community support:

1. **Frontend: React + Tailwind CSS**

    - **React**:
        - Offers component-based architecture and the virtual DOM for efficient UI rendering.
        - Large ecosystem of libraries (React Router, Redux, etc.).
    - **Tailwind CSS**:
        - Utility-first CSS framework that simplifies styling components with minimal custom CSS.
        - Fast to set up, consistent, and easy to maintain across a larger project.

    Why Chosen: React is well-known, widely adopted, and allows rapid prototyping. Tailwind simplifies the CSS workflow, especially in a multi-developer environment.

2. **Backend: Node.js + Express**

    - **Node.js**:
        - Non-blocking, event-driven I/O allows handling of many concurrent requests.
        - Vast ecosystem of packages in npm.
    - **Express.js**:
        - Minimalist web framework that makes routing and middleware management straightforward.
        - Integrates seamlessly with Node.js.

    Why Chosen: Node.js is a popular choice for REST APIs. Express is lightweight yet powerful.

3. **Database: PostgreSQL/SQLite (via Sequelize ORM)**

    - **Sequelize**:
        - Sequelize is a promise-based Node.js ORM (Object-Relational Mapper) that enables developers to interact with relational databases using object-oriented programming techniques instead of writing raw SQL queries. It supports multiple database systems including PostgreSQL, MySQL and SQLite.
        - Provides an abstraction layer, allowing easy model definition, querying, and migrations in JavaScript.
    - **SQLite and PostgreSQL**:
        - During development, we used a simple SQLite database under the Sequelize abstraction layer, for easy development and testing.
        - Upon reaching the mature stage of the project, for deployment purposes, we simply swapped the underlying SQLite database with a more robust and higher bandwidth PostgreSQL server by changing a few parameters.

    Why Chosen: For a simpler deployment and demonstration, a file-based database is very convenient. In future, the same code can be adapted to PostgreSQL by updating Sequelize configurations. Sequelize thus allowed us to focus on the actual interaction with the data instead of crafting SQL queries meticulously. We chose SQL over mongoDB as SQL more closely aligns with our goals of inheritance of user parameters from a base user class.

4. **Authentication & Security**

    - **JWT (JSON Web Tokens)**:
        - Used for stateless authentication between client and server.
    - **bcryptjs**:
        - For secure hashing and salting of user passwords.
    - **Helmet.js**:
        - Helps secure Express apps by setting various HTTP headers.

5. **Other Notable Libraries**

    - **express-rate-limit**:
        - Limits repeated requests to public APIs to mitigate brute-force attacks.
    - **Dotenv**:
        - Manages environment variables for local development and secure secrets like `JWT_SECRET` and database paths.

1.3 Justification for These Choices

- **Ease of Use**: JavaScript end-to-end (Node.js + React) simplifies knowledge sharing between front-end and back-end teams.
- **Robust Ecosystem**: Node, React, and Tailwind each have large communities.
- **Lightweight Development**: SQLite requires no additional server overhead, while Express + Node handles the REST API needs quickly.

- **High Level of Abstraction:** Sequelize provides an abstraction layer, allowing easy model definition, querying, and migrations in JavaScript. It allowed us to easily move to a more powerful database during deployment.

2 Codebase

2.1 GitHub Repository Link

https://github.com/TRACKit-CS253/TRACKit

2.2 File & Folder Organization

The structure of the codebase is as follows:

```
├── README.md
├── backend
│   ├── README.md
│   ├── .env
│   ├── checkFileUpload.js
│   ├── config
│   │   └── db.config.js
│   ├── controllers
│   │   ├── admin.controller.js
│   │   ├── announcement.controller.js
│   │   ├── auth.controller.js
│   │   ├── course.controller.js
│   │   ├── courseDescriptionEntry.controller.js
│   │   ├── event.controller.js
│   │   ├── faculty.controller.js
│   │   ├── forum.controller.js
│   │   ├── lecture.controller.js
│   │   ├── result.controller.js
│   │   ├── student.controller.js
│   │   └── user.controller.js
│   ├── middleware
│   │   ├── auth.middleware.js
│   │   ├── course.middleware.js
│   │   └── upload.middleware.js
│   ├── models
│   │   ├── admin.model.js
│   │   ├── announcement.model.js
│   │   ├── course.model.js
│   │   ├── courseDescriptionEntry.model.js
│   │   ├── events_calendar.js
│   │   ├── exam.model.js
│   │   ├── faculty.model.js
│   │   ├── forum_post.model.js
│   │   ├── forum_reply.model.js
│   │   ├── heading.model.js
│   │   ├── index.js
│   │   ├── lecture.model.js
│   │   ├── result.model.js
│   │   ├── student.model.js
│   │   ├── subheading.model.js
│   │   └── user.model.js
│   ├── objective.txt
│   ├── package-lock.json
│   ├── package.json
│   ├── routes
│   │   ├── admin.routes.js
│   │   ├── announcement.routes.js
│   │   ├── auth.routes.js
│   │   ├── course.routes.js
│   │   ├── courseDescriptionEntry.routes.js
│   │   ├── event.routes.js
│   │   ├── faculty.routes.js
│   │   ├── forum.routes.js
│   │   ├── lecture.routes.js
│   │   ├── result.routes.js
│   │   ├── student.routes.js
│   │   └── user.routes.js
│   ├── server.js
│   ├── uploads
│   │   ├── Chapter_1_v8.2_8a05c383.pptx
│   │   ├── files-1742830158971-173341411.pdf
│   │   ├── ...
│   └── utils
│       ├── initAdmin.js
│       ├── initState.js
│       └── statistics.js
└── frontend
```

```
├── README.md
├── .env
├── package-lock.json
├── package.json
├── public
│   ├── index.html
│   └── robots.txt
├── src
│   ├── App.css
│   ├── App.jsx
│   ├── assets
│   │   ├── ContactDeveloper.png
│   │   ├── addCourse.png
│   │   ├── faculty.png
│   │   ├── icon-7797704.png
│   │   ├── login.png
│   │   ├── manageCourse.png
│   │   ├── manageUsers.png
│   │   ├── student.png
│   │   ├── textures.jpg
│   │   ├── textures2.jpg
│   │   └── ved.png
│   ├── components
│   │   ├── Calendar_Course.jsx
│   │   ├── Calendar_Course_Home.jsx
│   │   ├── Calendar_Dashboard.jsx
│   │   ├── CourseMenu.jsx
│   │   ├── DashBoardMenu.jsx
│   │   ├── LogoutButton.jsx
│   │   ├── Notification.jsx
│   │   ├── ProtectedRoute.jsx
│   │   └── UserProfile.jsx
│   ├── contexts
│   │   ├── AuthContext.jsx
│   │   ├── CourseContext.jsx
│   │   ├── EventContext.jsx
│   │   └── NotificationContext.jsx
│   ├── index.css
│   ├── index.js
│   ├── pages
│   │   ├── Admin
│   │   │   ├── AddFaculty.jsx
│   │   │   ├── AddStudent.jsx
│   │   │   ├── Admin.jsx
│   │   │   ├── ContactDevelopers.jsx
│   │   │   ├── ManageCourses.jsx
│   │   │   ├── ManageUsers.jsx
│   │   │   └── createCourse.jsx
│   │   ├── ChangePassword.jsx
│   │   ├── Course
│   │   │   ├── Announcements.jsx
│   │   │   ├── Calendar.jsx
│   │   │   ├── CourseHome.jsx
│   │   │   ├── Courses.jsx
│   │   │   ├── Forum.jsx
│   │   │   ├── Lecture_Data.js
│   │   │   ├── Lectures.jsx
│   │   │   ├── Results.jsx
│   │   │   ├── data.js
│   │   │   └── data2.js
│   │   ├── Dashboard
│   │   │   ├── ContactUs.jsx
│   │   │   ├── Course.jsx
│   │   │   ├── Dashboard.jsx
│   │   │   ├── Performance.jsx
│   │   │   └── Profile.jsx
│   │   ├── ForgotPassword.jsx
│   │   └── Login.jsx
│   ├── services
│   │   └── auth.js
│   └── utils
│       ├── axiosConfig.js
│       ├── axiosInstance.js
│       └── fileUpload.js
└── tailwind.config.js
```

2.3 Backend API calls

The following is a summary of all the API endpoints in TrackIt, including their URLs, HTTP methods, expected request data, and response behaviour.

All API endpoints are prefixed by **/api** as shown. Most endpoints require a valid JWT token (provided via the Authorization header as Bearer <token>).

Admin-only endpoints additionally require the user's role to be admin. Course-specific endpoints often require the user to be associated with that course (either as a student or faculty) to succeed.

Typical response codes are:

- **200 OK** for successful GET/PUT/DELETE.
- **201 Created** for successful creation (POST),
- **400 Bad Request** for missing or invalid data,
- **401 Unauthorized** for missing/invalid token,
- **403 Forbidden** for insufficient permissions,
- **404 Not Found** for nonexistent resources, and
- **500 Internal Server Error** for unexpected failures.

2.3.1. Authentication

a. Login

- **URL**: `/auth/login`
- **Method**: `POST`
- **Access**: Public
- **Request Body**:

```
{
  "username": "string",
  "password": "string"
}
```

- **Response** (if successful):

```
{
  "success": true,
  "message": "Login successful",

  "token": "JWT token string",

  "user": {

    "id": "number",

    "username": "string",

    "email": "string",

    "firstName": "string",

    "lastName": "string",

    "userType": "student | faculty | admin",


    // Additional fields if applicable:

    // For student:

    "rollNumber": "string",

    "enrollmentYear": "number",

    "major": "string",


    // For faculty:

    "department": "string",

    "position": "string"

  }

}
```

- **Status**:
    - `200_OK` – Login successful

- `401_Unauthorized` – Invalid password
- `404_Not Found` – User not found
- `500_Internal Server Error` – On failure

2.3.2 Admin

a. Add Student

- **URL**: `/admin/student`
- **Method**: `POST`
- **Data**:

```
{
  "username": "string",
  "email": "string",
  "password": "string",
  "firstName": "string",
  "lastName": "string (optional)",
  "rollNumber": "string",
  "enrollmentYear": "integer (2000-2099)",
  "major": "string"
}
```

- **Status**:
  - If successful: `201_Created`
  - Else:
    - `400_Bad Request` (Missing fields / Duplicate data)
    - `500_Internal Server Error`

---

b. Add Faculty

- **URL**: `/admin/faculty`

- **Method**: `POST`
- **Data**:

```
{
  "username": "string",
  "email": "string",
  "password": "string",
  "firstName": "string",
  "lastName": "string (optional)",
  "department": "string",
  "position": "string"
}
```

- **Status**:
  - If successful: `201_Created`
  - Else:
    - `400_Bad Request`
    - `500_Internal Server Error`

---

c. Bulk Add Students

- **URL**: `/admin/bulk-students`
- **Method**: `POST`
- **Content-Type**: `multipart/form-data`
- **File**: CSV with columns: `username, email, password, firstName, lastName, rollNumber, enrollmentYear, major`
- **Status**:
  - If successful: `201_Created`

- Else:
    - `400_Bad Request` (Missing/invalid fields, duplicates)
    - `500_Internal Server Error`

d. Bulk Add Faculty

- **URL**: `/admin/bulk-faculty`
- **Method**: `POST`
- **Content-Type**: `multipart/form-data`
- **File**: CSV with columns: `username, email, password, firstName, lastName, department, position`
- **Status**:
    - If successful: `201_Created`
    - Else:
        - `400_Bad Request`
        - `500_Internal Server Error`

---

e. Get All Users

- **URL**: `/admin/users`
- **Method**: `GET`
- **Status**:
    - If successful: `200_OK`
    - Else: `500_Internal Server Error`

f. Update User

- **URL**: `/admin/user/:userId`
- **Method**: `PUT`
- **Data**: JSON fields to update
- **Status**:
  - If successful: `200_OK`
  - Else:
    - `404_Not Found`
    - `500_Internal Server Error`

g. Delete User

- **URL**: `/admin/user/:userId`
- **Method**: `DELETE`
- **Status**:
  - If successful: `200_OK`
  - Else:
    - `404_Not Found`
    - `500_Internal Server Error`

2.3.3. Announcements

a. Get All Announcements for a Course

- **URL**: `/announcement/course/:courseId`

- **Method**: GET
- **Access**: Authenticated users associated with the course
- **Response**:

```
{
  "success": true,
  "data": [
    {
      "id": "number",
      "courseId": "number",
      "facultyId": "number",
      "announcementHeading": "string",
      "announcementBody": "string",
      "createdAt": "timestamp",
      "Faculty": {
        "userId": "number",
        "department": "string",
        "position": "string",
        "User": {
          "firstName": "string",
          "lastName": "string",
          "email": "string",
          "username": "string"
        }
      }
```

```
        }
      ]
    }
```

- **Status**:
  - `200_OK` – if retrieval successful
  - `500_Internal Server Error` – on failure

---

b. Create a New Announcement

- **URL**: `/announcement`
- **Method**: `POST`
- **Access**: `Authenticated faculty in the course`
- **Data**:

```
{
  "courseId": "number",
  "announcementHeading": "string",
  "announcementBody": "string"
}
```

- **Status**:
  - `201_Created` – on successful creation
  - `500_Internal Server Error` – on failure

---

c. Update an Announcement

- **URL**: `/announcement/:courseId/:id`
- **Method**: `PUT`
- **Access**: `Authenticated faculty in the course`
- **Headers**:
  `Authorization: Bearer <token>`
- **Data**:

```
{
  "announcementHeading": "string (optional)",
  "announcementBody": "string (optional)"
}
```

- **Status**:
  - `200_OK` – on successful update
  - `404_Not Found` – if announcement does not exist
  - `500_Internal Server Error` – on failure

---

d. Delete an Announcement

- **URL**: `/announcement/:courseId/:id`
- **Method**: `DELETE`
- **Access**: `Authenticated faculty in the course`
- **Headers**:
  `Authorization: Bearer <token>`

- **Status**:
    - `200_OK` – on successful deletion
    - `404_Not Found` – if announcement does not exist
    - `500_Internal Server Error` – on failure

2.3.4. Course Management

a. Get All Courses

- **URL**: `/course`
- **Method**: `GET`
- **Access**: Public
- **Response**:

```
{

  "success": true,
  "data": [ ...courses ]

}
```

- **Status**:
    - `200_OK` – On success
    - `500_Internal Server Error` – On failure

---

b. Get Course by ID

- **URL**: `/course/:id`
- **Method**: `GET`
- **Access**: Public
- **Response**:

```
{
  "success": true,
  "data": {
    "id": "number",
    "code": "string",
    "name": "string",
    "description": "string",
    "credits": "number",
    "semester": "string",
    "faculty": [ ...users ],
    "students": [ ...users ]
  }
}
```

- **Status**:
  - `200_OK` – On success
  - `404_Not Found` – Course not found
  - `500_Internal Server Error` – On failure

---

c. Create a Course

- **URL**: `/course`
- **Method**: `POST`
- **Access**: Admin only
- **Request Body**:

```
{
  "code": "string",
  "name": "string",
  "description": "string",
  "credits": "number",
  "semester": "string"
}
```

- **Status**:
  - `201_Created` – Course created
  - `500_Internal Server Error` – On failure

---

d. Update Course

- **URL**: `/course/:id`
- **Method**: `PUT`
- **Access**: Admin only
- **Request Body**: Fields to update
- **Status**:
  - `200_OK` – Course updated
  - `404_Not Found` – Course not found
  - `500_Internal Server Error` – On failure

---

e. Delete Course

- **URL**: `/course/:id`
- **Method**: `DELETE`
- **Access**: Admin only
- **Headers**:
  `Authorization: Bearer <token>`
- **Status**:
  - `200_OK` – Course deleted
  - `404_Not Found` – Course not found
  - `500_Internal Server Error` – On failure

2.3.4 Course Enrollment Management (Admin only)

f. Add Faculty to Course

- **URL**: `/course/add-faculty`
- **Method**: `POST`
- **Access**: Admin only
- **Headers**:
  `Authorization: Bearer <token>`
- **Request Body**:

```
{
  "courseId": "number",
  "userId": "number"
}
```

- **Status**:

- `200_OK` – Faculty added
- `404_Not Found` – Course or Faculty not found
- `500_Internal Server Error` – On failure

---

g. Remove Faculty from Course

- **URL**: `/course/remove-faculty/:courseId/:userId`
- **Method**: `DELETE`
- **Access**: Admin only
- **Headers**:
  `Authorization: Bearer <token>`
- **Status**:
  - `200_OK` – Faculty removed
  - `404_Not Found` – Course or Faculty not found
  - `500_Internal Server Error` – On failure

---

h. Add Student to Course

- **URL**: `/course/add-student`
- **Method**: `POST`
- **Access**: Admin only
- **Headers**:
  `Authorization: Bearer <token>`
- **Request Body**:

```
{
  "courseId": "number",
  "userId": "number"
}
```

- **Status**:
  - `200_OK` – Student added
  - `404_Not Found` – Course or Student not found
  - `500_Internal Server Error` – On failure

---

i. Remove Student from Course

- **URL**: `/course/remove-student/:courseId/:userId`
- **Method**: `DELETE`
- **Access**: Admin only
- **Headers**:
  `Authorization: Bearer <token>`
- **Status**:
  - `200_OK` – Student removed
  - `404_Not Found` – Course or Student not found
  - `500_Internal Server Error` – On failure

2.3.5. Course Description Entries

a. Get All Course Description Entries for a Course

- **URL**: `/course-description-entry/course/:courseId`
- **Method**: `GET`
- **Access**: Authenticated users associated with the course
- **Response**:

```
{
  "success": true,
  "data": [
    {
      "id": "number",
      "courseId": "number",
      "facultyId": "number",
      "courseDescriptionEntryHeading": "string",
      "courseDescriptionEntryBody": "string",
      "Faculty": {
        "userId": "number",
        "department": "string",
        "position": "string",
        "User": {
          "firstName": "string",
          "lastName": "string",
          "email": "string",
          "username": "string"
        }
      }
```

```
        }
      ]
    }
```

- **Status**:
    - `200_OK` – On success
    - `500_Internal Server Error` – On failure

---

b. Create a New Course Description Entry

- **URL**: `/course-description-entry`
- **Method**: `POST`
- **Access**: Authenticated faculty in the course
- **Request Body**:

```
{
  "courseId": "number",
  "courseDescriptionEntryHeading": "string",
  "courseDescriptionEntryBody": "string"
}
```

- **Status**:
    - `201_Created` – Entry created successfully
    - `500_Internal Server Error` – On failure

---

c. Update a Course Description Entry

- **URL**: `/course-description-entry/:courseId/:id`

- **Method**: `PUT`
- **Access**: Authenticated faculty in the course
- **Headers**:
  `Authorization: Bearer <token>`
- **Request Body**:

```
{
  "courseDescriptionEntryHeading": "string (optional)",
  "courseDescriptionEntryBody": "string (optional)"
}
```

- **Status**:
  - `200_OK` – Entry updated successfully
  - `404_Not Found` – Entry not found
  - `500_Internal Server Error` – On failure

---

d. Delete a Course Description Entry

- **URL**: `/course-description-entry/:courseId/:id`
- **Method**: `DELETE`
- **Access**: Authenticated faculty in the course
- **Status**:
  - `200_OK` – Entry deleted successfully
  - `404_Not Found` – Entry not found
  - `500_Internal Server Error` – On failure

Event Object Structure

Each event (as stored or returned by the API) contains the following fields:

```
{

  "id": "number",
  "title": "string",
  "description": "string",
  "start": "ISO 8601 timestamp",
  "end": "ISO 8601 timestamp",
  "courseId": "number",
  "createdBy": "number"
}
```

Required Fields When Creating an Event

```
{

  "title": "string",
  "description": "string",
  "start": "ISO timestamp",
  "end": "ISO timestamp"
}
```

Note: `courseId` is sent in the **URL parameter**, not in the body (`/event/course/:courseId`)

2.3.6. Event Management

a. Get All Events for a Course

- **URL**: `/event/course/:courseId`
- **Method**: GET
- **Access**: Public
- **Description**: Returns all events associated with a specific course.
- **Response**:

```
{
    "success": true,
    "data": [ ...events ]

}
```

- **Status**:
  - `200_OK` – On success
  - `500_Internal Server Error` – On failure

---

b. Get Events for a User

- **URL**: `/event/user/:userId`
- **Method**: `GET`
- **Access**: Authenticated (token required)
- **Query Params (optional)**:
  `start`: Start date (ISO)
  `end`: End date (ISO)
- **Description**: Returns all events from courses that the user is enrolled in.
- **Response**:

```
{

  "success": true,
  "data": [ ...events ]

}
```

- **Status**:
  - `200_OK` – On success
  - `404_Not Found` – User not found
  - `500_Internal Server Error` – On failure

---

c. Create a New Event

- **URL**: `/event/course/:courseId`
- **Method**: POST
- **Access**: Authenticated (token required)
- **Request Body**:

```
{

  "title": "string",
  "description": "string",
  "start": "ISO timestamp",
  "end": "ISO timestamp"

}
```

- **Response**:

```
{

  "success": true,
 "message": "Event created successfully",
 "data": { ...event }

}
```

- **Status**:
  - `201_Created` – Event created
  - `400_Bad Request` – End time is before start time
  - `500_Internal Server Error` – On failure

---

d. Update an Event

- **URL**: `/event/:eventId`

- **Method**: PUT
- **Access**: Authenticated (token required)
- **Request Body** (all fields optional):

```
{
  "title": "string",
  "description": "string",
  "start": "ISO timestamp",
  "end": "ISO timestamp"
}
```

- **Response**:

```
{
  "success": true,
  "message": "Event updated successfully",
  "data": { ...updated event }
}
```

- **Status**:
  - `200_OK` – Event updated
  - `400_Bad Request` – End time before start time
  - `404_Not Found` – Event not found
  - `500_Internal Server Error` – On failure

---

e. Delete an Event

- **URL**: `/event/:eventId`
- **Method**: DELETE
- **Access**: Authenticated (token required)
- **Response**:

```
{
  "success": true,
  "message": "Event deleted successfully"

}
```

- **Status**:
  - `200_OK` – Event deleted
  - `404_Not Found` – Event not found
  - `500_Internal Server Error` – On failure

2.3.7 Faculty Profile

a. Get Faculty Profile

- **URL**: `/faculty/:id/profile`
- **Method**: GET
- **Access**: Authenticated
- **Authorization Logic**:
  - Faculty can access **their own profile**
  - Admins can access **any faculty profile**
- **Response**:

```
{

  "success": true,

  "data": {

    "userId": "number",

    "username": "string",

    "firstName": "string",

    "lastName": "string",

    "email": "string",
    "userType": "faculty",

    "department": "string",

    "position": "string"

  }

}
```

- **Status**:
  - `200_OK` – On success
  - `404_Not Found` – Faculty not found
  - `500_Internal Server Error` – On failure

---

b. Update Faculty Profile *(currently commented out)*

Planned Endpoint (currently inactive in the router)

- **URL**: `/faculty/:id/profile`
- **Method**: `PUT`
- **Access**: Authenticated
- **Authorization Logic**:
  - Faculty can update **their own profile**
  - Admins can update **any faculty profile**
- **Request Body**:

```
{

  "department": "string (optional)",

  "position": "string (optional)"

}
```

- **Response**:

```
{
  "success": true,
  "message": "Faculty profile updated successfully"
}
```

- **Status**:
  - `200_OK` – On successful update
  - `404_Not Found` – Faculty not found
  - `500_Internal Server Error` – On failure

2.3.8 Forum System

a. Get All Forum Posts for a Course

- **URL**: `/forum/course/:courseId`
- **Method**: `GET`
- **Access**: Authenticated users enrolled in the course
- **Description**: Returns all forum posts for a course, including replies and user details.
- **Response**:

```
{
  "success": true,
  "data": [
    {
      "id": "number",
      "query": "string",
      "courseId": "number",
      "userId": "number",
      "createdAt": "timestamp",
      "User": {
        "id": "number",
        "username": "string",
        "firstName": "string",
        "lastName": "string",
        "userType": "string"
      },
      "replies": [
        {
          "id": "number",
          "content": "string",
          "userId": "number",
          "createdAt": "timestamp",
          "User": { ... }
        }
      ]
    }
  ]
}
```

- **Status**:

- `200_OK` – Posts retrieved successfully
- `500_Internal Server Error` – On failure

---

b. Create a Forum Post

- **URL**: `/forum/course/:courseId`
- **Method**: `POST`
- **Access**: Authenticated users enrolled in the course
- **Request Body**:

```
{
  "query": "string"
}
```

- **Response**:

```
{
  "success": true,
  "message": "Post created successfully",
  "data": {
    "id": "number",
    "query": "string",
    "courseId": "number",
    "userId": "number",
    "createdAt": "timestamp",
    "User": { ... }
  }
}
```

- **Status**:
  - `201_Created` – Post created
  - `400_Bad Request` – Missing query content
  - `404_Not Found` – Course not found
  - `500_Internal Server Error` – On failure

---

c. Delete a Forum Post

- **URL**: `/forum/post/:postId`
- **Method**: `DELETE`
- **Access**: Post owner, faculty, or admin
- **Description**: Deletes a forum post and all its replies.
- **Status**:
  - `200_OK` – Post deleted successfully
  - `403_Forbidden` – Unauthorized access
  - `404_Not Found` – Post not found
  - `500_Internal Server Error` – On failure

---

d. Create a Reply to a Post

- **URL**: `/forum/post/:postId/reply`
- **Method**: `POST`
- **Access**: Authenticated users
- **Request Body**:

```
{
  "content": "string"
}
```

- **Response**:

```
{
  "success": true,
  "message": "Reply created successfully",
  "data": {
    "id": "number",
    "content": "string",
    "postId": "number",
    "userId": "number",
    "createdAt": "timestamp",
    "User": { ... }
  }
}
```

- **Status**:
  - `201_Created` – Reply created
  - `400_Bad Request` – Missing content
  - `404_Not Found` – Post not found
  - `500_Internal Server Error` – On failure

e. Delete a Reply

- **URL**: `/forum/reply/:replyId`
- **Method**: `DELETE`
- **Access**: Reply owner, faculty, or admin
- **Status**:
  - `200_OK` – Reply deleted
  - `403_Forbidden` – Unauthorized access
  - `404_Not Found` – Reply not found
  - `500_Internal Server Error` – On failure

2.3.9 Lecture Management

a. Get All Lectures for a Course (Grouped by Headings and Subheadings)

- **URL**: `/lecture/course/:courseId`
- **Method**: `GET`
- **Access**: Authenticated users enrolled in the course
- **Response**:

```
{
  "success": true,
  "data": [
    {
      "heading": "string",
      "subheadings": [
        {
          "subheading": "string",
          "lectures": [
            {
              "id": "number",
              "lectureTitle": "string",
              "lectureDescription": "string",
              "youtubeLink": "string",
              "fileUrls": [
                {
                  "url": "string",
                  "name": "string",
                  "type": ".pdf" // or other extensions
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

- **Status**:
  - `200_OK` – On success
  - `500_Internal Server Error` – On failure

b. Create a New Lecture

- **URL**: `/lecture`
- **Method**: `POST`
- **Access**: Faculty only
- **Upload**: Up to 10 files (PDFs or other types) via multipart/form-data
- **Request Body**:

```
{
  "courseId": "number",
  "heading": "string",
  "subheading": "string",
  "lectureTitle": "string",
  "lectureDescription": "string",
  "youtubeLink": "string"
}
```

- **Status**:
    - `201_Created` – On success
    - `404_Not Found` – Heading/Subheading not found
    - `500_Internal Server Error` – On failure

c. Create a Heading (Optionally with a Subheading)

- **URL**: `/lecture/heading`
- **Method**: `POST`

- **Access**: Faculty only
- **Request Body**:

```
{
  "courseId": "number",
  "heading": "string",
  "subheading": "string (optional)"
}
```

- **Status**:
    - `201_Created` – Heading (and subheading) created
    - `400_Bad Request` – Missing fields
    - `500_Internal Server Error` – On failure

---

d. Create a Subheading

- **URL**: `/lecture/subheading`
- **Method**: `POST`
- **Access**: Faculty only
- **Request Body**:

```
{
  "courseId": "number",
  "heading": "string",
  "subheading": "string"
}
```

- **Status**:
    - `201_Created` – On success
    - `404_Not Found` – Heading not found
    - `500_Internal Server Error` – On failure

---

e. Edit a Subheading

- **URL**: `/lecture/:courseId/subheading`
- **Method**: `PUT`
- **Access**: Faculty only
- **Request Body**:

```
{
  "heading": "string",
  "currentSubheading": "string",
  "newSubheading": "string"
}
```

- **Status**:
    - `200_OK` – Subheading updated
    - `404_Not Found` – Heading or subheading not found
    - `500_Internal Server Error` – On failure

---

f. Delete a Subheading (with its Lectures & Files)

- **URL**: `/lecture/:courseId/subheading`

- **Method**: `DELETE`
- **Access**: Faculty only
- **Request Body**:

```
{
  "heading": "string",
  "subheading": "string"
}
```

- **Status**:
  - `200_OK` – Deleted successfully
  - `404_Not Found` – Heading or subheading not found
  - `500_Internal Server Error` – On failure

---

g. Update a Lecture

- **URL**: `/lecture/:courseId/:id`
- **Method**: `PUT`
- **Access**: Faculty only
- **Upload**: Up to 10 additional files
- **Request Body**:

```
{
  "lectureTitle": "string (optional)",
  "lectureDescription": "string (optional)",
  "youtubeLink": "string (optional)",
  "heading": "string (optional)",
  "subheading": "string (optional)"
}
```

- **Status**:
  - `200_OK` – On success
  - `404_Not Found` – Lecture or headings not found
  - `500_Internal Server Error` – On failure

---

h. Delete a Lecture

- **URL**: `/lecture/:courseId/:id`
- **Method**: `DELETE`
- **Access**: Faculty only
- **Status**:
  - `200_OK` – Deleted successfully
  - `404_Not Found` – Lecture not found
  - `500_Internal Server Error` – On failure

2.3.10 Results Management

a. Get Results for a Student in a Course

- **URL**: `/result/student/:userId/course/:courseId`
- **Method**: `GET`
- **Access**: Student (self)
- **Description**: Retrieves all exam results for a student enrolled in a particular course.

- **Response**:

```
[
  {
    "examName": string,
    "weightage": number,
    "totalMarks": number,
    "mean": number,
    "median": number,
    "max": number,
    "deviation": number,
    "obtainedMarks": number | null
  }
]
```

Status:

```
{
  If successful: {
    200_OK
  }
  Else: {
    404_Not Found (Course not found),
    500_Internal Server Error
  }
}
```

b. Get Exam Names and IDs for a Course

URL: `/result/course/:courseId/exams`

Method: `GET`

Access: Faculty of that course

Description: Retrieves the list of exams created in a course.

Response:

```
[
  {
    "id": number,
    "examName": string
  }
]
```

Status:

```
{
  If successful: {
    200_OK
  }
  Else: {
    404_Not Found (Course not found),
    500_Internal Server Error
  }
}
```

c. Get Exam Details and Student Results

URL: `/result/course/:courseId/exam/:examId`

Method: `GET`

Access: Faculty of that course

Description: Returns statistical details of an exam and student-wise performance in that exam.

Response:

```
{
  "examName": string,
  "totalMarks": number,
  "weightage": number,
  "mean": number,
  "median": number,
  "max": number,
  "deviation": number,
  "results": [
    {
      "userId": number,
      "rollNumber": string,
      "name": string,
      "obtainedMarks": number | null
    }
  ]
}
```

Status:

```
{
  If successful: {
    200_OK
  }
  Else: {
    404_Not Found (Exam or Course not found),
    500_Internal Server Error
  }
}
```

---

d. Get Students Enrolled in a Course

URL: `/result/course/:courseId/students`

Method: `GET`

Access: Faculty of that course

Description: Returns basic student information (name, roll number) for all students enrolled in a given course.

Response:

```
[
  {
    "userId": number,
    "rollNumber": string,
    "user": {
      "firstName": string,
      "lastName": string
```

```
            }
          }
        ]
```

Status:

```
      {
        If successful: {
          200_OK
        }
        Else: {
          404_Not Found (Course not found),
          500_Internal Server Error
        }
      }
```

e. Publish a New Exam and Results

URL: `/result/course/:courseId/publish`

Method: `POST`

Access: Faculty of that course

Description: Publishes a new exam along with student-wise scores.

Data:

```
     {
       "examName": string,
       "weightage": number,
       "totalMarks": number,
    "results": [
     {
       "userId": number,
       "obtainedMarks": number
     }
    ]
  }
```

Response:

```
     {
       "message": string
     }
```

Status:

```
   {
     If successful: {
       201_Created
     }
     Else: {
       404_Not Found (Course or user not found),
       500_Internal Server Error
     }
   }
```

f. Modify an Existing Exam and Results

URL: `/result/exam/:examId/modify`

Method: `PUT`

Access: Faculty of that course

Description: Updates an existing exam and modifies associated student marks.

Data:

```
{
  "examName": string,
  "weightage": number,
  "totalMarks": number,
  "results": [
    {
      "userId": number,
      "obtainedMarks": number
    }
  ]
}
```

Response:

```
{
  "message": string
}
```

Status:

```
{
    If successful: {
      200_OK
    }
    Else: {
      404_Not Found (Exam not found),
      500_Internal Server Error
    }
  }
```

g. Delete an Exam and Its Results

URL: `/result/exam/:examId/delete`

Method: `DELETE`

Access: Faculty of that course

Description: Permanently deletes an exam and all associated student scores.

Response:

```
    {
      "message": string
    }
```

Status:

```
  {
    If successful: {
      200_OK
    }
    Else: {
      404_Not Found (Exam not found),
      500_Internal Server Error
    }
  }
```

---

h. Get All Exams with Full Statistics for a Course

URL: `/result/course/:courseId/exams/details`

Method: `GET`

Access: Faculty of that course

Description: Returns all exams with statistical summaries in a course.

Response:

```
[
  {
    "id": number,
    "examName": string,
    "totalMarks": number,
    "weightage": number,
    "mean": number,
    "median": number,
    "max": number,
    "deviation": number
  }
]
```

Status:

```
{
  If successful: {
    200_OK
  }
  Else: {
    404_Not Found (Course not found),
    500_Internal Server Error
  }
}
```

2.3.11 Student Management

a. Get Student Profile

URL: `/student/:id/profile`

Method: `GET`

Access: Student (self) or Admin

Description: Retrieves the profile of a student including user and academic information.

Response:

```
{
  "success": true,
  "data": {
    "id": number,
    "username": string,
    "email": string,
    "firstName": string,
    "lastName": string,
    "userType": string,
    "rollNumber": string,
    "enrollmentYear": number,
    "major": string
  }
}
```

Status:

```
    {
      If successful: {
        200_OK
      }
      Else: {
        404_Not Found (Student or User not found),
        500_Internal Server Error
      }
    }
```

---

b. Update Student Profile

URL: `/student/:id/profile`

Method: *PUT*

Access: Student (self) or Admin

Description: Updates student profile fields such as roll number, enrollment year, and major.

Data:

```
{
  "rollNumber": string,
  "enrollmentYear": number,
  "major": string
}
```

Response:

```
{
  "success": true,
  "message": "Student profile updated successfully"
}
```

Status:

```
{
  If successful: {
    200_OK
  }
  Else: {
    404_Not Found (Student not found),
    500_Internal Server Error
  }
}
```

2.3.12 User Management

a. Get User Profile

URL: `/users/:id`

Method: `GET`

Access: User (self) or Admin

Description: Retrieves user profile information along with type-specific details.

Response:

```
{
  "success": true,
  "data": {
    "id": number,
    "username": string,
    "email": string,
    "firstName": string,
    "lastName": string,
    "userType": string,
    "specificInfo": {
      // fields vary based on userType
    }
  }
}
```

Status:

```
        {
          If successful: {
            200_OK
          }
          Else: {
            403_Forbidden (Access denied),
            404_Not Found (User not found),
            500_Internal Server Error
          }
}
```

b. Get User Courses

URL: `/users/:id/courses`

Method: `GET`

Access: User (self) or Admin

Description: Retrieves list of courses that the user (student or faculty) is enrolled in.

Response:

```
          {
            "success": true,
            "data": [
              {
                "id": number,
                "code": string,
                "name": string,
                "description": string,
                "credits": number,
                "semester": string
              }
            ]
          }
```

Status:

```
         {
           If successful: {
             200_OK
           }
           Else: {
             403_Forbidden (Access denied),
             404_Not Found (User not found),
             500_Internal Server Error
           }
         }
```

---

c. Update User Profile

URL: `/users/:id`

Method: `PUT`

Access: User (self only)

Description: Allows a user to update their profile fields (excluding userType).

Data:

```
       {
```

```
    "username": string,

    "email": string,

    "firstName": string,

    "lastName": string,

    "password": string // optional

  }
```

Response:

```
    {

      "success": true,

      "message": "Profile updated successfully"

    }
```

Status:

```
    {

      If successful: {

        200_OK

      }

      Else: {

        403_Forbidden (Access denied),

        404_Not Found (User not found),

        500_Internal Server Error

      }

    }
```

d. Change Password

URL: `/users/:id/password`

Method: `PUT`

Access: User (self only)

Description: Allows a user to change their password by providing the old and new password.

Data:

```
{
  "oldPassword": string,
  "newPassword": string
}
```

Response:

```
{
  "success": true,
  "message": "Password changed successfully"
}
```

Status:

```
{
  If successful: {
    200_OK
  }
  Else: {
    400_Bad Request (Missing password fields),
      401_Unauthorized (Incorrect current password),

      403_Forbidden (Access denied),

      404_Not Found (User not found),

      500_Internal Server Error

    }

  }
```

3 Completeness

3.1 Implemented Features & Verification

1. **User Authentication & Role-Based Access Control**
    - **Implemented**:
        - Three distinct user roles: *Admin*, *Faculty*, and *Student*.
        - Secure login with JWT (JSON Web Tokens).
        - Sign-up restricted to Admins only (to prevent unauthorized user creation).
        - Password hashing and salting via *bcryptjs*, and additional security headers through *helmet*.
    - **Correctness Verification**:
        - Verified by attempting logins with valid credentials (expected *200 OK*) and invalid credentials (expected *401 Unauthorized*).
        - Checked token-based access for protected routes (e.g., Admin, Faculty endpoints) to ensure only users with the correct role can access.
2. **Course Management**
    - **Implemented**:
        - CRUD (Create/Read/Update/Delete) operations for courses, restricted to *Admin*.
        - Faculty assignment to courses.
        - Students automatically gain read-only permissions to their enrolled courses.
        - Public endpoint (`GET /course`) for listing all available courses.
    - **Correctness Verification**:
        - Created sample courses and verified that only the Admin token could create, edit, or delete courses.
        - Ensured course retrieval is publicly accessible (status *200 OK*) and that non-existent course IDs return *404 Not Found*.
3. **Announcements & Forum**

- **Implemented**:
    - **Announcements**: Faculty can post, edit, or delete announcements under a specific course. Students and Faculty can view them.
    - **Forum**: Students or Faculty can create forum posts and replies, restricted to course members. Deletion of posts or replies respects ownership or role (Faculty/Admin).
- **Correctness Verification**:
    - Validated creation and retrieval of announcements with correct user tokens.
    - Checked unauthorized attempts (e.g., student attempting to delete an announcement) return *403 Forbidden*.
    - Created, replied to, and deleted forum posts in a test environment to confirm correct role checks and error handling.

4. **Lecture Management**
    - **Implemented**:
        - Hierarchical lecture storage (Heading → Subheading → Lectures).
        - Faculty can upload lecture materials (files, YouTube links), then students can view them based on their enrollment.
        - Faculties can edit, move, or delete lectures, headings, and subheadings.
    - **Correctness Verification**:
        - Endpoints tested with valid faculty credentials for creation/deletion.
        - Attempted retrieval from a student account to confirm read-only access.
        - Verified that missing headings/subheadings return *404 Not Found* where appropriate.
5. **Event Management**
    - **Implemented**:
        - Ability to create, update, and delete events associated with a course (e.g., assignment deadlines, lecture schedules).
        - Students can view all events for their enrolled courses, and faculty can manage them.
    - **Correctness Verification**:

        - Created events via the faculty role and verified that students enrolled in that course can see them (*200 OK*).
        - Checked date-validation logic for start/end times.
        - Invalid or non-existent event IDs produce *404 Not Found*, confirming correct error handling.

6. **Results Management**
    - **Implemented**:
        - Admin or Faculty can publish exam results, define exam name, total marks, weightage, and per-student marks.
        - Statistical data (mean, median, max, deviation) automatically computed.
        - Students can view only their own results in a course; faculty can see aggregated and per-student results.
    - **Correctness Verification**:
        - Tested with sample data to confirm accurate statistics (mean, median, etc.).
        - Checked that only faculty can publish or modify exam results, while students can see only their personal scores.
7. **User & Student Management**
    - **Implemented**:
        - **Admin** can create, bulk-import, or delete users.
        - **Students/Faculty** can view and update their own profiles (with certain fields restricted).
        - **Role-based** restrictions ensure that Students cannot escalate their privileges.
    - **Correctness Verification**:
        - Attempted to update or delete a user profile from various roles to confirm *403 Forbidden* is correctly triggered for unauthorized actions.
        - Confirmed proper hashing of passwords during user creation and changes.
8. **Security Measures**
    - **Implemented**:

        - *bcryptjs* for password hashing.
        - *helmet* for setting secure HTTP headers.
        - *express-rate-limit* to mitigate brute-force login attempts.
        - JWT expiration and verification on all protected routes.
    - **Correctness Verification**:
        - Tested repeated invalid login attempts and confirmed rate-limiting logic.
        - Examined API responses for correct status codes on token expiration, invalid tokens, or missing tokens.

Overall, each major functionality was tested manually through tools like **Postman** (or an equivalent API testing framework) to confirm that request/response flows match the expected outcomes described in the API documentation. Error conditions (invalid data, insufficient permissions, missing fields) have also been verified to return the correct status codes (*400 Bad Request*, *401 Unauthorized*, *403 Forbidden*, *404 Not Found*, *500 Internal Server Error*, etc.).

3.2 Future Development Plans

While exam **results** have been addressed (with endpoints for publishing and modifying exam scores), the next step is to integrate a **comprehensive exam schedule** module that:

1. **Centralizes All Exam Dates**
    a. Faculty can post upcoming exam dates and times for each course.
    b. Students see a unified view of all their upcoming exams across different courses.
2. **Calendar Sync**
    a. Potentially provide a way for students to integrate exam dates into a personal or institutional calendar (ICS export or direct sync).
3. **Reminders & Notifications**
    a. Automated reminders for approaching exams (e.g., push notifications or email alerts).
    b. Additional messaging or alerts within the forum/announcement system.
4. **Clash Detection (Long-Term Roadmap)**
    a. Identify overlapping exams / events for different courses.

b. Prompt Admin/Faculty to adjust schedules or room allocations.

5. **Adding forgot password feature**

    a. Every user will have this option on the login page.

    b. We currently have this feature in the frontend, and partially implemented in the backend. We can implement this easily by integrating a proper email service.

    c. OTP(One Time Password) will be used to identify the authorised user.

6. **Attendance tracking**

    a. This will serve as the one stop solution for both students and faculty to monitor the attendance.

    b. This requires integration with the current database of the biometric office.

Appendix A - Group Log

The frontend part was first written, followed by the backend part. In each stage, tasks were divided among team members,and pair programming was used to carry out tasks.Thus,all team members are familiar with all stages of the development process as well as all aspects of the software.

The following are the group logs of team meetings and their minutes. These meetings primarily serve as review sessions for ongoing work. In addition to these discussions, each team member has contributed significant individual effort, and everyone's contributions are duly recognized and appreciated :

Date	Timings	Duration	Minutes
Feb 9 , 2025	14:00 - 16:00	2 hrs.	● Discussed the components to be present and distributed the division of the components ● Collectively decided upon the frameworks to be used to develop the frontend and backend parts
Feb 15, 2025	10:00 - 11:00	1 hrs.	● Started the work on frontend , and divided the frontend work among the team members
March 07, 2025	20:00-23:30	3.5 hrs.	● Reviewed the work on frontend, and decided upon the further changes to be made ● Started on the backend development, decided upon the work distribution of backend development
March 16, 2025	14:00 - 16:00	2 hrs.	● Reviewed the work of backend , discussed about its improvisation ● Discussed and distributed the work of connecting the backend with frontend
March 18, 2025	21:00 - 02:00	5 hrs.	● Reviewed the total work of the website and total structure ● Asked and discussed upon additional features and current feature improvisations that can be made ● Distributed improvisation work
March 25 , 2025	20:00- 23:00	3 hrs.	● Started upon bug testing and resolving work ● Shifted the hosting of site from localhost to provided server ● Started working on implementation document

March 28, 2025	15:00 - 17:00	2 hrs.	● Cross-checked and finalised the implementation doc for the submission