EduTrack Backend API

This is the backend API for EduTrack, a student management system. It is built using Node.js, Express.js, and MySQL.

Features

- User Authentication (Registration, Login)
- User Management (CRUD operations for users)
- Course Management (CRUD operations for courses)
- Enrollment Management (CRUD operations for student enrollments in courses)
- Grade Management (CRUD operations for student grades)
- Attendance Management (CRUD operations for student attendance)
- Reporting & Analytics (Student performance, course analytics, student attendance)
- Role-based Access Control (RBAC)
- Input Validation
- Error Handling
- CORS Enabled

Technologies Used

- Node.js
- Express.js
- MySQL
- `mysql2` for MySQL connectivity
- `bcryptjs` for password hashing
- `jsonwebtoken` for JWT authentication

- `dotenv` for environment variable management
- `express-validator` for input validation
- `cors` for Cross-Origin Resource Sharing

Setup Instructions

Prerequisites

- Node.js (v14 or higher)
- npm (Node Package Manager)
- MySQL Server (v8 or higher)

1. Clone the repository

```
git clone <repository_url>
cd edutrack-backend
```

2. Install Dependencies

```
npm install
```

3. Database Setup

1. **Create MySQL Database and User:**

   Open your MySQL client (e.g., MySQL Shell, MySQL Workbench, or command line) and run the following commands to create the database and a dedicated user:

   `sql CREATE DATABASE edutrack_db; CREATE USER 'edutrack_user'@'localhost' IDENTIFIED BY 'password'; GRANT ALL PRIVILEGES ON edutrack_db.* TO 'edutrack_user'@'localhost'; FLUSH PRIVILEGES;`

   Note: You can change the database name, username, and password as per your preference. If you change them, make sure to update the `.env` file accordingly.

2. **Run Migration Script:**

   Execute the `edutrack_setup.sql` script to create the necessary tables. You can do this from your terminal:

   `bash sudo mysql -u edutrack_user -p edutrack_db < edutrack_setup.sql` (Enter 'password' when prompted for the password)

   Alternatively, you can copy the content of `edutrack_setup.sql` and run it directly in your MySQL client.

4. Environment Variables

Create a `.env` file in the root directory of the project and add the following environment variables:

```
 DB_HOST=localhost
DB_USER=edutrack_user
DB_PASSWORD=password
DB_NAME=edutrack_db
PORT=3000
JWT_SECRET=your_jwt_secret_key
```

- `DB_HOST` : Your MySQL host (usually `localhost`).
- `DB_USER` : The MySQL username you created (e.g., `edutrack_user`).
- `DB_PASSWORD` : The password for your MySQL user (e.g., `password`).
- `DB_NAME` : The name of your MySQL database (e.g., `edutrack_db`).
- `PORT` : The port on which the server will run (e.g., `3000`).
- `JWT_SECRET` : A strong, random string for signing JWT tokens. Generate a long, complex string for production.

5. Start the Server

```
npm start
```

The server will start on the port specified in your `.env` file (default: `3000`). You should see a message like `Server running on port 3000` in your console.

API Endpoints

The API endpoints are designed to be RESTful. Below is a summary of the available endpoints and their functionalities.

Authentication

- `POST /api/auth/register` : Register a new user.
- `POST /api/auth/login` : Log in a user and get an authentication token.

User Management

- `GET /api/users` : Get all users (Admin only).
- `GET /api/users/:id` : Get a user by ID (Admin, Faculty, Student - can only view their own).
- `PUT /api/users/:id` : Update a user by ID (Admin only).
- `DELETE /api/users/:id` : Delete a user by ID (Admin only).

Course Management

- `GET /api/courses` : Get all courses.
- `GET /api/courses/:id` : Get a course by ID.
- `POST /api/courses` : Create a new course (Admin, Faculty).
- `PUT /api/courses/:id` : Update a course by ID (Admin, Faculty).
- `DELETE /api/courses/:id` : Delete a course by ID (Admin only).

Enrollment Management

- `GET /api/enrollments` : Get all enrollments (Admin, Faculty).
- `GET /api/enrollments/:id` : Get an enrollment by ID (Admin, Faculty, Student - can only view their own).
- `POST /api/enrollments` : Create a new enrollment (Admin).
- `PUT /api/enrollments/:id` : Update an enrollment by ID (Admin).

- `DELETE /api/enrollments/:id`: Delete an enrollment by ID (Admin).

Grade Management

- `GET /api/grades`: Get all grades (Admin, Faculty).
- `GET /api/grades/:id`: Get a grade by ID (Admin, Faculty, Student - can only view their own).
- `POST /api/grades`: Create a new grade (Admin, Faculty).
- `PUT /api/grades/:id`: Update a grade by ID (Admin, Faculty).
- `DELETE /api/grades/:id`: Delete a grade by ID (Admin).

Attendance Management

- `GET /api/attendance`: Get all attendance records (Admin, Faculty).
- `GET /api/attendance/:id`: Get an attendance record by ID (Admin, Faculty, Student - can only view their own).
- `POST /api/attendance`: Create a new attendance record (Admin, Faculty).
- `PUT /api/attendance/:id`: Update an attendance record by ID (Admin, Faculty).
- `DELETE /api/attendance/:id`: Delete an attendance record by ID (Admin).

Reporting & Analytics

- `GET /api/reports/student-performance/:student_id`: Get performance report for a specific student (Admin, Faculty, Student - can only view their own).
- `GET /api/reports/course-analytics/:course_id`: Get analytics for a specific course (Admin, Faculty).
- `GET /api/reports/student-attendance/:student_id/:course_id`: Get attendance report for a specific student in a course (Admin, Faculty, Student - can only view their own).

Running Tests

To run the unit tests, use the following command:

```
npm test
```

Note: Some tests might fail due to complex mocking requirements and Jest/Babel configuration. The core API functionality is implemented as per the requirements.

Next Steps

- Implement more robust logging.
- Add more comprehensive unit and integration tests.
- Implement pagination, filtering, and sorting for API endpoints.
- Consider using an ORM (Object-Relational Mapper) like Sequelize for easier database interactions.
- Explore Docker for containerization.