Application Frameworks - Assignment 01

The University Timetable Management System API is a RESTful API designed to facilitate efficient management of class schedules for students, faculty, and administrative staff within a university setting. This project aims to provide secure access, ensure data integrity, and offer user-friendly interfaces for interacting with the timetable system.

Setup Instructions

1. **Installation**:
    - Clone the repository: **git clone <repository-url>**
    - Install dependencies: **npm install**
2. **Environment Variables**:
    - Locate **.env** file in the root directory.
    - Define the required environment variables (e.g., database connection, JWT secret).
3. **Running the Application**:
    - Start the server: **node app.js**
    - The server will be running at **http://localhost:<port>**

API Endpoints

1. Courses

Base URL : http://localhost:3000/api/courses

1. Retrieve all courses

- **URL: /**
- **Method: GET**
- **Authentication required:** No
- **Description:** Retrieves a list of all courses.
- **Response:**
    - **Status:** 200 OK
    - **Body:** Array of course objects.

2. Retrieve a single course

- **URL: /:id**
- **Method: GET**
- **Authentication required:** No
- **Description:** Retrieves details of a single course by its ID.
- **Request Parameters:**
    - **id**: ID of the course to retrieve (in the URL path)
- **Response:**
    - **Status:** 200 OK
    - **Body:** Course object.
    - **Status:** 404 Not Found
    - **Body: { "error": "Course not found" }**

3. Create a new course

- **URL: /**
- **Method: POST**
- **Authentication required:** Yes (admin)

- **Description:** Creates a new course.
- **Request Body:**
    - **C_name**: Name of the course (string)
    - **C_code**: Course code (string)
    - **C_description**: Description of the course (string)
    - **C_credits**: Number of credits for the course (integer)
    - **C_faculty**: Faculty teaching the course (string)
- **Response:**
    - **Status:** 201 Created
    - **Body:** Created course object.
    - **Status:** 400 Bad Request
    - **Body: { "error": "Error message" }**

4. Delete a course

- **URL: /:id**
- **Method: DELETE**
- **Authentication required:** Yes (admin)
- **Description:** Deletes a course by its ID.
- **Request Parameters:**
    - **id**: ID of the course to delete (in the URL path)
- **Response:**
    - **Status:** 200 OK
    - **Body: { "message": "Course deleted successfully" }**
    - **Status:** 404 Not Found
    - **Body: { "error": "Course not found" }**

5. Update a course

- **URL: /:id**
- **Method: PATCH**
- **Authentication required:** Yes (admin)
- **Description:** Updates a course by its ID.

- **Request Parameters:**
    - **id**: ID of the course to update (in the URL path)
- **Request Body:** Fields to update in the course object.
- **Response:**
    - **Status:** 200 OK
    - **Body:** Updated course object.
    - **Status:** 404 Not Found
    - **Body: { "error": "Course not found" }**

6. Update a course's faculty

- **URL: /:id/update-faculty**
- **Method: PATCH**
- **Authentication required:** Yes (admin)
- **Description:** Updates the faculty teaching a course by its ID.
- **Request Parameters:**
    - **id**: ID of the course to update (in the URL path)
- **Request Body:**
    - **C_faculty**: New faculty name (string)
- **Response:**
    - **Status:** 200 OK
    - **Body:** Updated course object.
    - **Status:** 404 Not Found
    - **Body: { "error": "Course not found" }**

2. Enrollment

Base URL : http://localhost:3000/api/enrollments

1. Retrieve all enrollments

- **URL**: **/**

- **Method**: GET
- **Authentication Required**: Yes (admin or faculty)
- **Description:** Retrieves a list of all enrollments.
- **Response**:
    - **Status**: 200 OK
    - **Body**: Array of enrollment objects.

2. Retrieve enrollments by student ID

- **URL**: **/student/:studentId**
- **Method**: GET
- **Authentication Required**: Yes (admin or faculty)
- **Description**: Retrieves enrollments of a student by their ID.
- **Request Parameters**:
    - **studentId**: ID of the student (in the URL path)
- **Response**:
    - **Status**: 200 OK
    - **Body**: Array of enrollment objects.

3. Create a new enrollment

- **URL**: **/**
- **Method**: POST
- **Authentication Required**: Yes
- **Description**: Creates a new enrollment.
- **Request Body**:
    - **S_Id**: ID of the student (string)
    - **C_code**: Course code (string)
- **Response**:
    - **Status**: 201 Created
    - **Body**: Created enrollment object.
    - **Status**: 400 Bad Request
    - **Body**: { "error": "Error message" }

4. Update an enrollment by ID

- **URL**: **/:id**
- **Method**: PUT
- **Authentication Required**: Yes (admin or faculty)
- **Description**: Updates an enrollment by its ID.
- **Request Parameters**:
    - **id**: ID of the enrollment to update (in the URL path)
- **Request Body**:
    - Fields to update in the enrollment object.
- **Response**:
    - **Status**: 200 OK
    - **Body**: Updated enrollment object.
    - **Status**: 404 Not Found
    - **Body**: { "error": "Enrollment not found" }

5. Delete an enrollment by ID

- **URL**: **/:id**
- **Method**: DELETE
- **Authentication Required**: Yes (admin or faculty)
- **Description**: Deletes an enrollment by its ID.
- **Request Parameters**:
    - **id**: ID of the enrollment to delete (in the URL path)
- **Response**:
    - **Status**: 200 OK
    - **Body**: { "message": "Enrollment deleted successfully" }
    - **Status**: 404 Not Found
    - **Body**: { "error": "Enrollment not found" }

3. Timetable

Base URL : http://localhost:3000/api/timetables

1. Retrieve all timetables

- **URL**: **/**
- **Method**: GET
- **Authentication Required**: Yes
- **Description**: Retrieves a list of all timetables.
- **Response**:
    - **Status**: 200 OK
    - **Body**: Array of timetable objects.

2. Retrieve a single timetable by ID

- **URL**: **/:id**
- **Method**: GET
- **Authentication Required**: Yes
- **Description**: Retrieves details of a single timetable by its ID.
- **Request Parameters**:
    - **id**: ID of the timetable to retrieve (in the URL path)
- **Response**:
    - **Status**: 200 OK
    - **Body**: Timetable object.
    - **Status**: 404 Not Found
    - **Body**: { "error": "Timetable not found" }

3. Create a new timetable

- **URL**: **/**
- **Method**: POST
- **Authentication Required**: Yes (admin)
- **Description**: Creates a new timetable.

- **Request Body**:
    - **C_name**: Name of the course (string)
    - **day**: Day of the week (string)
    - **startTime**: Start time of the class (string)
    - **endTime**: End time of the class (string)
    - **faculty**: Faculty teaching the course (string)
    - **location**: Location of the class (string)
- **Response**:
    - **Status**: 201 Created
    - **Body**: Created timetable object.
    - **Status**: 400 Bad Request
    - **Body**: { "error": "Error message" }

4. Update a timetable by ID

- **URL**: **/:id**
- **Method**: PATCH
- **Authentication Required**: Yes (admin)
- **Description**: Updates a timetable by its ID.
- **Request Parameters**:
    - **id**: ID of the timetable to update (in the URL path)
- **Request Body**: Fields to update in the timetable object.
- **Response**:
    - **Status**: 200 OK
    - **Body**: Updated timetable object.
    - **Status**: 404 Not Found
    - **Body**: { "error": "Timetable not found" }

5. Delete a timetable by ID

- **URL**: **/:id**
- **Method**: DELETE
- **Authentication Required**: Yes (admin)

- **Description**: Deletes a timetable by its ID.
- **Request Parameters**:
    - **id**: ID of the timetable to delete (in the URL path)
- **Response**:
    - **Status**: 200 OK
    - **Body**: { "message": "Timetable deleted successfully" }
    - **Status**: 404 Not Found
    - **Body**: { "error": "Timetable not found" }

4. User

Base URL : http://localhost:3000/api/users

1. Register a new user

- **URL**: **/register**
- **Method**: POST
- **Authentication Required**: No
- **Description**: Registers a new user.
- **Request Body**:
    - **UserName**: User's username (string)
    - **password**: User's password (string)
    - **name**: User's name (string)
    - **email**: User's email address (string)
- **Response**:
    - **Status**: 201 Created
    - **Body**: **{ message: 'User registered successfully', user: newUser }**
    - **Status**: 400 Bad Request
    - **Body**: **{ "error": "Error message" }**

2. Login user

- **URL**: **/login**
- **Method**: POST

- **Authentication Required**: No
- **Description**: Authenticates a user.
- **Request Body**:
    - **UserName**: User's username (string)
    - **password**: User's password (string)
- **Response**:
    - **Status**: 200 OK
    - **Body**: **{ message: 'Login successful', token: JWT_token }**
    - **Status**: 401 Unauthorized
    - **Body**: **{ "error": "Invalid User Name" }** or **{ "error": "Invalid Password" }**

3. Get all users

- **URL**: **/**
- **Method**: GET
- **Authentication Required**: Yes (admin)
- **Description**: Retrieves all users.
- **Response**:
    - **Status**: 200 OK
    - **Body**: Array of user objects.

4. Get user by ID

- **URL**: **/:id**
- **Method**: GET
- **Authentication Required**: Yes (admin)
- **Description**: Retrieves a user by their ID.
- **Request Parameters**:
    - **id**: ID of the user to retrieve (in the URL path)
- **Response**:
    - **Status**: 200 OK
    - **Body**: User object.
    - **Status**: 404 Not Found

    - **Body**: **{ "error": "User not found" }**

5. Update user by ID

- **URL**: **/:id**
- **Method**: PUT
- **Authentication Required**: Yes (admin)
- **Description**: Updates a user by their ID.
- **Request Parameters**:
    - **id**: ID of the user to update (in the URL path)
- **Request Body**:
    - **name**: User's name (string)
    - **email**: User's email address (string)
- **Response**:
    - **Status**: 200 OK
    - **Body**: Updated user object.
    - **Status**: 400 Bad Request
    - **Body**: **{ "error": "Cannot update UserName, salt, or accessLevel" }**

6. Delete user by ID

- **URL**: **/:id**
- **Method**: DELETE
- **Authentication Required**: Yes (admin)
- **Description**: Deletes a user by their ID.
- **Request Parameters**:
    - **id**: ID of the user to delete (in the URL path)
- **Response**:
    - **Status**: 200 OK
    - **Body**: **{ "message": "User deleted successfully" }**
    - **Status**: 404 Not Found
    - **Body**: **{ "error": "User not found" }**

7. Change user's access level

- **URL**: **/:id/access-level**
- **Method**: PATCH
- **Authentication Required**: Yes (admin)
- **Description**: Changes a user's access level by their ID.
- **Request Parameters**:
    - **id**: ID of the user to update (in the URL path)
- **Request Body**:
    - **accessLevel**: New access level for the user (string)
- **Response**:
    - **Status**: 200 OK
    - **Body**: Updated user object.
    - **Status**: 404 Not Found
    - **Body**: **{ "error": "User not found" }**

5. Notifications

Base URL: http://localhost:3000/api/notifications

1. Get User Notifications

- **Description**: Retrieves notifications for the authenticated user.
- **Method**: GET
- **URL**: **/user-notifications**
- **Authentication Required**: Yes
- **Response**:
    - **Status**: 200 OK
    - **Body**: Object containing:
        - **unreadNotificationsCount**: Number of unread notifications (integer)
        - **matchingNotifications**: Array of notification objects

2. Update User Notification Read Status

- **Description**: Updates the read status of a user notification.
- **Method**: PATCH

- **URL**: **/user-notifications/:id**
- **Authentication Required**: Yes
- **Request Parameters**:
    - **id**: ID of the notification to update (in the URL path)
- **Request Body**:
    - **read_YN:** New read status (boolean)
- **Response**:
    - **Status**: 200 OK
    - **Body**: **{ "message": "User notification read status updated successfully" }**
    - **Status**: 404 Not Found
    - **Body**: **{ "error": "User notification not found" }**

Test Instructions

This Integration Testing will be done using Jest library.

1. Run following commands in the project root directory.

```
npm install --save-dev jest

npm install --save-dev supertest
```

2. run the tests by using following commands one by one in the project root directory.

```
npx jest Tests/timeTableController.test.js

npx jest Tests/userController.test.js

npx jest Tests/courseController.test.js

npx jest Tests/enrollmentController.test.js
```