1. Introduction

Overview

The Online Learning Platform is a full-stack application designed to facilitate student registration, course management, and enrollment. It provides separate functionalities for students and admin users (institute staff).

Objective

The objective of this project is to demonstrate proficiency in designing and implementing a scalable and efficient system for an Online Learning Platform.

Use Cases

- **Student**
  - Register an account
  - Login and logout
  - View available courses
  - Enroll in courses
- **Admin User**
  - Login and logout
  - Manage courses (CRUD operations)
  - Manage students (CRUD operations)
  - Manage enrollments (CRUD operations)

2. Technologies Used

Backend

- **Framework**: Node.js with Express.js
- **Database**: PostgreSQL
- **Authentication**: JWT Tokens

Frontend

- **Framework**: React.js
- **Styling**: Material-UI

Database

- **Database**: PostgreSQL

Authentication

- Implemented using JWT tokens for secure user authentication and authorization.

3. Folder Structure

```
├── backend/
```

```
│   ├── controllers/
│   ├── models/
│   ├── routes/
│   ├── services/
│   ├── app.js
│   └── ...
├── frontend/
│   ├── components/
│   ├── pages/
│   ├── App.js
│   └── ...
├── docs/
│   ├── API_Documentation.pdf
│   └── Design_Decisions.pdf
├── README.md
└── ...
```

4. Backend Implementation

Authentication

- Implemented JWT token-based authentication.
- Middleware for authentication and authorization.

Course Management

- CRUD operations for courses using Express.js and PostgreSQL.

Student Management

- CRUD operations for students with form validation.

Enrollment Management

- CRUD operations for enrollments ensuring data integrity.

5. Frontend Implementation

User Interface

- Responsive and visually appealing UI with Material-UI components.

User Authentication

- Login, logout, and registration forms integrated with backend APIs.

Course Listing

- Displaying courses fetched from backend with details and enrollment option.

Enrollment

- Ability for students to enroll in courses with interactive feedback.

6. API Documentation

Tools Used

For documenting the RESTful API endpoints of the Online Learning Platform, we have utilized **Swagger**. Swagger provides a clear and interactive interface for exploring and testing API endpoints, making it easier to understand the functionality of each API operation.

Endpoints

The API endpoints are categorized based on their functionality:

Authentication

- **POST /api/auth/register**
  - **Description:** Register a new user (student).
  - **Request Body:**

    ```
    json
    Copy code
    {
      "name": "string",
      "email": "string",
      "password": "string",
      "role": "string" // Default: "student"
    }
    ```

  - **Response:**
    - 200 OK: User registration successful.
    - 400 Bad Request: Validation errors.
    - 500 Internal Server Error: Server error.
- **POST /api/auth/login**
  - **Description:** Login and generate JWT token for authentication.
  - **Request Body:**

    ```
    json
    Copy code
    {
      "email": "string",
      "password": "string"
    }
    ```

  - **Response:**
    - 200 OK: Login successful. Returns token, user role, and userId.
    - 401 Unauthorized: Invalid credentials.
    - 500 Internal Server Error: Server error.
- **POST /api/auth/logout**
  - **Description:** Logout and invalidate JWT token.
  - **Authorization:** Bearer Token
  - **Response:**
    - 200 OK: Logout successful.

        - 500 Internal Server Error: Server error.

Course Management (Admin)

- **GET /api/courses**
    - **Description:** Retrieve all courses.
    - **Authorization:** Bearer Token
    - **Response:**
        - 200 OK: Array of course objects.
        - 401 Unauthorized: Token missing or invalid.
        - 500 Internal Server Error: Server error.
- **POST /api/courses**
    - **Description:** Create a new course.
    - **Authorization:** Bearer Token
    - **Request Body:**

```
json
Copy code
{
  "title": "string",
  "description": "string",
  "image": "string" // URL of course image
}
```

    - **Response:**
        - 201 Created: Course created successfully. Returns the created course object.
        - 400 Bad Request: Validation errors.
        - 401 Unauthorized: Token missing or invalid.
        - 500 Internal Server Error: Server error.
- **PUT /api/courses/{courseId}**
    - **Description:** Update an existing course.
    - **Authorization:** Bearer Token
    - **Request Body:**

```
json
Copy code
{
  "title": "string",
  "description": "string",
  "image": "string" // URL of course image
}
```

    - **Response:**
        - 200 OK: Course updated successfully. Returns the updated course object.
        - 400 Bad Request: Validation errors.
        - 401 Unauthorized: Token missing or invalid.
        - 404 Not Found: Course not found.
        - 500 Internal Server Error: Server error.
- **DELETE /api/courses/{courseId}**
    - **Description:** Delete a course by courseId.
    - **Authorization:** Bearer Token
    - **Response:**
        - 204 No Content: Course deleted successfully.

        - 401 Unauthorized: Token missing or invalid.
        - 404 Not Found: Course not found.
        - 500 Internal Server Error: Server error.

Student Management (Admin)

- **GET /api/students**
    - **Description:** Retrieve all students.
    - **Authorization:** Bearer Token
    - **Response:**
        - 200 OK: Array of student objects.
        - 401 Unauthorized: Token missing or invalid.
        - 500 Internal Server Error: Server error.
- **POST /api/students**
    - **Description:** Create a new student.
    - **Authorization:** Bearer Token
    - **Request Body:**

```
json
Copy code
{
  "name": "string",
  "email": "string",
  "password": "string"
}
```

    - **Response:**
        - 201 Created: Student created successfully. Returns the created student object.
        - 400 Bad Request: Validation errors.
        - 401 Unauthorized: Token missing or invalid.
        - 500 Internal Server Error: Server error.
- **PUT /api/students/{studentId}**
    - **Description:** Update an existing student.
    - **Authorization:** Bearer Token
    - **Request Body:**

```
json
Copy code
{
  "name": "string",
  "email": "string"
}
```

    - **Response:**
        - 200 OK: Student updated successfully. Returns the updated student object.
        - 400 Bad Request: Validation errors.
        - 401 Unauthorized: Token missing or invalid.
        - 404 Not Found: Student not found.
        - 500 Internal Server Error: Server error.
- **DELETE /api/students/{studentId}**
    - **Description:** Delete a student by studentId.
    - **Authorization:** Bearer Token

- **Response:**
  - 204 No Content: Student deleted successfully.
  - 401 Unauthorized: Token missing or invalid.
  - 404 Not Found: Student not found.
  - 500 Internal Server Error: Server error.

Enrollment Management (Admin)

- **GET /api/enrollments**
  - **Description:** Retrieve all enrollments.
  - **Authorization:** Bearer Token
  - **Response:**
    - 200 OK: Array of enrollment objects.
    - 401 Unauthorized: Token missing or invalid.
    - 500 Internal Server Error: Server error.
- **POST /api/enrollments**
  - **Description:** Create a new enrollment.
  - **Authorization:** Bearer Token
  - **Request Body:**

```
json
Copy code
{
  "userId": "integer",
  "courseId": "integer"
}
```

  - **Response:**
    - 201 Created: Enrollment created successfully. Returns the created enrollment object.
    - 400 Bad Request: Validation errors.
    - 401 Unauthorized: Token missing or invalid.
    - 500 Internal Server Error: Server error.
- **DELETE /api/enrollments/{enrollmentId}**
  - **Description:** Delete an enrollment by enrollmentId.
  - **Authorization:** Bearer Token
  - **Response:**
    - 204 No Content: Enrollment deleted successfully.
    - 401 Unauthorized: Token missing or invalid.
    - 404 Not Found: Enrollment not found.
    - 500 Internal Server Error: Server error.

Request/Response Formats

- **Request:** JSON format for sending data to the server.
- **Response:** JSON format for receiving data from the server.

7. Testing and Validation

Backend Tests

- Unit tests for controllers, services, and models using Jest.

Frontend Tests

- Component testing using React Testing Library.

8. Setup Instructions

Prerequisites

- Node.js
- PostgreSQL

Installation Steps

1. Clone the repository.
2. Install dependencies (`npm install`).
3. Set up PostgreSQL database.
4. Configure environment variables.

Running the Application

- Backend: `node index.js` in `backend/` directory.
- Frontend: `npm start` in `frontend/` directory.

9. Additional Considerations

Scalability

- Designed architecture to handle increased traffic and data growth.

Security

- Implemented best practices for authentication and data validation.

Documentation

- Clear and comprehensive documentation of design decisions and setup instructions.

10. Submission Details

Git Repository Link

- https://github.com/KasunWijerathna/Online-Learning-Platform-Interview.git.

Setup Instructions

- Detailed instructions on how to run the application and tests.

Documentation Files

- Include API documentation (Swagger/OpenAPI format) and design decisions.