**Full Stack Development with MERN**

**Project Documentation**

**1. Introduction**

* **Project Title:** Online Learning Platform.
* **Team Members:** 
  1. Thirumurugan S
  2. Naveen G
  3. Sanjay Kumar S
  4. Saranya M
  5. Ruban P

**2. Project Overview**

* **Purpose:** The **Online Learning Platform** aims to provide a comprehensive and interactive learning environment for students and instructors. It enables teachers to create and manage course content, allows students to enroll in and complete courses at their own pace, and offers administrative oversight to monitor and manage user engagement. This platform facilitates seamless online education, allowing students to access both free and paid courses and earn certificates upon completion.
* **Features:**
  + **Teacher Role**:
    - **Add and Manage Courses**: Teachers can create courses and add course sections to provide structured content.
    - **Course Deletion**: Teachers can delete courses if there are no active enrollments or for other specific reasons.
* **Student Role**:
  + **Course Enrollment**: Students can enroll in multiple courses and track their learning progress.
  + **Resume Course Functionality**: Students can pick up courses where they left off, ensuring continuity.
  + **Certificate Download**: Upon completion, students can download a certificate as proof of their achievement.
  + **Course Purchase**: For paid courses, students can make purchases to access exclusive content.
  + **Advanced Search and Filtering**: Students can filter courses by name, category, and other criteria to find relevant content.
* **Admin Role**:
  + **Comprehensive Course Management**: Admins have the ability to modify all courses, ensuring quality and consistency.
  + **User Monitoring and Management**: Admins can oversee and manage both teachers and students on the platform.
  + **Enrollment Tracking**: Admins can keep a record of all student enrollments, enabling better insights into platform engagement.
* **Goals:**
  + **Accessible Learning:** Make quality education available to anyone with internet access.
  + **Structured Course Management:** Provide teachers with tools to easily upload and manage course content**.**
  + **Streamlined User Management**: Enable admins to oversee users, track course enrollments, and maintain the integrity of course offerings.
  + **Progress Tracking:** Allow students to track their progress, resume courses, and earn certifications.
  + **E-commerce Integration:** Support the purchase and access of paid courses.

`

**3. Architecture**

* **Frontend:**

The frontend is built with **React**, which provides a dynamic and responsive user interface for different user roles (Teacher, Student, and Admin). The component-based architecture allows for modularity and reusability, making the interface intuitive and easy to maintain.

* **Component Structure**:
  + **Authentication Components**: Components like Login, Register, and Logout handle user authentication and access control.
  + **Dashboard Components**: Separate dashboards for each user role (Teacher, Student, Admin) display relevant information and provide access to specific features.
  + **Course Components**: Components for viewing, enrolling, and managing courses (CourseList, CourseDetail, EnrollButton).
  + **Reusable UI Elements**: Leveraging libraries like Material UI, Ant Design, and mdb-react-ui-kit, the UI features consistent styles, forms, tables, and modals for actions like adding or editing content.
  + **State Management**: State is managed using React’s Context API for global state (e.g., authentication status, user role) and component-level state for individual components. Axios is used to handle API calls to the backend.
  + **Routing**: React Router handles routing between pages. Private routes are created to restrict access based on user roles.
* **Backend:** The backend uses **Node.js** and **Express.js** to handle API requests and business logic, ensuring security, efficiency, and scalability.
  + **Authentication and Authorization**:
    - **JWT (JSON Web Token)**: Used for secure authentication. Each user receives a token upon login, which is stored in the client and sent with each request for authorization.
    - **Role-based Access Control**: Middleware functions verify each user’s role (Teacher, Student, Admin) to restrict access to specific endpoints.
  + **API Structure**:
    - **RESTful Endpoints**: Organized around core resources such as users, courses, enrollments, etc. Each endpoint follows RESTful principles, handling CRUD operations.
    - **File Handling**: Multer is used for handling media uploads, such as profile pictures or course materials.
* **Database:** The database is built with **MongoDB** and uses **Mongoose** as the Object Data Modeling (ODM) library to define schemas and relationships.
  + **Database Schema**:
    - **User Schema**: Defines fields like username, email, password (hashed using bcrypt), role (Student, Teacher, or Admin), and enrolledCourses (for Students).
    - **Course Schema**: Defines fields for courseName, description, price, sections (array of sections, each containing title, content, and resources), and enrolledStudents.
    - **Certificate Schema**: Captures information on completed courses, with fields such as studentId, courseId, and completionDate.
  + **Database Interactions**:
    - **CRUD Operations**: Each entity (User, Course, Certificate) has CRUD operations implemented in service functions. For example, students enrolling in courses triggers an update in the enrolledCourses field of the Student’s profile and adds the student’s ID to the enrolledStudents array in the Course document.
    - **Query Optimization**: Mongoose indexes are used to improve query performance for common searches (e.g., finding courses by name or category).
  + **Data Security**:
    - **Password Hashing**: Passwords are stored securely using bcrypt hashing.
    - **Data Validation**: Data is validated on both frontend and backend to ensure data integrity and prevent SQL injection or other attacks.

**4. Setup Instructions**

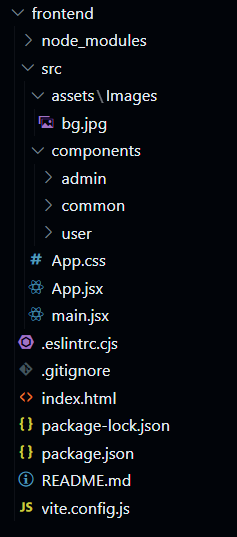
* **Prerequisites:**

Before setting up the project, ensure you have the following software dependencies installed:

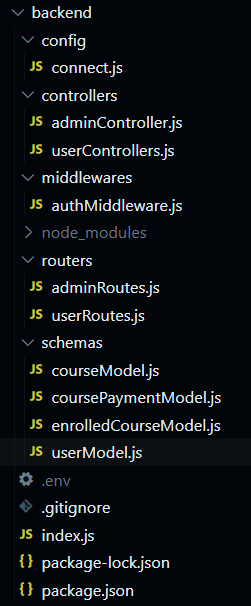
* + Node.js (v14.x or later
  + MongoDB
  + Git (for cloning the repository)
* **Installation:** Follow these steps to set up the project on your local machine:
  + **Clone the Repository:** git clone <repository-url>
  + **Navigate into the project directory:** cd <project-directory>
  + **Install backend Dependencies:** cd backend & npm install
  + **Install frontend dependencies:** cd ../frontend&npm install
* **Set Up Environment Variables**: In the backend directory, create a .env file to store your environment variables. Add the following variables to your .env file:
* DB\_URI=<your-mongodb-connection-string>
* JWT\_SECRET=<your-jwt-secret-key>
* PORT=<port-for-backend-server>
* **Run the Development Servers:**
  + **Start the backend server:** cd backend & npm start
  + **Start the frontend server:** cd ../frontend&npm run dev
* **Access the Application:** Open your browser and go to http://localhost:3000 to view the frontend. The backend server should be running on http://localhost:5000 (or the port specified in your .env file).

**5. Folder Structure**

* **Client:**



* **Server:**



**6. Running the Application**

* **Installation:** Follow these steps to set up the project on your local machine:
  + **Clone the Repository:** git clone <repository-url>
  + **Navigate into the project directory:** cd <project-directory>
  + **Install backend Dependencies:** cd backend & npm install
  + **Install frontend dependencies:** cd ../frontend&npm install
* **Set Up Environment Variables**: In the backend directory, create a .env file to store your environment variables. Add the following variables to your .env file:
* DB\_URI=<your-mongodb-connection-string>
* JWT\_SECRET=<your-jwt-secret-key>
* PORT=<port-for-backend-server>
* **Run the Development Servers:**
  + **Start the backend server:** cd backend & npm start
  + **Start the frontend server:** cd ../frontend&npm run dev
* **Access the Application:** Open your browser and go to http://localhost:3000 to view the frontend. The backend server should be running on http://localhost:5000 (or the port specified in your .env file).

**7. API Documentation**

1. **Register User**
   * **Endpoint**: /api/auth/register
   * **Method**: POST
   * **Body**: { "username": "string", "email": "string", "password": "string", "role": "string" }
   * **Response**: User details on success
2. **Login User**
   * **Endpoint**: /api/auth/login
   * **Method**: POST
   * **Body**: { "email": "string", "password": "string" }
   * **Response**: JWT token on success

**Course Management**

1. **Create Course** (Teacher only)
   * **Endpoint**: /api/courses
   * **Method**: POST
   * **Body**: { "courseName": "string", "description": "string", "price": "number" }
   * **Response**: Course details on success
2. **Get All Courses**
   * **Endpoint**: /api/courses
   * **Method**: GET
   * **Response**: List of courses
3. **Delete Course** (Teacher only)
   * **Endpoint**: /api/courses/:id
   * **Method**: DELETE
   * **Response**: Confirmation message

**Enrollment**

1. **Enroll in Course** (Student only)
   * **Endpoint**: /api/enrollments/:courseId
   * **Method**: POST
   * **Response**: Enrollment confirmation
2. **Get Enrolled Courses**
   * **Endpoint**: /api/enrollments
   * **Method**: GET
   * **Response**: List of enrolled courses

**Certificates**

1. **Download Certificate** (Student only)
   * **Endpoint**: /api/certificates/:courseId
   * **Method**: GET
   * **Response**: Certificate download URL

**Admin**

1. **Get All Users**
   * **Endpoint**: /api/admin/users
   * **Method**: GET
   * **Response**: List of all users
2. **Get Student Enrollments**
   * **Endpoint**: /api/admin/enrollments/:studentId
   * **Method**: GET
   * **Response**: List of student’s enrollments

**8. Authentication:** The project uses **JWT (JSON Web Tokens)** for user authentication and authorization. Each user (Student, Teacher, Admin) is assigned a token upon successful login, which is used to validate their identity and access permissions throughout the platform.

1.**User Registration**

* When a user registers, they provide their details such as username, email, password, and role (either "student," "teacher," or "admin").
* The password is **hashed** using bcryptjs before storing it in the database to ensure secure storage.

2.**Login and Token Generation**

* Upon login, the user provides their email and password.
* The server verifies the credentials and, if valid, generates a **JWT token** using a secret key stored in the environment variable JWT\_SECRET.
* This JWT token contains information like the user ID and role, which is then sent to the client and stored on the client side (e.g., in localStorage or a cookie).

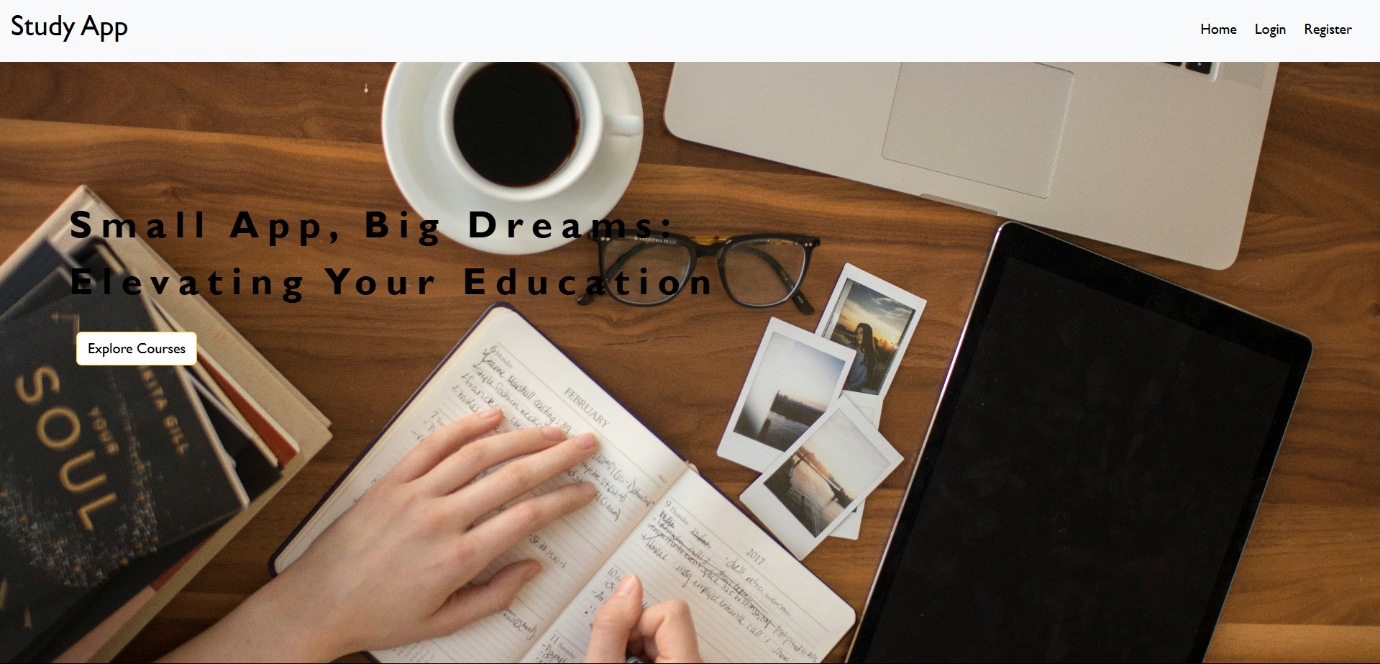
3.**Token-based Authorization**

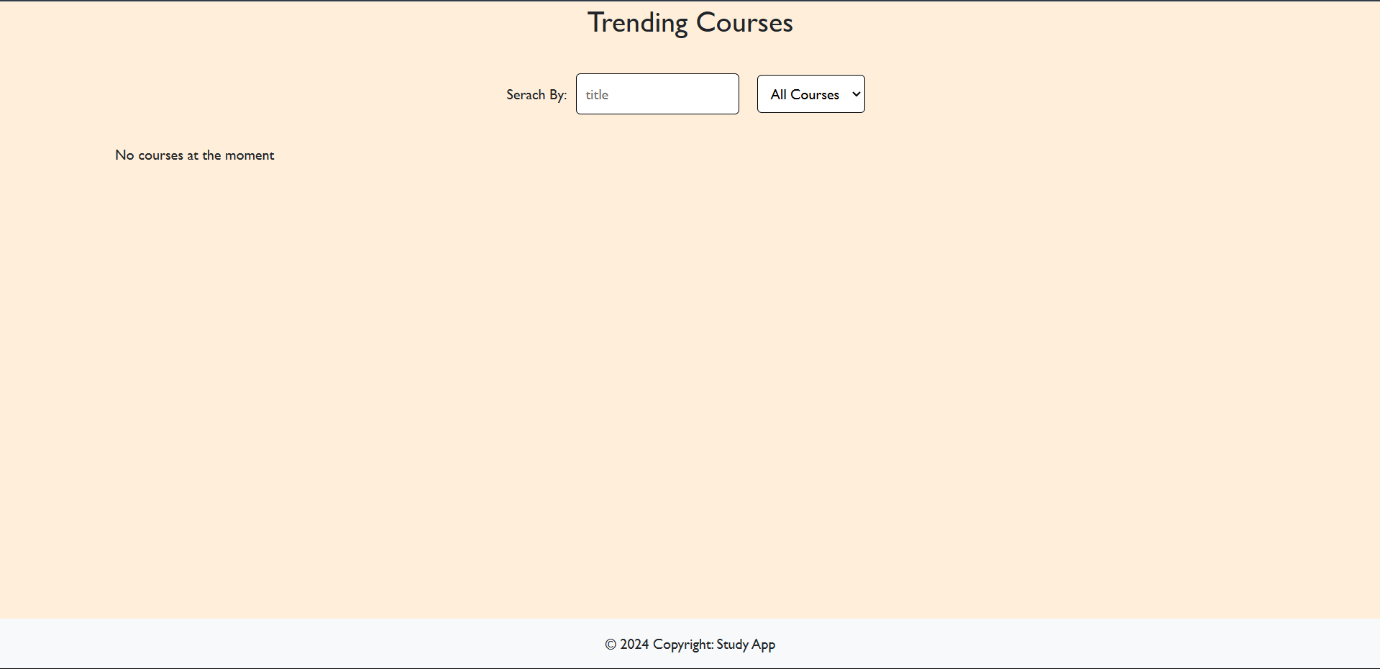
* Each request to protected routes (e.g., creating a course, enrolling in a course, managing users) requires a valid JWT token in the Authorization header as Bearer <token>.
* Middleware on the server side validates the token before granting access to the requested resource.

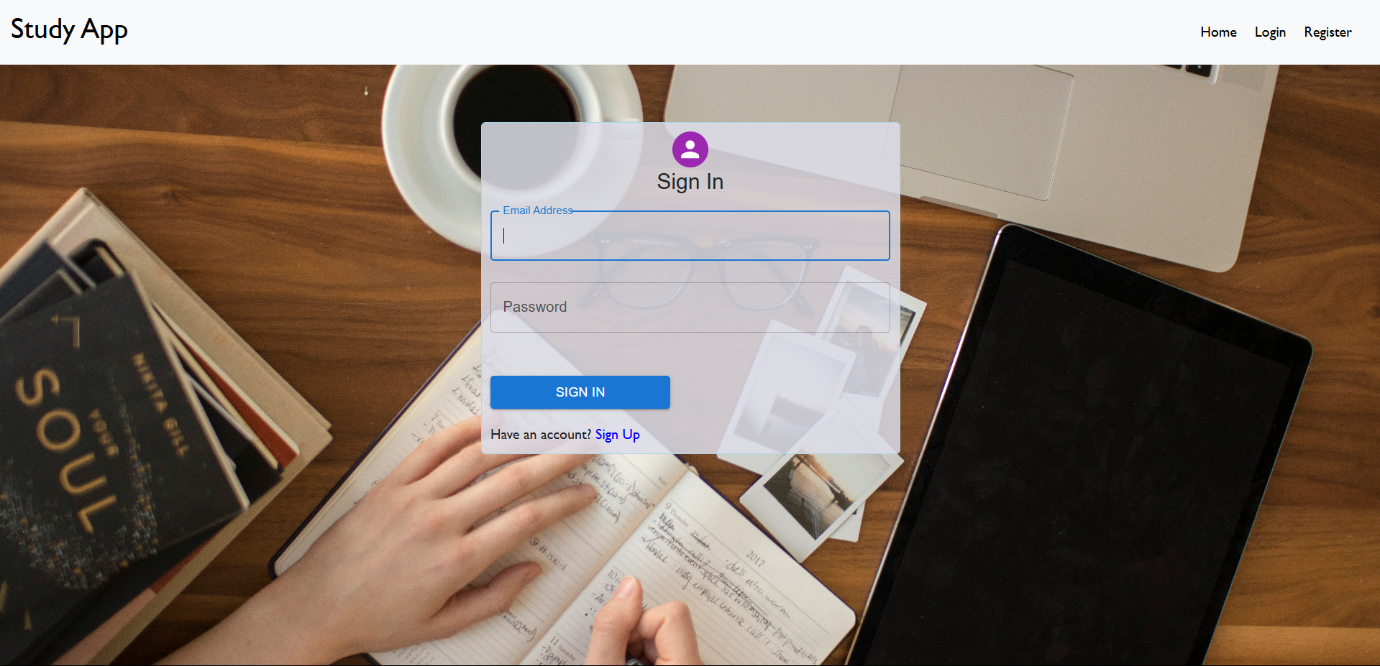
4. **Role-based Access Control**

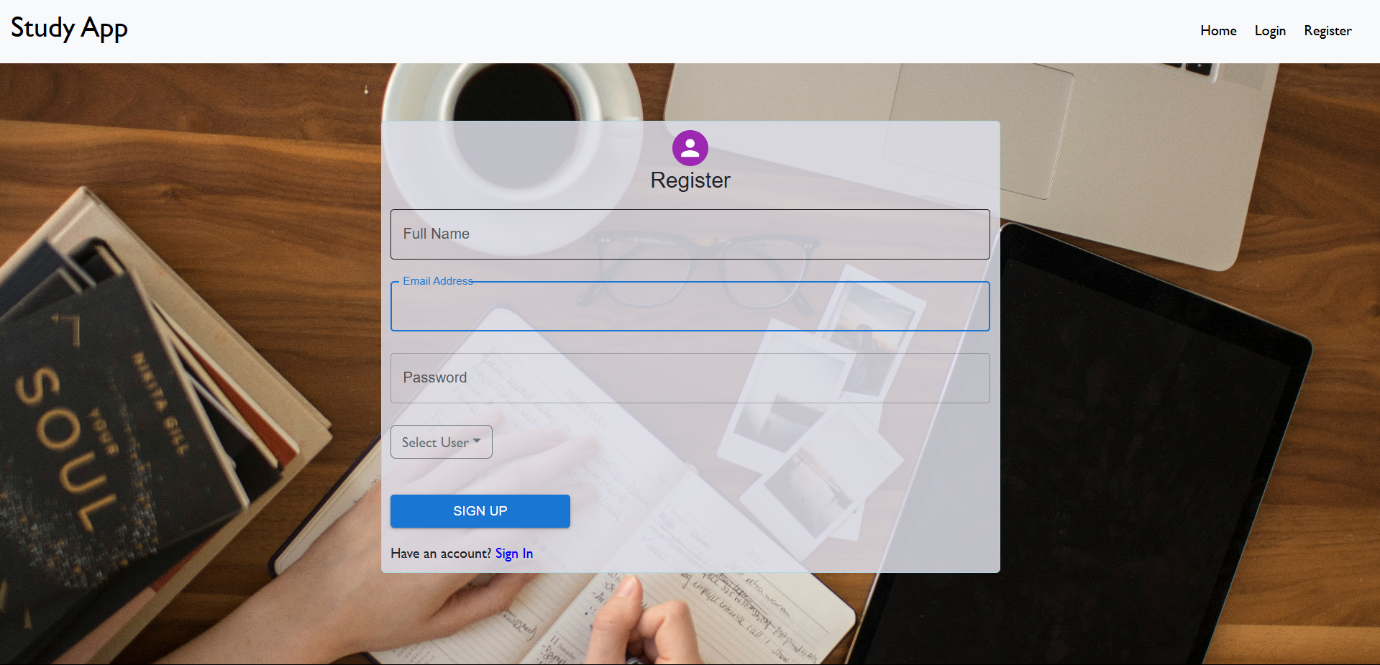
* Each role (Student, Teacher, Admin) has specific permissions. For example:
  + **Students** can access course content, enroll in courses, and download certificates.
  + **Teachers** can create and delete courses.
  + **Admins** can view and manage all users and courses.
* The server verifies the user’s role from the JWT token and checks permissions to ensure only authorized users can perform certain actions.

**9. User Interface**

****

****

****

****

**10. Testing**

**Testing Strategy**

1. **Unit Testing**
   * **Goal**: Test individual components and functions.
   * **Tools**:
     + **Jest** for functions and backend routes.
     + **React Testing Library** for React component interaction.
2. **Integration Testing**
   * **Goal**: Ensure frontend and backend work together smoothly.
   * **Tools**:
     + **Supertest** for backend API requests.
     + **Jest** to check data flow across components.
3. **End-to-End (E2E) Testing**
   * **Goal**: Test complete user interactions, such as login and course management.
   * **Tools**:
     + **Cypress** to simulate full user journeys and interactions.

**Testing Tools**

* **Jest**: Main tool for unit and integration tests.
* **React Testing Library**: Tests React components from a user perspective.
* **Supertest**: Simulates HTTP requests to test backend APIs.
* **Cypress**: Tests the entire application as a user would interact with it.

**Example Tests**

* **Unit**: Calculate course progress function.
* **Integration**: Confirm course enrollment updates backend and frontend.
* **E2E**: Login, enroll in a course, and download a completion certificate.

**Running Tests**

* **Frontend/Backend**: npm test for Jest and Supertest tests.
* **E2E**: npx cypress open to start Cypress for end-to-end testing.

**12. Known Issues**

1. **Cross-Browser Compatibility**

* **Issue**: The website may break or render incorrectly on some Linux and Mac browsers.
* **Cause**: Compatibility issues with certain CSS or JavaScript features across different browsers.
* **Workaround**: Recommend using the latest versions of Chrome or Firefox on Linux and Mac until resolved.

2. **Session Expiry Handling**

* **Issue**: Users may experience unexpected logouts if their session token expires during active use.
* **Cause**: Lack of session refresh functionality.
* **Workaround**: Refresh the page and log back in. Implementing a token refresh mechanism is planned.

3. **Delayed Certificate Generation**

* **Issue**: The certificate download feature may be slow or delayed after course completion.
* **Cause**: Server-side processing time for generating PDF certificates.
* **Workaround**: Wait a few moments and try re-downloading if the delay persists.

4. **Slow Loading on Large Course Content**

* **Issue**: Courses with large video files or numerous sections may load slowly.
* **Cause**: High data load affecting response time.
* **Workaround**: Optimize media content size. Implement lazy loading to improve performance in future updates.

**13. Future Enhancements**

1. **Token Refresh for Improved User Experience**

* **Description**: Implement a token refresh mechanism to prevent unexpected logouts due to session expiry, ensuring continuous access during extended sessions.

2. **Enhanced Analytics Dashboard**

* **Description**: Add an analytics dashboard for teachers and admins to track course performance, student engagement, and completion rates in real-time.

3. **Interactive Video Content**

* **Description**: Introduce interactive elements within video lectures, such as quizzes and clickable annotations, to enhance student engagement and learning outcomes.

4. **Payment Gateway Integration**

* **Description**: Integrate additional secure payment gateways to offer more options for purchasing paid courses, improving accessibility for a wider audience.

5. **Mobile Application Development**

* **Description**: Develop a mobile app for iOS and Android, providing users with a more convenient and optimized experience for learning on mobile devices.

6. **Gamification Features**

* **Description**: Add badges, points, and leaderboards to encourage student engagement and provide a rewarding learning experience.

7. **Multi-Language Support**

* **Description**: Enable content translation and multi-language options to make courses accessible to a global audience.

8. **AI-Powered Course Recommendations**

* **Description**: Use AI to recommend courses to students based on their learning history, improving user experience through personalized suggestions.