**Full Stack Development with MERN**

**1. Introduction**

* **Project Title:** Online Learning Platform
* **Team Members:**
  + Kirthik Venkatram – Full-Stack Developer
  + Sriram Aravindh – Backend Developer
  + Eric John – Frontend Developer

**2. Project Overview**

* **Purpose:**The Online Learning Platform is designed to bridge the gap between teachers and students, providing an efficient and user-friendly way to create, deliver, and **consume educational content online.**
* **Features:**
  + Role-based access: Admin, Teacher, and Student functionalities.
  + Teachers can create courses with sections that include videos or YouTube links.
  + Students can enroll in courses and track progress.
  + Certificates are auto-generated upon course completion.
  + Admin can manage all courses and users.

**3. Architecture**

* **Frontend:**
  + Built with React.js.
  + Component-based architecture for modularity.
  + Styled using React-Bootstrap and Material-UI.
* **Backend:**
  + Built with Node.js and Express.js.
  + Middleware for authentication (JWT).
  + RESTful API structure for seamless integration.
* **Database:**
  + MongoDB for storing user data, courses, sections, and progress tracking.
  + Schema examples:
    - Users: { name, email, role, enrolledCourses }
    - Courses: { title, description, categories, sections }
    - Sections: { title, description, contentLink, isCompleted }

**4. Setup Instructions**

* **Prerequisites:**
  + Node.js (version 20 or above)
  + MongoDB (local or cloud instance, e.g., MongoDB Atlas)
  + Git
* **Installation:**

1. **Clone the repository:**

git clone <https://github.com/KirthikVenkatram/Online-Learning-Platform.git>

1. **Navigate to the project directory:**

cd Online-Learning-Platform

1. **Install dependencies for both frontend and backend:**

cd backend && npm install

cd frontend && npm install

1. **Set up environment variables:**
   * Create a .env file in the server folder.
   * Add the following variables:

PORT=8000

MONGO\_URI=your-mongodb-connection-string

JWT\_SECRET=your-secret-key

**5. Folder Structure**

* **Client:**
  + src/components: Contains React components (e.g., Dashboard, CourseContent).
  + src/pages: Individual pages (e.g., StudentHome, AdminHome).
  + src/common: Shared utilities (e.g., NavBar, AxiosInstance).
* **Server:**
  + controllers: Logic for handling requests (e.g., userControllers.js).
  + routes: API routes (e.g., userRoutes.js).
  + models: Mongoose schemas for MongoDB (e.g., User, Course).

**6. Running the Application**

* **Frontend:** Go to the frontend directory and type npm run dev.
* **Backend:** Go to the backend directory and type npm start.

**7. API Documentation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endpoint** | **Method** | **Description** | **Parameters** | **Example Response** |
| /api/user/register | POST | Register a new user. | { name, email, password } | { success: true } |
| /api/user/login | POST | Log in a user and return a token. | { email, password } | { token, userDetails } |
| /api/user/addcourse | POST | Add a course (Teacher role). | { title, sections } | { success: true } |
| /api/user/coursecontent/:courseId | GET | Fetch course content. | courseId (URL param) | { courseContent } |

**8. Authentication**

1. **Token-based Authentication**:
   * JSON Web Tokens (JWT) are utilized for secure authentication and authorization.
   * When a user logs in, the server generates a JWT token signed with a secret key (JWT\_SECRET).
   * The token encodes essential user details (like ID and role) and has an expiration time to enhance security.
2. **Token Storage**:
   * Tokens are stored securely in the browser's local storage.
   * Every subsequent API request includes the token in the Authorization header for validation.
3. **Validation**:
   * Backend middleware intercepts protected routes and validates the token.
   * If the token is valid, the user's role and ID are extracted from the token payload and added to the request object for further processing.
   * If invalid or expired, the server responds with an unauthorized error.
4. **Role-based Access Control (RBAC)**:
   * Access to different parts of the application is determined by the user's role, which is stored in the JWT token.
     1. **Admin**:
        1. Manage users and courses globally.
        2. View platform-wide data and metrics.
     2. **Teacher**:
        1. Add and manage their own courses.
        2. View and track students enrolled in their courses.
     3. **Student**:
        1. Enroll in available courses.
        2. Track progress in enrolled courses.
        3. Earn certificates upon completion.

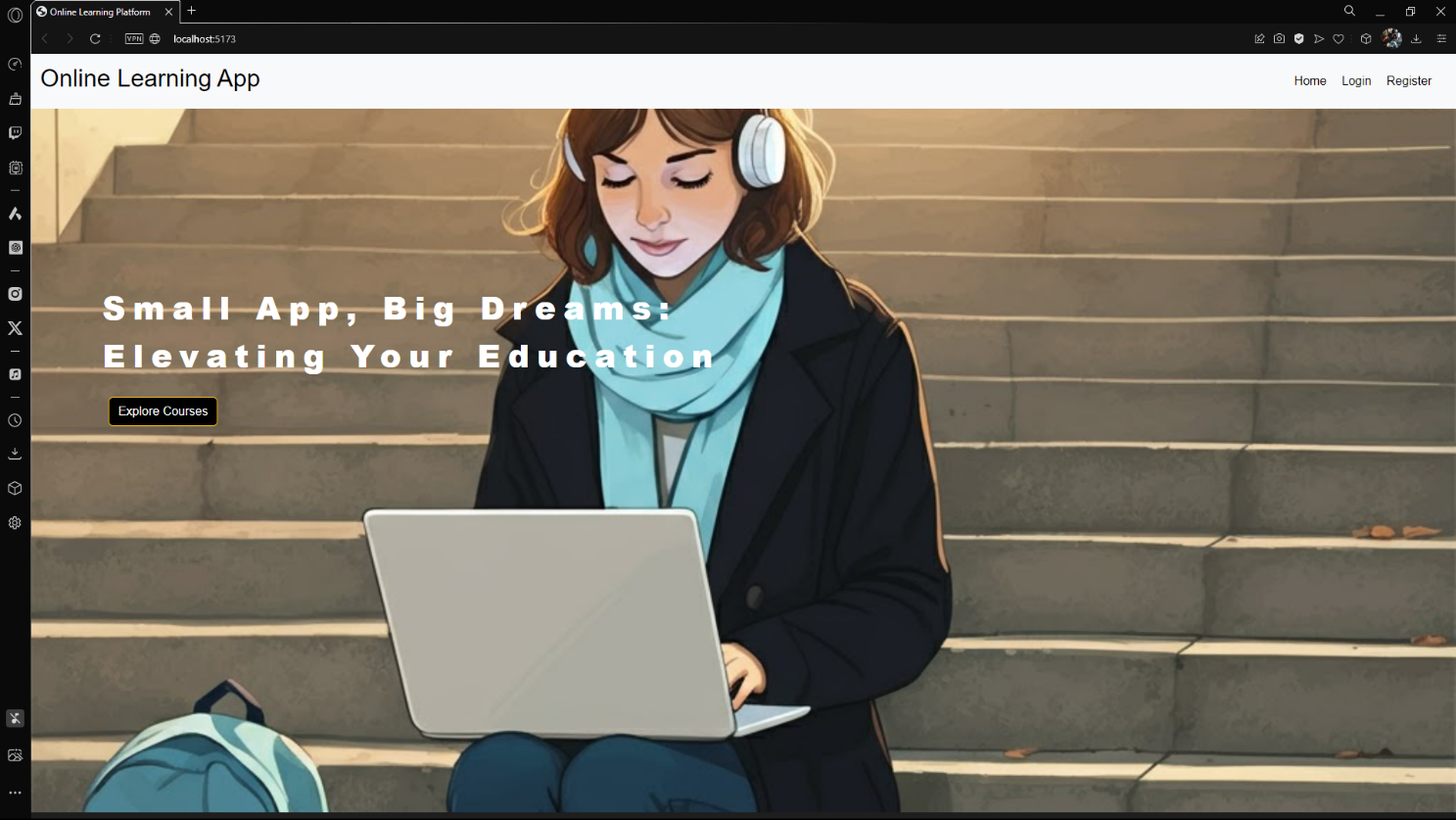
**9. User Interface**

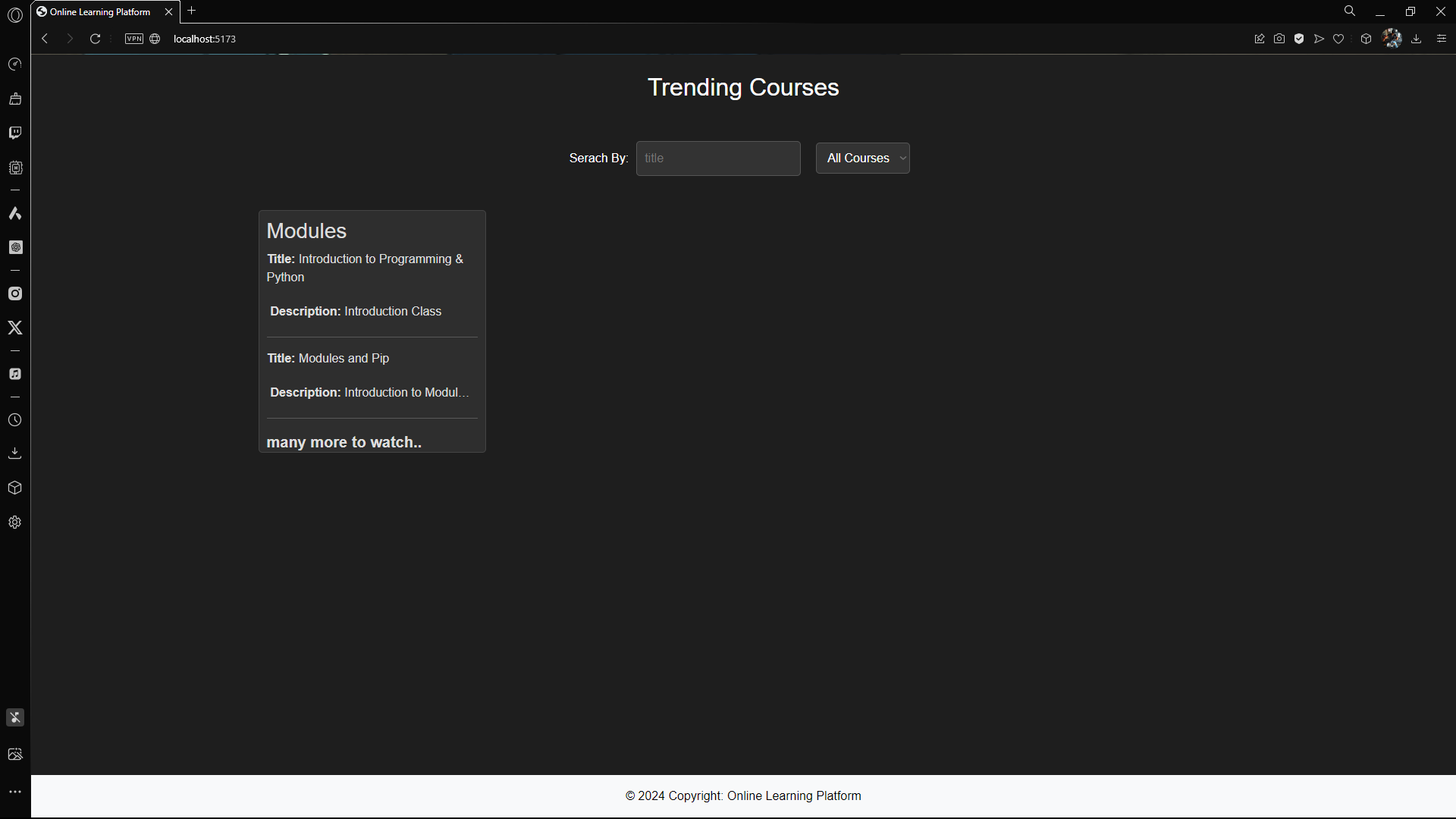
1. **Login/Register**:
   * Clean and simple forms to register or log in to the platform.
   * Error handling for invalid credentials and duplicate registrations.
2. **Dashboard**:
   * Dynamic navigation tailored to the user’s role:
     + Admin: Displays an overview of users, courses, and key metrics.
     + Teacher: Provides options to add courses, view enrolled students, and track course statistics.
     + Student: Lists enrolled courses and their progress.
3. **Course Content**:
   * Interactive page with an embedded video player for course sections.
   * Displays progress for each section.
   * A “Complete” button for marking sections as done, which dynamically updates the user's progress.

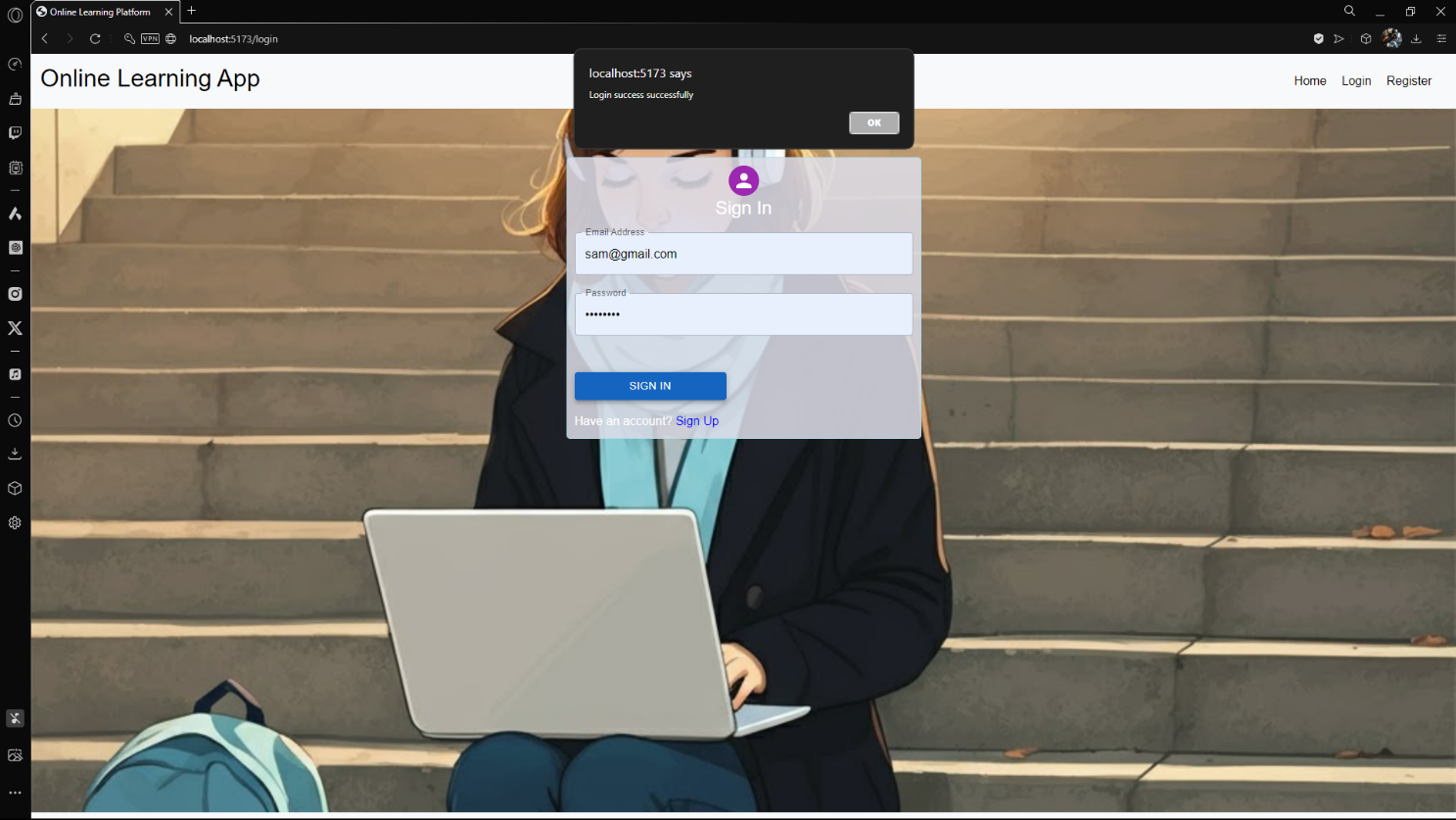
**10. Testing**

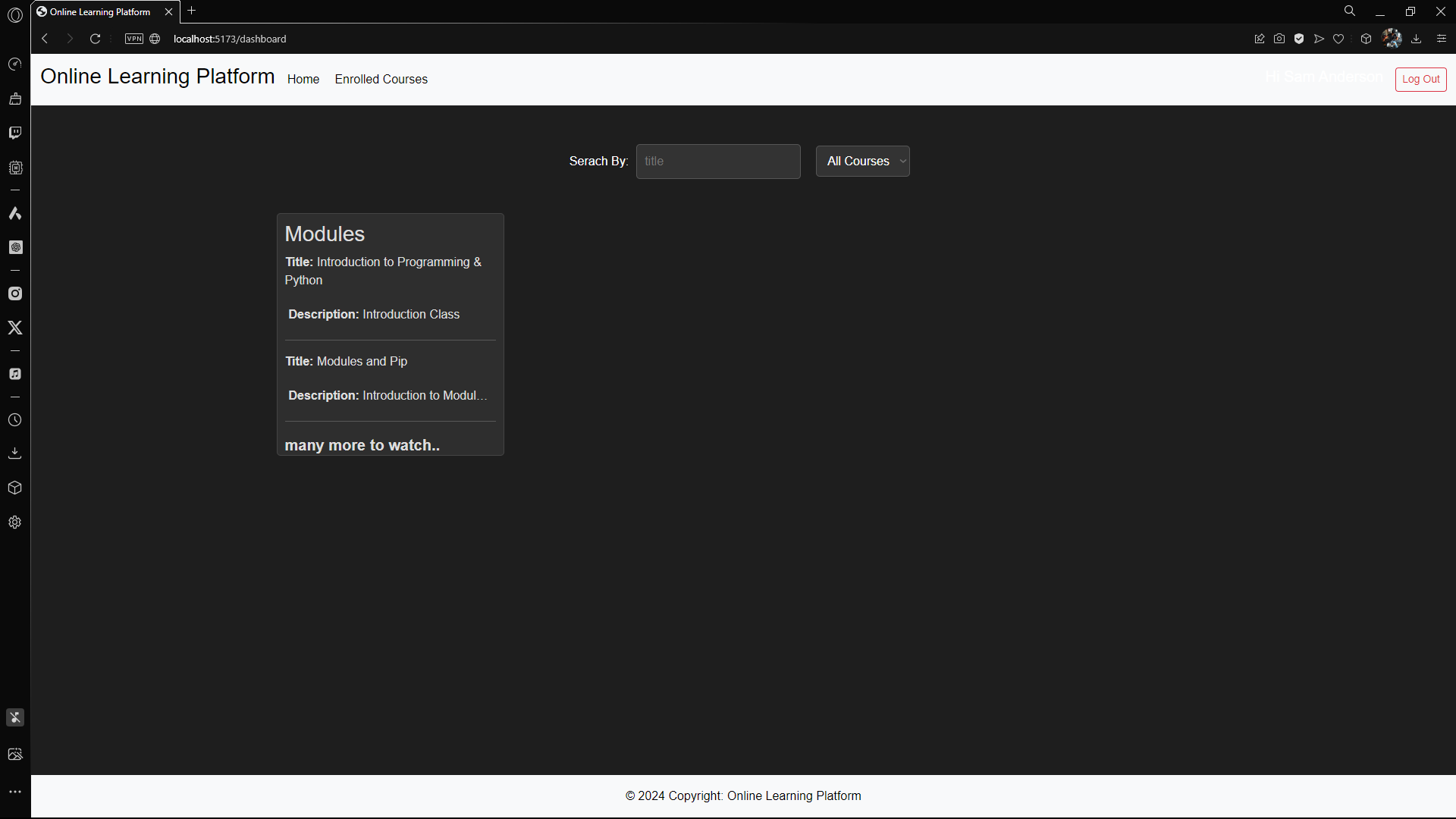
1. **Manual Testing**:
   * Focus on user flows:
     + Registration: Ensure user details are validated and stored correctly.
     + Login: Test token generation and storage.
     + Course creation (Teacher): Verify file uploads and YouTube link integrations.
     + Enrollment and completion (Student): Validate progress tracking and certificate generation.
2. **Integration Testing**:
   * Use Postman to test API endpoints:
     + Validate request/response structure for endpoints like /register, /login, /addcourse, etc.
     + Test role-based access by sending requests with different tokens.
3. **Edge Case Testing**:
   * Attempt invalid scenarios like:
     + Accessing restricted endpoints without authentication.
     + Enrolling in non-existent courses.
     + Handling empty or invalid form submissions.

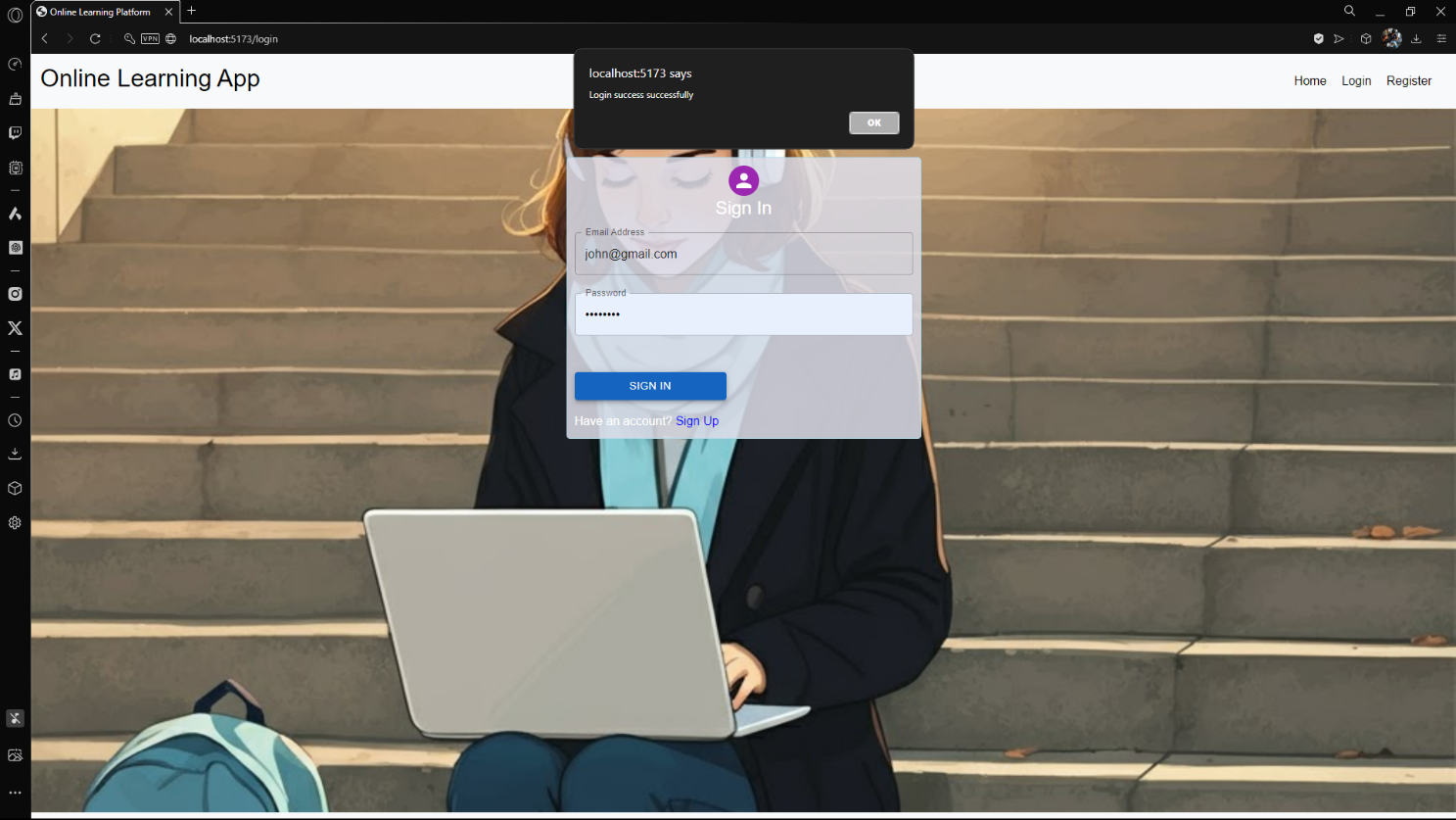
**11. Screenshots or Demo**

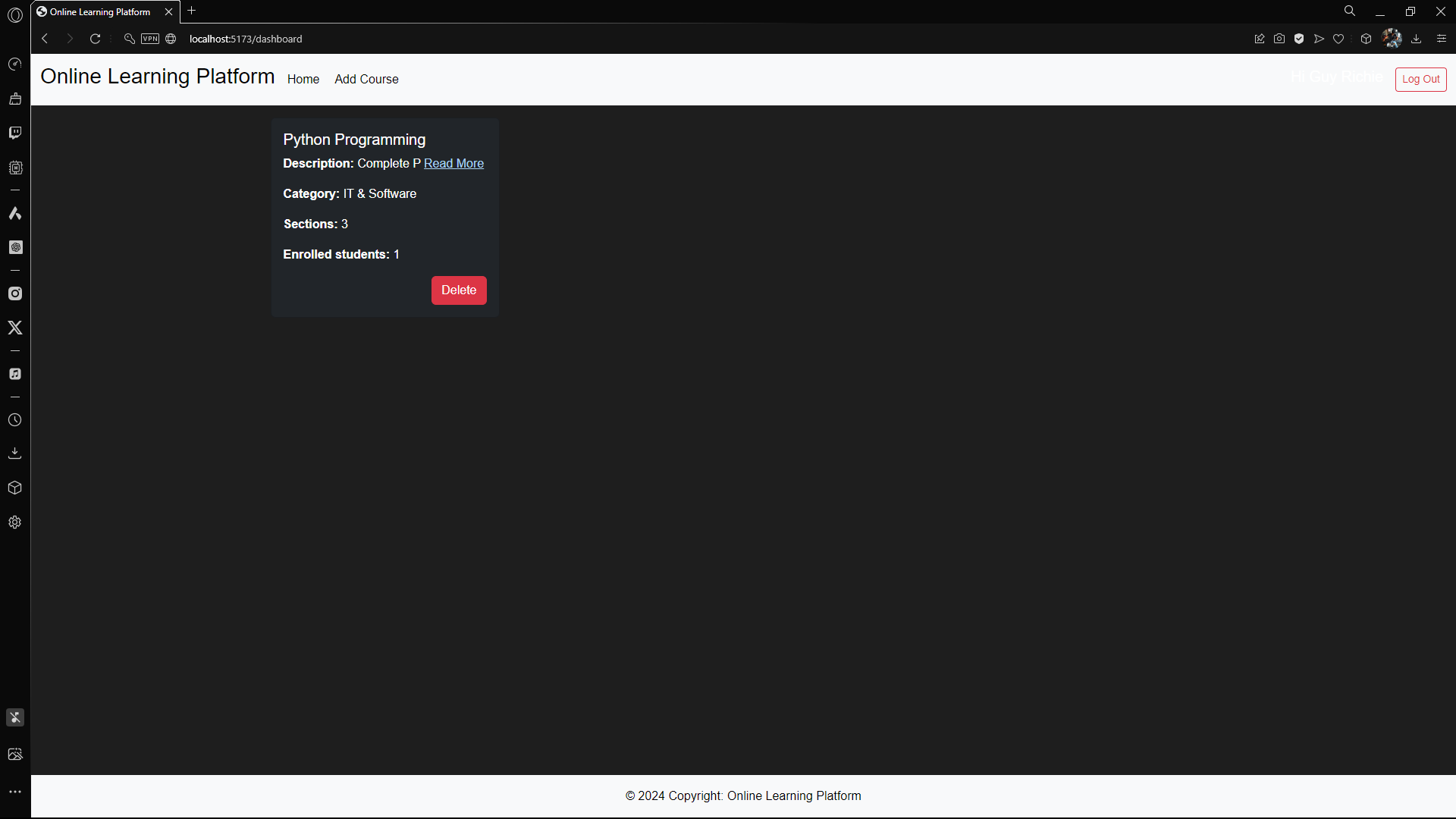












**12. Known Issues**

1. **Large File Uploads**:

* Uploading large video files (greater than 25MB) directly may lead to delays or errors due to GitHub file size limitations and server upload constraints.
* **Recommendation**: Encourage users to upload videos to YouTube and add the link instead.

1. **Incomplete Progress Tracking**:

* In rare cases, concurrent sessions by the same user might cause progress data inconsistencies.
* **Workaround**: Regularly sync progress data with the server to ensure accuracy.

**13. Future Enhancements**

1. **Real-time Notifications**:

* Notify users of important updates like course availability, enrollment confirmations, or section completions.
* Use WebSockets or Firebase Cloud Messaging for real-time alerts.

1. **Advanced Analytics**:

* For teachers:
  + Detailed insights into student engagement (e.g., section completion rates, video play statistics).
  + Exportable reports on student performance.
* For admins:
  + Aggregate data on platform usage, popular courses, and active users.

1. **Enhanced Admin Dashboard**:

* Include graphical insights like bar charts for user signups and pie charts for course enrollments.
* Provide search and filter options for managing users and courses efficiently.