**Full Stack Development with MERN**

**Project Documentation**

**1. Introduction**

**Project Title:** Online Learning Platform

**Team Members:**

* **Achyuth Sudharshan** – Team Lead [Frontend Developer]
* **Mohammed Hasan Mustafa** - Backend Developer
* **Azhagan** - Database Administrator
* **Alina Muskhan** - UI/UX Designer

**2. Project Overview**

**Purpose:**  
The purpose of the **Online Learning Platform** is to provide a comprehensive and user-friendly solution for both students and instructors in the field of e-learning. It aims to simplify the process of course creation, management, and enrollment by offering a seamless platform where instructors can design and offer educational content, while students can easily browse, enroll, and access learning materials. By implementing role-based permissions, the platform ensures that only authorized users can perform certain actions, such as course creation and management, enhancing the security and organization of the platform.

This system fosters a dynamic learning environment where instructors can directly interact with their students, track progress, and share knowledge in various formats like videos, quizzes, and assignments. For students, the platform offers a convenient and flexible way to learn at their own pace and convenience. The platform also facilitates administrative tasks, including enrollment management, content updates, and course feedback, all within a centralized system.

Overall, the project is designed to be scalable, allowing for future feature additions such as live sessions, certifications, and a broader range of interactive learning tools. This makes it an ideal foundation for an expanding e-learning ecosystem. The goal is to create an intuitive, secure, and feature-rich platform that can support diverse learning styles and needs, making online education accessible and efficient for all users.

**Features:**

1. **User Authentication: Secure login and registration for students and instructors**  
   The platform implements robust user authentication, ensuring that students and instructors can securely register, log in, and access their accounts. Users can register with basic details (such as email and password) and log in using their credentials to gain access to their respective dashboards. The authentication system is built using secure password hashing, ensuring user data is protected. Additionally, users are issued tokens (JWTs) upon successful login to authenticate requests to the platform.
2. **Role-Based Access Control: Specific permissions for instructors (course creation) and students (course enrollment)**  
   The platform enforces role-based access control (RBAC), where users are assigned roles such as "student" or "instructor." This allows for specific permissions tied to each role. Instructors are granted the ability to create, update, and delete courses, whereas students can only enroll in and access courses. This ensures that the right users can perform the correct actions on the platform, preventing unauthorized modifications and maintaining data integrity.
3. **Course Management: Create, update, and delete courses with descriptions, categories, and pricing options**  
   Instructors have full control over their courses. They can create new courses, providing details like title, description, category, and price. The platform allows instructors to update course content (e.g., adding or modifying lessons) and delete courses when necessary. Categories help organize the courses into relevant sections such as "Technology," "Business," or "Arts," making it easier for students to navigate and find courses of interest. Pricing options enable instructors to offer courses for free or for a set fee, supporting both paid and free learning opportunities.
4. **Enrollment Management: Track student enrollments and provide access to enrolled courses**  
   The platform enables students to browse available courses and enroll in those that interest them. Enrollment is tracked through the system, and students can only access the courses they have enrolled in. The system also provides instructors with access to a list of students enrolled in their courses, allowing them to manage students effectively, track progress, and offer support as needed. Additionally, the platform can allow students to view their enrollment history, track progress, and even leave feedback or ratings for the courses they complete.
5. **Interactive Learning Tools**  
   The platform can support various interactive learning tools to enhance the student experience. Features such as quizzes, assignments, discussion forums, and live chat can be integrated into the platform. These tools provide real-time feedback to students, help instructors assess student understanding, and create a more engaging and interactive learning environment.
6. **Course Certificates**  
   Upon completing a course, students may be issued a certificate of completion, which could be valuable for their professional development. This feature is beneficial for both students and instructors, as it adds a tangible outcome to the course and can serve as proof of skill acquisition.
7. **Course Progress Tracking**  
   Students can track their progress within a course. This feature allows students to see how far they've come, which lessons they've completed, and how much is left. Progress tracking can be displayed as a percentage or as a visual progress bar, helping students stay motivated and on track to finish their course.
8. **Course Rating and Feedback**  
   After completing a course, students can rate the course and provide feedback, helping other students make informed decisions about whether to enroll. Instructors can also use this feedback to improve course content, materials, and delivery methods.
9. **Notification System**  
   A notification system can keep users informed about updates related to courses, new enrollments, deadlines, course announcements, and more. Students can receive notifications when a new lesson is available, when an instructor updates the course material, or when an upcoming assignment is due.
10. **Search and Filtering**  
    Students can search for courses based on keywords, categories, price ranges, and other criteria. This helps students find the courses that best meet their learning goals. Advanced filtering options, such as course duration, skill level, and ratings, make course discovery more efficient.
11. **Admin Dashboard for Management**  
    The platform can offer an admin dashboard, giving platform administrators the ability to manage user accounts, monitor course creation and enrollment, and oversee other platform activities. Admins can also manage roles and permissions, making sure that only authorized users can perform certain actions.

**3. Architecture**

**Frontend:**

The **frontend** of the Online Learning Platform is designed using **React**, a popular JavaScript library for building user interfaces. The development approach follows modern best practices with **functional components** and **React hooks** to manage state and lifecycle methods. **React Router** is utilized for smooth navigation across different pages without reloading the entire page, ensuring a single-page application (SPA) experience.

**State management** is handled using **React's Context API**, which provides a way to share state globally without prop drilling. This is especially useful for managing authentication states, user roles (student or instructor), and the current course the user is interacting with. By using Context API, components can subscribe to the state they need without needing to pass props down manually.

For the **user interface (UI)**, **Material UI** is used as a design framework to ensure a modern, clean, and responsive layout. Material UI provides pre-designed components such as buttons, text fields, cards, and modals, which are easy to customize and integrate into the platform. The design is fully **responsive**, ensuring that the platform is optimized for use across various screen sizes, from mobile devices to desktops.

**Backend:**

The **backend** of the platform is built with **Node.js** and **Express.js**, which together provide a lightweight and efficient environment for building RESTful APIs. **Express.js** simplifies the routing and handling of HTTP requests, while **Node.js** provides a non-blocking I/O environment for fast, real-time applications.

The backend follows the **MVC (Model-View-Controller)** architecture, separating concerns by organizing code into models, views, and controllers:

* **Models** define the structure and schema of the data using **Mongoose** for MongoDB interaction.
* **Controllers** manage the logic for handling HTTP requests (such as creating, updating, or deleting courses, or enrolling students in courses).
* **Views** are handled by the frontend and are not part of the backend, as the backend only provides the API responses.

For **authentication and authorization**, the platform uses **JWT (JSON Web Tokens)**, a secure method for transmitting user credentials and claims between the client and server. Once a user logs in, they are issued a JWT token that is stored locally (usually in the browser's local storage). For secure password handling, **Bcrypt** is used to hash passwords before they are stored in the database, ensuring that even if the database is compromised, user passwords remain protected.

**Role-based access control** is a crucial feature, allowing the platform to differentiate between **students** and **instructors**. Students can only access courses and enroll, while instructors have permissions to create, update, and delete courses. This role-based system ensures that users only have access to the functionality that is relevant to their role.

**Database:**

The platform uses **MongoDB**, a NoSQL database known for its flexibility, scalability, and ease of use with JSON-like documents. **Mongoose**, an Object Data Modeling (ODM) library, is used to define schemas for the collections, manage data validation, and interact with the database more easily.

Key **MongoDB collections** include:

* **Users**: Stores user data, including email, password (hashed), role (student or instructor), and any other relevant profile information.
* **Courses**: Contains course-related data such as title, description, instructor ID (for linking courses to instructors), category, pricing, and lessons.
* **Enrollments**: Tracks the relationship between students and courses, allowing the platform to determine which students are enrolled in which courses.

The database design supports the relationships between **students** and **courses**. Each student can enroll in multiple courses, and each course can have multiple students. MongoDB’s flexible schema allows easy modification and scaling as new features are added, such as adding new attributes to courses or users.

Overall, the combination of **MongoDB**, **Node.js with Express**, **React**, and **Material UI** creates a robust and scalable architecture for the Online Learning Platform, providing a smooth and intuitive user experience while maintaining a powerful backend capable of supporting role-based access control, course management, and secure authentication.

**4. Setup Instructions**

**Prerequisites:**

To get started with the Online Learning Platform, ensure that the following prerequisites are installed on your system:

* **Node.js (v14 or above)**: The platform is built using Node.js, which is used to run both the frontend and backend. Install it from [Node.js official website](https://nodejs.org/).
* **MongoDB**: The platform requires a MongoDB database to store user data, courses, and enrollments. You can either set up MongoDB locally or use a cloud-based MongoDB service like [MongoDB Atlas](https://www.mongodb.com/cloud/atlas).

**Installation:**

Follow these steps to set up and run the project locally:

1. **Clone the repository**: Open a terminal or command prompt and run the following command to clone the project repository to your local machine:

git clone https://github.com/your-repo/online-learning-platform.git

1. **Navigate to the project directory**: Change into the project directory where the repository was cloned:

cd online-learning-app

1. **Install dependencies for frontend and backend**: The project has both a frontend and a backend, and you need to install the necessary dependencies for each.
   * For the **frontend** (React):

cd frontend npm install

* + For the **backend** (Node.js, Express):

cd backend npm install

1. **Configure environment variables in**.env**file**: In both the frontend and backend directories, you will need to configure certain environment variables.
   * Create a .env file in the **backend** directory and define the following variables:

Env

DB\_URI=mongodb://your-mongo-db-uri

JWT\_SECRET=your-secret-key

PORT=5000

* + - DB\_URI: MongoDB connection string, which can be either a local MongoDB URI or a URI from MongoDB Atlas.
    - JWT\_SECRET: A secret key used to sign and verify JSON Web Tokens (JWTs).
    - PORT: The port on which the backend server will run (e.g., 5000).

1. **Run the application**:
   * To start the **backend** server:

cd backend npm start

* + To start the **frontend** development server:

cd frontend npm start

**5. Folder Structure**

#### **Frontend**

The frontend is where the user interacts with the platform. It is built with React and provides all the pages and components for users.

* **src/components**: This folder holds reusable components that can be shared across different pages of the app. Examples include:
  + **Navbar**: The top navigation bar for easy navigation between different pages.
  + **CourseCard**: A component that displays course information in a card layout on the homepage or course listings.
* **src/pages**: This folder defines the main pages of the application that correspond to different routes. Examples include:
  + **Home**: The homepage, where users can browse available courses.
  + **Login**: The login page, where users can authenticate to the platform.
  + **Signup**: The registration page for new users.
  + **CourseDetails**: The page that displays detailed information about a course and allows students to enroll.
* **src/context**: This folder contains context files for **global state management** using React’s Context API. Examples include:
  + **AuthContext**: Manages user authentication status, JWT tokens, and user roles (student or instructor).
  + **CourseContext**: Manages the list of courses and enrollment status.

#### **Backend**

The backend handles all the business logic, authentication, database interactions, and API routes.

* **controllers**: This folder contains files that define the core business logic for the platform. Each controller is responsible for handling specific actions like:
  + **courseController.js**: Contains functions to create, update, delete, and list courses.
  + **userController.js**: Contains functions for user registration, login, and profile management.
  + **enrollmentController.js**: Handles student enrollments, managing the relationship between students and courses.
* **models**: Defines the **MongoDB schemas** for the main entities in the application, such as users, courses, and enrollments. These models are used by Mongoose to interact with the MongoDB database.
  + **User.js**: Defines the schema for storing user details, including their email, password (hashed), and role (student or instructor).
  + **Course.js**: Defines the schema for storing course information, such as title, description, instructor, and price.
  + **Enrollment.js**: Defines the schema for tracking which students are enrolled in which courses.
* **routes**: This folder contains the **API routes** for handling HTTP requests. Each route file is dedicated to a specific resource and contains the URL paths that map to the corresponding controller actions.
  + **userRoutes.js**: Defines routes related to user actions, such as registration, login, and profile.
  + **courseRoutes.js**: Defines routes for managing courses, like creating a new course, retrieving courses, and enrolling students.
* **middleware**: This folder contains custom **middleware functions** that are executed before hitting the route handler. Middleware can be used for tasks such as validating data, checking authentication status, and ensuring the correct user roles.
  + **authMiddleware.js**: Checks whether the request is authenticated (using JWT tokens) and ensures the correct role (student or instructor) is allowed to access the requested resource.

**6. Running the Application**

* **Frontend:**  
  Run npm start in the frontend directory to start the frontend server.
* **Backend:**  
  Run npm start in the backend directory to start the backend server.

**7.API Documentation**

This section provides the details for the available API endpoints in the Online Learning Platform. The platform has several endpoints that handle user registration, login, course creation, and enrollment.

**Authentication Endpoints**

1. **POST /api/auth/register**
   * **Description**: Registers a new user (student or instructor).
   * **Request Body**:

{ "email": "user@example.com", "password": "securepassword123", "role": "student" *// Can be "student" or "instructor"* }

* + **Response**:
    - **Status**: 201 Created (if successful)
    - **Body**:

{ "status": "success", "message": "User registered successfully" }

1. **POST /api/auth/login**
   * **Description**: Authenticates a user and returns a JWT token.
   * **Request Body**:

{ "email": "user@example.com", "password": "securepassword123" }

* + **Response**:
    - **Status**: 200 OK (if successful)
    - **Body**:

{ "status": "success", "token": "jwt\_token\_here" *// JWT token for authentication in subsequent requests* }

**Course Endpoints**

**POST /api/courses**

* + **Description**: Creates a new course (accessible only by instructors).
  + **Request Body**:

{ "title": "Introduction to Machine Learning", "description": "Learn the fundamentals of machine learning, algorithms, and applications.", "price": 49.99 }

* + **Response**:
    - **Status**: 201 Created (if successful)
    - **Body**:

{ "status": "success", "data": { "courseId": "123", "title": "Introduction to Machine Learning", "instructor": "instructor-id-12345" } }

1. **GET /api/courses**
   * **Description**: Fetches all available courses.
   * **Response**:
     + **Status**: 200 OK (if successful)
     + **Body**:

{ "status": "success", "data": [ { "courseId": "123", "title": "Introduction to Machine Learning", "instructor": "instructor-id-12345" }, { "courseId": "124", "title": "Advanced Data Science", "instructor": "instructor-id-67890" } ] }

1. **POST /api/courses/**

**/enroll**

* + **Description**: Enrolls a student in a specific course. The student must be authenticated.
  + **Request Body**:

json

Copy code

{ "studentId": "student-id-56789" }

* + **Response**:
    - **Status**: 200 OK (if successful)
    - **Body**:

{ "status": "success", "data": { "courseId": "123", "title": "Introduction to Machine Learning", "instructor": "instructor-id-12345" } }

**Example Responses**

* **Course Creation Response (POST /api/courses)**:

{ "status": "success", "data": { "courseId": "123", "title": "Introduction to Machine Learning", "instructor": "instructor-id-12345" } }

* **Fetching All Courses Response (GET /api/courses)**:

{ "status": "success", "data": [ { "courseId": "123", "title": "Introduction to Machine Learning", "instructor": "instructor-id-12345" }, { "courseId": "124", "title": "Advanced Data Science", "instructor": "instructor-id-67890" } ] }

* **Student Enrollment Response (POST /api/courses/**

**/enroll)**:

{ "status": "success", "data": { "courseId": "123", "title": "Introduction to Machine Learning", "instructor": "instructor-id-12345" } }

**8. Authentication**

Authentication is a crucial part of the platform, ensuring that users (both students and instructors) can only access the resources they are authorized to. Here's how it's implemented:

* **JWT-based Authentication**:
  + When users log in (via the POST /api/auth/login endpoint), they are issued a **JWT (JSON Web Token)**. This token contains user-specific information (like their role) and is used for **authorization** in subsequent requests.
* **Role-based Authorization**:
  + There are two types of users in the system: **Instructors** and **Students**.
    - **Instructors**: Can create and manage courses. They have restricted access to certain routes like POST /api/courses (for creating courses).
    - **Students**: Can only enroll in courses. They cannot access course creation or management functionalities.
* **Middleware for Token Validation**:
  + The backend uses **middleware** to validate the JWT token on protected routes. If the token is valid, the request proceeds; otherwise, the user receives a **401 Unauthorized** error.
  + Additionally, the middleware checks the **role** of the user, ensuring that only instructors can access certain routes like course creation or management.

### 9. ****User Interface****

The **frontend** is designed to provide a smooth and intuitive experience for both students and instructors. Key features include:

#### **Pages and Screens**

* **Login and Signup Pages**:
  + Allow users to register or log in. The UI includes form fields for entering the email, password, and selecting a role (student or instructor).
  + Validation messages are shown for incorrect credentials, and successful login redirects users to their respective dashboard (instructor or student).
* **Instructor Dashboard**:
  + **Course Management**: Instructors can view, create, edit, and delete courses. They can also see the list of students enrolled in their courses.
  + The dashboard will display information such as the course name, description, and the number of enrolled students.

Example dashboard layout:

* + **Create Course** button to add new courses.
  + **My Courses** section showing a list of courses the instructor has created.
  + **Student Enrollments** section showing the students enrolled in each course.
* **Course Catalog for Students**:
  + Displays a list of available courses for students to browse and enroll in.
  + Students can filter courses based on categories, popularity, or price.
  + **Enroll** button is available for each course, which adds the student to the course upon clicking.

Example course catalog layout:

* + A grid of **Course Cards** showcasing course details such as title, instructor name, price, and a brief description.
  + An **Enroll** button is available for each course.

#### **Design Considerations**:

* **Responsive Design**: The UI is designed to be responsive, using Material UI components, ensuring that it works seamlessly across different screen sizes.
* **Clear Navigation**: Navigation is intuitive with a top navbar for quick access to the dashboard, course catalog, and user settings.
* **Role-based Content**: Content is displayed based on the user’s role. Instructors see options to manage courses, while students only see available courses and enrollment options.

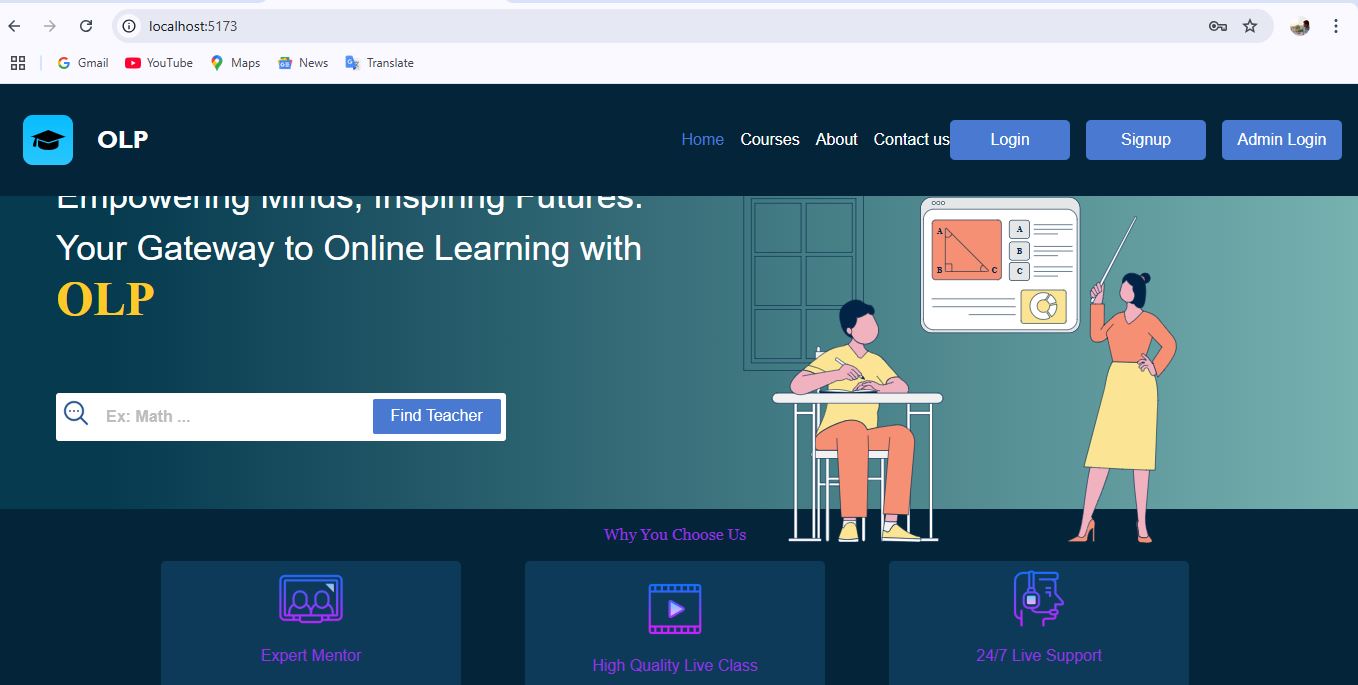
### 10. ****Testing****

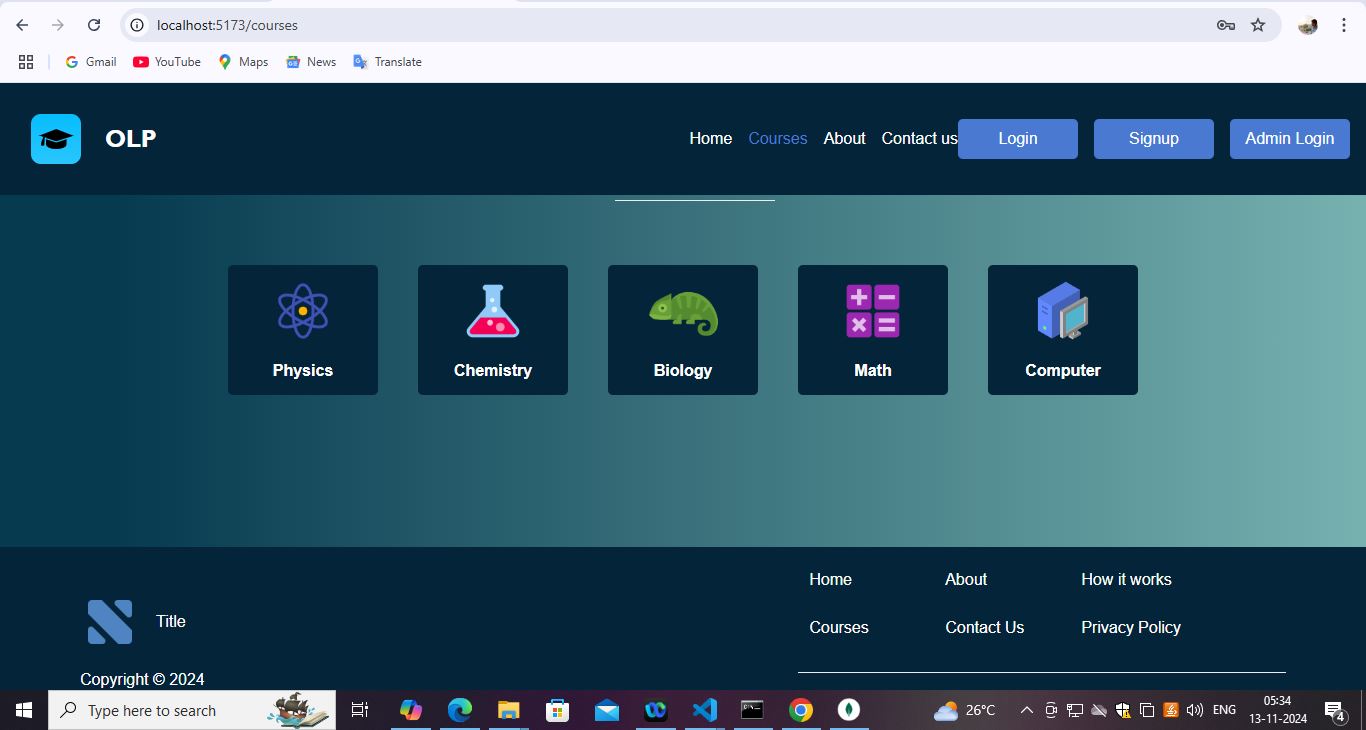
#### **Manual Testing**:

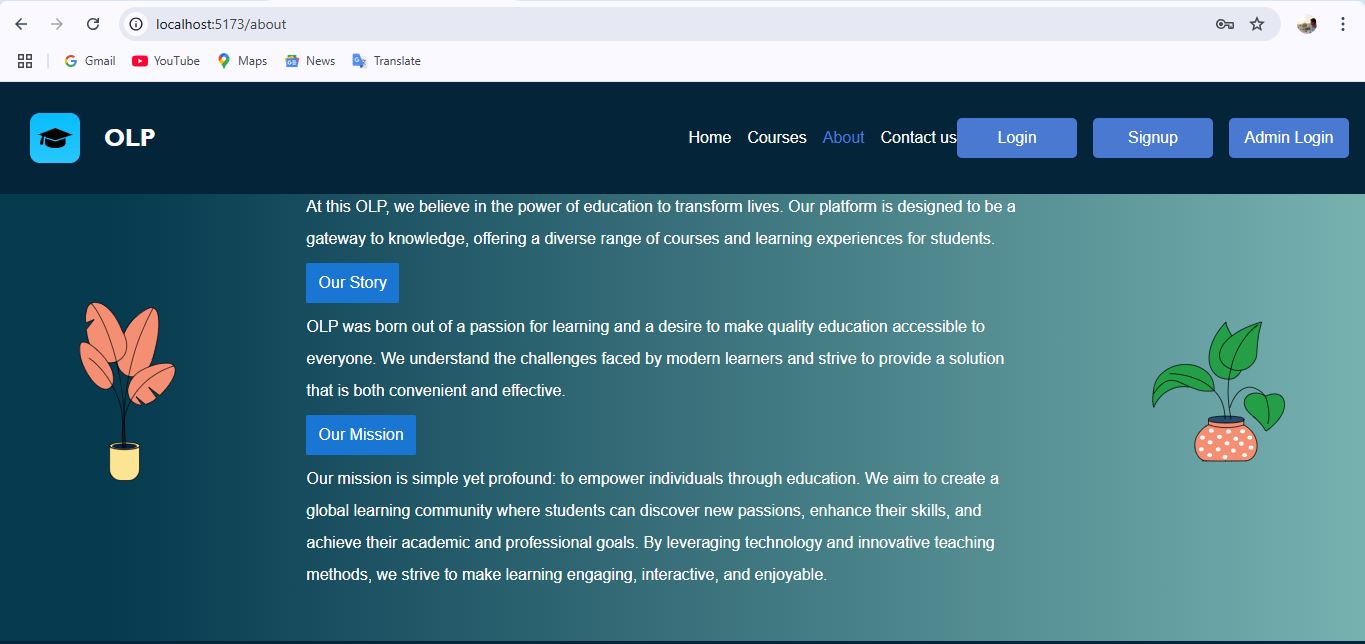
* Each endpoint was manually tested using **HTTPie** to ensure correct request-response cycles. The manual testing involved:
  + Registering new users (both students and instructors).
  + Logging in and receiving the JWT.
  + Accessing protected routes to ensure token validation and role-based access.
  + Testing the course creation and enrollment functionalities.

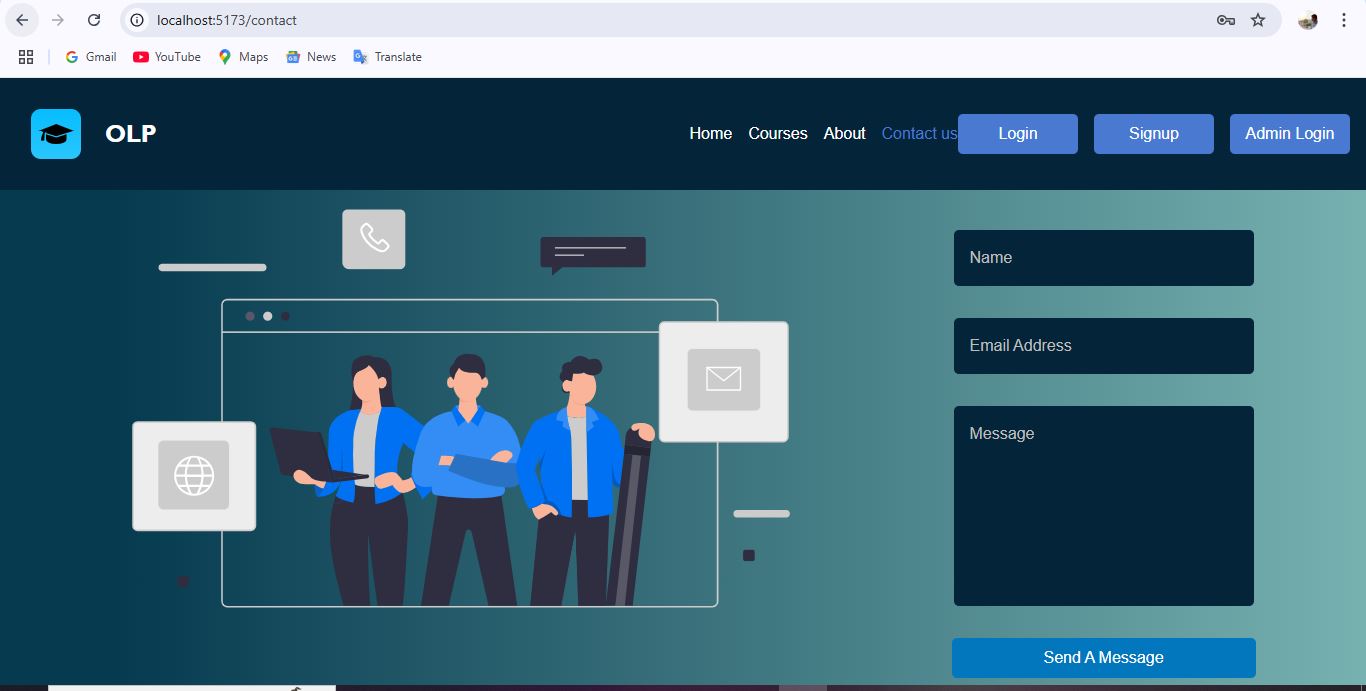
#### **Automated Testing:**

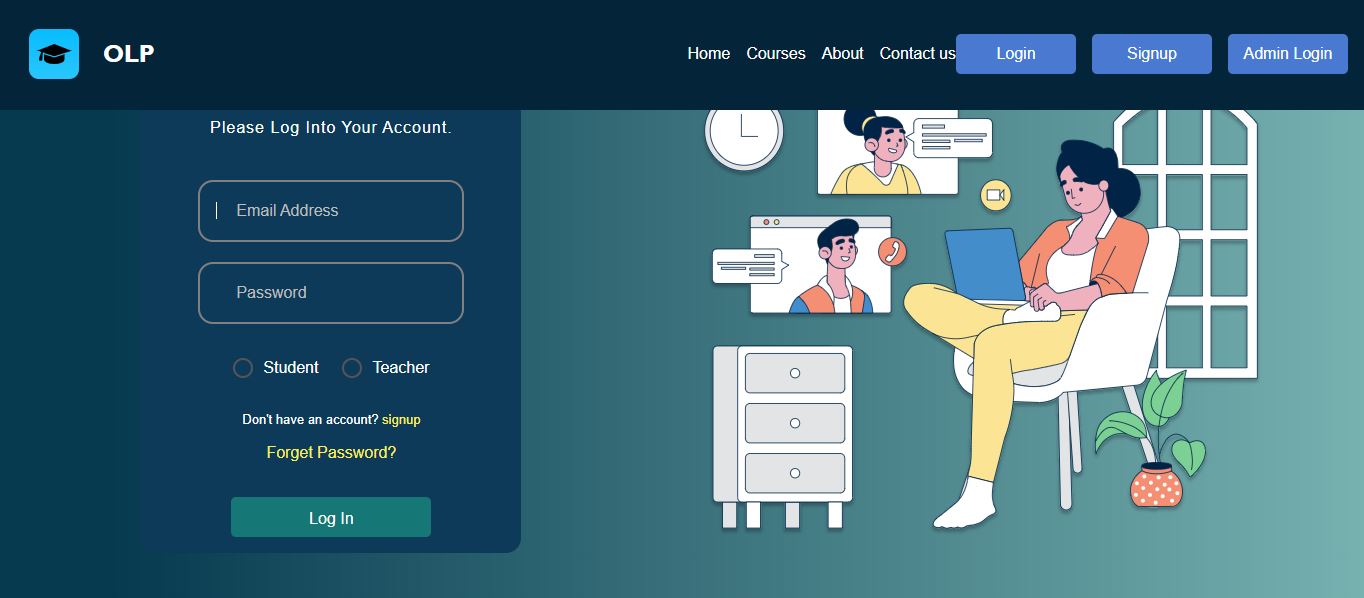
* **Mocha and Chai** were used for **unit** and **integration testing** of backend functionalities:
  + **Unit tests** were written for individual functions like user authentication, JWT generation, and validation.
  + **Integration tests** were written for entire workflows, such as user registration, login, course creation, and enrollment.

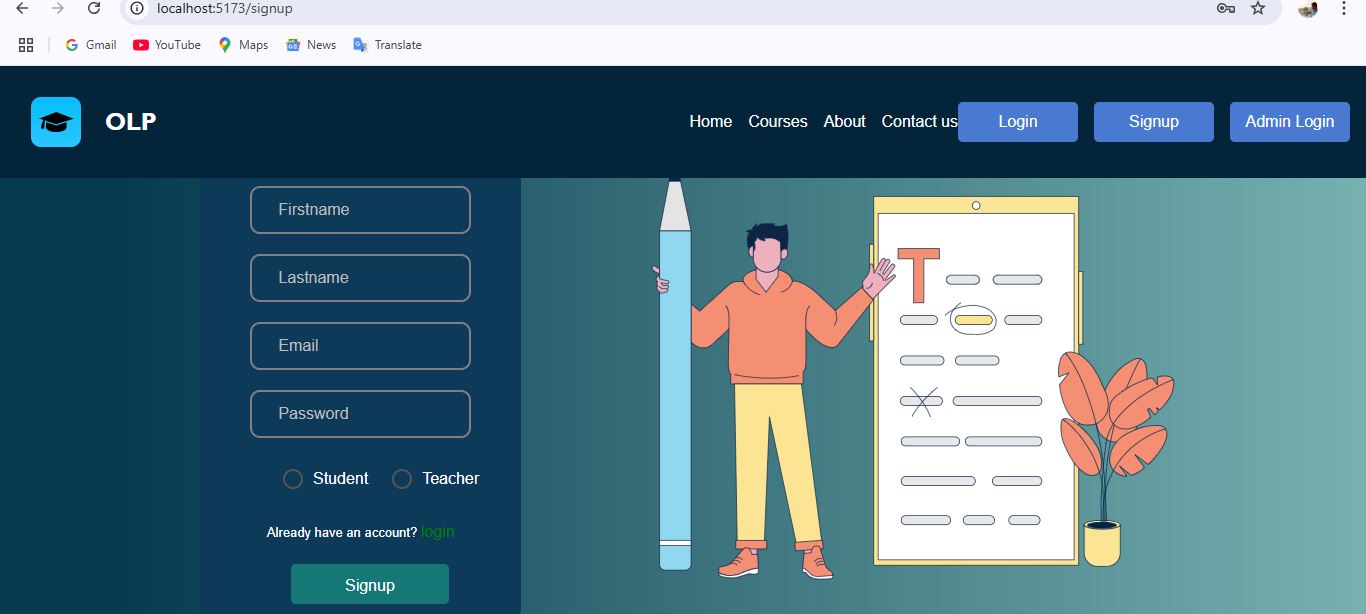
**11.Screenshots** 

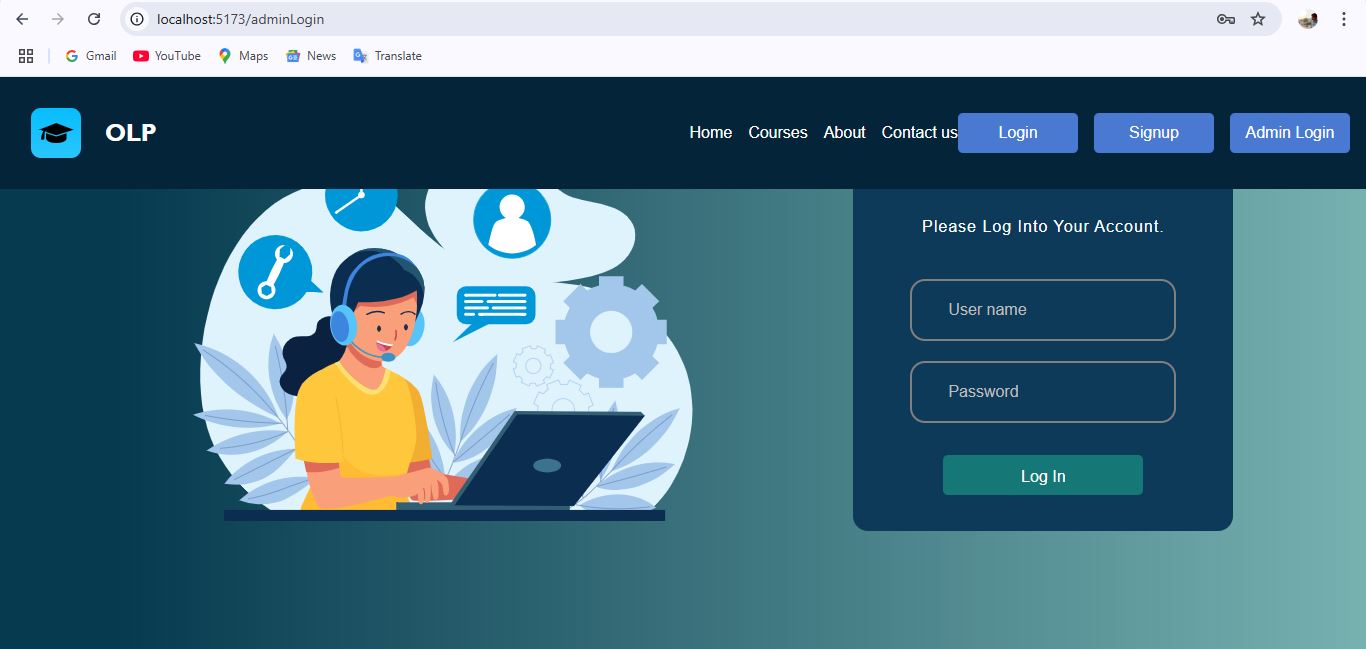
****

****

****

****

****

****

**12. Known Issues**

* **Error Handling:** Certain edge cases for error handling (e.g., invalid tokens) need refinement.
* **Session Expiry:** The session timeout feature can be implemented for enhanced security.

**13. Future Enhancements**

* **Chat Support:** Enable student-instructor chat for quick assistance.
* **Advanced Analytics:** Track student progress and provide instructors with course engagement insights.
* **Payment Integration:** Add a payment gateway to handle course purchase transactions.