1.Introduction

Online Learning Platform (OLP)

Team Members:

Team Member	Role
Mani kandan R	Lead (Oversees and contributes to all MERN stack development.)
Kaviya R	Developed backend
Prashanth D	Designed User Interface
Esther Jabina S	Developed Frontend
Bhuvanesh Kumar V	Test the Interface

2. Project Overview

Purpose

The purpose of the **Online Learning Platform (OLP)** is to create an accessible, scalable, and interactive learning environment. This platform aims to provide learners with the tools needed to enroll in courses, manage learning progress, and earn certifications—all within a user-friendly digital space. By utilizing the MERN stack, the project seeks to offer seamless user experiences across devices while supporting a variety of educational content and formats.

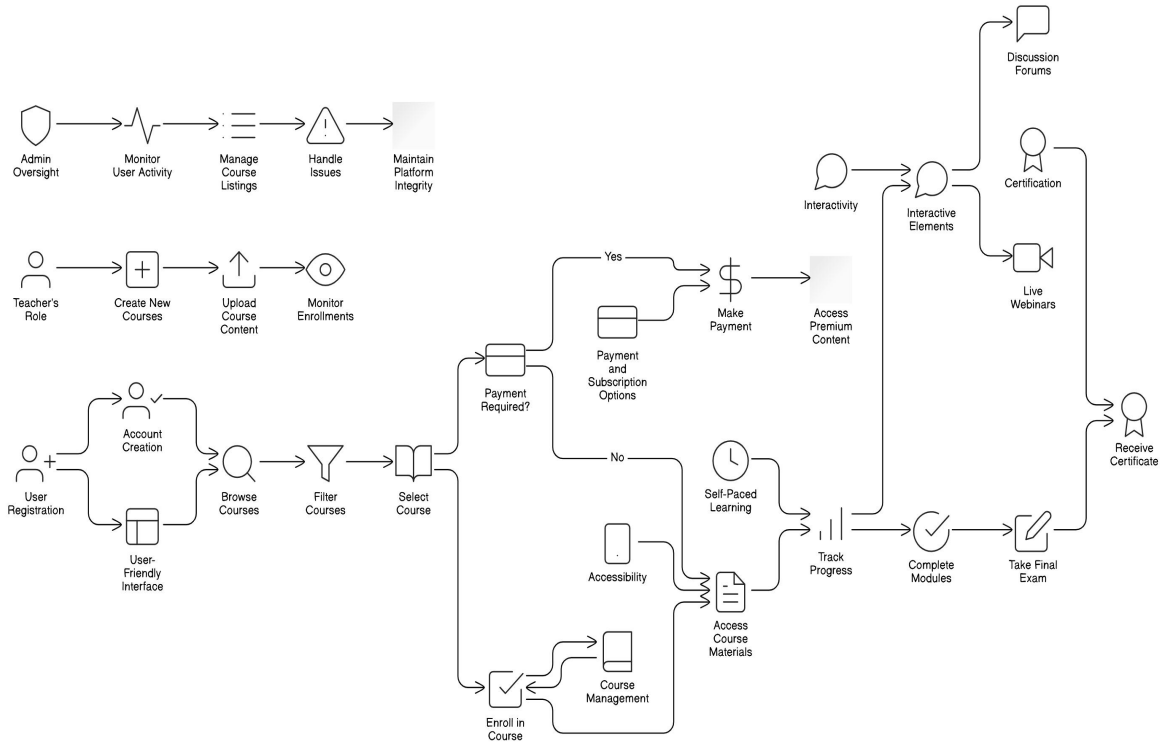
Goals

- **Accessibility**: Enable learning across various devices and ensure content is accessible anytime, anywhere.
- **Interactivity**: Foster engagement through features like discussion forums, live webinars, and collaborative tools.
- **Certification**: Allow learners to achieve certifications upon course completion, enhancing their learning credentials.
- **Scalability**: Design the platform to handle increasing user demand as it grows.

Features

- **User-Friendly Interface**: A clean, intuitive layout that allows users of all skill levels to navigate easily.
- **Course Management**: Facilitates instructors in creating, managing, and updating course content; enables learners to enroll and track progress.
- **Interactive Elements**: Includes discussion forums, chat features, and live webinar integration to support real-time interaction.
- **Self-Paced Learning**: Allows users to progress through courses at their own pace, with progress saved automatically.
- **Certification System**: Issues certificates to users upon successful course completion, which can be downloaded and shared.
- **Payment Integration**: Provides options for course payments, allowing access to premium content via secure payment methods.
- **Admin Dashboard**: Admins have control over platform operations, including course monitoring, user management, and issue resolution.

3. Architecture



Frontend Architecture (React)

The frontend is developed using React, focusing on creating a responsive and user-friendly interface for both students and instructors. Key elements include:

- **Component-Based Structure**: React components for user registration, course browsing, course management, payment, and access to interactive elements.
- **State Management**: Context API or a state management library (like Redux) to manage user authentication, enrollment status, and course progress.
- **Routing**: React Router for navigating between pages like course listings, payment options, and user dashboards.
- **Interactivity**: Interactive elements such as discussion forums and live webinars, utilizing third-party libraries or WebSocket for real-time communication.

Backend Architecture (Node.js & Express.js)

The backend, built with Node.js and Express.js, serves as the core of the application, managing requests and interacting with the database.

- **API Design**: RESTful APIs to handle user registration, course management, payment processing, and course content access.
- **Authentication & Authorization**: JWT (JSON Web Tokens) for secure user authentication and role-based access control for administrators, teachers, and students.
- **Course Management**: API endpoints for course creation, enrollment tracking, and content management.
- **Payment Gateway Integration**: Manages payment and subscription options, granting access to premium content after payment completion.

Database Schema and Interactions (MongoDB)

The database is structured to handle various entities such as users, courses, enrollments, and progress tracking.

- **User Schema**: Stores user details, role (admin, teacher, student), and enrollment history.
- **Course Schema**: Includes course details, content modules, teacher information, and access level (free/premium).
- **Enrollment Schema**: Tracks users' enrollment status, payment records, and course progress.
- **Progress Tracking**: Monitors module completion and generates certification upon course completion.

4. Setup Instructions

Prerequisites:

1. **Vite**: A fast build tool for React.
    - Install: npm create vite@latest and choose **React** template.
2. **Node.js & npm**: Required for server-side JavaScript.
    - Install: Node.js

3. **Express.js**: Web framework for backend APIs.
    - Install: npm install express
4. **MongoDB**: NoSQL database for storing app data.
    - Install: MongoDB
5. **React.js**: Frontend UI library.
    - Vite sets this up for you if you choose React template.
6. **Mongoose**: ODM for MongoDB in Node.js.
    - Install: npm install mongoose

Installation Steps

1. **Create Vite + React App**:
    - git clone <repo-url>
    - cd frontend
    - npm install
    - cd ../backend
    - npm install

2. **Set Up Backend (Express + MongoDB)**:
    - Create a backend folder, initialize Node.js, and install Express:
    - cd backend
    - npm init -y
    - npm install express mongoose

3. **Start Servers**:

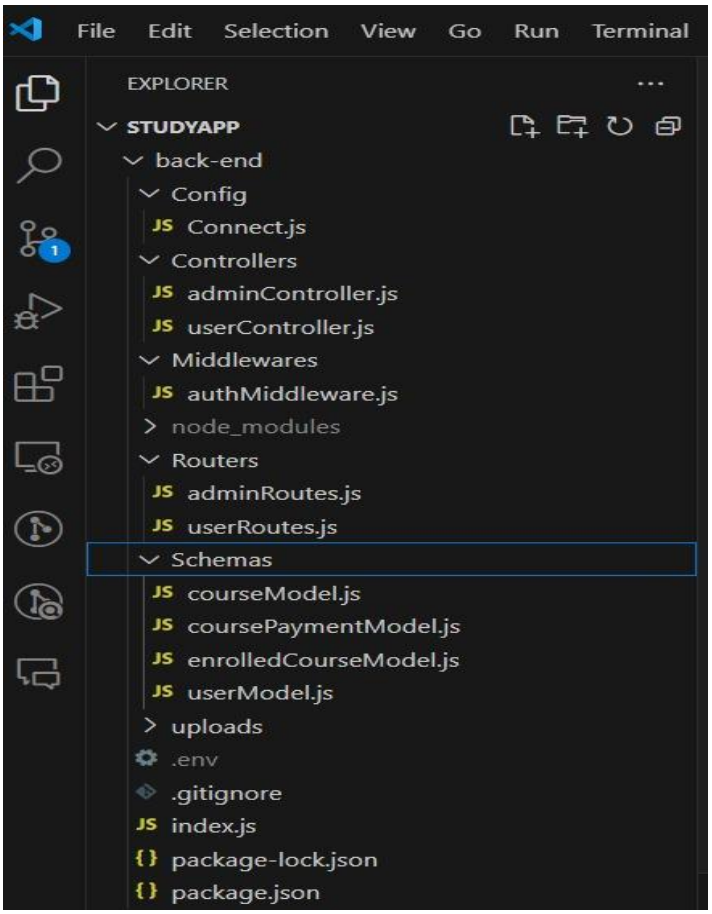
    Frontend: In the Vite project folder:
    - npm run dev
    - (Access at http://localhost:5173)

    Backend: In the backend folder:

    node server.js

5. Folder Structure

Client Side (React Frontend Structure)



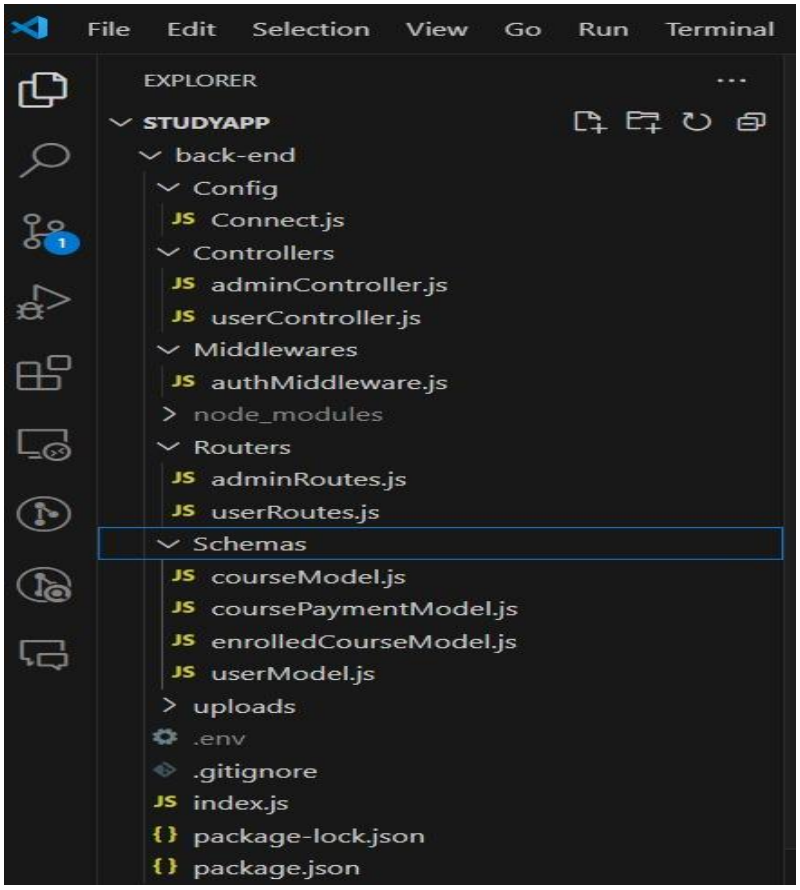
- **node_modules**: Contains the dependencies required for the React app to function. These are installed through npm and specified in the package.json file.
- **public**: Holds static files like index.html, which serves as the entry point for the React app. Any static assets accessible to the public can be placed here.
- **src**: The main source folder for the React app, containing:
- **Assets**: A directory for storing images, icons, or other static assets required by the application.
- **Components**: Contains the different components used across the app, divided into three subfolders:
    - **Admin**: Components used for admin functionalities, including AdminHome.jsx and AllCourses.jsx.

    - **Common**: Shared components accessible to various users, such as Dashboard.jsx, Home.jsx, NavBar.jsx, and authentication components (Login.jsx, Register.jsx).
    - **User**: Components specific to user roles:
        - **Student**: Components like CourseContent.jsx, EnrolledCourses.jsx, and StudentHome.jsx, meant for student functionalities.
        - **Teacher**: Components such as AddCourse.jsx and TeacherHome.jsx, meant for teacher functionalities.
- **App.css**: Contains global CSS styling for the application.
- **App.jsx**: The main component that sets up routing and renders the main application layout.
- **index.css**: Holds global styles that apply to the entire application.
- **main.jsx**: The entry point for rendering the React app.
- **Other Files**:
- **.gitignore**: Lists files and folders to ignore in Git version control.
- **eslint.config.js**: Configuration file for ESLint, which helps in maintaining code quality.
- **vite.config.js**: Configuration file for Vite, a tool used for fast and efficient development of the application

Server Side

1. **Config**: Contains configuration files, such as Connect.js, which handles the database connection setup.
2. **Controllers**: Contains the controller files, which define the logic for handling requests:
    - **adminController.js**: Defines functions to handle admin-related operations.
    - **userController.js**: Defines functions to handle user-related operations.
3. **Middlewares**: Holds middleware functions, like authMiddleware.js, which manage tasks such as authentication and authorization.

4. **Routers**: Contains route files that define the endpoints for different parts of the application:
    - **adminRoutes.js**: Defines the routes for admin functionalities.
    - **userRoutes.js**: Defines the routes for user functionalities.



5. **Schemas**: Stores Mongoose schema definitions for various collections in MongoDB:
    - **courseModel.js**: Defines the schema for courses.
    - **coursePaymentModel.js**: Defines the schema for course payments.
    - **enrolledCourseModel.js**: Defines the schema for enrolled courses.
    - **userModel.js**: Defines the schema for users.
6. **uploads**: Likely used to store uploaded files, though the exact purpose isn't specified in the structure.
7. **Other Files**:
    - **.env**: Environment configuration file to store sensitive information, such as database credentials.

- **.gitignore**: Lists files and folders to ignore in Git version control.
- **index.js**: The main entry point for the Node.js application.
- **package.json** and **package-lock.json**: Files listing project dependencies and other metadata.

6. Run the Application

1. **Start the Frontend Server (Vite + React)**

- Navigate to the frontend directory and run the following command:

  cd client

  npm run dev

  This command starts the Vite development server for the React frontend, typically accessible at http://localhost:5173 (or another port if 5173 is occupied).

2. **Start the Backend Server (Node.js + Express)**

   Open a new terminal window, navigate to the backend directory named as Server, and run the following command:

   cd server

   npm start

7. API Documentation

1. User Login

- **Endpoint**: /api/user/login
- **Method**: POST
- **Description**: Authenticates a user and generates a JWT token upon successful login.

Request Parameters:

- email (string): The email of the user.
- password (string): The password of the user.

Example :

```
{
  "email": "user@example.com",
  "password": "password123"
```

}

Response:

- success (boolean): Indicates whether the login was successful.
- message (string): A message providing additional information about the result of the login attempt.
- token (string): The JWT token generated for the user (if successful).
- userData (object): The user's data (if successful).

```
{
  "success": true,
  "message": "Login successful",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "userData": {
    "id": "12345",
    "name": "John Doe",
    "email": "user@example.com"
  }
}
```

Usage Example:

```
axiosInstance.post('http://localhost:5000/api/user/login', data)
  .then((res) => {
    if (res.data.success) {
      alert(res.data.message)

      localStorage.setItem("token", res.data.token);
      localStorage.setItem("user", JSON.stringify(res.data.userData));
      navigate('/dashboard')
      setTimeout(() => {
        window.location.reload()
      }, 1000)
    } else {
      alert(res.data.message)
    }
  })
  .catch((err) => {
    if (err.response && err.response.status === 401) {
      alert("User doesn't exist");
```

```
  }
  navigate("/login");
});
```

8. Authentication

Authentication in the Online Learning Platform (OLP) is managed using **JSON Web Tokens (JWT)**. JWT is used to securely verify the identity of users during login and ensure that only authorized users can access protected resources.

Authentication Flow:

- When a user logs in with their credentials (email and password), the server verifies the credentials against the database.
- If the credentials are correct, the server generates a JWT that contains the user's information and sends it back to the client.
- The client stores this token (typically in localStorage or a cookie) for use in subsequent requests.

Authorization:

Authorization controls access to different resources based on the user's role or privileges.

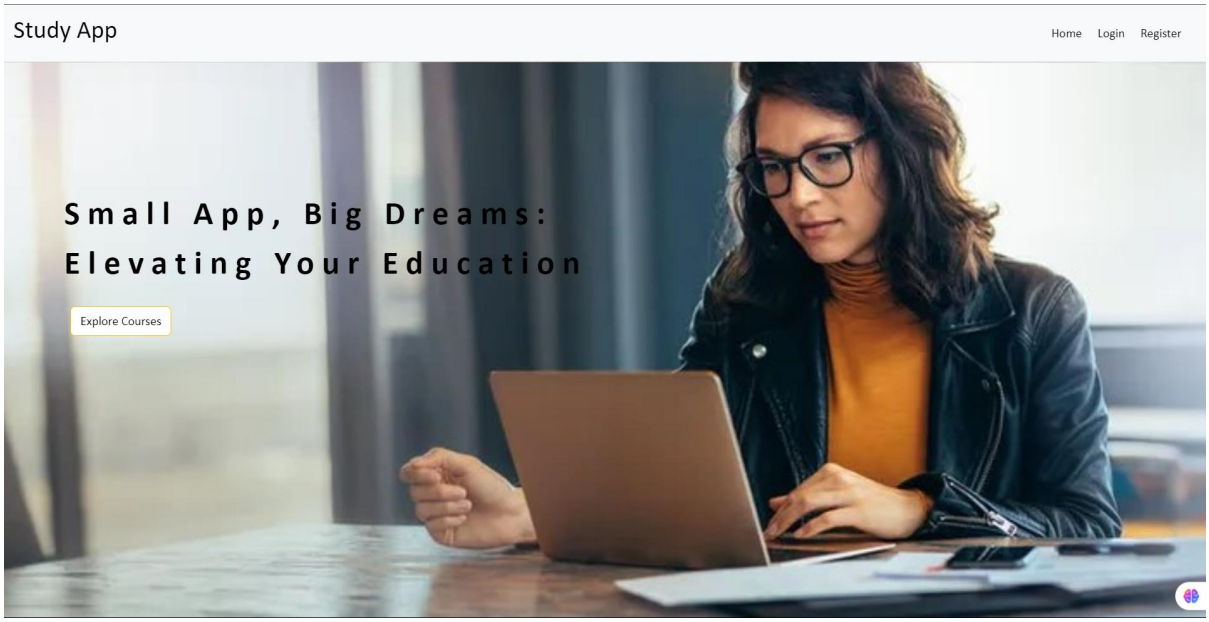
Authorization Flow:

- For each subsequent request to a protected route (e.g., accessing a course or dashboard), the client sends the JWT in the Authorization header of the HTTP request.
- The server validates the JWT and checks for its expiration. If the token is valid, the server grants access to the requested resource.
- Role-based authorization is applied by including roles within the JWT (e.g., admin, instructor, student), allowing the backend to check if a user has the correct permissions to perform a specific action or access a resource.
- If the token is expired or invalid, the server returns an error (e.g., 401 Unauthorized or 403 Forbidden).

Token Expiration:

JWTs have an expiration time, after which the token becomes invalid. When the token expires, users are required to re-authenticate or refresh their token to regain access.

9. User Interface



10. Testing

Testing Strategy: The testing strategy for the Online Learning Platform (OLP) combines manual and automated testing methods to ensure that all aspects of the platform function correctly and deliver a high-quality user experience. The primary focus is on functional and user experience testing, which ensures that both the frontend and backend work seamlessly together.

1. **Unit Testing**:
    - **Purpose**: To test individual components or functions in isolation and ensure they work as expected.
    - **Tool Used**: **Jest**

Jest is used to perform unit testing on React components. It helps verify that each component is rendered correctly, responds to user interactions, and handles the state as expected.

2. **End-to-End (E2E) Testing**:
    - **Purpose**: To test the entire application flow from the user's perspective, ensuring that the platform works as intended when users interact with it.
    - **Tool Used**: **Cypress**

Cypress is employed for end-to-end testing. It simulates real-world user interactions, such as signing up, enrolling in a course, and navigating through the learning platform, to ensure all critical workflows are functioning smoothly.

3. **Performance Testing**:
    - **Purpose**: To evaluate the platform's performance under various conditions, ensuring it handles traffic and user load efficiently.
    - **Tool Used**: **Google Lighthouse**

Google Lighthouse is used to assess performance metrics, such as load times, responsiveness, and other crucial user experience factors.

11. Demo Video Link:

https://drive.google.com/file/d/1f3lucfL5bzQkDK5I5DROM0TKtsbUlsuc/view?usp=sharing

12. Known Issues

1. **Authentication Token Expiry**
    a. **Issue**: Users may be logged out unexpectedly due to token expiry.
    b. **Impact**: Frequent re-logins required.
    c. **Solution**: Adjust token expiry settings and implement token refresh.

2. **Payment Gateway Failures**

a. **Issue**: Payment processing sometimes fails for premium courses.
b. **Impact**: Users unable to purchase courses.
c. **Solution**: Verify payment gateway settings and API keys.

3. **Course Enrollment Syncing**
    a. **Issue**: Enrollment data may not sync immediately across platform sections.
    b. **Impact**: Courses may not appear in the user dashboard.
    c. **Solution**: Refresh the page or check database syncing.
4. **Video Playback Issues**
    a. **Issue**: Videos may buffer or fail to load, especially on mobile devices.
    b. **Impact**: Disrupted learning experience.
    c. **Solution**: Implement adaptive bitrate streaming for better playback.
5. **Email Notification Delays**
    a. **Issue**: Delayed or missing email notifications for important events.
    b. **Impact**: Users may miss important course updates.
    c. **Solution**: Verify email configuration and SMTP server settings.

13. Future Enhancements

1. **User Analytics**: Track user engagement, completion rates, and popular courses for better insights.
2. **Advanced Search and Filtering**: Enable search by categories, difficulty, and personalized recommendations.
3. **Real-time Notifications**: Alert users about new content, webinars, and announcements.
4. **Mobile App**: Develop iOS and Android apps for on-the-go learning.
5. **Offline Access**: Allow users to download content for offline use.
6. **Gamification**: Add badges, points, and leaderboards to boost engagement.
7. **Multi-language Support**: Localize content to reach a global audience.
8. **Personalized Learning Paths**: Use AI to suggest tailored learning paths.
9. **Enhanced Discussion Forums**: Improve forums with tagging, upvotes, and live chat.
10. **Accessibility Features**: Add support for screen readers, captions, and adjustable fonts.

11. **Accredited Certifications**: Offer recognized certifications and shareable digital badges.
12. **Course Reviews**: Enable ratings and reviews to improve course selection.
13. **Integration with Learning Tools**: Connect with Google Classroom, Microsoft Teams, etc.
14. **Subscription Models**: Provide tiered access options and expanded payment choices.
15. **Admin Dashboard**: Enhance with data visualization and in-depth performance analytics.
16. **Content Management System (CMS)**: Create a CMS for easier course management and creation.