



STUDY APPLICATION MERN



A NAAN MUDHALVAN PROJECT REPORT

Submitted by

SUMIT KUMAR (310521104118)

VICKY KUMAR (310521104124)

PRINCE KUMAR RAM (310521104076)

ROHIT KUMAR PANDEY (310521104096)

In partial fulfillment for the award of the degree

of

**BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING**

**DHANALAKSHMI SRINIVASAN COLLEGE OF
ENGINEERING AND TECHNOLOGY
MAMALLAPURAM, CHENNAI- 603104**

ANNA UNIVERSITY :: CHENNAI-600025

NOVEMBER 2024



**DHANALAKSHMI SRINIVASAN
COLLEGE OF ENGINEERING & TECHNOLOGY**

(Approved by AICTE & Affiliated to Anna University, Chennai)

East Coast Road, Mamallapuram, Chennai - 603 104

Name

Reg. No

Department

Semester

Subject Name

Subject Code

*Certified that this is the Bonafide record of the Practicals done
for the above subject during the period
in the academic year 20 - 20*

Staff in Charge

Head of the Department

*submitted for the University Practical Examination
held on*

Internal Examiner

External Examiner

[illegible]

ACKNOWLEDGEMENT

I am deeply grateful to all those whose support and guidance made this project a reality. First and foremost, I extend my heartfelt thanks to my respected teachers for their unwavering encouragement and invaluable advice throughout the project. Your insights provided the foundation for this work.

I sincerely thank the esteemed guide from the **Naan Mudhalvan** team, whose expert suggestions and feedback were instrumental in shaping the direction and quality of this project.

My gratitude also goes to the faculty members of my institution for their cheerful readiness to assist me at every stage, ensuring I stayed motivated and on track.

A special thanks to my classmates for their collaborative spirit, innovative ideas, and constant support during the development process. Your input greatly enriched this project.

I would also like to express my heartfelt thanks to my parents for their blessings and encouragement, which are the driving forces behind all my endeavors.

Lastly, I am grateful to everyone who, in any capacity, contributed directly or indirectly to the completion of this project. Your help and encouragement have been invaluable.

Thank You!

DECLARATION

I hereby declare that the project titled "**Online Learning Platform Using MERN**" has been submitted by our team as part of the assignment given by the **Naan Mudhalvan Upskilling Platform**.

This project is original and not a duplicate or copy of any existing work. All the information presented within is accurate and truthful to the best of our knowledge.

1. Introduction

- Project Title: Online Learning Platform.

Team Members:

- Sumit Kumar
- Vicky Kumar
- Prince Kumar Ram
- Rohit Kumar Pandey

The **Online Learning Platform Using MERN** is a comprehensive web application developed as part of the Naan Mudhalvan Upskilling Platform initiative. This project aims to redefine the way students access and engage with educational resources by leveraging the power of modern web technologies. Built on the robust MERN stack (MongoDB, Express.js, React.js, and Node.js), the platform offers a seamless user experience with features such as secure authentication, interactive learning modules, progress tracking, and real-time notifications.

The platform is designed to be user-centric, ensuring that learners can access diverse study materials, participate in quizzes, and collaborate within a vibrant learning community. By combining cutting-edge technology with educational best practices, this project aspires to contribute to the broader goal of empowering students and enhancing digital learning experiences.

Scenario:

A non-profit organization aims to improve education in rural areas by offering an online platform. They adopt the **Online Learning Platform Using MERN**, leveraging its features like secure login, multilingual support, progress tracking, and interactive learning.

Challenges:

- Limited access to study materials.
- Low student engagement.
- Difficulty monitoring learning progress.

Implementation:

- Teachers upload multilingual study resources.
- Students engage with interactive quizzes and group discussions.
- Teachers track progress via dashboards for personalized support.

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.

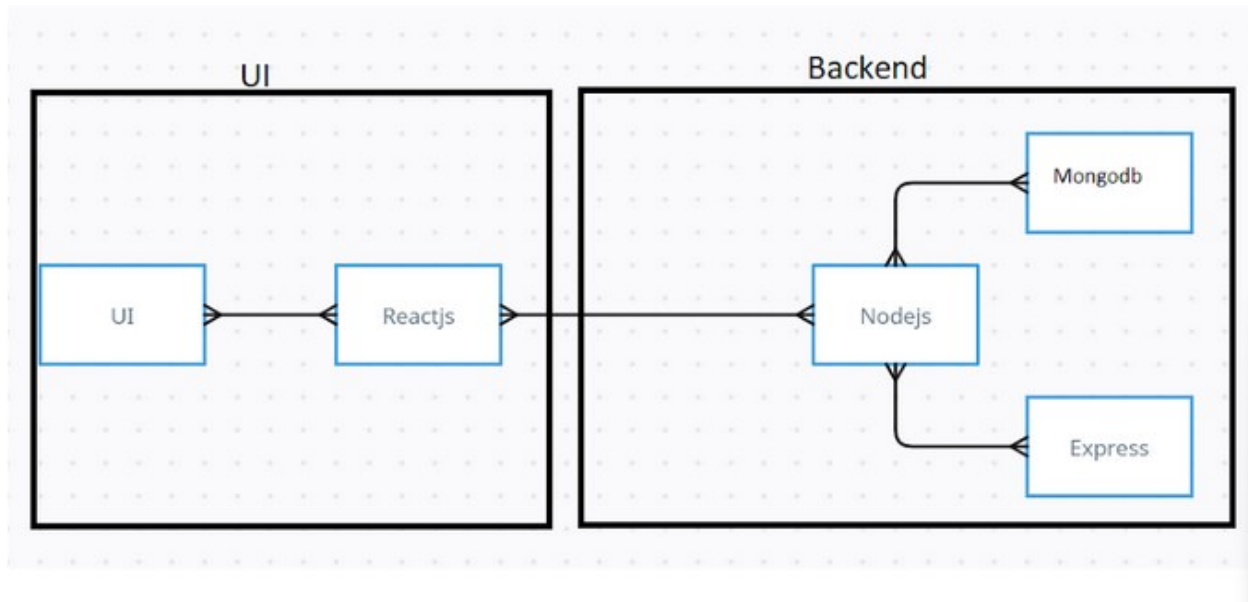
- **Admin Role:**

- 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.

Frontend Implementation (Client Side)

- **Framework:** Built with `React.js`, ensuring a responsive and dynamic user interface.
 - **Routing:** Utilized `React Router` for seamless navigation between pages like login, dashboard, and study material.
 - **State Management:** Implemented `Context API` or `Redux Toolkit` to manage user authentication and application state efficiently.
 - **API Communication:** Used `Axios` to interact with backend APIs for operations like fetching study materials or submitting quiz results.
 - **UI Design:** Styled with `CSS Modules` or `TailwindCSS` for a clean and modern look. Reusable components like `Navbar`, `Footer`, and `MaterialCard` streamline development.
-

Backend Implementation (Server Side)

- **Framework:** Developed with Express.js, providing a lightweight and flexible server framework.
- **Database:** MongoDB stores user data, study materials, and quiz results. Organized using Mongoose schemas for models like **User**, **Material**, and **Quiz**.
- **Authentication:** Implemented JWT (JSON Web Tokens) for secure user authentication, protecting private routes.
- **Endpoints:** Designed RESTful APIs to handle CRUD operations for users and materials, ensuring modularity and reusability.
 - **Example:**
POST /api/login for authentication.
GET /api/materials to fetch all study resources.
- **Middleware:** Added custom middleware for error handling and input validation using libraries like Joi.

Integration Between Client and Server

- The client communicates with the server via REST APIs, sending HTTP requests and receiving JSON responses.
- **Example workflow:**
 - A student logs in, and the frontend sends their credentials to the **/api/login** endpoint.
 - Upon successful authentication, a JWT is generated and stored in the client (usually in local storage or cookies).

Deployment

- **Frontend:** Deployed using **Netlify** or **Vercel** for a fast, scalable, and globally available user interface.
- **Backend:** Hosted on **Heroku** or **AWS EC2**, with the database deployed on **MongoDB Atlas** for high availability and reliability.
- **Environment Variables:** Managed using **.env** files for sensitive data like database connection strings or JWT secret

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`

`https://github.com/Sumitrazz/StudyApp-naan-mudhalvan-Project.git`

- 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>`

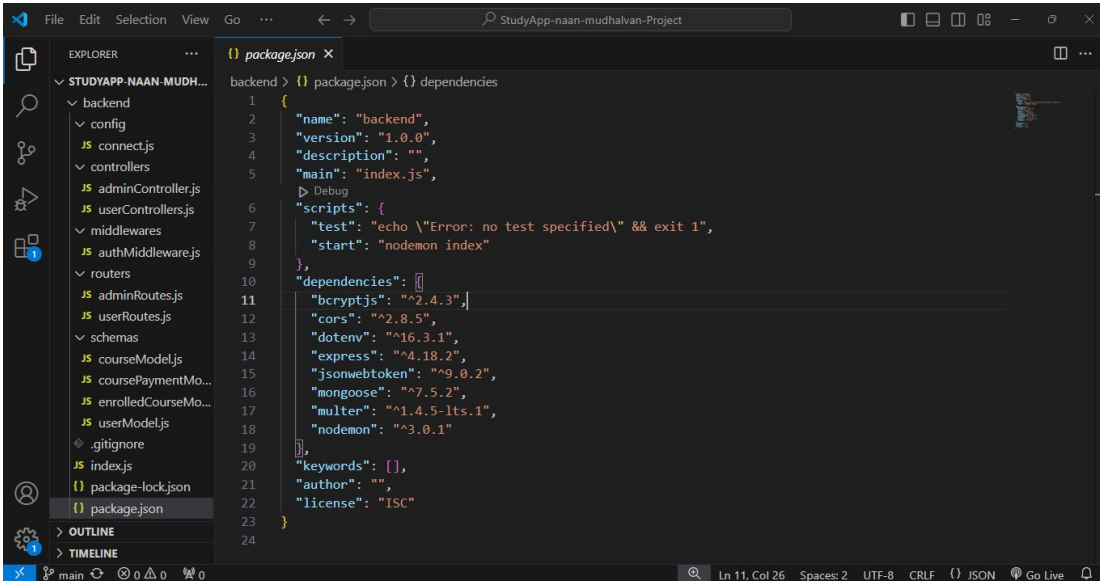
- Run the Development Servers:

- 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

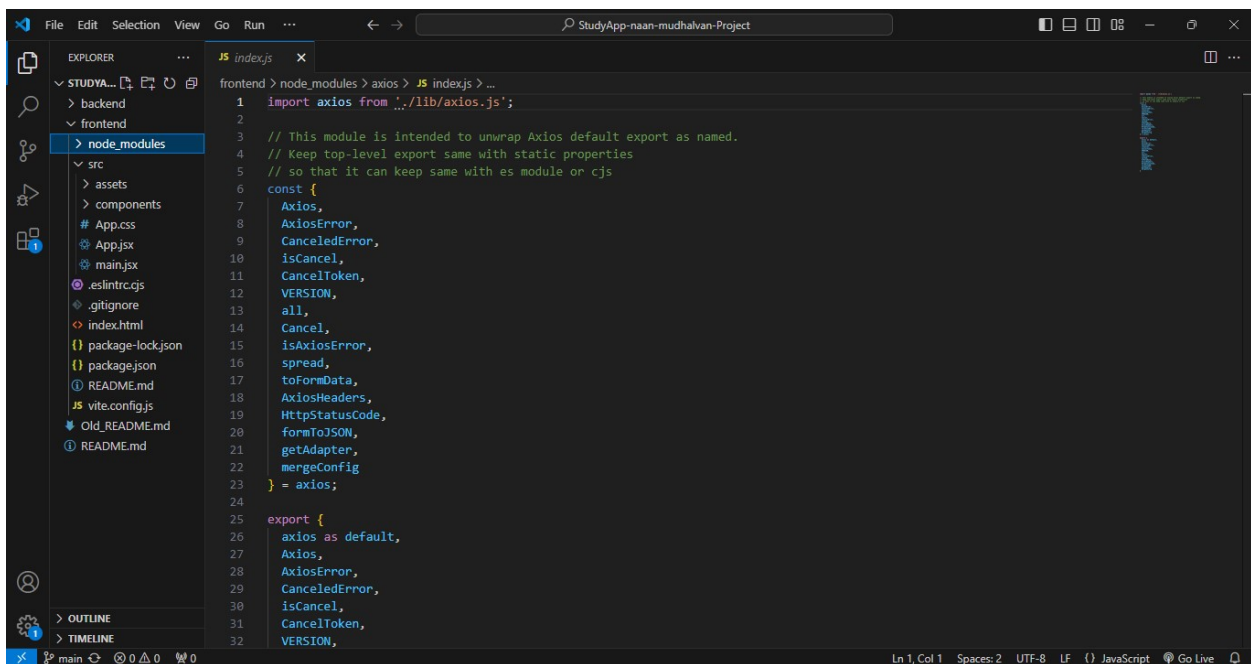
- Client:



The screenshot shows the VS Code editor with the 'package.json' file open in the 'backend' directory. The Explorer sidebar on the left shows the project structure, including 'config', 'connect.js', 'controllers', 'middlewares', 'routers', 'schemas', and 'index.js'. The main editor displays the following JSON content:

```
1 {  
2   "name": "backend",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \\\"Error: no test specified\\\" && exit 1",  
8     "start": "nodemon index"  
9   },  
10  "dependencies": {  
11    "bcryptjs": "^2.4.3",  
12    "cors": "^2.8.5",  
13    "dotenv": "^16.3.1",  
14    "express": "^4.18.2",  
15    "jsonwebtoken": "^9.0.2",  
16    "mongoose": "^7.5.2",  
17    "multer": "^1.4.5-lts.1",  
18    "nodemon": "^3.0.1"  
19  },  
20  "keywords": [],  
21  "author": "",  
22  "license": "ISC"  
23 }  
24
```

- Server:



The screenshot shows the VS Code editor with the 'index.js' file open in the 'frontend' directory. The Explorer sidebar on the left shows the project structure, including 'node_modules', 'src', 'assets', 'components', 'App.css', 'App.jsx', 'main.jsx', '.eslintrc.js', '.gitignore', 'index.html', 'package-lock.json', 'package.json', 'README.md', 'vite.config.js', 'Old_README.md', and 'README.md'. The main editor displays the following JavaScript code:

```
1 import axios from './lib/axios.js';  
2  
3 // This module is intended to unwrap Axios default export as named.  
4 // Keep top-level export same with static properties  
5 // so that it can keep same with es module or cjs  
6 const {  
7   Axios,  
8   axiosError,  
9   CancelError,  
10  isCancel,  
11  CancelToken,  
12  VERSION,  
13  all,  
14  Cancel,  
15  isAxiosError,  
16  spread,  
17  toFormData,  
18  AxiosHeaders,  
19  HttpStatusCode,  
20  formToJSON,  
21  getAdapter,  
22  mergeConfig  
23 } = axios;  
24  
25 export {  
26   axios as default,  
27   Axios,  
28   axiosError,  
29   CancelError,  
30   isCancel,  
31   CancelToken,  
32   VERSION,  
33 }
```

Server Folder Structure (Backend)

The server-side code, built with Node.js and Express.js, manages the application's APIs, business logic, and database interactions. Here's a typical structure:

1. **server.js**

- The entry point for the backend application.
- Configures the server, connects to the database, and sets up middleware and routes.
- Example contents:

```
const express = require('express');  
  
const mongoose = require('mongoose');  
  
const app = express();  
  
// Middleware and route setup
```

1. **routes/**

- Contains route definitions for handling different API endpoints.
- Example files:
 - **authRoutes.js**: Handles user authentication routes like login, signup, and logout.
 - **materialRoutes.js**: Manages endpoints for study materials (CRUD operations).

2. **controllers/**

- Implements the logic for handling requests received through routes.
- Example files:

- `authController.js`: Contains methods like `loginUser`, `registerUser`.
- `materialController.js`: Contains methods like `getAllMaterials`, `addMaterial`.

3. `models/`

- Contains MongoDB schemas and models for database collections.
- Example files:
 - `User.js`: Defines the schema for user data (e.g., name, email, password).
 - `Material.js`: Defines the schema for study materials.

4. `middleware/`

- Contains custom middleware for tasks like authentication or input validation.
- Example files:
 - `authMiddleware.js`: Verifies JWT tokens for protected routes.
 - `errorHandler.js`: Handles error responses consistently.

5. `config/`

- Stores configuration files, such as database connection logic.
- Example file:
 - `db.js`: Exports a function to connect to MongoDB.

6. `utils/`

- Contains utility functions used across the application.
- Example files:
 - `generateToken.js`: Creates JWT tokens.

7. `test/`

- Contains backend test cases written using frameworks like Mocha or Jest.
-

Client Folder Structure (Frontend)

The client-side code, built with React.js, manages the user interface and communicates with the server through APIs. Here's a typical structure:

1. **src/**

- The main directory for the React application.

2. **src/components/**

- Contains reusable UI components.
- Example components:
 - **Navbar.js**: Renders the navigation bar.
 - **MaterialCard.js**: Displays individual study material details.

3. **src/pages/**

- Contains full-page components for routing.
- Example files:
 - **HomePage.js**: Displays the homepage with features like a welcome message.
 - **LoginPage.js**: Contains the login form.
 - **Dashboard.js**: Displays user progress and study materials.

4. **src/hooks/**

- Contains custom React hooks for handling logic.
- Example:
 - **useAuth.js**: Manages authentication state.

5. **src/context/**

- Contains React Context files for global state management.
- Example:
 - `AuthContext.js`: Provides authentication context for components.

6. `src/redux/`

- If Redux Toolkit is used for state management, this folder holds slices and store setup.
- Example files:
 - `store.js`: Configures the Redux store.
 - `userSlice.js`: Handles user-related state.

7. `src/assets/`

- Stores static assets like images, icons, or fonts.

8. `src/utils/`

- Contains helper functions for tasks like API calls or formatting.
- Example:
 - `api.js`: Centralizes API endpoints.

9. `src/App.js`

- The main app component that handles routing and layout.

10. `public/`

- Contains public files like `index.html` or `favicon.ico`.

11. `src/styles/`

- Contains global styles or CSS modules.
-

6. Running the Application

- Installation: Follow these steps to set up the project on your local machine:
- Clone the Repository: `git clone https://github.com/Sumitrazz/StudyApp-naan-mudhalvan-Project.git`
- 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

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

Course Management

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

Enrollment

- Enroll in Course (Student only)
- Endpoint: /api/enrollments/:courseId
- Method: POST
- Response: Enrollment confirmation

- Get Enrolled Courses
- Endpoint: /api/enrollments
- Method: GET
- Response: List of enrolled courses

Certificates

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

Admin

- Get All Users
- Endpoint: /api/admin/users
- Method: GET
- Response: List of all users
- 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.

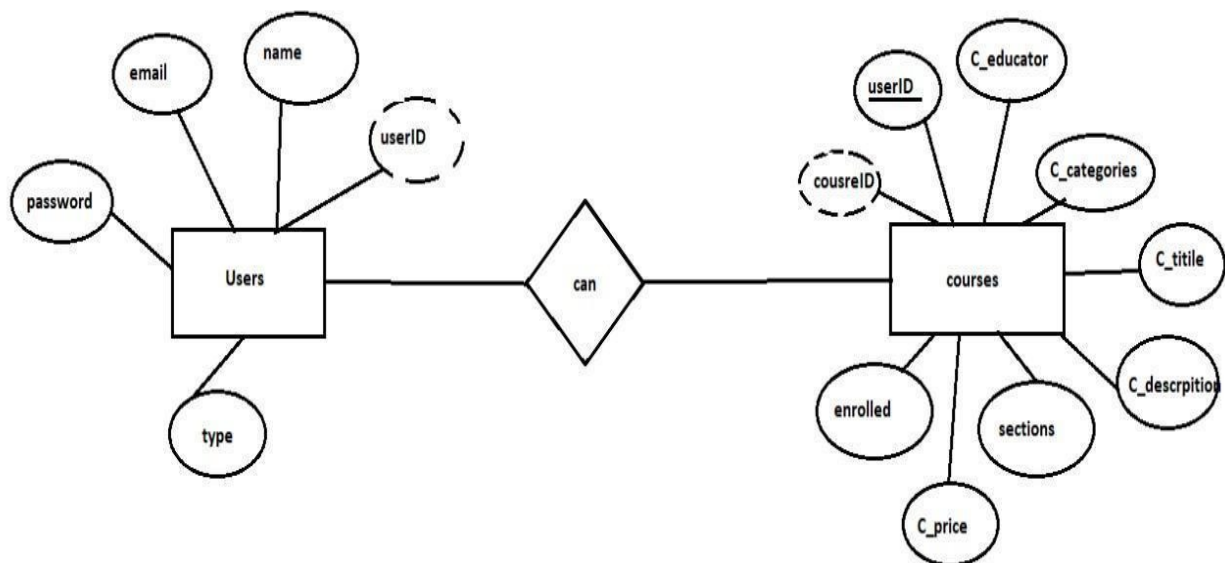
4. Role-based Access Control

- 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

ER-Diagram



Here there is 2 collections namely users, courses which have their own fields in

Users:

1. _id: (MongoDB creates by unique default)
2. name
3. email
4. password
5. type

Courses:

1. userID: (can act as a foreign key)
2. _id: (MongoDB creates by unique default)

3. C_educator
4. C_categories
5. C_title
6. C_description
7. sections
8. C_price
9. enrolled

10. Testing

Testing Strategy

- Unit Testing
- Goal: Test individual components and functions.
- Tools:
- Jest for functions and backend routes.
- React Testing Library for React component interaction.
- Integration Testing
- Goal: Ensure frontend and backend work together smoothly.
- Tools:
- Supertest for backend API requests.
- Jest to check data flow across components.
- 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.

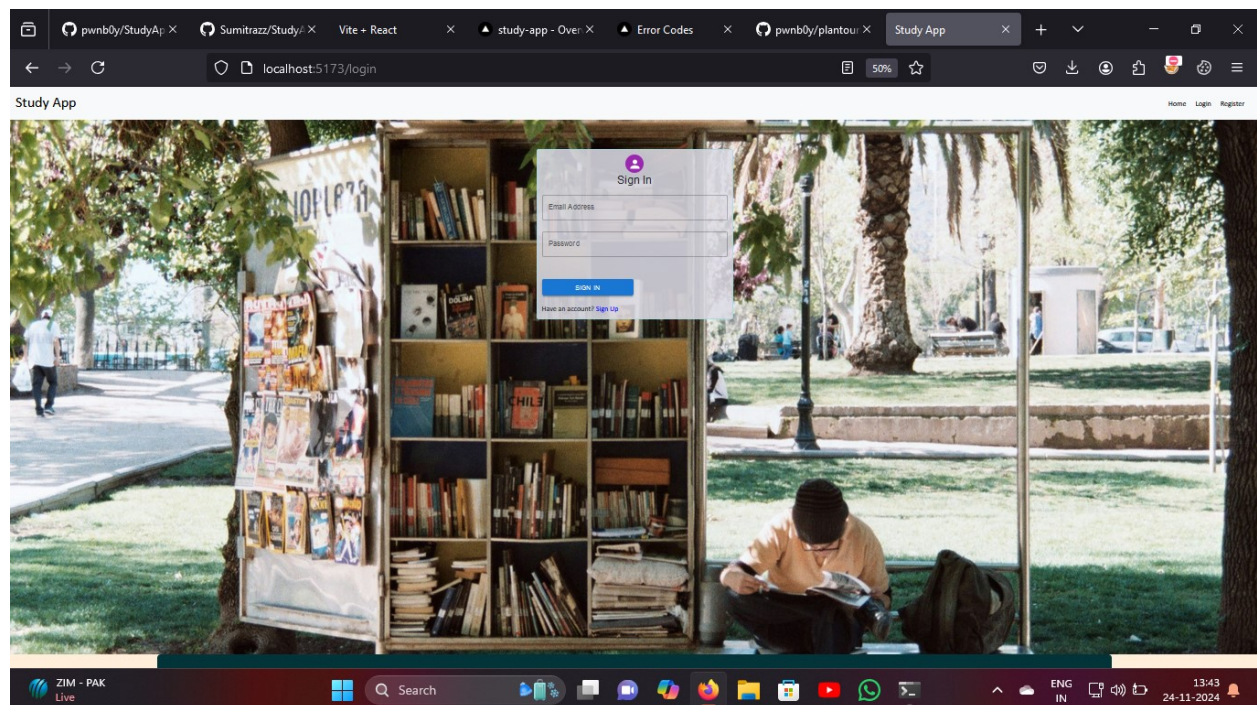
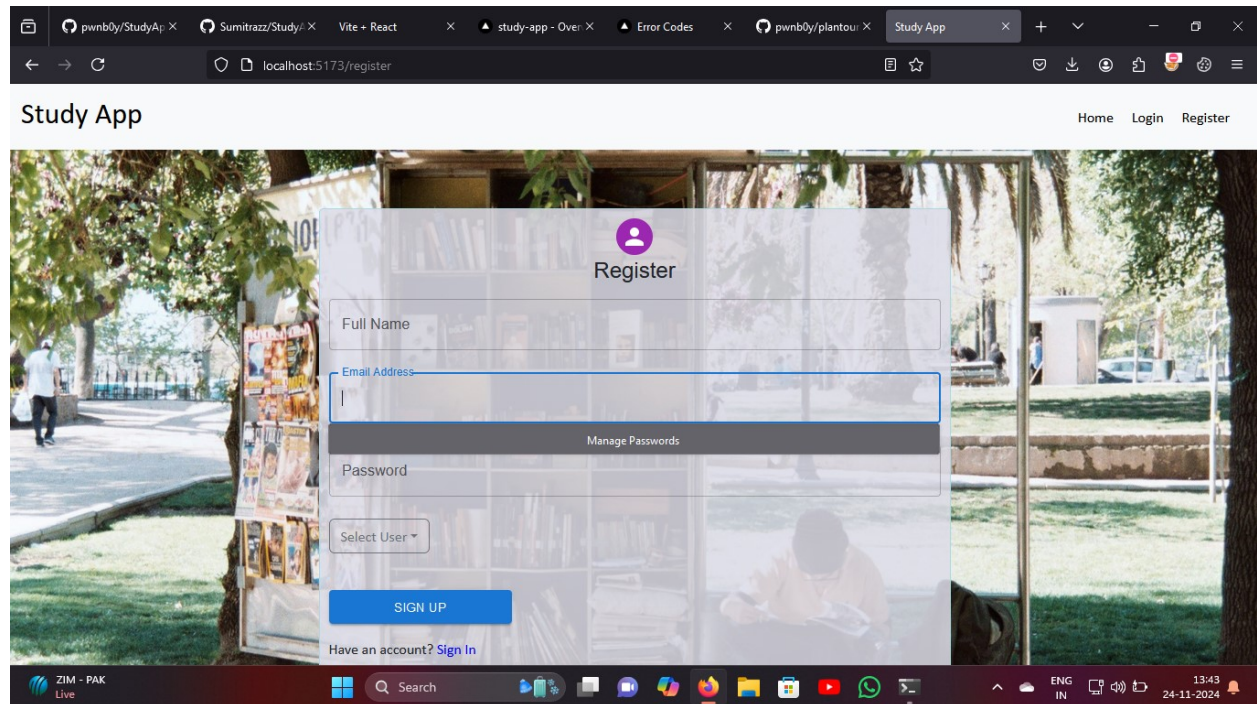
Example Tests

- 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.

11. Screenshots or Demo:



Study App

localhost:5173/login

50%

Why SimplyBook.me is an ideal fit for your educational and training businesses scheduling

Reminders

Don't let your students forget a date and time for a lesson or course. Remind them where they need to be in time to get there with automated and customisable reminders.

Calendar Sync

Don't let teaching and meeting times clash with your personal life and vacation time. Synchronise your personal and professional calendars to avoid double booking yourself! Each teacher can sync their personal calendars with their teaching schedules.

Intake Forms

If you need student or pupil information before they join your classes, you can import and create lesson specific intake forms to ensure they are taking the most appropriate course.

Kiosk

Let your students book themselves into available classes on site with a secure enhanced kiosk terminal. They can log in and join a class at your kiosk terminal, the system will forget all of their details and login information when they have finished.

Code Colours for Teachers

Create colour coordinated schedules for your teaching staff to more easily identify upcoming lessons, and overloaded schedules.

Group Booking

Allow companies and organisations to group book important courses and seminars for multiple staff members with the Group Bookings feature.

Daily Reports

Maintain detailed daily reports on your teaching schedule and external meeting, get it all in a report created for a daily overview of your trainer schedules.

Cancellation Policy

Don't get stuck with empty and unpaid seats in your classes that you can't fill. Ensure that your students know the cancellation policy and how long they need to give notice of non-attendance.

Study Session Cancellation Policy

To maintain an effective learning environment, it's important that students are aware of the cancellation policy for study sessions. Ensure that students understand how much notice they need to provide if they cannot attend a scheduled session, so that time can be allocated to others who may need it.

Teacher Cancellation Policy

To ensure smooth class management, it's crucial for teachers to communicate the cancellation policy clearly. Make sure students understand how much notice they need to provide if they cannot attend, allowing teachers to adjust their plans and accommodate other learners effectively.

[View All System Features](#)

[About Us](#)

LEARN ANYTHING

Benefits of Online Learning Expertise

Study App

localhost:5173/login

50%

LEARN ANYTHING

Benefits of Online Learning Expertise

Flexible Learning

Learn at your own pace with access to materials anytime, anywhere.

Expert Instructors

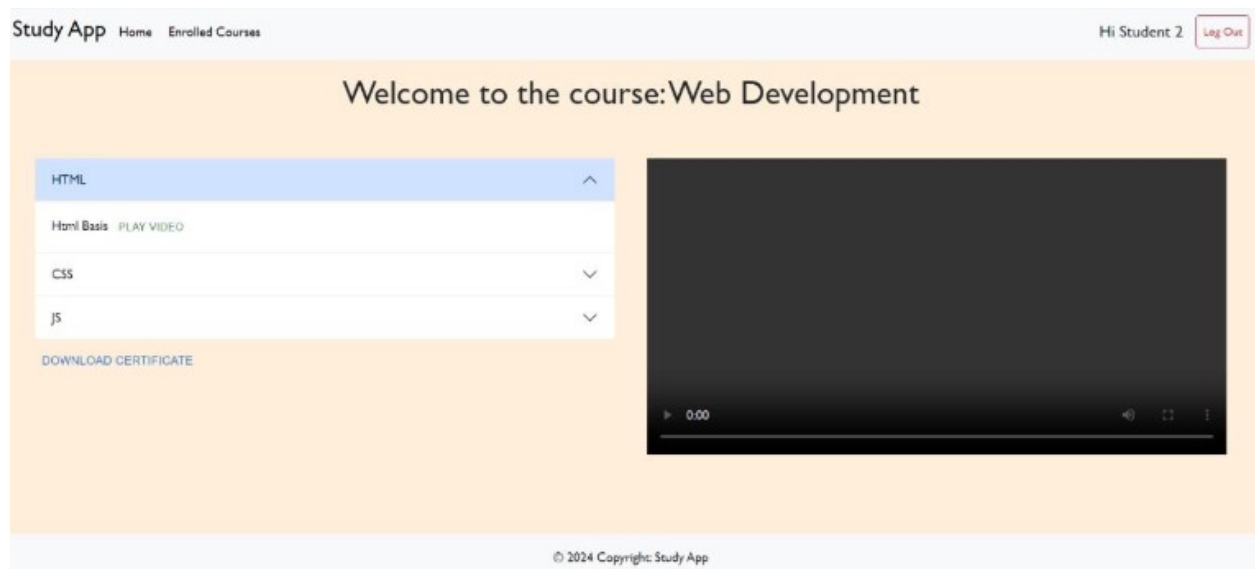
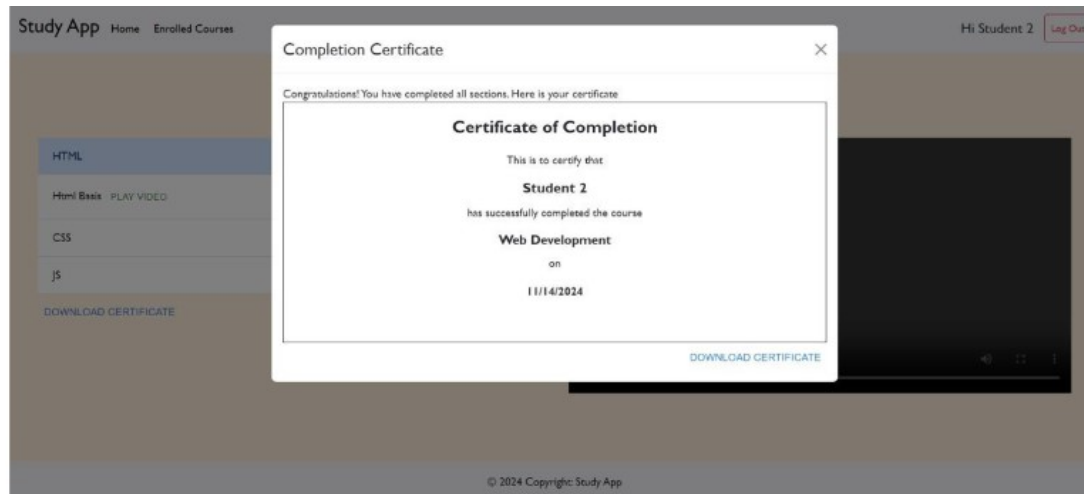
Learn from the best in the industry

StudyApp - Stay tuned and get the latest update

For far away behind the word mountains

Enter email address

© 2024 Copyright: Study App



● Demo Video link

 demo video studyApp.mp4

● Project Repo

<https://github.com/Sumitrazz/StudyApp-naan-mudhalvan-Project/tree/main>

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.