

NAAN MUDHALVAN PROJECT

MERN stack powered by MongoDB



PROJECT TITLE

Online Learning Platform using MERN Stack

Submitted by the team members of Final Year IT 'B'

SRIDEVI. S	311421205086
SUBASHINI. M	311421205087
SWETHA. B	311421205091
UMA MAHESHWARI. K	311421205098



DEPARTMENT OF INFORMATION TECHNOLOGY
MEENAKSHI COLLEGE OF ENGINEERING
12, VEMBULIAMMAN KOVIL STREET, WEST K.K. NAGAR,
CHENNAI - 600 078, TAMILNADU.
(AFFILATED TO ANNA UNIVERSITY)

Online Learning Platform Using MERN Stack

ABSTRACT :

The **Online Learning Platform (OLP)** is a comprehensive educational solution leveraging the MERN stack (MongoDB, Express, React, Node.js) to provide accessible, flexible, and user-friendly learning opportunities for students worldwide. OLP incorporates a suite of features designed to streamline online education, including course management, interactive learning tools, and flexible accessibility across devices.

Through a client-server architecture, OLP enables real-time data exchange and supports robust functionalities such as user authentication, role-based access, and payment processing for premium content. By offering a seamless and engaging user experience, this platform empowers students to learn at their own pace, interact with peers and instructors, and receive certifications upon course completion.

The platform also supports instructors in creating and managing content and enables administrators to oversee the system effectively. This documentation details the requirements, architecture, features and implementation steps necessary to develop the OLP using MERN, serving as a foundational guide for an effective online education ecosystem.

Keywords: Online Education, Web Application, MERN Stack, Vite

TABLE OF CONTENTS

ABSTRACT	2
1. INTRODUCTION	5
1.1. PROJECT TITLE	5
1.2. TEAM MEMBERS & THEIR ROLES	5
2. PROJECT OVERVIEW	5
2.1. PURPOSE	6
2.2. FEATURES	6
3. SYSTEM REQUIRMENTS	7
3.1. HARDWARE	7
3.2. SOFTWARE	7
3.3. NETWORK	7
4. ARCHITECTURE	7
4.1. TECHNICAL ARCHITECTURE	8
5. ER - DIAGRAM	8
a. USERS ENTITY	9
b. COURSES ENTITY	9
c. RELATIONSHIP - “CAN”	9
6. SETUP INSTRUCTIONS	10
A. FRONT-END DEVELOPMENT	10
B. BACK-END DEVELOPMENT	11
C. DATABASE DEVELOPMENT	12
6.1. PRE-REQUISITES	12
6.2. INSTALLATION	12
7. PROJECT FOLDER STRUCTURE	17
7.1. FRONT - END (CLIENT)	17
7.2. BACK - END (SERVER)	19
8. APPLICATION FLOW	23
I. Teacher	23
II. Student	23
III. Admin	23
9. RUNNING THE APPLICATION	23
10. API DOCUMENTATION	24
11. PROJECT IMPLEMENTATION :	37
12. AUTHENTICATION AND AUTHORIZATION:	59

13. USER INTERFACE SCREENSHOTS :	61
14. TESTING :	65
15. SCREENSHOTS OR DEMO LINK :	70
16. KNOWN ISSUES :	70
17. FUTURE ENHANCEMENT :	72
18. CONCLUSION :	74

1. INTRODUCTION :

1.1. PROJECT TITLE :

Online Learning Platform Using MERN Stack

1.2. TEAM MEMBERS & THEIR ROLE :

- **Sridevi. S - Role:** Project Lead & Front end Developer, responsible for coordinating the team, overseeing project milestones, and ensuring timely completion of deliverables and also focusing on designing & implementing the user interface using React & Material UI.
- **Subashini. M - Role:** Frontend Developer, focusing on designing & implementing the user interface using React & Material UI for an engaging and intuitive user experience.
- **Swetha. B - Role:** Backend Developer, responsible for building RESTful APIs, managing server-side functionality & ensuring data security using Node.js & Express.
- **Uma Maheshwari. K - Role:** Database Administrator, in charge of managing of the MongoDB database, handling data storage& ensuring efficient data retrieval & integrity.

2. PROJECT OVERVIEW :

In recent years, the demand for online education has surged, driven by the need for flexible, accessible, and diverse learning options that cater to different schedules, locations, and learning preferences. An **Online Learning Platform (OLP)** is a digital system designed to meet this demand, providing a comprehensive educational environment that allows users to engage with content, track progress, and gain certification for completing courses. Built using the MERN stack, OLP leverages MongoDB for data storage, Express.js for server-side functionality, React for a dynamic front-end interface, and Node.js for back-end development, ensuring a scalable, efficient, and responsive learning platform.

OLP is built with a focus on accessibility and interactivity, allowing learners of all backgrounds and technical proficiency to navigate the platform easily. It supports features like course enrollment, self-paced learning, discussion forums, live webinars, and progress tracking. Moreover, the platform offers flexibility for both free and paid courses, catering to broad audience.

With roles for students, instructors, and administrators, OLP provides a structured yet adaptable framework for managing and delivering educational content, enhancing the learning experience, and supporting instructors in content management and user engagement. This documentation outlines the technical architecture, functional requirements, and user workflows of

OLP, emphasizing how it integrates with modern educational needs to foster a robust online learning community.

2.1. PURPOSE :

The purpose of the **Online Learning Platform (OLP)** is :

- ❖ To provide a centralized online platform that facilitates flexible, self-paced learning, accessible to users worldwide.
- ❖ To enable users to enroll in courses, track learning progress, and earn certifications, supporting personal and professional development.
- ❖ To offer an interactive and user-friendly interface that allows learners to navigate content easily and engage in discussion forums, live webinars, and assignments.
- ❖ To provide instructors with a streamlined system for creating, organizing, and managing course content effectively.
- ❖ To allow administrators to monitor platform operations, manage user activities, and ensure data security and integrity.
- ❖ To create a scalable, efficient, and cost-effective solution for educational institutions and independent educators looking to deliver content online.

2.2. FEATURES :

The feature of the **Online Learning Platform (OLP)** encompasses the development, deployment, and maintenance of a comprehensive online education system using the MERN stack. Key components and features within this feature includes :

- ❖ **User Management:** Support user roles for students, instructors, and administrators, including registration, login, and profile management.
- ❖ **Course Management:** Enable instructors to create, update, organize, and publish course materials such as video lectures, assignments, and reading materials.
- ❖ **Interactivity Tools:** Offer discussion forums, chat rooms, and live webinar functionality to foster communication between students and instructors.
- ❖ **Progress Tracking:** Implement features for learners to monitor their course progress and revisit content as needed.
- ❖ **Certification:** Issue digital certificates upon course completion to recognize student achievement and skill acquisition.
- ❖ **Payment and Subscription Options:** Provide a payment gateway for paid courses, offering both one-time purchases and subscription models.

- ❖ **Cross-Device Accessibility:** Ensure compatibility across various devices (PCs, tablets, smartphones) to allow access from any location with an internet connection.
- ❖ **Front-end & Back-end Integration:** Leverage MERN stack for efficient front-end and back-end communication and database management.
- ❖ **Admin Panel:** Include an administrative dashboard for monitoring user activity, managing course listings, and handling platform maintenance tasks.

3. SYSTEM REQUIRMENTS :

3.1. HARDWARE :

- ❖ **Operating System** : Windows 8 or higher
- ❖ **RAM** : 4 GB or more (8 GB recommended for smooth development experience)

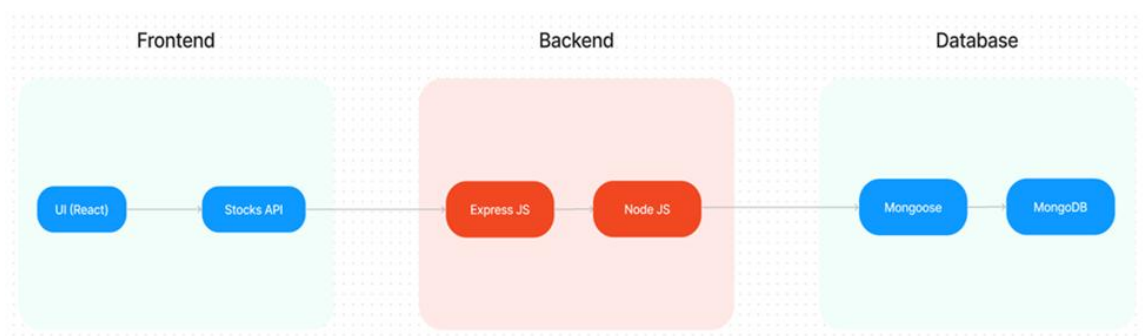
3.2. SOFTWARE :

- ❖ **Node.js** : LTS version for back-end and front-end development
- ❖ **MongoDB** : For database management using MongoDB Atlas or a local instance
- ❖ **React** : For front-end framework
- ❖ **Express.js** : For back-end framework
- ❖ **Git** : Version control
- ❖ **Code Editor** : e.g., Visual Studio Code
- ❖ **Web Browsers** : Two web browsers installed for testing compatibility (e.g., Chrome and Firefox)

3.3. NETWORK :

- ❖ **Bandwidth** : 30 Mbps

4. ARCHITECTURE :



4.1. TECHNICAL ARCHITECTURE :

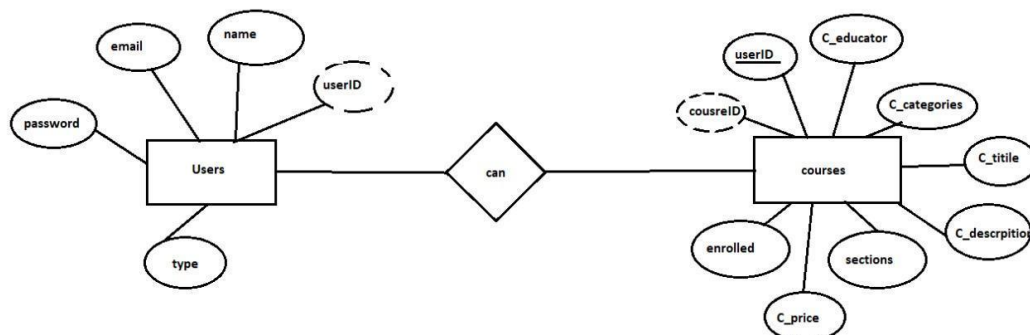
The technical architecture of OLP app follows a **client-server model**, where the **front-end serves as the client and the back-end acts as the server**. The front-end encompasses not only the user interface and presentation but also incorporates the axios library to connect with back-end easily by using RESTful Api's.

On the **front-end side**, it utilizes the bootstrap and material UI library to establish real-time and better UI experience for any user. On the **back-end side**, we employ Express.js frameworks to handle the server-side logic and communication. For data storage and retrieval, our back-end relies on MongoDB. MongoDB allows for efficient and scalable storage of user data and necessary information about the place.

For communication between the front-end and back-end, **RESTful API's** are implemented, facilitating smooth data exchange and modular integration. Additionally, **authentication and authorization** protocols, such as JWT (JSON Web Tokens), are used to secure user access, ensuring role-based permissions for students, instructors, and admins. The platform includes **real-time communication channels** like chat and messaging, fostering interaction between students and faculty, while **adaptive learning algorithms** provide personalized learning paths.

For analytics, **data visualization tools** are incorporated to display metrics on course engagement and completion, aiding administrators in making data-driven decisions. Finally, third-party integration, such as **payment gateways** for premium content and **video streaming services** for course delivery, enhance the platform's functionality, while systematic updates and **security audits** maintain platform integrity and user data protection. This comprehensive set of technical components enables the OLP to offer a secure, scalable, and user-friendly online learning experience.

5. ER - DIAGRAM :



Here there is 2 collections which have their own fields in :

- **USERS**
- **COURSES**

a. USERS ENTITY

- ✓ Represents the users of the platform.

Attributes:

- ❖ **userID:** Unique identifier for each user (likely primary key).
- ❖ **name:** Name of the user.
- ❖ **email:** Email address associated with the user account.
- ❖ **password:** User's password for secure login.
- ❖ **type:** Indicates the type of user (e.g., student, educator, admin).

b. COURSES ENTITY

- ✓ Represents the various courses available on the platform.

Attributes:

- ❖ **courseID:** Unique identifier for each course.
- ❖ **C_title:** Title of the course.
- ❖ **C_description:** Description of the course content.
- ❖ **C_educator:** The educator or instructor associated with the course.
- ❖ **C_categories:** Categories or tags to classify the course (e.g., Programming, Design).
- ❖ **sections:** Indicates sections or modules within the course.
- ❖ **C_price:** Price of the course, which could relate to whether it's a free or premium course.
- ❖ **enrolled:** Tracks the number of students or users enrolled in the course.

c. RELATIONSHIP - "CAN"

- ❖ This relationship shows the interaction between Users and Courses.
- ❖ **Users (students) can:**
 - Browse courses.
 - Enroll in courses.
 - Track progress in courses they are enrolled in.
- ❖ **Users (educators) can:**
 - Create, edit, and manage courses.

- Upload course content such as videos, text, and interactive elements.

❖ **Admin users can:**

- Oversee all user activities.
- Manage course listings, content, and enrollment data.

Summary in Use Case Points:

- ❖ **User Registration/Login:** Users can register and log in with a unique userID, email, and password.
- ❖ **Course Browsing and Enrollment:** Users can browse through available courses and enroll.
- ❖ **Course Management for Educators:** Educators can create and manage courses, add content, and handle student enrollments.
- ❖ **Content Interaction:** Enrolled students can interact with course materials and progress through sections/modules.
- ❖ **Payment Handling:** Courses may have prices, allowing for paid enrollment.
- ❖ **Data Tracking:** The platform tracks user enrollments and progress in each course.

6. SETUP INSTRUCTIONS :

❖ **Folder Setup**

- Create Frontend and Backend folders

A. FRONT-END DEVELOPMENT :

For Frontend, we use require dependencies as follows, they are:

- ❖ React
- ❖ Material UI
- ❖ Bootstrap
- ❖ React - bootstrap
- ❖ Axios
- ❖ Antd
- ❖ Mdb-react-ui-kit

```

{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@mui/icons-material": "^5.14.9",
    "@mui/material": "^5.14.9",
    "axios": "^1.5.0",
    "bootstrap": "^5.3.2",
    "html2canvas": "^1.4.1",
    "jspdf": "^2.5.1",
    "mdb-react-ui-kit": "^6.1.0",
    "mdb-ui-kit": "^6.4.0",
    "react": "^18.2.0",
    "react-bootstrap": "^2.8.0",
    "react-dom": "^18.2.0",
    "react-player": "^2.13.0",
    "react-router-dom": "^6.16.0"
  },
  "devDependencies": {
    "@types/react": "^18.2.15",
    "@types/react-dom": "^18.2.7",
    "@vitejs/plugin-react": "^4.0.3",
    "eslint": "^8.45.0",
    "eslint-plugin-react": "^7.32.2",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.3",
    "vite": "^4.4.5"
  }
}

```

B. BACK-END DEVELOPMENT :

❖ Setup express server

- Create index.js file in the server (backend folder).
- define port number, mongodb connection string and JWT key in env file to access it.
- Configure the server by adding cors, body-parser.

❖ Add authentication:

- For this you need to make middleware folder and in that make authMiddleware.js file for the authentication of the projects and can use in.

For Backend, we use require dependencies as follows, they are:

- ❖ Cors
- ❖ Bcryptjs
- ❖ Multer
- ❖ dotenv
- ❖ Express
- ❖ Mongoose
- ❖ Nodemon
- ❖ Jsonwebtoken

```

{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▶ Debug
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "nodemon index"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^7.5.2",
    "multer": "^1.4.5-lts.1",
    "nodemon": "^3.0.1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

```

C. DATABASE DEVELOPMENT :

❖ **Configure MongoDB**

- Import Mongoose
- Add database connection from config.js file present in config folder.
- Create a model folder to store all the DB schemas.

6.1. PRE-REQUISITES :

Here are the key prerequisites for developing a full-stack application. They are :

- ❖ **Vite + React**
- ❖ **Express**
- ❖ **Node.js**
- ❖ **Mongo DB**
- ❖ **Git**

6.2. INSTALLATION :

✓ **Vite:**

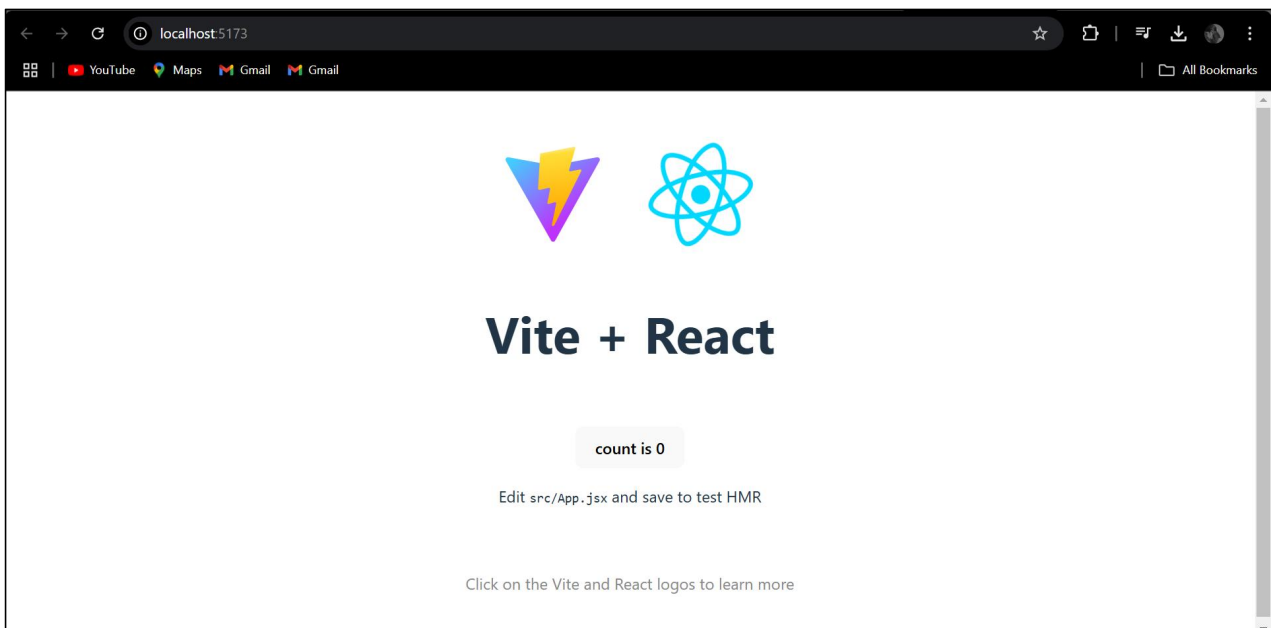
Vite is a new frontend build tool that aims to improve the developer experience for development with the local machine, and for the build of optimized assets for production (go live).

Vite (or ViteJS) includes: a development server with ES _native_ support and Hot Module Replacement; a build command based on rollup.

Installation : **npm create vite@latest**

```
Windows PowerShell
C:\Users\sivas>npm create vite@latest
> npx
> create-vite
Project name: ... frontend
Select a framework: » React
Select a variant: » JavaScript
Scaffolding project in C:\Users\sivas\frontend...
Done. Now run:
  cd frontend
  npm install
  npm run dev
C:\Users\sivas>cd frontend
C:\Users\sivas\frontend>npm install
added 248 packages, and audited 249 packages in 15s
101 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities
C:\Users\sivas\frontend>npm run dev
> frontend@0.0.0 dev
> vite

VITE v5.4.10 ready in 475 ms
➔ Local: http://localhost:5173/
➔ Network: use --host to expose
➔ press h + enter to show help
h
Shortcuts
press r + enter to restart the server
press u + enter to show server url
press o + enter to open in browser
press c + enter to clear console
press q + enter to quit
q
```



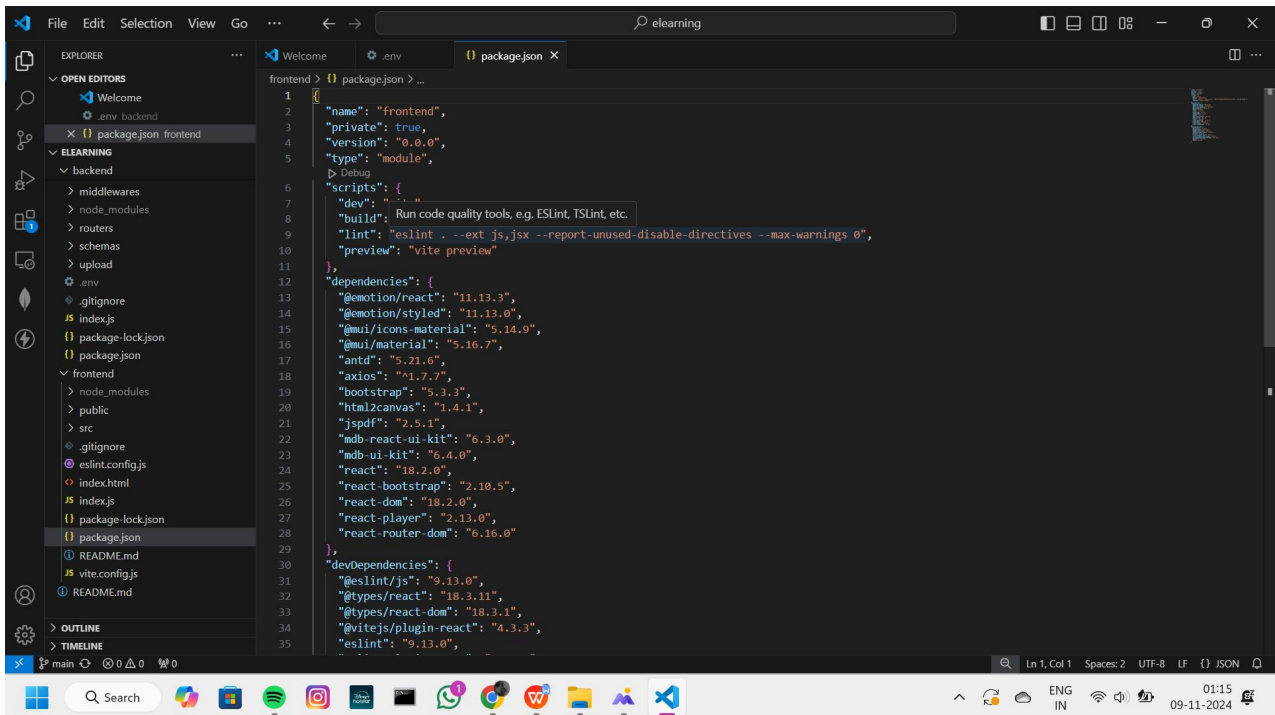
✓ Node.js and npm:

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications. Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

Download: <https://nodejs.org/en/download/>

Installation instructions: <https://nodejs.org/en/download/package-manager/>

Run “**npm init**” to get default dependencies

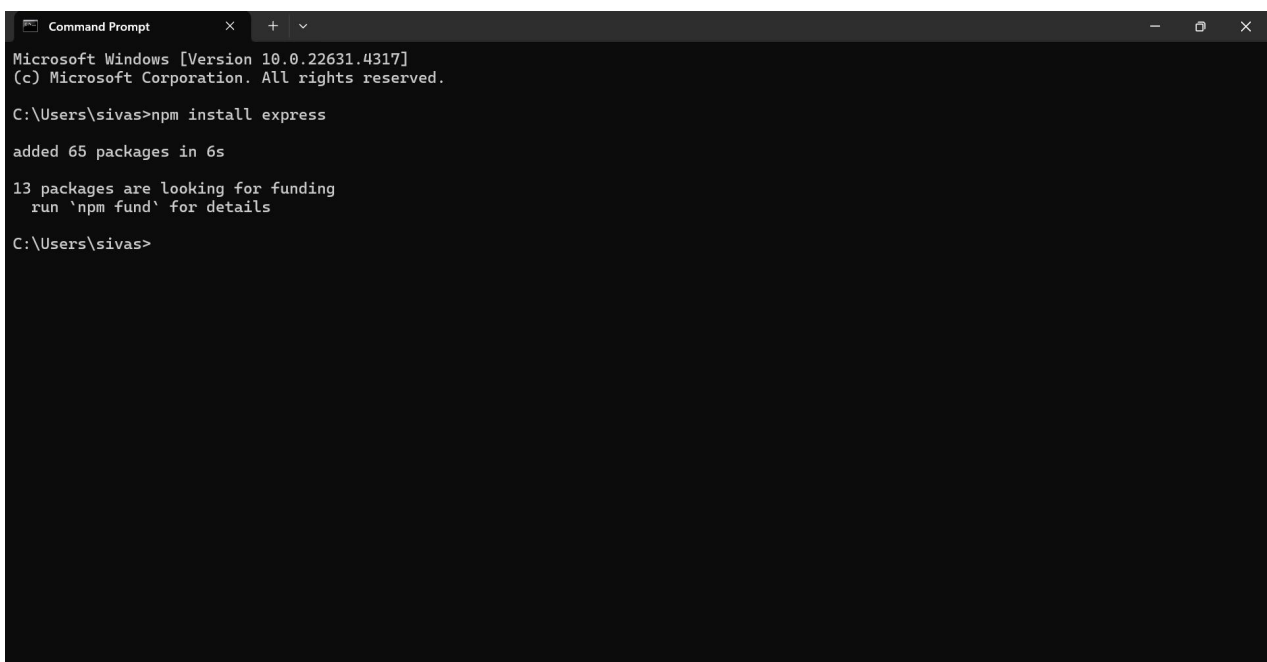


✓ Express.js:

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture. Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.

Installation: Open your command prompt or terminal and run the following command:

npm install express



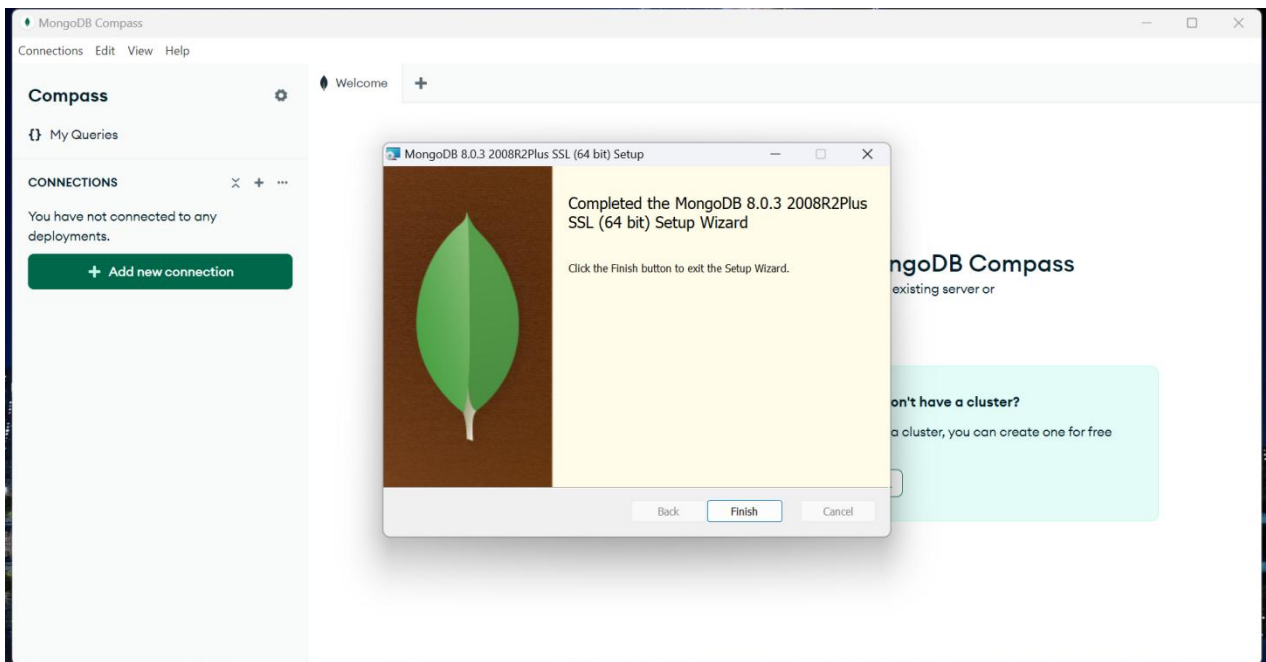
✓ MongoDB:

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

Download: <https://www.mongodb.com/try/download/community>

Installation instructions: <https://docs.mongodb.com/manual/installation/>



✓ React.js:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

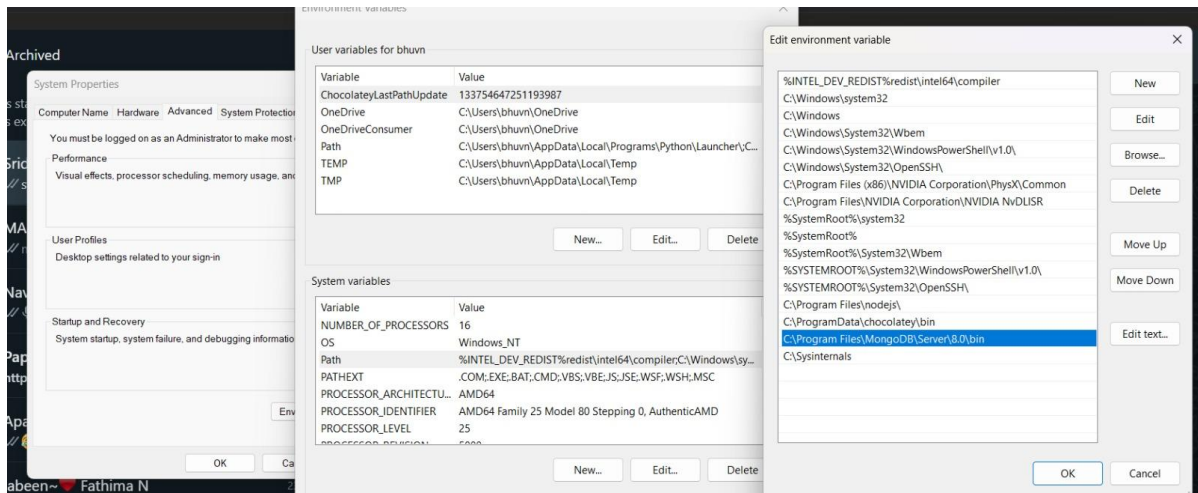
Follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

✓ HTML, CSS, and JavaScript:

Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

✓ Database Connectivity:

Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations. Before connecting you should set up an environment variable



✓ Front-end Framework:

Utilize React.js to build the user-facing part of the application, including entering booking room, status of the booking, and user interfaces for the admin dashboard. For making better UI we have also used some libraries like material UI and bootstrap.

Install the required dependencies by running the following commands:

cd frontend || npm install

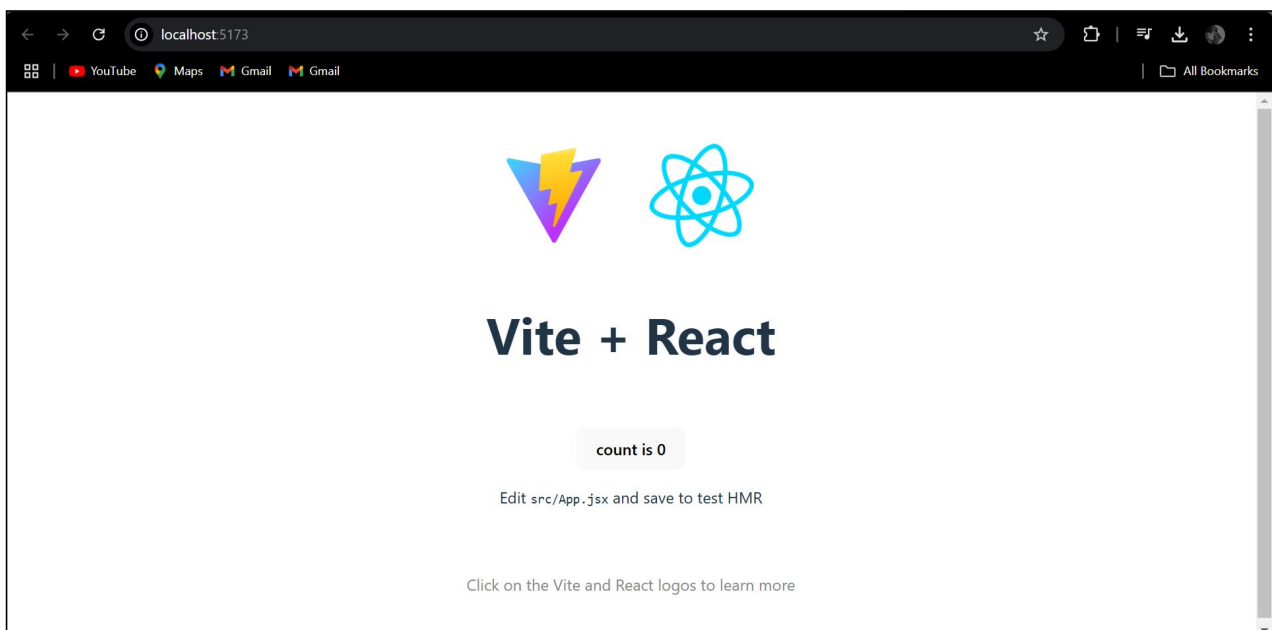
cd ../backend || npm install

Start the Development Server:

- To start the development server, execute the following command:

npm start

- The OLP app will be accessible at <http://localhost:5172>



The installation is successfully installed and set up the Online learning app on your local machine. You can now proceed with further customization, development, and testing as needed.

7. PROJECT FOLDER STRUCTURE :

7.1. FRONT - END (CLIENT) :

The first image is of front-end part which shows all the files and folders that have been used in UI development.

➤ Root Folders and Files :

❖ **Front-end:**

The main project folder containing all files and folders for the front-end part of the application.

❖ **node_modules:**

Contains all the dependencies and modules installed via npm (Node Package Manager). This folder is automatically generated when dependencies are installed.

❖ **public:**

Stores static files that can be served directly, such as index.html (the main HTML file for a React app), images, and other static assets. Files in this folder are accessible directly by the browser.

❖ **src:**

This is the main source folder for the React components and other application code. All core logic, components, and styles are stored here. Folders and Files Inside src contains :

➤ **assets:**

This folder typically contains images, icons, or other media assets used throughout the frontend.

➤ **components:**

Houses all the components used in the application. Components are organized into subfolders based on functionality or user roles (e.g., admin, common, user, student, teacher).

● **admin Folder :**

Contains components specific to the admin functionalities of the application.

- a. **AdminHome.jsx**: The main dashboard or homepage component for admin users.

- b. **AllCourses.jsx**: Displays all courses available on the platform, likely for administrative oversight.

- **common Folder :**

Contains shared components accessible by all users.

- a. **AllCourses.jsx**: Shows a list of all available courses (possibly a common view accessible by all users).
- b. **AxiosInstance.jsx**: Configures Axios for API requests, likely setting up base URLs or authorization headers.
- c. **Dashboard.jsx**: A general dashboard component that could serve as the main page for logged-in users.
- d. **Home.jsx**: The homepage or landing page for the application.
- e. **Login.jsx**: The login component for user authentication.
- f. **NavBar.jsx**: The navigation bar component, providing links to different parts of the application.
- g. **Register.jsx**: The registration component for new users.
- h. **UserHome.jsx**: The main page or dashboard for general users.

- **user Folder :**

Contains subfolders for different user roles with specific components

- **student:**

- a. **CourseContent.jsx**: Displays the content of a course for students.
- b. **EnrolledCourses.jsx**: Lists the courses a student is enrolled in.
- c. **StudentHome.jsx**: The main page or dashboard for students.

- **teacher:**

- a. **AddCourse.jsx**: Component for teachers to add or create new courses.
- b. **TeacherHome.jsx**: The main page or dashboard for teachers, showing relevant information for instructors.

➤ **App.css:**

Contains global styles for the application, defining the look and feel of the entire frontend.

➤ **App.jsx:**

The root component of the application. It usually contains routes to various pages and loads other major components.

➤ **main.jsx:**

The main entry point for the application, where the React app is rendered into the HTML document (usually index.html in the public folder).

➤ **.eslintrc.js:**

Configuration file for ESLint, a tool used to enforce coding standards and style in JavaScript code.

This structure is well-organized, with separate folders for different user roles (admin, student, teacher), as well as shared components for common functionality. Each user role has its specific set of components to encapsulate role-based features, making it easier to manage and extend the code. This setup supports scalability and maintainability for a comprehensive web application.

7.2. BACK - END (SERVER) :

The second image is of Backend part which is showing all the files and folders that have been used in backend development

➤ **Folder and File Structure**

❖ **Config:**

- This folder usually contains configuration files for setting up connections or other environmental variables needed by the application.
- It likely has settings for connecting to the database, such as database URL and credentials, along with any other application-level configurations.

❖ **Controllers:**

- **adminController.js**: Contains functions to handle admin-specific actions, such as managing courses, users, or payments.
- **userControllers.js**: Manages actions related to regular users, such as registration, login, enrolling in courses, or fetching user-specific information.

❖ **Middlewares:**

- **authMiddleware.js**: Contains middleware for handling authentication and authorization. It checks if users are authenticated before allowing access to certain routes, ensuring only logged-in users can access restricted areas of the application.

❖ **Routers:**

- **adminRoutes.js**: Defines API endpoints specific to admin functionalities and maps each endpoint to corresponding controller functions in adminController.js.
- **userRoutes.js**: Defines API endpoints for user functionalities and maps them to the functions in userControllers.js. This could include endpoints for login, registration, course enrollment, etc.

❖ **Schemas:**

- This folder includes all the models that define the structure of different collections or tables in the database.
- **courseModel.js**: Defines the schema for courses, including properties like title, description, price, educator, and other relevant course details.
- **coursePaymentModel.js**: Handles the schema for course payment details, including fields for transaction IDs, payment status, amount, etc.
- **enrolledCourseModel.js**: Manages information on users who have enrolled in specific courses, likely linking users to course IDs.
- **userModel.js**: Defines the user schema, including fields like name, email, password, and role (e.g., student or teacher).

❖ **uploads**

- This folder may be used for storing files uploaded by users, such as profile pictures or course materials. It ensures that any uploaded assets are kept in an organized directory.

❖ **Environment and Configuration Files**

- **.env**: Stores environment variables such as database credentials, API keys, and other sensitive information that should not be hard-coded.
- **.gitignore**: Specifies files and folders to be ignored by Git, such as node_modules, .env, and other directories that do not need to be tracked.

❖ Entry Points and Package Files

- **index.js**: The main entry point for the backend application. It typically sets up the server, connects to the database, and initializes middleware, routes, and error handling.
- **package.json**: Lists dependencies, scripts, and metadata for the project. It manages the backend libraries and tools required to run the server.
- **package-lock.json**: Records the exact versions of dependencies installed in node_modules, ensuring consistency across installations

This backend structure supports a modular approach to handling an online platform, where different sections are divided by purpose. Controllers handle specific actions, routes define accessible API endpoints, and middlewares enforce security and validation. Models structure the database, and configurations ensure easy setup across environments.

Here's an basic explanation for the third image which it includes:

❖ .gitignore

- Specifies files and directories that should be ignored by Git, preventing them from being tracked in version control. Common entries include node_modules, .env, and any other files that are either sensitive or unnecessary for sharing in the repository.

❖ index.html

- The main HTML file for the application. It serves as the entry point for the front-end application, where JavaScript bundles and stylesheets are injected. Typically, this file includes a root <div> where the front-end framework (e.g., React) mounts the application.

❖ package-lock.json

- Automatically generated by npm to track the exact versions of dependencies and sub-dependencies installed in the project. This file ensures that other developers or environments installing the project will have the exact same dependency versions, which improves consistency and reliability.

❖ package.json

- The main configuration file for the Node.js project. It includes metadata about the project (like its name and version), as well as scripts, dependencies, and other configurations. This file is essential for installing and managing packages and for running tasks like npm start or npm build.

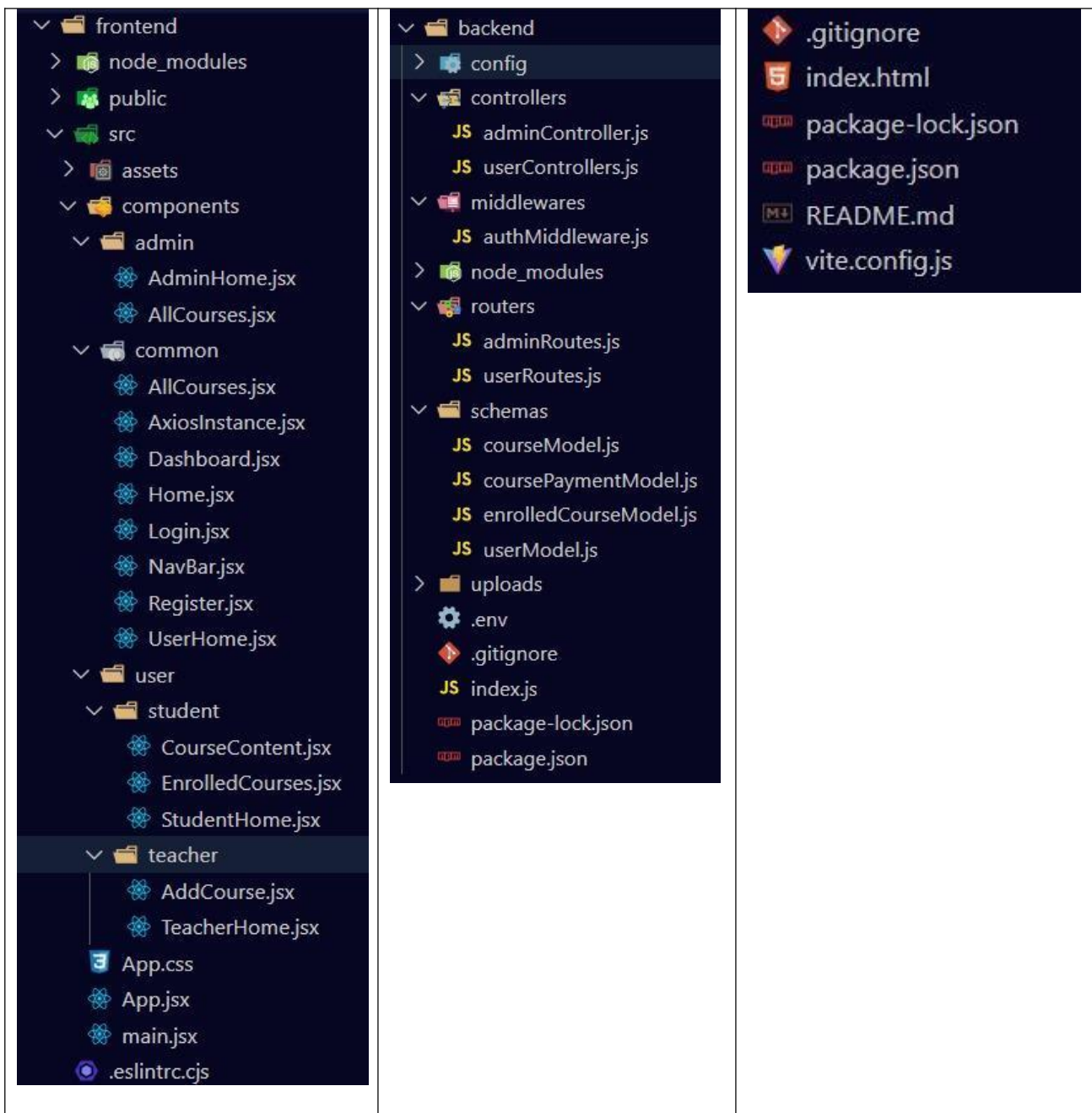
❖ README.md

- A markdown file that typically contains documentation for the project. It often includes an introduction, setup instructions, usage information, and any other details that help developers understand and contribute to the project.

❖ vite.config.js

- The configuration file for Vite, a fast build tool and development server for modern web projects. This file is used to customize Vite's behavior, such as setting up plugins, defining alias paths, and configuring the development and build environments.

This set of files represents the foundational setup for a Vite-based front-end project, including configuration, documentation, and project dependencies



8. APPLICATION FLOW :

The project has a user called– teacher and student and other will be Admin which takes care of all the user. The roles and responsibilities of these users can be inferred from the API endpoints defined in the code. Here is a summary:

I. Teacher:

- ❖ Can add courses for the student.
- ❖ Also delete the course if no student enrolled in it or any other reasons.
- ❖ Also add sections to courses.

II. Student:

- ❖ Can enroll in an individual or multiple course.
- ❖ Can start the course where it has stopped.
- ❖ Once the course is completed, they can download their certificate of completion of the course.
- ❖ For paid course, they need to purchase it and then they can start the course
- ❖ They can filter out the course by searching by name, category, etc

III. Admin:

- ❖ They can alter all the course that are present in the app.
- ❖ Watch out all kind of users in app.
- ❖ Record all the enrolled all the student that are enrolled in course.

9. RUNNING THE APPLICATION :

To run the Online Learning Platform (OLP) application locally, follow these steps:

For Backend:

- Open Terminal in VS Code or Command Prompt
- Navigate to the backend directory with the command : `cd backend`
- Install dependencies if not installed already : `npm install`
- Start the backend server : `npm start`
- The backend will run on **`http://localhost:8000`** in web browser to test API routes

For Frontend:

- Open Terminal in VS Code or Command Prompt
- Navigate to the frontend directory with the command : `cd frontend`
- Install dependencies if not installed already : `npm install`
- Start the frontend server : `npm run dev`
- The frontend will run on **http://localhost:5173** in web browser to access application

10. API DOCUMENTATION :

The following is a documentation of the API endpoints exposed by the backend of the OLP. These endpoints handle functionalities such as user registration, login, course management, enrollment, and more.

BACKEND CODE SNIPPETS WITH API

Backend > config > index.js

```
const express = require('express')
const cors = require('cors')
const dotenv = require('dotenv')
const DBConnection = require('./config/connect')
const path = require("path");

const app = express()
dotenv.config()
/////connection of DB////////
DBConnection()
const PORT = process.env.PORT
/////middleware////////
app.use(express.json())
app.use(cors())
app.use("/uploads", express.static(path.join(__dirname, "uploads")));
///ROUTES///
app.use('/api/admin', require('./routers/adminRoutes'))
app.use('/api/user', require('./routers/userRoutes'))
app.listen(PORT, () => console.log(`running on ${PORT}`))
```

Backend > config > connect.js

```
const mongoose = require("mongoose");
const connectionOfDb = () => {
  mongoose
    .connect(process.env.MONGO_DB, {
      useNewUrlParser: true,
```



```

    useUnifiedTopology: true,
  })
  .then(() => {
    console.log("Connected to MongoDB");
  })
  .catch((err) => {
    throw new Error(`Could not connect to MongoDB: ${err}`);
  });
};

```

```
module.exports = connectionOfDb;
```

Backend > middlewares > authMiddleware.js

```

const jwt = require("jsonwebtoken");
module.exports = async (req, res, next) => {
  try {
    const authorizationHeader = req.headers["authorization"];
    if (!authorizationHeader) {
      return res
        .status(401)
        .send({ message: "Authorization header missing", success: false });
    }

    const token = req.headers["authorization"].split(" ")[1];
    jwt.verify(token, process.env.JWT_KEY, (err, decode) => {
      if (err) {
        return res
          .status(200)
          .send({ message: "Token is not valid", success: false });
      } else {
        req.body.userId = decode.id;
        next();
      }
    });
  } catch (error) {
    console.error(error); // Handle or log the error appropriately
    res.status(500).send({ message: "Internal server error", success: false });
  }
};

```

Backend > controllers > userControllers.js

```

const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const userSchema = require("../schemas/userModel");
const courseSchema = require("../schemas/courseModel");
const enrolledCourseSchema = require("../schemas/enrolledCourseModel");
const coursePaymentSchema = require("../schemas/coursePaymentModel");
//////////for registering//////////
const registerController = async (req, res) => {
  try {
    const existsUser = await userSchema.findOne({ email: req.body.email });

```

```

    if (existsUser) {
      return res

      .status(200)
      .send({ message: "User already exists", success: false });
    }
    const password = req.body.password;
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);
    req.body.password = hashedPassword;
    const newUser = new userSchema(req.body);
    await newUser.save();
    return res.status(201).send({ message: "Register Success", success: true });
  } catch (error) {
    console.log(error);
    return res
      .status(500)
      .send({ success: false, message: `${error.message}` });
  }
};

////for the login
const loginController = async (req, res) => {
  try {
    const user = await userSchema.findOne({ email: req.body.email });
    if (!user) {
      return res
        .status(200)
        .send({ message: "User not found", success: false });
    }
    const isMatch = await bcrypt.compare(req.body.password, user.password);
    if (!isMatch) {
      return res
        .status(200)
        .send({ message: "Invalid email or password", success: false });
    }
    const token = jwt.sign({ id: user._id }, process.env.JWT_KEY, {
      expiresIn: "1d",
    });
    user.password = undefined;
    return res.status(200).send({
      message: "Login success successfully",
      success: true,
      token,
      userData: user,
    });
  } catch (error) {
    console.log(error);
    return res
      .status(500)
      .send({ success: false, message: `${error.message}` });
  }
};

//get all courses
const getAllCoursesController = async (req, res) => {
  try {
    const allCourses = await courseSchema.find();

```

```

    if (!allCourses) {
      return res.status(404).send("No Courses Found");
    }
    return res.status(200).send({
      success: true,
      data: allCourses,
    });
  } catch (error) {
    console.error("Error in deleting course:", error);
    res
      .status(500)
      .send({ success: false, message: "Failed to delete course" });
  }
};
////////posting course////////
const postCourseController = async (req, res) => {
  try {
    let price;
    // Extract data from the request body and files
    const {
      userId,
      C_educator,
      C_title,
      C_categories,
      C_price,
      C_description,
      S_title,
      S_description,
    } = req.body;
    const S_content = req.files.map((file) => file.filename); // Assuming you want to store the filenames
    in S_content
    // Create an array of sections
    const sections = [];
    for (let i = 0; i < S_title.length; i++) {
      sections.push({
        S_title: S_title[i],
        S_content: {
          filename: S_content[i],
          path: `/uploads/${S_content[i]}`,
        },
        S_description: S_description[i],
      });
    }
    if (C_price == 0) {
      price = "free";
    } else {
      price = C_price;
    }
    // Create an instance of the course schema
    const course = new courseSchema({
      userId,
      C_educator,
      C_title,
      C_categories,
      C_price: price,

```

```

    C_description,
    sections,

  });
  // Save the course instance to the database
  await course.save();
  res
    .status(201)
    .send({ success: true, message: "Course created successfully" });
} catch (error) {
  console.error("Error creating course:", error);
  res
    .status(500)
    .send({ success: false, message: "Failed to create course" });
}
};

///all courses for the teacher
const getAllCoursesUserController = async (req, res) => {
  try {
    const allCourses = await courseSchema.find({ userId: req.body.userId });
    if (!allCourses) {
      res.send({
        success: false,
        message: "No Courses Found",
      });
    } else {
      res.send({
        success: true,
        message: "All Courses Fetched Successfully",
        data: allCourses,
      });
    }
  } catch (error) {
    console.error("Error in fetching courses:", error);
    res
      .status(500)
      .send({ success: false, message: "Failed to fetch courses" });
  }
};

///delete courses by the teacher
const deleteCourseController = async (req, res) => {
  const { courseid } = req.params; // Use the correct parameter name
  try {
    // Attempt to delete the course by its ID
    const course = await courseSchema.findByIdAndDelete({ _id: courseid });
    // Check if the course was found and deleted successfully
    if (course) {
      res
        .status(200)
        .send({ success: true, message: "Course deleted successfully" });
    } else {
      res.status(404).send({ success: false, message: "Course not found" });
    }
  } catch (error) {
    console.error("Error in deleting course:", error);
    res

```

```

        .status(500)
        .send({ success: false, message: "Failed to delete course" });
    }
};
///  
enrolled course by the student
const enrolledCourseController = async (req, res) => {
    const { courseid } = req.params;
    const { userId } = req.body;
    try {
        const course = await courseSchema.findById(courseid);
        if (!course) {
            return res
                .status(404)
                .send({ success: false, message: "Course Not Found!" });
        }
        let course_Length = course.sections.length;
        // Check if the user is already enrolled in the course
        const enrolledCourse = await enrolledCourseSchema.findOne({
            courseId: courseid,
            userId: userId,
            course_Length: course_Length,
        });
        if (!enrolledCourse) {
            const enrolledCourseInstance = new enrolledCourseSchema({
                courseId: courseid,
                userId: userId,
                course_Length: course_Length,
            });

            const coursePayment = new coursePaymentSchema({
                userId: req.body.userId,
                courseId: courseid,
                ...req.body,
            });
            await coursePayment.save();
            await enrolledCourseInstance.save();
            // Increment the 'enrolled' count of the course by +1
            course.enrolled += 1;
            await course.save();
            res.status(200).send({
                success: true,
                message: "Enroll Successfully",
                course: { id: course._id, Title: course.C_title },
            });
        } else {
            res.status(200).send({
                success: false,
                message: "You are already enrolled in this Course!",
                course: { id: course._id, Title: course.C_title },
            });
        }
    } catch (error) {
        console.error("Error in enrolling course:", error);
        res
            .status(500)

```

```

        .send({ success: false, message: "Failed to enroll in the course" });
    }

};
/////sending the course content for learning to student
const sendCourseContentController = async (req, res) => {
    const { courseid } = req.params;
    try {
        const course = await courseSchema.findById({ _id: courseid });
        if (!course)
            return res.status(404).send({
                success: false,
                message: "No such course found",
            });
        const user = await enrolledCourseSchema.findOne({
            userId: req.body.userId,
            courseid: courseid, // Add the condition to match the courseid
        });
        if (!user) {
            return res.status(404).send({
                success: false,
                message: "User not found",
            });
        } else {
            return res.status(200).send({
                success: true,
                courseContent: course.sections,
                completeModule: user.progress,
                certificateData: user,
            });
        }
    } catch (error) {
        console.error("An error occurred:", error);
        return res.status(500).send({
            success: false,
            message: "Internal server error",
        });
    }
};
//////////completing module//////////
const completeSectionController = async (req, res) => {
    const { courseid, sectionid } = req.body; // Assuming you send courseid and sectionid in the request body
    // console.log(courseid, sectionid)
    try {
        // Check if the user is enrolled in the course
        const enrolledCourseContent = await enrolledCourseSchema.findOne({
            courseid: courseid,
            userId: req.body.userId, // Assuming you have user information in req.user
        });
        if (!enrolledCourseContent) {
            return res
                .status(400)
                .send({ message: "User is not enrolled in the course" });
        }
        // Update the progress for the section
    }
};

```

```

const updatedProgress = enrolledCourseContent.progress || [];
updatedProgress.push({ sectionId: sectionId });

// Update the progress in the database
await enrolledCourseSchema.findOneAndUpdate(
  { _id: enrolledCourseContent._id },
  { progress: updatedProgress },
  { new: true }
);
res.status(200).send({ message: "Section completed successfully" });
} catch (error) {
  console.error(error);
  res.status(500).send({ message: "Internal server error" });
}
};

//////////get all courses for particular user
const sendAllCoursesUserController = async (req, res) => {
  const { userId } = req.body;
  try {
    // First, fetch the enrolled courses for the user
    const enrolledCourses = await enrolledCourseSchema.find({ userId });
    // Now, let's retrieve course details for each enrolled course
    const coursesDetails = await Promise.all(
      enrolledCourses.map(async (enrolledCourse) => {
        // Find the corresponding course details using courseId
        const courseDetails = await courseSchema.findOne({
          _id: enrolledCourse.courseId,
        });
        return courseDetails;
      })
    );
    return res.status(200).send({
      success: true,
      data: coursesDetails,
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "An error occurred" });
  }
};

module.exports = {
  registerController,
  loginController,
  getAllCoursesController,
  postCourseController,
  getAllCoursesUserController,
  deleteCourseController,
  enrolledCourseController,
  sendCourseContentController,
  completeSectionController,
  sendAllCoursesUserController,
};

```

```

const userSchema = require("../schemas/userModel");
const courseSchema = require("../schemas/courseModel");
const enrolledCourseSchema = require("../schemas/enrolledCourseModel");
const coursePaymentSchema = require("../schemas/coursePaymentModel");
const getAllUsersController = async (req, res) => {
  try {
    const allUsers = await userSchema.find();
    if (allUsers === null || !allUsers) {
      return res.status(401).send({ message: "No users found" });
    }
    res.status(200).send({ success: true, data: allUsers });
  } catch (error) {
    console.log(error);
    return res
      .status(500)
      .send({ success: false, message: `${error.message}` });
  }
};

const getAllCoursesController = async (req, res) => {
  try {
    const allCourses = await courseSchema.find();
    if (allCourses === null || !allCourses) {
      return res.status(401).send({ message: "No courses found" });
    }
    res.status(200).send({ success: true, data: allCourses });
  } catch (error) {
    console.log(error);
    return res
      .status(500)
      .send({ success: false, message: `${error.message}` });
  }
};

const deleteCourseController = async (req, res) => {
  const { courseid } = req.params; // Use the correct parameter name
  try {
    // Attempt to delete the course by its ID
    const course = await courseSchema.findByIdAndDelete({ _id: courseid });
    // Check if the course was found and deleted successfully
    if (course) {
      res
        .status(200)
        .send({ success: true, message: "Course deleted successfully" });
    } else {
      res.status(404).send({ success: false, message: "Course not found" });
    }
  } catch (error) {
    console.error("Error in deleting course:", error);
    res
      .status(500)
      .send({ success: false, message: "Failed to delete course" });
  }
};

const deleteUserController = async (req, res) => {
  const { userid } = req.params; // Use the correct parameter name
  try {

```



```
// Attempt to delete the course by its ID
const user = await userSchema.findByIdAndDelete({ _id: userid });
// Check if the course was found and deleted successfully
if (user) {
  res
    .status(200)
    .send({ success: true, message: "User deleted successfully" });
} else {
  res.status(404).send({ success: false, message: "User not found" });
}
} catch (error) {
  console.error("Error in deleting user:", error);
  res
    .status(500)
    .send({ success: false, message: "Failed to delete course" });
}
};
module.exports = {
  getAllUsersController,
  getAllCoursesController,
  deleteCourseController,
  deleteUserController,
};
```

Backend > routers > userRoutes.js

```
const express = require("express");
const multer = require("multer");
const authMiddleware = require("../middlewares/authMiddleware");
const {
  registerController,
  loginController,
  postCourseController,
  getAllCoursesUserController,
  deleteCourseController,
  getAllCoursesController,
  enrolledCourseController,
  sendCourseContentController,
  completeSectionController,
  sendAllCoursesUserController,
} = require("../controllers/userControllers");
const router = express.Router();
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "uploads/");
  },
  filename: function (req, file, cb) {
    cb(null, file.originalname);
  },
});
const upload = multer({
  storage: storage,
});
```

```

router.post("/register", registerController);
router.post("/login", loginController);
router.post(
  "/addcourse",
  authMiddleware,
  // upload.single('C_image'),
  upload.array('S_content'),
  postCourseController
);
router.get('/getallcourses', getAllCoursesController);
router.get('/getallcoursesteacher', authMiddleware, getAllCoursesUserController);
router.delete('/deletecourse/:courseid', authMiddleware, deleteCourseController);
router.post('/enrolledcourse/:courseid', authMiddleware, enrolledCourseController);
router.get('/coursecontent/:courseid', authMiddleware, sendCourseContentController);
router.post('/completemodule', authMiddleware, completeSectionController);
router.get('/getallcoursesuser', authMiddleware, sendAllCoursesUserController);
module.exports = router;

```

Backend > routers > adminRoutes.js

```

const express = require("express");
const authMiddleware = require("../middlewares/authMiddleware");
const {
  getAllUsersController,
  getAllCoursesController,
  deleteCourseController,
  deleteUserController,
} = require("../controllers/adminController");
const router = express.Router();
router.get("/getallusers", authMiddleware, getAllUsersController);
router.get("/getallcourses", authMiddleware, getAllCoursesController);
router.delete('/deletecourse/:courseid', authMiddleware, deleteCourseController);
router.delete('/deleteuser/:userid', authMiddleware, deleteUserController);
module.exports = router;

```

Backend > schemas > userModel.js

```

const mongoose = require("mongoose");
const userModel = mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "name is required"],
      set: function (value) {
        return value.charAt(0).toUpperCase() + value.slice(1);
      },
    },
    email: {
      type: String,
      required: [true, "email is required"],
    },
    password: {

```

```

    type: String,
    required: [true, "password is required"],
  },
  type: {
    type: String,
    required: [true, "type is required"],
  },
  // enrolledCourses: [
  //   {
  //     type: mongoose.Schema.Types.ObjectId,
  //     ref: "course",
  //   },
  // ],
  },
  {
    timestamps: true,
  }
);

const userSchema = mongoose.model("user", userModel);
module.exports = userSchema;

```

Backend > schemas > enrolledCourseModel.js

```

const mongoose = require("mongoose");
const enrolledCourseSchema = mongoose.Schema(
  {
    courseId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "course",
    },
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "user",
    },
    course_Length: {
      type: Number,
      required: true,
    },
    progress: [{}],
    certificateDate: {
      type: Date,
    },
  },
  {
    timestamps: true,
  }
);
module.exports = mongoose.model("enrolledCourses", enrolledCourseSchema);

```

Backend > schemas > coursePaymentModel.js

```
const mongoose = require("mongoose");
const coursePaymentModel = mongoose.Schema(
  {
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "user",
    },
    courseId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "course",
    },
    cardDetails: {
      cardholdername: {
        type: String,
      },
      cardnumber: {
        type: Number,
      },
      cvvcode: {
        type: Number,
      },
      expmonthyear: {
        type: String,
      },
    },
    status: {
      type: String,
      default: "enrolled",
    },
  },
);
```

```

{
  timestamps: true,
  strict: false,
}
);
const coursePaymentSchema = mongoose.model("coursePayment", coursePaymentModel);
module.exports = coursePaymentSchema;

```

Backend > schemas > courseModel.js

```

const mongoose = require("mongoose");
const courseModel = mongoose.Schema(
{
  userId: {
    type: String,
    required: true,
  },
  C_educator: {
    type: String,
    required: [true, "name is required"],
  },
  C_title: {
    type: String,
    required: [true, "C_title is required"],
  },
  C_categories: {
    type: String,
    required: [true, "C_categories: is required"],
  },
  C_price: {
    type: String,
  },
  C_description: {
    type: String,
    required: [true, "C_description: is required"],
  },
  sections: {},
  enrolled: {
    type: Number,
    default: 0,
  },
},
{
  timestamps: true,
}
);
const courseSchema = mongoose.model("course", courseModel);
module.exports = courseSchema;

```

11. PROJECT IMPLEMENTATION :

Front - End Code Snippets

Frontend > index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Frontend > src > main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'

ReactDOM.createRoot(document.getElementById('root')).render(
  <App />,)
```

Frontend > src > App.jsx

```
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import { useState, useEffect, createContext } from "react";
import "./App.css";
import Home from "./components/common/Home";
import Login from "./components/common/Login";
import Register from "./components/common/Register";
import Dashboard from "./components/common/Dashboard";
import CourseContent from "./components/user/student/CourseContent";

export const UserContext = createContext();
function App() {
  const date = new Date().getFullYear();
  const [userData, setUserData] = useState();
  const [userLoggedIn, setUserLoggedIn] = useState(false);
  const getData = async () => {
    try {
      const user = await JSON.parse(localStorage.getItem("user"));
      if (user && user !== undefined) {
```

```

        setData(user);
        setUserLoggedIn(true);
    }
} catch (error) {
    console.log(error);
}
};
useEffect(() => {
    getData();
}, []);
return (
    <UserContext.Provider value={{ userData, userLoggedIn }}>
    <div className="App">
        <Router>
            <div className="content">
                <Routes>
                    <Route exact path="/" element={<Home />} />
                    <Route path="/login" element={<Login />} />
                    <Route path="/register" element={<Register />} />
                    {/* <Route path="/about" element={<About />} /> */}
                    <Route path="/forgotpassword" element={<ForgotPassword />} /> */}
                    {userLoggedIn ? (
                        <>
                            <Route path="/dashboard" element={<Dashboard />} />
                            <Route path="/courseSection/:courseId/:courseTitle" element={<CourseContent />} />
                        </>
                    ) : (
                        <Route path="/login" element={<Login />} />
                    )}
                </Routes>
            </div>
            <footer className="bg-light text-center text-lg-start">
                <div className="text-center p-3">
                    © {date} Copyright: Study App
                </div>
            </footer>
        </Router>
    </div>
</UserContext.Provider>
);
}
export default App;

```

Frontend > src > App.css

```

@import "../node_modules/bootstrap/dist/css/bootstrap.min.css";
* {
    padding: 0;
    margin: 0;
    box-sizing: border-box;
    font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;
    border: none;
    outline: none;
}

```

```
html,
body {
  width: 100%;
  height: 100%;
}

a {
  color: black;
  text-decoration: none;
  margin-right: 20px;
}

::-webkit-scrollbar {
  display: none;
}

#root {
  background-color: rgba(255, 224, 184, 0.523);
}

.App {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}

.content {
  flex: 1;
}

.content .first-container {
  background-image: url("../src/assets/Images/bg.jpg");
  background-size: cover;
  background-position: center;
}

.first-container,
.second-container {
  width: 100%;
  height: 92vh;
}

.content-home {
  position: absolute;
  top: 30%;
  left: 5%;
}

.content-home p {
  font-size: 42px;
  font-weight: 800;
  color: rgb(0, 0, 0);
  letter-spacing: 10px;
}
```



```
.content-home button {
  background-color: white;
  color: black;
}

.card-container,
.course-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 16px;
}

.card-container,
.course-container {
  display: flex;
  flex-wrap: wrap;
}

.filter-container {
  width: 100%;
  height: 10vh;
  display: flex;
  align-items: center;
  justify-content: center;
  margin-bottom: 10px;
}

.filter-container input,
select {
  border: 1px solid;
  padding: 10px;
  margin: 10px;
  border-radius: 5px;
}

.card,
.card1 {
  width: calc(33%-10px);
  height: 40%;
}

.read-more-link {
  cursor: pointer;
  color: blue;
  text-decoration: underline;
}

.card1 {
  border-radius: 5px;
  width: 300px;
  height: 320px;
  position: relative;
  border: 1px solid;
  overflow: hidden;
}
```

```

.description-container {
  width: 100%;
  overflow-x: auto;
  /* Enable horizontal scroll if content overflows */
}

.description {
  white-space: nowrap;
  /* Prevent text from wrapping */
  overflow: hidden;
  /* Hide overflowing content */
  text-overflow: ellipsis;
  /* Display ellipsis (...) for truncated text */
  padding: 5px;
  /* Add padding for better visibility */
}

.card1 .desc {
  width: 100%;
  height: 100%;
  width: 100%;
  transition: .5s;
  padding: 10px;
}

.card1:hover .desc {
  opacity: .5;
  transform: translateX(30%);
}

.card1 .details {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-image: url('../src/assets/Images/image.png');
  background-size: cover;

  transition: 1.9s;
  transform-origin: left;
  transform: perspective(2000px) rotateY(-90deg);
  border-radius: 5px;
}

.card1:hover .details {
  transform: perspective(2000px) rotateY(0deg);
}

.card1 .details .center {
  padding: 20px;
  text-align: center;
  background: #fff;
}

```

```
position: absolute;
top: 50%;
left: 30%;
transform: translate(-50%, -50%);
width: 80%;
height: 80%;
}
```

```
.card1 .details .center h1 {
margin: 0;
padding: 0;
color: #ff3636;
line-height: 20px;
font-size: 25px;
text-transform: uppercase;
}
```

```
.card1 .details .center h1 span {
font-size: 14px;
color: #262626;
}
```

```
.card1 .details .center p {
margin: 10px 0;
padding: 0;
color: #262626;
}
```

```
.card1 .details .center ul {
margin: 10px auto 0;
padding: 0;
display: table;
}
```

```
.card1 .details .center ul li {
list-style: none;
margin: 0 5px;
float: left;
}
```

```
.card1 .details .center ul li a {
display: block;
background: #262626;
color: #fff;
width: 30px;
height: 30px;
line-height: 30px;
text-align: center;
transform: .5s;
}
```

```
.card1 .details .center ul li a:hover {
background: #ff3636;
}
```

```
.course-content {
  width: 100%;
  padding: 20px;
  display: flex;
  flex-wrap: wrap;
}

.course-section {
  width: 50%;
  padding: 20px;
  box-sizing: border-box;
}

.course-video {
  width: 50%;
  padding: 20px;
  display: flex;
  justify-content: center;
  align-items: center;
}

@media (min-width: 940px) {
  .card {
    width: calc(25%-10px);
  }
}

.certificate {
  width: 100%;
  padding: 20px;
  border: 1px solid #000;
  text-align: center;
}

.certificate h1 {
  font-size: 24px;
  font-weight: bold;
  margin-bottom: 20px;
}

.certificate .content {
  margin-bottom: 40px;
}

.certificate .content h2 {
  font-size: 20px;
  margin-top: 10px;
  margin-bottom: 10px;
  font-weight: bold;
}

.certificate .content h3 {
  font-size: 18px;
```

```

margin-top: 10px;
margin-bottom: 10px;
font-weight: 700;
}

.certificate .content .date {
font-size: 16px;
font-weight: bold;
margin-top: 10px;
margin-bottom: 20px;
}

@media (max-width: 768px) {
.course-section {
width: 100%;
/* Take full width on smaller screens */
}

.course-video {
width: 100%;
/* Take full width on smaller screens */
}
}

```

Frontend > src > Components > user > teacher > TeacherHome.jsx

```

import React, { useContext, useEffect, useState } from 'react';
import { Button, Card, Container } from 'react-bootstrap';
import axiosInstance from '../common/AxiosInstance';

const TeacherHome = () => {
  const [allCourses, setAllCourses] = useState([]);
  const getAllCoursesUser = async () => {
    try {
      const res = await axiosInstance.get(`api/user/getallcoursesteacher`, {
        headers: {
          Authorization: `Bearer ${localStorage.getItem('token')}`,
        },
      });
      if (res.data.success) {
        setAllCourses(res.data.data);
      }
    } catch (error) {
      console.log('An error occurred:', error);
    }
  };

  useEffect(() => {
    getAllCoursesUser();
  }, []);

  const toggleDescription = (courseId) => {
    setAllCourses((prevCourses) =>
      prevCourses.map((course) =>

```

```

        course._id === courseId
        ? { ...course, showFullDescription: !course.showFullDescription }
        : course
    )
  );
};

const deleteCourse = async (courseId) => {
  const confirmation = confirm('Are you sure you want to delete')
  if (!confirmation) {
    return;
  }
  try {
    const res = await axiosInstance.delete(`api/user/deletecourse/${courseId}`, {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('token')}`,
      },
    })
    if (res.data.success) {
      alert(res.data.message)
      getAllCoursesUser()
    } else {
      alert("Failed to delete the course")
    }
  } catch (error) {
    console.log('An error occurred:', error);
  }
}

return (
  <Container className='card-container'>
    {allCourses?.length > 0 ? (
      allCourses.map((course) => (
        <Card key={course._id} className='card'>
          {/* <Card.Img variant='top' src='holder.js/100px180' /> */}
          <Card.Body>
            <Card.Title>{course.C_title}</Card.Title>
            <Card.Text>
              <p>
                <strong>Description: </strong>
                {course.showFullDescription
                  ? course.C_description
                  : course.C_description.slice(0, 10)} {' '}
                {course.C_description.length > 10 && (
                  <span
                    className='read-more-link'
                    onClick={() => toggleDescription(course._id)}
                  >
                    {course.showFullDescription ? 'Read Less' : 'Read More'}
                  </span>
                )}
              </p>
              <p>
                <strong>Category: </strong>
                {course.C_categories}
              </p>
            </p>
          </Card.Body>
        </Card>
      )
    )
  )
);

```

```

        <strong>Sections: </strong> {course.sections.length}
      </p>
      <p style={{color: '#c3b9b9'}}>
        <strong>Enrolled students: </strong> {course.enrolled}
      </p>
    </Card.Text>
    <div style={{float: 'right'}} className='d-flex'>
      <Button variant='primary' onClick={() => deleteCourse(course._id)}>Delete</Button>
    </div>
  </Card.Body>
</Card>
))
):(
  'No courses found!!'
)}
</Container>
);
};

```

export default TeacherHome;

Frontend > src > Components > user > teacher > AddCourse.jsx

```

import React, { useState, useContext } from 'react';
import { Button, Form, Col, Row } from 'react-bootstrap';
import { UserContext } from '../App';
import axiosInstance from '../common/AxiosInstance';

const AddCourse = () => {
  const user = useContext(UserContext);
  const [addCourse, setAddCourse] = useState({
    userId: user.userData._id,
    C_educator: "",
    C_title: "",
    C_categories: "",
    C_price: "",
    C_description: "",
    sections: [],
  });
  const handleChange = (e) => {
    const { name, value } = e.target;
    setAddCourse({ ...addCourse, [name]: value });
  };
  const handleCourseTypeChange = (e) => {
    setAddCourse({ ...addCourse, C_categories: e.target.value });
  };

  const addInputGroup = () => {
    setAddCourse({
      ...addCourse,
      sections: [

```

```

        ...addCourse.sections,
        {
            S_title: "",
            S_description: "",
            S_content: null,
        },
    ],
});
};

const handleChangeSection = (index, e) => {
    const updatedSections = [...addCourse.sections];
    const sectionToUpdate = updatedSections[index];
    if (e.target.name.endsWith('S_content')) {
        sectionToUpdate.S_content = e.target.files[0];
    } else {
        sectionToUpdate[e.target.name] = e.target.value;
    }
    setAddCourse({ ...addCourse, sections: updatedSections });
};

const removeInputGroup = (index) => {
    const updatedSections = [...addCourse.sections];
    updatedSections.splice(index, 1);
    setAddCourse({
        ...addCourse,
        sections: updatedSections,
    });
};

const handleSubmit = async (e) => {
    e.preventDefault()
    const formData = new FormData();
    Object.keys(addCourse).forEach((key) => {
        if (key === 'sections') {
            addCourse[key].forEach((section, index) => {
                if (section.S_content instanceof File) {
                    formData.append('S_content', section.S_content);
                }
                formData.append('S_title', section.S_title);
                formData.append('S_description', section.S_description);
            });
        } else {
            formData.append(key, addCourse[key]);
        }
    });

    for (const [key, value] of formData.entries()) {
        console.log(`${key}:`, value);
    }

    try {
        const res = await axiosInstance.post('/api/user/addcourse', formData, {
            headers: {
                Authorization: `Bearer ${localStorage.getItem('token')}`,
                'Content-Type': 'multipart/form-data',
            },
        });
    } catch (error) {
        console.log(error);
    }
};

```



```

    },
  });
  if (res.status === 201) {
    if (res.data.success) {
      alert(res.data.message);
    } else {
      alert('Failed to create course');
    }
  } else {
    alert('Unexpected response status: ' + res.status);
  }
} catch (error) {
  console.error('An error occurred:', error);
  alert('An error occurred while creating the course');
}
};
return (
  <div className=">
    <Form className="mb-3" onSubmit={handleSubmit}>
      <Row className="mb-3">
        <Form.Group as={Col} controlId="formGridJobType">
          <Form.Label>Course Type</Form.Label>
          <Form.Select value={addCourse.C_categories} onChange={handleCourseTypeChange}>
            <option>Select categories</option>
            <option>IT & Software</option>
            <option>Finance & Accounting</option>
            <option>Personal Development</option>
          </Form.Select>
        </Form.Group>
        <Form.Group as={Col} controlId="formGridTitle">
          <Form.Label>Course Title</Form.Label>
          <Form.Control name='C_title' value={addCourse.C_title} onChange={handleChange}
type="text" placeholder="Enter Course Title" required />
        </Form.Group>
      </Row>
      <Row className="mb-3">
        <Form.Group as={Col} controlId="formGridTitle">
          <Form.Label>Course Educator</Form.Label>
          <Form.Control name='C_educator' value={addCourse.C_educator}
onChange={handleChange} type="text" placeholder="Enter Course Educator" required />
        </Form.Group>
        <Form.Group as={Col} controlId="formGridTitle">
          <Form.Label>Course Price(Rs.)</Form.Label>
          <Form.Control name='C_price' value={addCourse.C_price} onChange={handleChange}
type="text" placeholder="for free course, enter 0" required />
        </Form.Group>
        <Form.Group as={Col} className="mb-3" controlId="formGridAddress2">
          <Form.Label>Course Description</Form.Label>
          <Form.Control name='C_description' value={addCourse.C_description}
onChange={handleChange} required as={"textarea"} placeholder="Enter Course description" />
        </Form.Group>
      </Row>
    </div>
  )

```

```

    {addCourse.sections.map((section, index) => (
      <div key={index} className="d-flex flex-column mb-4 border rounded-3 border-3 p-3
position-relative">
        <Col xs={24} md={12} lg={8}>
          <span style={{ cursor: 'pointer' }} className="position-absolute top-0 end-0 p-1"
onClick={() => removeInputGroup(index)}>
            </span>
          </Col>
        <Row className='mb-3'>
          <Form.Group as={Col} controlId="formGridTitle">
            <Form.Label>Section Title</Form.Label>
            <Form.Control
              name={`S_title`}
              value={section.S_title}
              onChange={(e) => handleChangeSection(index, e)}
              type="text"
              placeholder="Enter Section Title"
              required
            />
          </Form.Group>
          <Form.Group as={Col} controlId="formGridContent">
            <Form.Label>Section Content (Video or Image)</Form.Label>
            <Form.Control
              name={`S_content`}
              onChange={(e) => handleChangeSection(index, e)}
              type="file"
              accept="video/*,image/*"
              required
            />
          </Form.Group>
          <Form.Group className="mb-3" controlId="formGridAddress2">
            <Form.Label>Section Description</Form.Label>
            <Form.Control
              name={`S_description`}
              value={section.S_description}
              onChange={(e) => handleChangeSection(index, e)}
              required
              as={"textarea"}
              placeholder="Enter Section description"
            />
          </Form.Group>
        </Row>
      </div>
    )))

    <Row className="mb-3">
      <Col xs={24} md={12} lg={8}>
        <Button size='sm' variant='outline-secondary' onClick={addInputGroup}>
          + Add Section
        </Button>
      </Col>
    </Row>

    <Button variant="primary" type="submit">
      Submit

```

```

        </Button>
      </Form>
    </div>
  );
};

export default AddCourse;

```

Frontend > src > Components > user > student > StudentHome.jsx

```

import React from 'react'
import AllCourses from '../common/AllCourses'
import { Container } from 'react-bootstrap'

const StudentHome = () => {
  return (
    <
      <Container>
        <AllCourses />
      </Container>
    </>
  )
}

export default StudentHome

```

Frontend > src > Components > user > student > EnrolledCourses.jsx

```

import React, { useEffect, useState } from 'react'
import axiosInstance from '../common/AxiosInstance';
import { Link } from 'react-router-dom';
import { Button, styled, TableRow, TableHead, TableContainer, Paper, Table, TableBody, TableCell,
  TableCellClasses } from '@mui/material'

const StyledTableCell = styled(TableCell)(({ theme }) => ({
  [`&.${tableCellClasses.head}`]: {
    backgroundColor: theme.palette.common.black,
    color: theme.palette.common.white,
  },
  [`&.${tableCellClasses.body}`]: {
    fontSize: 14,
  },
}));

const StyledTableRow = styled(TableRow)(({ theme }) => ({
  '&:nth-of-type(odd)': {
    backgroundColor: theme.palette.action.hover,
  },
  // hide last border
  '&:last-child td, &:last-child th': {
    border: 0,
  },
}));

```

```

    },
  }));
const EnrolledCourses = () => {
  const [allEnrolledCourse, setAllEnrolledCourses] = useState([])
  const allCourses = async () => {
    try {
      const res = await axiosInstance.get('api/user/getallcoursesuser', {
        headers: {
          "Authorization": `Bearer ${localStorage.getItem("token")}`
        }
      })
      if (res.data.success) {
        setAllEnrolledCourses(res.data.data)
      }
      else {
        alert(res.data.message)
      }
    } catch (error) {
      console.log(error);
    }
  }
  useEffect(() => {
    allCourses()
  }, [])
  return (
    <TableContainer component={Paper}>
      <Table sx={{ minWidth: 700 }} aria-label="customized table">
        <TableHead>
          <TableRow>
            <StyledTableCell>Course ID</StyledTableCell>
            <StyledTableCell align="left">Course Name</StyledTableCell>
            <StyledTableCell align="left">Course Educator</StyledTableCell>
            <StyledTableCell align="left">Course Category</StyledTableCell>
            <StyledTableCell align="left">Action</StyledTableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {
            allEnrolledCourse?.length > 0 ? (
              allEnrolledCourse?.map((course) => (
                <StyledTableRow key={course._id}>
                  <StyledTableCell component="th" scope="row">
                    {course._id}
                  </StyledTableCell>
                  <StyledTableCell component="th" scope="row">
                    {course.C_title}
                  </StyledTableCell>
                  <StyledTableCell component="th" scope="row">
                    {course.C_educator}
                  </StyledTableCell>
                  <StyledTableCell component="th" scope="row">
                    {course.C_categories}
                  </StyledTableCell>
                </StyledTableRow>
              )
            ) : (
              <Table>
                <tbody>
                  <tr>
                    <td>No Courses Found</td>
                  </tr>
                </tbody>
              </Table>
            )
          }
        </TableBody>
      </Table>
    </TableContainer>
  )
}

```

```

        <StyledTableCell component="th" scope="row">
            <Link to={`/${courseSection}/${course._id}/${course.C_title}`}><Button size='small'
variant="contained" color="success">Go To</Button></Link>
        </StyledTableCell>
    </StyledTableRow>
    )))
    :
    (<p className='px-2'>yet to be enrolled courses</p>)
    }
</TableBody>
</Table>
</TableContainer>
)
}

export default EnrolledCourses

```

Frontend > src > Components > user > student > CourseContent.jsx

```

import React, { useState, useEffect, useContext } from 'react';
import { useParams, Link } from 'react-router-dom';
import { Accordion, Modal } from 'react-bootstrap';
import axiosInstance from '../common/AxiosInstance';
import ReactPlayer from 'react-player';
import { UserContext } from '../App';
import NavBar from '../common/NavBar';
import html2canvas from "html2canvas";
import { jsPDF } from "jspdf";
import { Button } from '@mui/material';

const CourseContent = () => {
    const user = useContext(UserContext)
    const { courseId, courseTitle } = useParams(); // Extract courseId from URL
    const [courseContent, setCourseContent] = useState([]);
    const [currentVideo, setCurrentVideo] = useState(null);
    const [playingSectionIndex, setPlayingSectionIndex] = useState(-1);
    const [completedSections, setCompletedSections] = useState([]);
    const [completedModule, setCompletedModule] = useState([]);
    const [showModal, setShowModal] = useState(false);
    const [certificate, setCertificate] = useState(null)
    // Extract sectionIds from completedModule
    const completedModuleIds = completedModule.map((item) => item.sectionId);
    const downloadPdfDocument = (rootElementId) => {
        const input = document.getElementById(rootElementId);
        html2canvas(input).then((canvas) => {
            const imgData = canvas.toDataURL('image/png');
            const pdf = new jsPDF();
            pdf.addImage(imgData, 'JPEG', 0, 0);
            pdf.save('download-certificate.pdf');
        });
    };
};

```

```

const getCourseContent = async () => {
  try {
    const res = await axiosInstance.get(`/api/user/coursecontent/${courseId}`, {
      headers: {
        "Authorization": `Bearer ${localStorage.getItem("token")}`
      }
    });
    if (res.data.success) {
      setCourseContent(res.data.courseContent);
      console.log(res.data.completeModule)
      setCompletedModule(res.data.completeModule)
      // setCompletedModule(res.data.completeModule[0]?.progress);
      setCertificate(res.data.certificateData.updatedAt)
    }
  } catch (error) {
    console.log(error);
  }
};

useEffect(() => {
  getCourseContent();
}, [courseId]);

const playVideo = (videoPath, index) => {
  setCurrentVideo(videoPath);
  setPlayingSectionIndex(index);
};

const completeModule = async (sectionId) => {
  if (completedModule.length < courseContent.length) {
    // Mark the current section as completed
    if (playingSectionIndex !== -1 && !completedSections.includes(playingSectionIndex)) {
      setCompletedSections([...completedSections, playingSectionIndex]);
      // Send a request to the server to update the user's progress
      try {
        const res = await axiosInstance.post(`/api/user/completemodule`, {
          courseId,
          sectionId: sectionId
        }, {
          headers: {
            Authorization: `Bearer ${localStorage.getItem('token')}`
          }
        });
        if (res.data.success) {
          // Handle success if needed
          alert(res.data.message);
          getCourseContent()
        }
      } catch (error) {
        console.log(error);
      }
    }
  } else {
    // Show the modal
    setShowModal(true);
  }
};

```

```

return (
  <
    <NavBar />
    <h1 className='my-3 text-center'>Welcome to the course: {courseTitle}</h1>
    <div className='course-content'>
      <div className="course-section">
        <Accordion defaultActiveKey="0" flush>
          {courseContent.map((section, index) => {
            // Extract sectionId from the section
            const sectionId = index;
            // Check if the sectionId is not in completedModuleIds
            const isSectionCompleted = !completedModuleIds.includes(sectionId);
            return (
              <Accordion.Item key={index} eventKey={index.toString()}>
                <Accordion.Header>{section.S_title}</Accordion.Header>
                <Accordion.Body>
                  {section.S_description}
                  {section.S_content && (
                    <
                      <Button color='success' className='mx-2' variant="text" size="small"
onClick={() => playVideo(`http://localhost:8000${section.S_content.path}`, index)}>
                        Play Video
                      </Button>
                      {isSectionCompleted && !completedSections.includes(index) && (
                        <Button
                          variant='success'
                          size='sm'
                          onClick={() => completeModule(sectionId)}
                          disabled={playingSectionIndex !== index}
                        >
                          Completed
                        </Button>
                      )}
                    </>
                  )}
                </Accordion.Body>
              </Accordion.Item>
            );
          })}
          {completedModule.length === courseContent.length && (
            <Button className='my-2' onClick={() => setShowModal(true)}>Download
Certificate</Button>
          )}
        </Accordion>
      </div>
      <div className="course-video w-50">
        {currentVideo && (
          <ReactPlayer
            url={currentVideo}
            width='100%'
            height='100%'
            controls
          />
        )}
      </div>
    </div>
  </

```

```

</div>
<Modal
  size="lg"
  show={showModal}
  onHide={() => setShowModal(false)}
  dialogClassName="modal-90w"
  aria-labelledby="example-custom-modal-styling-title"
>
  <Modal.Header closeButton>
    <Modal.Title id="example-custom-modal-styling-title">
      Completion Certificate
    </Modal.Title>
  </Modal.Header>
  <Modal.Body>
    Congratulations! You have completed all sections. Here is your certificate
    <div id='certificate-download' className="certificate text-center">
      <h1>Certificate of Completion</h1>
      <div className="content">
        <p>This is to certify that</p>
        <h2>{user.userData.name}</h2>
        <p>has successfully completed the course</p>
        <h3>{courseTitle}</h3>
        <p>on</p>
        <p className="date">{new Date(certificate).toLocaleDateString()}</p>
      </div>
    </div>
    <Button onClick={() => downloadPdfDocument('certificate-download')} style={{ float:
'right', marginTop: 3 }}>Download Certificate</Button>
  </Modal.Body>
</Modal>
</>
);
};

export default CourseContent;

```

Frontend > src > Components > common > UserHome.jsx

```

import React, { useContext, useEffect, useState } from 'react';
import { Container } from 'react-bootstrap';
import { UserContext } from '../App';
import TeacherHome from '../user/teacher/TeacherHome';
import AdminHome from '../admin/AdminHome';
import StudentHome from '../user/student/StudentHome';
// import axiosInstance from './AxiosInstance';

const UserHome = () => {
  const user = useContext(UserContext);
  let content;
  {
    switch (user.userData.type) {
      case "Teacher":
        content = <TeacherHome />

```



```

        break;
      case "Admin":
        content = <AdminHome />
      break;
      case "Student":
        content = <StudentHome />
      break;
      default:
        break;
    }
  }
  return (
    <Container>
      {content}
    </Container>
  );
};

```

```
export default UserHome;
```

Frontend > src > Components > common > Register.jsx

```

import React, { useState } from 'react'
import { Link, useNavigate } from 'react-router-dom';
import Navbar from 'react-bootstrap/Navbar';
import { Container, Nav } from 'react-bootstrap';
import Avatar from '@mui/material/Avatar';
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import Grid from '@mui/material/Grid';
import Box from '@mui/material/Box';
import Typography from '@mui/material/Typography';
import axiosInstance from './AxiosInstance'

import Dropdown from 'react-bootstrap/Dropdown';
const Register = () => {
  const navigate = useNavigate()
  const [selectedOption, setSelectedOption] = useState('Select User');
  const [data, setData] = useState({
    name: "",
    email: "",
    password: "",
    type: "",
  })
  const handleSelect = (eventKey) => {
    setSelectedOption(eventKey);
    setData({ ...data, type: eventKey });
  };
  const handleChange = (e) => {
    const { name, value } = e.target;
    setData({ ...data, [name]: value });
  };
};

```

```

const handleSubmit = (e) => {
  e.preventDefault()
  if (!data?.name || !data?.email || !data?.password || !data?.type) return alert("Please fill all fields");
  else {
    axiosInstance.post('/api/user/register', data)
      .then((response) => {
        if (response.data.success) {
          alert(response.data.message)
          navigate('/login')
        } else {
          console.log(response.data.message)
        }
      })
      .catch((error) => {
        console.log("Error", error);
      });
  }
};

return (
  <
    <Navbar expand="lg" className="bg-body-tertiary">
      <Container fluid>
        <Navbar.Brand><h2>Study App</h2></Navbar.Brand>
        <Navbar.Toggle aria-controls="navbarScroll" />
        <Navbar.Collapse id="navbarScroll">
          <Nav
            className="me-auto my-2 my-lg-0"
            style={{ maxHeight: '100px' }}
            navbarScroll
          >
            </Nav>
            <Nav>
              <Link to="/">Home</Link>
              {/* <Link to="/about">About</Link> */}
              <Link to="/login">Login</Link>
              <Link to="/register">Register</Link>
            </Nav>
          </Navbar.Collapse>
        </Container>
      </Navbar>
      <div className="first-container">
        <Container component="main" style={{ display: 'flex', justifyContent: 'center',
alignItems: 'center' }} >
          <Box
            sx={{
              marginTop: 8,
              marginBottom: 4,
              display: 'flex',
              flexDirection: 'column',
              alignItems: 'center',
              padding: '10px',
              background: '#dddde8db',
              border: '1px solid lightblue',
              borderRadius: '5px'
            }}
          >

```

```

    }}
  >
  <Avatar sx={{ bgcolor: 'secondary.main' }}>
    { /* <LockOutlinedIcon /> */ }
  </Avatar>
  <Typography component="h1" variant="h5">
    Register
  </Typography>
  <Box component="form" onSubmit={handleSubmit} noValidate>
    <TextField
      margin="normal"
      fullWidth
      id="name"
      label="Full Name"
      name="name"
      value={data.name}
      onChange={handleChange}
      autoComplete="name"
      autoFocus
    />
    <TextField
      margin="normal"
      fullWidth
      id="email"
      label="Email Address"
      name="email"
      value={data.email}
      onChange={handleChange}
      autoComplete="email"
      autoFocus
    />
    <TextField
      margin="normal"
      fullWidth
      name="password"
      value={data.password}
      onChange={handleChange}
      label="Password"
      type="password"
      id="password"
      autoComplete="current-password"
    />
    <Dropdown className='my-3'>
      <Dropdown.Toggle variant="outline-secondary" id="dropdown-basic">
        {selectedOption}
      </Dropdown.Toggle>
      <Dropdown.Menu>
        <Dropdown.Item onClick={() =>
handleSelect("Student")}>Student</Dropdown.Item>
        <Dropdown.Item onClick={() =>
handleSelect("Teacher")}>Teacher</Dropdown.Item>
      </Dropdown.Menu>
    </Dropdown>
  </Box>

```

```

    <Box mt={2}>
      <Button
        type="submit"
        variant="contained"
        sx={{ mt: 3, mb: 2 }}
        style={{ width: '200px' }}
      >
        Sign Up
      </Button>
    </Box>
    <Grid container>
      <Grid item>Have an account?
        <Link style={{ color: "blue" }} to={"/login"} variant="body2">
          {" Sign In"}
        </Link>
      </Grid>
    </Grid>
  </Box>
</Box>
</Container>
</div>
</>
)
}

```

export default Register

Frontend > src > Components > common > NavBar.jsx

```

import React, { useContext } from 'react'
import { Navbar, Nav, Button, Container } from 'react-bootstrap';
import { UserContext } from '../App';
import { NavLink } from 'react-router-dom';

const NavBar = ({ setSelectedComponent }) => {
  const user = useContext(UserContext)
  if (!user) {
    return null
  }
  const handleLogout = () => {
    localStorage.removeItem("token");
    localStorage.removeItem("user");
    window.location.href = "/";
  }
  const handleOptionClick = (component) => {
    setSelectedComponent(component);
  };
  return (
    <Navbar expand="lg" className="bg-body-tertiary">
      <Container fluid>
        <Navbar.Brand>
          <h3>Study App</h3>
        </Navbar.Brand>

```

```

    <Navbar.Toggle aria-controls="navbarScroll" />
    <Navbar.Collapse id="navbarScroll">
    <Nav className="me-auto my-2 my-lg-0" style={{ maxHeight: '100px' }} navbarScroll>
      <NavLink onClick={() => handleOptionClick('home')}>Home</NavLink>
      {user.userData.type === 'Teacher' && (
        <NavLink onClick={() => handleOptionClick('addcourse')}>Add Course</NavLink>
      )}
      {user.userData.type === 'Admin' && (
        <
          <NavLink onClick={() => handleOptionClick('cousres')}>Courses</NavLink>
        </>
      )}
      {user.userData.type === 'Student' && (
        <
          <NavLink onClick={() => handleOptionClick('enrolledcourese')}>Enrolled
Courses</NavLink>
        </>
      )}
    </Nav>
    <Nav>
      <h5 className='mx-3'>Hi {user.userData.name}</h5>
      <Button onClick={handleLogout} size='sm' variant='outline-danger'>Log Out</Button >
    </Nav>
  </Navbar.Collapse>
</Container>
</Navbar>
)
}

export default NavBar;

```

Frontend > src > Components > common > Login.jsx

```

import React, { useState } from 'react'
import { Link, useNavigate } from 'react-router-dom';
import Navbar from 'react-bootstrap/Navbar';
import { Container, Nav } from 'react-bootstrap';
import Avatar from '@mui/material/Avatar';
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import Grid from '@mui/material/Grid';
import Box from '@mui/material/Box';
import Typography from '@mui/material/Typography';
import axiosInstance from './AxiosInstance';

const Login = () => {
  const navigate = useNavigate()
  const [data, setData] = useState({
    email: '',
    password: '',
  })
  const handleChange = (e) => {
    const { name, value } = e.target;
    setData({ ...data, [name]: value });
  }

```

```

    });
    const handleSubmit = (e) => {
      e.preventDefault();
      if (!data?.email || !data?.password) {
        return alert("Please fill all fields");
      } else {
        axiosInstance.post('/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");
          });
      }
    });
    return (
      <
        <Navbar expand="lg" className="bg-body-tertiary">
          <Container fluid>
            <Navbar.Brand><h2>Study App</h2></Navbar.Brand>
            <Navbar.Toggle aria-controls="navbarScroll" />
            <Navbar.Collapse id="navbarScroll">
              <Nav
                className="me-auto my-2 my-lg-0"
                style={{ maxHeight: '100px' }}
                navbarScroll
              >
                </Nav>
                <Nav>
                  <Link to="/">Home</Link>
                  { /* <Link to="/about">About</Link> */ }
                  <Link to="/login">Login</Link>
                  <Link to="/register">Register</Link>
                </Nav>

                </Navbar.Collapse>
              </Container>
            </Navbar>
            <div className='first-container'>
              <Container component="main" style={{ display: 'flex', justifyContent: 'center', alignItems:
'center' }} >

```

```

<Box
  sx={{
    marginTop: 8,
    marginBottom: 4,
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
    padding: '10px',
    background: '#dddde8db',
    border: '1px solid lightblue',
    borderRadius: '5px'
  }}
>
  <Avatar sx={{ bgcolor: 'secondary.main' }}>
</Avatar>
  <Typography component="h1" variant="h5">
    Sign In
  </Typography>
  <Box component="form" onSubmit={handleSubmit} noValidate>
    <TextField
      margin="normal"
      fullWidth
      id="email"
      label="Email Address"
      name="email"
      value={data.email}
      onChange={handleChange}
      autoComplete="email"
      autoFocus
    />
    <TextField
      margin="normal"
      fullWidth
      name="password"
      value={data.password}
      onChange={handleChange}
      label="Password"
      type="password"
      id="password"
      autoComplete="current-password"
    />
    <Box mt={2}>
      <Button
        type="submit"
        variant="contained"
        sx={{ mt: 3, mb: 2 }}
        style={{ width: '200px' }}
      >
        Sign In
      </Button>
    </Box>
    <Grid container>
      <Grid item>Have an account?

```

```

        <Link style={{ color: "blue" }} to={"/register"} variant="body2">
          {" Sign Up"}
        </Link>
      </Grid>
    </Grid>
  </Box>
</Box>
</Container>
</div>
</>
)
}

```

export default Login

Frontend > src > Components > common > Home.jsx

```

import React from 'react'
import { Link } from 'react-router-dom'
import { Container, Nav, Button, Navbar } from 'react-bootstrap';
import AllCourses from './AllCourses';

const Home = () => {
  return (
    <
      <Navbar expand="lg" className="bg-body-tertiary">
        <Container fluid>
          <Navbar.Brand><h2>Study App</h2></Navbar.Brand>
          <Navbar.Toggle aria-controls="navbarScroll" />
          <Navbar.Collapse id="navbarScroll">
            <Nav
              className="me-auto my-2 my-lg-0"
              style={{ maxHeight: '100px' }}
              navbarScroll
            >
              </Nav>
            <Nav>
              <Link to={"/"}>Home</Link>
              <Link to={"/login"}>Login</Link>
              <Link to={"/register"}>Register</Link>
            </Nav>
          </Navbar.Collapse>
        </Container>
      </Navbar>
      <div id="home-container" className="first-container">
        <div className="content-home">
          <p>Small App, Big Dreams: <br /> Elevating Your Education</p>
          <Link to={"/register"}><Button variant="warning" className="m-2" size="md">Explore
Courses</Button></Link>
        </div>
      </div>
      <Container className="second-container">
        <h2 className="text-center my-4">Trending Courses</h2>

```



```

        <AllCourses />
      </Container>
    </>
  )
}

```

export default Home

Frontend > src > Components > common > Dashboard.jsx

```

// import React, { useState } from 'react';
// import NavBar from './NavBar';
// import UserHome from './UserHome'
// import { Container } from 'react-bootstrap';
// import AddCourse from '../user/teacher/AddCourse';
// const Dashboard = () => {
//   const [selectedComponent, setSelectedComponent] = useState('home');
//   const renderSelectedComponent = () => {
//     switch (selectedComponent) {
//       case 'home':
//         return <UserHome />
//       case 'addcourse':
//         return <AddCourse />
//       default:
//         return <UserHome />
//     }
//   };
//   return (
//     <
//       <NavBar setSelectedComponent={setSelectedComponent} />
//       <Container className='my-3'>
//         {renderSelectedComponent()}
//       </Container>
//     </>
//   );
// };

```

// export default Dashboard;

```

import React, { useContext, useState } from 'react';
import NavBar from './NavBar';
import UserHome from './UserHome'
import { Container } from 'react-bootstrap';
import AddCourse from '../user/teacher/AddCourse';
import StudentHome from '../user/student/StudentHome';
import AdminHome from '../admin/AdminHome';
import { UserContext } from '../App';
import EnrolledCourses from '../user/student/EnrolledCourses';
import CourseContent from '../user/student/CourseContent';
import AllCourses from '../admin/AllCourses';
const Dashboard = () => {
  const user = useContext(UserContext)
  const [selectedComponent, setSelectedComponent] = useState('home');
  const renderSelectedComponent = () => {

```

```

switch (selectedComponent) {
  case 'home':
    return <UserHome />
  case 'addcourse':
    return <AddCourse />
  case 'enrolledcourse':
    return <EnrolledCourses />
  case 'coursreSection':
    return <CourseContent />
  case 'cousres':
    return <AllCourses />
  default:
    return <UserHome />
}
};
return (
  <
    <NavBar setSelectedComponent={setSelectedComponent} />
    <Container className='my-3'>
      {renderSelectedComponent()}
    </Container>
  </>
);
};

export default Dashboard;

```

Frontend > src > Components > common > AxiosInstance.jsx

```

import axios from 'axios';
const axiosInstance = axios.create({
  baseURL: 'http://localhost:8000',
});
export default axiosInstance;

```

Frontend > src > Components > common > AllCourses.jsx

```

import React, { useState, useEffect, useContext } from 'react';
import axiosInstance from './AxiosInstance';
import { Button, Modal, Form } from 'react-bootstrap';
import { UserContext } from '../App';
import { Link, useNavigate } from 'react-router-dom';
import {
  MDBCol,
  MDBInput,
  MDBRow,
} from "mdb-react-ui-kit";
const AllCourses = () => {
  const navigate = useNavigate()
  const user = useContext(UserContext)
  const [allCourses, setAllCourses] = useState([]);
  const [filterTitle, setFilterTitle] = useState("");
  const [filterType, setFilterType] = useState("");

```

```

const [showModal, setShowModal] = useState(Array(allCourses.length).fill(false));
const [cardDetails, setCardDetails] = useState({
  cardholdername: "",
  cardnumber: "",
  cvvcode: "",
  expmonthyear: "",
});
const handleChange = (e) => {
  setCardDetails({ ...cardDetails, [e.target.name]: e.target.value })
}
const handleShow = (courseIndex, coursePrice, courseId, courseTitle) => {
  if (coursePrice === 'free') {
    handleSubmit(courseId)
    return navigate(`/courseSection/${courseId}/${courseTitle}`)
  } else {
    const updatedShowModal = [...showModal];
    updatedShowModal[courseIndex] = true;
    setShowModal(updatedShowModal);
  }
};
// Function to handle closing the modal for a specific course
const handleClose = (courseIndex) => {
  const updatedShowModal = [...showModal];
  updatedShowModal[courseIndex] = false;
  setShowModal(updatedShowModal);
};
const getAllCoursesUser = async () => {
  try {
    const res = await axiosInstance.get(`api/user/getallcourses`, {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('token')}`,
      },
    });
    if (res.data.success) {
      setAllCourses(res.data.data);
    }
  } catch (error) {
    console.log('An error occurred:', error);
  }
};
useEffect(() => {
  getAllCoursesUser();
}, []);
const isPaidCourse = (course) => {
  // Check if C_price contains a number
  return /^d/.test(course.C_price);
};
const handleSubmit = async (courseId) => {
  try {
    const res = await axiosInstance.post(`api/user/enrolledcourse/${courseId}`, cardDetails, {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('token')}`,
      },
    });
  }
});

```

```

    if (res.data.success) {
      alert(res.data.message);
      navigate(`/courseSection/${res.data.course.id}/${res.data.course.Title}`);
    } else {
      alert(res.data.message);
      navigate(`/courseSection/${res.data.course.id}/${res.data.course.Title}`);
    }
  } catch (error) {
    console.log('An error occurred:', error);
  }
}
}
return (
  <div className=" mt-4 filter-container text-center">
    <p className="mt-3">Serach By: </p>
    <input
      type="text"
      placeholder="title"
      value={filterTitle}
      onChange={(e) => setFilterTitle(e.target.value)}
    />
    <select value={filterType} onChange={(e) => setFilterType(e.target.value)}>
      <option value="">All Courses</option>
      <option value="Paid">Paid</option>
      <option value="Free">Free</option>
    </select>
  </div>
  <div className='p-2 course-container'>
    {allCourses?.length > 0 ? (
      allCourses
        .filter(
          (course) =>
            filterTitle === " ||
            course.C_title?.toLowerCase().includes(filterTitle?.toLowerCase())
        )
        .filter((course) => {
          if (filterType === 'Free') {
            return !isPaidCourse(course);
          } else if (filterType === 'Paid') {
            return isPaidCourse(course);
          } else {
            return true;
          }
        })
        .map((course, index) => (
          <div key={course._id} className='course'>
            <div className="card1">
              <div className="desc">
                <h3>Modules</h3>
                {course.sections.length > 0 ? (
                  course.sections.slice(0, 2).map((section, i) => (
                    <div key={i}>
                      <p><b>Title:</b> {section.S_title}</p>
                      <div className="description-container">
                        <div className="description">

```

```

        <b>Description:</b> {section.S_description}
    </div>
</div>
<hr />
</div>
))
):(
    <p>No Modules</p>
)}
<p style={{ fontSize: 20, fontWeight: 600 }}>many more to watch..</p>
</div>
<div className="details">
    <div className="center">
        <h1>
            {course.C_title}<br />
            <span>{course.C_categories}</span><br />
            <span style={{ fontSize: 10 }}>by: &nbsp;{course.C_educator}</span>
        </h1>
        <p>Sections: {course.sections.length}</p>
        <p>Price(Rs.): {course.C_price}</p>
        <p>Enrolled students: {course.enrolled}</p>
        {user.userLoggedIn === true ?
            <
                <Button
                    className=""
                    variant='outline-dark'
                    size='sm'
                    onClick={() => handleShow(index, course.C_price, course._id,
course.C_title)}
                >
                    Start Course
                </Button>
                <Modal show={showModal[index]} onHide={() => handleClose(index)}>
                    <Modal.Header closeButton>
                        <Modal.Title>
                            Payment for {course.C_title} Course
                        </Modal.Title>
                    </Modal.Header>
                    <Modal.Body>
                        <p style={{ fontSize: 15 }}>Educator: {course.C_educator}</p>
                        <p style={{ fontSize: 15 }}>Price: {course.C_price}</p>
                        <Form onSubmit={(e) => {
                            e.preventDefault()
                            handleSubmit(course._id)
                        }}>
                            <MDBInput className='mb-2' label="Card Holder Name"
name='cardholdername' value={cardDetails.cardholdername} onChange={handleChange} type="text"
size="md"
                                placeholder="Cardholder's Name" contrast required />
                            <MDBInput className='mb-2' name='cardnumber'
value={cardDetails.cardnumber} onChange={handleChange} label="Card Number" type="number"
size="md"
                                minLength="0" maxLength="16" placeholder="1234 5678 9012 3457"
required />
                        </Form>
                    </Modal.Body>
                </Modal>
            </
        </div>
    </div>

```

```

        <MDBRow className="mb-4">
            <MDBCol md="6">
                <MDBInput name='expmonthyear'
value={cardDetails.expmonthyear} onChange={handleChange} className="mb-2"
label="Expiration" type="text" size="md"
                placeholder="MM/YYYY" required />
            </MDBCol>
            <MDBCol md="6">
                <MDBInput name='cvvcode' value={cardDetails.cvvcode}
onChange={handleChange} className="mb-2" label="Cvv" type="number" size="md"
minLength="3"
                maxLength="3" placeholder="&#9679;&#9679;&#9679;" required
            />
        </MDBCol>
    </MDBRow>
    <div className="d-flex justify-content-end">
        <Button className='mx-2' variant="secondary" onClick={() =>
handleClose(index)}>
            Close
        </Button>
        <Button variant="primary" type='submit'>
            Pay Now
        </Button>
    </div>
</Form>
</Modal.Body>
</Modal>
</>
: <Link to={'/login'}><Button
    className=""
    variant='outline-dark'
    size='sm'
    >
        Start Course
    </Button></Link>
</div>
</div>
</div>
</div>
))
): (
    <p>No courses at the moment</p>
)
</div>
</>
);
};

export default AllCourses;

```

```

import React, { useState, useEffect } from 'react'
import { Button, styled, TableRow, TableHead, TableContainer, Paper, Table, TableBody, TableCell,
tableCellClasses } from '@mui/material'
import axiosInstance from '../common/AxiosInstance'
const StyledTableCell = styled(TableCell)(({ theme }) => ({
  [`&.${tableCellClasses.head}`]: {
    backgroundColor: theme.palette.common.black,
    color: theme.palette.common.white,
  },
  [`&.${tableCellClasses.body}`]: {
    fontSize: 14,
  },
}))
const StyledTableRow = styled(TableRow)(({ theme }) => ({
  '&:nth-of-type(odd)': {
    backgroundColor: theme.palette.action.hover,
  },
  // hide last border
  '&:last-child td, &:last-child th': {
    border: 0,
  },
}))
const AllCourses = () => {
  const [allCourses, setAllCourses] = useState([])
  const allCoursesList = async () => {
    try {
      const res = await axiosInstance.get('api/admin/getallcourses', {
        headers: {
          "Authorization": `Bearer ${localStorage.getItem("token")}`
        }
      })
      if (res.data.success) {
        setAllCourses(res.data.data)
      }
      else {
        alert(res.data.message)
      }
    } catch (error) {
      console.log(error);
    }
  }
  useEffect(() => {
    allCoursesList(),
  }, [])
  const deleteCourse = async (courseId) => {
    const confirmation = confirm('Are you sure you want to delete')
    if (!confirmation) {
      return;
    }
    try {
      const res = await axiosInstance.delete(`api/user/deletecourse/${courseId}`, {
        headers: {
          Authorization: `Bearer ${localStorage.getItem('token')}`,
          /* <Button size='small' color='info'>Update</Button> */
        },
      })
    }
  }
}

```

```

    })
    if (res.data.success) {
      alert(res.data.message)
      allCoursesList()
    } else {
      alert("Failed to delete the course")
    }
  } catch (error) {
    console.log('An error occurred:', error);
  }
}
return (
  <TableContainer component={Paper}>
    <Table sx={{ minWidth: 700 }} aria-label="customized table">
      <TableHead>
        <TableRow>
          <StyledTableCell>Course ID</StyledTableCell>
          <StyledTableCell align="center">Course Name</StyledTableCell>
          <StyledTableCell align="left">Course Educator</StyledTableCell>
          <StyledTableCell align="center">Course Category</StyledTableCell>
          <StyledTableCell align="left">Course Price</StyledTableCell>
          <StyledTableCell align="left">Course Sections</StyledTableCell>
          <StyledTableCell align="left">Enrolled Students</StyledTableCell>
          <StyledTableCell align="center">Action</StyledTableCell>
        </TableRow>
      </TableHead>
      <TableBody>
        {
          allCourses.length > 0 ? (
            allCourses.map((Course) => (
              <StyledTableRow key={Course._id}>
                <StyledTableCell component="th" scope="row">
                  {Course._id}
                </StyledTableCell>
                <StyledTableCell align="center" component="th" scope="row">
                  {Course.C_title}
                </StyledTableCell>
                <StyledTableCell align="center" component="th" scope="row">
                  {Course.C_educator}
                </StyledTableCell>
                <StyledTableCell align="center" component="th" scope="row">
                  {Course.C_categories}
                </StyledTableCell>
                <StyledTableCell align="center" component="th" scope="row">
                  {Course.C_price}
                </StyledTableCell>
                <StyledTableCell align="center" component="th" scope="row">
                  {Course.sections.length}
                </StyledTableCell>
                <StyledTableCell align="center" component="th" scope="row">
                  {Course.enrolled}
                </StyledTableCell>
                <StyledTableCell align="center" component="th" scope="row">
                  <Button onClick={() => deleteCourse(Course._id)} size='small'
color="error">Delete</Button>

```



```

        </StyledTableCell>
      </StyledTableRow>
    )))
    :
    (<p className='px-2'>No users found</p>)
  }
</TableBody>
</Table>
</TableContainer>
)
}

```

export default AllCourses

Frontend > src > Components > admin > AdminHome.jsx

```

import React, { useState, useEffect } from 'react'
import { Button, styled, TableRow, TableHead, TableContainer, Paper, Table, TableBody, TableCell,
tableCellClasses } from '@mui/material'
import axiosInstance from '../common/AxiosInstance'

const StyledTableCell = styled(TableCell)(({ theme }) => ({
  ['&.${tableCellClasses.head}']: {
    backgroundColor: theme.palette.common.black,
    color: theme.palette.common.white,
  },
  ['&.${tableCellClasses.body}']: {
    fontSize: 14,
  },
}));

const StyledTableRow = styled(TableRow)(({ theme }) => ({
  '&:nth-of-type(odd)': {
    backgroundColor: theme.palette.action.hover,
  },
  // hide last border
  '&:last-child td, &:last-child th': {
    border: 0,
  },
}));

const AdminHome = () => {
  const [allUsers, setAllUsers] = useState([])
  const allUsersList = async () => {
    try {
      const res = await axiosInstance.get('api/admin/getallusers', {
        headers: {
          "Authorization": `Bearer ${localStorage.getItem("token")}`
        }
      })
      if (res.data.success) {
        setAllUsers(res.data.data)
      } else {
        alert(res.data.message)
      }
    } catch (error) {

```

```

        console.log(error);
    }
}
useEffect(() => {
    allUsersList()
}, [])
const deleteUser = async (userId) => {
    const confirmation = confirm('Are you sure you want to delete')
    if (!confirmation) {
        return;
    }
    try {
        const res = await axiosInstance.delete(`api/user/deleteuser/${userId}`, {
            headers: {
                Authorization: `Bearer ${localStorage.getItem('token')}`,
            },
        })
        if (res.data.success) {
            alert(res.data.message)
            allUsersList()
        } else {
            alert("Failed to delete the user")
        }
    } catch (error) {
        console.log('An error occurred:', error);
    }
}
return (
    <TableContainer component={Paper}>
        <Table sx={{ minWidth: 700 }} aria-label="customized table">
            <TableHead>
                <TableRow>
                    <StyledTableCell>User ID</StyledTableCell>
                    <StyledTableCell align="left">User Name</StyledTableCell>
                    <StyledTableCell align="left">Email</StyledTableCell>
                    <StyledTableCell align="left">Type</StyledTableCell>
                    <StyledTableCell align="left">Action</StyledTableCell>
                </TableRow>
            </TableHead>
            <TableBody>
                {
                    allUsers.length > 0 ? (
                        allUsers.map((user) => (
                            <StyledTableRow key={user._id}>
                                <StyledTableCell component="th" scope="row">
                                    {user._id}
                                </StyledTableCell>
                                <StyledTableCell component="th" scope="row">
                                    {user.name}
                                </StyledTableCell>
                                <StyledTableCell component="th" scope="row">
                                    {user.email}
                                </StyledTableCell>
                                <StyledTableCell component="th" scope="row">

```

```

        {user.type}
      </StyledTableCell>
      <StyledTableCell component="th" scope="row">
        <Button onClick={() => deleteUser(user._id)} size='small'
color="error">Delete</Button>
        {/* <Button size='small' color="info">Update</Button> */}
      </StyledTableCell>
    </StyledTableRow>
  )))
  :
  (<p className='px-2'>No users found</p>)
}
</TableBody>
</Table>
</TableContainer>
)
}

export default AdminHome

```

12. AUTHENTICATION AND AUTHORIZATION:

The authentication and authorization system in the Online Learning Platform ensures secure user access and protects sensitive data. Below is a breakdown of how these components are managed:

1. Authentication Method: JSON Web Tokens (JWT)

JWT Generation: Upon successful login, the server generates a JSON Web Token (JWT) for the user. This token is encoded and signed using a secure key, ensuring it can be verified by the server.

Token Structure: The JWT typically contains:

- ❖ **Header:** Specifies the signing algorithm (e.g., HS256).
- ❖ **Payload:** Includes user information like user ID and roles (e.g., admin, student).
- ❖ **Signature:** Ensures the token's integrity by signing the header and payload with a secret key.

Token Expiration: Tokens are set to expire after a defined period (e.g., 24 hours) to enhance security. After expiration, users are required to log in again.

Secure Storage:

- ❖ **Frontend:** Tokens are stored in the browser's localStorage or sessionStorage, providing easy retrieval for authorization in frontend requests.

❖ **Backend:** The server stores only the signing key, not the tokens themselves, which ensures minimal server storage for authentication.

2. Authorization with Role-Based Access Control (RBAC)

User Roles: Different roles (e.g., students, instructors, admins) are assigned specific access permissions.

❖ **Students:** Access only their own courses, enrollments, and discussions.

❖ **Instructors:** Manage their course content and view discussions for their courses.

❖ **Admins:** Have complete access to manage users, courses, and platform-wide content.

Role-Based Endpoint Protection: Each endpoint verifies the user's role based on the JWT payload. This way:

❖ Admins can access restricted endpoints like user management and course deletion.

❖ Students are restricted to actions related to their enrolled courses and profile.

3. Token-Based Authorization Process

Token Validation: Each request to a protected route includes the JWT in the request headers as a Bearer token (e.g., Authorization: Bearer <JWT>).

❖ **Frontend:** Automatically appends the token to requests that require authorization.

❖ **Backend:** Verifies the JWT with the secret key, checking the token's validity and user role.

Middleware for Protected Routes:

❖ A middleware function intercepts requests to protected routes, validates the JWT, and extracts the user data.

❖ If the token is invalid or expired, the middleware rejects the request and sends a 401 Unauthorized response.

❖ If valid, the user's data (user ID, role) is appended to the request object, enabling access control in subsequent route handling.

4. Logout

- **Token Revocation:**

❖ **Frontend:** Logging out involves removing the token from `localStorage` or `sessionStorage`, effectively ending the session.

❖ **Backend:** Although JWTs are stateless, tokens can be manually blacklisted on logout to prevent future use if required.

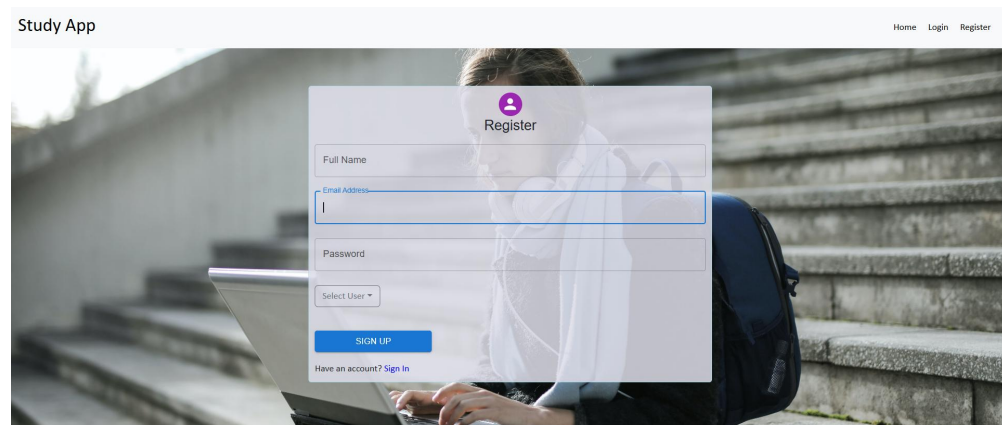
This setup ensures a robust authentication and authorization system, leveraging JWT for security and efficiency. By storing minimal information and only verifying JWTs without storing sessions, the OLP backend can remain scalable and efficient, while role-based controls ensure secure and restricted access to resources.

13. USER INTERFACE SCREENSHOTS :

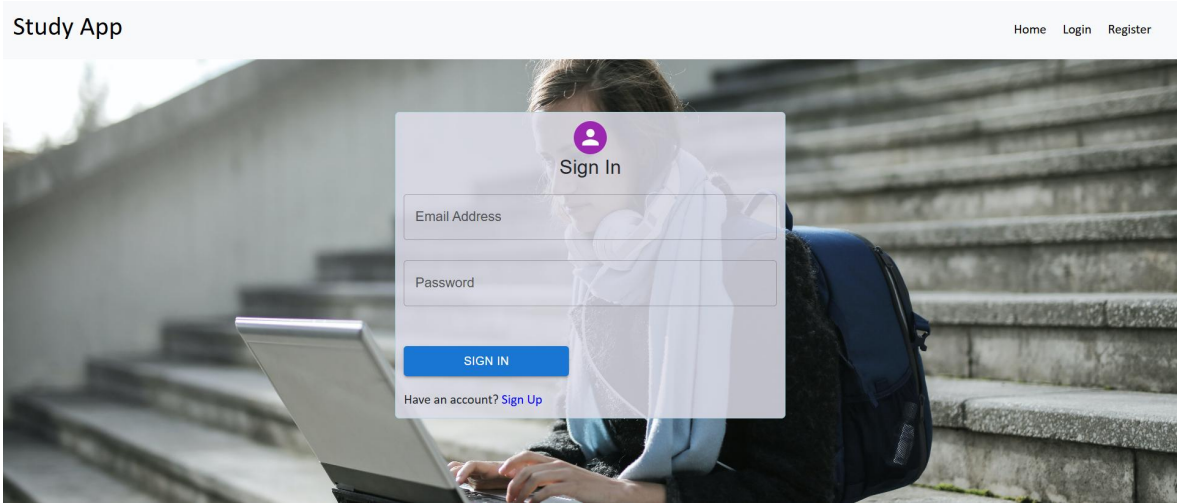
Student & teacher's Landing Page



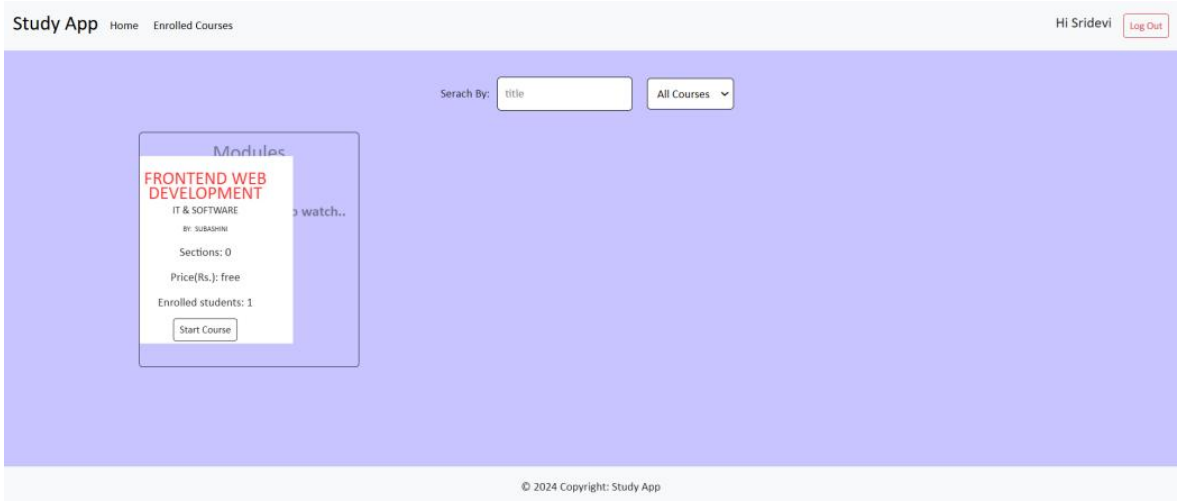
Student & teacher's Register Page



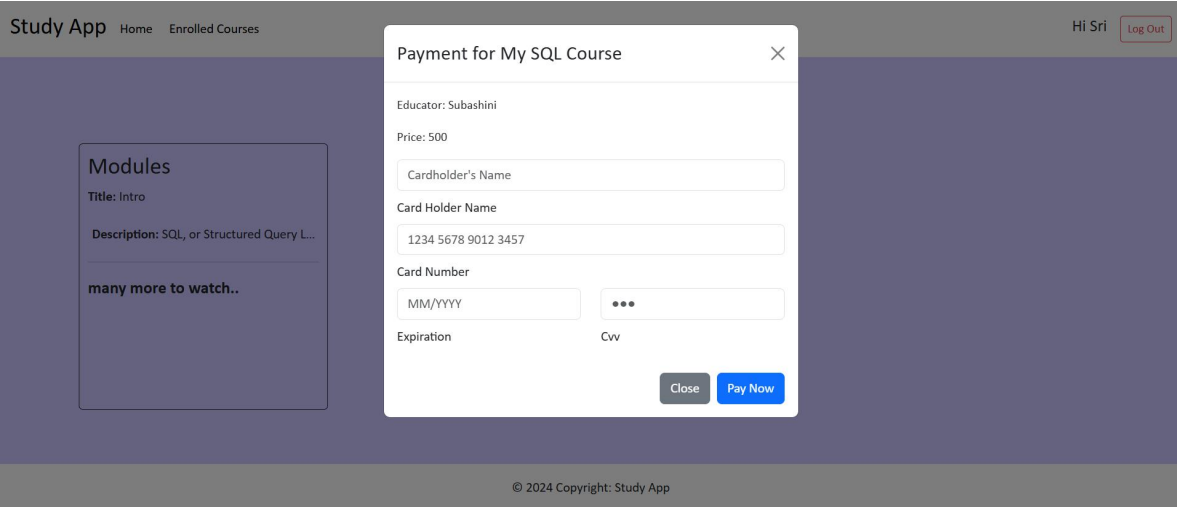
Student & teacher’s Login Page



Student Home Page



Payment Details



Student Enrolled Courses

Study App

HomeEnrolled Courses

Hi SrideviLog Out

Course ID	Course Name	Course Educator	Course Category	Action
672fb649224f8f1945d2b79b	Frontend Web Development	Subashini	IT & Software	GO TO
673083a65d498199b1409d05	Effective Communication Skills	Uma Maheshwari	Personal Development	GO TO
6730c01aa1ffac441db51d4b	My SQL	Swetha	IT & Software	GO TO

© 2024 Copyright: Study App

Student Course start

Study App

HomeEnrolled Courses


Hi SrideviLog Out

Welcome to the course: My SQL

Intro

SQL, or Structured Query Language, is a standardized programming language specifically designed for managing and manipulating relational databases. SQL enables users to perform tasks such as querying data, updating records, inserting new data, and deleting existing data in a database. [PLAY VIDEO](#)

DOWNLOAD CERTIFICATE



Download Certificate

Study App

HomeEnrolled Courses

Hi SrideviLog Out

Completion Certificate

Congratulations! You have completed all sections. Here is your certificate

Certificate of Completion

This is to certify that

Sridevi

has successfully completed the course

Frontend Web Development

on

11/10/2024

DOWNLOAD CERTIFICATE

© 2024 Copyright: Study App

Teacher Home Page

Study App

HomeAdd Course

Hi Sridevi

Log Out

My SQL

Description: SQL, or St [Read More](#)

Category: IT & Software

Sections: 1

Enrolled students: 1

Delete

© 2024 Copyright: Study App

Add Course

Study App

HomeAdd Course

Hi Sridevi

Log Out

Course Type

Select categories

Course Title

Enter Course Title

Course Educator

Enter Course Educator

Course Price(Rs.)

for free course, enter 0

Course Description

Enter Course description

+ Add Section

Submit

© 2024 Copyright: Study App

Study App

HomeAdd Course

Hi Sridevi

Log Out

Course Type

Select categories

Course Title

Enter Course Title

Course Educator

Enter Course Educator

Course Price(Rs.)

for free course, enter 0

Course Description

Enter Course description

Section Title

Enter Section Title

Section Content (Video or Image)

Choose FileNo file chosen

Section Description

Enter Section description

+ Add Section

Submit

14. TESTING :

Testing Strategy and Tools for the Online Learning Platform (OLP)

Testing is an essential part of ensuring the functionality, reliability, and security of the Online Learning Platform (OLP). Below is a detailed description of the testing strategy and tools used for the OLP project.

1. Types of Testing Implemented

a. Unit Testing

- **Purpose:** Test individual functions and components in isolation to ensure they behave as expected.
- **Frontend:**
 - ❖ Testing React components, such as course display, user authentication, form submissions, etc.
 - ❖ Ensuring functions like course filtering, search, and registration work as expected.
- **Backend:**
 - ❖ Testing backend API functions, such as user registration, course creation, user authentication, and payment handling.
 - ❖ Ensuring that API endpoints return correct status codes, data and handle errors gracefully.

b. Integration Testing

- **Purpose:** Validate that multiple components or services work together as expected.
- **Frontend:**
 - ❖ Verifying that React components interact correctly with API endpoints, e.g., fetching course data, enrolling in a course, and submitting assignments.
- **Backend:**
 - ❖ Testing API endpoints with real data, checking whether the backend correctly interacts with the database and returns expected results.

c. Functional Testing

- **Purpose:** Test the entire flow of the application to ensure it meets the user requirements.
- **Frontend:**
 - ❖ Simulating user actions such as logging in, browsing courses, enrolling, and interacting with course content.
 - ❖ Verifying UI elements like buttons, forms, and modals function as expected.
- **Backend:**
 - ❖ Verifying that all business logic works as expected, including authentication, database queries, and course management.

d. UI/UX Testing

- **Purpose:** Ensure the user interface is intuitive, responsive, and provides a smooth user experience.
- **Tools:**
 - ❖ **Cypress:** Used for end-to-end testing, ensuring the frontend behaves correctly in different browsers and screen sizes.
 - ❖ **Selenium WebDriver:** Automates user actions like login, browsing, and course interaction for cross-browser testing.
 - ❖ **Jest with React Testing Library:** For testing individual UI components and ensuring they render correctly and handle events properly.

e. Security Testing

- **Purpose:** Ensure the platform is secure, preventing common vulnerabilities like SQL injection, XSS, and CSRF.
- **Focus Areas:**
 - ❖ Testing the implementation of JWT tokens for user authentication.
 - ❖ Ensuring role-based access control (admin, instructor, student) is properly enforced.
 - ❖ Checking the security of sensitive data (passwords, payment information) and APIs.

- **Tools:**

- ❖ **OWASP ZAP:** Automated security scanning tool to check for vulnerabilities such as cross-site scripting (XSS), SQL injection, and insecure cookies.
- ❖ **Burp Suite:** Used to find security flaws in the platform, especially in the API layer.

f. Performance Testing

- **Purpose:** Test the platform's performance under various loads, ensuring it can handle multiple users simultaneously.

- **Tools:**

- ❖ **Apache JMeter:** Used for load testing by simulating multiple users interacting with the platform, helping to identify bottlenecks and performance issues.
- ❖ **Lighthouse (Chrome DevTools):** Measures web page performance, including page load times, resource utilization, and overall speed.

2. Tools Used for Testing

a. Frontend Testing Tools

- **Jest:** A JavaScript testing framework for writing unit and integration tests for React components.
 - ❖ Example: Testing React components like course listings, course detail view, user authentication forms, etc.
 - ❖ **React Testing Library:** Used alongside Jest to test React components by simulating user interactions (e.g., button clicks, form submissions).
- **Cypress:** A powerful end-to-end testing framework that automates UI testing to simulate real user behavior and test interactions with the platform.
- **Selenium WebDriver:** Used for cross-browser testing to ensure compatibility across different browsers like Chrome, Firefox, and Edge.

b. Backend Testing Tools

- **Mocha:** A flexible Node.js testing framework for writing backend tests.
 - ❖ Example: Testing API endpoints like course creation, user registration, and content retrieval.

- **Chai:** Assertion library used alongside Mocha to write backend tests. It allows developers to easily assert conditions and handle mock data.
- **Supertest:** A tool to test HTTP requests and responses. Used to test RESTful API endpoints in the backend.
- **Jest:** Can also be used on the backend to test Node.js functions, databases, and interactions with external services.

c. Security Testing Tools

- **OWASP ZAP (Zed Attack Proxy):** A popular open-source security testing tool that helps detect vulnerabilities like XSS, SQL injection, and weak authentication methods.
- **Burp Suite:** Another security tool used for scanning APIs and web applications for vulnerabilities and helping secure the platform.

d. Performance Testing Tools

- **Apache JMeter:** Performance testing tool used to simulate concurrent users and measure the platform's response under load. It helps identify system bottlenecks, slow database queries, and server limitations.
- **Lighthouse:** A tool integrated into Chrome DevTools that measures page load performance and suggests improvements to enhance user experience, including mobile responsiveness and accessibility.

3. Testing Process

Step 1: Write Unit Tests

- Developers write tests for individual functions and components. These tests cover the core functionality of both the frontend and backend. For instance, unit tests might cover how course data is fetched and displayed or how authentication works.

Step 2: Run Tests Locally

- Before committing changes, developers run tests in their local development environment using tools like Jest for React and Mocha for backend. This helps catch bugs early.

Step 3: Integration Testing

- Once unit tests are successful, integration tests are run to ensure different modules (like frontend and backend) work together seamlessly. For example, checking that the frontend can retrieve and display data from the backend.

Step 4: Functional Testing

- Functional tests are conducted to simulate real user actions such as logging in, searching for courses, enrolling, completing assignments, and receiving certificates.

Step 5: UI/UX Testing

- Automated UI tests are run using Cypress and Selenium to ensure that the platform is user-friendly and behaves correctly across different browsers and screen sizes.

Step 6: Security Testing

- The platform is tested for security vulnerabilities using tools like OWASP ZAP and Burp Suite. Penetration tests are performed to identify weaknesses in the authentication and authorization mechanisms, API security, and data protection.

Step 7: Performance Testing

- Load tests are executed using JMeter to simulate multiple users accessing the platform simultaneously. Performance metrics like response time, server resource usage, and scalability are measured.

Step 8: Bug Fixing and Optimization

- Any issues found during testing are fixed, and optimizations are made for performance, security, and usability.

4. Continuous Integration and Deployment (CI/CD)

- **CI Tools:** Jenkins or GitHub Actions are configured to automatically run tests whenever new code is pushed to the repository. This ensures that the platform is always tested for stability before deployment.
- **CD Tools:** Jenkins or Heroku Pipelines are used to deploy the platform automatically after successful tests, ensuring that the most stable version of the platform is available to users.

5. Manual Testing

- **Exploratory Testing:** Manual testing is performed to explore and test features that might not be covered by automated tests. Testers try various real-world scenarios to find edge cases.
- **User Acceptance Testing (UAT):** End-users test the platform to ensure it meets their needs, and usability feedback is gathered for further improvements.

This comprehensive testing strategy ensures the Online Learning Platform is robust, secure, and user-friendly, delivering a seamless experience for learners, instructors, and administrators.

15. DEMO LINK :

Click the below demo video link here to view full application process :

https://drive.google.com/file/d/1n2PcdCdaP-p_hvV3JMoqLepB3Q6GrMuX/view?usp=drive_link

16. KNOWN ISSUES :

While extensive testing has been conducted to ensure the functionality and stability of the Online Learning Platform (OLP), there are a few known issues that users and developers should be aware of. These issues are actively being worked on for resolution in upcoming releases.

1. Authentication Delays

- ❖ **Issue:** Occasionally, users experience a slight delay when logging in or signing up, especially during peak usage times.
- ❖ **Impact:** Users may have to wait for a few extra seconds before gaining access to the platform.
- ❖ **Status:** Investigating server-side performance optimizations to speed up the authentication process.

2. Payment Gateway Integration

- ❖ **Issue:** Some users have reported issues with completing payments via the integrated payment gateway, particularly during high-traffic periods.

- ❖ **Impact:** Users may experience errors when attempting to complete a transaction or enroll in paid courses.
- ❖ **Status:** Ongoing communication with the payment gateway provider for a smoother and more reliable payment experience.

3. Video Playback Issues

- ❖ **Issue:** Some video content on the platform experiences buffering or fails to load correctly, particularly on slower internet connections or older browsers.
- ❖ **Impact:** Users may experience interruptions in course videos or may not be able to view them at all.
- ❖ **Status:** Currently working on improving video delivery by using a more efficient content delivery network (CDN).

4. Search Function Inconsistencies

- ❖ **Issue:** The search function may not always return the most relevant or accurate results, especially when filtering by multiple categories (e.g., course difficulty or subject).
- ❖ **Impact:** Users may have difficulty finding the right courses or course materials.
- ❖ **Status:** Enhancing the search algorithm to provide better search results and optimize filtering options.

5. Course Completion Certificate Errors

- ❖ **Issue:** In rare cases, users have reported that they did not receive their course completion certificates even after successfully completing all course requirements.
- ❖ **Impact:** Students may be unable to download or access their certificates upon finishing a course.
- ❖ **Status:** Investigating the backend logic for generating certificates and implementing additional validation checks.

6. Course Enrollment Confirmation

- ❖ **Issue:** There have been reports that some users do not receive immediate confirmation emails or pop-up notifications after enrolling in a course.
- ❖ **Impact:** Users may be unsure whether their enrollment was successful or not.
- ❖ **Status:** Reviewing the email confirmation process and enhancing the user interface to display clear enrollment confirmation messages.

These are the currently known issues in the Online Learning Platform. We are actively addressing them to improve the overall user experience. Users are encouraged to report any new issues they encounter so that we can continue improving the platform.

17. FUTURE ENHANCEMENT :

While the Online Learning Platform (OLP) is functional and provides core features for learners and instructors, there are several potential future enhancements that could improve the user experience, expand the platform's capabilities, and address growing needs in the online education space. Below are some key future improvements and features that could be implemented:

1. Advanced AI-Driven Personalization

- **Feature:** Introduce AI-powered recommendations that tailor course suggestions based on user learning history, preferences, and performance.
- **Benefit:** Provides learners with more personalized content, ensuring they discover relevant courses that align with their interests and skill levels.
- **Details:** Use machine learning algorithms to analyze user behavior and suggest courses, articles, and videos to boost learning engagement.

2. Gamification of Learning

- **Feature:** Incorporate gamified elements such as leaderboards, points, badges, and achievements for learners who complete tasks, assignments, and courses.
- **Benefit:** Increases student motivation, making learning more interactive and fun.
- **Details:** Introduce challenges, quizzes, and competitions that reward users with points, badges, and rankings to foster a competitive learning environment.

3. Live Virtual Classes

- **Feature:** Implement live instructor-led sessions and real-time classes where students can interact with instructors and classmates via video.
- **Benefit:** Enhances the learning experience by providing real-time communication and fostering engagement between instructors and students.
- **Details:** Use video conferencing tools and integration with virtual classrooms, allowing students to ask questions, participate in discussions, and get immediate feedback.

4. Mobile App Development

- **Feature:** Develop native mobile apps for iOS and Android to provide better access to courses on the go.
- **Benefit:** Increases accessibility for learners who prefer using mobile devices and want to learn on the move.
- **Details:** The mobile app would provide a smooth and optimized learning experience, with offline support for accessing course materials without an internet connection.

5. Multi-Language Support

- **Feature:** Introduce multi-language support for the platform, enabling students and instructors to interact in their preferred language.
- **Benefit:** Expands the platform's reach to a global audience and improves accessibility for non-English speaking users.
- **Details:** Implement a language selector and translate course content, quizzes, and interface elements into popular languages.

6. Integration with Third-Party Tools

- **Feature:** Integrate the platform with external tools like Google Classroom, Zoom, or Slack to provide a more collaborative and seamless experience for students and instructors.
- **Benefit:** Enhances communication and collaboration, and ensures that users can use the tools they are already familiar with.
- **Details:** Implement APIs to integrate with popular education tools, enabling smooth data exchange and real-time collaboration.

7. Improved Analytics Dashboard

- **Feature:** Enhance the dashboard for both students and instructors with deeper analytics and insights into learning progress, trends, and areas for improvement.
- **Benefit:** Enables both learners and instructors to track performance and identify knowledge gaps.
- **Details:** Introduce visual data such as heatmaps, progress bars, and detailed charts that show students' strengths and weaknesses based on quiz results and assignment scores.

8. Blockchain for Certification Verification

- **Feature:** Use blockchain technology to issue and verify course completion certificates, ensuring their authenticity.
- **Benefit:** Prevents fraud and ensures that employers or educational institutions can trust the certification issued by the platform.
- **Details:** Integrate blockchain to securely store certificate data and provide a verified, immutable record of learners' achievements.

9. AI-Powered Tutoring and Chatbots

- **Feature:** Integrate AI chatbots and tutoring assistants that can provide real-time help to learners and answer questions on course materials.
- **Benefit:** Provides immediate assistance to learners, enhancing the learning experience with 24/7 support.
- **Details:** Develop intelligent tutoring systems using NLP (Natural Language Processing) to respond to learner queries, explain complex topics, and offer guidance.

The Online Learning Platform (OLP) has the potential to evolve significantly with these enhancements. As the demand for flexible and interactive online education grows, these features will improve user engagement, offer personalized learning experiences, and increase the overall value of the platform for both learners and instructors.

18. CONCLUSION :

The conclusion of this project focuses on integrating a full-stack application, combining a React front end with a Node.js and Express back end, connected to a MongoDB database. By implementing modular code structure, secure authentication, and efficient data handling, this project successfully delivers an online learning platform (OLP) that allows different user roles—admin, teacher, and student—to manage and access course-related functionalities.

Through the integration of various tools and libraries like **Axios** for HTTP requests, **Multer** for handling file uploads, **JWT** for authentication, and **Mongoose** for database management, the platform provides a seamless and responsive user experience. Additionally, security measures, such as using **bcryptjs** for password hashing and **CORS** for controlling cross-origin access, enhance the robustness and integrity of the system.

In conclusion, this project showcases an effective implementation of a scalable and interactive learning management system by utilizing modern web development practices and technologies. It serves as a foundation for further enhancements, such as incorporating additional features, refining the user interface, or optimizing performance for large-scale deployment.

Github Project Link : <https://github.com/sridevis15/Online-learning-platform>