MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE

Kodambakkam, Chennai-600024.

## DEPARTMENT OF INFORMATION TECHNOLOGY

# PROJECT NAME: STUDENT LEARNING PLATFORM USING MERN

PROJECT SUBMITTED BY,

| NAME | REG. NO. |
|---|---|
| AAFREIN A | 311520205001 |
| AISHWARYA.L | 311520205007 |
| JENIFER.M | 311520205024 |
| MANOJ KUMAR P | 311520205029 |

## TABLE OF CONTENTS

# INTRODUCTION

This online learning platform is a feature-rich web application developed using the MERN stack (MongoDB, Express.js, React.js, Node.js). It offers robust functionalities including user authentication, course creation, lecture management, enrollment, payment processing, and an interactive dashboard. Designed to streamline education delivery, it empowers educators and learners with a seamless and engaging digital experience.

## Purpose

The purpose of this project is to create a comprehensive and user-friendly online learning platform that bridges the gap between educators and learners. It aims to simplify the delivery of educational content, enhance engagement through interactive tools, and provide a scalable solution for managing courses, lectures, and student progress. By integrating essential features such as authentication, payment processing, and dashboards, the platform empowers instructors to focus on teaching while offering students a seamless and accessible learning experience.

## Scope of User Roles

### 1. Admin

- Full access to the platform with the ability to manage all aspects.
- Key Responsibilities:
  - Monitor overall platform performance and usage.
  - Manage users, including instructors and students (add, update, delete).
  - Oversee course content and approve or reject courses submitted by instructors.
  - Handle platform settings, including payment and notification configurations.
  - Generate reports on platform metrics such as revenue, active users, and course completion rates.

### 2. Instructor

- Authorized to create and manage educational content on the platform.
- Key Responsibilities:
  - Add, update, and delete courses and lectures.
  - View and manage enrolled students.
  - Track the performance of courses and student progress.
  - Interact with students through announcements and course updates.

- Monitor earnings and payment history.

**3. Student**

- Primary user of the platform, focused on accessing educational content.
- Key Responsibilities:
  - Browse and enroll in available courses.
  - Access lectures, assignments, and quizzes within enrolled courses.
  - Track progress and view course completion certificates (if applicable).
  - Manage personal profiles and payment history.
  - Provide feedback or ratings on courses.

**4. Guest**

- Users who have not registered or logged in.
- Key Responsibilities:
  - Browse available courses and platform features.
  - View course previews and instructor profiles.
  - Register or sign up to access additional features.

This structured role-based system ensures clear functionality boundaries, enhances security, and improves user experience by providing tailored features for each type of user.

**Technologies Used**

- **MongoDB:** NoSQL database for storing user, courses, and lecture data.
- **Express.js:** Web application framework for building the backend API and handling HTTP requests.
- **React.js**: JavaScript library for building the user interface with dynamic, reusable components.
- **Node.js**: JavaScript runtime for building the server-side logic of the application.

## PRIMARY OBJECTIVES

**1. Facilitate Seamless Learning:**

- Provide a user-friendly platform where students can easily access educational content and track their progress.

**2. Empower Educators:**

- Enable instructors to create, manage, and deliver courses and lectures efficiently with minimal technical complexity.

**3. Ensure Secure Access**:

- Implement robust authentication and authorization mechanisms to protect user data and restrict access based on roles.

**4. Streamline Payments and Enrollment**:

- Integrate secure payment systems to manage course fees and simplify the enrollment process for students.

**5. Enhance Engagement and Interaction:**

- Offer tools like dashboards, announcements, and progress tracking to foster better engagement between educators and learners.

**6. Support Scalability:**

- Design the platform to handle a growing number of users, courses, and concurrent activities without compromising performance.

**7. Promote Accessibility and Inclusivity:**

- Ensure the platform is accessible across devices, responsive in design, and easy to navigate for users with diverse needs and backgrounds.

## TECHNOLOGY STACK

| Category | Technology |
|---|---|
| Frontend | React.js, React Router, Axios, Bootstrap/CSS3 |
| Backend | Node.js, Express.js, JWT Authentication |
| Database | MongoDB, Mongoose |
| Deployment & Hosting | MongoDB Atlas |
| Version Control | Git, GitHub |
| Security | Bcrypt.js, CORS |
| Testing | Postman |

### Frontend Dependencies:

- **react:** Core library for building the user interface.
- **react-router-dom**: For routing and navigation between pages.
- **axios:** For making HTTP requests to interact with the backend API**.**

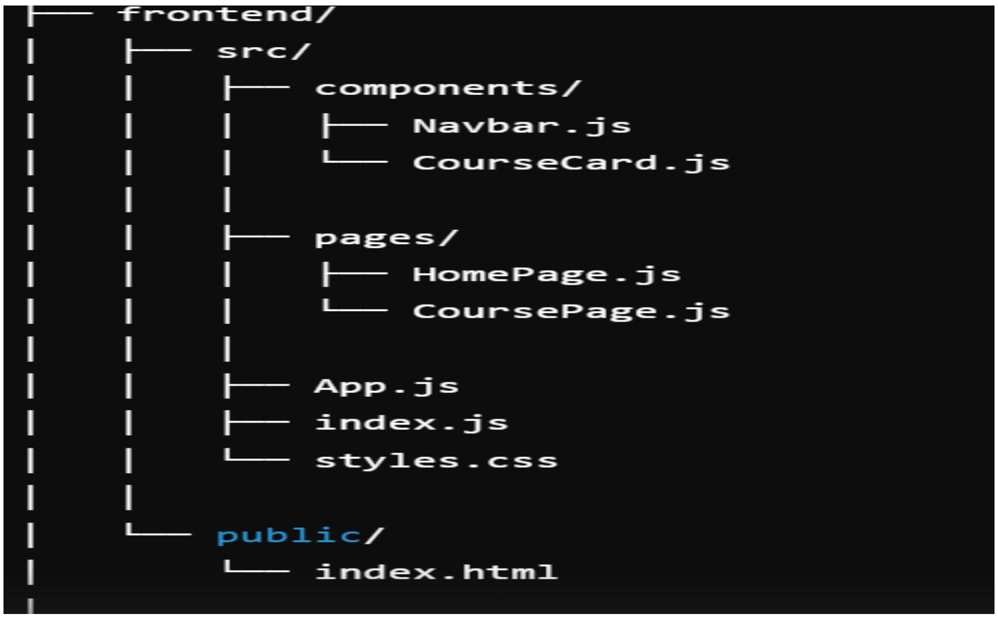
### Backend Dependencies:

- **express**: Web framework for Node.js to handle routing and API requests.
- **jsonwebtoken (JWT)**: For generating and verifying JSON Web Tokens for user authentication.
- **bcryptjs**: For securely hashing and comparing user passwords.
- **mongoose**: ODM library for interacting with MongoDB and managing data models and validation.

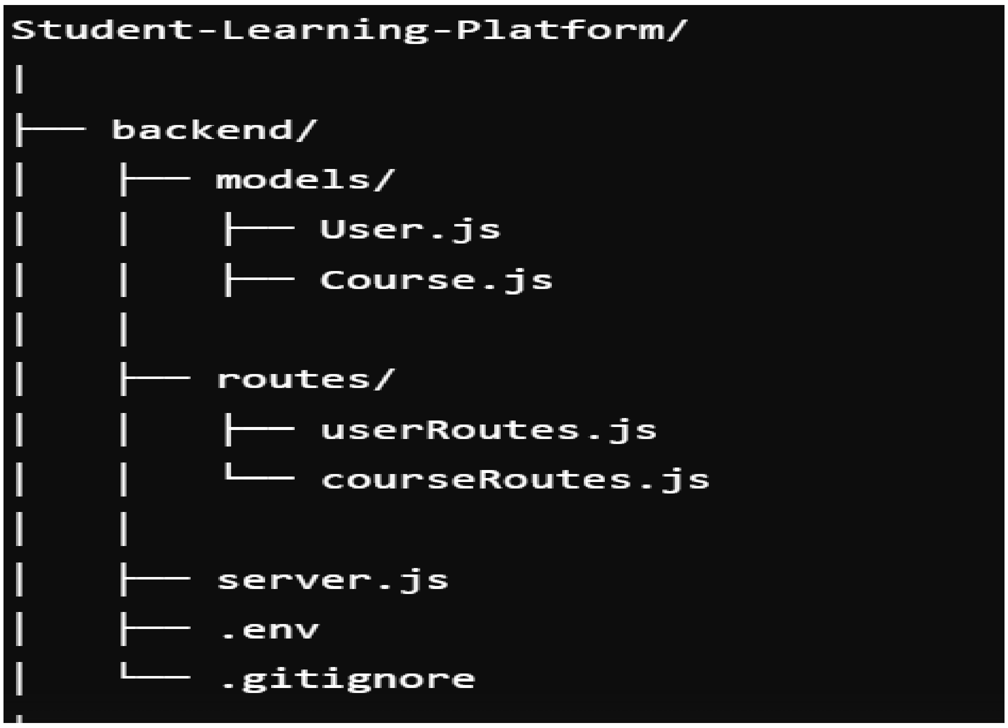
### Development Tools:

- **nodemon**: Automatically restarts the server during development when code changes are made.
- **dotenv**: For managing environment variables (e.g., API keys, database credentials).
- **postman**: For testing API endpoints during development.

## SYSTEM ARCHITECTURE



```
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   │   ├── Navbar.js
│   │   │   └── CourseCard.js
│   │   │
│   │   ├── pages/
│   │   │   ├── HomePage.js
│   │   │   └── CoursePage.js
│   │   │
│   │   ├── App.js
│   │   ├── index.js
│   │   └── styles.css
│   │
│   └── public/
│       └── index.html
│
```



```
Student-Learning-Platform/
│
├── backend/
│   ├── models/
│   │   ├── User.js
│   │   ├── Course.js
│   │
│   ├── routes/
│   │   ├── userRoutes.js
│   │   └── courseRoutes.js
│   │
│   ├── server.js
│   ├── .env
│   └── .gitignore
│
```

## BACKEND DEVELOPMENT

**Backend Development Summary for Online Learning Platform (MERN)**

**1. Authentication & Authorization**

- **JWT:** Secures routes and handles user authentication.
- **bcryptjs:** Hashes passwords for secure storage.
- **Role-based Access Control (RBAC):**
  - **Student:** Access course-related routes and profile.
  - **Teacher:** Manage courses and view enrollments.
  - **Admin:** Manage users, courses, and enrollments.

**2. Models**

- **User Model:** User details (name, email, password, role, enrolledCourses, etc.).
- **Course Model:** Course info (title, description, teacher, schedule, enrolledStudents, etc.).
- **Enrollment Model:** Links students to courses (studentId, courseId, status).
- **Teacher Model:** Teacher-specific details (bio, courses managed).

**3. Middleware**

- **authMiddleware.js:** Verifies JWT for authentication.
- **checkAdmin.js:** Grants access to admin-only routes.
- **checkTeacher.js:** Restricts access to teacher-specific routes.
- **checkStudent.js:** Restricts access to student-specific routes.
- **roleMiddleware.js:** Confirms user role for route access.

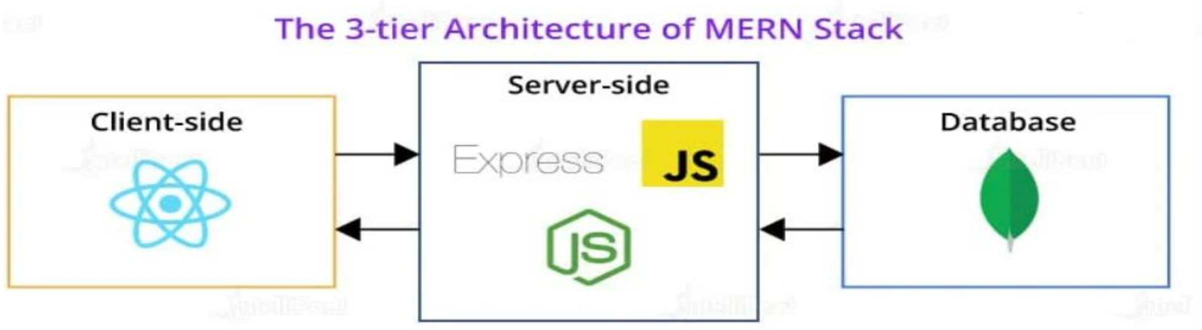
**4. Routes**

- **User Routes:** Register, login, and profile management.
- **Course Routes:** view, add, update, or delete courses (role-specific).
- **Enrollment Routes:** Enroll, update status, and view enrollments.
- **Teacher Routes:** Manage teachers and their courses.
- **Admin Routes:** Manage users, assign teachers, and monitor data.

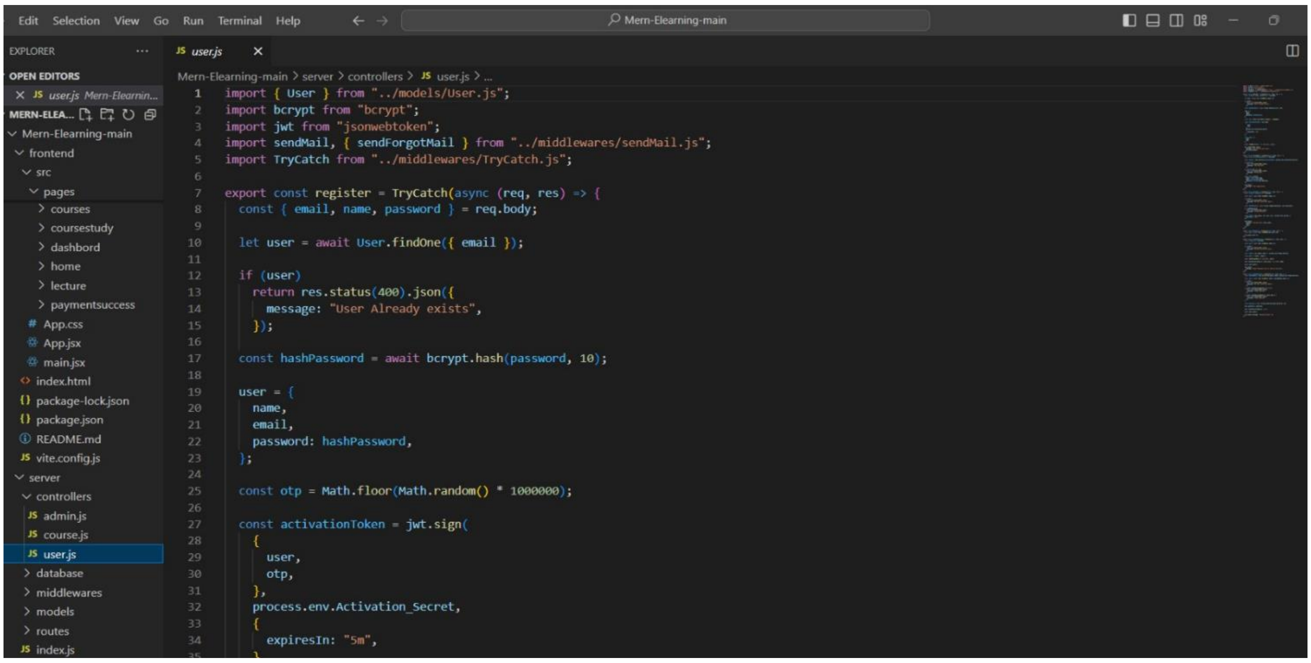
**5. Security**

- **JWT Authentication:** Protects sensitive routes, with tokens stored in frontend storage.
- **RBAC:** Role-based route access enforced via middleware.
- **Password Hashing:** Secure password storage using bcryptjs.
- **Data Validation:** Ensures proper input formats (e.g., email, password).

  This structure focuses on scalability, security, and clear role-based management.



## Server.js



```
import { User } from "../models/User.js";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import sendMail, { sendForgotMail } from "../middlewares/sendMail.js";
import TryCatch from "../middlewares/TryCatch.js";

export const register = TryCatch(async (req, res) => {
  const { email, name, password } = req.body;

  let user = await User.findOne({ email });

  if (user)
    return res.status(400).json({
      message: "User Already exists",
    });

  const hashPassword = await bcrypt.hash(password, 10);

  user = {
    name,
    email,
    password: hashPassword,
  };

  const otp = Math.floor(Math.random() * 1000000);

  const activationToken = jwt.sign(
    {
      user,
      otp,
    },
    process.env.Activation_Secret,
    {
      expiresIn: "5m",
```

1) **Dependencies**:

- **express:** Used to create the server and handle HTTP requests.
- **mongoose:** Used for interacting with the **MongoDB** database.
- **cors:** Middleware to allow cross-origin requests, enabling the frontend to communicate with the backend.
- **dotenv:** Loads environment variables from a .env file (e.g., database URL, port).

2) **Middleware**:

- **express.json():** Parses incoming requests with JSON payloads.
- **cors():** Allows the frontend (running on port 3000) to make requests to the backend.

3) **MongoDB Connection**:

- If successful, it logs a connection message; otherwise, it catches and logs any errors.

```
MONGO_URI=mongodb+srv://<username>:<password>@cluster0.mongodb.net/<database_name>?retryWrites=true&w=majority
```

4) **Routes**:

- /api/users: Manages user-related routes (e.g., registration, login).
- /api/users: Manages user-related routes (e.g., registration, login, profile).
- /api/courses: Manages course-related routes (e.g., listing, creation, updates).

- /api/enrollments: Manages enrollment-related routes (e.g., course enrollment, status updates).
- /api/teachers: Manages teacher-related routes (e.g., profile, course management).
- /api/admin: Manages admin-related routes (e.g., user and course management).

5) **Error Handling**:

- A global error handler catches unhandled errors and responds with a 500 status code and a generic error message.

6) **Server**:

- The application listens on the specified port (5000 by default) and logs a success message when the server is running.

## FRONTEND DEVELOPMENT

### Technologies Used

- **React.js**: A JavaScript library for building dynamic user interfaces with a component-based structure.
- **React Router**: A library for routing and navigation within the React application.
- **Axios**: A promise-based HTTP client for making requests to the backend API.
- **CSS**: For styling the application and creating a responsive design.
- **React Hooks**: For managing state and lifecycle methods in function components.
- **JWT**: For storing and managing user authentication tokens on the client-side.

### Key Features Implemented

**1.Landing Page:**

Displays options to Register or Login.

Navigates to corresponding pages.

**2.Registration and Login:**

User Registration: Collects name, email, password, and role (Admin, Teacher, Student).

User Login: Authenticates and stores JWT token for secure access.

**3.Dashboard:**

Student Dashboard: Lists courses and enrollments.

Teacher Dashboard: Manages courses and student progress.

Admin Dashboard: Views and manages users, courses, and enrollments.

**4.Role-Based Access:**

Custom views for Admin, Teacher, and Student based on roles.

**5.Course Management**:

Student: Browse, enroll, view courses.

Teacher: Create, update, and manage courses.

Admin: Manage users, courses, and enrollments**.**

## A. User Registration

The **User Registration** process allows a user to sign up and create an account. Once the registration is successful, the user can log in and access the **User Dashboard**.

### Core Steps:

**1 .User Registration**

The User Registration process allows a user to sign up and create an account. Once the registration is successful, the user can log in and access the User Dashboard.

Core Steps:

1. Input Fields:
   - o Name: Full name of the user.
   - o Email: Unique email address.
   - o Password: Strong password.

**2.Validation:**

- o Ensure that all required fields are filled.
- o Validate that the email is not already taken by another user..

**3.JWT Token Generation**:

After successful registration, a **JWT** token is generated, which is used for authenticating the user in subsequent requests.

**4.Response**:

- o After successful registration, the server returns a success message, the **JWT token**, and basic user details like name, email, and role

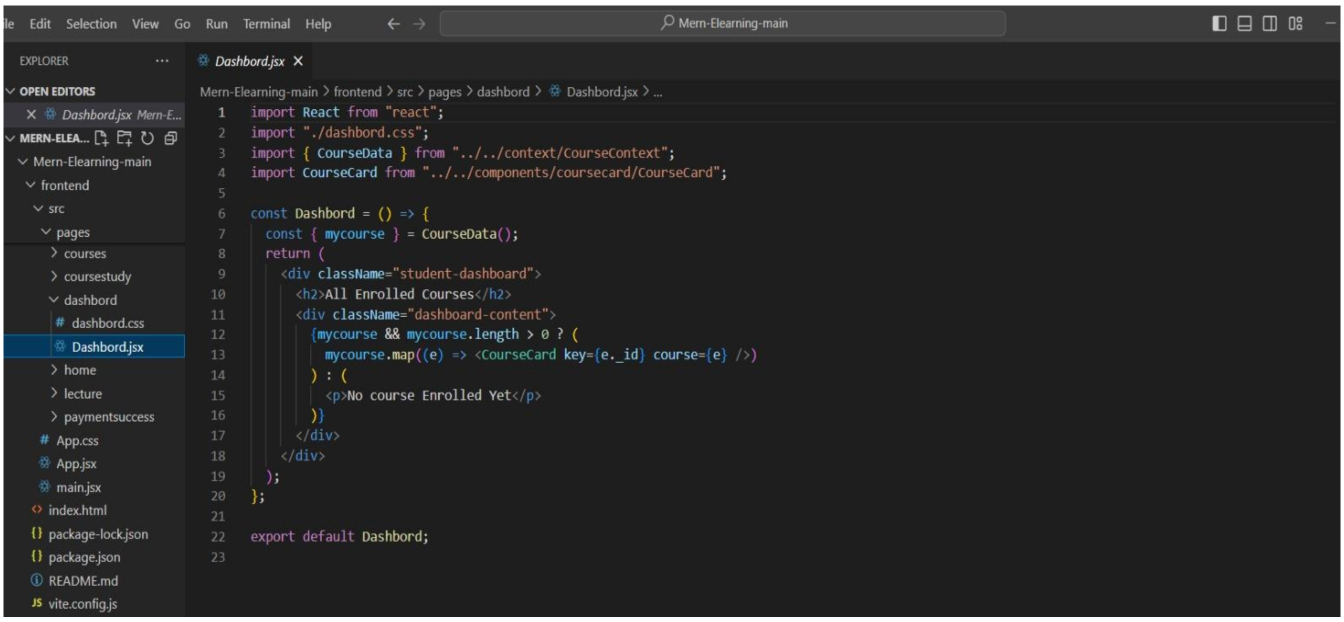
## B. User Login

The **User Login** process allows registered users to log in with their email and password. Upon successful authentication, a **JWT token** is returned, which grants access to protected routes (such as the User Dashboard).

### Core Steps:

1. **Input Fields**:
    - **Email**: The email used during registration.
    - **Password**: The password entered by the user during registration.

2. **Validation**:
    - Check that the email exists in the database.
    - Validate the password using **bcrypt** to compare the entered password with the hashed password in the database.
3. **JWT Token**:
    - On successful login, a **JWT** token is generated and sent back to the frontend. This token is used for authorizing further requests.
4. **Response**:
    - The server returns the **JWT token** and user details (name, email, role) in the response.



```
import React from "react";
import "./dashbord.css";
import { CourseData } from "../../context/CourseContext";
import CourseCard from "../../components/coursecard/CourseCard";

const Dashbord = () => {
  const { mycourse } = CourseData();
  return (
    <div className="student-dashboard">
      <h2>All Enrolled Courses</h2>
      <div className="dashboard-content">
        {mycourse && mycourse.length > 0 ? (
          mycourse.map((e) => <CourseCard key={e._id} course={e} />)
        ) : (
          <p>No course Enrolled Yet</p>
        )}
      </div>
    </div>
  );
};

export default Dashbord;
```

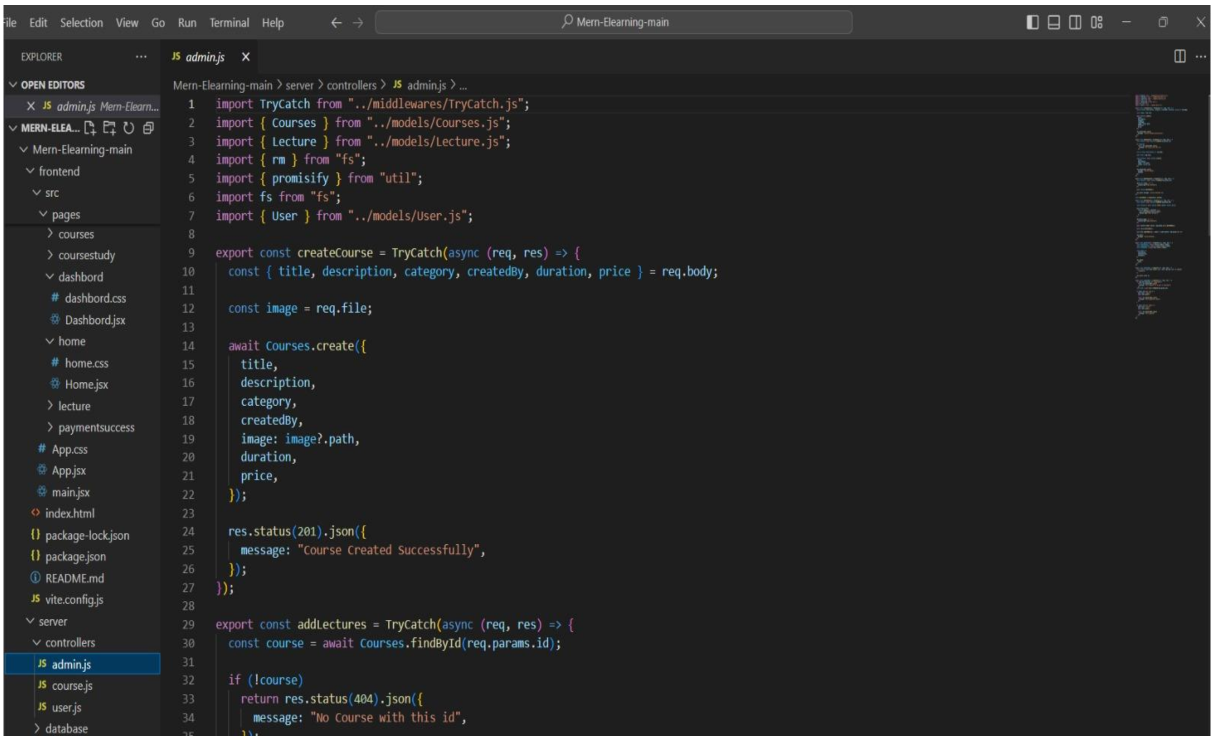
# AUTHENTICATION AND AUTHORIZATION

## 1. Authentication

**Authentication** is the process of verifying the identity of a user. In this application, users (patients, students, and subscribers) need to log in with valid credentials (email and password) to prove their identity.

### Steps Involved in Authentication:

1. **User Registration**:
   - The user provides their information (name, email, password).
   - The password is hashed using a secure hashing algorithm (e.g., bcrypt).
   - A **JWT token** is generated and sent to the user after successful registration.
   - The token is used for all future requests to verify the user's identity.
2. **User Login**:
   - The user enters their **email** and **password**.
   - The email is checked against the database to find a matching user.
   - The JWT token is then used to authenticate subsequent requests made by the user.
3. **JWT Token**:
   - The **JWT token** is a string that encodes the user's information (e.g., user ID, role) and has an **expiration time**.
   - The token is included in the **Authorization** header of every request made to protected routes.

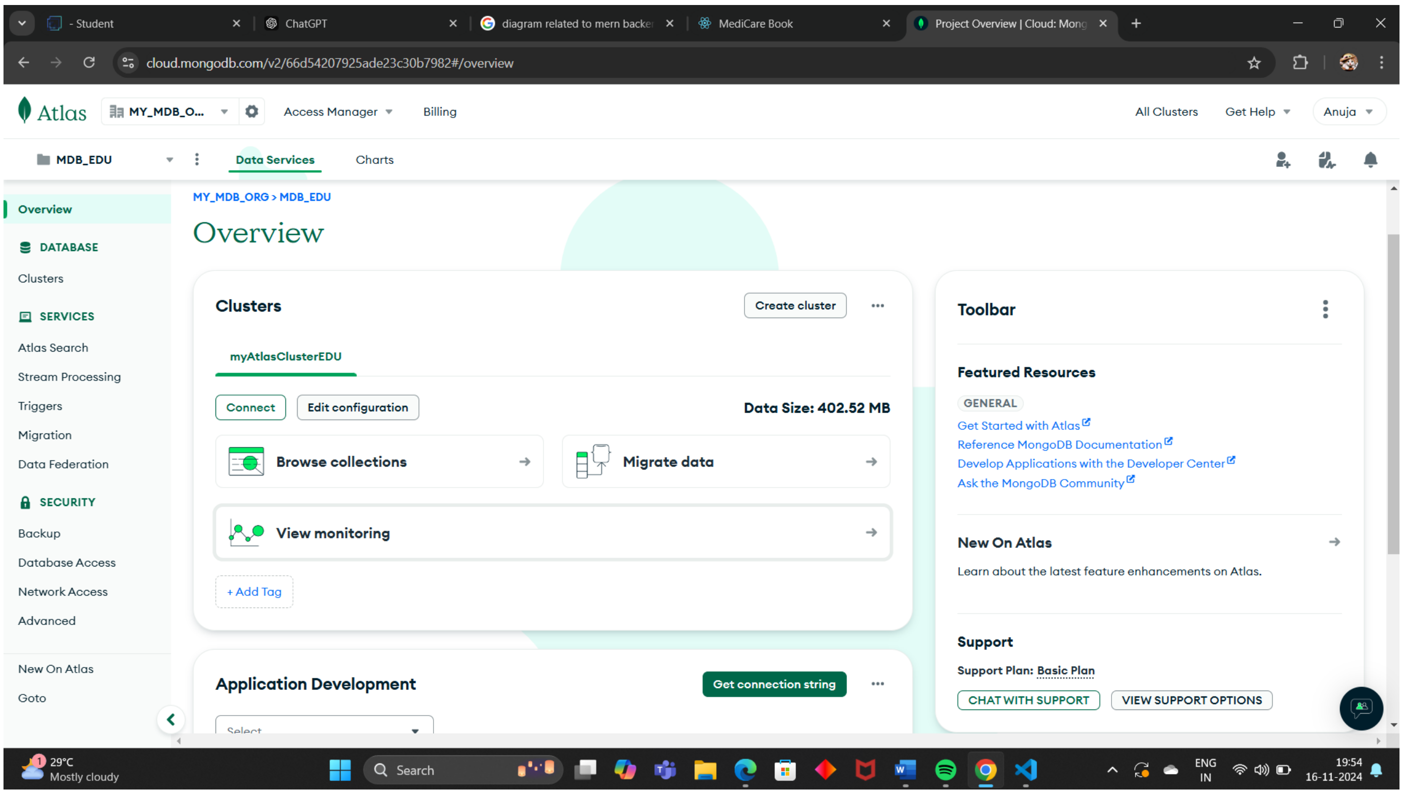


```javascript
import TryCatch from "../middlewares/TryCatch.js";
import { Courses } from "../models/Courses.js";
import { Lecture } from "../models/Lecture.js";
import { rm } from "fs";
import { promisify } from "util";
import fs from "fs";
import { User } from "../models/User.js";

export const createCourse = TryCatch(async (req, res) => {
  const { title, description, category, createdBy, duration, price } = req.body;

  const image = req.file;

  await Courses.create({
    title,
    description,
    category,
    createdBy,
    image: image?.path,
    duration,
    price,
  });

  res.status(201).json({
    message: "Course Created Successfully",
  });
});

export const addLectures = TryCatch(async (req, res) => {
  const course = await Courses.findById(req.params.id);

  if (!course)
    return res.status(404).json({
      message: "No Course with this id",
```

## MONGODB ATLAS DATABASE

MongoDB Atlas is a Database-as-a-Service (DBaaS) that allows users to set up and manage MongoDB databases in the cloud without the need for physical hardware or complex manual setups. It provides several features like:

- **Scalability**: Atlas automatically scales your MongoDB database to handle growing data requirements.
- **Security**: It offers robust security features like encryption, authentication, and network isolation.
- **Backup and Recovery**: Atlas provides automated backups and recovery options.
- **Monitoring and Alerts**: Atlas offers built-in monitoring and alerting features to track database performance.
- **Global Clusters**: Atlas supports multi-region clusters to improve data locality and reduce latency.



## USER INTERFACE DESIGN

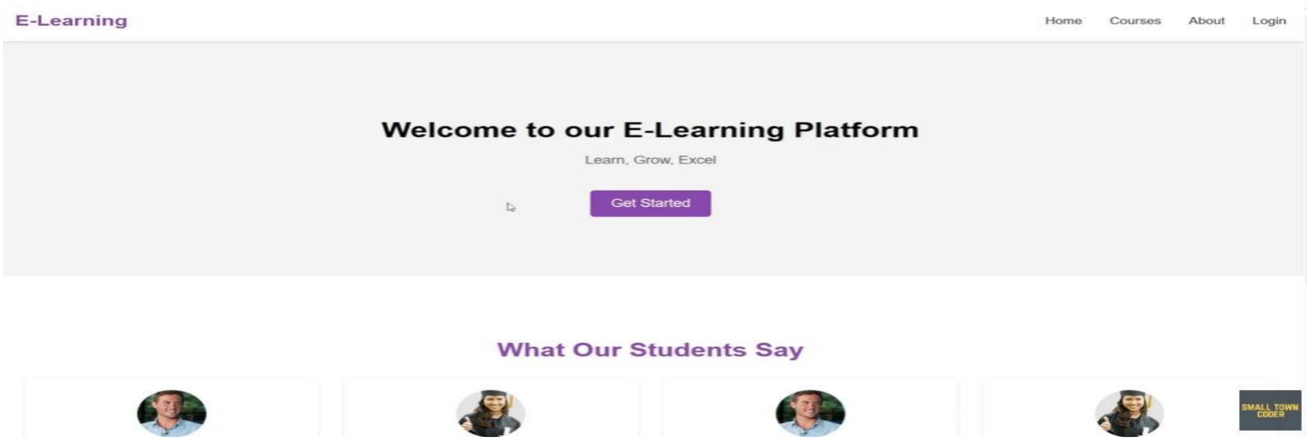
### 1. Landing Page (Common for All Users)

**Objective**: The landing page serves as the entry point of the application, where users can register or log in.

- **Features**:

- Registration Button: Redirects to the registration form.

- Login Button: Redirects to the login form.
- Welcome Message: A brief introduction, e.g., "Enroll in courses and enhance your skills."
- Minimalistic Design: Clean layout with easy navigation and clear Call-to-Action (CTA) buttons.



Welcome to our E-Learning Platform

Learn, Grow, Excel

What Our Students Say

**1. login Page (student/Teacher/Admin Registration)**

**Objective:** The Register page allows users to create a new account by providing basic information (name, email, password, and role).

**Input Fields:**

Name: User's full name.

Email: Unique email address.

Password: Secure password.

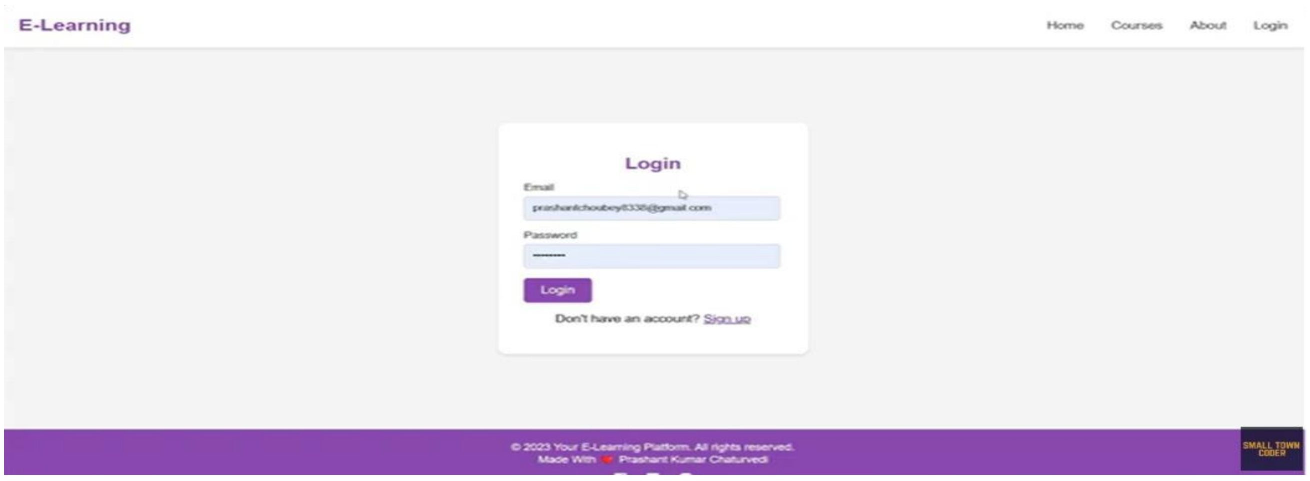
Confirm Password: To verify correct password entry.

Role: Dropdown to select role (Student, Teacher, Admin) for dashboard access.

Buttons:

Register: Submits form data and creates the account.

Already have an account?: Link to the Login page.

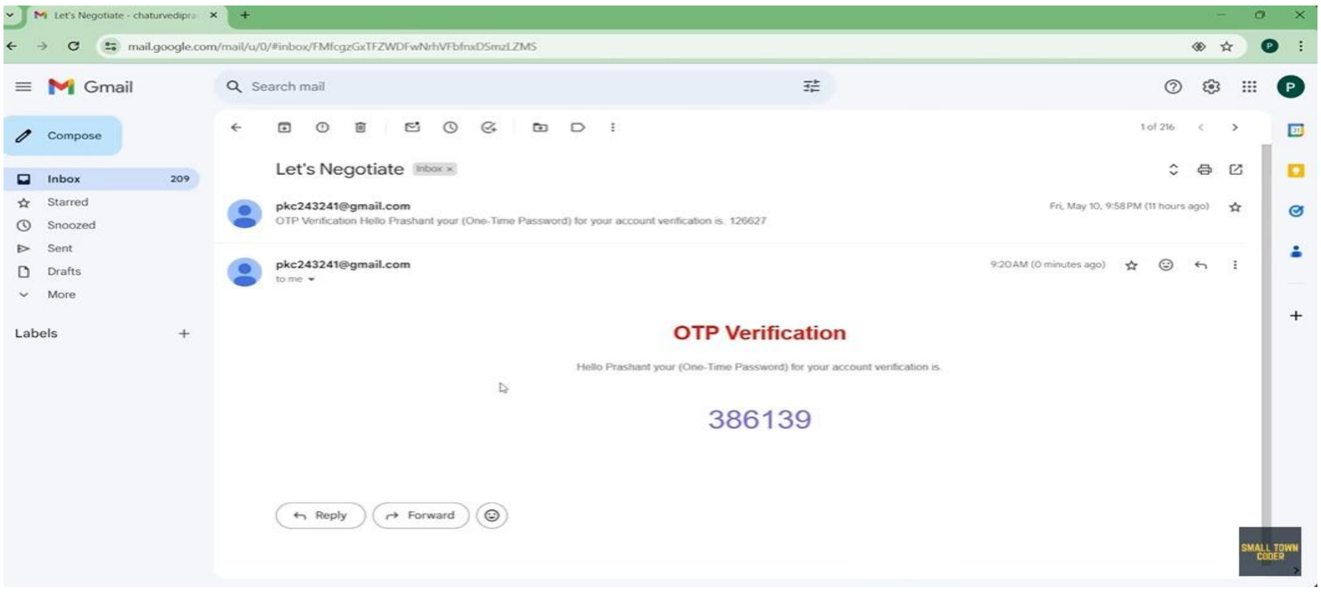


Email: prashantchoubey8338@gmail.com

Password: ••••••••

Don't have an account? Sign up

© 2023 Your E-Learning Platform. All rights reserved.
Made With ❤ Prashant Kumar Chaturvedi



mail.google.com/mail/u/0/#inbox/FMfcgzGxTFZWDFwNrhVFbfnxDSmzLZMS

Let's Negotiate Inbox

pkc243241@gmail.com
OTP Verification Hello Prashant your (One-Time Password) for your account verification is. 126627
Fri, May 10, 9:58 PM (11 hours ago)

pkc243241@gmail.com
to me
9:20 AM (0 minutes ago)

OTP Verification

Hello Prashant your (One-Time Password) for your account verification is.

386139

2. **Course Management**

Objective: Enables teachers and admins to create, update, and manage courses.

Input Fields:

Title: Name of the course.

Description: Brief overview of the course content.

Teacher: Dropdown to select the course instructor.

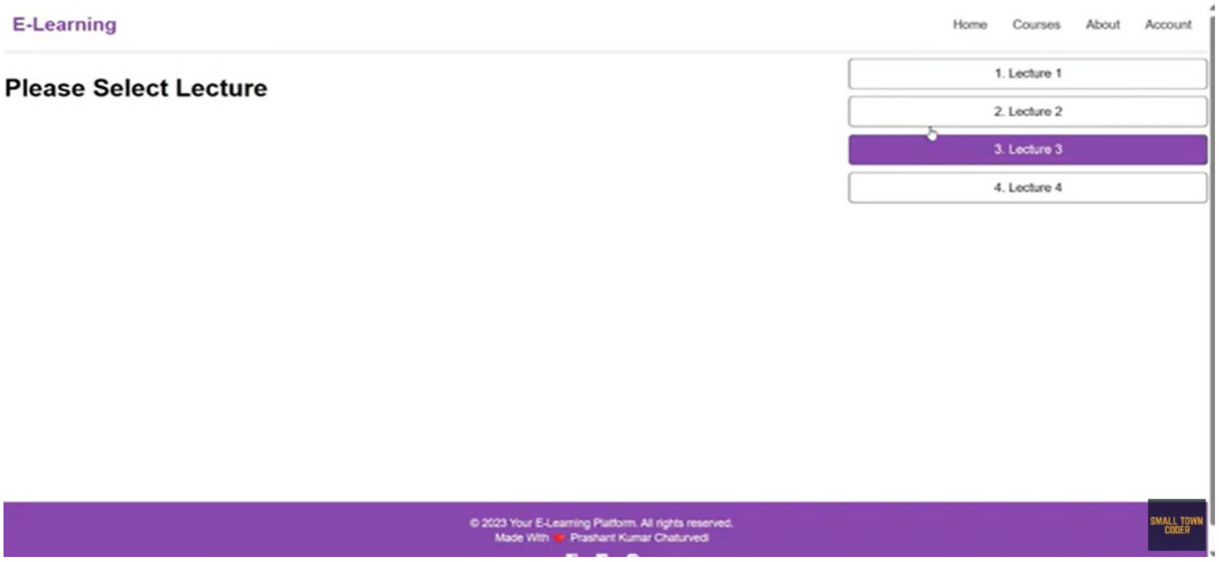
Duration: Duration of the course (in hours or weeks).

Buttons:

Create Course: Submits the form to create a new course.

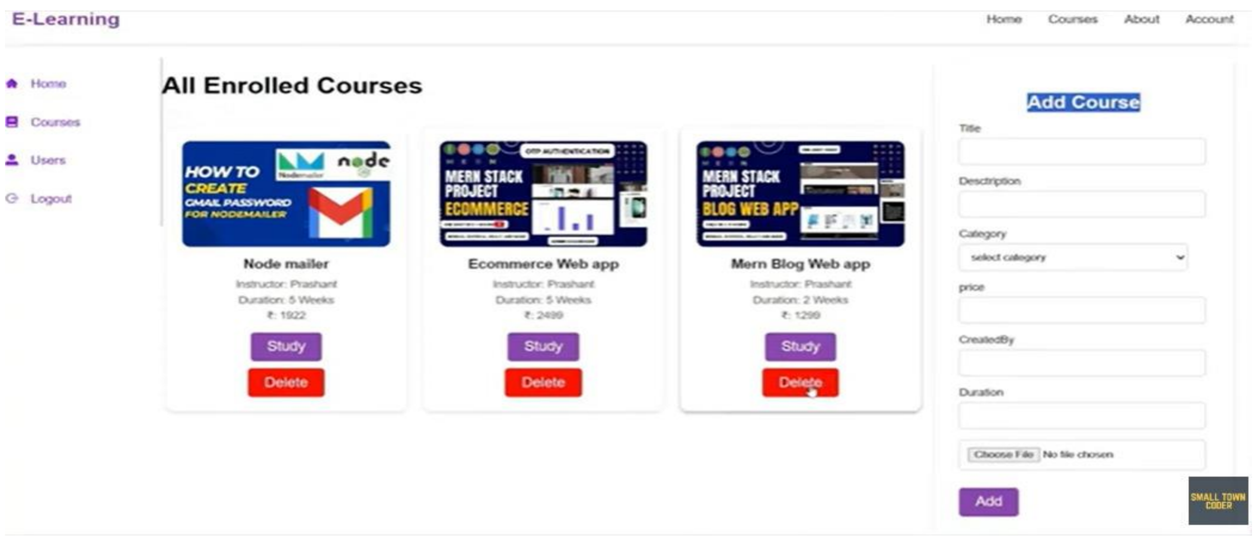
Edit Course: Allows updates to course details.

Delete Course: Removes the course from the platform.



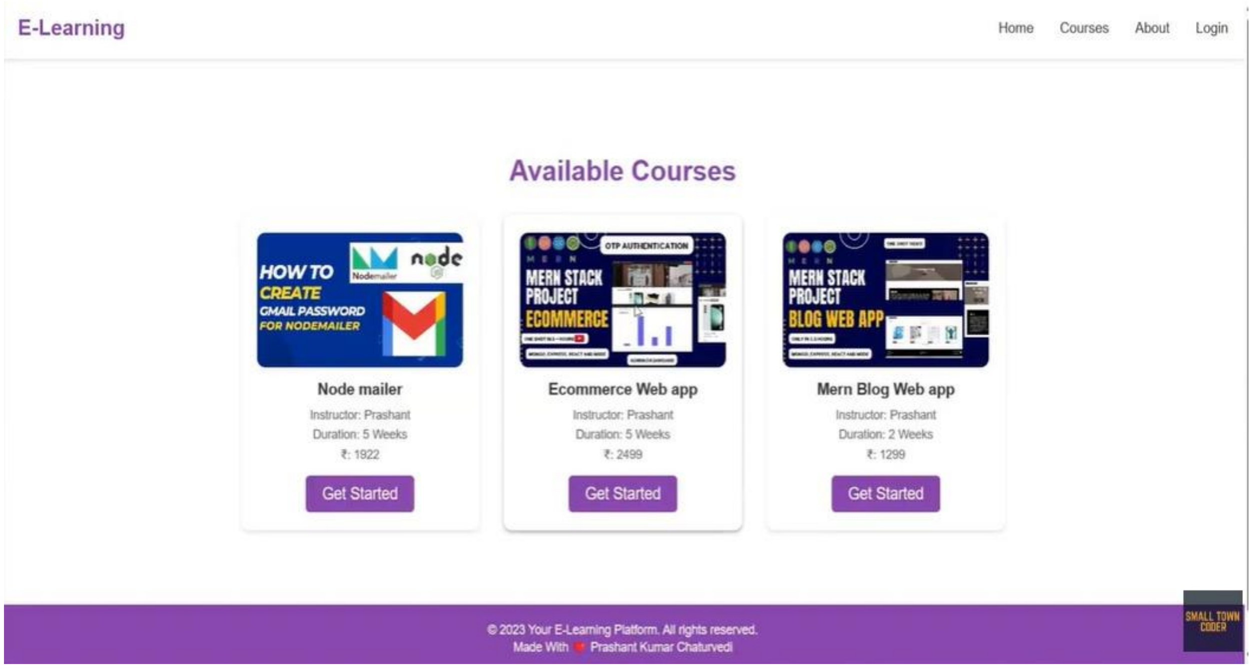
Please Select Lecture

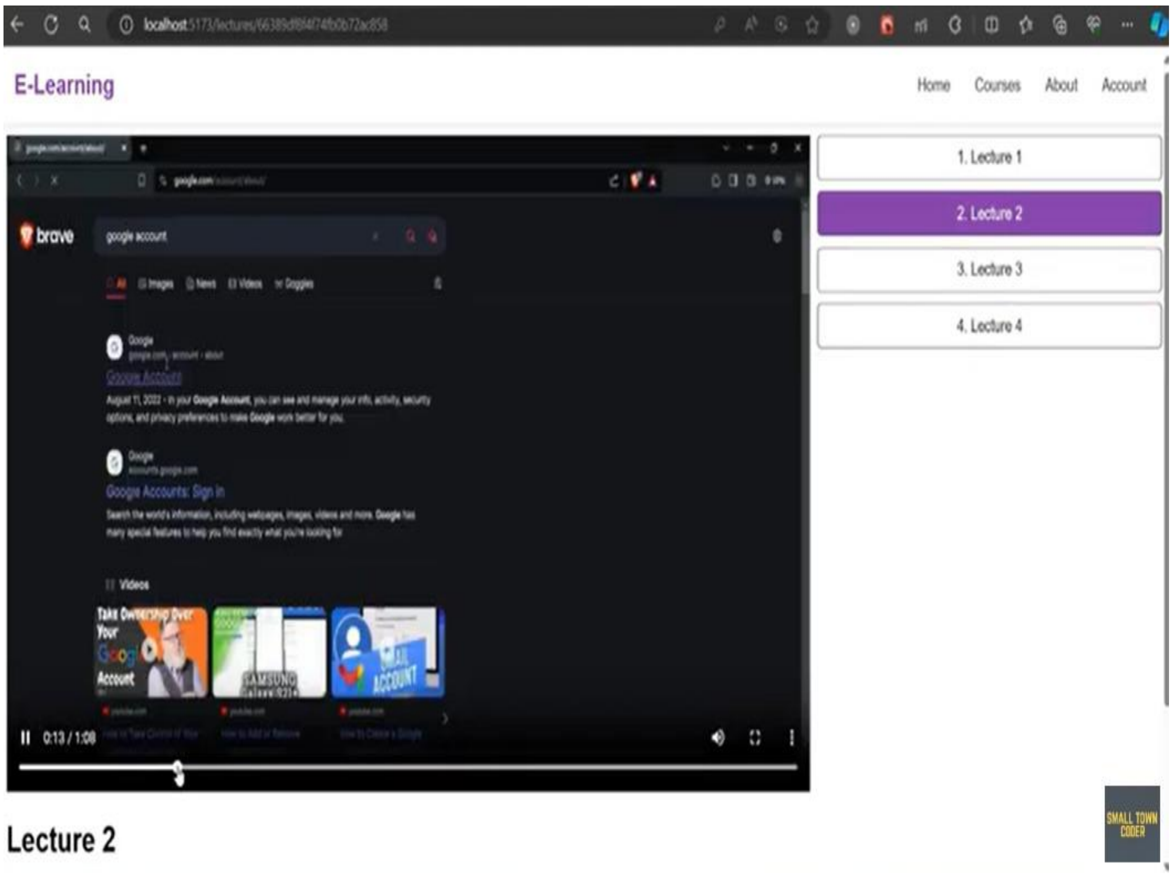
© 2023 Your E-Learning Platform. All rights reserved.
Made With ❤ Prashant Kumar Chaturvedi



## 3. Dashboard

- Objective: Displays available courses and supports video conferencing for lectures.
- Available Courses: Lists all courses the user is enrolled in or can browse, with details like title, teacher, and schedule.
- Upcoming Lectures: Shows upcoming lectures with the option to join via video conferencing.
- Video Conferencing: Integrates a platform like Zoom or Google Meet for live classes, accessible directly from the dashboard.
- Course Progress: Displays the user's progress in each course (completed, in progress, etc.).
- Notifications: Alerts for new lectures, assignments, or announcements.





## 4. Payment

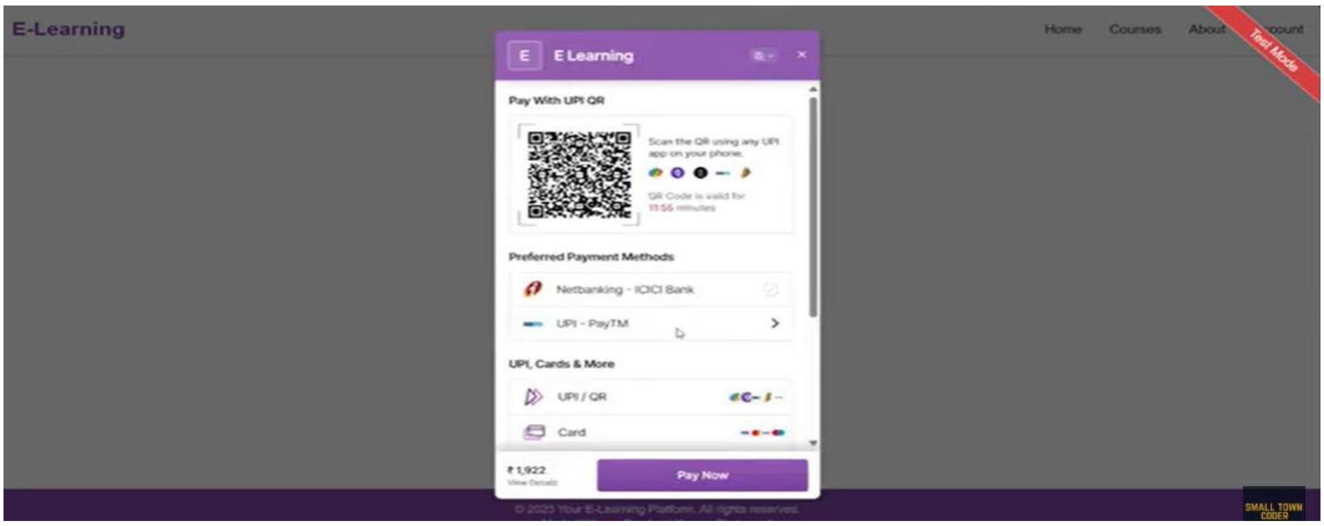
Objective: Manages course payments and subscriptions

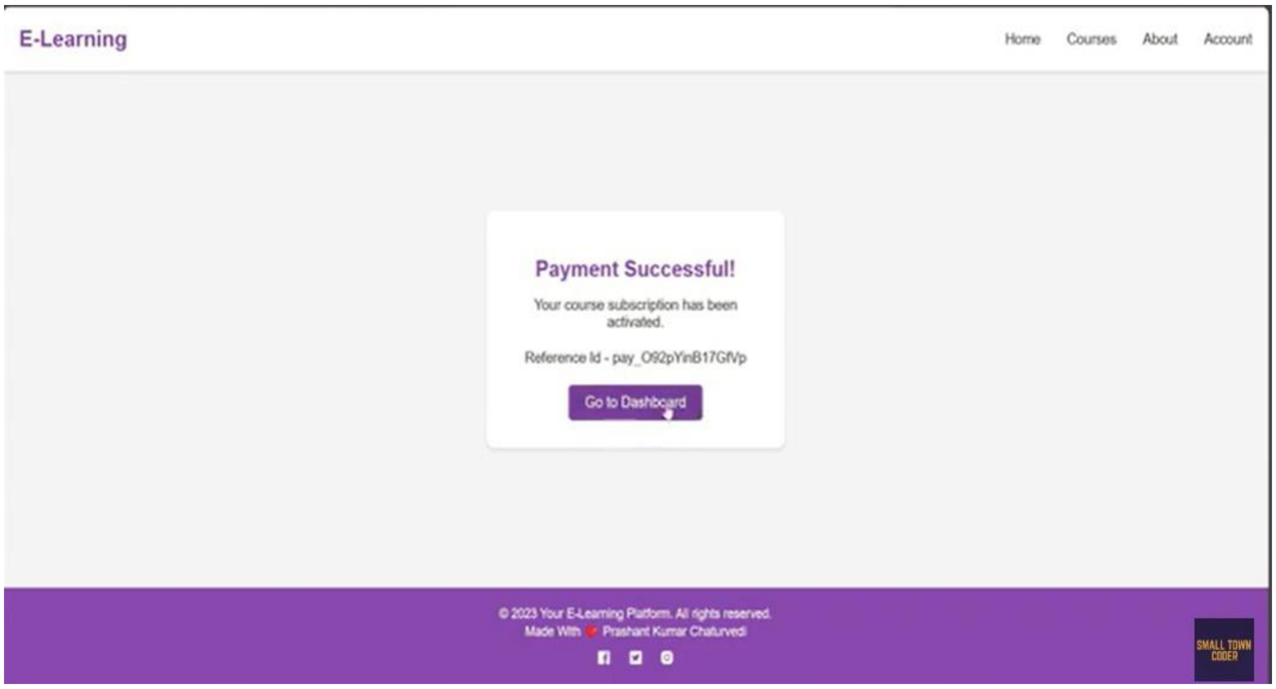
Payment Gateway Integration: Enables secure payment processing via credit card, PayPal, etc.

Invoice Generation: Automatically generates invoices after successful payment.

Payment History: Displays a log of previous transactions and receipts.

Subscription Plans: Allows students to subscribe to individual courses or a platform-wide membership.





Payment Successful!

Your course subscription has been activated.

Reference Id - pay_O92pYinB17GfVp

© 2023 Your E-Learning Platform. All rights reserved.
Made With ❤ Prashant Kumar Chaturvedi

**Challenge: User Authentication and Security**

• Problem: Ensuring secure user authentication while preventing unauthorized access, especially since sensitive health data is involved.

• Solution: Implement multi-factor authentication (MFA) for enhanced security, use hashed passwords for storage, and JWT tokens for secure session management. Regular security audits and data encryption (HTTPS) will ensure safety.

# CONCLUSION

The Online Learning Platform provides an integrated and efficient solution for students, teachers, and administrators to engage in the learning process seamlessly. By incorporating essential features such as user authentication, role-based access, course management, video conferencing for live lectures, and a comprehensive payment system, the platform ensures an interactive and user-friendly environment. The use of JWT for secure authentication, along with the ability to manage courses, track progress, and enable real-time communication between students and teachers, makes this platform an ideal tool for modern education.

Overall, this platform is designed to enhance the learning experience, provide flexible access to educational resources, and streamline administrative tasks, offering a robust and scalable solution for online education

# REFERENCES

1. **MERN Stack Documentation**
   - MongoDB: https://www.mongodb.com/docs/
   - Express.js: https://expressjs.com/
   - React: https://reactjs.org/docs/
   - Node.js: https://nodejs.org/en/docs/
2. **JWT Authentication**
   - JSON Web Tokens (JWT): https://jwt.io/
   - Understanding JWT: https://www.digitalocean.com/community/tutorials/understanding-jwt-encoding-decoding
3. **MongoDB Atlas**
   - MongoDB Atlas Setup: https://www.mongodb.com/cloud/atlas
4. **React (Frontend Framework)**
   - React Documentation: https://reactjs.org/docs/getting-started.html
5. **Node.js and Express Error Handling**
   - Error Handling in Express: https://expressjs.com/en/guide/error-handling.html