

# **ONLINE LEARNING PLATFORM**

## **Smart Internz (Naan Mudhalvan)**

Project report in partial fulfillment for the requirement of the degree of

### **BACHELOR OF TECHNOLOGY**

### **IN**

### **INFORMATION TECHNOLOGY**

Submitted by

ABILASH.R	211121205001
AKASH.S	211121205002
ASHWIN.R	211121205004
LOGESH.K	211121205017



**MADHA ENGINEERING COLEGE**

KUNDRATHUR, CHENNAI - 600 069

**NOV 2024**

## ABSTRACT

The increasing demand for flexible, accessible, and interactive education has led to the development of online learning platforms that empower both learners and instructors. This project, **Online Learning Platform**, aims to create a robust, user-friendly system for managing and delivering educational content in a virtual environment. The platform provides essential features such as user registration, secure login, course creation and enrollment, and real-time progress tracking.

The system was developed using a modern technology stack, including **React.js** for the frontend, **Node.js** and **Express.js** for the backend, and **MongoDB** for database management. Key functionalities like authentication are implemented using **JSON Web Tokens (JWT)** to ensure secure access, and the platform's responsive design enables accessibility across various devices. The use of the **MVC architecture** ensures a clear separation of concerns, contributing to the system's scalability and maintainability.

Throughout the development process, emphasis was placed on creating an interactive and engaging user experience. Students can explore courses, enroll, and monitor their progress, while instructors can efficiently upload and manage course materials. Additionally, the platform supports a flexible, self-paced learning approach, making it an ideal solution for learners with varying schedules and preferences.

By leveraging technology, this project demonstrates the potential of online learning platforms to make education more accessible, interactive, and impactful in today's digital age.

## TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	<b>ACKNOWLEDGEMENT</b>	iii
	<b>ABSTRACT</b>	iv
	<b>LIST OF FIGURES</b>	viii
1.	<b>Introduction</b>	1
	1.1 Problem Statement	1
	1.2 Project Objectives	1
	1.3 Project Scopes	1
	1.4 Goals of the Project	3
	1.5 Constraints of the Project	4
2.	<b>Project Plan</b>	5
	2.1 Project Initiation	5
	2.2 Requirement Gathering & Analysis	5
	2.3 System Design & Architecture	6
	2.4 Development Phase	6
	2.5 Testing Phase	7
	2.6 Deployment & Launch	7
	2.7 Post-Launch Support & Documentation	7
3.	<b>Requirement Analysis</b>	8
	3.1 Functional Requirements	8
	3.2 Scenario-Based Case Study	9
	3.3 System Requirements	10

	3.4 Technical Architecture	10
4.	<b>Er Diagram</b>	11
5.	<b>Pre-Requisites</b>	12
	5.1 Vite	12
	5.2 Node.js & npm	12
	5.3 Express.Js	12
	5.4 MongoDB	13
	5.5 React.Js	13
	5.6 HTML, CSS, and JavaScript	13
	5.7 Database Connectivity	13
	5.8 Front-end Framework	14
6.	<b>Project Structure</b>	15
7.	<b>Application Flow</b>	17
8.	<b>Project Flow</b>	18
	8.1 Project Setup and Configuration	18
9.	<b>Literature Review</b>	20
	9.1 Existing Online Learning Platforms	20
	9.2 Gaps in Existing Systems	20
	9.3 Technologies Used in Learning Platforms	21
	9.4 Relevance to Naan Mudhalvan Initiative	21
	9.5 Key Insights for Project Development	21
10.	<b>System Design</b>	22

	10.1 High-Level Design (HLD)	22
	10.2 Low-Level Design (LLD)	24
	10.2.1 Database Design	24
	10.2.2 APIs	24
	10.2.3 UI Design	25
	10.2.4 Sequence Diagram	25
	10.3 Security Considerations	25
11.	<b>Implementation</b>	26
	11.1 Overview of the Implementation Process	26
	11.2 Technologies Used	26
	11.3 System Architecture	36
	11.4 Database Design	36
	11.5 Key Features Implementation	37
	11.6 Code Snippets	37
	11.7 Testing	39
	11.8 Challenges Faced During Implementation	39
	11.9 Conclusion	39
12.	<b>Appendices</b>	40
13.	<b>Conclusion</b>	57
14.	<b>References</b>	58

## **LIST OF FIGURES**

<b>S.NO</b>	<b>FIGURE</b>	<b>PAGE NO</b>
1	Technical architecture	10
2	ER-Diagram	11
3	Frontend structure	15
4	Backend structure	16
5	Configuration files	16
6	6.1 Data Flow Part 1 (Level 0)	23
	6.2 Data Flow Part 2 (Level 1)	23
7	Sequence Diagram	25

# 1.INTRODUCTION

## 1.1 Problem Statement

Traditional learning environments and many online platforms often lack streamlined functionality for basic course management and user administration. Students require a simple, user-friendly interface to explore and purchase courses, while teachers need tools to manage their course offerings. Additionally, administrators must have robust controls to oversee the platform's users and content. The absence of such a basic yet efficient system creates barriers for effective educational collaboration. This project aims to address these challenges by building a straightforward online learning platform that caters to the essential needs of students, teachers, and administrators, using the MERN stack for efficiency and scalability.

## 1.2 Project Objectives

The objectives of this **Simple Online Learning Platform** are:

- To enable students to log in, browse courses, and make purchases easily.
- To provide teachers with tools to add or remove courses effectively.
- To allow administrators to oversee the system by managing users and courses.
- To ensure secure and role-based access for students, teachers, and administrators.
- To deliver a simple, responsive, and scalable platform using the MERN stack.

## 1.3 Project Scope

The **Simple Online Learning Platform** focuses on addressing the basic needs of students, teachers, and administrators in an educational ecosystem. It is designed to be straightforward yet effective, leveraging the MERN stack for scalability, flexibility, and ease of maintenance. The scope includes the following features and functionalities:

- **Student Module:**

- Students can register and log in securely.
- They can browse available courses with detailed descriptions.
- The platform allows students to purchase courses and access them after successful payment.

- **Teacher Module:**

Teachers can register and log in with role-based authentication.

They can add new courses, providing titles, descriptions, and other relevant details.

Teachers can also delete courses they have created.

- **Admin Module:**

Administrators have full control over the platform.

They can add, update, or remove any user (students or teachers).

Admins can manage courses by adding, editing, or removing them.

- **General Features:**

A responsive user interface ensures accessibility across devices (desktop, tablet, and mobile).

Secure authentication using JSON Web Tokens (JWT) for all user roles.

A basic database structure to handle user and course data efficiently.

- **Future Expansion:**

While the initial scope is simple, the platform's architecture allows for adding advanced features like course reviews, progress tracking, live classes, or analytics in the future.



This project focuses on providing a simple, functional solution tailored to small educational institutions or start-ups seeking an easy-to-manage learning platform.

#### **1.4 Goals of the Project**

The primary goals of the **Simple Online Learning Platform** project are:

##### **User-Centric Design:**

To create an intuitive, easy-to-navigate platform that provides a smooth experience for students, teachers, and administrators, ensuring minimal learning curve for all users.

##### **Secure and Role-Based Authentication:**

To implement secure login and authentication processes for students, teachers, and administrators, ensuring proper role-based access control and data security.

##### **Efficient Course Management:**

To enable teachers to easily add, update, and remove courses, while allowing students to browse, purchase, and access courses in a streamlined manner.

##### **Admin Control and Oversight:**

To empower administrators with comprehensive control over user management (students and teachers) and course management, ensuring smooth operations and the ability to maintain the integrity of the platform.

##### **Scalability and Flexibility:**

To build a scalable solution using the MERN stack that can be easily expanded in the future, allowing for the integration of additional features such as assessments, live classes, or analytics.

##### **Responsive and Accessible Interface:**

To develop a platform that is fully responsive, providing an optimal experience on various devices (desktop, tablet, mobile) and ensuring accessibility for a wide range of users.

**Cost-Effective and Simple Implementation:**

To deliver a cost-effective solution using modern open-source technologies that are easy to implement, maintain, and scale, reducing development and operational costs for educational institutions.

These goals aim to provide a robust, user-friendly, and secure online learning platform that caters to the essential needs of all users, while laying the foundation for future growth and feature expansion.

**1.5 Constraints of the Project****Limited Features in Initial Version:**

The platform is designed to be simple in its first version, with core functionalities such as user authentication, basic course management, and administrative controls. Advanced features like real-time interactions, live classes, and detailed progress tracking are outside the scope for the initial release.

**Scalability Limitations:**

While the system is designed to be scalable, the initial deployment may not handle extremely large volumes of users or courses without additional optimizations. The performance of the platform could be limited if there is a rapid increase in traffic or content.

**Platform Dependency:**

The platform is currently dependent on the MERN stack (MongoDB, Express, React, Node.js), which, while flexible and scalable, may require additional resources and expertise for deployment, especially for teams not already familiar with these technologies.

**Security Concerns:**

Though security measures like role-based authentication and secure user login are implemented, there may still be vulnerabilities, especially if the platform scales rapidly or integrates additional complex features. Ongoing security audits will be needed as the platform grows.

## 2. PROJECT PLAN

This project was developed by **Abilash, Akash, Ashwin** and **Logesh** over the last one and a half months. The core requirements were inspired by the **Naan Mudhalvan portal**. The platform provides functionalities where students can view and buy courses, teachers can manage courses, and admins can oversee users and course management.

### 2.1 Project Initiation

**Completed:** First Week

**Project Scope:** The project scope was defined based on the Naan Mudhalvan portal, ensuring the platform included essential features such as role-based access, course viewing, buying, adding, and removing.

**Team Formation:** The team, consisting of **Logesh, Abilash, Akash,** and **Ashwin**, was formed to handle various parts of the project: backend (Node.js + Express), frontend (React), and testing.

**Technology Selection:** MERN stack (MongoDB, Express, React, Node.js) was chosen for its full-stack capabilities and ability to handle the project's requirements efficiently.

### 2.2 Requirement Gathering & Analysis

**Completed:** Week 1-2

**Requirements Clarification:** The project's functional requirements were clearly defined by analyzing the Naan Mudhalvan portal, ensuring alignment with the core features of the platform.

**Wireframes and Design:** Wireframes were quickly drafted in Figma, and design mockups were shared with peers and professors to ensure all user flows were correctly mapped out.

## 2.3 System Design & Architecture

**Completed:** Week 2-3

**Database Schema Design:** MongoDB collections for users (students, teachers, admins) and courses were created, including relationships between them.

**System Architecture:** A simple yet effective architecture was designed with the frontend in React and the backend in Node.js with Express, all interacting with MongoDB.

**User Flow Mapping:** The flow of the platform from student registration to course buying and teacher course management was finalized.

## 2.4 Development Phase

**Completed:** Week 3-7

**Backend Development:**

**User Authentication:** JWT-based login system was implemented for students, teachers, and admins.

**Course Management:** Developed API routes for adding, updating, and deleting courses. Teachers had access to manage their own courses, while admins could manage everything.

**CRUD Operations:** Set up necessary routes for managing users and courses with appropriate security checks.

**Frontend Development:**

Created user interfaces based on the wireframes, including student dashboards, course lists, and teacher/admin dashboards.

React Router was used for navigation, and React state management handled the user sessions and course details.

Bootstrap was used for responsive and mobile-friendly UI design.

- **Role-Based Access Control:** Role-based conditional rendering was implemented to display appropriate features (like course management for teachers and admins).

## 2.5 Testing Phase

**Completed:** Week 7-8

**User Testing:** Some users (students) tested the platform, providing feedback that helped identify bugs and improve the UI/UX.

## 2.6 Deployment and Launch

**Completed:** Week 8-9

**Live Environment Testing:** Final testing was conducted on the live platform to ensure smooth operation.

**Project Presentation:** A detailed presentation was given to professors, explaining the development process, features, and how the Naan Mudhalvan requirements were met.

## 2.7 Post-Launch Support & Documentation

**Completed:** Ongoing (until project submission)

**Documentation:** The final report was written, explaining the design choices, development workflow, and how the platform meets the requirements.

**Code Documentation:** Clear comments were added to the codebase to aid future developers.

**Final Tweaks:** Any bugs were fixed, and UI enhancements were made based on feedback from the testing phase.

**Project Submission:** The project, including the live URL, source code, and report, was submitted for academic evaluation.

### 3. REQUIREMENT ANALYSIS

#### 3.1 Functional Requirements

An online learning platform(OLP) is a digital platform that provides a variety of tools and resources to facilitate learning and education over the internet. These platforms have become increasingly popular, especially in recent years, as they offer flexibility and accessibility for learners of all ages and backgrounds. Here are some key features and a description of an online learning platform:

**3.1.1 User-Friendly Interface:** Online learning platforms typically have an intuitive and user-friendly interface that makes it easy for learners, regardless of their technical proficiency, to navigate and access the content.

**3.1.2 Course Management:** Instructors or course creators can upload, organize, and manage course materials. Learners can enroll in courses and track their progress.

**3.1.3 Interactivity:** Many platforms include interactive elements like discussion forums, chat rooms, and live webinars, which foster communication and collaboration among learners and instructors.

**3.1.4 Certification:** Learners can earn certificates or badges upon completing courses or meeting certain criteria, which can be valuable for employment or further education.

**3.1.5 Accessibility:** Content is often accessible on various devices, including computers, tablets, and smartphones, making learning possible from anywhere with an internet connection.

**3.1.6 Self-Paced Learning:** Learners can typically access course materials at their own pace. This flexibility allows for learning that fits into individual schedules and preferences.

**3.1.7 Payment and Subscription Options:** There may be free courses, but some content may require payment or a subscription. Platforms often offer multiple pricing models.

#### 3.2 Scenario-based Case Study:

## Scenario: Learning a New Skill

**3.2.1 User Registration:** Sarah, a student interested in learning web development, visits the Online Learning Platform and creates an account. She provides her email and chooses a password.

**3.2.2 Browsing Courses:** Upon logging in, Sarah is greeted with a user-friendly interface displaying various courses categorized by topic, difficulty level, and popularity.

She navigates through the course catalog, filtering courses by name and category until she finds a "Web Development Fundamentals" course that interests her.

**3.2.3 Enrolling in a Course:** Sarah clicks on the course and reads the course description, instructor details, and syllabus. Impressed, she decides to enroll in the course.

After enrolling, Sarah can access the course materials, including video lectures, reading materials, and assignments.

**3.2.4 Learning Progress:** Sarah starts the course and proceeds through the modules at her own pace. The platform remembers her progress, allowing her to pick up where she left off if she needs to take a break.

**3.2.5 Interaction and Support:** Throughout the course, Sarah engages with interactive elements such as discussion forums and live webinars where she can ask questions and interact with the instructor and other learners.

**3.2.6 Course Completion and Certification:** After completing all the modules and assignments, Sarah takes the final exam. Upon passing, she receives a digital certificate of completion, which she can download and add to her portfolio.

**3.2.7 Paid Courses:** Sarah discovers an advanced web development course that requires payment. She purchases the course using the platform's payment system and gains access to premium content.

**3.2.8 Teacher's Role:** Meanwhile, John, an experienced web developer, serves as a teacher on the platform. He creates and uploads new courses on advanced web development topics, adds sections to existing courses, and monitors course enrolments.

**3.2.9 Admin Oversight:** The admin oversees the entire platform, monitoring user activity, managing course listings, and ensuring smooth operation. They keep track of enrolled students, handle any issues that arise, and maintain the integrity of the platform.

### 3.3 System Requirements

#### 3.3.1 Hardware Required:

Windows 8 machine

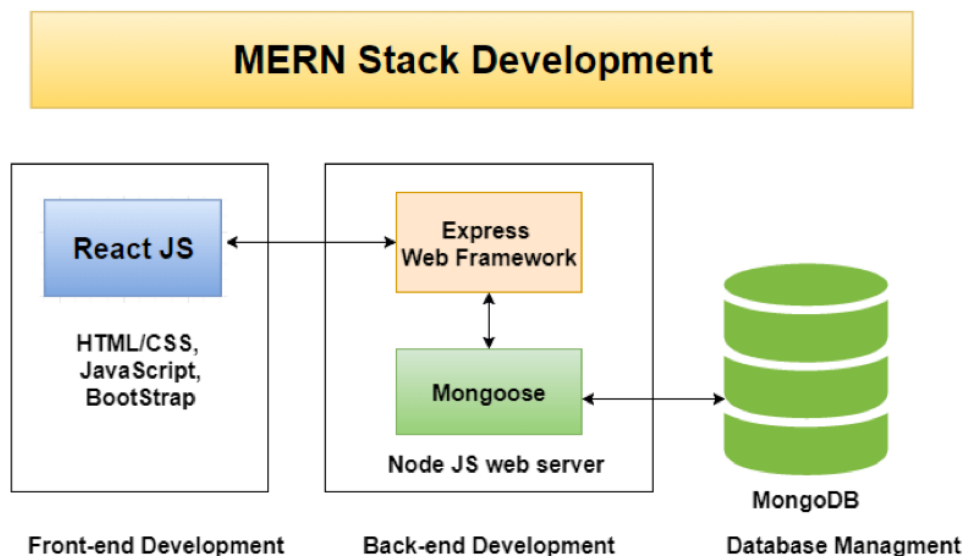
#### 3.3.2 Software Required:

Install with two web browsers

#### 3.3.3 System Required:

Bandwidth of 30mbps

### 3.4 Technical Architecture

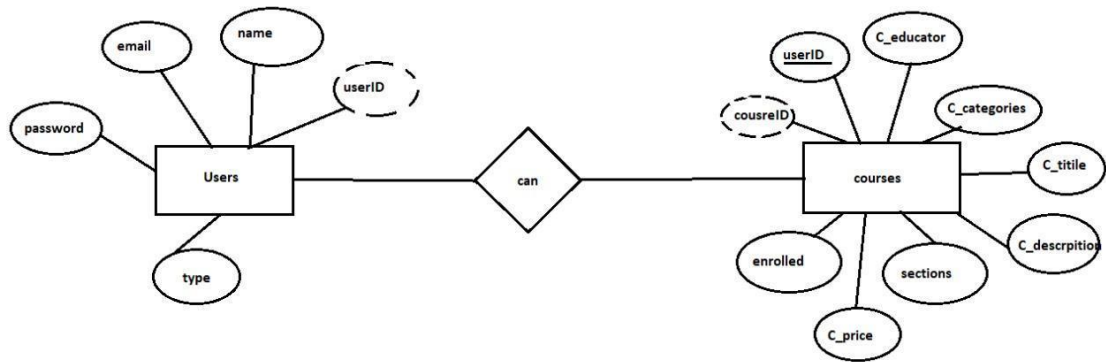


**Fig 3.1** Technical architecture

The technical architecture of OLP app follows a client-server model, where the frontend serves as the client and the backend acts as the server. The frontend encompasses not only the user interface and presentation but also incorporates the axios library to connect with backend easily by using RESTful Apis.



## 4. ER-DIAGRAM



**Fig 4.1** ER-Diagram

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

Users:

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

Courses:

1. `userID`: (can act as a foreign key )
2. `_id`: (MongoDB creates by unique default)
3. `C_educator`
4. `C_categories`
5. `C_title`
6. `C_description`
7. `sections`
8. `C_price`
9. `Enrolled`

## 5. PRE-REQUISITES

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, React.js:

### 5.1 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.

**npm create vite@latest**

### 5.2 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/>

**npm init**

### 5.3 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**

## 5.4 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/>

## 5.5 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>

**5.6 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.

**5.7 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. To Connect the Database with Node JS, go through the below provided link:

<https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/>

**5.8 Front-end Framework:** Utilize Reactjs 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 Dependencies:

- Navigate into the cloned repository directory:

```
cd containment-zone
```

- 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>

You have 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

## 6.PROJECT STRUCTURE

A **project structure** outlines the organization and arrangement of components, files, and resources within a project. It serves as a blueprint for how a project is designed, developed, and managed. A well-defined project structure improves collaboration, simplifies development and debugging, and ensures consistency across the project lifecycle.



Fig 6.1 Frontend structure

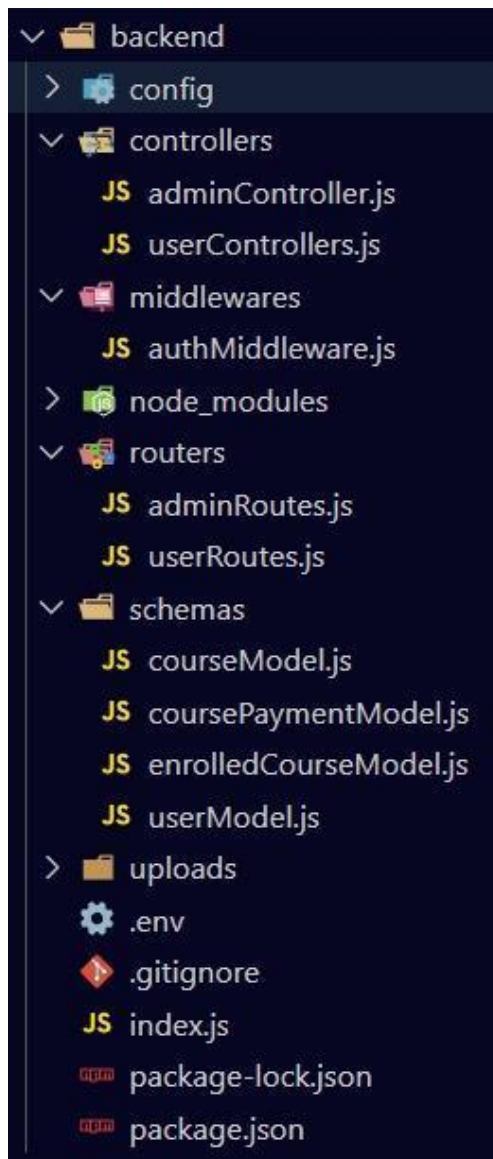


Fig 6.2 Backend structure

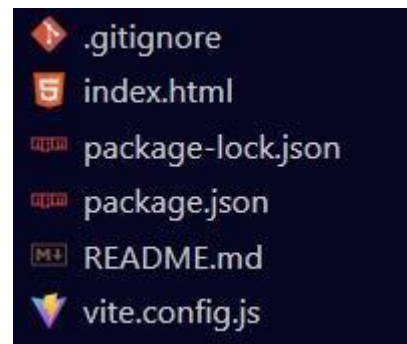


Fig 6.3 Configuration files

The image 6.1 is of frontend part which is showing all the files and folders that have been used in UI development

The image 6.2 is of Backend part which is showing all the files and folders that have been used in backend development

The image 6.3 is of configuration part. These files collectively help in defining, configuring, and documenting a project's structure, dependencies, and workflows.

## **7. 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:

### **7.1 Teacher:**

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

### **7.2 Student:**

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

### **7.3 Admin:**

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

## **8. PROJECT FLOW**

### **8.1 Project Setup and Configuration**

#### **8.1.1 Frontend Development**

- React
- Bootstrap
- Material UI
- Axios
- mdb-react-ui-kit
- react-bootstrap

#### **8.1.2 Backend Development**

- cors
- bcrypt.js
- expresss
- dotenv
- mongoose
- multer
- nodemon
- jwttoken

#### **8.1.3 Database Development**

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



```
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"],

    },},{timestamps: true,});

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

module.exports = userSchema;
```

## 9. LITERATURE REVIEW

The **literature review** analyses existing systems, platforms, or projects related to the one you have developed. It identifies gaps, challenges, and opportunities that your project addresses. For a project based on the **Naan Mudhalvan portal requirements**, here's a tailored literature review:

### 9.1. Existing Online Learning Platforms

Many online learning platforms cater to diverse user groups, offering a range of features. Some notable examples include:

- **Coursera:**
  - Offers a wide range of courses from top universities.
  - Provides certificates but focuses heavily on pre-designed curricula.
  - Lack of customization for regional or institutional needs.
- **Udemy:**
  - Allows instructors to create and publish courses.
  - Limited student engagement tools compared to modern LMS (Learning Management Systems).
- **Khan Academy:**
  - Provides free education primarily targeting school-level learners.
  - Does not emphasize skill development for advanced learners or employability.

### 9.2. Gaps in Existing Systems

The above platforms, while comprehensive, lack certain features relevant to your project:

- **Regional Customization:**
  - Existing platforms often do not cater to the specific needs of regional learners or provide content in local languages.
- **Government Collaboration:**
  - Limited or no integration with government initiatives or skill development programs like **Naan Mudhalvan**.

- **Cost and Accessibility:**
  - Many platforms require subscriptions or fees, limiting access for underprivileged students.
- **Institution-Specific Features:**
  - They lack the ability to integrate features like administrative tools, attendance management, or fee tracking, which your project incorporates.

### 9.3. Technologies Used in Learning Platforms

- **Frontend Technologies:** React.js, Angular, and Vue.js are popular for creating interactive and user-friendly interfaces.
- **Backend Technologies:** Node.js, Django, and Spring Boot provide robust server-side solutions.
- **Database Systems:** MongoDB, MySQL, and PostgreSQL are commonly used for managing course data, user profiles, and progress tracking.

### 9.4. Relevance to Naan Mudhalvan Initiative

The **Naan Mudhalvan** portal focuses on enhancing skills among students and bridging the gap between academic learning and industry requirements. Your project aligns with this vision by:

- Incorporating course content tailored to the local demographic and skill requirements.
- Integrating tools to track learning progress, offer certifications, and ensure accessibility.

### 9.5. Key Insights for Project Development

- **Self-Paced Learning:** Flexible access to materials is crucial for catering to diverse learners.
- **Accessibility:** Ensuring the platform works across devices (desktop, mobile, tablet) broadens its reach.

## 10. SYSTEM DESIGN

The system design phase focuses on creating a blueprint that outlines how your project will be structured, organized, and implemented. For your project based on the **Naan Mudhalvan portal requirements**, the design can be broken into two primary components: **High-Level Design (HLD)** and **Low-Level Design (LLD)**.

### 10.1. High-Level Design (HLD)

#### 10.1.1 System Architecture

The system can be designed as a **three-tier architecture**:

##### 1. **Presentation Layer (Frontend):**

- Responsible for the user interface and user experience.
- Technologies: React.js or any frontend framework for responsive design.

##### 2. **Business Logic Layer (Backend):**

- Handles the server-side logic, APIs, and communication with the database.
- Technologies: Node.js with Express.js or Django.

##### 3. **Data Layer (Database):**

- Stores user information, course data, progress, and other relevant records.
- Database: MySQL or MongoDB for relational/non-relational data.

### 10.1.2 System Modules

The project is divided into modules:

1. **User Management:**

- Role-based access for students, instructors, and admins.
- Features: Registration, login, profile updates.

2. **Course Management:**

- Upload, organize, and access course materials.
- Categories for skill levels or topics.

3. **Progress Tracking:**

- Monitor student performance and generate progress reports.

4. **Certification System:**

- Award certificates upon course completion.

5. **Admin Panel:**

- Manage users, monitor system performance, and generate analytics.

### 10.1.3 Data Flow Diagram (DFD)

- **Level 0:** Shows the interaction between users and the system.

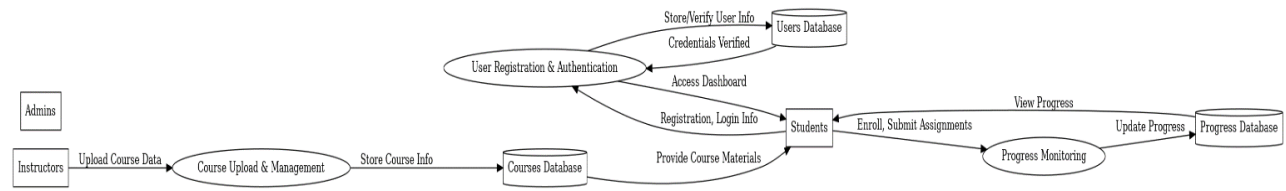


**Fig 10.1** Interaction between **Students** and the **System**.



**Fig 10.2** Interaction between **Instructors, Admins, and the System**.

- **Level 1:** Breaks down specific actions like user registration, course upload, and progress monitoring.



**Fig 10.3** Level 1 Data Flow Diagram(DFD).

## 10.2. Low-Level Design (LLD)

### 10.2.1 Database Design

- **Tables:**
  - Users: Store user details (ID, name, role, email).
  - Courses: Store course information (ID, title, description, instructor).
- **Relationships:**
  - One-to-Many: Instructors to Courses.
  - Many-to-Many: Students to Courses.

### 10.2.2 APIs

Design RESTful APIs for interaction between the frontend and backend:

1. **User APIs:**
  - Login: POST /api/login
  - Register: POST /api/register
  - Profile Update: PUT /api/users/:id
2. **Course APIs:**
  - Get Courses: GET /api/courses
  - Add Course: POST /api/courses
  - Enroll in Course: POST /api/enroll/:courseId
3. **Progress APIs:**
  - Track Progress: GET /api/progress/:userId

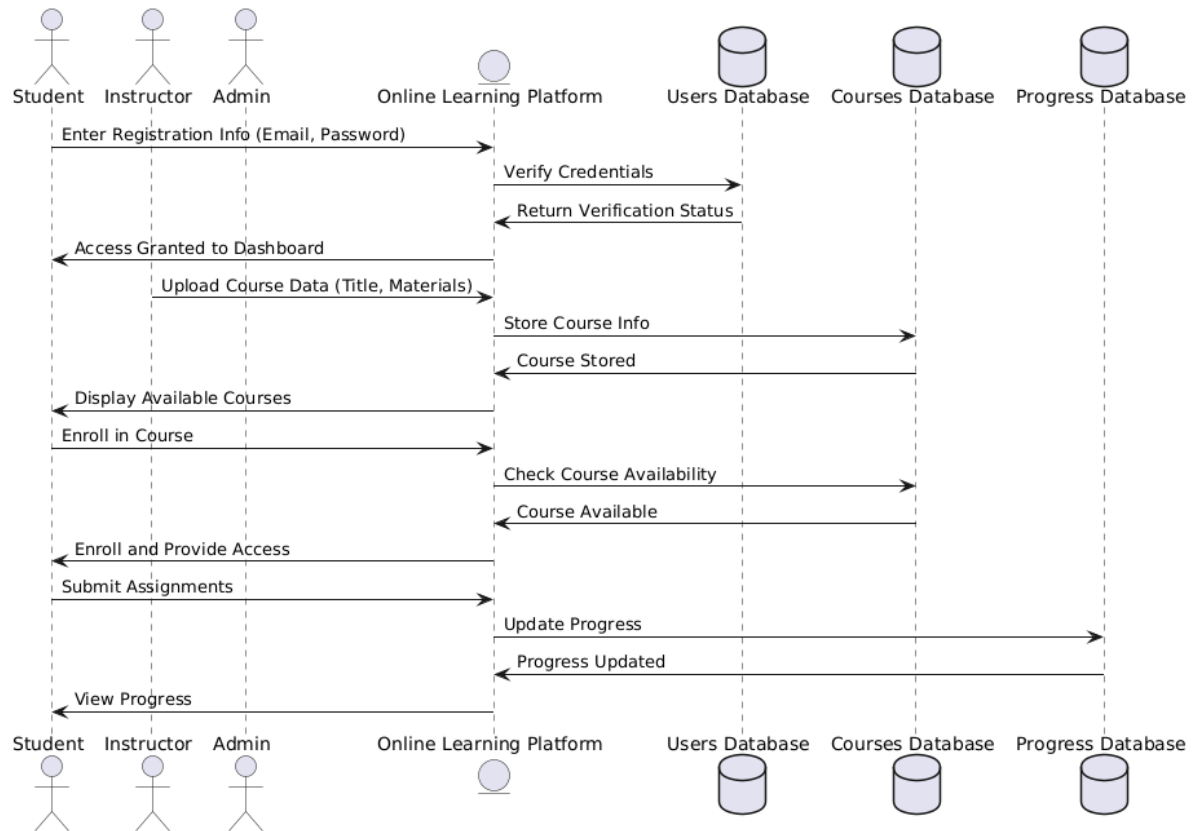
### 10.2.3 UI Design

- **Wireframes:**

- Wireframes for major screens such as the homepage, course page, progress tracking dashboard, and admin panel.

### 10.2.4 Sequence Diagram

- Illustrates the flow of operations, such as a student enrolling in a course, accessing materials, and receiving a certificate.



**Fig 10.4** Sequence Diagram

## 10.3. Security Considerations

- **Authentication:** Implement JWT (JSON Web Tokens) for secure user sessions.
- **Authorization:** Role-based access control to ensure only authorized users can perform certain actions.
- **Data Security:** Encrypt sensitive data like passwords using hashing algorithms.

## 11. IMPLEMENTATION

The **Implementation** chapter of your project report details how the system is developed, including the tools, technologies, and methodologies used to bring the design into action. Below is a breakdown of the sections you can include in the **Implementation** chapter for your **Online Learning Platform** project.

### 11.1. Overview of the Implementation Process

The **implementation** phase transforms the system's design into a working model, and the process involves setting up the environment, writing code, integrating features, and testing the system. This section provides a brief overview of the steps followed to implement the online learning platform.

### 11.2. Technologies Used

#### 11.2.1 Frontend Development

The frontend is responsible for the user interface and user experience. For your project, the following technologies were used:

- **HTML/CSS/JavaScript:** These are the core technologies for developing the structure, styling, and interactivity of the platform.

 **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>
```

```
@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);

}
```

- **React.js:** A powerful JavaScript library used for building dynamic and responsive user interfaces. React's component-based architecture allows for efficient rendering of the user interface.

 App.js

```
function App() {  
  
  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) {

      setUserData(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 /> } />

          </Routes>

        </div>

      </Router>

    </div>

    </UserContext.Provider>

  );
}

```

```

<Route path="/register" element={<Register />} />

{userLoggedIn ? (
  <>
    <Route path="/dashboard" element={<Dashboard />} />
    <Route path="/courseSection/:courseId/:courseTitle" element={<CourseContent />} />
  </>
) : (
  <Route path="/login" element={<Login />} />
)}
</Routes>
</div>
</Router>
</div>
</UserContext.Provider>
);
}

export default App;

```

- **Bootstrap:** A popular framework used to style the user interface, ensuring that the platform is responsive and mobile-friendly.

### **Bootstrap form**

```

import Button from 'react-bootstrap/Button';

import Form from 'react-bootstrap/Form';

```

```

function BasicExample() {

return (

<Form>

<Form.Group className="mb-3" controlId="formBasicEmail">

<Form.Label>Email address</Form.Label>

<Form.Control type="email" placeholder="Enter email" />

<Form.Text className="text-muted">

We'll never share your email with anyone else.

</Form.Text>

</Form.Group>

<Form.Group className="mb-3" controlId="formBasicPassword">

<Form.Label>Password</Form.Label>

<Form.Control type="password" placeholder="Password" />

</Form.Group>

<Form.Group className="mb-3" controlId="formBasicCheckbox">

<Form.Check type="checkbox" label="Check me out" />

</Form.Group>

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

Submit

</Button>

</Form>

);

}

```

```
export default BasicExample;
```

### 11.2.2 Backend Development

The backend is responsible for data processing, server-side logic, and handling requests between the frontend and databases. The following technologies were used for backend development:

- **Node.js**: A runtime environment for executing JavaScript on the server. Node.js allows for handling multiple simultaneous requests efficiently, which is crucial for an online platform with potentially many users.
- **Express.js**: A Node.js web application framework used to build the REST API for the system, which facilitates communication between the client-side and the server.

 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}`))

```

- **MongoDB:** A NoSQL database used to store user, course, and progress information. MongoDB is highly scalable and flexible, allowing for rapid development and easy integration with the application.

🚦 Config.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;
```

### 11.2.3 Authentication

The platform requires user authentication to secure access to the courses and progress data.

- **JWT (JSON Web Tokens)**: Used for user authentication and authorization. JWT ensures that users are properly authenticated when accessing their account or course materials.

 Controller.js

```
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}` });

}

};

```

#### 11.2.4 Deployment

The deployment of the application ensures that the system is accessible to users on the web.

- **Vercel:** Used for hosting the frontend, ensuring that the platform is easily accessible by students and instructors.

- **Heroku:** Used for deploying the backend API and database, ensuring scalability and security

### 11.3. System Architecture

The system architecture outlines how different components interact within the platform. The **Model-View-Controller (MVC)** architecture is used for clear separation of concerns:

- **Model:** Represents the data and the business logic. In the case of the online learning platform, the data includes users, courses, and progress.
- **View:** Represents the user interface, including the course listings, registration page, and student progress pages.
- **Controller:** Handles user inputs, processes them, and returns the appropriate responses. For example, handling user registration or retrieving course data.

### 11.4. Database Design

#### 11.4.1 Database Schema

The platform uses **MongoDB** as the database to store and manage data. The database schema includes the following collections:

- **Users:** Stores user information such as name, email, password (hashed), role (student/instructor), and other details.
- **Courses:** Stores course details including course name, description, instructor ID, and course materials.
- **Progress:** Tracks student progress for each course, including assignments, grades, and completion status.

## 11.5. Key Features Implementation

### 11.5.1 User Registration and Login

The user registration and login system is implemented with authentication. The steps include:

- **Sign-Up:** The user submits their registration details, which are validated and stored in the database.
- **Login:** The system checks the entered credentials against the stored data and issues a JWT token for authentication.

### 11.5.2 Course Upload and Management

Instructors can upload course details, including the title, description, and materials. The course information is saved in the database, and the system provides options for students to access and enroll in the course.

### 11.5.3 Progress Tracking

Students can track their progress by enrolling in courses, submitting assignments, and viewing their grades. The system updates and displays progress in real time.

### 11.5.4 Responsive UI

The platform is designed to be responsive, so students and instructors can use the platform across various devices (computers, tablets, smartphones). The user interface adapts to different screen sizes, ensuring a smooth user experience.

## 11.6. Code Snippets

### 11.6.1 User Authentication Code Example (Backend)

```
const jwt = require('jsonwebtoken');  
  
const bcrypt = require('bcrypt');
```

```

const User = require('../models/User');

const loginUser = async (req, res) => {

const { email, password } = req.body;

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

if (!user || !(await bcrypt.compare(password, user.password))) {

return res.status(400).json({ message: "Invalid credentials" });

}

const token = jwt.sign({ id: user._id }, 'secretKey', { expiresIn: '1h' });

res.json({ message: "Login successful", token });

};

```

### 11.6.2 Course Upload Code Example (Backend)

```

const Course = require('../models/Course');

const uploadCourse = async (req, res) => {

const { title, description, materials } = req.body;

const newCourse = new Course({ title, description, materials, instructorId: req.user.id });

await newCourse.save();

res.status(201).json({ message: "Course uploaded successfully", course: newCourse });

}

```

## 11.7. Testing

Testing was performed to ensure the correct functionality of the platform. The following tests were conducted:

- **Unit Tests:** To verify individual functions and components (e.g., login, registration, course management).
- **Integration Tests:** To ensure that the different parts of the system work together as expected.
- **UI/UX Tests:** To validate the responsiveness and user experience of the platform.

## 11.8. Challenges Faced During Implementation

Some challenges faced during implementation include:

- **Handling asynchronous operations** in Node.js.
- **Ensuring secure user authentication** with JWT and password hashing.
- **Managing course material uploads** in a scalable and user-friendly manner.

## 11.9. Conclusion

The **implementation** of the Online Learning Platform has been successfully carried out by using modern technologies such as Node.js, React.js, and MongoDB. The system allows students to access courses, track progress, and earn certifications, while instructors can upload and manage course content. The platform is designed to be scalable and secure, ensuring a seamless learning experience for users.

This **Implementation** chapter provides a detailed overview of how the project was executed, the technologies used, and how the features were implemented. Let me know if you need further elaboration or additions!

## 12. APPENDICES

The **Appendices** section contains supplementary material that supports the main content of the report, such as code snippets, diagrams, and detailed explanations. Below is an outline of what could be included in the appendices:

### Appendix A: Source Code

- **Backend Code:** A complete list of code files related to backend implementation (Node.js, Express, MongoDB connections, etc.).

```
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}`))
```

```
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,

};

```

- **Frontend Code:** Code files for the frontend (React components, styling, etc.).

```

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App.jsx';

import reportWebVitals from './reportWebVitals.js';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

```

```

<React.StrictMode>

<App />

</React.StrictMode>

);

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 [userData, setUserData] = useState();

  const [userLoggedIn, setUserLoggedIn] = useState(false);

  const getData = async () => {

    try {

      const user = await JSON.parse(localStorage.getItem("user"));

```

```

if (user && user !== undefined) {

  setUserData(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 /> } />

            {userLoggedIn ? (

              <

```

```
<Route path="/dashboard" element={<Dashboard />} />

<Route path="/courseSection/:courseId/:courseTitle" element={<CourseContent />} />

</>

):(

<Route path="/login" element={<Login />} />

)}

</Routes>

</div>

</Router>

</div>

</UserContext.Provider>

);

}

export default App;
```

## Appendix B: Database Schema

```
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"],
```

```

    },
    },
    {
    timestamps: true,
    }
    );

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

module.exports = userSchema;


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;

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;

```

```

const enrolledCourseSchema = mongoose.Schema(
{
  courseId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "course",

```

```

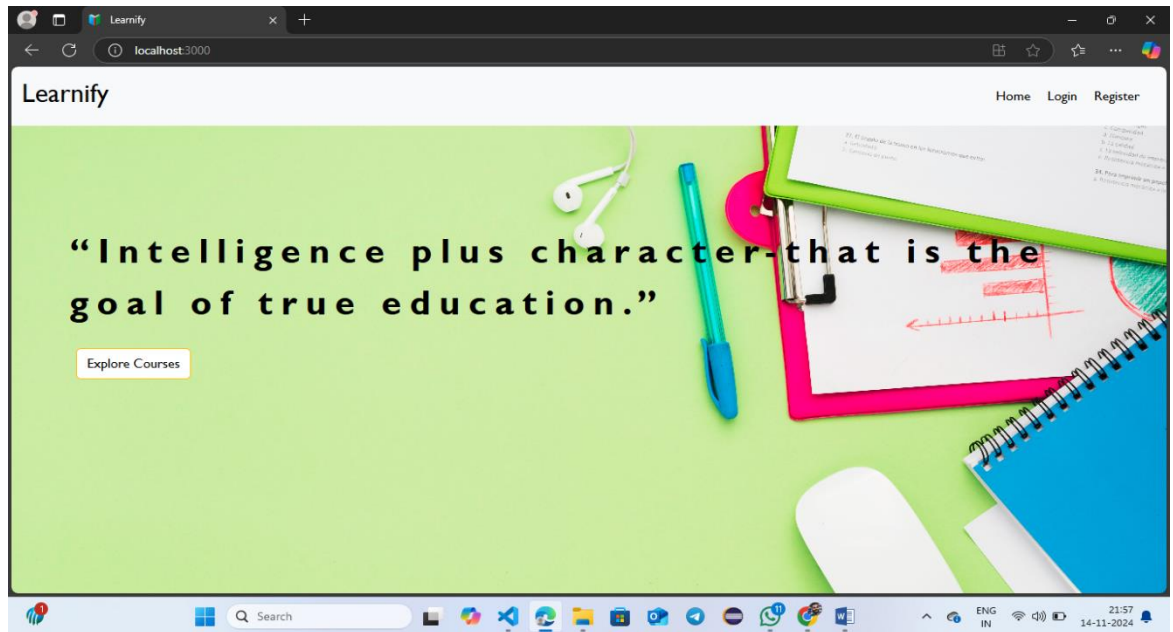
    },
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "user",
    },
    course_Length: {
      type: Number,
      required: true,
    },
    progress: [{}],
    certificateDate: {
      type: Date,
    },
  },
  {
    timestamps: true,
  }
);

module.exports = mongoose.model("enrolledCourses", enrolledCourseSchema);

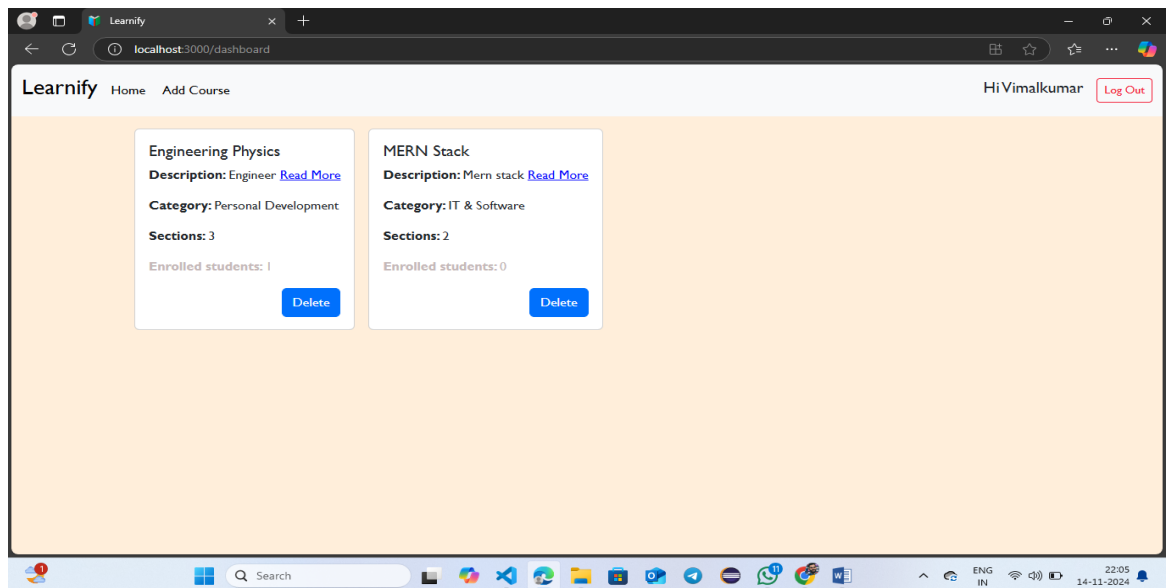
```

## Appendix C: Screenshots of the System

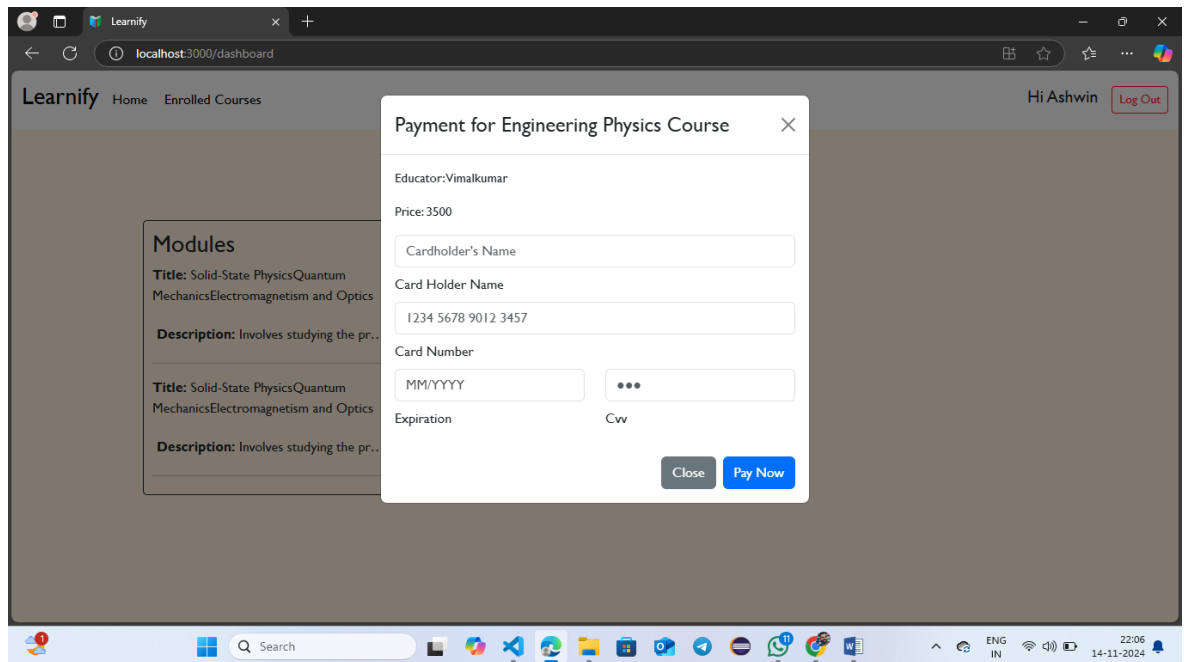
### Home Page



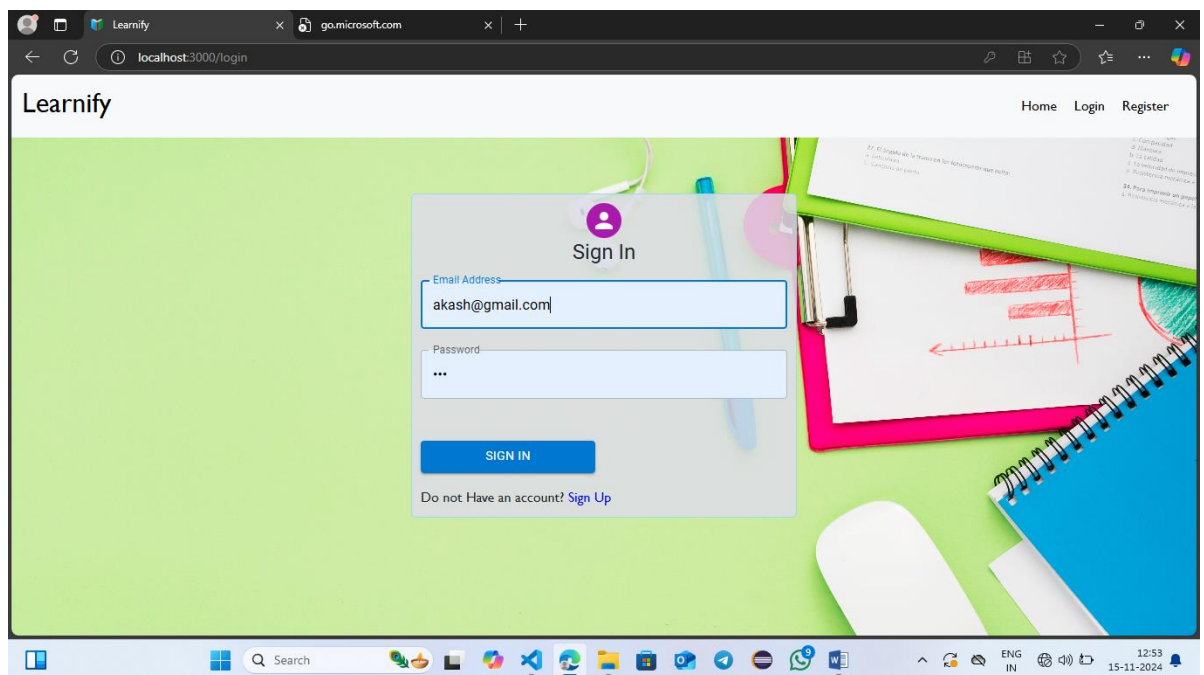
### The Dashboard after login.



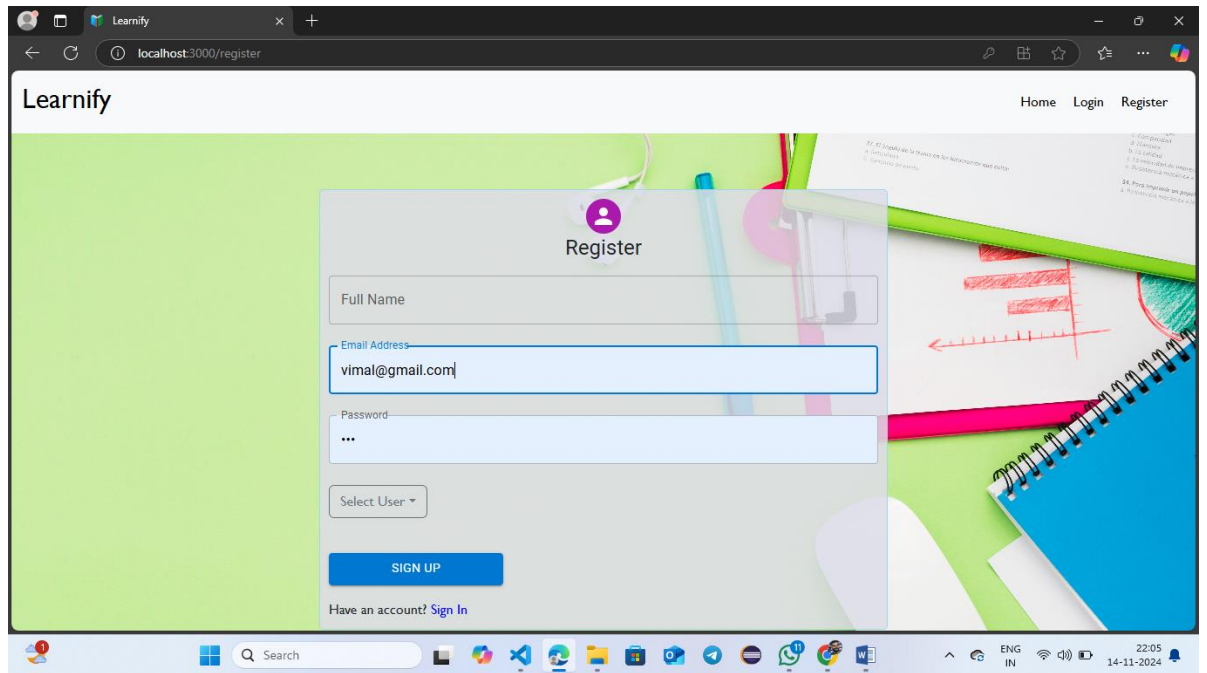
## Course Enrollment page.



## User Login page.



## User Sign-up page.



The screenshot shows a web browser window with the URL `localhost:3000/register`. The page title is "Learnify". In the top right corner, there are links for "Home", "Login", and "Register". The main content area features a "Register" form with a purple user icon. The form includes the following fields and elements:

- Full Name**: An empty text input field.
- Email Address**: A text input field containing the email `vimal@gmail.com`.
- Password**: A text input field with masked characters (dots).
- Select User**: A dropdown menu.
- SIGN UP**: A blue button.
- Have an account? [Sign In](#)**: A link at the bottom of the form.

The background of the page is a light green wall with a desk setup featuring a blue spiral notebook, a pink highlighter, and a green pen. The Windows taskbar is visible at the bottom, showing the search bar and various application icons.

## 13.CONCLUSION

The development and implementation of the **Online Learning Platform** have successfully achieved the objectives set out at the beginning of the project. This platform provides an intuitive and user-friendly interface that caters to both students and instructors, offering features such as user registration, course management, and progress tracking. The platform allows for a flexible and self-paced learning experience, making education more accessible to users across the globe.

The project was implemented using modern technologies, such as **React.js** for the frontend, **Node.js** and **Express.js** for the backend, and **MongoDB** for database management. The system supports authentication via **JWT**, ensuring secure user access, while **responsive design** ensures that the platform is accessible on various devices, from desktops to smartphones.

Key features, such as course uploads by instructors, real-time progress tracking, and the ability for students to enroll in and view courses, were implemented successfully. Despite some challenges, including managing asynchronous operations and handling secure data transmission, the final product provides a robust learning platform that meets the needs of both students and instructors.

This project demonstrates how technology can be used to enhance education by providing a platform that is scalable, secure, and user-friendly. The system can be further expanded to include features like discussion forums, live webinars, and advanced reporting, ensuring that it can continue to meet the evolving needs of users.

## 14. REFERENCES

The **References** section includes all the sources of information, tools, libraries, and frameworks used throughout the project. Below is a sample list of references that you might have used:

### 1. **Books:**

- Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education.
- Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill.

### 2. **Websites:**

- React.js Documentation. (2024). *React – A JavaScript library for building user interfaces*. Retrieved from <https://reactjs.org/>
- MongoDB Documentation. (2024). *MongoDB: The Database for Modern Applications*. Retrieved from <https://www.mongodb.com/>
- Node.js Documentation. (2024). *Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine*. Retrieved from <https://nodejs.org/>

### 3. **Research Papers:**

- Li, Z., & Zhang, X. (2018). *E-learning System Design: Principles and Best Practices*. International Journal of Educational Technology, 15(3), 56-67.

### 4. **Libraries and Frameworks:**

- **Node.js**. (2024). *Node.js Documentation*. Retrieved from <https://nodejs.org/>
- **Express.js**. (2024). *Express - Fast, unopinionated, minimalist web framework for Node.js*. Retrieved from <https://expressjs.com/>
- **JWT.io**. (2024). *JWT: JSON Web Token*. Retrieved from <https://jwt.io/>
- **Bootstrap**. (2024). *Bootstrap: The most popular HTML, CSS, and JS library in the world*. Retrieved from <https://getbootstrap.com/>