Online Learning Platform using MERN

Project Title : LearnHub - Your Center for Skill Enhancement

Team ID : LTVIP2025TMID55995

Team Size : 4

Team Leader : Mylapalli Yesebu

Team Member : Nambu Sree Jaya Krishna Siva Sai Bulli Pavan

Team Member : Sajjala Ganesh Siva Surya Srinivas

Team Member : P Sai Govind

Section 1 : INTRODUCTION

An **Online Learning Platform (OLP)** is a digital environment providing students and instructors with the resources and tools they require to provide education through the internet. OLPs have been widely accepted as they are versatile, available, and can meet different learning requirements in different settings and ages.

Some of the major characteristics of online learning platforms include:

User-Centric Interface: Designed to become intuitive and easy to use, with the option of being utilized by users with varying technical expertise.

Course Management: Enables lecturers to write, upload, and manage materials, while students can take courses, access resources, and monitor learning progress.

Interactivity: Provides interactive elements such as discussion boards, live webinars, and chat, which support communication and collaboration amongst students-staff and students-students.

Certification: Issues digital certifications or badges upon successful course completion, giving the learner recognition that can benefit career advancement or future learning.

Accessibility: Built for different devices—desktops, tablets, and smartphones—to make learning anywhere, anytime possible with an internet connection.

Self-Paced Learning: Enables students to learn at their convenience, providing flexibility and individualized learning paths.

Payment and Subscription Models: Free and paid content, with various price models such as single-upfront payments, subscriptions, or freemium availability.

Section 2: PROJECT OVERVIEW

Scenario: Acquiring a New Skill on an Online Learning Platform
The case study represents how various users—students, teachers, and admins—engage with an online learning platform to serve their respective purposes.

Student Journey: Sarah's Learning Experience
1. User Registration
Sarah, a student looking to learn web development, comes to the platform and registers by entering her email and creating a password.

2. Browsing Courses
When she logs in, she is greeted by a simple, intuitive interface that lists a multitude of courses arranged

by category, difficulty, and popularity. She navigates the catalog employing filters on course name and category, before finally choosing a course called "Web Development Fundamentals."

3. Course Enrollment
Sarah reads the course syllabus, instructor bio, and course description. Liking what she sees, she signs up for the course. After signing up, she has access to the course content, such as video lectures, assignments, and notes.

4. Progress in Learning
She starts learning at her convenience. The website automatically records her progress, so she can pick up from where she stopped if she wants to stop and continue later.

5. Interaction and Support
Throughout the course, Sarah engages with discussion forums and attends live webinars in order to clear her doubts. These interactive features intensify her learning and keep her in touch with instructors and peers.

6. Course Completion and Certification
Once she has completed all the modules and assignments, Sarah sits for the final exam. Having passed, she is issued an electronic certificate of completion, which she downloads and includes in her academic portfolio.

7. Accessing Paid Content
Sarah subsequently investigates a paid advanced web development course. She buys the course using the site's secure payment system and gains access to advanced content.

Teacher's Role: Instructor John
John, a seasoned web developer, plays the role of a teacher. He creates and posts advanced course content, includes sections with videos and resources, and is responsible for course updates. He can see enrollments and reply to student interactions.

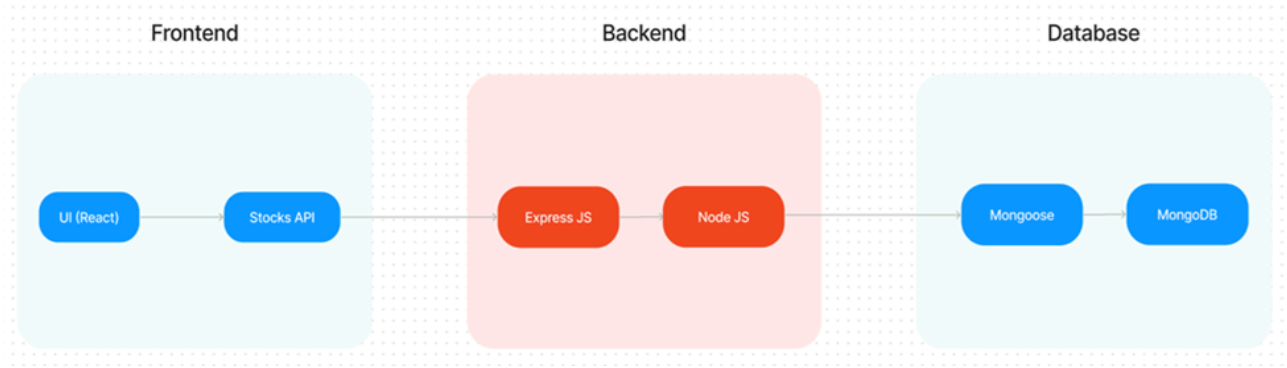
Admin Oversight
The Admin oversees the entire platform for seamless functioning. Tasks involve:

- Overseeing user activity
- Managing and authenticating course listings
- Addressing support requests and technical issues
- Ensuring platform integrity and data security

This situation describes how a well-architected online learning platform offers a seamless experience across all user roles—enabling students, facilitating educators, and aiding administrative management.

Section 3: TECHNICAL ARCHITECTURE

The technical architecture of the Online Learning Platform (OLP) is a client-server architecture where the client is the frontend and the server is the backend. They communicate with each other and exchange information using RESTful APIs, enabling the flow of information in a seamless and secure way.



At the frontend, React.js is utilized to build the application, complemented by UI libraries like Bootstrap and Material UI to render a user-friendly and responsive interface experience. The Axios library is utilized in making optimal interactions with the backend through API requests.

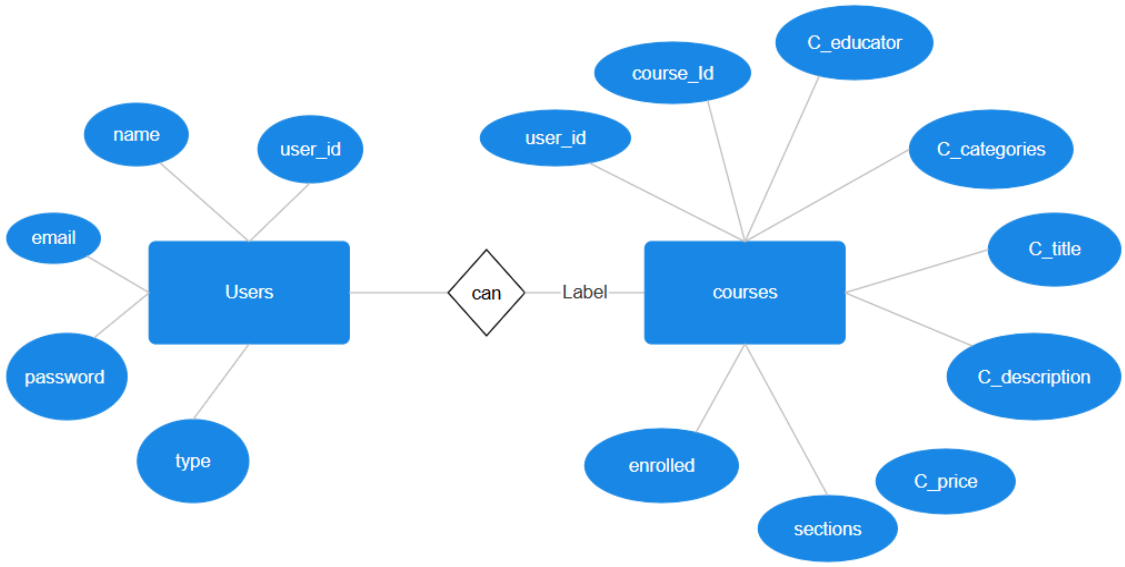
The backend is built with Express.js and Node.js, which execute server-side logic, API routing, authentication, and business logic. The backend takes responsibility for managing user roles, course actions, and student progress.

Data storage is provided by MongoDB Atlas, a NoSQL cloud database. MongoDB stores user data, course data, sections, and enrollment data in an efficient and scalable manner through Mongoose models.

All three technologies integrate well together to form a solid, scalable MERN stack architecture. They enable real-time data handling, role-based access control, and a seamless learning environment for all stakeholders—learners, teachers, and administrators.

Section 4: ER - Diagram

The database structure of the Online Learning Platform is designed using **MongoDB**, which stores data in flexible and scalable **collections**.



Here there are 2 collections, namely users and courses, that have their own fields in

Users Collection:

1. User_id (Unique identifier automatically created by MongoDB)
2. name (full name of the user)
3. email (User's email used for login)
4. password (Encrypted password)
5. Type (Role of the user like student, teacher, etc)

Courses Collection:

1. User_id (Unique course identifier auto-generated)
2. Courses_id (ID of the teacher who created the course, foreign reference to `Users`)
3. C_educator (name of the course instructor)
4. C_categories (Course category, e.g., Web Dev, AI, etc.)
5. C_title (Title of the course)
6. C_description (Brief summary of the course content)
7. Sections (List of course sections with title, content, and description)
8. C_price (Course price (0 for free courses))
9. Enroll (List of enrolled student user IDs)

Section 5: PRE-REQUISITES

To develop a full-stack Online Learning Platform (OLP) using the **MERN stack—MongoDB, Express.js, React.js, and Node.js**—the following tools, libraries, and skills are essential:

Vite (Frontend Build Tool):

Vite is a modern frontend build tool that provides a fast and optimized development experience. It includes, A lightweight development server with native ES module support, Hot Module Replacement (HMR), and Production builds powered by **Rollup.**
To create a new Vite-based React app:

```
npm create vite@latest
```

Node.js and npm:

Node.js is a JavaScript runtime that lets you run JavaScript on the server-side. It is paired with **npm (Node Package Manager)** to install dependencies.

- Download: nodejs.org
- Installation Help: Node.js Setup Guide
- To initialize a Node.js project:

```
npm init
```

Express.js (Backend Framework):

Express.js is a fast and minimal web framework for Node.js that simplifies routing, middleware, and server-side logic. To install:

```
npm install express
```

MongoDB (Database):

MongoDB is a NoSQL document-oriented database that stores data in a flexible, JSON-like structure. It's known for scalability and real-time performance.

- Download: MongoDB Community
- Setup Guide: Installation Docs

For database connectivity, use **Mongoose** (ODM) with Node.js:
🔗 Mongoose + Node.js + MongoDB Guide

React.js (Frontend Library):

React.js is a popular JavaScript library used for building fast, dynamic user interfaces via reusable components.

Get started: [React Docs]

HTML, CSS, and JavaScript:

Basic knowledge of **HTML** for creating the structure of application, **CSS** for styling, and **JavaScript(JS)** 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. To Connect the Database with Node JS go through the below provided link:

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

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:

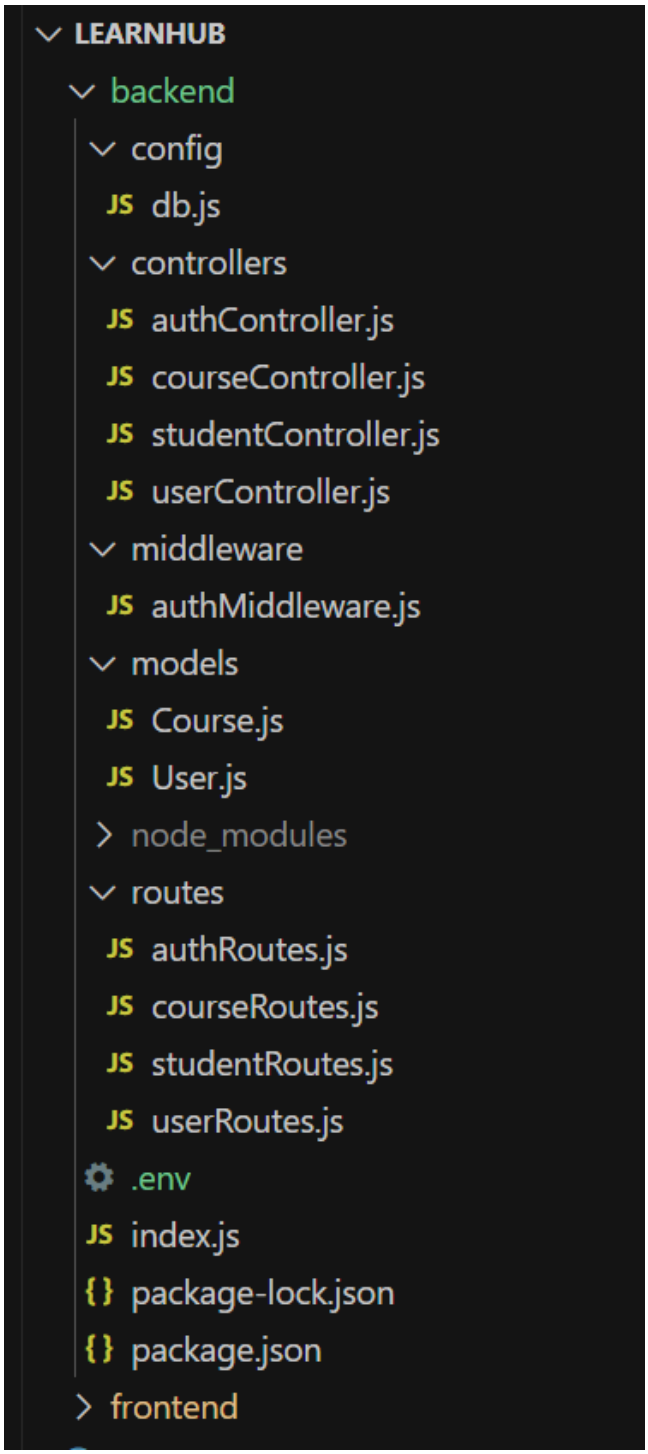
  To start the development server, execute the following command:

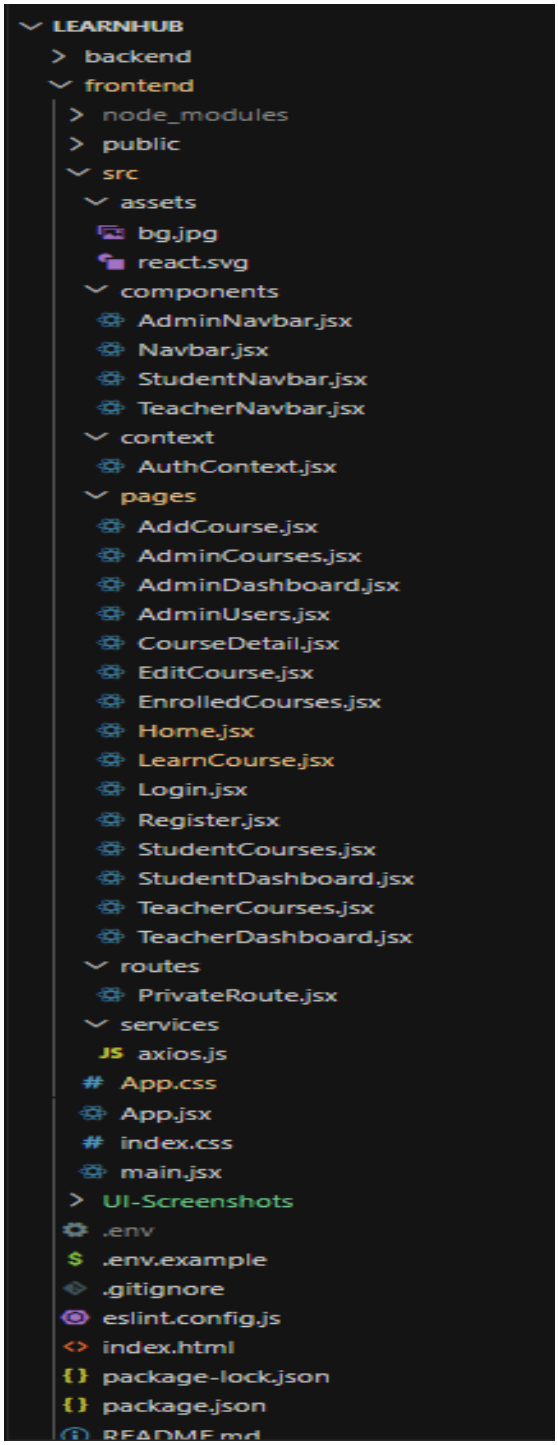
  ```
  npm start
  ```
- The OLP app will be accessible at http://localhost:5172

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.

Section 6: PROJECT STRUCTURE

The LearnHub application is organized into two main parts: the **Frontend** and the **Backend**. Each part contains a structured set of files and folders that support modular, scalable development and role-based functionality.





Frontend (Client-side):

The frontend of the application is built using **React.js** with **Vite** as the build tool. It contains all the components, pages, services, routing, and styling necessary for rendering the user interface.

Key folders and files include:

- components/ – Reusable UI components like Navbar, Dashboards, etc.
- pages/ – Role-based pages: Admin, Teacher, and Student views
- routes/ – Custom protected routes (e.g., PrivateRoute.jsx)
- context/ – Global state management (e.g., AuthContext)
- services/ – Axios setup for API calls
- App.jsx – Route definitions and layout rendering
- main.jsx – Entry point of the React application
- assets/ – Static files like images or logos
- .env – Stores environment variables (API base URL)

Backend (Server-side):

The backend is developed using **Node.js** and **Express.js**. It includes route files, controller logic, and MongoDB-based models. This layer handles all the business logic, API endpoints, authentication, and database communication.

Key folders and files include:

- routes/ – Route definitions (auth, user, course, student)
- controllers/ – Handles the logic for each route
- models/ – Mongoose schemas for User and Course
- middleware/ – Auth middleware to protect routes
- config/db.js – MongoDB connection setup
- index.js – Entry point of the server
- .env – Environment variables for DB URI, JWT secret, etc.

Section 7: Application Flow

LearnHub has three different roles of users—**Admin**, **Teacher**, and **Student**—each with their own different access rights and roles. The application flow is developed using role-based routing and API control to enable secure and tailored experiences for each type of user.

Teacher Role

Teachers are tasked with developing and handling course content for students. Their features are:

- Create New Courses: Enter detailed course details like title, description, category, and price.
- Add Course Sections: Add multiple sections (e.g., videos, images, text) in a course.
- Delete Courses: Delete any course, as long as no students are taking it.

Student Role

The students are the main learners on the platform. Their learning process comprises:

- Course Enrollment: Browse and join free or paid courses.
- Resume Learning: Resume learning where they left off.
- Progress Tracking: The learning progress is automatically stored on the platform.
- Certificate Download: Once a course is finished, students can download a completion certificate.
- Search and Filter: Find courses by name, category, or other filtering.
- Buy Paid Courses (upcoming improvement): Access to premium material upon secure payment.

Admin Role

Admins have full control of the site. Their duty is:

- User Management: See and manage all users (Students and Teachers), role updates, and deletion (except Admins).
- Course Management: View, revise, and remove any course present in the platform.
- Enrollment Monitoring: Monitor students' enrollments in all courses and maintain platform integrity.
- By having this role-based architecture, there is ensured separation of duties, secure access control, as well as an optimized exp aligned with each user's role in the learning environment.

Section 6: API DEVELOPMENTATION

Milestone 1: Project Setup & Configuration

The first milestone focused on setting up the project's folder structure and configuring the required tools and dependencies for both the frontend and backend.

Folder Structure Initialization

1. A main project directory was created, containing two subdirectories:
    - frontend/ – for client-side development using React.js
    - backend/ – for server-side logic using Node.js and Express

Backend Setup

The backend/ folder was initialized with a **Node.js** project using:

```
npm init -y
```

Installed Dependencies:

The following packages were installed to support API development, authentication, data handling, and file management:

Package	Purpose
`express`	Core Node.js web framework for building APIs
`cors`	Enables Cross-Origin Resource Sharing between frontend and backend
`dotenv`	Loads environment variables from a `.env` file
`mongoose`	MongoDB Object Data Modeling (ODM) library for schema-based DB access
`bcryptjs`	For hashing user passwords securely
`jsonwebtoken`	For implementing role-based JWT authentication
`multer`	Middleware for handling file uploads (e.g., images, videos)
`nodemon`	Auto-restarts the server during development

To install these, the following command was used:

```
npm install express cors dotenv mongoose bcryptjs jsonwebtoken multer
npm install --save-dev nodemon
```

This configuration ensured that the backend was ready for further development of user authentication, course APIs, and database integration.

Milestone 2 – Backend Development

This milestone involved setting up the **Express server**, configuring middleware, and implementing **authentication logic** to secure the backend routes.

Express Server Setup

The backend development began by creating a new file named:

`backend/index.js`

This file serves as the **entry point of the server**.

In this step:

- The **Express server** was initialized
- A custom **port number** was defined
- The **MongoDB connection string** and **JWT secret key** were loaded from the `.env` file using the `dotenv` package

Server Configuration

To support HTTP requests and data parsing, the following middleware were configured:

Middleware	Purpose
`cors`	Enables communication between frontend and backend
`express.json()`	Parses incoming request bodies as JSON

Authentication Middleware

To protect routes and implement **role-based access**, a new folder named `middleware/` was created.

Inside it, a file called `authMiddleware.js` was added. This file handles, **JWT verification,** Extracting user data from tokens, and Blocking unauthorized access to private routes

This middleware was later imported and reused in protected routes such as:

- Course creation (**Teacher**)
- User management (**Admin**)
- Learning page (**Student**)

Environment Configuration

The `.env` file included the MOGO_URI, JWT_SECRET_KEY, and PORT number

This allows for secure and modular management of environment-specific settings.

With the Express server running and middleware in place, the backend was ready for further development of routes, models, and controllers.

Reference: backend.mp4 backend

Milestone 3 – Database Integration

This milestone focused on configuring the database for the application using **MongoDB Atlas**, and defining the necessary schemas to structure and store data efficiently.

MongoDB Configuration

To interact with MongoDB, the `mongoose` library was installed and imported into the project:

```
npm install mongoose
```

A dedicated file was created for handling database connections:

`backend/config/db.js`

This file uses `mongoose.connect()` to establish a connection between the Express server and MongoDB. The database connection string is securely loaded from the `.env` file using `process.env.MONGO_URI`.

Database Models

To structure application data, a new folder named `models/` was created:

`backend/models/`

Inside it, **Mongoose schemas** were defined for key entities such as:

- `User.js` – Defines the user fields (name, email, password, role)
- `Course.js` – Defines course details, including title, educator, description, price, and sections

These models allow the application to perform **CRUD operations** on MongoDB in a schema-based and organized way.

Reference: database.mp4 backend

Milestone 4 – Frontend Development

This milestone involved setting up the **frontend user interface** using React.js and integrating essential UI and utility libraries to support a smooth, responsive, and modern web experience.

Frontend Setup

The frontend was created using **Vite**, a fast build tool optimized for React.js:

```
npm create vite@latest
```

The development environment was initialized with the necessary tools to build a component-based, modular UI.

Installed Libraries and Tools

To enrich the design and functionality of the user interface, the following libraries were added:

Library	Purpose
React.js	Core frontend library for building UI components
Bootstrap	CSS framework for responsive design and layout
Material UI	Modern React UI components with customizable themes
Axios	For making HTTP requests to backend APIs
Ant Design (Antd)	Enterprise-level UI toolkit for React apps
MDB React UI Kit	React components based on Material Design by Bootstrap
React-Bootstrap	Bootstrap components built as React components

All dependencies were installed using:

```
npm install bootstrap @mui/material axios antd mdb-react-ui-kit
react-bootstrap
```

Component-Based UI Design

The application UI was divided into role-based pages and shared components:

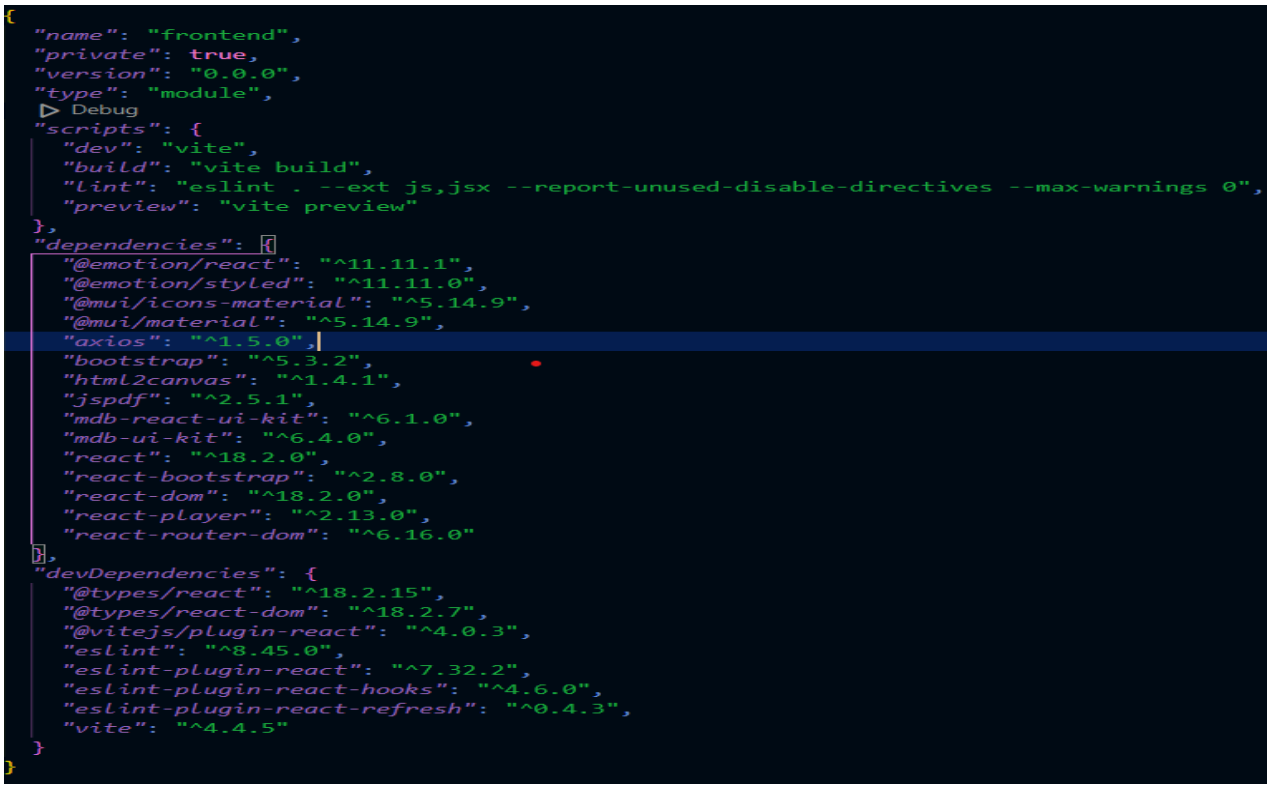
- `Navbar.jsx`, `Sidebar.jsx`, and dashboards for Admin, Teacher, and Student roles
- Pages like `Login`, `Register`, `EnrolledCourses`, `LearnCourse`, etc.
- Form handling and layout managed using **Bootstrap and MUI Grid system**

API Integration

Using **Axios**, frontend components were connected to backend endpoints to:

- Authenticate users
- Fetch, create, and update course data
- Track and update student progress

Reference: **frontend.mp4** frontend



```
{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  ▷ Debug
  "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"
  }
}
```

Milestone 5 – Project Implementation

After completing the development of both frontend and backend components, the project was fully integrated and tested. This final milestone involved **running the application end-to-end** to ensure that all modules were functioning correctly and that the user experience was smooth and responsive.

Final Testing and Debugging

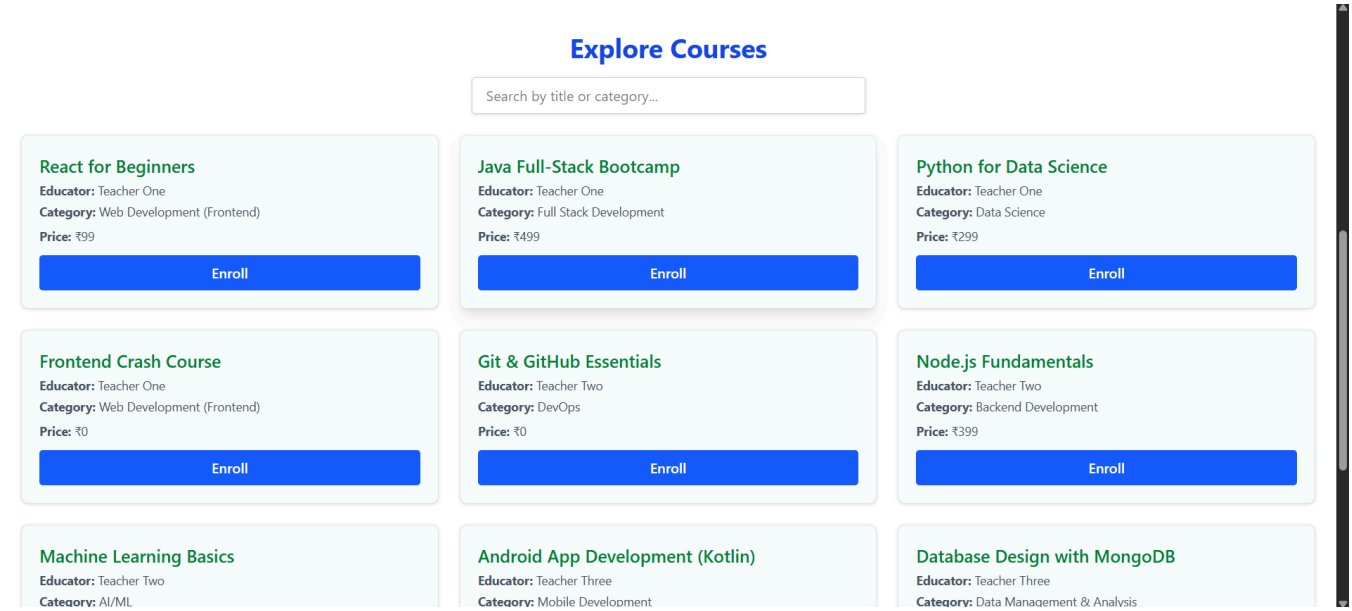
The application was launched locally, and thorough manual testing was performed to:

- Verify route protection and access control for Admin, Teacher, and Student roles
- Check course creation, enrollment, learning flow, and certificate generation
- Identify and fix any UI bugs, data handling issues, or unexpected behaviors

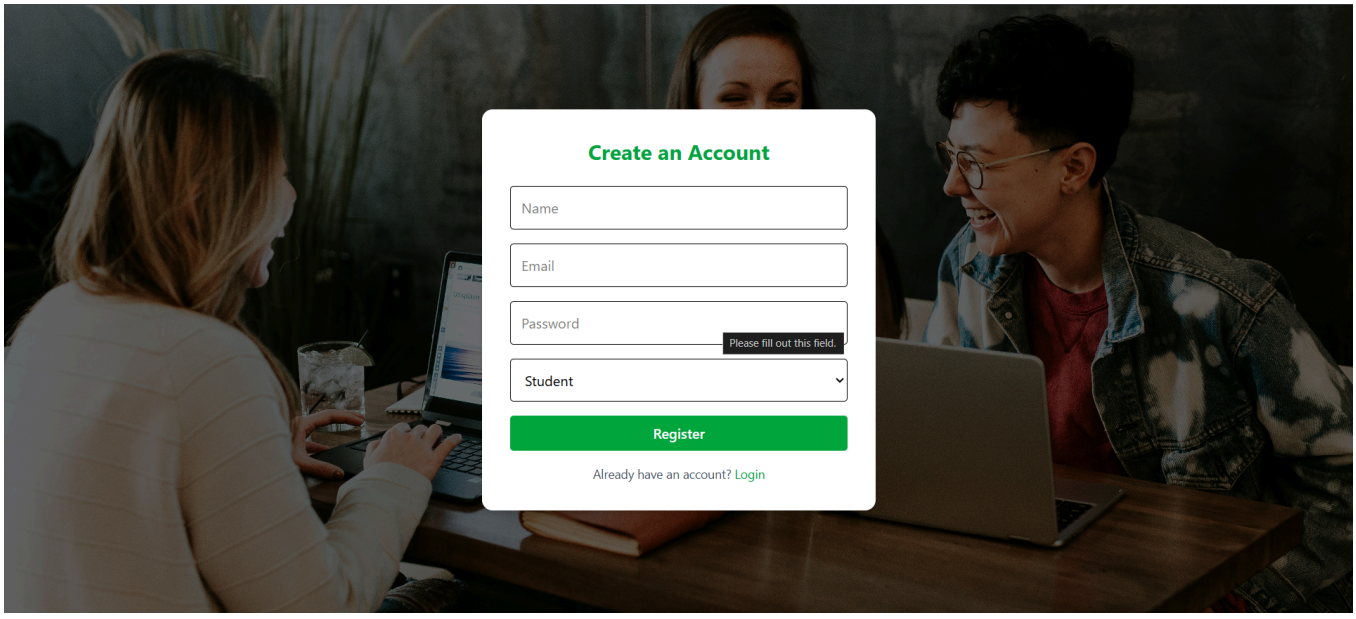
User Interface Previews

The following are sample views of the key UI screens from the LearnHub application:

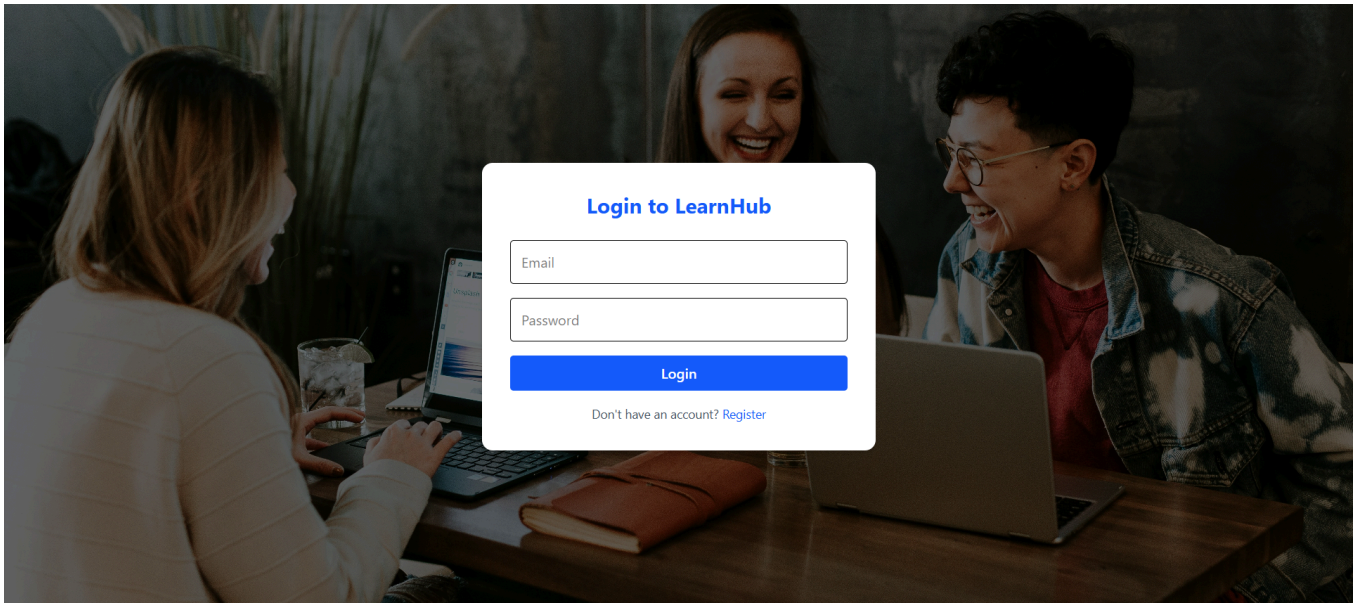
🏠 Landing/Home Page



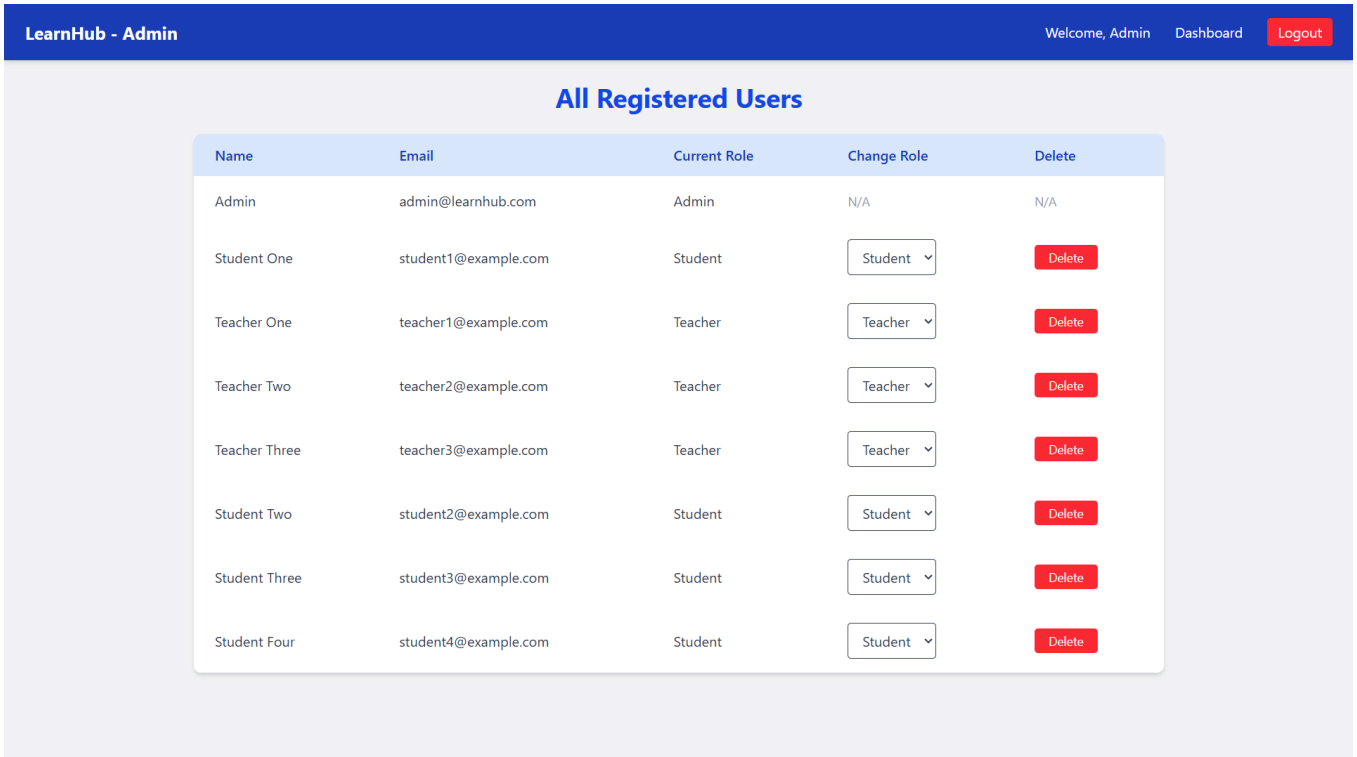
📝 Register Page



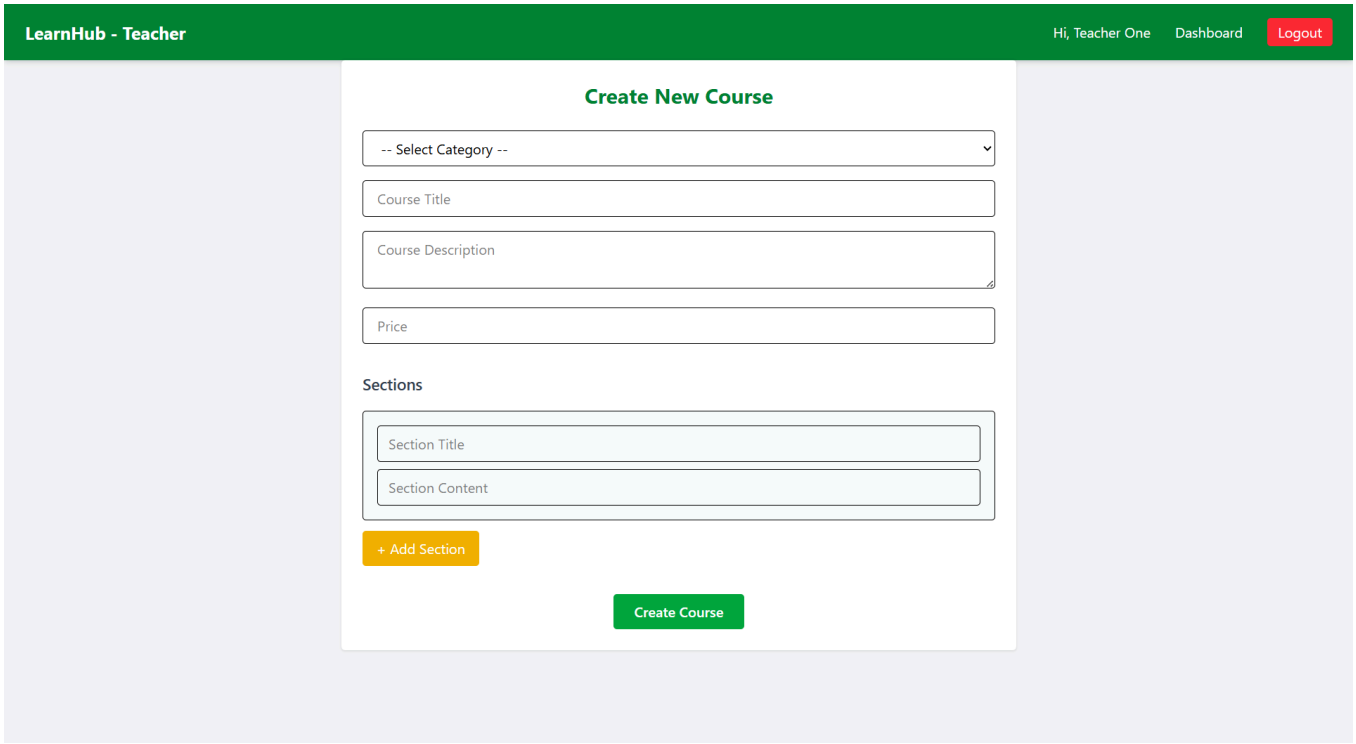
🔐 Login Page



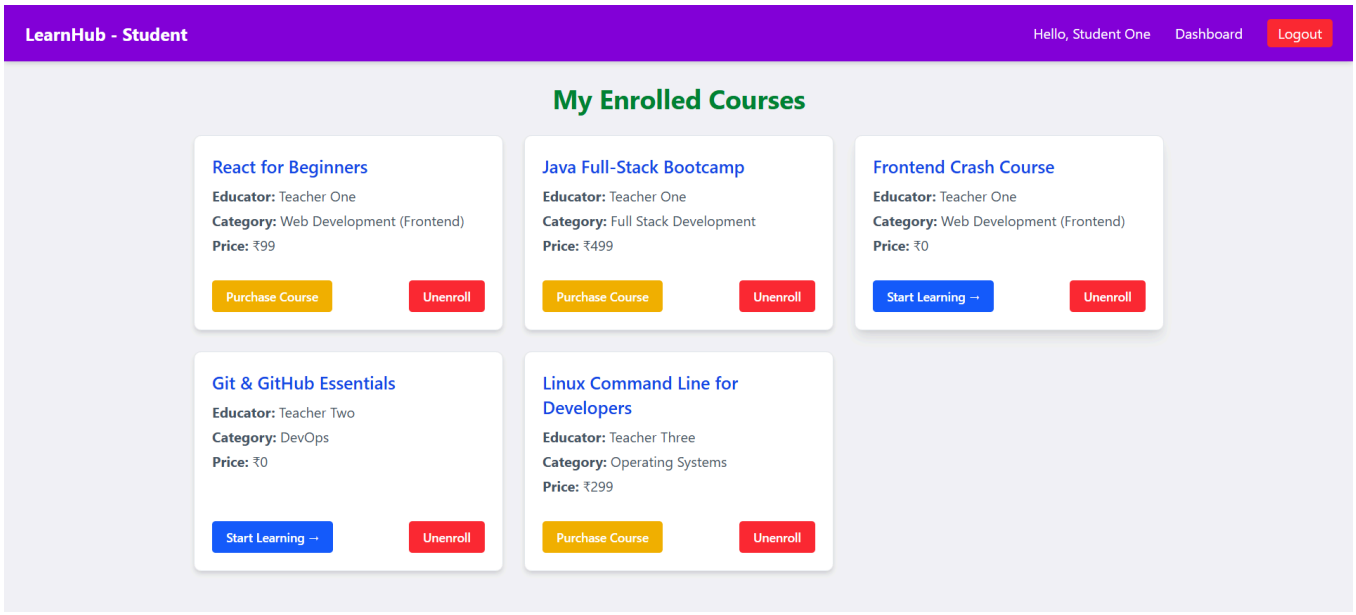
🧑‍💼 Admin Dashboard

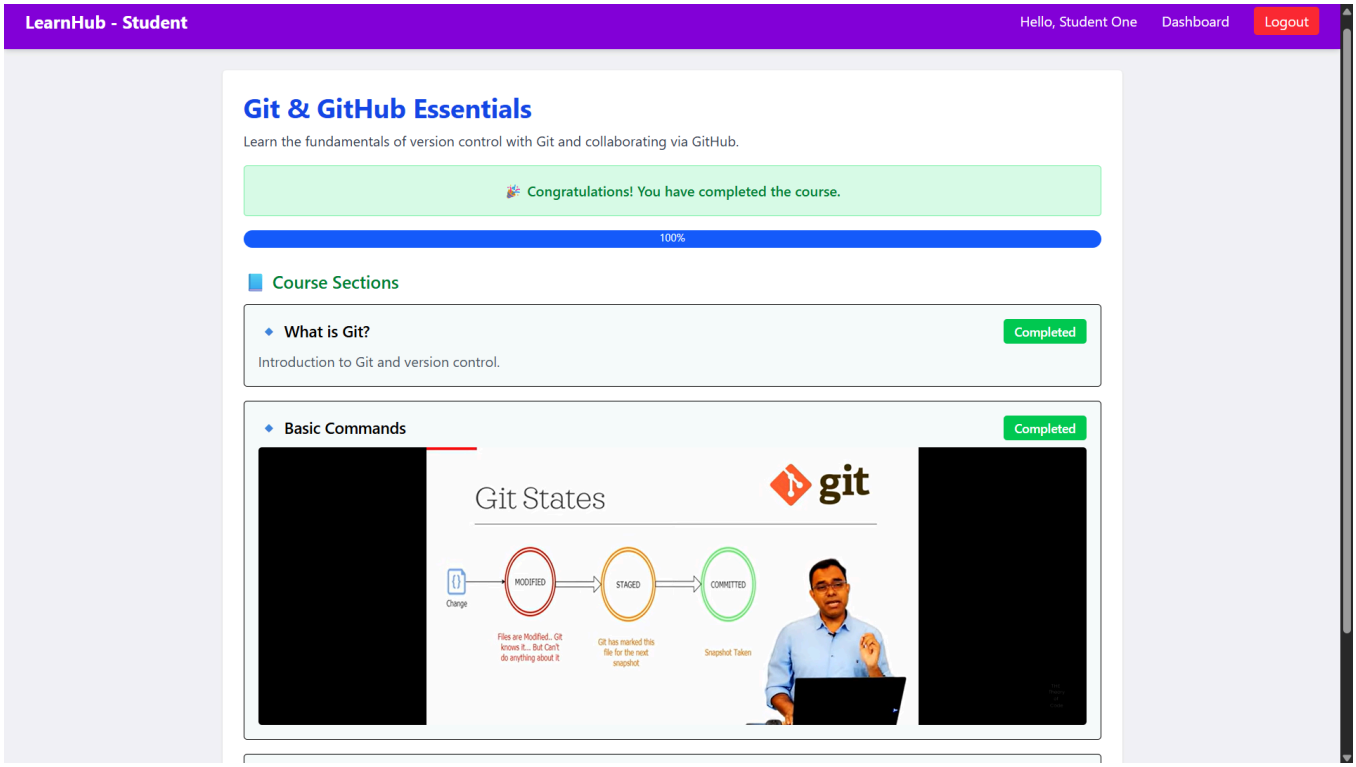


👩‍🏫 Teacher Dashboard



🎓 Student Dashboard





Project Links

- **Source Code Repository**: [GitHub – LearnHub MERN Internship]
- **Demo Video**: [Click here to watch]

This milestone concluded the LearnHub project with a successful implementation, full test coverage of major flows, and deployment-ready code.