**Full Stack Development with MERN**

**Project Documentation**

**1. Introduction**

**• Project Title:** LearnHub: Your Center for Skill Enhancement

**• Team Members:** Pavani G - Frontend Developer

Ganaparthi Nageswari- Backend Developer

Chemikala Lasya - Tester

**2. Project Overview**

**• Purpose**:  **LearnHub** is a scalable and responsive online learning platform built with **MERN Stack**, aiming to provide learners with accessible, self-paced, and Certified

Education in various fields.

**• Goals of the project include:**

* ***Make Learning Easy and Available for Everyone***: It help people learn new skills—especially tech skills—without needing to travel or pay huge fees. Whether you're in a city or a small village, you should be able to learn from home.
* ***Let Students Learn at Their Own Speed***: Life can get busy. So instead of fixed schedules, LearnHub lets users study whenever they have time—whether that’s at night, on weekends, or between other responsibilities.
* ***Help Teachers Share What They Know***: Instructors shouldn’t need technical knowledge to teach online. Our goal is to give them simple tools to create, upload, and manage their courses—and reach more students.
* ***Make Online Learning More Than Just Watching Videos***: We want to build a space where learners can ask questions, join discussions, and even attend live webinars—just like in a real classroom.
* ***Build Something That Can Grow with Time***: We’re not just building for today—we’re designing LearnHub to grow, evolve, and adapt with time as new needs and features arise.

**• Features:**

* User registration/login
* Browse and enroll in categorized courses
* Instructor dashboard to upload and manage content
* Course creation, editing, and management by teachers and admins
* Admin panel for platform control
* Progress tracking and certifications
* Payment gateway for premium content
* Student enrollment in courses and access to course content
* Media uploads and streaming for course videos and materials
* Progress tracking for students
* Real-time updates and notifications for important activities
* Data security and privacy for all user information

**3. Architecture**

**• Frontend:**

We built the frontend using **React** to keep everything fast and interactive.  
Each part of the platform—student, teacher, admin—has its own dedicated components. **React Router** handles navigation, and **Hooks** + **Context API** manage data and state. Design libraries like **Bootstrap**, **Material UI**, and **Ant Design** make it look clean and modern. We used **Axios** to send and receive data from the backend quickly and smoothly.

**• Backend:**

We built the backend using **Node.js** and **Express.js** to keep things fast and organized.  
Everything is neatly divided—there are separate routes for users, admins, and courses. We use **controllers** to handle the main logic and **middleware** to manage login checks and data validation. All actions like signing in, creating courses, or uploading content are handled through **RESTful APIs**. For security, we use **JWT** to protect user sessions and **bcrypt.js** to safely hash passwords. This structure keeps the backend clean, secure, and easy to scale as the platform grows.

**• Database:**

We used **MongoDB** as our database to store all the important data like users, courses, and enrollments. To work with the database easily, we used **Mongoose**, a library that helps us structure and manage data. Each piece of data is connected—for example, courses link to their teachers, and enrollments link to courses.  
With **Mongoose models**, we can create, read, update, and delete data smoothly and consistently. This setup helps keep everything organized, reliable, and ready for fast searching and filtering.

**4. Setup Instructions :-**

**• Prerequisites:**

* Node.js (v16 or above)
* MongoDB (latest stable version)
* npm (Node Package Manager)
* React (frontend library)
* Express.js (backend framework)
* Mongoose (MongoDB ODM for Node.js)
* Axios (HTTP client for frontend)
* Bootstrap, Material UI, Ant Design, mdb-react-ui-kit (UI libraries)
* bcryptjs (password hashing)
* jsonwebtoken (JWT authentication)
* multer (file uploads)
* dotenv (environment variable management)
* cors (Cross-Origin Resource Sharing)
* nodemon (development server auto-reload)

**• Installation:**

**1. Clone the Repository :**

git clone https://github.com/Nageswari87/LearnHub/tree/main

cd LearnHub

**2. Install Dependencies :**

**Backend-**

cd backend

npm install

**Frontend-**

cd frontend

npm install

**3. Set Up Environment Variables**

* In the backend folder, create a .env file.
* Add the following variables (example values, update as needed):

MONGO\_URI=mongodb://localhost:27017/learnhub

JWT\_SECRET=your\_jwt\_secret

PORT=5000

* In the frontend folder, if needed, create a .env file for frontend environment variables (e.g., API base URL):

VITE\_API\_URL=http://localhost:5000

**4. Start the Application**

**Backend-**

cd backend

npm start

**Frontend-**

cd frontend

npm run dev

**5. Folder Structure**

* **Client (React Frontend)-**

**frontend/**

**├── public/**

**├── src/**

**│ ├── components/**

**│ ├── assets/**

**│ ├── App.jsx**

**│ └── main.jsx**

**├── package.json**

**└── vite.config.js**

* public/: Contains static files and the main HTML template.
* src/components/: Houses all reusable UI components, organized by user roles.
* src/assets/: Stores images and other static resources.
* App.jsx: The main application component.
* main.jsx: Entry point for rendering the React app.
* package.json: Lists dependencies and scripts for the frontend.
* vite.config.js: Configuration for the Vite build tool.
* **Server (Node.js Backend)-**

**backend/**

**├── config/**

**├── controllers/**

**├── middlewares/**

**├── routers/**

**├── schemas/**

**├── uploads/**

**├── index.js**

**└── package.json**

* config/: Handles database connection setup.
* controllers/: Contains business logic for handling requests.
* middlewares/: Custom middleware for authentication and validation.
* routers/: Defines Express API routes.
* schemas/: Mongoose models for MongoDB collections.
* uploads/: Stores uploaded files such as course videos.
* index.js: Entry point for starting the backend server.
* package.json: Lists dependencies and scripts for the backend.

**6. Running the Application**

**Start the Backend Server-**

cd backend

npm start

**Start the Frontend Server-**

cd frontend

npm run dev

* **The backend server will run at: http://localhost:5000**
* **The frontend development server will run at: http://localhost:5173**

**7. API Documentation**

**User Authentication**

* **POST /api/auth/register**

**Description: Register a new user (student, teacher, or admin)**

**Request Body:**

{

  "name": "John Doe",

  "email": "john@example.com",

  "password": "password123",

  "role": "student"

}

**Response:**

{

   "message": "User registered successfully",

   "user": { ... }

}

* **POST /api/auth/login**

**Description: User login**

**Request Body:**

{

   "email": "john@example.com",

  "password": "password123"

}

**Response:**

{

  "token": "jwt\_token\_here",

   "user": { ... }

}

**Courses**

* **GET /api/courses**

**Description: Get all available courses**

**Response:**

[

  {

    "\_id": "courseId",

    "title": "Course Title",

    "description": "Course Description",

    "teacher": "teacherId"

  },

  ...

**]**

* **POST /api/courses**

**Description: Create a new course (teacher/admin only)**

**Request Body:**

**{**

  "title": "Course Title",

  "description": "Course Description"

}

**Response:**

{

  "message": "Course created successfully",

  "course": { ... }

}

**Enrollment**

* **POST /api/enroll**

**Description: Enroll a student in a course**

**Request Body:**

{

  "courseId": "courseId",

  "studentId": "studentId"

}

Response:

{

  "message": "Enrollment successful"

}

**Media Upload**

* **POST /api/upload**

**Description: Upload course content (video, etc.)**

**Request: multipart/form-data with file and courseId**

**Response:**

{

  "message": "File uploaded successfully",

  "fileUrl": "/uploads/filename.mp4"

}

* **POST /api/addcourse**

**Response:**

{

"message": "Course added successfully",

"course": { ... }

}

**Get All Courses  
 GET /api/getallcourses  
 *Response:***

[

  {

    "\_id": "60f1b2c4e1a2b3d4e5f6a7b8",

    "C\_title": "Introduction to React",

    "C\_description": "Learn the basics of React.",

    "teacher": "60f1b2c4e1a2b3d4e5f6a7b7"

  },

  {

    "\_id": "60f1b2c4e1a2b3d4e5f6a7b9",

    "C\_title": "Node.js Fundamentals",

    "C\_description": "Backend development with Node.js.",

    "teacher": "60f1b2c4e1a2b3d4e5f6a7b6"

  }

]

**Delete a Course  
 DELETE /api/deletecourse/:courseid  
 *Response:***

{

  "message": "Course deleted successfully"

}

**Add Section to Course  
 POST /api/addsection/:courseId  
 *Response:***

{

  "message": "Section added successfully",

  "section": {

    "\_id": "60f1b2c4e1a2b3d4e5f6a7c1",

    "title": "Getting Started",

    "content": "Introduction to the course...",

    "videoUrl": "/uploads/section1.mp4"

  }

}

**Enroll in a Course  
 POST /api/enrolledcourse/:courseid  
 *Response:***

{

  "message": "Enrollment successful",

  "enrollment": {

    "\_id": "60f1b2c4e1a2b3d4e5f6a7d1",

    "student": "60f1b2c4e1a2b3d4e5f6a7b5",

    "course": "60f1b2c4e1a2b3d4e5f6a7b8"

  }

**}**

**Unenroll from a Course  
 DELETE /api/enrolledcourse/:courseid  
 *Response:***

{

  "message": "Unenrolled from course successfully"

}

**Get Course Content  
 GET /api/coursecontent/:courseid  
 *Response:***

{

  "courseId": "60f1b2c4e1a2b3d4e5f6a7b8",

  "sections": [

    {

      "title": "Getting Started",

      "content": "Introduction to the course...",

      "videoUrl": "/uploads/section1.mp4"

    },

    {

      "title": "Advanced Topics",

      "content": "Deep dive into React...",

      "videoUrl": "/uploads/section2.mp4"

    }

  ]

}

**8. Authentication**

**Authentication and authorization in this project are handled using JSON Web Tokens (JWT).**

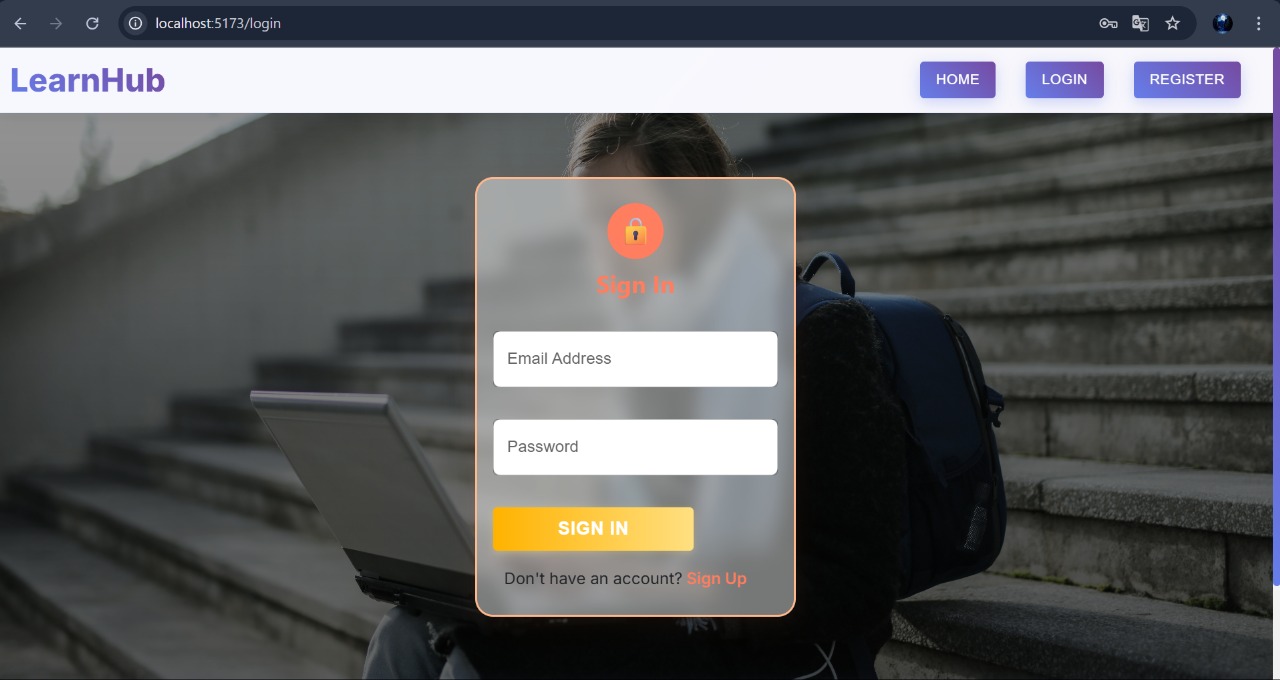
* **User Login:**When a user logs in, the backend verifies their credentials. If valid, a JWT is generated and sent to the client.
* **Token Usage:**The JWT contains user information and is stored on the client side. For all protected routes, the client includes the token in the Authorization header as a Bearer token.
* **Authorization:**Middleware on the backend checks the validity of the token for each protected request. It also verifies the user’s role (admin, teacher, student) to control access to specific resources and actions.
* **Session Management:**No server-side sessions are used; authentication is stateless and fully managed via JWTs.
* **Security:**Passwords are hashed using bcryptjs before storage. Tokens are signed with a secret key and can be set to expire after a certain period for added security.

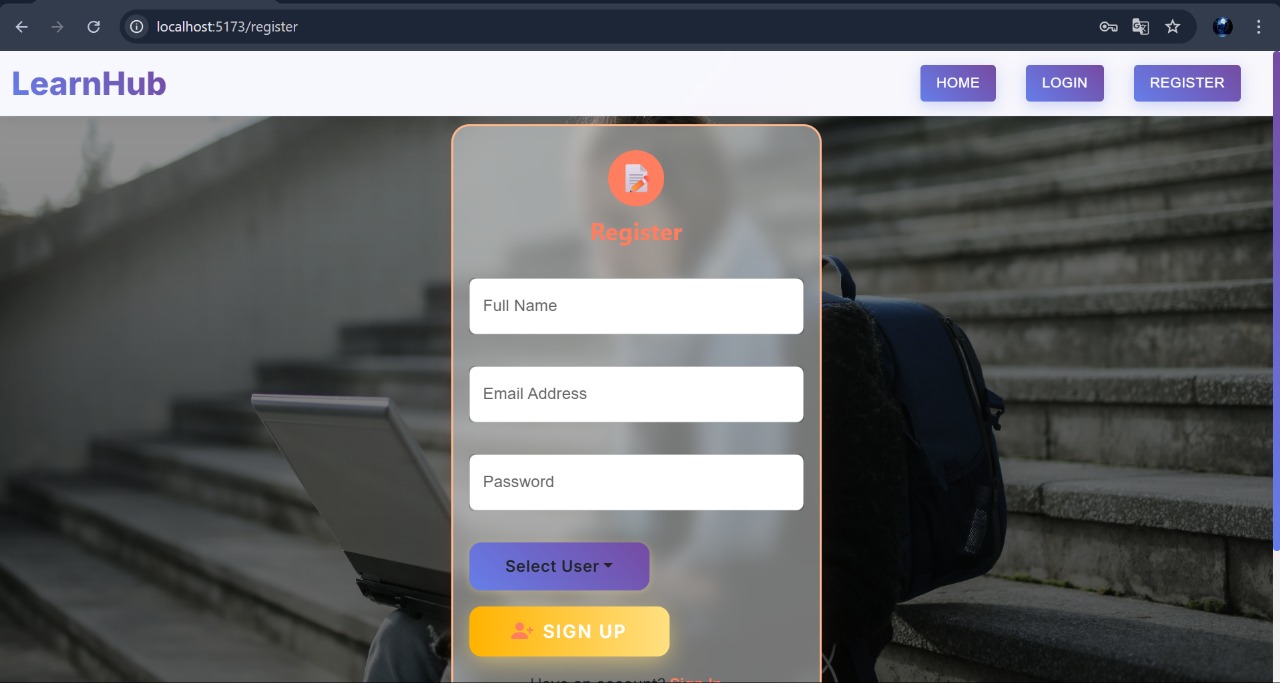
**9. User Interface**

**🔳 Home Page**

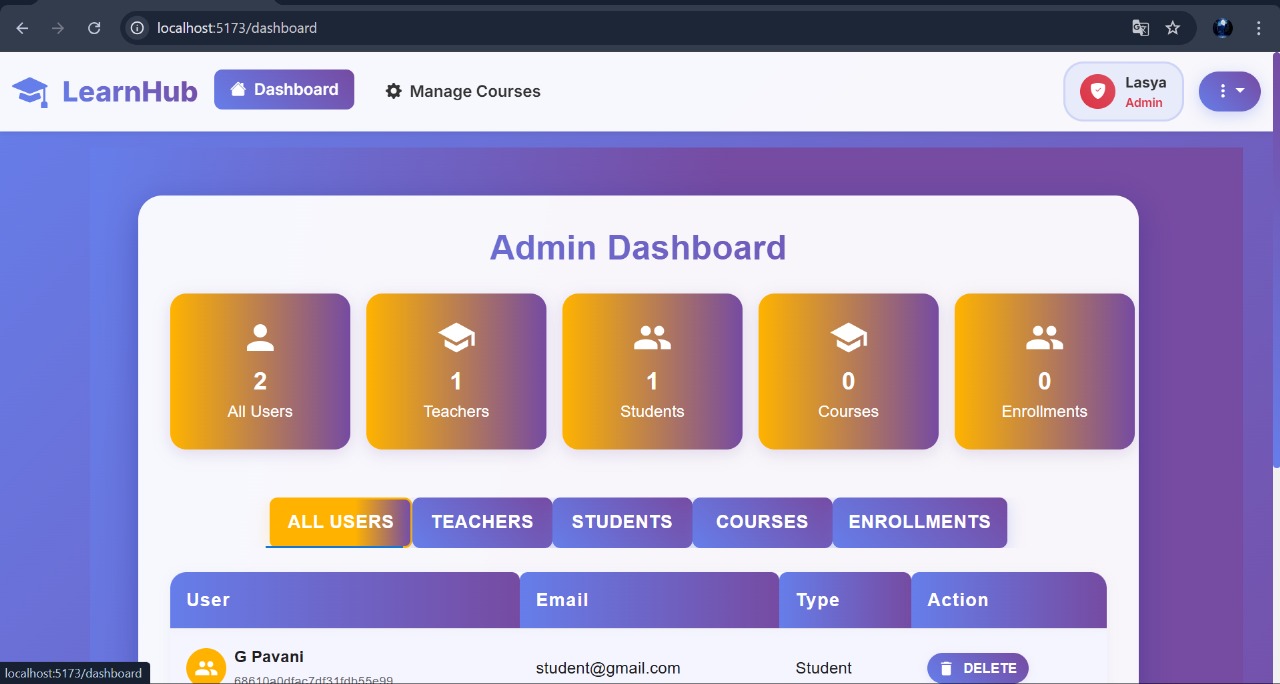


**🔳 Login Page**

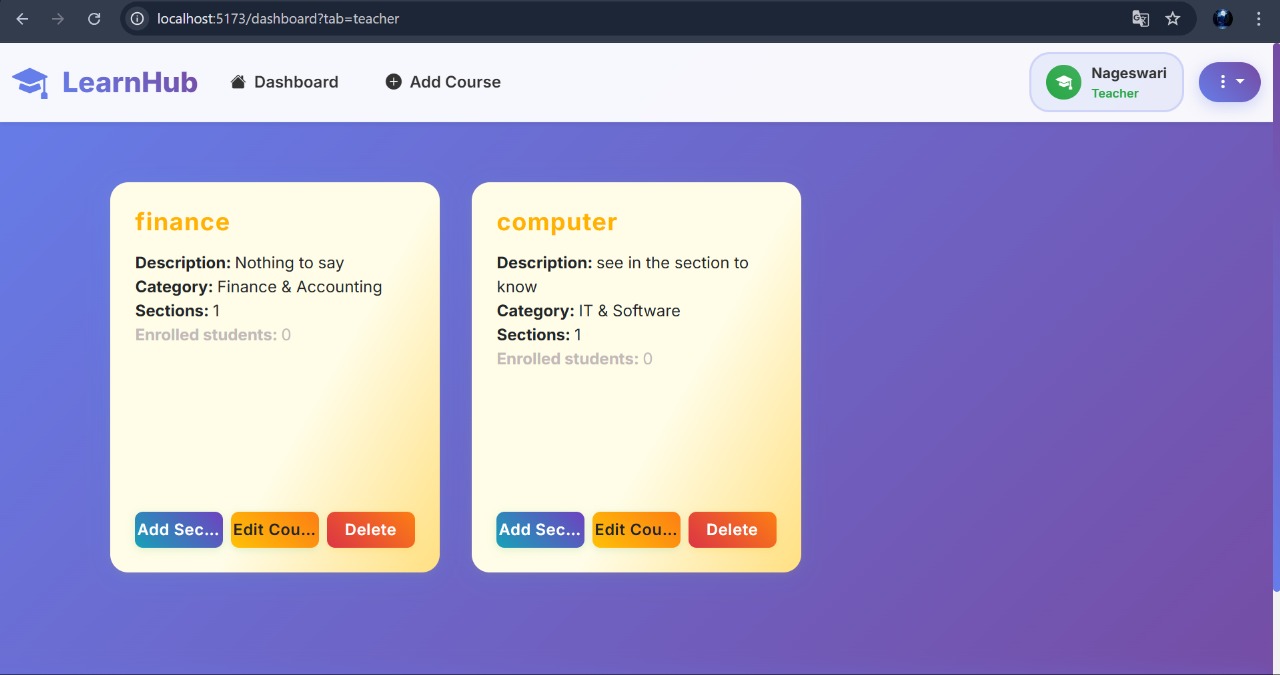


**🔳 Registration Page** 

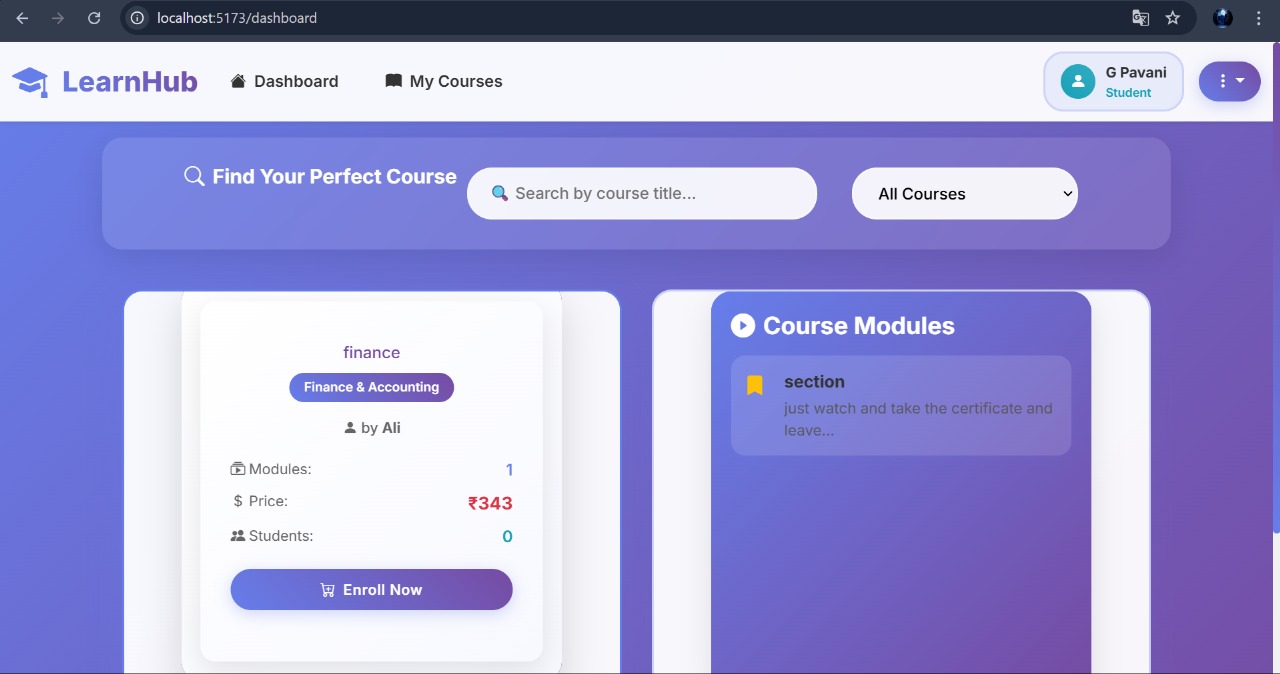
**🔳 Admin Dashboard**



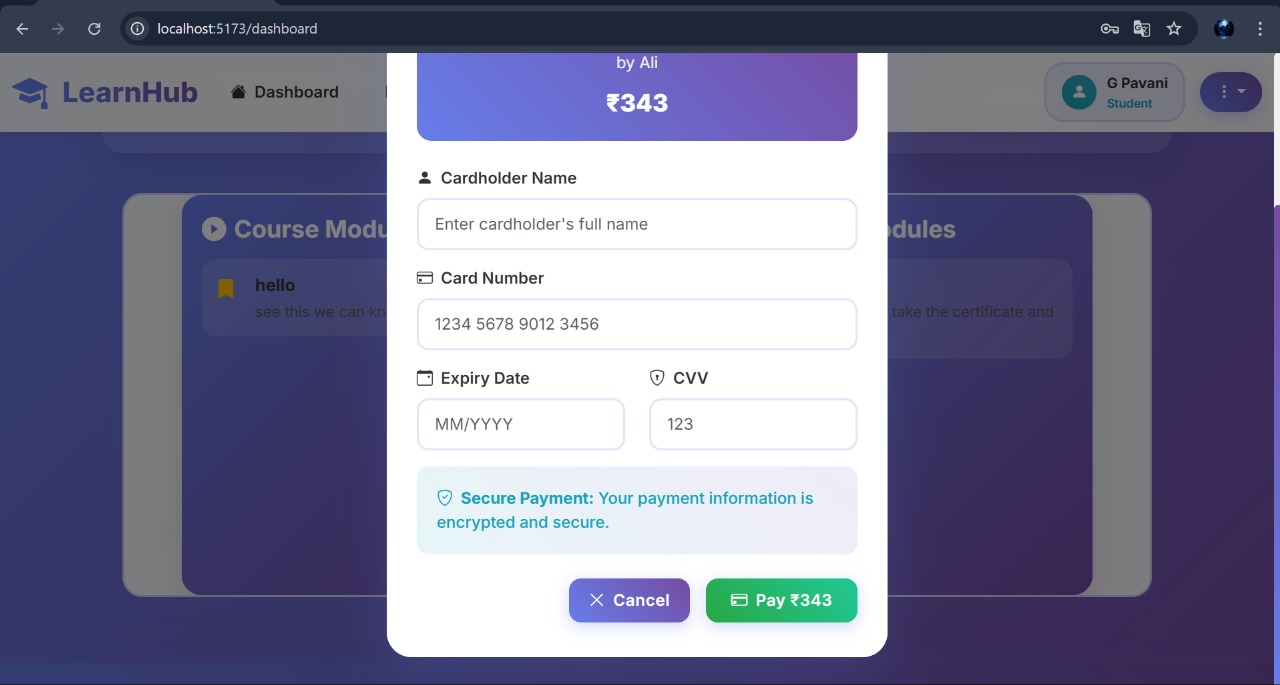
**🔳 Teacher Dashboard**



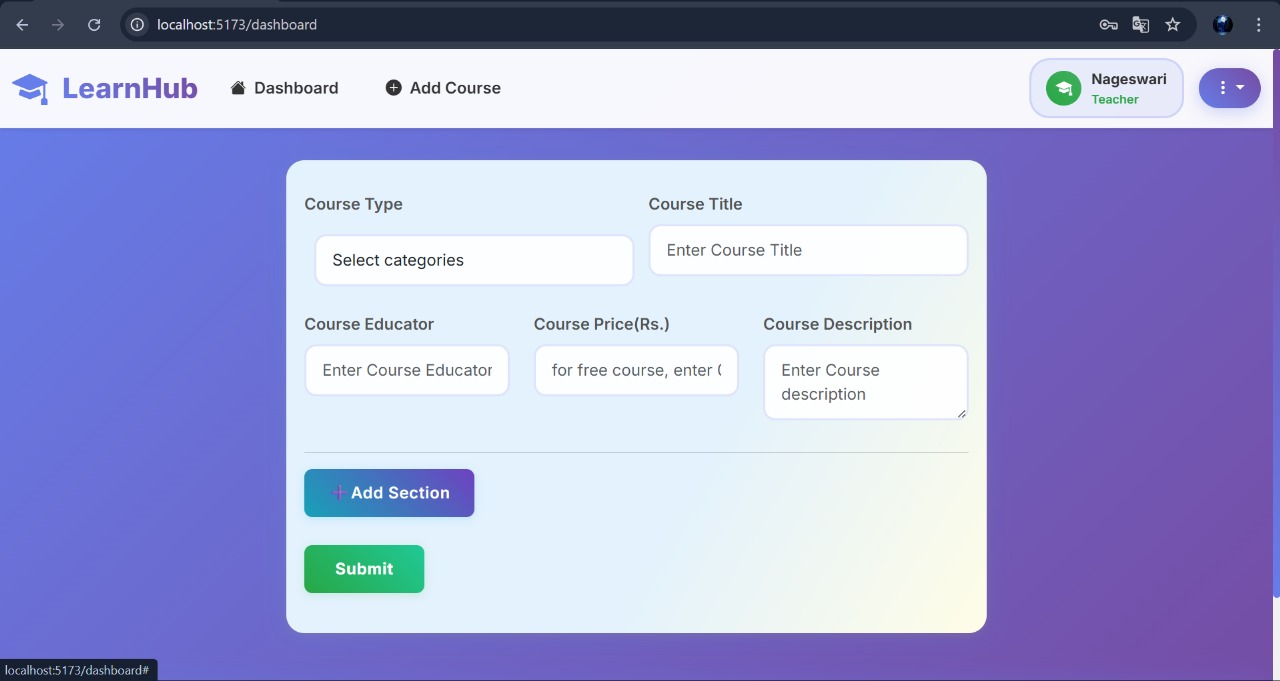
**🔳 Student Dashboard**



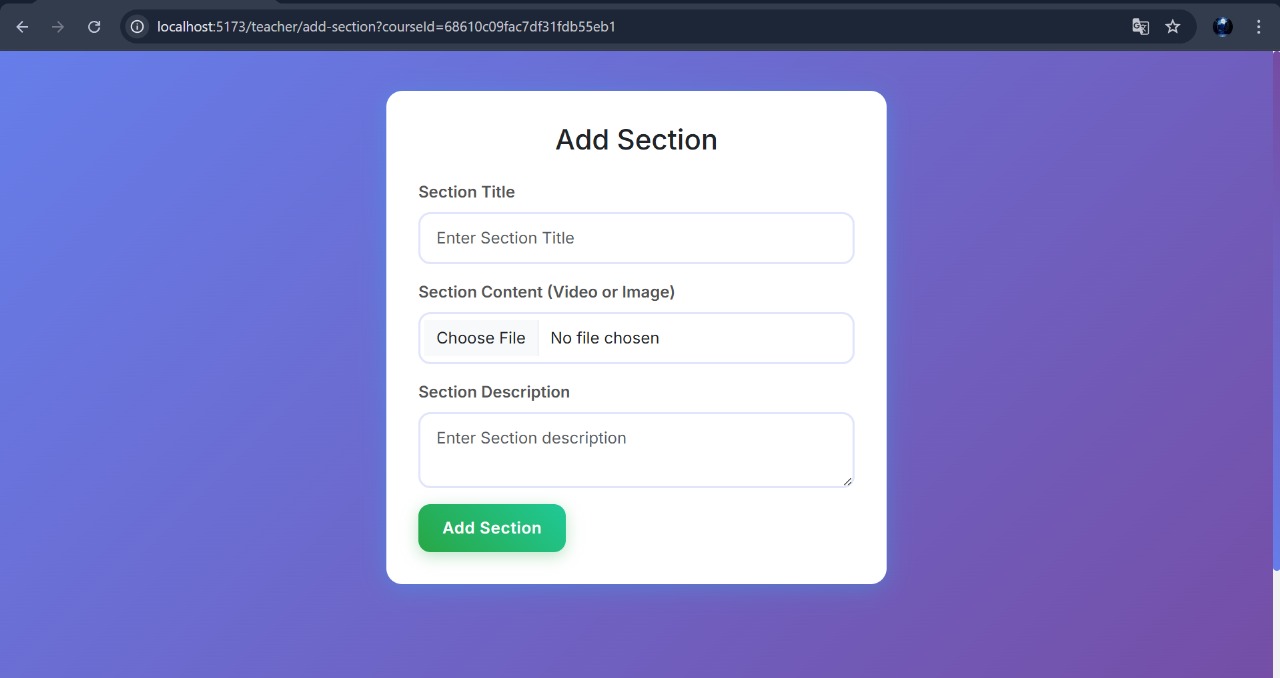
**🔳Payment Page**

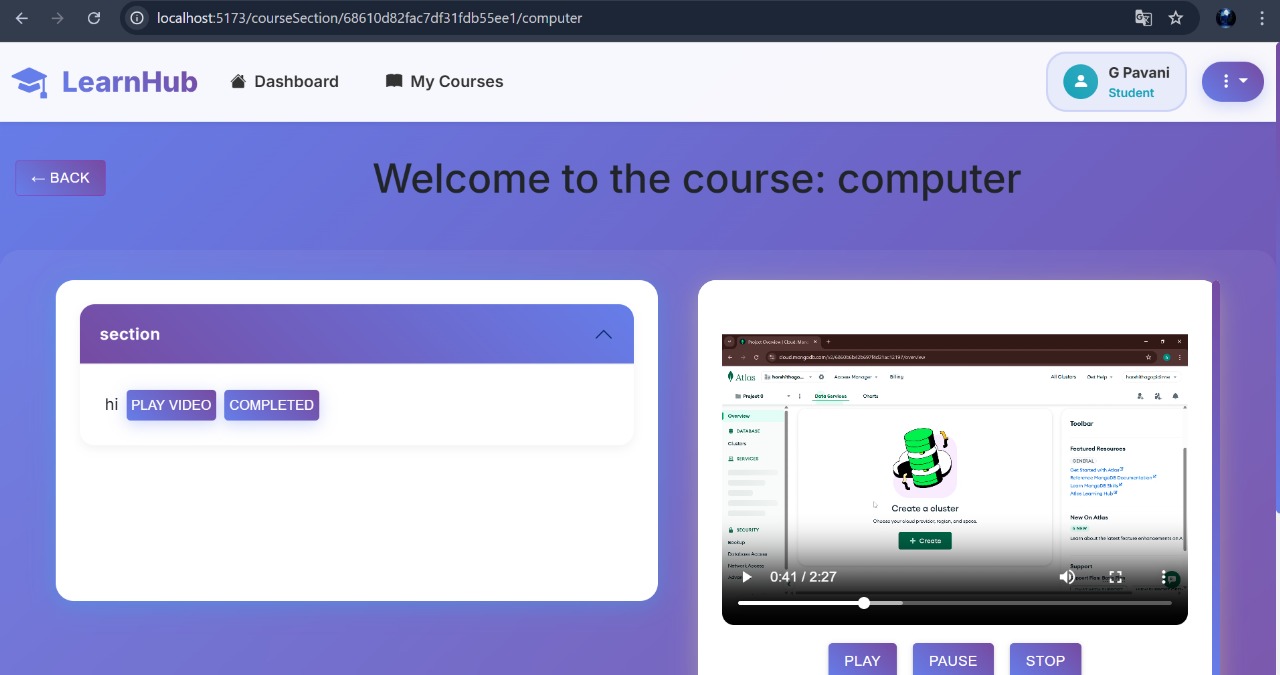


**🔳 Course Creation**



**🔳 Media Upload**

****🔳 Media Playback**

****

**10. Testing**

**The project uses a combination of manual and automated testing strategies to ensure reliability and correctness.**

* **Manual Testing:**All major user flows (registration, login, course creation, enrollment, media upload, and dashboard navigation) were manually tested across different browsers and devices to **verify** functionality and responsiveness.
* **Automated Testing:**Backend APIs can be tested using tools like Postman for endpoint validation and response checking**.**
* **Error Handling:**The application was tested for invalid inputs, unauthorized access, and edge cases to ensure proper error messages and security.
* **Performance Testing:**Key endpoints were checked for response time and stability under typical usage.

11. Demo video

• The Demo video drive link to showcase the application.

[Demo Video link](https://drive.google.com/drive/folders/1W9hfNiOxcpRJFvKxIXpKDka_A2j0sjM-?usp=sharing)

**12. Known Issues**

* Mobile responsiveness may need tuning on very small screens.
* No email verification is implemented for new user registrations.
* Error messages may not always be user-friendly or descriptive.

**13. Future Enhancements**

* Implement email verification and password reset functionality for improved security.
* Add real-time chat and discussion forums for better user interaction.
* Develop a mobile application for iOS and Android platforms.
* Integrate advanced analytics and reporting tools for teachers and admins.
* Enable support for additional media types and interacticontent.
* Improve UI/UX for enhanced accessibility and mobile responsiveness.
* Add multi-language support to reach a broader audience.
* Integrate third-party learning tools and APIs.
* Implement real-time notifications for important events and updates.
* Automate testing and continuous integration for more robust development.