LearnHub

1.Introduction

- **Project Title:** Learn Hub – E Learning Platform
- **Team ID :** LTVIP2025TMID55977
- **Team Members:**

  List team members and their roles.

  - **Chalamala Sunitha** – Login Form and Register Form
  - **Ganchi Ravitha** – Backend
  - **Margani Geethanjali**– Admin and student Dashboard, Course Player
  - **Muppana Pavan Durga Manikanteswara Rao**– Student Dashboard, Course Assignment

2.Project Overview

- **Purpose:**

  The purpose of this project, LearnHub – Online Learning Platform, is to provide a seamless, accessible, and feature-rich web application that enables educators to create and manage courses while allowing students to browse, enroll, and complete those courses with ease.

- **The platform aims to:**

  - Digitize and simplify the course delivery and learning experience.
  - Provide centralized control for teachers and admins to manage content and learners.
  - Ensure secure, scalable, and user-friendly interfaces for all user types (students, teachers, admins).
  - It supports remote education and encourages continuous learning with features like interactive video sections, course progress tracking, and auto-generated certificates upon completion.

- **Key Features & Functionalities:**

Role	Features
Student	- Register/Login - Browse Courses - Enro l in Courses - Access Video & Content Sections - Mark Courses as Completed - Download Certificate
Teacher	- Create Courses - Add Course Sections (Video/Article) - Assign Courses to Multiple Students

	- View Enrolled Students - Delete Courses
Admin	- View All Registered Users (Students & Teachers) - Remove Courses from System - View Platform-Wide Statistics
Common	- Secure JWT-based Authentication - Role-Based Access Control (RBAC) - Responsive Design & Dashboard - Protected Routes for Each Role

3.Architecture

Architecture

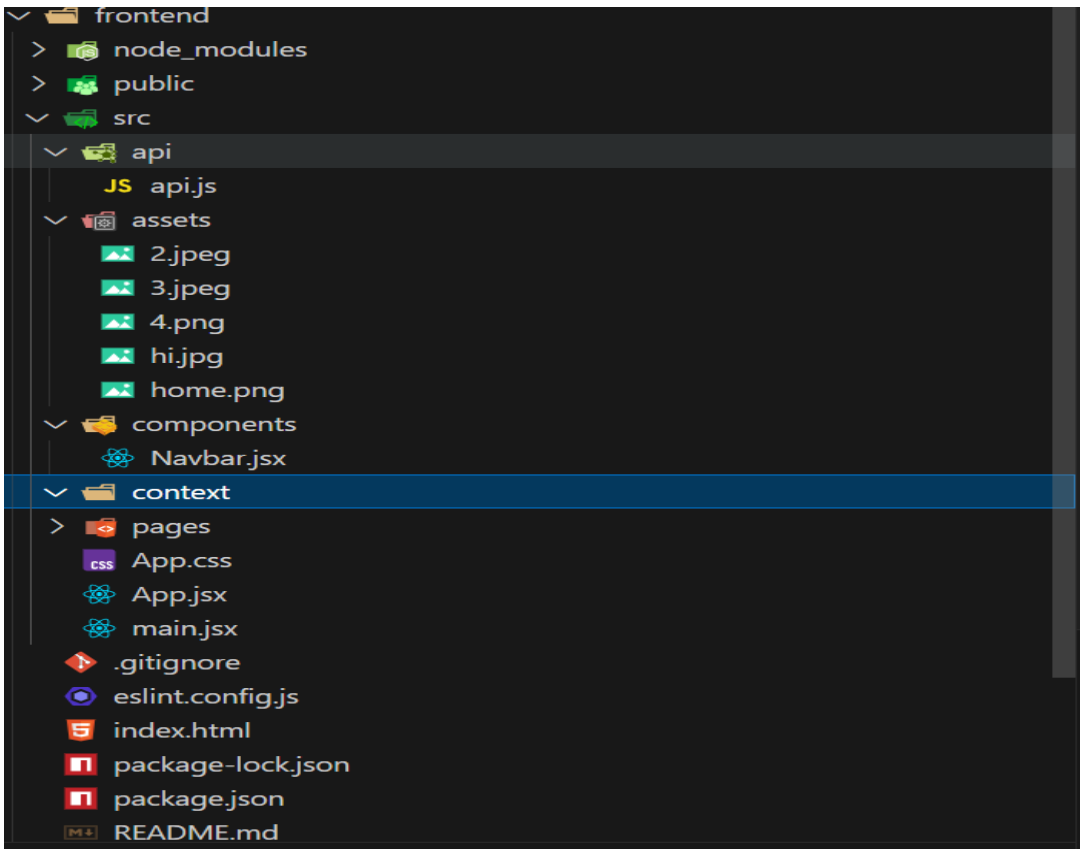
Frontend: React.js Architecture

The frontend is built using React.js with a component-based architecture. It uses react-router-dom for routing and react-bootstrap for consistent, responsive UI styling**.**

Key Components:

- **App.jsx:** Handles routing and layout rendering**.**
- **Pages:** Login, Register, Dashboard, Browse Courses, CoursePlayer, etc.
- **Components:** Reusable components like Navbar, Course Card, etc**.**
- **Private Route:** A wrapper for role-protected pages.
- **Role-based UI rendering:** Students, Teachers, and Admins see different dashboard options.

Structure:



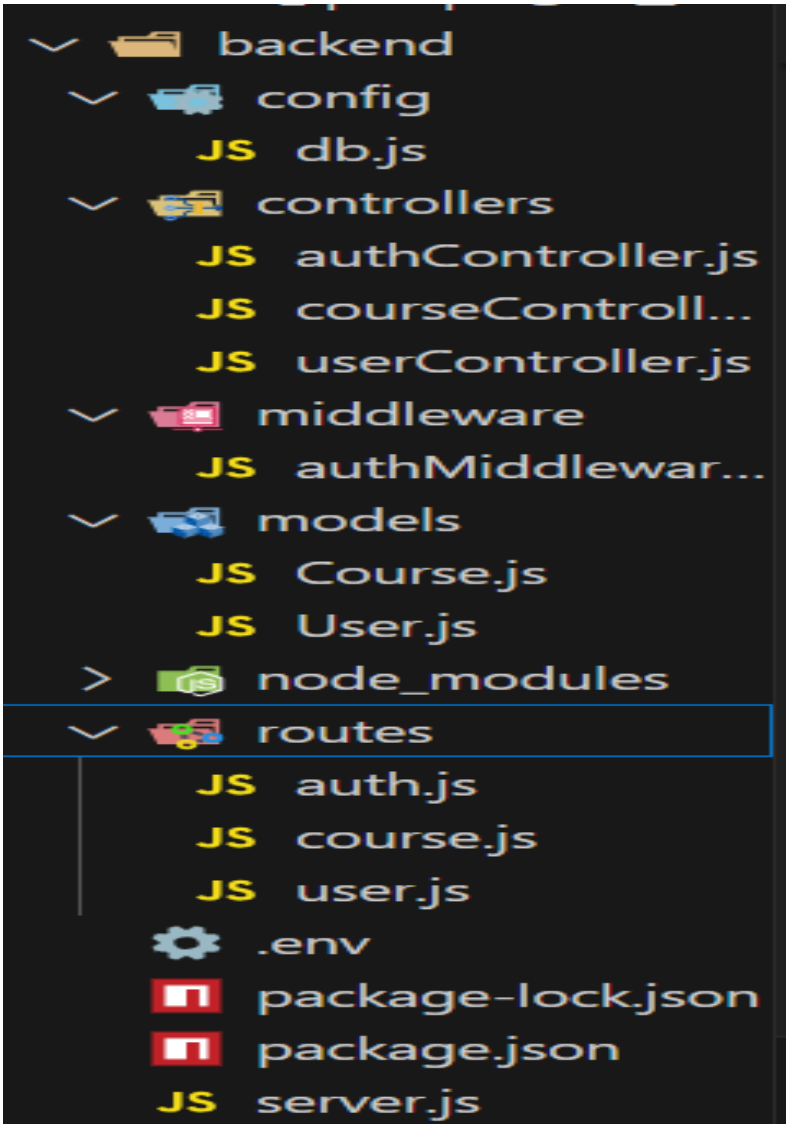
Features:

- Single Page Application (SPA)
- Dynamic rendering based on user role
- State management using local Storage and React hooks
- CSS modularization for cleaner styles

Backend: Node.js + Express.js Architecture

The backend is built using Node.js and Express.js, following RESTful API principles.

Structure:



- **Features:**
- JWT-based authentication middleware
- Role-based route protection
- RESTful endpoints for CRUD operations
- Modular controller-service-model pattern

Database: MongoDB with Mongoose

MongoDB is used for storing user and course data, with Mongoose as the ODM (Object Data Modeling) tool.

Database Features:

- Relational references between users and courses (via ObjectId)
- Query filters for search and category-based browsing
- Embedded documents for course sections
- Efficient updates using MongoDB's $push, $pull, and $set

4.Setup Instructions

Prerequisites

Before setting up the project locally, ensure you have the following software installed:

Dependency	Required Version	Description
Node.js	v16.x or higher	Runtime environment for executing JavaScript code on server side
npm / yarn	npm v8.x+	Package manager for Node.js
MongoDB	v5.x or higher	NoSQL database used to store app data
Git	Any recent	For cloning the project repository
VS Code / IDE	Recommended	For writing and running code

2. 🛠️ Installation & Setup

Step 1: Install Frontend Dependencies

- cd frontend
- npm install

Step 2: Install Backend Dependencies

- cd/backend
- npm install

Step 3: Environment Variables

Create a. env file inside the server folder with the following content:

.env

PORT=8000
MONGO_URI=mongodb+srv://olp:olp%401234@olp-server-1.wlsw0ih.mongodb.net/?retryWrites=true&w=majority&appName=OLP-SERVER-1
JWT_SECRET=supersecretkey

Step 4: Run Backend Server

npm run dev

This will run the Express backend on http://localhost:8000

Step 5: Run Frontend Client

npm start

This will start the React app on http://localhost:5731

3. Testing Login Credentials:
You can create sample users (students, teachers, admin) using Thunder Client POST /api/auth/register.

5. Folder Structure

This section describes the structure of both the React frontend (Client) and the Node.js backend (Server) of the project**.**

🖥️ **Client (Frontend - React)**

The frontend is built using React.js with routing, components, and page separation. Here's the typical folder layout:

graphql

Copy Edit

client/

├── public/ **# Static assets (HTML, favicon, etc.)**

│ └── index.html

├── src/

│ ├── assets/ **# Images, icons, background assets**

│ ├── components/ **# Reusable UI components like Navbar, Cards**

│ │ └── Navbar.jsx

│ ├── pages/ **# Page-level components (routes)**

│ │ ├── Login.jsx

│ │ ├── Register.jsx

│ │ ├── Dashboard.jsx

│ │ ├── CreateCourse.jsx

│ │ └── CoursePlayer.jsx

│ ├── api/ **# API base configuration (e.g., BASE_API)**

│ │ └── api.js

│ ├── App.jsx **# Root component with route config**

│ ├── main.jsx **# ReactDOM render entry point**

│ ├── Home.css **# Global styles for login/register background**

│ └── index.css **# Base CSS for styling**

└── package.json **# Frontend dependencies and scripts**

📌 **Highlights:**

- Components folder handles reusable elements like navigation.
- Pages handle route-specific views.
- api.js stores the base URL (BASE_API) for easy environment switching.
- Home.css styles the auth background elegantly.

⚙️ **Server (Backend - Node.js + Express)**

The backend is developed using Express.js and MongoDB (via Mongoose). Here's the structure:

server/

├── controllers/ # Business logic for routes

│ ├── authController.js

│ ├── courseController.js

│ └── userController.js

├── models/ # Mongoose schemas for MongoDB collections

│ ├── User.js

│ └── Course.js

├── routes/ # Route definitions grouped by feature

│ ├── authRoutes.js

│ ├── courseRoutes.js

│ └── userRoutes.js

├── middleware/ # Custom middleware (e.g., auth check)

│ └── authMiddleware.js

├── config/ # Config files (e.g., DB connection)

│ └── db.js

├── .env # Environment variables (port, DB URI, JWT)

├── server.js # Entry point of backend server

└── package.json # Backend dependencies and scripts

📌 **Highlights:**

- Controllers keep logic separate from route definitions.
- Middleware authenticates and protects routes.
- Models define schemas for MongoDB collections.

6. Running the Application

To run the Online Learning Platform locally, follow the steps below for both the frontend and backend:

🖥️ **Frontend (React)**

Steps:

1. Open a terminal.
2. **Navigate to the client directory:**

bash

Copy Edit

cd client

3. **Install dependencies:**

npm install

4. **Start the development server:**

bash

Copy Edit

npm start

5. **Visit: http://localhost:5731**

This will launch the frontend React app in your default browser.

⚙️ **Backend (Node.js + Express)**

Steps:

1. Open another terminal window**.**
2. **Navigate to the server directory:**

bash

CopyEdit

cd server

3. **Install backend dependencies:**

bash

CopyEdit

npm install

4. **Create a .env file (if not created) and add:**

Env

PORT=8000

MONGO_URI=mongodb+srv://olp:olp%401234@olp-server-1.wlsw0ih.mongodb.net/?retryWrites=true&w=majority&appName=OLP-SERVER-1

JWT_SECRET=supersecretkey

5. **Start the backend server:**

npm start

6. The backend will be available at**: http://localhost:8000**

✅ **Once both servers are running:**

- Frontend will interact with the backend via BASE_API defined in /client/src/api/api.js.
- Make sure MongoDB is running locally (mongod command) before starting the backend.

7. API Documentation

The backend API is built with Node.js and Express.js, and exposes RESTful endpoints for authentication, course management, and user interactions.

All endpoints use the base URL:
http://localhost:8000/api/

export const BASE_API = "http://localhost:8000/api";

Authentication Routes

Endpoint	Method	Description	Body Parameters	Response Example
/auth/register	POST	Register a new user	name, email, password, type	{ message: "User registered successfully" }
/auth/login	POST	Login and receive token	email, password	{ token: "...", user: {...} }

User Routes

Endpoint	Method	Description	Headers	Response Example
/users/students	GET	Get list of all student users	Authorization: Bearer <token>	[{ _id, name, email }, ...]

Course Routes

Endpoint	Method	Description	Body / Params	Headers
/courses/create	POST	Create a new course	C_title, C_description, C_price, C_categories	Authorization
/courses/:id/add-section	POST	Add a section to a course	title, videoUrl, externalLink	Authorization
/courses/:id	GET	Get course by ID	id in URL param	Authorization
/courses/:id/complete	POST	Mark a course as completed by a student	none	Authorization
/courses/:id/enrol	POST	Enrol the current user into a course	none	Authorization
/courses/:id/assign	POST	Assign course to students (teacher only)	{ studentIds: [] }	Authorization
/courses	GET	Get all courses with optional filters	query: search, category	-
/courses/all	GET	Get all courses (teacher/admin view)	none	Authorization
/courses/:id	DELETE	Delete a course	id in URL param	Authorization

---

🛠️ **Example Usage**

Request: Create Course

http: POST /api/courses/create

Authorization: Bearer <token>

Content-Type: application/json

```
{
  "C_title": "JavaScript Basics",
  "C_description": "Intro to JS",
  "C_categories": "Programming",
  "C_price": 0
}
```

Response:

json

CopyEdit

```
{
  "_id": "64f1023...",
  "C_title": "JavaScript Basics",
  "userID": "6488...",
  "enrolled": [],
  "sections": []
}
```

8. Authentication

Overview

The project uses token-based authentication powered by JWT (JSON Web Tokens) to securely manage user sessions across the frontend and backend. This ensures that only authenticated users can access protected routes and perform authorized actions such as creating courses, enrolling in them, or managing student data.

Authentication Flow

1. **User Login:**
    - Endpoint: POST /api/auth/login
    - On successful login, the server returns**:**

json

```
{
  "token": "jwt_token_here",
  "user": {
    "_id": "user_id",
    "name": "John Doe",
    "email": "john@example.com",
    "type": "student"
  }
}
```

2. **Token Storage:**
    - The token is securely stored on the client side using localStorage:

js

```
localStorage.setItem("token", token);
localStorage.setItem("user", JSON.stringify(user));
```

3. **Token Usage:**
    - For all protected routes, the frontend includes the token in the Authorization header:

```
Authorization: Bearer <token>
```

4. **Token Verification:**
    - The backend uses a middleware to decode and verify the JWT token before granting access to protected routes:

js

```
const jwt = require("jsonwebtoken");
const verifyToken = (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.status(401).json({ message: "Access denied" });
  try {
    const verified = jwt.verify(token, process.env.JWT_SECRET);
    req.user = verified;
    next();
  } catch {
    res.status(401).json({ message: "Invalid token" });
  }
};
```

User Roles and Access Control

The application supports three user roles:

- **Student:** Can browse and enrol in courses, view their courses, and mark them complete.
- **Teacher:** Can create, assign, and manage courses and sections.
- **Admin:** (Optional) Can view all users and delete any course.

Authorization checks are implemented using role-based conditions:

js

```
if (req.user.type !== "teacher") {
  return res.status(403).json({ message: "Access denied" });
```

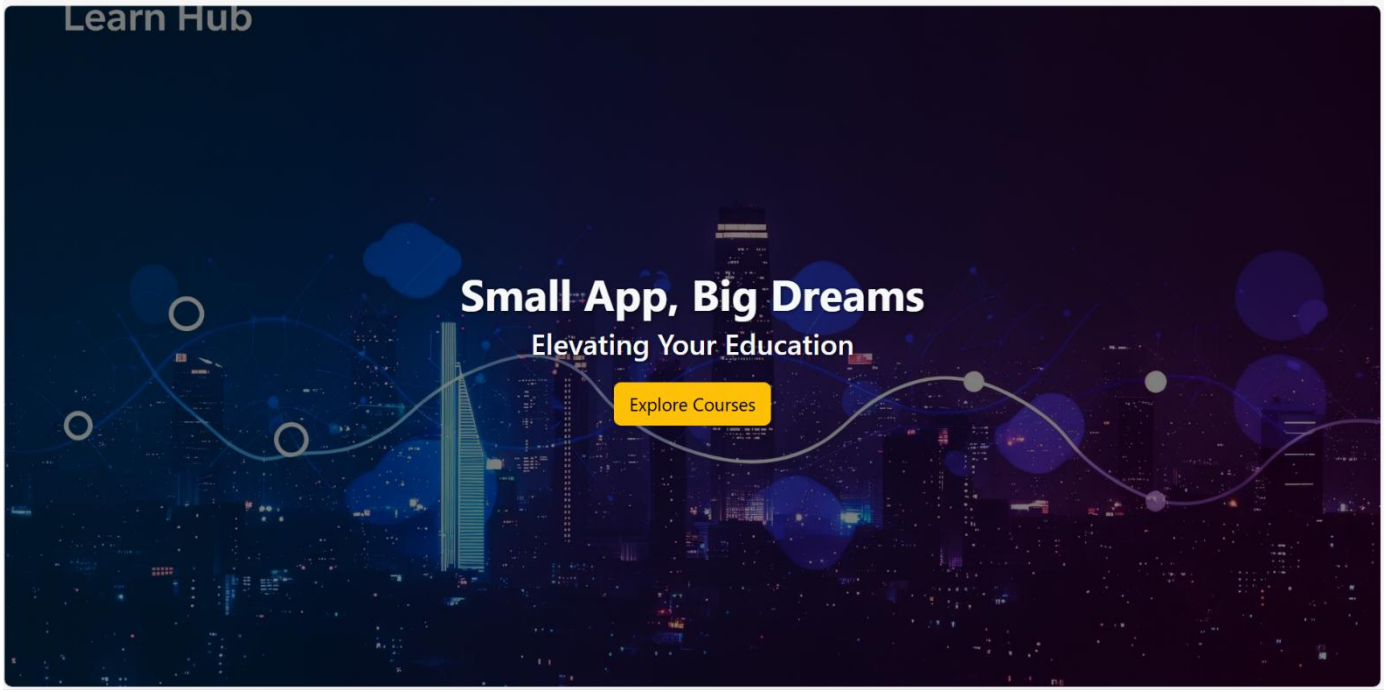
}

Security Measures

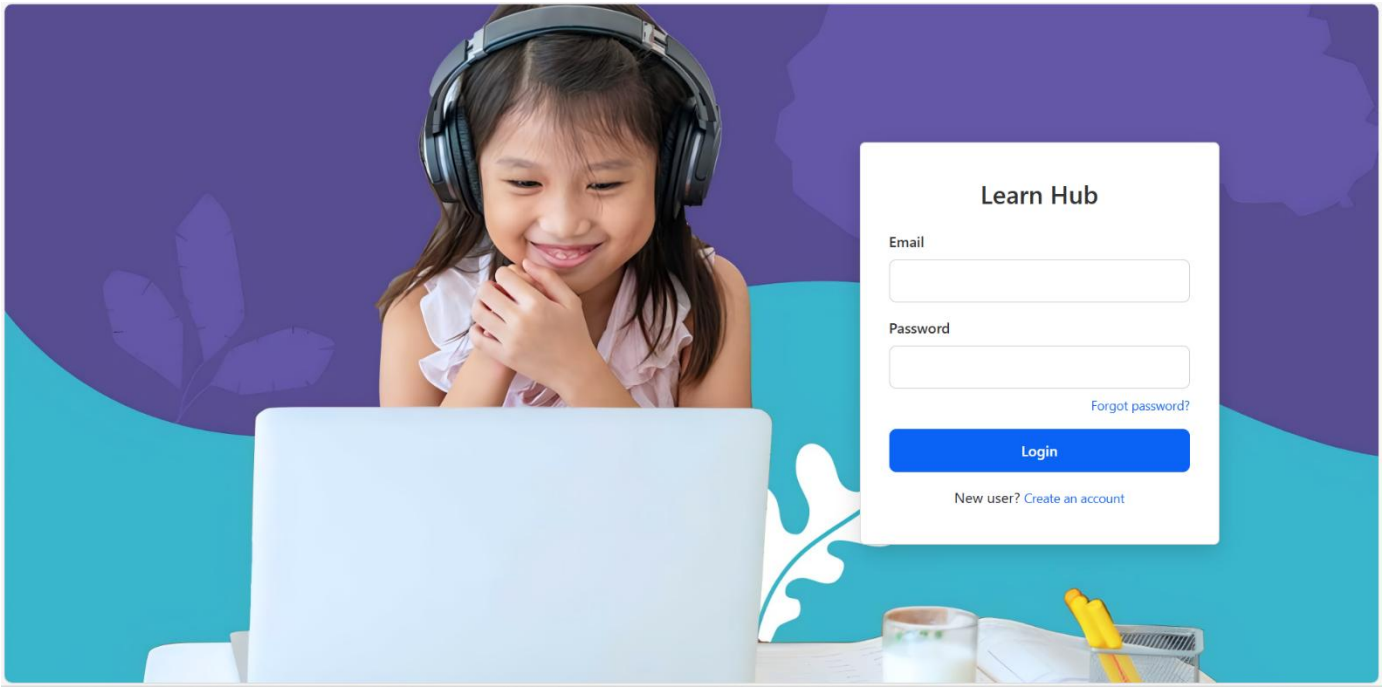
- Tokens are signed using a secret key defined in .env (JWT_SECRET).
- Sensitive actions (e.g. course deletion, student assignment) are protected via role-based checks.
- Passwords are hashed using bcrypt before storing in the database.

9. User Interface

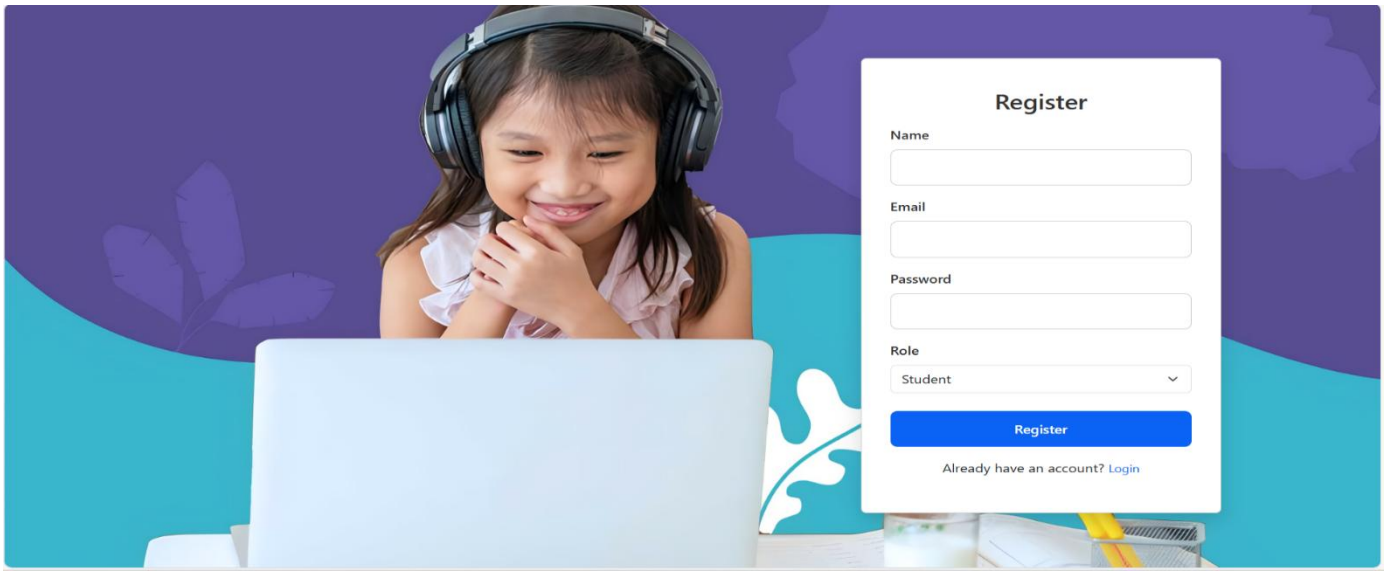
Home Page:

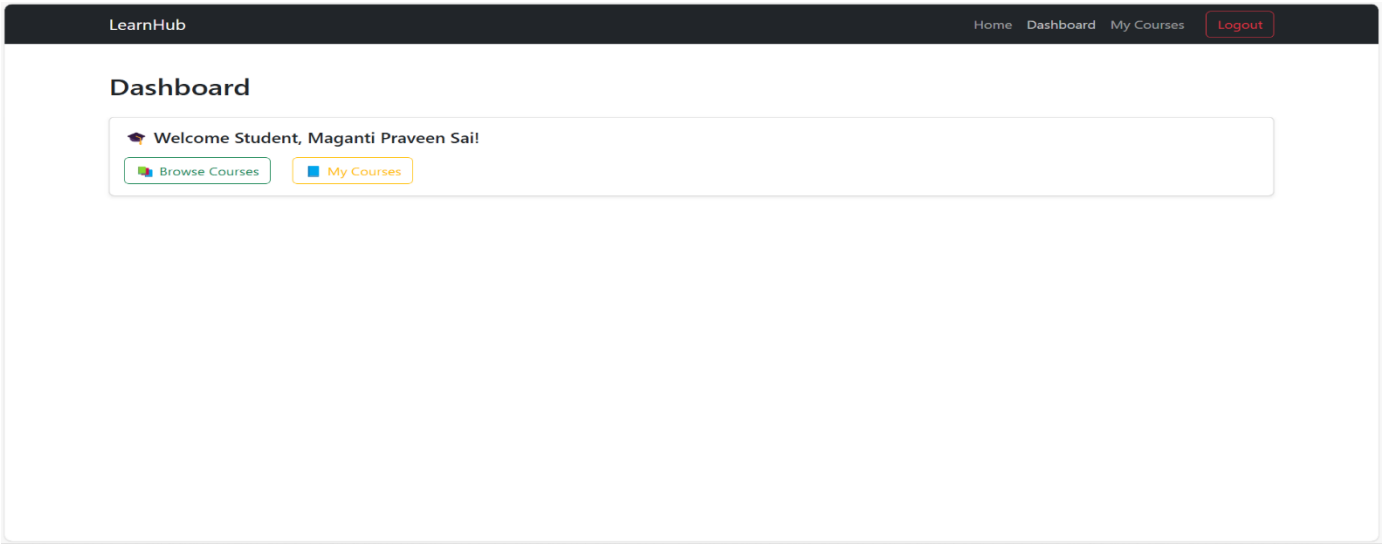


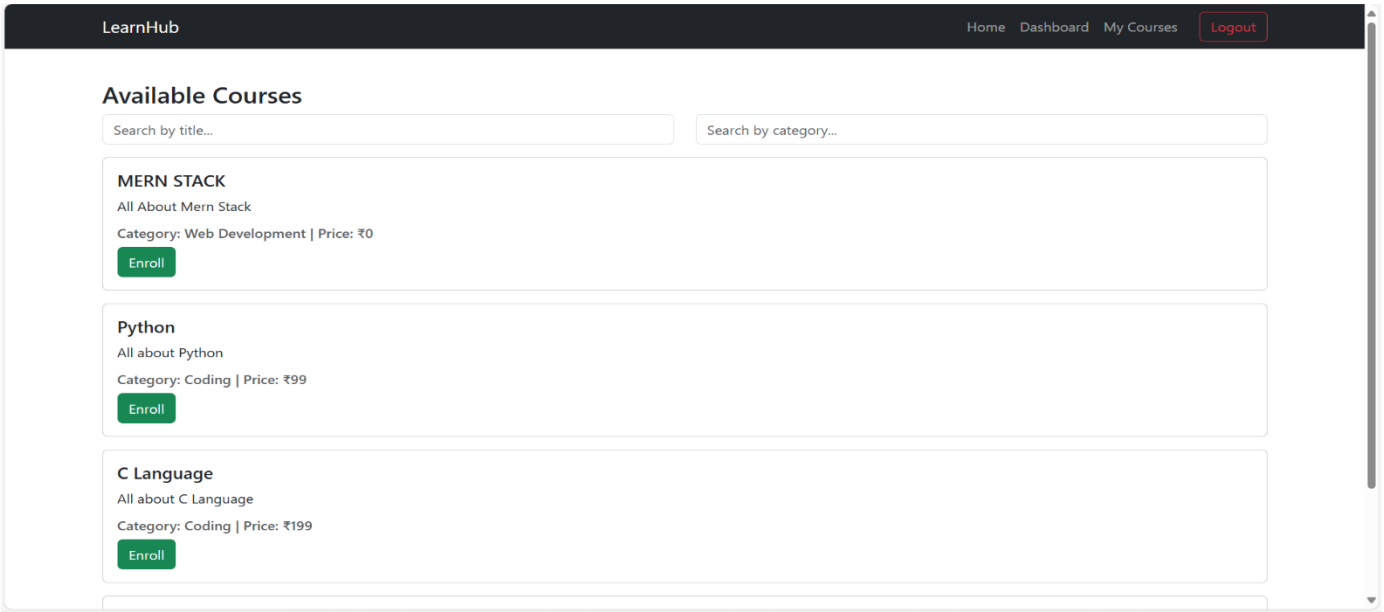
Login Page:

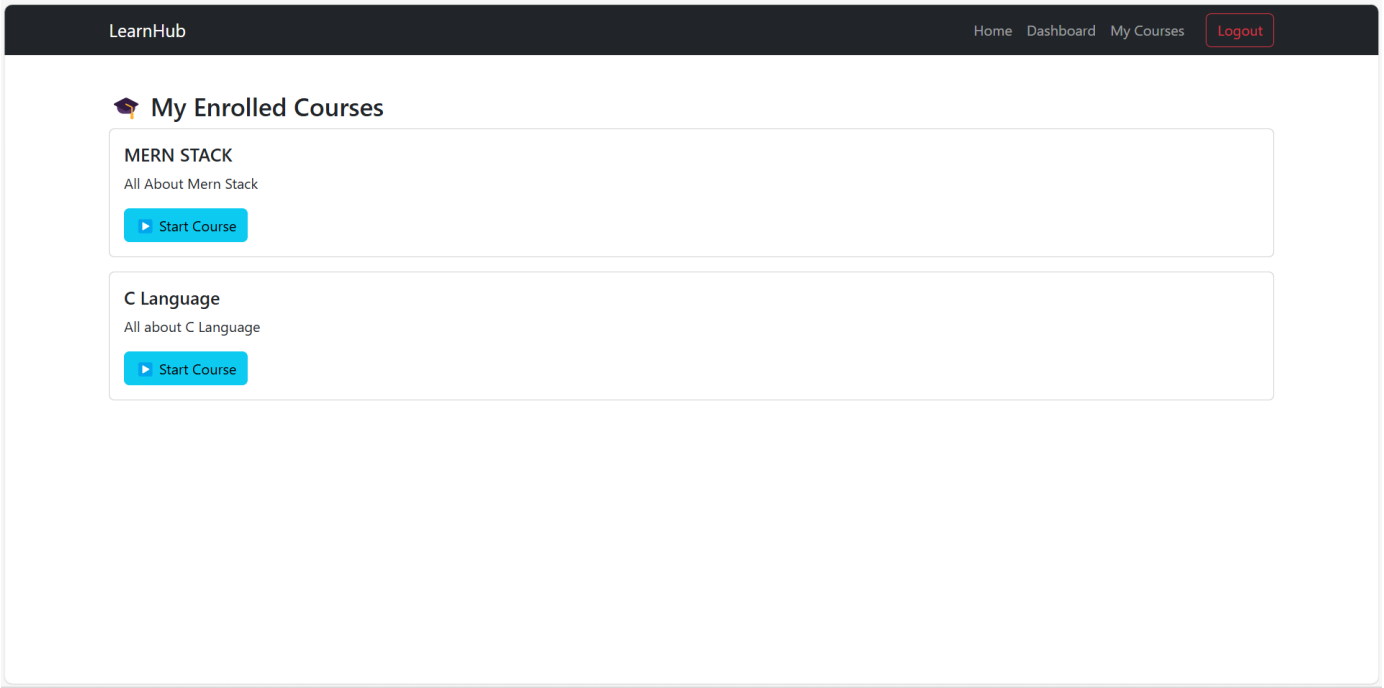


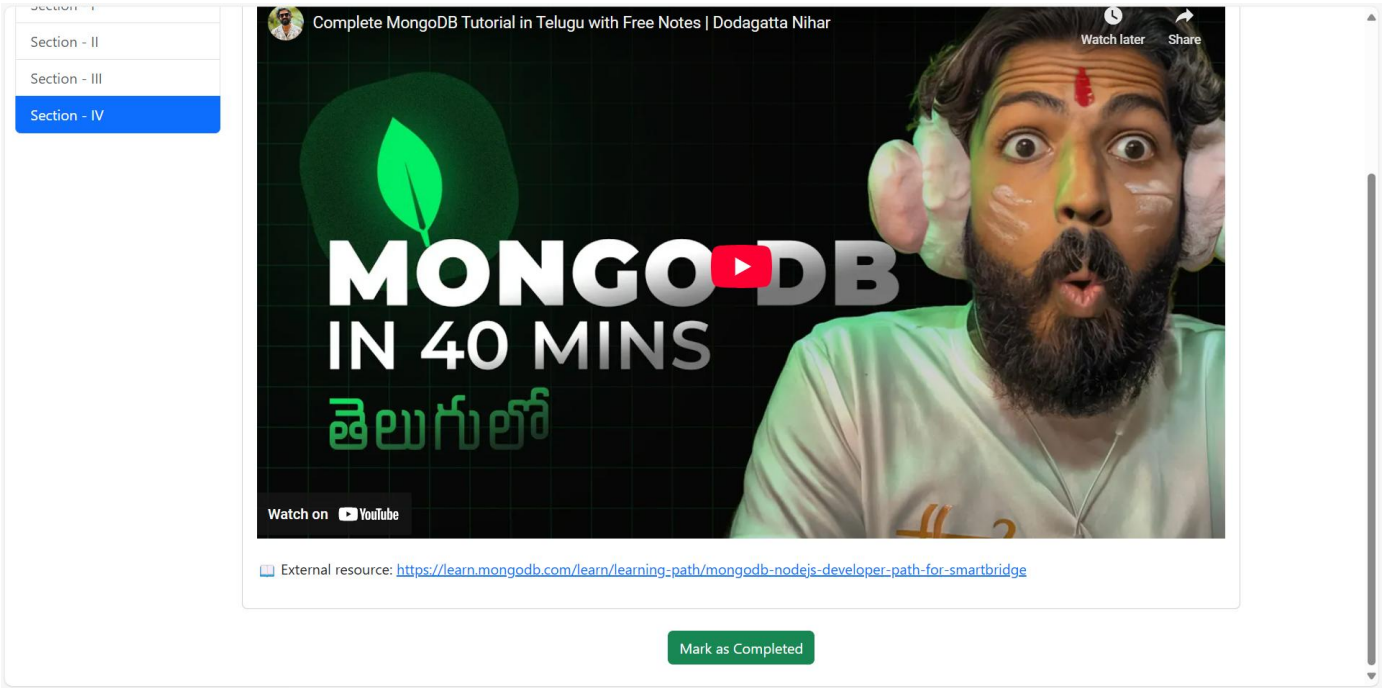
Register Page:

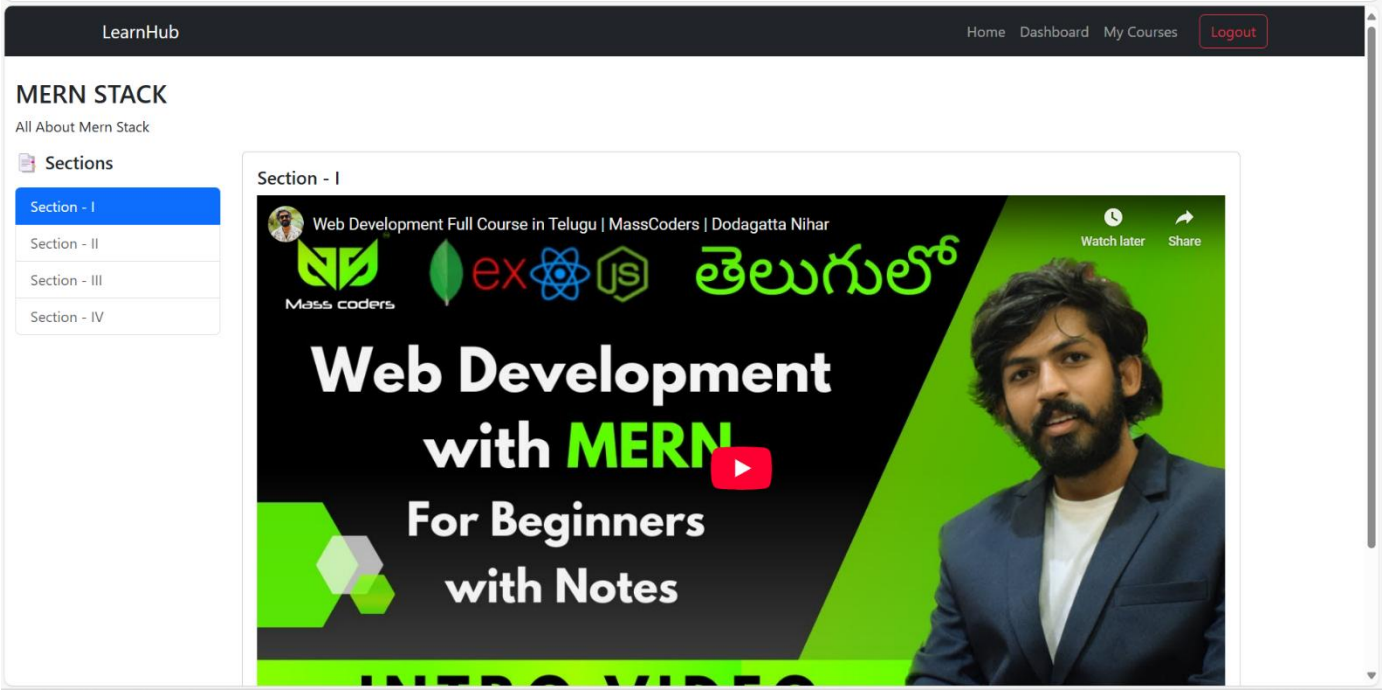



LearnHub
Home
Dashboard
My Courses
Logout
Dashboard
Welcome Student, Maganti Praveen Sai!
Browse Courses
My Courses


LearnHub
Home
Dashboard
My Courses
Logout
Available Courses
Search by title...
Search by category...
MERN STACK
All About Mern Stack
Category: Web Development | Price: ₹0
Enroll
Python
All about Python
Category: Coding | Price: ₹99
Enroll
C Language
All about C Language
Category: Coding | Price: ₹199
Enroll


LearnHub
Home
Dashboard
My Courses
Logout
My Enrolled Courses
MERN STACK
All About Mern Stack
Start Course
C Language
All about C Language
Start Course


Section - II
Section - III
Section - IV
Complete MongoDB Tutorial in Telugu with Free Notes | Dodagatta Nihar
Watch later
Share
MONGODB
IN 40 MINS
తెలుగులో
Watch on YouTube
External resource: https://learn.mongodb.com/learn/learning-path/mongodb-nodejs-developer-path-for-smartbridge
Mark as Completed


LearnHub
Home
Dashboard
My Courses
Logout
MERN STACK
All About Mern Stack
Sections
Section - I
Section - II
Section - III
Section - IV
Section - I
Web Development Full Course in Telugu | MassCoders | Dodagatta Nihar
Watch later
Share
Mass coders
ex
js
తెలుగులో
Web Development
with MERN
For Beginners
with Notes

Certificate of Completion

This certifies that

MAGANTI PRAVEEN SAI

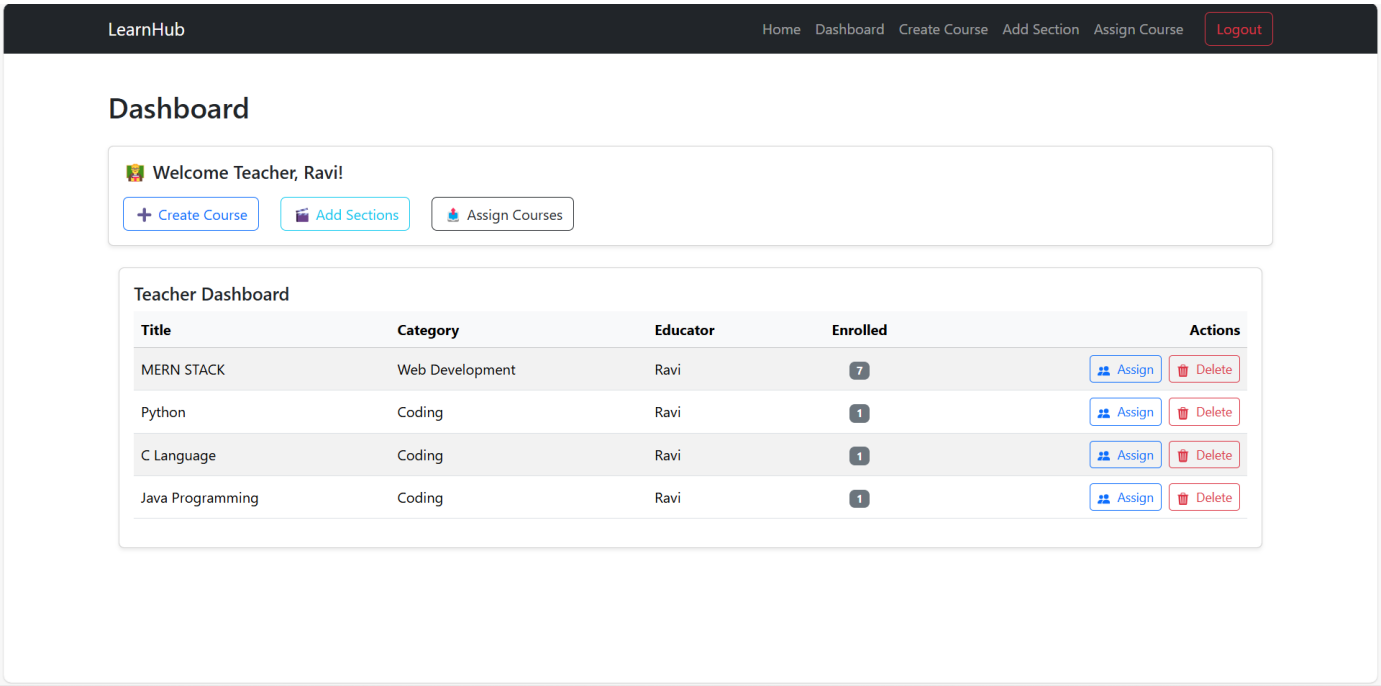
has successfully completed the course

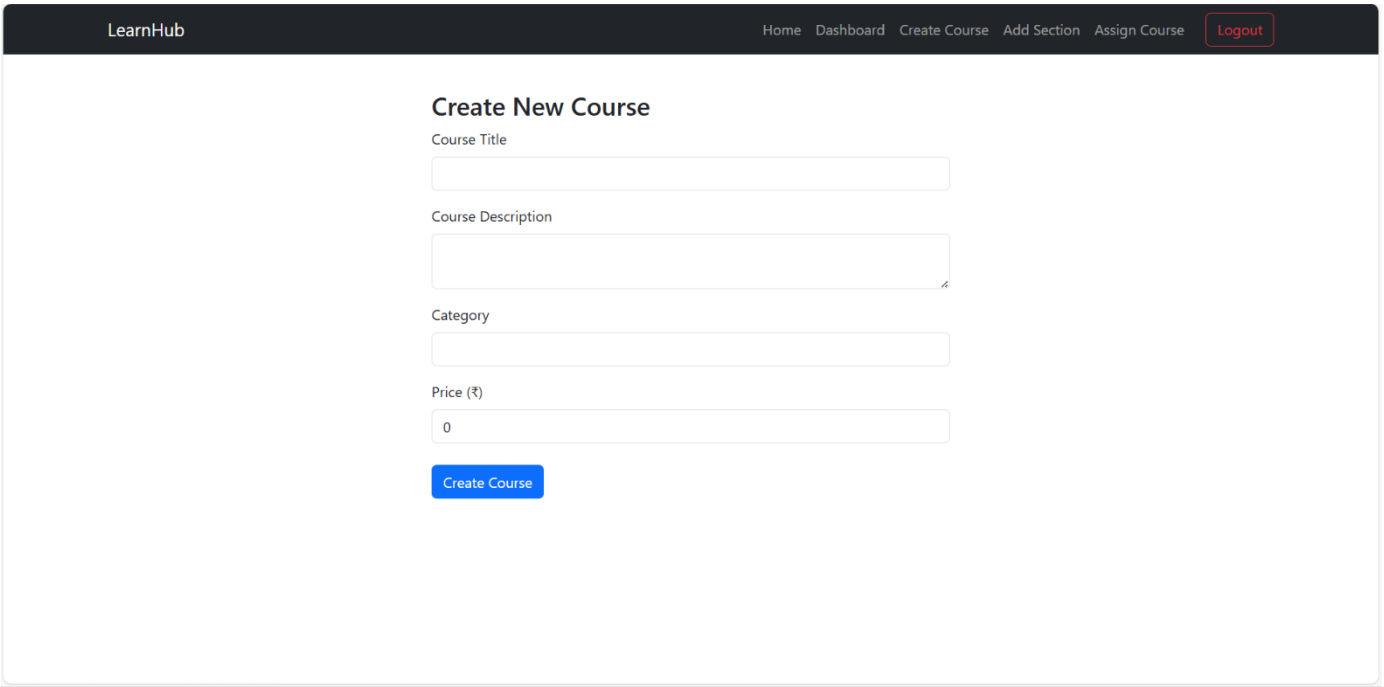
"MERN STACK"

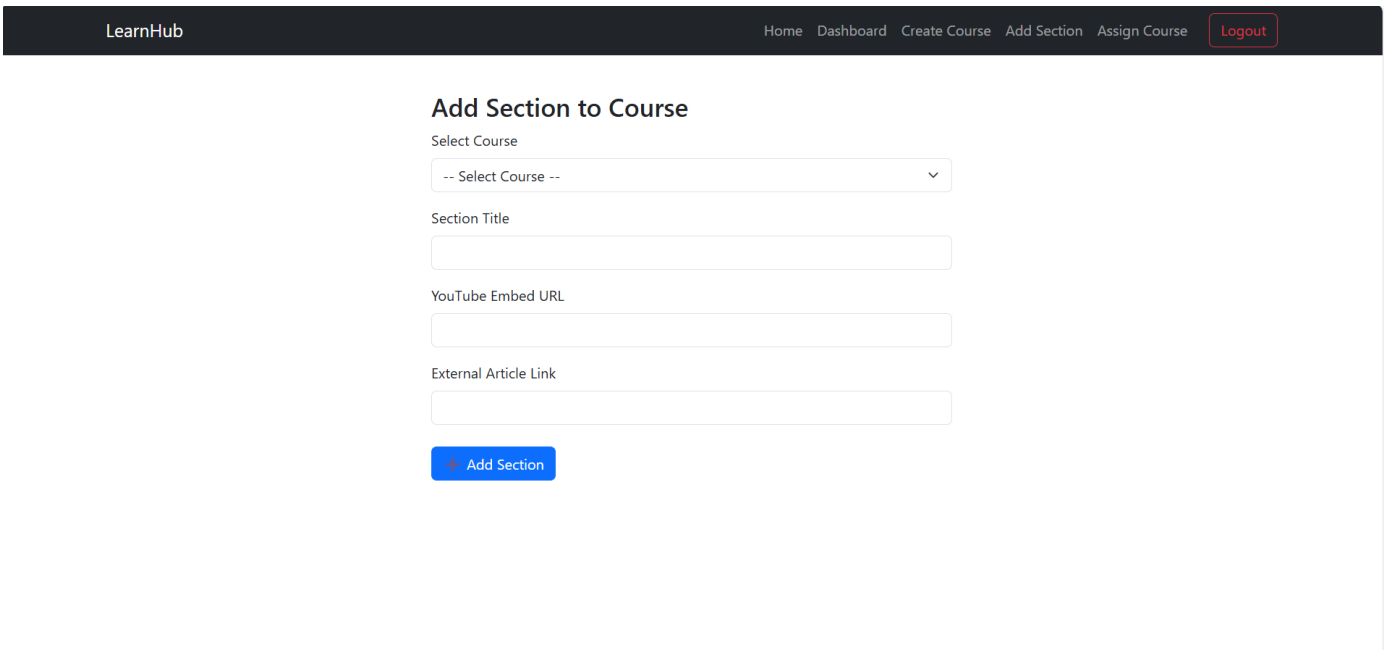
Instructor: Ravi

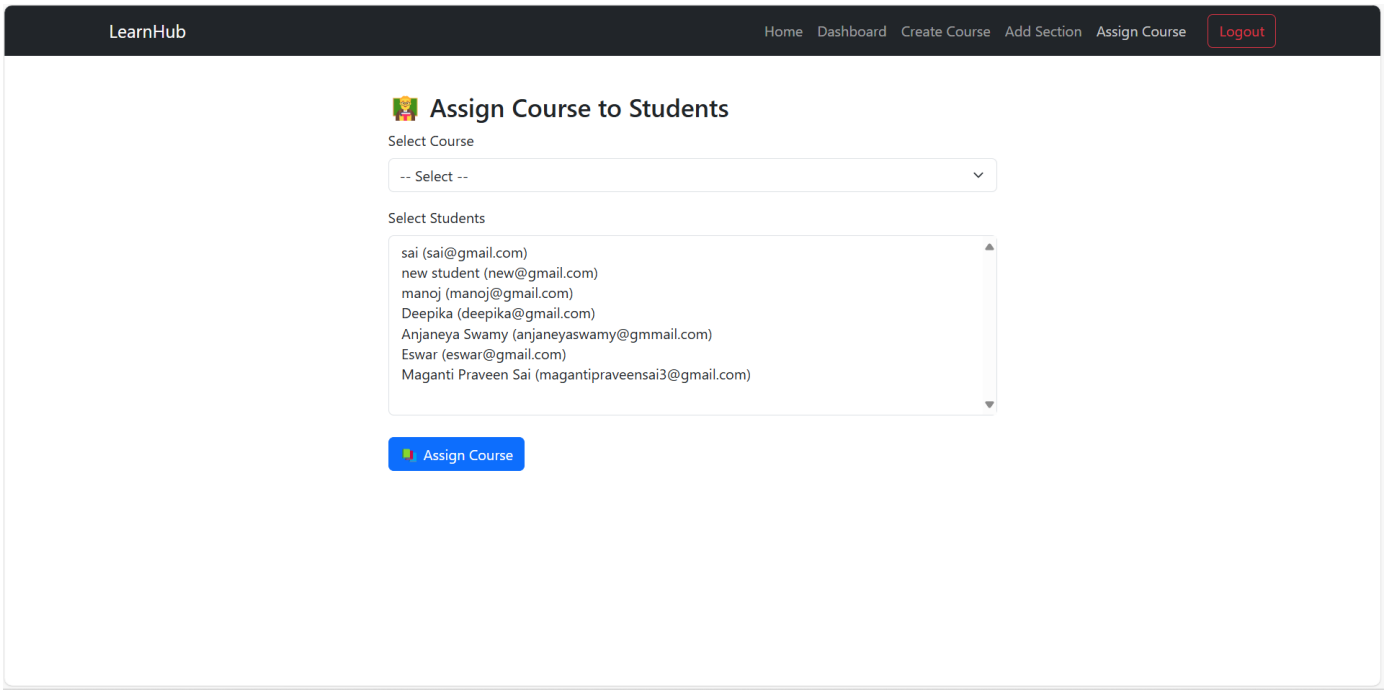
Signature

Date: 28/6/2025




LearnHub
Home
Dashboard
Create Course
Add Section
Assign Course
Logout
Create New Course
Course Title
Course Description
Category
Price (₹)
0
Create Course


LearnHub
Home
Dashboard
Create Course
Add Section
Assign Course
Logout
Add Section to Course
Select Course
-- Select Course --
Section Title
YouTube Embed URL
External Article Link
Add Section


LearnHub
Home
Dashboard
Create Course
Add Section
Assign Course
Logout
Assign Course to Students
Select Course
-- Select --
Select Students
sai (sai@gmail.com)
new student (new@gmail.com)
manoj (manoj@gmail.com)
Deepika (deepika@gmail.com)
Anjaneya Swamy (anjaneyaswamy@gmmail.com)
Eswar (eswar@gmail.com)
Maganti Praveen Sai (magantipraveensai3@gmail.com)
Assign Course

10.Testing

Testing ensures that the application functions correctly and meets requirements. This project follows a **manual and functional testing strategy** during development using tools like **Thunder Client**, **Postman**, and **browser-based testing** to validate UI and API interactions.

Testing Tools Used

Tool	Purpose
Thunder Client	API testing directly within VS Code. Fast validation of endpoints, headers, tokens, and responses.
Postman	Advanced API testing, collection creation, and request automation.
Browser DevTools	UI testing, inspecting DOM, checking console logs, network responses, and performance.
React Developer Tools	Debug React components, inspect props, state, and routing.

Testing Strategy

Type	Description
Unit Testing (Manual)	Each individual API route (register, login, course CRUD) was tested for expected input/output and error scenarios using Thunder Client.
Integration Testing	Simulated real-world flows like login → dashboard → create course → assign course → course player.
Functional Testing	Verified that all user roles (student, teacher) experience the app according to their permissions.
UI/UX Testing	Manually tested responsiveness, form validations, and navigation flows on desktop screens.
Token Authorization Testing	Verified that protected routes return 401 Unauthorized if token is missing or invalid.

Sample Test Cases

Scenario	Expected Result
Register with valid credentials	User is created and response is 201
Login with incorrect password	401 Unauthorized error
Create course without token	401 Unauthorized
Student trying to access /create-course	403 Forbidden
Complete course and mark as completed	Certificate download becomes available
Enroll in course already enrolled	400 Already enrolled error

11. Demo

- https://youtu.be/MQhVBS6kaYE

12. Known Issues

While the platform is functional and stable for most use cases, there are a few known issues and limitations developers or users should be aware of during testing and deployment:

Current Known Issues

Issue	Description	Status
No Password Reset Flow	"Forgot Password?" link is present but not implemented with actual password reset functionality.	❌ Not Implemented
No Role-Based Route Restriction at API Level	Though frontend restricts routes based on user type, backend APIs do not fully validate user roles.	⚠️ Partial
No Email Verification	After registration, there is no email confirmation mechanism to verify user identity.	❌ Not Implemented
Admin Registration Flow	Admin registration is not supported through the UI. Admin users must be added manually via Thunder Client or database.	✅ Intended Behavior

Mobile Responsiveness	Some UI components (e.g., forms and modals) may not be fully optimized for smaller mobile screens.	⚠️ Needs Improvement
Missing Pagination	All courses and users load at once without pagination. This can affect performance as the dataset grows.	⚠️ Enhancement Suggested
No Real File Upload for Videos	Course sections use YouTube embed links; there's no feature for uploading or streaming custom videos.	✅ By Design
No Multi-Tenant Support	All teachers can view all courses. There's no hard isolation between instructors' content.	✅ Intended Design for Simplicity

---

🔧 **Planned Fixes / Improvements**

- 🔄 Implement password reset via email (using NodeMailer / third-party service).
- 🔐 Enhance backend route protection with role-based access control (RBAC).
- 📱 Improve mobile layout responsiveness using media queries.
- 📊 Add course analytics and tracking for teachers.
- ⏳ Introduce pagination for large user/course datasets.

13. Future Enhancements

As the platform evolves, several features and improvements can be incorporated to enhance its functionality, performance, and user experience. The following list outlines potential areas of future development:

Feature Enhancements

Feature	Description
🔐 **Password Reset Functionality**	Implement full "Forgot Password" flow with secure email-based password reset using tokens and expiry time.
📧 **Email Verification**	Add email verification during user registration to ensure valid and trusted users.

📈 **Course Analytics**	Enable instructors to view engagement metrics such as views per section, completion rate, and active learners.
📝 **Quiz/Assessment Module**	Add quizzes, assignments, and automated evaluation to help students test their understanding.
📁 **Course Categories & Filters**	Provide course tagging, filters, and search enhancements for easier browsing.
📥 **Video Upload Support**	Allow educators to upload and host their own videos, optionally using a cloud storage solution like AWS S3.
🌐 **Language Support (i18n)**	Support multiple languages for a global audience using i18n frameworks.
💬 **Discussion Forum / Comments**	Introduce student-teacher discussion boards or comment sections under each section.
🛒 **Payment Integration**	Allow teachers to set prices and integrate payment gateways like Razorpay, Stripe, or PayPal.
🔎 **Advanced Search**	Enable keyword-based search across titles, descriptions, educators, and categories.
📱 **Progressive Web App (PWA)**	Convert the platform into a PWA for offline usage and mobile installation.
📄 **Certificates with Verification**	Add certificate download with unique serial number and verification link for authenticity.
📧 **Notifications**	Integrate email and/or in-app notifications for course updates, assignments, or deadlines.
🎨 **Theme Customization**	Allow users to choose between light/dark themes or customize UI preferences.
🛡️ **Admin Dashboard Enhancements**	Provide analytics, role management, and better visibility into platform usage for admins.

Technical Enhancements

Enhancement	**Benefit**
Role-Based API Security	Enforce fine-grained access control at backend for better protection.

Caching and CDN	Improve performance by integrating Redis caching and using a CDN for static assets.
Unit & Integration Testing	Add automated test coverage to ensure stable future deployments.
CI/CD Integration	Enable automated deployments using GitHub Actions, Netlify, or Vercel for smoother release cycles.
Dockerization	Package the app using Docker for easier deployment and environment consistency.