LearnHub

Your Center for Skill Enhancement

Full Stack Development with MERN

Project Documentation

Team ID: LTVIP2026TMIDS52632

Team Leader: Nagaram Chakradhar Singh

Members: Nandanavanam Nikhil | P Joseph Prasanth | Pacha Kranthi Kumar

GitHub: github.com/NCHAKRADHAR-SINGH1/learnhub-project

Date: February 2026

Table of Contents

1. Introduction

1.1 Project Title

LearnHub: Your Center for Skill Enhancement — a full-featured online learning platform (OLP) built on the MERN stack (MongoDB, Express.js, React.js, Node.js). The application enables educators to publish courses and learners to browse, enrol, watch video content, track progress, and earn verifiable digital certificates.

1.2 Team Information

Team ID	LTVIP2026TMIDS52632
Team Size	4 Members
Team Leader	Nagaram Chakradhar Singh
Member 2	Nandanavanam Nikhil
Member 3	P Joseph Prasanth
Member 4	Pacha Kranthi Kumar
Repository	github.com/NCHAKRADHAR-SINGH1/learnhub-project
Technology Stack	MongoDB · Express.js · React.js · Node.js (MERN)
Submission Date	February 2026

1.3 Team Roles & Responsibilities

#	Name	Role	Responsibility
1	Nagaram Chakradhar Singh	Team Leader / Full Stack Dev	Project architecture, backend API, authentication, deployment
2	Nandanavanam Nikhil	Frontend Developer	React UI development, routing, responsive design, payment UI
3	P Joseph Prasanth	Backend Developer	REST API design, course management, database schema, enrollment logic
4	Pacha Kranthi Kumar	Backend / QA Engineer	Certificate generation, progress tracking, testing, documentation

2. Project Overview

2.1 Purpose

LearnHub addresses the growing need for accessible, affordable, and structured online skill development. The platform bridges the gap between expert instructors who wish to monetize their knowledge and learners — students or working professionals — who seek quality self-paced courses. By providing a unified marketplace for course creation, discovery, purchase, and consumption, LearnHub democratises education and enables continuous learning from any device.

2.2 Key Features & Functionalities

User Registration & Authentication	Role-based sign-up (Student / Instructor) with JWT-secured login.
Course Management (Instructor)	Create courses with title, category (IT & Software, etc.), price, educator name, description, and multiple sections each containing a video or image asset.
Course Catalogue (Student)	Browse all published courses on a responsive card-based home page with category, section count, enrolled-student count, and pricing information.
Payment Gateway	Integrated card-based payment modal (card-holder name, card number, expiry, CVV) to enrol in paid courses. Free courses are enrolled immediately.
Progress Tracking	Section-level completion markers; once all sections are marked COMPLETED the certificate becomes available for download.
Video Playback	Embedded in-page video player on the Course Section page for watching uploaded lesson videos.
Certificate Generation	Auto-generated PDF Certificate of Completion bearing the learner's name, course title, and completion date; downloadable from the course section page.
Instructor Dashboard	View all self-published courses with stats (enrolled students) and option to delete courses.
Enrolled Courses View	Students can view all their enrolled courses via the "Enrolled Courses" nav link.

2.3 Target Audience

- Students learning technical skills (programming, IT, software development)
- Working professionals seeking career up-skilling or reskilling
- Expert instructors and educators looking to create and monetize online courses
- Organizations providing internal training programs for their staff

2.4 Technology Stack Summary

Layer	Technology	Version / Details
Frontend	React.js (Vite)	18.x — component-based SPA
UI Framework	Bootstrap + Material UI	Responsive grid & pre-built components
HTTP Client	Axios	Promise-based REST API consumption
Backend	Node.js + Express.js	20.x LTS — RESTful API server
Database	MongoDB + Mongoose	NoSQL document store with ODM
Auth	JSON Web Tokens (JWT)	Stateless token-based session management
File Storage	Multer	Multipart video/image upload handling
PDF Generation	PDFKit / pdfmake	Certificate PDF generation
Dev Tools	Vite, Nodemon, dotenv	Hot-reload, env management

3. Architecture

3.1 System Architecture Overview

LearnHub follows a classic three-tier client-server architecture deployed as two independent processes. The React SPA (served by Vite on port 5173) communicates with the Express.js REST API (port 5000) over HTTP/HTTPS. MongoDB Atlas or a local MongoDB instance serves as the persistence layer. All inter-service communication is JSON over RESTful endpoints authenticated with JWT bearer tokens.

Presentation Tier	React.js SPA — Vite Dev Server (port 5173) Bootstrap · Material UI · Axios HTTP client
Application Tier	Node.js + Express.js REST API (port 5000) JWT Auth · Multer · Mongoose ODM · Certificate Gen
Data Tier	MongoDB — Collections: users, courses, enrollments, payments, certificates

3.2 Frontend Architecture (React.js)

The frontend is a single-page application bootstrapped with **Vite + React 18**. React Router v6 handles client-side navigation between routes. Application state is managed with React's built-in **useState** and **useEffect** hooks. Bootstrap provides the responsive grid and utility classes while Material UI components are used for form elements and modals. **Axios** is the HTTP client used for all API calls, with the base URL configured via environment variables.

Key Frontend Routes

- `/` — Home — public course catalogue
- `/login` — User sign-in page
- `/register` — New user registration with role selection
- `/dashboard` — Instructor dashboard — manage courses, add new course
- `/dashboard (student)` — Student home — browse all available courses
- `/courseSection/:courseId/:sectionId` — Course player — video, progress, certificate

Component Hierarchy

- App.jsx — root component, React Router provider
- Navbar.jsx — dynamic navigation (guest / student / instructor)
- Home.jsx — course grid listing with enrol buttons
- Login.jsx / Register.jsx — auth forms
- Dashboard.jsx — instructor course management + AddCourse form
- CourseSection.jsx — video player + section list + certificate modal
- PaymentModal.jsx — card payment form
- CertificateModal.jsx — PDF certificate preview + download

3.3 Backend Architecture (Node.js + Express.js)

The backend is a RESTful API server built with **Express.js** running on **Node.js v20 LTS**. The server follows an MVC-inspired folder structure: routes define URL patterns, controllers contain business logic, and models encapsulate Mongoose schemas. **Multer** middleware handles multipart file uploads (video/image). Cross-Origin Resource Sharing (CORS) is enabled to allow requests from the React dev server. JWT tokens are verified by a custom auth middleware on all protected routes.

Middleware Pipeline

- cors() — allows cross-origin requests from localhost:5173 / production domain
- express.json() — parses JSON request bodies
- express.urlencoded() — parses URL-encoded form data
- multer({ storage }) — handles file upload to /uploads directory
- verifyToken (custom) — validates JWT Bearer token on protected routes

3.4 Database Architecture (MongoDB)

MongoDB is used as the primary database with **Mongoose** as the Object Document Mapper. The schema design uses a combination of embedded sub-documents (course sections within a course document) and references (user ID in enrollment records) for optimised query performance.

3.4.1 Users Collection

Field	Type	Description	Required
_id	ObjectId	Auto-generated unique identifier	Auto
name	String	Full name of the user	Yes
email	String	Unique email address (index)	Yes
password	String	Bcrypt-hashed password	Yes
role	String	Enum: "student" | "instructor"	Yes
createdAt	Date	Account creation timestamp	Auto

3.4.2 Courses Collection

Field	Type	Description	Required
_id	ObjectId	Auto-generated unique identifier	Auto
title	String	Course title	Yes
category	String	e.g., "IT & Software"	Yes
educator	String	Instructor name	Yes
price	Number	0 for free courses, else Rs. amount	Yes
description	String	Course description text	Yes
sections	[Section]	Embedded array of section sub-documents	Yes
createdBy	ObjectId	Ref → users._id (instructor)	Yes
enrolledStudents	Number	Count of enrolled learners	Auto

Section Sub-document Schema

Field	Type	Description	Required
_id	ObjectId	Auto-generated	Auto
title	String	Section / module title	Yes
description	String	Section description	No
contentUrl	String	Server path to video or image file	Yes
contentType	String	Enum: "video" | "image"	Auto

3.4.3 Enrollments Collection

Field	Type	Description	Required
_id	ObjectId	Auto-generated	Auto
userId	ObjectId	Ref → users._id	Yes
courseId	ObjectId	Ref → courses._id	Yes
completedSections	[ObjectId]	Array of completed section IDs	No
certificateIssued	Boolean	True after all sections complete	No
enrolledAt	Date	Enrollment timestamp	Auto

3.4.4 Payments Collection

Field	Type	Description	Required
_id	ObjectId	Auto-generated	Auto
userId	ObjectId	Ref → users._id	Yes
courseId	ObjectId	Ref → courses._id	Yes
amount	Number	Amount paid in Rs.	Yes
cardHolderName	String	Name on card	Yes
paidAt	Date	Payment timestamp	Auto

4. Setup Instructions

4.1 Prerequisites

- **Node.js** — v18.x or v20.x LTS (nodejs.org)
- **npm** — v9+ (Bundled with Node.js)
- **MongoDB** — v6.x Community or Atlas cloud (mongodb.com)
- **Git** — Any recent version (git-scm.com)
- **VS Code (recommended)** — Any version (code.visualstudio.com)

4.2 Clone the Repository

```
# Clone the repository
git clone https://github.com/NCHAKRADHAR-SINGH1/learnhub-project.git
cd learnhub-project
```

4.3 Backend Installation

```
# Navigate to the server directory
cd server

# Install dependencies
npm install

# Dependencies installed:
# express, mongoose, cors, dotenv, jsonwebtoken,
# bcryptjs, multer, pdfkit, nodemon
```

4.4 Environment Variables (Backend)

Create a `.env` file in the `server/` directory:

```
# server/.env
PORT=5000
MONGO_URI=mongodb://localhost:27017/learnhub
# or for MongoDB Atlas:
# MONGO_URI=mongodb+srv://:@cluster0.mongodb.net/learnhub
JWT_SECRET=your_super_secret_jwt_key_here
JWT_EXPIRES_IN=7d
UPLOAD_DIR=./uploads
```

4.5 Frontend Installation

```
# Navigate to client directory
cd ../client

# Install dependencies
npm install

# Dependencies installed:
# react, react-dom, react-router-dom, axios,
# bootstrap, @mui/material, @emotion/react
```

4.6 Environment Variables (Frontend)

Create a `.env` file in the `client/` directory:

```
# client/.env
VITE_API_BASE_URL=http://localhost:5000/api
```

Note: Never commit `.env` files to version control. Both directories have `.gitignore` entries for `.env`.

5. Folder Structure

5.1 Repository Root

```
learnhub-project/
■■■ client/ # React frontend (Vite)
■■■ server/ # Node.js + Express backend
■■■ README.md
■■■ .gitignore
```

5.2 Client (Frontend) Structure

```
client/
■■■ public/
■ ■■■ vite.svg
■■■ src/
■ ■■■ assets/ # Static images, icons
■ ■■■ components/ # Reusable UI components
■ ■ ■■■ Navbar.jsx
■ ■ ■■■ CourseCard.jsx
■ ■ ■■■ PaymentModal.jsx
■ ■ ■■■ CertificateModal.jsx
■ ■■■ pages/ # Route-level page components
■ ■ ■■■ Home.jsx
■ ■ ■■■ Login.jsx
■ ■ ■■■ Register.jsx
■ ■ ■■■ Dashboard.jsx
■ ■ ■■■ CourseSection.jsx
■ ■■■ services/ # Axios API service layer
■ ■ ■■■ api.js
■ ■■■ context/ # React Context providers
■ ■ ■■■ AuthContext.jsx
■ ■■■ App.jsx # Root component + Router
■ ■■■ main.jsx # Vite entry point
■■■ .env
■■■ index.html
■■■ package.json
■■■ vite.config.js
```

5.3 Server (Backend) Structure

```
server/
■■■ config/
■ ■■■ db.js # MongoDB connection setup
■■■ controllers/
■ ■■■ authController.js # register, login
■ ■■■ courseController.js # CRUD for courses
■ ■■■ enrollController.js # enroll, get enrolled
■ ■■■ paymentController.js # process payment
■ ■■■ progressController.js # mark section complete
■ ■■■ certificateController.js # generate & download PDF
■■■ middleware/
■ ■■■ authMiddleware.js # JWT token verification
■ ■■■ uploadMiddleware.js # Multer configuration
■■■ models/
■ ■■■ User.js
■ ■■■ Course.js
```

```
■ ■■■ Enrollment.js
■ ■■■ Payment.js
■■■ routes/
■ ■■■ authRoutes.js
■ ■■■ courseRoutes.js
■ ■■■ enrollRoutes.js
■ ■■■ paymentRoutes.js
■ ■■■ progressRoutes.js
■ ■■■ certificateRoutes.js
■■■ uploads/ # Uploaded videos & images
■■■ .env
■■■ server.js # App entry point
■■■ package.json
```

6. Running the Application

6.1 Start MongoDB

```
# Local MongoDB (macOS / Linux)
mongod --dbpath /data/db

# Windows (run as administrator)
net start MongoDB

# Or ensure MongoDB Atlas cluster is running and MONGO_URI is set
```

6.2 Start the Backend Server

```
cd server
npm start # production
# or
npm run dev # development (nodemon — auto-restart on changes)

# Expected output:
# Server running on port 5000
# MongoDB connected successfully
```

6.3 Start the Frontend

```
cd client
npm run dev # starts Vite dev server

# Expected output:
# VITE v5.x.x ready in 300ms
# Local: http://localhost:5173/
```

6.4 Access the Application

Open your browser and navigate to:

- `http://localhost:5173` — React frontend (main app)
- `http://localhost:5000/api` — REST API base URL
- `http://localhost:5000/api/courses` — Example: get all courses

6.5 Build for Production

```
# Build React frontend
cd client
npm run build
# Output: client/dist/ (static files)

# Serve static files from Express (add to server.js):
# app.use(express.static(path.join(__dirname, '../client/dist')))
```

7. API Documentation

All API endpoints are prefixed with `/api`. Protected routes require an `Authorization: Bearer <token>` header. Responses are JSON. HTTP status codes follow REST conventions (200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Server Error).

7.1 Authentication Endpoints

Method	Endpoint	Description	Auth Required
POST	/api/auth/register	Register a new user (student or instructor)	No
POST	/api/auth/login	Login and receive a JWT token	No
GET	/api/auth/me	Get the currently authenticated user profile	Yes

POST /api/auth/register — Request Body

```
{
"name": "Gopi",
"email": "gopi@gmail.com",
"password": "secret123",
"role": "student" // "student" | "instructor"
}
```

POST /api/auth/register — Response (201)

```
{
"message": "User registered successfully",
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
"user": { "_id": "...", "name": "Gopi", "email": "gopi@gmail.com", "role": "student" }
}
```

POST /api/auth/login — Request Body

```
{
"email": "gopi@gmail.com",
"password": "secret123"
}
```

POST /api/auth/login — Response (200)

```
{
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
"user": { "_id": "...", "name": "Gopi", "role": "student" }
}
```

7.2 Course Endpoints

Method	Endpoint	Description	Auth Required
GET	/api/courses	Fetch all published courses	No
GET	/api/courses/:id	Get single course by ID	No
POST	/api/courses	Create a new course (instructor)	Yes
DELETE	/api/courses/:id	Delete a course (owner only)	Yes

Method	Endpoint	Description	Auth Required
GET	/api/courses/instructor	Get all courses by logged-in instructor	Yes

POST /api/courses — Request (multipart/form-data)

```
Form fields:
title: "C++"
category: "IT & Software"
educator: "radha"
price: 1800
description: "Comprehensive C++ course"
sections[0][title]: "Introduction to C++"
sections[0][description]: "Basics of C++"
sections[0][content]:
```

GET /api/courses — Response (200)

```
[
{
"_id": "69984a2c8446685f9a945100",
"title": "c++",
"category": "IT & Software",
"educator": "radha",
"price": 1800,
"description": "c++",
"sections": [{ "_id": "...", "title": "c++", "contentUrl": "/uploads/..." }],
"enrolledStudents": 0,
"createdBy": "...",
"createdAt": "2026-02-20T..."
},
...
]
```

7.3 Enrollment Endpoints

Method	Endpoint	Description	Auth Required
POST	/api/enroll	Enrol in a free course	Yes
GET	/api/enroll/my	Get all enrolled courses for current user	Yes
GET	/api/enroll/:courseId	Check if enrolled in a specific course	Yes

7.4 Payment Endpoints

Method	Endpoint	Description	Auth Required
POST	/api/payment	Process payment and trigger enrollment	Yes

POST /api/payment — Request Body

```
{
"courseId": "69984a2c8446685f9a945100",
"amount": 1800,
"cardHolderName": "gopi",
"cardNumber": "12345512",
"expiration": "12/2025",
```

```
"cvv": "234"
}
```

POST /api/payment — Response (200)

```
{
"message": "Enroll Successfully",
"enrollmentId": "..."
}
```

7.5 Progress Endpoints

Method	Endpoint	Description	Auth Required
POST	/api/progress/complete	Mark a section as completed	Yes
GET	/api/progress/:courseId	Get completion status for a course	Yes

7.6 Certificate Endpoints

Method	Endpoint	Description	Auth Required
GET	/api/certificate/:courseId	Download PDF certificate (all sections complete)	Yes

8. Authentication

8.1 Authentication Strategy

LearnHub uses **JSON Web Token (JWT)** based stateless authentication. On successful login or registration, the server signs a JWT with the user's ID and role using a secret key stored in environment variables. The token is returned to the client and stored in `localStorage`. Subsequent API requests include the token in the `Authorization: Bearer <token>` header.

8.2 Token Generation

```
// authController.js (simplified)
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

const login = async (req, res) => {
const { email, password } = req.body;
const user = await User.findOne({ email });
if (!user) return res.status(404).json({ message: "User not found" });

const valid = await bcrypt.compare(password, user.password);
if (!valid) return res.status(401).json({ message: "Invalid credentials" });

const token = jwt.sign(
{ id: user._id, role: user.role },
process.env.JWT_SECRET,
{ expiresIn: process.env.JWT_EXPIRES_IN }
);
res.json({ token, user: { _id: user._id, name: user.name, role: user.role } });
};
```

8.3 Token Verification Middleware

```
// middleware/authMiddleware.js
const jwt = require('jsonwebtoken');

const verifyToken = (req, res, next) => {
const authHeader = req.headers.authorization;
if (!authHeader?.startsWith("Bearer "))
return res.status(401).json({ message: "No token provided" });

const token = authHeader.split(" ")[1];
try {
const decoded = jwt.verify(token, process.env.JWT_SECRET);
req.user = decoded; // { id, role }
next();
} catch (err) {
return res.status(403).json({ message: "Invalid or expired token" });
}
};
```

8.4 Role-Based Access Control

The `req.user.role` field (decoded from JWT) is used to enforce role-based access:

- Only users with role **"instructor"** can create or delete courses.

- Only users with role **"student"** can enrol in courses, make payments, mark progress, and download certificates.
- The backend validates ownership: instructors can only delete courses they created (matched by `createdBy === req.user.id`).
- Frontend navigation adapts dynamically: the Navbar shows "Add Course" and instructor dashboard links only for instructors.

8.5 Password Security

Passwords are hashed using **bcryptjs** with a salt factor of 10 before storage. Plain-text passwords are never persisted in the database.

9. User Interface

The UI is built with React, Bootstrap, and Material UI components. The application features a warm beige background (`#FFF8F0`), clean card-based layouts, and a consistent blue-purple colour scheme for interactive elements. Below are the key screens extracted from the live demo.

9.1 Landing Page — Sign In

The Login page presents a centred glassmorphism-style card over a full-page background image of a student studying. Fields include Email Address and Password. A "Sign Up" link navigates to the registration page.

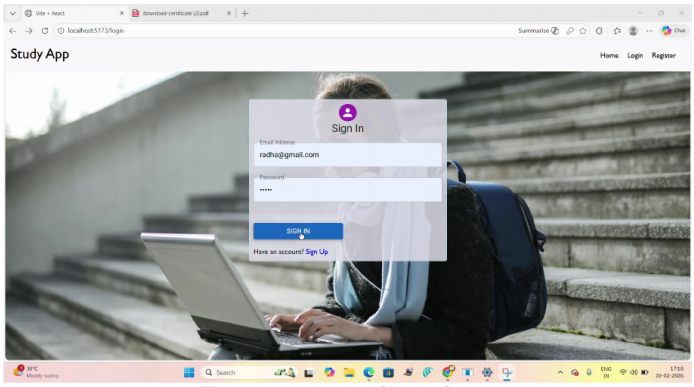

Figure 9.1 — Login / Sign In Screen

9.2 User Registration

The Registration screen collects Full Name, Email, Password, and a role selector ("Student" or "Instructor"). After successful registration the user is redirected to the login page.

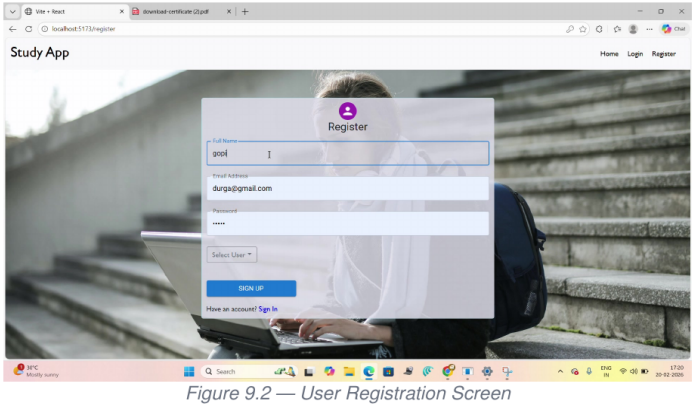

Figure 9.2 — User Registration Screen

9.3 Instructor Dashboard — Add Course

Instructors land on the Dashboard after login. The "Add Course" form allows selection of a course category from a dropdown, entry of the course title, educator name, price (0 for free), description, and

one or more sections. Each section has a title, description, and a file picker for a video or image asset.

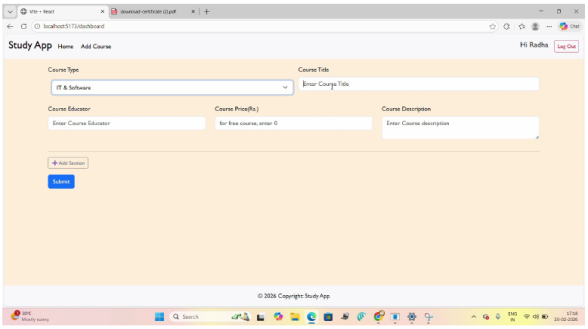

Figure 9.3 — Add Course Form (Empty)

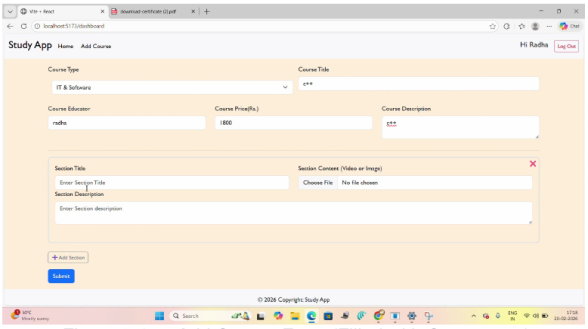

Figure 9.4 — Add Course Form (Filled with C++ course)

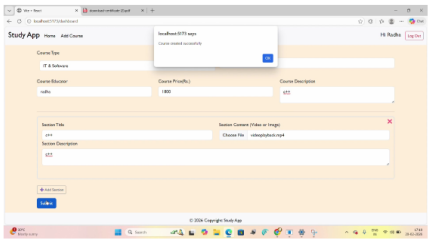

Figure 9.5 — Course Created Successfully confirmation

9.4 Instructor Course Management

The instructor home view lists all their published courses as cards. Each card displays the course title, description, category, number of sections, enrolled student count, and a Delete button.

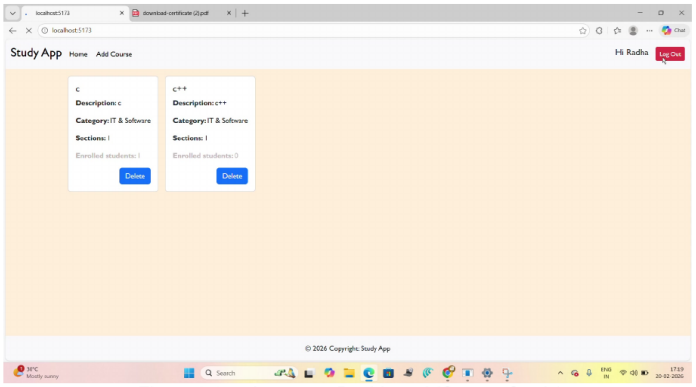

Figure 9.6 — Instructor Dashboard (Course Cards)

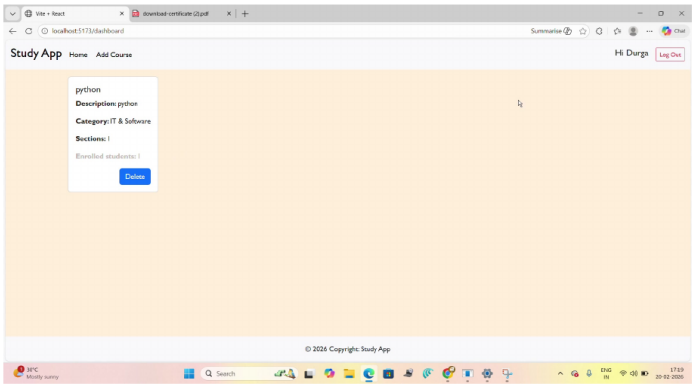

Figure 9.7 — Instructor Dashboard (Python course)

9.5 Student Home — Course Catalogue

After a student logs in they see the course catalogue. Course cards show module titles, descriptions, and an "Enroll" / "Pay Now" button. Available courses include Java, Python, C, and C++ in the IT & Software category.

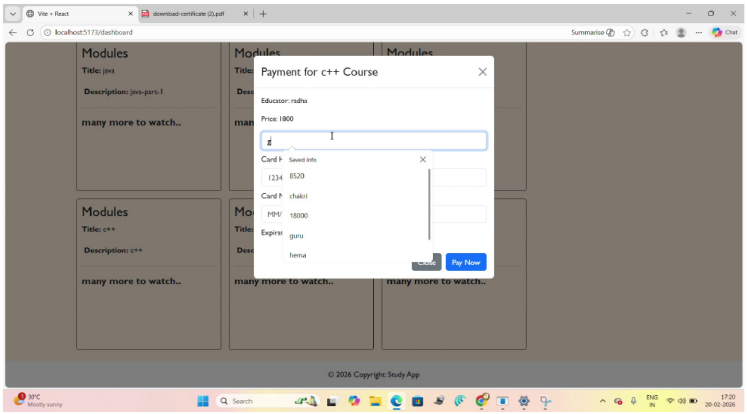

Figure 9.8 — Student Course Catalogue (Course grid)

9.6 Payment Modal

Clicking "Enroll" on a paid course opens the Payment modal, which displays the course name, educator, and price. Students fill in their card-holder name, 16-digit card number, expiration date, and CVV, then click "Pay Now".

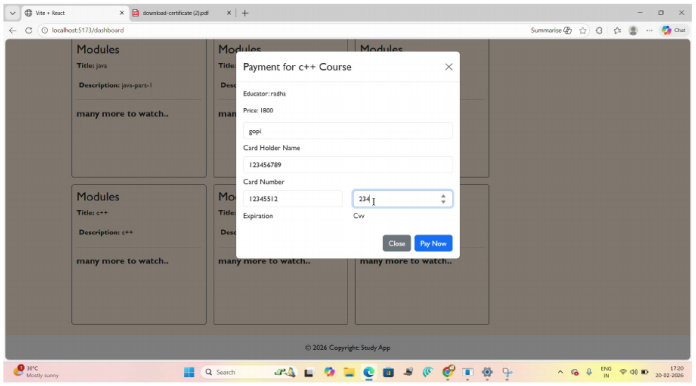

Figure 9.9 — Payment for C++ Course Modal

[Screenshot: Figure 9.10 — Enrollment Successful Confirmation]

9.7 Course Section — Video Player & Progress

The Course Section page has two panes. On the left, an accordion lists all sections with a "PLAY VIDEO" link and a "COMPLETED" status badge. On the right, the video player embeds the uploaded lesson video. Marking all sections complete reveals the "DOWNLOAD CERTIFICATE" link.

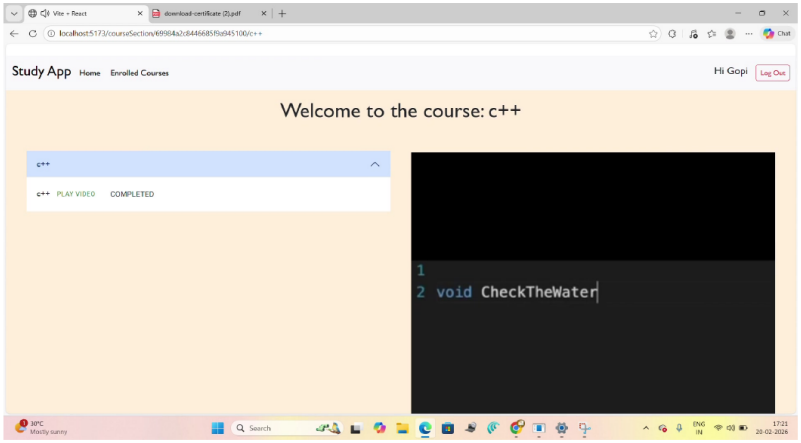

Figure 9.11 — Course Section Player (C++ lesson)

[Screenshot: Figure 9.12 — Course Section Player (COMPLETED status)]

9.8 Certificate Download

Once all sections are marked complete, the student can click "DOWNLOAD CERTIFICATE". A modal displays a preview of the Certificate of Completion bearing the student's name, course title, and completion date. A "DOWNLOAD CERTIFICATE" link exports the certificate as a PDF file.

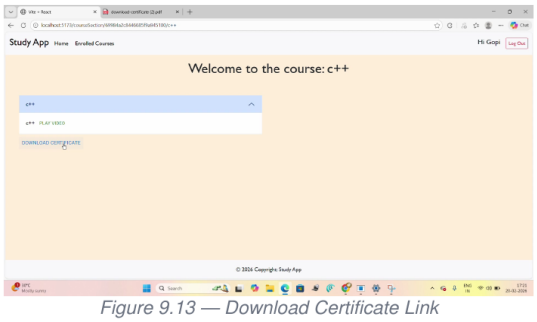

Figure 9.13 — Download Certificate Link

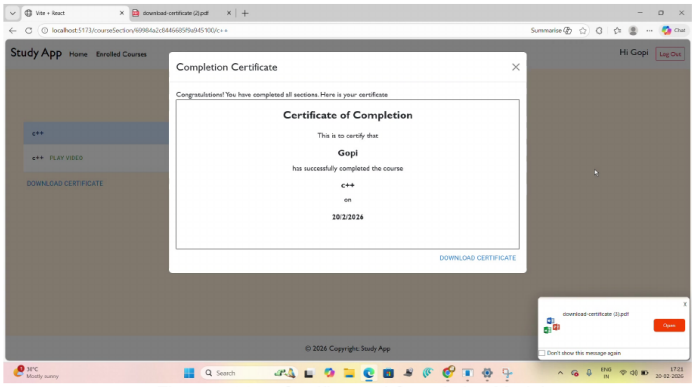

Figure 9.14 — Certificate of Completion Modal

10. Testing

10.1 Testing Strategy

LearnHub adopts a multi-layer testing approach combining manual functional testing, API-level testing, and frontend component testing. Given the project timeline, emphasis is placed on functional validation of all user flows.

10.2 Manual Functional Testing

Each core user flow was manually tested end-to-end:

TC #	Test Case	Steps / Input	Expected Result	Status
TC-01	Instructor Registration	Register as instructor with valid data	User created, token returned, redirected to dashboard	Pass
TC-02	Student Registration	Register as student with valid data	User created, token returned	Pass
TC-03	Login — Valid Credentials	Sign in with correct email/password	JWT token issued, dashboard loads	Pass
TC-04	Login — Invalid Credentials	Sign in with wrong password	Error message displayed	Pass
TC-05	Create Course	Instructor creates C++ course with video section	"Course created successfully" alert	Pass
TC-06	Delete Course	Instructor deletes a course	Course removed from dashboard	Pass
TC-07	Browse Courses	Student views all available courses	Course cards rendered correctly	Pass
TC-08	Paid Enrollment (Payment)	Student pays for C++ (Rs. 1800)	"Enroll Successfully" confirmation	Pass
TC-09	Free Enrollment	Student enrols in free course	Enrolled without payment modal	Pass
TC-10	Video Playback	Student plays course video	Video plays in embedded player	Pass
TC-11	Mark Section Complete	Student marks section as COMPLETED	Status badge updates to COMPLETED	Pass
TC-12	Download Certificate	Student downloads certificate after completion	PDF certificate downloaded with correct name & course	Pass
TC-13	Enrolled Courses View	Student views enrolled courses list	All enrolled courses displayed	Pass
TC-14	Logout	User clicks Log Out button	Token cleared, redirected to login	Pass

10.3 API Testing (Postman / Thunder Client)

All REST endpoints were tested using **Postman** and the VS Code extension **Thunder Client**. The following scenarios were validated:

- JWT token not attached → 401 Unauthorized response
- Expired token → 403 Forbidden response
- Creating a course as a student role → 403 Forbidden
- Enrolling without payment for a paid course → 400 Bad Request
- Downloading certificate before all sections complete → 403 Forbidden
- Invalid course ID → 404 Not Found
- Missing required fields on registration → 400 with field errors

10.4 Browser Compatibility

- Google Chrome 121+ — **Pass**
- Microsoft Edge 121+ — **Pass**
- Mozilla Firefox 122+ — **Pass**
- Safari 17+ — **Not tested**

11. Screenshots & Demo

The following section consolidates key application screenshots captured from the live demo video recorded on **20 February 2026**. The complete demo video is available in the project repository.

11.1 Demo Video

A full walkthrough demo (approx. 3 minutes 30 seconds) demonstrating all major flows — instructor course creation, student registration, payment, video playback, progress tracking, and certificate download — is available at:

`https://github.com/NCHAKRADHAR-SINGH1/learnhub-project/blob/main/demo.mp4`

11.2 Demo Flow Summary

- **Step 1 — Instructor Login:** Instructor "Radha" logs in with radha@gmail.com and is directed to the instructor dashboard.
- **Step 2 — Course Creation:** Radha creates a new "c++" course under "IT & Software" category, priced at Rs. 1800, uploads a video lesson (videoplayback.mp4) for the section.
- **Step 3 — Course Confirmation:** "Course created successfully" alert appears. Dashboard refreshes to show both "c" and "c++" cards.
- **Step 4 — Instructor Logout:** Radha logs out. The app returns to the public login page.
- **Step 5 — Student Registration:** New student "Gopi" registers using gopi@gmail.com with role "Student".
- **Step 6 — Student Login & Catalogue:** Gopi logs in and sees the full course catalogue: Java, Python, C, C++.
- **Step 7 — Payment & Enrollment:** Gopi selects C++ (Rs. 1800), fills the payment modal, and clicks "Pay Now". "Enroll Successfully" confirmation appears.
- **Step 8 — Video Playback:** Gopi enters the C++ course, plays the embedded lesson video showing C++ code.
- **Step 9 — Section Completion:** Section is marked COMPLETED. The "DOWNLOAD CERTIFICATE" link appears below the section list.
- **Step 10 — Certificate Download:** Gopi clicks "DOWNLOAD CERTIFICATE". The Certificate of Completion modal shows "Gopi has successfully completed the course c++ on 20/2/2026". PDF is downloaded.

11.3 Additional Screenshots

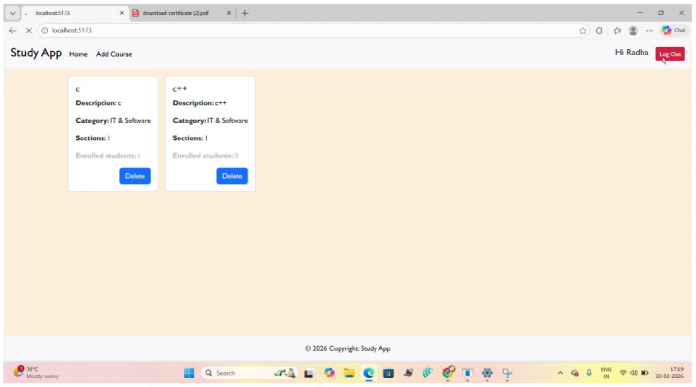

Figure — Instructor Dashboard — Course Cards

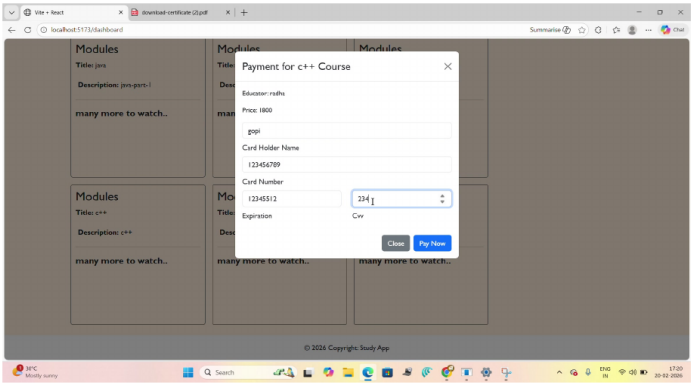

Figure — Payment Modal — C++ Course

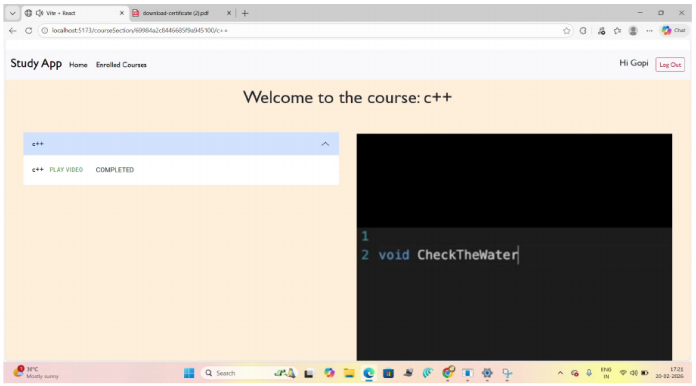

Figure — Course Section — Video Player

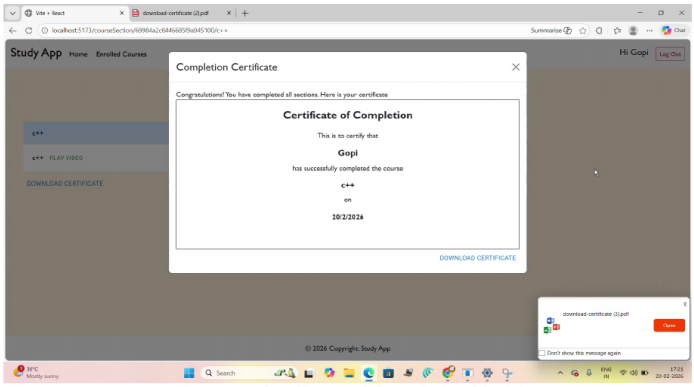

Figure — Certificate of Completion Modal

12. Known Issues

The following known limitations and bugs have been identified during development and testing. They are documented for transparency and are targeted for resolution in upcoming iterations.

Severity	Issue	Description
HIGH	**Payment Security**	The current payment implementation is a mock/simulation. Card details are stored in plain text in the payments collection. A production system must integrate a PCI-DSS compliant payment gateway (e.g., Razorpay, Stripe) and must never store raw card details.
HIGH	**No HTTPS**	The application currently runs on HTTP. Deployment to production must enforce HTTPS using TLS/SSL certificates (e.g., Let's Encrypt via Nginx or a cloud provider).
MEDIUM	**File Storage on Disk**	Uploaded videos and images are stored on the server's local filesystem. This is not scalable for production. A cloud object store such as AWS S3, Cloudinary, or Azure Blob Storage should be used.
MEDIUM	**No Email Verification**	User accounts are activated immediately after registration without email verification. Adding email confirmation (via Nodemailer / SendGrid) would reduce fake account creation.
MEDIUM	**No Pagination**	The course catalogue and instructor dashboard load all records in a single API call. Large datasets will degrade performance. Pagination or infinite-scroll should be implemented.
LOW	**Browser Alert for Confirmations**	Success and error messages are displayed using the browser's native window.alert() which is intrusive. React toast notifications (e.g., react-toastify) would provide a better UX.
LOW	**No Course Search / Filter**	The home page lists all courses without a search bar or category filter. Adding search and filter capabilities would improve discoverability.
LOW	**No Password Reset**	There is no "Forgot Password" / reset flow. Users who forget their password cannot recover their accounts without direct database intervention.
LOW	**No Input Sanitization**	Form inputs are not sanitized against XSS or injection attacks on the frontend. Server-side validation with express-validator should be added.

13. Future Enhancements

LearnHub can be enhanced by integrating a secure payment gateway such as Razorpay or Stripe to enable safe and reliable transactions. This will improve payment security and provide proper transaction management.

The platform can also include a course rating and review system, allowing students to share feedback and help others select suitable courses. Adding search and filtering options will further improve usability and make course discovery easier.

Email notifications can be implemented to inform users about important events such as registration, course enrollment, and certificate generation. Discussion forums will also allow better interaction between instructors and students.

A mobile application can be developed to improve accessibility and allow users to learn from anywhere. Additionally, analytics tools will help instructors track course performance and student progress.

Finally, deploying the platform on cloud infrastructure will improve scalability, performance, and reliability, making the system suitable for real-world usage.