
Project Documentation: Edemy E-Learning LMS Platform

1. Introduction

Edemy is a full-stack e-learning web application built using the **MERN** stack (MongoDB, Express.js, React, Node.js). The platform allows students to browse, purchase, and consume educational courses, while providing a dedicated dashboard for educators to create and manage their content. The application integrates modern third-party services for robust authentication, payment processing, and media management.

2. Key Features

- **Dual-Role User System:** Supports both **Student** and **Educator** roles with distinct functionalities.
- **Secure Authentication:** User sign-up, sign-in, and session management are handled by **Clerk**.
- **Database Synchronization:** User data is automatically synced from Clerk to the application's MongoDB database via **webhooks**.
- **Course Creation (Educator):** Educators can create detailed courses, including titles, rich-text descriptions, pricing, and multi-level course content (chapters and lectures).
- **Media Management:** Course thumbnails are uploaded to and served from **Cloudinary**.
- **Payment Processing:** Secure course enrollment is handled through **Stripe Checkout**.
- **Enrollment & Progress Tracking:** Successful payments are verified via Stripe webhooks, and the database is updated to reflect student enrollments and track their progress through courses.
- **Course Interaction (Student):** Students can browse all courses, view detailed course pages, purchase courses, and access their enrolled courses in a dedicated "My Enrollments" section.
- **Role-Based Access Control:** API endpoints for educator-specific actions are protected by custom middleware.

3. Technology Stack

Category	Technology / Service
Frontend	**React**, **Vite**, **React Router**, **Tailwind CSS**, **Axios**, react-toastify
Backend	**Node.js**, **Express.js**, **Mongoose** (for MongoDB ODM)
Database	**MongoDB Atlas** (NoSQL, document-based database)
Services	**Clerk** (Authentication), **Stripe** (Payments), **Cloudinary** (Image Storage), **Vercel** (Deployment)
Libraries	svix (Webhook Verification), multer (File Upload Handling)

Export to Sheets

4. Project Structure

The project is organized into a monorepo structure with two main directories: server for the backend and client (or user) for the frontend.

Backend (server) Structure

/server

```
├ configs/          # Database, Cloudinary, Multer configurations
├ controllers/      # Core application logic for each route
├ middlewares/      # Custom middleware (e.g., protectEducator)
├ models/           # Mongoose schemas (User, Course, Purchase)
├ routes/           # API route definitions
├ .env              # Environment variables (secret keys)
└ server.js         # Main server entry point
```

Frontend (client or user) Structure

/client

```
└── src/
    ├── assets/       # Static assets (images, dummy data)
    ├── components/   # Reusable React components (student & educator)
    ├── context/      # React Context for global state (AppContext)
    ├── pages/        # Top-level page components (student & educator)
    ├── App.jsx       # Main application component with routing
    └── main.jsx      # Application entry point, providers setup
```

5. Backend Architecture (API)

Database Schemas

- **User.js**: Stores user information synced from Clerk, including _id (Clerk ID), name, email, imageUrl, and an array of enrolledCourses.
- **Course.js**: Stores all course details, including title, description, price, content (nested chapters and lectures), and references the educator and enrolledStudents.
- **Purchase.js**: Creates a record for each transaction, linking a userId to a courseId with a status (pending, completed, failed).
- **CourseProgress.js**: Tracks which lectures a specific user has completed for a specific course.

API Endpoints

The API is structured as a RESTful service.

- **Webhook Routes:**
    - POST /clerk: Receives user.created events from Clerk to create users in the database.
    - POST /stripe: Receives checkout.session.completed events from Stripe to finalize course enrollments.
- **Course Routes (/api/course):**
    - GET /all: Fetches all published courses for the public course list.
    - GET /:id: Fetches the details of a single course.
- **User Routes (/api/user):**
    - GET /data: Fetches the logged-in user's data from the database.
    - POST /purchase: Creates a Purchase record and a Stripe Checkout session.
    - POST /add-rating: Adds a student's rating to a course.
- **Educator Routes (/api/educator):**
    - GET /update-role: Updates a user's Clerk metadata to grant them the 'educator' role.
    - POST /add-course: Creates a new course (protected).
    - GET /courses: Fetches all courses created by the logged-in educator (protected).

6. Frontend Architecture

State Management

Global state is managed using the **React Context API**. The AppContext.jsx serves as a centralized store for:

- Fetched data like allCourses and userData.
- Authentication state like isEducator.
- Functions that make API calls to the backend (e.g., fetchAllCourses, fetchUserData).

Routing

Client-side routing is handled by **React Router**. The main router is set up in App.jsx, which defines paths for all pages. It includes a nested route structure for the educator dashboard, using a layout component (Educator.jsx) and an <Outlet /> to render child pages.

7. Setup and Installation

Backend

1. Navigate to the server directory: cd server

2. Install dependencies: npm install
3. Create a .env file and add all required keys from MongoDB, Clerk, Stripe, and Cloudinary.
4. Start the server: npm run server

Frontend

1. Navigate to the client directory: cd client
2. Install dependencies: npm install
3. Create a .env.local file and add VITE_CLERK_PUBLISHABLE_KEY and VITE_BACKEND_URL.
4. Start the development server: npm run dev

8. Deployment

The application is deployed on **Vercel** via a connected GitHub repository.

- **CI/CD:** Vercel provides a built-in CI/CD pipeline. Pushing code to the main branch on GitHub automatically triggers a new build and deployment.
- **Environment Variables:** All backend secrets from the .env file must be configured in the Vercel project's **Settings → Environment Variables**.
- **Webhook Configuration:** After the initial deployment, the live Vercel URL must be used to update the endpoint URLs in the **Clerk** and **Stripe** webhook dashboards.