🎯 Project Objective

To develop a **full-stack LMS application** where:

- **Admins** can manage users, mentors, and approve/publish courses.
- **Mentors** can create courses, upload lessons, and evaluate students.
- **Students** can enroll, learn, and track their progress.

The goal is to provide a seamless, scalable, and user-friendly experience for all three roles.

🧠 Core Features by Role

👑 Admin

1. Manage all users (create, view, update, delete).
2. Assign or change user roles (promote mentor, demote user, etc.).
3. Approve or reject mentor-created courses before publishing.
4. View platform analytics — total users, active courses, enrollments, etc.
5. Manage categories, subcategories, and tags.
6. Handle reports, complaints, and feedback.

👨‍🏫 Mentor

1. Create, edit, and delete courses.
2. Add lessons (video, PDF, text content, or links).
3. Create quizzes or assignments for enrolled students.
4. Track student enrollments and performance.

5. Send announcements or updates to students.
6. Manage course visibility (draft/published).

👩‍🎓 Student/User

1. Browse and search available courses.
2. Enroll in free or paid courses.
3. Watch lessons, download resources, and take quizzes.
4. View their learning progress and completion percentage.
5. Submit assignments and receive grades.
6. Leave reviews or feedback for courses.

🧩 Technology Stack

Layer	Technology	Description
Frontend	Angular 19	Dynamic UI, role-based dashboards, responsive design
Backend	Node.js + Express.js	REST API server, authentication, and business logic
Database	MongoDB	NoSQL database for users, courses, and quiz data
Authentication	JWT (JSON Web Tokens)	Secure login and role-based access control
UI Library	Angular Material / TailwindCSS	For a clean, responsive interface
Optional Integrations	AWS S3 (file uploads), Socket.IO (live classes), Stripe (payments)	

🏗️ System Modules

1. Authentication & Authorization

- Register, Login, Logout
- Role-based access control (Admin, Mentor, User)
- JWT token management (frontend & backend)

2. User Management (Admin Module)

- View all users and their roles
- Edit or deactivate users
- Promote users to mentor roles
- Dashboard overview with analytics (user growth, course count, etc.)

3. Course Management (Mentor Module)

- Create new courses with title, description, category, and pricing
- Add multiple lessons to each course (video/text/document)
- Upload resources and materials
- Manage enrolled students
- Publish/unpublish courses (subject to admin approval)

4. Course Catalog (User Module)

- Browse or search courses
- Filter by category, mentor, or price (free/paid)
- View detailed course content

- Enroll and start learning

5. Learning Module

- View lessons sequentially (lock next lessons until completion)
- Take quizzes or assignments
- Track progress and scores
- Certificate generation after course completion

6. Quiz & Assignment Module

- Mentor can create quizzes per course
- Students take quizzes and get instant results
- Auto or manual grading
- Score tracking in student dashboard

7. Review & Feedback

- Students can review courses (rating + comments)
- Admin can moderate feedback
- Mentor can respond to reviews

8. Notification System (optional)

- Announcements from mentors/admin
- Email or in-app notifications for enrollments, quiz results, etc.

🧱 System Architecture

1. Frontend (Angular)

- **Modules:** Auth, Admin, Mentor, Student
- **Services:** AuthService, ApiService, CourseService, QuizService
- **Guards:** AuthGuard (login check), RoleGuard (role-based routes)
- **Components:**
    - Login/Register
    - Dashboards (Admin, Mentor, Student)
    - Course List / Course Detail
    - Quiz Page
    - Profile Page

2. Backend (Node + Express)

- **Routes:**
    - `/auth` → Login, Register
    - `/users` → User Management
    - `/courses` → Course CRUD, Enrollment
    - `/quizzes` → Quiz CRUD and results
- **Middleware:** JWT authentication, role authorization, input validation
- **Models:** User, Course, Lesson, Quiz, Enrollment
- **Controllers:** handle business logic (create, update, view, etc.)

3. Database (MongoDB)

- **Collections:**

- `users`: { name, email, password, role, status }
- `courses`: { title, mentorId, lessons[], students[], published }
- `lessons`: { courseId, title, content, videoUrl }
- `quizzes`: { courseId, questions[], results[] }
- `enrollments`: { userId, courseId, progress, status }

🧭 User Flow Summary

🔸 Student

1. Register → Login → Browse courses
2. Enroll in a course → View lessons → Take quiz → Complete course
3. Receive completion certificate & rate the course

🔸 Mentor

1. Login → Create Course → Add Lessons/Quizzes
2. Track Enrollments → Grade Students → View Feedback

🔸 Admin

1. Login → Manage Users & Roles → Approve Courses
2. Monitor Platform Activity → View Reports

📊 Dashboard Breakdown

Role	Dashboard Widgets
Admin	Total Users, Total Courses, Pending Approvals, Active Mentors, Platform Statistics
Mentor	My Courses, Enrollments Count, Recent Student Submissions, Ratings Overview
Student	My Courses, Progress Tracker, Certificates Earned, Quiz Scores

---

🗂️ Folder Organization (suggested)

Backend

```
backend/
├── config/
├── controllers/
├── middleware/
├── models/
├── routes/
├── utils/
├── app.js
└── server.js
```

Frontend (Angular)

```
frontend/
├── src/app/
│    ├── core/
│    ├── features/
│    ├── shared/
│    └── app-routing.module.ts
├── assets/
└── environments/
```

🧾 Deliverables

Phase	Deliverable	Responsibility
Phase 1	Authentication (Login/Register + Role Setup)	Backend + Frontend Team
Phase 2	Course Management (Mentor & Admin)	Mentor Module Developer
Phase 3	Course Catalog + Enrollment	Student Module Developer
Phase 4	Quiz System + Progress Tracking	Backend & Frontend Integration
Phase 5	Dashboard & Reports	UI Developer + Backend API
Phase 6	Testing + Deployment	QA + DevOps

⚙️ Development Guidelines

1. Use JWT-based authentication.
2. Use environment-based configuration for API URLs.
3. Apply role-based route guards on frontend and backend.
4. Use Angular Material components for UI consistency.
5. Follow RESTful API conventions.
6. Maintain reusable services and shared modules.
7. Store sensitive credentials in `.env` files only.