Project: Heliora

Project Overview

Heliora is a full-stack, cross-platform (web and mobile) learning management system (LMS) designed to empower teachers to create, manage, and monitor interactive courses, while enabling students to engage with content and track their progress. The platform focuses on delivering a seamless, professional-grade user experience with modern features like real-time analytics, gamification, and cross-platform accessibility. As a beginner, you'll face intermediate-level challenges in implementing a scalable backend, responsive frontend, and a mobile app, all while ensuring security, performance, and usability. The goal is to create a portfolio piece that screams "hire me" to tech giants.

Target Audience

- **Teachers**: Educators or professionals creating courses on subjects, skills, or tech stacks.
- **Students**: Learners seeking structured, interactive courses with progress tracking.
- **Administrators**: Platform managers overseeing user roles and content moderation.

Tech Stack (Mandatory)

To ensure a modern, employable skill set, you must use the following tech stack. Deviations will not be tolerated:

- **Frontend (Web)**: Angular, Tailwind CSS, Angular Material
    - Angular for structured, component-based UI with TypeScript, Tailwind for rapid, responsive styling, and Angular Material for pre-built UI components.
- **Mobile App**: Flutter
    - Flutter for high-performance, native-like cross-platform iOS and Android apps, leveraging Dart for a structured development experience.
- **Backend**: Node.js, Express.js
    - Build a RESTful API for scalability and ease of integration.
- **Database**: MongoDB
    - NoSQL for flexible course and user data storage.
- **Authentication**: JSON Web Tokens (JWT)
    - Secure user authentication and role-based access.
- **File Storage**: AWS S3
    - Store course materials (videos, documents) securely.
- **Real-Time**: Socket.IO
    - Enable real-time updates for student enrollment and progress tracking.
- **Deployment**: Vercel (web), Firebase App Distribution (mobile), Render (backend), MongoDB Atlas (database)
    - Vercel for web hosting, Firebase for mobile app distribution, Render for backend, and MongoDB Atlas for cloud database.
- **Version Control**: Git, GitHub
    - Maintain a clean commit history with descriptive messages.
- **Testing**: Jest (backend), Karma/Jasmine (Angular), flutter_test (Flutter), Postman (API testing)
    - Ensure code reliability and API functionality.
- **Documentation**: Swagger/OpenAPI
    - Generate interactive API documentation.
- **Code Quality**: ESLint, Prettier, Angular CLI

- Enforce consistent code style and modular architecture.

Features and Requirements

The project must include the following features, categorized by user type and platform. Each feature has specific requirements to ensure a professional-grade implementation.

1. User Authentication and Roles

- **Feature**: Secure registration, login, and role-based access for teachers, students, and admins.
- **Requirements**:
  - Users register with email, password, and role (teacher/student/admin).
  - JWT-based authentication with refresh tokens for session management.
  - Passwords hashed using bcrypt.
  - Role-based middleware in Express.js to restrict access (e.g., only teachers create courses).
  - Forgot password functionality with email-based reset links (use a mock email service like Nodemailer for development).
  - Mobile app mirrors web authentication flow using Flutter's flutter_secure_storage for token storage.

2. Teacher Dashboard (Web and Mobile)

- **Feature**: Teachers create, manage, and monitor courses.
- **Requirements**:
  - **Course Creation**:
    - Form to input course title, description, subject/skill (e.g., Python, Calculus), and difficulty level.
    - Drag-and-drop interface to build a course outline (e.g., modules like "Introduction," "Advanced Topics").
    - Upload content per module: videos (MP4, max 100MB), documents (PDF, max 10MB), or text-based exercises.
  - **Course Management**:
    - Edit/delete courses or individual modules.
    - Publish/unpublish courses to control student access.
  - **Student Monitoring**:
    - Real-time dashboard showing student enrollment, progress (%), and exercise scores.
    - Filterable table (by course, student, or completion status) with export to CSV.
    - Push notifications (mobile) for student milestones (e.g., course completion).
  - **UI/UX**:
    - Responsive design with Tailwind CSS and Angular Material (web).
    - Consistent mobile UI using Flutter widgets.
    - Accessible design (WCAG 2.1 compliance, e.g., ARIA labels, keyboard navigation).

3. Student Dashboard (Web and Mobile)

- **Feature**: Students browse, enroll, and complete courses.
- **Requirements**:
  - **Course Discovery**:
    - Searchable course catalog with filters (subject, difficulty, teacher).

        - Course preview page showing outline, duration, and teacher bio.
    - **Course Enrollment**:
        - One-click enrollment with confirmation modal.
        - Display enrolled courses in a dashboard with progress bars.
    - **Content Consumption**:
        - Step-by-step module navigation (e.g., video → document → exercise).
        - Video player with playback controls (HTML5 video or Flutter's video_player).
        - Document viewer for PDFs (use ng2-pdf-viewer for Angular, flutter_pdfview for Flutter).
        - Exercise submission form supporting multiple-choice or text answers.
    - **Progress Tracking**:
        - Visual progress tracker (e.g., circular progress bar) per course.
        - Gamification: Earn badges for milestones (e.g., "First Module Completed").
    - **UI/UX**:
        - Mobile-first design with smooth transitions.
        - Offline support for course content (cache content using Flutter's hive or sqflite).

4. Admin Dashboard (Web Only)

- **Feature**: Admins manage users and moderate content.
- **Requirements**:
    - View, edit, or delete user accounts (teachers/students).
    - Approve/reject courses before publishing (moderation queue).
    - Basic analytics: total users, courses, and enrollments.
    - UI: Simple table-based interface with pagination using Angular Material.

5. Real-Time Features

- **Feature**: Real-time updates for critical actions.
- **Requirements**:
    - Use Socket.IO for:
        - Notifying teachers when students enroll or complete modules.
        - Updating student progress in real-time on teacher dashboards.
    - Mobile app handles Socket.IO events consistently with web (using ngx-socket-io for Angular, socket_io_client for Flutter).

6. Performance and Scalability

- **Requirements**:
    - Backend API handles 100 concurrent users without latency (>500ms).
    - Optimize database queries with MongoDB indexes.
    - Lazy-load course content on web (Angular's loadChildren) and mobile (Flutter's ListView.builder).
    - Cache API responses using Redis (optional, for bonus points).

7. Security

- **Requirements**:
    - HTTPS for all API calls.
    - Input validation and sanitization to prevent XSS/SQL injection (using Angular's DomSanitizer and Flutter's input validation).
    - Rate limiting on login and API endpoints (Express Rate Limit).

- Secure file uploads (validate file types, scan for malware using AWS Lambda).

8. Testing and Documentation

- **Requirements**:
  - Write unit tests for at least 70% of backend API endpoints using Jest.
  - Test Angular components with Karma/Jasmine and Flutter widgets with flutter_test.
  - Test all API routes manually with Postman; provide a Postman collection.
  - Document API with Swagger/OpenAPI (include endpoints like /courses, /users).
  - Write a README.md with:
    - Project setup instructions.
    - Tech stack overview.
    - Deployment steps for Vercel, Firebase App Distribution, Render, and MongoDB Atlas.
    - Screenshots of web and mobile UIs.

Non-Functional Requirements

- **Code Quality**:
  - Follow ESLint, Prettier, and Angular CLI for consistent code style.
  - Modular code with reusable Angular components, Flutter widgets, and Express middleware.
  - Clear separation of concerns (e.g., controllers, services, models in backend).
- **Performance**:
  - Web app loads in <2 seconds (use Lighthouse for optimization).
  - Mobile app startup time <3 seconds.
- **Cross-Platform**:
  - Web app supports Chrome, Firefox, and Safari.
  - Mobile app supports iOS 15+ and Android 10+.
- **Accessibility**:
  - Minimum WCAG 2.1 Level AA compliance.
- **Error Handling**:
  - Graceful error messages for users (e.g., "Course not found").
  - Log errors to console (use Winston for backend logging).

Deliverables

- **Source Code**: Hosted on a public GitHub repository with a clean commit history.
- **Deployed Application**:
  - Web app live on Vercel.
  - Mobile app distributed via Firebase App Distribution (testable on iOS and Android).
- **Documentation**:
  - README.md with setup, deployment, and screenshots.
  - Swagger API documentation.
  - Postman collection for API testing.
- **Demo Video**: 5-minute video showcasing all features (web and mobile) with a voiceover explaining your approach.

Success Criteria

To meet my expectations and avoid termination, your project must:

- Implement all mandatory features with no critical bugs.
- Achieve a professional, polished UI/UX that rivals commercial LMS platforms.
- Demonstrate intermediate-level skills in full-stack development (e.g., REST API, MongoDB queries, Angular state management, Flutter widget architecture).
- Be completed within 2 months (80-120 hours, 10-15 hours/week).
- Impress hypothetical tech giant recruiters with:
    - Modern tech stack (Angular, Flutter, Node.js, MongoDB).
    - Scalable architecture (microservices-like API, AWS S3).
    - Attention to detail (accessibility, testing, documentation).

Timeline (8 Weeks, 10-15 Hours/Week)

- **Week 1**: Project setup (GitHub, Node.js, Angular, Flutter, MongoDB).
- **Week 2**: Authentication (JWT, user roles, forgot password).
- **Week 3**: Teacher dashboard (course creation, content upload).
- **Week 4**: Student dashboard (course enrollment, content consumption).
- **Week 5**: Admin dashboard and real-time features (Socket.IO).
- **Week 6**: Testing (Jest, Karma/Jasmine, flutter_test, Postman) and performance optimization.
- **Week 7**: Documentation (Swagger, README) and deployment (Vercel, Firebase App Distribution, Render, MongoDB Atlas).
- **Week 8**: Final polish, demo video, and submission.

Additional Notes

- **Challenges for Beginners**:
    - Learning Flutter and Dart alongside Angular.
    - Implementing Socket.IO for real-time updates.
    - Integrating AWS S3 for file storage.
    - Ensuring cross-platform compatibility.
- **Why This Project Stands Out**:
    - Combines web and mobile development, showcasing versatility.
    - Uses a modern, in-demand tech stack (Angular, Flutter, Node.js, MongoDB).
    - Includes advanced features like real-time updates and gamification.
    - Demonstrates full-stack proficiency (frontend, backend, database, cloud).
- **Resources**:
    - Official docs for Angular, Flutter, Node.js, MongoDB, AWS S3.
    - Tutorials on JWT, Socket.IO, and Tailwind CSS.
    - Free tiers of Vercel, Firebase, Render, AWS, and MongoDB Atlas for deployment.

Failure to deliver a fully functional, professional-grade application will result in a poor performance review. I expect you to rise to the challenge and produce a project that could theoretically land you interviews at top tech companies. Get to work.