3. Application Architecture Documentation

Brief overview of the application's architecture.

Backend (Node.js / Express)

RESTful API using Node.js and Express, following **Model-Route-Controller** pattern:

- **Models (`/models`):** Mongoose Schemas define data structures (e.g., `User.js` , `Subject.js`) for MongoDB collections, specifying fields, types, and validations.
- **Routes (`/routes`):** API endpoints (e.g., `/api/users` , `/api/subjects/:id`) map HTTP requests to controllers, with middleware for security.
- **Controllers (`/controllers`):** Handle business logic, process requests, interact with models for CRUD operations, and return JSON responses.
- **Authentication & Authorization (`/middleware`):** `protect` middleware verifies JWT for user authentication; `authorize` middleware restricts endpoint access by role (e.g., 'Admin', 'Teacher').

Frontend (Angular)

Single-Page Application (SPA) built with Angular, using **standalone components**:

- **Component-Based UI (`/app`):** Reusable components (e.g., `LoginComponent` , `UserListComponent`) manage templates, styles, and logic.
- **Services (`/_services`):** Injectable services (`AuthService` , `UserService`) handle backend API communication, keeping components focused on UI.
- **Routing (`app.routes.ts`):** Angular Router controls navigation; `authGuard` (`/_guards/auth-guard.ts`) checks login status and roles via `AuthService` for protected routes, ensuring client-side security and smooth UX.