Full Stack Coding Challenge: Secure Task Management System

Overview

Design and implement a **secure Task Management System** using **role-based access control (RBAC)** in a **modular NX monorepo**. The system must allow users to manage tasks securely, ensuring **only authorized users** can access and modify data based on their **roles and organizational hierarchy**.

⏱ **Time Limit**: 8 hours maximum – please do not spend more than 8 hours on this assessment

Monorepo Structure (NX Workspace)

Please name your repository with your first name's first letter, last name, a hyphen (-) and a randomly generated uuid. For example, John Doe would be jdoe-0a19fc14-d0eb-42ed-850d-63023568a3e3.

None

```
apps/
 api/ → NestJS backend
 dashboard/ → Angular frontend

libs/
 data/ → Shared TypeScript interfaces & DTOs
 auth/ → Reusable RBAC logic and decorators
```

Core Features

Backend (NestJS + TypeORM + SQLite/PostgreSQL)

Data Models

- **Users**

● **Organizations** (2-level hierarchy)
● **Roles**: `Owner`, `Admin`, `Viewer`
● **Permissions**
● **Tasks** (resource)

Access Control Logic

● Implement **decorators/guards** for checking access.
● Enforce **ownership & org-level access**.
● Implement **role inheritance logic**.
● Scope **task visibility based on role**.
● Implement **basic audit logging** (to console or file).

API Endpoints

● `POST /tasks` – Create task (with permission check) ●
`GET /tasks` – List accessible tasks (scoped to role/org) ●
`PUT /tasks/:id` – Edit task (if permitted)
● `DELETE /tasks/:id` – Delete task (if permitted) ● `GET`
`/audit-log` – View access logs (Owner/Admin only)

Authentication Requirement

Do not use mock auth.

● Implement **real authentication using JWT**.
● Authenticate via login and include token in all requests. ●
Include token verification middleware/guard in all endpoints.

Frontend (Angular + TailwindCSS)

Task Management Dashboard

● Create/Edit/Delete tasks
● Sort, filter, and categorize (e.g., "Work",
"Personal") ● Drag-and-drop for reordering/status
changes
● Responsive design (mobile → desktop)

Authentication UI

● Include **login UI** to authenticate against backend. ● Upon login, store the JWT and attach it to all API requests.

State Management

● Use any state management solution you prefer.

Bonus Features (Optional)

● Task completion visualization (e.g., bar chart)
● Dark/light mode toggle
● Keyboard shortcuts for task actions

Testing Strategy

● **Backend**: Use **Jest** to test RBAC logic, authentication, and endpoints. ● **Frontend**: Use **Jest/Karma** to test components and state logic.

README (Must Include)

Setup Instructions

● How to run both backend and frontend apps
● `.env` setup (JWT secrets, DB config)

Architecture Overview

● NX monorepo layout and rationale
● Explanation of shared libraries/modules

Data Model Explanation

● Describe schema and include ERD/diagram

Access Control Implementation

● How roles, permissions, and organization hierarchy work
● How JWT auth integrates with access control

API Docs

- Endpoint list with sample requests/responses

Future Considerations

- Advanced role delegation
- Production-ready security: JWT refresh tokens, CSRF protection, RBAC caching
- Scaling permission checks efficiently

Evaluation Criteria

- Secure and correct RBAC implementation
- JWT-based authentication
- Clean, modular architecture in NX
- Code clarity, structure, and maintainability
- Responsive and intuitive UI
- Test coverage
- Documentation quality
- Bonus for elegant UI/UX or advanced features

Important

To keep your assessment properly logged and reviewed, submit your completed work through our official submission portal:

https://forms.gle/1iJ2AHzMWsWecLUE6

You have 4 business days from the moment you submit this form to complete and submit your assessment. Assessments sent after that window may not be reviewed.

This portal ensures your work is tracked correctly and routed to the hiring team for evaluation.