**Full Stack Specifications**

**1. Web Frontend**

* **Framework:** Next.js (latest stable version)
* **Language:** TypeScript
* **Styling:** Tailwind CSS (consistent styling with mobile)
* **State Management:** React Context API
* **Routing:** Next.js built-in routing
* **Responsibilities:**
  + Render pages and components
  + Consume backend API via axios
  + Validate API responses using TypeScript types
  + Handle forms, UI state, and client-side logic
  + Connect to backend real-time server via Socket.IO client and update UI automatically on events
  + Use the roles/permissions to show/hide UI elements and functionality

**2. Mobile App**

* **Framework:** React Native + Expo (latest stable)
* **Language:** TypeScript
* **Styling:** NativeWind (consistent styling with web)
* **State Management:** React Context API
* **Navigation:** React Navigation
* **Responsibilities:**
  + Render mobile screens and components
  + Consume backend API via axios
  + Validate API responses using TypeScript types
  + Maintain identical data structures to web frontend
  + Connect to backend real-time server via Socket.IO client and update UI automatically on events
  + Use the roles/permissions to show/hide UI elements and functionality

**3. Backend API**

* **Framework:** Node.js + Express (latest stable)
* **Language:** TypeScript
* **ORM:** Prisma
* **Database:** PostgreSQL
* **Responsibilities:**
  + Expose REST API endpoints
  + Validate request bodies and query parameters
  + Return typed responses matching frontend/mobile expectations
  + Handle authentication and authorization using JWT
  + Interact with PostgreSQL exclusively through Prisma
  + Integrate Socket.IO server for real-time updates
  + Emit events to web and mobile clients immediately after any data changes
  + Ensure real-time events respect permissions

**4. Database**

* **Database:** PostgreSQL
* **Access:** Prisma client (generated TypeScript types)
* **Responsibilities:**
  + Store all persistent data: users, content, app data
  + Enforce referential integrity and constraints

**5. Cache / Session Store**

* **Technology:** Redis
* **Purpose:** Cache frequently accessed data, manage sessions
* **Responsibilities:**
  + Store temporary session data
  + Cache API responses or computed objects for fast retrieval
  + Use TypeScript interfaces for all stored objects
  + Optionally store transient real-time states (e.g., presence or active sessions)

**6. File Storage**

* **Technology:** MinIO (S3-compatible)
* **Purpose:** Store user-uploaded files and assets
* **Responsibilities:**
  + Upload, download, delete files
  + Manage buckets and object metadata
  + Use TypeScript interfaces for file metadata

**7. Authentication**

* **Method:** JWT (JSON Web Tokens)
* **Responsibilities:**
  + Issue JWT tokens after login
  + Verify JWT tokens on every protected endpoint
  + Use a TypeScript interface to define the JWT payload: user ID, roles, permissions
  + JWT payload to include roles/permissions

**8. Hosting & Deployment**

* **OS:** Ubuntu 22.04 LTS
* **Reverse Proxy:** Nginx (latest stable)
* **Containerization:** Podman (latest stable)
* **Responsibilities:**
  + Host backend API, web frontend, and static assets
  + Proxy HTTP requests to correct container
  + Manage container lifecycle and logs

**Rules for the AI**

1. Use exactly the technologies listed above. Any additional technologies that you want to use should be consulted with me beforehand.
2. Follow strict TypeScript usage wherever TypeScript is indicated (frontend, mobile, backend, cache, file storage, JWT payloads)
3. All API requests/responses between frontend, mobile, and backend must use the same data structures
4. Business logic only exists in backend API; frontend and mobile handle presentation and state only
5. Infrastructure is strictly hosting and deployment; AI should never put business logic in Nginx/Ubuntu/Podman
6. All real-time functionality must use Socket.IO server on backend and Socket.IO client on web and mobile
7. Backend must emit events immediately after any data changes, and clients must update automatically—no polling or alternative real-time methods allowed
8. Shared theme/design tokens exported in a TS file so that web and mobile have consistent styling
9. For complex components, create shared component files where possible, so web and mobile can reuse logic while applying platform-specific wrappers. Always put state, behaviour, and permission checks in the shared logic file. Only styling or platform-specific tweaks go in .web.tsx or .native.tsx