**Interview Prompt: Build and Design an RBAC System Using Next.js**

**Objective**

Your task is to design and implement a simple Role-Based Access Control (RBAC) system using **Next.js**. The system should allow for managing users, roles, and permissions while demonstrating clean design, modularity, and secure coding practices.

**Requirements**

**Core Functionality**

1. **User Authentication**

• Implement basic authentication (can use mock data or an in-memory database).

• Each user should have a unique identifier, username, password, and assigned roles.

2. **Roles and Permissions**

• Define a few roles (e.g., Admin, Editor, Viewer).

• Each role should have specific permissions (e.g., “create”, “read”, “update”, “delete”).

• Allow multiple roles to be assigned to a user.

3. **Access Control**

• Protect specific pages or API routes in the application based on roles/permissions.

• For example:

• Admins can access a page to manage roles and permissions.

• Editors can create and edit content.

• Viewers can only view content.

**Technical Requirements**

1. Use **Next.js** to:

• Create protected pages.

• Implement server-side or client-side authentication checks.

2. Use Next.js API routes to fetch user roles and permissions.

3. Demonstrate middleware or higher-order components (HOCs) for access control.

**Stretch Goals**

• Allow dynamic assignment of roles and permissions via a simple UI.

• Implement JWT or session-based authentication for added security.

• Add a simple database layer using SQLite, Prisma, or any mock database.

**Evaluation Criteria**

• **Correctness:** The system should enforce RBAC correctly based on roles and permissions.

• **Design:** The code should be modular, readable, and scalable.

• **Security:** Proper handling of authentication, authorization, and protected routes.

• **Efficiency:** Minimal re-renders and optimized data fetching.

• **UX:** A clean and simple UI for demonstration purposes.

**Example Scenarios to Test**

1. A Viewer tries to access a restricted Admin page (should be denied).

2. An Admin assigns a role to another user dynamically.

3. A route only accessible by Editors is tested.

**Tools You Can Use**

• **Framework:** Next.js (React-based)

• **Authentication:** NextAuth.js, JWT, or custom solutions

• **Styling:** TailwindCSS, Chakra UI, or plain CSS

• **Database (Optional):** SQLite, Prisma, or in-memory objects

Let me know if you want to adjust the complexity or scope!