Deployed Link : https://rbac-assignment-frontend.netlify.app/

Directory Structure:

The project is organized into several components to keep the code modular and maintainable. Here's the structure:

```
src/
├── components/
│   ├── Users/
│   │   ├── UserTable.jsx          # Displays the list of users
│   │   ├── UserFormModal.jsx      # Modal for adding/editing users
│   │   ├── UserActions.jsx        # Handles actions like editing and deleting users
│   │   └── UserContext.js         # React Context to manage user state
│   ├── Roles/
│   │   ├── RoleTable.jsx          # Displays the list of roles
│   │   ├── RoleFormModal.jsx      # Modal for adding/editing roles
│   │   └── RoleContext.js         # React Context to manage role state
│   ├── Permissions/
│   │   ├── PermissionGrid.jsx     # Displays the permission grid for roles
│   │   └── PermissionContext.js   # React Context to manage permissions state
│   └── Layout/
│       ├── Navbar.jsx             # Navbar component
├── pages/
│   ├── UsersPage.jsx              # Users page
│   ├── RolesPage.jsx              # Roles page
│   └── PermissionsPage.jsx        # Permissions page
├── utils/
│   ├── api.js                     # Utility functions for mock API calls
│   └── mockData.js                # Mock data for users, roles, and permissions
├── App.js                         # Main App component with routing and context
└── index.js                       # Entry point of the application
```

How the Application Works:

1. User Management :

Users can be viewed, added, edited, and deleted from the user table.

Users can be assigned roles, and their status can be toggled between active and inactive.

Search Users: A search input is provided to filter users by their attributes. The search query dynamically updates the displayed users list as you type.

2. Role Management :

Roles can be created, edited, and deleted.

Each role can have specific permissions assigned (e.g., Read, Write, Delete).

You can assign multiple roles to a user and manage their access levels via these roles.

3. Permissions Management :

Permissions are displayed in a dynamic grid format, where you can assign Read, Write, or Delete permissions to each role.

Permissions are managed for each role, and these roles are then assigned to users.

4. API Integration & Loading Indicators :

All CRUD operations (add, edit, delete) are simulated using mock API calls.

Each action displays a loader until the operation is complete.

After a successful operation, a success message appears indicating the result (e.g., "User added successfully").

Success messages automatically disappear after a timeout.

5. React Context for State Management :

UserContext, RoleContext, and PermissionContext are used to manage and share state across the application.

This ensures that the data remains consistent and accessible across different components.

6. Routing and Navigation :

The app uses React Router for navigation, with three main pages:

/users: Displays the user management interface.

/roles: Displays the role management interface.

/permissions: Displays the permission grid.

7. Responsive UI :

The layout uses Bootstrap to ensure the app is responsive and adapts to different screen sizes.

On mobile devices, the sidebar collapses into a hamburger menu, and the layout adjusts for smaller screens.

Future Improvements :

If this project were to be further developed, some features that could be added include:

Authentication & Authorization: Add login functionality to ensure only authorized users can access the dashboard.

Backend Integration: Implement a real API to manage users, roles, and permissions dynamically.

Real-Time Data Sync: Allow the app to update data in real-time via WebSockets or polling from a backend.

Bulk Actions: Enable bulk user actions, such as assigning roles or deleting users.

Detailed User Logs: Track changes made to users, roles, and permissions for auditing purposes.