« Le routage »

But du TP : Utilisez uniquement le hook useState.

Le point de départ est le tableau que vous avez dans le TP3.

1. Gestion de l'Authentification (Simulée)

Implémentez un mécanisme de gestion de l'état de l'utilisateur. L'idéal est d'utiliser le Context API de React pour centraliser l'état (Pas de bibliothèque externe pour le moment, mais vous pouvez vous inspirer de react router).

AuthContext : Créez un contexte qui maintient les informations suivantes :

isAuthenticated (boolean)
userRole (string: 'admin', 'user', 'guest', ou null)
login (fonction qui prend username et password et met à jour l'état)
logout (fonction)

Logique de Connexion : Dans la fonction login du contexte, vérifiez les identifiants par rapport à usersData. Si la connexion est réussie, stockez le rôle et mettez isAuthenticated à true.

Formulaire de Connexion : Dans PageConnexion, utilisez le contexte pour appeler la fonction login lors de la soumission du formulaire. En cas de succès, redirigez l'utilisateur vers la page /users (utilisez le hook useNavigate).

2. Implémentation de la Route Protégée

Créez un composant générique PrivateRoute (ou ProtectedRoleRoute) pour encadrer les routes nécessitant une connexion.

Ce composant doit :

Recevoir les rolesRequis (un tableau de chaînes de caractères, ex: ['admin', 'user']) en tant que prop.

Utiliser le AuthContext pour vérifier :

Si l'utilisateur est connecté (isAuthenticated). Si non, rediriger vers /login.
Si l'utilisateur a un des rôles requis (userRole est dans rolesRequis).

Redirection Conditionnelle :

Si l'utilisateur est connecté ET possède le bon rôle, affiche le composant enfant (<Outlet /> ou element).

Si l'utilisateur est connecté MAIS n'a pas le bon rôle, rediriger vers /forbidden.

3. React Router

Utilisez la bibliothèque « React router » pour implémenter la gestion de l'authentification.