React router (navigation multi pages)

Pour mettre en place un système de navigation entre plusieurs pages dans une application React, il est conseillé d'utiliser le routeur React comme React Router. Voici étape par étape comment intégrer React Router dans vos application existante (en partant de votre fichier App.js) :

1. Installer React Router

Dans vos projet React, ouvre un terminal à la racine et tapez :

```
bash
npm install react-router-dom@6
```

2. Créer les pages composants

Pour commencer, créez plusieurs fichiers composants pour représenter les différentes pages de vos application. Par exemple :

- `Home.js` (page d'accueil)
- `About.js` (page À propos)
- `Profile.js` (page profil, vous avez déjà un composant Profile que vous pouvez réutiliser)

3. Configurer React Router dans App.js

Modifie vos fichier `App.js` pour importer les composants du routeur et définir les routes :

Voici l'import :

```
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
```

On importe également ces pages qui sont dans un dossier pages pour être bien organisé,

```
import Home from './pages/Home';
import About from './pages/About';
```

Ensuite on met en place le router et les routes,

```
<Router>
  <nav>
    <Link to="/">Home</Link> | <Link to="/about">About</Link> | <Link to="/pr
  </nav>

  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
    <Route path="/profile" element={<Profile user={user} />} />
  </Routes>
</Router>
```

4. Détails supplémentaires

- Le composant `<Router>` englobe toute l'application ou au moins la zone où vous voulez activer la navigation.
- Le composant `<Routes>` contient les différentes routes possibles de votre app.
- Chaque `<Route>` a un `path` (URL) et un `element` qui est le composant à afficher.
- Les liens sont créés avec `<Link to="...">` et remplacent les balises `<a>`. Ils permettent la navigation sans rechargement complet de la page.

Une fois cette structure en place, vous pourrez naviguer entre vos pages en cliquant sur les liens, et React Router prendra en charge le rendu du composant correspondant à l'URL.Si besoin, il est possible d'ajouter plus de pages et routes, ou gérer des routes dynamiques (ex : `/profile/:id`).

Imaginons que l'on veut protéger une page pour un admin, alors faisons cela, on va ajouter un state admin dans le App.js,

```
function App() {

  const [isAdmin, setIsAdmin] = useState(true); // ou false pour non-admin
```

On crée un composant de protection appelé PrivateRoute.js dans le folder Components, et on et cela en place :

```
import { Navigate } from 'react-router-dom';

// Définition d'un composant fonctionnel nommé PrivateRoute qui prend en paramètres :
// - isAdmin : un booléen indiquant si l'utilisateur est administrateur
// - children : les composants enfants à rendre si la condition est remplie
function PrivateRoute({ isAdmin, children }) {
    // Si l'utilisateur n'est pas admin (isAdmin === false)
    if (!isAdmin) {
        // On redirige vers la page d'accueil ("/") en remplaçant l'historique,
        // ce qui évite à l'utilisateur de revenir en arrière vers la page protégée.
        return <Navigate to="/" replace />;
    }
    // Si l'utilisateur est admin (isAdmin === true),
    // le composant affiche directement les enfants passés en propriété.
    return children;
}
// Export du composant pour pouvoir l'importer dans d'autres fichiers React.
export default PrivateRoute;
```

Ensuite on utilise PrivateRoute dans le App.js, par exemple pour la route profile, on l'importe bien évidemment, et on modifie la route profile,

```jsx
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/about" element={<About />} />
  <Route
    path="/profile"
    element={
      <PrivateRoute isAdmin={isAdmin}>
        <Profile user={user} />
      </PrivateRoute>
    } />
</Routes>
```

Evidemment avec un système de connexion/inscription et définition d'un rôle user, avant la question, pour définir un rôle admin a un user cela se fait dans l'api backend de cette manière.

1. Au niveau backend (API)

- Lors de la création ou mise à jour d'un utilisateur, on attribue un champ rôle, souvent une propriété telle que `role` ou `isAdmin`.
- Exemple d'objet utilisateur stocké en base de données :

```json
{
  "id": 123,
  "username": "lisa",
  "email": "lisa@example.com",
  "role": "admin"  // ou "user"
}
```

- Le serveur doit vérifier ce rôle lors des requêtes protégées.

2. Lors de la connexion

- Le backend renvoie, avec le jeton d'authentification (par exemple JWT), les informations du profil utilisateur, incluant son rôle.

```json
{
  "sub": "123",
  "username": "lisa",
  "role": "admin",
  "iat": 1234567890,
  "exp": 1234567999
}
```

3. Côté React

- Dans le contexte d'authentification (`AuthContext` ou `AuthProvider`), on stocke ces informations utilisateur, dont le rôle.

```
const [user, setUser] = useState({
  id: 123,
  username: 'lisa',
  role: 'admin',
  token: '...'
});
```

Puis pour protéger des zones admin, on vérifie dans le composant ou la route :

```
if (user.role !== 'admin') {
  return <Navigate to="/" />;
}
```

4. Contrôle d’accès dans les routes privées

- Le composant `PrivateRoute` (ou `ProtectedRoute`) utilisera la vérification du rôle.

```
function PrivateRoute({ user, children }) {
  if (!user || user.role !== 'admin') {
    return <Navigate to="/" replace />;
  }
  return children;
}
```

En résumé : Le rôle admin est défini et géré côté backend avec les données utilisateur.L’information sur le rôle est envoyée à React à la connexion et stockée en état d’authentification.React utilise cette information pour conditionner l’accès aux routes et composants sensibles.