DOCUMENT INDIVIDUEL NADJIDE OMAR

Présentation d'une de mes parties de code (LOGIN)

Login :

Dans la première partie du code vous y verrez tous les imports nécessaire pour notre component Login.

Login

```
import React, { useState, useContext } from "react";
import { useNavigate } from "react-router-dom";
import { Box, Button, TextField, Typography, Paper } from "@mui/material";
import { AuthContext } from "../AuthContext";
```

Ensuite dans notre fonction Login () dans un premier temps il y a l'initialisation de plusieurs variables d'état.

Login

```
export default function Login() {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [errorMessage, setErrorMessage] = useState("");
  const navigate = useNavigate();
  const { setUser } = useContext(AuthContext);
```

Le hook useState est utilisé pour créer trois variables d'état : username, password et errorMessage. Les variables username et password sont probablement utilisées pour stocker les entrées de l'utilisateur à partir d'un formulaire de connexion, tandis que errorMessage est utilisée pour stocker tout message d'erreur qui pourrait survenir pendant le processus de connexion. Chaque variable d'état est associée à une fonction de mise à jour correspondante (setUsername, setPassword et setErrorMessage), qui est utilisée pour mettre à jour la valeur de la variable d'état.

Le hook useNavigate de react-router-dom est utilisé pour obtenir la fonction navigate, qui permet de changer de route.

Le hook useContext est utilisé pour accéder à AuthContext, qui fournit une fonction setUser. Cette fonction est utilisée pour mettre à jour l'utilisateur actuel dans l'état de l'application après une connexion réussie.

La fonction la plus importante de notre component login est « HandleSubmit »

Celle-ci va être déclenchée au moment du submit du formulaire. Elle commence par empêcher le comportement par défaut de soumission du formulaire avec event.preventDefault().

Ensuite, elle envoie une requête POST à l'API à l'adresse http://127.0.0.1:8000/login, en incluant le nom d'utilisateur et le mot de passe que l'utilisateur a entrés dans le corps de la requête.

La réponse de l'API est ensuite traitée. Si la réponse n'est pas OK (ce qui indique une erreur), la réponse est convertie en JSON, puis une erreur est lancée avec le détail de l'erreur. Si la réponse est OK, la réponse est convertie en JSON pour être utilisée dans l'étape suivante.

Dans le deuxième bloc .then, les données de la réponse sont utilisées pour mettre à jour l'application. Le jeton d'accès est stocké dans le stockage local du navigateur, l'utilisateur actuel est mis à jour avec la fonction setUser, et l'application est redirigée vers la route racine (/).

Si une erreur se produit à un moment donné pendant ce processus, elle est capturée dans le bloc catch. Le message d'erreur à afficher à l'utilisateur est mis à jour avec la fonction setErrorMessage et l'erreur est également enregistrée dans la console.

AuthContext est un objet de contexte qui est utilisé pour partager les informations de l'utilisateur actuel à travers plusieurs composants de l'application.

La partie return du component

```
Login

return (
    <Box
      display="flex"
      flexDirection="column"
      alignItems="center"
      justifyContent="center"
      minHeight="100vh"
    >
      <Paper elevation={3} sx={{ padding: 4, width: "400px" }}>
        <Typography variant="h5" sx={{ marginBottom: 2, textAlign: "center" }}>
          Login
        </Typography>
        {errorMessage && (
          <Typography variant="body2" color="error">
            {errorMessage}
          </Typography>
        )}
        <Box
          component="form"
          onSubmit={handleSubmit}
          sx={{ display: "flex", flexDirection: "column", gap: 2 }}
        >
          <TextField
            label="Username"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
            fullWidth
          />
          <TextField
            label="Password"
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            fullWidth
          />
          <Button
            type="submit"
            variant="contained"
            sx={{ marginTop: 2 }}
            disabled={!username || !password}
          >
            Login
          </Button>
        </Box>
      </Paper>
    </Box>
  );
```

Cette partie du code est le rendu du composant Login. Il s'agit de la partie de l'interface utilisateur qui sera affichée à l'écran lorsque ce composant est utilisé.

On commence par créer un conteneur principal avec l'élément Box. Ce conteneur est configuré pour afficher ses enfants en colonne, centrés à la fois horizontalement et verticalement, et pour prendre toute la hauteur de la fenêtre de visualisation.

À l'intérieur de ce conteneur, on a un élément Paper qui contient le formulaire de connexion. Cet élément a une ombre (elevation={3}), un padding, une largeur fixe et est centré dans le conteneur principal.

Le titre "Login" est affiché en utilisant un élément Typography. Si un message d'erreur est présent (c'est-à-dire si errorMessage est vrai), il est également affiché en utilisant un autre élément Typography, mais avec une couleur d'erreur.

Ensuite, le formulaire lui-même est créé avec un autre élément Box. Ce formulaire est configuré pour afficher ses enfants en colonne et pour avoir un espace entre eux. Lorsque le formulaire est soumis, la fonction handleSubmit est appelée.

Le formulaire contient deux champs de texte pour le nom d'utilisateur et le mot de passe. Chaque fois que l'utilisateur change la valeur de l'un de ces champs, la fonction correspondante (setUsername ou setPassword) est appelée avec la nouvelle valeur.

Enfin, il y a un bouton "Login" pour soumettre le formulaire. **Ce bouton est désactivé si le nom d'utilisateur ou le mot de passe est vide.**