ProjectManager | MERN

Autenticación de Usuarios

1. Importar en el componente `<Login/>` las siguientes dependencias y componentes:

```
import React, {useState} from 'react';
import {Link, useNavigate} from 'react-router-dom';
import { Alert } from '../components/Alert';
import clientAxios from '../config/clientAxios';
```

2. Establecer los estados para manejar las alertas

```
const [alert, setAlert] = useState({});
```

3. Crear la función (manejador) `handleShowAlert()`

```
  const handleShowAlert = (msg, time = true) => {
    setAlert({
      msg
    });

    if(time){
      setTimeout(() => {
        setAlert({});
      }, 3000);
    }
    reset()
  }
```

4. Implementar la condición correspondiente para mostrar las alertas en el cuerpo del componente

```
     {
        alert.msg && <Alert  {...alert}/>
      }
```

5. Implementar el custom hook `useForm()` y aplicar destructuring a `formValues`

```
const {formValues, handleInputChange, reset} = useForm({
  email : "",
  password : ""
})

const {email, password} = formValues;
```

6. Agregar los atributos que corresponden a cada input en el formulario. Ejemplo:

```
name="email"
value={email}
onChange={handleInputChange}
```

7. Crear e implementar la función (manejador) `handleSubmit` en el evento `onSubmit={handleSubmit}` del formulario:

```
 const handleSubmit = async (e) => {
  e.preventDefault();
}
```

8. Implementar las validaciones pertinentes

```
  if([email, password].includes("")) {
    handleShowAlert("Todos los campos son obligatorios");
    return null
  }
```

9. Verificar lo que devuelve en enpoint de autenticación

```
return res.status(200).json({
    ok : true,
    msg :'Usuario Logueado',
    user : {
        nombre : user.name,
        _id : user._id,
    },
    token : generateJWT({
        id: user._id
    })
})
```

10. Hacer el pedido a la API utilizando `clientAxios`

```
 try {

      const {data} = await clientAxios.post('/auth/login',{
        email,
        password
      })

      console.log(data)

    } catch (error) {
      console.log(error);
      handleShowAlert(error.response.data.msg)
    }
};
```

Implementación del "State Global" utilizando **Context**

Context provee una forma de pasar datos a través del árbol de componentes sin tener que pasar props manualmente en cada nivel. Ver documentación

1. Crear el archivo `src/context/AuthProvider.jsx` con el siguiente código inicial:

```
import {useState, createContext} from 'react';
import { createContext } from 'react'
import { clientAxios } from '../config/clientAxios';

const AuthContext = createContext();

const AuthProvider = ({children}) => {

    const [auth, setAuth] = useState({});

    return (
        <AuthContext.Provider
            value= {{
                auth,
                setAuth,
            }}
        >

            {children}

        </AuthContext.Provider>
    )
}

export {
    AuthProvider
}
```

```
export default AuthContext;
```

2. Importar e implementar el `<AuthProvider/>` en el componente `<App/>`

```
import { AuthProvider } from './context/AuthProvider';

return (
  <BrowserRouter>
  <AuthProvider>
    <Routes>
      <Route
        path="/"
        element={<AuthLayout/>}
      >
        <Route index element={<Login/>}/>
        {/* demás rutas públicas */}
      </Route>

    </Routes>
    </AuthProvider>
  </BrowserRouter>
)
```

3. Crear el custom hook `src/hooks/useAuth` que a traves de `useContext()` de REACT, devolverá el estado global de la aplicación:

```
import {useContext} from 'react';
import AuthContext from '../context/AuthProvider';

const useAuth = () => {
    return useContext(AuthContext)
}

export default useAuth
```

Nota: Este custom hook obtiene el estado global de `AuthContext`.

4. Ahora podemos setear el *state global* de autentiación con los datos traidos de la API, haciendo utilizando el hook `useAuth()` en el componente `<Login/>` y guardar el token en `sessionStorage`

```
const {setAuth} =useAuth();
setAuth(data.user);
sessionStorage.setItem('token',data.token);
```

5. Una vez guardado los datos del usuario en el *state global* limpiamos las alertas y redirigimos al apartado privado de la aplicación, no sin antes de crear las correspondientes rutas (ver siguiente paso)

```
handleShowAlert({});
//navigate('/projects')
```

6. Crear en el backend un *middleware* que verifique si exite un token en la petición.

```
const { verify } = require("jsonwebtoken");
const User = require("../models/User");

module.exports = async (req, res, next) => {

  try {

        if(!req.headers.authorization){
            throw createHttpError(401,"Se requiere un token");
        }

        const token = req.headers.authorization;
        const decoded = verify(token, process.env.JWT_SECRET);

        req.user = await User.findById(decoded.id).select("name");

        next()

      } catch (error) {

          return errorResponse(res,error, "CHECK-TOKEN")

      }
  }
};
```

7. Crear un nuevo estado en el componente `<AuthProvider/>` que servirá para detectar cuando se está haciendo la consulta a la API

```
const [loading, setLoading] = useState(true);
```

*Nota: Incluir **loading** en objeto que va en props value*

8. Completar el componente `<AuthProvider/>` implementando en un `useEffect()` un pedido a la API para autenticar el usuario a partir del token

```
useEffect(() => {

    const authUser = async () => {
        const token = sessionStorage.getItem('token');
        if(!token){
            setLoading(false)
            return null
        }

        const config = {
            headers : {
                "Content-Type" : "application/json",
                Authorization : token
            }
        }

        try {
            const {data} = await clientAxios.get('/users/profile',config);
            setAuth(data.user);

        } catch (error) {
            console.error(error.response?.data);
            sessionStorage.removeItem('token')
        } finally {
            setLoading(false)

        }
    }

    authUser()

}, []);
```

Rutas protegidas

1. Crear el layout `<ProtectedLayout/>` que se encargará de mostrar todas las vistas de acceso restrigido, siempre y cuando haya un usuario autenticado.

```
import React from "react";
import { Navigate, Outlet } from "react-router-dom";
import useAuth from "../hooks/useAuth";

export const ProtectedLayout = () => {
  const { auth, loading } = useAuth();

  if (loading) {
    return <p>Cargando...</p>;
  }

  return (
```

```
    <>
      {auth._id ? (
        <main className="container mx-auto mt-5 md:mt-10 p-5 md:flex md:justify-
center">
          <Outlet />
        </main>
      ) : (
        <Navigate to="/" />
      )}
    </>
  );
};
```

2. Crear el componente `<Projects/>` en la carpeta `pages`

```
import React from 'react'

export const Projects = () => {
  return (
    <div>Projects</div>
  )
}
```

3. Implementar en el componente `<App/>` las rutas privadas que hagan uso del componente `<ProtectedLayout>`

```
<Route path="/projects" element={<ProtectedLayout />}>
  <Route index element={<Projects />} />
</Route>
```