

# React, Node et MySql Login (Partie 5)

## Stockage de l'info de login côté React (Version 2)

Avant de créer un context qui va nous servir à stocker les infos de login, vous pouvez tester à nouveau l'appli en ajoutant de l'affichage conditionnel

Si l'on est connecté on affiche le bouton logout, sinon le formulaire de connexion.

```
src > screen > LoginScreen.jsx > LoginScreen
33
34   return (
35     <>
36       {auth && (
37         <button className="btn btn-primary" onClick={handleLogout}>
38           Logout
39         </button>
40       )}
41       {!auth && (
42         <form onSubmit={handleSubmit}>
43           <div className="mb-3">
```

Tests :

localhost:3000

Email address

Pincode

Login

localhost:3000

Logout

Application

Manifest

Service Workers

Storage

Local Storage

http://localhost:3000/

Application

Manifest

Service Workers

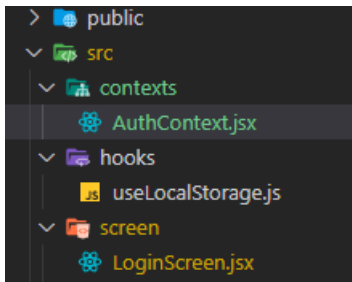
Storage

Local Storage

http://localhost:3000/

Key	Value
auth	{"id":1,"email":"bedulaurent..."}

## Étape 10 : Création d'un contexte pour stocker l'info de login.



```
src > contexts > AuthContext.jsx > ...
1  import { createContext, useState, useMemo } from "react";
2
3  const AuthContext = createContext();
4
5  const AuthProvider = ({ children }) => {
6
7      const [auth, setAuth] = useState(null);
8      const authMemo = useMemo(() => ({ auth, setAuth }), [auth]);
9
10     return (
11         <AuthContext.Provider value={authMemo}>
12             {children}
13         </AuthContext.Provider>
14     );
15 };
16
17 export { AuthContext, AuthProvider };
```

Ln 3 : Création du context

Ln 5 : Déclaration du provider

Ln 7 : Création du state pour stocker les infos de login

Ln 8 : Création d'un memo pour améliorer les performances

(voir : <https://fr.reactjs.org/docs/hooks-reference.html#usememo> )

Ln 10 à 14 : Le composant retourne le provider du context.

Ln 17 : on exporte le context et son provider dans un objet

Pour le contexte, allez voir <https://fr.reactjs.org/docs/context.html>

## Étape 11 : Utilisation du contexte

On englobe l'application avec le provider du contexte

```
src > App.js > App
1  import "bootstrap/dist/css/bootstrap.min.css";
2  import "bootstrap/dist/js/bootstrap.bundle.js";
3  import "./App.css";
4  import LoginScreen from "../screen/LoginScreen";
5  import { AuthProvider } from "../contexts/AuthContext";
6
7  function App() {
8      return (
9          <AuthProvider>
10             <div className="App container pt-5">
11                 <LoginScreen/>
12             </div>
13         </AuthProvider>
14     );
15 }
16
17 export default App;
```

On importe le contexte et on remplace le hook `useLocalStorage` par un `useContext` initialisé avec notre `AuthContexte` dans `LoginScreen`

```
src > screen > LoginScreen.jsx > LoginScreen
1 | import { useContext } from "react";
2 | import useLocalStorage from "../hooks/useLocalStorage";
3 | import { AuthContext } from "../contexts/AuthContext";
4 |
5 | function LoginScreen(props) {
6 |   //const [auth, setAuth] = useLocalStorage("auth", null);
7 |   const {auth, setAuth} = useContext(AuthContext);
8 |
9 |   const handleSubmit = (e) => {
10 |     e.preventDefault();
```

Rien d'autre à faire, en testant on obtient le même comportement qu'avec `useLocalStorage` sauf que l'info n'est plus stockée dans le navigateur mais directement dans le contexte. Dans ce screen, l'utilisateur est bien authentifié car le bouton Logout est affiché, mais rien n'est stocké en `localStorage`



Logout

