



Front End Developer

Técnicas de Programação Avançada para a Web

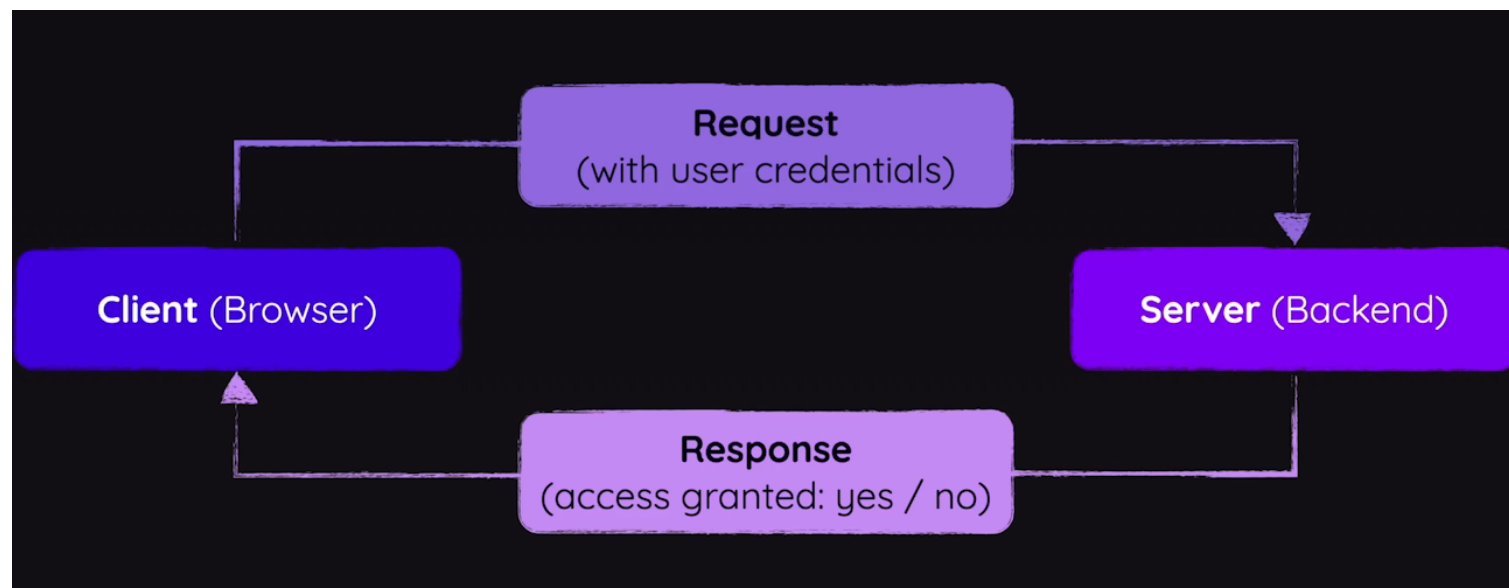
Sara Monteiro

sara.monteiro.prt@msft.cesae.pt

Autenticação – para que serve?

Iremos finalizar o estudo do módulo com as funcionalidades de Autenticação. Para que serve então?

➔ Para proteger conteúdo reservado a certos perfis de users



Actualização do Conteúdo de UI baseado na Autenticação

Quando recebemos do pedido ao backend a indicação que o login foi feito com sucesso podemos armazenar esses dados no `localStorage` e usá-lo para esconder ou mostrar conteúdos na nossa aplicação. Por exemplo, o `logout` só faz sentido mostrar se o user estiver autenticado, e o `login` só deve aparecer quando não está autenticado.

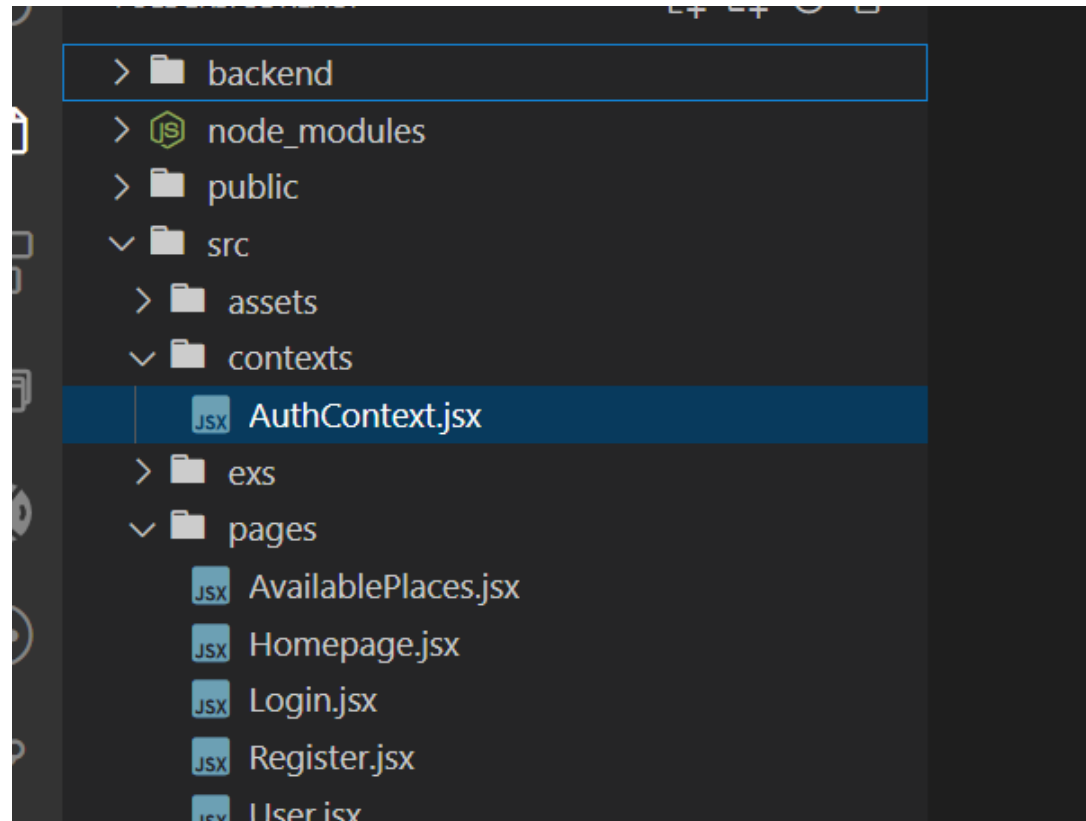
Mas estes dados precisam de ser renderizados em toda a aplicação sempre que há uma mudança de rota. Para tal usaremos o recurso `ContextAPI`.

Autenticação: ContextAPI

- recurso do React que permite a partilha de estados e funções entre componentes sem a necessidade de "prop drilling" (passar props manualmente de um componente pai para vários níveis de componentes filhos).
- para uso do ContextAPI temos que criar o um ficheiro de contexto em que será envolvida toda a app e os ficheiros para proteger as rotas conforme as especificidades. Podemos também usar para proteger certos componentes

ContextAPI: criação de Contexto

Para organizar devidamente o código criaremos dentro da raiz do projecto um folder de *contexts*.



ContextAPI: criação de Contexto

O nosso contexto irá definir as funções relacionadas com autenticação e definir que os filhos encapsulados com a mesma herdam estes dados.

Ao fazer login, o local storage irá guardar o role para que possamos com ele distinguir depois as autorizações.

```
contexts > Jsx AuthContext.jsx > [AuthProvider]
import { createContext, useState, useEffect } from "react";

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);

  useEffect(() => {
    const storedRole = localStorage.getItem("role");

    if (storedRole) {
      setUser({ role: storedRole });
    }
  }, []);
}
```

ContextAPI: função de Login

A função de login é registada no ficheiro de contexto e chamada na página de login com os dados colocado pelo user.

```
const login = async (authData) => {  
  const response = await fetch("http://localhost:3000/login", {  
    method: "POST",  
    headers: {  
      "Content-Type": "application/json",  
    },  
    body: JSON.stringify(authData),  
  });  
  
  if (!response.ok) {  
    throw new Error("Failed to authenticate");  
  }  
  
  const data = await response.json();  
  localStorage.setItem("role", data.role);  
  setUser({ role: data.role });  
  return true;  
};
```

ContextAPI: função de Login

```
export default function Login() {  
  const { login } = useContext(AuthContext);  
  const [enteredValues, setEnteredValues] = useState({ email: '', password: '' });  
  const navigate = useNavigate();  
  
  async function handleSubmit(event) {  
    event.preventDefault();  
    const success = await login(enteredValues);  
    if (success) {  
      navigate('/');  
    } else {  
      alert('Email ou senha inválidos.');    }  
  }  
}
```


ContextAPI: encapsular as rotas no contexto

O nosso contexto irá definir as funções relacionadas com autenticação e definir que os filhos encapsulados com a mesma herdam estes dados.

```
✓ const router = createBrowserRouter([
✓   {path: '/',
✓     element: <LayoutMaster/>,
✓     errorElement: <ErrorPage/>,
✓     children:[
        {path: '/', element: <HomePage/>},
        {path: '/shopping-list', element: <ShoppingList/>},
        {path: '/contacts/:name', element: <Contacts/>},
        {path: '/places', element: <AvailablePlaces/>},
        {path: '/SWars', element: <StarWars/>},
      ]
    },
  ],);

✓ return (
✓   <AuthProvider>
    <RouterProvider router={router} />
  </AuthProvider>
);
```

Actualização do Conteúdo de UI baseado na Autenticação

```
import { Link } from "react-router-dom"
import { useContext } from 'react'
import { AuthContext } from '../contexts/AuthContext'

export default function HomePage(){
  const { user, logout } = useContext(AuthContext);

  return <div><h1>As minhas funcionalidades</h1>
  <ul>
    <li><Link to='/user/sara'>Sara</Link></li>
    <li><Link to='/AvailablePlaces'>Available Places</Link></li>
    {!user ? (
      <li><Link to="/login">Login</Link></li>
    ) : (
      <li><button onClick={logout}>Logout</button></li>
    )}
  </ul>
</div>
}
```

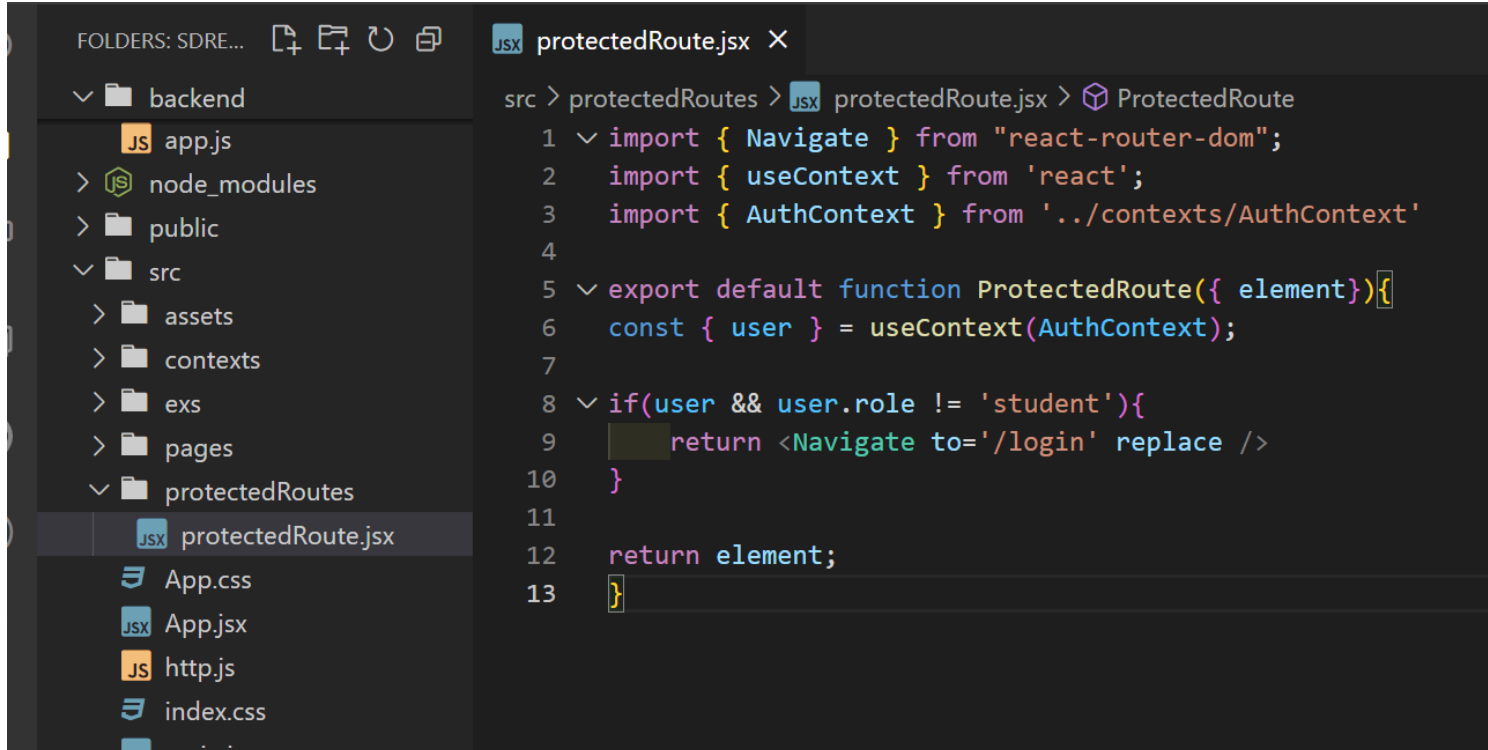
Protecção de Rotas

Neste momento já sabemos proteger componentes ou conteúdos para utilizadores autenticados. E se quisermos proteger toda uma funcionalidade para um determinado role, por exemplo?

Ao fazer `console.log(user)` verificamos que contém o role do user, logo podemos proteger rotas conforme o role.

Protecção de Rotas

Vamos então criar uma função que nos valide de acordo com o que queremos e aplicar a mesma nas rotas. Por exemplo, caso queira que a available places apareça apenas a estudantes:



```
FOLDERS: SDRE...  
  backend  
    app.js  
    node_modules  
    public  
    src  
      assets  
      contexts  
      exs  
      pages  
      protectedRoutes  
        protectedRoute.jsx  
    App.css  
    App.jsx  
    http.js  
    index.css  
    main.jsx  
  protectedRoute.jsx X  
src > protectedRoutes > protectedRoute.jsx > ProtectedRoute  
1  import { Navigate } from "react-router-dom";  
2  import { useContext } from 'react';  
3  import { AuthContext } from '../contexts/AuthContext'  
4  
5  export default function ProtectedRoute({ element }) {  
6    const { user } = useContext(AuthContext);  
7  
8    if (user && user.role !== 'student') {  
9      return <Navigate to="/login" replace />  
10    }  
11  
12    return element;  
13  }
```

Protecção de Rotas

```
✓ const router = createBrowserRouter([
  {path: '/', element: <HomePage/>},
  {path: '/user/:name', element: <User/> },
  {path: '/AvailablePlaces', element:<ProtectedRoute element={<AvailablePlaces/>}/> },
  {path: '/Login', element: <Login/> },
  {path: '/Register', element: <Signup/> },
])

✓ return (
```

Rotas com token de validação

Por segurança, hoje em dia muitas apis requerem validação de token para serem consultadas.

Caso isso aconteça a consultar APIs externas ou internas, no pedido deve ser enviado o token ao Backend.

```
export async function updatePlacesData(userPlaces){  
  const response = await fetch('http://localhost:3000/user-places',  
    {  
      method: 'PUT',  
      body: JSON.stringify({places: userPlaces}),  
      headers: {  
        "Content-Type": "application/json",  
        "Authorization": 'Bearer ' + token  
      }  
    })  
}
```

Recursos

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- <https://react.dev/>