1. Atividades Práticas - Autenticação com React

Nesta atividade prática, vamos usar o hook useContext para gerenciar a autenticação de usuários em um contexto de React. O objetivo é simular um sistema de login com informações sobre nome de usuário, senha e um estado que verifica se o usuário está logado.

1.1 Criando o projeto e iniciando o servidor local

Crie um projeto chamado **autenticacao**, digitando o seguinte comando no terminal:

```
npm create vite@latest autenticacao -- --template react
```

Este processo de configuração inicial do projeto leva alguns segundos. Ao terminar, o Vite repassa instruções para que você termine de instalar as dependências do seu projeto. Desta forma, digite o seguinte comando para **entrar no diretório** recém criado do nosso projeto:

```
cd autenticacao
```

Em seguida, **instale as dependências** necessárias do projeto executando no terminal o seguinte comando:

```
npm install
```

Até aqui, você criou um projeto React usando o Vite e adicionou todas as dependências ao projeto. Você pode, agora, **abrir a aplicação no editor Visual Studio Code**. Para isso, dentro do diretório do projeto digite o seguinte comando no terminal:

```
code .
```

Após a execução deste comando, o Visual Studio Code deverá abrir com a pasta raiz do seu projeto sendo acessada. Em seguida, você inicializará um servidor local e executará o projeto em seu navegador. Digite a seguinte linha de comando no terminal:

```
npm run dev
```

Ao executar esse script, você iniciará um servidor local de desenvolvimento, executará o código do projeto, iniciará um observador que detecta alterações no código e abrirá o projeto em um navegador web. Ele irá rodar a aplicação em modo desenvolvimento em http://localhost:5173/.

1.2 Passo 1: Criar o Contexto de Autenticação

Nosso objetivo é criar um contexto de autenticação com as informações do usuário, senha e status de login. Em seguida, vamos utilizar o **useContext** para acessar e modificar essas informações em componentes filhos. Finalmente, vamos implementar um formulário de login que simule a autenticação de um usuário com base em um nome de usuário e senha.

Vamos começar criando o contexto que armazenará informações de autenticação. Assim, crie um diretório chamado **context** dentro do diretório **src**. Em seguida, neste dentro do diretório **context**, crie um arquivo chamado **AuthContext.jsx**, com o código conforme quadro abaixo:

```
import { createContext } from 'react';
// Criando o contexto de autenticação
const AuthContext = createContext();
export default AuthContext;
```

Observe que o componente **AuthContext** define o contexto que armazenará as informações de autenticação, neste caso, o estado do nome, senha e status de login do usuário.

Em seguida, vamos criar o componente que será responsável por definir e exportar o **provider**, que irá gerenciar o estado de autenticação e as funções de login e logout. Este componente usará o **AuthContext** definido anteriormente. Dessa forma, dentro do diretório **context**, crie um arquivo chamado **AuthProvider.jsx**, com o código conforme quadro abaixo:

```
import { useState } from "react";
import AuthContext from "./AuthContext"; // Importa o AuthContext
// Provider para gerenciar o estado de autenticação
function AuthProvider({ children }) {
  const [authState, setAuthState] = useState({
    name: "",
    password: "",
    isLoggedIn: false,
  });
  // Função de login
  const login = (name, password) => {
    setAuthState({
      name,
      password,
      isLoggedIn: true,
    });
  };
```

```
  // Função de logout
  const logout = () => {
    setAuthState({
      name: "",
      password: "",
      isLoggedIn: false,
    });
  };
  return (
    <AuthContext.Provider value={{ authState, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}
export default AuthProvider;
```

Neste exemplo:

- **AuthProvider:** Garante que os componentes filhos possam acessar e modificar o estado de autenticação (**authState**), que contém o nome, senha e se o usuário está logado.
- **login e logout:** Funções para logar e deslogar o usuário, atualizando o estado de autenticação.

1.3 Passo 2: Criar o Componente de Login

Agora, vamos criar um formulário de login onde o usuário poderá inserir seu nome e senha, e clicar no botão para fazer login. Primeiro, crie um diretório chamado **components** dentro do diretório **src**. Em seguida, dentro do diretório **components**, crie um arquivo chamado **LoginPage.jsx** com o seguinte código abaixo:

```
import { useContext, useState } from "react";
import AuthContext from "../context/AuthContext"; // Importa o AuthContext

function LoginPage() {
  const { authState, login, logout } = useContext(AuthContext);
  const [name, setName] = useState("");
  const [password, setPassword] = useState("");
```

```
  const handleLogin = () => {
    if (name && password) {
      login(name, password);
      setName("");
      setPassword("");
    } else {
      alert("Por favor, preencha todos os campos.");
    }
  };
  return (
    <div>
      {authState.isLoggedIn ? (
        <div>
          <h2>Bem-vindo, {authState.name}!</h2>
          <button onClick={logout}>Logout</button>
        </div>
      ) : (
        <div>
          <h2>Login</h2>
          <input
            type="text"
            placeholder="Nome de usuário"
            value={name}
            onChange={(e) => setName(e.target.value)}
          />
          <input
            type="password"
            placeholder="Senha"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
          <button onClick={handleLogin}>Login</button>
        </div>
      )}
    </div>
  );
}

export default LoginPage;
```

Observe que neste código:

- **Estado Local (name e password):** Controla os campos de nome de usuário e senha.
- **useContext(AuthContext):** Consome o contexto de autenticação, acessando o estado (**authState**) e as funções **login** e **logout**.
- **Lógica de Login:** Se o nome e a senha estiverem preenchidos, a função **login** é chamada e atualiza o estado de autenticação no contexto.
- **Renderização Condicional:** Se o usuário estiver logado (**authState.isLoggedIn**), uma mensagem de boas-vindas é exibida com o botão de logout. Caso contrário, o formulário de login é exibido.

1.4 Passo 3: Criar o Componente de Cabeçalho

Agora, vamos criar um componente **Header** que mostra se o usuário está logado ou não, e exibe seu nome quando autenticado. Assim, dentro do diretório **components**, crie um arquivo chamado **Header.jsx** com o seguinte código abaixo:

```
import { useContext } from "react";
import AuthContext from "../context/AuthContext"; // Importa o AuthContext
function Header() {
  const { authState } = useContext(AuthContext);
  return (
    <header style={{ backgroundColor: "#f1f1f1", padding: "10px" }}>
      <h1>Minha Aplicação</h1>
      {authState.isLoggedIn ? (
        <p>Usuário logado: {authState.name}</p>
      ) : (
        <p>Nenhum usuário logado</p>
      )}
    </header>
  );
}
export default Header;
```

Observe neste exemplo que:

- O **Header** consome o estado de autenticação para mostrar se o usuário está logado e, se estiver, exibir o nome do usuário logado.
- Usamos **useContext** para obter o valor atual do **authState**.

1.5 Passo 4: Integrar Tudo no Componente Principal

Agora, vamos integrar o **AuthProvider** no componente principal (**App**) e envolver a aplicação para garantir que todos os componentes possam acessar o contexto de autenticação. Assim, altere o arquivo **App.jsx** com o código abaixo:

```
import AuthProvider from "./context/AuthProvider"; // Importa o AuthProvider
import Header from "./components/Header";
import LoginPage from "./components/LoginPage";
function App() {
  return (
    <AuthProvider>
      <Header />
      <LoginPage />
    </AuthProvider>
  );
}
export default App;
```

Neste código, observe que **AuthProvider** envolve os componentes **Header** e **LoginPage**, fornecendo o contexto de autenticação para que esses componentes possam acessar o estado global de autenticação (**authState**), e as funções de **login** e **logout**.

1.6 Passo 5: Testando a Aplicação

Ao iniciar a aplicação, o componente LoginPage exibirá o formulário de login e o Header mostrará que nenhum usuário está logado. Veja o resultado no navegador, conforme abaixo:

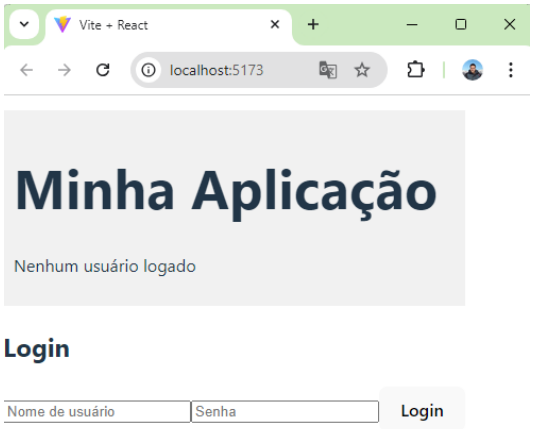

Figura 1: Exemplo da nossa aplicação sem usuário autenticado

Em seguida, insira um nome de usuário e uma senha e clique em "Login". Veja que após o login, o **Header** exibirá o nome do usuário logado, e o componente **LoginPage** mostrará uma mensagem de boas-vindas e um botão de logout.

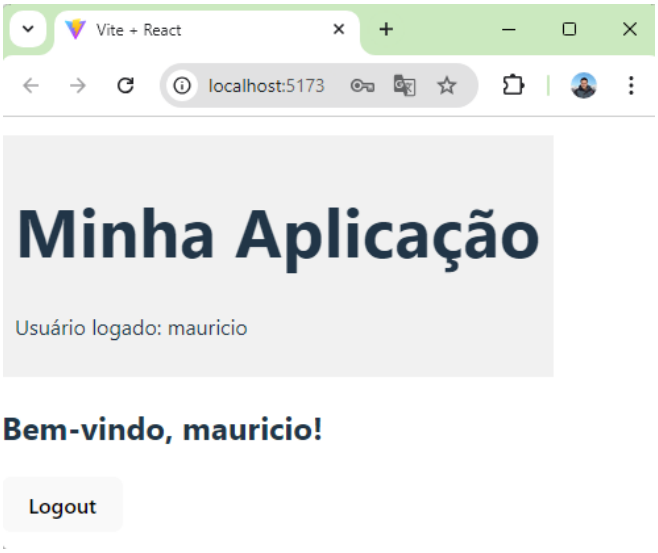

Figura 2: Exemplo da nossa aplicação com o usuário “mauricio” autenticado

Para finalizar, clique em "Logout" para encerrar a sessão. Veja que voltamos para a tela inicial mostrada na Figura 1.