1. Atividades Práticas - Rotas Públicas e Privadas

Nesta atividade prática, vamos criar um exemplo simples passo a passo para ilustrar como podemos trabalhar com rotas públicas e privadas com React Router.

1.1 Criando o projeto, instalando dependências e iniciando o servidor local

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

```
npm create vite@latest my-routes-auth -- --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 my-routes-auth
```

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

```
npm install
```

Antes de começar a usar o **React Router** em seus projetos, você precisa instalar essa biblioteca no seu projeto React:

```
npm install react-router-dom
```

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 .
```

Em seguida, você inicializará um servidor local e executará o projeto em seu navegador:

```
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: Criando a estrutura de diretórios

Primeiro, precisamos criar a estrutura de diretórios e arquivos de nossa aplicação. Crie o seu projeto conforme imagem abaixo:

```
src/
|-- components/
|    |-- Dashboard.jsx
|    |-- Login.jsx
|-- layouts/
|    |-- RootLayout.jsx
|-- pages/
|    |-- About.jsx
|    |-- Home.jsx
|-- utils/
|    |-- auth.jsx
|-- App.jsx
|-- main.jsx
|-- styles.css
```

Observe que no diretório "src", há as seguintes subpastas: "components", que contém os arquivos "Dashboard.jsx" e "Login.jsx"; "layouts", com o arquivo "RootLayout.jsx"; "pages", contendo "About.jsx" e "Home.jsx"; e "utils", onde está o arquivo "auth.jsx". Fora das pastas, estão os arquivos "App.jsx", "main.jsx" e "styles.css".

1.3 Passo 2: Configuração Básica da Aplicação

Neste passo, vamos iniciar a configuração básica da nossa aplicação. Desta forma, altere o arquivo **main.jsx** com o seguinte código:

```jsx
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from './App';
import './styles.css';  // Adiciona estilos globais
createRoot(document.getElementById("root")).render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

1.4 Passo 3: Criação do Arquivo de Autenticação

Agora, vamos criar a lógica de autenticação. O estado de autenticação será armazenado apenas em memória usando o **useState** e o React **Context API**.

Desta forma, altere o arquivo **auth.jsx**:

```
import React, { createContext, useContext, useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';

// Criação do contexto de autenticação
const AuthContext = createContext();

// Provedor de autenticação
export function AuthProvider({ children }) {
  const [isAuthenticated, setIsAuthenticated] = useState(false); // Estado de
autenticação (em memória)
  const login = () => {
    setIsAuthenticated(true); // Simula o login
  };

  const logout = () => {
    setIsAuthenticated(false); // Simula o logout
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}
// Hook personalizado para acessar o contexto de autenticação
export function useAuth() {
  return useContext(AuthContext);
}
```

```
// Componente para proteger as rotas privadas
export function RequireAuth({ children }) {
  const { isAuthenticated } = useAuth();
  const navigate = useNavigate();

  useEffect(() => {
    if (!isAuthenticated) {
      navigate('/login'); // Redireciona para login se não estiver autenticado
    }
  }, [isAuthenticated, navigate]); // UseEffect irá rodar quando
isAuthenticated ou navigate mudar

  // Retorna null enquanto o redirecionamento não foi feito
  if (!isAuthenticated) {
    return null;
  }

  // Renderiza os filhos se estiver autenticado
  return children;
}
```

Observe neste código:

- **AuthProvider:** Gerencia o estado de autenticação em memória com **useState**. O estado **isAuthenticated** será **true** ou **false**, dependendo se o usuário está logado ou não.
- **useAuth:** Um hook personalizado para acessar o contexto de autenticação.
- **RequireAuth:** Componente que protege as rotas privadas. Se o usuário não estiver logado, ele será redirecionado para a página de login.
- **useEffect:**
  - O **navigate('/login')** está dentro de um **useEffect** para garantir que o redirecionamento ocorra após a renderização do componente, evitando o erro.
  - O **useEffect** depende de duas variáveis: **isAuthenticated** (para monitorar o estado de autenticação) e **navigate** (para garantir que o efeito seja atualizado corretamente se **navigate** mudar).

1.5 Passo 4: Configuração do App com Autenticação e Rotas

Agora, configuraremos a aplicação no arquivo **App.jsx** para incluir as rotas públicas e privadas, usando o contexto de autenticação.

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import RootLayout from "./layouts/RootLayout";
import Home from "./pages/Home";
import About from "./pages/About";
import Login from "./components/Login";
import Dashboard from "./components/Dashboard";
import { AuthProvider, RequireAuth } from "./utils/auth";

const router = createBrowserRouter([
  {
    path: "/",
    element: <RootLayout />,
    children: [
      { path: "/", element: <Home /> },
      { path: "/about", element: <About /> },
      { path: "/login", element: <Login /> },
      {
        path: "/dashboard",
        element: (
          <RequireAuth>
            <Dashboard />
          </RequireAuth>
        ),
      },
    ],
  },
]);
function App() {
  return (
    <AuthProvider>
      <RouterProvider router={router} />
    </AuthProvider>
  );
}
export default App;
```

Observe neste código:

- **AuthProvider:** O **AuthProvider** envolve toda a aplicação para fornecer o estado de autenticação globalmente.
- **RequireAuth:** Esse componente é usado para proteger a rota **/dashboard**. Se o usuário não estiver autenticado, ele será redirecionado para a página de login.

1.6 Passo 5: Componente de Layout Aplicação

Neste passo, configuraremos o componente que terá o layout básico de nossa aplicação. Assim, altere o arquivo **RootLayout.jsx**:

```
import { Outlet, Link } from 'react-router-dom';
import { useAuth } from '../utils/auth';
function RootLayout() {
  const { isAuthenticated, logout } = useAuth();
  return (
    <div>
      <header>
        <nav>
          <ul>
            <li><Link to="/">Home</Link></li>
            <li><Link to="/about">About</Link></li>
            {isAuthenticated ? (
              <>
                <li><Link to="/dashboard">Dashboard</Link></li>
                <li><button className='login-button'
onClick={logout}>Logout</button></li>
              </>
            ) : (
              <li><Link to="/login">Login</Link></li>
            )}
          </ul>
        </nav>
      </header>
      <main>
        <Outlet />
      </main>
    </div>
  );
}
export default RootLayout;
```

1.7 Passo 6: Páginas Públicas

Neste passo, configuraremos as páginas públicas de nossa aplicação. Assim, altere o arquivo **Home.jsx**:

```
function Home() {
  return (
    <div className="home-container">
      <h1>Home Page</h1>
      <p>Bem-vindo à nossa aplicação de autenticação em memória!</p>
    </div>
  );
}
export default Home;
```

Em seguida, altere o arquivo **About.jsx**:

```
function About() {
  return (
    <div className="about-container">
      <h1>About Page</h1>
      <p>Esta é uma aplicação React Router com autenticação.</p>
    </div>
  );
}
export default About;
```

1.8 Passo 7: Página de Login

A página de Login permite que o usuário se autentique em nossa aplicação. Quando o botão de login é clicado, o estado de autenticação é atualizado no contexto, e o usuário é redirecionado para o componente **Dashboard**.

Desta forma, altere o arquivo **Login.jsx**:

```
import { useNavigate } from 'react-router-dom';
import { useAuth } from '../utils/auth';
```

```
function Login() {
  const { login } = useAuth();
  const navigate = useNavigate();
  const handleLogin = () => {
    login();
    navigate('/dashboard');  // Redireciona para o Dashboard após login
  };
  return (
    <div>
      <h1>Login Page</h1>
      <button className="button" onClick={handleLogin}>Login</button>
    </div>
  );
}
export default Login;
```

1.9 Passo 8: Página Protegida

A página Dashboard é uma rota protegida que só pode ser acessada quando o usuário estiver logado. Desta forma, altere o arquivo **Dashboard.jsx**:

```
function Dashboard() {
  return (
    <div>
      <h1>Dashboard</h1>
      <p>Bem-vindo ao painel de controle. Esta é uma página protegida.</p>
    </div>
  );
}
export default Dashboard;
```

1.10 Passo 9: Estilos Globais

Finalmente, vamos definir um estilo básico para a aplicação. Desta forma, altere o arquivo **styles.css**:

```css
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f5f5f5;
}

header {
    background-color: #333;
    padding: 10px 0;
}

header nav ul {
    display: flex;
    justify-content: center;
    list-style: none;
    padding: 0;
}

header nav ul li {
    margin: 0 20px;
}

header nav ul li a,
header nav ul li button {
    color: white;
    text-decoration: none;
    font-size: 18px;
    background: none;
    border: none;
    cursor: pointer;
}

header nav ul li button:hover,
header nav ul li a:hover {
    color: #ffd700;
}
```

```css
.button {
    background-color: #4CAF50;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

.button:hover {
    background-color: #45a049;
}

.login-button {
    color: white;
    cursor: pointer;
}

main {
    margin: 40px auto;
    padding: 20px;
    width: 80%;
    max-width: 800px;
    background-color: white;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

Este código define o estilo básico de nossa aplicação utilizando CSS. O corpo da página (body) utiliza a fonte Arial, remove as margens e o padding padrão, e aplica uma cor de fundo cinza claro. O cabeçalho (header) tem um fundo escuro, e o menu de navegação dentro dele utiliza flexbox para centralizar os itens, removendo a lista padrão. Os links e botões no menu são brancos, sem sublinhado, e mudam para uma cor dourada ao passar o mouse. Há também um estilo para botões com uma cor verde de fundo e bordas arredondadas, que escurece levemente ao ser sobreposto. O conteúdo principal (main) é centralizado, com margens e padding ajustados, e possui uma área delimitada com sombra para destacar o conteúdo.

1.11 Passo 10: Testar a aplicação

Agora, vamos testar nossa aplicação. Perceba quando fizemos o primeiro acesso que podemos visualizar todas as páginas públicas.

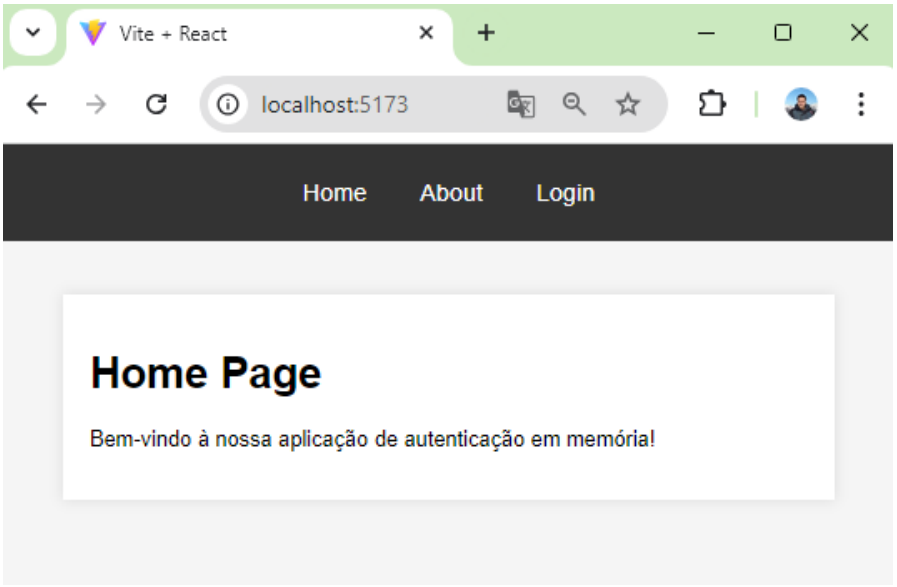

Figura 1: Página inicial da aplicação

Agora, você pode acessar a tela de Login e clicar no botão “Login” para realizar sua autenticação na aplicação e ter acesso às páginas privadas.

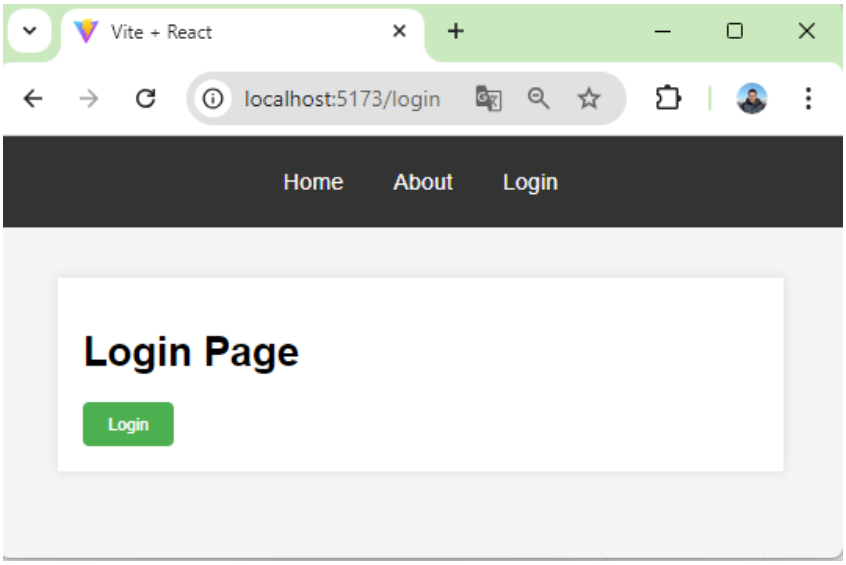

Figura 2: Página de Login

Após fazer sua autenticação, você terá acesso às páginas privadas.

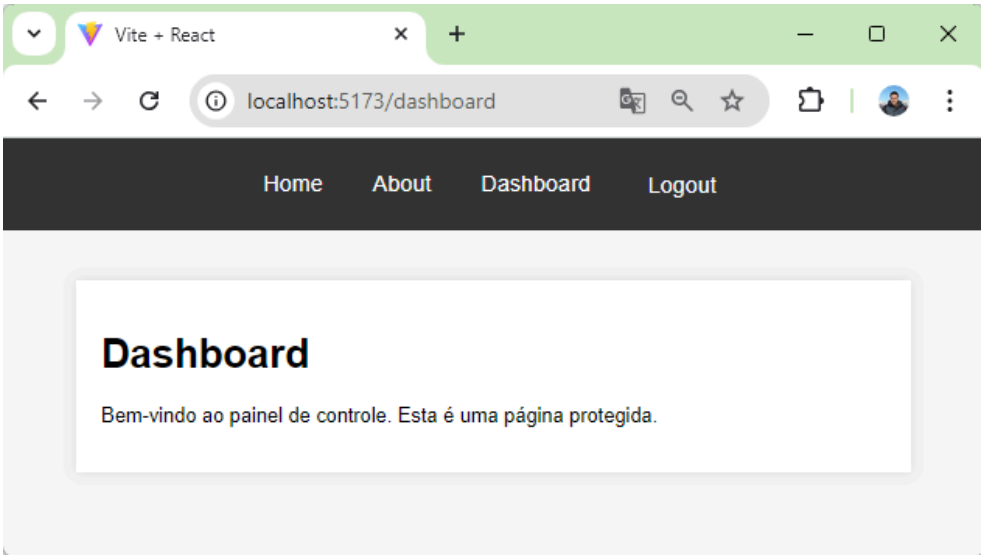

Figura 3: Página privada de nossa aplicação