Introdução ao React Router: Navegação entre páginas e configuração de rotas

Sumário:

1. Introdução ao React Router

Em sites tradicionais, o navegador solicita um documento de um servidor web, faz o download e avalia os arquivos de CSS e JavaScript, e renderiza o HTML enviado pelo servidor. Quando o usuário clica em um link, o processo começa tudo de novo para carregar uma nova página. Isso pode tornar o site mais lento, pois precisa carregar tudo do zero a cada navegação.

O **React Router** é uma biblioteca que facilita a navegação em aplicações Single Page Applications (SPA). Em vez de recarregar a página ao navegar entre links, o React Router atualiza apenas o conteúdo relevante, melhorando a experiência do usuário. Neste contexto, o React Router permite o roteamento no lado do cliente (client-side routing), realizando um comportamento diferente. Ao invés de solicitar uma nova página completa ao servidor, o React Router permite que a URL seja alterada no navegador e que a página atualizada seja renderizada no próprio navegador (usando JavaScript). Isso significa que a navegação entre "páginas" do site é instantânea, e o navegador só busca novos dados (via API, por exemplo) para preencher essa nova página, sem precisar recarregar tudo.

Esse processo melhora a performance, pois não há a necessidade de baixar todos os arquivos (como CSS e JavaScript) a cada navegação, e permite a criação de transições mais dinâmicas e fluídas, como animações.

1.1 Configuração de um projeto de exemplo

Para auxiliar na compreensão dos conceitos relacionados ao React Router, considere o escopo de desenvolvimento de uma aplicação web sobre venda de produtos onlines.

1.1.1 Configuração Básica do React Router

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
```

Ou, se estiver usando yarn:

```
yarn add react-router-dom
```

1.1.2 Estrutura do Projeto

Em seguida, considere que o nosso projeto contém a seguinte estrutura de diretórios e arquivos:

```
src/
|-- pages/
|    |-- About.jsx
|    |-- ErrorPage.jsx
|    |-- Home.jsx
|    |-- ProductDetails.jsx
|    |-- Products.jsx
|-- App.jsx
|-- index.css
|-- main.jsx
```

Figura 1: Estrutura do projeto de exemplo

Descrição: A imagem mostra a estrutura de um projeto de código-fonte, com a pasta principal chamada "src". Dentro dela, há uma subpasta chamada "pages", que contém cinco arquivos JSX: "About.jsx", "ErrorPage.jsx", "Home.jsx", "ProductDetails.jsx" e "Products.jsx". Fora da pasta "pages", há três outros arquivos: "App.jsx", "index.css" e "main.jsx". Cada item é listado com indentação para indicar a hierarquia de diretórios.

1.1.3 Criação das páginas básicas do projeto

Neste passo, vamos considerar a criação das páginas **Home, About** e **Products** de nosso projeto. Neste contexto, considere o seguinte código para o arquivo **Home.jsx**:

```jsx
function Home() {
  return (
    <div>
      <h1>Home Page</h1>
      <p>Bem-vindo à nossa loja online!</p>
    </div>
  );
}
export default Home;
```

Em seguida, considere o seguinte código para o arquivo **About.jsx**:

```
function About() {
  return (
    <div>
      <h1>About Us</h1>
      <p>Conheça mais sobre nossa loja e nossos valores.</p>
    </div>
  );
}
export default About;
```

Finalmente, considere o seguinte código para o arquivo **Products.jsx**:

```
function Products() {
  return (
    <div>
      <h1>Lista de Produtos</h1>
      <p>Listagem de Produtos</p>
    </div>
  );
}
export default Products;
```

1.1.4 Definição do Layout Básico da aplicação

Agora, vamos alterar o arquivo **App.jsx** para conter o layout básico de nossa aplicação, contendo o código sobre a navegação comum entre as páginas, além do espaço para atualizarmos o conteúdo de cada página acessada e o rodapé:

```
function App() {
  return (
    <div>
      <header>
        <h1>Minha Loja Virtual</h1>
        <nav>
          <ul>
            <li><a href="/" alt="">Home</a></li>
            <li><a href="/about" alt="">Sobre</a></li>
            <li><a href="/products" alt="">Produtos</a></li>
          </ul>
        </nav>
      </header>
```

```
      <main>
        <p>Conteúdo que virá de cada página.</p>
      </main>
      <footer>
        <p>&copy; 2024 Minha Loja. Todos os direitos reservados.</p>
      </footer>
    </div>
  );
}
export default App;
```

Este código define um componente chamado **App**, que representa a estrutura básica de uma loja virtual. Ele importa um arquivo CSS (App.css) para estilização e retorna um layout composto por três seções principais: um cabeçalho (**<header>**), um conteúdo principal (**<main>**) e um rodapé (**<footer>**). No cabeçalho, há um título "Minha Loja Virtual" e uma barra de navegação com links para as páginas "Home", "Sobre" e "Produtos", utilizando a tag **<a>** para redirecionamento. O **<main>** contém um parágrafo com texto placeholder que será substituído pelo conteúdo específico de cada página. Por fim, o rodapé exibe uma mensagem de copyright para "Minha Loja", indicando o ano 2024.

Antes de testar nossa aplicação, sugiro fazer uma pequena alteração estilo CSS de nossa aplicação. Desta forma, pensando em uma forma prática de realizar isso, vamos alterar o arquivo **index.css** com o seguinte código:

```
body {
  margin: 0;
  display: flex;
  justify-content: center; /* Centraliza horizontalmente */
  align-items: flex-start; /* Alinha o conteúdo no topo */
  min-width: 320px;
  min-height: 100vh; /* Garante que o body ocupe a altura da tela */
  background-color: #f4f4f4; /* Cor de fundo opcional */
  font-family: Arial, sans-serif;
}

div {
  max-width: 1400px; /* Largura máxima do conteúdo */
  width: 100%; /* Usa 100% da largura disponível */
  padding: 20px;
  box-sizing: border-box; /* Inclui o padding na largura total */
```

```
  background-color: #fff; /* Fundo branco para o conteúdo */
  margin-top: 20px; /* Espaçamento opcional a partir do topo */
}

/* Cabeçalho */
header {
  background-color: #007BFF;
  color: white;
  padding: 20px;
  text-align: center;
}
header h1 {
  margin: 0;
  font-size: 2.5em;
}

/* Navegação */
nav ul {
  list-style: none; /* Remove marcadores de lista */
  padding: 0;
  margin: 20px 0 0 0;
  display: flex; /* Coloca os itens da lista em linha */
  justify-content: center; /* Centraliza os itens de navegação */
}
nav ul li {
  margin: 0 15px; /* Espaçamento entre os itens */
}
nav ul li a {
  color: white;
  text-decoration: none; /* Remove sublinhado dos links */
  font-size: 1.2em;
}
nav ul li a:hover {
  text-decoration: underline; /* Sublinha ao passar o mouse */
}

/* Conteúdo principal */
main {
  padding: 20px;
  text-align: center;
}
```

```css
/* Rodapé */
footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 10px;
  margin-top: 20px;
}
footer p {
  margin: 0;
  font-size: 0.9em;
}
```

Veja na figura abaixo o resultado desta aplicação quando você acessa a rota raiz (“/”):

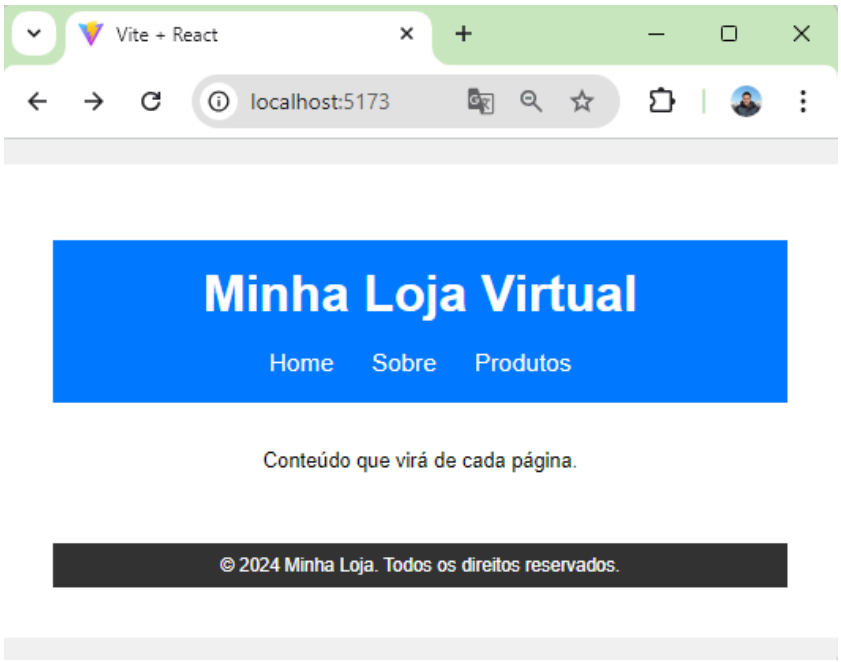

Figura 2: Exemplo de acesso à rota raiz “/”

Descrição: A imagem mostra a interface de uma página web chamada "Minha Loja Virtual". No topo, há uma barra azul com o nome da loja em letras grandes e brancas. Abaixo do título, há um menu de navegação com links para "Home", "Sobre" e "Produtos". No centro da página, aparece o texto: "Conteúdo que virá de cada página." Na parte inferior, há um rodapé preto com o aviso de copyright: "© 2024 Minha Loja. Todos os direitos reservados."

Entretanto, neste momento, ainda não conseguimos navegar entre as páginas, mesmo alterando a URL diretamente no navegador. Precisamos, então, configurar as rotas da aplicação.

1.2 Configuração das Rotas

Agora, precisamos criar um **Browser Router** e configurar nossa primeira rota. Isso habilitará o roteamento no lado do cliente para o nosso aplicativo web. Browser Router é o roteador específico do **React Router** que usa a API de **histórico do navegador**. Isso é essencial para habilitar o roteamento no lado do cliente, ou seja, permitir a navegação entre diferentes páginas da sua aplicação sem recarregar a página.

O **createBrowserRouter** e o **RouterProvider** fazem parte da biblioteca React Router e são utilizados para configurar e gerenciar o roteamento dentro de uma aplicação React. O **createBrowserRouter** cria um roteador do tipo "Browser Router". Este tipo de roteador utiliza a API de Histórico do Navegador (History API) para controlar as rotas da aplicação, ou seja, permite uma navegação baseada em URLs, sem recarregar a página (usando o mecanismo de "Single Page Application"). Já o **RouterProvider** é um componente React que "envolve" a aplicação e provê o roteador criado (com **createBrowserRouter**) para ela. Ele gerencia toda a lógica de navegação entre as rotas da aplicação, permitindo que o roteador passe informações sobre a rota atual para os componentes da aplicação.

Neste contexto, considere o seguinte código de configuração inicial do roteador no arquivo **main.jsx**:

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from "./App.jsx";
import Home from "./pages/Home.jsx";
import About from "./pages/About.jsx";
import Products from "./pages/Products.jsx";
import "./index.css";
import { createBrowserRouter, RouterProvider } from "react-router-dom";

const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
  },
  {
    path: "/home",
    element: <Home />,
  },
  {
```

```
    path: "/about",
    element: <About />,
  },
  {
    path: "/products",
    element: <Products />,
  },
]);
createRoot(document.getElementById("root")).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>
);
```

Neste código, observe que:

- **createBrowserRouter:** Este comando configura o roteador para o aplicativo.
  - O roteador espera um array de rotas, e cada rota é definida como um objeto que inclui:
    - **path:** Define o caminho da URL.
    - **element:** O componente que será renderizado quando o usuário acessar a rota especificada.
    - **children (opcional):** Pode ser usado para definir rotas "filhas" (sub-rotas) que podem herdar ou estar aninhadas em rotas principais.
- **<RouterProvider router={router} />:** conecta o sistema de roteamento à aplicação.
  - O **<RouterProvider>** é um componente que recebe um objeto **router** como propriedade (**prop**). Esse objeto contém toda a configuração de rotas que você definiu usando o **createBrowserRouter**.
  - Ele age como um "provedor" de rotas, que significa que ele disponibiliza o roteador para que todos os componentes internos da árvore do React possam acessar e utilizar as funcionalidades de roteamento.
  - O roteador define qual componente deve ser exibido com base no caminho da URL.

Quando a aplicação inicia, o <**RouterProvider**> analisa a URL atual (por exemplo, “**/**”) e verifica qual componente deve ser exibido. Neste exemplo, se a URL for a raiz (“/”), o componente **App** será renderizado.

Agora podemos testar as rotas da aplicação. Acesse a rota “about” e veja o resultado:

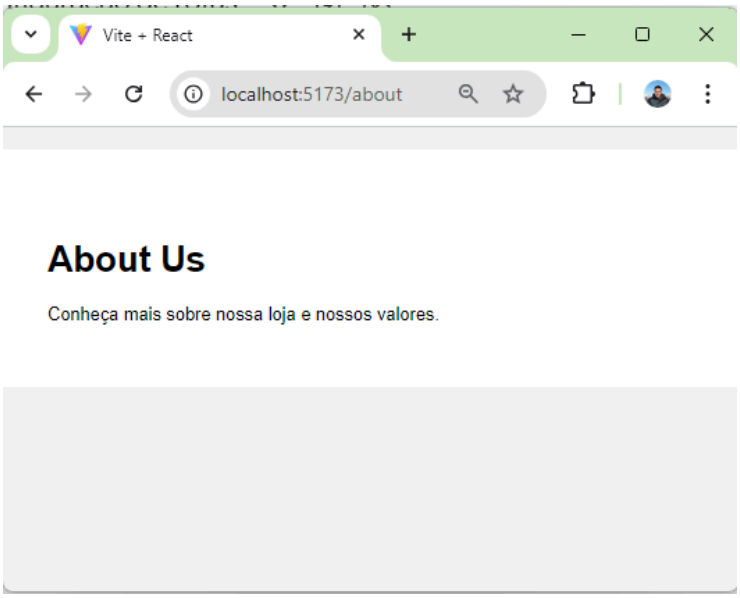

Figura 3: Exemplo de acesso à rota raiz “about”

Descrição: A imagem mostra uma página da web com o título "About Us" em negrito, seguido pela frase "Conheça mais sobre nossa loja e nossos valores."

Faça o mesmo para as outras rotas (“home” e “products”) e verifique se estão todas acessíveis.

1.3 Roteamento no Lado do Cliente

O roteamento no lado do cliente (client-side routing) é uma técnica utilizada em Single Page Applications (SPAs) para atualizar a URL **sem fazer uma nova requisição ao servidor** para carregar uma página completa. Em vez disso, o aplicativo manipula a mudança de URL localmente e atualiza apenas a parte necessária da interface. Isso proporciona uma navegação mais rápida e fluida, já que o usuário não precisa esperar pelo carregamento de uma nova página.

No cenário atual, a aplicação está fazendo uma requisição completa (full document request) ao servidor quando o usuário clica em um link de nossa aplicação. Esse comportamento é comum em sites tradicionais, onde cada clique em um link carrega uma nova página, resultando em uma experiência mais lenta.

Veremos algumas estratégias para evitarmos este comportamento no navegador.

1.3.1 Uso do Componente <Link>

O componente **<Link>** é um elemento do React Router utilizado para navegação no lado do cliente. Embora sua função seja semelhante à da tag **<a>** do HTML, ele oferece uma vantagem: em vez de recarregar toda a página ao clicar em um link, o React Router intercepta o evento, atualiza a URL no navegador e modifica apenas a interface de usuário, sem a necessidade de recarregar o conteúdo já carregado. Isso proporciona uma navegação mais fluida e rápida dentro da aplicação.

Neste contexto, em vez de usar uma tag <a> como esta:

```
<a href="/about">Sobre</a>
```

Usaremos o componente **<Link>** do React Router assim:

```
import { Link } from "react-router-dom";
<Link to="/about">Sobre</Link>
```

Onde:

- **<Link>:** É o componente que substitui a tag **<a>** para criar links no React Router.
- **to:** A prop **to** define a URL para a qual o link deve redirecionar. O React Router lida com a navegação internamente, sem recarregar a página inteira.

Neste contexto, vamos adicionar o componente **Link** à nossa aplicação. Assim, considere alterar o arquivo **App.jsx** com o seguinte código:

```
import { Link } from "react-router-dom";
function App() {
  return (
    <div>
      <header>
        <h1>Minha Loja Virtual</h1>
        <nav>
          <ul>
            <li><Link to="/home">Home</Link></li>
            <li><Link to="/about">Sobre</Link></li>
            <li><Link to="/products">Produtos</Link></li>
          </ul>
        </nav>
      </header>
```

```
      <main>
        <p>Conteúdo que virá de cada página.</p>
      </main>
      <footer>
        <p>&copy; 2024 Minha Loja. Todos os direitos reservados.</p>
      </footer>
    </div>
  );
}
export default App;
```

Faça um teste em nossa aplicação web. Observe que, quando clicamos em um link gerado pelo componente **<Link>** do React Router, a URL muda imediatamente no navegador, e o novo conteúdo aparece na tela sem um recarregamento completo da página.

1.3.2 Hook useNavigate

O **useNavigate** é um hook fornecido pelo React Router que permite a navegação programática, ou seja, redirecionamentos feitos por ações de código, como após o envio de formulários, cliques de botão, ou quando há necessidade de redirecionamento com base em lógica de negócios.

O **useNavigate** é útil quando você deseja navegar para outra rota sem depender de um **<Link>**. Por exemplo, ao submeter um formulário, você pode querer redirecionar o usuário para uma página diferente após o sucesso. Quando você chama a função retornada pelo **useNavigate**, ela muda a URL da mesma forma que o **<Link>**, e o React Router lida com a renderização do componente correto.

Por exemplo, altere o arquivo Home.jsx com o seguinte código:

```
import { useNavigate } from 'react-router-dom';

function Home() {
  const navigate = useNavigate();

  const handleClick = () => {
    // Faz algo (por exemplo, autenticação ou validação)
    navigate('/about'); // Redireciona para a página "Sobre Nós"
  };
```

```
  return (
    <div>
      <h1>Home Page</h1>
      <p>Bem-vindo à nossa loja online!</p>
      <button onClick={handleClick}>Ir para a página Sobre Nós</button>
    </div>
  );
}
export default Home;
```

Teste nossa aplicação, acessando a página “Home”. Para isso, informe a seguinte URL em seu navegador: http://localhost:5173/home

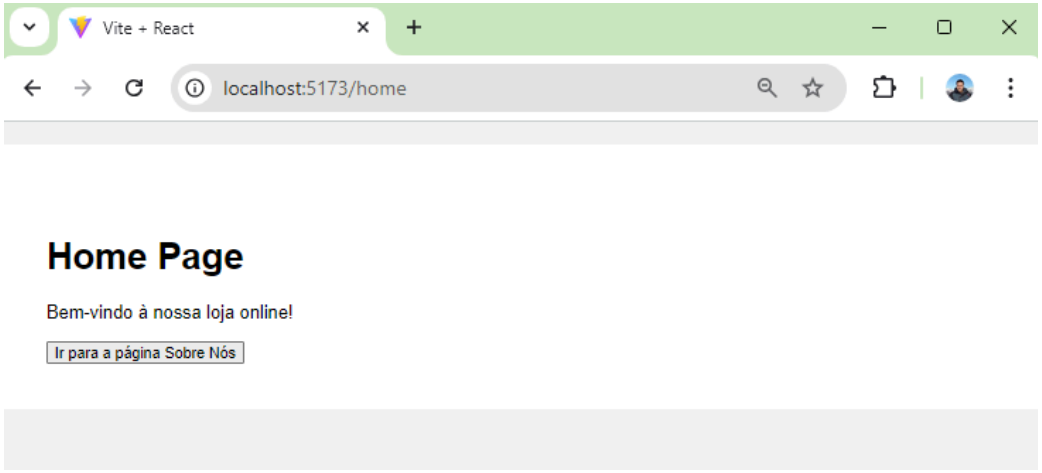

Figura 4: Exemplo de acesso à página “home” contendo o botão de redirecionamento

Descrição: A imagem mostra uma página web aberta no navegador, com a URL "localhost:5173/home" e a aba intitulada "Vite + React". O conteúdo da página exibe o título "Home Page" em negrito, seguido pela mensagem "Bem-vindo à nossa loja online!" logo abaixo. Há também um botão com o texto "Ir para a página Sobre Nós".

Após, clique no botão “Ir para a página Sobre Nós” e veja que fomos redirecionados para a página “Sobre”.

2. Rotas Aninhadas (Nested Routes)

As rotas aninhadas (nested routes) no React Router são um recurso poderoso que permite estruturar a navegação de uma aplicação de forma hierárquica. Elas permitem que você defina rotas dentro de outras rotas, criando uma estrutura de componentes e caminhos que refletem layouts aninhados ou páginas com sub-páginas.

Em nossa aplicação, observe que, ao clicarmos em “Home”, “Sobre” ou “Produtos”, seu conteúdo é carregado em um novo layout (sem cabeçalho, menu ou rodapé). Queremos, entretanto, que os componentes **Home**, **About** e **Products** sejam renderizados dentro do layout definido em **App**. Desta forma, vamos alterar as rotas de nossa aplicação em **main.jsx**:

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from "./App.jsx";
import Home from "./pages/Home.jsx";
import About from "./pages/About.jsx";
import Products from "./pages/Products.jsx";
import "./index.css";
import { createBrowserRouter, RouterProvider } from "react-router-dom";
const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children: [
      {
        path: "/",
        element: <Home />,
      },
      {
        path: "/about",
        element: <About />,
      },
      {
        path: "/products",
        element: <Products />,
      },
    ],
  },
]);
```

```
createRoot(document.getElementById("root")).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>
);
```

Observe neste exemplo:

- **path: "/" (Rota Pai):**
  - A rota raiz (/) utiliza o componente **App** como o layout principal da aplicação. Ele atua como o "pai" para todas as rotas filhas.
  - **errorElement: <ErrorPage />:** Define que, se qualquer erro ocorrer dentro desta rota pai ou em suas rotas filhas (tais como **/about** ou **/products**), o componente **ErrorPage** será exibido.
- **children (Rotas Filhas):**
  - As rotas filhas estão aninhadas dentro da rota pai e são definidas como caminhos relativos (/, **/about**, **/products**).
  - Cada uma dessas rotas pode gerar erros (como falhas ao carregar dados), e o **errorElement** da rota pai cuidará desses erros, exibindo o **ErrorPage**.
- **ErrorPage:**
  - Este é o componente que será renderizado quando ocorrer um erro. Ele pode exibir uma mensagem amigável ou instruções para o usuário.

Em seguida, precisamos alterar o arquivo **App.jsx** para definir melhor um espaço reservado onde o conteúdo da rota filha será renderizado:

```
import { Outlet, Link } from "react-router-dom";

function App() {
  return (
    <div>
      <header>
        <h1>Minha Loja Virtual</h1>
        <nav>
          <ul>
            <li><Link to="/">Home</Link></li>
            <li><Link to="/about">Sobre</Link></li>
            <li><Link to="/products">Produtos</Link></li>
          </ul>
        </nav>
      </header>
```

```
      <main>
        {/* Outlet é o "espaço reservado" onde o conteúdo da rota filha será
renderizado */}
        <Outlet />
      </main>
      <footer>
        <p>&copy; 2024 Minha Loja. Todos os direitos reservados.</p>
      </footer>
    </div>
  );
}
export default App;
```

Observe que componente **Outlet** é o ponto de inserção onde as rotas filhas serão renderizadas dentro do layout da rota pai. Quando o usuário visita, por exemplo, **/about**, o conteúdo da rota filha (**<About />**) será renderizado no lugar do **Outlet** dentro do layout de Dashboard.

Agora, teste novamente nossa aplicação e perceba que o conteúdo de “Home”, “Sobre” e “Produtos” é renderizado dentro do layout especificado no componente **App.**

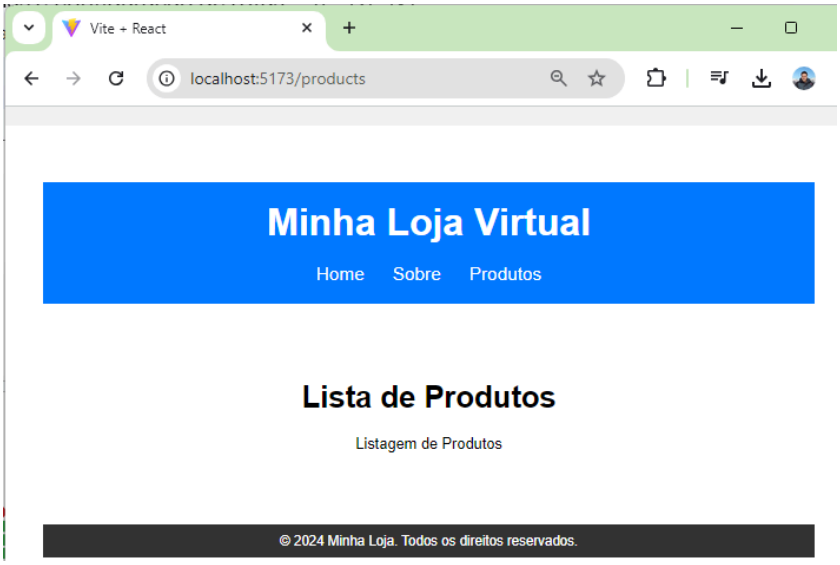

Figura 5: Exemplo de acesso à rota “products” com rotas aninhadas

Descrição: A imagem exibe a página "Shop" da "Minha Loja Virtual". No topo, há uma barra azul com o nome da loja em letras brancas grandes, acompanhada de um menu de navegação com os links "Home", "Sobre" e "Produtos". Abaixo, o título "Lista de Produtos" está em negrito, seguido pela frase "Listagem de Produtos". Na parte inferior, há um rodapé preto com um aviso de copyright"

3. Rotas Dinâmicas no React Router

As rotas dinâmicas no React Router permitem que você crie URLs com parâmetros variáveis, que podem ser usados para exibir conteúdo específico com base na URL. Esses parâmetros são chamados de parâmetros de rota e são uma maneira poderosa de tornar a navegação em uma aplicação React mais dinâmica e flexível.

No React Router, você pode definir parâmetros dinâmicos nas rotas utilizando **:** seguido do **nome do parâmetro**. Esses parâmetros são capturados pela URL e podem ser usados para carregar ou exibir informações específicas no componente.

Por exemplo, em nossa loja virtual, você pode ter uma rota dinâmica para cada produto com base no seu ID ou nome, como **/products/:productId**. Isso permite que sua aplicação carregue e exiba informações específicas de um produto com base no parâmetro da URL:

```
<Route path="/products/:productId" element={<Product />} />
```

Aqui**, :productId** é o parâmetro dinâmico que pode variar. A URL **/products/123** capturaria **123** como o **productId**.

Logo, vamos alterar o arquivo **Products.jsx** para mostrar uma listagem de produtos:

```
import { Link } from "react-router-dom";
const productList = [
  { id: 1, name: "Produto A" },
  { id: 2, name: "Produto B" },
  { id: 3, name: "Produto C" },
];
function ProductList() {
  return (
    <div>
      <h1>Lista de Produtos</h1>
      <ul
        style={{
          listStyleType: "none",
          padding: 0,
        }}>

         {productList.map((product) => (
          <li key={product.id} style={{ margin: "10px 0" }}>
```

```
            <Link
              to={`/products/${product.id}`}
              style={{
                textDecoration: "none",
                color: "#007BFF",
              }}>
              {product.name}
            </Link>
          </li>
        ))}
      </ul>
    </div>
  );
}
export default ProductList;
```

Agora, precisamos definir uma página de detalhes de produtos que simule a obtenção de dados de um produto específico com base no ID passado na URL. Altere **ProductDetails.jsx**:

```
import { useLoaderData } from 'react-router-dom';
const products = [
  { id: 1, name: 'Produto A', description: 'Descrição do Produto A', price: 100 },
  { id: 2, name: 'Produto B', description: 'Descrição do Produto B', price: 200 },
  { id: 3, name: 'Produto C', description: 'Descrição do Produto C', price: 300 },
];
export function loader({ params }) {
  const product = products.find((p) => p.id === Number(params.id));
  if (!product) {
    throw new Error('Produto não encontrado!');
  }
  return product;
}
function ProductDetails() {
  const product = useLoaderData();
  return (
    <div>
      <h1>{product.name}</h1>
      <p>{product.description}</p>
      <p>Preço: R$ {product.price}</p>
    </div>
  );
}
export default ProductDetails;
```

Observe que:

- **products:** Uma lista de produtos fictícios com informações como id, name, description e price.
- **loader({ params }):**
  - Esta função é chamada automaticamente pelo React Router quando a rota correspondente é acessada.
  - **params.id:** Captura o parâmetro id da URL, procura um produto na lista com o ID correspondente e retorna esse produto.
  - Se o produto não for encontrado, lança um erro com a mensagem "Produto não encontrado!".
- **ProductDetails:**
  - Um componente que exibe os detalhes do produto. Ele usa o hook **useLoaderData()** para acessar os dados do produto carregados pelo **loader**.
  - Quando você define um loader para uma rota no React Router, esse **loader** é executado automaticamente quando a rota é acessada. Assim, quando o usuário acessa uma rota de produto com um ID (por exemplo, **/products/1**), o **loader** é executado, encontra o produto com o **id: 1** na lista e passa esses dados para o componente **ProductDetails**.
  - Renderiza o nome, a descrição e o preço do produto.

OBS: O **loader** é executado antes do componente ser renderizado, o que garante que os dados estarão disponíveis quando o componente for exibido. Sugere-se utilizá-lo em operações como: busca de dados de uma API, busca de dados de um banco de dados, ou qualquer outra necessidade de lidar com lógica assíncrona (promises, fetches),

Em seguida, vamos alterar o arquivo **main.jsx** para que tenha uma rota dinâmica utilizando parâmetros de URL para o componente **ProductDetails**.

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from "./App.jsx";
import Home from "./pages/Home.jsx";
import About from "./pages/About.jsx";
import Products from "./pages/Products.jsx";
import ProductDetails,{loader as productLoader,} from "./pages/ProductDetails";
```

```
import "./index.css";
import { createBrowserRouter, RouterProvider } from "react-router-dom";
const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children: [
      {
        path: "/",
        element: <Home />,
      },
      {
        path: "/about",
        element: <About />,
      },
      {
        path: "/products",
        element: <Products />,
      },
      {
        path: "/products/:id",
        element: <ProductDetails />,
        loader: productLoader, // Carrega os dados do produto
      },
    ],
  },
]);
createRoot(document.getElementById("root")).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>
);
```

Observe que:

- Rotas Dinâmicas:
  - **path: "/products/:id":** Esta é uma rota dinâmica que captura o valor **:id** diretamente da URL. Por exemplo, se o usuário acessar **/products/123**, o valor **id** será **123**.

- **element: <ProductDetails />:** O componente **ProductDetails** será renderizado para exibir informações específicas sobre o produto cujo **ID** foi capturado.
- **loader: productLoader:** O **loader** é uma função que será executada antes de renderizar o componente. Ele pode buscar dados necessários para a exibição do produto com base no **id** da URL. Nesse caso, **productLoader** pode ser responsável por buscar os detalhes do produto em uma API ou banco de dados.

Agora, teste sua aplicação e acesse a rota “products”:

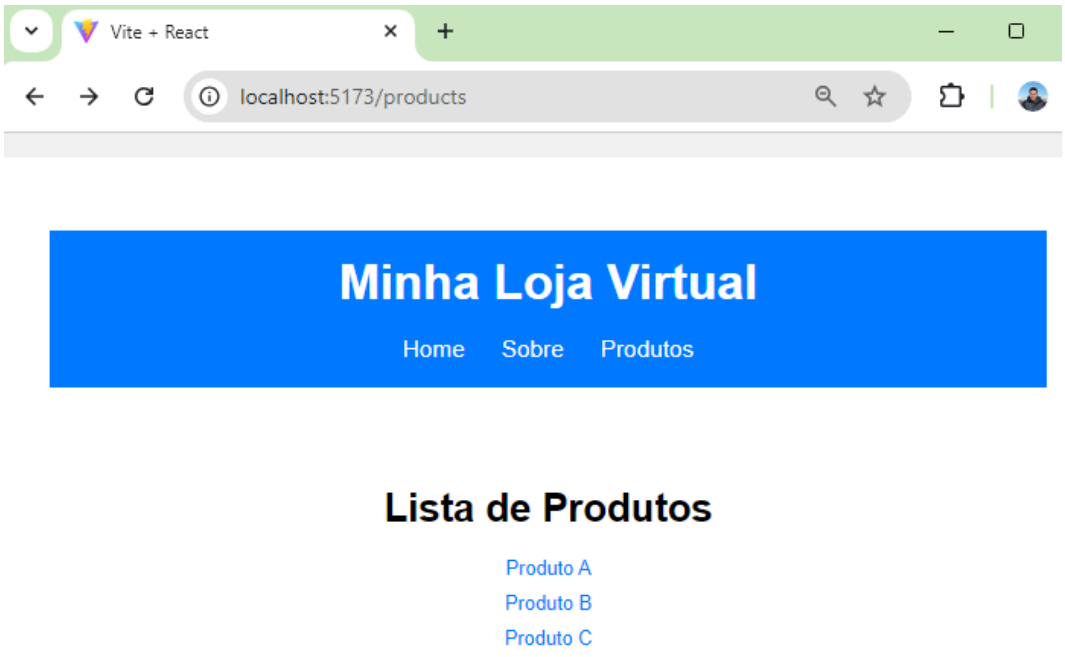

Figura 6: Exemplo de acesso à rota “products” com listagem de produtos

Descrição:A imagem exibe a página "Lista de Produtos" da "Minha Loja Virtual". No topo, há uma barra azul com o nome da loja em letras brancas grandes, acompanhada de um menu de navegação com os links "Home", "Sobre" e "Produtos". Abaixo, está o título "Lista de Produtos" em negrito, seguido por uma lista de links com os itens "Produto A", "Produto B" e "Produto C", em azul.

Em seguida, selecione um produto, tal como “Produto B”:

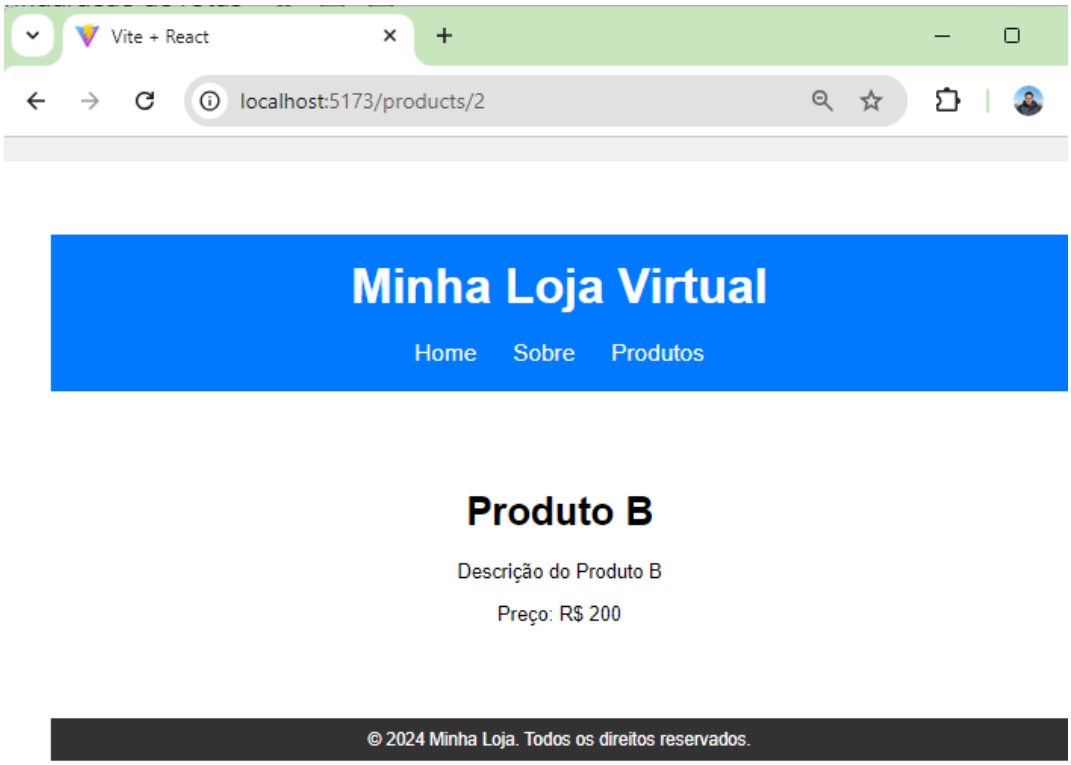

Figura 7: Exemplo de acesso à rota dinâmica “products” passando o ID 2

Descrição:A imagem exibe a página de detalhes do "Produto B" da "Minha Loja Virtual". No topo, há uma barra azul com o nome da loja em letras brancas grandes, junto com um menu de navegação contendo os links "Home", "Sobre" e "Produtos". Abaixo, está o título "Produto B" em negrito, seguido pela descrição "Descrição do Produto B" e o preço "R$ 200". Na parte inferior, há um rodapé preto com o aviso de copyright: "© 2024 Minha Loja. Todos os direitos reservados."

4. Tratando Erros de Rota

No contexto de uma aplicação React que utiliza o React Router para gerenciar rotas, é importante lidar corretamente com erros que ocorrem durante o carregamento, renderização ou mutação de dados dentro de uma rota específica. Quando um erro ocorre em uma rota, o React Router consegue capturar esse erro e fazer com que um componente de erro designado seja exibido (rota de fallback).

Faça um teste em nossa aplicação web. Acesse uma rota inexistente, tal como “nenhum”, e veja o resultado no navegador.

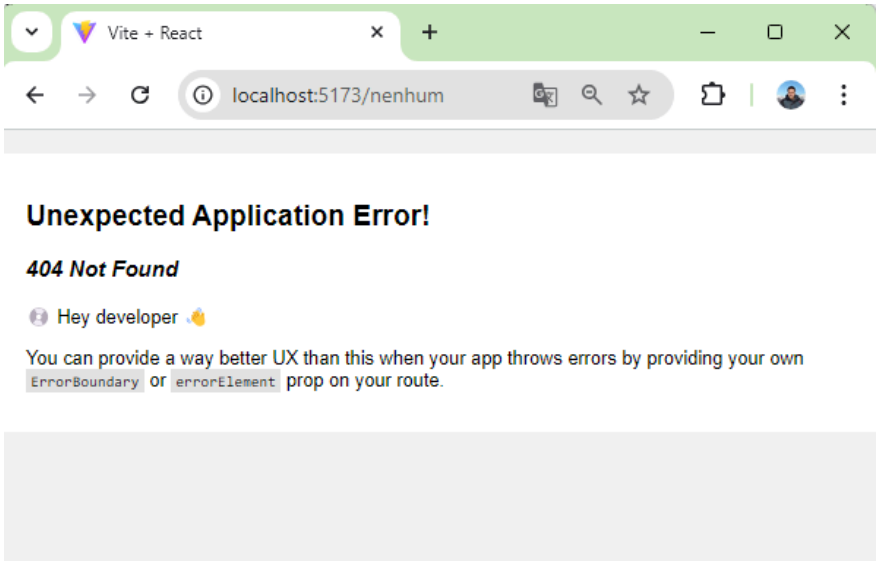

Figura 8: Exemplo de acesso à rota “nenhum”

Descrição: A imagem mostra uma página de erro exibindo a mensagem "Unexpected Application Error!" em destaque, seguida por "404 Not Found". Abaixo, há um aviso para o desenvolvedor com um emoji de mão acenando, sugerindo que uma melhor experiência de usuário (UX) pode ser fornecida ao lidar com erros, recomendando o uso de "ErrorBoundary" ou "errorElement". O fundo da página é branco e simples.

Para tratar esses cenários, é comum criar uma rota de erro personalizada, que será exibida sempre que nenhuma outra rota corresponder à URL acessada. Para isso, usaremos o **useRouteError**, que é um hook disponibilizado pelo React Router para capturar e lidar com erros que ocorrem durante o carregamento, renderização ou mutação de dados dentro de uma rota específica. Ele facilita a implementação de páginas de erro personalizadas, permitindo que você acesse o erro diretamente dentro de um componente.

Desta forma, precisamos alterar o arquivo **main.jsx** da seguinte forma:

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from "./App.jsx";
import Home from "./pages/Home.jsx";
import About from "./pages/About.jsx";
import Products from "./pages/Products.jsx";
import ProductDetails, {
  loader as productLoader,
} from "./pages/ProductDetails";
import ErrorPage from "./pages/ErrorPage.jsx"; // Componente de erro
personalizado
import "./index.css";
import { createBrowserRouter, RouterProvider } from "react-router-dom";

const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    errorElement: <ErrorPage />, // Define o componente de erro para esta rota
    children: [
      {
        path: "/",
        element: <Home />,
      },
      {
        path: "/about",
        element: <About />,
      },
      {
        path: "/products",
        element: <Products />,
      },
      {
        path: "/products/:id",
        element: <ProductDetails />,
        loader: productLoader, // Carrega os dados do produto
      },
    ],
  },
]);
```

```
createRoot(document.getElementById("root")).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>
);
```

Observe que ao definir o roteador com **createBrowserRouter**, você pode especificar um **errorElement** para cada rota. Esse elemento é o componente de erro que será exibido quando a rota falhar (neste caso, o componente **ErrorPage**).

Em seguida, adicione o seguinte código ao arquivo **ErrorPage.jsx**:

```
import { useRouteError } from "react-router-dom";

function ErrorPage() {
  const error = useRouteError(); // Captura o erro da rota

  return (
   <div>
     <h1>Oops! Algo deu errado.</h1>
     <p>Desculpe, não conseguimos carregar a página.</p>
     <p>
      <i>{error.statusText || error.message}</i> {/*Exibe a mensagem do erro*/}
     </p>
   </div>
  );
}
export default ErrorPage;
```

Observe que dentro do componente **ErrorPage**, o hook **useRouteError** captura o objeto de erro gerado pela falha da rota. Esse erro pode incluir detalhes como mensagens de falha, status de resposta HTTP, entre outros. No exemplo, o erro capturado pelo **useRouteError** é exibido na página de erro através de **error.statusText** (que pode ser a descrição do status HTTP) ou **error.message** (a mensagem de erro).

Acesse novamente uma rota inexistente, tal como “nenhum”, e veja o resultado no navegador.

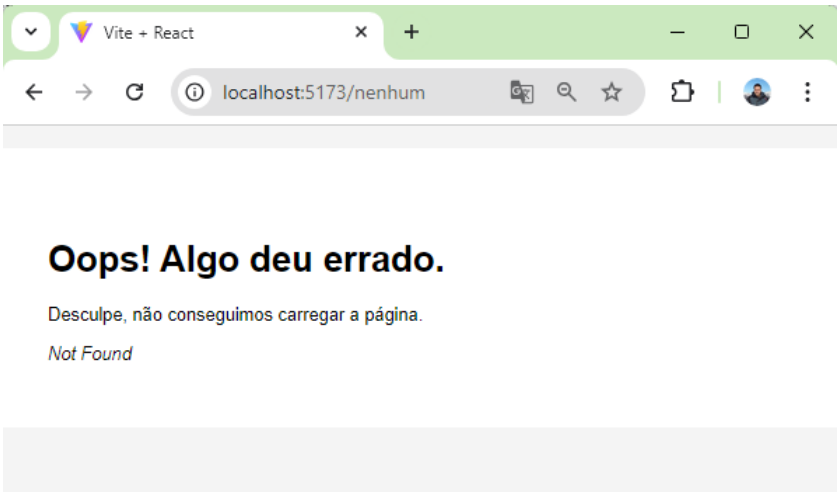

Figura 9: Exemplo de acesso à rota “nenhum” com tratamento de erros

Descrição: A imagem mostra uma página de erro exibindo a mensagem "Oops! Algo deu errado." em destaque, seguida pelo texto "Desculpe, não conseguimos carregar a página." Logo abaixo, há a frase "Not Found".

O hook **useRouteError** é útil nas seguintes situações:

- Erros de carregamento de dados: Quando uma rota falha ao buscar dados de uma API ou servidor.
- Falhas de renderização: Quando um erro ocorre ao tentar renderizar um componente em uma rota.
- Erros de mutação: Quando uma operação de mutação (como uma solicitação POST, PUT ou DELETE) falha.
- Exceções inesperadas: Qualquer exceção ou erro não tratado dentro da árvore de componentes do React Router.

Referências

O material desenvolvido para este capítulo baseou-se nas seguintes obras:

- DEVLIN BASILAN DULDULAO, D. B.; CABAGNOT, R. J. L. Practical Enterprise React: Become an Effective React Developer in Your Team. USA: APRESS, 2021.
- ELROM, E. React and Libraries: Your Complete Guide to the React Ecosystem. USA: APRESS, 2021.
- MDN WEB DOCS. Getting started with React. Disponível em: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started. Acessado em 21 jun. 2024.
- REACT DOCS. Disponível em https://pt-br.react.dev/. Acessado em 15 jun. 2024.
- ROLDÁN, C. S. React 17: Design Patterns and Best Practices. UK: Packt Publishing, 2021.
- SCOTT, A. D. JavaScript Everywhere. USA: O'Reilly Media, 2020.
- UZAYR, S. B.; CLOUD, N.; AMBLER, T. JavaScript Frameworks for Modern Web Development: The Essential Frameworks, Libraries, and Tools to Learn Right Now. USA: APRESS, 2019.
- W3SCHOOLS. React Tutorial. Disponível em: https://www.w3schools.com/react. Acessado em 17 jun. 2024.