1. Atividades Práticas - Rotas Dinâmicas com React Router

Nesta atividade prática, vamos criar um exemplo simples passo a passo para ilustrar como podemos trabalhar com rotas dinâmicas 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-app -- --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-app
```

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/
|-- pages/
|   |-- CustomerList.jsx
|   |-- CustomerList.css
|   |-- CustomerDetails.jsx
|   |-- CustomerDetails.css
|-- App.jsx
|-- App.css
|-- main.jsx
```

Observe que:

- **CustomerList.jsx** e **CustomerDetails.jsx** têm seus próprios arquivos CSS separados, permitindo uma melhor modularização e manutenção do código.
- Os estilos globais (como a estilização do body) ficam em um arquivo separado chamado **App.css,** que é importado no componente principal da aplicação (**App.jsx**).

Com essa abordagem, você organiza melhor os arquivos, tornando mais fácil encontrar e modificar cada componente individualmente.

1.3 Passo 2: Criar os arquivos de estilo

Neste passo, vamos criar os arquivos de estilo de nossa aplicação. Desta forma, comeÇe alterando o arquivo **App.css**:

```css
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f0f0f0;
}
h1, h2 {
  color: #333;
  text-align: center;
}
```

Este código CSS define estilos básicos para a página. O seletor body aplica a fonte "Arial" (ou outra fonte sem serifa se "Arial" não estiver disponível), remove margens e preenchimentos padrão e define um fundo cinza claro (**#f0f0f0**). Os seletores **h1** e **h2** aplicam uma cor de texto cinza escuro (**#333**) e centralizam o texto dentro desses títulos (**text-align: center**). Isso proporciona um layout limpo e consistente para o corpo e os títulos da página.

Em seguida, altere o arquivo **CustomerList.css**:

```
/* Estilos para a lista de clientes */
.customer-list {
    list-style: none;
    padding: 0;
}
.customer-list li {
    padding: 10px;
    border-bottom: 1px solid #ccc;
}
.customer-list li a {
    text-decoration: none;
    color: #007bff;
    font-weight: bold;
}
.customer-list li a:hover {
    text-decoration: underline;
}
/* Estilos para o container principal */
.container {
    max-width: 800px;
    margin: 20px auto;
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}
```

Este código CSS estiliza uma lista de clientes e um container principal. A classe **.customer-list** remove os marcadores da lista (**list-style: none**) e remove o preenchimento padrão. Os itens da lista (**.customer-list li**) recebem um espaçamento interno de **10px** e uma linha de borda inferior cinza clara (**#ccc**). Os links dentro da lista (**.customer-list li a**) têm o texto estilizado sem sublinhado (**text-decoration: none**), com cor azul (**#007bff**) e texto em negrito. Ao passar o mouse sobre o link (**:hover**), o sublinhado reaparece. A classe **.container** define o layout

principal com largura máxima de **800px**, centralizando o conteúdo com **margin: 20px auto**, fundo branco, espaçamento interno, bordas arredondadas, e uma leve sombra para destacar o container.

Finalmente, altere o arquivo de estilos chamado **CustomerDetails.css**:

```css
/* Estilos para os detalhes do cliente */
.customer-details {
    margin-top: 20px;
    padding: 20px;
    background-color: #fafafa;
    border: 1px solid #ddd;
    border-radius: 4px;
}
.customer-details p {
    margin: 10px 0;
}
.customer-details strong {
    color: #333;
}
/* Estilos do link voltar */
.back-link {
    display: inline-block;
    margin-top: 10px;
    color: #007bff;
    text-decoration: none;
}
.back-link:hover {
    text-decoration: underline;
}
```

Este código CSS estiliza a seção de detalhes do cliente e o link "voltar". A classe **.customer-details** aplica uma margem superior de 20px, preenchimento de 20px, fundo cinza claro (**#fafafa**), borda fina (**#ddd**), e bordas levemente arredondadas. Os parágrafos dentro desta seção (**.customer-details p**) têm uma margem de 10px vertical. O elemento **** dentro desta seção recebe uma cor cinza escuro (**#333**) para destaque. A classe **.back-link** estiliza o link

"voltar" como um bloco em linha com uma cor azul (**#007bff**) e sem sublinhado inicialmente, mas com sublinhado ao passar o mouse (**:hove**r).

1.4 Passo 3: Criando os componentes CustomerList e CustomerDetails

Agora, vamos criar os componentes **CustomerList** e **CustomerDetails** de nossa aplicação. Vamos começar alterando o arquivo **CustomerList.jsx**:

```
import { Link, Outlet, useLoaderData } from "react-router-dom";
import "./CustomerList.css"; // Importa o CSS específico do componente

function CustomerList() {
  const customers = useLoaderData(); // Carrega a lista de clientes do loader
  return (
    <div className="container">
      <h1>Lista de Clientes</h1>
      <ul className="customer-list">
        {customers.map((customer) => (
          <li key={customer.id}>
            {/* Link para a página de detalhes do cliente */}
            <Link to={`/customers/${customer.id}`}>{customer.name}</Link>
          </li>
        ))}
      </ul>
      {/* Renderiza os detalhes do cliente na mesma página */}
      <Outlet />
    </div>
  );
}
export default CustomerList;
```

Este código define o componente **CustomerList** no React, que exibe uma lista de clientes. Ele utiliza o hook **useLoaderData** do React Router para carregar os dados dos clientes de forma assíncrona. Cada cliente é mapeado para um item de lista (**<li>**), com um link (**<Link>**) que navega para uma rota específica baseada no **ID** do cliente **(/customers/:id)**, levando à página de detalhes do cliente. O componente **Outlet** permite que os detalhes do cliente sejam renderizados dentro da mesma página, criando uma experiência de navegação fluida sem

recarregar a página. O arquivo CSS específico **CustomerList.css** é importado para estilizar o componente.

Em seguida, vamos alterar o arquivo **CustomerDetails.jsx**:

```
import { useLoaderData, Link } from "react-router-dom";
import './CustomerDetails.css'; // Importa o CSS específico do componente
function CustomerDetails() {
  const customer = useLoaderData(); // Dados do cliente carregados pelo loader
  return (
    <div className="customer-details">
      <h2>Detalhes do Cliente</h2>
      <p><strong>Nome:</strong> {customer.name}</p>
      <p><strong>Email:</strong> {customer.email}</p>
      <p><strong>Telefone:</strong> {customer.phone}</p>
      <p><strong>Endereço:</strong> {customer.address}</p>
      {/* Link para voltar à lista de clientes */}
      <Link to="/customers" className="back-link">← Voltar à lista de
clientes</Link>
    </div>
  );
}
export default CustomerDetails;
```

Este código define o componente **CustomerDetails**, que exibe os detalhes de um cliente específico. Ele usa o hook **useLoaderData** do React Router para obter os dados do cliente, previamente carregados pelo loader. Dentro do componente, as informações do cliente, como nome, e-mail, telefone e endereço, são exibidas com o uso de elementos **<p>** e **** para formatação. Há também um link (**<Link>**) para retornar à lista de clientes, estilizado com a classe **back-link**, que aponta para a rota **/customers**. O arquivo CSS **CustomerDetails.css** é importado para estilizar o componente.

1.5 Passo 4: Definição do Router

Agora, definimos o **router** de nossa aplicação. Para isso, altere o arquivo **App.jsx**:

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import './App.css'; // Importa estilos globais
import CustomerList from './pages/CustomerList';
import CustomerDetails from './pages/CustomerDetails';

// Dados estáticos dos clientes
```

```
const customersData = [
  { id: 1, name: "John Doe", email: "john@example.com", phone: "(555) 123-4567",
address: "123 Main St, Springfield" },
  { id: 2, name: "Jane Smith", email: "jane@example.com", phone: "(555) 987-6543",
address: "456 Elm St, Springfield" },
  { id: 3, name: "Alice Johnson", email: "alice@example.com", phone: "(555)
555-5555", address: "789 Oak St, Springfield" }
];
// Loader para carregar a lista de clientes
const customersLoader = () => {
  return customersData; // Retorna a lista de clientes diretamente
};
// Loader para carregar os detalhes de um cliente específico
const customerDetailsLoader = ({ params }) => {
  const customerId = parseInt(params.customerId, 10); // Converte o ID para número
  const customer = customersData.find(c => c.id === customerId); // Encontra o
cliente no array
  if (!customer) {
    throw new Error("Cliente não encontrado"); // Caso o cliente não exista
  }
  return customer;
};
// Definimos o router com as rotas dinâmicas e loaders
const router = createBrowserRouter([
  {
    path: "/customers",
    children: [
      {
        index: true, // Define esta rota como indexada (padrão é "/customers")
        element: <CustomerList />, // Página da lista de clientes
        loader: customersLoader,   // Carrega a lista de clientes
      },
      {
        path: ":customerId", // Rota dinâmica para detalhes do cliente
        element: <CustomerDetails />,
        loader: customerDetailsLoader, // Carrega os detalhes do cliente específico
      },
    ],
  },
]);
function App() {
  return <RouterProvider router={router} />;
}
export default App;
```

Este código define um roteador com duas rotas principais sob **/customers**: uma rota indexada que carrega e exibe uma lista de clientes utilizando o componente **CustomerList**, e uma rota dinâmica que exibe os detalhes de um cliente específico através de **CustomerDetails**. Os dados dos clientes são estáticos e são carregados pelos loaders: o **customersLoader** carrega a lista completa de clientes, e o **customerDetailsLoader** busca os detalhes de um cliente específico com base no **customerId** presente na URL. A navegação entre a lista e os detalhes dos clientes é feita sem recarregar a página. O componente **RouterProvider** é usado para gerenciar o roteamento dentro do aplicativo.

1.6 Passo 5: Testar a aplicação

Agora, você pode rodar o projeto para verificar se as rotas estão funcionando corretamente:

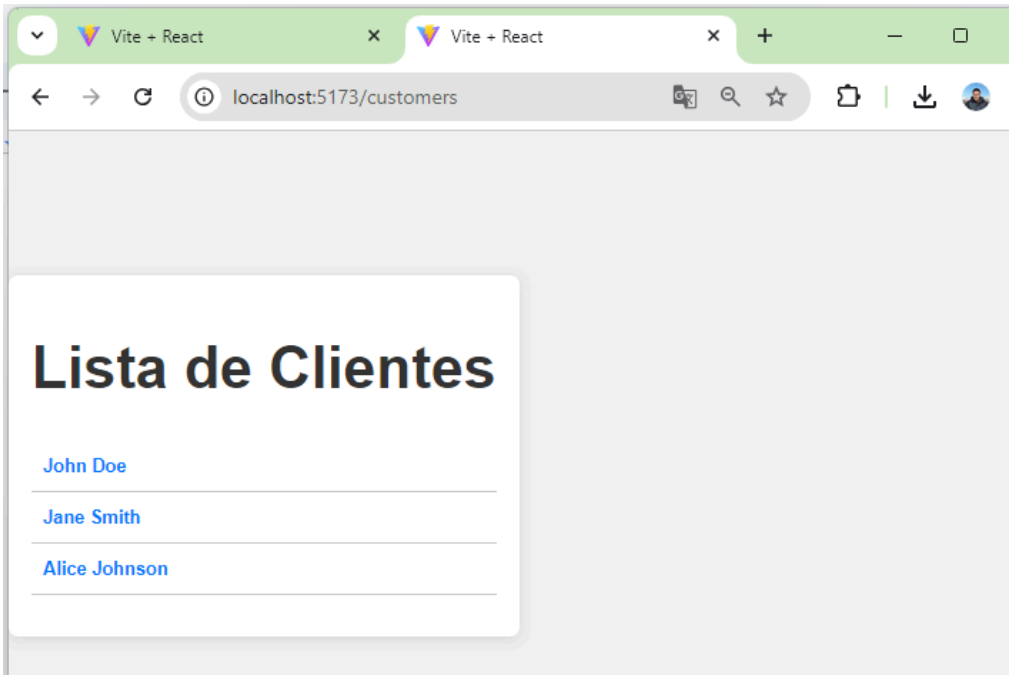

Figura 1: Exemplo de acesso à rota “customers”

Procure acessar os dados de um cliente específico, por exemplo, “Jane Smith”:

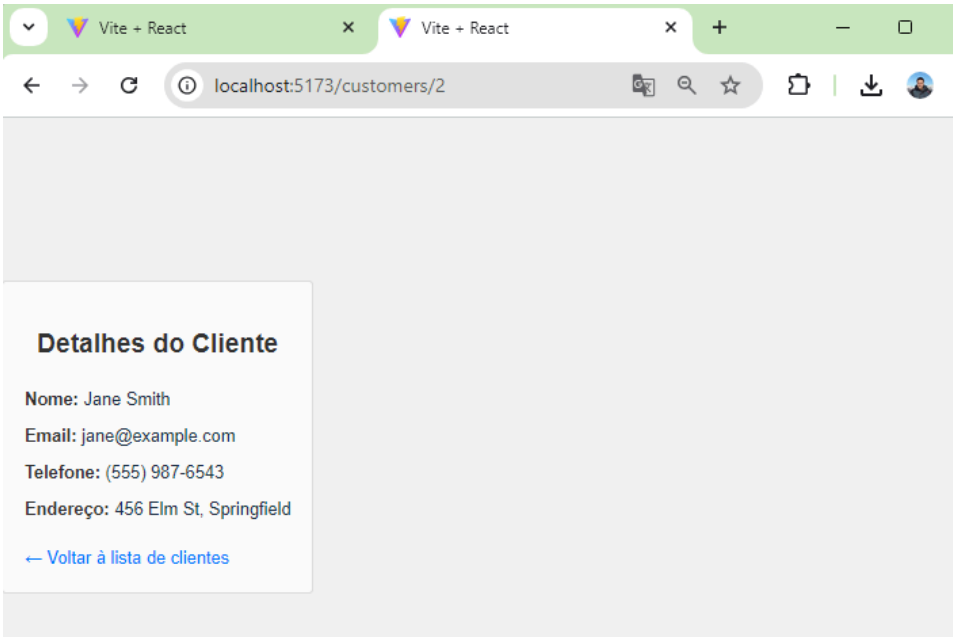

Figura 2: Exemplo de acesso à rota “customers” passando o ID = 2