

Objetivos:

- I. Routes;
- II. Rotas com restrição de acesso;
- III. Hook useNavigate.

I. Routes

Uma aplicação React geralmente é do tipo SPA (Single-Page Application), ou seja, toda a renderização ocorre em uma única página (documento web). No entanto, é possível criar rotas com base na URL para renderizar componentes diferentes, proporcionando ao aplicativo um comportamento semelhante ao de múltiplas páginas, como em um portal.

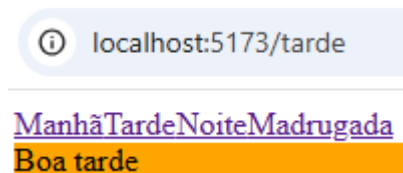
O React Router (<https://reactrouter.com/en/main>) é uma biblioteca amplamente utilizada para implementar roteamento em aplicativos React. Ele fornece componentes essenciais, como BrowserRouter, Routes e Route, que permitem definir rotas e renderizar componentes específicos de acordo com a URL atual.

Instalação do React Router: para instalar o pacote necessário, execute o seguinte comando no terminal:

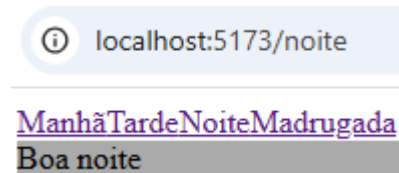
```
npm i react-router
```

O código a seguir renderiza o componente de acordo com a rota especificada na URL.

A rota **/tarde** renderiza o componente **Tarde**



A rota **/noite** renderiza o componente **Noite**



```
import { BrowserRouter, Link, Route, Routes } from "react-router";
```

```
export default function App() {
  return (
    <BrowserRouter>
      <Menu />
      <Rotas />
    </BrowserRouter>
  );
}
```

```
function Rotas() {
  return (
    <Routes>
      <Route path="/" element={<Erro />} />
      <Route path="/manha" element={<Manha />} />
    </Routes>
  );
}
```

```

    <Route path="/tarde" element={<Tarde />} />
    <Route path="/noite" element={<Noite />} />
    <Route path="/madrugada" element={<Madrugada />} />
  </Routes>
);
}

function Menu() {
  return (
    <div>
      <Link to="/manha">Manhã</Link>
      <Link to="/tarde">Tarde</Link>
      <Link to="/noite">Noite</Link>
      <Link to="/madrugada">Madrugada</Link>
    </div>
  );
}

function Manha() {
  return <div style={{ backgroundColor: "yellow" }}>Bom dia</div>;
}

function Tarde() {
  return <div style={{ backgroundColor: "orange" }}>Boa tarde</div>;
}

function Noite() {
  return <div style={{ backgroundColor: "#aaa" }}>Boa noite</div>;
}

function Madrugada() {
  return <div style={{ backgroundColor: "#777" }}>Bom sono</div>;
}

function Erro() {
  return <div>Rota inexistente</div>;
}

```

Explicação do código

- **BrowserRouter**: é o componente que habilita o roteamento no React. Ele deve envolver todos os componentes que precisam acessar as rotas. Observe que as rotas e definição dos links foram definidas dentro do contexto do BrowserRouter:

```

export default function App() {
  return (
    <BrowserRouter>
      <Menu />
      <Rotas />
    </BrowserRouter>
  );
}

```

```
    </BrowserRouter>
  );
}
```

- **Routes:** agrupa todas as definições de rotas da aplicação:

```
function Rotas() {
  return (
    <Routes>
      <Route path="/" element={<Erro />} />
      <Route path="/manha" element={<Manha />} />
      <Route path="/tarde" element={<Tarde />} />
      <Route path="/noite" element={<Noite />} />
      <Route path="/madrugada" element={<Madrugada />} />
    </Routes>
  );
}
```

- **Route:** define cada rota individualmente:
 - **path:** define o caminho da rota na URL. Use **path="/"** (rota coringa) para capturar rotas não definidas e exibir uma página de erro personalizada;
 - **element:** define o componente que será renderizado quando a rota for acessada.

Cada rota (**Route**) renderiza um componente específico (**Manha**, **Tarde**, **Noite**, **Madrugada**).

- Menu de navegação: utiliza o componente **Link** para criar links que permitem navegar entre as rotas:
 - **to:** especifica o caminho da rota;
 - Sempre inicie o caminho das rotas com **/** ao usar o componente **Link**.

```
function Menu() {
  return (
    <div>
      <Link to="/manha">Manhã</Link>
      <Link to="/tarde">Tarde</Link>
      <Link to="/noite">Noite</Link>
      <Link to="/madrugada">Madrugada</Link>
    </div>
  );
}
```

II. Rotas com restrição de acesso

O React Router é uma rota para um componente, então se a rota deixa de existir, o componente não será acessado por uma URL. No exemplo a seguir:

- Usuários não logados acessam somente as rotas **/manha** e **/tarde**;
- Usuários logados acessam somente as rotas **/noite** e **/madrugada**.

Para gerenciar essa restrição, utilizamos:

- Context API para gerenciar o estado de autenticação, que está no estado logado do LoginProvider;

- Renderização condicional para diferenciar entre rotas diurnas e noturnas:

```
function Rotas() {
  const { logado } = useLogado();
  return logado ? <RotasNoturno /> : <RotasDiurno />;
}
```

A seguir o código de exemplo:

```
import { createContext, useContext, useState } from "react";
import { BrowserRouter, Link, Route, Routes } from "react-router";
```

```
export default function App() {
  return (
    <LoginProvider>
      <BrowserRouter>
        <Menu />
        <Rotas />
      </BrowserRouter>
    </LoginProvider>
  );
}
```

```
function Rotas() {
  const { logado } = useLogado();
  return logado ? <RotasNoturno /> : <RotasDiurno />;
}
```

// rotas acessíveis apenas para usuários não logados

```
function RotasDiurno() {
  return (
    <Routes>
      <Route path="/" element={<Erro />} />
      <Route path="/manha" element={<Manha />} />
      <Route path="/tarde" element={<Tarde />} />
    </Routes>
  );
}
```

// rotas acessíveis apenas para usuários logados

```
function RotasNoturno() {
  return (
    <Routes>
      <Route path="/" element={<Erro />} />
      <Route path="/noite" element={<Noite />} />
      <Route path="/madrugada" element={<Madrugada />} />
    </Routes>
  );
}
```

```
function Menu() {
```

```
const { logado, setLogado } = useLogado();
return (
  <div>
    <button onClick={() => setLogado(!logado)}>
      {logado ? "Logout" : "Login"}
    </button>
    <Link to="/manha">Manhã</Link>
    <Link to="/tarde">Tarde</Link>
    <Link to="/noite">Noite</Link>
    <Link to="/madrugada">Madrugada</Link>
  </div>
);
}

function Manha() {
  return <div style={{ backgroundColor: "yellow" }}>Bom dia</div>;
}

function Tarde() {
  return <div style={{ backgroundColor: "orange" }}>Boa tarde</div>;
}

function Noite() {
  return <div style={{ backgroundColor: "#aaa" }}>Boa noite</div>;
}

function Madrugada() {
  return <div style={{ backgroundColor: "#777" }}>Bom sono</div>;
}

function Erro() {
  return <div style={{ backgroundColor: "red" }}>Rota inexistente</div>;
}

interface ContextProps {
  logado: boolean;
  setLogado: (value: boolean) => void;
}

interface ChildrenProps {
  children: React.ReactNode;
}

const LoginContext = createContext<ContextProps | null>(null);

function LoginProvider({ children }: ChildrenProps) {
  const [logado, setLogado] = useState(false);

  return (
```

```

    <LoginContext.Provider value={{ logado, setLogado }}>
      {children}
    </LoginContext.Provider>
  );
}

function useLogado() {
  const context = useContext(LoginContext);
  if (!context) {
    throw new Error("useLogado deve ser usado dentro de um LoginProvider");
  }
  return context;
}

```

Na prática a restrição de acesso foi construída usando intercâmbio das rotas através do retorno do componente Rotas. Essa organização de código faz com que a definição das rotas seja trocada mudando o valor do estado logado.

III. Hook useNavigate

O hook useNavigate fornece acesso ao objeto navigate e pode ser utilizado para redirecionar programaticamente o usuário para outra rota.

No exemplo a seguir:

- Ao acessar uma rota desconhecida, será exibido o componente **Erro**, assim como é mostrado ao lado;
- No componente **Erro**, há um botão que, ao ser clicado, chama a função `navigate("/manha")`, redirecionando o usuário para a rota `/manha`.

```
import { BrowserRouter, Link, Route, Routes, useNavigate } from "react-router";
```

```

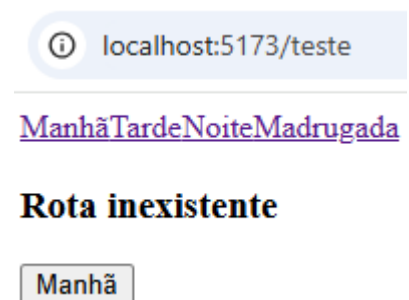
export default function App() {
  return (
    <BrowserRouter>
      <Menu />
      <Rotas />
    </BrowserRouter>
  );
}

```

```

function Rotas() {
  return (
    <Routes>
      <Route path="*" element={<Erro />} />
    </Routes>
  );
}

```



```

    <Route path="/manha" element={<Manha />} />
    <Route path="/tarde" element={<Tarde />} />
    <Route path="/noite" element={<Noite />} />
    <Route path="/madrugada" element={<Madrugada />} />
  </Routes>
);
}

function Menu() {
  return (
    <div>
      <Link to="/manha">Manhã</Link>
      <Link to="/tarde">Tarde</Link>
      <Link to="/noite">Noite</Link>
      <Link to="/madrugada">Madrugada</Link>
    </div>
  );
}

function Manha() {
  return <div style={{ backgroundColor: "yellow" }}>Bom dia</div>;
}

function Tarde() {
  return <div style={{ backgroundColor: "orange" }}>Boa tarde</div>;
}

function Noite() {
  return <div style={{ backgroundColor: "#aaa" }}>Boa noite</div>;
}

function Madrugada() {
  return <div style={{ backgroundColor: "#777" }}>Bom sono</div>;
}

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

  return (
    <div>
      <h3>Rota inexistente</h3>
      <div>
        <button onClick={() => navigate("/manha")}>Manhã</button>
      </div>
    </div>
  );
}

```

IV. Exercícios

Exercício 1 – Complete o código a seguir para os componentes serem roteados para os caminhos /legume, /verdura e /fruta. Qualquer URL desconhecida deverá ser roteada para o componente Erro. Adicione também um menu com links para as rotas.

localhost:5173/legume

[Frutas](#) [Legumes](#) [Verduras](#)
Beterraba
Cenoura

```
function Legume() {
  return (
    <div style={{ backgroundColor: "PeachPuff" }}>
      <div>Beterraba</div>
      <div>Cenoura</div>
    </div>
  );
}

function Verdura() {
  return (
    <div style={{ backgroundColor: "palegreen" }}>
      <div>Alface</div>
      <div>Couve</div>
    </div>
  );
}

function Fruta() {
  return (
    <div style={{ backgroundColor: "LemonChiffon" }}>
      <div>Laranja</div>
      <div>Manga</div>
    </div>
  );
}

function Erro() {
  return <h3>Rota inexistente</h3>;
}
```

Exercício 2 – Alterar o aplicativo do Exercício 1 para ter rotas para os componentes Carro e Moto. Adicione o controle de rotas, as rotas para /legume, /verdura e /fruta devem estar disponíveis quando o usuário estiver logado e as rotas /carro e /moto devem estar disponíveis quando o usuário não estiver logado.

Quando está logado:

localhost:5173/verdura

Logout [Frutas](#) [Legumes](#) [Verduras](#) [Carro](#) [Moto](#)

Alface

Couve

Quando não está logado:

localhost:5173/carro

Login [Frutas](#) [Legumes](#) [Verduras](#) [Carro](#) [Moto](#)

Gol

Uno

```
function Carro() {
  return (
    <div style={{ backgroundColor: "LightSkyBlue" }}>
      <div>Gol</div>
      <div>Uno</div>
    </div>
  );
}
```

```
function Moto() {
  return (
    <div style={{ backgroundColor: "Wheat" }}>
      <div>CG160</div>
      <div>DT180</div>
    </div>
  );
}
```

Exercício 3 – Complete o código a seguir para criar rotas para os componentes Form e List.

Requisitos:

- A aplicação deverá ser inicializada no componente Form;
- O componente Form deverá ter um link para a rota /list;
- O componente List deverá ter um link para a rota /form.

Componente Form:

localhost:5173/form

Nome

Idade

Salvar [Ir para Lista](#)

Componente List:

localhost:5173/list

Cadastro:

1. Ana - 22
2. Pedro - 25
3. Lúcia - 24

[Voltar para o formulário](#)

```
function Form() {
  const [name, setName] = useState("");
  const [age, setAge] = useState("");
```

```

const {add} = usePerson();

return (
  <div>
    <div>
      <label htmlFor="name">Nome</label>
      <input id="name" value={name} onChange={(e) => setName(e.target.value)} />
    </div>
    <div>
      <label htmlFor="age">Idade</label>
      <input id="age" value={age} onChange={(e) => setAge(e.target.value)} />
    </div>
    <div>
      <button onClick={() => add({name,age})}>Salvar</button>
    </div>
  </div>
);
}

function List() {
  const {list} = usePerson();

  return (
    <div>
      <p>Cadastro:</p>
      <ol>
        {
          list.map((item,index) => <li key={index}>{item.name} - {item.age}</li>)
        }
      </ol>
    </div>
  );
}

function Erro() {
  return <h3>Rota inexistente</h3>;
}

interface Person {
  name: string;
  age: string;
}

interface ContextProps {
  list: Person[];
  add: (person:Person) => void;
}

interface ChildrenProps {

```

```
    children: React.ReactNode;
  }

  const PersonContext = createContext<ContextProps | null>(null);

  function PersonProvider({ children }: ChildrenProps) {
    const [list, setList] = useState<Person[]>([]);

    function add(person: Person){
      setList( (prev) => [...prev, person]);
    }

    return (
      <PersonContext.Provider value={{ list, add }}>
        {children}
      </PersonContext.Provider>
    );
  }

  function usePerson() {
    const context = useContext(PersonContext);
    if (!context) {
      throw new Error("usePerson deve ser usado dentro de um PersonProvider");
    }
    return context;
  }
}
```