React Routes

Link de acesso à versão no Notion:

https://cherry-client-b8f.notion.site/React-Routes-810eb7faa6674d9caf13b32c80d2cf67?pvs=25

O que são rotas?

É o recurso que permite que o usuário navegue entre diferentes partes da aplicação. No react isso é gerenciado por uma biblioteca chamada **React Router Dom**. Para iniciarmos, é necessário fazer a instalação através do NPM.

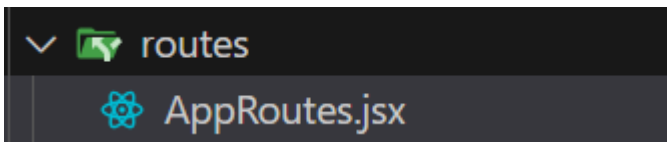
```
npm i react-router-dom
```

Link para a documentação oficial: https://reactrouter.com/

Utilizando createBrowserRouter

Iniciaremos pelo arquivo **main.jsx**

Primeiro passo é criar uma pasta chamada **routes**, dentro de src, e um componente dentro dessa pasta, que irá conter todas as rotas do projeto.



Neste componente, teremos que importar o elemento responsável pela criação das rotas, que é o **createBrowserRouter**.

```
import {createBrowserRouter} from 'react-router-dom'
```

Após a importação, começaremos a criar nosso roteador. Ele seguirá a estrutura abaixo:

```


//import dos elementos que usaremos para as configuração das rotas
import {createBrowserRouter} from 'react-router-dom'

/*variável que irá guardar todas as rotas que iremos criar dentro do
método createBrowserRouter()*/
export const router = createBrowserRouter([
  {},
  {},
])
```

O createBrowserRouter é uma função, que recebe um array de objetos, onde cada objeto tem as propriedades das rotas que iremos configurar.

Toda rota recebe duas propriedades: **path e element.**

path: indica o caminho da nossa aplicação. Por exemplo: /sobre, /contato. A barra(/) sem nada na frente, sempre indicará o caminho raiz da nossa aplicação.

element: indica o componente que será carregado quando aquela rota for acessada. Lembre-se sempre que a **importação do componente é obrigatória** para conseguirmos acessá-lo.

```
//AppRoutes.jsx

import {createBrowserRouter} from 'react-router-dom'
import App from '../App.jsx'

export const router = createBrowserRouter([
 {
```

```
    path: '/',
    element: <App/>
  }
])
```

Para inserir outras rotas, basta criar novos objetos, com outros caminhos e elementos a serem carregados.

```
//AppRoutes.jsx

import {createBrowserRouter} from 'react-router-dom'
import App from '../App.jsx'

export const router = createBrowserRouter([
  {
    path: '/',
    element: <App/>
  },
  {
    path: '/sobre',
    element: <Sobre/>
  }
])
```

Agora precisamos indicar para o React que esta variável router é a responsável por direcionar o usuário. Para isso iremos até o arquivo **main.jsx**, e importaremos, além do router que acabamos de criar, outro elemento do React Router Dom que é o **RouterProvider**.

```
import { RouterProvider } from "react-router-dom";
import { router } from "./routes/AppRoutes";
```

Após isso, devemos alterar a função que renderiza por padrão o <App /> inicialmente na nossa aplicação, atribuindo o router como responsável pelo roteamento da nossa aplicação.

```
//Main.jsx
import { RouterProvider } from "react-router-dom";
import { router } from "./routes/AppRoutes";

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

Como eu insiro um link para uma rota

Para inserir um link para acessar a rota, precisamos utilizar o elemento **Link** do React Router DOM. No arquivo do nosso menu, iremos substituir o links que utilizam **<a href=””>** pelo componente **<Link to=””>**, onde o caminho que estará dentro do atributo “to” será a rota de destino para qual aquele link deve levar o usuário.

Diferente da tag <a>, o Link não gera um recarregamento de todo o site, sendo o ideal para navegação entre rotas.

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

/*Exemplo de menu contendo Links para as rotas configuradas*/
export default function NavBar() {
    <ul>
        <li>
            <Link to='/'>Home</Link>
```

```
        </li>
        <li>
            <Link to='/sobre'>Sobre</Link>
        </li>
    </ul>
}
```

Definindo layouts

Para que páginas compartilhem o mesmo layout, precisaremos criar componentes que servirão como arquivos de layout e configurar rotas “filhas”, que herdarão este layout, utilizando o elemento **children** para eles.

```
//AppRoutes.jsx

import {createBrowserRouter} from 'react-router-dom'
import App from '../App.jsx'

export const router = createBrowserRouter([
  {
    path: '/',
    element: <RootLayout />,
    children: [
       //a rota index indica qual componente será renderizado por padrão
      {index:true, element: <ConteudoPrincipal/>},
      {path:'sobre', element: <Sobre/>}
    ]
  }
])
```

Agora as rotas dentro de children, seguirão o layout do elemento pai, que no nosso caso, é o componente **<RootLayout />**.

Para que essas rotas apareçam no conteúdo, nós devemos informar ao componente pai, onde inseri-las. Fazemos isso com o componente **<Outlet/>**:

```
import {Outlet} from 'react-router-dom'

export default function RootLayout(){
    return(
        <>
        <Header/>
            <Outlet/>
        <Footer/>
        </>
    )
}
```

O **Outlet** é um componente do react-router-dom que permite abrirmos uma janela que irá mostrar o conteúdo das rotas filho na tela, de acordo com a rota acessada pelo usuário.

Lidando com página não encontrada

Em alguns momentos nossos usuários podem acabar acessando rotas que não existem. Nesse caso, precisamos configurar um componente que será renderizado nesses casos. Podemos fazer isso de duas maneiras:

1ª opção: Error element

É o elemento padrão do react-router-dom que trata rotas com erro:

```
//AppRoutes.jsx

export const router = createBrowserRouter([
  {
    path: '/',
    element: <RootLayout />,
    errorElement: <PageNotFound/>
    children: [
      {index:true, element: <ConteudoPrincipal/>},
      {path:'sobre', element: <Sobre/>}
    ]
  }
])
```

2ª opção: Path *

É uma rota que será habilitada para qualquer rota que não foi prevista anteriormente no nosso mapeamento.

```
//AppRoutes.jsx

export const router = createBrowserRouter([
  {
    path: '/',
    element: <App/>,
    children: [
      {index:true, element: <ConteudoPrincipal/>},
      {path:'sobre', element: <Sobre/>},
      {path: '*', element: <PageNotFound/>}
    ]
  }
])
```

Lidando com layouts distintos

Caso quisermos utilizarmos outros layouts na nosso aplicação, precisamos configurar outras rotas que funcionarão como layout e adicionarmos filhos à elas. O arquivo deve ficar como no exemplo abaixo:

```
//AppRoutes.jsx

import {createBrowserRouter} from 'react-router-dom'
import App from './App'
import PageNotFound from './Pages/PageNotFound'
import Sobre from './Pages/Sobre'
import Servicos from './Pages/Servicos'
import Contato from './Pages/Contato'
import ConteudoPrincipal from './Componentes/ConteudoPrincipal'

const router = createBrowserRouter([
  {
    path: '/',
    element: <RootLayout/>,
    children: [
      {index:true, element: <ConteudoPrincipal/>},
      {path:'sobre', element: <Sobre/>},
      {path:'servicos', element: <Servicos/>},
      {path:'contato', element: <Contato/>},
      {path:'*', element: <PageNotFound/>}
```

```
    ]
  },
  {
    path: '/admin',
    element: <AdminLayout/>,
    children: [
      {index:true, element: <FormLogin/>},
      {path:'/admin/home', element: <AdminHome/>},
      {path:'/admin/perfil', element: <Profile/>}
    ]
  }
])
```

Neste caso o componente **<AdminLayout/>** deve conter a estrutura desejada e o **Outlet** onde as rotas filhas serão renderizadas. Da mesma maneira, o **<RootLayout/>** também deve conter um **Outlet** para renderizar as rotas que são filhas dele.

Redirecionamento e navegação programática

No React Router, **navigate** é a **função que permite fazer navegação programática**, ou seja, mudar de rota via código, sem o usuário clicar em um **<Link>**. Ela é obtida usando o hook **useNavigate**.

Abaixo um exemplo do uso:

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

export default function MyComponent() {
  const navigate = useNavigate(); // pega a função navigate

  function goToHome() {
    navigate("/"); // redireciona para a home
  }

  return <button onClick={goToHome}>Ir para Home</button>;
}
```

Principais usos

Ir para uma rota específica:

```
navigate("/dashboard");
```

Voltar ou avançar na navegação (como history):

```
navigate(-1); // volta uma página
navigate(1);  // avança uma página
```

Substituir a rota (não adiciona no histórico):

- **replace: true** → o usuário não consegue voltar usando o botão "voltar" do navegador.
- Útil para redirecionamentos após login ou logout.

```
navigate("/login", { replace: true });
```

Rotas dinâmicas

Muitas vezes precisamos de atributos dinâmicos nas rotas, para que possamos acessar um post específico em um blog, informações de um produto selecionado pelo usuário ou exibir o perfil de um usuário específico, por exemplo.

Neste caso temos que criar parâmetros na rota, que receberá um valor dinâmico toda vez que for chamada

No React Router, você deve configurar a rota com um **:parâmetro** (dois pontos na frente):

```
//AppRoutes.jsx

import { createBrowserRouter} from "react-router-dom";
import App from "./App";
import Post from "./Post";

export const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
  },
  {
    path: "/posts/:id",
    element: <Post />,
  },
]);
```

Criando links para rotas dinâmicas

Para criar links para as rotas dinâmicas, basta passar um valor na url, que será armazenado no parâmetro:

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

export default function App() {
  return (
    <div>
      <h1>Lista de Posts</h1>
      <ul>
        <li>
          <Link to="/posts/1">Post 1</Link>
        </li>
        <li>
          <Link to="/posts/2">Post 2</Link>
        </li>
        <li>
          <Link to="/posts/3">Post 3</Link>
        </li>
      </ul>
    </div>
  );
}
```

Normalmente esse valor é variável, vindo de uma base de dados, como no exemplo abaixo:

```
const posts = [
  { id: 10, title: "React Básico" },
  { id: 11, title: "React Avançado" },
];

export default function App() {
  return (
    <div>
      <h1>Posts</h1>
      <ul>
        {posts.map((post) => (
```

```
        <li key={post.id}>
          <Link to={`/posts/${post.id}`}>{post.title}</Link>
        </li>
      ))}
    </ul>
  </div>
 );
}
```

Acessando o parâmetro

Para acessar o parâmetro, usaremos um outro hook do React Router Dom que é o **useParams**. Ele será responsável por capturar o parâmetro enviado, assim conseguimos acessar as informações do elemento do qual queremos exibir mais informações.

```
/* Importamos o useParams*/
import { useParams } from "react-router-dom";

export default function PaginaPost() {
   /*Chamamos o useParams, trazendo o que foi enviado na rota e guardando no objeto params*/
  const params = useParams();

  return (
    <div>
    <!-- Acessamos a propriedade do objeto com um ponto -->
      <h1>Post #{params.id}</h1>
      <p>Carregando conteúdo do post...</p>
    </div>
  );
}
```

Múltiplos parâmetros

É possível passar diversos parâmetros, por exemplo para acessar os comentários de um post dentro de um blog.

```
import Comment from "./Comment";

export const router = createBrowserRouter([
  {
    path: "/posts/:id/:commentId",
    element: <Comment />,
  },
]);
```

Rotas privadas

Rotas privadas no React Router são rotas que **só podem ser acessadas se o usuário estiver autenticado**. Se não estiver, ele é redirecionado para a tela de login (ou outra página).

No React, isso normalmente é feito criando um **componente wrapper** que verifica a autenticação antes de renderizar o conteúdo.

Passo a passo da lógica

1. Você tem uma rota "protegida", por exemplo **/dashboard**.
2. Antes de renderizar, você verifica se o usuário está logado.
3. Se estiver, mostra o conteúdo; se não, redireciona para **/login**.

Implementando as rotas privadas

Primeiramente, iremos simular um componente que retorna **true/false** se existe um token de autenticação no localStorage.

```
// auth.jsx
export const isAuthenticated = () => {
  // Simulação: retorna true se o usuário estiver logado
  return localStorage.getItem("token") !== null;
};
```

Para configurar a rota, criaremos um componente que fará a função de wrapper, chamado **PrivateRoute**.

```
//AppRoutes.jsx
import { createBrowserRouter } from "react-router-dom";
import Login from "./Login";
import Dashboard from "./Dashboard";
import PrivateRoute from "./PrivateRoute";

const router = createBrowserRouter([
  { path: "/login", element: <Login /> },
  {
    path: "/dashboard",
    element: (
      <PrivateRoute>
        <Dashboard />
      </PrivateRoute>
    )
  },
]);
```

Dentro do componente **PrivateRoute**, iremos inserir a lógica que irá renderizar o componente privado caso o usuário esteja autenticado. Caso não esteja, irá redirecionar para a rota de /login.

```
import { Navigate } from "react-router-dom";
import { isAuthenticated } from "./auth";

export default function PrivateRoute({ children }) {
  if (!isAuthenticated()) {
    // se não estiver logado → redireciona para login
    return <Navigate to="/login" replace />;
  }
  // se estiver logado → renderiza o conteúdo
  return children;
}
```

Referências

https://dev.to/tywenk/how-to-use-nested-routes-in-react-router-6-4jhd

https://www.robinwieruch.de/react-router-nested-routes/

https://www.dhiwise.com/post/the-power-of-createbrowserrouter-optimizing-your-react-app

https://reactrouter.com/en/main/start/tutorial

https://css-tricks.com/learning-react-router/

https://hygraph.com/blog/routing-in-react

https://coderpad.io/blog/development/guide-to-react-router/

https://medium.com/@nomannayeem/react-router-7-the-ultimate-guide-to-the-new-features-and-framework-capabilities-06e7f06981f6