

FRONT-END DESIGN ENGINEERING

[React – API Connect File]

[Aula 15]
[Atualizado em: 06/05/2024]



REACT – API CONNECT FILE

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Usar uma conexão centralizada com uma API externa tem várias vantagens e desvantagens.

✓ Vantagens:

- ✓ **Reutilização de código:** Centralizar a conexão com a API em um único arquivo permite reutilizar a lógica de acesso à API em todo o aplicativo. Isso reduz a duplicação de código e facilita a manutenção.
- ✓ **Facilidade de gerenciamento:** Com uma conexão centralizada, é mais fácil gerenciar configurações de autenticação, como tokens de acesso ou chaves de API. Qualquer alteração nessas configurações pode ser feita em um único local.
- ✓ **Padronização:** Ao centralizar a conexão com a API, você pode estabelecer padrões consistentes para lidar com erros, autenticação, tratamento de dados, entre outros aspectos relacionados à comunicação com a API.

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Usar uma conexão centralizada com uma API externa tem várias vantagens e desvantagens.

✓ Desvantagens:

- ✓ **Acoplamento:** Uma conexão centralizada pode levar a um alto acoplamento entre diferentes partes do aplicativo. Mudanças na API ou nos requisitos de comunicação podem afetar muitas partes do código, tornando-o menos flexível.
- ✓ **Complexidade:** À medida que o aplicativo cresce, a lógica de comunicação com a API centralizada pode se tornar complexa e difícil de gerenciar. Isso pode dificultar a compreensão do código e introduzir potenciais pontos de falha.
- ✓ **Desempenho:** Dependendo da arquitetura do aplicativo e da carga de trabalho, uma conexão centralizada pode resultar em gargalos de desempenho, especialmente se várias partes do aplicativo estiverem disputando recursos da API simultaneamente.

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

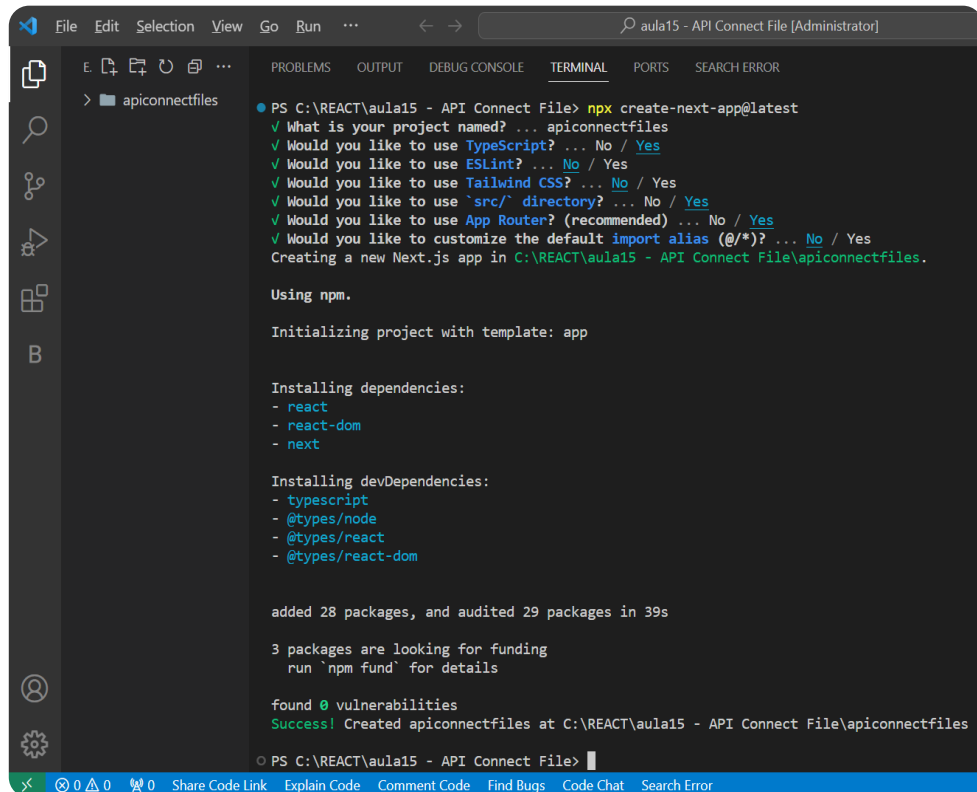
- ✓ Em resumo, usar uma conexão centralizada com uma API externa pode ser uma escolha sólida em muitos casos, pois oferece benefícios como reutilização de código e facilidade de gerenciamento. No entanto, é importante estar ciente das possíveis desvantagens, como acoplamento e complexidade, e considerar cuidadosamente o impacto que essa abordagem terá no seu aplicativo.

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ O projeto é uma aplicação web que acessa uma API externa para exibir uma lista de posts. Cada post poderá ser visualizado individualmente em uma página separada.

O acesso à API é centralizado em um único arquivo para melhor organização e reutilização de código.



```
PS C:\REACT\aula15 - API Connect File> npm create-next-app@latest
✓ What is your project named? ... apiconnectfiles
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like to use `src/` directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to customize the default import alias (@/*)? ... No / Yes
Creating a new Next.js app in C:\REACT\aula15 - API Connect File\apiconnectfiles.

Using npm.

Initializing project with template: app

Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- typescript
- @types/node
- @types/react
- @types/react-dom

added 28 packages, and audited 29 packages in 39s

3 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Success! Created apiconnectfiles at C:\REACT\aula15 - API Connect File\apiconnectfiles

PS C:\REACT\aula15 - API Connect File>
```

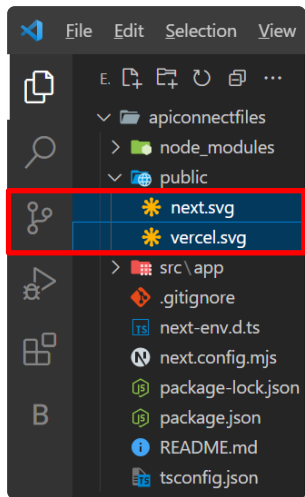
REACT – API CONNECT FILE

✓ Introdução ao API Connect File

✓ Vamos apagar alguns arquivos e manter outros

✓ Na pasta PUBLIC

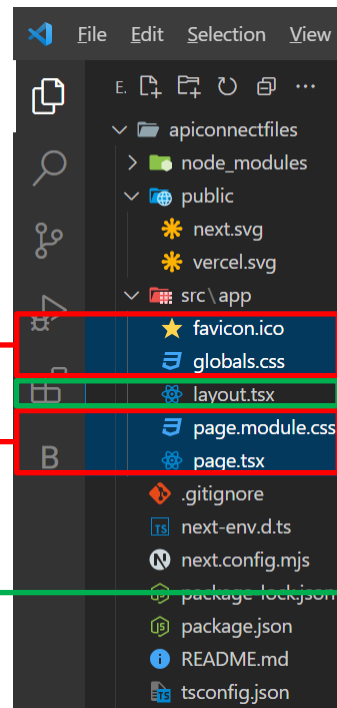
- ✓ next.svg
- ✓ vercel.svg



excluir esses arquivos

✓ Na pasta SRC > APP

- ✓ favicon.ico
- ✓ globals.css
- ✓ page.module.css
- ✓ Page.tsx

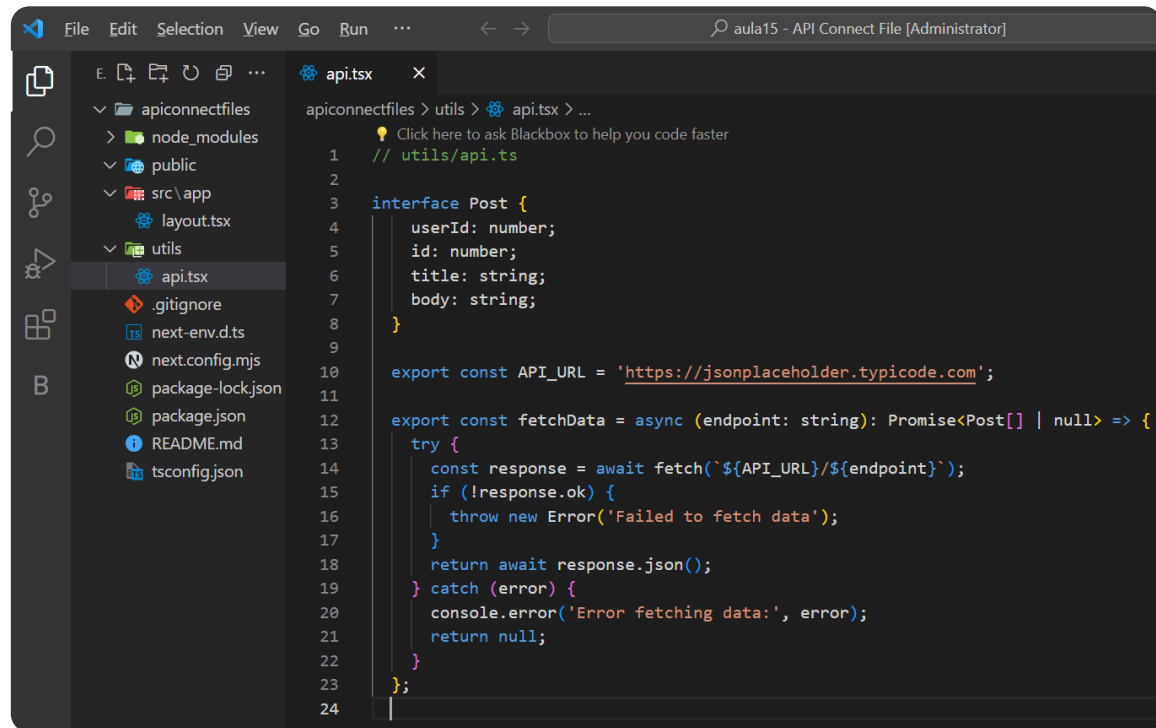


NÃO excluir este arquivo

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Vamos criar o componente
utils > api.tsx



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the following structure:

- apiconnectfiles
 - node_modules
 - public
 - src\app
 - layout.tsx
 - utils
 - api.tsx
- .gitignore
- next-env.d.ts
- next.config.mjs
- package-lock.json
- package.json
- README.md
- tsconfig.json

The code editor shows the content of `api.tsx`:

```
1 // Click here to ask Blackbox to help you code faster
2 // utils/api.ts
3
4 interface Post {
5   userId: number;
6   id: number;
7   title: string;
8   body: string;
9 }
10
11 export const API_URL = 'https://jsonplaceholder.typicode.com';
12
13 export const fetchData = async (endpoint: string): Promise<Post[] | null> => {
14   try {
15     const response = await fetch(`${API_URL}/${endpoint}`);
16     if (!response.ok) {
17       throw new Error('Failed to fetch data');
18     }
19     return await response.json();
20   } catch (error) {
21     console.error('Error fetching data:', error);
22     return null;
23   }
24 }
```


REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Analisando linha a linha o arquivo api.xtsx

```
1 // utils/api.tsx
2
3 interface Post {
4   userId: number;
5   id: number;
6   title: string;
7   body: string;
8 }
```

// Aqui estamos definindo uma interface TypeScript chamada Post. Interfaces são usadas para definir a estrutura de objetos em TypeScript. Esta interface especifica os tipos de propriedades que esperamos encontrar em um objeto Post, incluindo userId, id, title e body, todos com tipos específicos.

```
9
```

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Analisando linha a linha o arquivo api.xtsx

```
10 export const API_URL = 'https://jsonplaceholder.typicode.com';  
    // Esta linha exporta uma constante chamada API_URL que contém a URL base da API que será usada para acessar os  
    dados. A URL aponta para a API JSONPlaceholder, que fornece dados de teste.  
11  
12 export const fetchData = async (endpoint: string): Promise<Post[] | null> => {  
    // Aqui, exportamos uma função chamada fetchData. Esta função recebe um parâmetro endpoint que indica o caminho  
    específico da API que desejamos acessar. A função retorna uma promessa (Promise) que resolve para um array de  
    objetos do tipo Post ou null.  
13     try {  
14         const response = await fetch(`${API_URL}/${endpoint}`);  
15         if (!response.ok) {  
16             throw new Error('Failed to fetch data');  
17         }  
        // Usamos um bloco try-catch para lidar com possíveis erros durante a chamada à API. Dentro do bloco try,  
        fazemos a chamada à API usando fetch e verificamos se a resposta é bem-sucedida (response.ok). Se não for,  
        lançamos um erro. Se a resposta for bem-sucedida, usamos response.json() para extrair os dados JSON da  
        resposta.
```

REACT – API CONNECT FILE

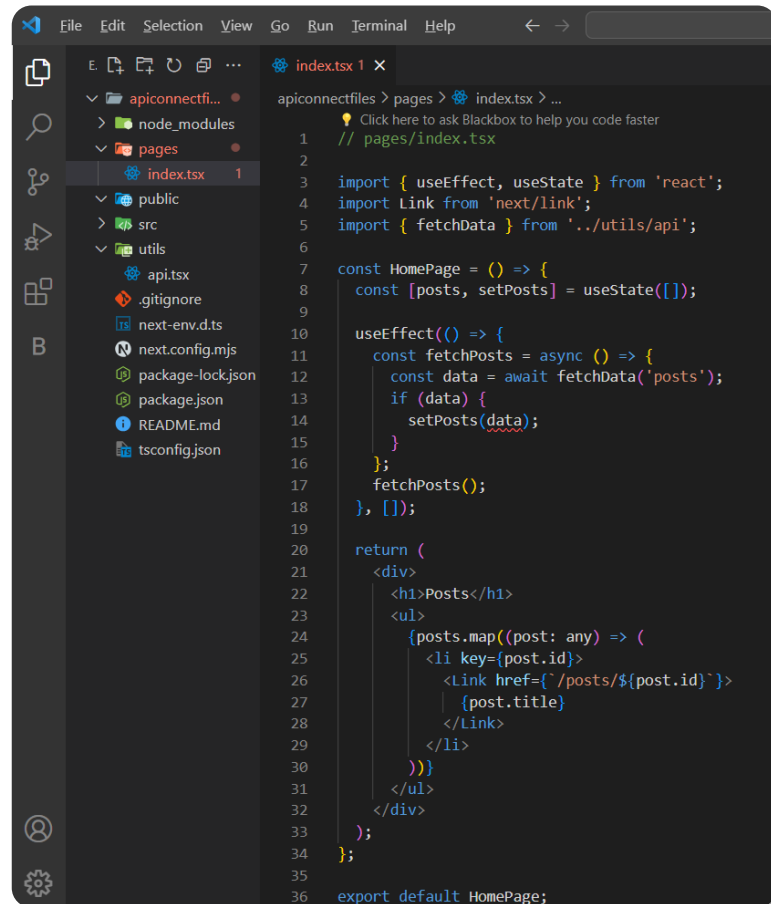
✓ Introdução ao API Connect File

- ✓ Analisando linha a linha o arquivo api.xtsx

```
18         return await response.json();  
        // Se a chamada à API for bem-sucedida, retornamos os dados JSON como um array de objetos do tipo Post.  
19     } catch (error) {  
        console.error('Error fetching data:', error);  
        // Se ocorrer um erro durante a chamada à API, capturamos o erro e o registramos usando console.error. Em  
        // seguida, retornamos null para indicar que não conseguimos obter os dados.  
21         return null;  
22     }  
23 }
```

REACT – API CONNECT FILE

- ✓ Introdução ao API Connect File
- ✓ Vamos criar o componente `pages > index.tsx`



The screenshot shows a code editor with a dark theme. On the left, a file explorer shows the project structure: `apiconnectfi...`, `node_modules`, `pages` (selected), `public`, `src`, and `utils`. Under `pages`, `index.tsx` is selected. The main editor area shows the content of `index.tsx`:

```
1 // pages/index.tsx
2
3 import { useEffect, useState } from 'react';
4 import Link from 'next/link';
5 import { fetchData } from '../utils/api';
6
7 const HomePage = () => {
8   const [posts, setPosts] = useState([]);
9
10  useEffect(() => {
11    const fetchPosts = async () => {
12      const data = await fetchData('posts');
13      if (data) {
14        setPosts(data);
15      }
16    };
17    fetchPosts();
18  }, []);
19
20  return (
21    <div>
22      <h1>Posts</h1>
23      <ul>
24        {posts.map((post: any) => (
25          <li key={post.id}>
26            <Link href={`/${posts}/${post.id}`}>
27              {post.title}
28            </Link>
29          </li>
30        ))}
31      </ul>
32    </div>
33  );
34 };
35
36 export default HomePage;
```

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

✓ Analisando linha a linha o arquivo index.tsx

```
1 // pages/index.tsx
2
3 import { useEffect, useState } from 'react';
  // Aqui estamos importando os hooks useEffect e useState do React. O useEffect é usado para executar efeitos
  colaterais em componentes funcionais, enquanto o useState é usado para adicionar estado a componentes funcionais.
4
5 import Link from 'next/link';
  // Estamos importando o componente Link do Next.js. O componente Link é usado para criar links de navegação entre
  páginas em um aplicativo Next.js.
6
7 import { fetchData } from '../utils/api';
  //Estamos importando a função fetchData do arquivo api.ts localizado na pasta utils. Essa função é usada para
  buscar os dados da API.
8
9 const HomePage = () => {
  // Estamos definindo um componente funcional chamado
10
11   const [posts, setPosts] = useState([]);
  // Estamos declarando um estado chamado posts usando o hook useState. O estado inicial é um array vazio []. O
  setPosts é uma função que usaremos para atualizar o estado posts.
12
13 }
```

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Analisando linha a linha o arquivo index.tsx

```
10  useEffect(() => {  
    // Estamos utilizando o hook useEffect para buscar os posts da API assim que o componente HomePage for  
    montado. O segundo argumento [] indica que esta função de efeito será executada apenas uma vez, após a  
    montagem do componente.  
  
    const fetchPosts = async () => {  
11      // Estamos definindo uma função assíncrona chamada fetchPosts que usa a função fetchData para buscar os  
      posts da API. Se os dados forem retornados com sucesso, atualizamos o estado posts com esses dados usando  
      setPosts.  
  
12      const data = await fetchData('posts');  
  
13      if (data) {  
14        setPosts(data);  
15      }  
16    };  
  
17    fetchPosts()  
    // Chamamos a função fetchPosts para iniciar a busca dos posts assim que o componente HomePage for montado.  
18  }, []);  
19
```

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Analisando linha a linha o arquivo index.tsx

```
20     return (  
    // estamos retornando a estrutura JSX que representa o conteúdo do componente HomePage  
  
    <div>  
21        // Estamos definindo uma função assíncrona chamada fetchPosts que usa a função fetchData para buscar os  
        posts da API. Se os dados forem retornados com sucesso, atualizamos o estado posts com esses dados usando  
        setPosts.  
  
22        <h1>Posts</h1>  
  
23        <ul>  
  
24            {posts.map((post: any) => (  
25                <li key={post.id}>  
  
                    <Link href={`/${posts}/${post.id}`}>  
26                        // Estamos usando o componente Link do Next.js para criar links dinâmicos para cada post. Cada  
                        link redireciona para a página individual do post, cujo ID é passado como parte da URL.  
  
27                        {post.title}  
                        // Estamos exibindo o título do post dentro do link.  
  
28                    </Link>
```

REACT – API CONNECT FILE

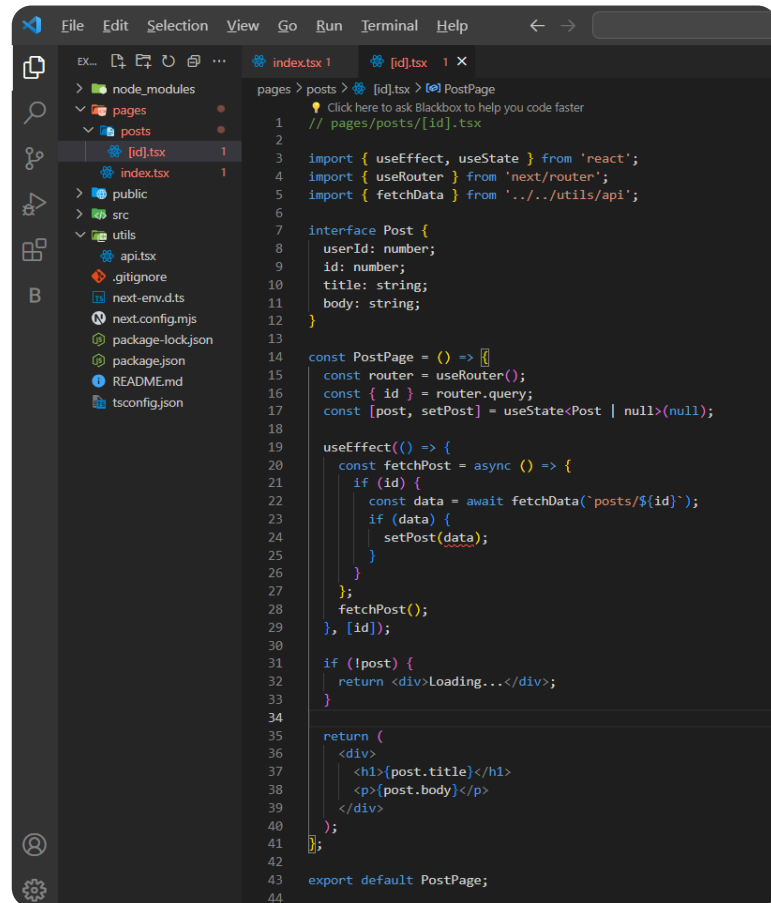
✓ Introdução ao API Connect File

- ✓ Analisando linha a linha o arquivo index.tsx

```
29         </li>
30     }}}
31 </ul>
32 </div>
33 )
34 }
35
36 export default HomePage;
37
```


REACT – API CONNECT FILE

- ✓ Introdução ao API Connect File
- ✓ Vamos criar o componente `pages > posts > [id].tsx`



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the following structure:

- node_modules
- pages
 - posts
 - [id].tsx (selected)
- public
- src
- utils
 - api.tsx
 - .gitignore
 - next-env.d.ts
 - next.config.mjs
 - package-lock.json
 - package.json
 - README.md
 - tsconfig.json

The code editor shows the content of the `[id].tsx` file:

```
1 // Click here to ask Blackbox to help you code faster
2 // pages/posts/[id].tsx
3
4 import { useEffect, useState } from 'react';
5 import { useRouter } from 'next/router';
6 import { fetchData } from '../../utils/api';
7
8 interface Post {
9   userId: number;
10   id: number;
11   title: string;
12   body: string;
13 }
14
15 const PostPage = () => {
16   const router = useRouter();
17   const { id } = router.query;
18   const [post, setPost] = useState<Post | null>(null);
19
20   useEffect(() => {
21     const fetchPost = async () => {
22       if (id) {
23         const data = await fetchData(`posts/${id}`);
24         if (data) {
25           setPost(data);
26         }
27       }
28     };
29     fetchPost();
30   }, [id]);
31
32   if (!post) {
33     return <div>Loading...</div>;
34   }
35
36   return (
37     <div>
38       <h1>{post.title}</h1>
39       <p>{post.body}</p>
40     </div>
41   );
42 }
43
44 export default PostPage;
```

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Antes vamos explicar novamente, como já visto em aulas anteriores, o porquê criar arquivos com o nome entre colchetes
 - ✓ A criação de um arquivo com colchetes `[id].tsx` em uma pasta `pages` no Next.js é uma convenção para criar uma rota dinâmica. Quando um arquivo é criado com colchetes em seu nome, isso indica ao Next.js que a rota associada a esse arquivo é dinâmica e pode receber parâmetros na URL.
 - ✓ Por exemplo, ao criar o arquivo `[id].tsx` dentro da pasta `pages/posts`, estamos indicando que queremos criar uma rota que pode receber um parâmetro `id` na URL. O `id` é então disponibilizado como parte do objeto `query` através do hook `useRouter`, permitindo-nos acessá-lo para buscar dados específicos na API.
 - ✓ Essa abordagem torna a criação de rotas dinâmicas muito mais fácil e limpa, pois não é necessário criar arquivos separados para cada rota dinâmica. Em vez disso, podemos usar um único arquivo `[param].tsx` e lidar com a lógica para diferentes valores de parâmetros dentro desse arquivo. Isso torna o código mais organizado e fácil de manter.

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Analisando linha a linha o arquivo [id].tsx

```
1 // pages/index.tsx
2
3 import { useEffect, useState } from 'react';
  // Aqui estamos importando os hooks useEffect e useState do React. O useEffect é usado para executar efeitos
  colaterais em componentes funcionais, enquanto o useState é usado para adicionar estado a componentes funcionais.
4 import { useRouter } from 'next/router';
  // Estamos importando o hook useRouter do Next.js. O useRouter é um hook que nos fornece acesso ao objeto router
  do Next.js, permitindo-nos acessar e manipular a rota atual.
5 import { fetchData } from '../utils/api';
  // Estamos importando a função fetchData do arquivo api.ts localizado na pasta utils. Essa função é usada para
  buscar os dados da API.
6
```

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Analisando linha a linha o arquivo [id].tsx

```
7 interface Post {  
8     userId: number;  
9     id: number;  
10    title: string;  
11    body: string;  
12 }  
13  
14 const PostPage = () => {  
    // Estamos definindo um componente funcional chamado PostPage.  
  
    const router = useRouter();  
15    // Estamos utilizando o hook useRouter para acessar o objeto router do Next.js, que contém informações sobre a  
    rota atual.  
  
    const { id } = router.query;  
16    // Estamos desestruturando o objeto query do router para obter o parâmetro id da rota. Esse parâmetro é  
    fornecido na URL da página.  
  
    const [post, setPost] = useState<Post | null>(null);  
17    // Estamos definindo um estado chamado post usando o hook useState. O estado inicial é null, indicando que  
    inicialmente não temos nenhum post carregado. Post | null é um tipo de união que indica que post pode ser um  
    objeto do tipo Post ou null. O setPost é uma função que usaremos para atualizar o estado post.  
18
```

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

- ✓ Analisando linha a linha o arquivo [id].tsx

```
19      useEffect(() => {  
        // Estamos utilizando o hook useEffect para buscar os detalhes do post da API sempre que o ID do post na URL  
        // mudar. O segundo argumento [id] indica que esta função de efeito será executada sempre que o valor de id mudar.  
  
        const fetchPost = async () => {  
          // Estamos definindo uma função assíncrona chamada fetchPost que usa a função fetchData para buscar os  
          // detalhes do post da API com base no ID fornecido.  
  
          if (id) {  
            const data = await fetchData(`posts/${id}`);  
  
            if (data) {  
              setPost(data);  
            }  
          }  
        }  
      };  
  
      fetchPost();  
      // Chamamos a função fetchPost para iniciar a busca dos detalhes do post assim que o componente PostPage for  
      // montado ou sempre que o ID do post mudar.  
  
    }, [id]);  
30
```

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

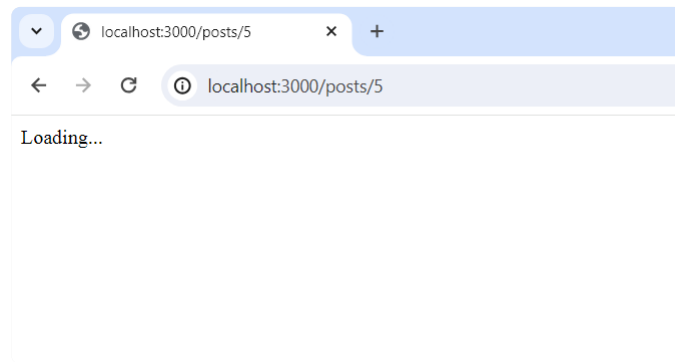
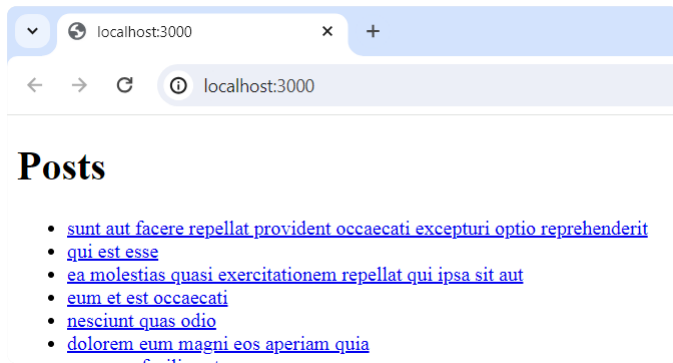
- ✓ Analisando linha a linha o arquivo [id].tsx

```
31     if (!post) {  
32         // Estamos verificando se o estado post é null. Se for, exibimos uma mensagem de carregamento.  
33         return <div>Loading...</div>;  
34     }  
  
35     return (  
36         // Finalmente, estamos retornando a estrutura JSX que representa os detalhes do post. Exibimos o título do post  
37         // dentro de um elemento h1 e o corpo do post dentro de um elemento p.  
38         <div>  
39             <h1>{post.title}</h1>  
40             <p>{post.body}</p>  
41         </div>  
42     );  
  
43     export default PostPage;
```

REACT – API CONNECT FILE

✓ Introdução ao API Connect File

✓ Resultado final



FRONT-END DESIGN ENGINEERING

ANTONIO, C. Pro React: Build Complex Front-End Applications in a Composable Way With React. Apress, 2015.

BOSWELL, D; FOUCHER, T. The Art of Readable Code: Simple and Practical Techniques for Writing Better Code. Estados Unidos: O'Reilly Media, 2012.

BRITO, Robin Cris. Android Com Android Studio - Passo A Passo. Editora Ciência Moderna.

BUNA, S. React Succinctly. Estados Unidos: [s.n], 2016. Disponível em: <www.syncfusion.com/ebooks/reactjs_succinctly>. Acesso em: 12 de janeiro de 2023.

FACEBOOK (2019a). React: Getting Started. React Docs, 2019. Disponível em: <reactjs.org/docs/react-api.html>. Acesso em: 13 de janeiro de 2023.

FACEBOOK (2019b). React Without ES6. React Docs, 2019. Disponível em: <reactjs.org/docs/react-without-es6.html>. Acesso em: 10 de janeiro de 2023.

FACEBOOK (2019c). React Without JSX. React Docs, 2019. Disponível em: <reactjs.org/docs/react-without-jsx.html>. Acesso em: 10 de janeiro de 2023.

FREEMAN, Eric ROBSON, Elisabeth. Use a Cabeça! Programação em HTML5. Rio de Janeiro: Editora Alta Books, 2014

GACKENHEIMER, C. Introduction to React: Using React to Build scalable and efficient user interfaces.[s.i.]: Apress, 2015.

HUDSON, P. Hacking with React. 2016. Disponível em: <www.hackingwithreact.com/read/1/3/introduction-to-jsx>. Acesso em: 13 janeiro de 2023.

KOSTRZEWA, D. Is React.js the Best JavaScript Framework in 2018? 2018. Disponível em: <hackernoon.com/is-react-js-the-best-javascript-framework-in-2018-264a0eb373c8>. Acesso em: janeiro de 2023.

MARTIN, R. Clean Code: A Handbook of Agile Software Craftsmanship. Estados Unidos: Prentice Hall, 2009.

MDN WEB DOCS. Guia JavaScript. Disponível em <<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide>>. Acessado em 29 de janeiro de 2023.

NELSON, J. Learn React's Fundamentals Without the Buzzwords? 2018. Disponível em: <jamesknelson.com/learn-react-fundamentals-sans-buzzwords>. Acesso em: 12 janeiro de 2023.

FRONT-END DESIGN ENGINEERING

NIELSEN, J. Response Times: The 3 Important Limits. 1993. Disponível em: <www.nngroup.com/articles/response-times-3-important-limits>. Acesso em: 10 janeiro de 2023.

O'REILLY, T. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. 2005. Disponível em: <www.oreilly.com/pub/a/web2/archive/what-is-web-20.html#mememap>. Acesso em: 10 de janeiro de 2023.

PANDIT, N. What Is ReactJS and Why Should We Use It? 2018. Disponível em: <www.c-sharpcorner.com/article/what-and-why-reactjs>. Acesso em: 12 de janeiro de 2023.

RAUSCHMAYER, A. Speaking JavaScript: An In-Depth Guide for Programmers. Estados Unidos: O'Reilly Media, 2014.

REACTIVA. O arquivo package-lock.json. Disponível em: <<https://nodejs.reativa.dev/0020-package-lock-json/index>>. Acessado em 13 de janeiro de 2023.

_____. O guia do package.json. Disponível em: <<https://nodejs.reativa.dev/0019-package-json/index>>. Acessado em 13 de janeiro de 2023.

RICOY, L. Desmitificando React: Uma Reflexão para Iniciantes. 2018. Disponível em: <medium.com/trainingcenter/desmitificando-react-uma-reflex%C3%A3o-para-iniciantes-a57af90b6114>. Acesso em: 13 janeiro de 2023.

SILVA, Maurício Samy. Ajax com jQuery: requisições Ajax com a simplicidade de jQuery. São Paulo: Novatec Editora, 2009.

_____. Construindo Sites com CSS e XHTML. Sites Controlados por Folhas de Estilo em Cascata. São Paulo: Novatec, 2010.

_____. CSS3 - Desenvolva aplicações web profissionais com o uso dos poderosos recursos de estilização das CSS. São Paulo: Novatec Editora, 2010.

STACKOVERFLOW. Most Popular Technologies: Web Frameworks. Developer Survey Results, StackOverflow, 2019. Disponível em: <insights.stackoverflow.com/survey/2019#technology>. Acesso em: 13 de janeiro de 2023.

W3C. HTML5 - A linguagem de marcação que revolucionou a web. São Paulo: Novatec Editora, 2010.

_____. A vocabulary and associated APIs for HTML and XHTML. Disponível em <<https://www.w3.org/TR/2018/SPSD-html5-20180327/>>. Acessado em 28 de abril de 2020, às 20h53min.

FRONT-END DESIGN ENGINEERING

W3C. Cascading Style Sheets, level 1. Disponível em <<https://www.w3.org/TR/2018/SPSD-CSS1-20180913/>>. Acessado em 28 de abril de 2020, às 21h58min.

_____. Cascading Style Sheets, level 2 Revision 2. Disponível em <<https://www.w3.org/TR/2016/WD-CSS22-20160412/>>. Acessado em 28 de abril de 2020, às 22h17min.

_____. Cascading Style Sheets, level 2. Disponível em <<https://www.w3.org/TR/2008/REC-CSS2-20080411/>>. Acessado em 28 de abril de 2020, às 22h03min.

_____. Cascading Style Sheets, level 3. Disponível em <<https://www.w3.org/TR/css-syntax-3/>>. Acessado em 28 de abril de 2020, às 22h18min.

_____. HTML 3.2 Reference Specification. Disponível em <<https://www.w3.org/TR/2018/SPSD-html32-20180315/>>. Acessado em 28 de abril de 2020, às 19h37min.

_____. HTML 4.0 Specification. Disponível em <<https://www.w3.org/TR/1998/REC-html40-19980424/>>. Acessado em 28 de abril de 2020, às 19h53min.

_____. HTML 4.01 Specification. Disponível em <<https://www.w3.org/TR/2018/SPSD-html401-20180327/>>. Acessado em 28 de abril de 2020, às 20h04min.

_____. Cascading Style Sheets, level 2 Revision 1. Disponível em <<https://www.w3.org/TR/CSS2/>>. Acessado em 28 de abril de 2020, às 22h13min.

WIKIPEDIA. JavaScript. Disponível em <<https://pt.wikipedia.org/wiki/JavaScript>>. Acessado em 29 de abril de 2020, às 10h.