React com Nexjts

Prof: Rafael Vinicius Germinari Gonçalves

Mercado

Média salarial por Frameworks / Ferramentas

Perguntamos quais eram as principais ferramentas e frameworks que o participante costuma usar mais no dia a dia.

01º .NET (Standard, Core, Framework) R$ 9.033,27 2166 participantes	02º Spring Boot R$ 10.405,61 1739 participantes	03º React R$ 8.441,70 1463 participantes
04º Node.js R$ 9.883,38 1046 participantes	05º Outro R$ 9.211,92 953 participantes	06º Nenhum R$ 8.659,86 800 participantes
07º Laravel R$ 8.141,53 787 participantes	08º Angular R$ 7.958,29 628 participantes	09º Flutter R$ 8.810,65 453 participantes

https://pesquisa.codigofonte.com.br/2024

O que é React ?

Uma biblioteca JavaScript para a construção de interfaces de usuário (UIs).

Foi criada pelo Facebook e é amplamente utilizada para desenvolver interfaces de usuário dinâmicas e interativas.

A principal vantagem do React é sua abordagem declarativa e baseada em componentes, o que facilita a construção e manutenção de UIs complexas

React Documentação

A documentação do react pede para que a gente inicie um novo projeto react já com alguns framework popular na comunidade como:

- Nextjs
- Gatsby
- Remix
- Expo

O que é Framework ?

frameworks de software contêm módulos de código reutilizáveis com base em padrões e protocolos de software específicos.

Frameworks também podem definir e aplicar determinadas regras de arquitetura de software ou processos de negócios, para que novas aplicações possam ser desenvolvidas de maneira padronizada.

O que é Nextjs

Next.js é um poderoso framework para React que simplifica a construção de aplicações web robustas, rápidas e escaláveis, fornecendo uma série de funcionalidades avançadas que não estão disponíveis no React puro. Ele melhora a performance, SEO, e a experiência de desenvolvimento, tornando-se uma escolha popular entre desenvolvedores front-end.

Tecnologias do Nextjs

React

Biblioteca de javascript para a construção de interfaces de usuário

Tailwind

Tailwind CSS é uma biblioteca de utilitários CSS que permite criar designs personalizados de forma rápida e eficiente.

Nodejs

Node.js é uma plataforma de runtime para JavaScript que permite a execução de código JavaScript fora do navegador.

Tecnologias do Nextjs

Front end

React

Biblioteca de javascript para a construção de interfaces de usuário

Tailwind

Tailwind CSS é uma biblioteca de utilitários CSS que permite criar designs personalizados de forma rápida e eficiente.

Back end

Nodejs

Node.js é uma plataforma de runtime para JavaScript que permite a execução de código JavaScript fora do navegador.

Iniciar projeto

“ *npx create-next-app@latest* ”

Criando o projeto

Vamos iniciar a criação de um projeto utilizando Next.js em conjunto com TypeScript. Para começar, criaremos uma pasta que servirá como repositório para os arquivos do nosso projeto. Utilizaremos o comando "**npx create-next-app@latest ./**" para gerar os arquivos iniciais do nosso projeto.

- **ESlint:** identificar possíveis problemas de código
- **TailWind CSS:** Tailwind CSS é um framework de CSS utilitário. Ele fornece classes pré-definidas que você pode usar para estilizar seus elementos HTML
- src(source): Apenas perguntando se gostaria de adicionar o projeto dentro da pasta src.
- App Router: Sistema de roteamento usado para navegação entre as paginas e outras coisas.
- Import alias: Apenas o nome que vai ser usado na importação.

```
PS D:\Area de trabalho\senac\aula-react-nextjs> npx create-next-app@latest ./
√ 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
```

Para executar o projeto utilizamos o comando "npm run dev"

Estrutura do projeto

Estrutura basica de página

Isso permite que o componente seja exportado como a exportação padrão do arquivo.

É utilizado para exportar um único valor ou entidade como a "exportação padrão" de um módulo

```
export default function Index(){
  return (
    <div>
      <h1>Hello world</h1>
    </div>
  );
}
```

O retorno dentro do componente é a parte onde o JSX é renderizado. Esse JSX descreve como a interface do usuário deve se parecer quando o componente for exibido.

JSX e TSX

JSX é uma extensão da sintaxe JavaScript que permite incorporar estruturas semelhantes a **XML** (como HTML) diretamente no código JavaScript. **JSX** é frequentemente associado ao **React**, mas também é utilizado em outras bibliotecas/frameworks. Permite a criação de componentes de interface de usuário de uma maneira mais declarativa.

Portanto, um arquivo com a extensão **.tsx** contém código **TypeScript** que pode incluir **JSX**. Isso é comumente usado em projetos **React** para escrever componentes React com **TypeScript**, aproveitando as vantagens da tipagem estática e da sintaxe **JSX**.

```
1  var nome = "Hello World"
2  export default function Home() {
3    return (
4      <>
5      <h1>{nome}</h1>
6      </>
7    )
8  }
```

Nome do arquivo page.tsx

Tailwind

O Tailwind CSS é um framework CSS baseado em utilitários (utility-first), onde você escreve classes diretamente no HTML para estilizar os elementos.

Vantagens

- ✅ Muito rápido para estilizar.
- ✅ Não precisa sair do HTML para escrever CSS.
- ✅ Altamente personalizável.
- ✅ Reduz código CSS repetido.

Sem

```
.botao {
    background-color: blue;
    color: white;
    padding: 1rem;
    border-radius: 0.5rem;
  }
```

```
<button class="botao">Clique aqui</button>
```

Com

```
1  <button className="bg-blue-
   500 text-white p-4 rounde
   d">Clique aqui</button>
```

Exercicio 1

- Criar uma barra de navegação simples com Tailwind
- Uso de **flex** e **space-x**
- Personalização básica de cores e espaçamentos

Exercicio 2

- Criar uma barra de navegação lateral com Tailwind
- Personalização básica de cores e espaçamentos

Exercicio 3

- Tela de login simples
- utilize tags do tailwind **focus:ring-blue-500**
- **focus:outline-none**
- **focus:ring-2**
- **shadow-lg**

Componentes

🧩 O que é um componente?
Um componente é como um bloco de construção reutilizável da interface do seu aplicativo.
É uma **função** (ou classe, mas hoje usamos mais função) que retorna uma parte da tela — um **"pedacinho do app"**.

```
1   function MeuComponente(){
2     return(
3       <div>Hello World</div>
4     );
5   }
6   export default function Home() {
7     return (
8       <>
9         <MeuComponente/>
10      </>
11    )
12  }
```

Sempre coloque os nomes dos componentes em Maiúsculo

TypeScript

Foi desenvolvido pela Microsoft e é mantido pela comunidade open source. Em essência, o TypeScript permite aos desenvolvedores escrever código JavaScript que pode incluir tipos estáticos, fornecendo assim benefícios significativos durante o desenvolvimento e manutenção de projetos.

```
1  const add = (a:number, b:number) => {
2  }
3
```

```
1  function add({a,b}:{a:number,b:number}){
2    return a+b;
3  }
```

Propriedades

type Props = { ... } define as propriedades esperadas pelo componente.

O componente BoasVindas recebe os dados com esse tipo (Props).

Isso ajuda o VS Code a mostrar erro se esquecer ou errar o tipo de uma prop.

```
1   type Props = {
2       nome: string;
3       idade: number;
4   };
5
6   export default function BoasVindas({ nome, idade }: Props) {
7       return (
8           <div className="p-4 bg-green-100 rounded">
9               <h1 className="text-xl font-bold">Olá, {nome}!</h1>
10              <p>Você tem {idade} anos.</p>
11          </div>
12      );
13  }
```

💡 O que são types?

types no TypeScript permitem descrever a forma de um dado — ou seja, como um objeto, função ou variável deve ser estruturado.

✅ **Vantagens:**

- Autocompletar e IntelliSense no editor.
- Evita erros como acessar uma propriedade que não existe.
- Deixa o código mais seguro, claro e fácil de manter.

```
1   type usuario = {
2     id: number;
3     nome: string;
4     idade: number;
5     endereco:{
6       rua: string;
7       cidade: string;
8       estado: string;
9     }
10    tarefas: string[];
11  };
12
```

Propriedades children

No **React**, **children** é uma propriedade especial que permite que um componente inclua elementos ou componentes filhos entre suas tags de abertura e fechamento. Isso facilita a composição de componentes e a criação de estruturas de componentes mais flexíveis e reutilizáveis. Quando você utiliza a propriedade **children**, você está permitindo que outros componentes ou elementos sejam passados como filhos para dentro do componente pai.

```
1  function MeuComponente({  children }: { children:React.ReactNode }) {
2    return (
3      <>
4        <li>{children}</li>
5      </>
6    );
7  }
8  export default function Home() {
9    return (
10     <>
11       <MeuComponente>
12         <div>
13           <h1>Hellow World</h1>
14         </div>
15       </MeuComponente>
16     </>
17   )
18 }
```

🧩 **Eles também podem ser um pouco mais complexos.**

para deixar uma propriedade como “não obrigatória” é utilizado o símbolo “?” antes dos :

```
1  type Props = {
2      titulo: string;
3      cor: 'azul' | 'vermelho';
4      subtitulo?: string;
5      funcao?: () => void;
6      className?: string;
7  };
8
9  export default function Button({ titulo, cor, subtitulo, funcao, className }: Props) {
10     let corBase = '';
11
12     switch (cor) {
13         case 'azul':
14             corBase = 'bg-blue-900';
15             break;
16         case 'vermelho':
17             corBase = 'bg-red-500';
18             break;
19         default:
20             break;
21     }
22
23     return (
24         <button
25             onClick={funcao}
26             className={`${corBase} ${className} text-white p-4 rounded-lg`}
27         >
28             <div className="text-2xl font-bold">{titulo}</div>
29             {subtitulo && <div className="text-sm opacity-80">{subtitulo}</div>}
30         </button>
31     );
32 }
33
```

Eventos

No ecossistema React, eventos referem-se a interações do usuário ou outras ações que acionam a execução de código específico. Essas interações englobam uma variedade de ações, como cliques do mouse, teclas pressionadas e envio de formulários.

Quando se trata da passagem de propriedades (props) para funções que serão invocadas, é comum utilizar a sintaxe **() =>** para indicar que se trata de uma função anônima. Essa abordagem permite a transferência de dados dinâmicos e configurações entre componentes, promovendo a flexibilidade e reusabilidade no desenvolvimento de aplicações React.

```
1  function handleClick() {
2    alert("Click");
3  }
4
5  export default function Home() {
6    return (
7      <>
8        <button onClick={handleClick}>click</button>
9      </>
10   );
11 }
```

Evento normal

```
1  function handleClick({int}: {int:number}) {
2    alert(`${int}`);
3  }
4
5  export default function Home() {
6    return (
7      <>
8        <button onClick={() => handleClick({int:34})}>click</button>
9      </>
10   )
11 }
```

Evento com passagem de propriedade

Componentes – exercício 1

Utilize essa estrutura para desenvolver o exercício:

```
/app
  ├── page.tsx
  └── components/
        ├── Header.tsx
        ├── Card.tsx
        └── Footer.tsx
```

Componentes – exercício 2

- Usar **children** para injetar conteúdo
- Passagem de **funçao** como parametro
- As props devem ser **titulo, children e acao**

Área do Usuário

Bem-vindo ao sistema! esse pedaço veio do children.

Executar ação

Componentes – exercício 3

Menu

Início

Usuários

Sair

Lista de Usuários

Alice
alice@email.com
Ver detalhes

Bruno
bruno@email.com
Ver detalhes

Carla
carla@email.com
Ver detalhes

Listas map

No contexto de JavaScript, o método **map()** surge como uma função de array que possibilita a iteração sobre cada elemento de um array, efetuando uma operação específica para cada um deles. Essa operação resulta na criação de um novo array contendo os resultados dessas transformações. Importante ressaltar que o método **map()** não altera o array original, mas gera um novo conjunto de resultados.

No código fornecido, uma constante denominada **lista** é declarada, armazenando uma matriz de números de 1 a 9. Em seguida, a expressão **{lista.map((numeros, index) => (...))}** utiliza o método **map()** para percorrer cada elemento da matriz **lista**.

Para cada número presente, é criado um elemento **<li>** (item de lista) no JSX. A propriedade **key={index}** é empregada para fornecer uma chave única a cada item, uma prática essencial para que o **React** possa identificar quais itens foram adicionados, removidos ou alterados durante atualizações.

Cada **<li key={index}>{numeros}</li>** é responsável por renderizar um elemento **<li>** para cada número na lista, exibindo o próprio número. Para melhor organização, todos esses elementos **<li>** são encapsulados em um elemento ** **. Essa prática é comum para agrupar múltiplos elementos JSX e proporcionar uma estrutura mais organizada no código.

```
1  function MeuComponente(){
2    const lista = [1,2,3,4,5,6,7,8,9]
3
4    return(
5      <div>
6        {lista.map((numeros,index) => (
7          <li key={index}>
8            {numeros}
9          </li>
10       ))}
11     </div>
12   );
13 }
14 export default function Home() {
15   return (
16     <>
17       <MeuComponente/>
18     </>
19   )
20 }
```

🤔 O que é o if ternário?

O operador ternário é uma forma compacta de escrever uma condição if/else. Ele tem esta estrutura:

condição ? valorSeVerdadeiro : valorSeFalso

✅ **Como funciona no React?**
Você vai usá-lo dentro do JSX para mostrar ou esconder algo, ou alterar conteúdo com base em uma condição.

```
1  export default function Index() {
2    const estaLogado = true;
3
4    return (
5      <div>
6        {estaLogado ? <p>Bem-vindo!</p> : <p>Por favor, faça login.</p>}
7      </div>
8    );
9  }
```

UseState hook

O que é o useState?
useState é um hook do React que permite armazenar e atualizar valores no estado de um componente. Ele é muito usado para controlar inputs, exibir/ocultar elementos, armazenar contadores, entre outros.

você precisa adicionar **'use client'** no topo do arquivo para habilitar o uso de **useState**

```
1  'use client';
2
3  import { useState } from 'react';
4
5  export default function Index() {
6    const [contador, setContador] = useState(0);
7
8    function incrementar() {
9      setContador(contador + 1);
10   }
11   return (
12     <div className="p-4 border rounded w-fit">
13       <p className="text-lg">Valor atual: {contador}</p>
14       <button
15         onClick={incrementar}
16         className="mt-2 px-4 py-2 bg-blue-500 text-white rounded"
17       >
18         Somar +1
19       </button>
20     </div>
21   );
22 }
```

📌 **Exercício 1 – Contador com incremento e decremento**
Descrição: Crie dois botões: um para somar e outro para subtrair.
Extra: Adicione uma verificação para que o número nunca fique abaixo de zero.

📌 **Exercício 2 – Mostrar/esconder texto**
Descrição: Crie um botão que ao ser clicado alterna entre mostrar e esconder um parágrafo de texto.
Dica: Use boolean no useState.

📌 **Exercício 3 – Campo de nome com mensagem**
Descrição: Crie um campo de input. Conforme o usuário digita, exiba abaixo "Olá, [nome]!"
Extra: Se o campo estiver vazio, exiba "Digite seu nome".

📌 **Exercício 4 – Lista de tarefas simples**
Descrição: Crie um campo para digitar uma tarefa e um botão para adicioná-la a uma lista (array).
Dica: Use useState([]) e atualize com setTarefas([...tarefas, novaTarefa])

📌 **Exercício 5 – Contador de cliques em várias cores**
Descrição: Mostre três botões com cores diferentes. Ao clicar em cada botão, ele soma +1 ao seu próprio contador.
Exemplo: Vermelho: 3 cliques, Azul: 5 cliques, Verde: 1 clique.

📌 **Exercício 6 – Simulador de likes**
Descrição: Crie um botão de "Curtir" que alterna entre "Curtir" e "Descurtir", e mostra o total de curtidas.
Dica: Use um booleano para o estado do botão e um número para o total de curtidas.

API (Application Programming Interface)

API, que significa Interface de Programação de Aplicações (em inglês, Application Programming Interface), é um conjunto de regras e definições que permite a interação entre diferentes softwares. Em outras palavras, uma API define como os componentes de software devem se comunicar.

As APIs são fundamentais para o desenvolvimento de software, pois permitem que diferentes sistemas, plataformas e linguagens de programação se comuniquem de maneira eficiente. Elas fornecem um conjunto de funções, procedimentos e protocolos que facilitam a integração de aplicativos e serviços.

Existem vários tipos de APIs, incluindo APIs web, que são amplamente utilizadas na integração de sistemas pela internet. As APIs web permitem que aplicativos se comuniquem por meio de solicitações HTTP/HTTPS, geralmente utilizando formatos de dados como JSON ou XML para a troca de informações.

As APIs podem ser públicas, permitindo acesso a desenvolvedores externos, ou privadas, restritas a um conjunto específico de usuários. Elas desempenham um papel crucial na construção de ecossistemas de software, possibilitando a criação de aplicativos mais poderosos e a integração de diferentes serviços e plataformas.

Como funciona

A dinâmica da API opera da seguinte maneira: o cliente inicia uma requisição à API, que, por sua vez, submete a solicitação ao banco de dados. O banco de dados responde com os dados correspondentes, os quais a API converte para o formato JSON ou XML.

Finalmente, a API retorna essa resposta formatada para o cliente que a requisitou. Essa interação eficiente permite a comunicação fluida e a transferência de informações entre o cliente e o banco de dados, facilitando a integração e a obtenção de dados de forma organizada e estruturada.

Server x client componentes

Uso de uma API Server Side

```
1  type usersProps = {
2      id: number;
3      name: string;
4      username: string;
5      email: string;
6  };
7  //server side
8  export default async function UserPage() {
9
10     const response = await fetch('https://jsonplaceholder.typicode.com/users');
11     const users: usersProps[] = await response.json();
12     console.log(users);
13     return (
14         <div>
15             <h1 className="text-2xl font-bold mb-4">Lista de Usuários</h1>
16             <ul className="space-y-2">
17                 {users.map(user => (
18                     <li key={user.id} className="p-4 bg-gray-100 rounded shadow">
19                         <h2 className="text-xl font-semibold">{user.name}</h2>
20                         <p className="text-gray-600">Username: {user.username}</p>
21                         <p className="text-gray-600">Email: {user.email}</p>
22                     </li>
23                 ))}
24             </ul>
25
26         </div>
27     )
28 }
```

Uso de uma API Client Side

```
'use client'

import { useEffect, useState } from "react";


type User = {
    id: number;
    name: string;
    email: string;
};

export default function ClientSide() {
    const [users, setUsers] = useState<User[]>([]);

    async function fetchUsers() {
        const response = await fetch('https://jsonplaceholder.typicode.com/users');
        const data = await response.json();
        setUsers(data);
    }

    useEffect(() => {
        fetchUsers();
    }, []);

    return (
        <div>
            <h1>Usuários</h1>
            <ul>
                {users.map(user => (
                    <li key={user.id}>{user.name}</li>
                ))}
            </ul>
        </div>
    );
}
```

Client side dentro de Server Side

Posso criar um componente button que é cliente side pelo uso do "use client" e invocalo dentro de uma pagina que é server side.

o que eu não posso fazer é tentar invocar um componente server side dentro de um client side.

```
1  'use client'
2
3  import { useState } from "react";
4
5
6  export default function Button() {
7      const [count, setCount] = useState(0);
8      return (
9          <button className="bg-blue-500 text-white p-4 rounded-lg" onClick={() => setCount(count + 1)}>
10             Clique Aqui {count}
11         </button>
12     );
13 }
```

Exercício API

Criar uma tela que consome a Fake Store API e exibe uma lista de produtos, com nome, imagem, preço e descrição curta.
URL: https://fakestoreapi.com/docs

Rotas

app
dashboard
page.js
settings
page.js
api
route.js
/dashboard
Routable
/dashboard/settings
Routable
/api
Routable

Link

Link é um componente fornecido pelo Next.js que permite criar links para navegar entre as páginas em sua aplicação.

Ele é usado principalmente para navegação declarativa do lado do cliente. Ao clicar em um link, ele impede a atualização padrão completa da página no navegador e busca o novo conteúdo da página no lado do cliente, resultando em uma experiência do usuário mais rápida e suave.

Geralmente, é utilizado em seu código JSX ou TSX de componentes para criar links clicáveis que direcionam para outras páginas dentro da sua aplicação Next.js.

```
1  import Link from "next/link"
2
3  export default function Home() {
4    return (
5      <>
6        <Link href={"/Exemplo1"}>Click</Link>
7      </>
8    )
9  }
```

href é o caminho da pasta onde contem o arquivo page

useRouter

useRouter é um hook fornecido pelo Next.js que permite acessar o objeto do roteador e suas propriedades dentro do seu componente.

Ele fornece controle programático sobre o roteador e permite navegar para diferentes páginas ou realizar outras ações relacionadas ao roteamento em resposta a interações ou eventos do usuário dentro do seu componente.

É útil quando você precisa lidar com navegação ou acessar informações específicas da rota diretamente na lógica do seu componente.

É importante importar o useRouter do next navigation

```
1  'use client'
2  import { useRouter } from "next/navigation";
3
4  export default function Home() {
5    const route = useRouter();
6
7    return (
8      <>
9        <button onClick={() => route.push("/Exemplo1")}>Click</button>
10     </>
11   )
12 }
```

A função push vai enviar o usuário para a rota em questão.

Route Groups

Pastas aninhadas são normalmente mapeadas para caminhos de URL. No entanto, você pode marcar uma pasta como um Grupo de Rota para evitar que ela seja incluída no caminho da URL da rota.
Isso permite que você organize seus segmentos de rota e arquivos do projeto em grupos lógicos sem afetar a estrutura do caminho da URL.

Os grupos de rotas são úteis para:

- Organizar rotas em grupos, por exemplo, por seção do site, intenção ou equipe.
- Possibilitar layouts aninhados no mesmo nível do segmento de rota:
  - Criar vários layouts aninhados no mesmo segmento, incluindo vários layouts raiz.
  - Adicionar um layout a um subconjunto de rotas em um segmento comum.

Um grupo de rota pode ser criado envolvendo o nome de uma pasta entre parênteses: **(nomeDaPasta).**

Rota dinamica

As rotas dinâmicas (**Dynamic Routes**) no Next.js permitem criar páginas que podem se adaptar a diferentes parâmetros na URL. Isso é útil quando você deseja criar páginas que compartilham a mesma estrutura, mas cujo conteúdo depende de valores específicos.

A sintaxe para criar rotas dinâmicas no Next.js envolve a utilização de colchetes (**[]**) dentro do nome do arquivo da página, indicando que aquele segmento da rota é dinâmico e pode receber diferentes valores.

Rota dinamica

```
1  import { promises } from "dns";
2
3
4  type postsProps = {
5      id: number;
6      title: string;
7  };
8
9  export default async function Page({ params }: { params: Promise<{ id: string }> }) {
10     const { id } = await params;
11
12     const response = await fetch(`https://jsonplaceholder.typicode.com/posts?userId=${id}`);
13     const posts: postsProps[] = await response.json();
14
15     return (
16         <div className="p-4">
17             <h1 className="text-2xl font-bold">Rota Dinâmica</h1>
18             <p className="mt-2">ID: {id}</p>
19             <h2 className="text-xl font-semibold mt-4">Posts:</h2>
20             <ul className="mt-2">
21                 {posts.map((post) => (
22                     <li key={post.id} className="border-b py-2">
23                         {post.title}
24                     </li>
25                 ))}
26             </ul>
27         </div>
28     );
29 }
```

Loading ui e streaming

Loading.tsx

🧩 loading.tsx é um componente de fallback

É automaticamente renderizado enquanto a rota (ou parte dela) está sendo carregada no servidor.

🔧 Quando é mostrado?

- Enquanto o componente assíncrono de page.tsx ainda está sendo renderizado no servidor.
- Especialmente útil quando page.tsx faz fetch de dados demorados (ex: banco externo, API).

Por que usar?

- Melhorar experiência do usuário com feedback imediato.
- Evitar páginas brancas ou travadas.
- Controlar exatamente o que é exibido durante o carregamento.

OBS: ele vai carregar apenas o componente onde ele está caso tenha um menu dentro da layout.tsx o layout vai ser carrego primeiro e depois a pagina entra em loading dos conteudo da pagina.

Loading.tsx

```
1  export default function Loading() {
2      return (
3          <div className="flex items-center justify-center h-screen">
4              <div className="animate-spin rounded-full h-16 w-16 border-b-2 border-blue-500"></div>
5          </div>
6      );
7  }
```

Comando para simular atraso na pagina
usar antes da requisição dos dados

```
await new Promise((resolve) => setTimeout(resolve, 4000)); // Simulando um atraso de 4 segundos
```

Streaming

Streaming é a capacidade do Next.js enviar parte do HTML ao navegador conforme vai ficando pronto, sem esperar tudo carregar.

🔍 Antes: sem streaming
Tudo é montado no servidor → só então enviado → navegador mostra a página inteira.
⏱️ Resultado: página demora mais para aparecer.

✅ Com Streaming (via React Server Components)

- Next.js começa a enviar o HTML imediatamente.
- Vai "streamando" partes conforme ficam prontas.
- O usuário vê a interface em pedaços, mas mais rápido.

Exercicio

📌 Enunciado do Exercício
Título: Página de Detalhes com Streaming e Fallback
Descrição:
Quando o aluno clicar no botão “Ver mais” de um produto, ele deve ser redirecionado para a página: **/produtos/ID**

Crie essa rota dinâmica (**[id]/page.tsx**) que:

- Busca os dados do produto por ID:

https://fakestoreapi.com/products/{id}

Mostra a imagem grande, título, preço, categoria, descrição completa.

- Crie um **loading.tsx** dentro da pasta [id] para exibir um fallback durante o carregamento.
- (Extra) Mostre um botão "Voltar para produtos".