

Desenvolvimento Front-end com Frameworks

Fundamentos de React

Agenda

Etapas 3: Programas Assíncronos.

- JavaScript Assíncrono.
- Estados dos Componentes.
- Construção de Componentes Básicos.



JavaScript Assíncrono

Quando escrevemos código **JavaScript síncrono**, fornecemos uma lista de **instruções que são executadas imediatamente na ordem**.

Por exemplo, se quiséssemos usar JavaScript para lidar com alguma manipulação simples de DOM, escreveríamos o código para fazer isso.

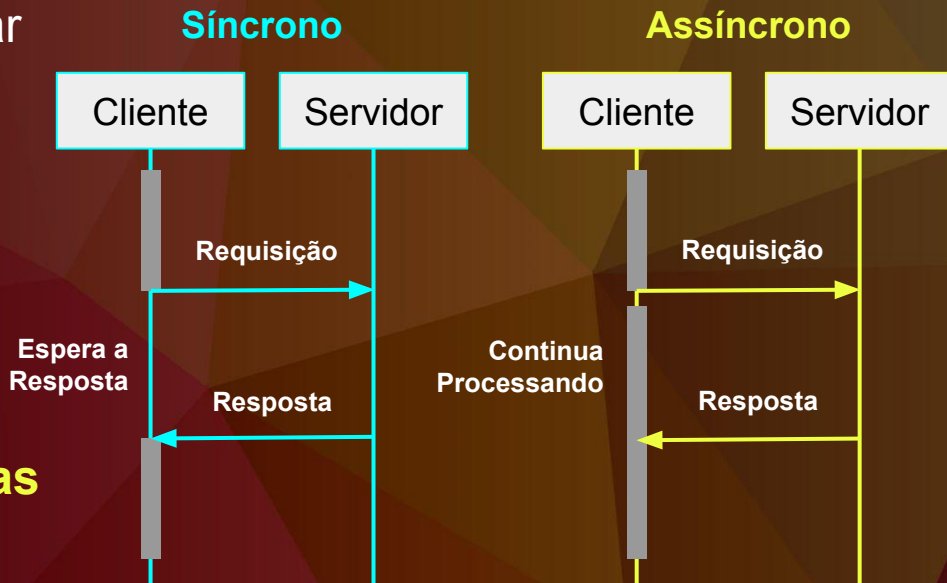
Enquanto cada operação está acontecendo, nada mais está acontecendo.

```
const header = document.getElementById("heading");  
header.innerHTML = "Hey!";
```

Com a web moderna, precisamos realizar tarefas assíncronas:

- Acessar um banco de dados.
- Transmitir conteúdo de vídeo ou áudio.
- Buscar dados de uma API.

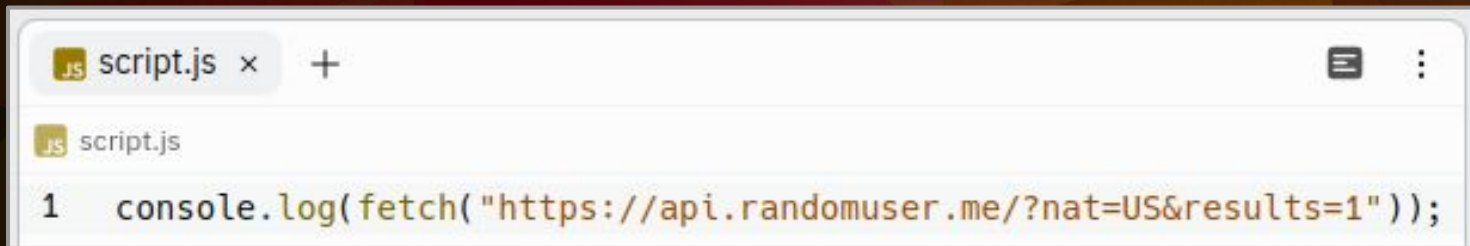
Com JavaScript, as tarefas assíncronas não bloqueiam o processo principal.



Fazer uma solicitação para uma API costumava ser bastante complicada em JavaScript.

Tínhamos que escrever mais de 20 linhas de código aninhado apenas para carregar alguns dados em nosso aplicativo.

Então a **função fetch apareceu e simplificou nossas vidas**.

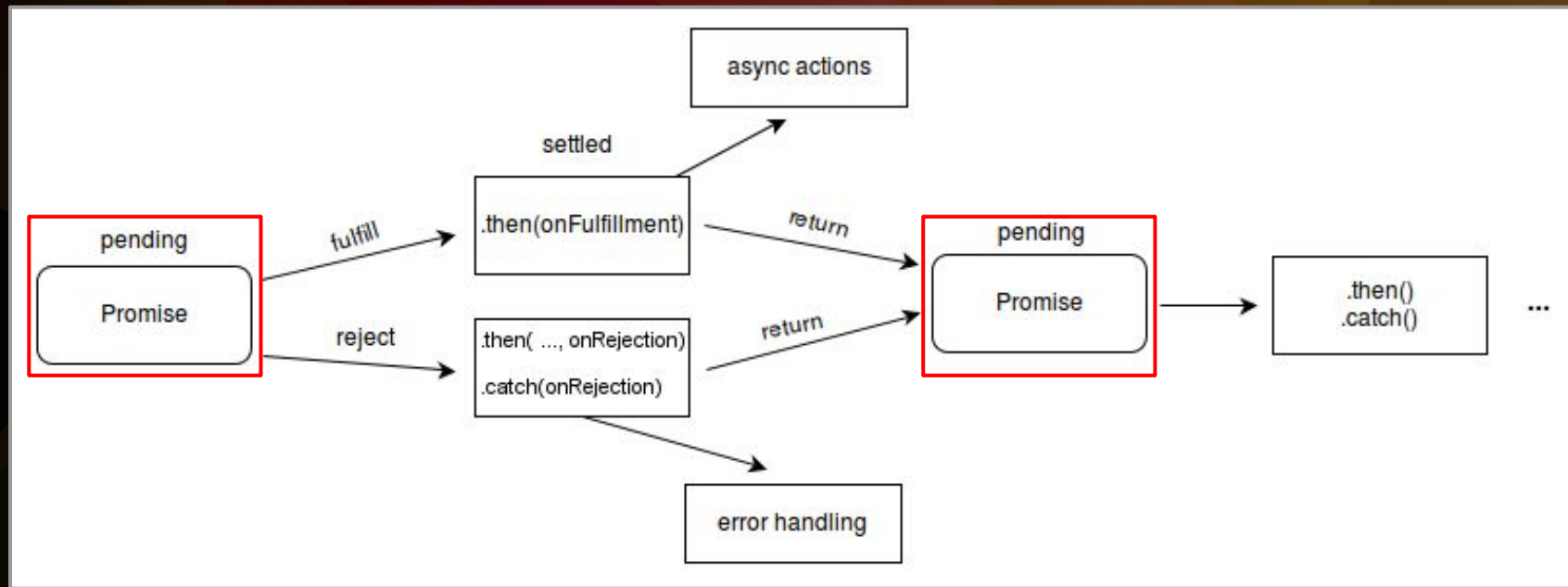


```
.js script.js x +  
.js script.js  
1 console.log(fetch("https://api.randomuser.me/?nat=US&results=1"));
```

Quando rodamos o **fetch**, vemos que retorna um objeto **Promise** (promessa).

A promessa é um objeto que representa se a operação assíncrona está pendente, foi concluída ou falhou.

Pense nisso como o navegador dizendo: “Ei, vou tentar o meu melhor para obter esses dados. De qualquer forma, voltarei para contar como foi.”



Promise representa a eventual conclusão (ou falha) de uma operação assíncrona e seu valor resultante.

Promise é um proxy para um valor não necessariamente conhecido quando o objeto é criado.

Ele permite que você associe os manipuladores ao valor de sucesso ou ao motivo da falha de uma ação assíncrona.

Isso permite que métodos assíncronos retornem valores como métodos síncronos: em vez de retornar imediatamente o valor final, o método assíncrono retorna uma promessa de fornecer o valor em algum momento no futuro.

async e **await** são uma sintaxe que simplifica a programação assíncrona, facilitando o fluxo de escrita e leitura do código; assim é possível escrever código que funciona de forma assíncrona, porém **é lido e estruturado de forma síncrona**.

Definindo uma função como **async**, podemos utilizar a palavra-chave **await** antes de qualquer expressão que retorne uma promessa. Dessa forma, a execução da função externa (a função `async`) será pausada até que a Promise seja resolvida.

Estados dos Componentes

Estados dos Componentes

Dados é o que dá vida aos nossos componentes **React**.

Nossas interfaces de usuário são ferramentas que precisam saber como manipular e alterar dados de maneira eficaz.

O estado de um aplicativo **React** é orientado por dados que podem ser alterados e que interferem diretamente em como nossas interfaces são apresentadas.

O exemplo anterior utilizou propriedades, mas produziu componentes visuais estáticos. Agora vamos implementar a ação de clicar no componente para que a avaliação seja obtida através do usuário.

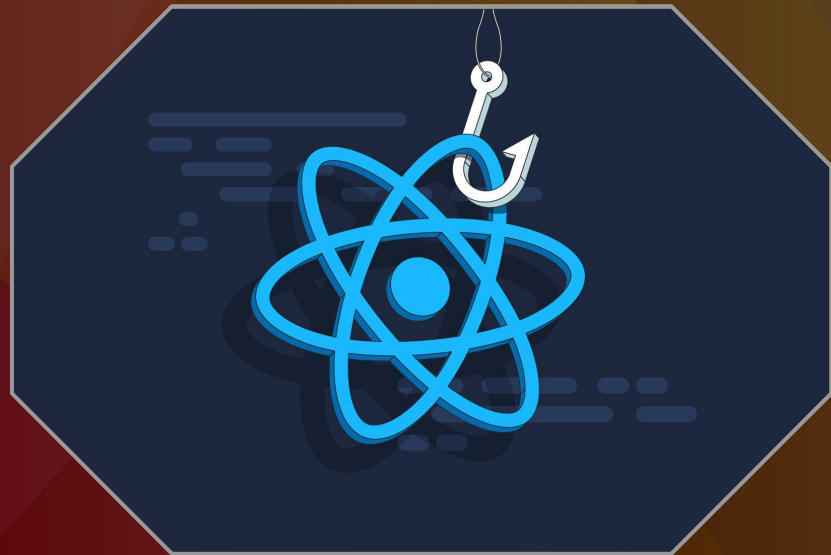
O Que São Hooks?

Os hooks são funções do **React** que permitem fazer conexões de elementos ao ciclo de vida da aplicação.

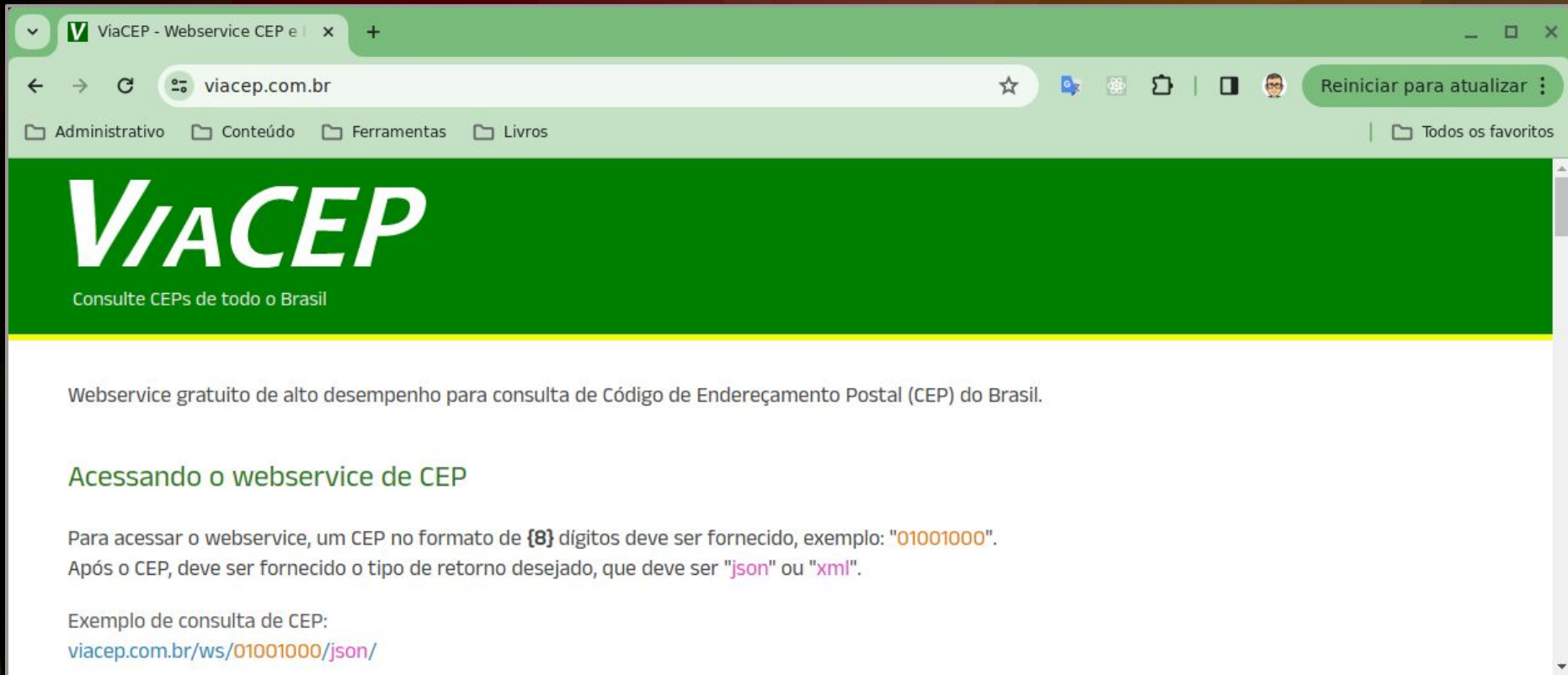
Eles foram criados especificamente para componentes funcionais.

O *hook* **useState** nos permite rastrear o estado em um componente.

Estado geralmente se refere a dados que precisam ser rastreados em um aplicativo.



Construção de Componentes Básicos



codesandbox.io/dashboard/recent

Google Lens

Create

Popular

Browser

Server

Frontend

Backend

Workspace

react



React

CodeSandbox

8.2M



React (TS)

CodeSandbox

235.5k



Next.js

CodeSandbox

107.4k



Preact

CodeSandbox

19.5k



React + Tailwind

CodeSandbox

14.5k



React (JS)

CodeSandbox

14.5k



React Native with Expo

CodeSandbox

9.0k



AI Code Completion

Codelum

4.5k



Create-T3-app

CodeSandbox

3.0k



Gatsby

CodeSandbox

1.5k



Docusaurus

CodeSandbox

1.1k



Storybook (React)

CodeSandbox

386



React + Chakra

CodeSandbox

374



Hono + Next.js

CodeSandbox

202



React (Rsbuild + TS)

CodeSandbox

163



React Router (Remix)

CodeSandbox

136

1

2

```
# App.css M x
src > # App.css > ...
1  .container {
2    background-color: lightblue;
3    border: 1px solid black;
4    margin: 5px;
5    padding: 5px;
6    display: inline-table;
7  }
```

3

```
JS enderecos.js U x
src > infra > JS enderecos.js > ...
1  export default async function obterEndereco(cep) {
2    let retorno = {};
3    const url = `https://viacep.com.br/ws/${cep}/json/`;
4    await fetch(url)
5      .then((resposta) => resposta.json())
6      .then((endereco) => {
7        retorno = endereco;
8        console.log(endereco);
9      })
10     .catch((erro) => retorno.erro = erro);
11    return retorno;
12  }
```

A declaração **async** estabelece uma ligação de uma nova função assíncrona a um determinado nome.

Neste caso, a declaração **async** visa o agrupamento de instruções assíncronas.

A palavra-chave **await** é usada no corpo da função para conter o retorno automático de um objeto **Promise**. Em vez disso, aguarda o fim do processo assíncrono.

4

```
Tela1.jsx U x
src > pages > Tela1.jsx > ...
1  import { useState } from "react";
2  import obterEndereco from "../infra/enderecos";
3
4  export default function Tela1() {
5
6      const [endereco, setEndereco] = useState({});
7
8      async function handleChange(event) {
9          let cep = event.target.value;
10         if (cep.length === 8) {
11             let endObtido = await obterEndereco(cep);
12             console.log(endObtido.cep);
13             setEndereco(endObtido);
14         }
15     }
```

Aqui um exemplo
básico do hook
useState

5

Tela1.jsx U x

src > pages > Tela1.jsx > ...

```
4   export default function Tela1() {  
17     return (  
18       <div className="container">  
19         Digite o CEP: <input type="text" onChange={handleChange} size={8} />  
20         <p>{endereco.cep}</p>  
21         {endereco.cep &&  
22           <div>  
23             <p>CEP: {endereco.cep}</p>  
24             <p>Logradouro: {endereco.logradouro}</p>  
25             <p>Complemento: {endereco.complemento}</p>  
26             <p>Bairro: {endereco.bairro}</p>  
27             <p>Localidade: {endereco.localidade}</p>  
28             <p>UF: {endereco.uf}</p>  
29           </div>  
30         }  
31       </div>  
32     );  
33   }
```

Aqui estamos fazendo
a renderização
condicional

React App

localhost:3000

AdministrativoConteúdoFerramentasLivros

Digite o CEP: 01001000

01001-000

CEP: 01001-000

Logradouro: Praça da Sé

Complemento: lado ímpar

Bairro: Sé

Localidade: São Paulo

UF: SP

Reiniciar para atualizar

Todos os favoritos

Console

1

top

Filtro

Todos os níveis

1 problema: 1

undefined

enderecos.js:8

{cep: '22775-045', logradouro: 'Rua A

lfredo Ceschiatti', complemento: '',

▶ bairro: 'Jacarepaguá', localidade: 'R

io de Janeiro', ...}

enderecos.js:8

{cep: '22775-045', logradouro: 'Rua A

lfredo Ceschiatti', complemento: '',

▶ bairro: 'Jacarepaguá', localidade: 'R

io de Janeiro', ...}

22775-045

{cep: '22775-045', logradouro: 'Rua A

lfredo Ceschiatti', complemento: '',

▶ bairro: 'Jacarepaguá', localidade: 'R

io de Janeiro', ...}

01001-000

