

# **Análise e Desenvolvimento de Sistemas**

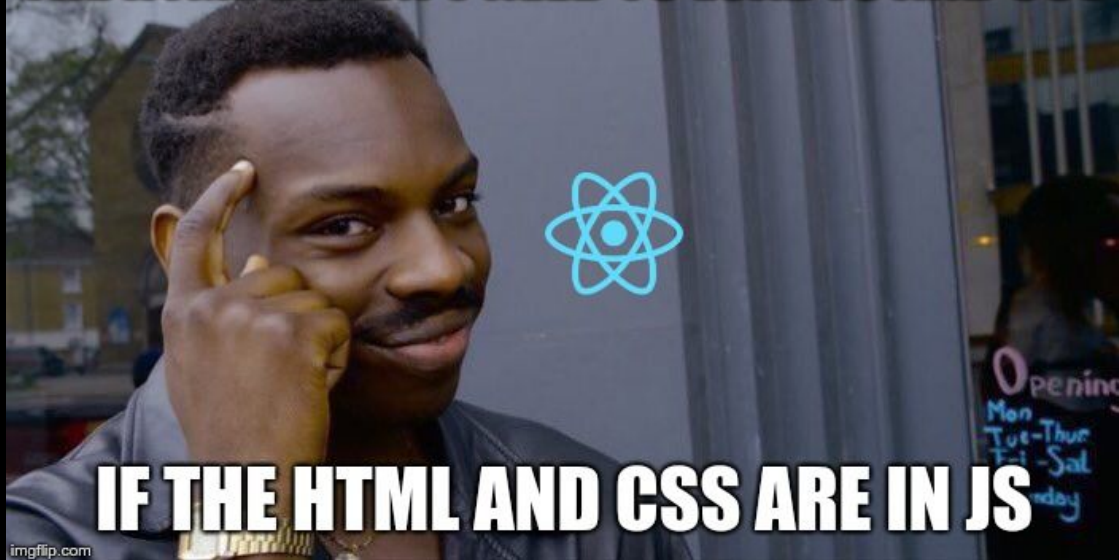
## **Frameworks Web I**

### **Aula 02 - React JS: parte 1**

---

---

**THE HTML DOESN'T NEED TO LOAD JS AND CSS**

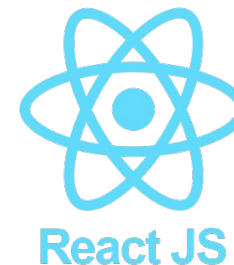


**IF THE HTML AND CSS ARE IN JS**

imgflip.com

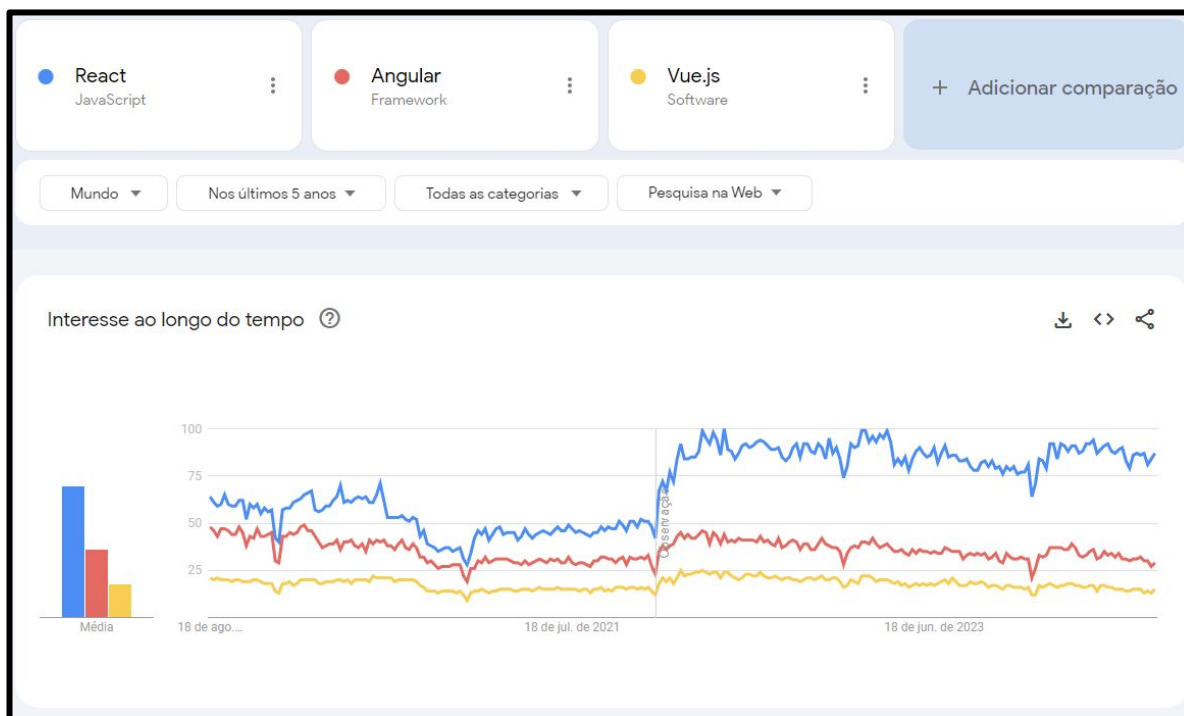
O que é?

# O que é



React é um framework JavaScript criado pelo Facebook que é usado para criar interfaces de usuário (UI) em aplicativos web.

- Ele é popular por ser fácil de usar, é flexível e escalável, e é usado por muitas empresas de tecnologia, incluindo o Facebook, Instagram e Airbnb.



# O que é

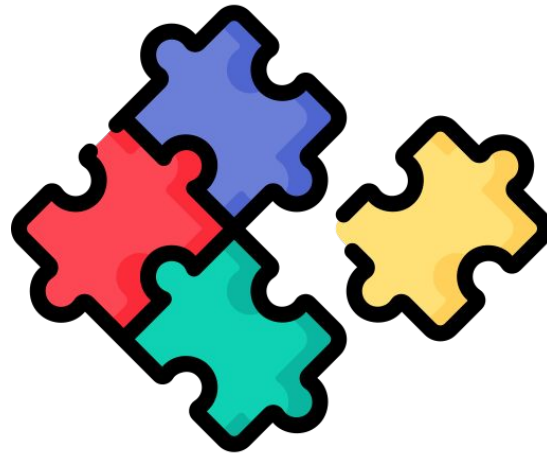
Para saber mais



# O que é

O React usa a linguagem JavaScript para criar componentes, que são pequenos pedaços de código que representam uma parte específica da interface do usuário (UI) de um aplicativo.

- Cada componente tem um estado, que é uma variável que armazena as informações que mudam dentro do componente, como os dados de um formulário ou a cor de um botão.



# O que é

Quando o usuário interage com o aplicativo, como clicar em um botão ou preencher um formulário, o estado dos componentes é atualizado e reflete as mudanças na UI.

- Isso é feito com o uso de funções de callback, que são funções que são chamadas quando uma ação é executada pelo usuário.



# O que é

DOM (Document Object Model) é utilizado pelo navegador para representar a página Web.

- Após o browser ler o HTML, cria-se um objeto que faz uma representação estruturada e define meios de como essa estrutura pode ser acessada.
- Nós podemos acessar e manipular o DOM com Javascript, é a forma mais fácil e usada.
- Com ele pode-se criar aplicações que atualizam os dados da página sem que seja necessário uma atualização.

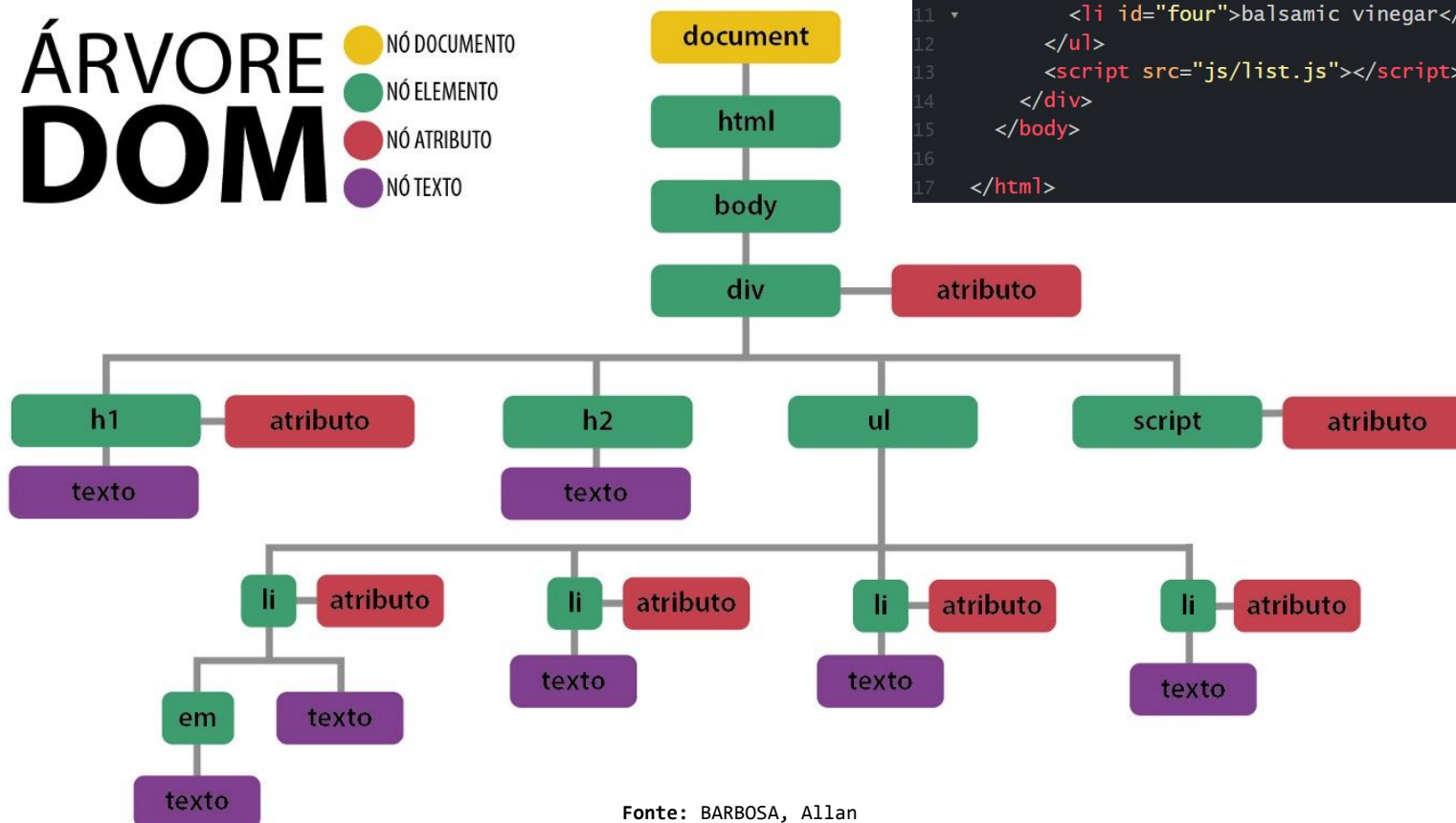


# O que é

## Exemplo

# ÁRVORE DOM

- NÓ DOCUMENTO
- NÓ ELEMENTO
- NÓ ATRIBUTO
- NÓ TEXTO



```
1 <html>
2
3 <body>
4   <div id="page">
5     <h1 id="header">List</h1>
6     <h2>Buy groceries</h2>
7     <ul>
8       <li id="one" class="hot"><em>fresh</em> figs</li>
9       <li id="two" class="hot">pine nuts</li>
10      <li id="three" class="hot">honey</li>
11      <li id="four">balsamic vinegar</li>
12    </ul>
13    <script src="js/list.js"></script>
14  </div>
15 </body>
16
17 </html>
```

# O que é

O React também usa o que é chamado de Virtual DOM (Document Object Model Virtual), que é uma representação em memória da UI do aplicativo.

- Quando o estado dos componentes muda, o Virtual DOM é atualizado e comparado com o DOM real para determinar quais mudanças precisam ser feitas na UI.
- Isso é muito mais rápido do que atualizar o DOM diretamente, o que torna o React.JS muito rápido e eficiente.

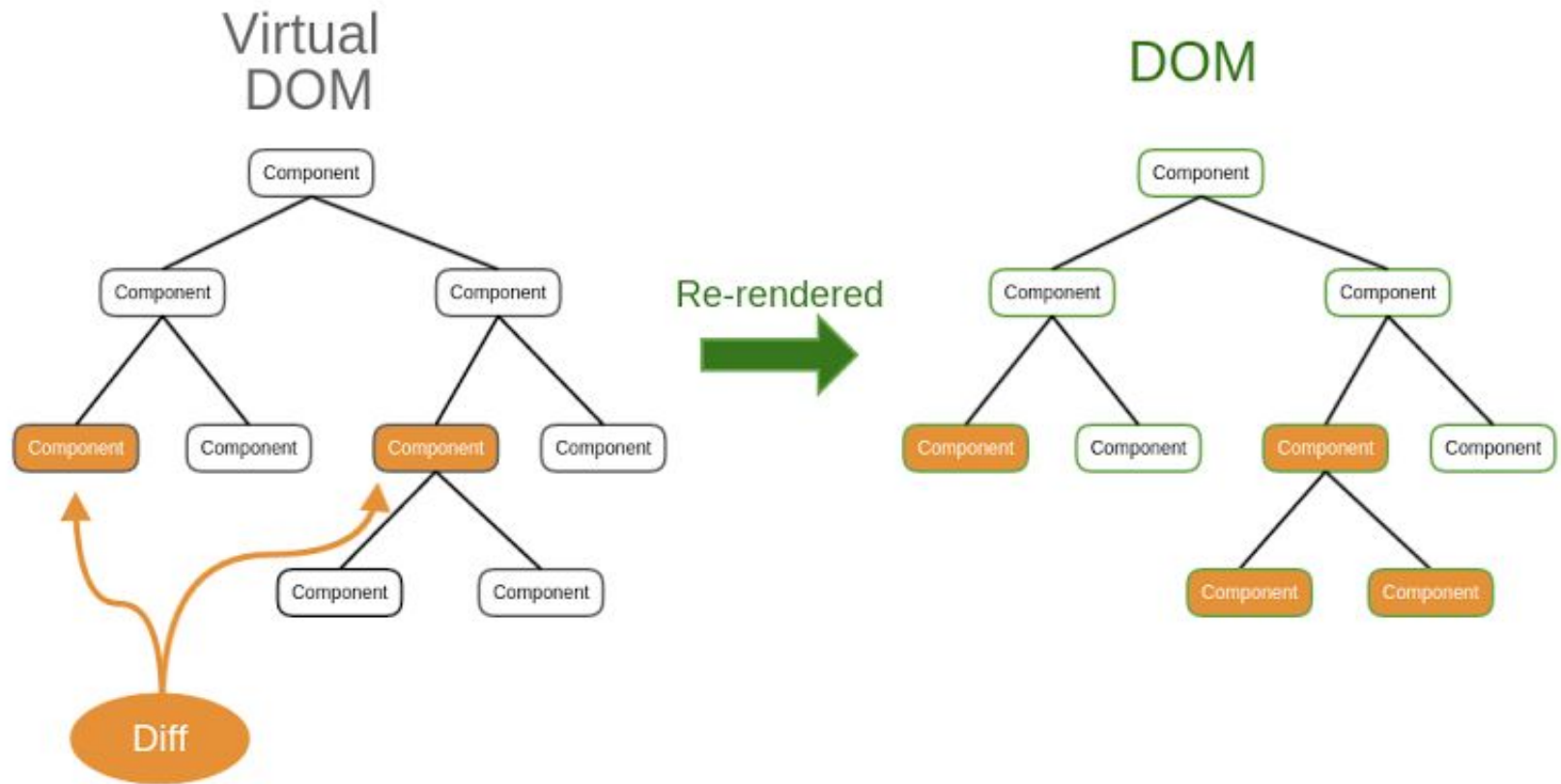
# O que é

O Virtual DOM é uma representação em memória do DOM, que é atualizada muito mais rapidamente do que o DOM real.

- Quando um componente do React é atualizado, o Virtual DOM é atualizado primeiro, e depois as alterações são sincronizadas com o DOM real. Isso torna a atualização da interface de usuário muito mais rápida e eficiente.

Em resumo, o DOM é a representação de um documento HTML ou XML no navegador, enquanto o Virtual DOM é uma representação em memória do DOM que é usada pelo React para atualizar a interface de usuário de forma mais rápida e eficiente.

# O que é



# Configurando o ambiente

# Configurando o ambiente

Ter um ambiente de desenvolvimento bem estruturado é fundamental para uma boa experiência de codificação

- Antes de iniciarmos o React, precisamos ter o Node.js e o npm (Node Package Manager) instalados na nossa máquina.
  - **Node.js** é um runtime de JavaScript que permite executar JavaScript em sua máquina, não apenas no navegador.
  - **npm** é um gerenciador de pacotes para Node.js e é usado para instalar e gerenciar dependências

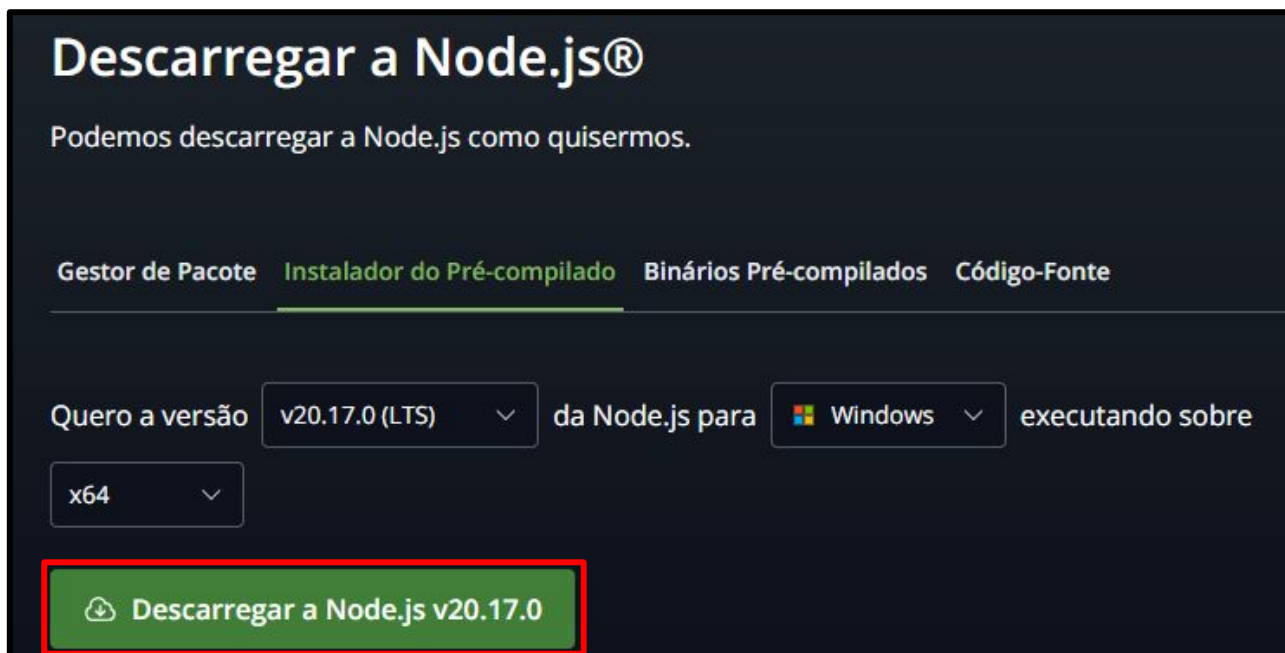


# Instalando no Windows

# Configurando o ambiente

Primeiramente, vamos instalar o Node.js na máquina, para isso acesse: [nodejs.org](https://nodejs.org)

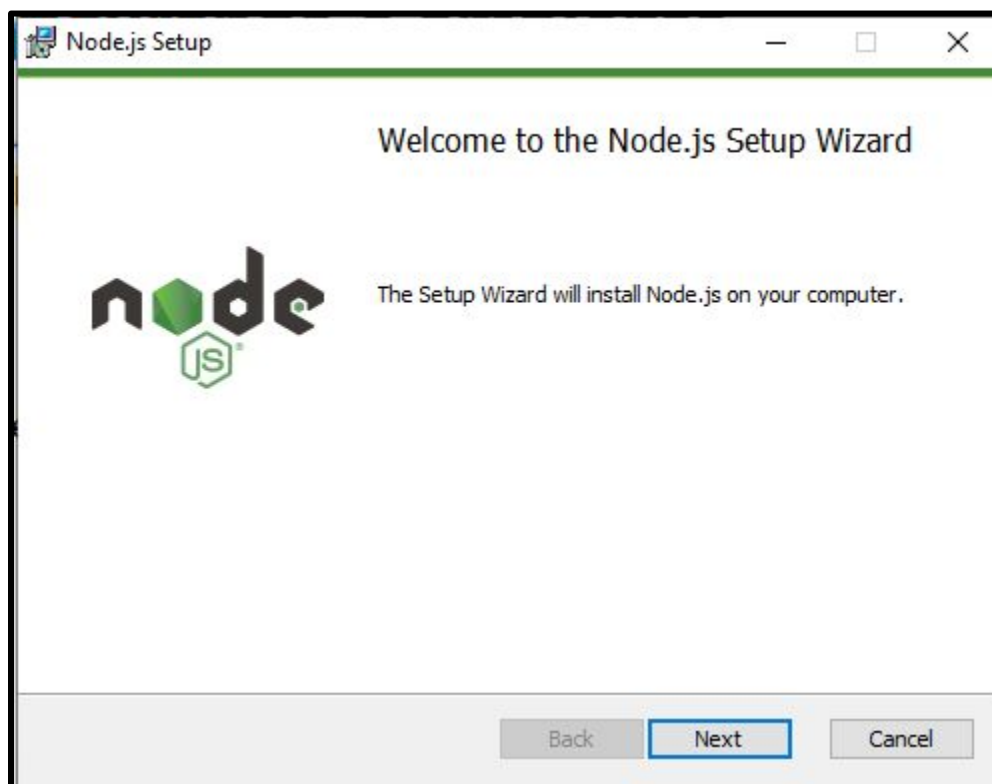
- A tela abaixo será exibida. Você deverá então selecionar a versão LTS, ou seja, a versão de suporte estendido (em inglês, Long-Term Support, ou LTS)





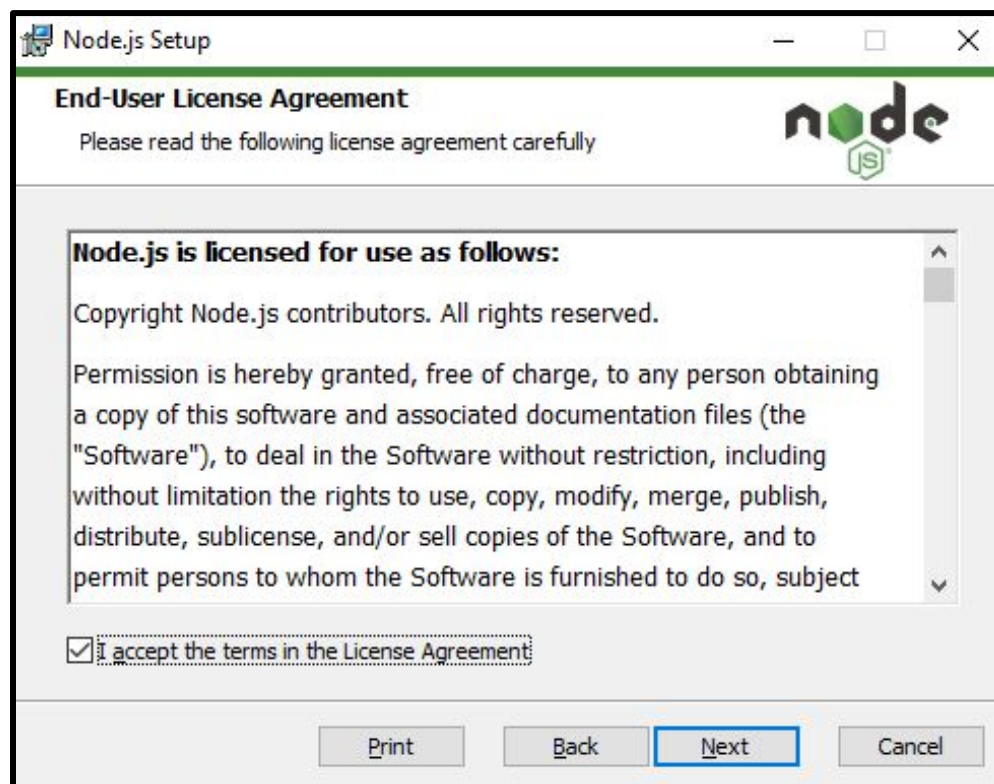
# Configurando o ambiente

Após baixar o arquivo, abra o instalador e prossiga com a instalação normalmente



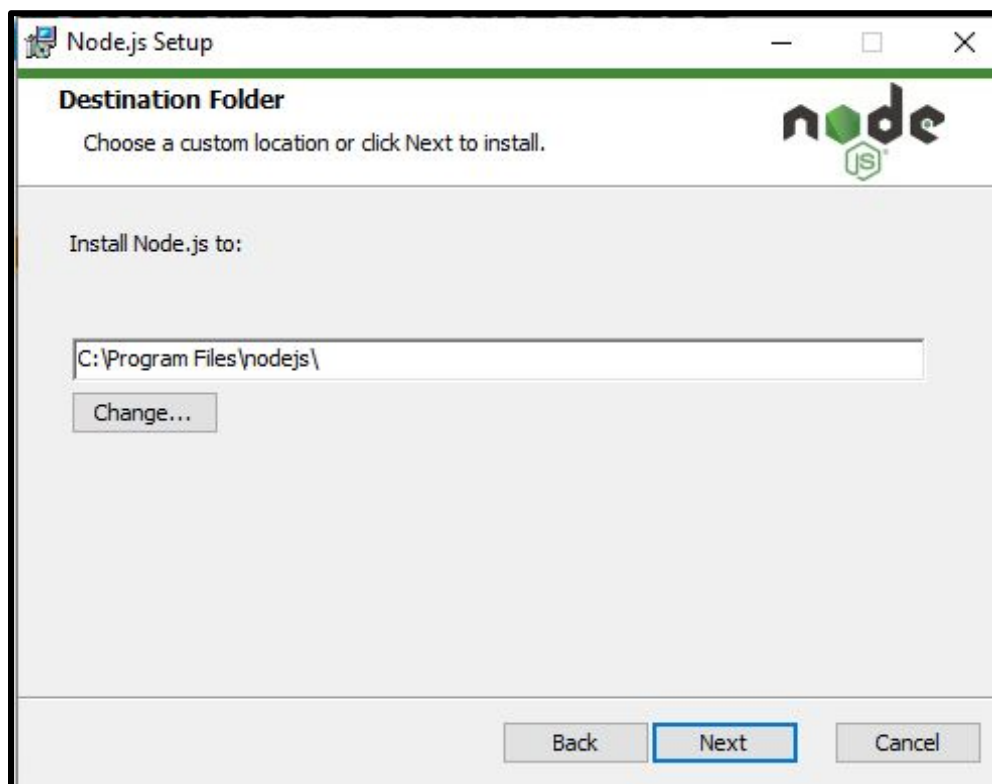
# Configurando o ambiente

Prossiga com a instalação



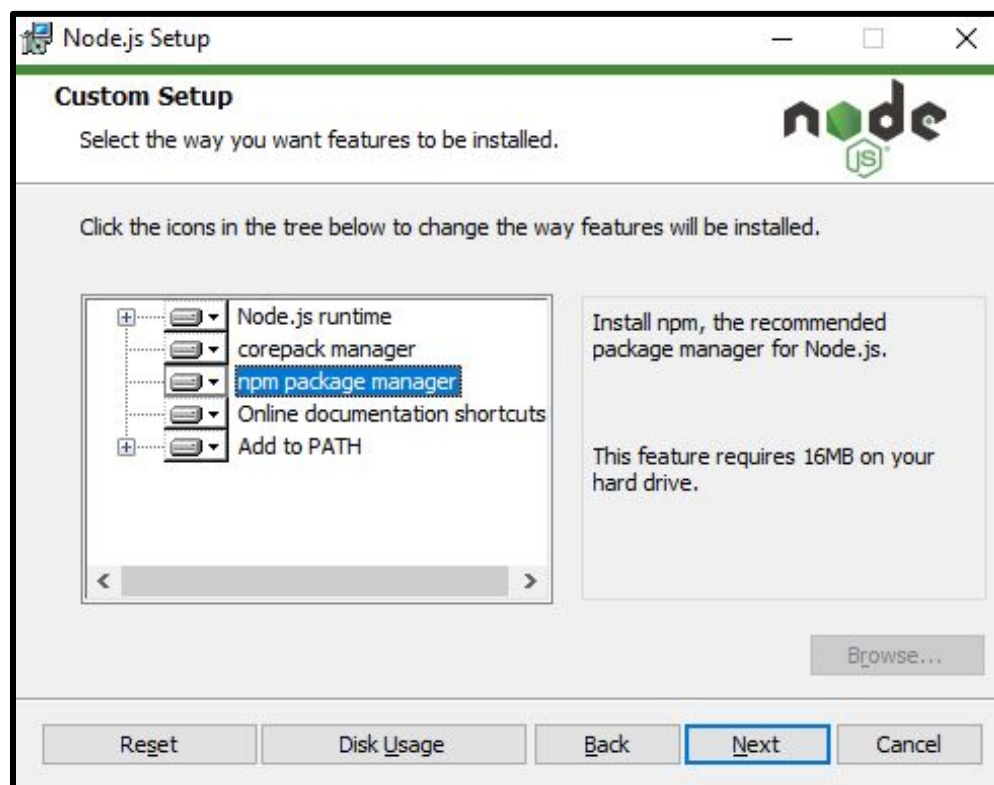
# Configurando o ambiente

Prossiga com a instalação



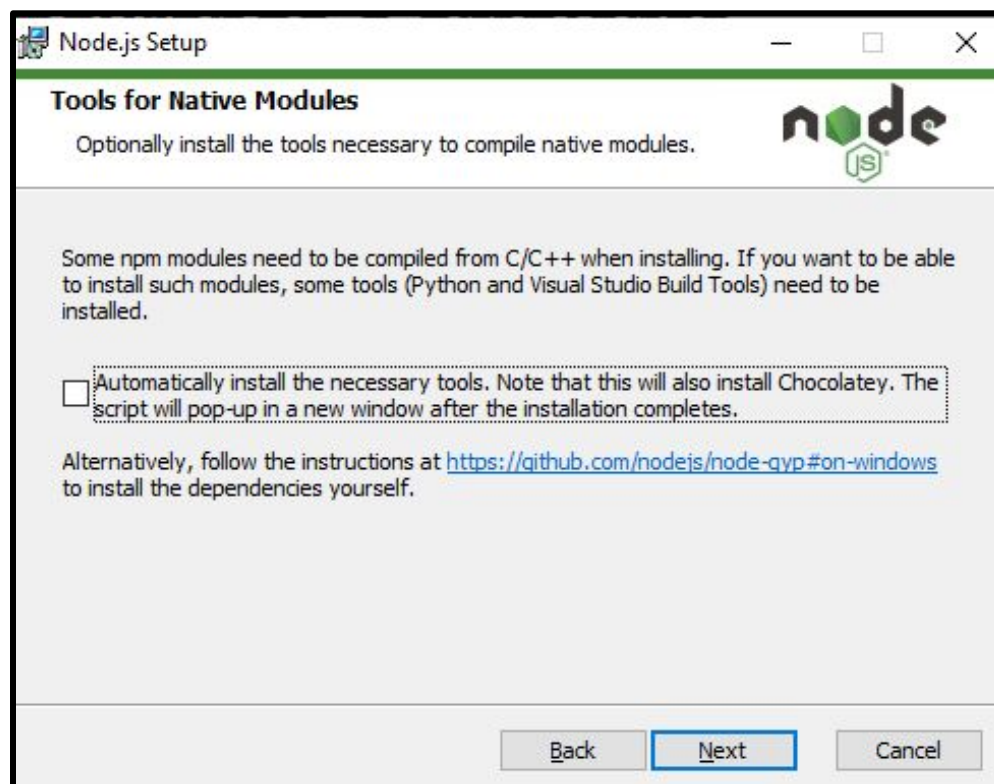
# Configurando o ambiente

Perceba que aqui, o instalador já vai instalar o npm junto com o Node :)



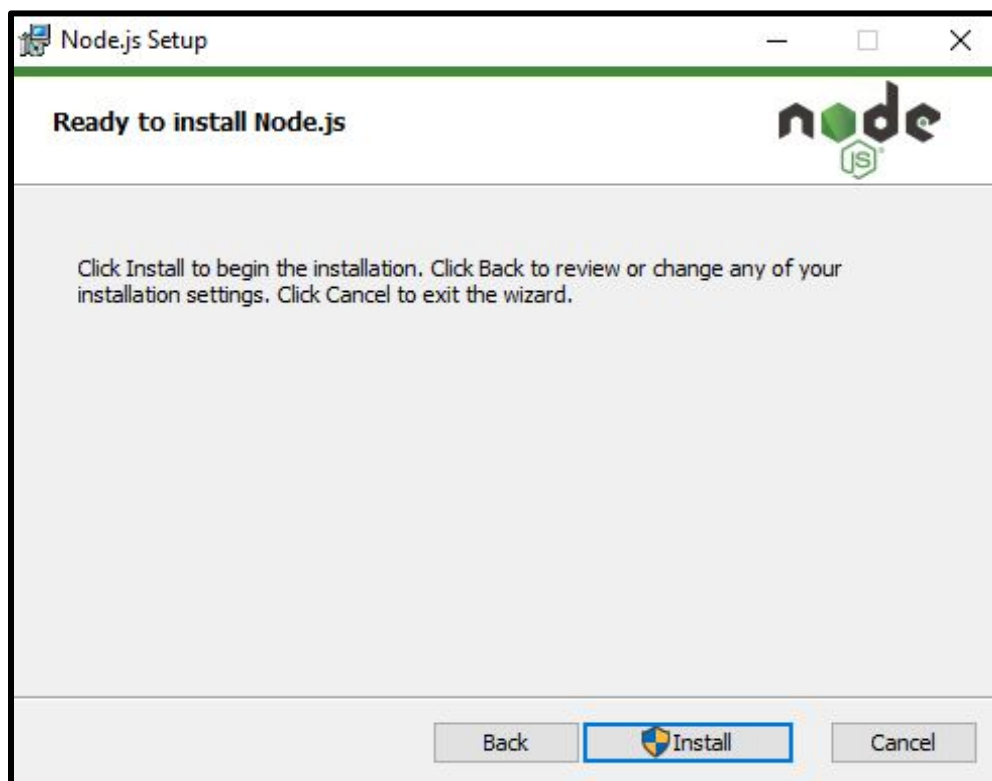
# Configurando o ambiente

## Prossiga com a instalação



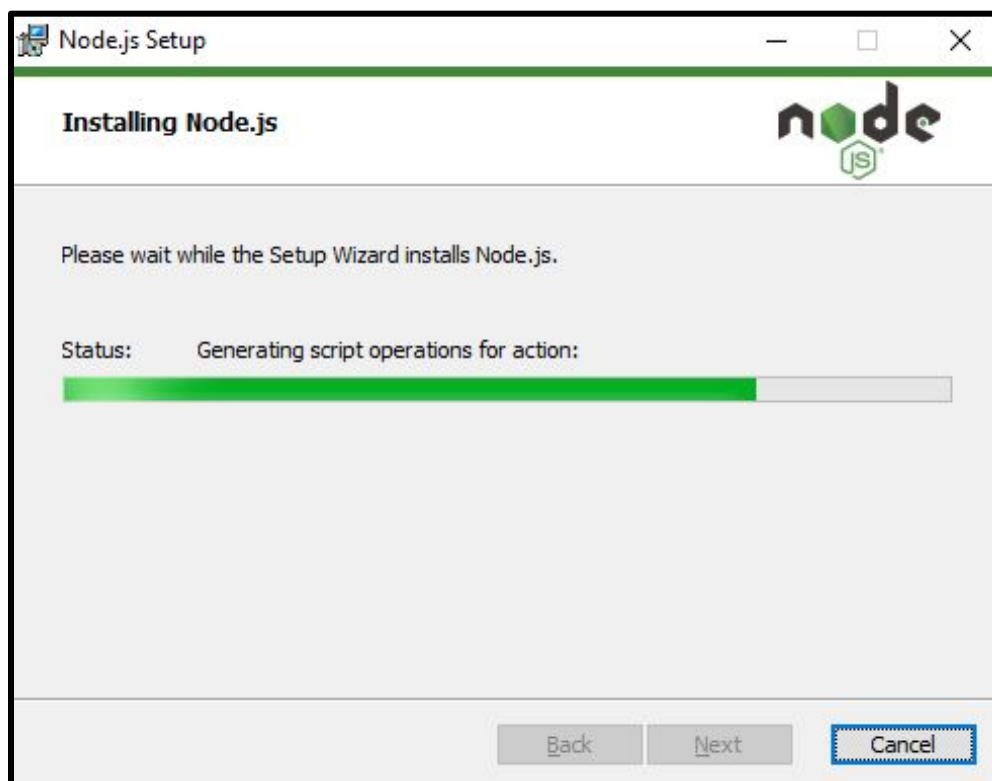
# Configurando o ambiente

Prossiga com a instalação



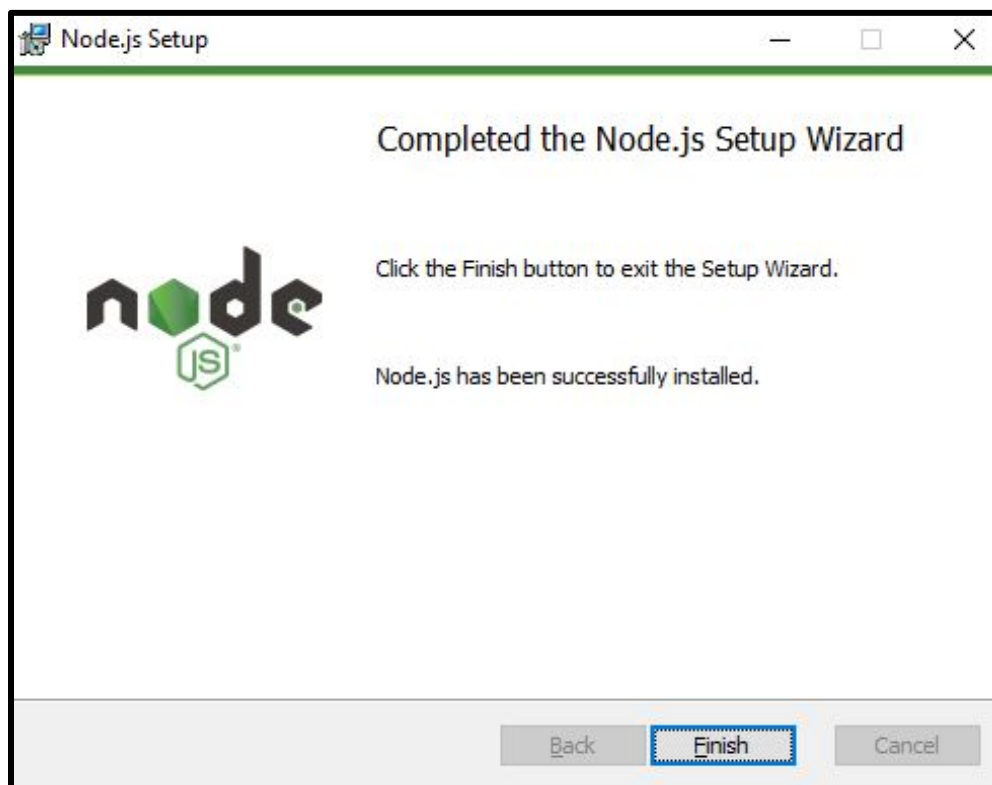
# Configurando o ambiente

Aguarde a instalação dos arquivos



# Configurando o ambiente

Pronto! O Node.js e o npm foram instalados na máquina :)





# Configurando o ambiente

Para verificar se a instalação está ok, abra um prompt (ex: Power Shell) e digite os comandos a seguir. Ao executá-los, se tudo estiver certo, as versões das ferramentas devem ser exibidas na tela

```
node -v
```

```
npm -v
```



```
Windows PowerShell
PS C:\Users\joaop> node -v
v20.17.0
PS C:\Users\joaop> npm -v
10.8.2
```

# Configurando o ambiente

Ainda no prompt, atualize a versão do npm, executando o comando abaixo

A screenshot of a Windows PowerShell terminal window. The title bar at the top says "Windows PowerShell". The command prompt shows the path "PS C:\Users\joaop>" followed by the command "npm install -g npm" in a light blue font. The terminal background is black.

```
Windows PowerShell
PS C:\Users\joaop> npm install -g npm
```

# Instalando no Linux

# Configurando o ambiente

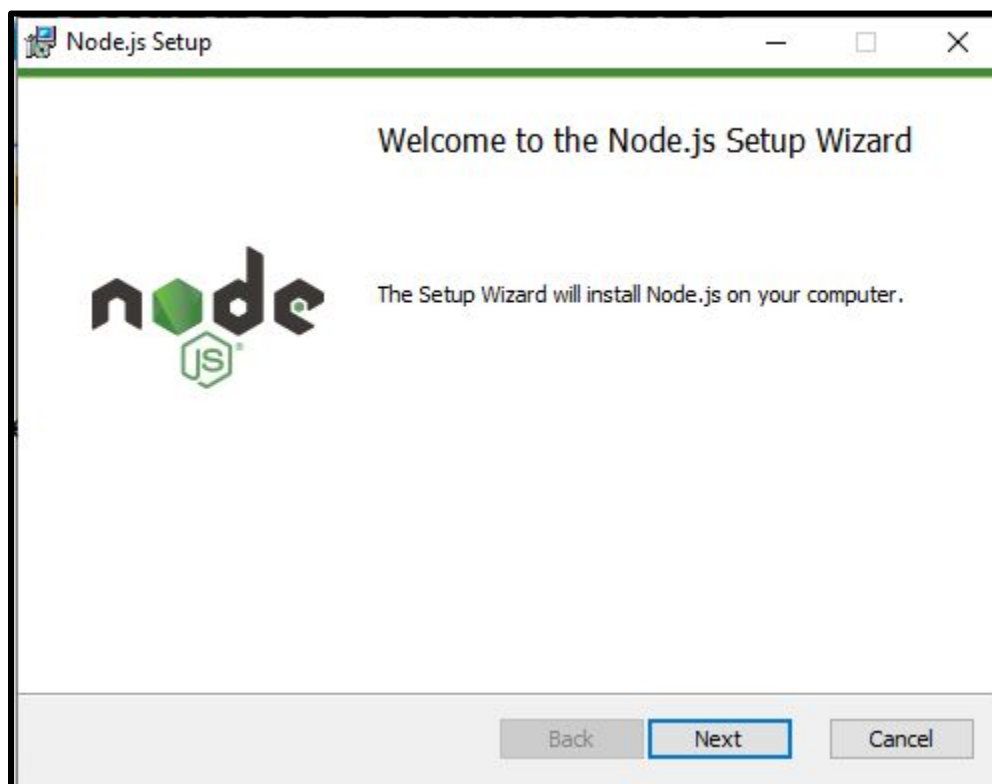
Primeiramente, vamos instalar o Node.js na máquina, para isso abra um terminal e execute os dois comandos abaixo

```
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -
```

```
sudo apt-get install -y nodejs
```

# Configurando o ambiente

Após baixar o arquivo, abra o instalador e prossiga com a instalação normalmente

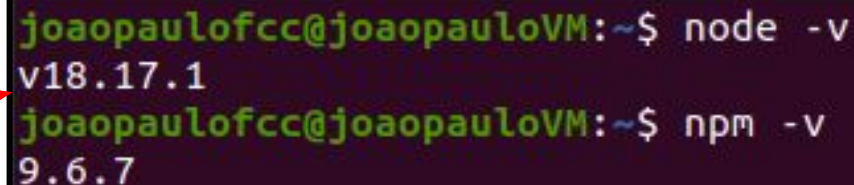


# Configurando o ambiente

Para verificar se a instalação está ok, abra um terminal e digite os comandos a seguir. Ao executá-los, se tudo estiver certo, as versões das ferramentas devem ser exibidas na tela

```
node -v
```

```
npm -v
```



```
joaopaulofcc@joaopauloVM:~$ node -v  
v18.17.1  
joaopaulofcc@joaopauloVM:~$ npm -v  
9.6.7
```

Se suas versões forem mais  
novas não tem problema :)

# Configurando o ambiente

Ainda no terminal, atualize a versão do npm, executando o comando abaixo

```
joaopaulofcc@joaopauloVM:~$ sudo npm install -g npm  
removed 16 packages, and changed 43 packages in 4s  
28 packages are looking for funding  
run `npm fund` for details
```

# Visual Studio Code



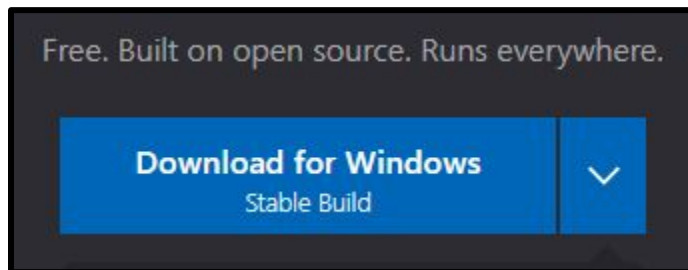
# Configurando o ambiente

Em seguida, precisaremos de instalar algum editor de código, uma opção simples e gratuita de utilizar é o Visual Studio Code (VS Code).

Para realizar a instalação do VS Code, acesse:

<https://code.visualstudio.com>

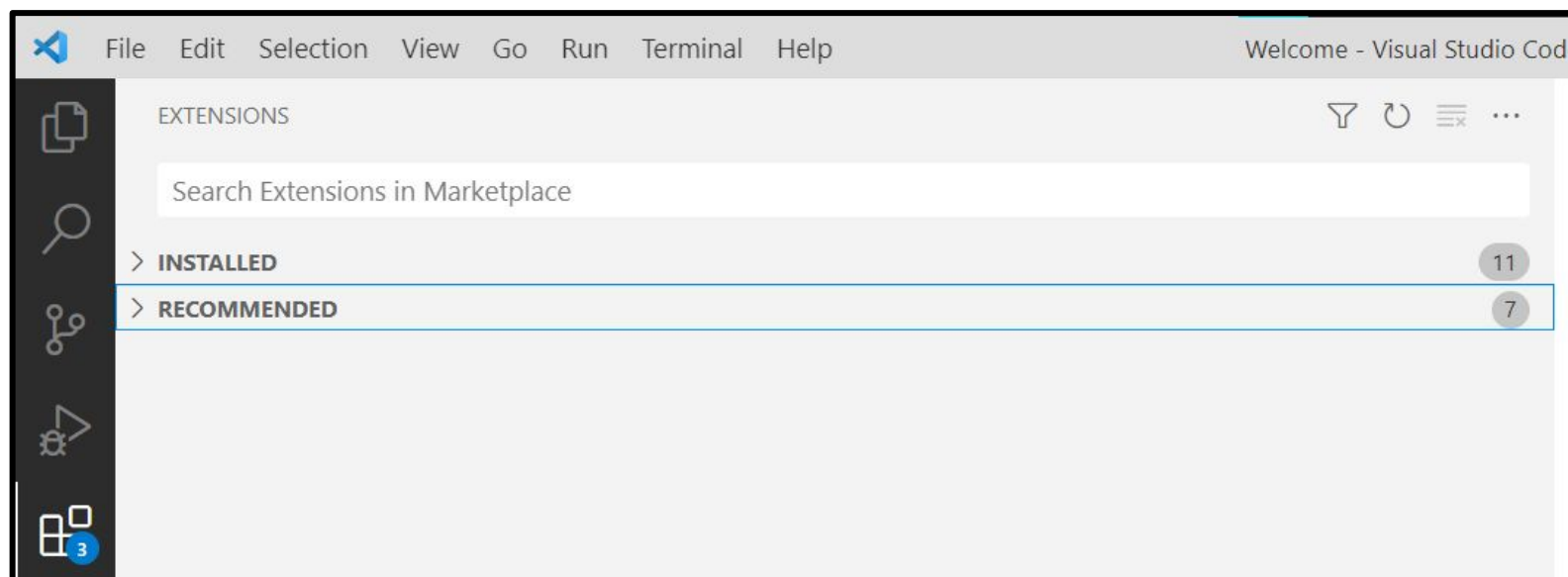
- Baixe o instalador e prossiga normalmente com a instalação



# Extensões VS Code


# Configurando o ambiente

Depois de configurar o VS Code, considere instalar algumas extensões que facilitarão o desenvolvimento do React








# Configurando o ambiente



A extensão “ES7 React/Redux/GraphQL/React-Native snippets” oferece uma coleção de snippets que podem acelerar sua codificação React



## ES7 React/Redux/GraphQL/React-Native snippets v1.9.3

rodrigoallades |  835,220 |      (4)

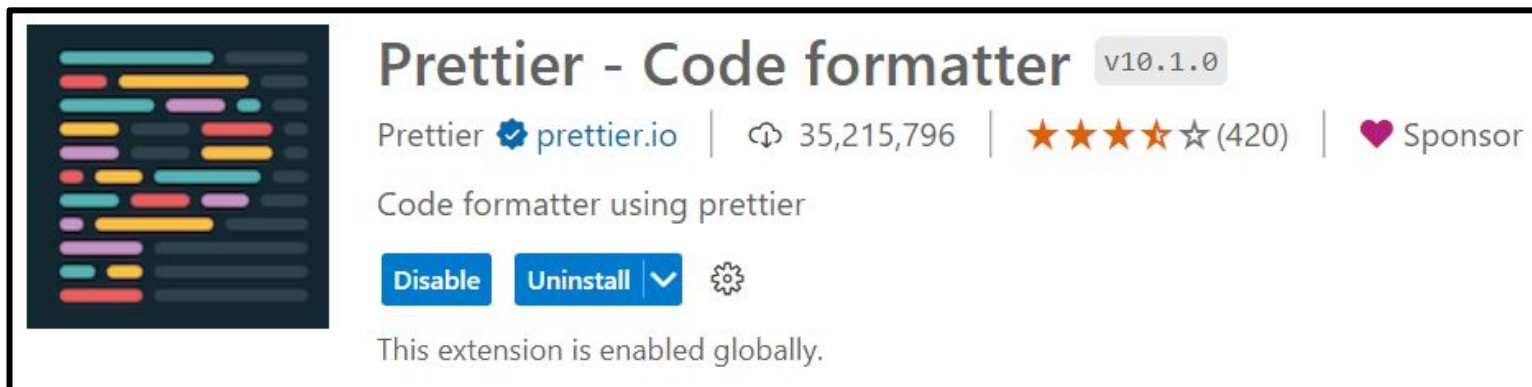
Simple extensions for React, Redux and Graphql in JS/TS with ES7 syntax (forked from dsznajder)

Disable Uninstall  

This extension is enabled globally.

# Configurando o ambiente

A extensão “Prettier - Code formatter” é outra ótima extensão para formatar automaticamente seu código para garantir que ele siga um estilo consistente

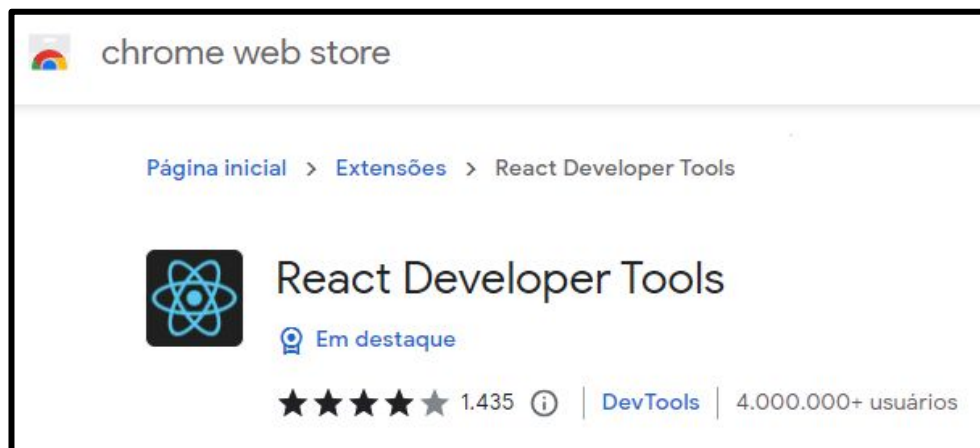


# Extras

# Configurando o ambiente

## React Developer Tools

- É uma extensão de navegador essencial disponível para Chrome e Firefox.
- Ele permite que você inspecione hierarquias de componentes no DOM virtual, observe props e estado de componentes e veja atualizações de componentes em tempo real.



# Criando um projeto



# Criando um projeto

Também é possível criar um projeto usando o comando abaixo:

```
"npx create-react-app my-react-app"
```

Após configurar o ambiente de desenvolvimento nos passos anteriores, você já está pronto para criar um projeto React

- Para isso, abra um prompt e digite o comando mostrado abaixo, onde meu-app" é o nome que eu quero dar para minha aplicação

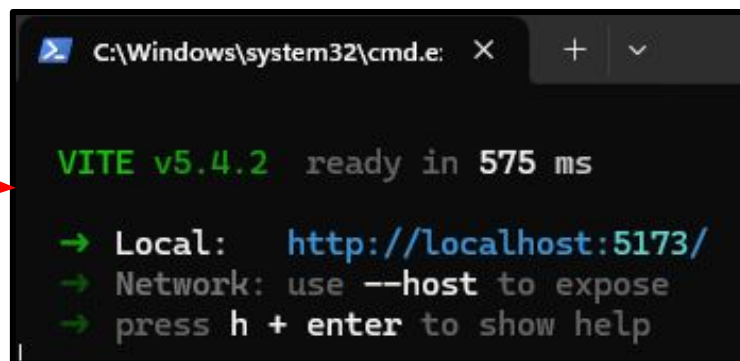
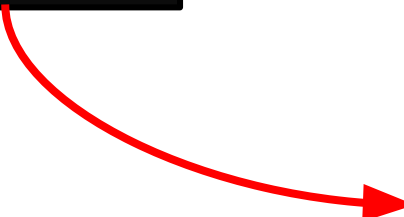
```
npm create vite@latest my-react-app -- --template react
```

Após a execução do comando, aguarde alguns minutos até que todas as dependências sejam baixadas :)

# Criando um projeto

Após a execução, acesse a pasta onde foi criado o projeto (a pasta tem o mesmo nome escolhido no comando) e inicie o servidor de desenvolvimento :)

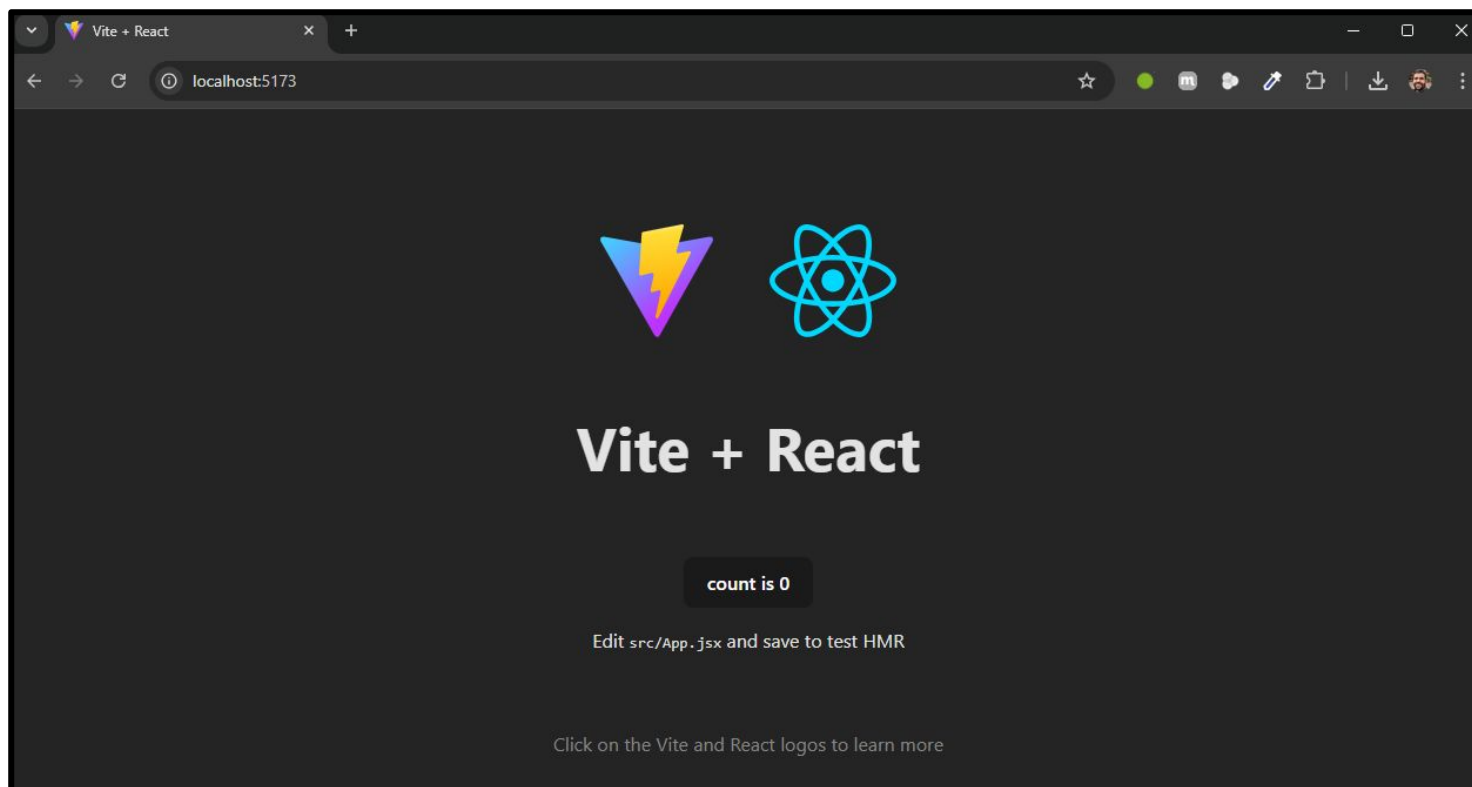
```
cd meu-app  
npm install  
npm run dev
```



```
C:\Windows\system32\cmd.e: X + v  
  
VITE v5.4.2 ready in 575 ms  
→ Local:   http://localhost:5173/  
→ Network: use --host to expose  
→ press h + enter to show help
```

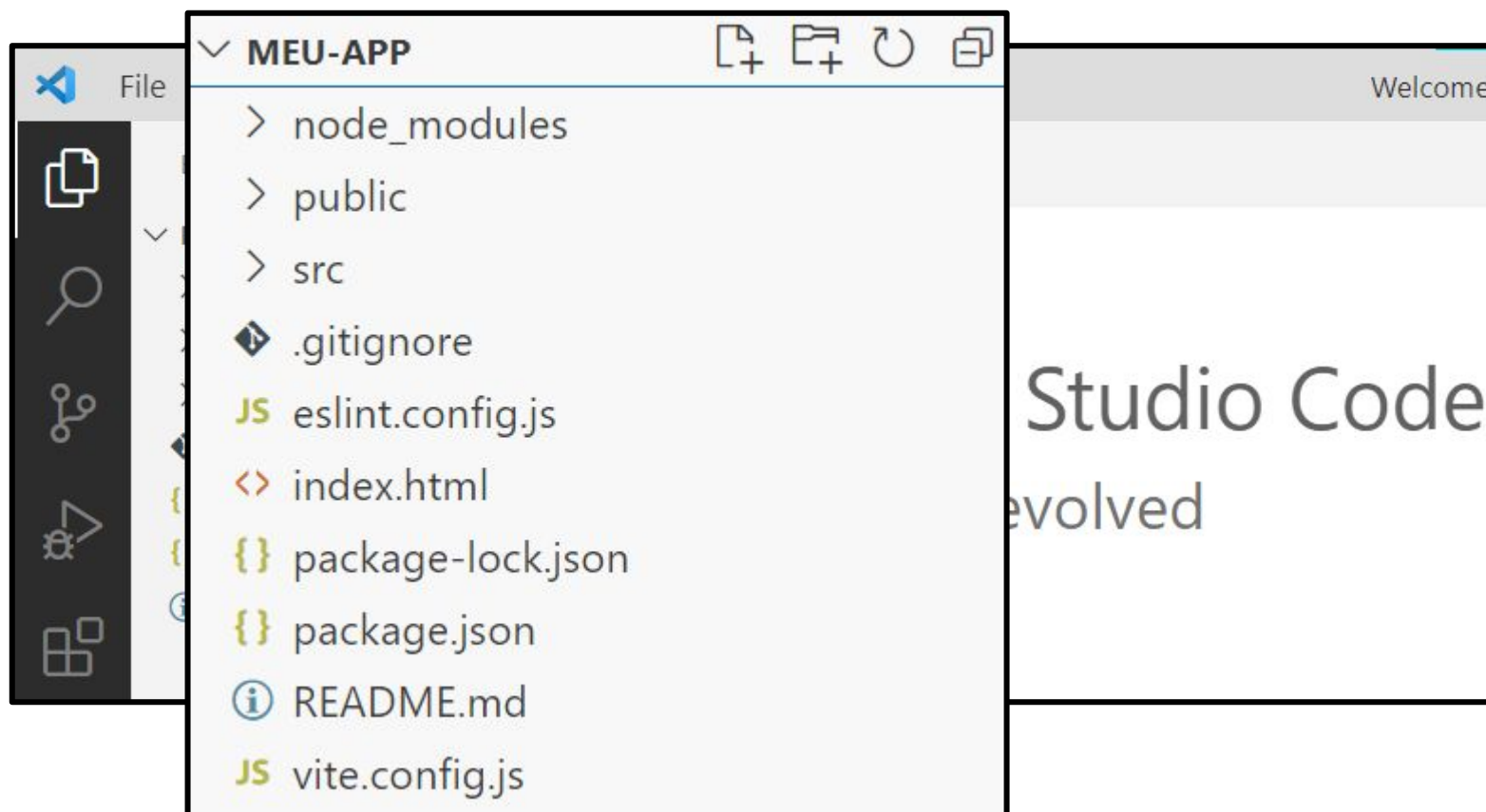
# Criando um projeto

Automaticamente será aberta uma janela no navegador com a aplicação default do Vite + React sendo executada :)



# Criando um projeto

Para editar o projeto, basta abrir esta pasta no VS Code :)



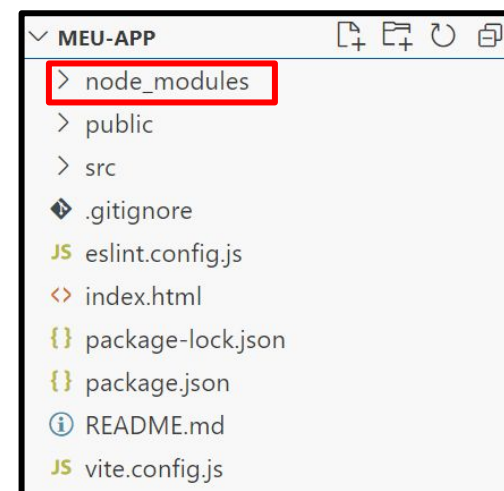
# Estrutura de arquivos

# Estrutura de arquivos

A pasta `node_modules` contém todos os módulos e as funcionalidades necessárias para projetar com uso da biblioteca React

- É também nesta pasta que serão armazenados os módulos e bibliotecas auxiliares que você instala especificamente para uma determinada aplicação.
- Vários módulos e bibliotecas são nativos do React e já residem nesta pasta quando se cria a estrutura padrão.

Não precisamos nos preocupar com esta pasta agora



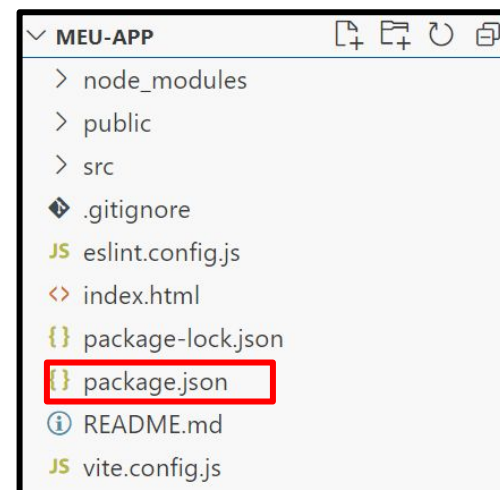
# Estrutura de arquivos

Pense neste arquivo como uma “lista de ingredientes” necessários para fazer funcionar a aplicação :)

O arquivo “package.json” contém um JSON que descreve o nome e versão de todas as dependências da aplicação

- Ao personalizar sua aplicação instalando funcionalidades, posteriormente elas e suas versões serão automaticamente gravadas neste arquivo.

```
{} package.json X
{} package.json > ...
1  {
2    "name": "meu-app",
3    "private": true,
4    "version": "0.0.0",
5    "type": "module",
6    "scripts": {
7      "dev": "vite",
8      "build": "vite build",
9      "lint": "eslint .",
10     "preview": "vite preview"
11   },
12   "dependencies": {
13     "react": "^18.3.1",
14     "react-dom": "^18.3.1"
15   },
16   "devDependencies": {
17     "@eslint/js": "^9.9.0",
```

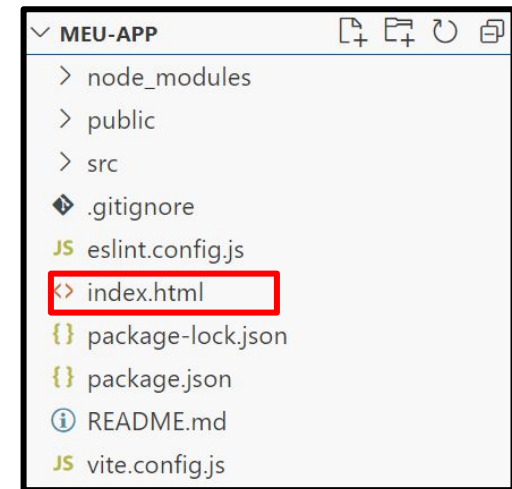


# Estrutura de arquivos

## Arquivo “index.html” com marcação HTML5

- É nesse arquivo que encontramos o container da aplicação. Além deste arquivo, outros também estão presentes nesta pasta.

```
<? index.html X
<? index.html > ...
1  <!doctype html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Vite + React</title>
8    </head>
9    <body>
10     <div id="root"></div>
11     <script type="module" src="/src/main.jsx"></script>
12   </body>
13 </html>
14
```

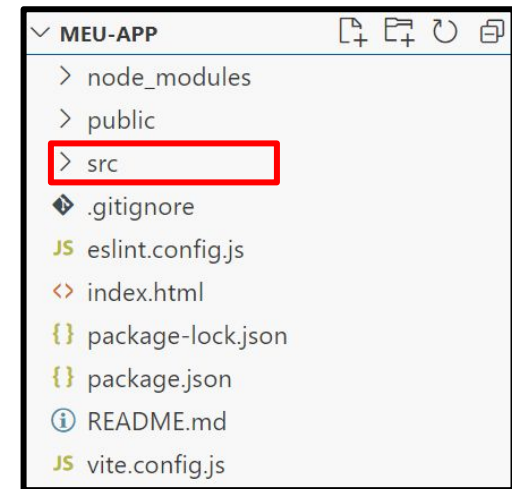




# Estrutura de arquivos

A pasta “src” contém os arquivos nos quais trabalharemos na maior parte do tempo



- O arquivo “App.js” é o componente de mais alto nível que serve de container geral para toda a aplicação. É este arquivo que vai injetar toda a marcação HTML dentro do container do arquivo index.html
- Já os arquivos “App.css” e “index.css” são os arquivos para estilização dos elementos da página




# Hot reload

# Hot reload

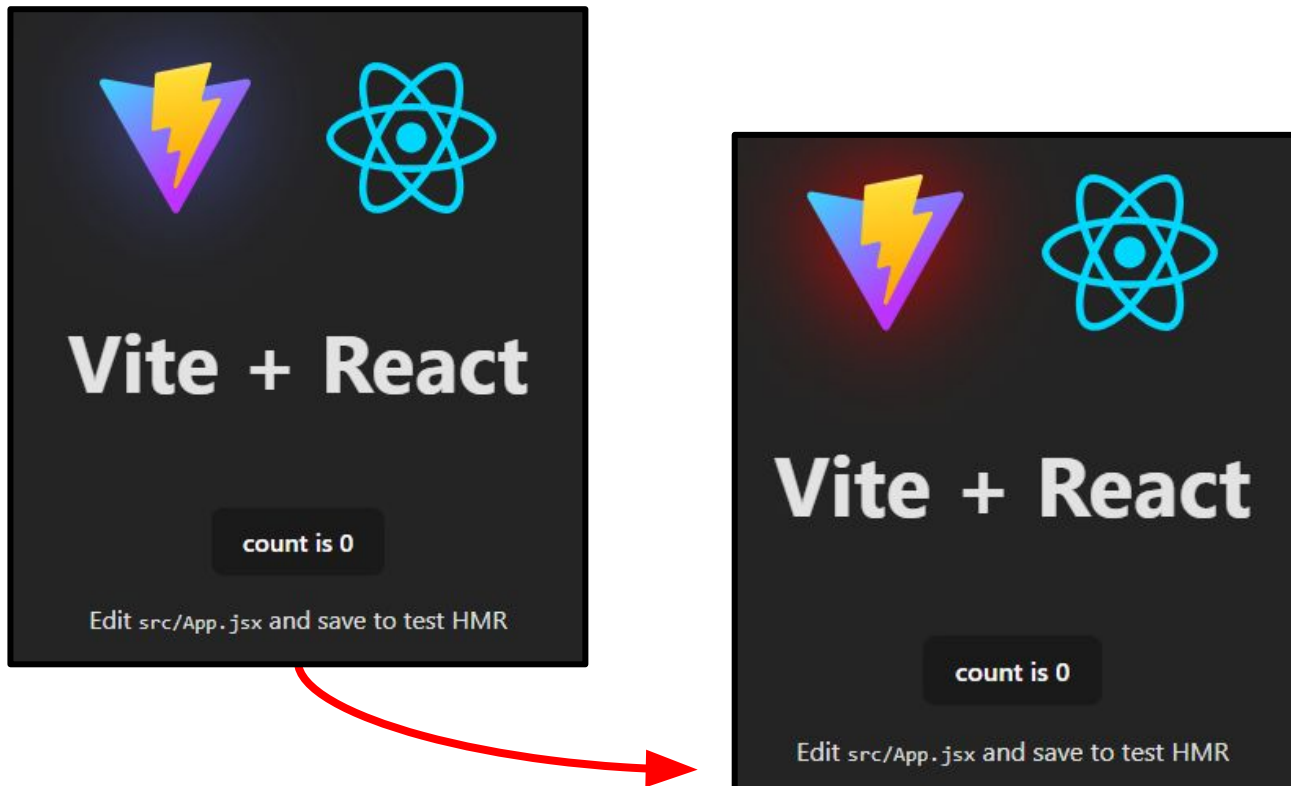
Com o servidor de desenvolvimento rodando no endereço localhost, acesse o arquivo “src/App.css” e faça a seguinte alteração no trecho abaixo:

```
14  .logo:hover {  
15  |    filter: drop-shadow(0 0 2em  #646cffaa);  
16  |  
17  .logo.react:hover {  
18  |    filter: drop-shadow(0 0 2em  #61dafbaa);  
19  |  }
```

```
14  .logo:hover {  
15  |    filter: drop-shadow(0 0 2em  red);  
16  |  
17  .logo.react:hover {  
18  |    filter: drop-shadow(0 0 2em  #61dafbaa);  
19  |  }
```

# Hot reload

Assim que você salvar qualquer alteração feita em um arquivo do projeto, automaticamente o servidor de desenvolvimento carrega e atualiza a página mostrando imediatamente estas alterações sem precisar reiniciar o servidor :)



# Conceitos básicos

# JSX

# Conceitos básicos

JSX, (JavaScript XML) foi apresentado pelo Facebook junto com o React

- JSX é uma extensão de sintaxe para JavaScript que permite escrever código semelhante a HTML em seus arquivos JavaScript.
- Com o JSX o React **mais intuitivo e seu código mais legível.**

Exemplo de código JSX / JavaScript equivalente:



```
const texto = <h1>Olá Mundo!</h1>;
```



```
const texto = React.createElement("<h1", null, "Ola Mundo!");
```

# Conceitos básicos

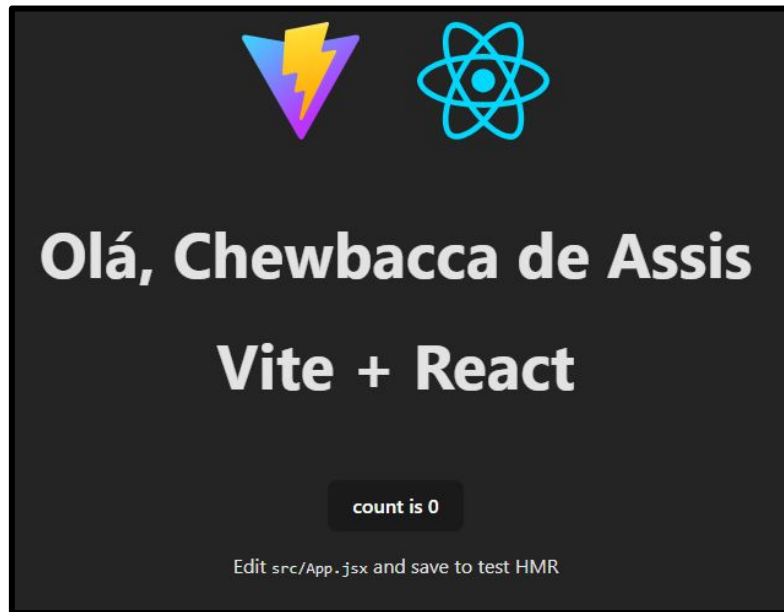
Um dos recursos mais interessantes do JSX é sua capacidade de interpolar expressões JavaScript dentro da sintaxe semelhante a HTML usando chaves {}

- É possível incorporar qualquer expressão JavaScript dentro do JSX envolvendo-a entre chaves.



# Conceitos básicos

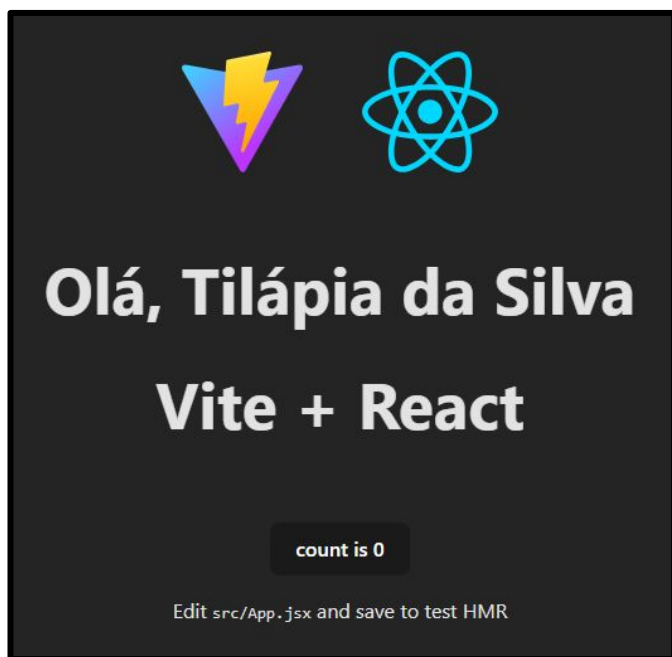
## Exemplo



```
App.jsx x
src > App.jsx > App
1  import { useState } from 'react'
2  import reactLogo from './assets/react.svg'
3  import viteLogo from '/vite.svg'
4  import './App.css'
5
6  const name = "Chewbacca de Assis";
7  const element = <h1>Olá, {name}</h1>
8
9  function App() {
10   const [count, setCount] = useState(0)
11
12   return (
13     <div>
14       <a href="https://vitejs.dev" target="_blank">
15         <img src={viteLogo} className="logo" alt="Vite logo" />
16       </a>
17       <a href="https://react.dev" target="_blank">
18         <img src={reactLogo} className="logo" alt="React logo" />
19       </a>
20     </div>
21     <p>
22       {element}
23     </p>
24     <h1>Vite + React</h1>
25   )
26 }
```

# Conceitos básicos

Além da interpolação, o JSX também permite que você crie props (abreviação de properties), que são entradas para um componente React



```
App.jsx 1 •
src > App.jsx > App
1  import { useState } from 'react'
2  import reactLogo from './assets/react.svg'
3  import viteLogo from '/vite.svg'
4  import './App.css'
5
6  function Welcome(props) {
7    return <h1>Olá, {props.name}</h1>
8  }
9
10 function App() {
11   const [count, setCount] = useState(0)
12
13   return (
14     <div>
15       <a href="https://vitejs.dev" target="_blank">
16         <img src={viteLogo} className="logo" alt="Vite logo" />
17       </a>
18       <a href="https://react.dev" target="_blank">
19         <img src={reactLogo} className="logo" alt="React logo" />
20       </a>
21     </div>
22     <p>
23       <Welcome name = "Tilápia da Silva" />
24     </p>
25   )
26 }
```

# Conceitos básicos

O JSX também suporta filhos, permitindo que você crie UIs complexas a partir de componentes menores e reutilizáveis.

```
function Welcome(props) {  
  return <h1>Olá, {props.name}</h1>  
}  
  
function App() {  
  return (  
    <div>  
      <Welcome name="José da Silva"/>  
      <Welcome name="Tião Tucunaré"/>  
    </div>  
  );  
}
```



# Conceitos básicos



## Resumindo

- Não é obrigatório utilizar JSX nos projetos em React, você pode escrever em JavaScript.
- Porém, sua sintaxe semelhante a HTML fornece uma maneira mais direta de descrever como sua interface do usuário deve ser.
- Ao usar JSX, você pode escrever um código mais legível e de manutenção facilitada, tornando seu processo de desenvolvimento web mais eficiente.

# Renderizando elementos

# Conceitos básicos

## Renderização de elementos em React é um conceito fundamental

- Elementos são os menores blocos de construção de um aplicativo React.
- Um elemento no React é um objeto que descreve o que você deseja ver na tela. Ao contrário dos elementos DOM do navegador, os elementos React são leves e fáceis de criar.

## Exemplo de um elemento React



```
const element = <h1>Olá mundo!</h1>;
```

# Conceitos básicos

Os elementos React são renderizados para o DOM por meio da função `createRoot()` e seu método `render()`

- **Função `createRoot`:** recebe um argumento, um elemento HTML.
- **Método `render`:** é chamado para definir o componente React que deve ser renderizado.

# Conceitos básicos

## Como funciona o método render?

- No projeto React, existe um arquivo `index.html`. Neste arquivo existe um única `<div>`. É nesta div que o aplicativo React será renderizado.

```
index.html > ...
1  <!doctype html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Vite + React</title>
8    </head>
9    <body>
10     <div id="root"></div>
11     <script type="module" src="/src/main.jsx"></script>
12   </body>
13 </html>
```



# Conceitos básicos

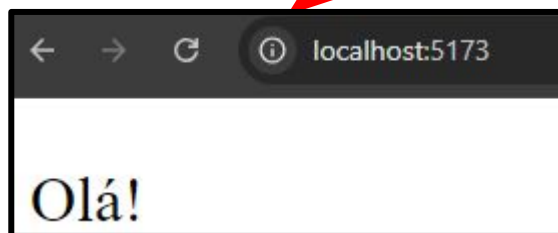
OBS: o id do elemento não precisa ser chamado de "root", mas esta é a convenção padrão.

## Exemplo 1

- O exemplo abaixo, adicionado no arquivo “main.jsx” (ou “index.jsx”), adicionar um parágrafo dentro do elemento com id “root”.

```
main.jsx X
src > main.jsx > ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3
4 const container = document.getElementById('root');
5 const root = ReactDOM.createRoot(container);
6 root.render(<p>Olá!</p>);
```

```
<div id="root">
  <p>Olá!</p> == $0
</div>
```



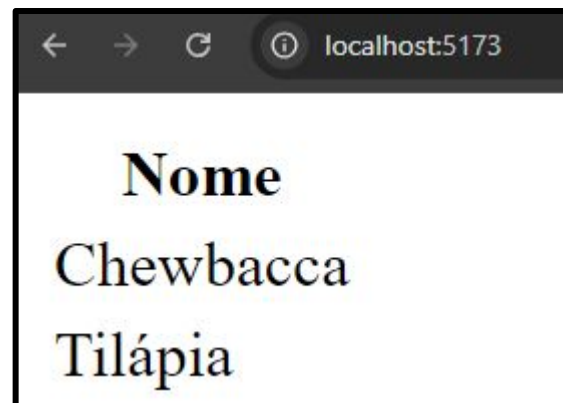
# Conceitos básicos

OBS: o id do elemento não precisa ser chamado de "root", mas esta é a convenção padrão.

## Exemplo 2

- Crie uma variável que contenha código HTML e exiba-a no nó "root":

```
main.jsx ×
src > main.jsx > ...
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  const minhaTabela = (
5    <table>
6      <tr>
7        <th>Nome</th>
8      </tr>
9      <tr>
10       <td>Chewbacca</td>
11     </tr>
12     <tr>
13       <td>Tilápia</td>
14     </tr>
15   </table>
16 );
17
18 const container = document.getElementById('root');
19 const root = ReactDOM.createRoot(container);
20 root.render(minhaTabela);
```



# Conceitos básicos

## Exemplo 3

- Criando um relógio que é atualizado a cada segundo

```
main.jsx ×
src > main.jsx > ...
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  function relógio() {
5    const element = (
6      <div>
7        <h1>Olá mundo</h1>
8        <h2>Agora são {new Date().toLocaleTimeString()}</h2>
9      </div>
10   );
11
12   const container = document.getElementById('root');
13   const root = ReactDOM.createRoot(container);
14   root.render(element);
15 }
16
17 setInterval(relógio, 1000);
```

[Acesse o código comentado](#)

**Olá mundo!**

**Agora são 09:24:40**

# Conceitos básicos

## Exemplo 3 - Observações

- Neste exemplo, a função `relogio()` é executada a cada segundo, criando um novo elemento com a hora atual e renderizando-o no DOM.
- Isso pode parecer ineficiente, mas o React foi projetado para lidar com atualizações frequentes com alto desempenho.

Quando um novo elemento é passado para `ReactDOM.render()`, o React compara esse novo elemento com o renderizado anteriormente. Em seguida, ele calcula a maneira mais eficiente de atualizar o DOM para corresponder à árvore mais recente, atualizando apenas as partes alteradas do DOM real.

# Componentes e props

# Conceitos básicos

Componentes e props são elementos base do React que permitem a criação de interfaces de usuário complexas a partir de peças menores e reutilizáveis

- Componentes são pedaços de código independentes e reutilizáveis.
- Eles servem ao mesmo propósito que as funções JavaScript, mas trabalham isoladamente e retornam HTML.

Componentes podem ser de dois tipos: de classe e de função. Por enquanto vamos nos concentrar mais nos componentes de função.

# Conceitos básicos

Ao criar um componente React, o nome do componente DEVE começar com letra maiúscula

## Componente de Classe

- Um componente de classe deve incluir a instrução `extends React.Component`.
- Esta instrução cria uma herança para `React.Component` e dá ao seu componente acesso às funções do `React.Component`.
- O componente também requer um método `render()`, este método retorna HTML.



```
class Car extends React.Component {  
  render() {  
    return <h2>Olá! Eu sou um carro!</h2>;  
  }  
}
```

# Conceitos básicos

Ao criar um componente React,  
o nome do componente DEVE  
começar com letra maiúscula

## Componente de Função

- Um componente de função também retorna HTML e se comporta da mesma maneira que um componente de classe, mas os componentes de função podem ser escritos usando muito menos código e são mais fáceis de entender.



```
function Car() {  
  return <h2>Olá! Eu sou um carro!</h2>;  
}
```



props

# Conceitos básicos

Props, abreviação de propriedades, são entradas para um componente

- Eles são passados para um componente da mesma forma que os argumentos são passados para uma função
- Props permitem personalizar componentes e torná-los reutilizáveis em diferentes cenários.

Props são somente leitura e um componente nunca deve modificar seus próprios props. Isso reforça a ideia de "fluxo de dados unidirecional" levando a um comportamento previsível.

# Conceitos básicos

## Exemplo 1

- Neste exemplo, name é um prop sendo passado para o componente Welcome. Veja que o exemplo é mostrado tanto como componente de classe quanto de função



```
class Welcome extends React.Component {  
  render() {  
    return <h1>Olá, {this.props.name}</h1>  
  }  
}
```

```
function Welcome(props) {  
  return <h1>Olá, {props.name}</h1>  
}
```

# Conceitos básicos

## Exemplo 2

- Neste exemplo, Avatar e Username seriam componentes menores que recebem props de seu componente pai, Profile. O modelo de composição permite que você crie UIs complexas como uma combinação de componentes mais simples

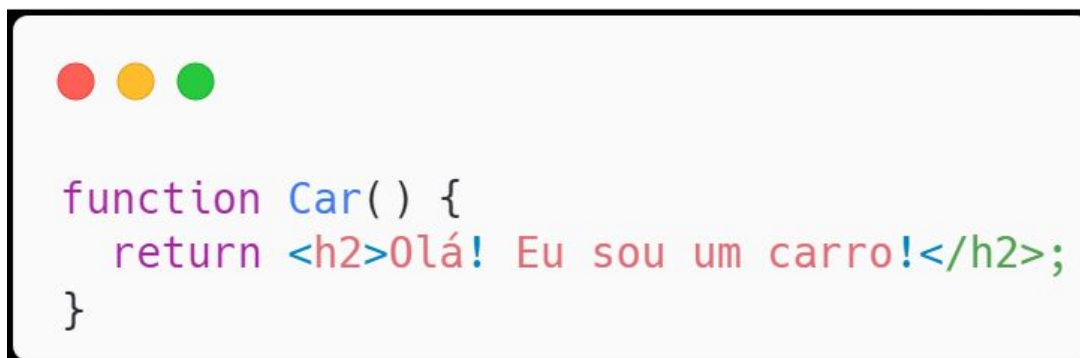


```
function Profile(props) {  
  return {  
    <div>  
      <Avatar imageUrl={props.user.imageUrl} />  
      <Username name={props.user.name} />  
    </div>  
  };  
}
```

# Renderizando componentes

# Conceitos básicos

Vamos retomar o componente visto anteriormente



```
function Car() {  
  return <h2>Olá! Eu sou um carro!</h2>;  
}
```

- A aplicação tem um componente chamado Car, que retorna um elemento <h2>.

# Conceitos básicos

Para usar este componente, vamos usar sintaxe semelhante ao HTML normal

```
main.jsx ×  
src > main.jsx > ...  
1 import React from 'react';  
2 import ReactDOM from 'react-dom/client';  
3  
4 function Car() {  
5   return <h2>Olá! Eu sou um carro!</h2>;  
6 }  
7  
8 const root = ReactDOM.createRoot(document.getElementById('root'));  
9 root.render(<Car />);
```



# Conceitos básicos

Podemos também testar utilizando props no componente

```
main.jsx ×
src > main.jsx > ...
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  function Car(props) {
5    return <h2>Olá! Eu sou um carro {props.color}</h2>;
6  }
7
8  const root = ReactDOM.createRoot(document.getElementById('root'));
9  root.render(<Car color = "vermelho"/>);
```





# Conceitos básicos

Podemos testar também chamar componentes dentro de outros componentes:

```
main.jsx ×
src > main.jsx > ...
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  function Car() {
5    return <h2>O carro!</h2>;
6  }
7
8  function Garage() {
9    return (
10     <>
11       <h1>Quem está na garagem?</h1>
12       <Car />
13     </>
14   );
15 }
16
17 const root = ReactDOM.createRoot(document.getElementById('root'));
18 root.render(<Garage />);
```

Quem está na garagem?  
O carro!

```
<div id="root">
  <h1>Quem está na garagem?</h1>
  <h2>O carro!</h2>
</div>
```

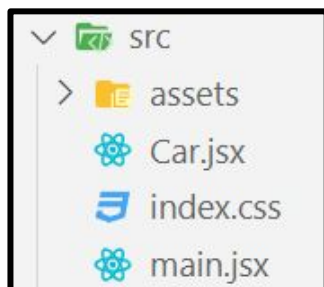
[Acesse o código comentado](#)

# Conceitos básicos

O React tem tudo a ver com a reutilização de código e é recomendável dividir seus componentes em arquivos separados

- Para fazer isso, crie um novo arquivo com extensão .js e coloque o código dentro dele. OBS: o nome do arquivo deve começar com letra maiúscula!

```
Car.jsx  X
src > Car.jsx > ...
1  function Car() {
2      return <h2>Olá! Eu sou um carro!</h2>;
3  }
4
5  export default Car;
```



```
main.jsx  X
src > main.jsx > ...
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import Car from './Car.js';
4
5  const root = ReactDOM.createRoot(document.getElementById('root'));
6  root.render(<Car />);
```

# Conceitos básicos

Também é possível programar o main.jsx como mostrado abaixo

- Aqui estamos utilizando outra forma de renderizar o componente Car.
- Nesta forma utilizamos StrictMode e importações nomeadas.



```
main.jsx ×  
src > main.jsx  
1 import { StrictMode } from 'react'  
2 import { createRoot } from 'react-dom/client'  
3 import Car from './Car.jsx'  
4  
5 createRoot(document.getElementById('root')).render(  
6   <StrictMode>  
7     <Car />  
8   </StrictMode>,  
9 )
```

# Conceitos básicos

## Vantagens da segunda forma:

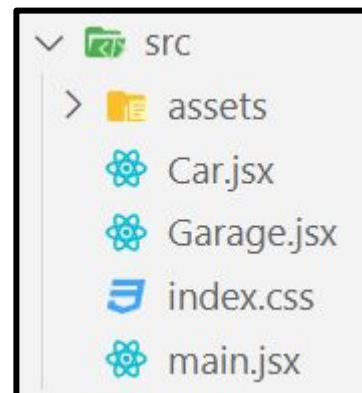
- O StrictMode é uma ferramenta valiosa no desenvolvimento React, pois ajuda a identificar problemas potenciais no código, tornando-o mais robusto. Além disso, o uso de importações nomeadas pode otimizar o tamanho do pacote final do seu projeto, melhorando o desempenho de carregamento, especialmente em aplicações maiores.

## Quando a primeira forma pode ser útil:

- Em projetos muito pequenos ou simples, onde a otimização do tamanho do pacote não é uma preocupação crítica e você não precisa das verificações extras do StrictMode, a primeira forma pode ser mais concisa e direta.

# Conceitos básicos

Mais um exemplo :)



Quem está na garagem?

Olá! Eu sou um Ford Mustang!

```
Garage.jsx
src > Garage.jsx > ...
1  import Car from './Car.jsx';
2
3  function Garage() {
4    const carInfo = { name: "Ford", model: "Mustang" };
5    return (
6      <>
7        <h1>Quem está na garagem?</h1>
8        <Car brand={ carInfo } />
9      </>
10   );
11 }
12
13 export default Garage;
```

```
Car.jsx
src > Car.jsx > ...
1  function Car(props) {
2    return <h2>Olá! Eu sou um {props.brand.name} {props.brand.model}!</h2>;
3  }
4
5  export default Car;
```

```
main.jsx
src > main.jsx
1  import { StrictMode } from 'react'
2  import { createRoot } from 'react-dom/client'
3  import Garage from './Garage.jsx'
4
5  createRoot(document.getElementById('root')).render(
6    <StrictMode>
7      <Garage />
8    </StrictMode>,
9  )
```

# State e Ciclo de vida

# State e Ciclo de vida

State e ciclo de vida são aspectos fundamentais do React que permitem a criação de interfaces de usuário interativas e dinâmicas

- State é semelhante aos props, porém privado e controlado pelo componente.
- Enquanto as props permitem que os componentes pais passem dados para seus filhos, o state é um recurso disponível apenas para componentes de classe e permite que eles criem e gerenciem seus próprios dados.

# State e Ciclo de vida

## Exemplo

- Neste exemplo, `this.state` é inicializado no construtor com a hora atual. O state pode então ser acessado e exibido dentro do método `render`.

```
Clock.jsx  X
src > Clock.jsx > ...
1  import React from 'react';
2
3  class Clock extends React.Component {
4
5      constructor(props) {
6          super(props);
7          this.state = {date: new Date()};
8      }
9
10     render() {
11         return (
12             <div>
13                 <h2>Hora atual: {this.state.date.toLocaleTimeString()}</h2>
14             </div>
15         );
16     }
17 }
18
19 export default Clock;
```

[Acesse o código comentado](#)



```
main.jsx  X
src > main.jsx
1  import { StrictMode } from 'react'
2  import { createRoot } from 'react-dom/client'
3  import Clock from './Clock.jsx'
4
5  createRoot(document.getElementById('root')).render(
6      <StrictMode>
7          <Clock />
8      </StrictMode>,
9  )
```



# State e Ciclo de vida

State pode ser atualizado usando o método `this.setState()`

- É importante usar este método para atualizar o estado porque ele informa ao React que o componente e seus filhos precisam ser renderizados novamente com o state atualizado.



```
this.setState({date: new Date()});
```

# State e Ciclo de vida

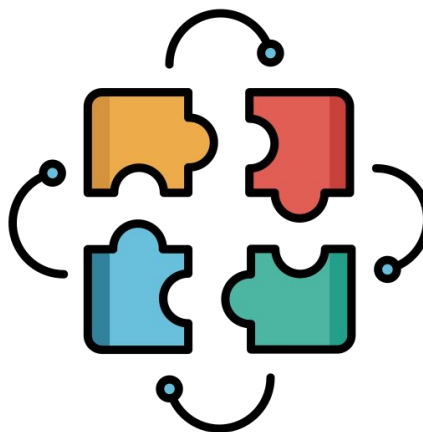
## Isso nos leva ao conceito de métodos de ciclo de vida

- Esses são métodos especiais que o React chamará em pontos específicos durante a vida de um componente em seu aplicativo, permitindo que você adicione código que deve ser executado em resposta a determinados eventos.
- Os métodos de ciclo de vida são divididos em três categorias principais:
  - montagem (mounting)
  - atualização (updating)
  - desmontagem (unmounting).

# State e Ciclo de vida

## Montagem (mounting)

- Esses métodos são chamados quando uma instância de um componente está sendo criada e inserida no DOM.
- Os principais métodos nesta categoria são `constructor()`, `componentDidMount()` e `render()`.



# State e Ciclo de vida

## Atualização (updating)

- Esses métodos são chamados quando um componente está sendo renderizado novamente como resultado de alterações em seus props ou state.
- Os principais métodos nesta categoria são `shouldComponentUpdate()`, `componentDidUpdate()` e `render()`.



# State e Ciclo de vida

## Desmontagem (unmounting)

- Esses métodos são chamados quando um componente está sendo removido do DOM. O método principal nesta categoria é `componentWillUnmount()`.



# State e Ciclo de vida

Um uso típico desses métodos é iniciar ou interromper eventos como cronômetros ou solicitações de rede em resposta a alterações no ciclo de vida do componente.

**Exemplo:** Aqui está uma versão do componente Clock que inicia um cronômetro em `componentDidMount()` e o limpa em `componentWillUnmount()`:

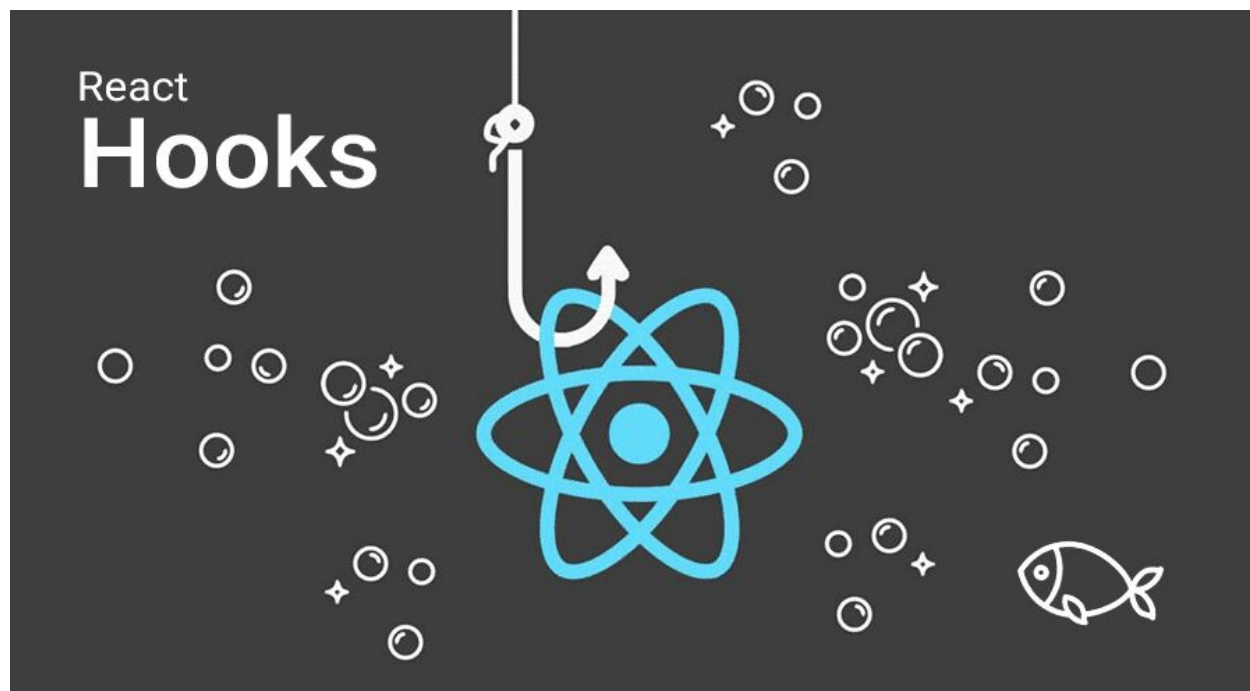
**Hora atual: 12:06:15**

[Acesse o código comentado](#)

```
Clockjsx x
src > Clockjsx > ...
1  import React from 'react';
2
3  class Clock extends React.Component {
4
5      constructor(props) {
6          super(props);
7          this.state = {date: new Date()};
8      }
9
10     componentDidMount() {
11         this.timerID = setInterval( () => this.tick(), 1000);
12     }
13
14     componentWillUnmount() {
15         clearInterval(this.timerID);
16     }
17
18     tick() {
19         this.setState({date: new Date()});
20     }
21
22     render() {
23         return (
24             <div>
25                 <h2>Hora atual: {this.state.date.toLocaleTimeString()}</h2>
26             </div>
27         );
28     }
29 }
30
31 export default Clock;
```

# State e Ciclo de vida

Atualmente, os React Hooks substituíram os componentes de classe, oferecendo várias vantagens, mas discutiremos eles posteriormente na disciplina :)



# State e Ciclo de vida

```
Clock.jsx ×
src > Clock.jsx > ...
1  import React, { useState, useEffect } from 'react';
2
3  const Clock = () => {
4    const [date, setDate] = useState(new Date());
5
6    useEffect(() => {
7      const timerID = setInterval(() => {
8        tick();
9      }, 1000);
10
11      return () => {
12        clearInterval(timerID);
13      };
14    }, []);
15
16    const tick = () => {
17      setDate(new Date());
18    };
19
20    return (
21      <div>
22        <h2>Hora atual: {date.toLocaleTimeString()}</h2>
23      </div>
24    )
25  }
26
27  export default Clock;
```

[Acesse o código comentado](#)

Como curiosidade, aqui está o mesmo código implementado usando Hooks



# Tratando eventos

# Tratando eventos

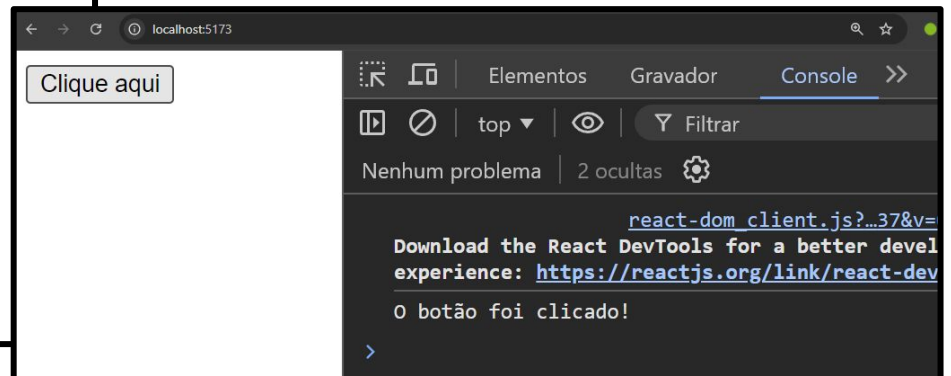
A manipulação de eventos no React é semelhante à manipulação de eventos em elementos DOM, mas existem algumas diferenças sintáticas

- Os eventos do React são nomeados usando camelCase, em vez de letras minúsculas
- Com JSX você passa uma função como o manipulador de eventos, em vez de uma string.

# Tratando eventos

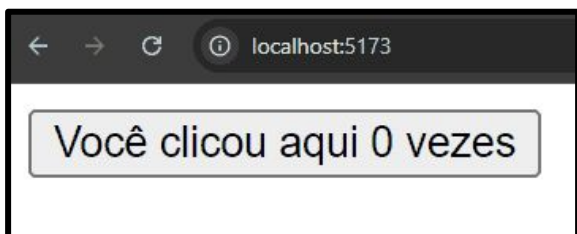
Aqui está um exemplo de um manipulador de eventos em um componente funcional usando uma arrow function:

```
Button.jsx X
src > Button.jsx > ...
1  import React from 'react';
2
3  const Button = () => {
4    const handleClick = () => {
5      console.log('O botão foi clicado!');
6    }
7
8    return (
9      <button onClick={handleClick}>
10        Clique aqui
11      </button>
12    );
13  }
14
15  export default Button;
```



# Tratando eventos

Os manipuladores de eventos também são o local onde você pode atualizar o estado do componente. Vamos expandir o exemplo anterior para aumentar um contador toda vez que o botão for clicado:



[Acesse o código comentado](#)

```
Button.jsx X
src > Button.jsx > ...
1  import React from 'react';
2
3  const Button = () => {
4    const [count, SetCount] = React.useState(0);
5
6    const handleClick = () => {
7      SetCount(count + 1)
8    }
9
10   return (
11     <button onClick={handleClick}>
12       Você clicou aqui {count} vezes
13     </button>
14   );
15 }
16
17 export default Button;
```

# Renderização condicional

# Renderização condicional

Podemos usar operadores JavaScript como if, else, operador ternário ou operador lógico && para criar ou não elementos na UI, dependendo do estado atual

```
Greeting.jsx X
src > Greeting.jsx > ...
1  const UserGreeting = () => <h1>Bem vindo novamente!</h1>;
2  const GuestGreeting = () => <h1>Por favor faça login!</h1>;
3
4  const Greeting = (props) => {
5    const isLoggedIn = props.isLoggedIn;
6
7    if (isLoggedIn) {
8      return <UserGreeting />;
9    }
10   return <GuestGreeting />;
11 }
12
13 export default Greeting;
```

**Por favor faça login!**

**Bem vindo novamente!**

[Acesse o código comentado](#)

```
main.jsx X
src > main.jsx
1  import { StrictMode } from 'react'
2  import { createRoot } from 'react-dom/client'
3  import Greeting from './Greeting.jsx'
4
5  createRoot(document.getElementById('root')).render(
6    <StrictMode>
7      <Greeting isLoggedIn={false} />
8    </StrictMode>,
9  )
```

# Renderização condicional

**Outro exemplo:** neste exemplo, o componente LoginControl mantém o estado atual com a variável de estado isLoggedIn. Ele renderiza condicionalmente um <LogoutButton /> ou um <LoginButton /> com base no estado atual.

```
import React from 'react';

const LoginControl = () => {
  const [isLoggedIn, setIsLoggedIn] = React.useState(false);

  let button;
  if (isLoggedIn) {
    button = <LogoutButton />;
  } else {
    button = <LoginButton />;
  }

  return (
    <div>
      <Greeting isLoggedIn={isLoggedIn} />
      {button}
    </div>
  );
}

export default LoginControl;
```

# Renderização condicional

[Acesse o código comentado](#)

Outra maneira poderosa de expressar a renderização condicional em JSX é usar o operador lógico `&&` do JavaScript

- Neste exemplo, o elemento `<h2>` só será incluído na saída se `unreadMessages.length > 0` for verdadeiro.

main.jsx

```
Mailbox.jsx x
src > Mailbox.jsx > ...
1  const Mailbox = (props) => {
2    const unreadMessages = props.unreadMessages;
3
4    return (
5      <div>
6        <h1>Olá!</h1>
7        {unreadMessages.length > 0 &&
8          <h2>
9            Você tem {unreadMessages.length} mensagens não lidas.
10         </h2>
11      </div>
12    );
13  };
14
15
16  export default Mailbox;
```

Mailbox.jsx

```
main.jsx x
src > main.jsx > ...
1  import { StrictMode } from 'react'
2  import { createRoot } from 'react-dom/client'
3  import Mailbox from './Mailbox.jsx'
4
5  let messagesList = ['Mensagem 1', 'Mensagem 2'];
6
7  createRoot(document.getElementById('root')).render(
8    <StrictMode>
9      <Mailbox unreadMessages={messagesList} />
10    </StrictMode>,
11  )
```

**Olá!**

**Você tem 2 mensagens não lidas.**



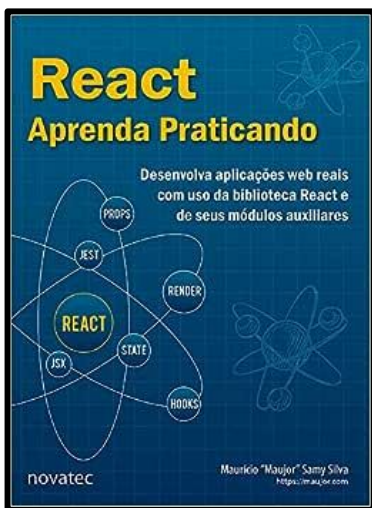
Avalie a aula!

Avalie a aula de hoje :)



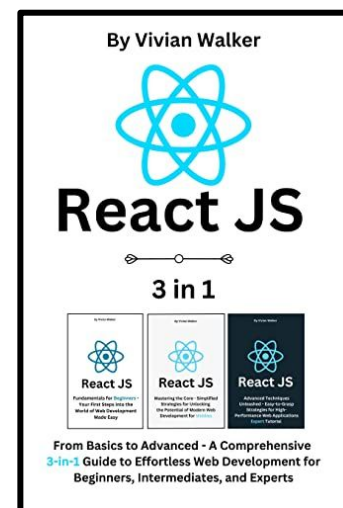
# Referências

# Referências



SILVA, Maurício Samy. **React - Aprenda praticando**. Novatec. 2021.

Walker, Vivian. **React JS: From Basics to Advanced - A Comprehensive 3-in-1 Guide to Effortless Web Development for Beginners, Intermediates, and Experts**.



# Referências



W3Schools. React Tutorial. Disponível em:  
<<https://www.w3schools.com/react/>>. Acesso em 07 ago. 2024.

Alura. React: o que é, como funciona e um Guia dessa popular ferramenta JS. Disponível em:  
<<https://www.alura.com.br/artigos/react-js>>. Acesso em 07 ago. 2024.



# Referências



Todos os ícones utilizados são gratuitos e livres para uso pessoal e comercial.

Eles foram retirados do site <https://www.flaticon.com/>.

## **Autores:**

Robert Angle, Flat Icons, Freepik, monkik

# **Análise e Desenvolvimento de Sistemas**

## **Frameworks Web I**

### **Aula 02 - React JS: parte 1**

---

---