Interface Web II

Autor: Marcos de Melo

• Utilização de ambiente de desenvolvimento integrado para construção da interface de sistemas Web.

II.10 INTERFACES WEB II

Função: Desenvolvimento de websites interativos com programação lado cliente
Classificação: Execução

Atribuições e Responsabilidades

Planejar projetos para Web.

Valores e Atitudes

Incentivar a criatividade.
Estimular o interesse na resolução de situações-problema.
Responsabilizar-se pela produção, utilização e divulgação de informações.

Competência Profissional	Habilidades
1. Desenvolver sistemas para Internet com foco na interface com o usuário e programação lado cliente.	1.1 Codificar software em linguagem para a Web. 1.2 Utilizar linguagem de script para Web. 1.3 Utilizar interface baseada em navegador para interação com usuário.

Bases Tecnológicas

Processamento script lado cliente (*Javascript*)

- Sintaxe básica, variáveis, tipos e escopo;
- Controle de fluxo e manipulação de erro;
- Laços e interação;
- Funções e manipulação *Document Object Model (DOM);*
- *XML e JSON;*
- Solicitações assíncronas *(AJAX);*
- *Cookies.*

Biblioteca *Javascript cross-browser (JQuery)*

- Instalação, função $() e seletores;
- Eventos;
- Manipulação.

Carga horária (horas-aula)

Teórica	00	**Prática Profissional**	80	**Total**	**80 Horas-aula**

Sumário

ReactJS

O **ReactJS** não é uma linguagem nova, mas sim uma "super ferramenta" feita com JavaScript para construir **interfaces de usuário (UI)**.

Para entender o React, vamos usar uma analogia que todo mundo conhece: **Lego**.

O Conceito de Componentes (As Peças de Lego)

No desenvolvimento web tradicional, você cria uma página HTML gigante. Se precisar de um botão igual em dez páginas, você copia e cola o código dez vezes.

No **React**, você cria o botão **uma vez** como uma peça de Lego (chamamos de **Componente**) e o encaixa onde quiser.

- **Vantagem:** Se você decidir mudar a cor do botão, muda em um lugar só, e todos os botões do site atualizam sozinhos.

O "Estado" (A Memória do Componente)

Imagine um painel de máquina industrial. Ele tem uma lâmpada que acende quando a temperatura passa de 80°C. No React, chamamos essa informação (a temperatura) de **State** (Estado).

- O React fica "vigiando" esse Estado.
- **Regra de Ouro:** Sempre que o Estado muda, o React redesenha a peça na tela automaticamente. Você não precisa dar um "F5" ou mandar o JavaScript manipular o HTML manualmente via document.getElementById.

O DOM Virtual (O Engenheiro Eficiente)

O **DOM** (Document Object Model) é a estrutura da sua página no navegador. Alterar o DOM diretamente é "pesado" e lento, como reformar uma casa inteira só para trocar uma lâmpada.

O React usa o **Virtual DOM**:

1. Ele tira uma "foto" da tela atual.
2. Quando algo muda, ele tira uma "nova foto".
3. Ele compara as duas e **só altera o que mudou**. Se você mudou apenas o texto de um parágrafo, o React não mexe no resto da página. Isso o torna ultra veloz.

JSX (HTML dentro do JavaScript)

No curso técnico, você aprende que HTML é uma coisa e JavaScript é outra. O React mistura os dois usando o **JSX**. Parece HTML, mas é JavaScript. Isso permite que você use lógica (como um if ou um for) direto na estrutura da página.

Exemplo Prático:

JavaScript

```
function Saudacao() {
  const hora = new Date().getHours();
  return (
    <h1>{hora < 12 ? "Bom dia!" : "Boa tarde!"}</h1>
  );
}
```

Por que aprender React no curso técnico?

É importante ressaltar que o ReactJS não é uma linguagem de programação e sim uma biblioteca Javascript, ou seja, ao escolher o react para desenvolvimento web a linguagem de programação principal que o aluno e futuro desenvolvedor irá usar será o JavaScript. O mercado de trabalho adora o React porque ele facilita o trabalho em equipe (cada um faz uma "peça" do site) e é mantido pela Meta (Facebook), o que garante que ele não vai sumir amanhã.

Principais Motivos para usar:

- **Domínio de Mercado:** É a biblioteca frontend mais utilizada no mundo, o que se traduz em um volume massivo de vagas e maior facilidade para contratação de times.
- **Ecossistema Inigualável:** Virtualmente qualquer biblioteca de componentes ou padrão de design é lançado primeiro para React.
- **Evolução Tecnológica:** A introdução de **React Server Components (RSC)** e novas APIs de renderização mantém a biblioteca relevante frente a alternativas como Vue ou Svelte.
- **Versatilidade:** O conhecimento em React permite migrar facilmente para o desenvolvimento mobile com o React Native ou para aplicações robustas de servidor com Next.js.

Pontos de Atenção:

Concorrência: Por ser a "porta de entrada" padrão, o mercado para níveis iniciantes (júnior) pode estar mais saturado e competitivo.

Complexidade: Embora a curva de aprendizado inicial seja amigável, dominar conceitos avançados como useEffect, hooks customizados e gerenciamento de estado exige tempo.

Criar um Projeto React JS

Escolher como iniciar um projeto React hoje em dia é como escolher o alicerce de uma casa: cada opção oferece um nível diferente de suporte, velocidade e "mordomia".

Ferramentas de criação

Existem muitas ferramentas de criação de projetos ReactJS, mas vamos elucidar 3 deles aqui agora.

1. Create React App (CRA)

O **CRA** foi, por muito tempo, a ferramenta oficial do Facebook. Ele "esconde" toda a configuração complexa (como Webpack e Babel) para que você foque apenas no código.

- **Como funciona:** Ele cria uma **Client-Side Rendering (CSR)**. Isso significa que o navegador baixa um arquivo HTML vazio e um pacote JavaScript gigante que monta a página na hora.
- **Status atual: Descontinuado/Legado.** A própria documentação do React não recomenda mais o uso do CRA. Ele se tornou lento para projetos grandes e não recebe atualizações significativas há tempos.
- **Vantagem:** Nenhuma hoje em dia, exceto se você estiver estudando um curso antigo que o utilize especificamente.

Observações: *em resumo, podemos perceber que esta opção não é para ser usado, visto que está em desuso, só o citamos aqui porque foi o primeiro de todos feito pelo Facebook.*

2. Vite

O **Vite** (francês para "rápido") é o sucessor espiritual do CRA para aplicações que rodam puramente no lado do cliente.

- **O Diferencial:** Ele utiliza algo chamado *Native ESM*, o que torna a inicialização do servidor de desenvolvimento e o "hot reload" (atualização da tela ao salvar) instantâneos, independentemente do tamanho do projeto.
- **Foco:** Agilidade e modernidade. Ele é extremamente leve e suporta TypeScript, JSX e CSS de forma nativa e ultra veloz.
- **Onde brilha:** Dashboards, painéis administrativos ou aplicativos onde o SEO (rankeamento no Google) não é o foco principal.

3. Next.js

O **Next.js** não é apenas um "criador de projeto", é um **framework completo** construído sobre o React.

- **Como funciona:** Ele permite **Server-Side Rendering (SSR)** e **Static Site Generation (SSG)**. Ou seja, a página já vem pronta (ou quase pronta) do servidor para o navegador.
- **Recursos Extras:** Já vem com sistema de rotas (pastas), otimização de imagens, API Routes (você pode criar um backend dentro dele) e uma performance de carregamento inicial superior.
- **Foco:** Aplicações profissionais, e-commerces, blogs e qualquer sistema que precise de SEO de ponta e altíssima performance.

Comparativo Rápido

Característica	Create React App	Vite	Next.js
Velocidade de Dev	Lenta	Ultra Rápida	Rápida
Renderização	Cliente (CSR)	Cliente (CSR)	Servidor (SSR/SSG) e Cliente
Configuração	Fechada	Flexível	Estruturada (Framework)
SEO	Ruim	Ruim	Excelente
Uso Recomendado	Evitar hoje	Apps Simples/Dashboards	Apps Complexos/Produtos

Qual é o melhor?

A resposta curta: **Depende do seu objetivo**, mas o CRA está fora da disputa.

1. **Escolha o VITE se:** Você quer aprender a base do React puro, está criando um projeto de estudos simples ou um aplicativo que vive atrás de um login (onde o Google não precisa indexar as páginas). É a escolha ideal para quem quer velocidade e simplicidade sem "mágicas" de framework.
2. **Escolha o NEXT.JS se:** Você está construindo um produto real, uma loja, um blog ou um site institucional. Se você quer seguir as melhores práticas do mercado atual e precisa de um sistema robusto que resolva roteamento e performance de forma nativa, o Next.js é o padrão ouro da indústria.

Veredito: Para a maioria dos desenvolvedores em 2026, o **Next.js** é a ferramenta mais completa, enquanto o **Vite** é a melhor alternativa para projetos leves.

Criar um projeto com o **Vite** é um processo extremamente rápido e direto. Diferente do antigo CRA, o Vite não instala tudo de uma vez, ele primeiro, baixa uma pequena ferramenta de "scaffolding" (andaime) que te pergunta o que você quer usar.

Criando um projeto com Vite

Vamos compreender agora como criar projetos web utilizando a linguagem javascript, utilizando a biblioteca de javascript ReactJS. Entenderemos como o ReactJS funciona e quais os recursos ele possui para desenvolvermos sistemas web mais rápido. Para isso, vamos utilizar o criador de projetos Vite que já falamos sobre ele anteriormente.

Passo 1: Pré-requisitos

Certifique-se de ter o **Node.js** instalado na sua máquina. Você pode verificar abrindo seu terminal e digitando:

Terminal

```
node -v
```

(Recomenda-se a versão 18 ou superior).

Certifique-se de ter um editor de códigos para programar no projeto, recomendamos o ***Visual Studio Code***, pois utilizaremos ele nos exemplos das aulas.

Passo 2: Executar o comando de criação

Abra o terminal na pasta onde deseja salvar seu projeto e digite o seguinte comando:

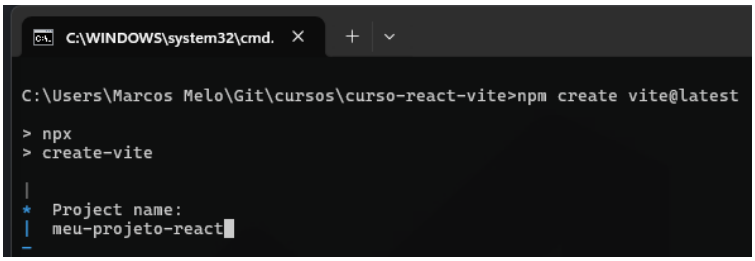
Terminal

```
npm create vite@latest
```

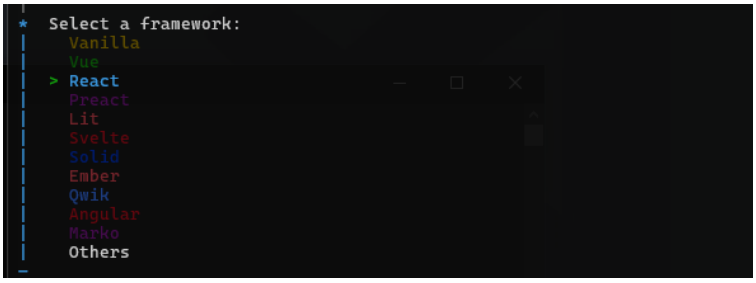
Passo 3: Configurar o projeto (Menu Interativo)

O terminal vai iniciar um assistente. Siga estes passos usando as setas do teclado e o Enter:

1. **Project name:** Digite o nome da pasta do seu projeto (ex: meu-projeto-react).

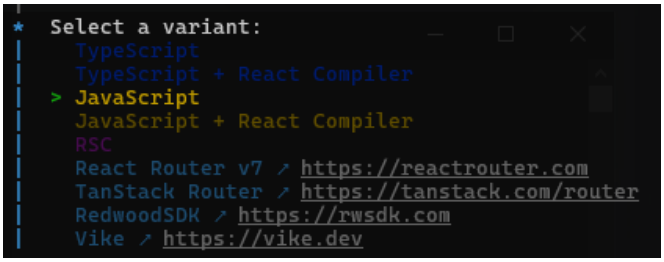


2. **Select a framework:** Use as setas para selecionar **React**.



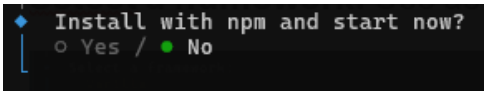
3. **Select a variant:**

- Escolha **JavaScript** (se quiser o básico).
- Escolha **TypeScript** (se quiser tipagem e um código mais profissional).



4. **Install with npm and start now?**

Nesta etapa da criação é perguntado se você quer instalar a dependências e já executar o projeto, selecione **No** porquê vamos fazer isso posteriormente.



Passo 4: Instalar as dependências

Se você respondeu **No** na última etapa, o Vite cria a estrutura de pastas instantaneamente, mas ele não instala os pacotes do Node. Entre na pasta e instale-os manualmente:

Terminal

```
cd meu-projeto-react
npm install
```

Passo 5: Rodar o projeto

Antes de abrir o projeto vamos abrir a pasta do projeto no editor de códigos VSCODE.

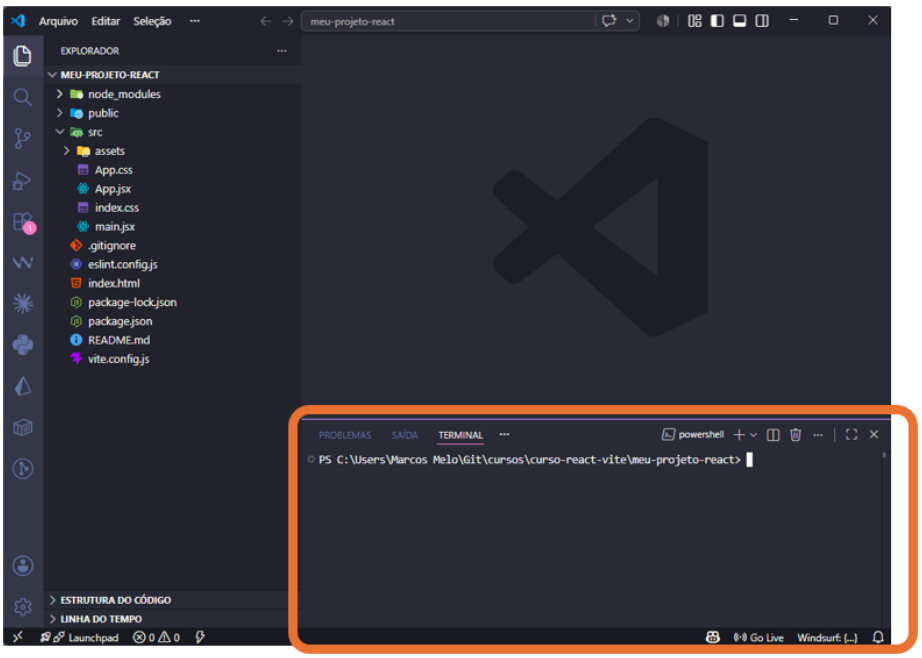
Digite o comando a seguir para abrir no VSCode.

Terminal

```
code .
```

Após seu projeto abrir no editor de códigos VSCODE, não é mais necessário ficar aberto a janela do terminal, então pode fechá-la.

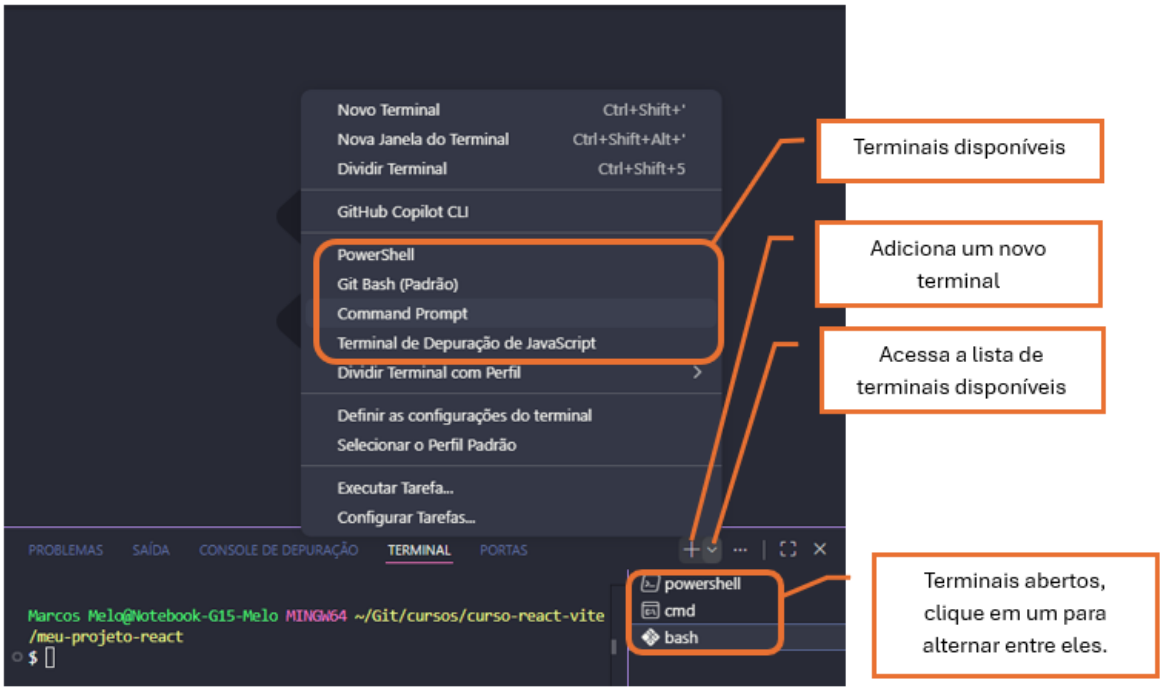
Mas fechar o terminal, não quer dizer que não vamos realizar comandos no terminal, pelo contrário vamos executar comandos no terminal a todo momento durante a criação do projeto e para isso o VSCODE possui o recurso de carregar o terminal dentro dele bastando digitar **CTRL** + **J** para acessá-lo.



Mudando ou abrindo mais de um terminal

O VSCODE pode abrir mais de um terminal, para que seja possível executar novos comandos sem ter que esperar comandos anteriores terminarem, ou até mesmo, ter que parar a execução de um comando de serviço para rodar o próximo comando.

Veja na imagem a seguir como adicionar novos terminais e como alternar entre eles



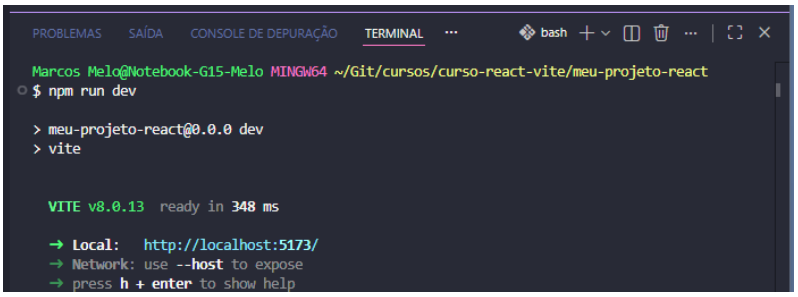
Iniciando o projeto no navegador

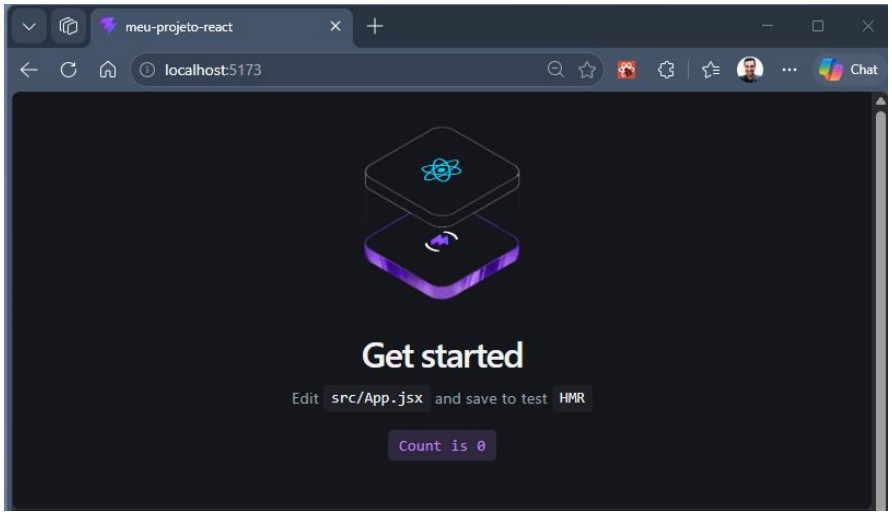
O criador de projetos Vite, cria um projeto com toda a estrutura para já funcionar no navegador, ele ainda não faz o que queremos, mas ao longo do desenvolvimento vamos acrescentando, então para ver o projeto funcionando, basta iniciar o servidor de desenvolvimento com o comando a seguir no terminal:

Terminal

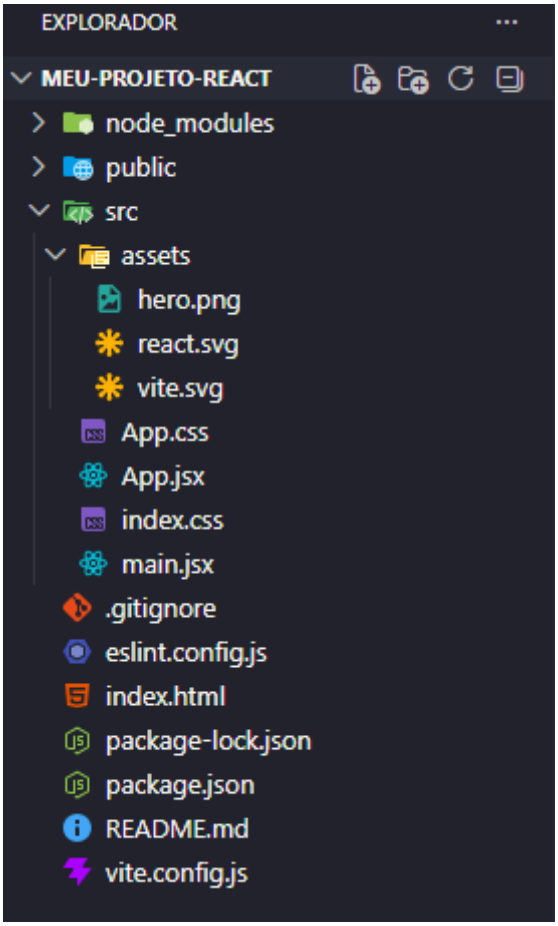
```
npm run dev
```

Se tudo deu certo, o terminal te mostrar um endereço (geralmente ***http://localhost:5173***). Segure Ctrl e clique no link ou copie e cole no navegador para ver seu projeto rodando!





Resumo da Estrutura de Pastas



Ao abrir a pasta no seu editor (como o VSCode), você verá algo assim:

- **index.html**: Fica na raiz, o que torna o carregamento mais rápido.
- **src/**: Onde você passará 99% do seu tempo escrevendo componentes.
- **vite.config.js**: Onde você adiciona plugins ou configurações extras se precisar.

Dica de Atalho: *Se você já souber exatamente o que quer e quiser pular o menu interativo, pode rodar tudo em uma única linha:*

Terminal

```
# Exemplo criando um projeto React com TypeScript direto
npm create vite@latest meu-projeto -- --template react-ts
```

O arquivo index.html

O arquivo index.html na raiz de um projeto React é a **porta de entrada única** de toda a sua aplicação web.

Em aplicações React modernas (Single Page Applications ou SPA), o navegador carrega apenas este arquivo HTML uma única vez.

A Função Principal

O index.html serve como um "esqueleto" vazio. Ele não contém o conteúdo das suas páginas, mas sim um ponto de ancoragem onde o React irá injetar dinamicamente toda a interface que você programar em JavaScript.

🔍 Estrutura Interna Comum

Os Componentes Chave

- **<head>**: Centraliza as configurações de SEO, links de fontes externas (como Google Fonts) e scripts globais de terceiros (como Google Analytics).
- ** **: É a linha mais importante. O React busca por esse id="root" e renderiza todo o seu código JavaScript dentro dele.
- **<script type="module" src="/src/main.jsx"></script>:** É a chamada para o script do arquivo JavaScript principal da biblioteca ReacJS, responsável por renderizar dentro do JavaScript todo o conteúdo html da estrutura do layout do site.

Como a Mágica Acontece

1. O navegador do usuário acessa seu site e baixa o index.html.
2. O arquivo carrega e exibe uma tela em branco (ou com uma mensagem de carregamento).
3. A ferramenta de build (Vite, Webpack) insere automaticamente uma tag <script> apontando para o seu código React (main.jsx ou index.js).

4. O JavaScript do React entra em ação, encontra a div com o id root e monta toda a interface da sua aplicação ali dentro.

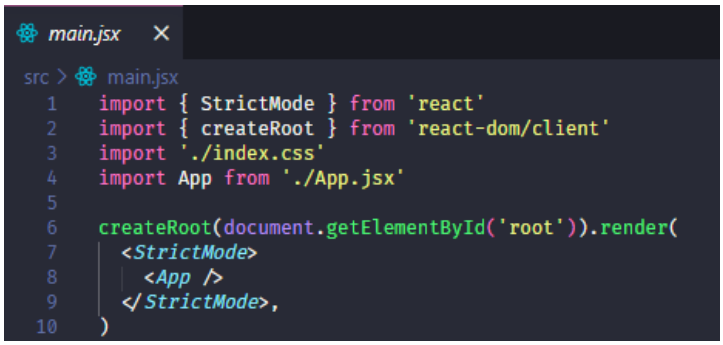
O arquivo main.jsx

O arquivo main.jsx (ou index.js em projetos mais antigos) é a **ponte de concreto** que liga o seu código React ao arquivo index.html.

Enquanto o index.html fornece o esqueleto vazio, o main.jsx é o responsável por pegar todo o seu aplicativo e "injetá-lo" dentro daquela .

🔍 Estrutura Interna Comum

Em um projeto moderno (usando Vite e React 18+), o arquivo geralmente se parece com isto:



Os Componentes Chave

- **import React e createRoot**: Importa as ferramentas necessárias da biblioteca react-dom para desenhar e gerenciar os elementos no navegador.
- **import App from './App.jsx'**: Importa o componente principal da sua aplicação. O App funciona como a "caixa mãe" que contém todas as outras páginas e componentes do seu site.
- **import './index.css'**: Aplica os estilos CSS globais (como resets, fontes e cores de fundo) que valerão para todo o site.
- **document.getElementById('root')**: Uma instrução padrão do JavaScript para procurar a div com o id root lá no index.html.
- **createRoot(...).render(...)**: Cria a raiz do React no navegador e renderiza (desenha) o componente <App /> dentro dela.
- **<StrictMode>**: Uma ferramenta interna do React que ajuda a encontrar erros comuns no código durante a fase de desenvolvimento. Ela não gera nenhum visual na tela.

O Fluxo da Inicialização

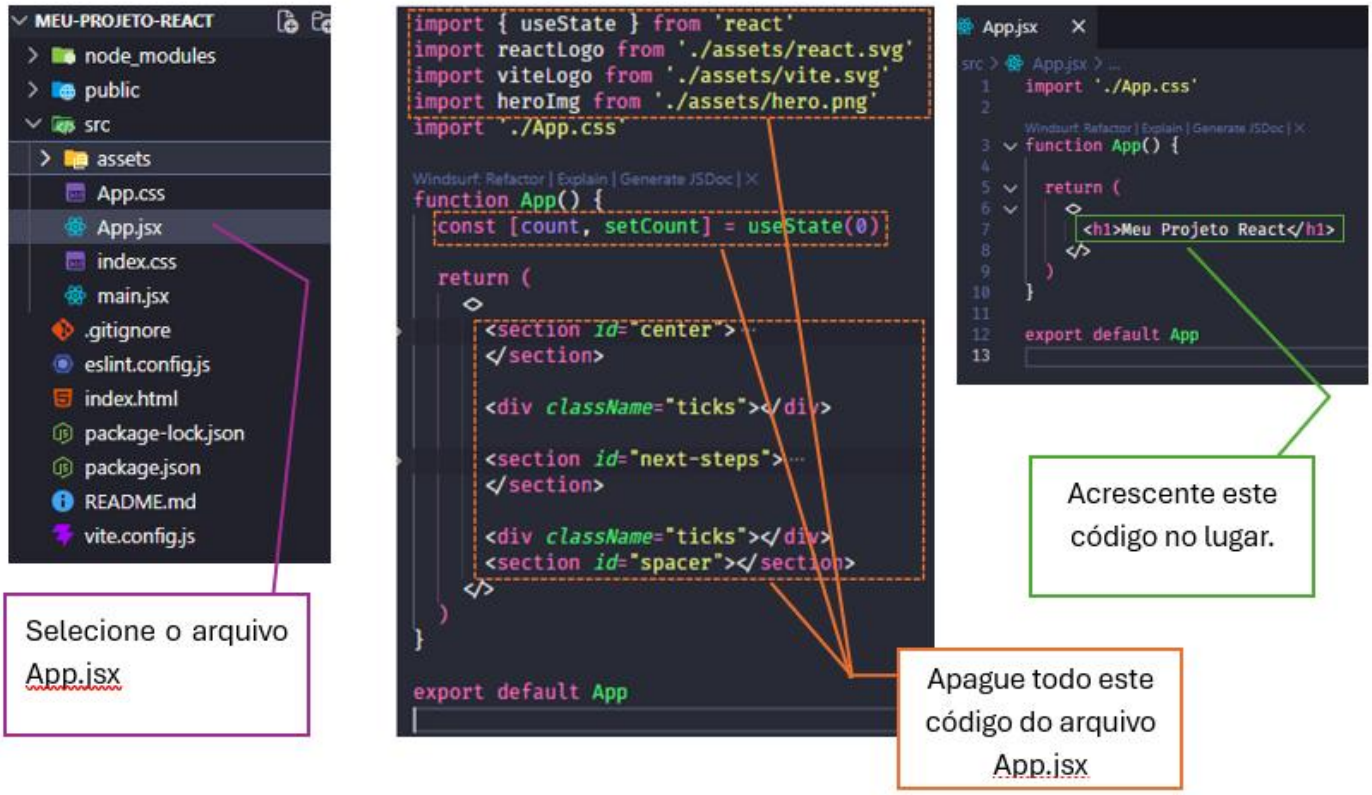
1. O navegador lê o index.html.
2. O HTML chama o script main.jsx.
3. O main.jsx localiza a div #root.

4. O **main.jsx** limpa o conteúdo dessa div e coloca o componente <App /> dentro dela.
5. A partir desse momento, o React assume o controle total da tela.

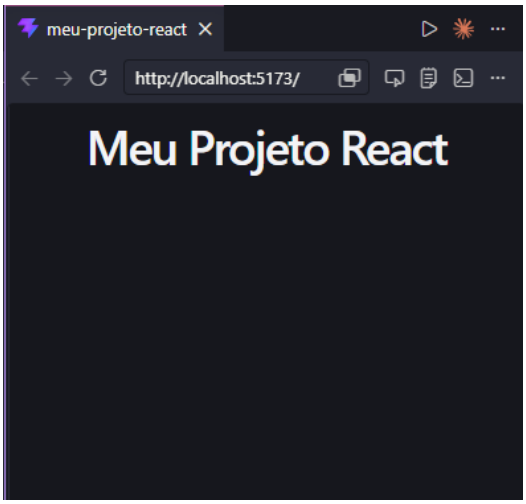
Exibindo String na tela

Agora que aprendemos como tudo funciona, vamos modificar o arquivo principal, o App.jsx que contém todo o código html que será renderizado no layout do site.

Modifique o código como mostrado a seguir;

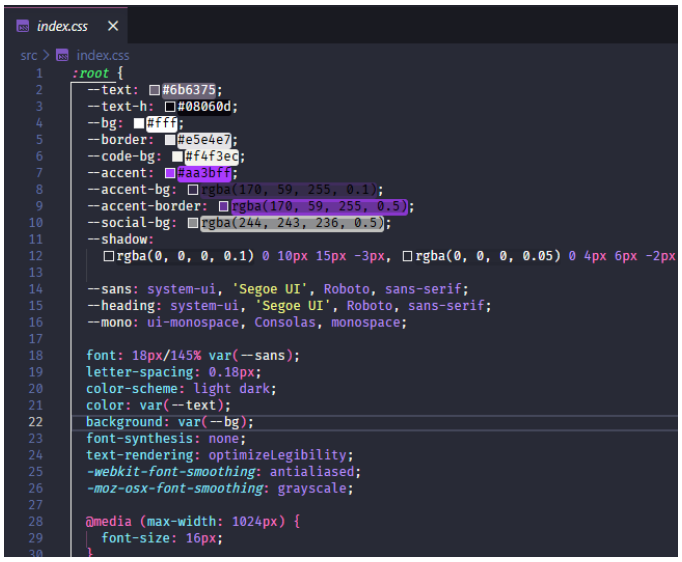


Resultado



Conhecendo CSS

O criador de projetos ***Vite*** já cria uma estrutura de CSS pronta para o modelo de exemplo, e são mostrados nas imagens abaixo.

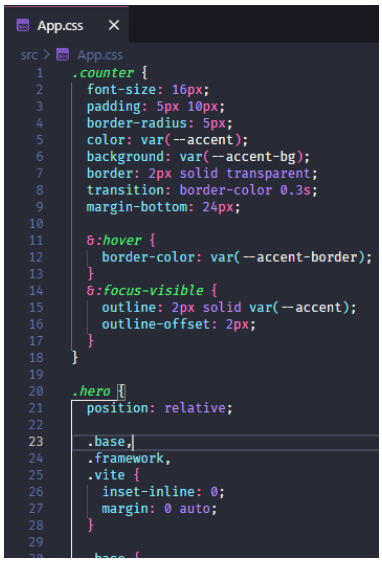


```css
:root {
  --text: #6b6375;
  --text-h: #08060d;
  --bg: #fff;
  --border: #e5e4e7;
  --code-bg: #f4f3ec;
  --accent: #aa3bff;
  --accent-bg: rgba(170, 59, 255, 0.1);
  --accent-border: rgba(170, 59, 255, 0.5);
  --social-bg: rgba(244, 243, 236, 0.5);
  --shadow:
    rgba(0, 0, 0, 0.1) 0 10px 15px -3px, rgba(0, 0, 0, 0.05) 0 4px 6px -2px

  --sans: system-ui, 'Segoe UI', Roboto, sans-serif;
  --heading: system-ui, 'Segoe UI', Roboto, sans-serif;
  --mono: ui-monospace, Consolas, monospace;

  font: 18px/145% var(--sans);
  letter-spacing: 0.18px;
  color-scheme: light dark;
  color: var(--text);
  background: var(--bg);
  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;

  @media (max-width: 1024px) {
    font-size: 16px;
  }
```



```css
.counter {
  font-size: 16px;
  padding: 5px 10px;
  border-radius: 5px;
  color: var(--accent);
  background: var(--accent-bg);
  border: 2px solid transparent;
  transition: border-color 0.3s;
  margin-bottom: 24px;

  &:hover {
    border-color: var(--accent-border);
  }
  &:focus-visible {
    outline: 2px solid var(--accent);
    outline-offset: 2px;
  }
}

.hero {
  position: relative;

  .base,
  .framework,
  .vite {
    inset-inline: 0;
    margin: 0 auto;
  }
```

O arquivo index.css

O arquivo **index.css** funciona como a folha de estilo global da sua aplicação. É o lugar ideal para definir regras CSS que devem ser aplicadas a todo o projeto, independentemente do componente que está na tela.

Geralmente, ele é importado logo na "porta de entrada" do app (no arquivo **main.jsx** ou **main.tsx**), o que garante que suas regras sejam carregadas assim que a página abre.

Para que ele é mais utilizado?

Resets e Normalizações: Limpar as margens padrões dos navegadores, definir **box-sizing: border-box** em todos os elementos e garantir um comportamento consistente entre navegadores (Chrome, Firefox, Safari, etc.).

Estilização do Layout Base: Configurar as propriedades do **html** e do **body**, como a cor de fundo padrão da página, o tamanho de fonte base e a altura mínima (**min-height: 100vh**).

Tipografia e Fontes: Definir a família de fontes principal do projeto (**font-family**) e importar fontes externas (como o **Google Fonts**).

Variáveis Globais (Custom Properties): Centralizar as cores da sua paleta, espaçamentos e temas (como **Dark/Light Mode**) usando variáveis nativas do CSS (**:root { --cor-primaria: #646cff; }**).

Integração de Frameworks: Se você for usar o **Tailwind CSS**, por exemplo, é dentro do index.css que você insere as diretivas **@tailwind base;**, **@tailwind components;** e **@tailwind utilities;**

index.css vs App.css (Qual a diferença?)

Quando você cria um projeto padrão no Vite, ele vem com esses dois arquivos. A separação lógica costuma ser esta:

Arquivo	Escopo	Objetivo Principal
index.css	**Global**	Afeta toda a estrutura da página (html, body, resets e variáveis).
App.css	**Componente**	Afeta especificamente o componente principal (App.jsx) e seus filhos diretos, servindo para o layout inicial da aplicação.

💡 ***Dica de organização:*** *Conforme seu projeto cresce, depender apenas de estilos globais pode gerar conflitos de classes (uma regra escrita para um botão no topo pode quebrar um botão no rodapé). Para evitar isso, costuma-se manter no index.css apenas o que realmente for global, e usar* ***CSS Modules*** *(NomeDoComponente.module.css),* ***Styled Components*** *ou* ***Tailwind*** *para os estilos específicos de cada componente.*

Removendo os estilos Padrões

Como estamos no início do curso, aprendendo passo-a-passo tudo sobre o react, vamos remover os estilos para colocá-los novamente com detalhes posteriormente, não será preciso deletar os arquivos, mas vamos apagar o seu conteúdo.

Apague o conteúdo dos arquivos **index.css** e **app.css**.

Componentização

Para entender a estrutura de componentes no React de forma didática, pense no React como um **jogo de Lego**.

Em vez de construir uma página web inteira em um único arquivo de texto gigante (como fazíamos no HTML tradicional), no React nós criamos **pequenas peças coloridas e independentes (os componentes)**. Depois, juntamos essas peças para formar a interface do usuário.

Aqui está o passo a passo de como essa estrutura funciona na prática:

1. O que é, afinal, um Componente?

Visualmente, um componente é qualquer pedaço da tela que isola uma função ou design: um botão, uma barra de pesquisa, um card de produto ou até a página inteira.

Tecnicamente, no React moderno, **um componente é apenas uma função JavaScript**. A única diferença é que essa função retorna HTML (que no React chamamos de **JSX**).

```javascript
// Isto é um componente React completo, limpo e isolado
function BotaoEnviar() {
  return (
    <button className="btn-enviar">
      Enviar Dados
    </button>
  );
}
```

2. A Árvore de Componentes (Hierarquia)

Os componentes não ficam soltos; eles se encaixam uns dentro dos outros, criando uma hierarquia parecida com uma árvore genealógica (Pai, Filhos, Netos).

Imagine a estrutura de um painel de controle simples:

- **App** (O componente pai de todos, a "caixa do Lego")
    - **Header** (Componente filho)
        - **Logo** (Componente neto)
        - **PerfilUsuario** (Componente neto)
    - **Sidebar** (Componente filho com os menus)
    - **Dashboard** (Componente filho com o conteúdo principal)
        - **CardGrafico** (Componente neto)
        - **TabelaVendas** (Componente neto)

No código, o componente pai (App) apenas importa e "chama" os filhos como se fossem novas tags HTML:

```javascript
import Header from './components/Header';
import Sidebar from './components/Sidebar';
import Dashboard from './components/Dashboard';

function App() {
  return (
    <div className="layout-app">
      <Header />
      <div className="conteudo-flex">
        <Sidebar />
        <Dashboard />
      </div>
    </div>
  );
}
```

3. Os Três Pilares de um Componente

Para que essa estrutura funcione de forma dinâmica, todo componente se apoia em três conceitos fundamentais:

A. Reutilização (Reusability)

Você escreve o código uma vez e o usa quantas vezes quiser. Se você tem um componente <CardProduto/>, você pode listá-lo 20 vezes na tela mudando apenas os dados internos.

B. Propriedades (Props)

Como os componentes são isolados, eles precisam de uma forma de conversar. As **Props** são os argumentos que o componente pai passa para o componente filho. Pense nelas como os atributos do HTML (como o src de uma tag img).

C. Estado (State)

As *Props* são fixas (o filho apenas lê o que o pai mandou). Se o componente precisa memorizar alguma informação que muda com o tempo — como o texto de um campo de busca ou se um menu está aberto ou fechado —, ele usa o **Estado** (através do hook useState). O estado é a memória interna do componente.

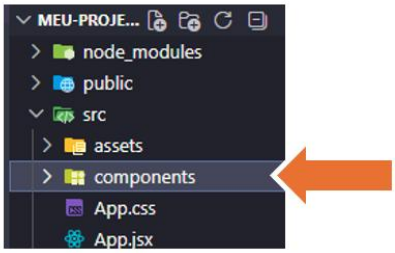
Resumo das Vantagens

- **Manutenção fácil:** Se o botão do sistema quebrar, você mexe apenas no arquivo do botão, sem risco de quebrar o resto do site.
- **Organização:** Cada componente fica em seu próprio arquivo (ex: Card.jsx e Card.css andam juntos).
- **Produtividade:** Sua equipe pode focar em criar componentes genéricos e robustos que serão reaproveitados em vários projetos.

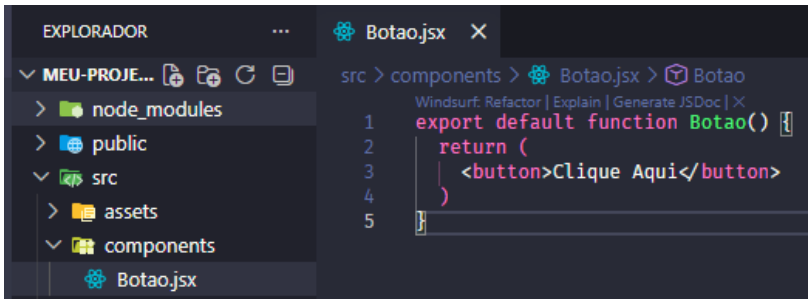
Primeiro Componente #01

Agora que já foi explicado como é o funcionamento de um componente vamos explicar mais detalhadamente criando o primeiro componente da aplicação, podemos criar o componente diretamente na raiz e dentro da pasta **./src** do projeto, porém, para ficar mais organizado e seguir um

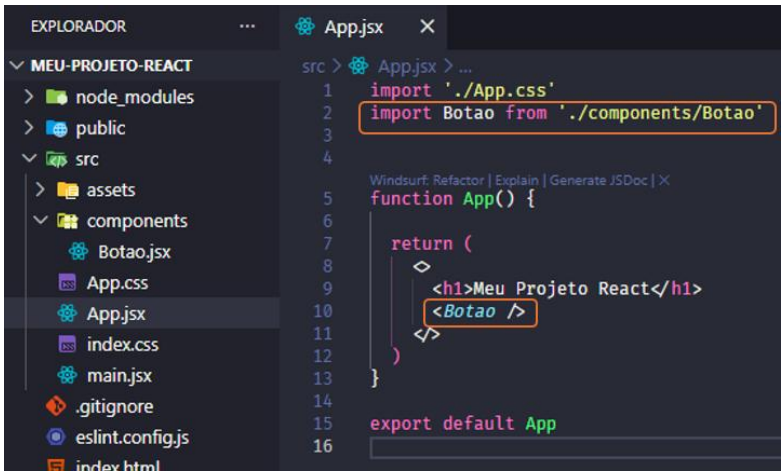
padrão de desenvolvimento, vamos criar dentro da pasta **src** uma pasta chamada **components**, para criar dentro dela todos os arquivos de componentes.



Crie dentro da pasta components o arquivo **Botao.jsx** e dentro deste componente o código deste componente, assim como, na imagem a seguir;



Após criar o arquivo do componente Botao.jsx, para utilizá-lo é preciso importá-lo dentro do componente principal, o **App.jsx**. Acrescente no arquivo App.jsx o código em destaque a seguir;

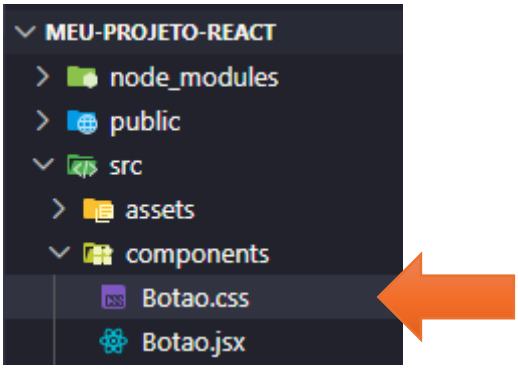


Execute o projeto novamente com o comando **npm run dev** e veja o resultado no navegador.

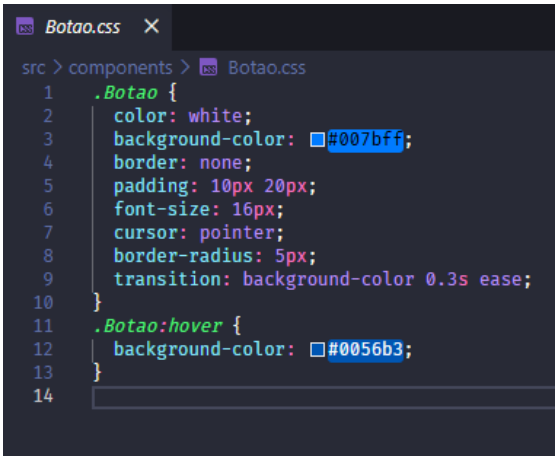
Aplicando CSS diretamente no Componente

Já comentamos anteriormente que, quando a aplicação ficar muito complexa, com muitos componentes aplicar estilos CSS pode gerar problemas de renderização de estilos CSS indevidamente e que uma das soluções seria criar um arquivo css exclusivo para o componente. Vamos criar um arquivo css para o nosso componente ***Botao.jsx*** e daremos a ele o mesmo nome do componente.

- Crie o arquivo ***Botao.css*** na pasta components.



- Selecione o arquivo Botao.css e digite o código a seguir nele, este código vai aplicar estilização no componente em que este arquivo css for importado.



- Importando o arquivo do Botao.css no arquivo Botao.jsx.

- Veja no navegador o botão terá a aparência estilizada conforme mostrado na imagem abaixo.

Reutilização de componentes

Componentes podem ser reutilizados mais de uma vez no projeto, é só chamar o componente mais de uma vez.

App.jsx

```jsx
import './App.css'
import Botao from './components/Botao'


function App() {

  return (
    <>
      <h1>Meu Projeto React</h1>
      <Botao />
      <Botao />
    </>
  )
}

export default App
```

Componente com Propriedade

Botao.jsx

src > components > Botao.jsx > Botao

```jsx
import './Botao.css';
Windsurf: Refactor | Explain | Generate JSDoc |
export default function Botao(props) {
  return (
    <button className="Botao">{props.texto}</button>
  )
}
```

App.jsx

src > App.jsx > ...

```jsx
import './App.css'
import Botao from './components/Botao'


Windsurf: Refactor | Explain | Generate JSDoc |
function App() {

  return (
    <>
      <h1>Meu Projeto React</h1>
      <Botao texto="Entrar" />
      <Botao texto="Cadastrar" />
    </>
  )
}

export default App
```

Propriedades são Somente Leitura

React Fragment

Desafio Número Aleatório

Componente com Filho #01

Componente com Filho #02

Repetição

Desafio Repetição

Renderização Condicional #01

Renderização Condicional #02

Renderização Condicional #03

Comunicação Direta

Comunicação Indireta

Componente com Estado

Componente Controlado

Styled Components

É uma das bibliotecas mais populares para estilização no ecossistema React. Ela utiliza o conceito de **CSS-in-JS**, o que significa que você escreve código CSS real diretamente dentro dos seus arquivos JavaScript/TypeScript.

Por que usar Styled Components?

1. **Escopo Automático:** O CSS que você escreve para um componente **nunca vaza** para outros. A biblioteca gera classes únicas automaticamente.
2. **Estilização Dinâmica:** Você pode passar **props** para seus estilos. Se um botão precisa mudar de cor quando está "ativo", você faz isso com lógica JS dentro do CSS.
3. **Manutenção Simples:** É muito mais fácil encontrar onde um estilo está definido, pois ele vive junto com o componente.
4. **Sintaxe Real de CSS:** Diferente de estilos inline (style={{ backgroundColor: 'red' }}), você usa a sintaxe padrão do CSS.

Como instalar no Vite

No terminal do seu projeto, rode:

Terminal

```
npm install styled-components
```

Dica: *Se estiver usando TypeScript, as definições de tipo já vêm inclusas nas versões mais recentes.*

Exemplo Prático de Uso

```
// Criamos um componente chamado 'BotaoColorido'
// Ele é, essencialmente, um elemento <button> com estilos
const BotaoColorido = styled.button`
  background-color: #6200ee;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 16px;

  &:hover {
    background-color: #3700b3; // O '&' referencia o próprio elemento
  }
`;

// No seu componente React:
function App() {
  return (
    <div>
      <BotaoColorido>Clique aqui!</BotaoColorido>
    </div>
  );
}
```

Imagine que você quer criar um botão que muda de cor se for "primário":

JavaScript

```
import styled from 'styled-components';

// Criando um componente de botão estilizado
const Button = styled.button`
  background: ${props => props.$primary ? "#BF4F74" : "white"};
  color: ${props => props.$primary ? "white" : "#BF4F74"};
  font-size: 1em;
  margin: 1em;
  padding: 0.25em 1em;
  border: 2px solid #BF4F74;
  border-radius: 3px;
  cursor: pointer;

  &:hover {
    opacity: 0.8;
  }
`;
```

```
export default function App() {
  return (
    <div>
      <Button>Botão Normal</Button>
      <Button $primary>Botão Primário</Button>
    </div>
  );
}
```

Dica Extra: *Se você estiver usando o* ***VS Code****, instale a extensão* ***vscode-styled-components****. Ela adiciona realce de sintaxe (syntax highlighting) e autocompletar ao CSS dentro do JavaScript, o que é essencial para a produtividade.*

Styled Components vs. Tailwind CSS

Atualmente, a disputa no mercado está entre essas duas abordagens. Veja qual escolher:

Característica	Styled Components	Tailwind CSS
Aprendizado	Fácil (é CSS puro)	Médio (precisa aprender as classes)
Organização	Código limpo no JSX	JSX pode ficar "poluído" de classes
Performance	Processado no JS (Runtime)	Gerado no build (Zero Runtime)
Uso Ideal	Sistemas de Design complexos	Prototipagem rápida e sites leves