

React JS: Da Gênese ao Ecossistema Moderno com Vite

Uma jornada técnica através da evolução do React, desde sua criação no Facebook até as inovações do React 19, explorando a arquitetura moderna com Vite e as melhores práticas de desenvolvimento front-end.

```
1  NameArt ()  
2  {  
3      #include <iostream>  
4      cout << " (de)start fails ()"  
5      r = Ametix ();  
6      try{Ametix::start();}catch(...){cout << "start failed ("<< r<<")";}  
7      cout << endl;  
8  }  
9  
10  NameArt (char Ametix ())  
11  {  
12      #include <iostream>  
13      cout << " (de)start success ("<< Ametix << ")";  
14      r = Ametix ();  
15      try{Ametix::start();}catch(...){cout << "start failed ("<< r<<")";}  
16      cout << endl;  
17  }  
18  
19  >> cout << " (de)start success ("<< r<< ")";  
20  cout << endl;  
21  cout << " (de)start success ("<< r<< ")";  
22  cout << endl;  
23  }  
24  >> cout << " (de)start success ("<< r<< ")";  
25  cout << endl;
```

A Origem do React: Resolvendo o Caos da Interface



O Problema Original

Em 2011, o Facebook enfrentava um desafio crítico: a sincronização de interfaces complexas com dados em constante mudança. O modelo tradicional de atualização do DOM causava bugs e inconsistências.

A Solução Revolucionária

Jordan Walke criou o React para implementar **Data Flow Unidirecional**, eliminando a complexidade do two-way binding e tornando o comportamento da UI previsível e debugável.



A Quebra de Paradigma: JSX e Componentização

JSX: HTML no JavaScript

Uma sintaxe controversa que mistura marcação com lógica, quebrando décadas de separação de concerns. O resultado? Componentes verdadeiramente encapsulados.

```
<Button onClick={handleClick}>  
  {isLoading ? 'Carregando...' : 'Enviar'}  
</Button>
```

Componentes como Blocos

A UI deixa de ser uma estrutura monolítica e se torna um sistema de componentes reutilizáveis, testáveis e compostos hierarquicamente.

```
const App = () => (  
  <Layout>  
    <Header />  
    <Content />  
  </Layout>  
>);
```

Vite: O Novo Motor do Desenvolvimento React



Create React App (CRA)

Webpack bundling completo a cada alteração. Tempo de inicialização: 30-60s em projetos médios. HMR lento em aplicações grandes.

Bundling Tradicional

- Processa todos os arquivos no boot
- Bundle completo antes do servidor
- Lentidão proporcional ao tamanho

Vite (Próxima Geração)

Native ES Modules + Esmake. Inicialização instantânea. HMR em milissegundos independente do tamanho do projeto.

Abordagem On-Demand

- Serve módulos nativos via HTTP
- Compila apenas o que é requisitado
- Performance constante e escalável



Tecnologias por Trás do Vite

1

Native ES Modules

Navegadores modernos importam módulos JavaScript nativamente sem necessidade de bundling. O Vite serve os arquivos diretamente via HTTP/2, eliminando o overhead de build durante o desenvolvimento.

2

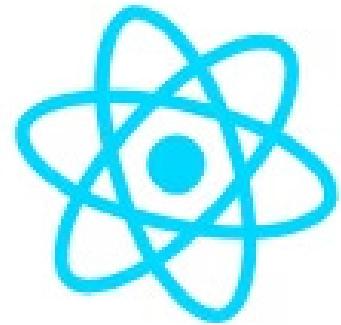
Esbuild (Go)

Escrito em Go, o Esbuild é 10-100x mais rápido que bundlers JavaScript tradicionais. Realiza o pre-bundling de dependências node_modules com velocidade incomparável.

3

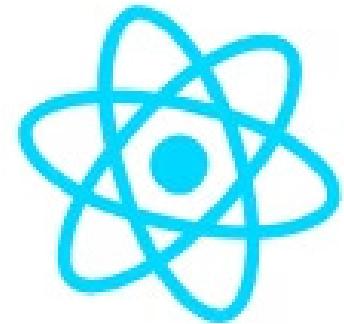
Rollup (Produção)

Para builds de produção, o Vite usa Rollup, gerando bundles otimizados com tree-shaking avançado, code splitting automático e assets otimizados.



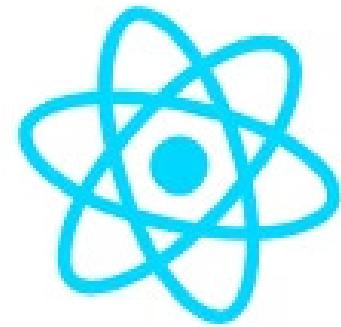
npm create vite@latest

Criação de projeto com a versão mais recente do VITE →
Perguntas são feitas



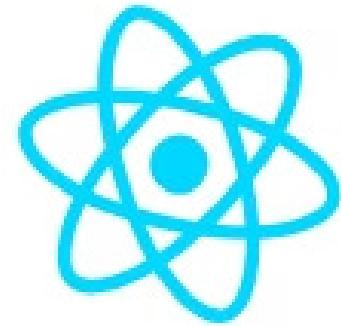
`npm create vite@latest
projeto -- --template react`

- `projeto`: O nome da pasta que será criada.
- `--`: Informa ao npm que os próximos argumentos são para o script do Vite, não para o npm.
- `--template react`: Define que você quer o React com **JavaScript** puro.



npm install (ou npm i)

- Instala dependências



npm run dev

- Vamos executar o projeto → Iniciar o servidor de desenvolvimento

Está usando contêiner? → Adicione `--host` no script dev em `package.json`

Anatomia de um Projeto Vite + React

01

index.html (Raiz)

Diferente do CRA, fica na raiz do projeto. Contém o ponto de entrada com `<div id="root">` e referência direta ao `main.jsx` via ES Module.

03

App.jsx (Componente Raiz)

Primeiro componente funcional da aplicação. Define a estrutura base e orquestra subcomponentes.

02

main.jsx (Inicialização)

Usa `createRoot` da API React 18+ para renderização concorrente. Importa e monta o componente App no DOM.

04

vite.config.js (Motor)

Configura plugins (como `@vitejs/plugin-react`), aliases de path, variáveis de ambiente e otimizações de build.

Estrutura de Pastas: O Código na Prática

index.html

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module"
      src="/src/main.jsx">
    </script>
  </body>
</html>
```

App.jsx

```
import { useState } from 'react'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <div className="App">
      <h1>Vite + React</h1>
      <button onClick={() =>
        setCount(c => c + 1)}>
        Count: {count}
      </button>
    </div>
  )
}

export default App
```

main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(
  document.getElementById('root')
).render(<App />)
```

vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

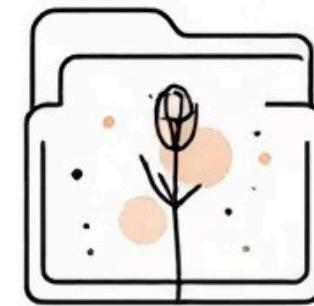
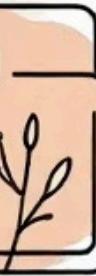
export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: { '@': '/src' }
  }
})
```

Public vs src/assets: Quando Usar Cada Pasta

/public

Arquivos servidos diretamente sem processamento. Mantêm o nome original. Ideal para: favicon.ico, robots.txt, manifest.json, assets referenciados por URL absoluta.

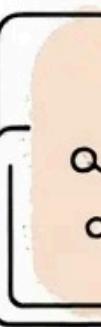
```
<link rel="icon"  
      href="/favicon.ico" />
```



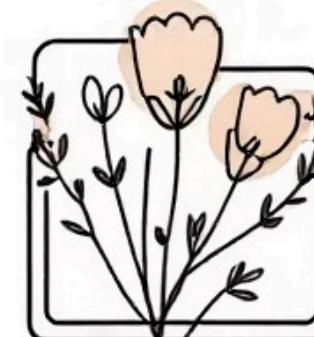
/src/assets

Arquivos importados via JavaScript e processados pelo bundler. Recebem hash no nome para cache busting. Ideal para: imagens usadas em componentes, fontes, ícones SVG.

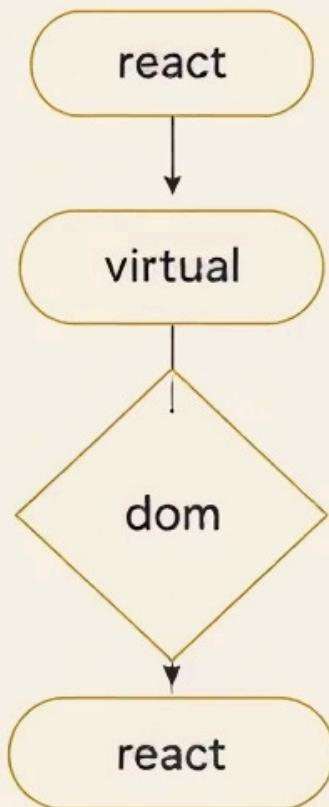
```
import logo from './assets/logo.svg'  
<img src={logo} />
```



- ❑ **Regra prática:** Se o arquivo precisa de cache busting ou otimização (compressão, lazy loading), use src/assets. Se é estático e referenciado por URL, use public.



O Funcionamento Interno: JSX → DOM Real



- 1 1. JSX Transform
Conversão de JSX em chamadas `React.createElement()` que retornam objetos JavaScript puros.
- 2 2. Virtual DOM
React mantém uma representação em memória da UI. É uma árvore de objetos leve e rápida de manipular.
- 3 3. Reconciliação
Algoritmo de diffing compara a nova árvore com a anterior, identificando mudanças mínimas necessárias.
- 4 4. Commit ao DOM
Apenas as diferenças são aplicadas ao DOM real, minimizando operações custosas e mantendo performance.

"O Virtual DOM não é sobre velocidade bruta, é sobre **desenvolver com confiança** sem micro-otimizações manuais do DOM."

React 19: O Futuro Chegou



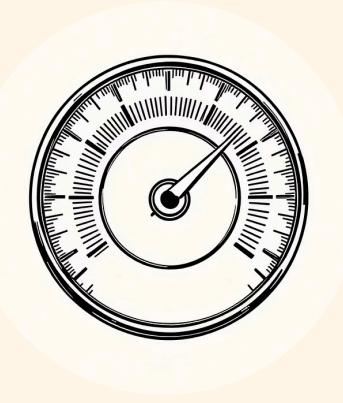
React Compiler

Otimização automática de performance. Elimina a necessidade de `useMemo`, `useCallback` e `memo`. O compilador analisa o código e aplica memoization inteligente onde necessário.



Actions e useActionState

Nova API para formulários com tratamento de estados pending/error integrado. Suporta operações assíncronas nativas e Server Actions no Next.js.



Performance Nativa

Renderização concorrente aprimorada, Suspense estável para data fetching, e otimizações de hidratação que reduzem o tempo de interatividade em 40%.

- ❑ **Breaking changes mínimos:** React 19 mantém compatibilidade com código existente, mas prepara o ecossistema para o futuro com React Server Components e arquiteturas híbridas.