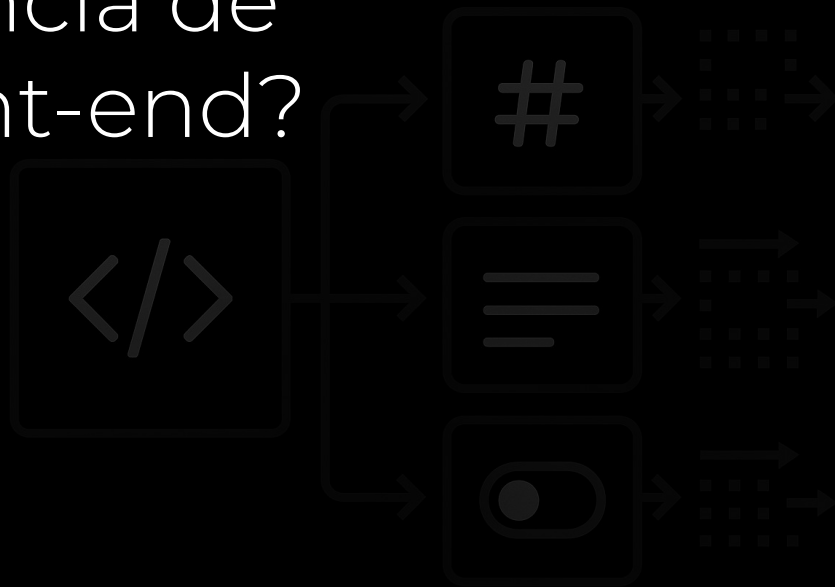


<web-components />

O fim da dependência de frameworks no front-end?

Cícero Feijó

professorfeijo.com.br



Sobre mim:



Cícero Feijó

Desenvolvedor há
mais de 20 anos.

Engenheiro de
Software
Especialista no
Sicredi.

Instrutor há mais
de 15 anos.

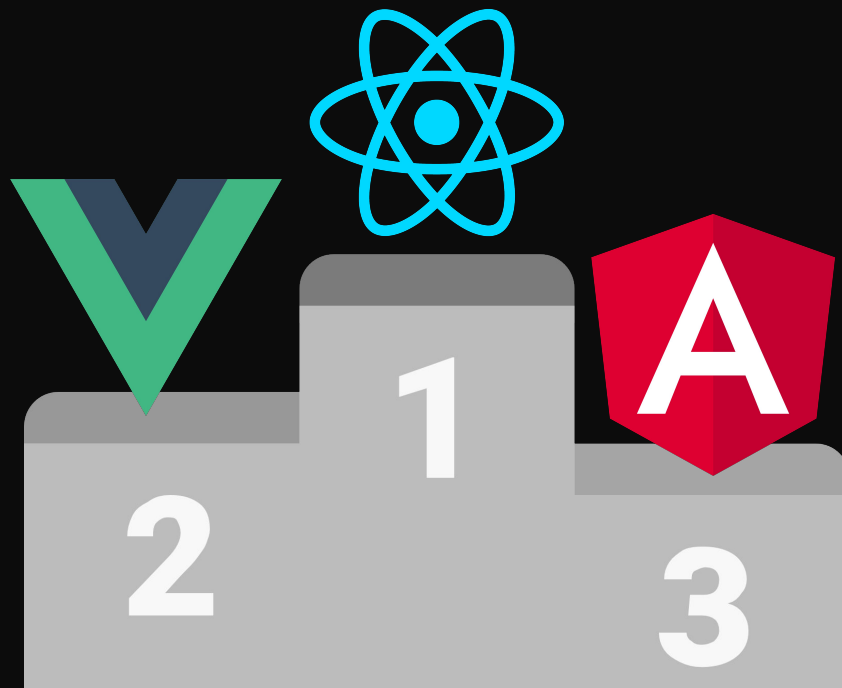
Ministro cursos
para iniciantes e
veteranos.

Tenho um canal
com quase 3 mil
inscritos.

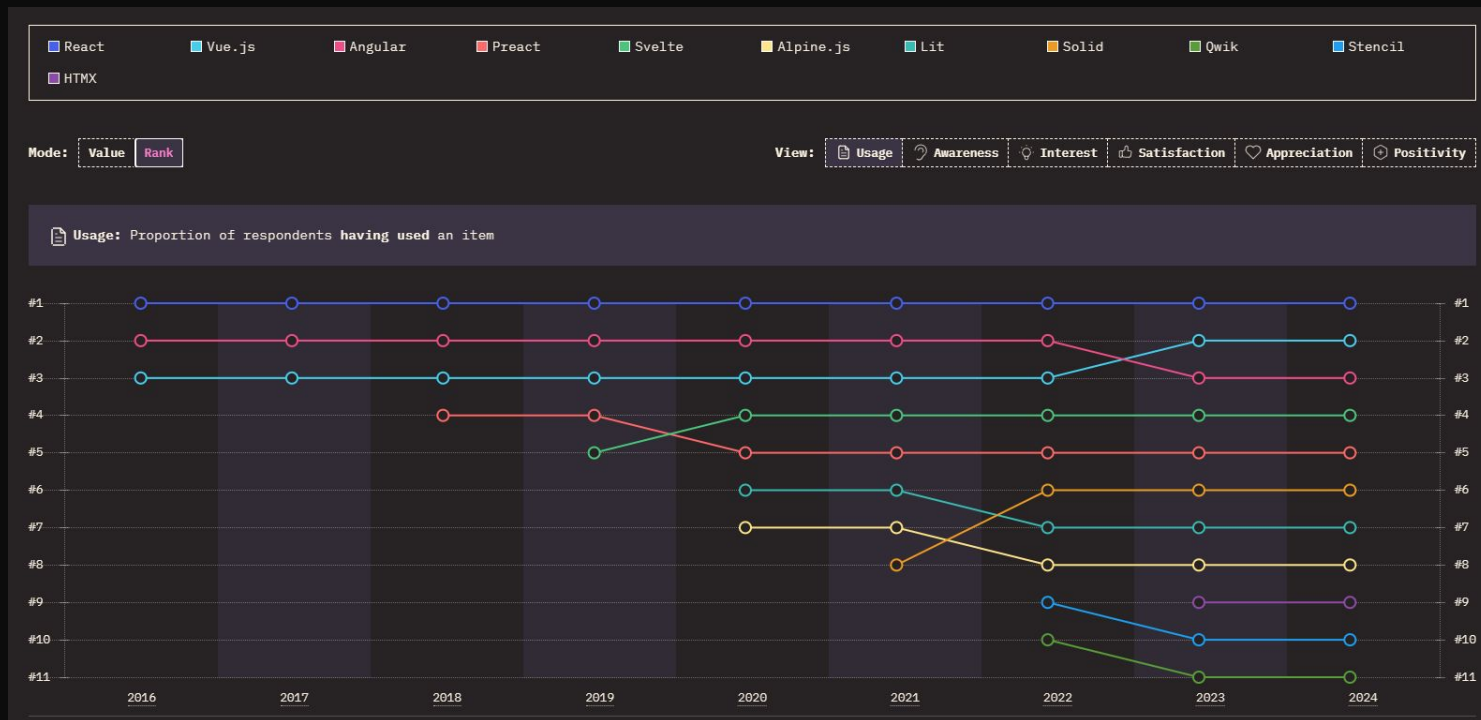


Atuei em
empresas como
Terra, HP, Azion
entre outras.

Visão atual de frameworks - adoção:

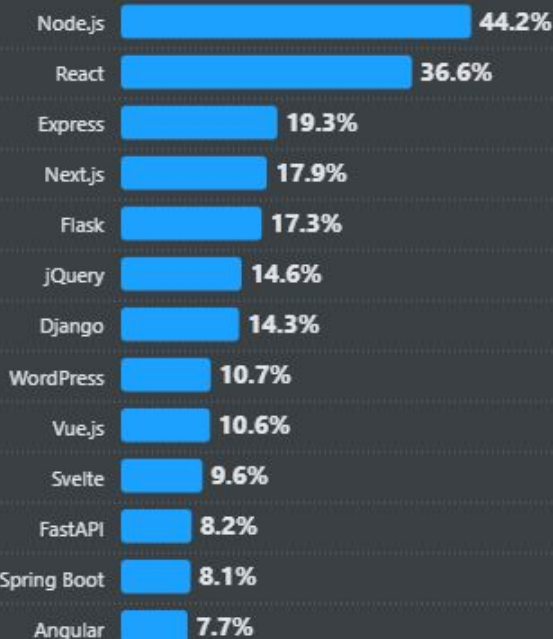


Visão atual de frameworks - adoção:

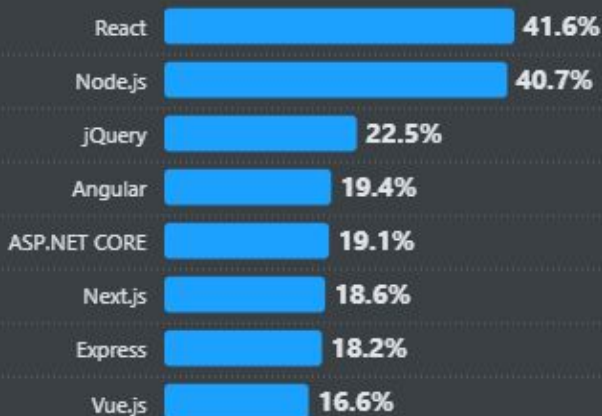


Visão atual de frameworks - aprendizado:

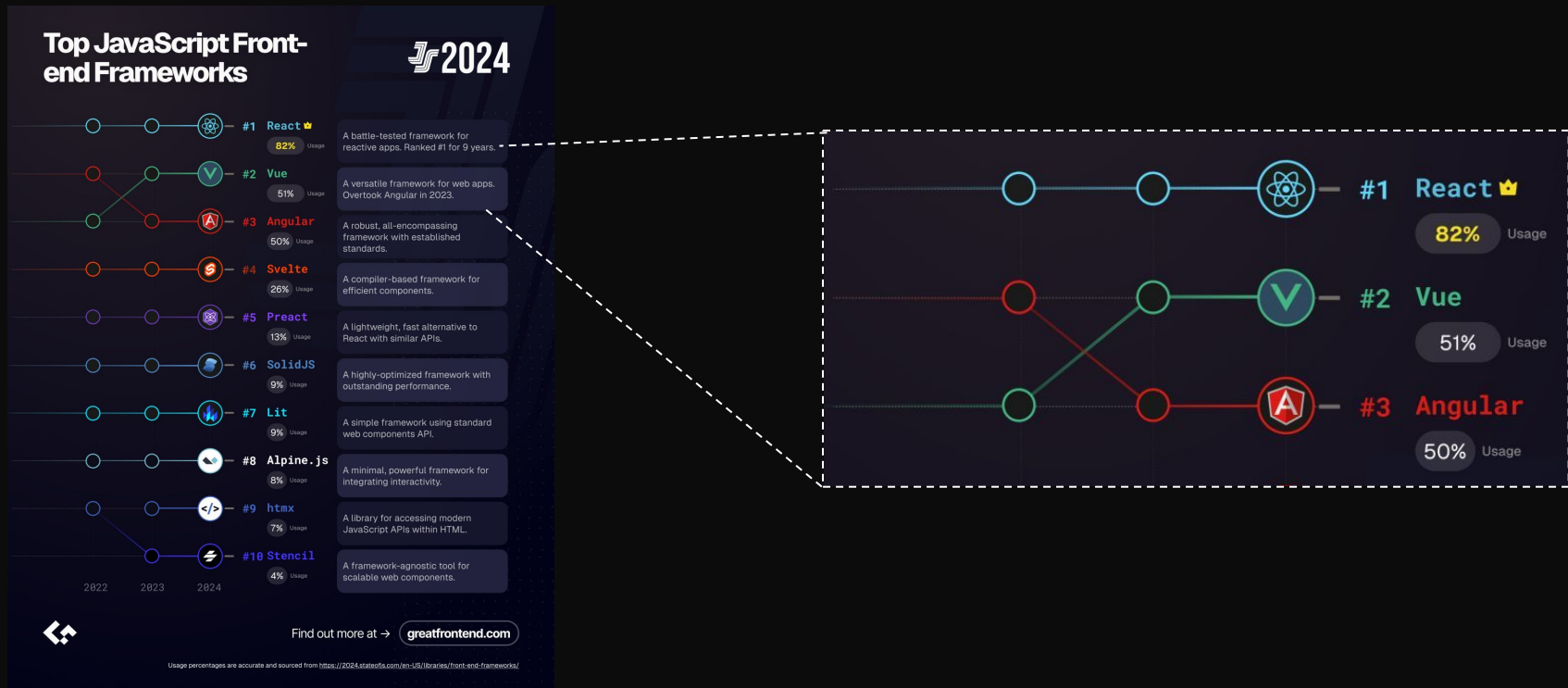
Learn to Code



Professional Developers



Visão atual de frameworks - mercado:



O problema: a dependência excessiva de frameworks

Obsolescência rápida:

Frameworks mudam rápido — o que é mainstream hoje pode ser legado amanhã.

Lock-in tecnológico:

Componentes React não funcionam em Angular e vice-versa.

Soluções reinventadas:

Cada framework implementa seu próprio sistema de componentes, reatividade, roteamento, etc.

Esforço Cognitivo:

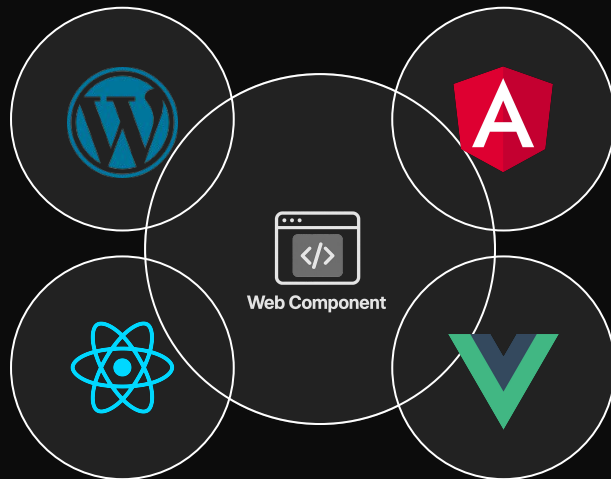
Curva de aprendizado e sobrecarga de stack para novos devs.

E se pudéssemos escrever componentes que funcionam sozinhos ou em conjunto com todos esses frameworks?

A Solução proposta: Web Components

Web Components são um conjunto de padrões da plataforma web que permitem criar elementos customizados reutilizáveis, encapsulados e com escopo próprio, usando tecnologias **nativas** do navegador.

Independem do contexto: um botão web component pode ser executado em uma Angular App, React App, Vue, Wordpress ou em um HTML puro.



Por que usar Web Components?

Vantagens:

- Pode ser usado sozinho ou com qualquer framework.
- Reutilização real em múltiplos projetos.
- Encapsulamento de estilo e comportamento.
- Suporte crescente em navegadores modernos.

Quando considerar usar:

- Design Systems.
- Bibliotecas de UI independentes.
- Integração entre múltiplos frameworks.
- Micro frontends.

Cuidados ao usar Web Components

Desvantagens:

- ⚠ Curva de aprendizado para o "bare metal".
 - Falta de reatividade nativa.
- ⚠ Dificuldades com gerenciamento de estado global.
- ⚠ Integração com frameworks pode ter fricção.
 - SEO e acessibilidade ainda exigem atenção manual.
- ⚠ Menor ecossistema e comunidade.
 - Performance em larga escala pode variar.

Cuidados ao usar Web Components

Curva de aprendizado para o "bare metal"

Criar Web Components sem bibliotecas como *Lit* ou *Stencil* pode ser **verboso** e **trabalhoso**.

Você precisa lidar com:

- manipulação de DOM nativo.
- métodos de ciclo de vida (*connectedCallback*, etc.).
- Shadow DOM e *attachShadow()*.

Exige conhecimento profundo da plataforma web e manipulação DOM, o que pode afastar devs acostumados com abstrações de frameworks e libs.



Cuidados ao usar Web Components

Dificuldades com gerenciamento de estado global

Componentes encapsulados funcionam bem isoladamente, mas compartilhar estado entre eles pode ser complexo.

É necessário:

- levantar eventos (*CustomEvent*).
- usar contextos externos (como *Redux*, *Signals*, *context API* de frameworks).
- ou criar soluções próprias de comunicação entre componentes.



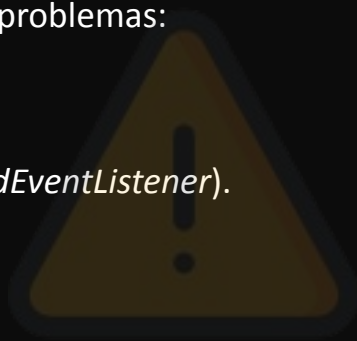
Cuidados ao usar Web Components

Integração com frameworks pode ter fricção

Apesar de interoperáveis, o uso de Web Components em frameworks como React pode ter problemas:

- React não propaga corretamente propriedades booleanas/personalizadas.
- Precisa de workarounds para lidar com eventos personalizados (*onCustomEvent* $\neq$ *addEventListener*).
- Angular exige configuração adicional para reconhecer elementos personalizados.

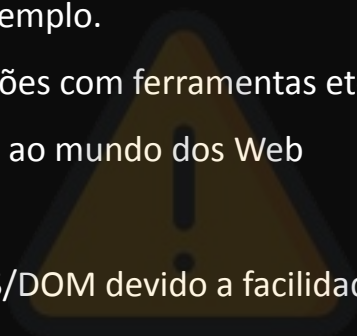
```
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
```



Cuidados ao usar Web Components

Menor ecossistema e comunidade

- A comunidade Web Components é menor comparada a React, Angular ou Vue por exemplo.
- Menor disponibilidade de componentes prontos, tutoriais, soluções prontas, integrações com ferramentas etc.
- Frameworks dominantes impõem padrões e práticas que não se aplicam diretamente ao mundo dos Web Components.
- O mercado não tem preparado muito desenvolvedores com know hall profundo de JS/DOM devido a facilidade dos frameworks.



Composição de Web Components

Custom Elements:

define novos elementos HTML.

Shadow DOM:

encapsula estilo e estrutura evitando os “leaks” de regras.

HTML Templates:

define estrutura que pode ser clonada e usada em runtime.

(opcional) ES Modules:

para importar/exportar componentes.

Básico:

```
class Button extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: "open" });
    shadow.innerHTML = `
      <style>
        button {
          background: #ff9900;
          color: white;
          border: none;
          padding: 10px;
          border-radius: 5px;
        }
      </style>
      <button><slot></slot></button>
    `;
  }
}

customElements.define("my-button", Button);
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Exemplo</title>
    <script src="button.js"></script>
  </head>
  <body>
    <my-button>button</my-button>
  </body>
</html>
```

Eventos:

Nativos do HTML:

podemos utilizar eventos como onClick, onBlur, etc. de forma transparente.

```
<my-button onClick="alert('Funciona')">botão</my-button>
```

Customizados:

podemos customizar eventos, entretanto eles não são atachados nativamente ao elemento, sendo necessário fazê-lo via JavaScript.

```
<my-button meu-evento="alert('Não Funciona')">botão</my-button>
```

Eventos customizados:

```
class Button extends HTMLElement {
  constructor() {
    ...
    const btn = shadow.querySelector("button");
    // Criando um evento custom
    btn.addEventListener("click", () => {
      const attr = this.getAttribute("meu-evento");
      // Executa o valor vindo do atributo
      if (attr) {
        // executa o código do atributo como JS
        new Function(attr)();
      }
      this.dispatchEvent(
        new CustomEvent("meu-evento", {
          bubbles: true, composed: true,
          detail: { info: "clicado" },
        })
      );
    });
  }
}

customElements.define("my-button", Button);
```

```
...
<my-button meu-evento="alert('Funciona')">poi</my-button>
<script>
  document
    .querySelector("my-button")
    .addEventListener("meu-evento", (e) => {
      alert("evento recebido!");
      console.log(e.detail); // { info: "clicado" }
    });
</script>
...
```

Exemplos de mercado:

Design Systems:

Shoelace – biblioteca completa de Web Components.

<https://shoelace.style>



- Biblioteca de Web Components moderna e framework-agnóstica.
- Compatível com React, Vue, Angular ou HTML puro.
- Oferece UI pronta: botões, modais, abas, tooltips e mais.
- Acessível, personalizável via CSS e com foco em usabilidade.
- Ideal para projetos modulares e reutilizáveis.

Exemplos de mercado:

Design Systems:

Adobe Spectrum – usou Web Components para padronizar UI em todos os produtos.

<https://spectrum.adobe.com>



- Implementação oficial do design system da Adobe usando Web Components.
- Segue as diretrizes visuais e de acessibilidade do Spectrum.
- Compatível com qualquer framework ou aplicação HTML.
- Componentes com temas (light, dark, etc.) e foco em design profissional.
- Excelente para interfaces refinadas e aplicações corporativas.

Exemplos de mercado:

Design Systems:

FAST by Microsoft – Web Components modernos com performance e extensibilidade.

<https://fast.design>



- Plataforma para criar Web Components leves, rápidos e altamente personalizáveis.
- Compatível com qualquer framework (React, Angular, Vue) ou HTML puro.
- Oferece um sistema de design dinâmico e extensível via tokens e design system provider.
- Permite criar seus próprios componentes ou usar os já prontos (FAST Components).
- Foco em desempenho, acessibilidade e escalabilidade em larga escala.

Exemplos de mercado:

Casos reais:

Ionic Framework – todos os componentes visuais são Web Components.

<https://ionicframework.com/>

Salesforce Lightning Web Components (LWC) – usa padrão de Web Components com extensões.

<https://developer.salesforce.com/developer-centers/lightning-web-components>

Ferramentas úteis

Lit – facilita a criação de Web Components reativos.

Stencil – gera Web Components com suporte a TypeScript, JSX e lazy loading.

WebComponents.dev – playground online para testar e compartilhar.

Stencil

É um compilador open-source criado pela **Ionic** para gerar Web Components reutilizáveis, performáticos e compatíveis com qualquer framework.

Características principais:

- Implementação baseada em JSX + TypeScript.
- Produz Web Components nativos com Shadow DOM, lazy loading e encapsulamento.
- Gera output compatível com React, Angular, Vue (via bindings).
- Ideal para Design Systems e bibliotecas de UI compartilháveis.
- Possui uma CLI para criação do scaffolding de componentes.

Stencil

```

v src
  v components
    v my-button
      > test
      # my-button.css
      my-button.tsx
  
```

```

import { Component, Host, h } from
 '@stencil/core';
  
```

```

@Component({
  tag: 'my-button',
  styleUrls: 'my-button.css',
  shadow: true,
})
  
```

```

export class MyButton {
  render() {
    return (
      <Host>
        <button>
          <slot></slot>
        </button>
      </Host>
    );
  }
}
  
```

my-button.tsx

```

:host {
  display: block;
  
```

```

button {
  background: #4f46e5;
  color: white;
  border: none;
  padding: 12px 20px;
  font-size: 1rem;
  font-weight: 500;
  border-radius: 10px;
  cursor: pointer;
  box-shadow: 0 4px 10px rgba(79, 70, 229, 0.3);
  
```

```

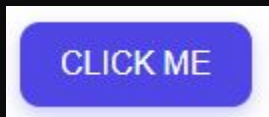
&:hover {
  background: #4338ca;
}
  
```

```

}
  
```

my-button.css

Stencil



```

<my-button class="hydrated">
  #shadow-root (open)
    <style>
      :host {
        display: block;
        button {
          background: #4f46e5;
          color: white;
          border: none;
          padding: 12px 20px;
          font-weight: 500;
          border-radius: 10px;
          cursor: pointer;
          box-shadow: 0 4px 10px rgba(79, 70, 229, 0.3);
          &:hover {
            background: #4338ca;
          }
        }
      }
    </style>
    <button> == $0
      <slot>
        ↪ #text reveal
      </slot>
    </button>
    "CLICK ME"
  </my-button>

```

Stencil exportando para React

@stencil/react-output-target

Comportamento	Web Component via Stencil	React JSX Clássico
Atributos primitivos (title, id)	✓ Sim	✓ Sim
Booleanos (disabled)	⚠ Pode falhar sem wrapper	✓ Sim
Eventos nativos (click)	✓ Sim	✓ Sim
Eventos personalizados	⚠ React não ouve direto	✓ nativo via props
Tipagem em TS	✓ Com wrapper	✓ Completa
ref com métodos internos	⚠ DOM direto	✓ via instância React
Reatividade e estado interno	⚠ Manual	✓ Declarativa

Stencil exportando para outros modelos:

React – @stencil/react-output-target

Vue – @stencil/vue-output-target

Angular – uso do web component nativo + CUSTOM_ELEMENTS_SCHEMA

Visão de futuro

- O futuro é interoperabilidade e componentes duráveis.
- Web Components reduzem acoplamento e aumentam a vida útil do seu código.
- "Talvez não eliminem frameworks, mas nos libertam da obrigação de tê-los."

Mas o porém:

- A DEVX de componentes nativos com as features de frameworks ainda estão muito à frente.
- O know hall de times que já conhecem Angular, React e Vue é muito maior que web-component.
- Sempre devemos pesar o que queremos: agilidade, qualidade, esforço cognitivo do time, etc.

Obrigado!

me encontre aqui!



professorfeijo.com.br



source code dos slides



github.com/csfeijo/web-components-tdc