

# Web Components

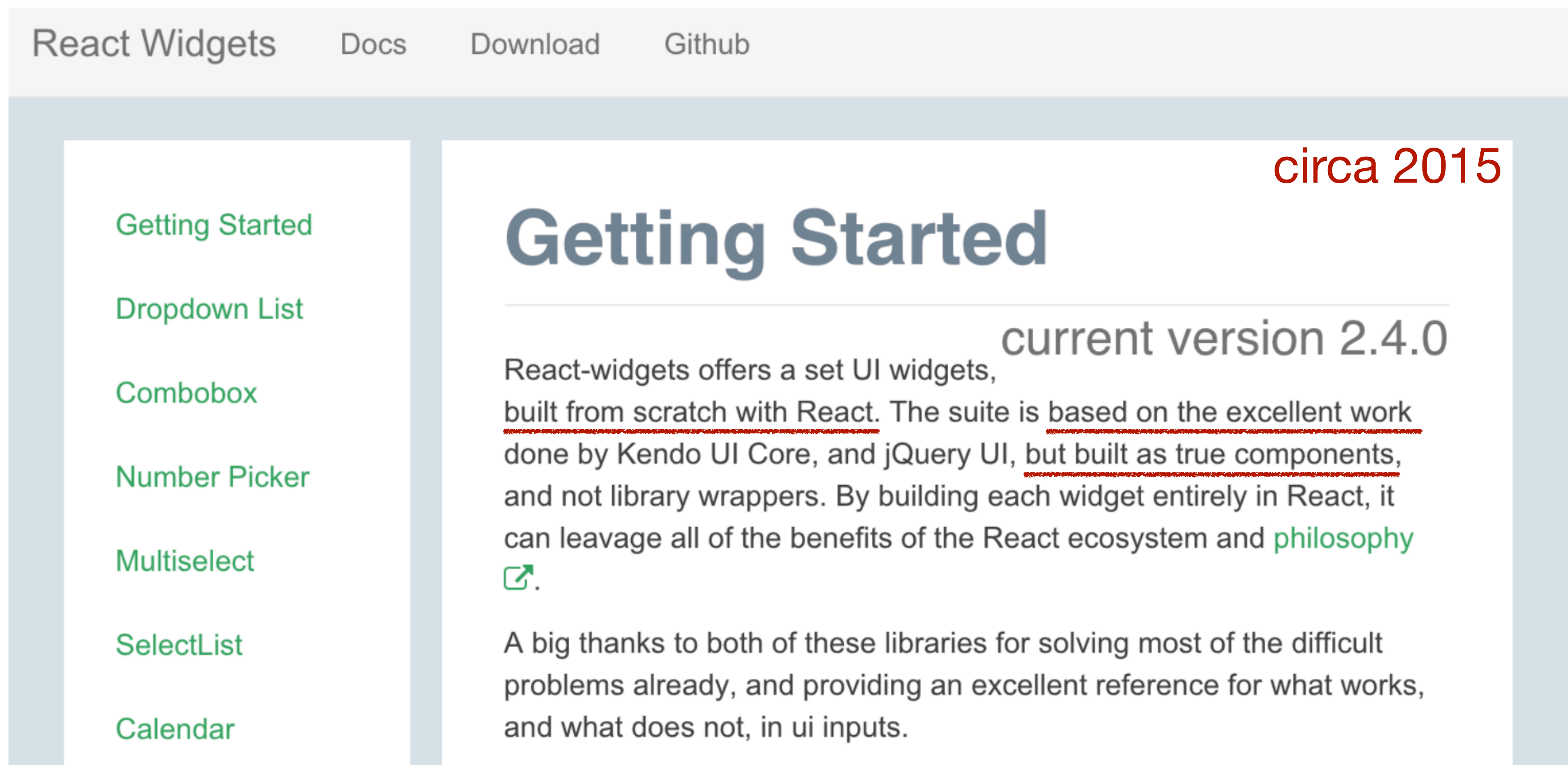
**Interface Pessoa-Máquina - 25/26 - LEI / UM**

Hugo Pacheco  
[hpacheco@di.uminho.pt](mailto:hpacheco@di.uminho.pt)

# Components na Web

- Um termo mesmo muito overloaded em web development
  - Qualquer template ou pedaço reutilizável de HTML, CSS ou JS
  - Abstrações que frameworks utilizam para modularização de código
    - ! Framework-specific
      - Um dos fortes argumentos é reutilização, mas apenas interna...
    - ! Frameworks e as abstrações que utilizam evoluem
      - Código tem que evoluir com elas...
    - ! Frameworks aparecem e desaparecem, sobem e descem de popularidade
      - Não faz sentido reinventar a roda a cada nova framework...

# Components na Web
































# Web Components

- Um novo Web Standard
  - “A suite of different technologies allowing you to create **reusable HTML custom elements**”
  - Na forma de extensões aos standards DOM + HTML + CSS
- + **Reactivity:** Comportam-se como elementos HTML normais, podem ter atributos e estes podem ser modificados para alterar o comportamento
- + **Encapsulation:** Implementação escondida do resto da página
- + **Modularity:** Desenvolvidos e testados independentemente
- + **Reusability:** Possível utilizar o mesmo componente em vários contextos

# Web Components

- Uma abordagem **suportada** por **todos** os principais browsers

| Browser support   |  CHROME   |  OPERA    |  SAFARI   |  FIREFOX  |  EDGE     |
|---|--|--|--|--|--|
|  HTML TEMPLATES   |  STABLE  |  STABLE  |  STABLE  |  STABLE  |  STABLE  |
|  CUSTOM ELEMENTS |  STABLE |  STABLE |  STABLE |  STABLE |  STABLE |
|  SHADOW DOM      |  STABLE |  STABLE |  STABLE |  STABLE |  STABLE |
|  ES MODULES      |  STABLE |  STABLE |  STABLE |  STABLE |  STABLE |

# Web Components

- Uma abordagem **framework-agnostic**
  - Custom HTML element tratado como qualquer `<div>` por qualquer framework
- ! Na prática há alguns desafios
  - ! Nas convenções que a framework utiliza para comunicar com o custom element (argumentos, eventos)

Custom Elements Everywhere

Making sure frameworks and custom elements can be BFFs 

# Web Components: Recursos

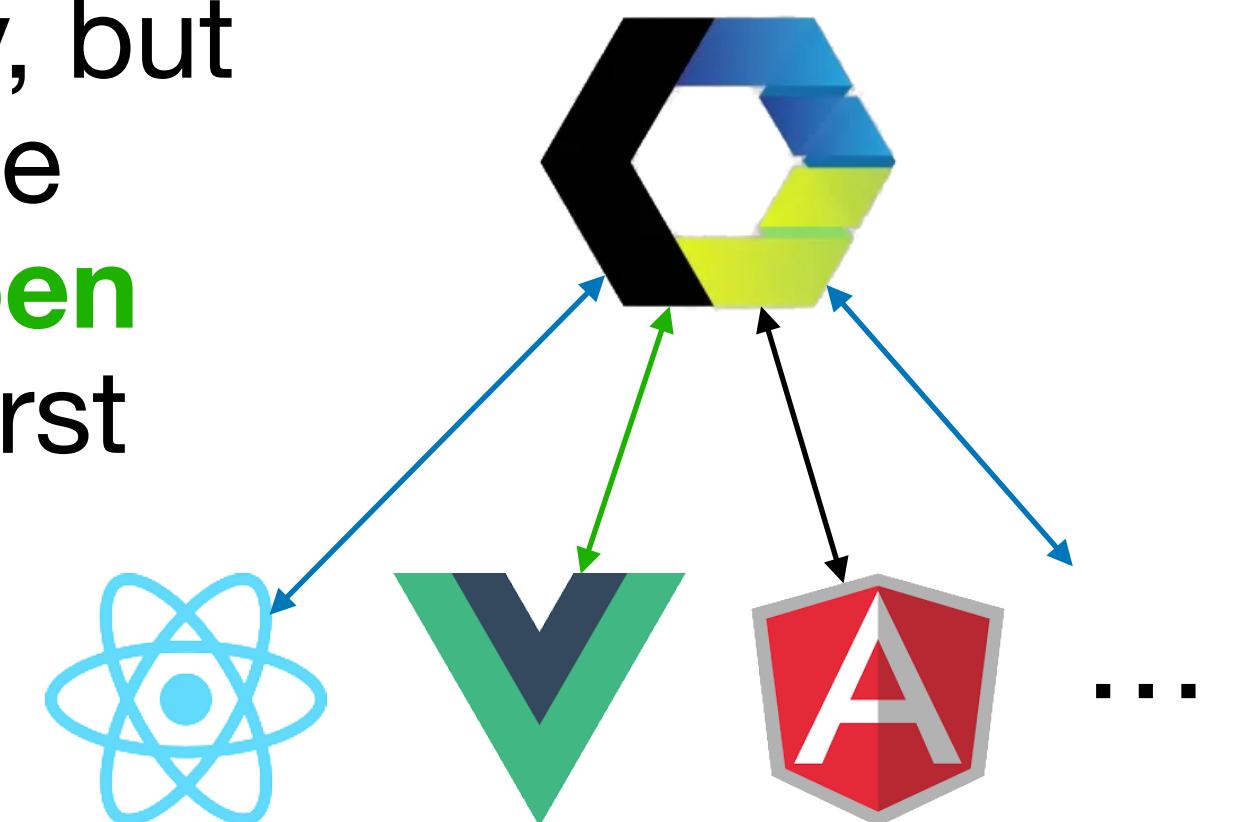
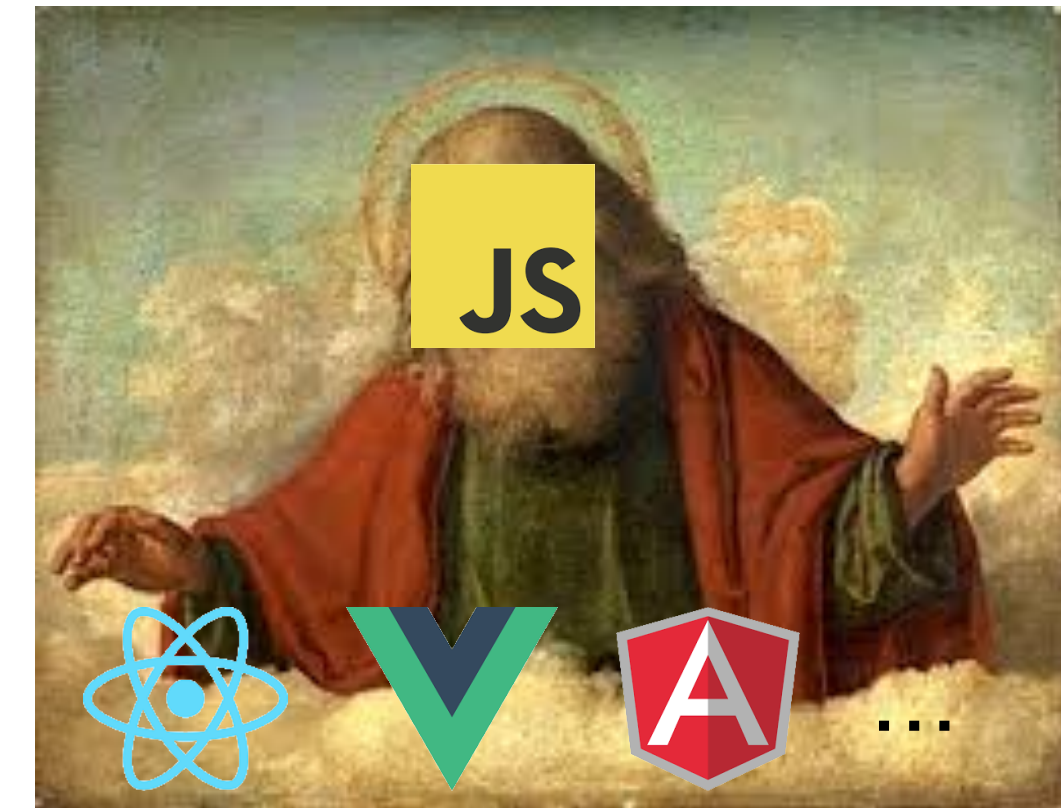
- Vários repositórios de web components prontos a usar:
  - <https://www.webcomponents.org/>
  - <https://component.gallery/>
  - <https://genericcomponents.netlify.app/>
  - <https://github.com/mdn/web-components-examples>
  - <https://github.com/davatron5000/awesome-standalones>
  - [https://github.com/scottaohara/accessible\\_components](https://github.com/scottaohara/accessible_components)
  - <https://shoelace.style/>
  - <https://kickstand-ui.com/>
  - ...

# Web Components: Recursos

- Várias bibliotecas para simplificar a criação de web components:
  - <https://lit.dev/> by Google
  - <https://fast.design/> by Microsoft
  - <https://stenciljs.com/>
  - <https://hybrids.js.org/>
  - <https://www.dataformsjs.com/>
  - <https://slim.js.org/>
  - <https://github.com/devpunks/snuggsi>
  - ...

# Web Components: Adopção

- Browsers: proposta pela Google (2011) → suporte generalizado (2020)
- Frameworks:
  - “Web Components have **struggled to gain agreement and adoption at a time when JavaScript frameworks have grown in stature and capability**. If you’re coming from React, Vue.js, or Angular, Web Components can look complex and clunky”
  - Uma **transição** em **andamento**
    - “A decade ago, few would have tackled a site without jQuery, but **browser vendors took the excellent parts** and added native alternatives (such as `querySelector`). The **same will happen for JavaScript frameworks**, and Web Components is that first tentative step.”



# Web Components: Adopção

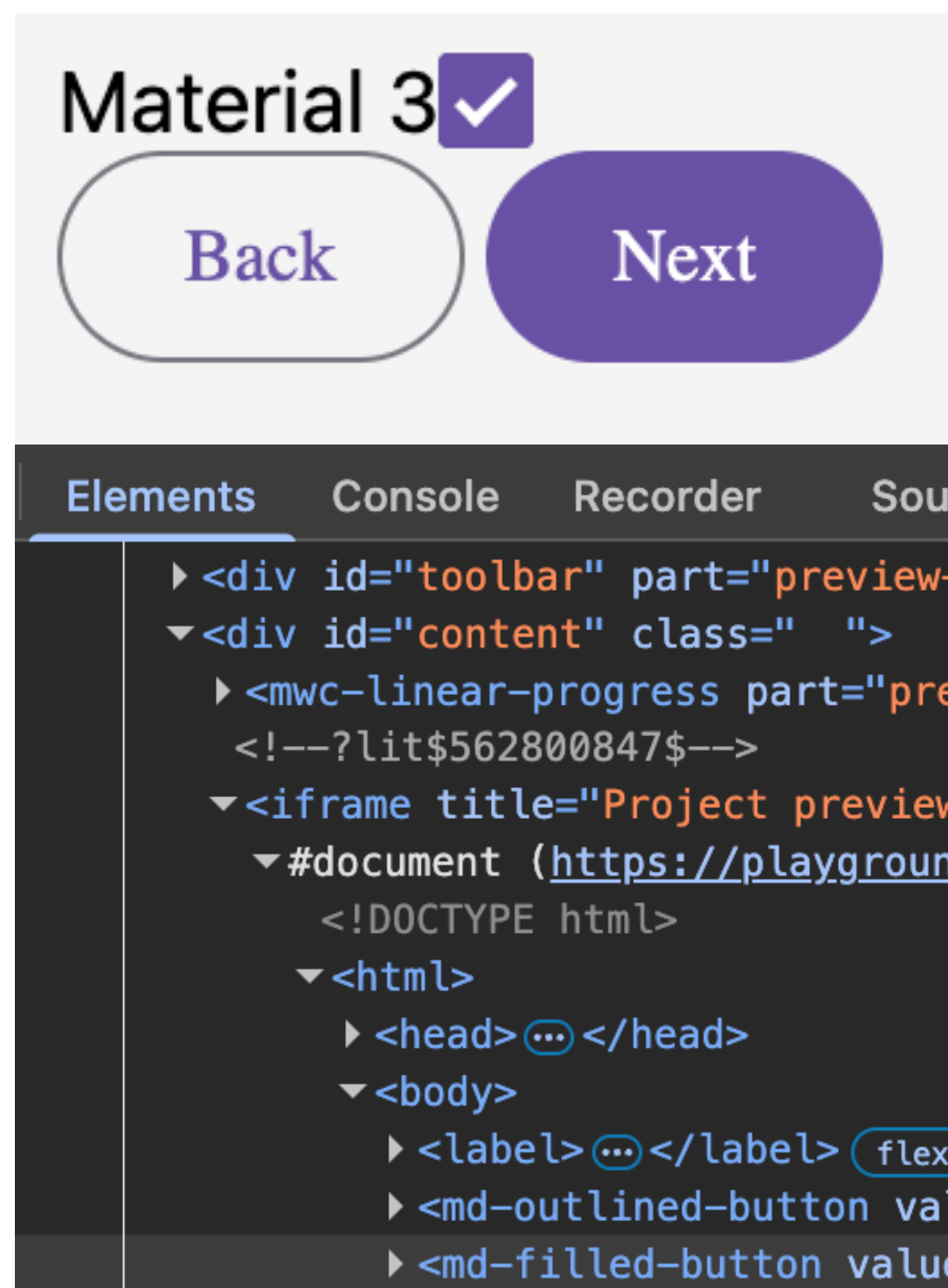
- Design Systems
  - Standards de design dentro de uma empresa ou gama de produtos
  - Para **uso em múltiplos ambientes de desenvolvimento** (browsers, frameworks, etc) e por várias equipas
- “This might have been the deciding factor for **companies like Adobe, Microsoft, and Google to use Web Components in their design systems**. Knowing that they can deliver a consistent experience in all these different contexts is incredibly valuable”



# Web Components: Design Systems

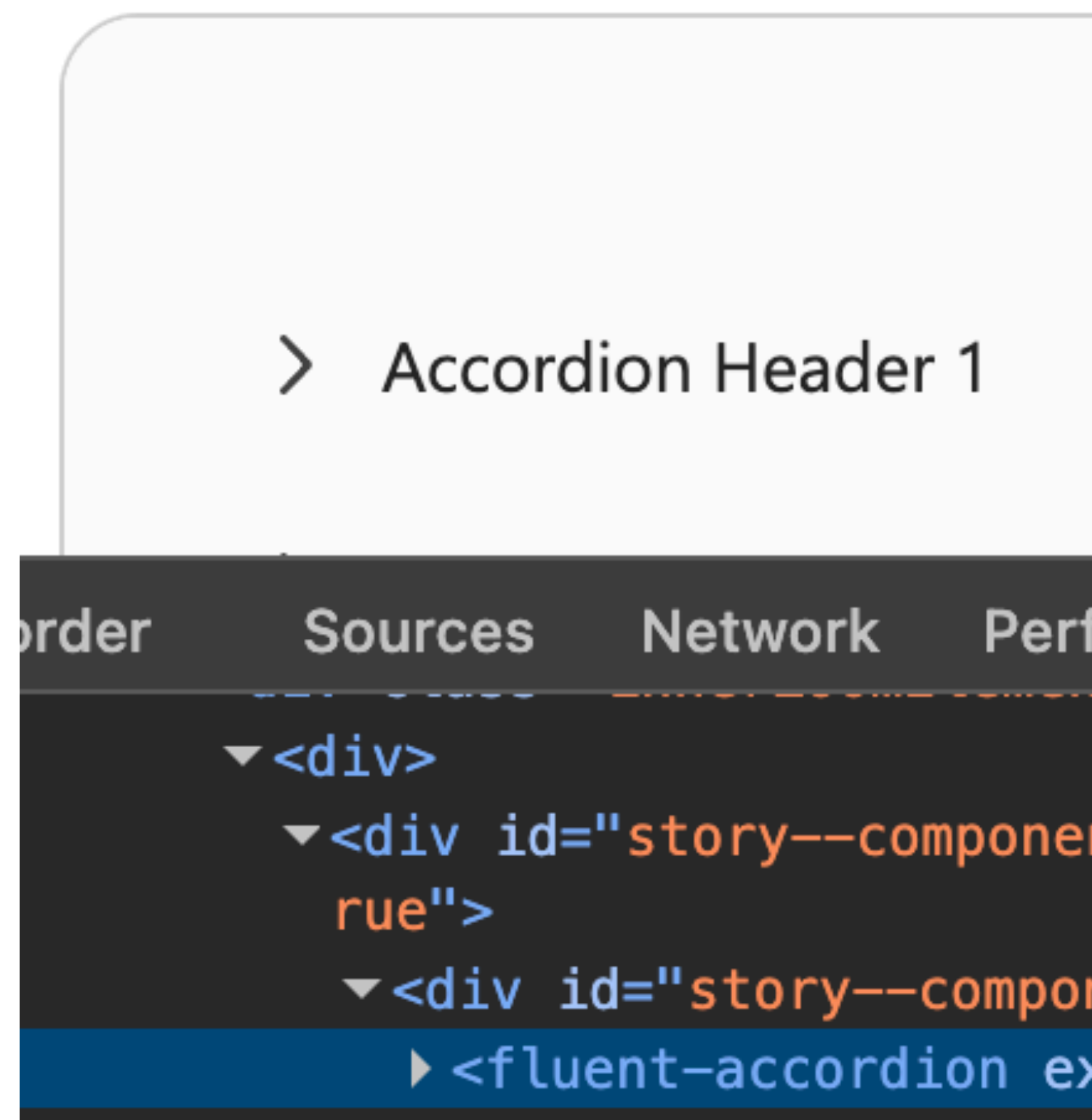
## Google Material Design

<https://m3.material.io/develop/web>



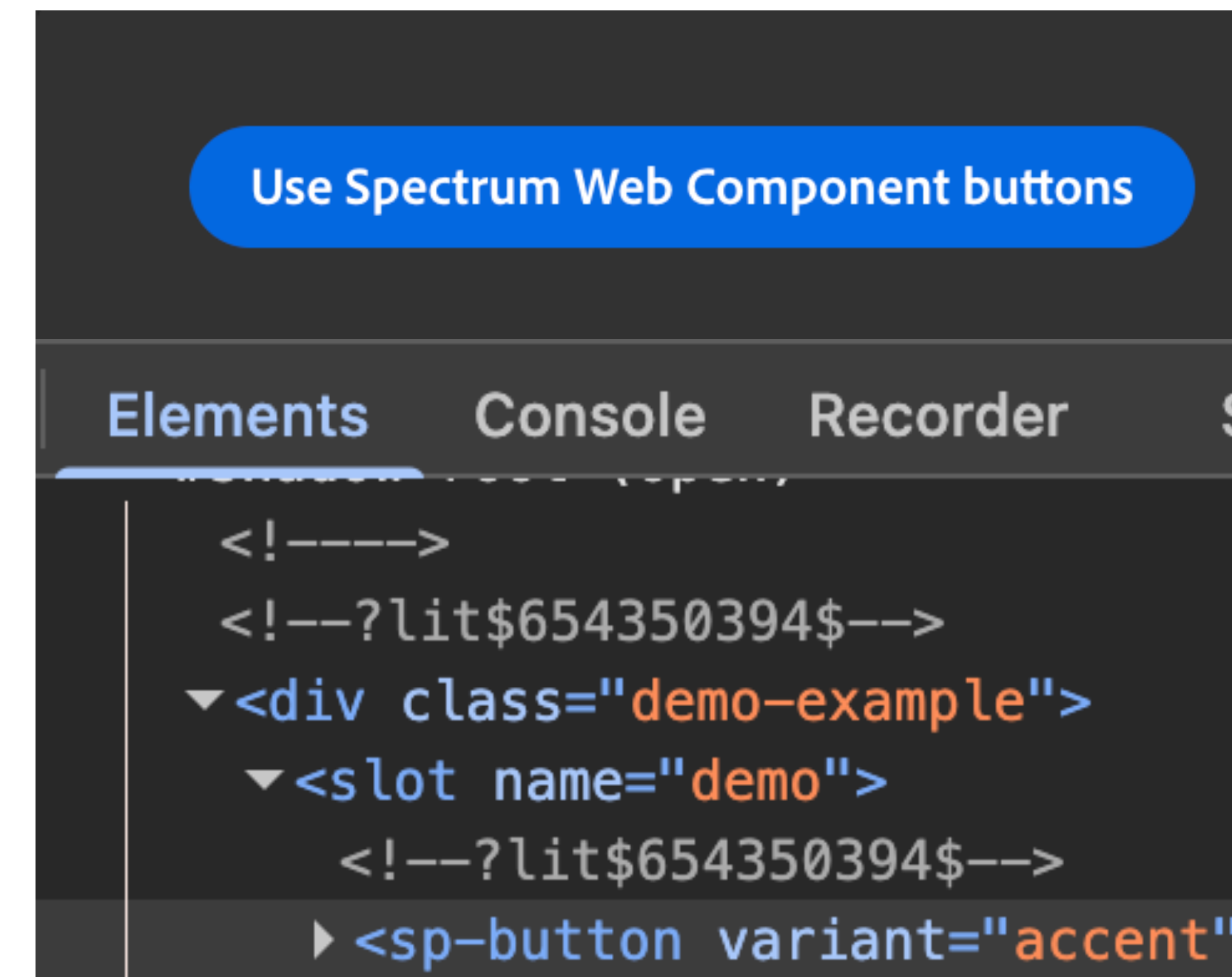
## Microsoft Fluent Design

<https://fluent2.microsoft.design/get-started/develop>



## Adobe Spectrum

<https://opensource.adobe.com/spectrum-web-components/>



# Web Components: Tecnologia

- Uma sinergia entre 4 tecnologias distintas
  - **Custom Elements**
    - Definir e registrar novos elementos; única tecnologia estritamente necessária
  - Shadow DOM
    - Abstrair o DOM do elemento para o resto da página
  - HTML Templates
    - Escrever excertos de HTML parametrizável
  - CSS Selectors
    - Elemento combina estilos da página com estilos internos

# Web Components: Custom Elements

- Basta criar
  - Uma classe JS

```
class MyElement extends HTMLElement {  
  constructor() {  
    super();  
  }  
}
```

- Uma tag **HTML** (convenção especial prefixo-nome para evitar clash)

```
customElements.define("my-element", MyElement);
```

# Exemplo (Contador)

- Criar um contador que apresenta número de clicks

💡 Encapsulamento de estado com atributos/métodos privados

💡 Podemos registrar eventos dentro do elemento

💡 Podemos alterar propriedades do elemento com **CSS**

```
class ClickCounter extends HTMLElement {
  #clicks = 0;
  constructor() {
    super();
    this.#render();
    this.addEventListener('click', evt => {
      this.#clicks++;
      this.#render();
    });
  }
  #render () {
    this.innerHTML = `Clicked ${this.#clicks}x`;
  }
}
customElements.define('click-counter', ClickCounter);
```

# Exemplo (Contador)

- Definir um atributo `clicks`

```
<click-counter id="counter"  
clicks="3"></click-counter>
```

- 💡 Inicialização com atributo no **HTML**

- 💡 Atributo modificável via **JS**, e.g., `counter.clicks=5`

- 💡 *Attribute reflection:* sincronização **JS ↔ HTML**

```
class ClickCounter extends HTMLElement {
```

```
...
```

```
  get clicks() { return this.#clicks; }  
  set clicks(v) { this.#clicks = v;  
    this.setAttribute("clicks", v); }
```

```
  static get observedAttributes() { return ["clicks"] }
```

```
  attributeChangedCallback(name, oldValue, newValue) {  
    if (name === "clicks") this.#clicks = newValue;  
    this.#render();
```

```
}
```

# Exemplo (Contador)

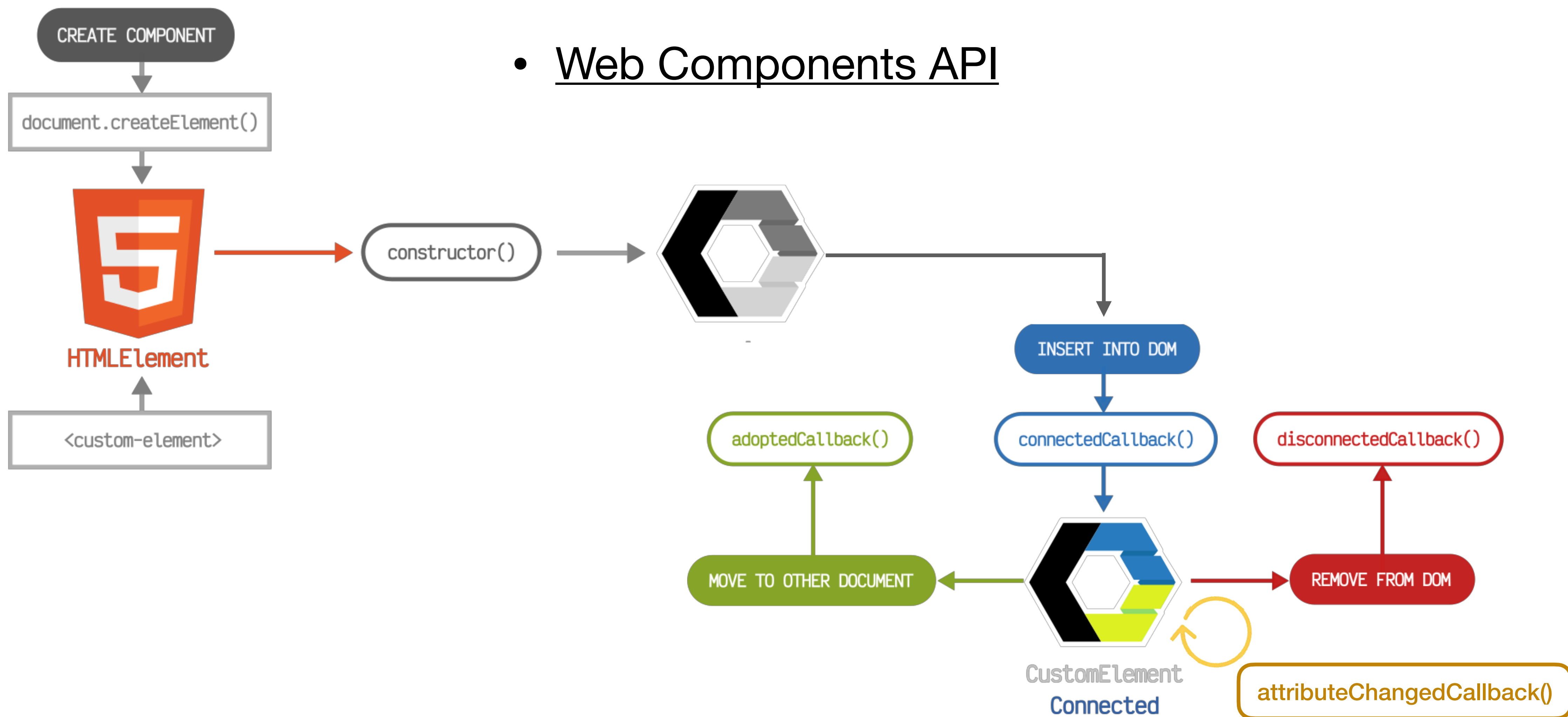
- Refletir informação do parent element, neste caso o **id**
- 💡 Elemento não é “conectado” ao DOM no momento em que é criado
- 💡 Podemos reagir quando elemento é:
  - “conectado” ao DOM
  - “desconectado” do DOM
  - “adoptado” por outro DOM (página, frame)

```
<div id="container">  
  <click-counter></click-counter>  
</div>
```

```
class ClickCounter extends HTMLElement {  
  #parent = "";  
  
  ...  
  
  connectedCallback() {  
    this.#parent = this.parentElement.id;  
    this.#render();  
  }  
}
```

# Ciclo de vida de um Custom Element

- Web Components API



# Web Components: Shadow DOM

- **HTML** e **CSS** são abertos por defeito
  - Não há isolamento na página, tudo está definido num scope global
  - Excepção feita para `<i f r a m e >`, que encapsula completamente uma sub-página
- ! Problema:
  - **JS** e **CSS** da página pode interferir com DOM e visualização de um custom element
  - **JS** e **CSS** de um custom element podem interferir com DOM e visualização da página
- 💡 Solução = custom element tem dois DOMs
  - **Light DOM:** o DOM por defeito construído a partir do **HTML**, não encapsulado
  - **Shadow DOM:** é renderizado pelo browser mais invisível para o DOM da página
- Ao contrário de `<i f r a m e >`, custom element oferece uma API para interação com a página

# Exemplo (Shadow)

- Criar um Shadow DOM
  - 💡 Light DOM e Shadow DOM coexistem
  - 💡 Se existir Shadow DOM, Light DOM não é visualizado
  - 💡 Se Shadow DOM estiver “aberto”, é visualizado
  - 💡 `<slot>` permite acrescentar Light DOM ao Shadow DOM

```
<shadow-element id="shadow">  
  <p>This is Light DOM</p>  
</shadow-element>
```

```
class ShadowElement extends HTMLElement {  
  constructor() {  
    super();  
    this.attachShadow({ mode: 'open' });  
    this.shadowRoot.innerHTML = `<p>This  
is Shadow DOM</p><slot></slot>`;  
  }  
}
```

open/closed

```
customElements.define('shadow-element',  
ShadowElement);
```

# Web Components: CSS Selectors

- 💡 Shadow DOM também ajuda a encapsular estilos CSS

```
this.shadowRoot.innerHTML = `<style> ... </style>`;
```

- Shadow styles
  - Não visíveis da página
  - Não se podem referir a elementos da página
  - Pseudo-class `:host` refere-se ao próprio custom element, apelidado de “shadow root”
    - `:host(selector)` quando host satisfaz `selector`
- ! Propriedades CSS “herdadas” atravessam fronteiras do Shadow DOM

# Exemplo (Contador + Shadow CSS)

- Contador de clicks com cor variável

```
class ClickCounter extends HTMLElement {
  #span;
  ...
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    const styles = document.createElement('style');
    styles.textContent = `
:host {
  ...
  background: hsl(var(--hue), var(--saturation), var(--lightness));
}`;
    this.shadowRoot.appendChild(styles);
    this.#span = document.createElement('span');
    this.shadowRoot.appendChild(this.#span);
    this.#render();
    this.addEventListener('click', evt => { this.#clicks++; this.#updateHue();
this.#render();});
  }
}
```

Shadow style

Atualiza --vue

```
body {
  --hue: 0;
  --saturation: 70%;
  --lightness: 50%;
}
```

Definido na página; herança permite passar variáveis CSS ao custom element

# Web Components: HTML Templates

💡 Já vimos JavaScript Template Literals

```
`Clicked ${this.#clicks}x`
```

- HTML Template (elemento `<template>`)
  - Código **HTML** que não é renderizados pelo browser
  - Elementos substituíveis com keyword `slot`

```
<template id="hello-world">  
  <slot name="msgtext"></slot>  
</template>
```

💡 Pode ser instanciado por **HTML**

```
<h1 slot="msgtext">Hello Default!</h1>
```

💡 Pode ser instanciado por **JS**

# Exemplo (Contador + HTML Templates)

- Contador de clicks com cor variável

```
class ClickCounter extends HTMLElement {  
  ...  
  constructor() {  
    —  
    const tpl = template.content.cloneNode(true);  
    this.shadowRoot.append(tpl);  
    ...  
  }  
  #render() {  
    const slot = this.shadowRoot.querySelectorAll('slot')[0];  
    slot.textContent = `${this.#clicks}`;  
  }  
}
```

```
<template id="template">  
  <style>  
    ...  
  </style>  
  <span>Clicked <slot></slot>x</span>  
</template>
```

HTML  
Template

Instancia template

Substitui `<slot>`

# Web Components: Tradeoffs

- Assente nas tecnologias tradicionais (HTML + CSS + JS)
- Formas de lifecycle management e composição mais restritas do que usando frameworks
- Sem Virtual DOM: JS tem que gerir manualmente updates ao DOM
- + Garantem consistência em diversos ambientes
- + Possível converter SPAs (de qualquer framework) para Web Components
- + Várias bibliotecas e frameworks dedicadas a construir Web Components



# Web Components

- **Web Components** instanciam/implementam **Concepts**
  - Partilham os mesmos princípios: modularidade, independência, reutilização
  - Refinam concepts ao definirem UI concretas
- Diferenças principais (devido ao diferente nível de abstração)
  - ↗ **Granularidade:** components mais finos (e.g. vários botões para emojis vs reaction concept)
  - ↗ **Lifecycle management:** concepts assumem-se universais, não são criados ou removidos
  - ↗ **Sincronização:** sincronização de concepts pode ser simulada com troca de eventos entre components; para além de alterar estado, side-effects na UI

# Exemplo (User + Session)

- Uma combinação familiar de **concepts**

## **concept** User

**purpose** authenticate users

### **principle**

after a user registers with a username and password,  
they can authenticate as that user by providing a matching  
username and password:

register (n, p, u); authenticate (n, p, u') {u' = u}

### **state**

registered: **set** User

username, password: registered -> **one** String

### **actions**

register (n, p: String, out u: User)

authenticate (n, p: String, out u: User)

## **concept** Session [User]

**purpose** authenticate user for extended period

### **principle**

after a session starts (and before it ends),  
the getUser action returns the user identified at the start:  
start (u, s); getUser (s, u') {u' = u}

### **state**

active: **set** Session

user: active -> **one** User

### **actions**

start (u: User, **out** s: Session)

getUser (s: Session, **out** u: User)

end (s: Session)

# Exemplo (User + Session)

- Podemos implementar cada **concept** como um **web component**
- Cada component implementa também uma UI (com Material 3 widgets)
- Sincronização feita pelo DOM, alterna entre UIs

