Web Components API

(this is for real!)

Belén Albeza
@ladybenko

A tale as old as time

From documents to apps

aDIVtox: Otra chorrada by BenKo'99

- Home
- Noticias
- Download
- Kursillo de DIV
- Links
- Libro de visitas
- E-mail

Página optimizada para 800x600 mínimo 256 kolores

Home

¡Salu2 a todos! En primer lugar kiero pedir diskulpas por la tardanza de aktualización de esta web. Pero komo kompensación tenéis una web en mi opinión mucho mejor que la anterior. ¿Novedades? Ahora tenéis el kurzillo de DIV komprimido (60K), una sección más (FreeDiv), las noticias más freskas ke se kuecen en el IRC y una kompleta remodelación de la página. ¿Mola el nuevo look? ;-)

Han pasado 0274 todos estos aDIVtox

Última aktualización 05/04/99. WEBMASTER: BenKo ' 99 | benko@olemail.co

1999

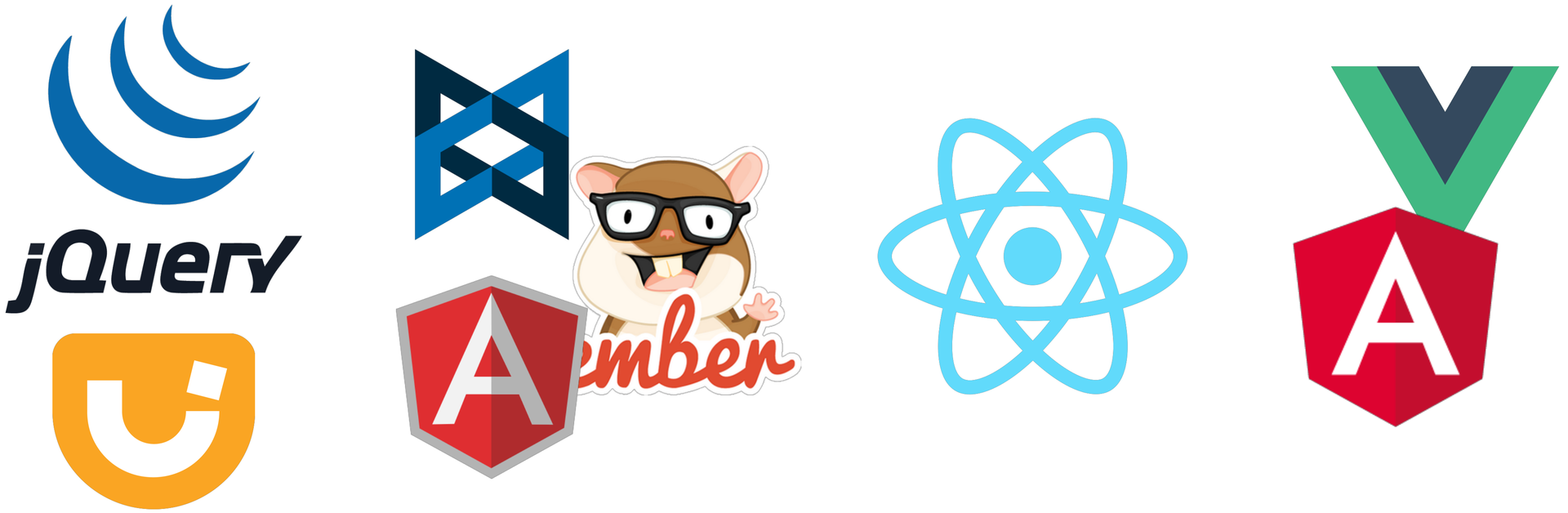
Firefox DevTools

2019

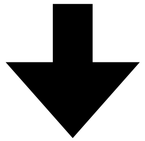

It’s 2019
We can’t style a dropdown

A common need

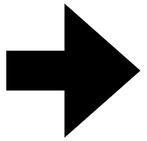
Reusable UI widgets



```
<User username="desatranques">
```



```
<span class="user">
  <img
   class="user__avatar"
   src="desatranques.png">
  Desatranques Jaén
</span>
```



Desatranques Jaén®

Web Components API

- A standardised API to make re-usable UI components
- **Web Components API =**
  Custom Elements + Shadow DOM +
  <template> + <slot>
- Since it's implemented by the browser, it can do things frameworks can't 😱

Shadow DOM v1 - WD

Usage

Global 71.17

Method of establishing and maintaining functional boundaries between DOM trees and how these trees interact with each other within a document, thus enabling better functional encapsulation within the DOM & CSS.

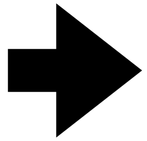
Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mobile *	Chrome for Android	Firefox for Android
		2-57										
		2 58										
		3 59-62	4-52	3.1-9.1	10-39	3.2-9.3						
6-10	12-17	63	53-70	1 10-11.1	40-56	1 10-11.4		2.1-4.4.4	7	12-12.1		
11	18	64	71	1 12	57	1 12.1	all	67	10	46	70	63
		65-66	72-74	1 TP								

Custom Elements

- Our own HTML tags! 🎉
- We can extend from HTMLElement or other subclass
- https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_custom_elements

```
customElements.define(
  'x-user',
  class extends HTMLElement {
    constructor() {
      super();
      // ...
    }
  }
);
```

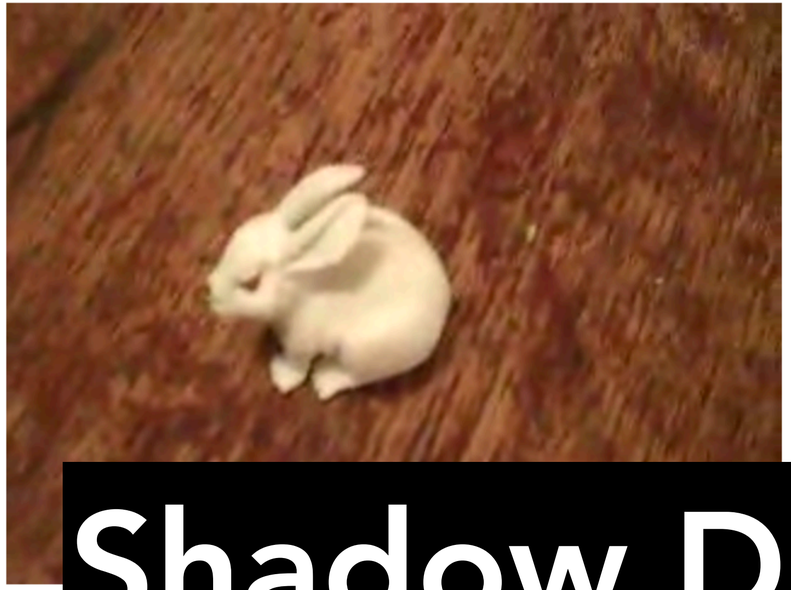


<x-user>

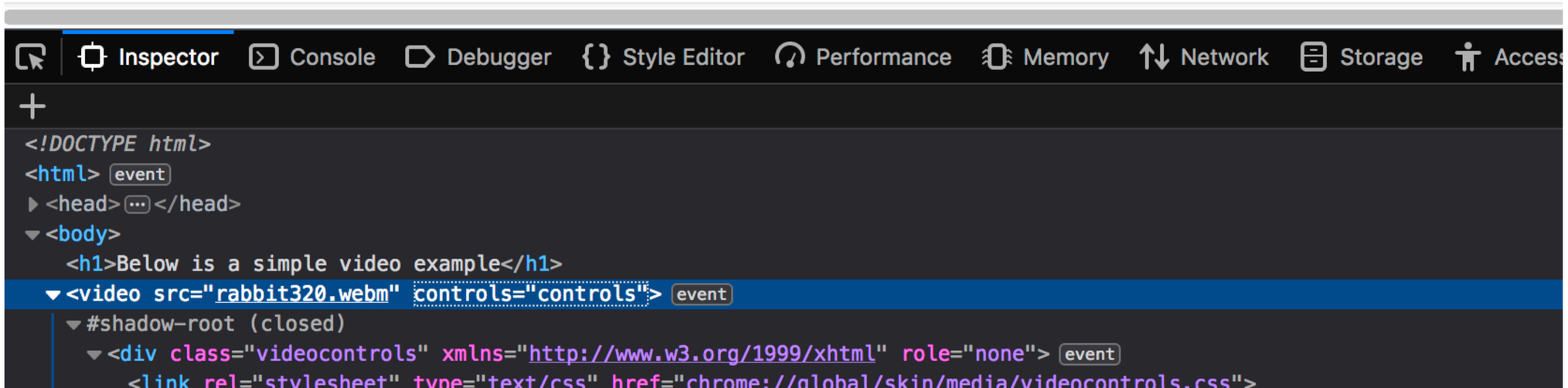
Shadow DOM

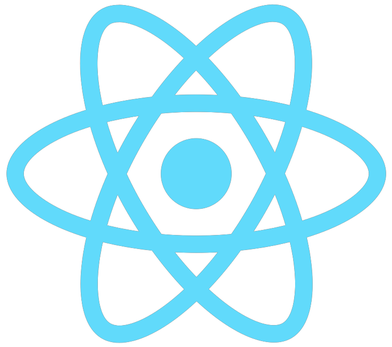
- The DOM is the tree that represents HTML nodes in our document
- The Shadow DOM is a fragment of the DOM that is **isolated** from the “light DOM”
- https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM

Below is a simple video example



Shadow DOM is not new...





```
<span class="user">
  <img
   class="user__avatar"
   src="desatranques.png">
  Desatranques Jaén
</span>
```

Web Components

```
<x-user>
#shadow-root
  <span class="user">
    <img
     class="user__avatar"
     src="desatranques.png">
    Desatranques Jaén
  </span>
</x-user>
```

```
customElements.define('x-user', class extends HTMLElement {
  constructor() {
    super();

    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML =
      '<span class="user">...</span>';
  }
});
```

DevTools + Web Components

- You can inspect, manipulate, change CSS rules, etc. of Custom Elements with a Shadow root like if they were regular HTML elements!

Demo: A simple component

https://belen-albeza.github.io/webcomponents-examples/wc-simple.html

<template>

- A tag to aid into creating new DOM with JavaScript
- It isn’t rendered by the browser, but:
  - We can inspect it with DevTools
  - We can easily clone it and then do an `appendChild`

```
<template id="user-tpl">
  <style>
    .user {
      border: 1px solid black;
    }
  </style>

  <span class="user">
    <img
     src="avatar.png">
    Anonymous
  </span>
</template>
```

```
const template = document.querySelector('#user-tpl');

this.attachShadow({ mode: 'open' });
this.shadowRoot.appendChild(
  template.content.cloneNode(true));
```

Demo: <template>

https://belen-albeza.github.io/webcomponents-examples/wc-template.html

Shadow DOM encapsulation

- JavaScript
  - Some isolation. Ex: document.querySelector won’t go inside, but Shadow DOM can access light DOM
- CSS
  - No classname collisions, only inherited properties go through
  - Styles don’t leak outside

<slot>

- Allows the user of a component to inject DOM inside.
- Multiple slots are allowed for a component
- Ej:

```
<x-message level="success">
  <p>Once upon a time…</p>
</x-message>
```

Demo: <slot>

https://belen-albeza.github.io/webcomponents-examples/wc-slots.html

Custom attributes

- We can make up **our own attributes** and render the content of the component depending on them
- But we have to **listen for attribute changes** if we want the component to reflect those changes.

```
// which attributes to watch
static get observedAttributes() {
  return ['username'];
}

// callback for changes
attributeChangedCallback(attr, oldValue, value) {
  // re-render component here
  // …
}
```

Demo: Custom attributes

https://belen-albeza.github.io/webcomponents-examples/wc-attrs.html

What about CSS?

- You don’t need a system to avoid class collisions from the outside (no BEM, or CSS Modules, or CSS in JS…)
- …or from the inside out! **STYLES DON’T LEAK**
- Properties & variables that are inherited permeate to the Shadow DOM (`color`, `font`…)

Demo CSS

https://belen-albeza.github.io/webcomponents-examples/wc-styles.html

Recap: pros of Web Components

- Encapsulation
- They are HTML Elements
  - Same API's you know and love: `querySelector`, `innerHTML`, `appendChild`, `addEventListener`…
  - You can use regular DevTools to debug them
- A standardised API

So… can I ditch React?

No.

JS Frameworks have many features

- Reusable components
- Template system
- **Data-binding**
- Other: routers, architecture (Angular, Redux, Vuex…), optimizations…

So what's the point?

- Web Components are here to **power up** frameworks.
- …or to allow micro-libraries to exist to make shareable components
- A **standard API** means that a WC component created with a framework can be used on a different one –or without any framework at all!

Some JS Frameworks already support WC

- https://custom-elements-everywhere.com/

Web Components made with Vue

- Step 1: Code your component as usual
- Step 2: Build with vue-cli and the `--target wc` flag

Demo: a Web Component made with Vue

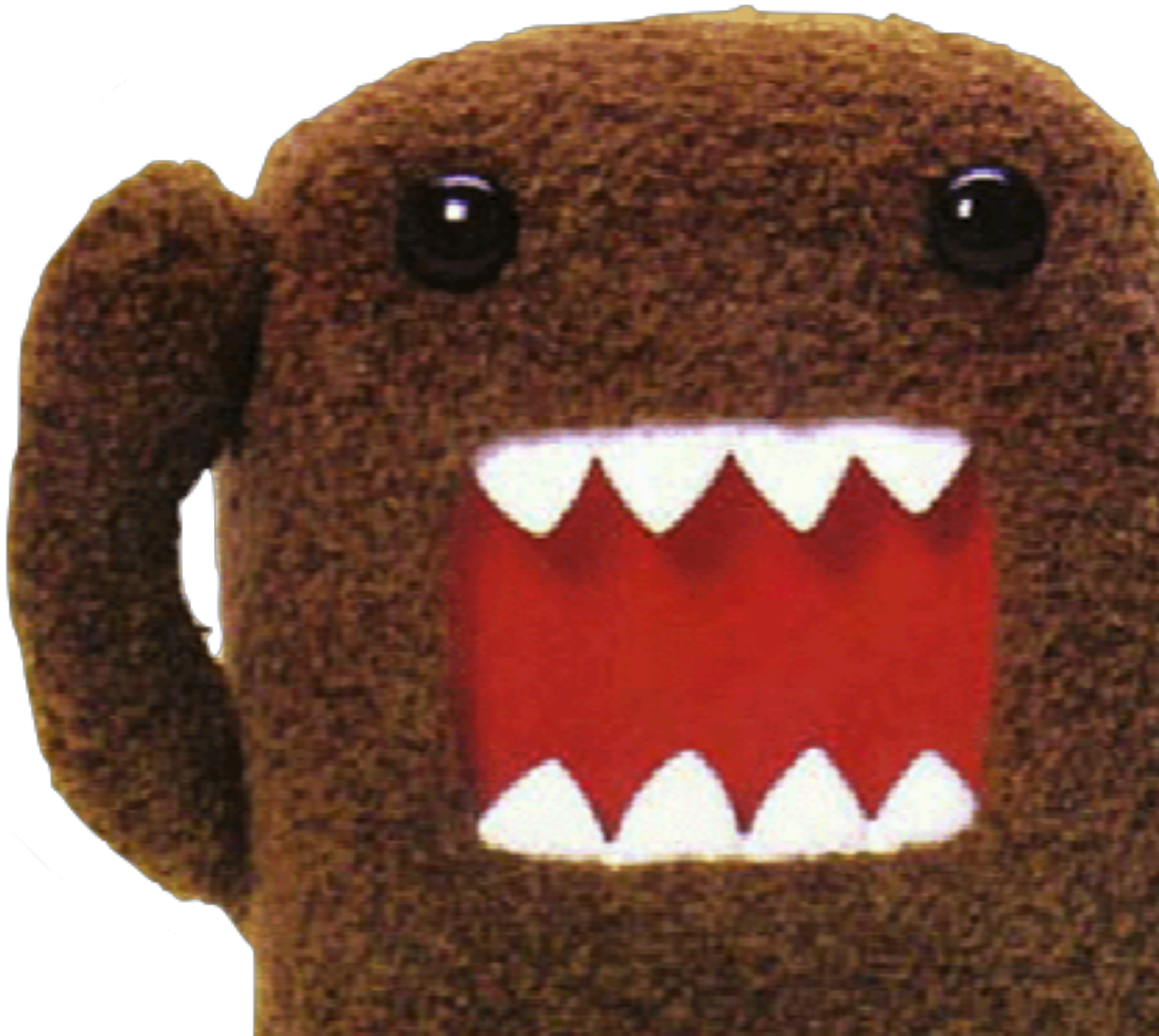
https://belen-albeza.github.io/webcomponents-examples/wc-vue.html

Demos + repo

https://belen-albeza.github.io/webcomponents-examples/

Thanks!

Questions?



Belén Albeza
@ladybenko