

# </> htmx 2.0 & Web Components

A Perfect Match for Frontend  
Development



@LostInBrittany



# Horacio Gonzalez



@LostInBrittany

Espagnol Perdu en Bretagne

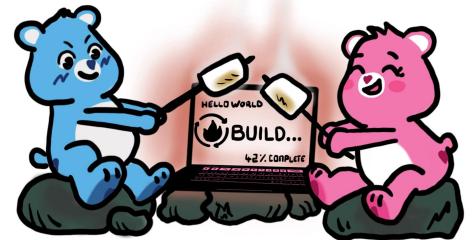
Head of DevRel



clever cloud



Finist  
Devs



clever cloud

@LostInBrittany

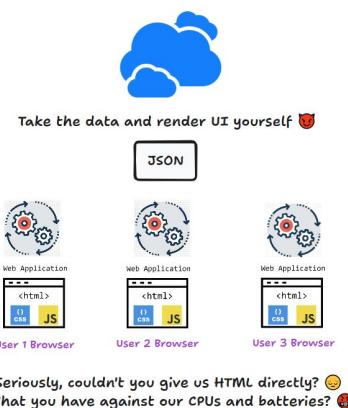




# </> htmx

Créez des interfaces utilisateur modernes avec  
la simplicité et la puissance de l'hypertexte

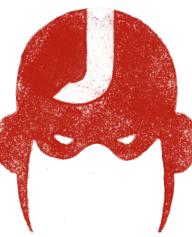
" Modern Approach "



" HTMX Approach "



# Les limites arbitraires du HTML



- Pourquoi seules les balises `<a>` et `<form>` peuvent-elles effectuer des requêtes HTTP(S) ?
- Pourquoi seuls les événements de clic et d'envoi peuvent-ils les déclencher ?
- Pourquoi seules les méthodes GET et POST sont-elles disponibles ?
- Pourquoi les balises `<a>` et `<form>` forcent le remplacement de l'intégralité de l'écran ?
- Pourquoi autant des limites arbitraires à HTML ?



# Objectif: de l'interactivité dans l'hypertexte



htmx étend les fonctionnalités de HTML

- effectuer des requêtes AJAX
- effectuer des transitions CSS
- gérer des WebSockets
- gérer les événements envoyés par le serveur

avec seulement des attributs HTML déclaratifs.

</> htmx





# Mais à quoi ça sert tout ça ?

Car c'est bien beau et sémantique, mais j'ai du mal à voir l'intérêt...



# Et si on faisait un petit saut arrière dans le temps ?



À une époque révolue, où les dinosaures codaient avec struts



clever cloud

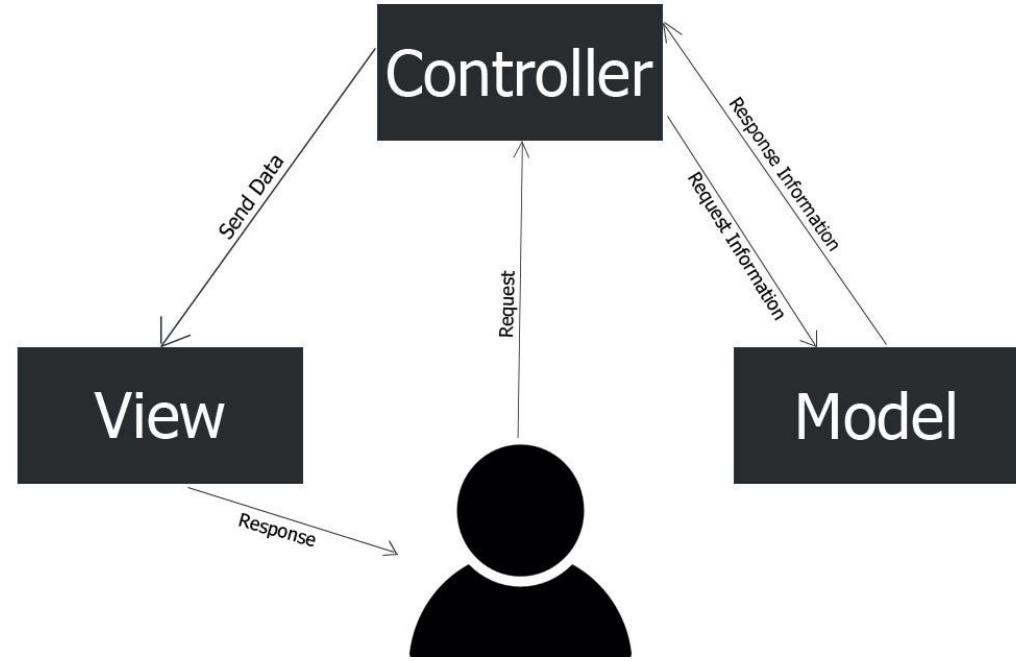
@LostInBrittany



# Souvenez-vous du modèle MVC



## Model-View-Controller



Avec la navigation page par page

# L'époque dorée d'une génération de frameworks



Sinatra



django

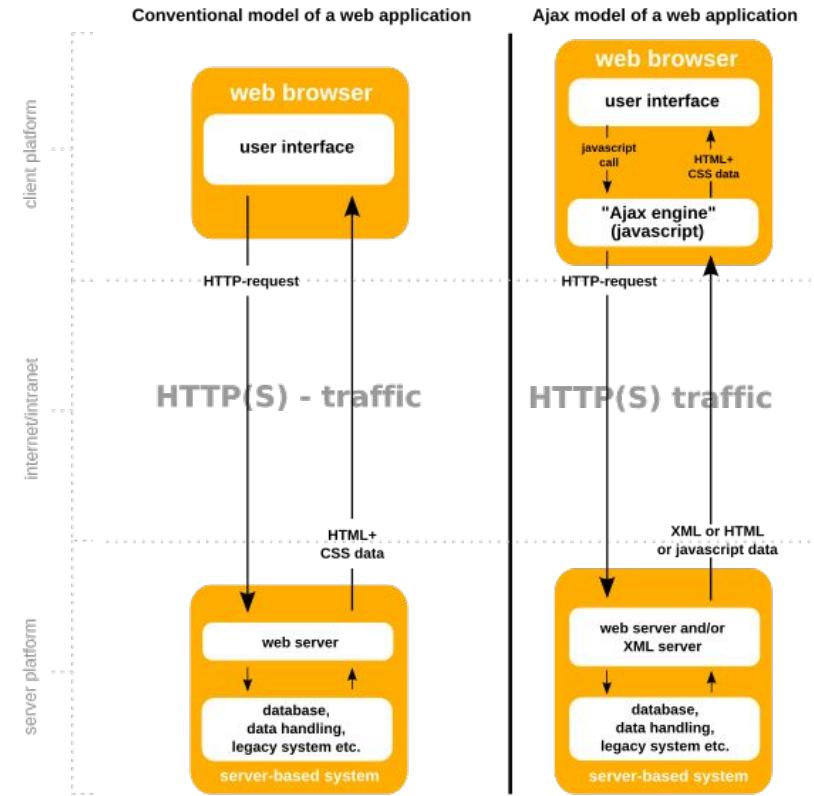


Generant des vues HTML

# Vers 2005 on voit arriver AJAX



AJAX



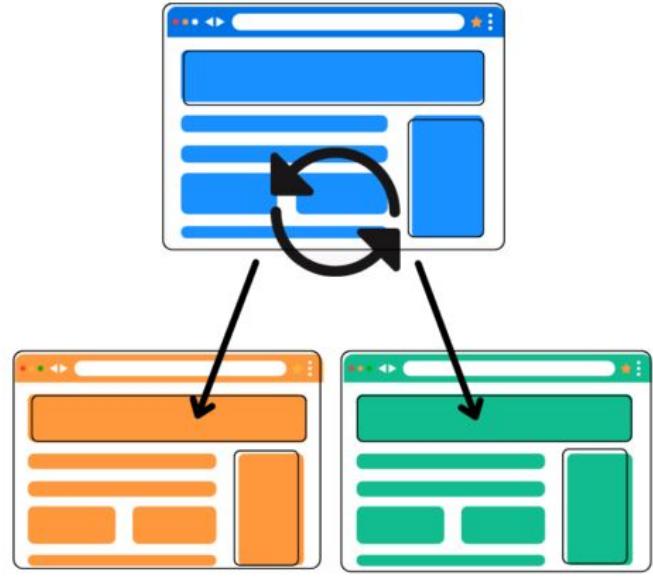
C'est la naissance du Web 2.0

# Et les pages webs deviennent des apps dynamiques

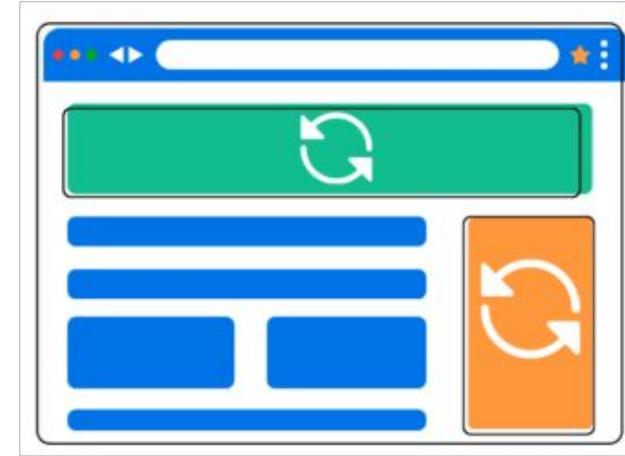


À coup de JavaScript, avec JQuery

# On bascule vers des Single Page Apps

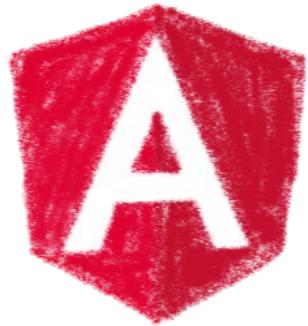
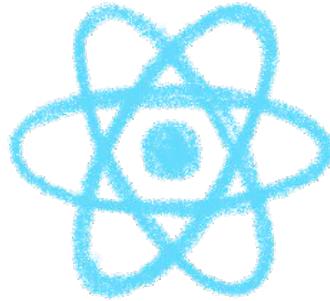


**MPA**  
Multi-page app

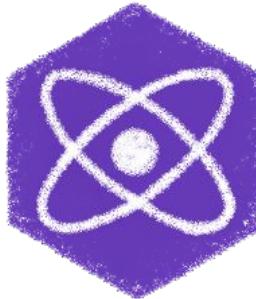


**SPA**  
Single-page app

# Des apps de plus en plus complexes



Lit

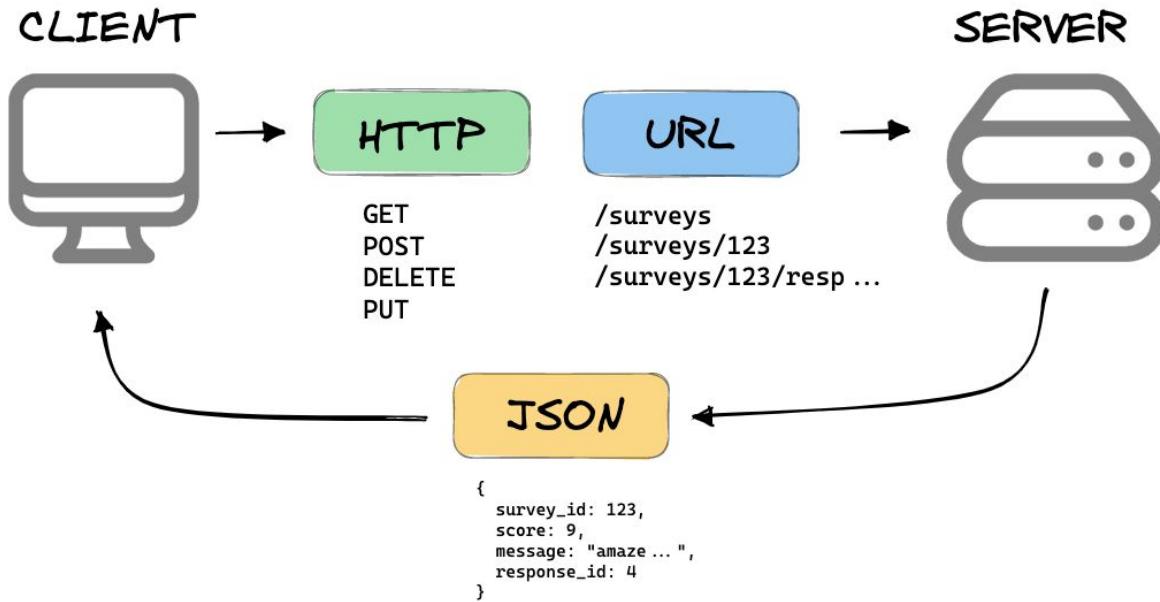


Avec l'apparition des frameworks JavaScript

# Et les backends deviennent des API REST

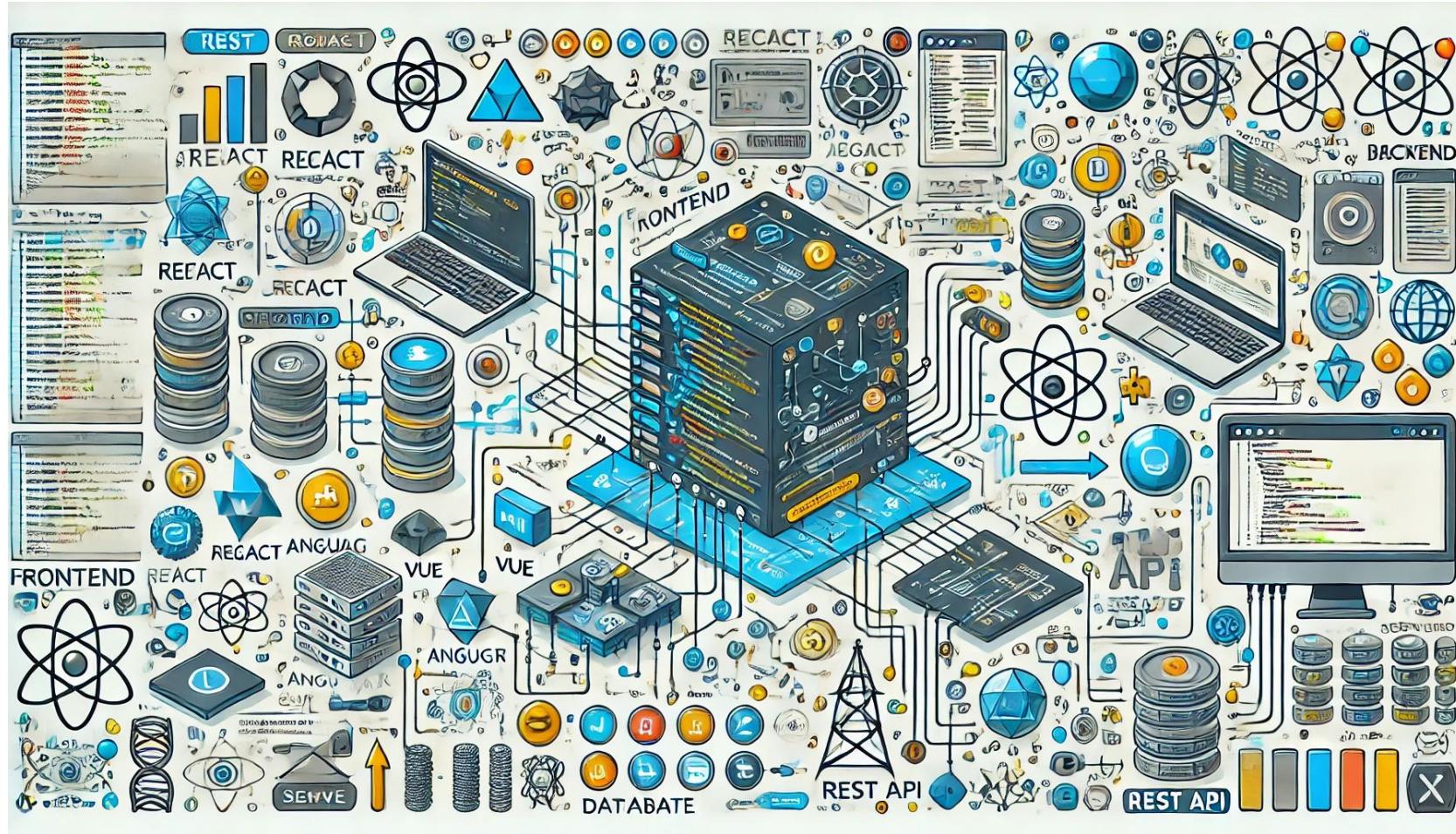


WHAT IS A REST API?



Servant du JSON

# On a gagné des fonctionnalités



A simple, hand-drawn style illustration of a baby's face. The face is light blue with a darker blue outline. It has large, expressive eyes with dark pupils, a small nose, and a wide, open mouth. A pacifier with a white 'S' on it is in the mouth. The cheeks are slightly blushing. The drawing is done with a textured, crayon-like appearance.

Mais on a perdu de la simplicité et le côté sémantique



# Pour beaucoup d'app c'est overkill



On ne veut qu'un site web avec un peu de dynamisme !

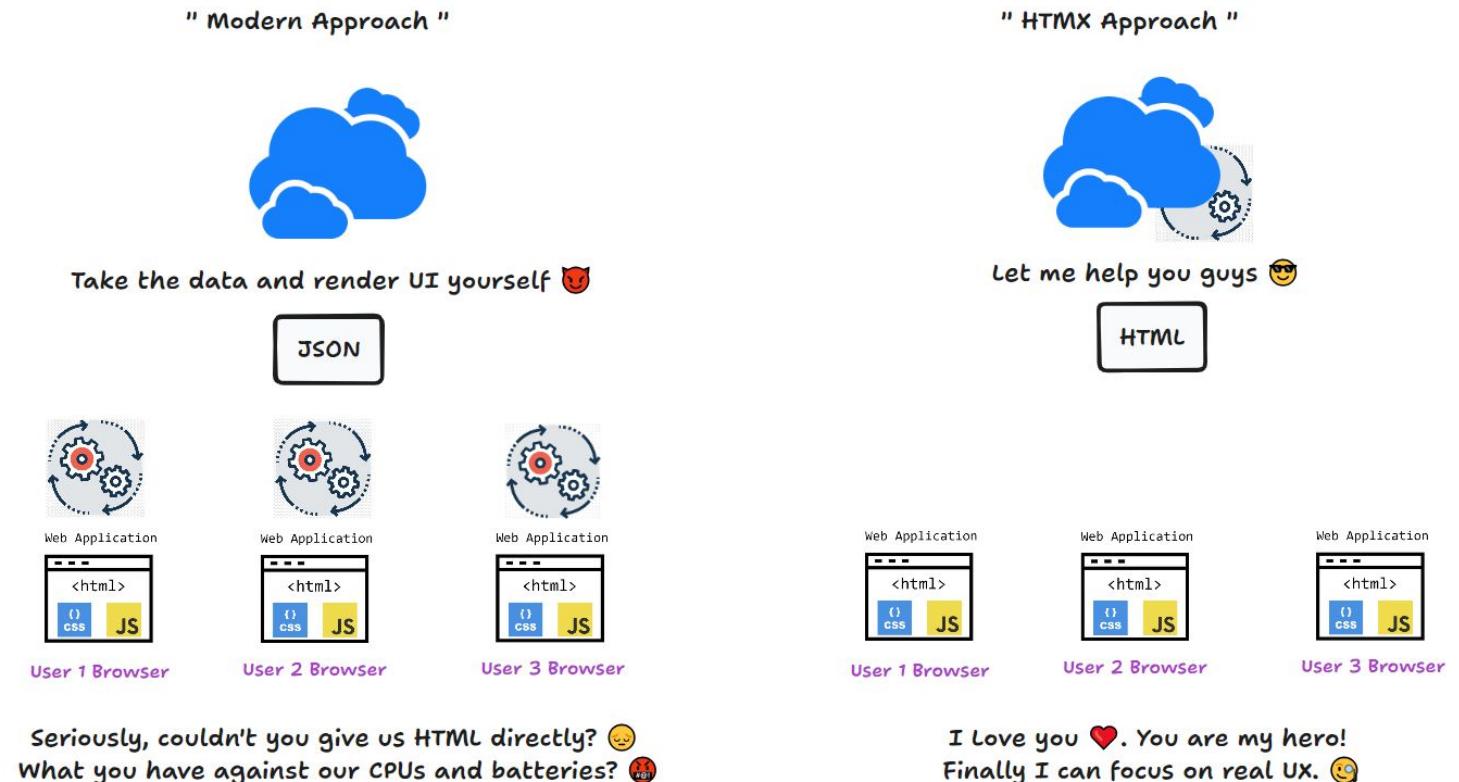
# </> htmx peut être la bonne solution

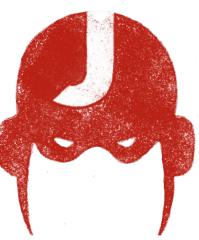


# </> htmx

C'est du HTML étendu

- Simplicité
- Sémantique
- Dynamisme





# On veut du code !

Des exemples ! Des exemples !

```
example-01.html
<script src="https://unpkg.com/htmrx.org@2.0.2"></script>

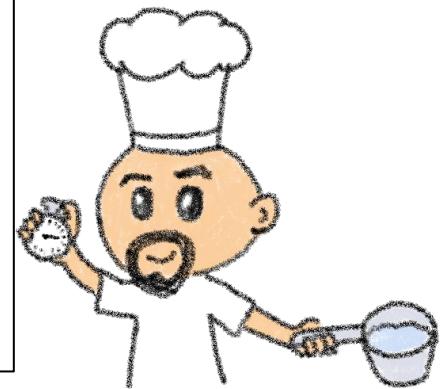
<!-- have a button that POST on a click via AJAX
     and replace the content of #status div
     with the response --&gt;

&lt;button hx-post="/clicked" hx-target="#status"&gt;
  Click Me
&lt;/button&gt;

&lt;div id="status"&gt;Not yet clicked&lt;/div&gt;</pre>

</> htmrx


```



clever cloud

@LostInBrittany

# Trop de blabla, on veut un exemple !



```
example-01.html

<script src="https://unpkg.com/htmrx.org@2.0.2"></script>

<!-- have a button that POST on a click via AJAX
     and replace the content of #status div
     with the response --&gt;

&lt;button hx-post="/clicked" hx-target="#status"&gt;
    Click Me
&lt;/button&gt;

&lt;div id="status"&gt;Not yet clicked&lt;/div&gt;</pre>

</> htmx


```

# GET, POST, PUT, DELETE...



example-02.html

```
<script src="https://unpkg.com/htmx.org@2.0.2"></script>

<div>
  <button hx-get="/test-methods" hx-target="#status">Send GET</button>
  <button hx-post="/test-methods" hx-target="#status">Send POST</button>
  <button hx-put="/test-methods" hx-target="#status">Send PUT</button>
  <button hx-delete="/test-methods" hx-target="#status">Send DELETE</button>
</div>

<div id="status">No request sent</div>
```

</> htmx

# Remplacer l'élément par la réponse



example-03.html

```
<script src="https://unpkg.com/htmxx.org@2.0.2"></script>

<div id="test-replace">
  <button hx-get="/test-replace/innerHTML">
    If you click, this message will be replaced
  </button>
  <button hx-get="/test-replace/outerHTML" hx-swap="outerHTML">
    If you click, this button will become a div
  </button>
  <button hx-get="/test-replace/delete" hx-swap="delete">
    If you click, this button will disappear when the response is received
  </button>
  <button hx-get="/test-replace/none" hx-swap="none">
    If you click, nothing changes, the response is ignored
  </button>
</div>
```

</> htmxx

# Choisir quand émettre la requête



example-04.html

```
<script src="https://unpkg.com/htmrx.org@2.0.2"></script>

<!-- By default, AJAX requests are triggered by the "natural" event of an element: --&gt;
&lt;div id="test-triggers"&gt;
    &lt;button hx-get="/trigger/natural" hx-target="#status"&gt;
        In a button the natural event is a click
    &lt;/button&gt;
    &lt;button hx-trigger="mouseover" hx-get="/trigger/mouseover" hx-target="#status"&gt;
        This button triggers on mouseover
    &lt;/button&gt;
    &lt;button hx-trigger="mouseenter" hx-get="/trigger/mouseenter" hx-target="#status"&gt;
        This button triggers on mouseenter
    &lt;/button&gt;
    &lt;button hx-trigger="mouseleave" hx-get="/trigger/mouseleave" hx-target="#status"&gt;
        This button triggers on mouseleave
    &lt;/button&gt;
&lt;/div&gt;

&lt;div id="status"&gt;No AJAX request sent yet&lt;/div&gt;</pre>
```

</> htmx



clever cloud

@LostInBrittany



# Choisir quand émettre la requête



example-05.html

```
<script src="https://unpkg.com/htmxa.org@2.0.2"></script>

<!-- By default, AJAX requests are triggered by the "natural" event of an element: --&gt;
&lt;div id="test-triggers"&gt;
  &lt;button hx-trigger="every 5s" hx-get="/trigger/5seconds" hx-target="#status"&gt;
    Sends request every 5 seconds, no event needed
  &lt;/button&gt;
  &lt;button hx-trigger="click[ctrlKey]" hx-get="/trigger/ctrlclick" hx-target="#status"&gt;
    Sends request on click while pressing Ctrl
  &lt;/button&gt;
  &lt;button hx-trigger="click[ctrlKey] once" hx-get="/trigger/ctrlclickonce" hx-target="#status"&gt;
    Sends request on the first click while pressing Ctrl
  &lt;/button&gt;
&lt;/div&gt;

&lt;div id="status"&gt;No AJAX request sent yet&lt;/div&gt;</pre>
```

</> htmx



clever cloud

@LostInBrittany



# Un spinner pour faire patienter



example-06.html

```
<script src="https://unpkg.com/htmxa.org@2.0.2"></script>

<!-- By default, AJAX requests are triggered by the "natural" event of an element: --&gt;
&lt;div id="test-triggers"&gt;
  &lt;button hx-trigger="every 5s" hx-get="/trigger/5seconds" hx-target="#status"&gt;
    Sends request every 5 seconds, no event needed
  &lt;/button&gt;
  &lt;button hx-trigger="click[ctrlKey]" hx-get="/trigger/ctrlclick" hx-target="#status"&gt;
    Sends request on click while pressing Ctrl
  &lt;/button&gt;
  &lt;button hx-trigger="click[ctrlKey] once" hx-get="/trigger/ctrlclickonce" hx-target="#status"&gt;
    Sends request on the first click while pressing Ctrl
  &lt;/button&gt;
&lt;/div&gt;

&lt;div id="status"&gt;No AJAX request sent yet&lt;/div&gt;</pre>
```

</> htmx



clever cloud

@LostInBrittany



# Des extensions presque à l'infini



## </> htmx extensions

This site is a searchable collection of extensions for [htmx 2.0](#). They are not guaranteed to work with the htmx 1.x codebase.

[Core](#) extensions are actively maintained by the htmx team.

[Community](#) extensions are contributed by the community or rarely touched by the htmx team (although they still work!)

### ► Contributing

#### [Core](#)

search core extensions

Name	Description
<a href="#">sse</a>	Provides support for <a href="#">Server Sent Events</a> directly from HTML.
<a href="#">ws</a>	Provides bi-directional communication with <a href="#">Web Sockets</a> servers directly from HTML



# On veut du code !

Faisons un exemple

## To-Do List

Donner le talk au FinistDevs



Manger de la pizza

 addTask

# Tout le code est disponible



Screenshot of a GitHub repository page for `introduction-to-htmx`.

The repository is private and contains the following files:

- frontend
- img
- server
- .gitignore
- README.md
- package-lock.json
- package.json

The README file contains the following content:

## Introduction to </> htmx

This repository stores all the code for my talk *htmx 2.0 & Web Components: A Perfect Match for Frontend Development*, that I have given at:

- 2024-09-24 - [FinistDevs](#)

Repository statistics:

- 1 Commit
- 0 stars
- 1 watching
- 0 forks

Language distribution:

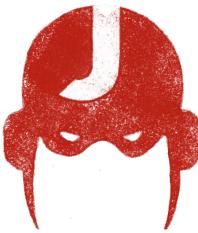
- JavaScript: 66.7%
- HTML: 33.3%

Suggested workflows:

Based on your tech stack

<https://github.com/LostInBrittany/introduction-to-htmx>

# Des démos de htmx



## To-Do List

Donner le talk au FinistDevs



Manger de la pizza



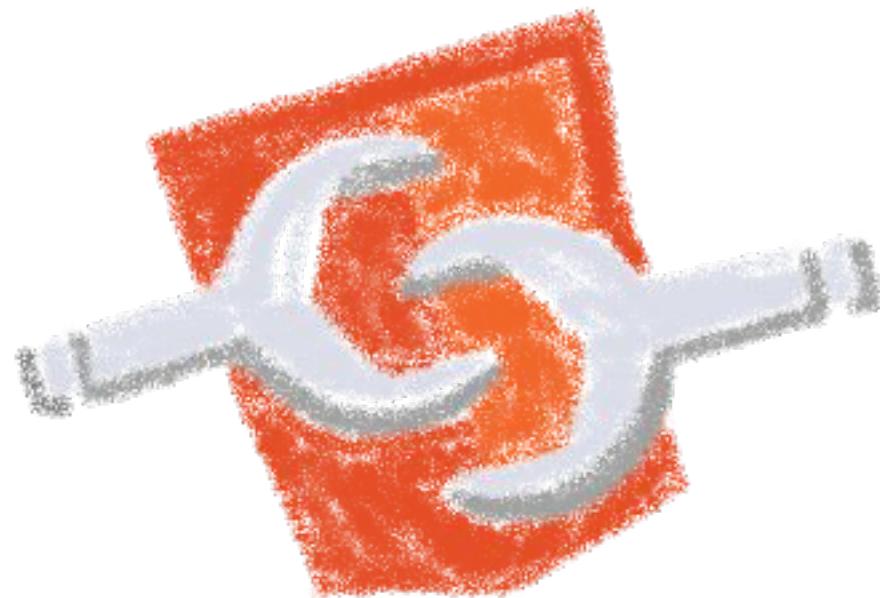
addTask

Du Hello World à la To-do List



# What the heck are web components?

The 3 minutes context

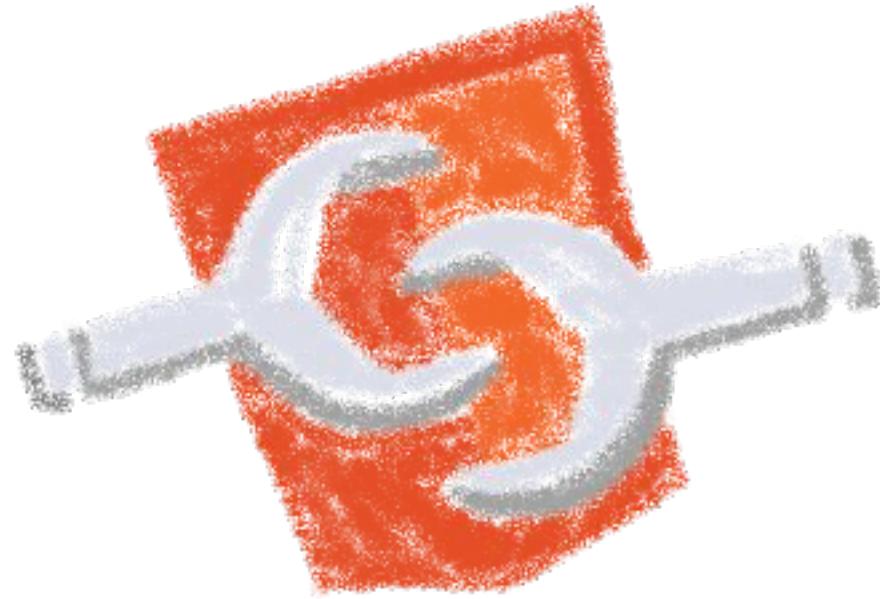


clever cloud

@LostInBrittany

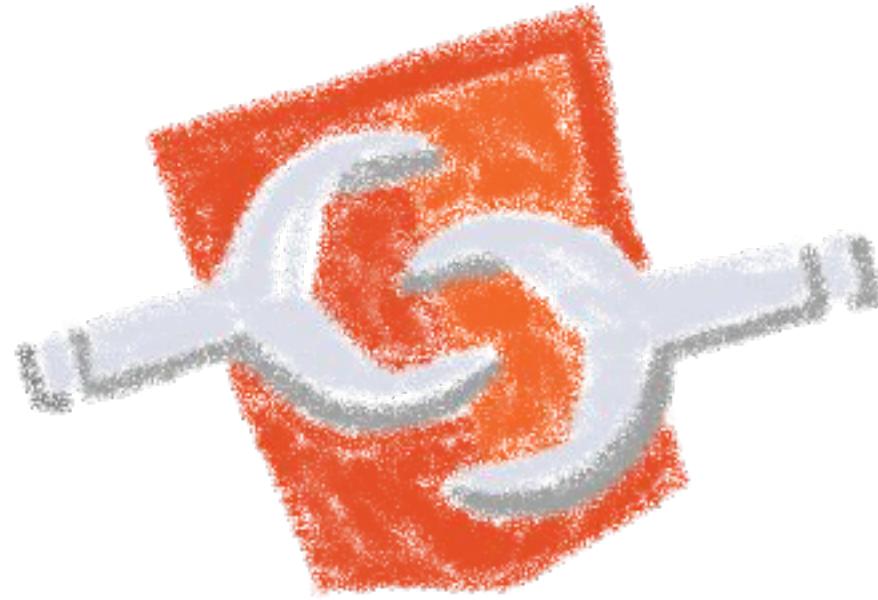


# Web Components



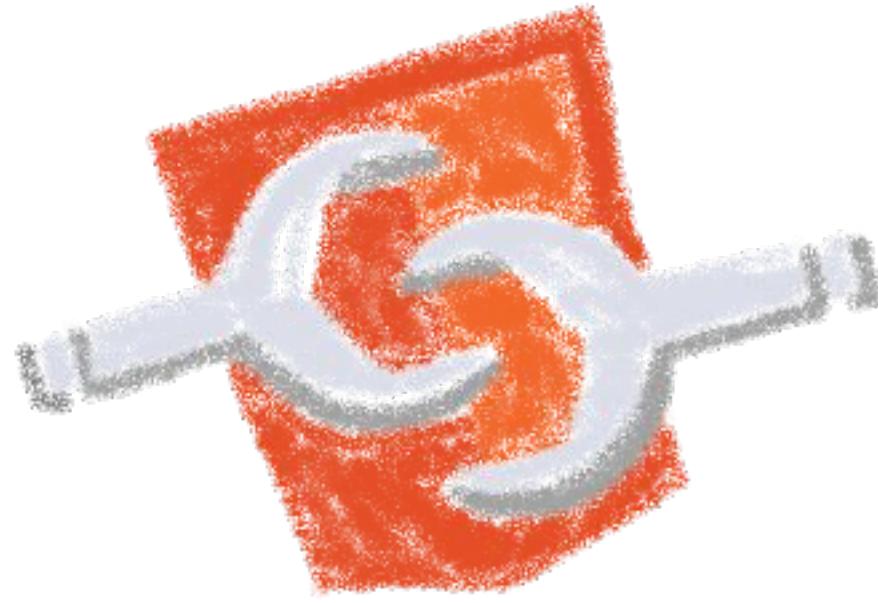
Web standard W3C

# Web Components



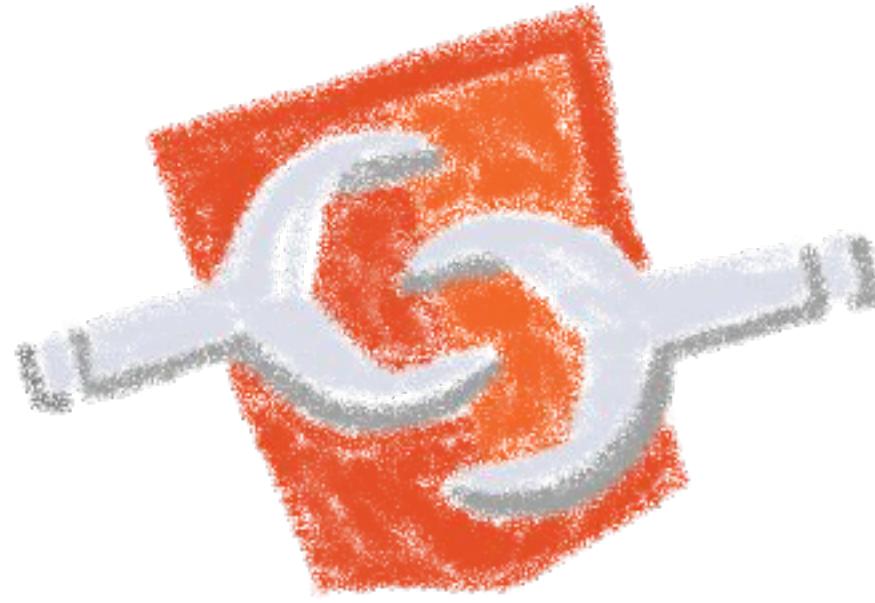
Available in all modern browsers:  
Firefox, Safari, Chrome

# Web Components



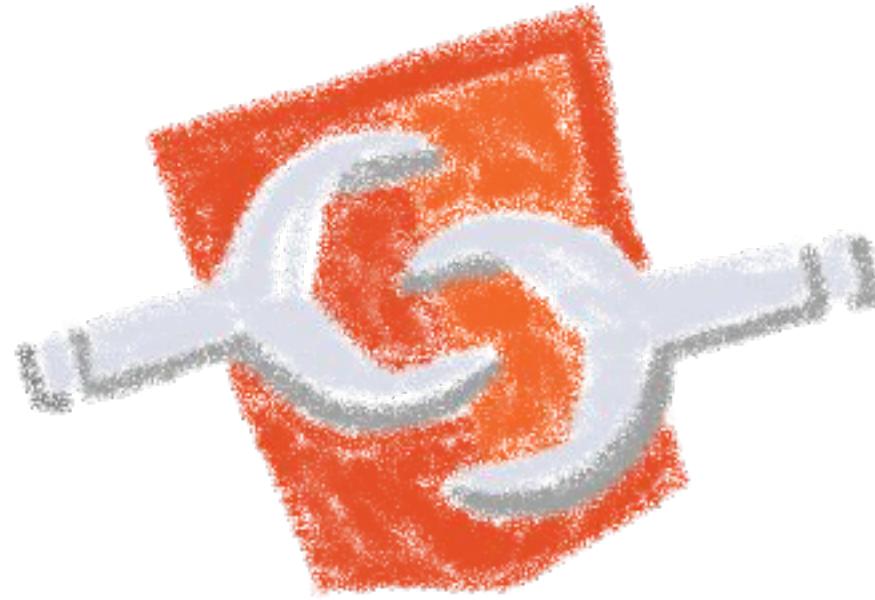
Create your own HTML tags  
Encapsulating look and behavior

# Web Components



Fully interoperable  
With other web components, with any framework

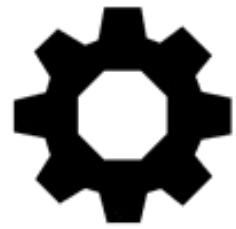
# Web Components



CUSTOM ELEMENTS



SHADOW DOM



TEMPLATES



clever cloud

@LostInBrittany



# Custom Element



`</>` To define your own HTML tag

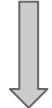
```
<body>
  ...
<script>
  window.customElements.define('my-element',
    class extends HTMLElement {...});
</script>
<my-element></my-element>
</body>
```

# Shadow DOM



To encapsulate subtree and style in an element

Hello, world!



```
<button>Hello, world!</button>
<script>
var host = document.querySelector('button');
const shadowRoot = host.attachShadow({mode:'open'});
shadowRoot.textContent = 'こんにちは、影の世界!';
</script>
```

こんにちは、影の世界!

# Template



To have clonable document template

```
<template id="mytemplate">
  <img src="" alt="great image">
  <div class="comment"></div>
</template>
```

```
var t = document.querySelector('#mytemplate');
// Populate the src at runtime.
t.content.querySelector('img').src = 'logo.png';
var clone = document.importNode(t.content, true);
document.body.appendChild(clone);
```



clever cloud

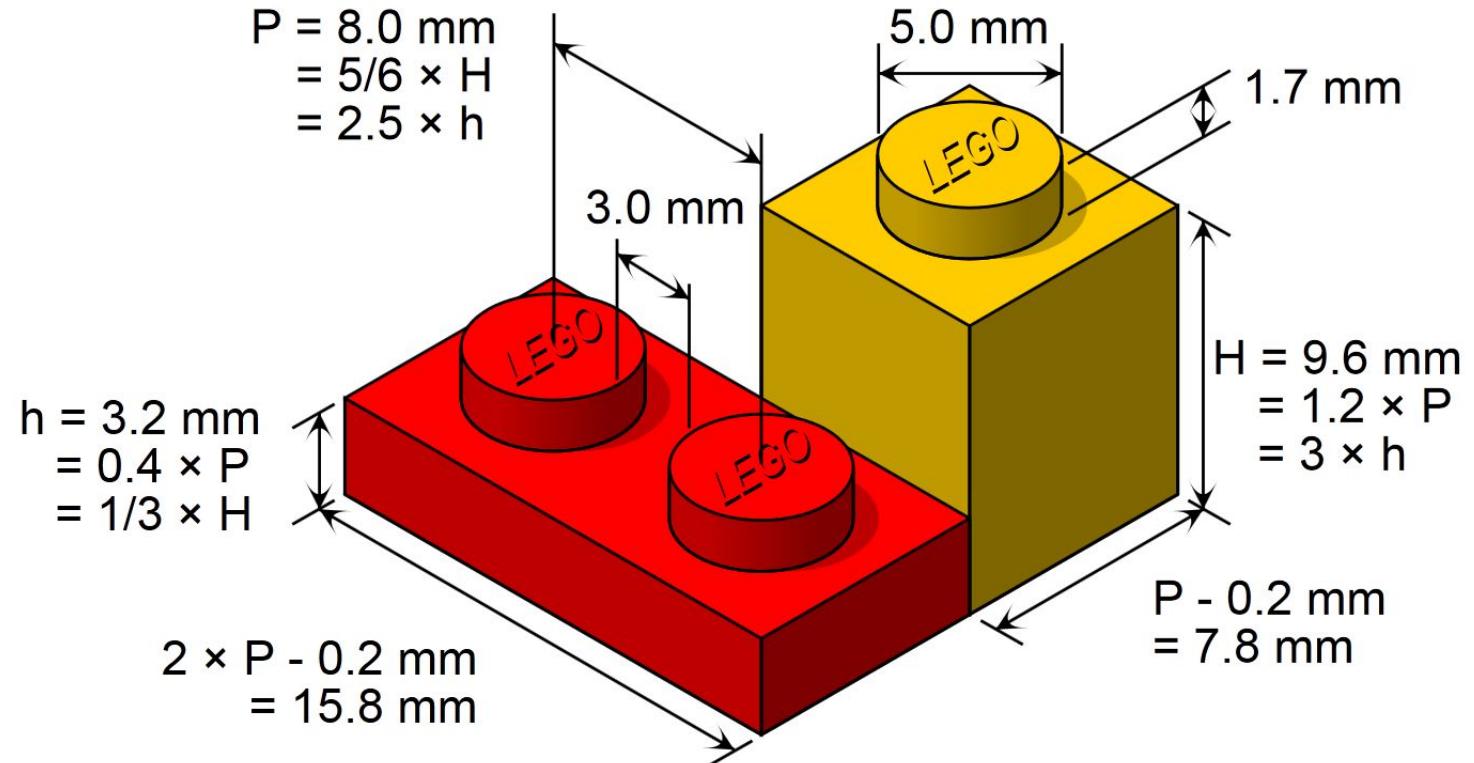
@LostInBrittany 

A small cartoon character with glasses and a pencil, holding a sign that says '@LostInBrittany'.

# But in fact, it's just an element...



- Attributes
- Properties
- Methods
- Events





## Simple. Fast. Web Components



### Skip the boilerplate

Building on top of the Web Components standards, Lit adds just what you need to be happy and productive: reactivity, declarative templates and a handful of thoughtful features to reduce boilerplate and make your



### Tiny footprint, instant updates

Weighing in at around 5 KB (minified and compressed), Lit helps keep your bundle size small and your loading time short. And rendering is blazing fast, because Lit touches only the dynamic parts of your UI when updating



### Web Components

### Interoperable & future-ready

Every Lit component is a native web component, with the superpower of interoperability. Web components work anywhere you use HTML, with any framework or none at all. This makes Lit ideal for building shareable



clever cloud

@LostInBrittany 

# Modern lightweight web components



Simple.  
Fast.  
Web Components.

> npm i lit

Get Started



Simple

Skip the boilerplate

Building on top of the Web Components standards, Lit adds just what you need to be happy and productive: reactivity, declarative templates and a handful of thoughtful features to reduce boilerplate and make your



Fast

Tiny footprint, instant updates

Weighing in at around 5 KB (minified and compressed), Lit helps keep your bundle size small and your loading time short. And rendering is blazing fast, because Lit touches only the dynamic parts of your UI when updating



Web Components

Interoperable & future-ready

Every Lit component is a native web component, with the superpower of interoperability. Web components work anywhere you use HTML, with any framework or none at all. This makes Lit ideal for building shareable

## For the new web paradigm



clever cloud

@LostInBrittany



# LitElement



```
import { LitElement, html } from 'lit-element';

// Create your custom component
class CustomGreeting extends LitElement {
  // Declare properties
  static get properties() {
    return {
      name: { type: String }
    };
  }
  // Initialize properties
  constructor() {
    super();
    this.name = 'World';
  }
  // Define a template
  render() {
    return html`<p>Hello, ${this.name}!</p>`;
  }
}
// Register the element with the browser
customElements.define('custom-greeting', CustomGreeting);
```

Lightweight web-components using lit-html



clever cloud

@LostInBrittany



# Based on lit-html



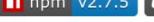
lit / packages / lit-html / README.md ...

 abdonrd Unify the npm badges (#3680) X d85f082 · 5 months ago History

Preview Code Blame 58 lines (36 loc) · 2.7 KB Raw Copy Download Edit ☰

## lit-html 2.0

Efficient, Expressive, Extensible HTML templates in JavaScript

lit-html is the template system that powers the [Lit](#) library for building fast web components. When using `lit-html` to develop web components, most users should import lit-html via the [lit](#) package rather than installing and importing from `lit-html` directly.

### About this release

This is a stable release of `lit-html` 2.0 (part of the Lit 2.0 release). If upgrading from previous versions of `lit-html`, please note the minor breaking changes from lit-html 1.0 in the [Upgrade Guide](#).

An efficient, expressive, extensible  
HTML templating library for JavaScript

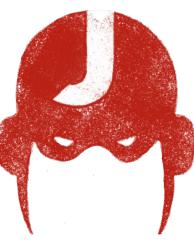
# Do you know tagged templates?



```
function uppercaseExpression(strings, ...expressionValues) {  
  var finalString = ''  
  for ( let i = 0; i < strings.length; i++ ) {  
    if (i > 0) {  
      finalString += expressionValues[i - 1].toUpperCase()  
    }  
    finalString += strings[i]  
  }  
  return finalString  
}  
  
const expressions = [ 'Sophia Antipolis', 'RivieraDev', 'Thank you'];  
console.log(uppercase`Je suis à ${expression[0]} pour ${expression[1]}. ${expression[2]}!`)
```

Little known functionality of template literals

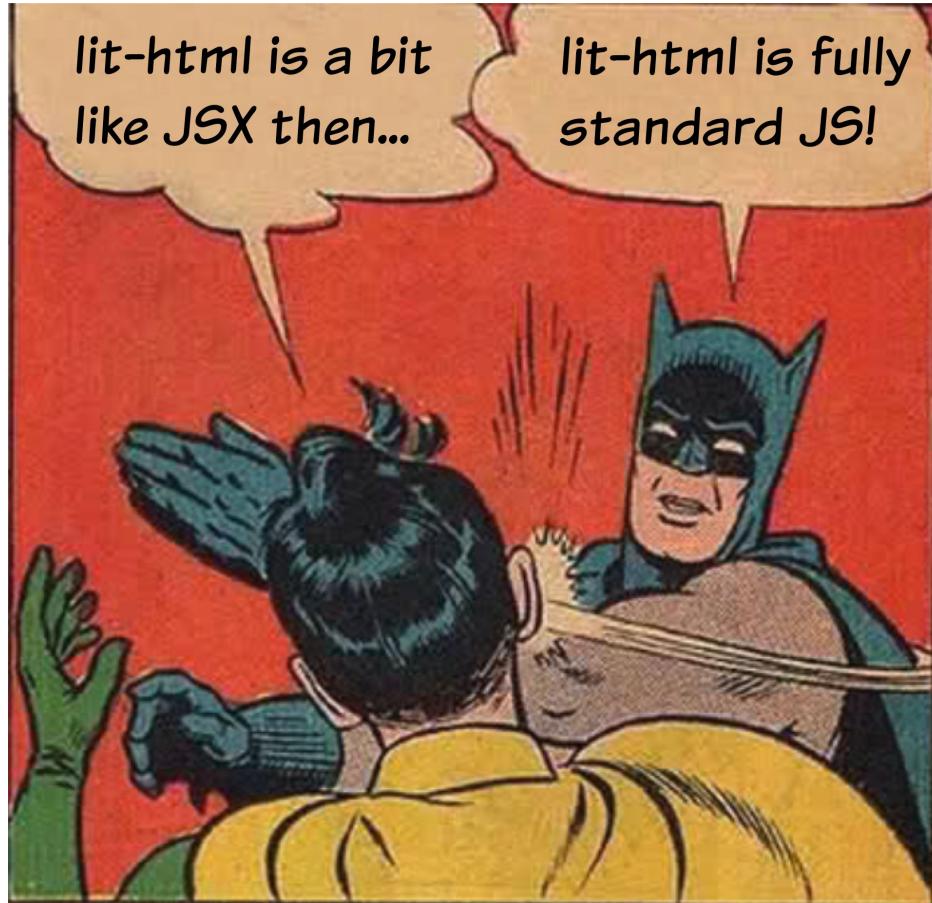
# lit-html Templates



```
let myTemplate = (data) => html`  
  <h1>${data.title}</h1>  
  <p>${data.body}</p>  
`;
```

Lazily rendered  
Generates a TemplateResult

# It's a bit like JSX, isn't it?



The good sides of JSX... but in the standard!

# LitElement



```
import { LitElement, html } from 'lit-element';

// Create your custom component
class CustomGreeting extends LitElement {
  // Declare properties
  static get properties() {
    return {
      name: { type: String }
    };
  }
  // Initialize properties
  constructor() {
    super();
    this.name = 'World';
  }
  // Define a template
  render() {
    return html`<p>Hello, ${this.name}!</p>`;
  }
}
// Register the element with the browser
customElements.define('custom-greeting', CustomGreeting);
```

Lightweight web-components using lit-html

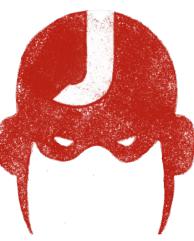


clever cloud

@LostInBrittany



# Une petit démo de Lit



Faisons un compteur : `my-lit-counter`



# Lit & </> htmx

Amour au premier <tag>



Lit

</> htmx



clever cloud

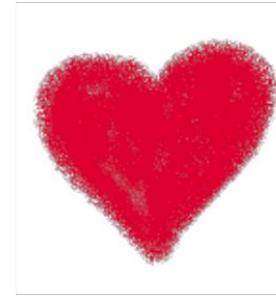
@LostInBrittany



# htmx pour structure, lit pour encapsuler l'intelligence



</> htmx



Lit

Pour htmx les éléments lits sont comme n'importe quel tag

# Une petit démo de htmx + Lit



Faisons un compteur : `my-lit-counter`





# That's all, folks!

Thank you all!



clever cloud

@LostInBrittany

