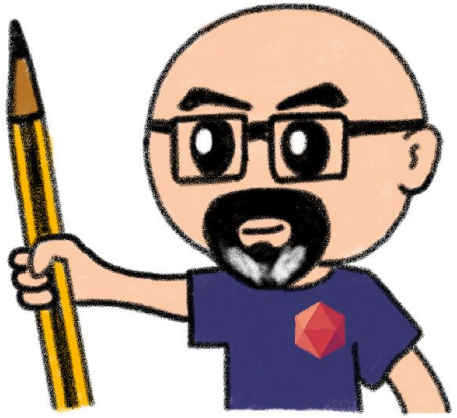
</> htmx 2.0 & Web Components

A Perfect Match for Frontend Development

clever cloud

@LostInBrittany

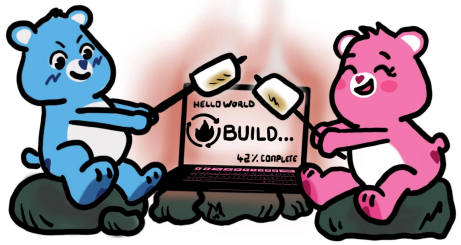


Horacio Gonzalez

@LostInBrittany

Espagnol Perdu en Bretagne

Finist Devs

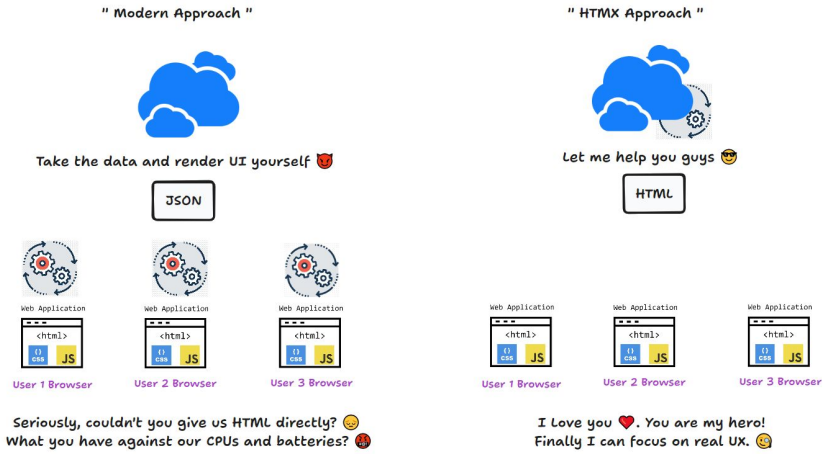


Head of DevRel

clever cloud

</> htmx

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



clever cloud

@LostInBrittany

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...

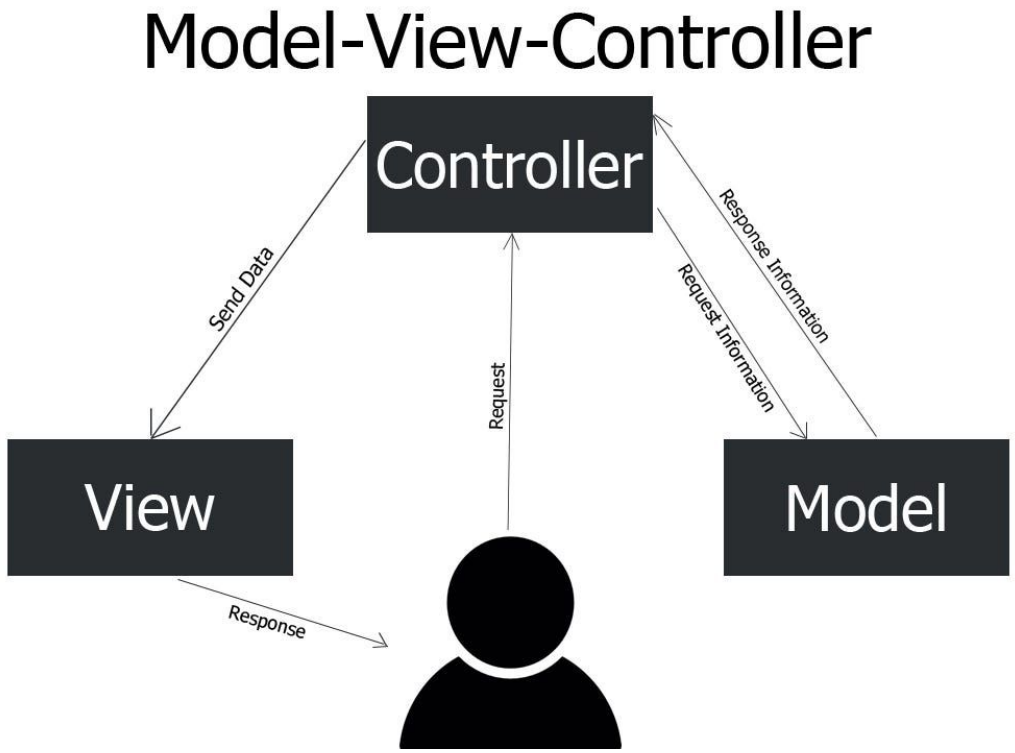
clever cloud

@LostInBrittany

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

À une époque révolue, où les dinosaurs codaient avec struts

Souvenez-vous du modèle MVC



Avec la navigation page par page

L'époque dorée d'une génération de frameworks

RAILS

Sinatra

play

STRUTS

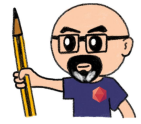
GRAILS

django

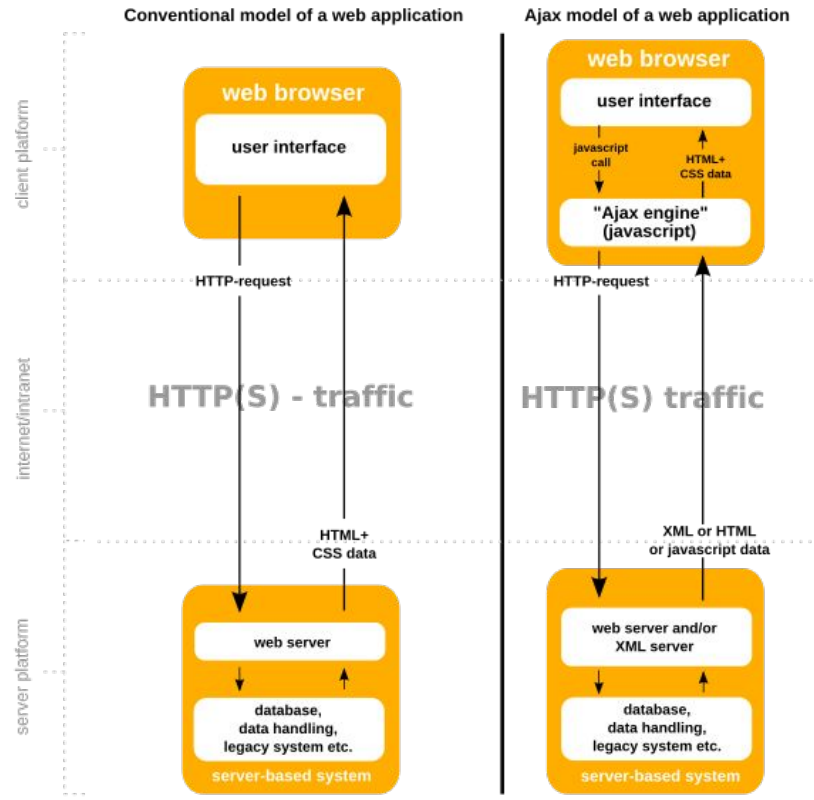
Generant des vues HTML

clever cloud

@LostInBrittany



Vers 2005 on voit arriver AJAX



C'est la naissance du Web 2.0

clever cloud

@LostInBrittany

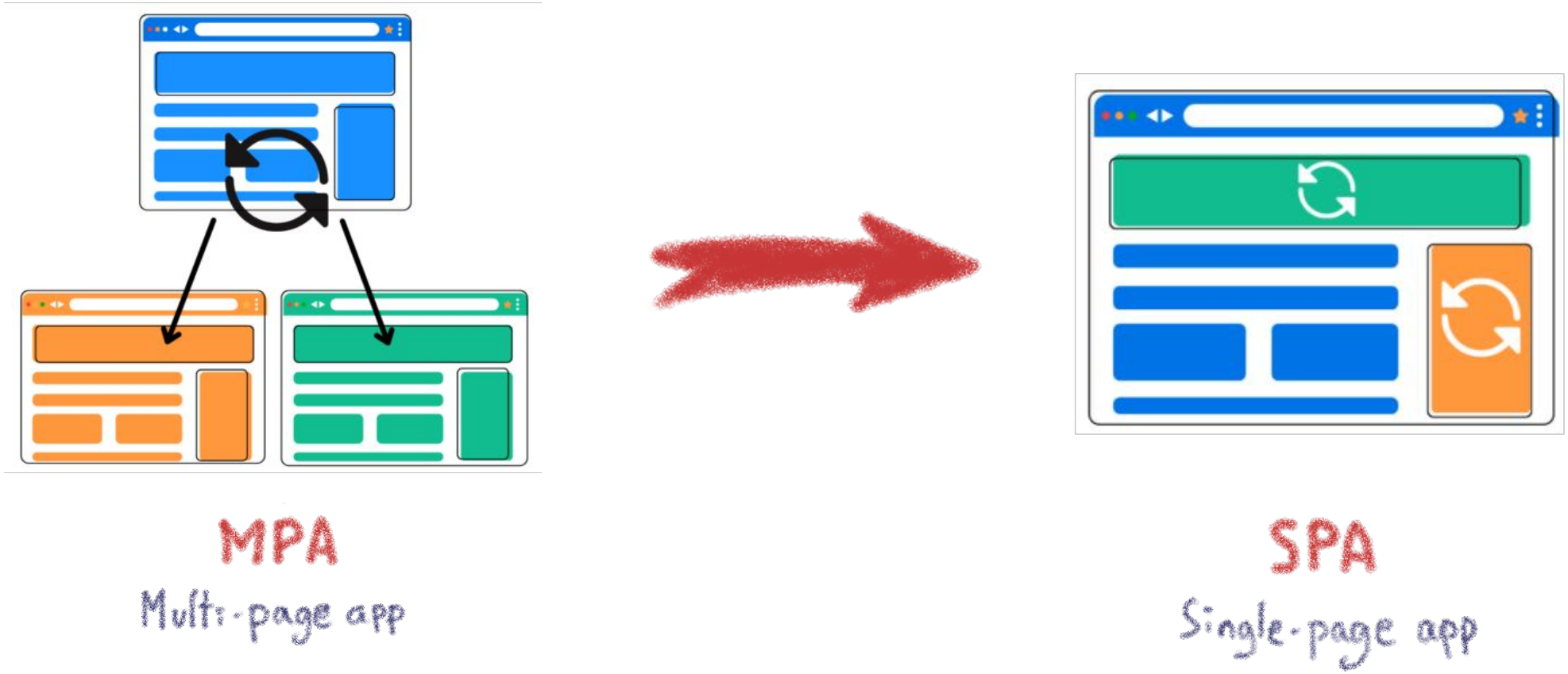
Et les pages webs deviennent des apps dynamiques

À coup de JavaScript, avec JQuery

clever cloud

@LostInBrittany

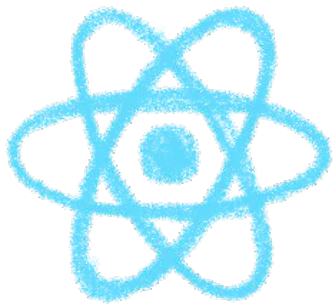
On bascule vers des Single Page Apps



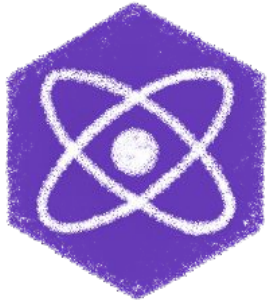
clever cloud

@LostInBrittany

Des apps de plus en plus complexes



Lit

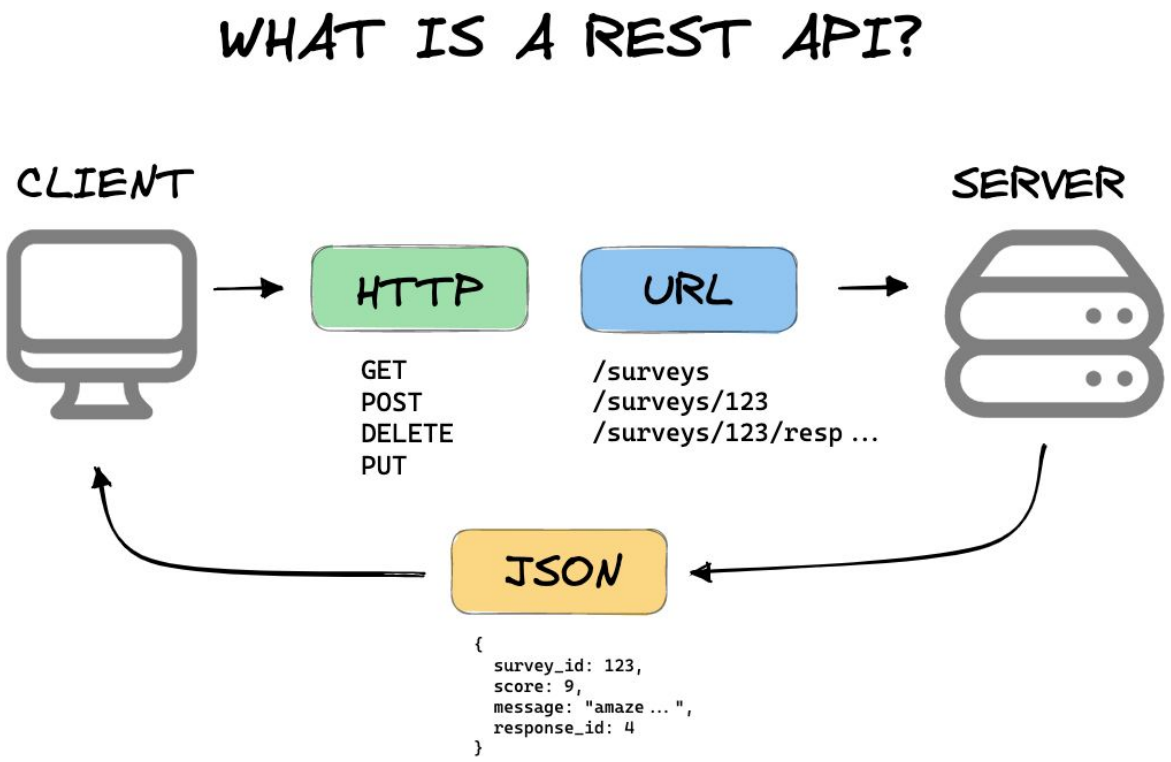


Avec l'apparition des frameworks JavaScript

clever cloud

@LostInBrittany

Et les backends deviennent des API REST

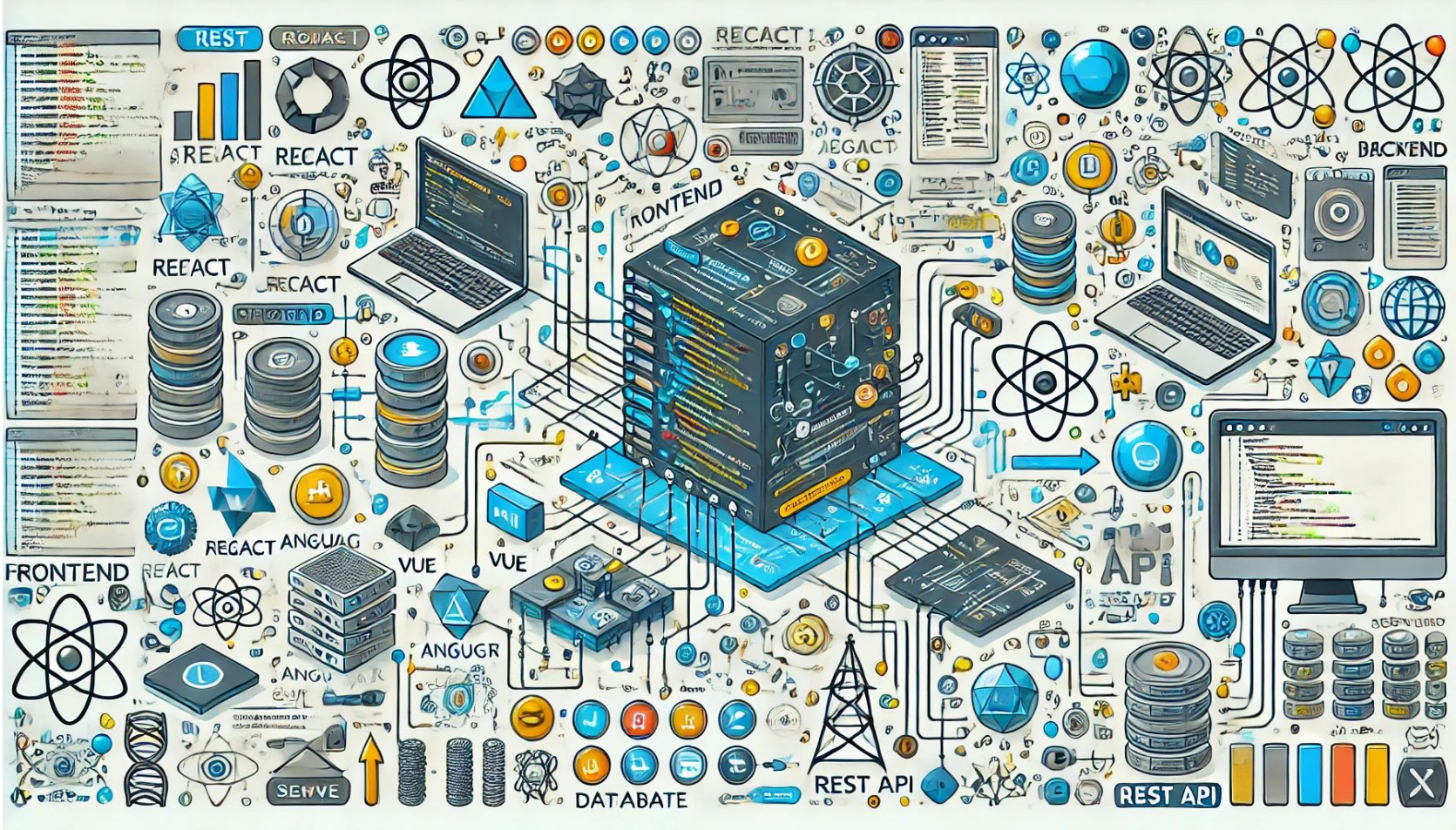


Servant du JSON

clever cloud

@LostInBrittany

On a gagné des fonctionnalités



Mais on a perdu de la simplicité et le côté sémantique

clever cloud

@LostInBrittany

Pour beaucoup d'app c'est overkill

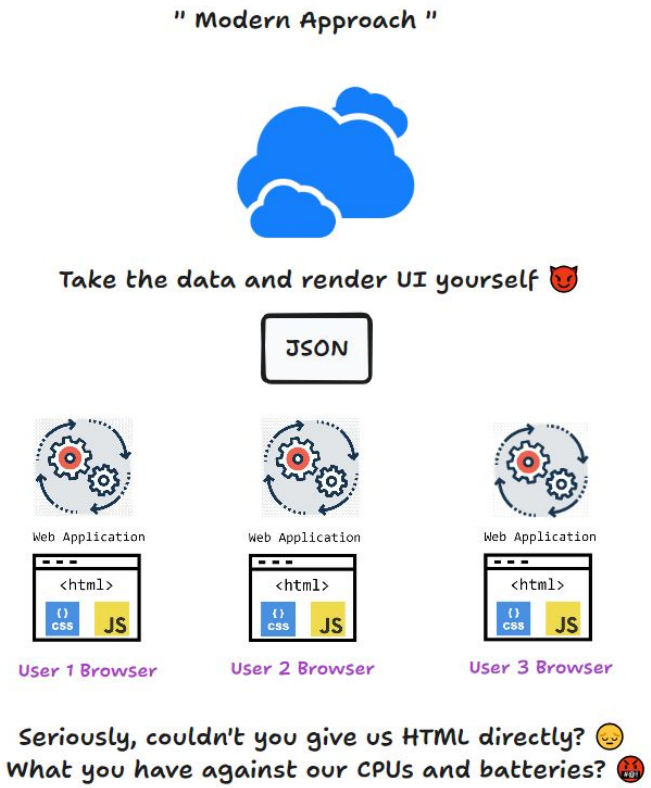
On ne veut qu'un site web avec un peu de dynamisme !

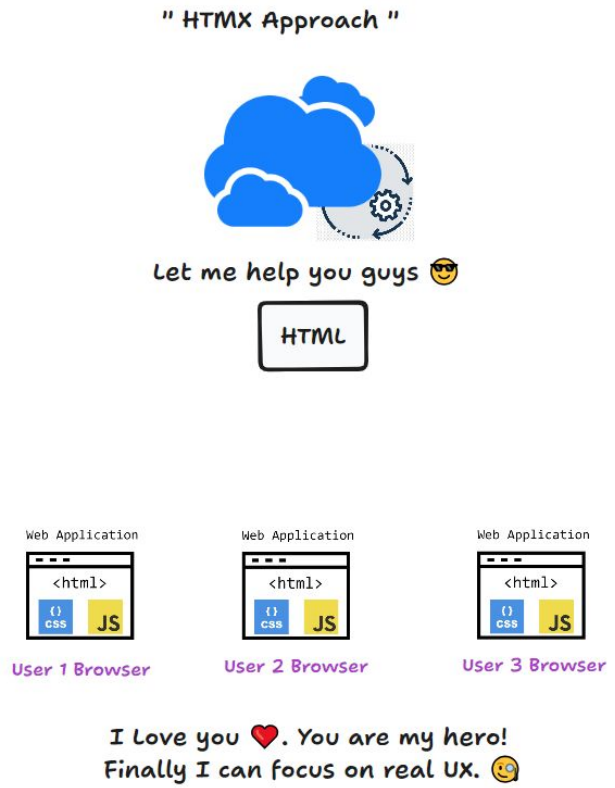
</> htmx peut être la bonne solution

</> htmx

C'est du HTML étendu

- Simplicité
- Sémantique
- Dynamisme





clever cloud

@LostInBrittany

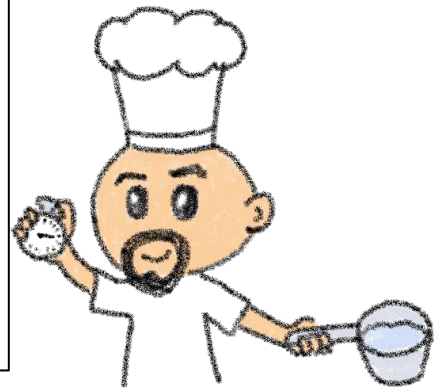
On veut du code !

Des exemples ! Des exemples !

example-01.html

```
<script src="https://unpkg.com/htmx.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 -->

<button hx-post="/clicked" hx-target="#status">
   Click Me
</button>

<div id="status">Not yet clicked</div>
```

</> htmx



clever cloud

@LostInBrittany

Trop de blabla, on veut un exemple !

example-01.html

```
<script src="https://unpkg.com/htmx.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 -->

<button hx-post="/clicked" hx-target="#status">
   Click Me
</button>

<div id="status">Not yet clicked</div>
```

</> htmx

clever cloud

@LostInBrittany

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

clever cloud

@LostInBrittany

Remplacer l'élément par la réponse

example-03.html

```
<script src="https://unpkg.com/htmx.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 dissapear 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>
```

</> htmx

Choisir quand émettre la requête

example-04.html

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

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

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

</> htmx

clever cloud

@LostInBrittany

Choisir quand émettre la requête

example-05.html

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

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

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

</> htmx

clever cloud

@LostInBrittany

Un spinner pour faire patienter

example-06.html

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

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

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

</> 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**

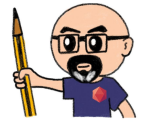
Core

search core extensions

Name	Description
sse	Provides support for Server Sent Events directly from HTML.
ws	Provides bi-directional communication with Web Sockets servers directly from HTML

clever cloud

@LostInBrittany



On veut du code !

Faisons un exemple

To-Do List

Donner le talk au FinistDevs 📝

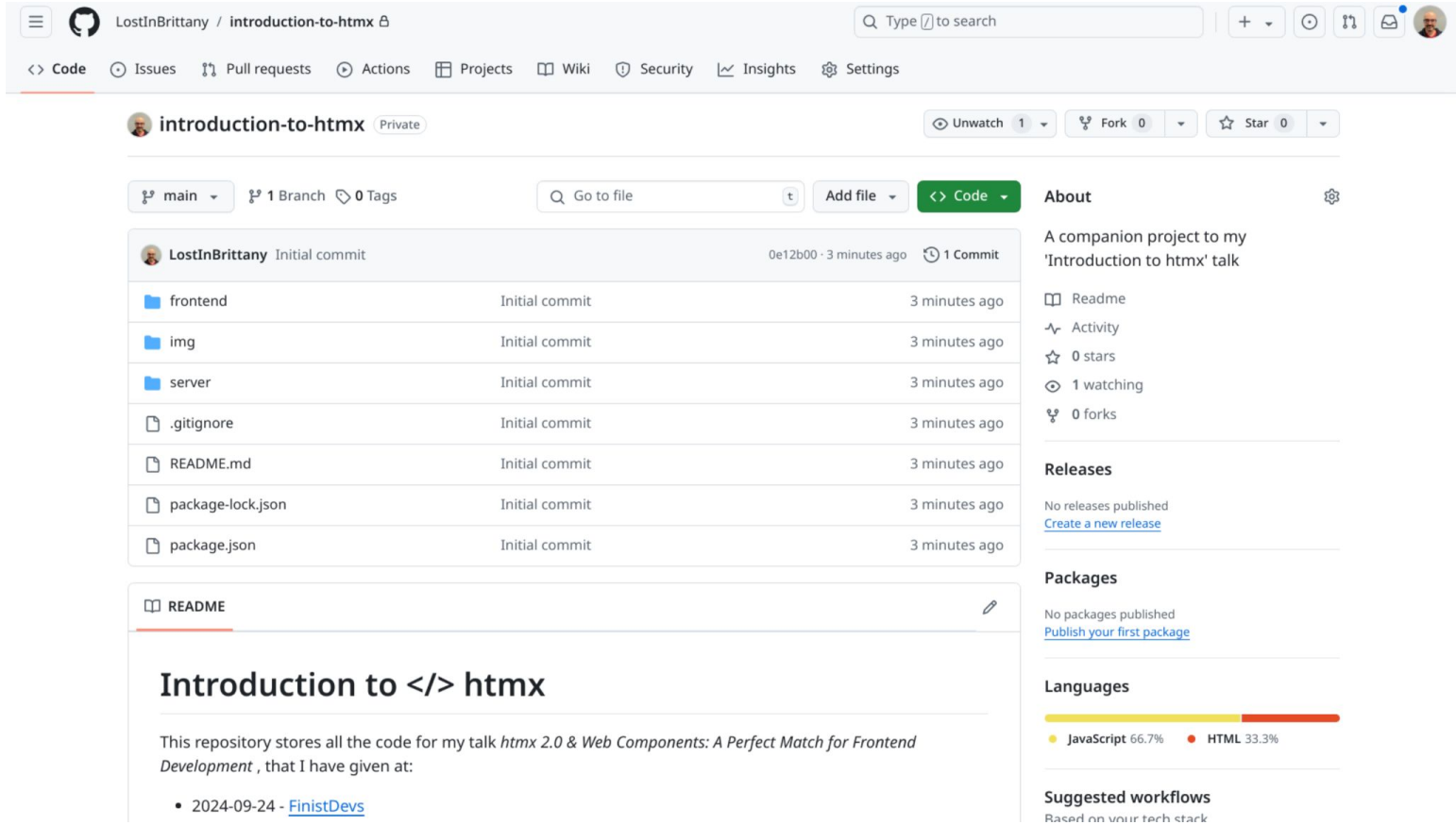
Manger de la pizza 📝

addTask

clever cloud

@LostInBrittany

Tout le code est disponible



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

clever cloud

@LostInBrittany

Des démos de htmx

Du `Hello World` à la `To-do List`

clever cloud

@LostInBrittany

What the heck are web component?

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

</>

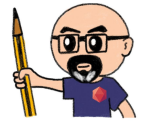
CUSTOM ELEMENTS

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>
```

clever cloud

@LostInBrittany

Shadow DOM

To encapsulate subtree and style in an element

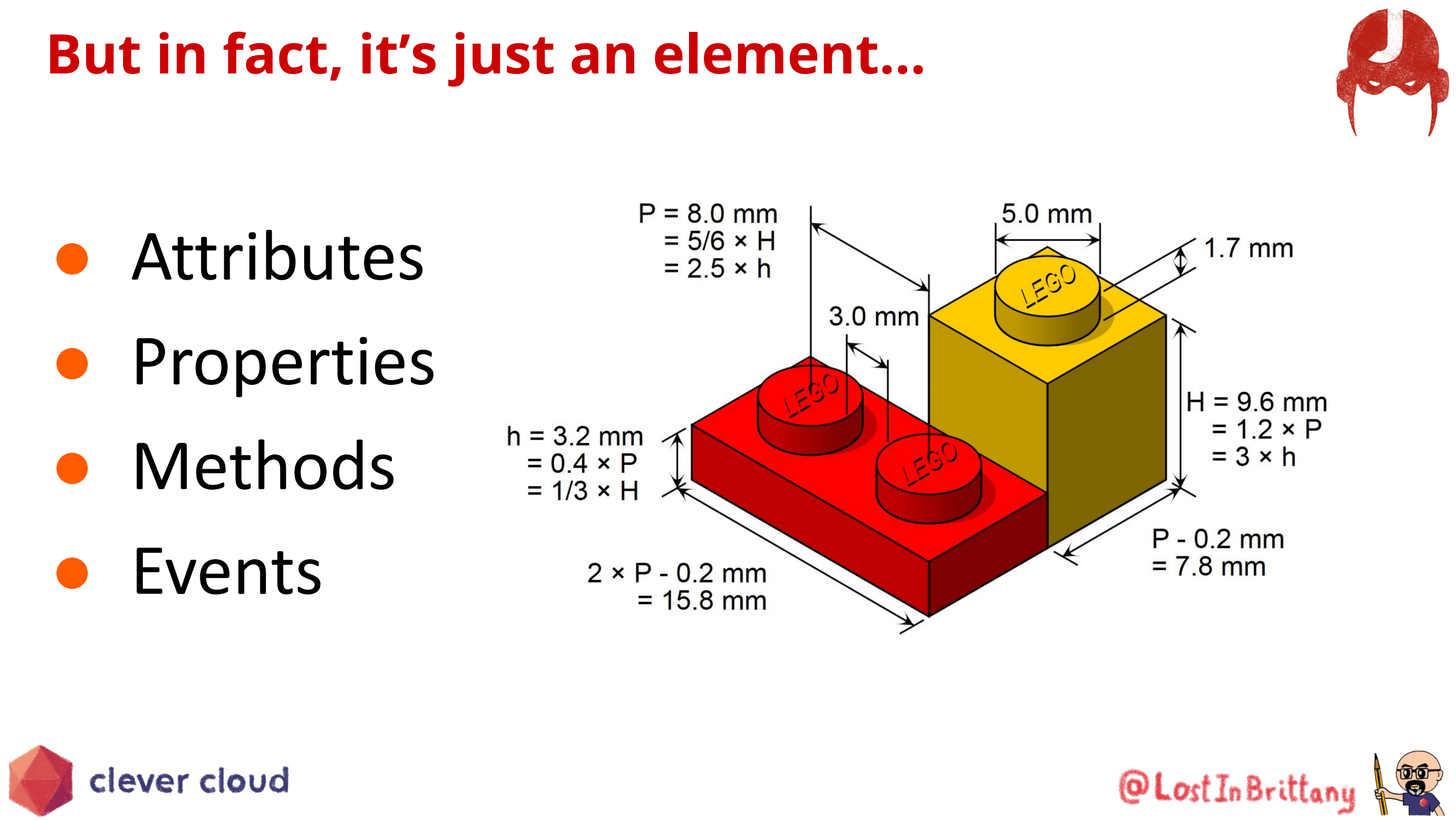

```
<button>Hello, world!</button>
<script>
var host = document.querySelector('button');
const shadowRoot = host.attachShadow({mode:'open'});
shadowRoot.textContent = 'こんにちは、影の世界!';
</script>
```

Template

To have clonable document template

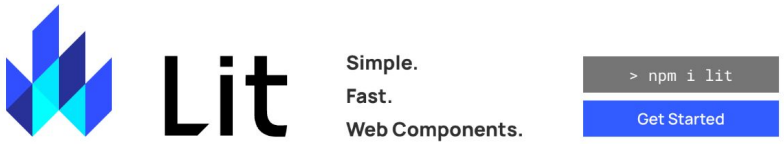
```
<template id="mytemplate">
  <img src="" alt="great image">
  <div class="comment"></div>
</template>

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);
```

But in fact, it's just an element…



- Attributes
- Properties
- Methods
- Events

Simple. Fast. Web Components



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

Modern lightweight web components

Lit

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

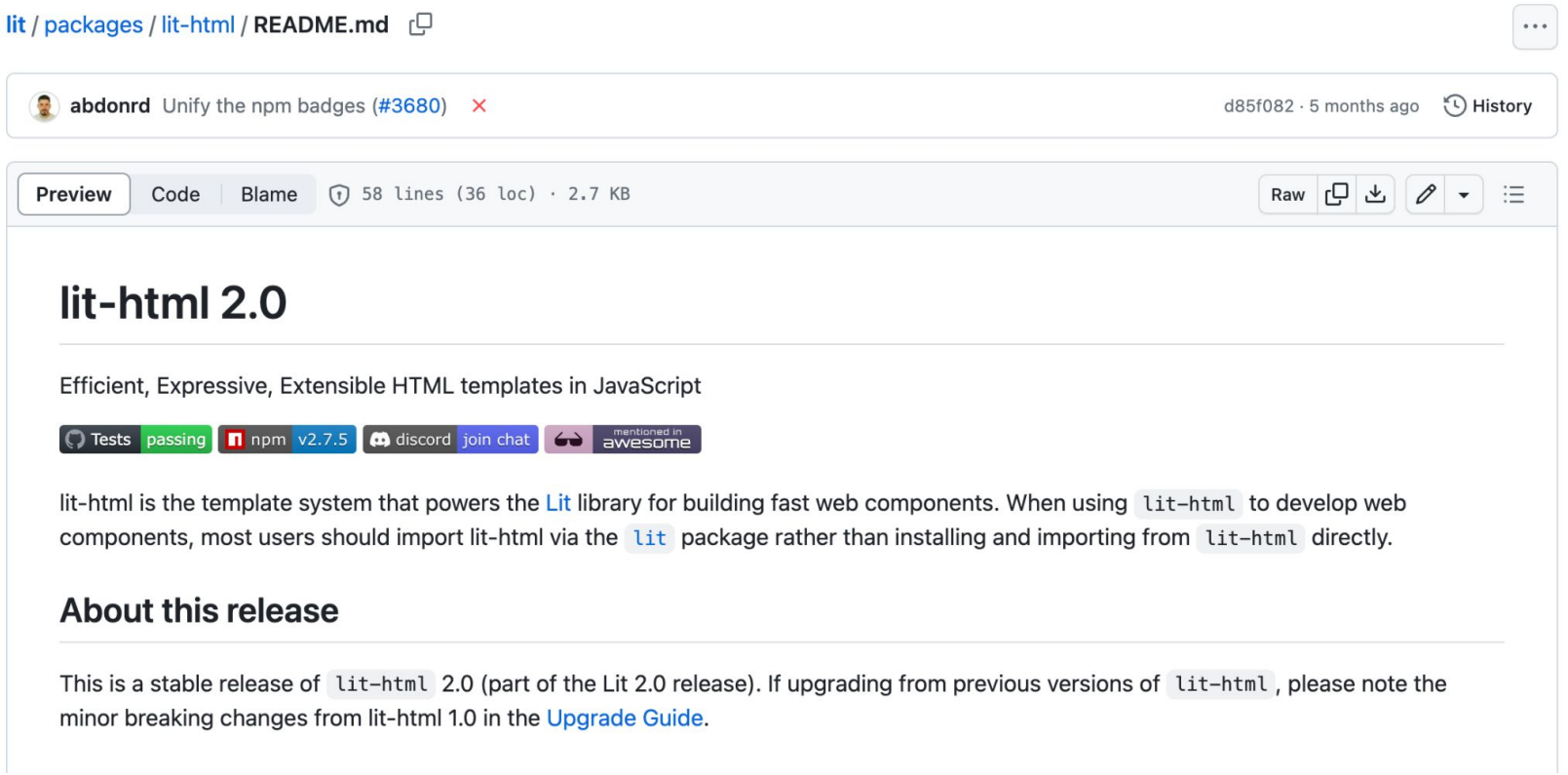
```
import { LitElement, html } from 'lit-element';

// 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

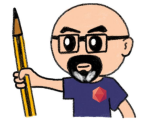
Based on lit-html



An efficient, expressive, extensible HTML templating library for JavaScript

clever cloud

@LostInBrittany



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

clever cloud

@LostInBrittany

lit-html Templates

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

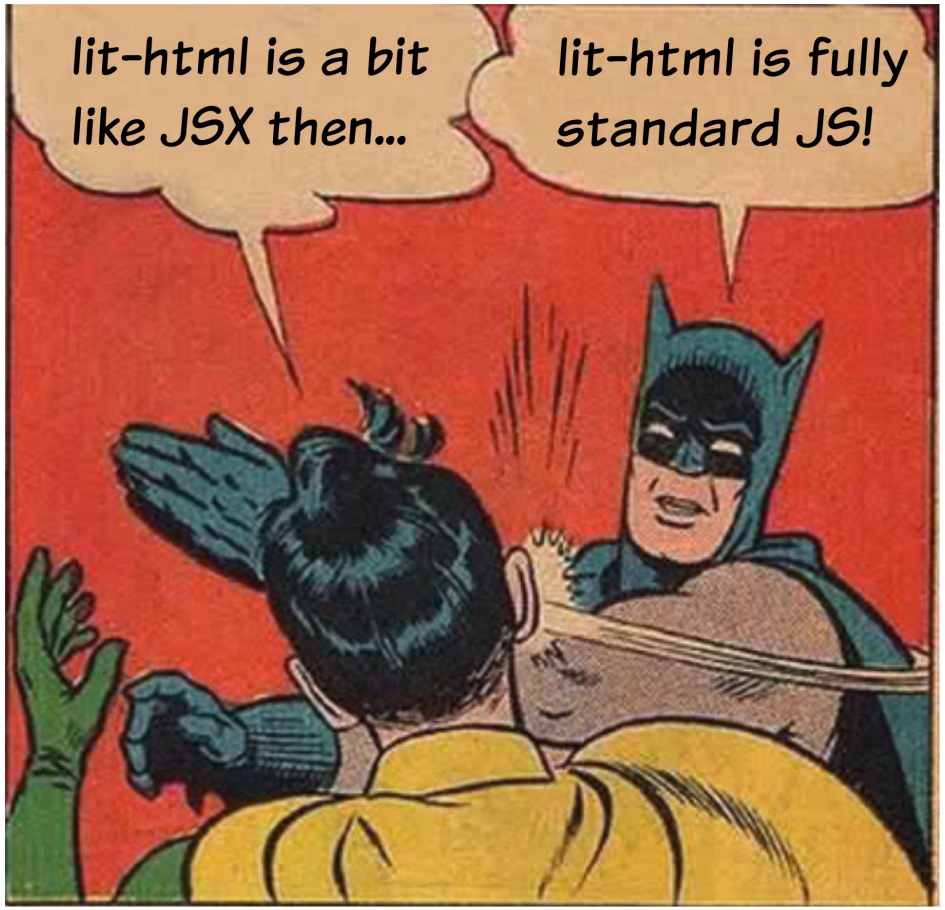
Lazily rendered

Generates a TemplateResult

clever cloud

@LostInBrittany

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 élément lits sont comme n'importe quel tag

clever cloud

@LostInBrittany

Une petit démo de htmx + Lit

Faisons un compteur : `my-lit-counter`

clever cloud

@LostInBrittany

That's all, folks!

Thank you all!

clever cloud

@LostInBrittany