



{RIVIERADEV}

# WebComponents in 2023: A Complete Exploration

Horacio Gonzalez

2023-07-10



OVHcloud



@LostInBrittany

# Who are we?

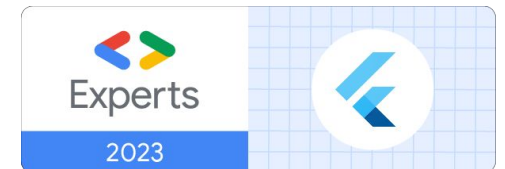
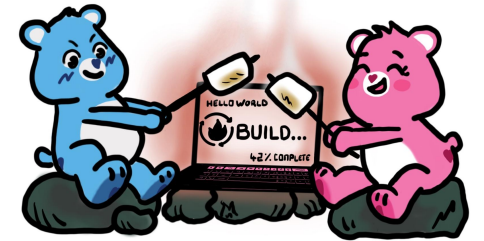
Introducing myself and  
introducing OVHcloud



# Horacio Gonzalez

@LostInBrittany

Spaniard Lost in Brittany





**Web Cloud & Telecom**



**Private Cloud**



**Public Cloud**



**Storage**



**Network & Security**



**30 Data Centers**  
in 12 locations



**34 Points of Presence**  
on a 20 TBPS Bandwidth Network



**2200 Employees**  
worldwide



**115K Private Cloud**  
VMS running



**300K Public Cloud**  
instances running



**380K Physical Servers**  
running in our data centers



**1 Million+ Servers**  
produced since 1999



**1.5 Million Customers**  
across 132 countries



**3.8 Million Websites**  
hosting



**1.5 Billion Euros Invested**  
since 2016



**P.U.E. 1.09**  
Energy efficiency indicator



**20+ Years in Business**  
Disrupting since 1999

# We want the code!



LostInBrittany / web-components-interop

Unwatch 1 Star 1 Fork 2

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

The git repository to support my 'A world outside Polymer' talk Edit

Manage topics

11 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

LostInBrittany Updating 2018-12	Latest commit 554da13 5 minutes ago
node_modules	Updating 2018-12 5 minutes ago
step-01	Updating 2018-12 5 minutes ago
step-02	Updating 2018-12 5 minutes ago
step-03	Updating 2018-12 5 minutes ago
step-04	Updating 2018-12 5 minutes ago
step-05	Updating 2018-12 5 minutes ago
step-06	Updating Slim to last version 8 months ago
README.md	Updating 10 months ago
package-lock.json	Updating 2018-12 5 minutes ago
package.json	Updating 2018-12 5 minutes ago

<https://github.com/LostInBrittany/web-components-in-2023/>

# What the heck are web component?

The 3 minutes context



# 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

# 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

```
<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>
```

Hello, world!



こんにちは、影の世界!

# Template



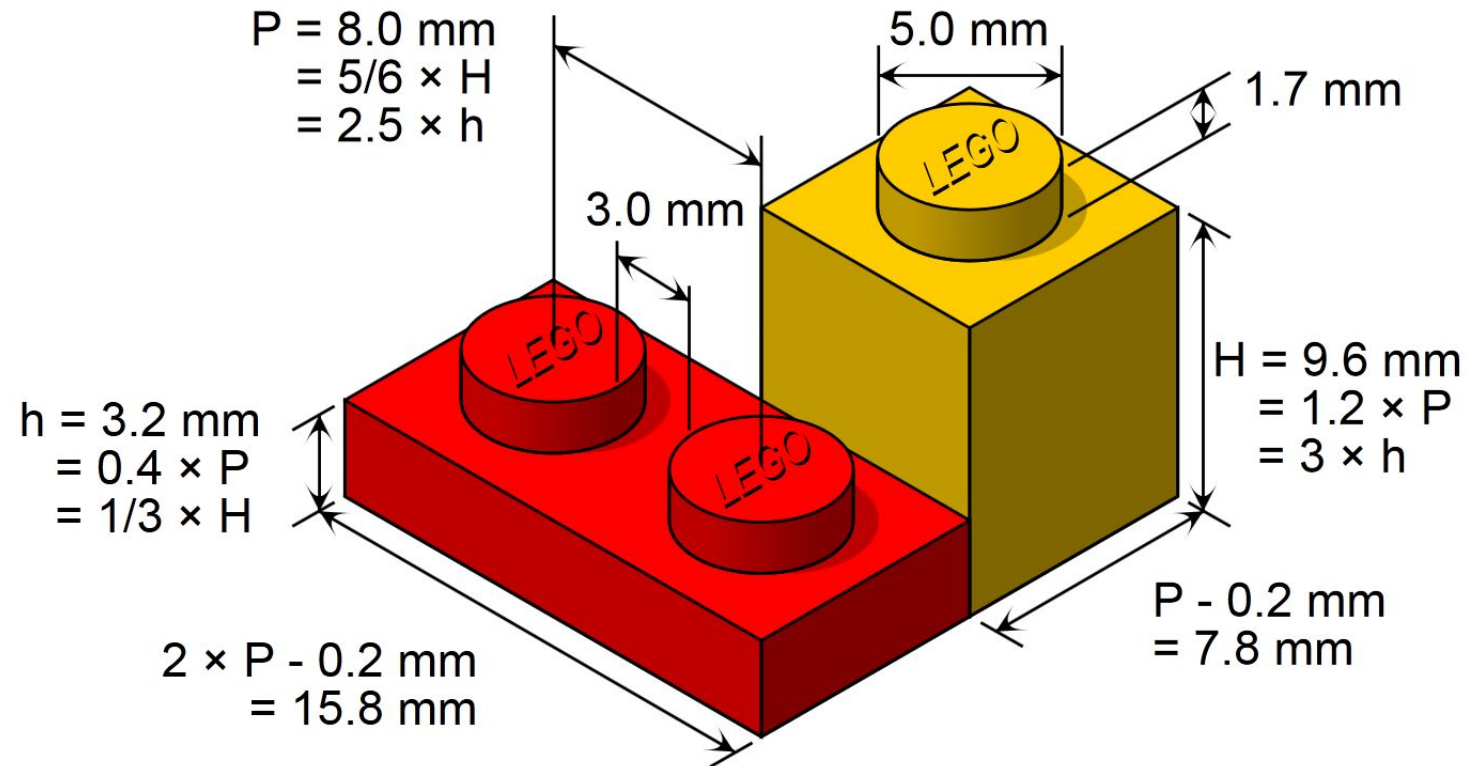
To have clonable document template

```
<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);
```

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

- Attributes
- Properties
- Methods
- Events



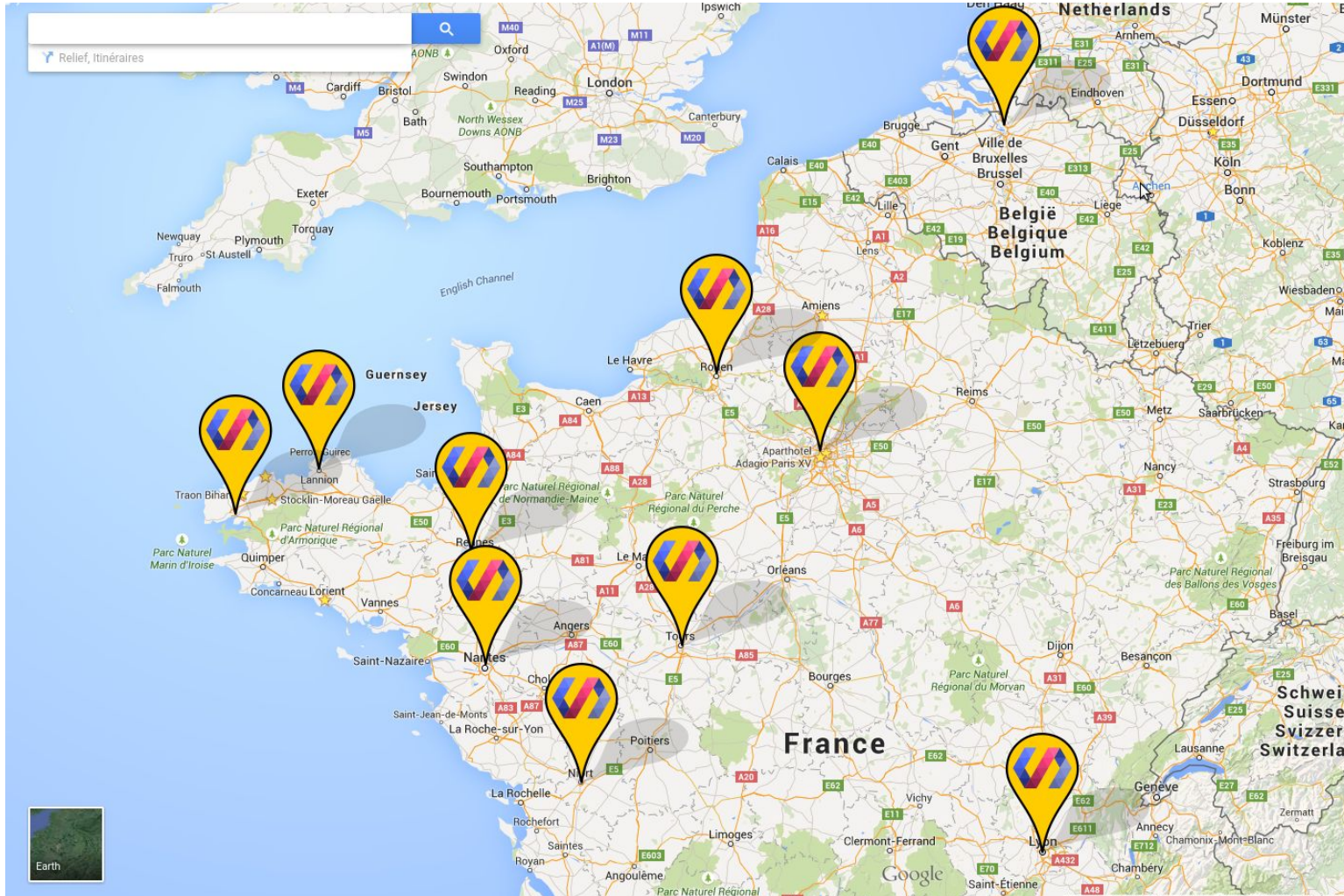
# Sometimes I feel a bit grumpy

The stories of the grumpy old speaker...





# On Polymer tour since 2014



# Web components == Revolution



[bu.edu](http://bu.edu)

# Building a world brick by brick



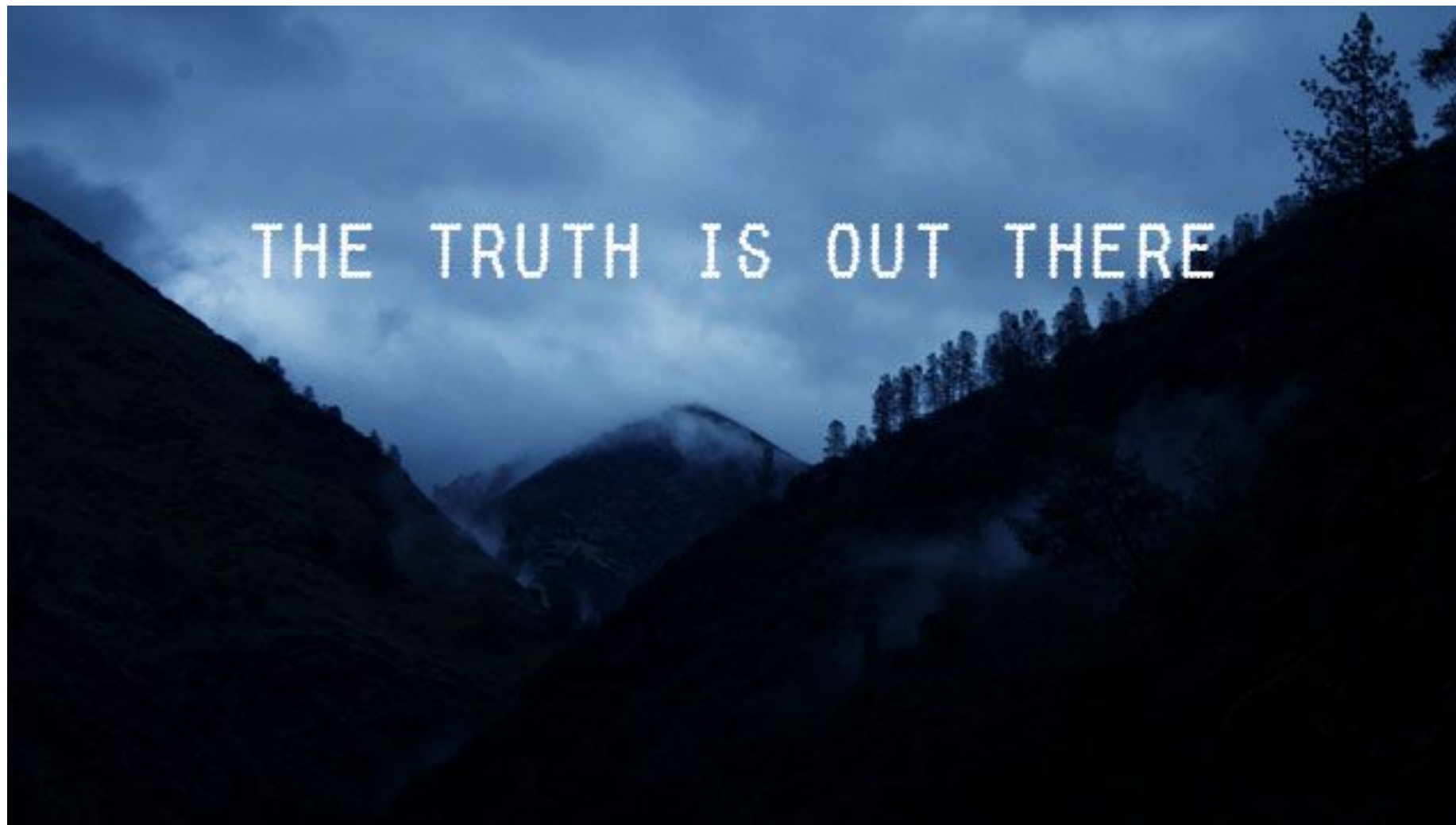
[BitRebels](#) [Brickset](#)

# Is the promise unfulfilled?

It's 2023 now, where is your revolution, dude?



# Is it a conspiracy?



# Am I only a dreamer?



# Well, revolution IS there



But it's a silent one..

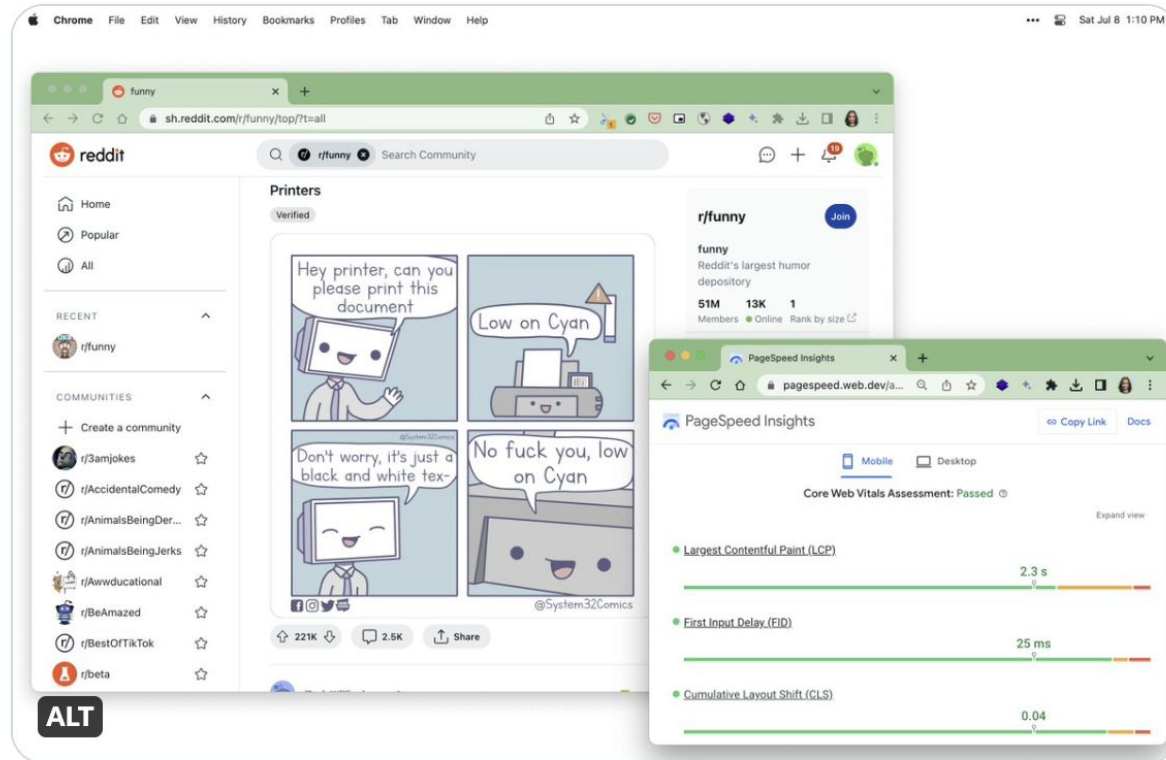
# I as looking for a great example



**Addy Osmani** ✓  
@addyosmani



The new Reddit UI is built with [@buildWithLit](#) & Web Components:  
[sh.reddit.com](#). It's fast (on Core Web Vitals) and a good experience so far.



10:27 PM · Jul 8, 2023 · 217.3K Views



# New Reddit runs on Web Components



reddit.com/r/reactjs

## What's your opinion on the new Reddit design ditching React in favor of Lit for mostly performance reasons?

Old : <https://i.imgur.com/ZaGmgPr.png>

New (you need to be logged out): <https://i.imgur.com/83jzoNx.png>

They've announced it here:  
[https://www.reddit.com/r/reddit/comments/11zso11/an\\_improved\\_web\\_experience/](https://www.reddit.com/r/reddit/comments/11zso11/an_improved_web_experience/)

I've confirmed the JS framework they've used with an extension. The performance boost is huge, and it's mostly frontend related. You can check by clicking around or using the chrome Lighthouse audit (the new one scores 99).

154 upvotes · 103 comments

u/StackOverflowTeams · Promoted

Stack Overflow for Teams - The knowledge-sharing platform for technology teams - Get it free!

try.stackoverflow.co

Sort by: Best · 103 comments

+ Add a Comment

r/scrivener

I've just wondered if there's any news on a major new update to...

20 upvotes · 52 comments

r/kde

Why the default panel UX fails and how to f...

self.kde

6 upvotes · 20 comments

r/feedthebeast

What is your opinion on the Replay Mod only supporting Fabric after...

4 comments

r/emacs

What's the future for smooth scrolling over images?

14 upvotes · 10 comments

r/skyrimmods

PSA: Papyrus Profiler - script load and a surprising result

64 upvotes · 36 comments

r/reactjs

Anyone want a mentor? I would like to help

157 upvotes · 92 comments

Elements Console >> 1 1 1

```
<div class="main w-100 mx-auto pb-xl col-start-1 col-end-1 row-start-1 row-end-auto xs:col-start-1 xs:col-end-6 s:col-start-4 s:col-end-10 m:col-end-11 l:col-start-4 l:col-end-13 xl:col-start-5 xl:col-end-14">  
  <shreddit-title title="What's your opinion on the new Reddit design ditching React in favor of Lit for mostly performance reasons? : r/reactjs">... </shreddit-title> == $0  
  <shreddit-post class="block pt-xs nd:pt-xs bg-[color:var(--shreddit-content-background)] box-border mb-xs nd:visible nd:pb-2xl" permalink="/r/reactjs/comments/12yxva6/whats_you_opinion_on_the_new_reddit_design/" content-href="https://www.reddit.com/r/reactjs/comments/12yxva6/whats_you_opinion_on_the_new_reddit_design/" " view-context="CommentsPage" vote-type comment-count="103" is-crosspostable is-desktop-viewport should-track-post-view-events user-logged-in created-timestamp="2023-04-25T22:12:50.744000+0000" domain="self.reactjs" id="t3_12yxva6" post-title="What's your opinion on the new Reddit design ditching React in favor of Lit for mostly performance reasons">... </shreddit-post>
```

1.1.col-start-4.1.col-end-13.xl.col-start-5.xl.col-end-14. shreddit-title

Styles Computed Layout Event Listeners DOM Breakpoints >>

Filter :hov .cls + -

```
element.style {  
}
```

# Often hidden in plain sight



# Vanilla Web Components



# Let's build a vanilla Web Component

```
README.md
```

## Web Components in 2023 - Vanilla Web Components

### A *Hello world* Custom Elements

We can define a new Custom Elements by extending the `HTMLElement` class:

File `src/hello-world.js`

```
class HelloWorld extends HTMLElement {  
  
  // This gets called when the HTML parser sees your tag  
  constructor() {  
    super(); // always call super() first in the ctor.  
    this.msg = 'Hello World!';  
  }  
  
  // Called when your element is inserted in the DOM or  
  // immediately after the constructor if it's already in the DOM  
  connectedCallback() {  
    this.innerHTML = `

${this.msg}</p>`;  
  }  
}  
  
customElements.define('hello-world', HelloWorld);


```



Using only HTML, CSS & JS, no library needed

# A very basic web component

```
class HelloWorld extends HTMLElement {  
  // This gets called when the HTML parser sees your tag  
  constructor() {  
    super(); // always call super() first in the ctor.  
    this.msg = 'Hello World!';  
  }  
  // Called when your element is inserted in the DOM or  
  // immediately after the constructor if it's already in the DOM  
  connectedCallback() {  
    this.innerHTML = `

${this.msg}</p>`;  
  }  
}  
customElements.define('hello-world', HelloWorld);


```



# Custom Elements:

- Let you define your own HTML tag with bundled JS behavior
- Trigger lifecycle callbacks
- Automatically “upgrade” your tag when inserted in the document

# Custom Elements don't:

- Scope CSS styles
  - Shadow DOM
- Scope JavaScript
  - ES2015
- “Reproject” children into `<slot>` elements
  - Shadow DOM

# Adding ShadowDOM

```
class HelloWithShadowdom extends HTMLElement {  
  // This gets called when the HTML parser sees your tag  
  constructor() {  
    super(); // always call super() first in the ctor.  
    this.msg = 'Hello World from inside the ShadowDOM!';  
    this.attachShadow({ mode: 'open' });  
  }  
  // Called when your element is inserted in the DOM or  
  // immediately after the constructor if it's already in the DOM  
  connectedCallback() {  
    this.shadowRoot.innerHTML = `

`${this.msg}`

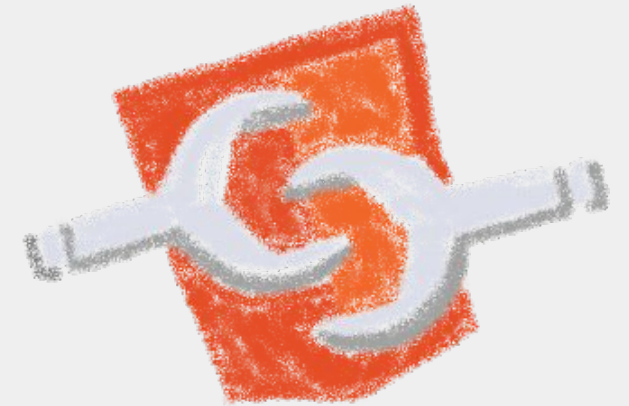
`;  
  }  
}  
customElements.define('hello-with-shadowdom', HelloWithShadowdom);
```



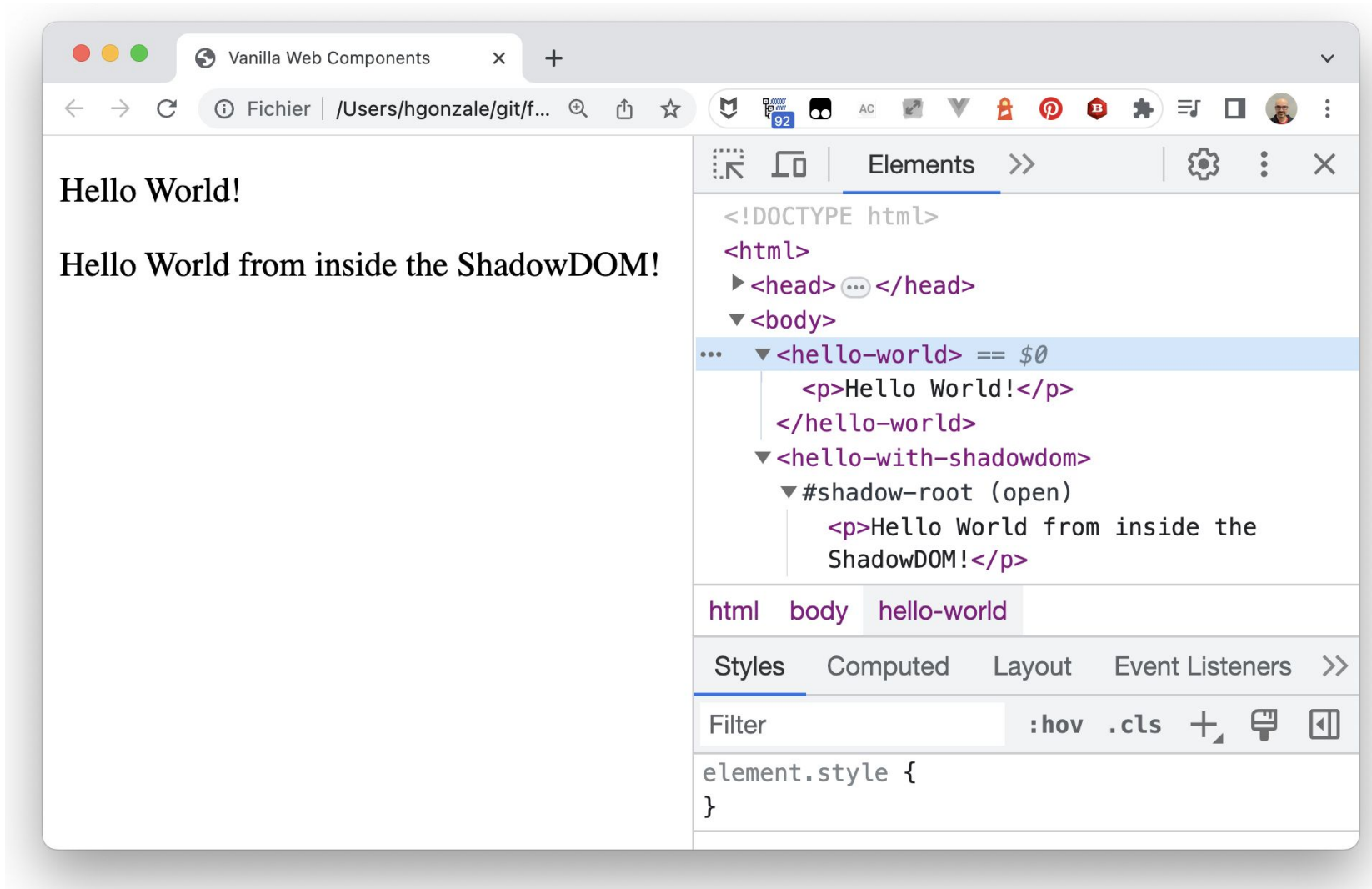


# Using web components

```
<!DOCTYPE html>
<html>
<head>
  <title>Vanilla Web Components</title>
  <script src="./hello-world.js"></script>
  <script src="./hello-with-shadowdom.js"></script>
</head>
<body>
  <hello-world></hello-world>
  <hello-with-shadowdom></hello-with-shadowdom>
</body>
</html>
```



# Using web components



The screenshot shows a web browser window with the title "Vanilla Web Components". The page content consists of two paragraphs: "Hello World!" and "Hello World from inside the ShadowDOM!". The browser's developer tools are open to the "Elements" panel, showing the following DOM tree:

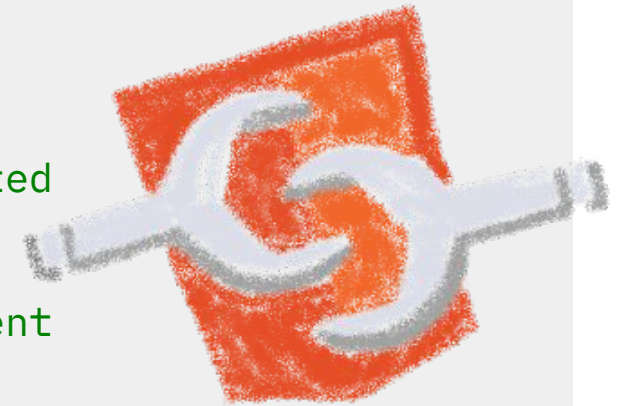
```
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <hello-world> == $0
      <p>Hello World!</p>
    </hello-world>
    <hello-with-shadowdom>
      #shadow-root (open)
        <p>Hello World from inside the ShadowDOM!</p>
    </hello-with-shadowdom>
  </body>
</html>
```

The "hello-world" element is selected, and the breadcrumb below the DOM tree shows the path: `html > body > hello-world`. The "Styles" panel is also visible, showing a filter and a CSS rule:

```
element.style {
}
```

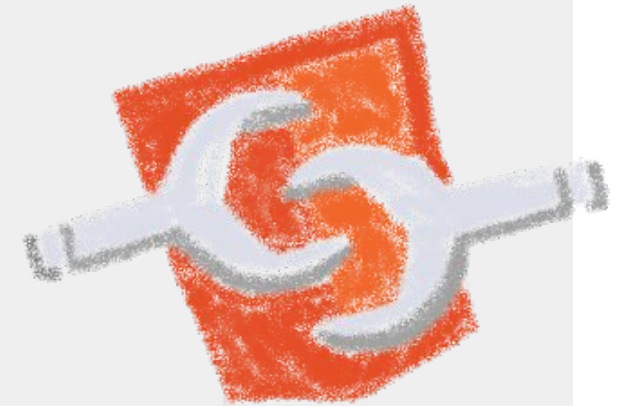
# Lifecycle callbacks

```
class MyElementLifecycle extends HTMLElement {
  constructor() {
    // Called when an instance of the element is created or upgraded
    super(); // always call super() first in the ctor.
  }
  static get observedAttributes() {
    // Tells the element which attributes to observe for changes
    return [];
  }
  connectedCallback() {
    // Called every time the element is inserted into the DOM
  }
  disconnectedCallback() {
    // Called every time the element is removed from the DOM.
  }
  attributeChangedCallback(attrName, oldVal, newVal) {
    // Called when an attribute was added, removed, or updated
  }
  adoptedCallback() {
    // Called if the element has been moved into a new document
  }
}
```



# my-vanilla-counter element

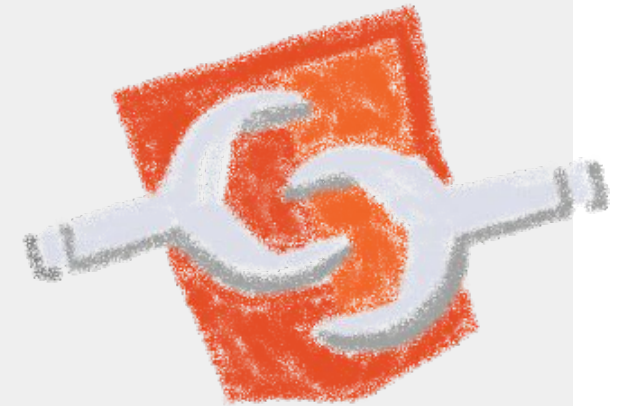
```
class MyVanillaCounter extends HTMLElement {  
  constructor() {  
    super();  
    this._counter = 0;  
    this.attachShadow({ mode: 'open' });  
  }  
  connectedCallback() {  
    this.render();  
    this.display();  
  }  
  static get observedAttributes() { return [ 'counter' ] }  
  // We reflect attribute changes into property changes  
  attributeChangedCallback(attr, oldVal, newVal) {  
    if (oldVal !== newVal) {  
      this[attr] = newVal;  
    }  
  }  
}
```



# my-counter custom element

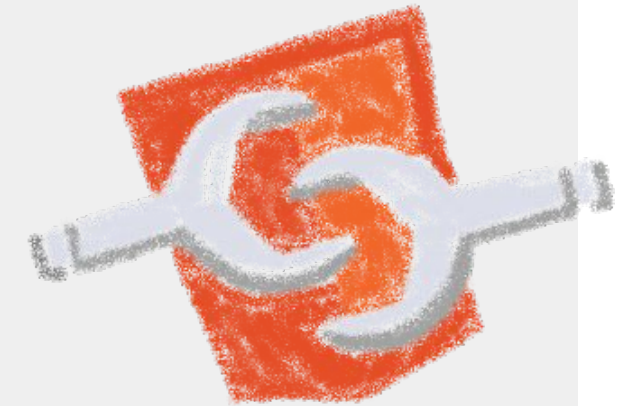
```
// Getter and setters for counter
get counter() { return this._counter; }
set counter(value) {
  if (value !== this._counter) {
    this._counter = Number.parseInt(value);
    this.setAttribute('counter', value);
    this.display();
  }
}

increment() {
  this.counter = this.counter + 1;
  this.dispatchEvent(new CustomEvent('increased',
    {detail: {counter: this.counter}}));
}
```



# my-counter custom element

```
render() {  
  let container = document.createElement('div');  
  container.style.display = 'flex';  
  ...  
  this.style.fontSize = '5rem';  
}  
  
display() {  
  this.output.innerHTML = `${this.counter}`;  
}  
  
}  
customElements.define('my-vanilla-counter', MyVanillaCounter);
```



# my-counter-with-templates

```
let template = `  
  <style>  
    ...  
  </style>  
  <div class="container">  
    <div id="icon">  
        
    </div>  
    <div id="value">  
      0  
    </div>  
  </div>  
`;  
;
```

# my-counter-with-templates

```
render() {
  let templ = document.createElement('template');
  templ.innerHTML = template;

  this.shadowRoot.appendChild(templ.content.cloneNode(true));

  let button = this.shadowRoot.getElementById('icon');
  button.addEventListener('click', this.increment.bind(this));
}

display() {
  console.log(this.shadowRoot.getElementById('value'))
  this.shadowRoot.getElementById('value').innerHTML =
    `${this.counter}`;
}
```



# Coding my-counter



web-components-in-2023 / step-02 /

Add file ▾

**LostInBrittany** Step 02 - my-vanilla-counter' 4606639 · 4 hours ago History

Name	Last commit message	Last commit date
..		
img	Step 02 - my-vanilla-counter'	4 hours ago
src	Step 02 - my-vanilla-counter'	4 hours ago
README.md	Step 02 - my-vanilla-counter'	4 hours ago
index.html	Step 02 - my-vanilla-counter'	4 hours ago

README.md

## Web Components in 2023 - Vanilla **my-counter** Element

For most of the examples in this workshop, we are going to build the same element in very different ways. The element is a very simple counter, that can be increased using a button.

# my-counter custom element



# Why those libs?

Why people don't use vanilla?



**Lit**



**stencil**



**hybrids**

# Web component standard is low level



At it should be!

# Standard == basic bricks

Standard exposes an API to:

- Define elements
- Encapsulate DOM



# Libraries are helpers



They give you higher-level primitives

# Different high-level primitives



Each one tailored to a use

# Sharing the same base



High-performant, low-level, in-the-platform  
web components standard



# Libraries aren't a failure of standard

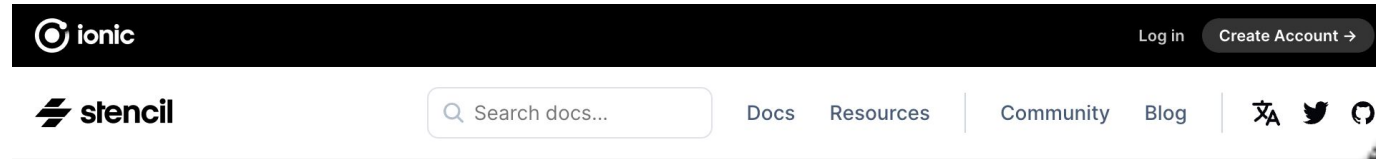


They happen by design



**A library for building reusable,  
scalable component libraries**

# Not another library



Stencil 4 is here! Read all about what's new in latest major release [Read the Blog →](#)

## Build. Customize. Distribute. **Adopt.**

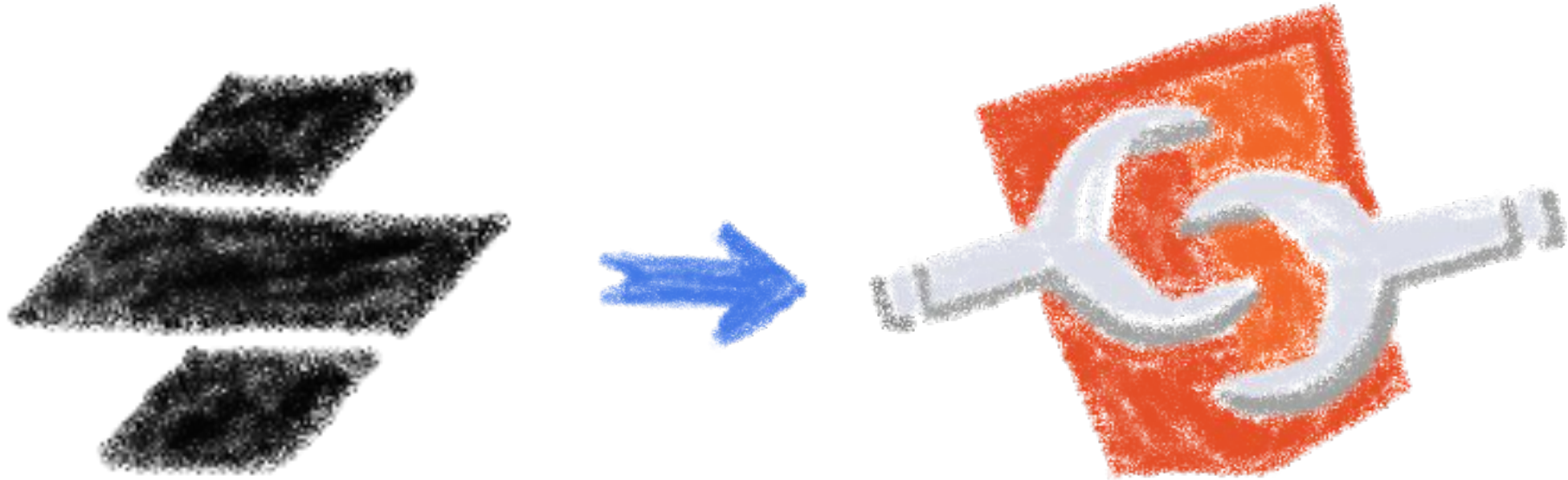
Stencil is a library for building reusable, scalable component libraries.  
Generate small, blazing fast Web Components that run everywhere.

[Get started →](#)

`npm init stencil` 

## A Web Component toolchain

# A build time tool



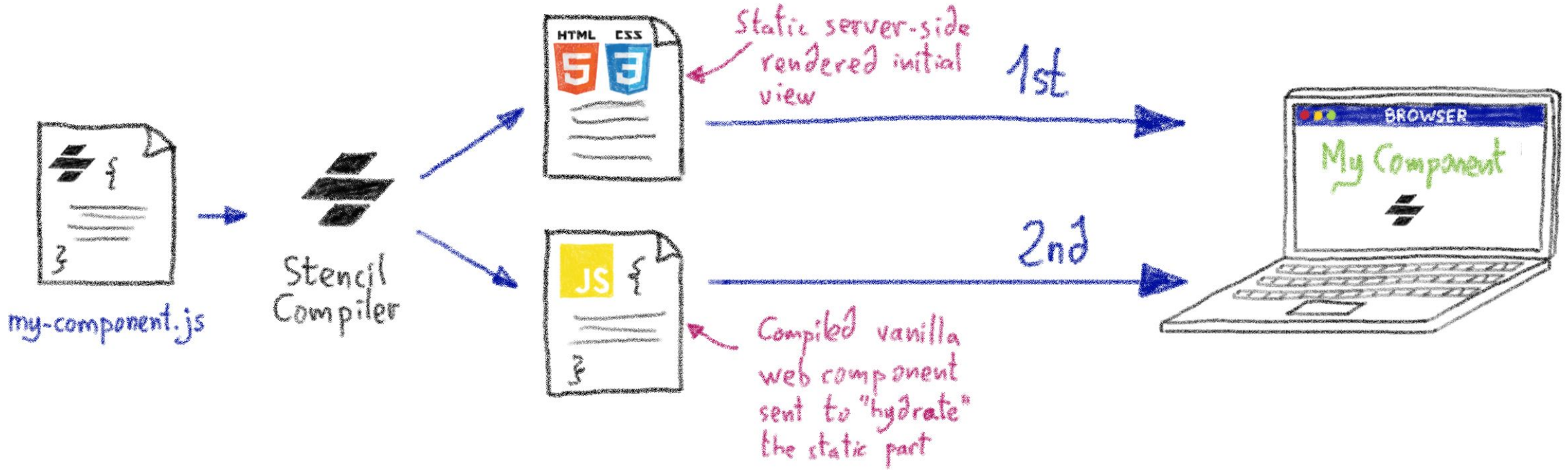
To generate standard web components

# Fully featured

- Web Component-based
- Asynchronous rendering pipeline
- TypeScript support
- Reactive Data Binding
- Component pre-rendering
- Simple component lazy-loading
- JSX support
- Dependency-free components



# And the cherry on the cake



## Server-Side Rendering

# Stencil leverages the web platform



Stencil doesn't fight the web platform. It embraces it.



## Simple

With intentionally small tooling, a tiny API, and zero configuration, Stencil gets out of the way and lets you focus on your work.



## Lightweight

A tiny runtime, pre-rendering, and the raw power of native Web Components make Stencil one of the fastest compilers around.

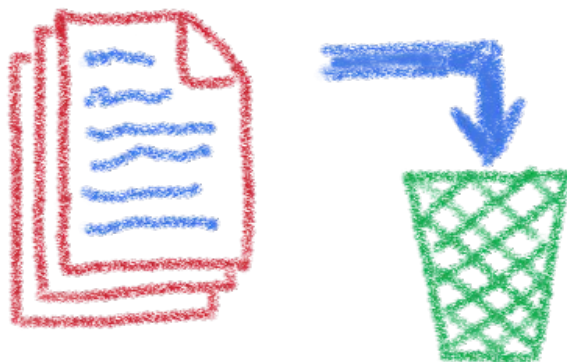


## Future proof

Build cross-framework components and design systems on open web standards, and break free of Framework Churn.

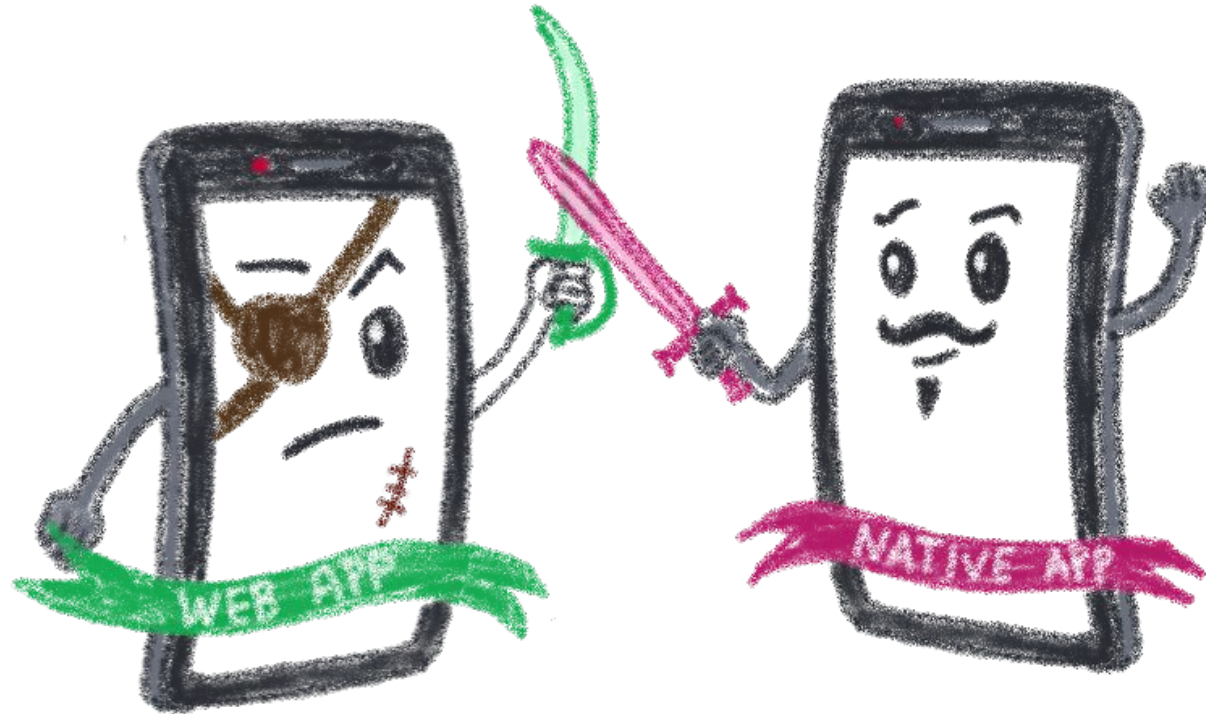
# The Stencil story

A company tired of putting good code in the bin





# Once upon a time there was a fight



Between native apps and web app on mobile

# A quest to the perfect solution



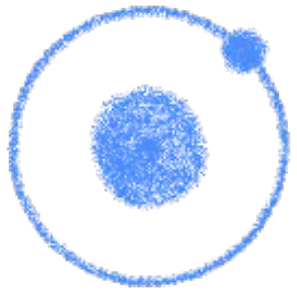
Hybrid apps, leveraging on web technologies

# A company wanted to do it well



The perfect technology for mobile web and hybrid apps

# The time is 2013

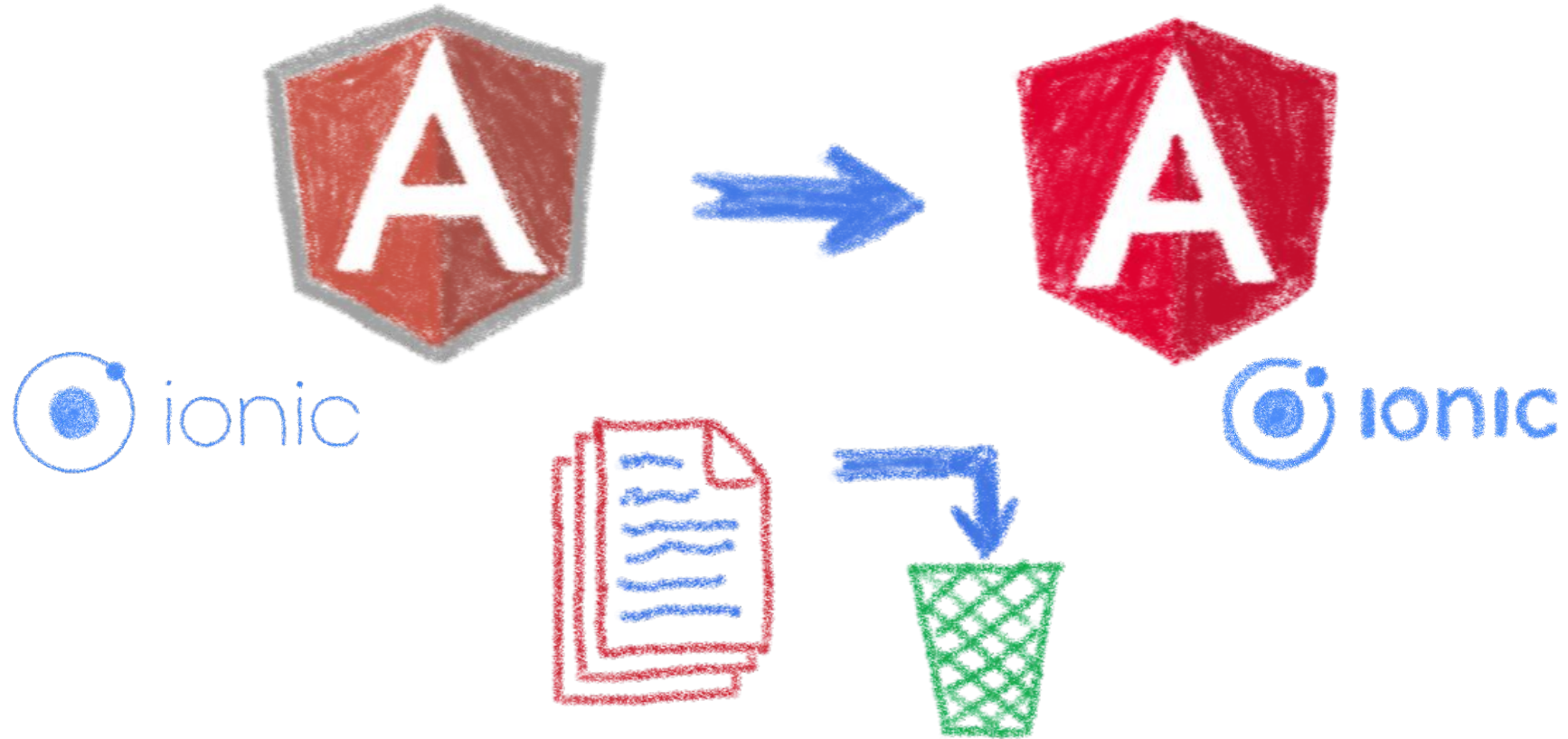


ionic



So what technology would you use?

# Really soon after launch...



*Hey folks, we are killing AngularJS!*

# What did Ionic people do?



Let's put everything in the trash bin and begin anew

# But times have changed...

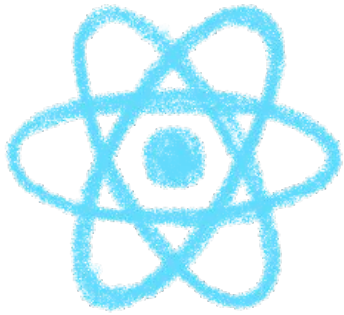


Knockout.



In 2013 Angular JS was the prom queen

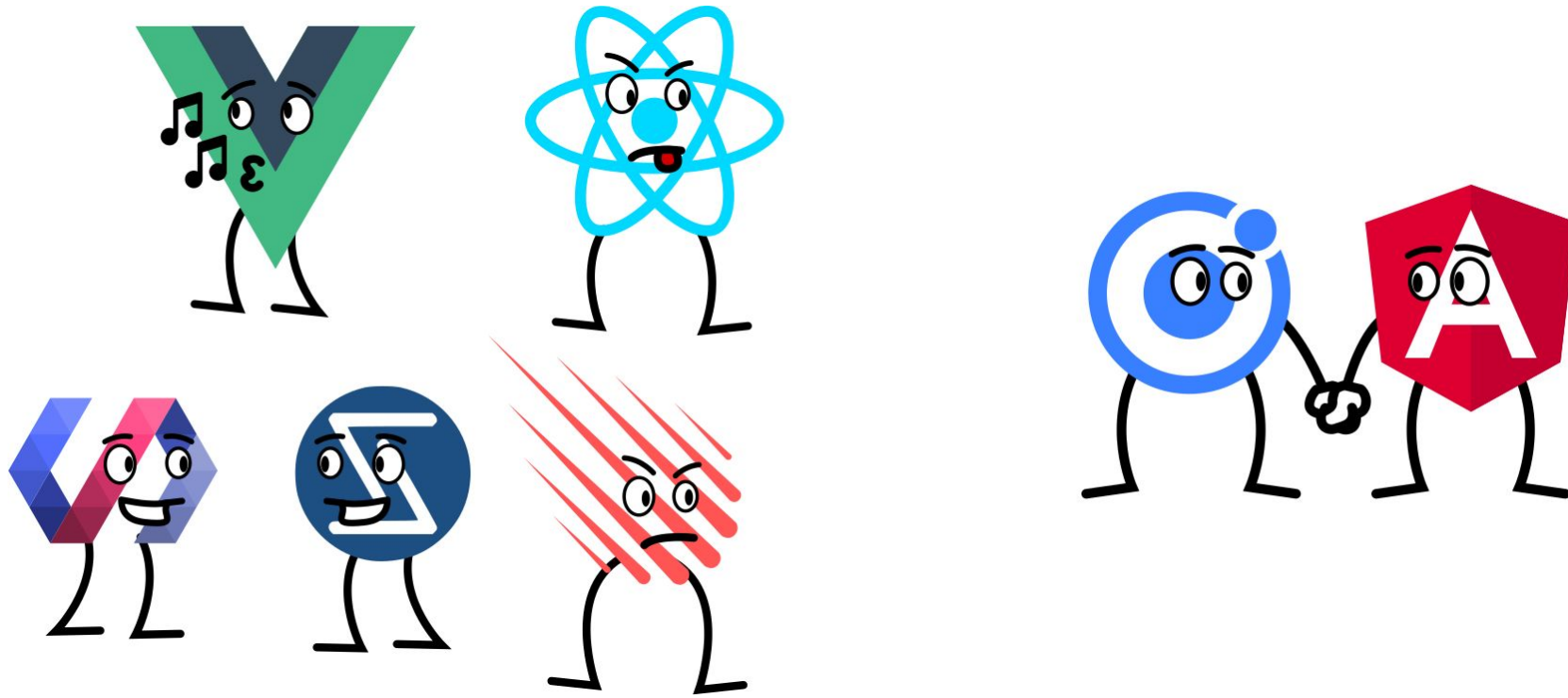
# Times have changed...



In 2017 Angular is only one more in the clique

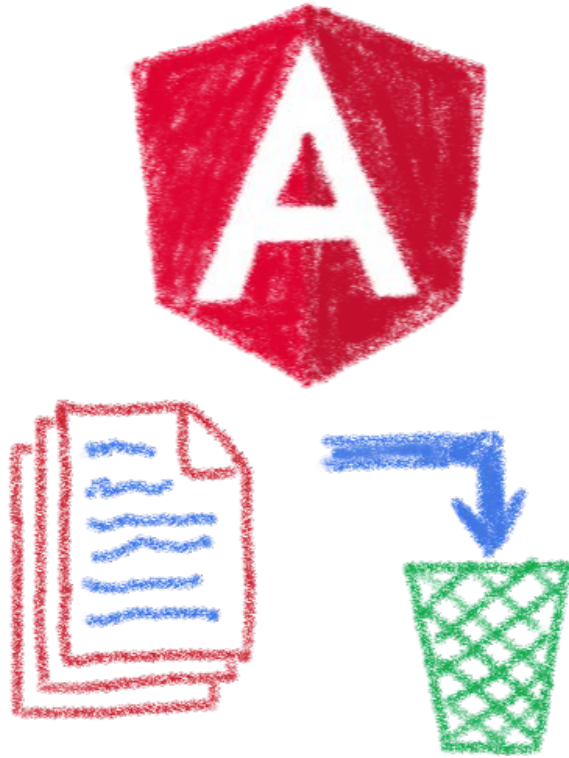


# Angular limits adoption of Ionic



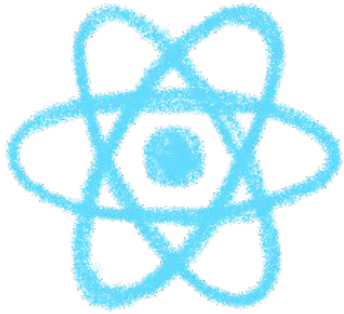
Devs and companies are very vocal about JS Frameworks

# What did Ionic people do?



Let's put everything in the trash bin and begin anew...  
But on which framework?

# What about web components?



A nice solution for Ionic problems:  
Any framework, even no framework at all!

# But what Web Component library?



hybrids



snuggsi ツ



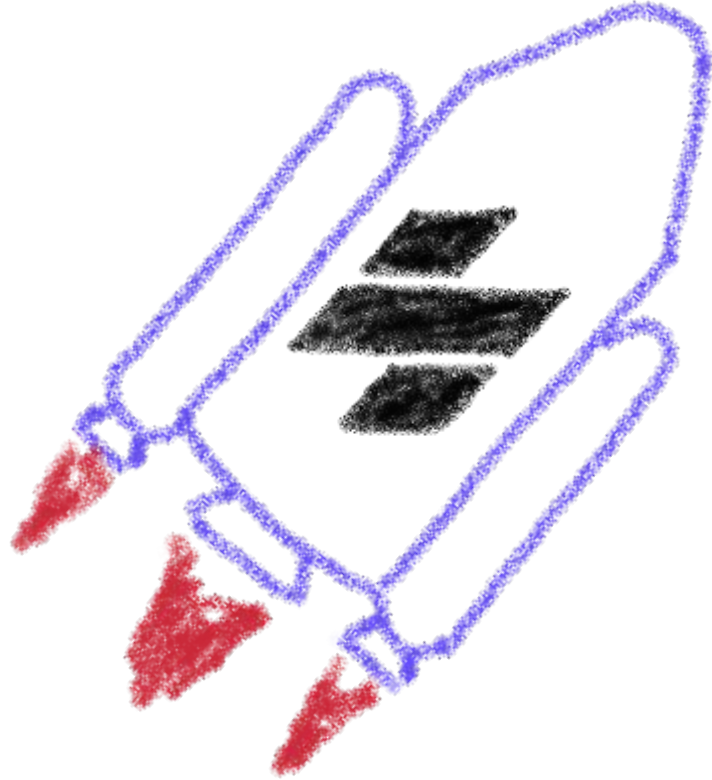
SkateJS



slim.js

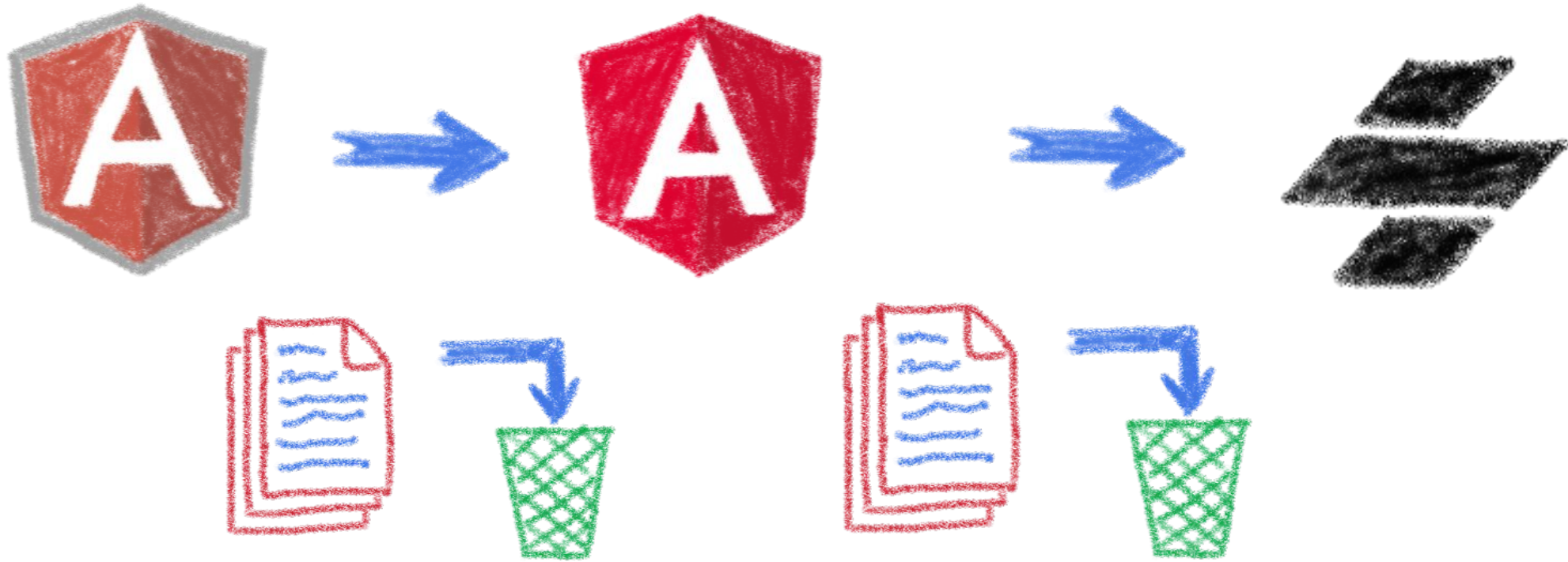
There were so many of them!

# Let's do something different



A fully featured web component toolchain  
With all the bells and whistles!

# Ionic rewrote all their code again



From Ionic 4 is fully based on Stencil

# Now Ionic works on any framework



Or without framework at all

# And we have Stencil



To use it in any of our projects



# Hands on Stencil

Simply use npm init

```
npm init stencil
```

Choose the type of project to start

```
? Select a starter project.
```

```
Starters marked as [community] are developed by the Stencil Community,  
rather than Ionic. For more information on the Stencil Community, please see  
https://github.com/stencil-community > - Use arrow-keys. Return to submit.
```

```
> component           Collection of web components that can be used anywhere  
  app [community]     Minimal starter for building a Stencil app or website  
  ionic-pwa [community] Ionic PWA starter with tabs layout and routes
```

# Hands on Stencil

And the project is initialized in some seconds!

```
✓ Pick a starter > component
✓ Project name > my-stencil-counter
✓ All setup in 17 ms

$ npm start
  Starts the development server.
$ npm run build
  Builds your components/app in production mode.
$ npm test
  Starts the test runner.

We suggest that you begin by typing:
$ cd my-stencil-counter
$ npm install
$ npm start

Happy coding! 🎈
```

# Let's look at the code



```
1 import { Component, Prop, h } from '@stencil/core';
2 import { format } from '../utils/utils';
3
4 @Component({
5   tag: 'my-component',
6   styleUrls: 'my-component.css',
7   shadow: true
8 })
9 export class MyComponent {
10  /**
11   * The first name
12   */
13  @Prop() first: string;
14
15  /**
16   * The middle name
17   */
18  @Prop() middle: string;
19
20  /**
21   * The last name
22   */
23  @Prop() last: string;
24
25  private getText(): string {
26    return format(this.first, this.middle, this.last);
27  }
28
29  render() {
30    return <div>Hello, World! I'm {this.getText()}</div>;
31  }
32 }
33
```

# Some concepts

```
import { Component, Prop, h } from '@stencil/core';
import { format } from '../../utils/utils';

@Component({
  tag: 'my-component',
  styleUrls: 'my-component.css',
  shadow: true
})
export class MyComponent {

  @Prop() first: string;
```

## Decorators

# Some concepts

```
@Prop() first: string;  
  
@Prop() middle: string;  
  
@Prop() last: string;  
  
@State() nickname: string;
```

## Properties and States

# Some concepts

```
render() {  
  return <div>Hello, World! I'm {this.getText()}</div>;  
}
```

## Asynchronous rendering using JSX

# Some concepts

```
@Prop() value: number;

@Watch(value)
valueChanged(newValue: boolean, oldValue: boolean) {
  console.log(`The new value is ${newValue}, it was ${oldValue} before`);
}
```

## Watch

# Some concepts

```
@Event() actionCompleted: EventEmitter;

someAction(message: String) {
  this.actionCompleted.emit(message);
}
```

```
@Listen('actionCompleted')
actionCompletedHandler(event: CustomEvent) {
  console.log('Received the custom actionCompleted event: ', event.detail);
}
```

Emitting events



# Some concepts

```
@Method()  
async sayHello() {  
  this.hello = true;  
}  
  
render() {  
  return (  
    <Host>  
      <h2>{ this.hello ? `Hello sthlm.js` : ''}</h2>  
    </Host>  
  );  
}
```

## Asynchronous public methods

# Some concepts

```
@Component({  
  tag: 'my-component',  
  styleUrls: 'my-component.css',  
  shadow: true  
})  
export class MyComponent {
```

## Optional Shadow DOM

# Coding my-stencil-counter



web-components-in-2023 / step-03 /

Add file ▾

**LostInBrittany** Step 03 - Stencil my-counter element bed1385 · 1 minute ago History

Name	Last commit message	Last commit date
..		
img	Step 03 - Stencil my-counter element	1 minute ago
my-stencil-counter	Step 03 - Stencil my-counter element	1 minute ago
README.md	Step 03 - Stencil my-counter element	1 minute ago

README.md

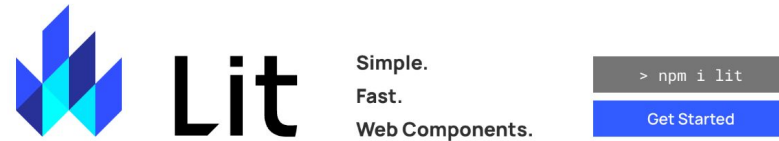
## Web Components in 2023 - Stencil `my-counter` Element

Now we are going to use Stencil to create another version of our counter, `my-stencil-counter`.

To create Stencil components, we need to use `nodejs` and `npm`. If you don't have them in your computer, the easiest way would be to use the [GitPod workspace](#), that has all the required tooling.



# 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

# Do you remember Polymer



The first Web Component library

# It is deprecated...

## Polymer Library

The Polymer library is our original web components library. Version 3.0 of the Polymer library brings web components into the mainstream, embracing JavaScript modules and npm.

Update your apps and elements for seamless interop with popular frameworks and tools and easy access to the full JavaScript ecosystem.

WHAT'S NEW

UPGRADE GUIDE

## Welcome to the future...

Starting a new project? Try our next-gen products, headed for release in the coming months:

### LitElement

An ultra-light, highly performant custom element base class with a simple but expressive API.

### PWA Starter Kit

Your one-stop shop for building Progressive Web Apps that make the most of the modern web.

### Material Web Components

Pixel-perfect realizations of Google's Material Design spec that work anywhere on the web.

# And that means good news!

# Let's try to see clearer



Let's dive into Polymer history...

# A tool built for another paradigm



No web component support on browsers  
No React, Angular or Vue innovations

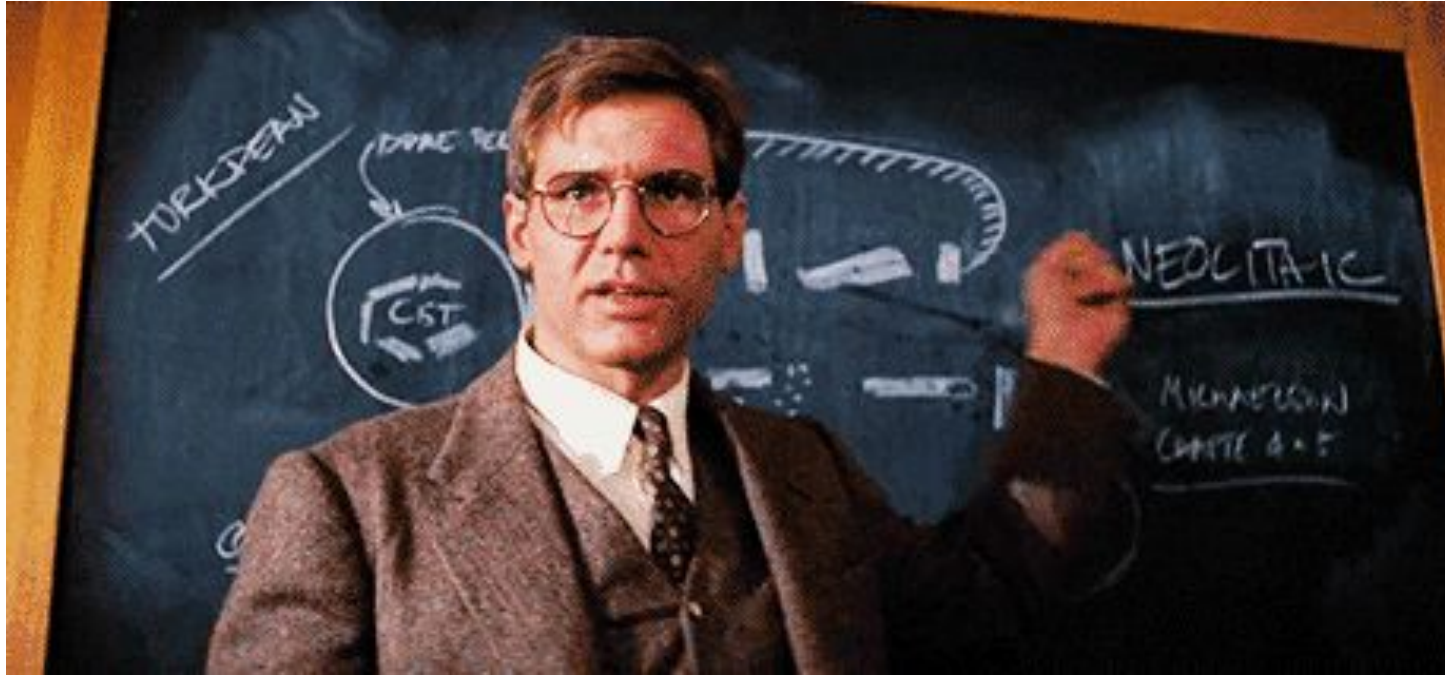


# No so well suited for the current one



The current platform is way more powerful  
The state of art has evolved

# Let's learn from its lessons



The current platform is way more powerful  
The state of art has evolved

# And let it rest...



There will have no Polymer 4...

# So Polymer as we know it is dead...



But the Polymer Project is indeed alive!

# But I have invested so much on it!



## What to do?

# That's why web components are top



You can keep using all your Polymer components and create the new ones with a new library... And it simply works!

# Born from the Polymer team



For the new web paradigm

# 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





# Based on lit-html



lit / packages / lit-html / README.md

abdonrd Unify the npm badges (#3680) 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

Tests passing npm v2.7.5 discord join chat mentioned in awesome

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]}!`
```

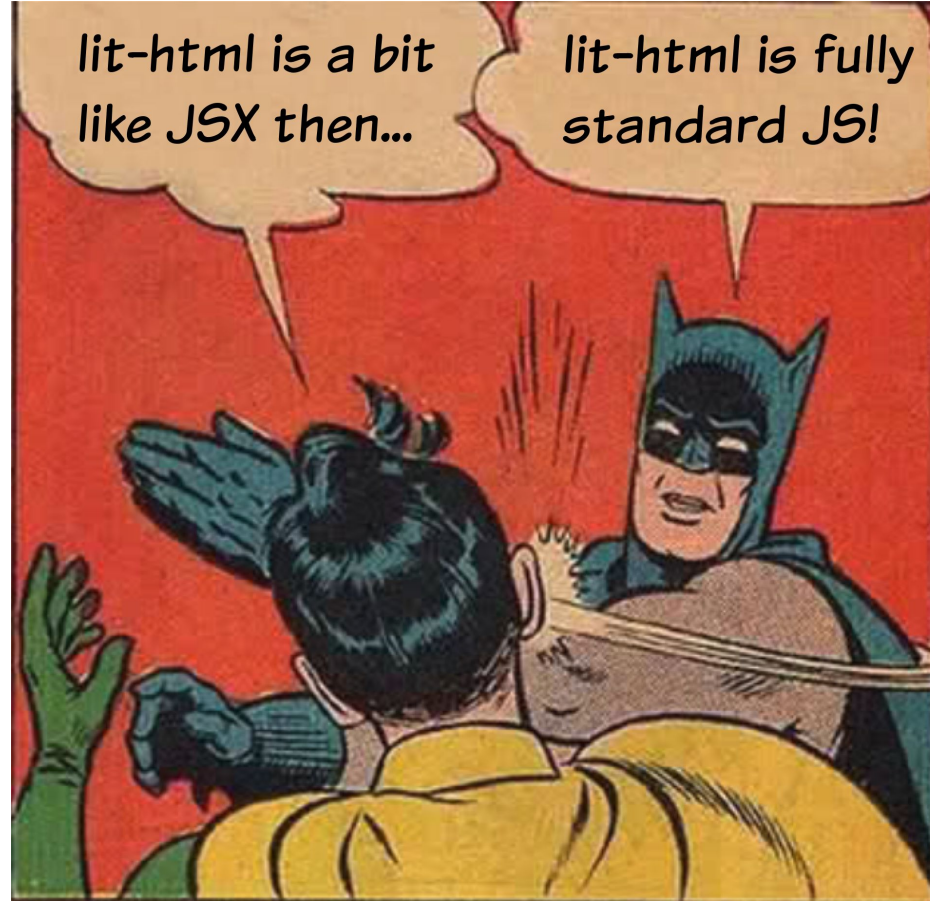
Little known functionality of template literals

# 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!

```
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

# Coding my-lit-counter

## Web Components in 2023 - Lit `my-counter` Element

Now we are going to use Lit to create another version of our counter, `my-lit-counter`.

To create Lit components, we need to use `nodejs` and `npm`. If you don't have them in your computer, the easiest way would be to use the [GitPod workspace](#), that has all the required tooling.

### Creating the project

In the `step-04` folder, create a new lit project using the Open Web Components ( `open-wc` ) generator, and call it `my-lit-counter` :

```
npm init @open-wc
```



After running `init` you will be provided with a prompt so that you can choose the type of project to start. Please choose *Scaffold a new project* to create your first Lit component.

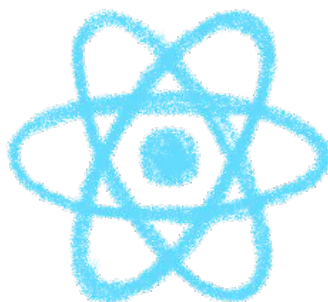
```
$ npm init @open-wc
```



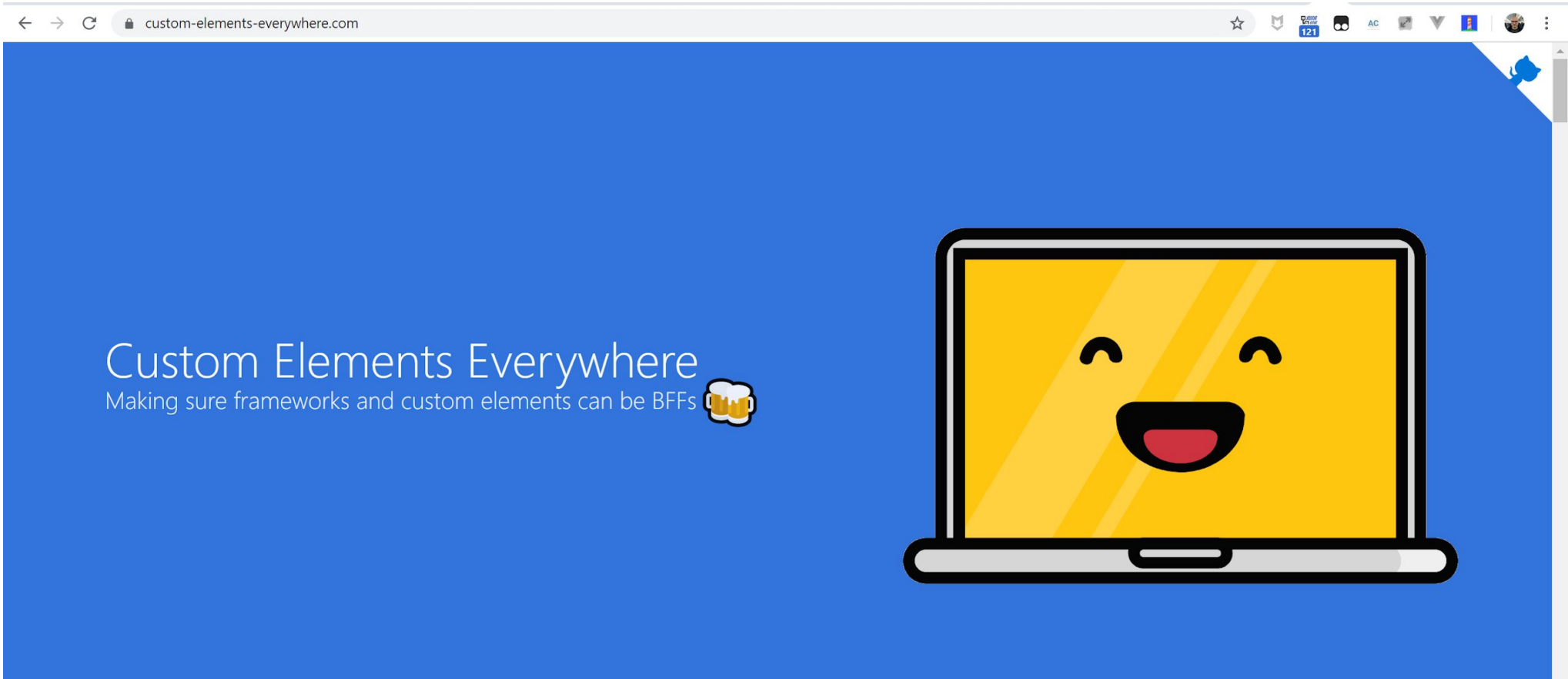
```
.....
```

# Web Components & Frameworks

Less “*either/or*” and more “*both/and*”



# Compatibility is on Web Components side



Web Components everywhere, baby!

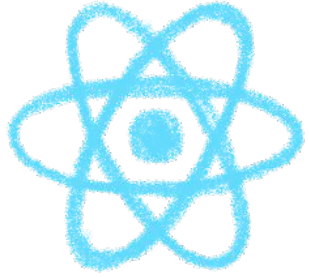


# They are the interoperable alternative



Any framework... or no framework at all

# You can have a single implementation



And it simply works everywhere\*

# \*React don't fully support them yet



A screenshot of the Lit documentation website. The page title is "React". The left sidebar shows a navigation menu with categories like "INTRODUCTION", "COMPONENTS", "TEMPLATES", "COMPOSITION", "MANAGING DATA", "TOOLS AND WORKFLOWS", "SERVER RENDERING", "FRAMEWORKS" (with "React" selected), "LOCALIZATION", "API", "RELEASES", and "RELATED LIBRARIES". The main content area has a blue header "React" followed by a callout box stating: "This package is part of the Lit Labs family of experimental packages. See the Lit Labs page for guidance on using Labs software in production." Below this, the text explains that the @lit-labs/react package provides utilities for creating React wrapper components and custom hooks from reactive controllers. It also notes that the wrapper enables setting properties on custom elements, mapping DOM events to React-style callbacks, and enables correct type-checking in JSX by TypeScript. The wrappers are targeted at two audiences: users of web components who wrap components and controllers for their own use, and vendors who publish React wrappers for their users. The right sidebar shows a "Contents: React" section with a tree view: "Why are wrappers needed?", "createComponent" (with sub-items "Usage" and "How it works"), and "useController" (with sub-items "Usage" and "How it works"). The top navigation includes "Documentation v2", "Tutorials", "Playground", "Blog", a search bar, and a GitHub icon.

Long story made short: use [lit-labs/react](https://github.com/lit-labs/react)

# When you need interoperability



Nothing beats the standard

# Angular can generate web components



- Introduction
- Getting started >
- Understanding Angular >
  - Overview
  - Components >
    - Overview
    - Component lifecycle
    - View encapsulation
    - Component interaction
    - Component styles
    - Sharing data between child and parent directives and components
    - Content projection

## Angular elements overview

Angular elements are Angular components packaged as *custom elements* (also called Web Components), a web standard for defining new HTML elements in a framework-agnostic way.

For the sample application that this page describes, see the [live example](#) / [download example](#).

[Custom elements](#) are a Web Platform feature currently supported by Chrome, Edge (Chromium-based), Firefox, Opera, and Safari, and available in other browsers through polyfills (see [Browser Support](#)). A custom element extends HTML by allowing you to define a tag whose content is created and controlled by JavaScript code. The browser maintains a `CustomElementRegistry` of defined custom elements, which maps an instantiable JavaScript class to an HTML tag.

The `@angular/elements` package exports a `createCustomElement()` API that provides a bridge from Angular's component interface and change detection functionality to the built-in DOM API.

- Angular elements overview
  - Using custom elements
    - How it works
    - Transforming components to custom elements
    - Mapping
    - Browser support for custom elements
    - Example: A Popup Service
    - Typings for custom elements

## Angular Elements

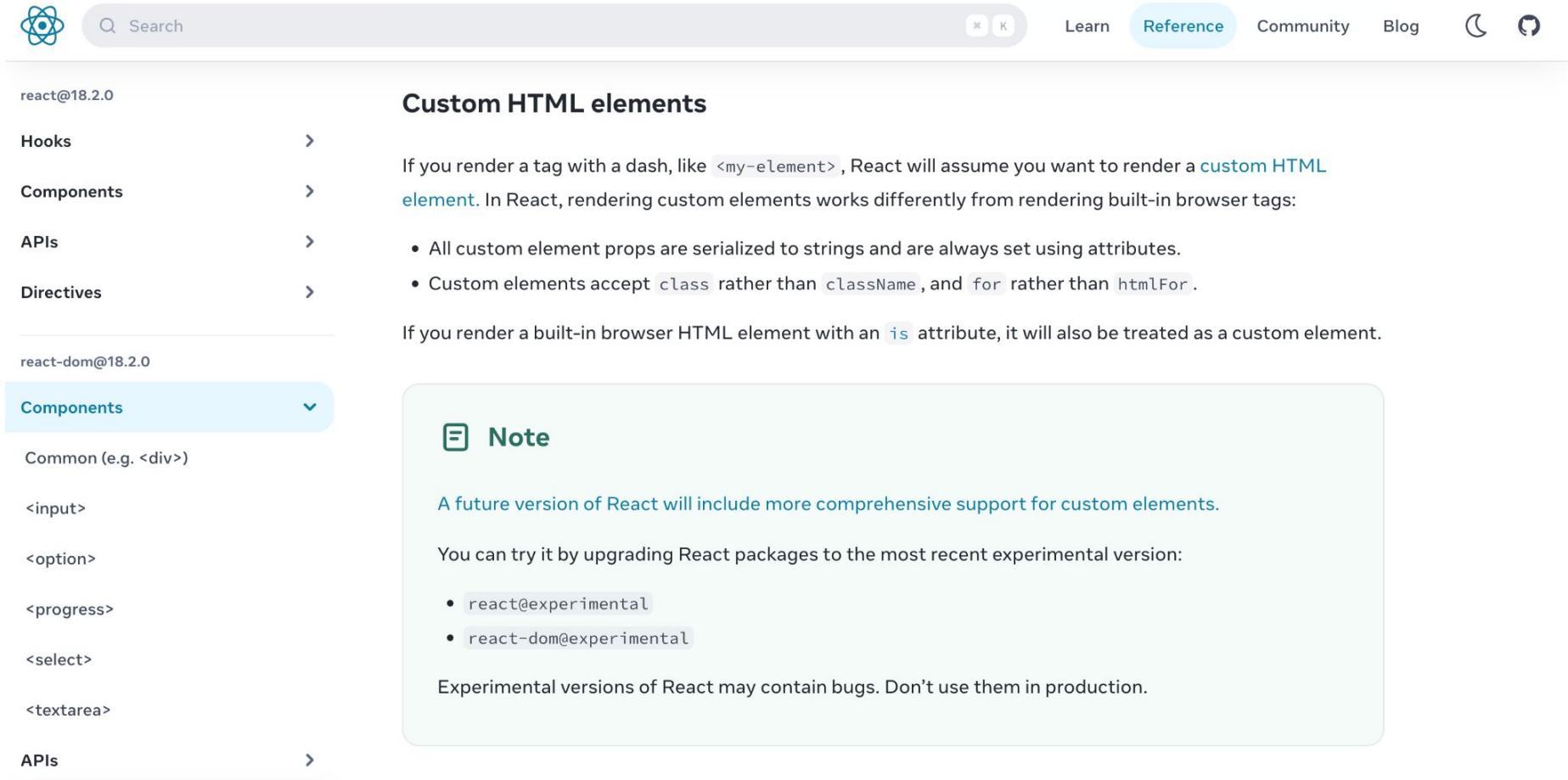
# Vue can generate web components



The screenshot shows the Vue.js documentation page for "Vue and Web Components". The page has a navigation bar with links for Docs, API, Playground, Ecosystem, About, Sponsor, and Partners. A search bar is present. On the left, there is a sidebar with "API Preference" (Options and Composition) and a list of navigation items under "Getting Started" and "Essentials". The main content area has the title "Vue and Web Components" and a paragraph explaining that "Web Components" is an umbrella term for web native APIs. Below this is a section titled "Using Custom Elements in Vue" which states that Vue scores a perfect 100% in the Custom Elements Everywhere tests. On the right, there is a "ON THIS PAGE" section with links to "Using Custom Elements in Vue", "Building Custom Elements with V...", and "Web Components vs. Vue Comp...". Below that is a "SPONSORS" section with logos for appwrite, Vue Mastery, Vue School, VEHIKL, PASSIONATE PEOPLE, storyblok, ionic, CodeDict, and a placeholder for "Your logo".

With `defineCustomElement()`

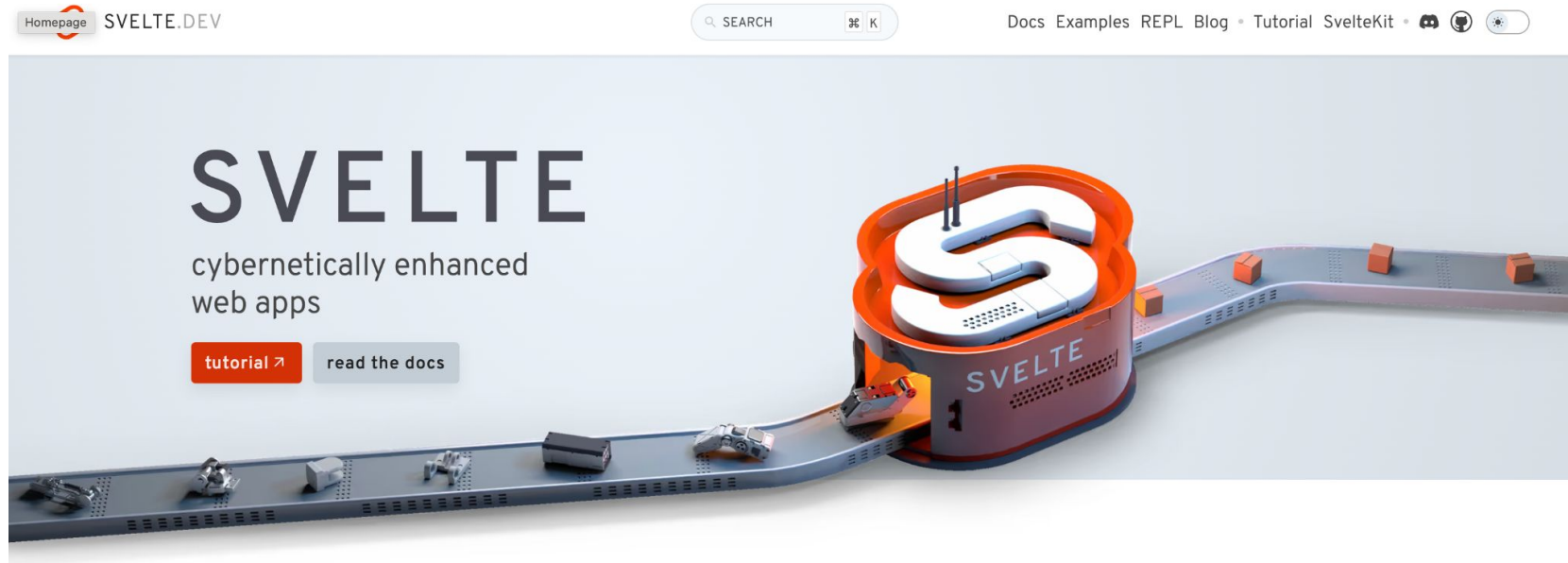
# React can generate web components



The screenshot shows the React documentation page for "Custom HTML elements". The left sidebar lists navigation options for "react@18.2.0" (Hooks, Components, APIs, Directives) and "react-dom@18.2.0" (Components, Common (e.g. <div>), <input>, <option>, <progress>, <select>, <textarea>, APIs). The main content area is titled "Custom HTML elements" and explains that tags with a dash (like `<my-element>`) are rendered as custom HTML elements. It lists two bullet points: "All custom element props are serialized to strings and are always set using attributes." and "Custom elements accept `class` rather than `className`, and `for` rather than `htmlFor`." A note box states: "A future version of React will include more comprehensive support for custom elements. You can try it by upgrading React packages to the most recent experimental version:" followed by two bullet points: `react@experimental` and `react-dom@experimental`. It concludes with "Experimental versions of React may contain bugs. Don't use them in production."

But it can generate them too

# What about Svelte?



## compiled

Svelte shifts as much work as possible out of the browser and into your build step. No more manual optimisations – just faster, more efficient apps.

## compact

Write breathtakingly concise components using languages you already know – HTML, CSS and JavaScript. Oh, and your application bundles will be tiny as well.

## complete

Built-in scoped styling, state management, motion primitives, form bindings and more – don't waste time trawling npm for the bare essentials. It's all here.

<https://svelte.dev>

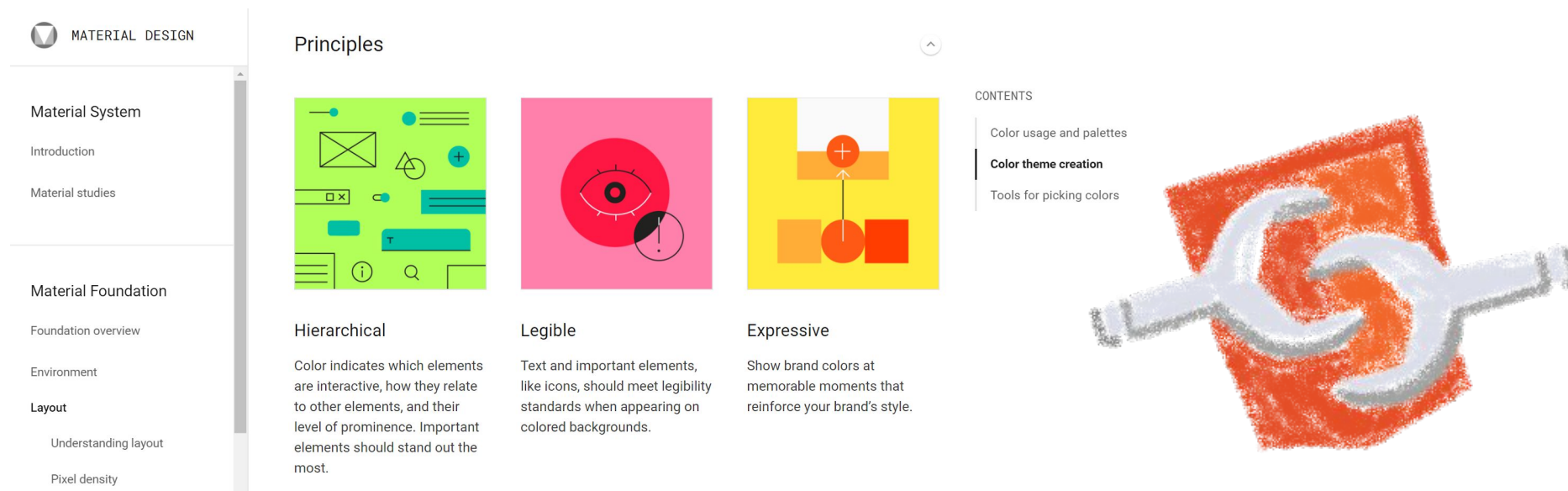


## Let's look in detail one case



# Web Components & Design Systems

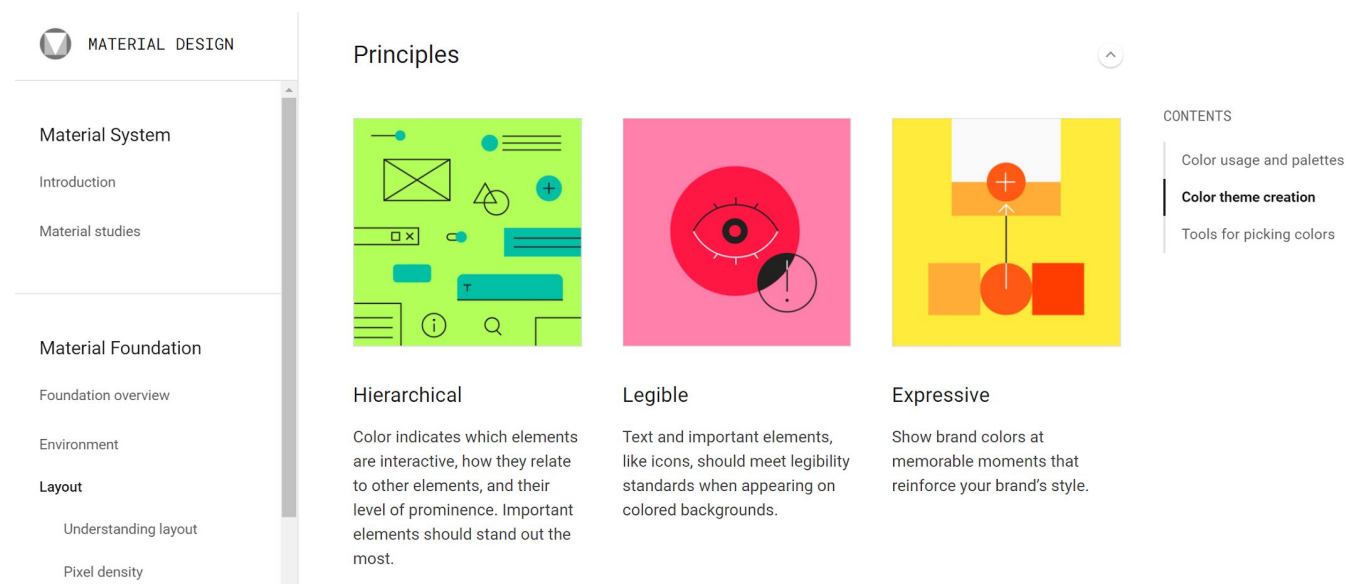
## One of the best cases for Web Components



The screenshot displays the Material Design website. On the left is a navigation sidebar with sections: MATERIAL DESIGN, Material System (Introduction, Material studies), Material Foundation (Foundation overview, Environment), and Layout (Understanding layout, Pixel density). The main content area is titled 'Principles' and features three columns: 'Hierarchical' (with a green background and icons), 'Legible' (with a pink background and an eye icon), and 'Expressive' (with a yellow background and a plus sign icon). Each column has a brief description of the principle. To the right of the principles is a 'CONTENTS' sidebar listing 'Color usage and palettes', 'Color theme creation', and 'Tools for picking colors'. Below the contents is a large, stylized graphic of a wrench and a screwdriver on an orange background.

# So, what are Design Systems?

## And why should I look at them?



The screenshot shows the Material Design documentation interface. On the left is a navigation sidebar with sections: MATERIAL DESIGN, Material System (Introduction, Material studies), Material Foundation (Foundation overview, Environment, Layout (Understanding layout, Pixel density)). The main content area is titled 'Principles' and features three cards: 'Hierarchical' (with a green background and icons), 'Legible' (with a pink background and an eye icon), and 'Expressive' (with a yellow background and a plus sign icon). Below each card is a brief description. On the right is a 'CONTENTS' sidebar with links for 'Color usage and palettes', 'Color theme creation', and 'Tools for picking colors'.

**Principles**

- Hierarchical**  
Color indicates which elements are interactive, how they relate to other elements, and their level of prominence. Important elements should stand out the most.
- Legible**  
Text and important elements, like icons, should meet legibility standards when appearing on colored backgrounds.
- Expressive**  
Show brand colors at memorable moments that reinforce your brand's style.

**CONTENTS**

- Color usage and palettes
- Color theme creation**
- Tools for picking colors

# A talk for devs by a dev



I am not a designer, neither I play one on TV...

# The same or different?

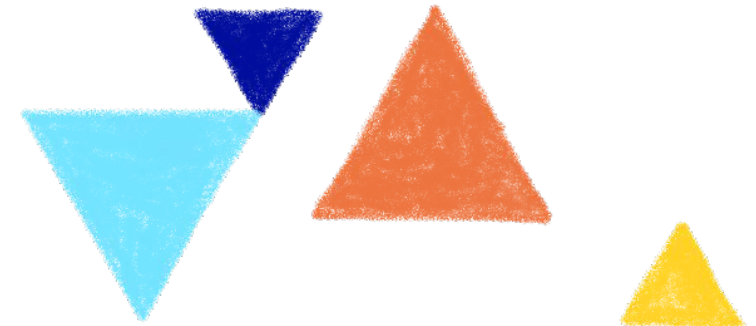
Design System

Component Catalog

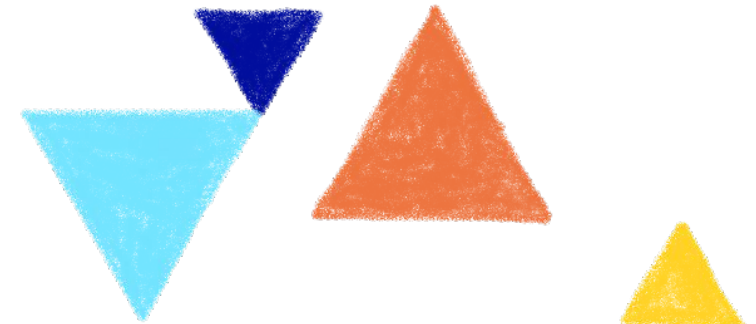
Style Guide



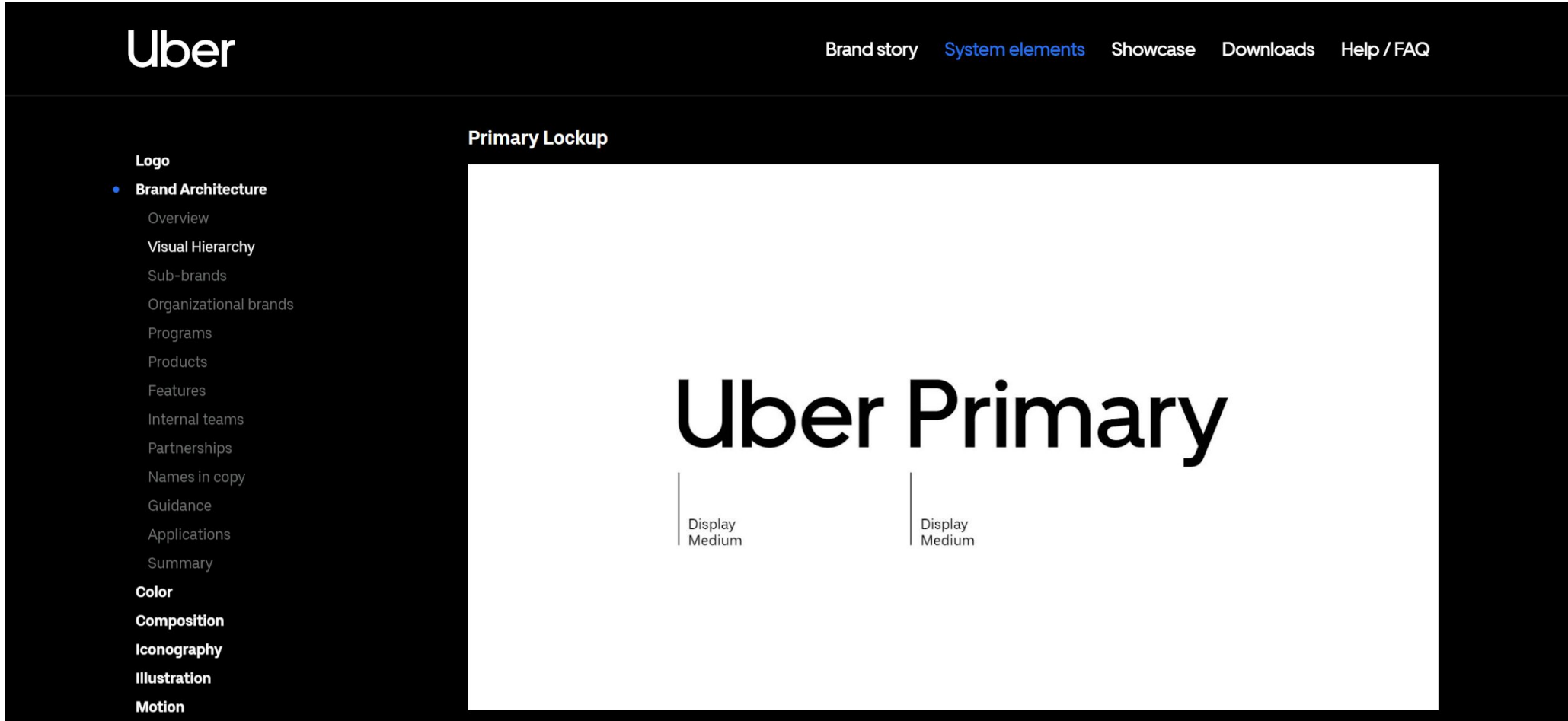
A **document** listing the **styles**, **patterns**, **practices**, and **principles** of a brand **design standards**



Style guides define the **application's look and feel**

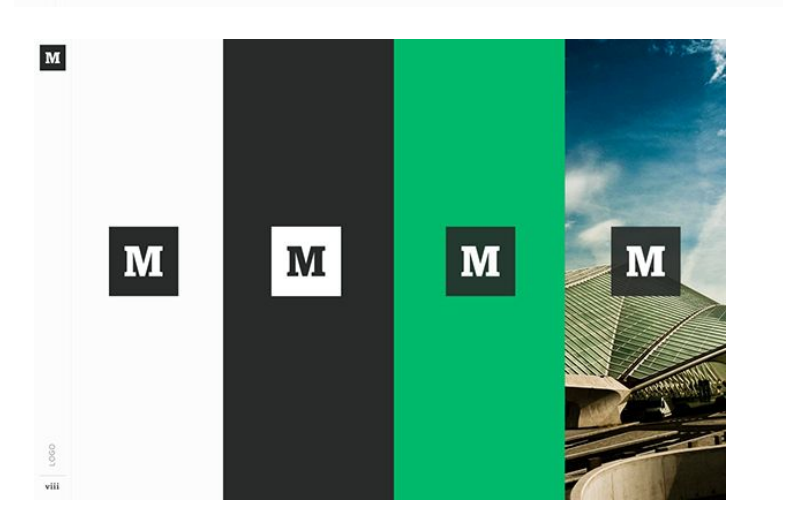
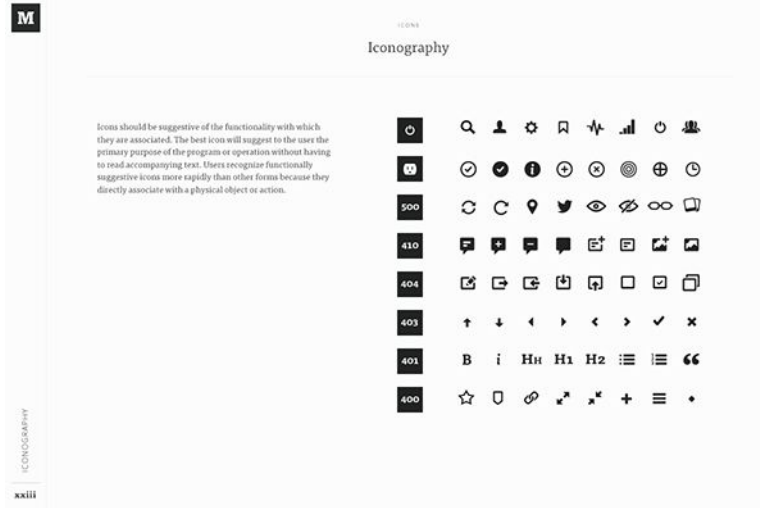
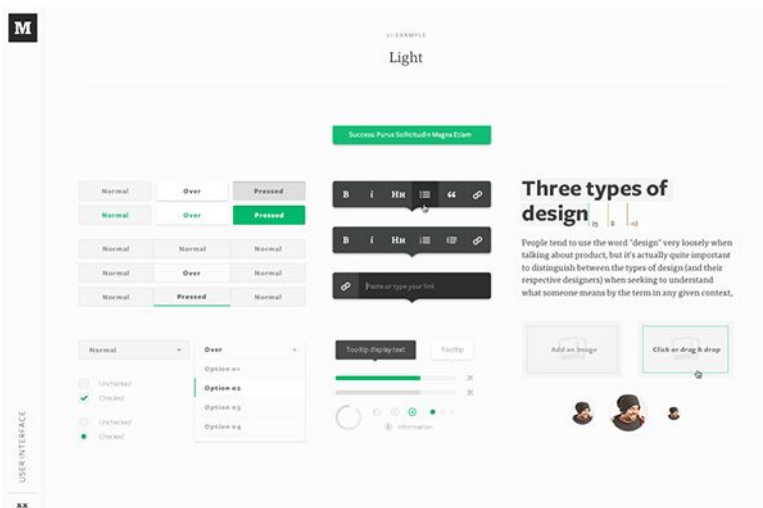
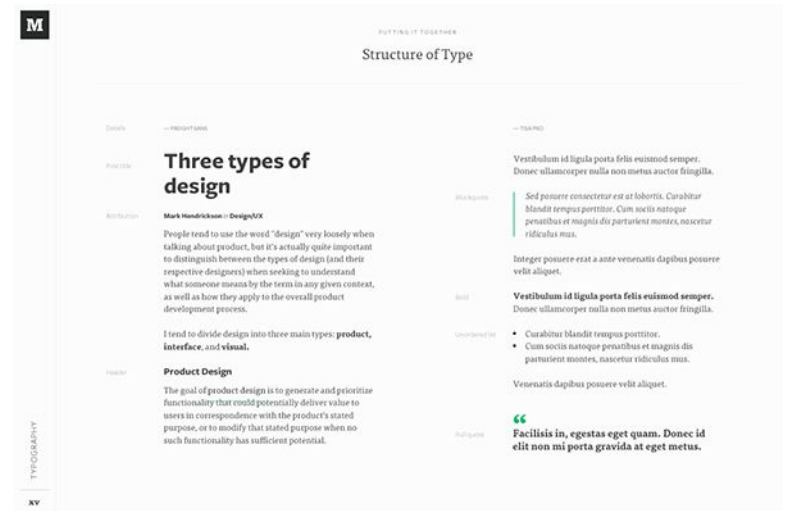


# Style Guide Example: Uber



<https://brand.uber.com/>

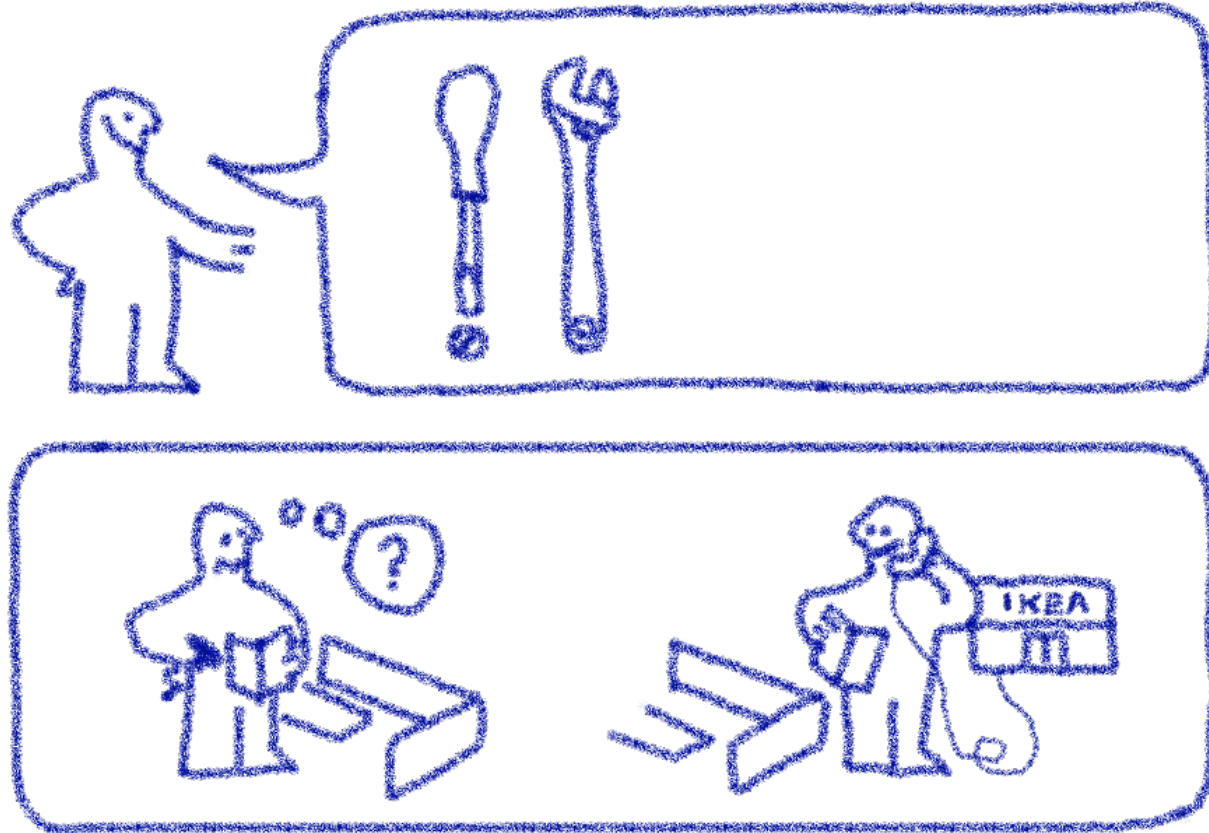
# Style Guide Example: Medium



<https://www.behance.net/gallery/7226653/Medium-Brand-Development>



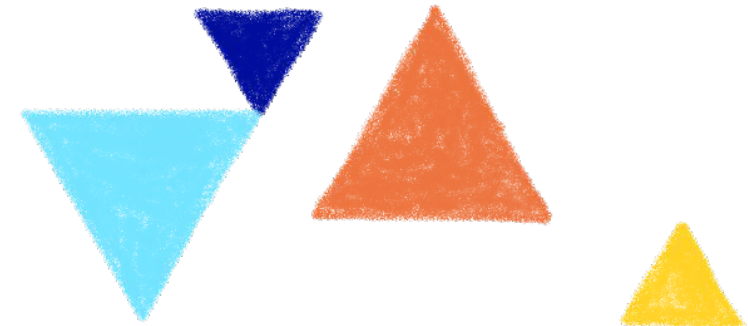
# Style Guides alone are ambiguous



Interpretation needed to adapt the preconisation to the use case

# Component Catalogs

A **component catalog** is a **repository** of components, with one or several **implementations**, code **examples** and **technical documentation**



# Component Catalog example: Bootstrap

A simple primary alert—check it out!

A simple secondary alert—check it out!

A simple success alert—check it out!

A simple danger alert—check it out!

A simple dark alert—check it out!

```
<div class="alert alert-primary" role="alert">  
  A simple primary alert—check it out!  
</div>  
<div class="alert alert-secondary" role="alert">  
  A simple secondary alert—check it out!  
</div>  
<div class="alert alert-success" role="alert">  
  A simple success alert—check it out!  
</div>
```



<https://getbootstrap.com/>

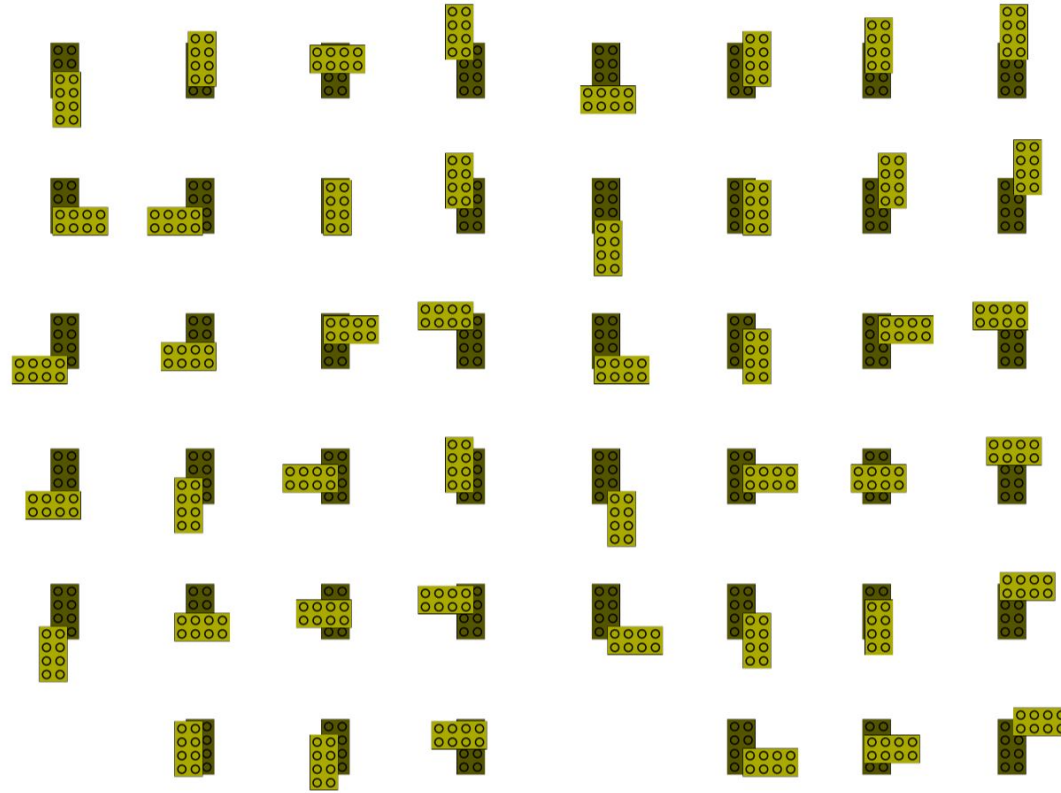
# Component Catalog Example: ING's Lion



A screenshot of the Storybook web application. The left sidebar shows a navigation menu with categories like 'INTRO', 'FORMS', and 'Input Datepicker'. The 'Main' item under 'Input Datepicker' is selected. The main canvas area displays a 'Date' component, which is a calendar for June 2020. The date '22' is highlighted with a green border. The calendar has a header 'Date' with a close button 'x', and navigation arrows '&lt;' and '&gt;' for the month. The days of the week are listed as 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'. The dates are arranged in a grid: 31, 1, 2, 3, 4, 5, 6; 7, 8, 9, 10, 11, 12, 13; 14, 15, 16, 17, 18, 19, 20; 21, 22, 23, 24, 25, 26, 27; 28, 29, 30, 1, 2, 3, 4.

<https://lion-web-components.netlify.app/>

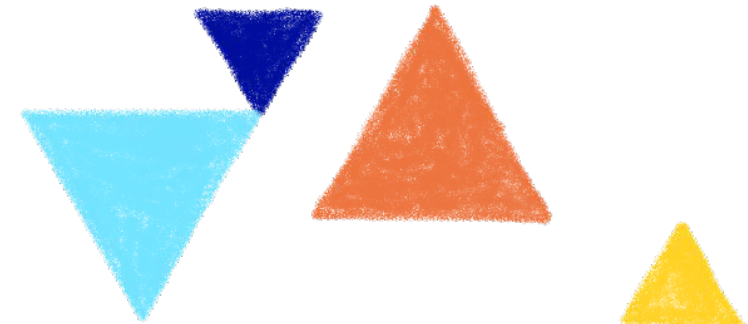
# Catalogs alone create inconsistency



Like using the same LEGO bricks  
to create very different objects

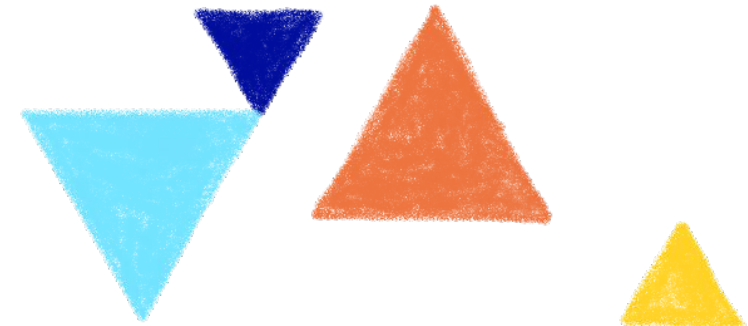
# Design Systems

A Design System is like a **common visual language** for **product teams**



# Design systems

A Design System is a set of **design standards**, **documentations**, and **principles**, alongside with the toolkit (**UI patterns** and **code components**) to achieve those standards



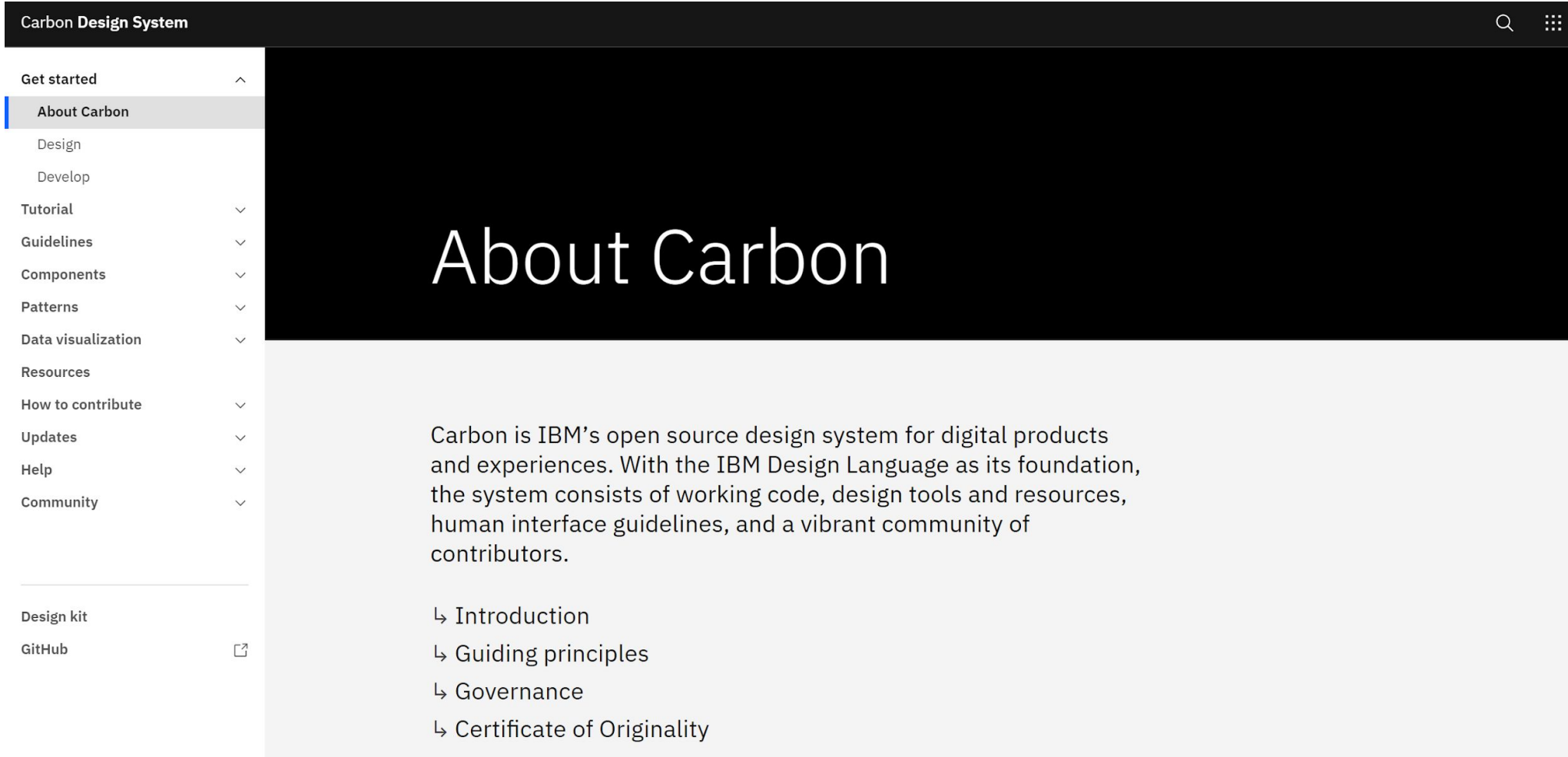
# Design systems

Design System  $\approx$   
Style Guide + Component Catalog





# Example: Carbon Design System



The screenshot shows the Carbon Design System website. The header is dark with the title 'Carbon Design System' and search and menu icons. A sidebar on the left lists navigation items: 'Get started', 'About Carbon' (highlighted), 'Design', 'Develop', 'Tutorial', 'Guidelines', 'Components', 'Patterns', 'Data visualization', 'Resources', 'How to contribute', 'Updates', 'Help', and 'Community'. Below these are 'Design kit' and 'GitHub'. The main content area has a dark header with 'About Carbon' in white. Below this, a paragraph describes Carbon as IBM's open source design system. A list of links follows: Introduction, Guiding principles, Governance, and Certificate of Originality.

<https://www.carbondsdesignsystem.com/>

# Example: Firefox's Photon Design System

## Photon Design System

### Photon Design

- Principles
- Getting started
- Design for Scale
- Design for Performance
- Design for Inclusion

### Visuals

### Motion

### Copy

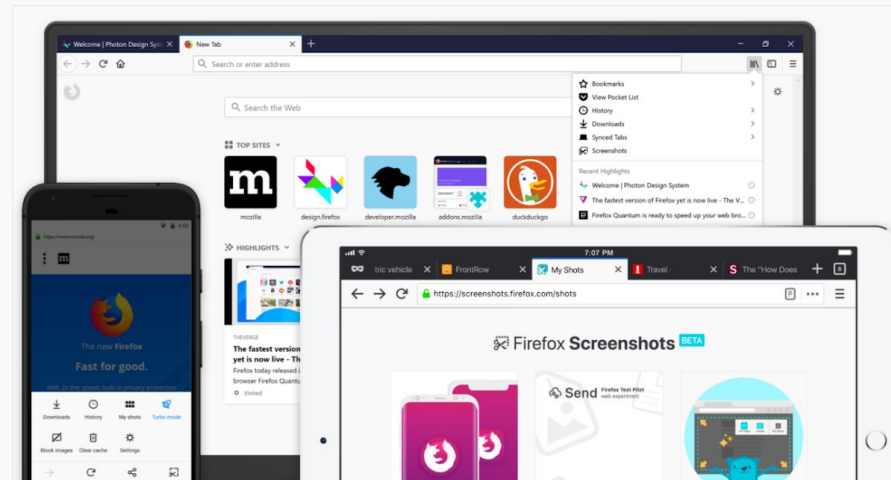
### Patterns

### Components

### Resources

## Photon Design System

Launch recognizable, enjoyable Firefox products and features faster.



Photon is the Firefox design language to build modern, intuitive, delightful experiences, for products across all platforms – from mobile to desktop, from TV to the next big thing.

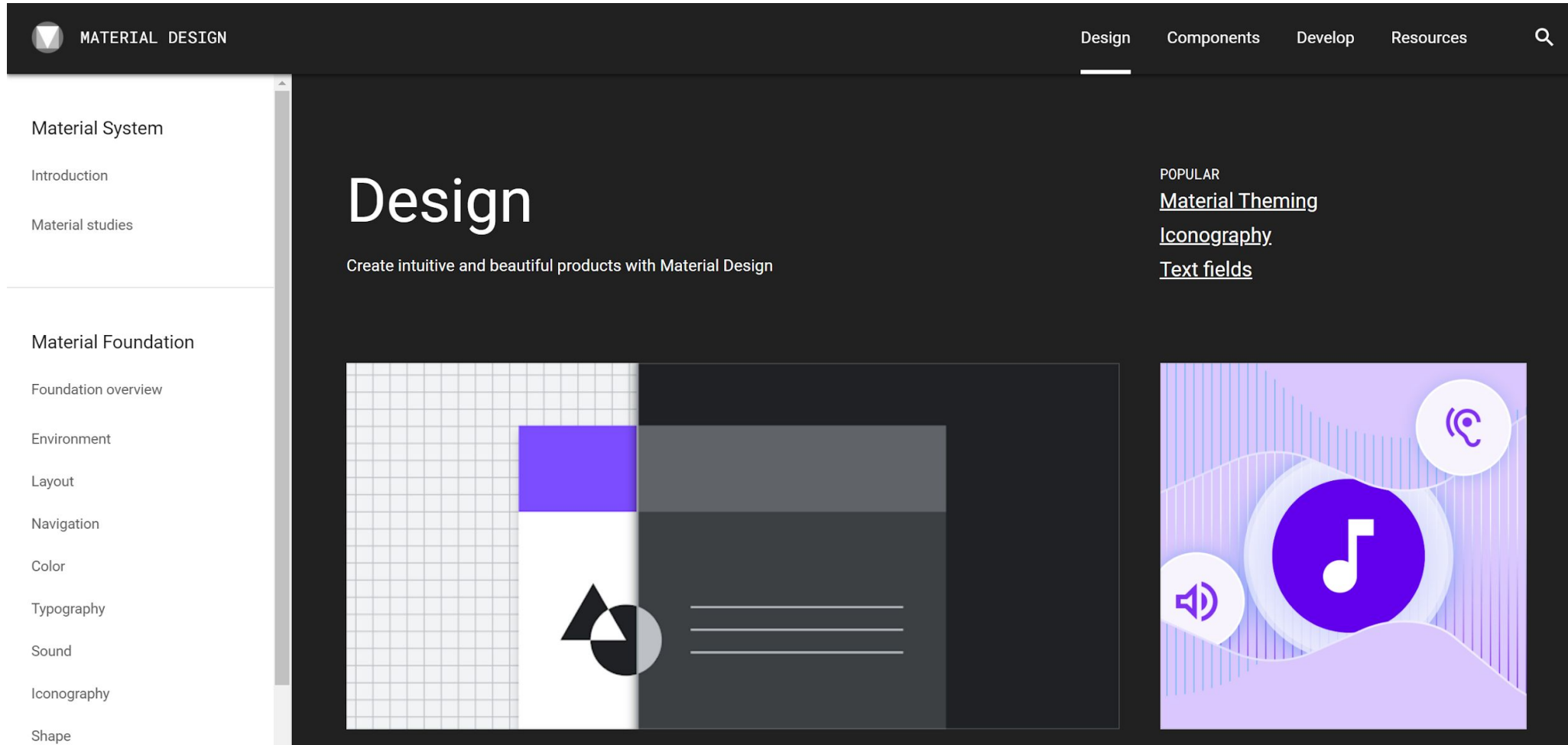
The Photon Design System houses guidelines, reusable UI components, templates, and other resources to help you create products for Firefox users. It is flexible and always evolving to serve the best Firefox experience for every situation.

Using this system will help make your work more efficient, and our products more consistent, while still looking, feeling, and sounding uniquely Firefox.

You can help us improve the system and ensure it remains current and relevant.

<https://design.firefox.com/photon/>

# Example: Material Design



<https://material.io/>

# The component catalog

The poor relative of the Design System family

# Let's choose a simple example

A simple primary alert—check it out!

A simple secondary alert—check it out!

A simple success alert—check it out!

A simple danger alert—check it out!

A simple dark alert—check it out!

```
<div class="alert alert-primary" role="alert">
  A simple primary alert—check it out!
</div>
<div class="alert alert-secondary" role="alert">
  A simple secondary alert—check it out!
</div>
<div class="alert alert-success" role="alert">
  A simple success alert—check it out!
</div>
```



Bootstrap

## Bootstrap based component catalogs







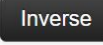

# A long time ago



## Buttons

### Default buttons

Button styles can be applied to anything with the `.btn` class applied. However, typically you'll want to apply these to only `<a>` and `<button>` elements for the best rendering.

Button	class=""	Description
	<code>btn</code>	Standard gray button with gradient
	<code>btn btn-primary</code>	Provides extra visual weight and identifies the primary action in a set of buttons
	<code>btn btn-info</code>	Used as an alternative to the default styles
	<code>btn btn-success</code>	Indicates a successful or positive action
	<code>btn btn-warning</code>	Indicates caution should be taken with this action
	<code>btn btn-danger</code>	Indicates a dangerous or potentially negative action
	<code>btn btn-inverse</code>	Alternate dark gray button, not tied to a semantic action or use
	<code>btn btn-link</code>	Deemphasize a button by making it look like a link while maintaining button behavior



Bootstrap

Components defined in HTML, CSS and some jQuery

# Then it was AngularJS time...


UI Bootstrap Directives Getting started Previous docs

# UI Bootstrap

Bootstrap components written in pure AngularJS by the AngularUI Team

[Code on Github](#) [Download \(2.5.0\)](#) [Create a Build](#)

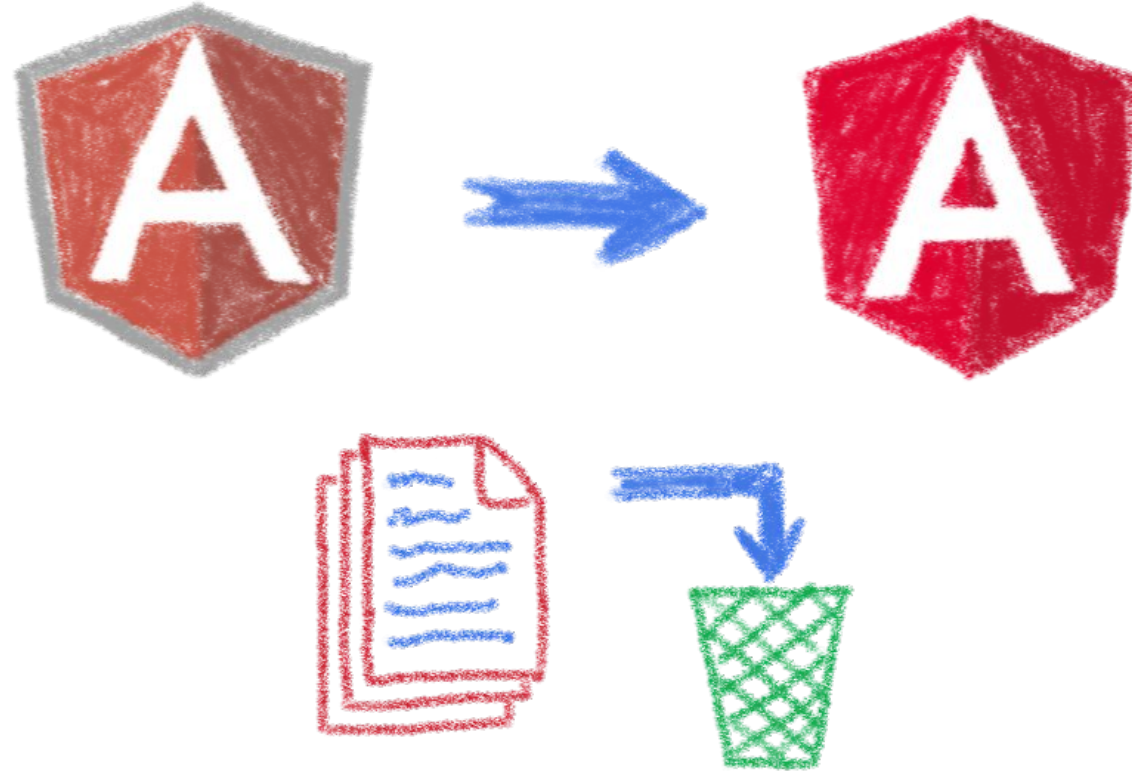
[Star 14,640](#) [Fork 7,184](#) [Tweet](#)



Getting started

## And new reference implementations were needed

# But you know the sad story...



All UI Bootstrap based catalogs woke up with an obsolete implementation



# Worry no more, let's do Angular!



Home Getting Started Components

1240,914  
npm



## Bootstrap widgets

### The angular way

Angular widgets built from the ground up using only Bootstrap 4 CSS with APIs designed for the Angular ecosystem.

No dependencies on 3rd party JavaScript.

[Demo](#)

[Installation](#)

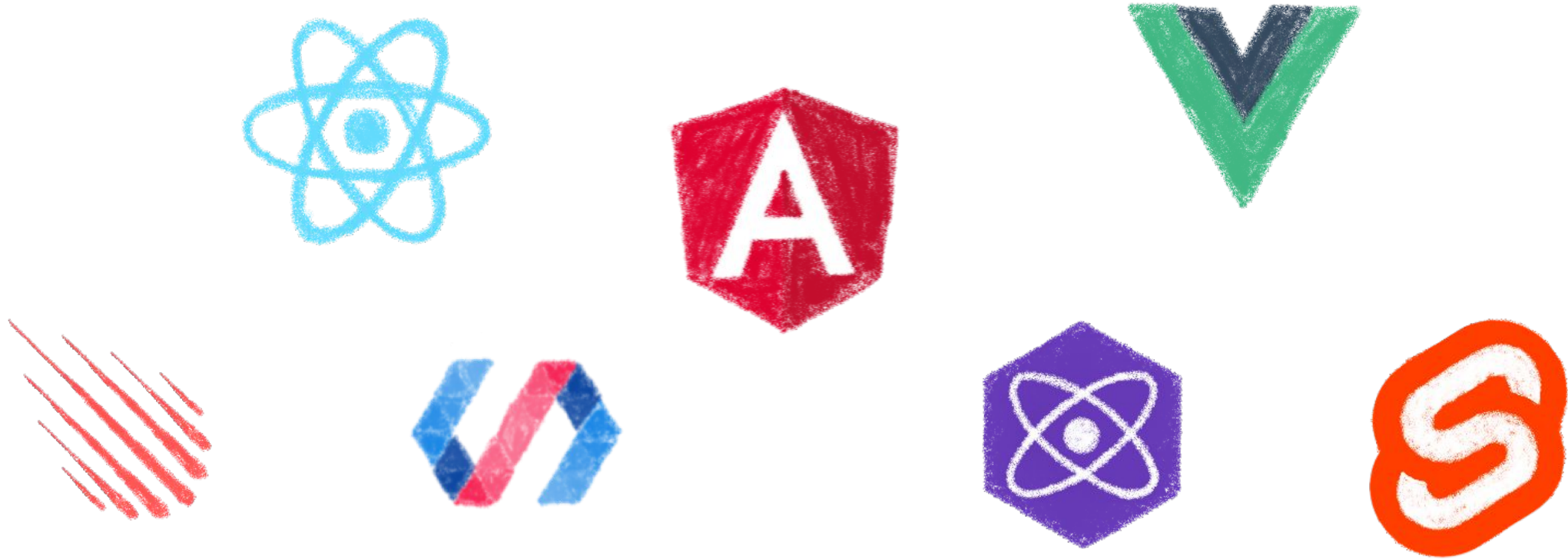
Currently at v6.1.0



# ng-bootstrap to the rescue



# But times had changed...



In 2017 Angular is only one more in the clique

# React is the new Big Thing™



Black Lives Matter. [Support the Equal Justice Initiative.](#)


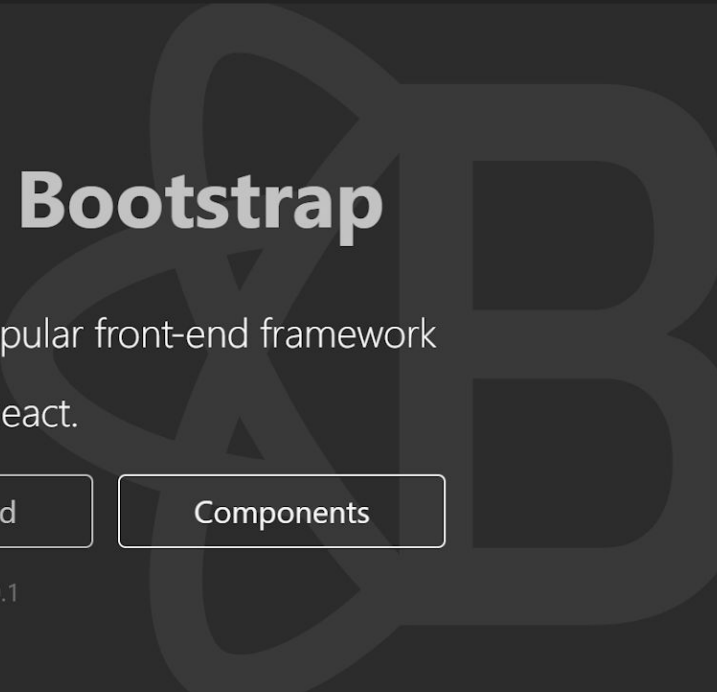
Home Getting Started Components v1.0.1 (Bootstrap 4.5)

## React Bootstrap

The most popular front-end framework  
Rebuilt for React.

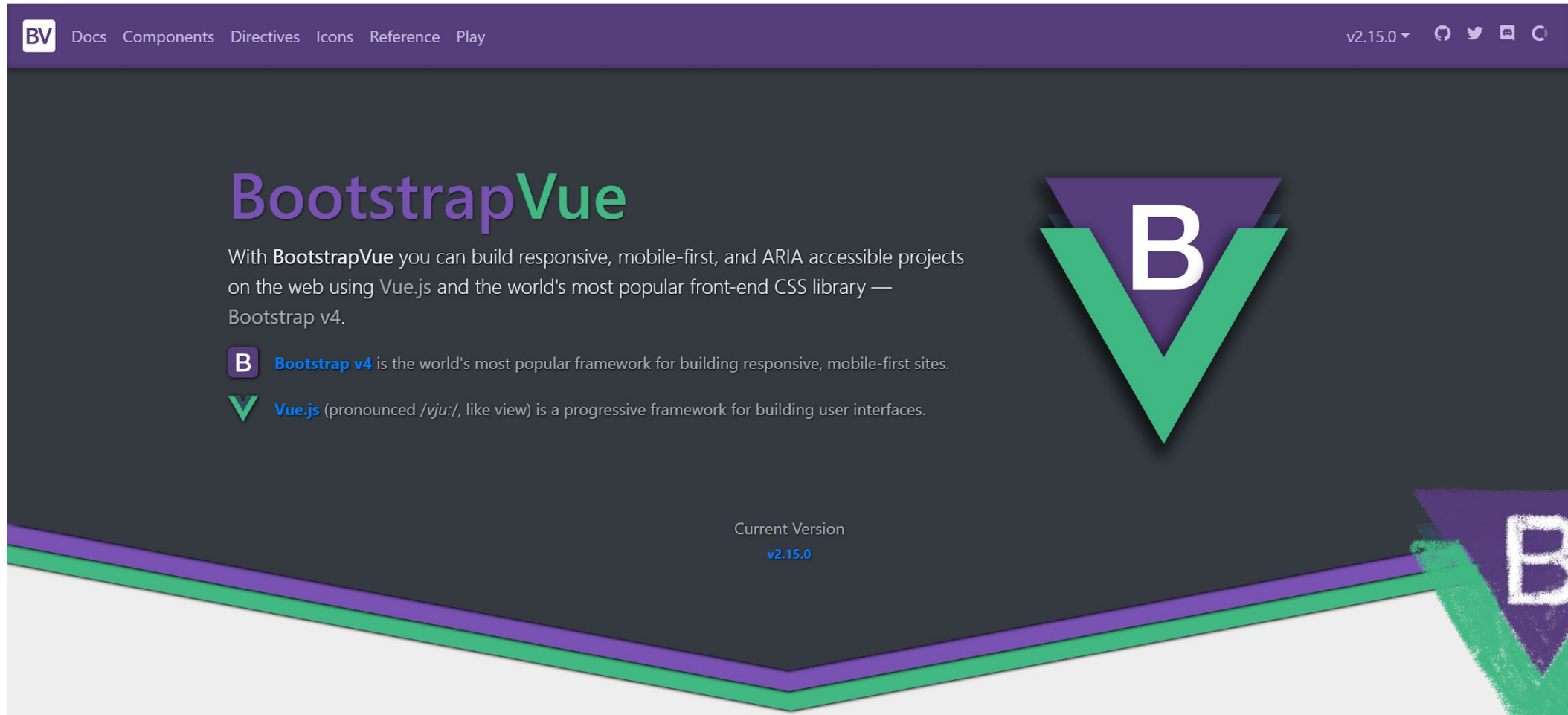
[Get started](#) [Components](#)

Current version: 1.0.1



So let's build React Bootstrap...

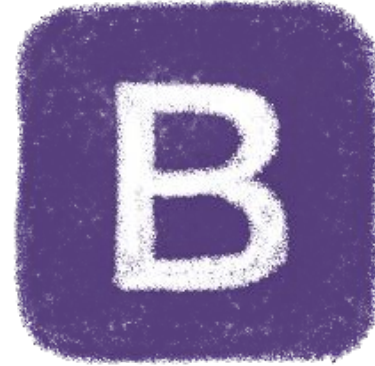
# Wait, what about Vue?



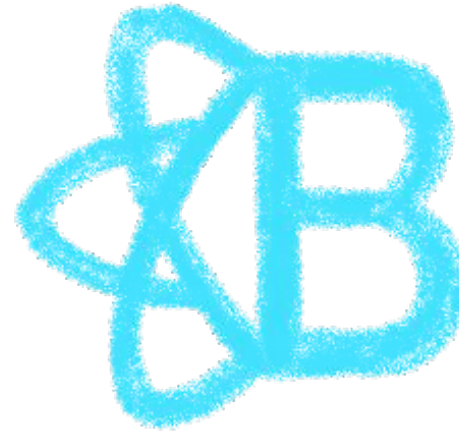
The screenshot shows the BootstrapVue documentation page. At the top, there is a navigation bar with links for 'Docs', 'Components', 'Directives', 'Icons', 'Reference', and 'Play'. The version 'v2.15.0' is displayed in the top right corner. The main heading is 'BootstrapVue'. Below it, a paragraph states: 'With BootstrapVue you can build responsive, mobile-first, and ARIA accessible projects on the web using Vue.js and the world's most popular front-end CSS library — Bootstrap v4.' There are two callout boxes: one with a 'B' icon for 'Bootstrap v4' and one with a 'V' icon for 'Vue.js'. At the bottom of the page, it says 'Current Version v2.15.0'. The page is overlaid with a large, stylized 'BV' logo consisting of a purple triangle with a white 'B' and a green triangle with a white 'V'.

We also need BootstrapVue

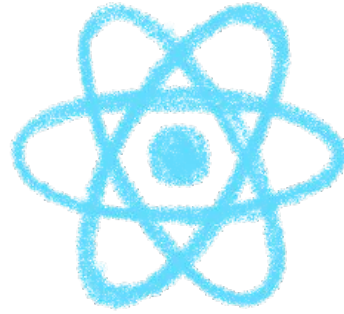
# OK, I think you see my point...



Bootstrap

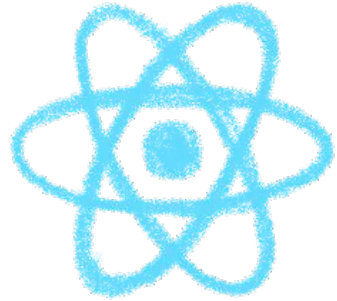


# Most Design System do a choice



Either they choose a canonical implementation or they ship and maintain several implementations

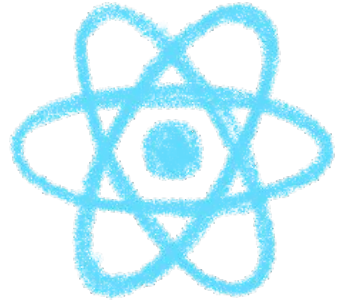
# Both choices are problematic



Shipping only one implementation:

Web dev ecosystem changes quickly and almost nobody keeps the same framework for years...

# Both choices are problematic

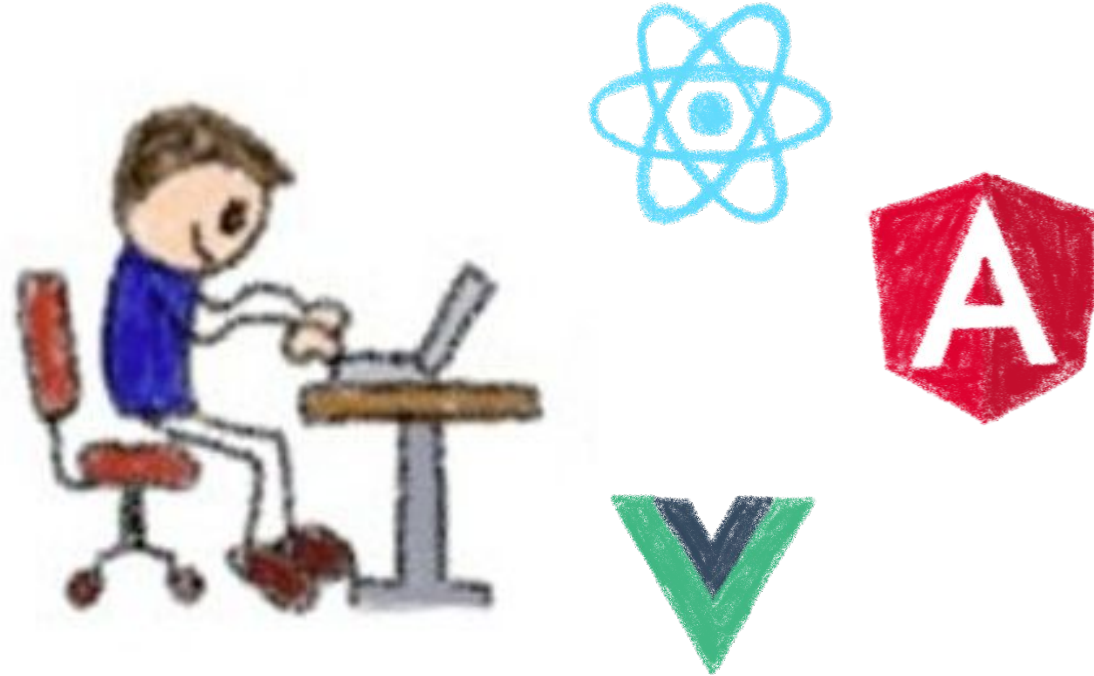


Shipping several implementations:

You need to maintain all the implementation...  
and you still miss some others

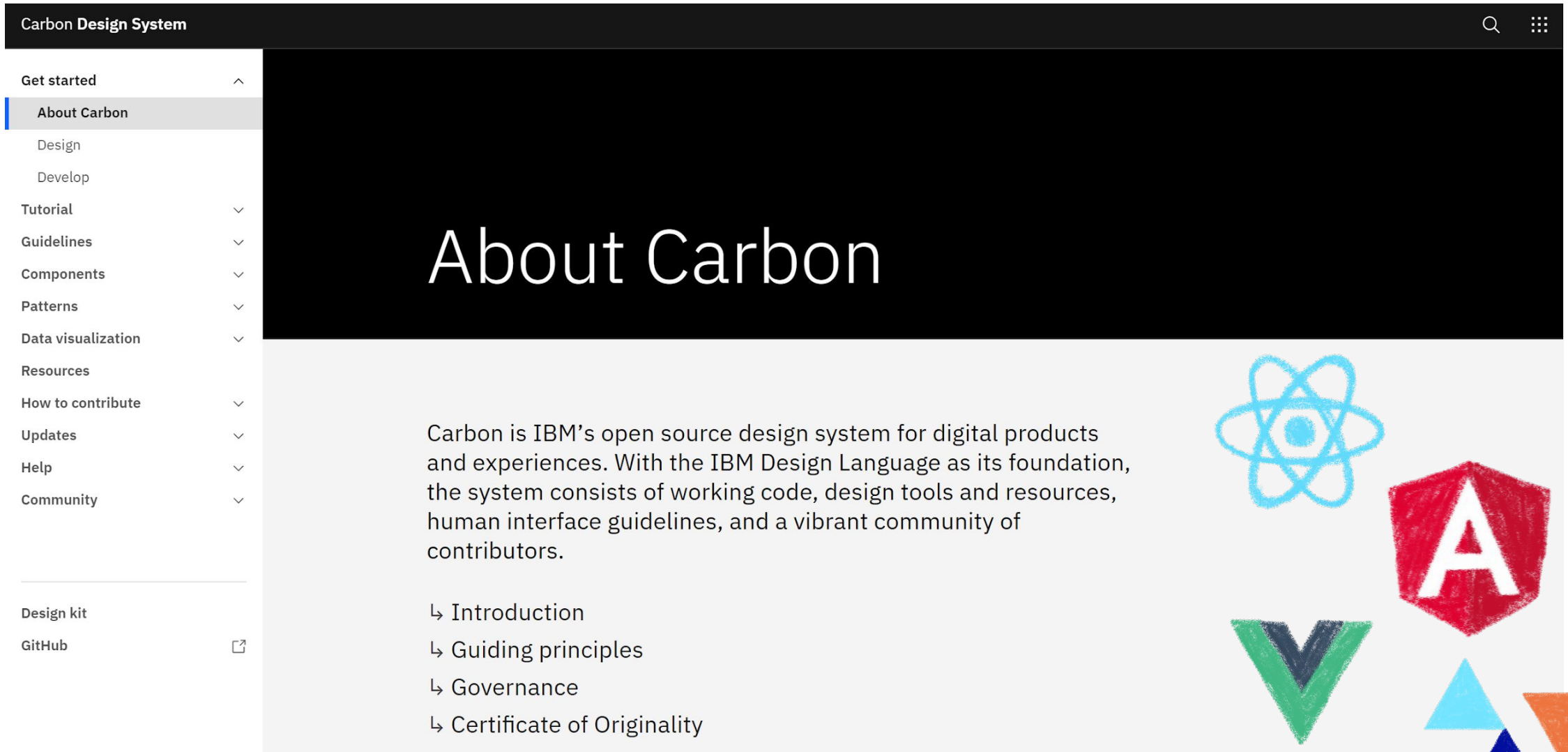


# Incomplete catalogs are problematic



People will need to recode the components  
in their chosen framework...  
Coherence is not guaranteed!!!

# Example: Carbon Design System



The screenshot shows the Carbon Design System website. The header is dark with the title "Carbon Design System" and search/menus icons. A sidebar on the left lists navigation items: "Get started", "About Carbon" (highlighted), "Design", "Develop", "Tutorial", "Guidelines", "Components", "Patterns", "Data visualization", "Resources", "How to contribute", "Updates", "Help", "Community", "Design kit", and "GitHub". The main content area has a dark background with the title "About Carbon" in white. Below this, on a light background, is a paragraph of text and a list of links. To the right of the text are several colorful, hand-drawn style icons: a blue atom, a red shield with a white 'A', a green 'V', a blue triangle, an orange triangle, and a yellow triangle.

Carbon Design System

Get started ^

About Carbon

Design

Develop

Tutorial v

Guidelines v

Components v

Patterns v

Data visualization v

Resources

How to contribute v

Updates v

Help v

Community v

Design kit

GitHub

## About Carbon

Carbon is IBM's open source design system for digital products and experiences. With the IBM Design Language as its foundation, the system consists of working code, design tools and resources, human interface guidelines, and a vibrant community of contributors.

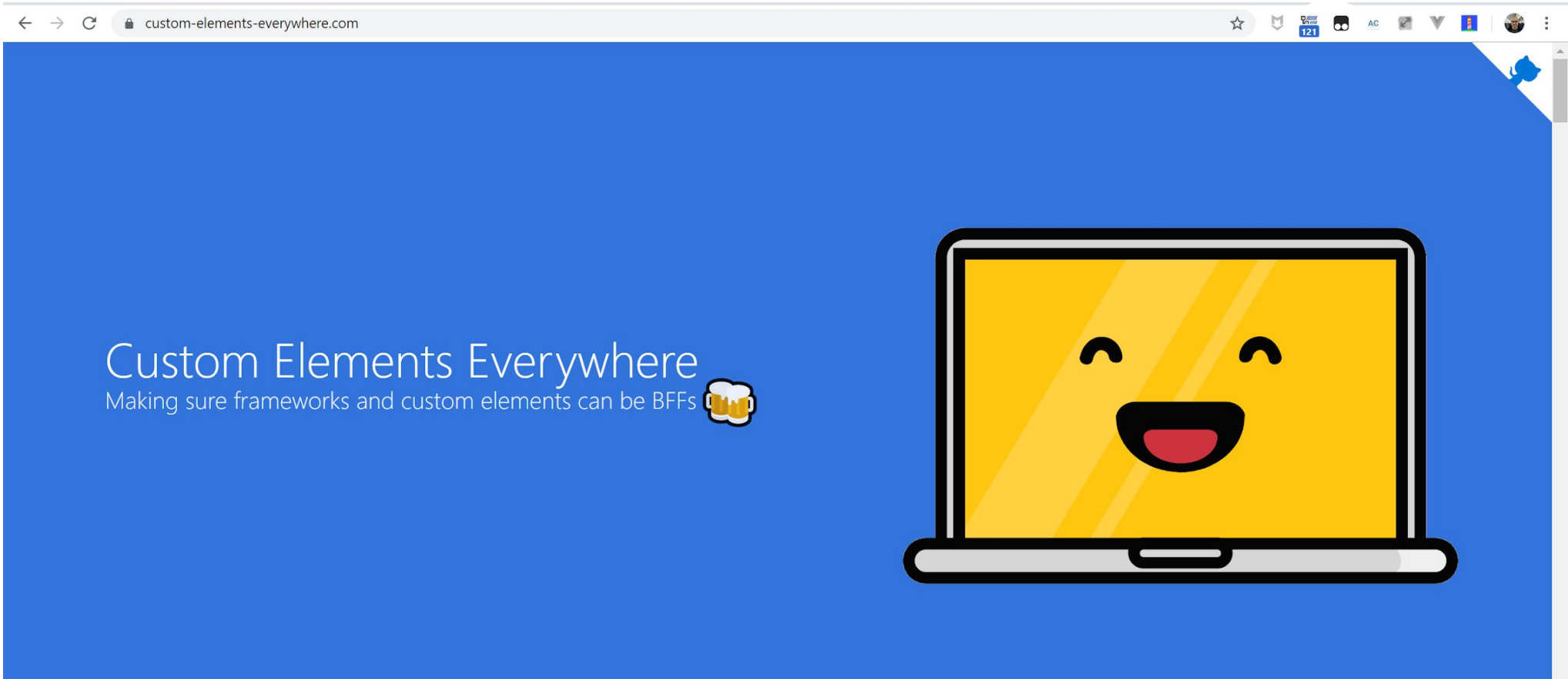
- ↳ Introduction
- ↳ Guiding principles
- ↳ Governance
- ↳ Certificate of Originality

# Web Components & Design Systems

A match made in heaven

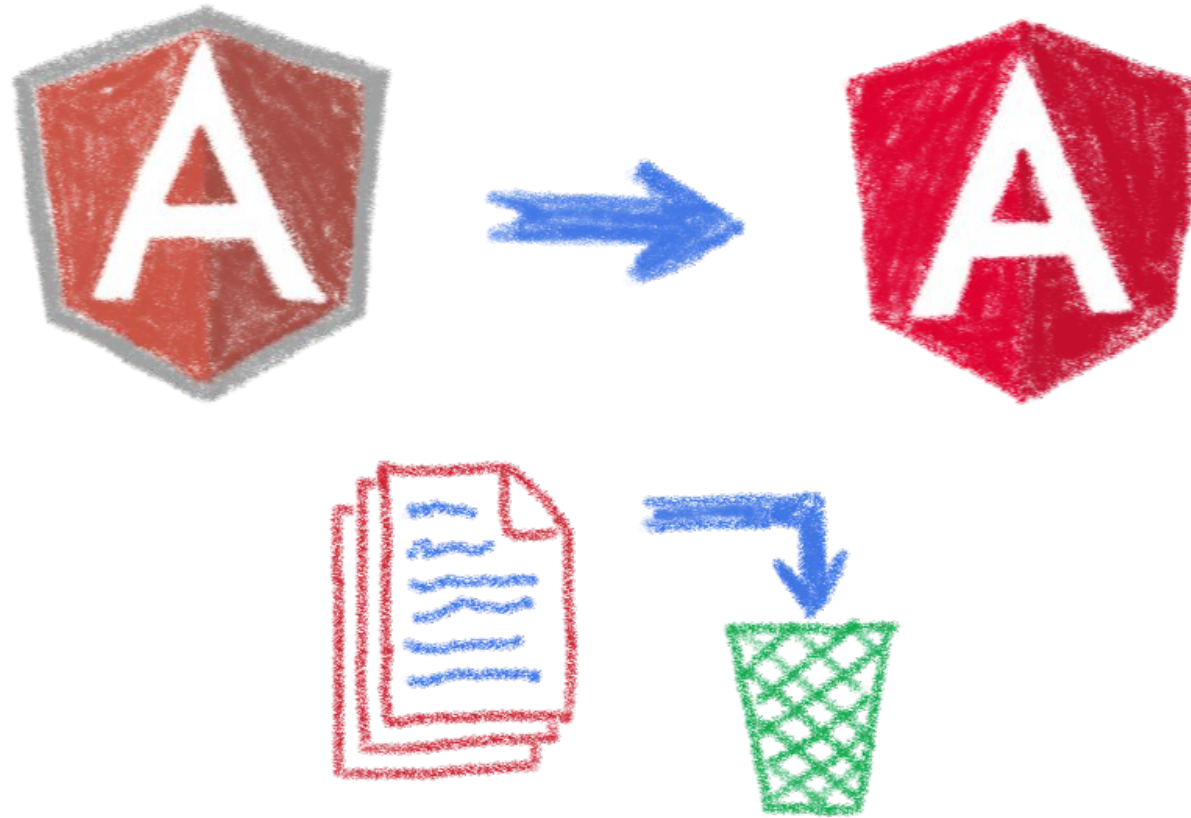


# Compatibility is on Web Components side



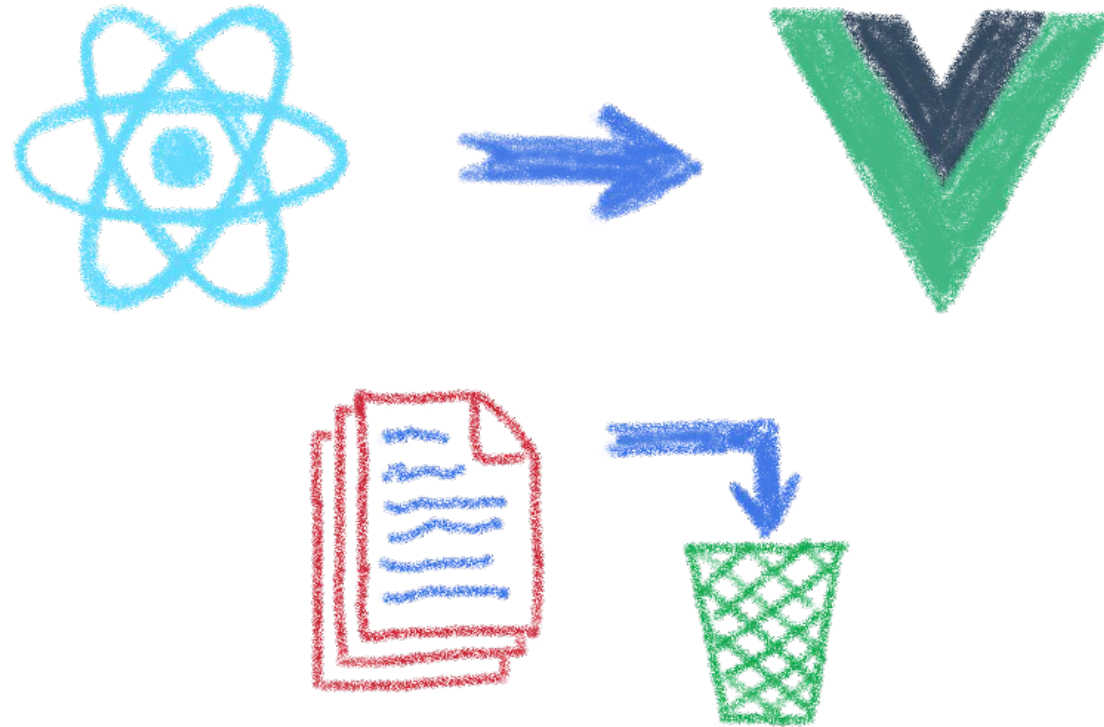
Web Components everywhere, baby!

# Do you remember AngularJS?



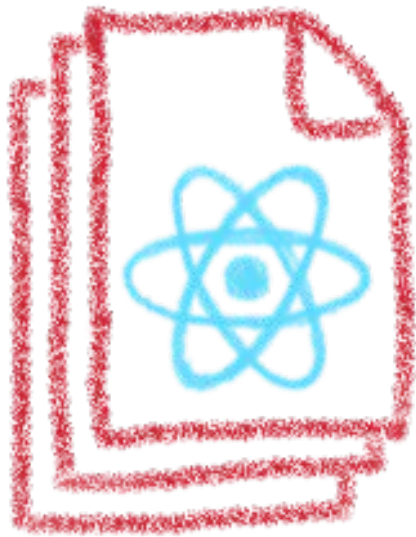
And all the code put in the trash bin  
when Angular arrived...

# The pain of switching frameworks?



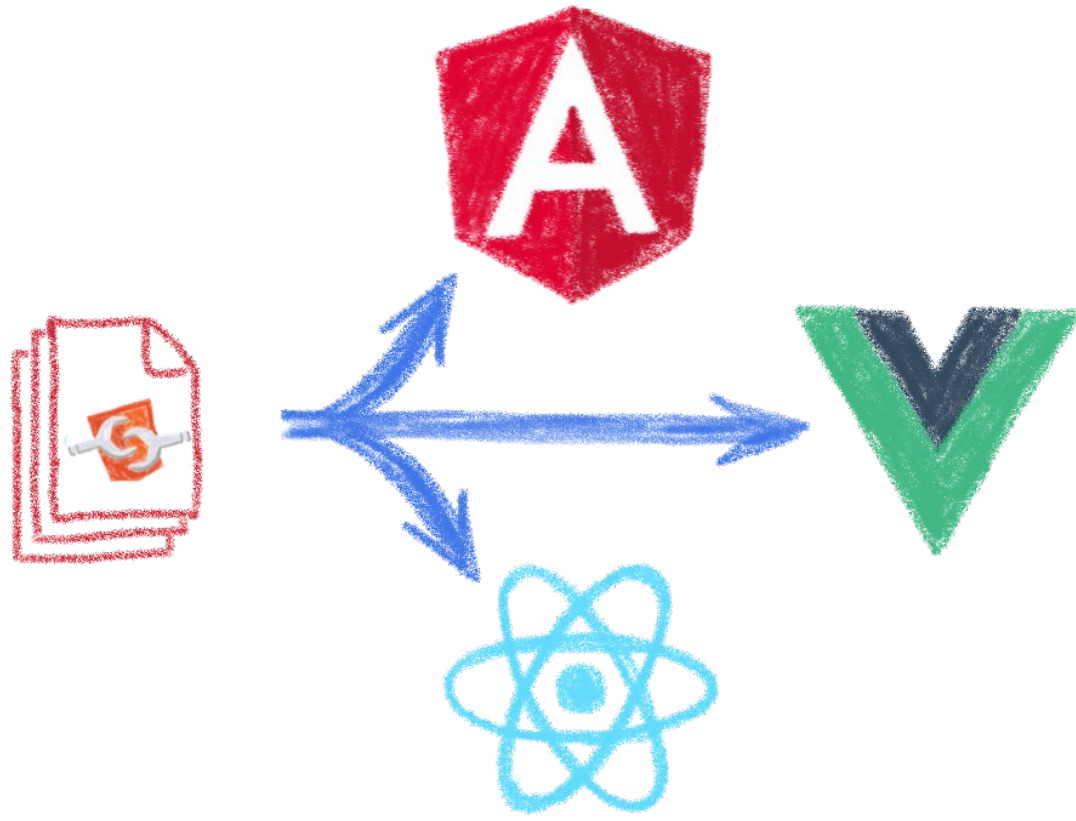
Rewriting once again your code...

# The impossibility of sharing UI code?



Between apps written with different frameworks

# Web Components change that



In a clean and standard way



# They are the interoperable alternative



Any framework... or no framework at all

# They are truly everywhere

↑ **spacexfsw**   102 points · 15 days ago

↓ The Crew Displays onboard Dragon runs Chromium with HTML, Javascript & CSS. We don't use LESS. - Sofian

We follow an agile process, we have high bar for unit test coverage and we have integration tests that runs with and without flight hardware. We also take a lot of pride in manually verifying and documenting our new features to make sure they work as intended and we have no regression. - Sofian

We use Web Components extensively. - Sofian

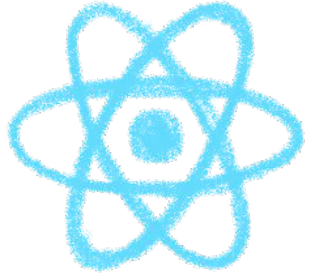
We use a reactive programming library that we developed in house. - Sofian

Different team members uses different editors, I use VSCode but I might be just a little bit biased :) - Sofian

I will have to get back but overall code is our craft here and we make sure it's clean and tidy. I wouldn't expect something too outrageous. Fair warning, we have linters on everything. - Sofian

 Even in the spaaaaaaace 

# You can have a single implementation



And it simply works everywhere

# When you need interoperability

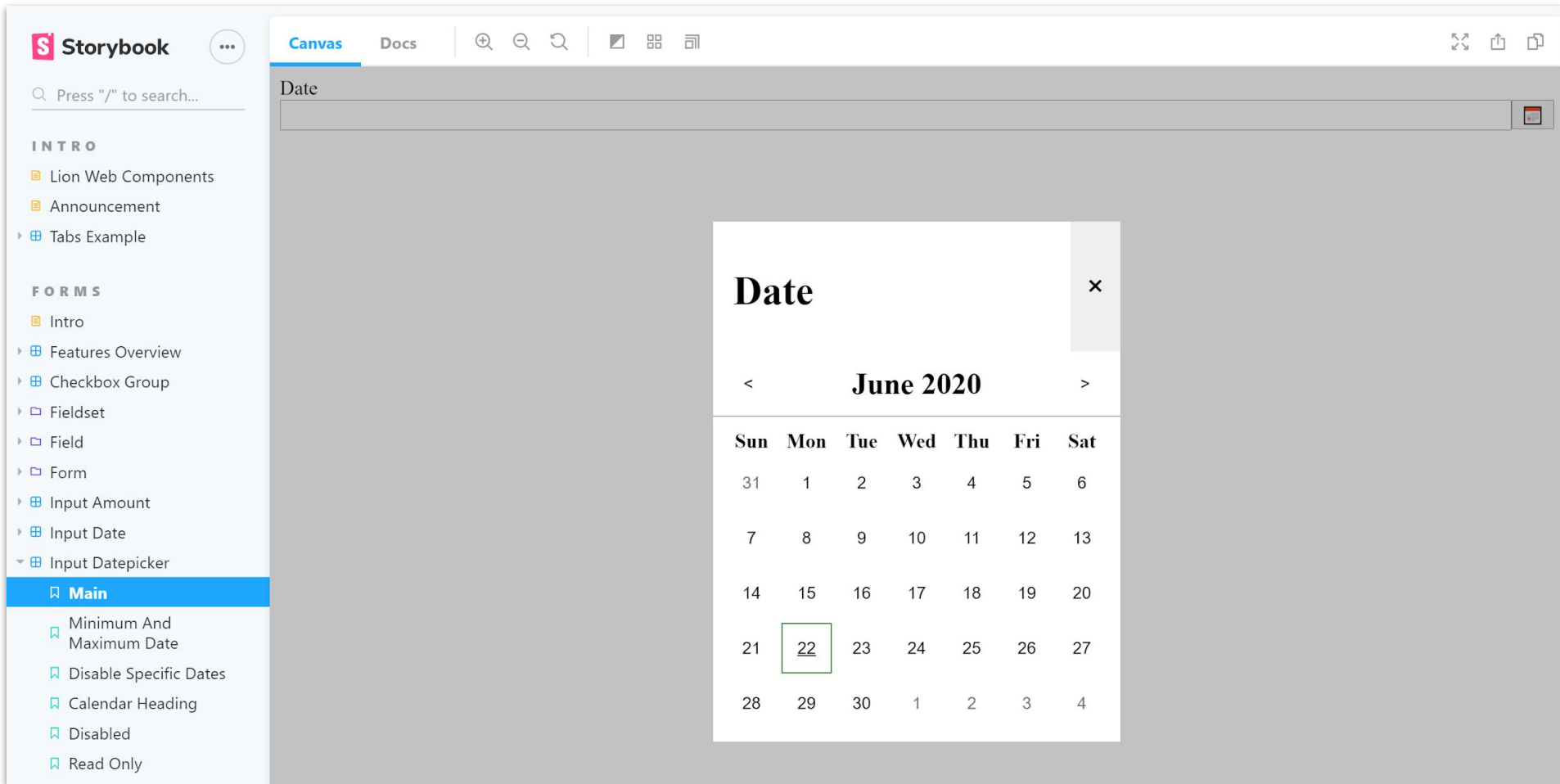


Nothing beats the standard

# But how to do it?

Designing, developing and managing  
a catalog of Web Components

# Learning from the best



<https://lion-web-components.netlify.app/>

# Learning from the best

A screenshot of a web browser showing the documentation for Clever Cloud's Web Components. The page has a light blue header with the Clever Cloud logo and navigation tabs for "Canvas", "Docs", and "Notes". A left sidebar contains a search bar and a table of contents with sections for "HOME" (Readme, Changelog, Contributing, Release), "DOCS" (Architecture Decision Records, localization, writing stories, guidelines), and "ATOMS" (a list of component tags like <cc-beta>, <cc-button>, etc.). The main content area has a title "Collection of Web Components by Clever Cloud", followed by a section "What is this?" which explains the project's purpose and lists some low-level components. Below that is a section "Why is it public?" with three numbered points explaining the motivation for sharing the code.

clever cloud

Canvas Docs Notes

Press "/" to search...

HOME

- Readme
- Changelog
- Contributing
- Release

DOCS

- Architecture Decision Records
- How to translate and localize?
- How to write stories
- Web components guidelines

ATOMS

- <cc-beta>
- <cc-button>
- <cc-datetime-relative>
- <cc-expand>
- <cc-flex-gap>
- <cc-img>

## Collection of Web Components by Clever Cloud

### What is this?

This project contains a collection of Web Components made by Clever Cloud.

Some of those components are low-level like `<cc-button>`, `<cc-input-text>` or `<cc-loader>`, the other components are more high-level and specific to Clever Cloud's domain model.

We use them on different Web UIs we have (public and internal).

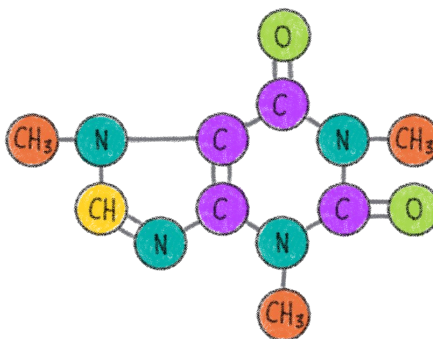
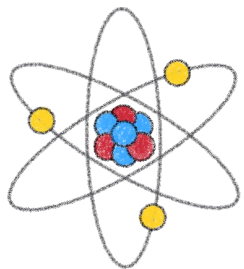
### Why is it public?

1. We want to share our knowledge and experience with Web Components along with the tooling we used to build them. We hope it will help others for their own components.
2. We use those components ourselves but we also want our clients and partners to use them in their own custom Web UIs based on our products.
3. We think it's a great way for our clients to give feedbacks (and even contributions) on small parts of our Web UIs.

<https://github.com/CleverCloud/clever-components>

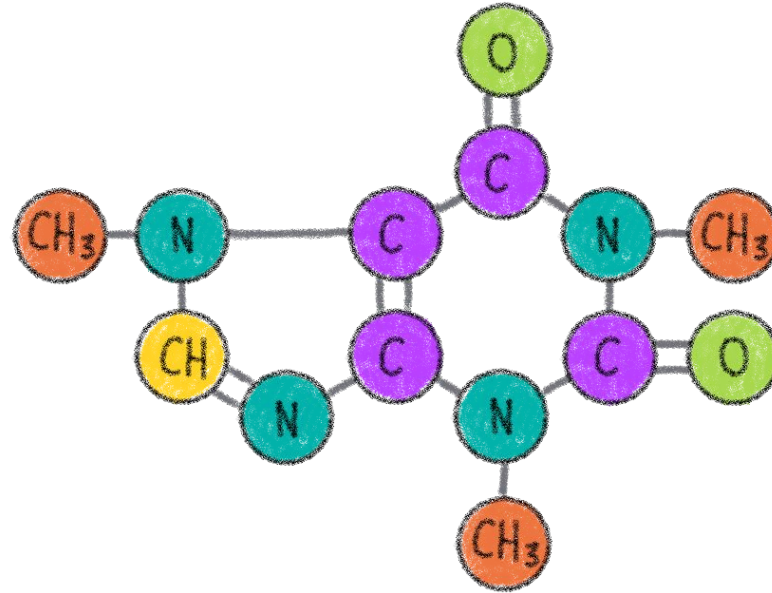
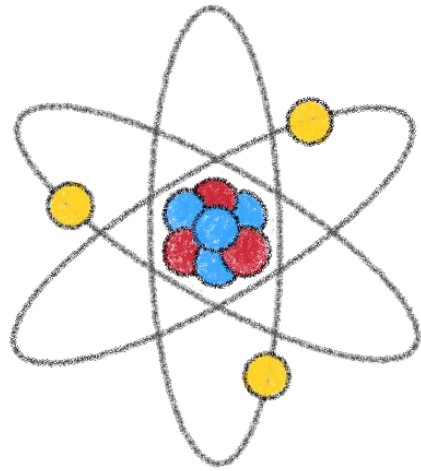
# What kind of components?

From little atomic blocs to big smart components,  
and everything in between



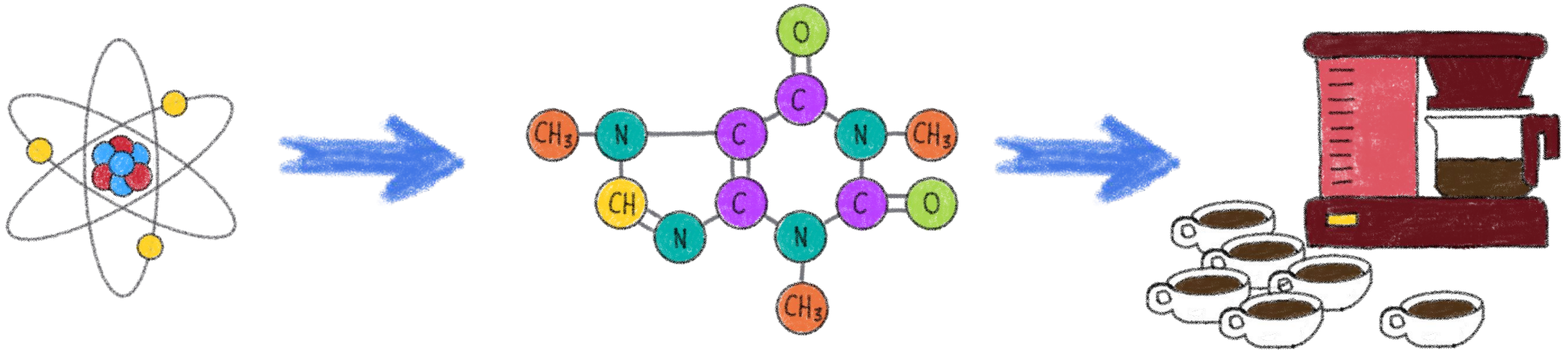


# A matter of size and complexity



What kind(s) of components you want to build

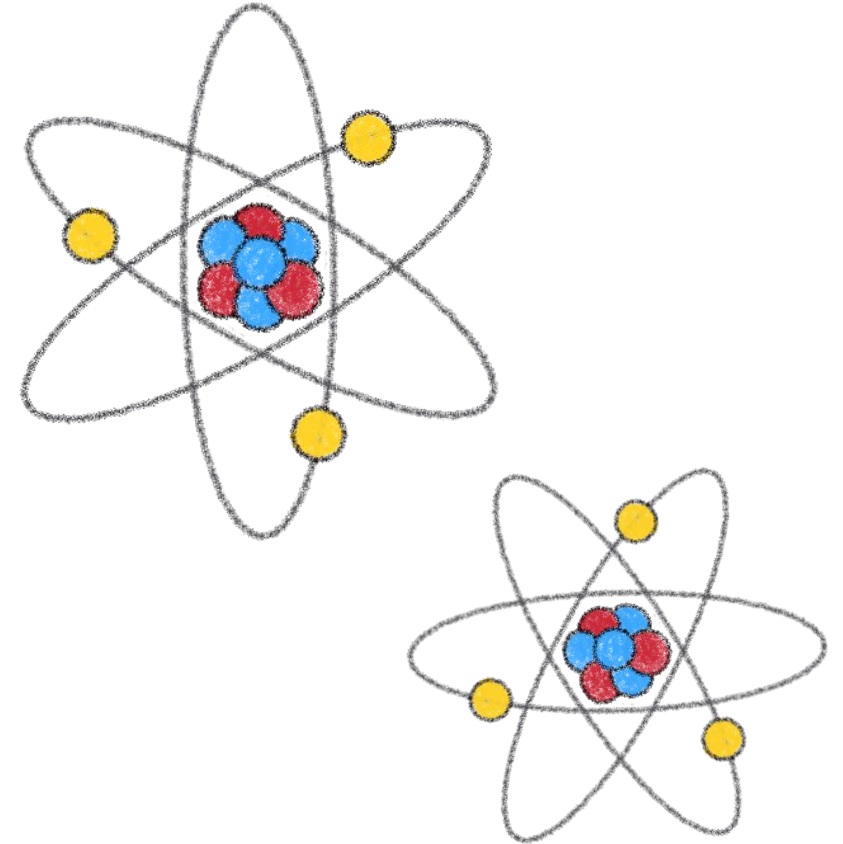
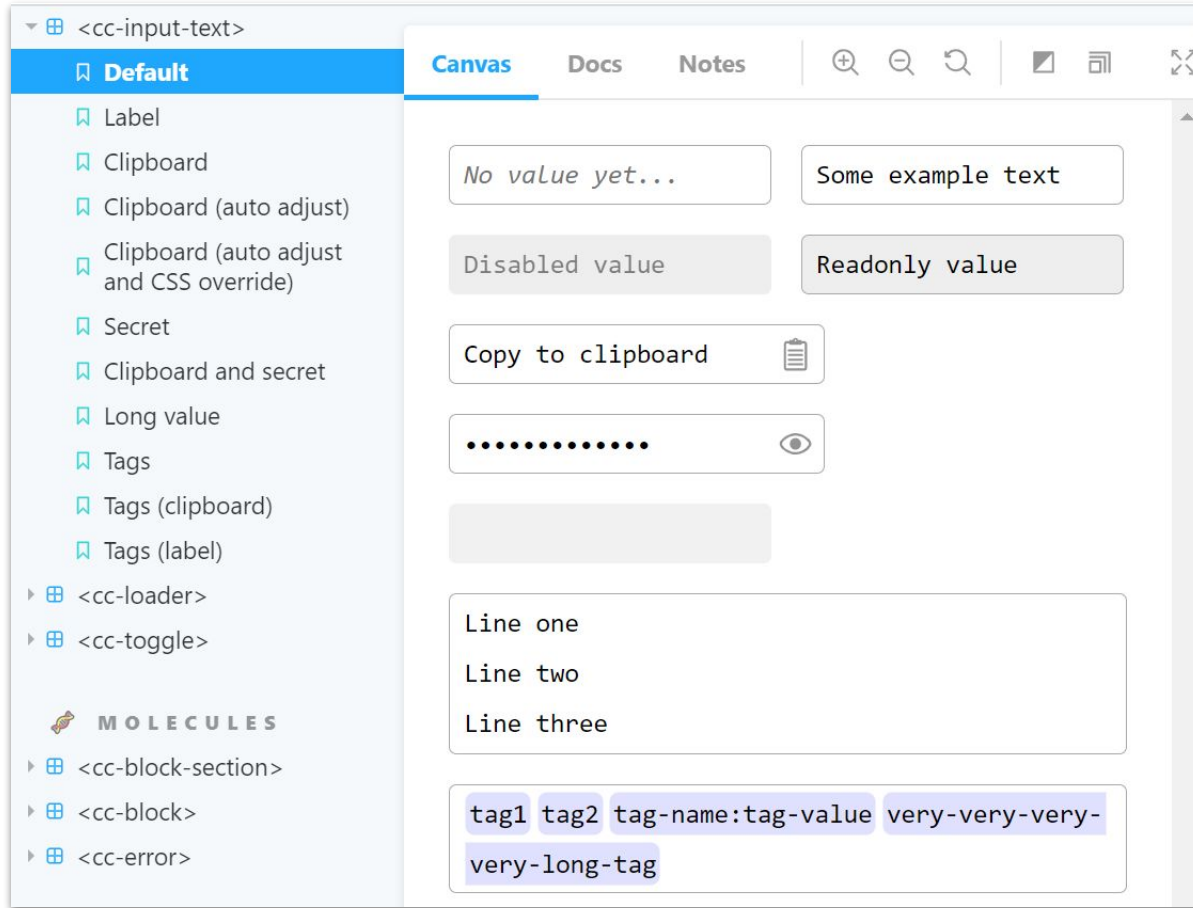
# Build from the bottom and go up



Eat your own dog food

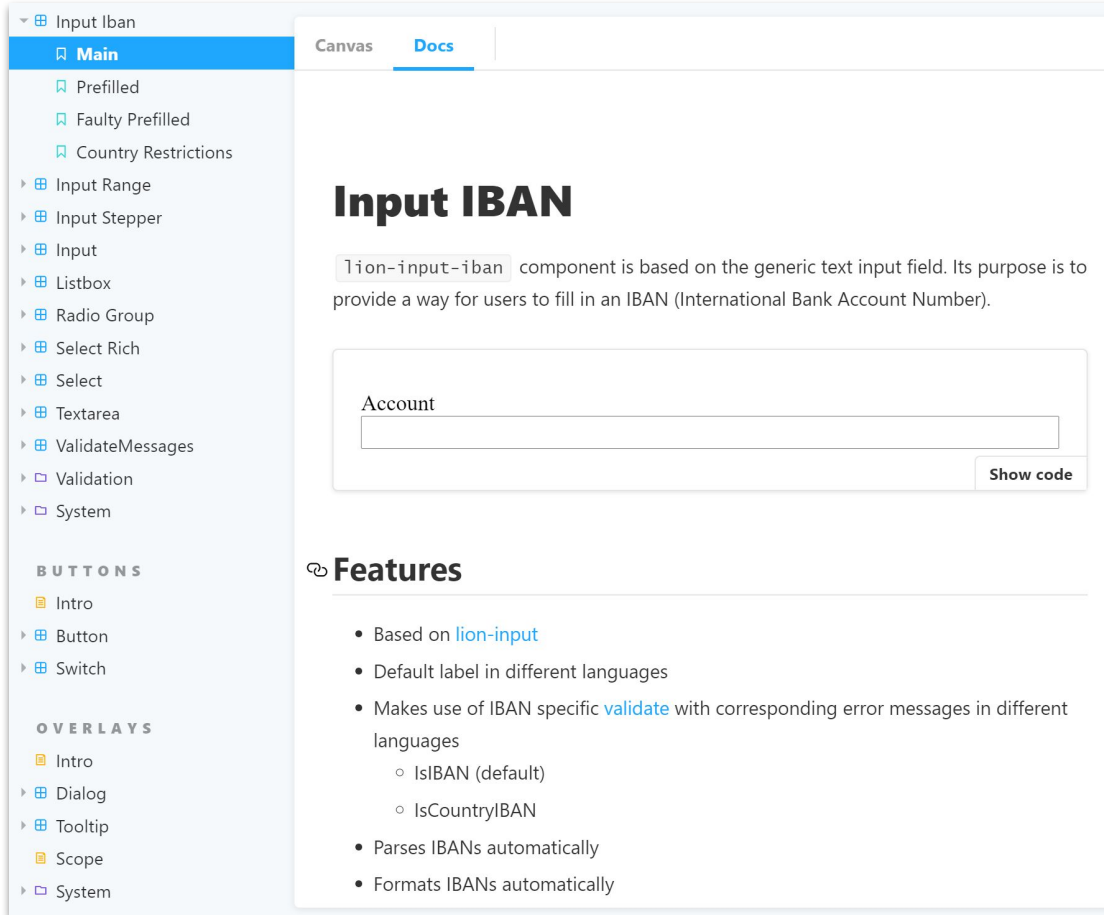


# And how to choose the atoms?

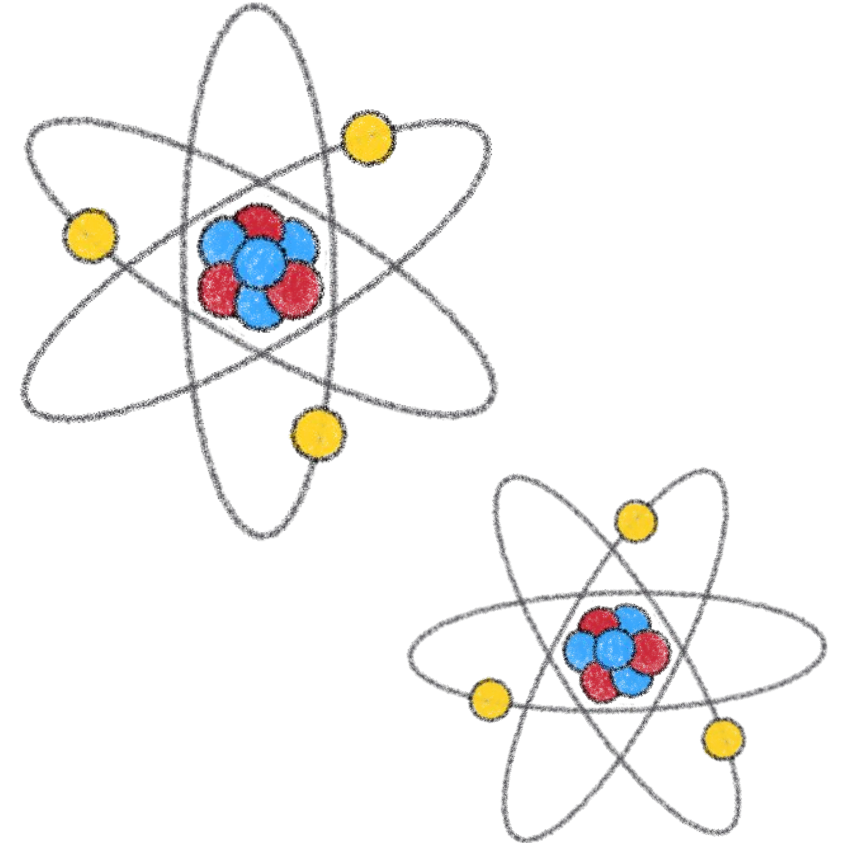


Flexibility and configurability are key

# And how to choose the atoms?

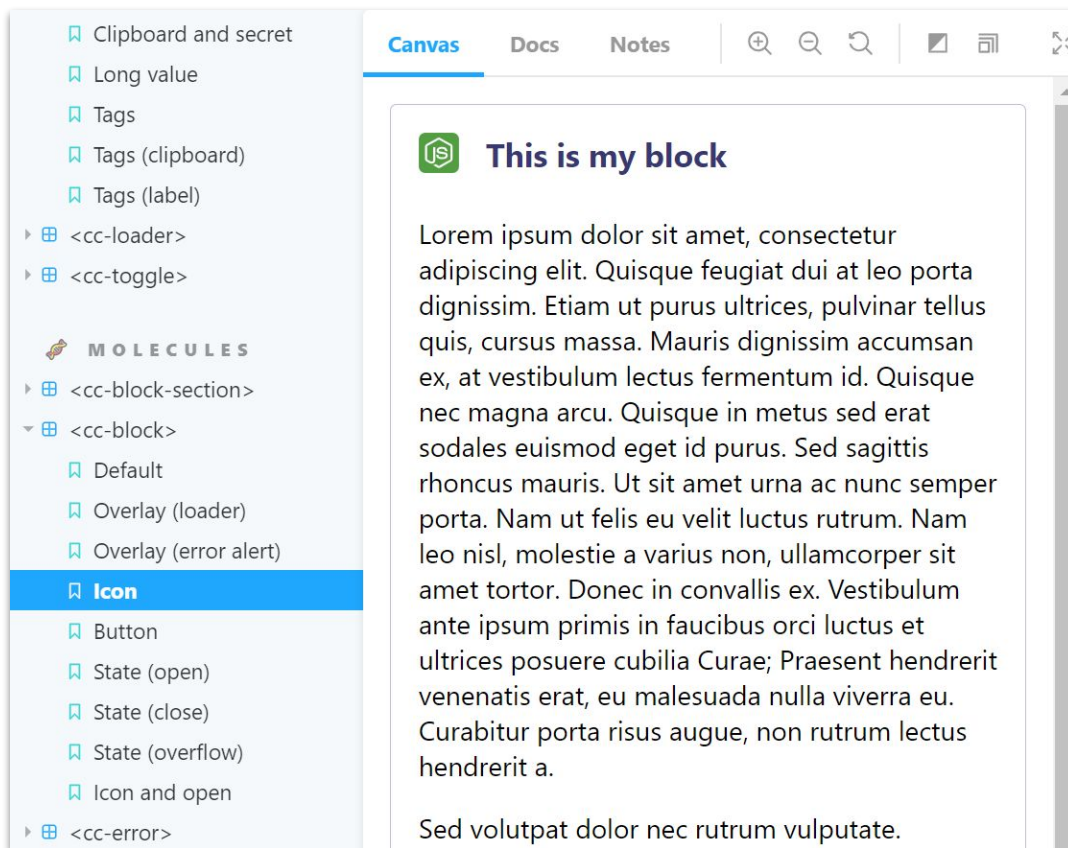


The screenshot shows a documentation page for the 'Input IBAN' component. On the left is a navigation sidebar with categories like 'Main', 'Buttons', and 'Overlays'. The main content area has a 'Docs' tab selected. The title is 'Input IBAN'. Below the title is a paragraph: 'lion-input-iban component is based on the generic text input field. Its purpose is to provide a way for users to fill in an IBAN (International Bank Account Number)'. Below this is a visual representation of the component: a text input field with the label 'Account' and a 'Show code' button. Underneath is a 'Features' section with a list of bullet points: 'Based on lion-input', 'Default label in different languages', 'Makes use of IBAN specific validate with corresponding error messages in different languages' (with sub-points for 'IsIBAN (default)' and 'IsCountryIBAN'), 'Parses IBANs automatically', and 'Formats IBANs automatically'.



Encode often used patterns

# And what about the molecules?



Clipboard and secret  
Long value  
Tags  
Tags (clipboard)  
Tags (label)  
> <cc-loader>  
> <cc-toggle>

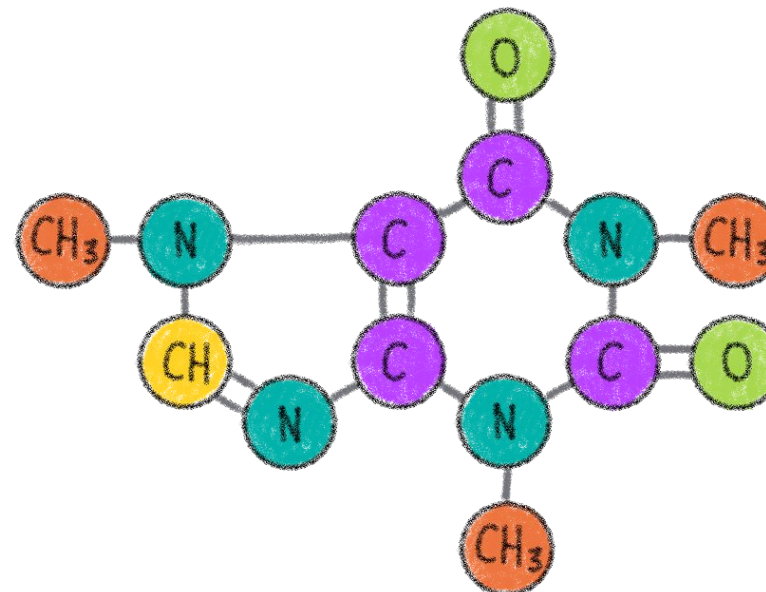
**MOLECULES**

> <cc-block-section>  
▼ <cc-block>  
  Default  
  Overlay (loader)  
  Overlay (error alert)  
  **Icon**  
  Button  
  State (open)  
  State (close)  
  State (overflow)  
  Icon and open  
> <cc-error>

**This is my block**

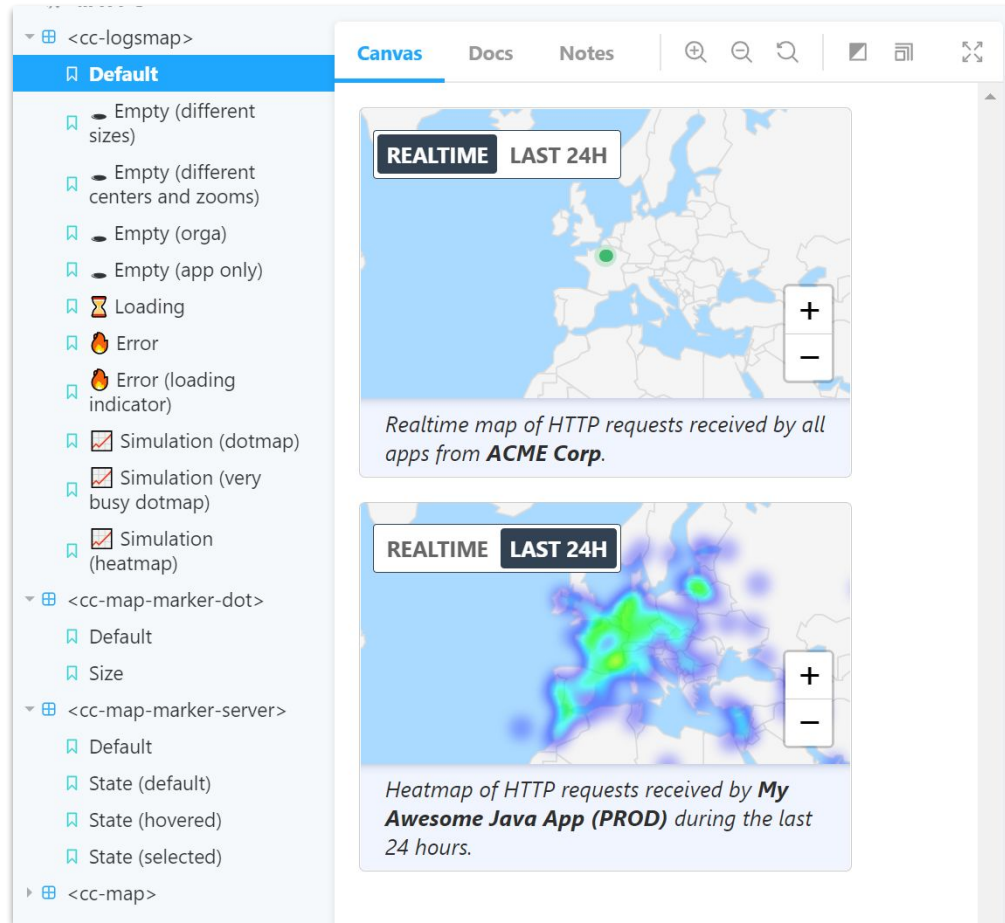
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque feugiat dui at leo porta dignissim. Etiam ut purus ultrices, pulvinar tellus quis, cursus massa. Mauris dignissim accumsan ex, at vestibulum lectus fermentum id. Quisque nec magna arcu. Quisque in metus sed erat sodales euismod eget id purus. Sed sagittis rhoncus mauris. Ut sit amet urna ac nunc semper porta. Nam ut felis eu velit luctus rutrum. Nam leo nisl, molestie a varius non, ullamcorper sit amet tortor. Donec in convallis ex. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent hendrerit venenatis erat, eu malesuada nulla viverra eu. Curabitur porta risus augue, non rutrum lectus hendrerit a.

Sed volutpat dolor nec rutrum vulputate.



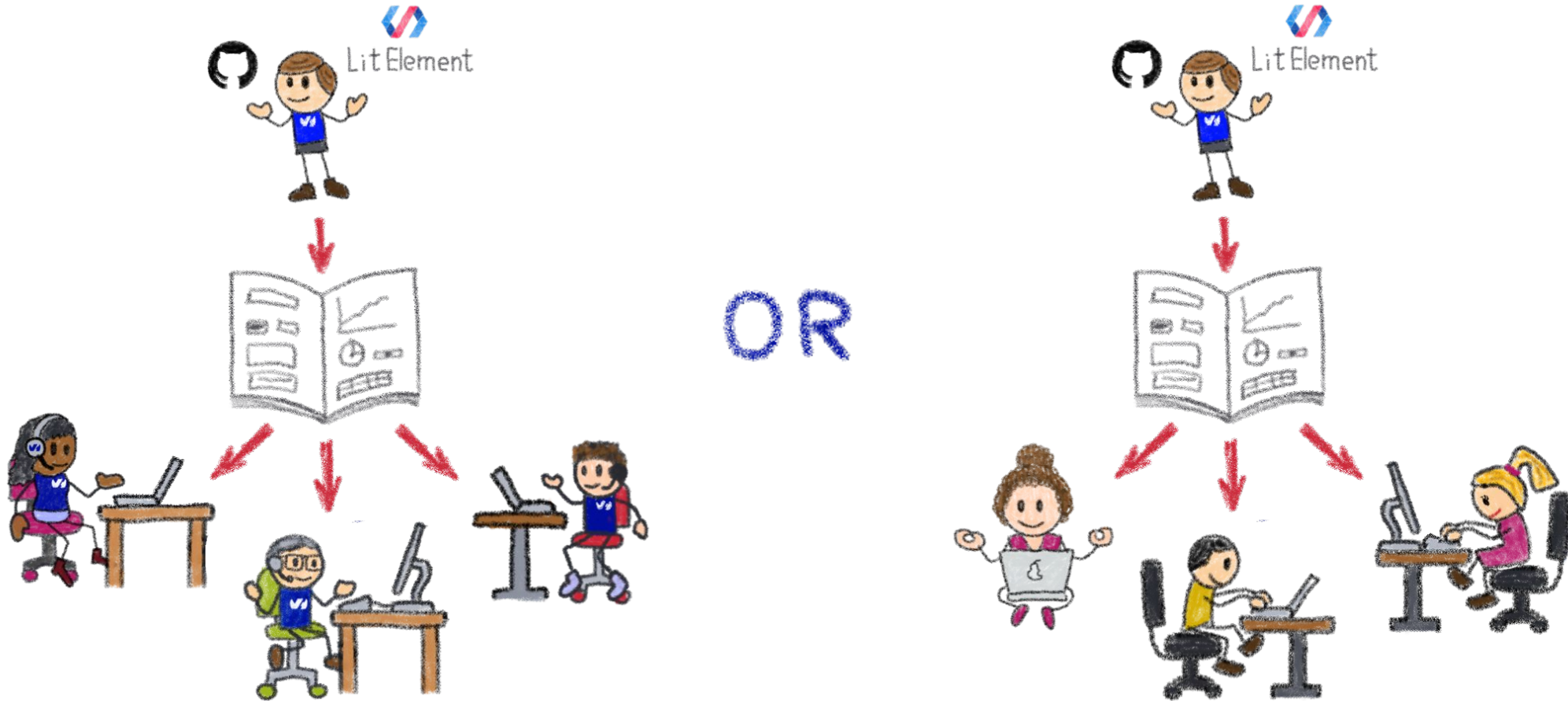
Capitalize on your atoms  
Keep the flexibility and configurability

# Big smart business components



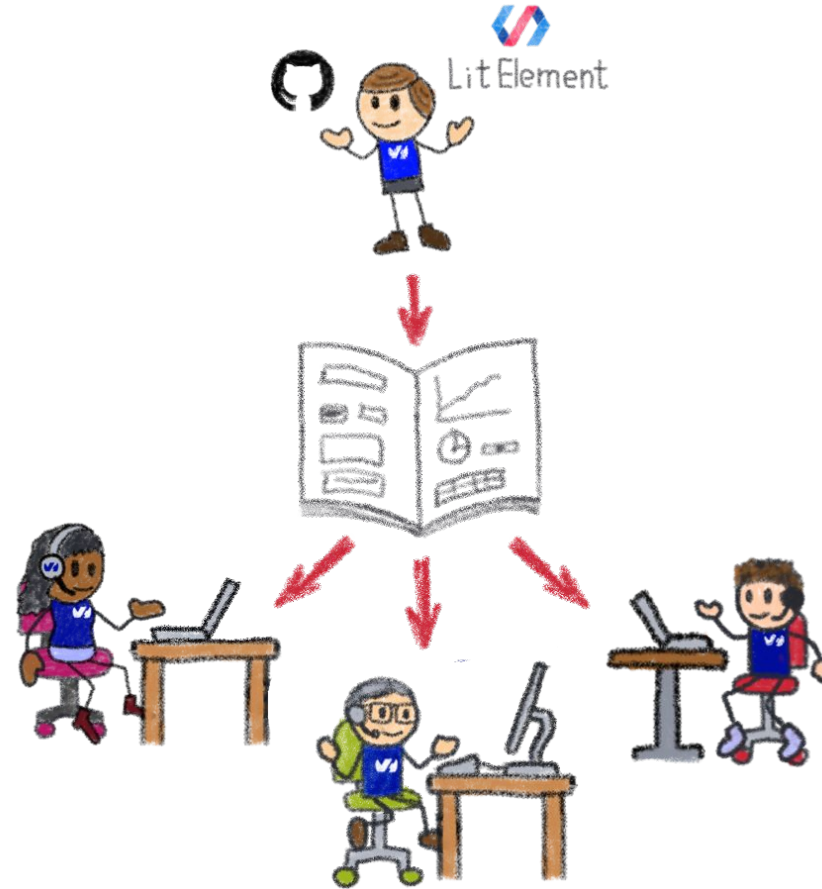
## Encoding your business logic

# Internal or external customers?



Who are your target users?

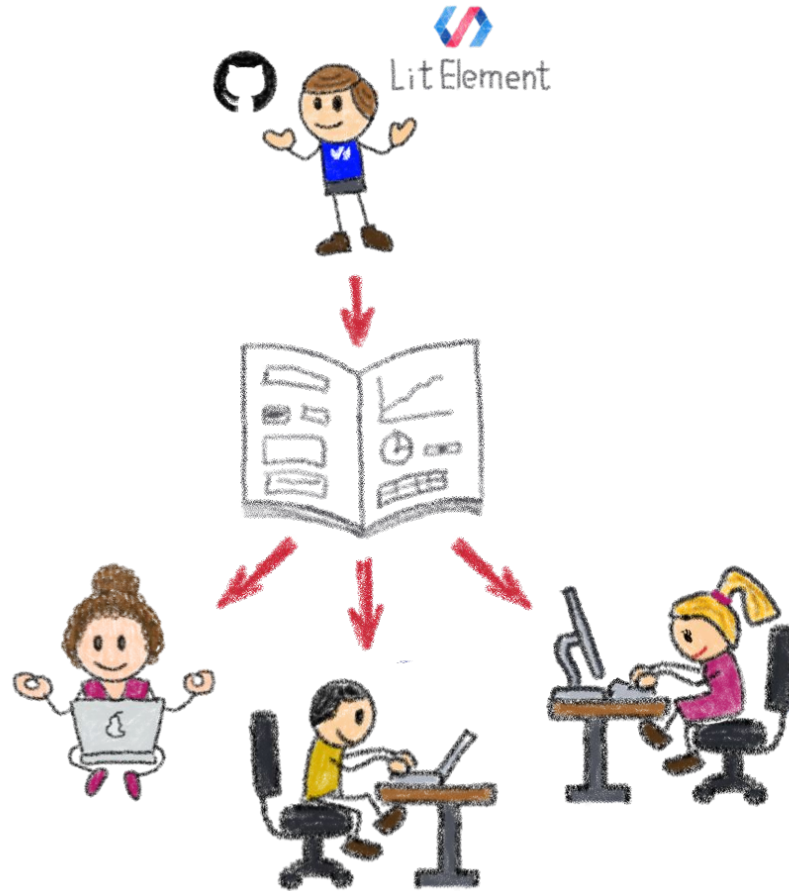
# Internal customers need off-the-shelf components



A well defined and coherent look-and-feel



# External customers need to be able to tweak



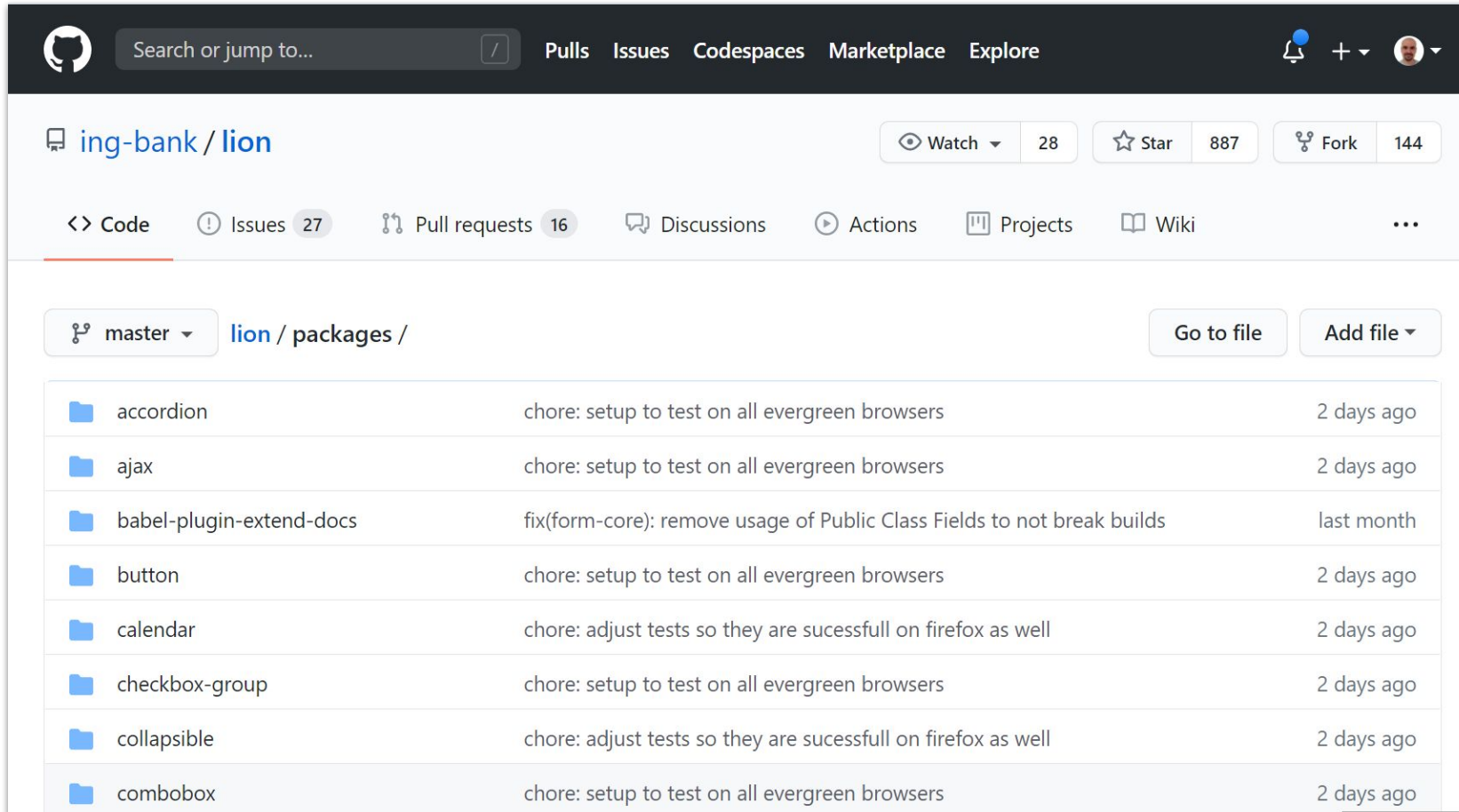
Theming and customizing components

# How to organize the catalog

## Packages, imports and pragmatism



# A single repository



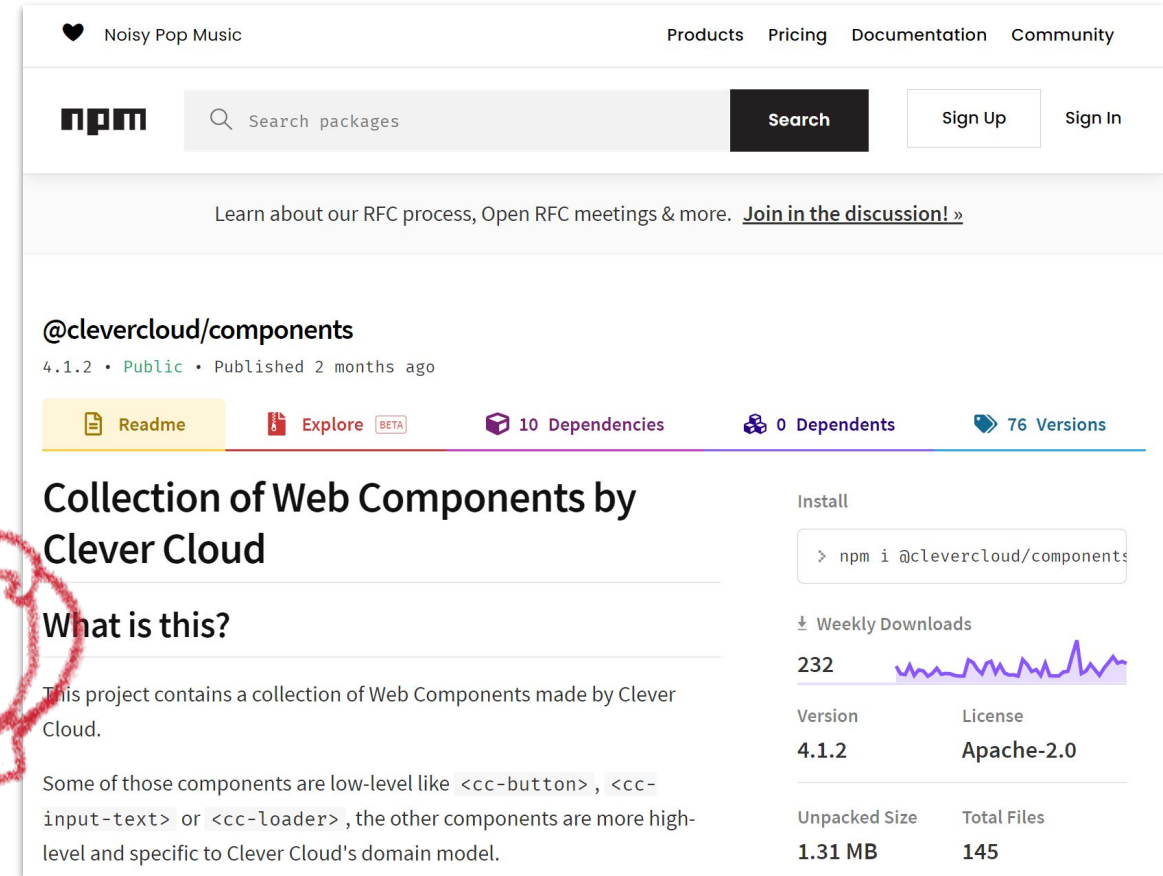
Single source of truth for the catalog

# Two schools of thought

Lion web components is logically organized in groups of systems.

The accessibility column indicates whether the functionality is accessible in its core. Aspects like styling and content determine actual accessibility in usage.

Package	Version	Description	Accessibility
-- Form System --		A system that lets you make complex forms with ease, including: validation, translations.	✓
combobox	npm v0.1.2	Text box controlling popup listbox	✓
form	npm v0.7.1	Wrapper for multiple form elements	✓
form-core	npm v0.6.3	Core functionality for all form controls	✓
form-integrations	npm v0.3.5	Shows form elements in an integrated way	✓
fieldset	npm v0.15.1	Group for form inputs	✓
checkbox-group	npm v0.12.1	Group of checkboxes	✓
input	npm v0.16.1	Input element for strings	✓
input-amount	npm v0.8.1	Input element for amounts	✓
input-date	npm v0.8.1	Input element for dates	✓
input-datepicker	npm v0.17.0	Input element for dates with a datepicker	✓
input-email	npm v0.9.1	Input element for e-mails	✓
input-iban	npm v0.16.1	Input element for IBANs	✓
input-range	npm v0.5.1	Input element for a range of values	✓



Noisy Pop Music Products Pricing Documentation Community

npm Search packages Search Sign Up Sign In

Learn about our RFC process, Open RFC meetings & more. [Join in the discussion!](#)

@clevercloud/components  
4.1.2 • Public • Published 2 months ago

Readme Explore BETA 10 Dependencies 0 Dependents 76 Versions

## Collection of Web Components by Clever Cloud

### What is this?

This project contains a collection of Web Components made by Clever Cloud.

Some of those components are low-level like `<cc-button>`, `<cc-input-text>` or `<cc-loader>`, the other components are more high-level and specific to Clever Cloud's domain model.

Install

```
> npm i @clevercloud/components
```

Weekly Downloads

232

Version	License
4.1.2	Apache-2.0

Unpacked Size	Total Files
1.31 MB	145

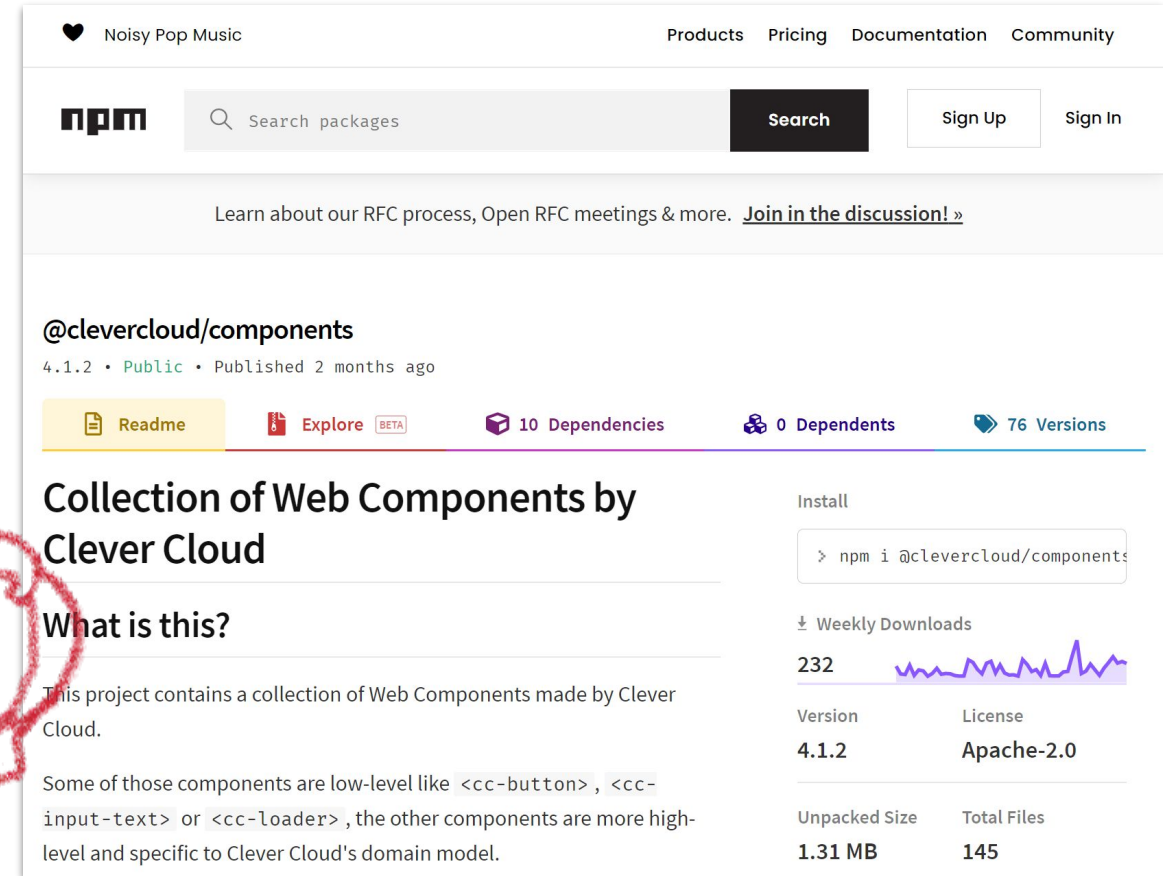
## A packet per component or a global one

# Two schools of thought

Lion web components is logically organized in groups of systems.

The accessibility column indicates whether the functionality is accessible in its core. Aspects like styling and content determine actual accessibility in usage.

Package	Version	Description	Accessibility
-- Form System --		A system that lets you make complex forms with ease, including: validation, translations.	✓
combobox	npm v0.1.2	Text box controlling popup listbox	✓
form	npm v0.7.1	Wrapper for multiple form elements	✓
form-core	npm v0.6.3	Core functionality for all form controls	✓
form-integrations	npm v0.3.5	Shows form elements in an integrated way	✓
fieldset	npm v0.15.1	Group for form inputs	✓
checkbox-group	npm v0.12.1	Group of checkboxes	✓
input	npm v0.10.1	Input element for strings	✓
input-amount	npm v0.8.1	Input element for amounts	✓
input-date	npm v0.8.1	Input element for dates	✓
input-datepicker	npm v0.12.0	Input element for dates with a datepicker	✓
input-email	npm v0.9.1	Input element for e-mails	✓
input-iban	npm v0.10.1	Input element for IBANs	✓
input-range	npm v0.5.1	Input element for a range of values	✓



Noisy Pop Music Products Pricing Documentation Community

npm Search packages Search Sign Up Sign In

Learn about our RFC process, Open RFC meetings & more. [Join in the discussion!](#)

@clevercloud/components  
4.1.2 • Public • Published 2 months ago

Readme Explore BETA 10 Dependencies 0 Dependents 76 Versions

## Collection of Web Components by Clever Cloud

### What is this?

This project contains a collection of Web Components made by Clever Cloud.

Some of those components are low-level like `<cc-button>`, `<cc-input-text>` or `<cc-loader>`, the other components are more high-level and specific to Clever Cloud's domain model.

Install

```
> npm i @clevercloud/components
```

Weekly Downloads

232

Version	License
4.1.2	Apache-2.0

Unpacked Size	Total Files
1.31 MB	145

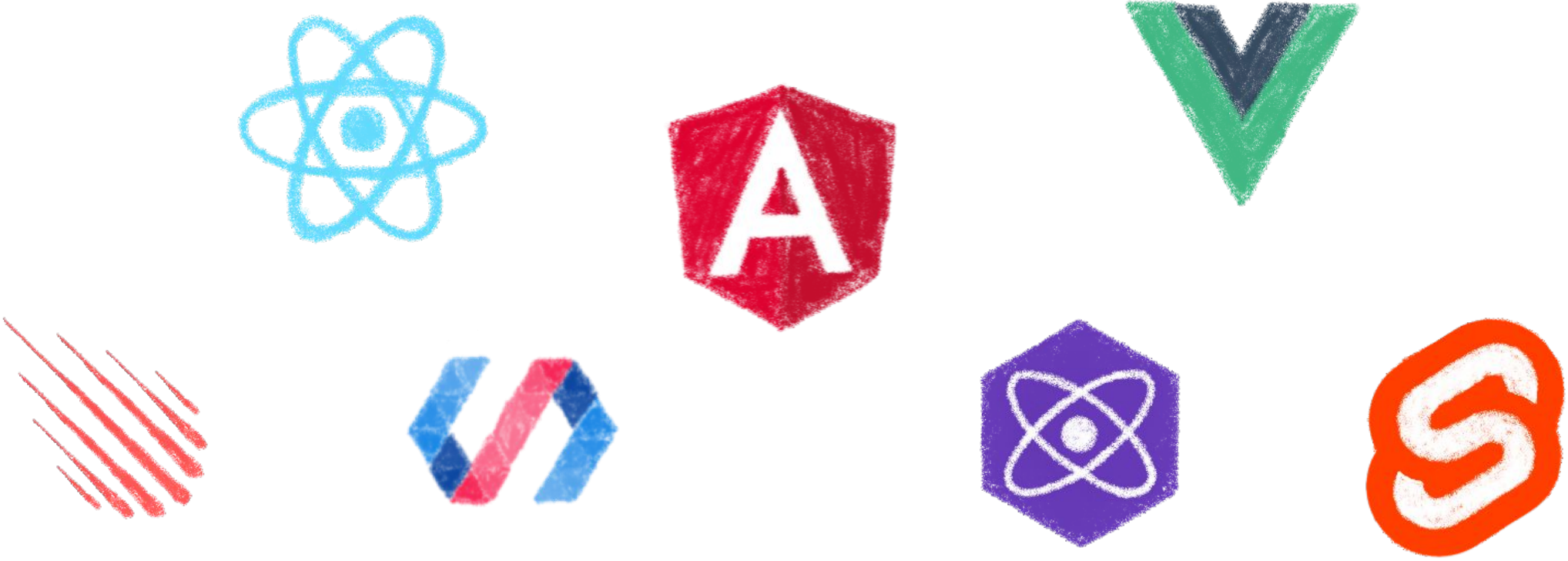
## Individual versioning vs global one

# Driving-up adoption

Making devs use your components

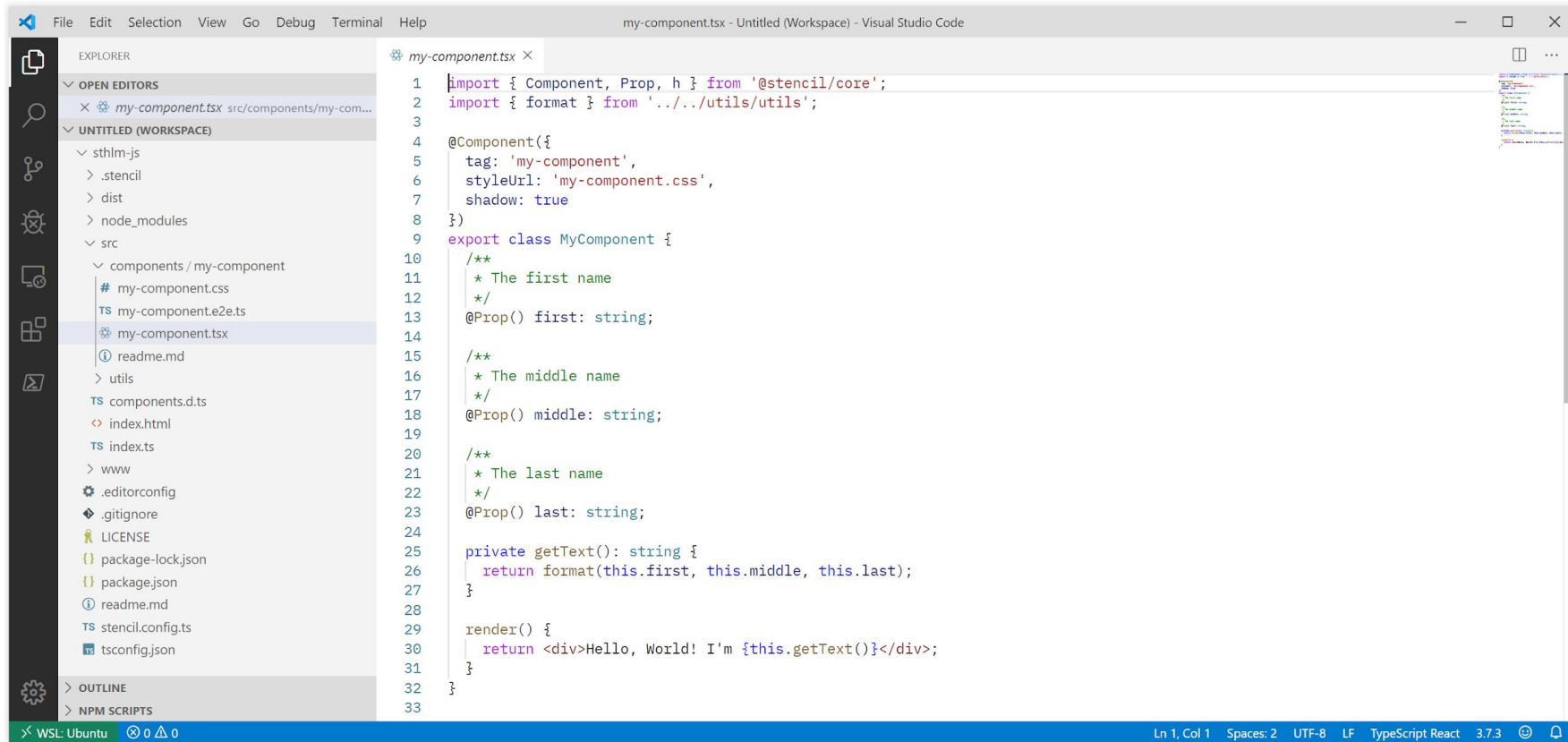


# Think who are your target users



Users of any framework current or future...

# They aren't used to your library



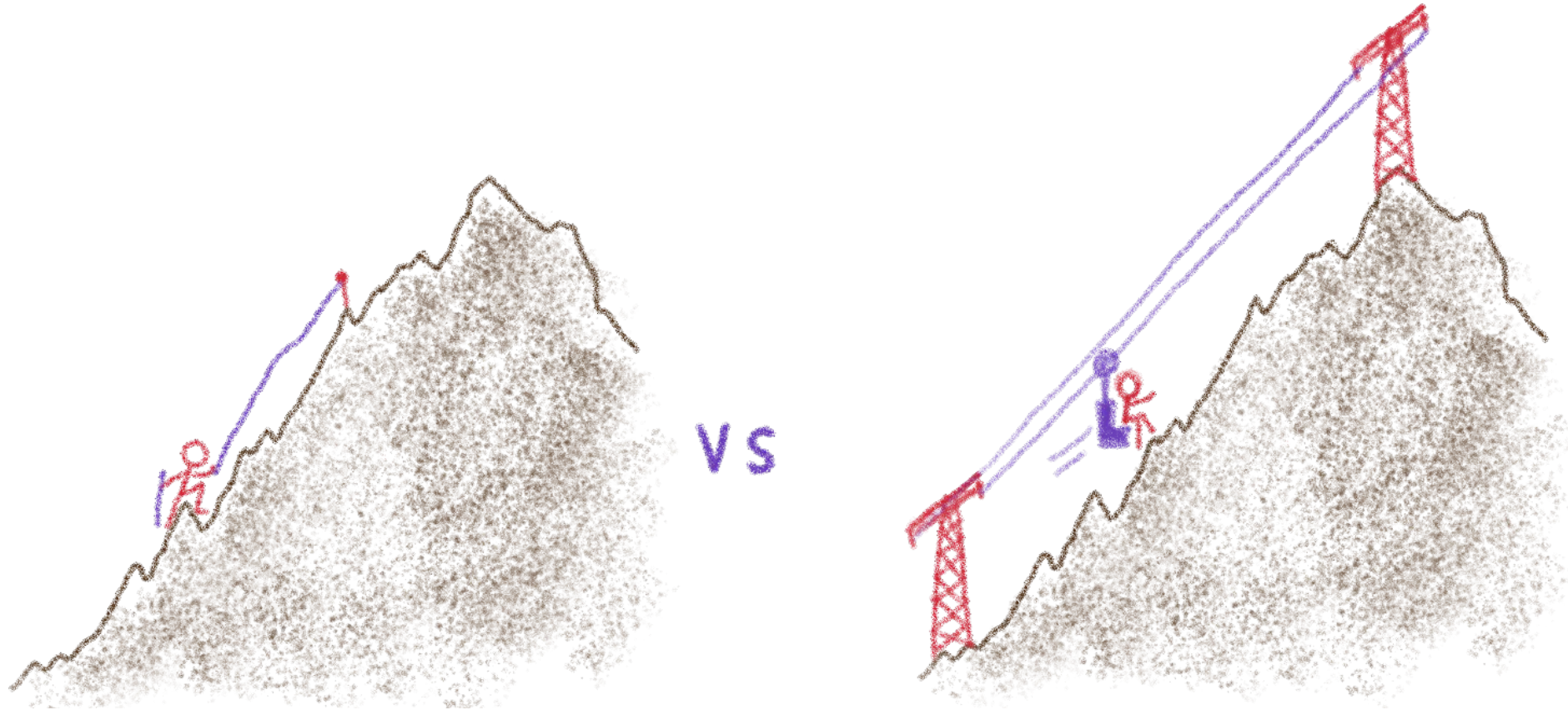
```
1 import { Component, Prop, h } from '@stencil/core';
2 import { format } from '../utils/utils';
3
4 @Component({
5   tag: 'my-component',
6   styleUrls: 'my-component.css',
7   shadow: true
8 })
9 export class MyComponent {
10   /**
11    * The first name
12    */
13   @Prop() first: string;
14
15   /**
16    * The middle name
17    */
18   @Prop() middle: string;
19
20   /**
21    * The last name
22    */
23   @Prop() last: string;
24
25   private getText(): string {
26     return format(this.first, this.middle, this.last);
27   }
28
29   render() {
30     return <div>Hello, World! I'm {this.getText()}</div>;
31   }
32 }
33
```



And they shouldn't need to be



# Go the extra mile to drive up adoption



So they don't need to do it

# Make it easy to use

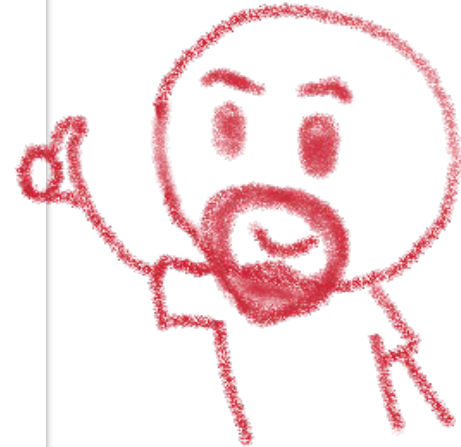
## How to install

```
npm i @lion/<package-name>
```

## How to use

## Use a Web Component

```
<script type="module">  
  import '@lion/input/lion-input.js';  
</script>  
  
<lion-input name="firstName"></lion-input>
```



As easy as a HTML tag

# Document every composant

## Input IBAN

`lion-input-iban` component is based on the generic text input field. Its purpose is to provide a way for users to fill in an IBAN (International Bank Account Number).

```
import { html } from 'lit-html';
import { loadDefaultFeedbackMessages } from '@lion/validate-messages';
import { IsCountryIBAN } from './src/validators.js';

import './lion-input-iban.js';

export default {
  title: 'Forms/Input Iban',
};

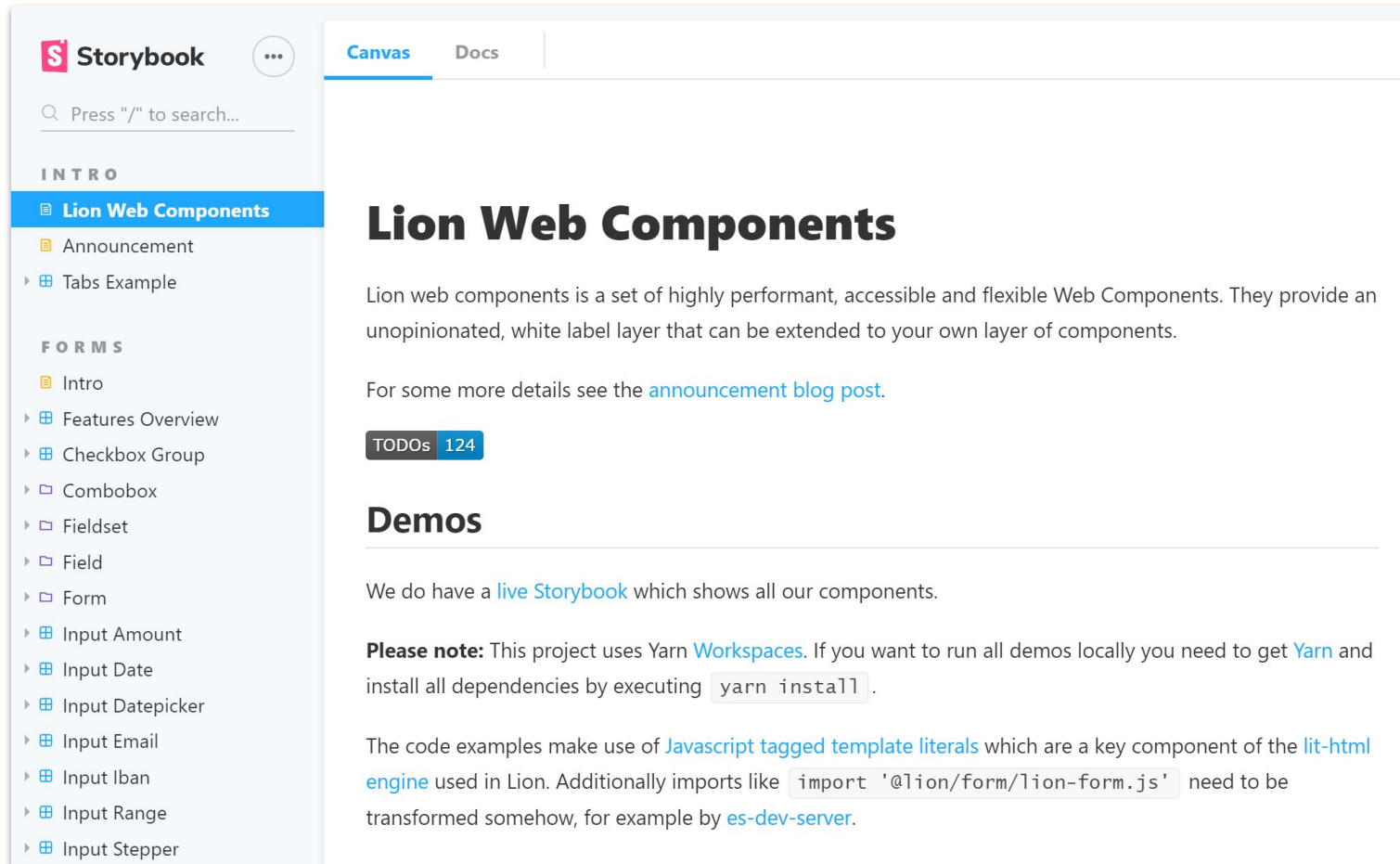
loadDefaultFeedbackMessages();

export const main = () => {
  return html` <lion-input-iban label="Account" name="account"></lion-input-iban> `;
};
```



How to use, inputs/outputs, examples...

# Documentation isn't enough

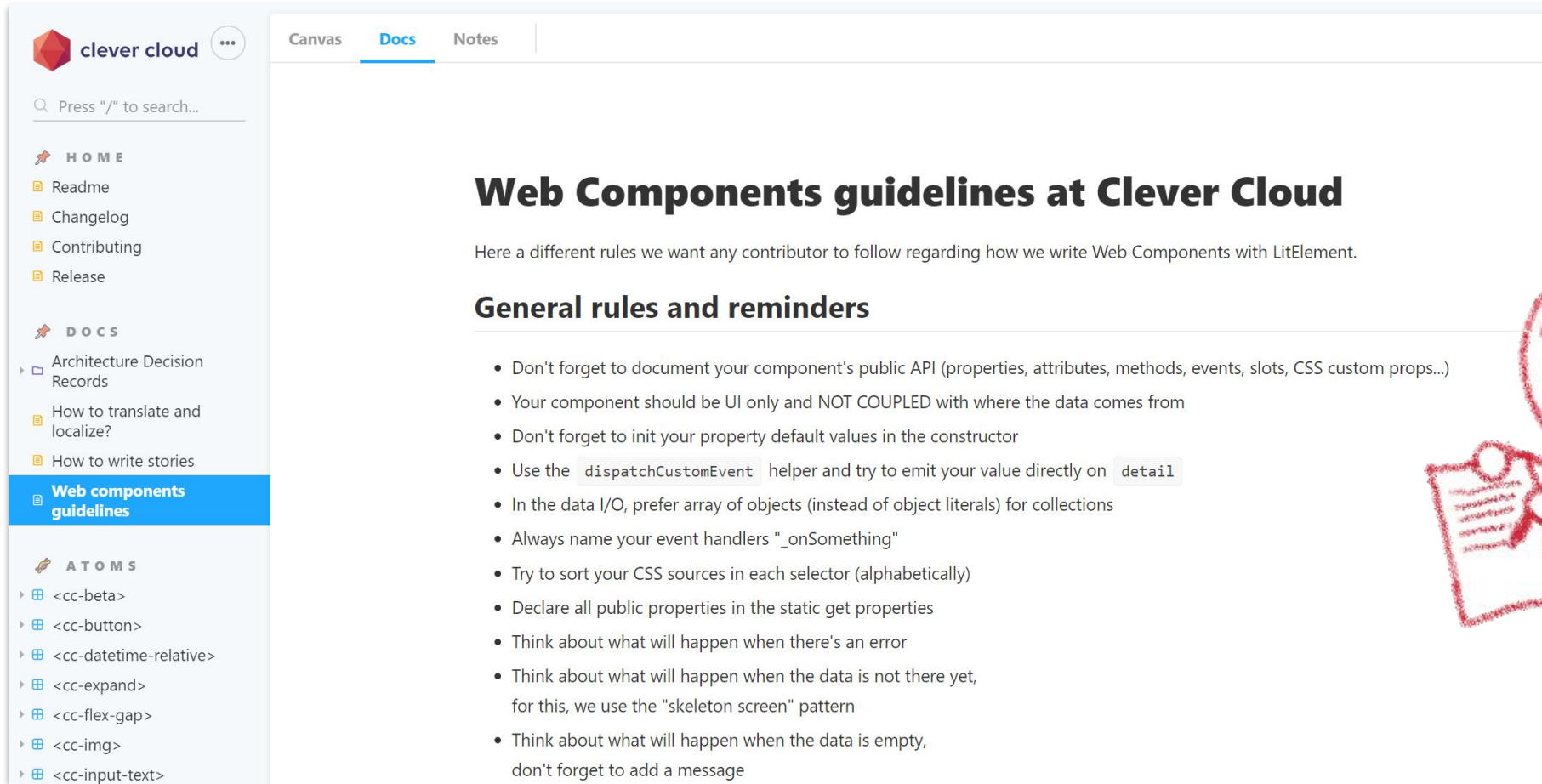


The screenshot shows the Storybook website for Lion Web Components. The left sidebar contains a navigation menu with sections for 'INTRO' (Lion Web Components, Announcement, Tabs Example) and 'FORMS' (Intro, Features Overview, Checkbox Group, Combobox, Fieldset, Field, Form, Input Amount, Input Date, Input Datpicker, Input Email, Input Iban, Input Range, Input Stepper). The main content area is titled 'Lion Web Components' and includes a description, a link to an announcement blog post, a 'TODOs 124' badge, and a 'Demos' section with a link to a live Storybook.



 **Storybook** make adoption easy

# Keeping a coherent writing style



The screenshot shows the 'Web Components guidelines at Clever Cloud' page. The left sidebar contains a navigation menu with sections: HOME (Readme, Changelog, Contributing, Release), DOCS (Architecture Decision Records, How to translate and localize?, How to write stories, **Web components guidelines**), and ATOMS (<cc-beta>, <cc-button>, <cc-datetime-relative>, <cc-expand>, <cc-flex-gap>, <cc-img>, <cc-input-text>). The main content area has the title 'Web Components guidelines at Clever Cloud', a sub-header 'General rules and reminders', and a list of guidelines.

## Web Components guidelines at Clever Cloud

Here are different rules we want any contributor to follow regarding how we write Web Components with LitElement.

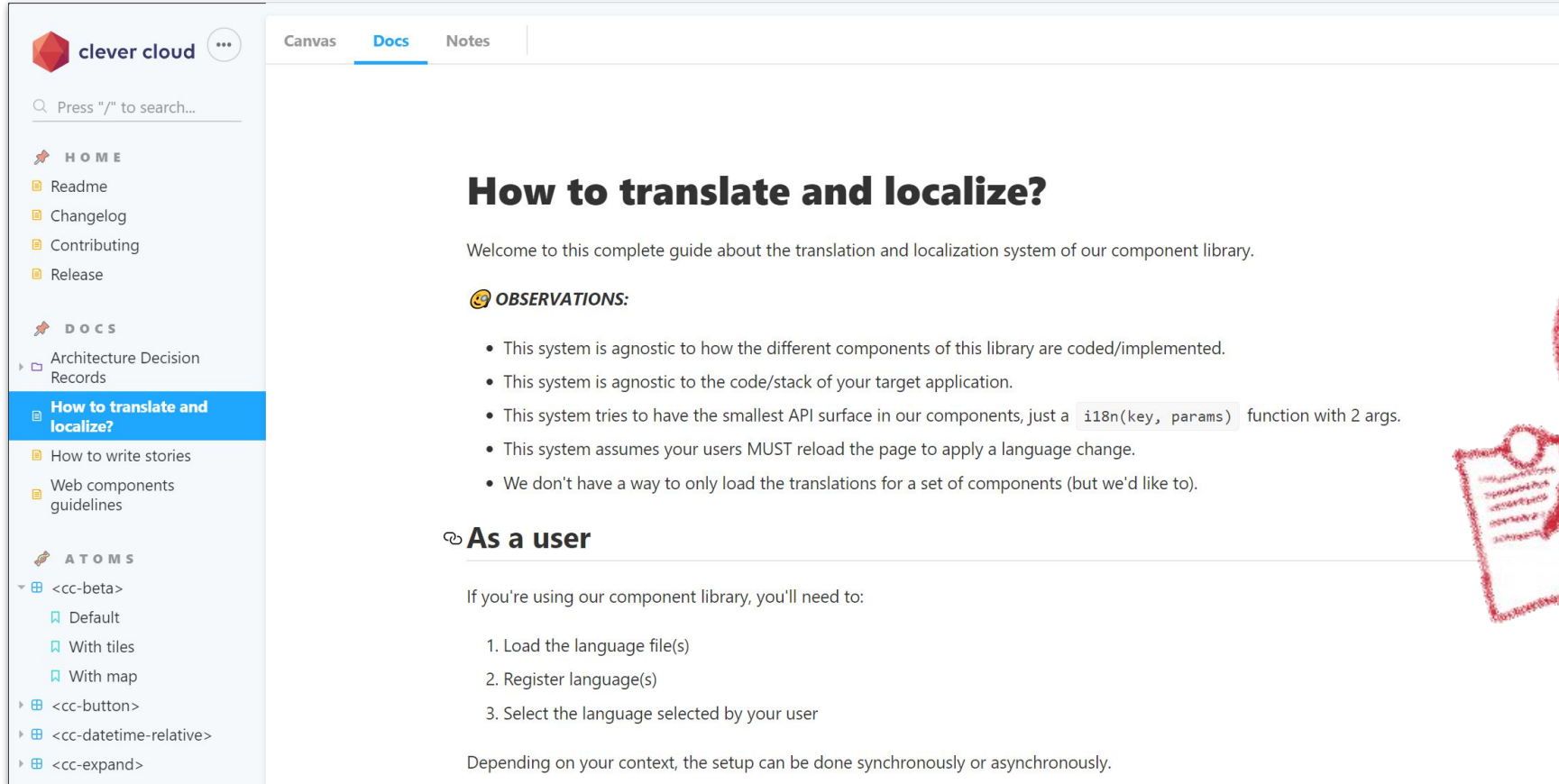
### General rules and reminders

- Don't forget to document your component's public API (properties, attributes, methods, events, slots, CSS custom props...)
- Your component should be UI only and NOT COUPLED with where the data comes from
- Don't forget to init your property default values in the constructor
- Use the `dispatchCustomEvent` helper and try to emit your value directly on `detail`
- In the data I/O, prefer array of objects (instead of object literals) for collections
- Always name your event handlers "\_onSomething"
- Try to sort your CSS sources in each selector (alphabetically)
- Declare all public properties in the static get properties
- Think about what will happen when there's an error
- Think about what will happen when the data is not there yet, for this, we use the "skeleton screen" pattern
- Think about what will happen when the data is empty, don't forget to add a message



## Write down your guidelines

# i18n shouldn't be an afterthought



The screenshot shows the 'clever cloud' documentation interface. The left sidebar contains a navigation menu with sections: HOME (Readme, Changelog, Contributing, Release), DOCS (Architecture Decision Records, 'How to translate and localize?' (highlighted), How to write stories, Web components guidelines), and ATOMS (<cc-beta>, <cc-button>, <cc-datetime-relative>, <cc-expand>). The main content area is titled 'How to translate and localize?' and includes a welcome message, a section for 'OBSERVATIONS' with five bullet points, and a section for 'As a user' with a list of three steps. A red hand-drawn character holding a document is overlaid on the right side of the screenshot.

**How to translate and localize?**

Welcome to this complete guide about the translation and localization system of our component library.

**OBSERVATIONS:**

- This system is agnostic to how the different components of this library are coded/implemented.
- This system is agnostic to the code/stack of your target application.
- This system tries to have the smallest API surface in our components, just a `i18n(key, params)` function with 2 args.
- This system assumes your users MUST reload the page to apply a language change.
- We don't have a way to only load the translations for a set of components (but we'd like to).

**As a user**

If you're using our component library, you'll need to:

1. Load the language file(s)
2. Register language(s)
3. Select the language selected by your user

Depending on your context, the setup can be done synchronously or asynchronously.

## Prepare everything for internationalization

