DEVOXX

WebComponents in 2023: A Complete Exploration

Horacio Gonzalez

2023-10-02

OVHcloud

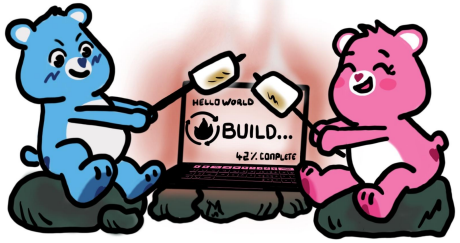
Who are we?

Introducing myself and introducing OVHcloud

Horacio Gonzalez

@LostInBrittany

Spaniard Lost in Brittany



OVHcloud

DevRel Leader

Finist Devs

GDG Finistère

Experts 2023

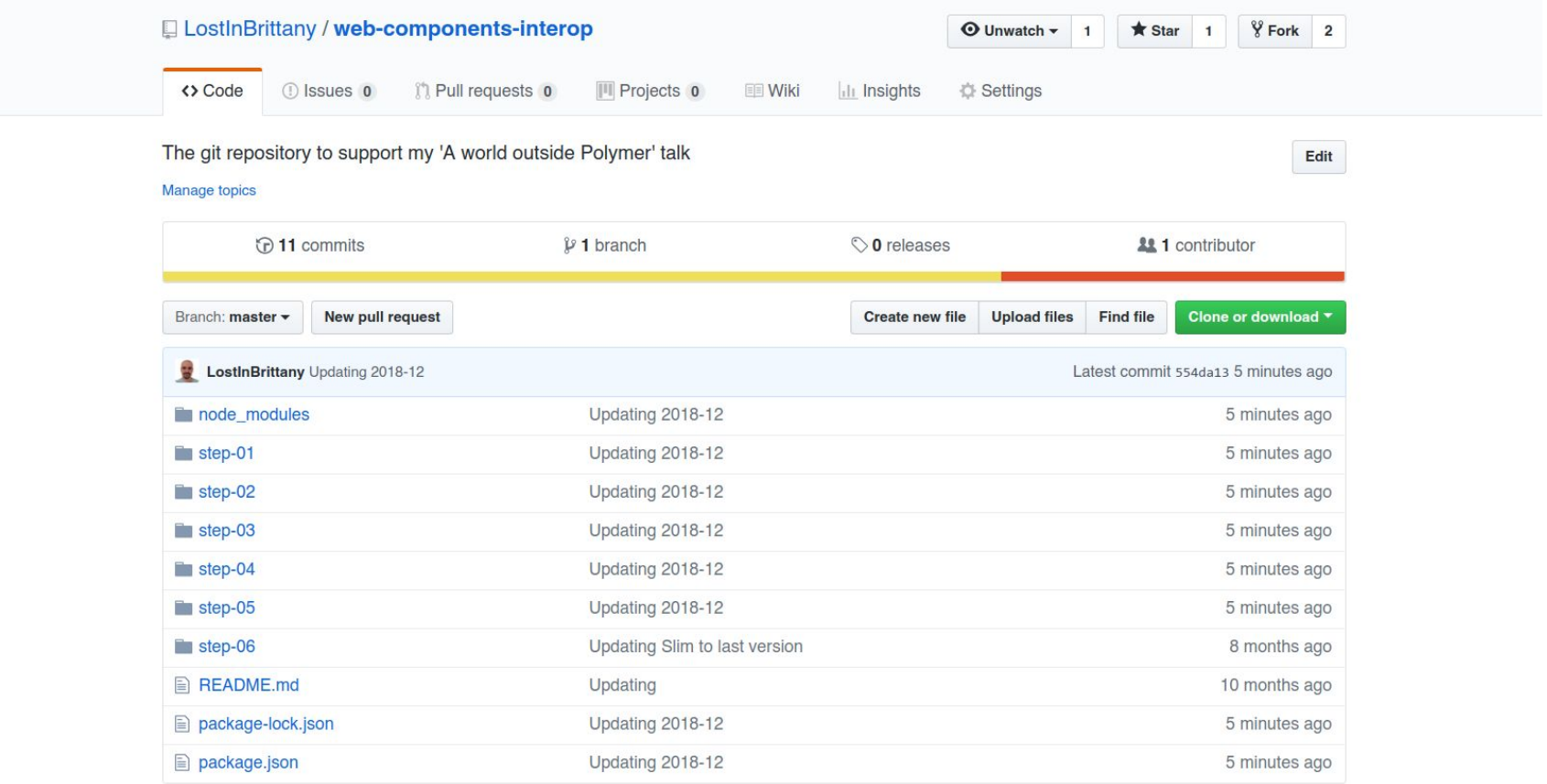
Experts 2023

OVHcloud

- **Web Cloud & Telcom**
- **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!



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>
```

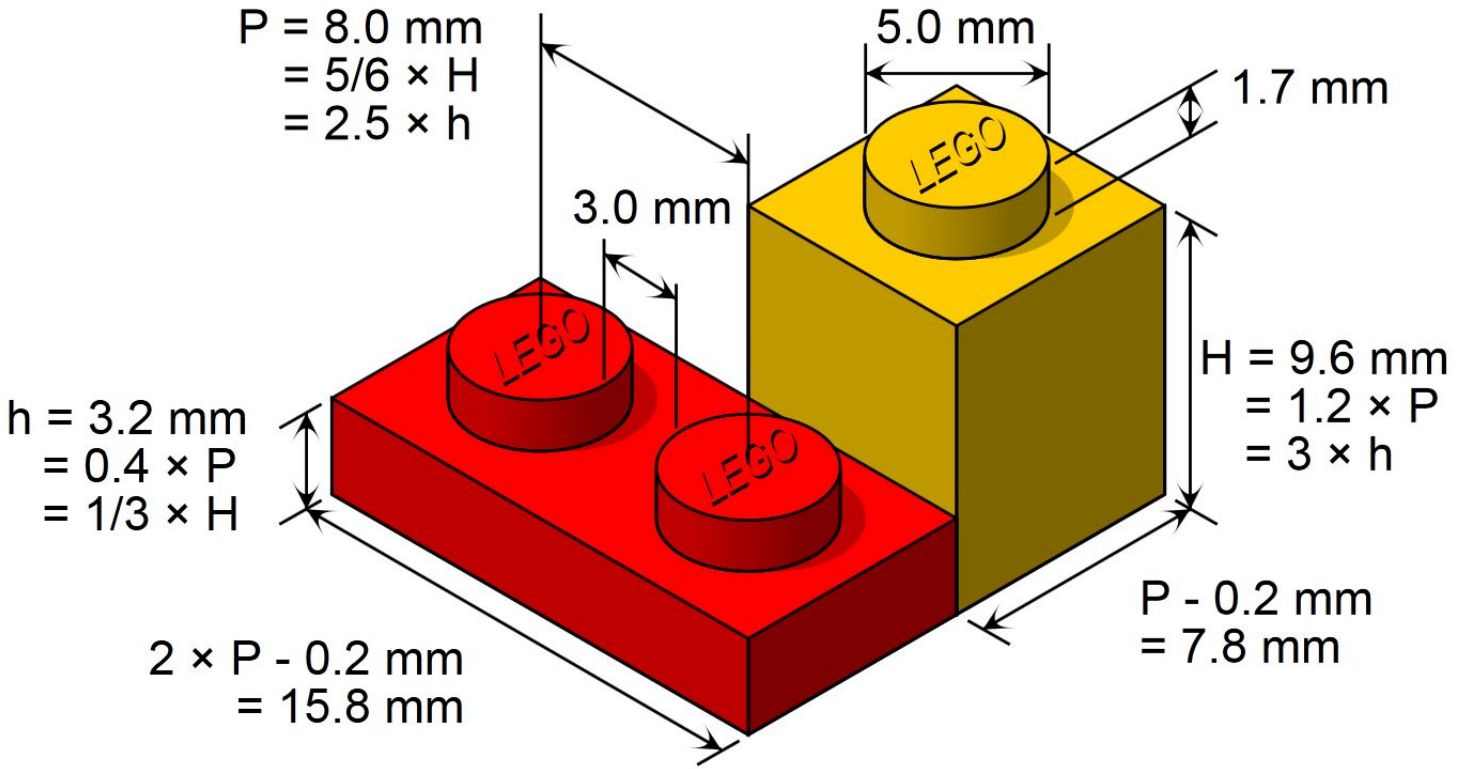
Template

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

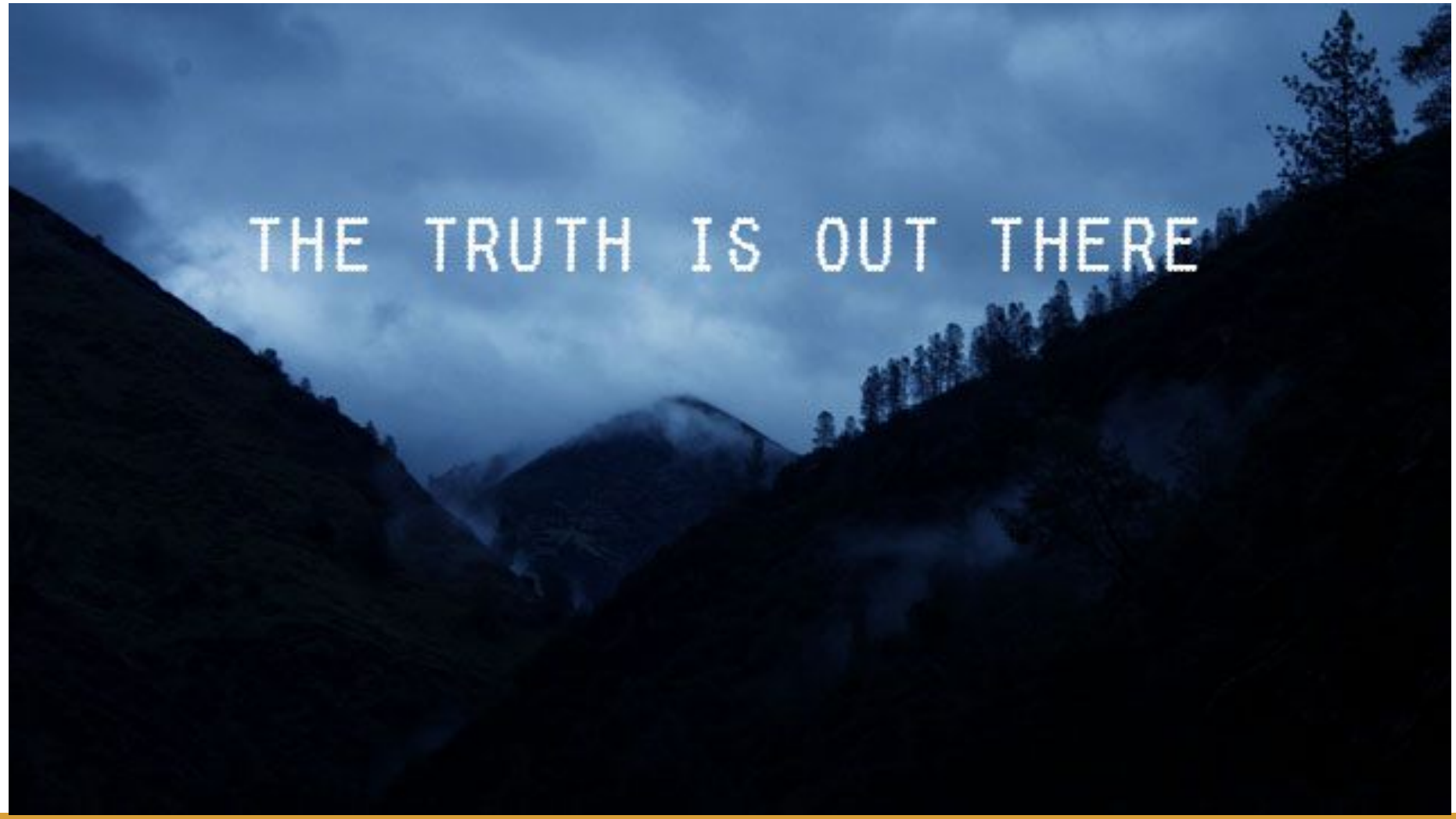
Building a world brick by brick

BitRebels Brickset

Is the promise unfulfilled?

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

Is it a conspiracy?

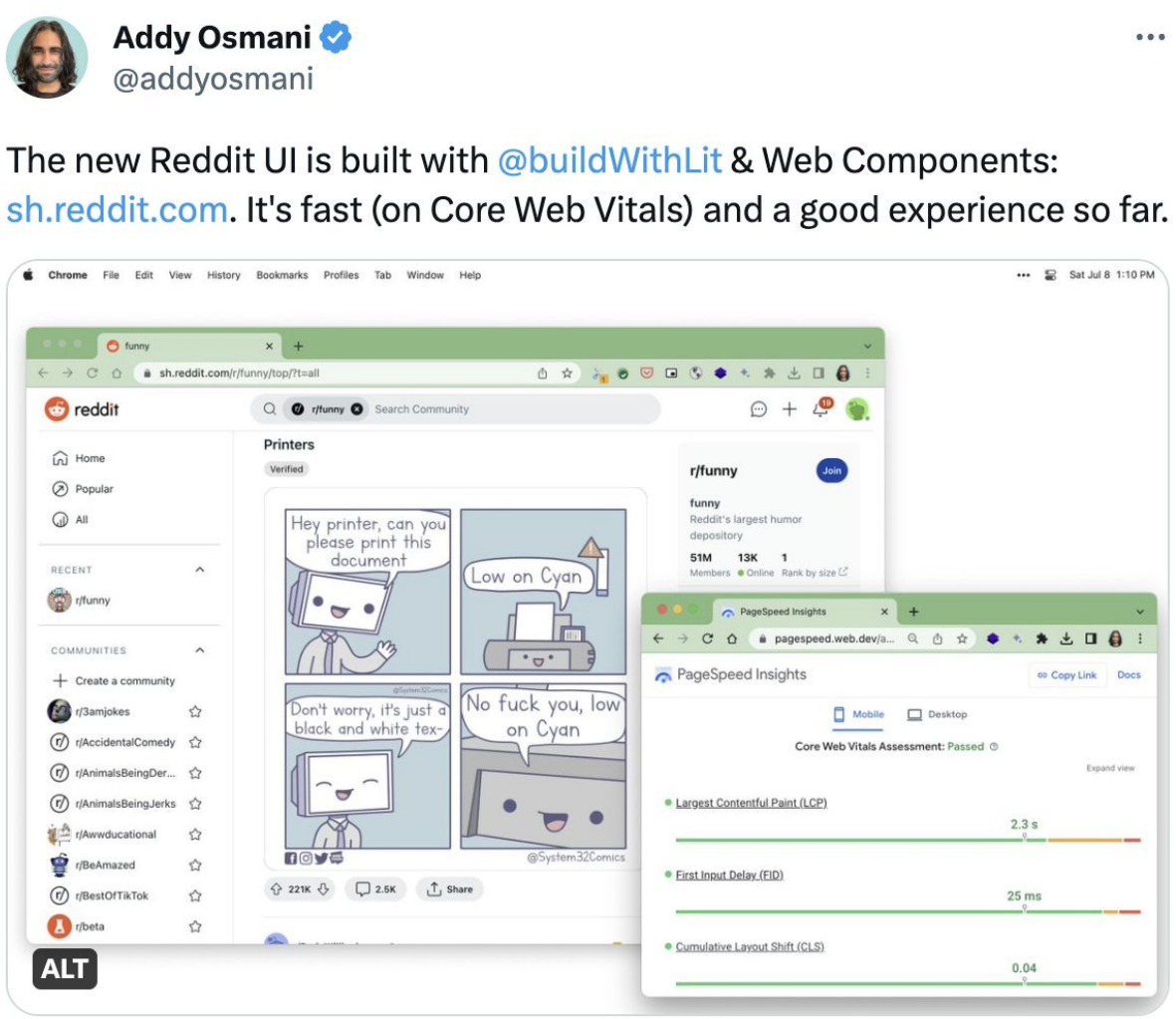


Am I only a dreamer?

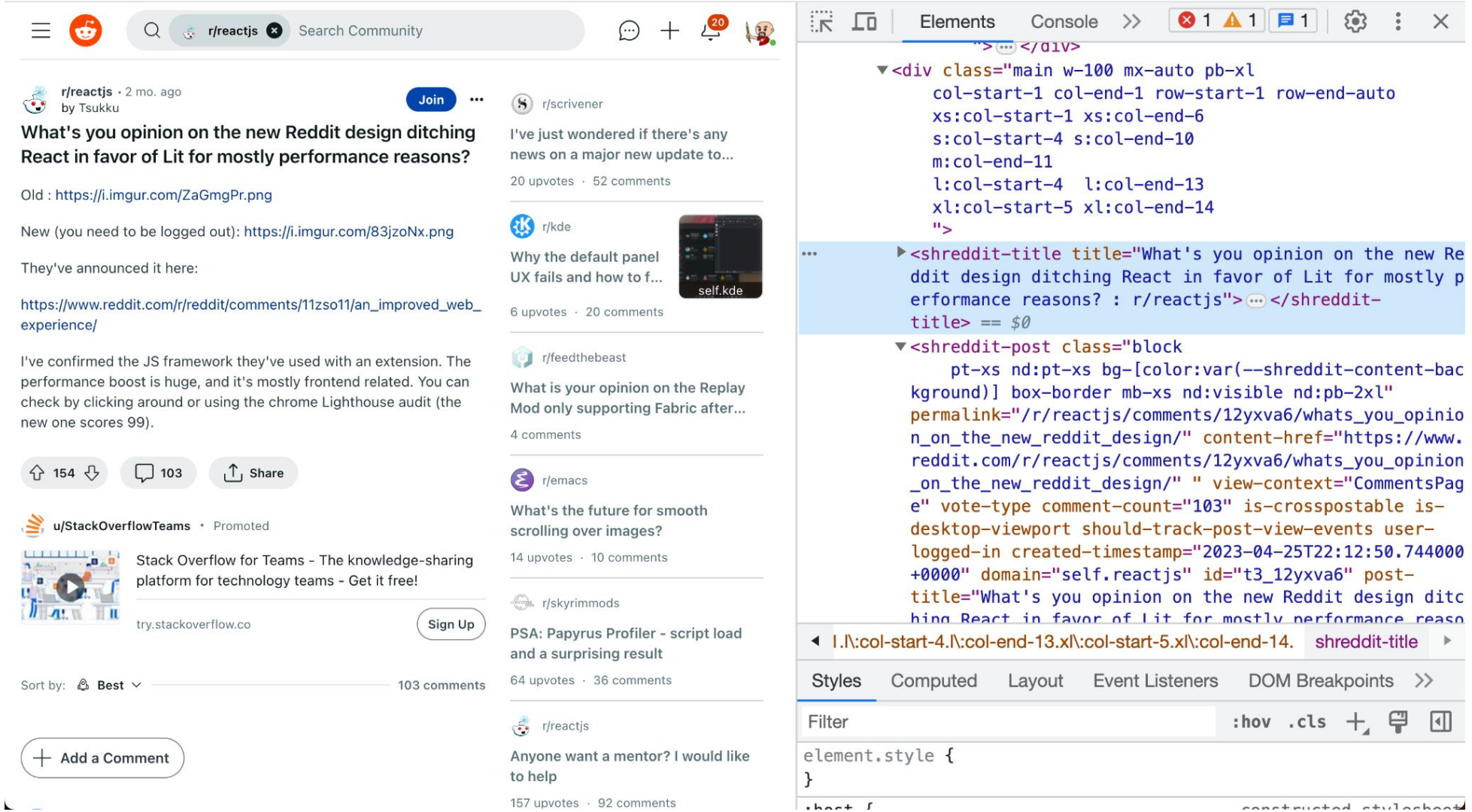
Well, revolution IS there

But it's a silent one...

I as looking for a great example



New Reddit runs on Web Components



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 = `<p>${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 = `<p>${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

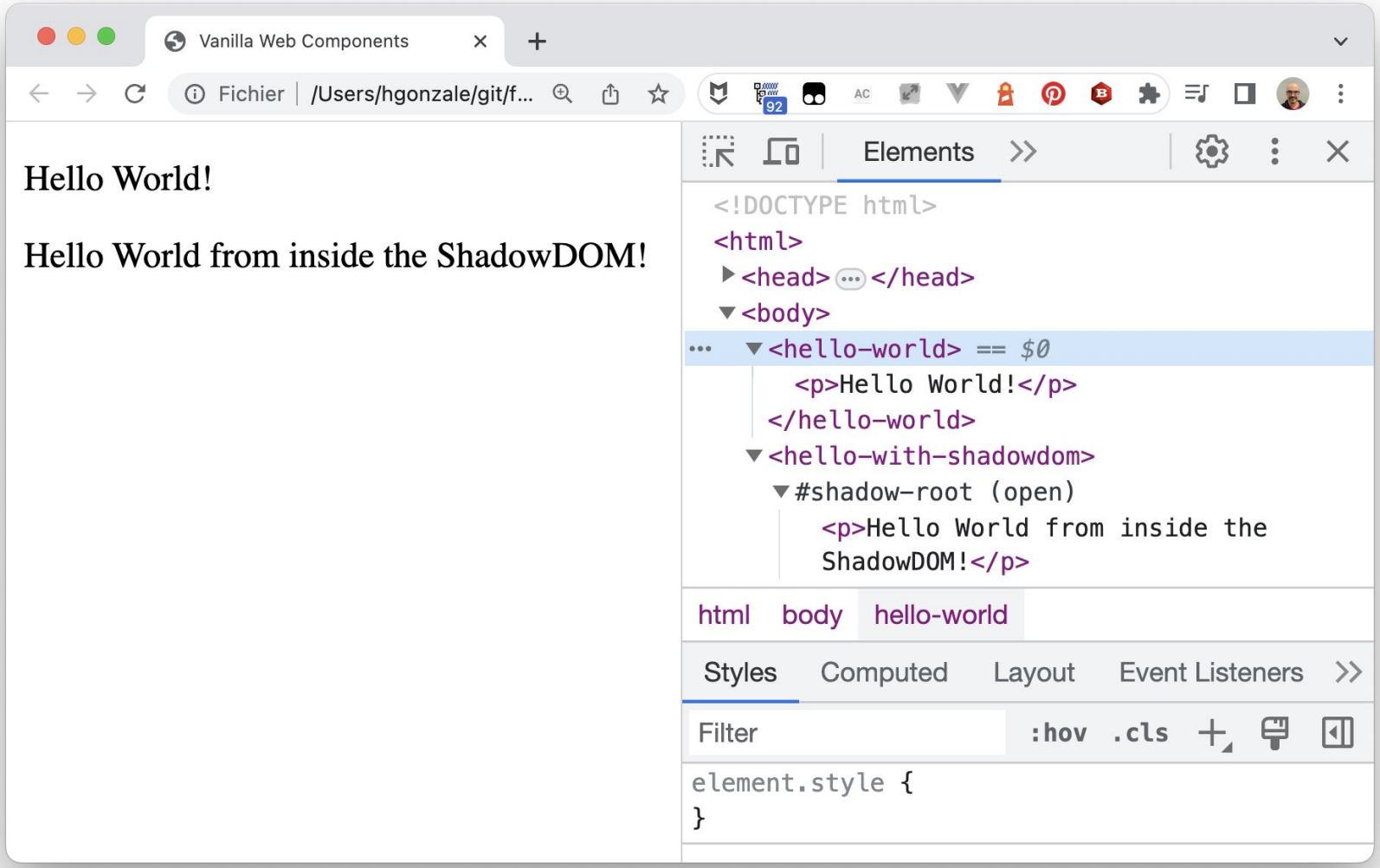
Adding ShadowDOM

```
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 = `<p>${this.msg}</p>`;
  }
}
customElements.define('hello-with-shadowdom', HelloWithShadowdom);
```

Using web components

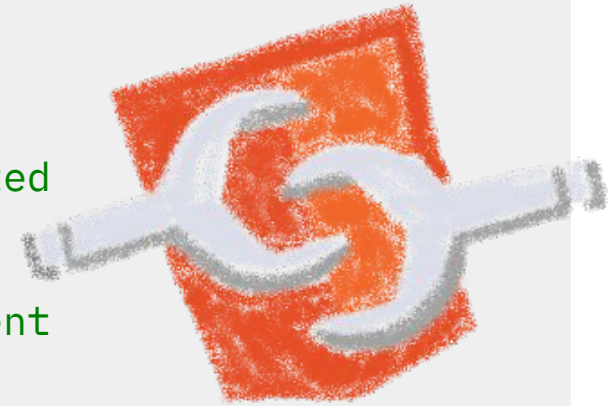
```
<!DOCTYPE html>
<html>
<head>
  <title>Vanilla Web Components</title>
  <script src="./hello-world.js"></script>
  <script src="./hello-with-shadowdom.js"></script>
</head>
<body>
  <hello-world></hello-world>
  <hello-with-shadowdom></hello-with-shadowdom>
</body>
</html>
```

Using web components



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 observer 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">
       <img src="${import.meta.url}/../img/logo.png">
   </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

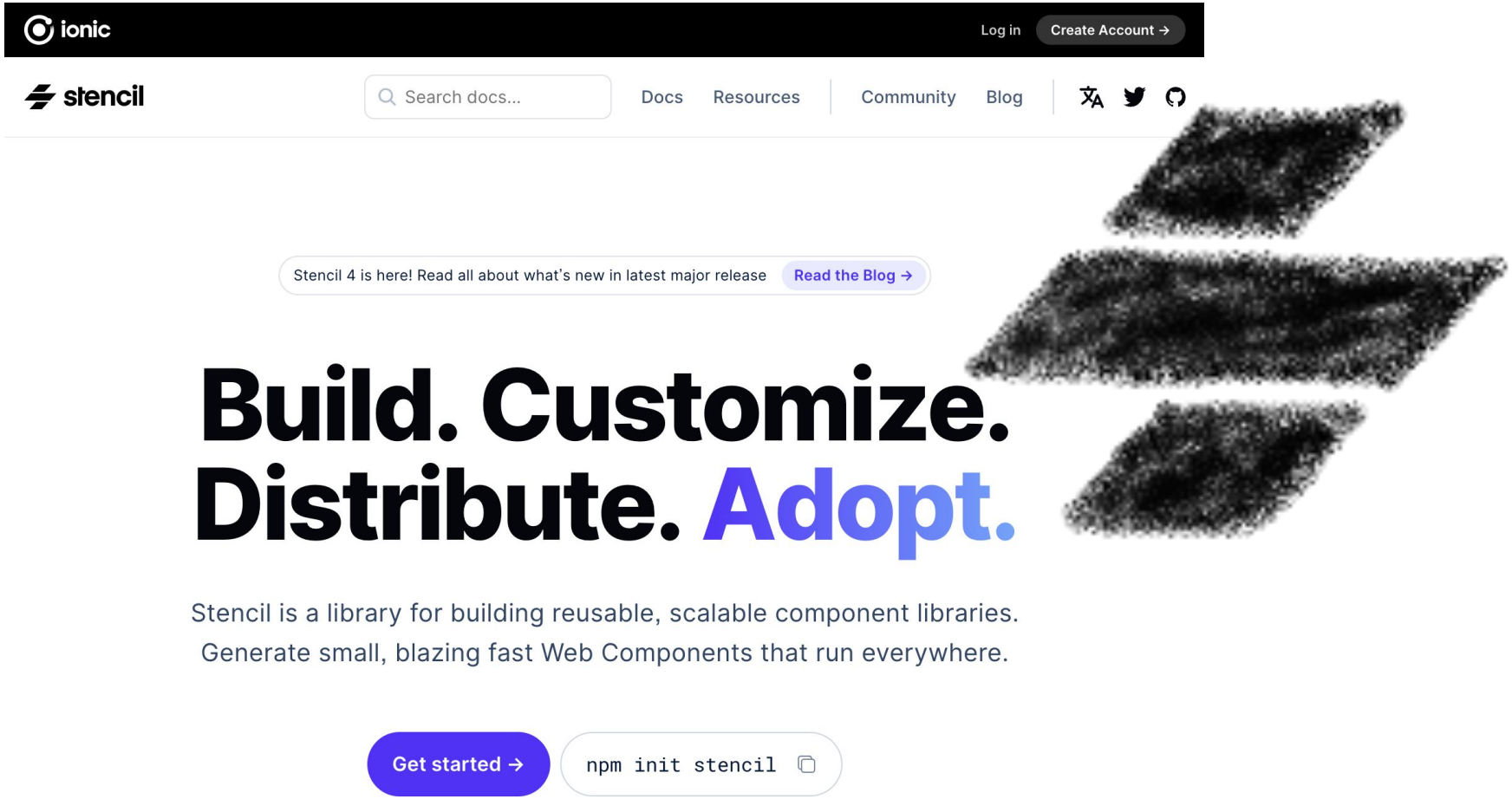
High-performant, low-level, in-the-platform web components standard

Libraries aren't a failure of standard

They happen by design

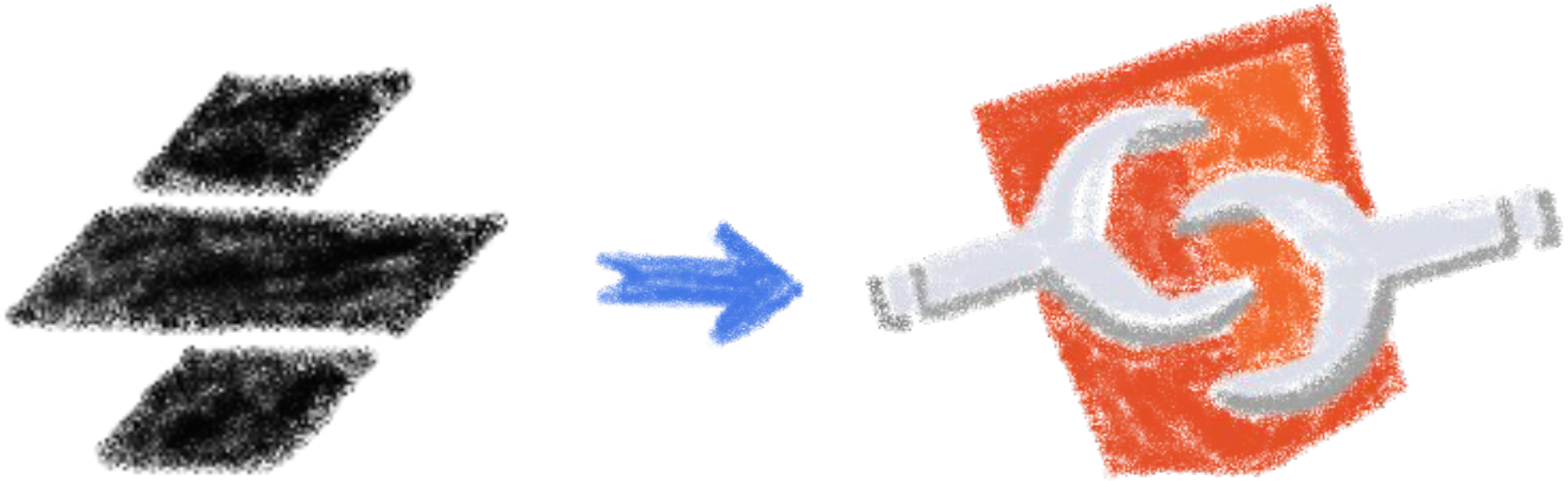
A library for building reusable, scalable component libraries

Not another library



A Web Component toolchain

A build time tool

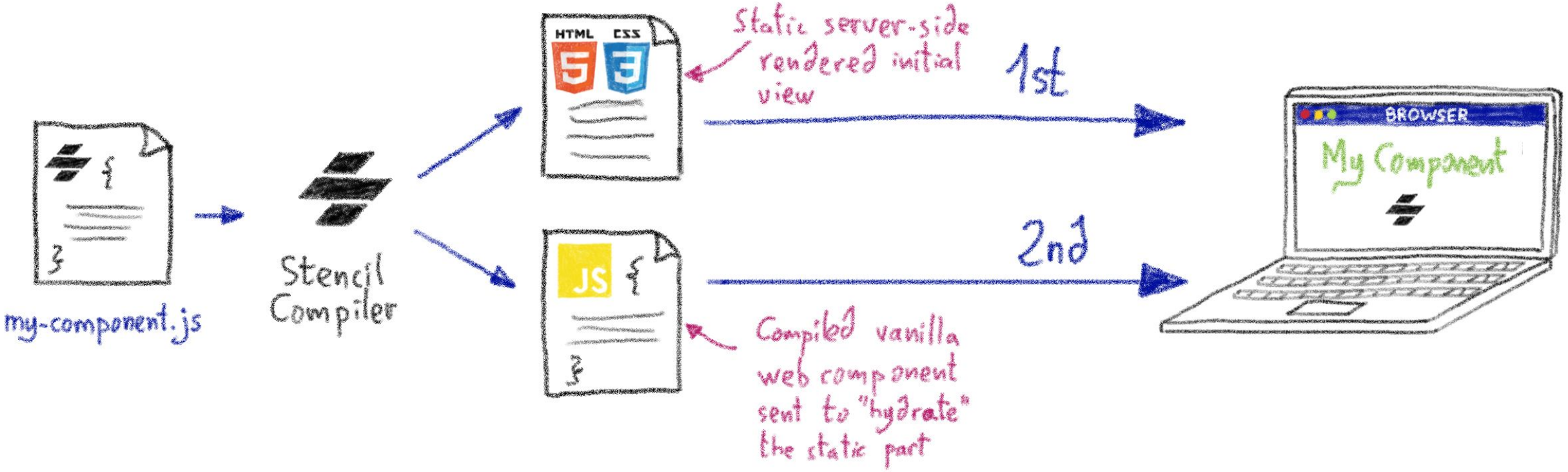


To generate standard web components

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

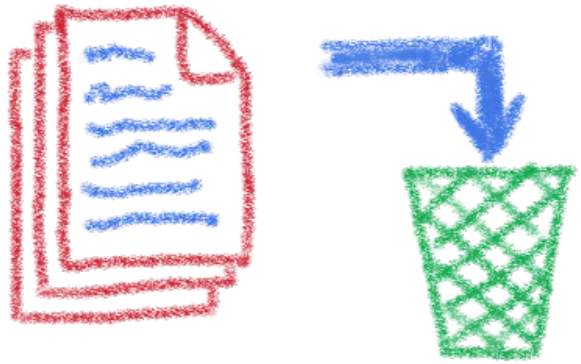
A tiny runtime, pre-rendering, and the raw power of native Web Components make Stencil one of the fastest compilers around.

Future proof

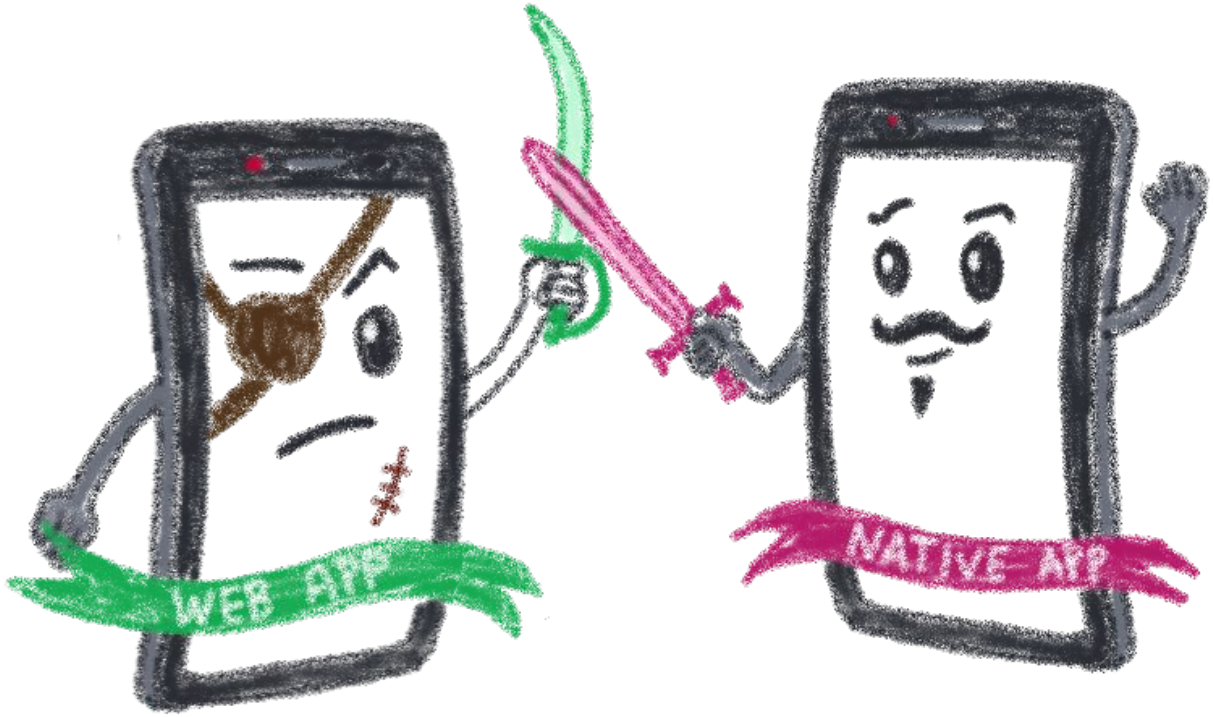
Build cross-framework components and design systems on open web standards, and break free of Framework Churn.

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

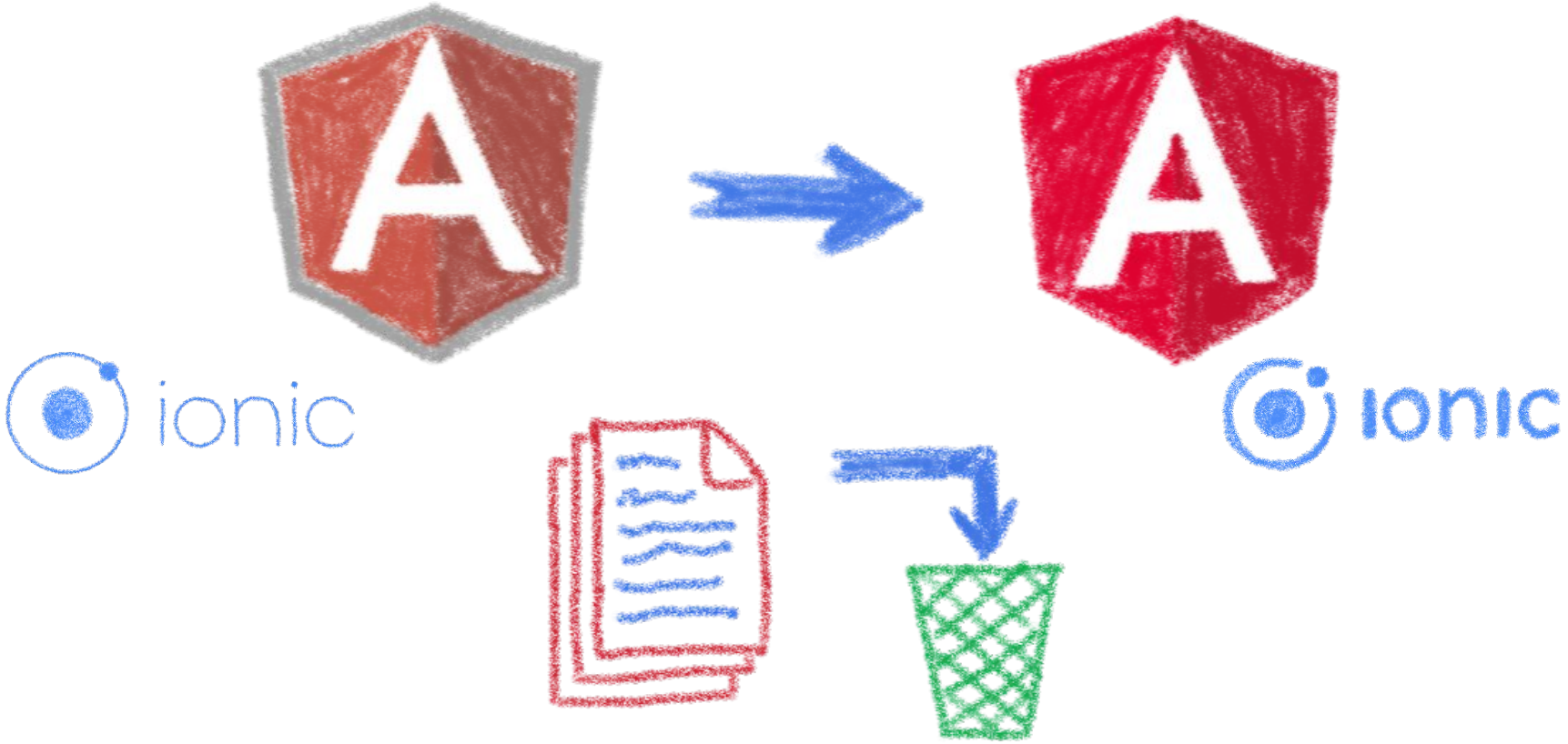
A company wanted to do it well

The perfect technology for mobile web and hybrid apps

The time is 2013

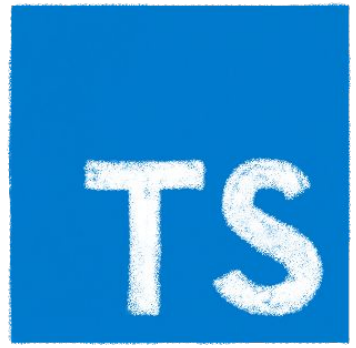
So what technology would you use?

Really soon after launch...



Hey folks, we are killing AngularJS!

What did Ionic people do?

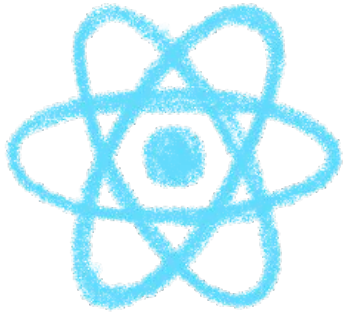


Let's put everything in the trash bin and begin anew

But times have changed...

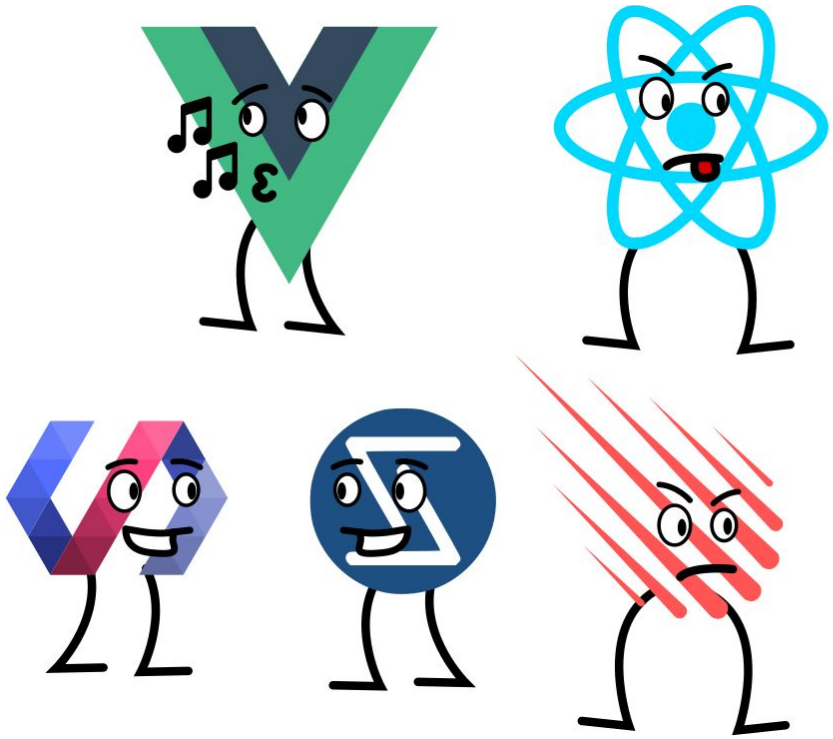
In 2013 Angular JS was the prom queen

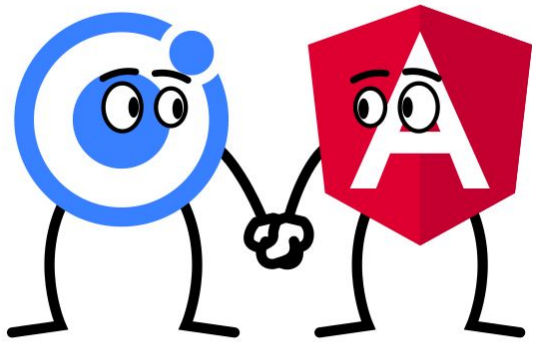
Times have changed...



In 2017 Angular is only one more in the clique

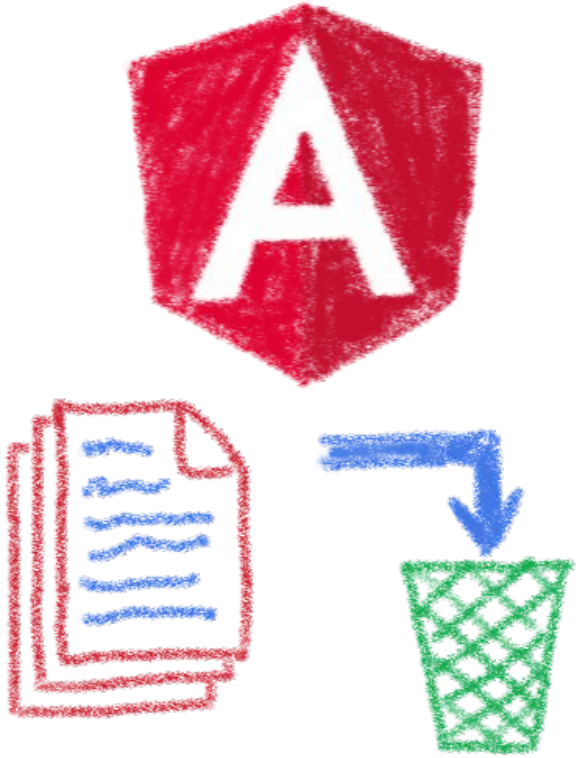
Angular limits adoption of Ionic





Devs and companies are
very vocal about JS Frameworks

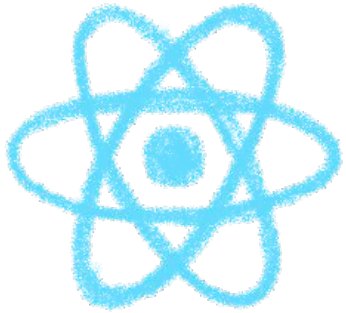
What did Ionic people do?



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

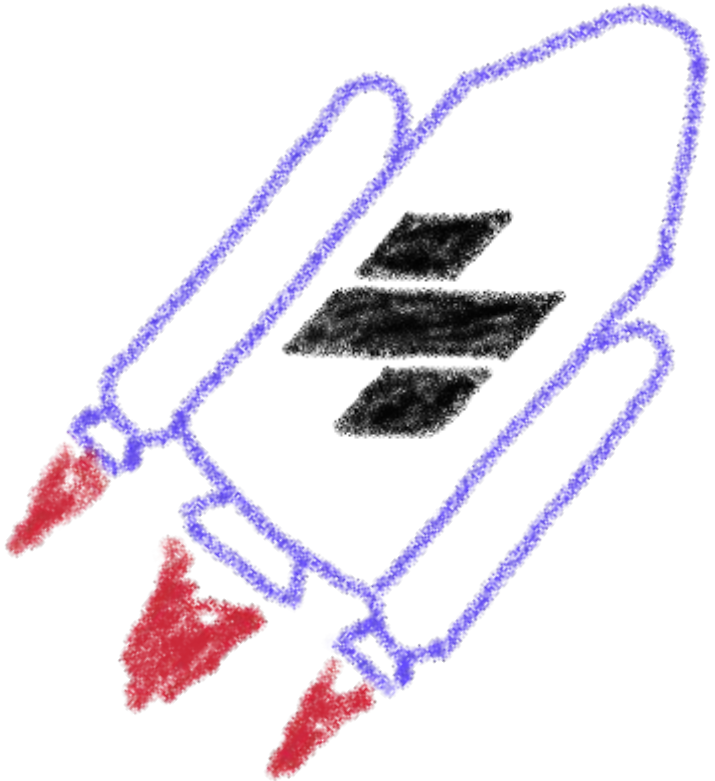
smart
HTML ELEMENTS

SkateJS

slim.js

There were so many of them!

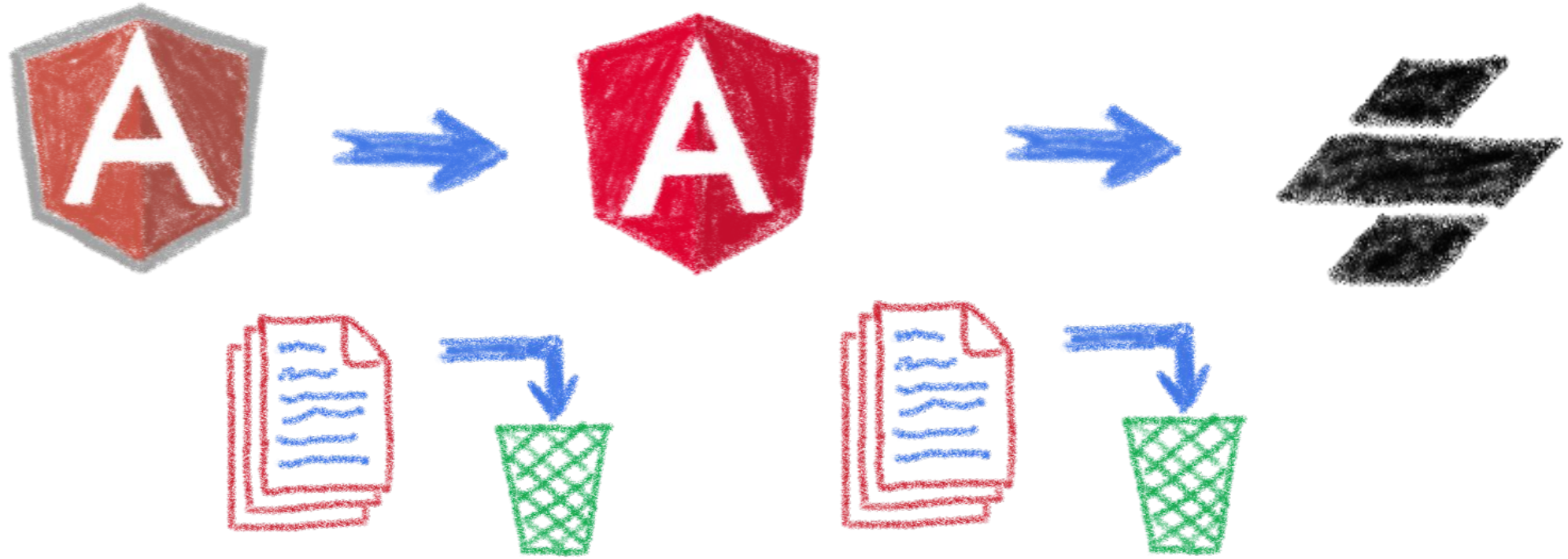
Let's do something different



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

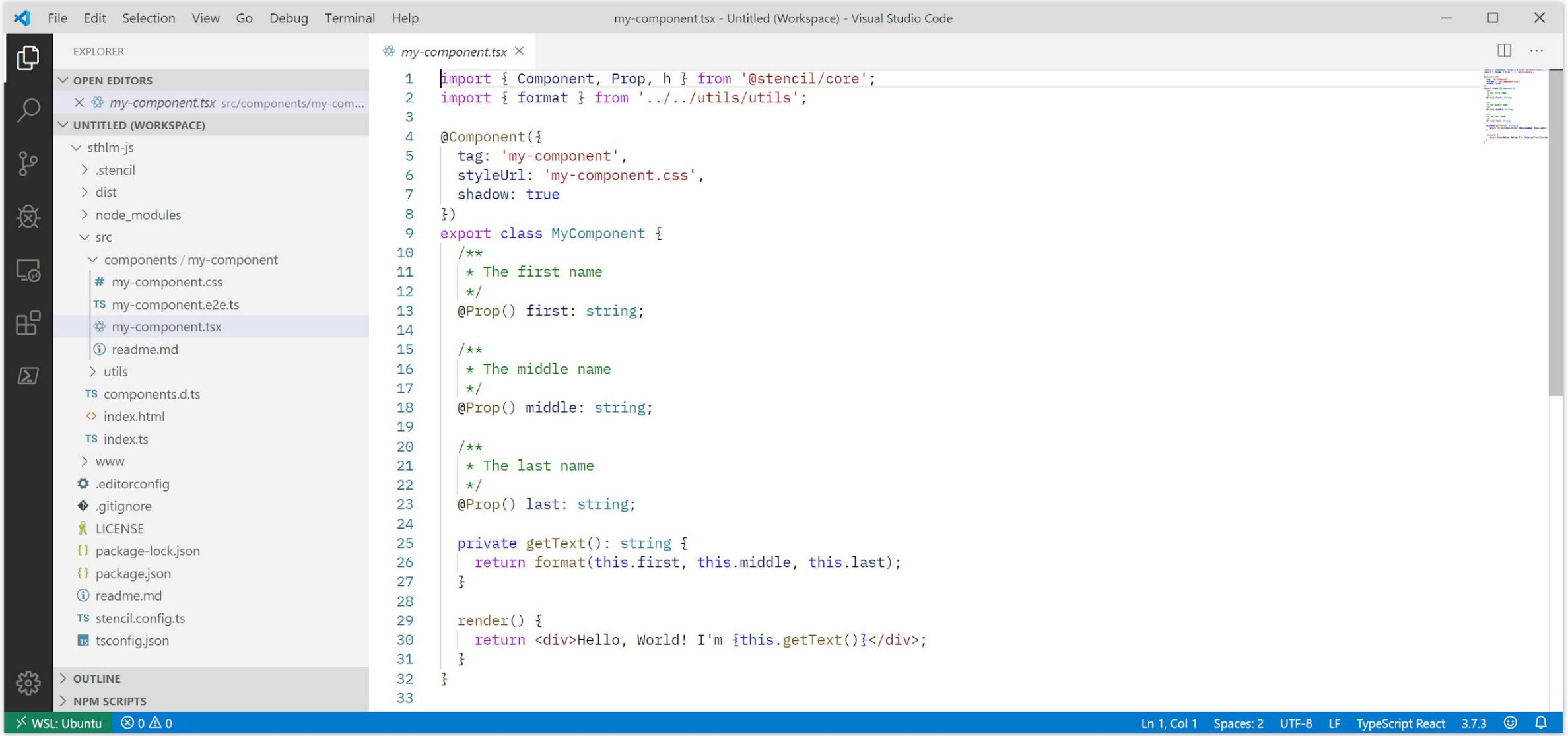
```
? Select a starter project.

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



Some concepts

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

@Component({
  tag: 'my-component',
  styleUrl: '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',
  styleUrl: '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

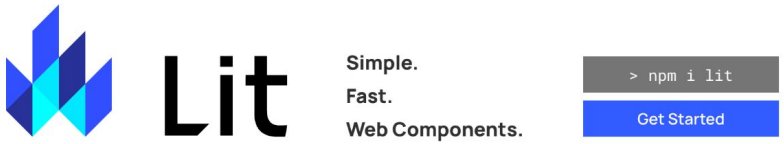
README.md

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

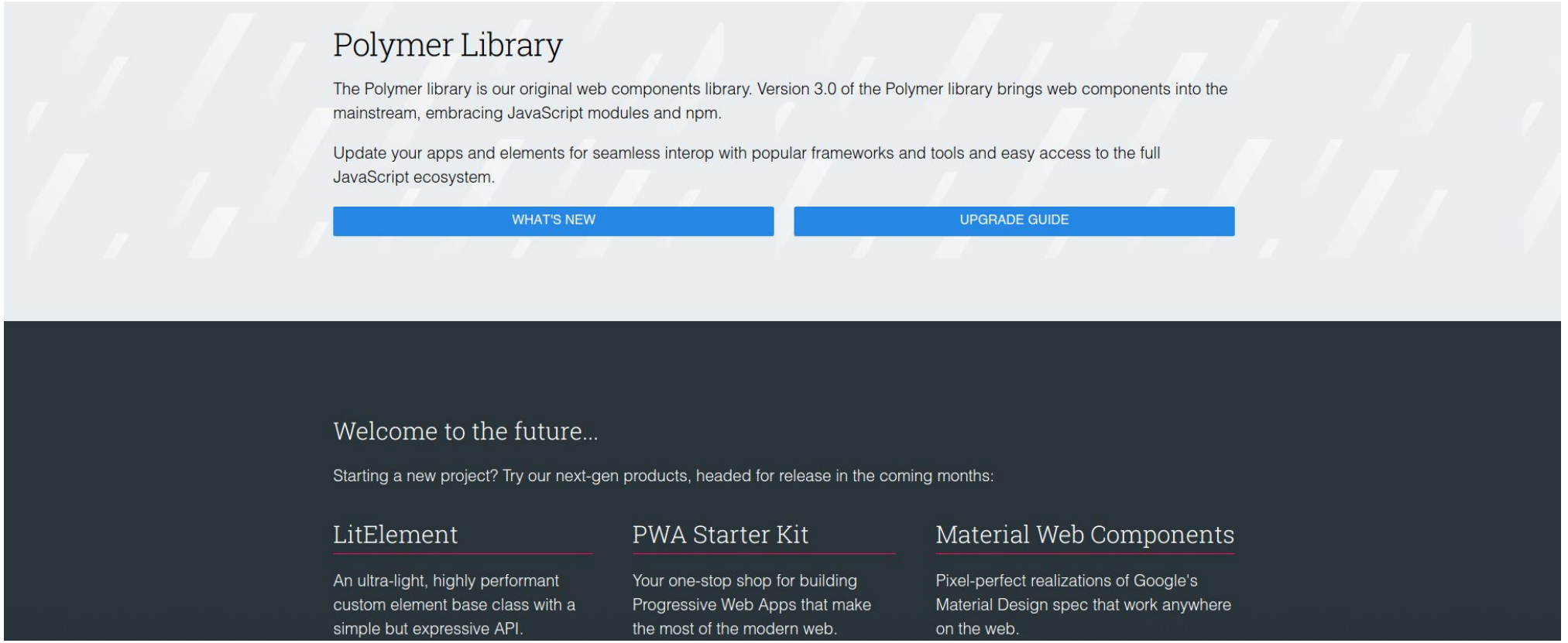
Interoperable & future-ready

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...



And that means good news!

Let's try to see clearer

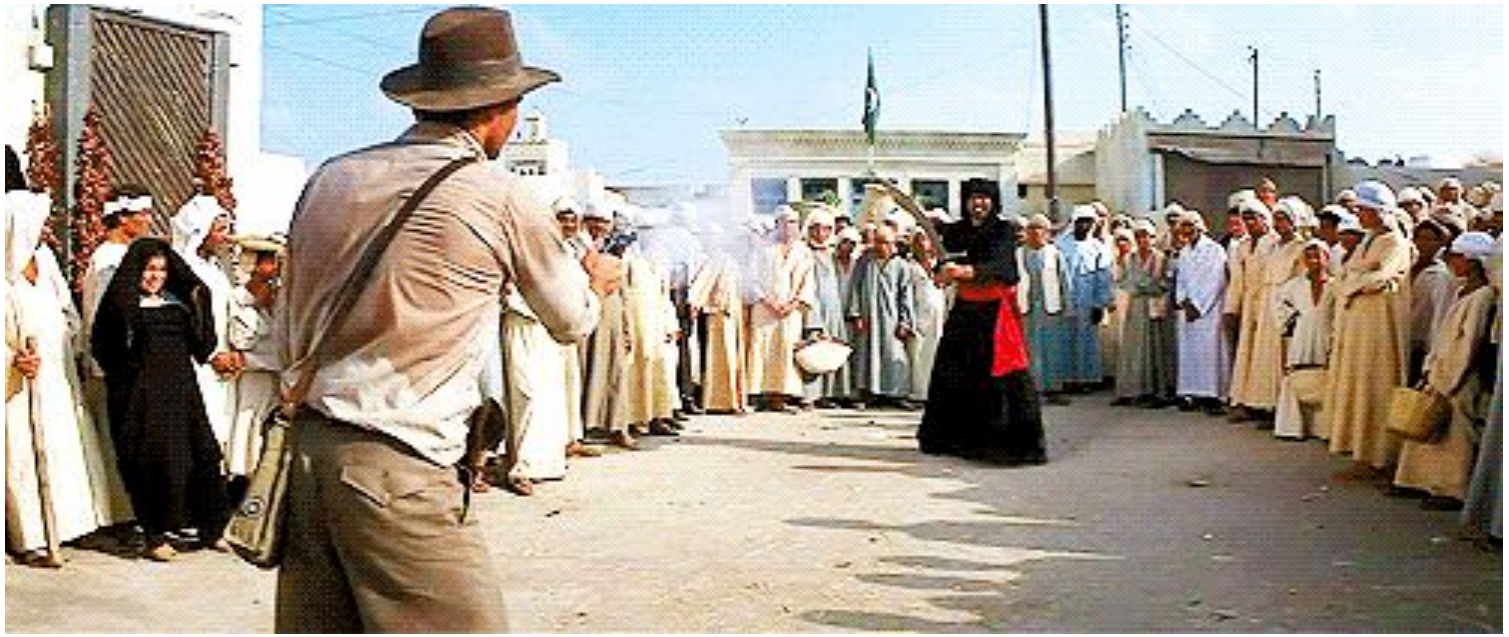
Let's dive into Polymer history...

A tool built for another paradigm

No web component support on browsers

No React, Angular or Vue innovations

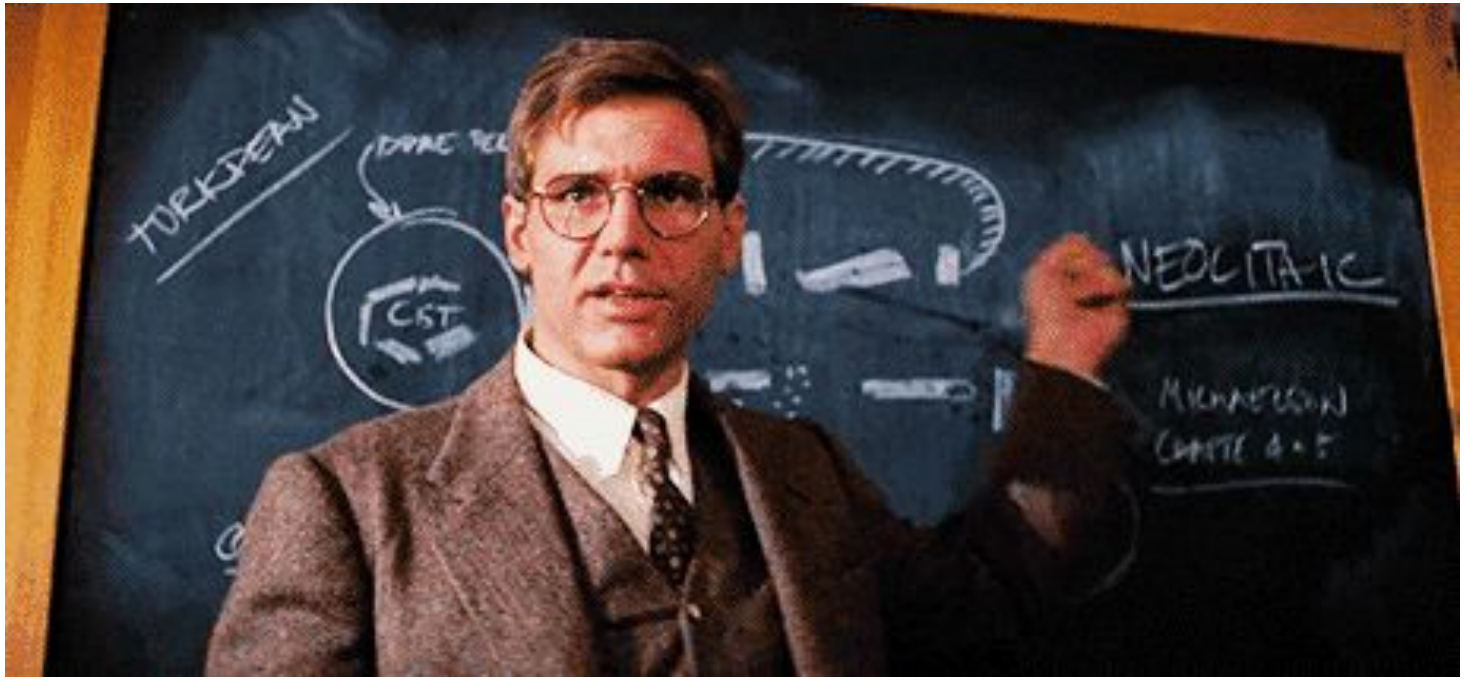
No so well suited for the current one



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?

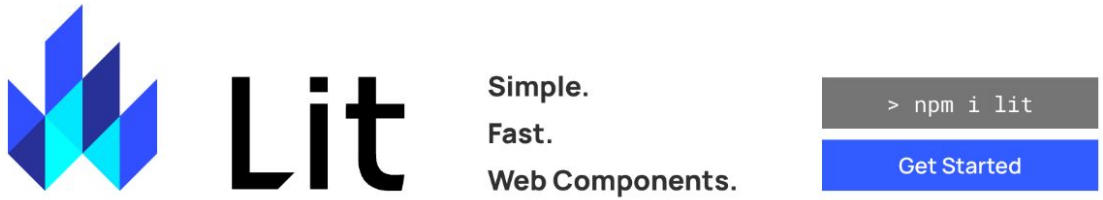
That's why web components are top

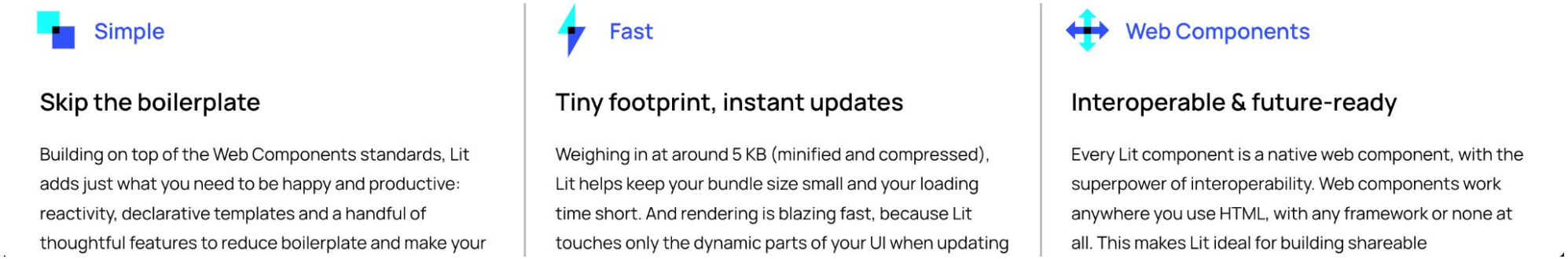
You can keep using all your Polymer components and create the new ones with a new library… And it simply works!

Born from the Polymer team

For the new web paradigm

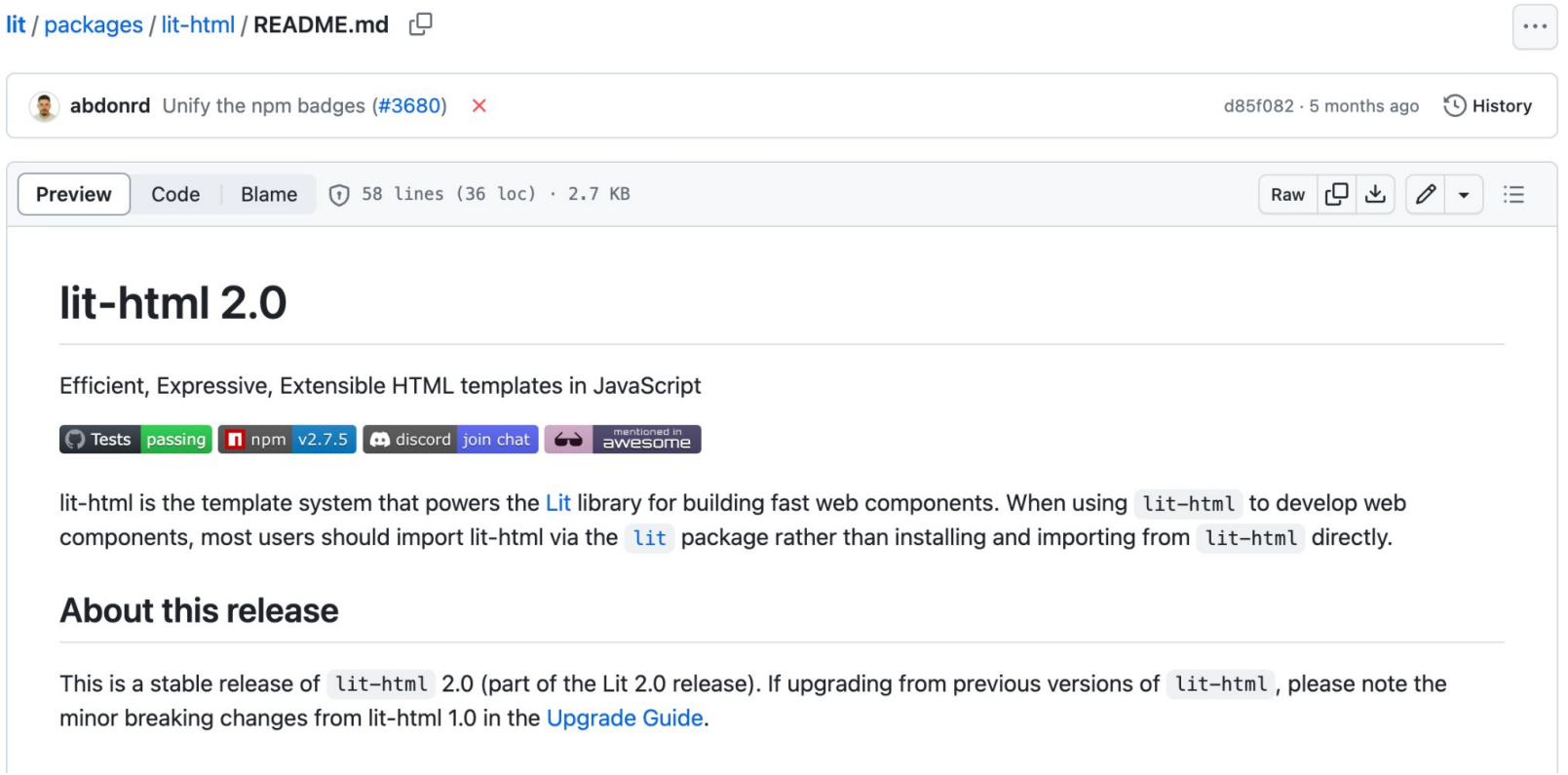
Modern lightweight web components





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

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

Little known functionality of template literals

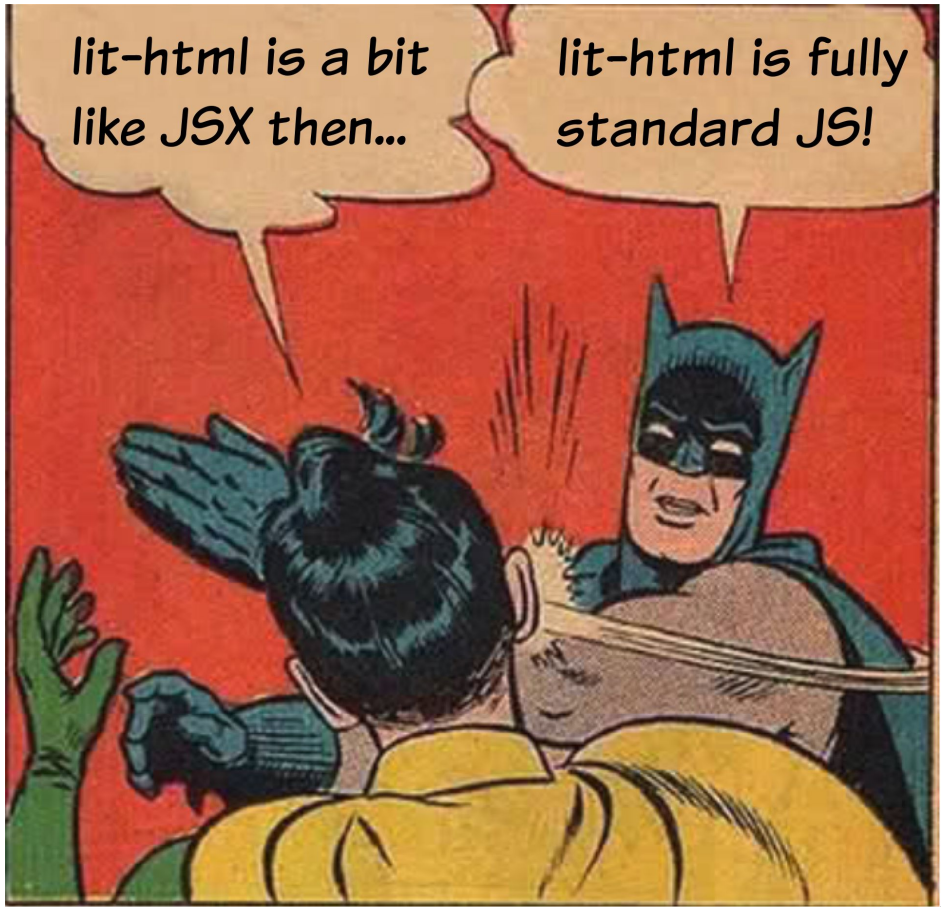
lit-html Templates

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

Lazily rendered

Generates a TemplateResult

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



The good sides of JSX… but in the standard!

LitElement

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

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

Lightweight web-components using lit-html

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` :

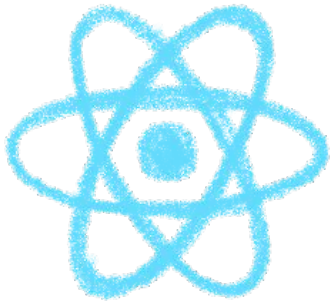
```
npm init @open-wc
```

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.

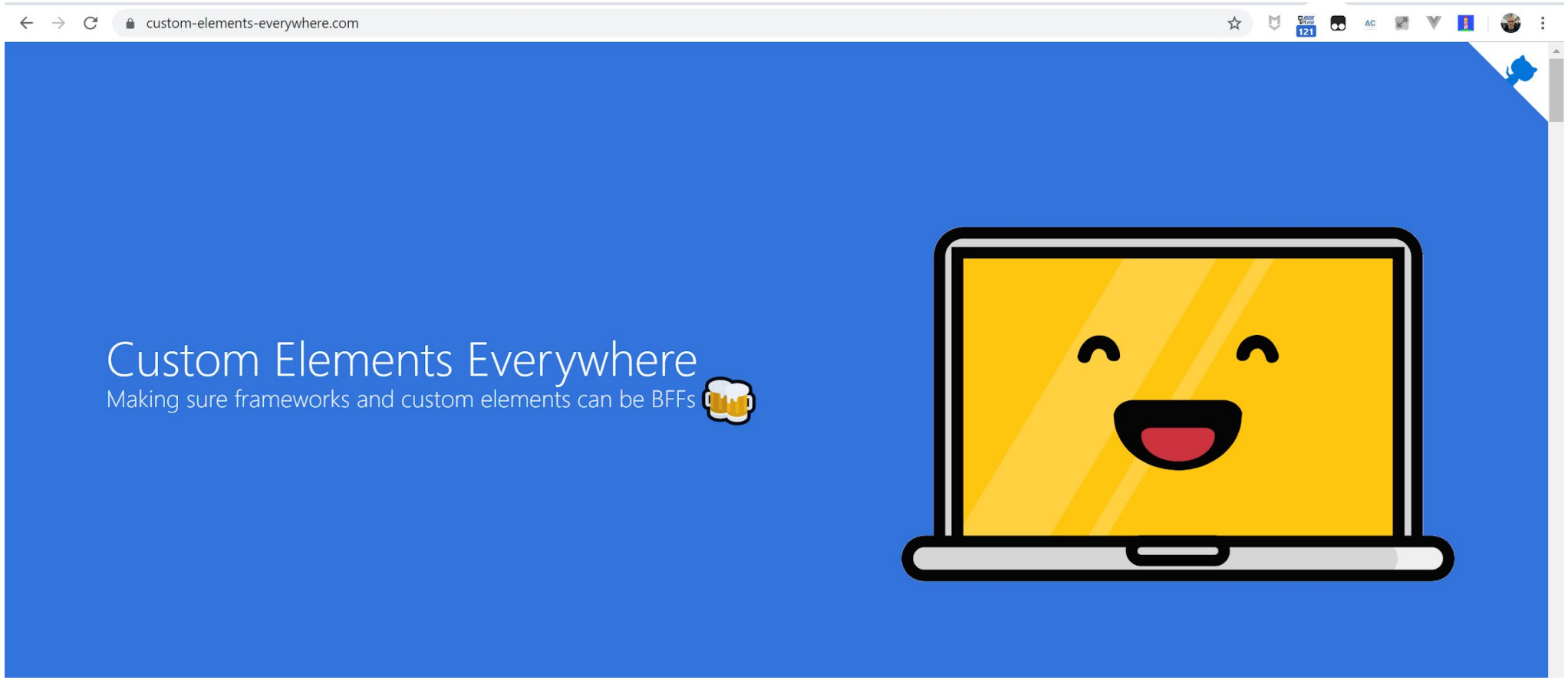
```
$ npm init @open-wc

        _.,,,,,,,,,,._
```

Web Components & Frameworks

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



Compatibility is on Web Components side

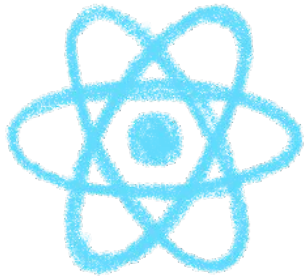


Web Components everywhere, baby!

They are the interoperable alternative

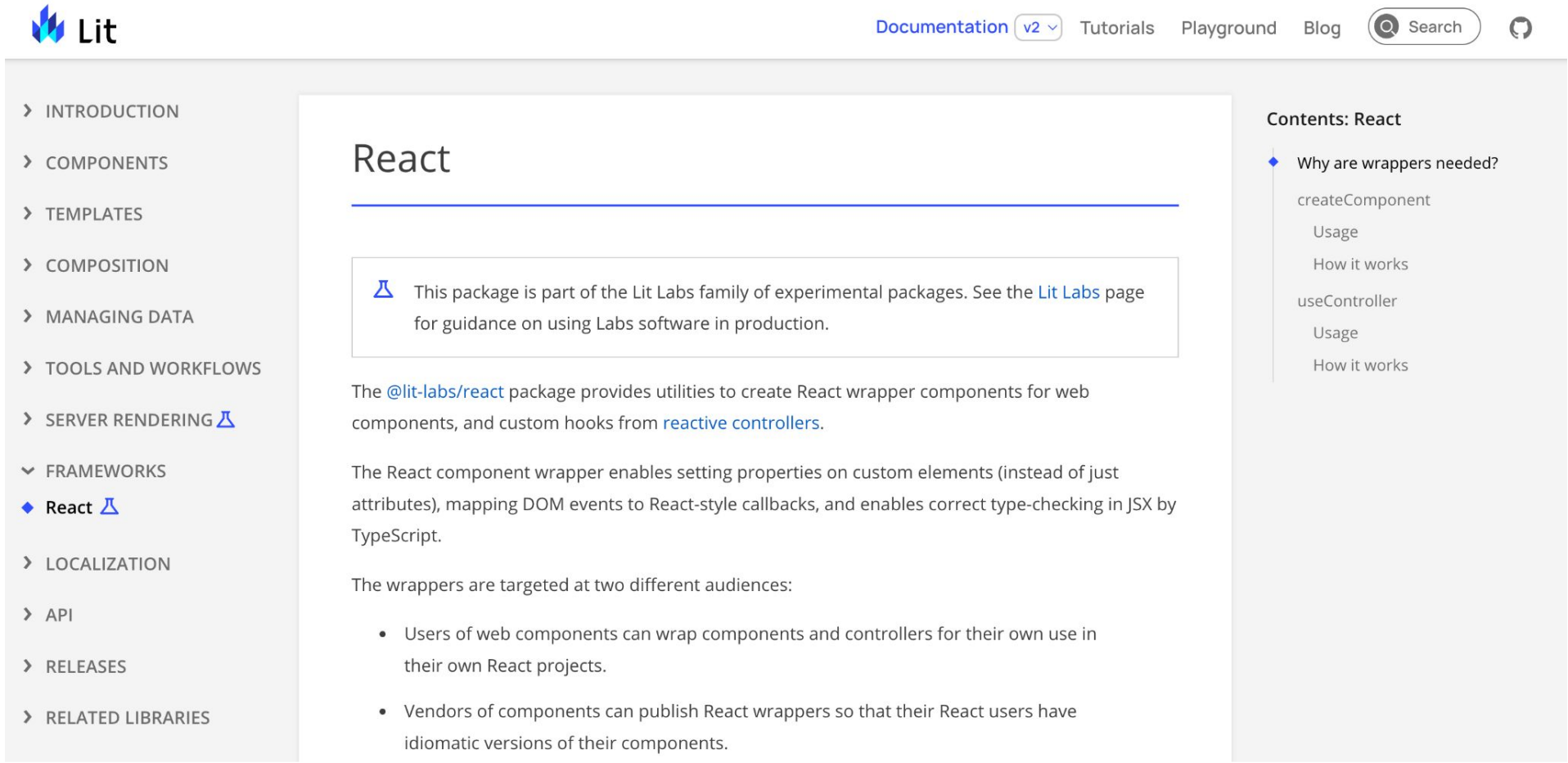
Any framework… or no framework at all

You can have a single implementation



And it simply works everywhere*

*React don't fully support them yet

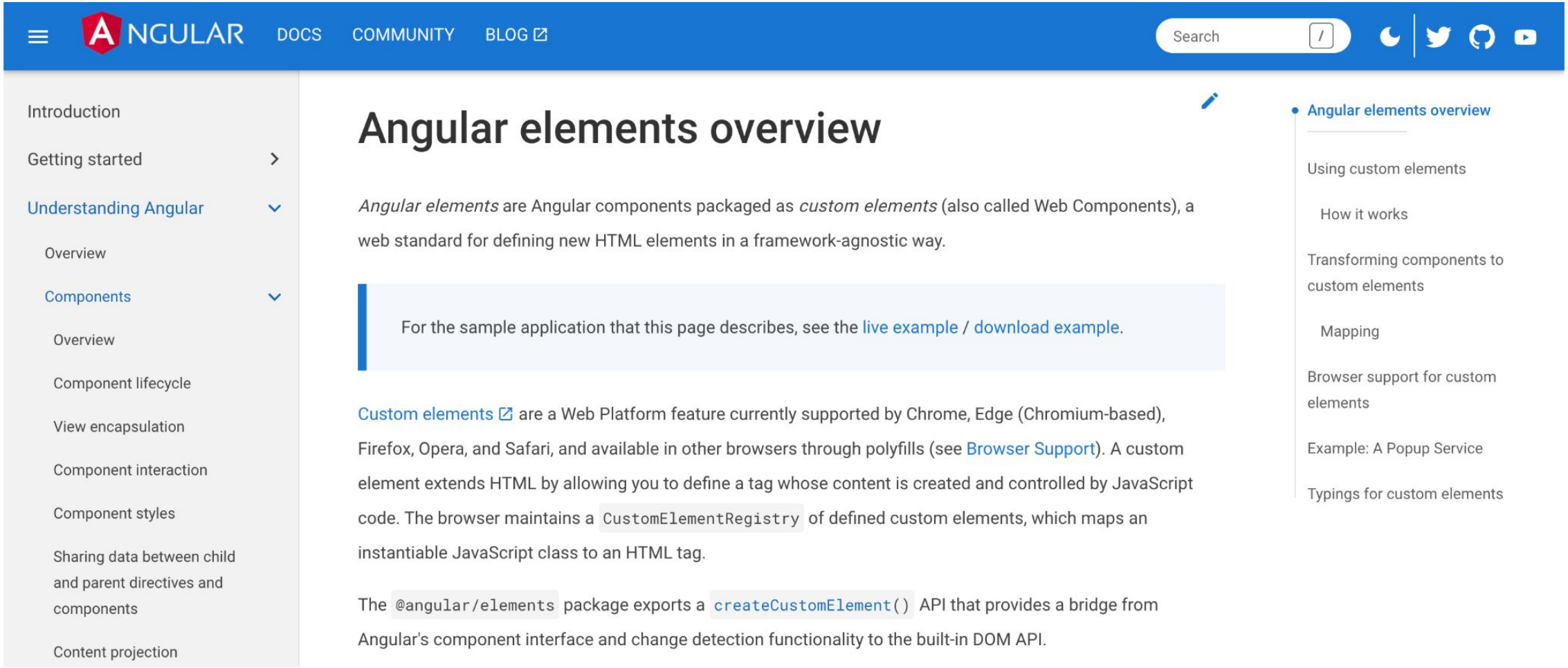


Long story made short: use `lit-labs/react`

When you need interoperability

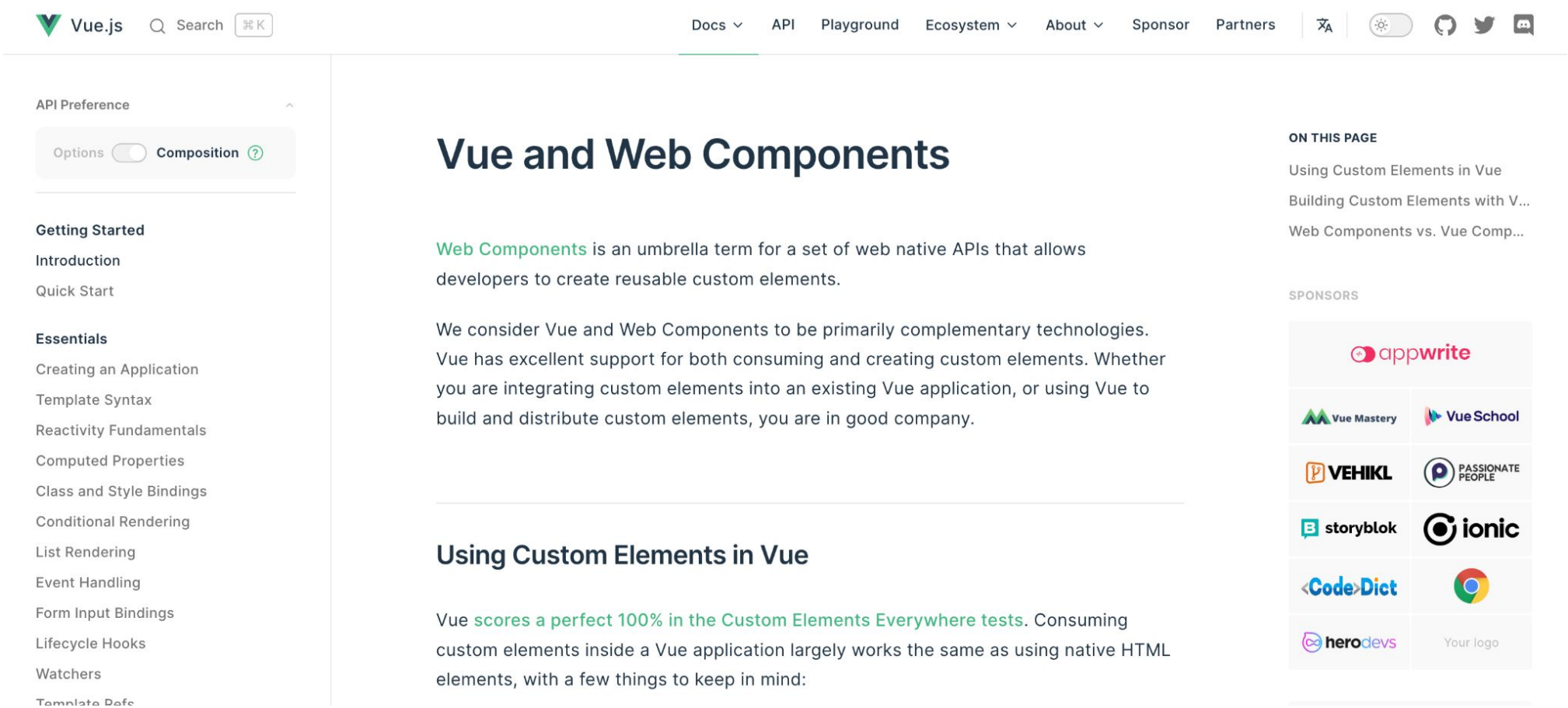
Nothing beats the standard

Angular can generate web components



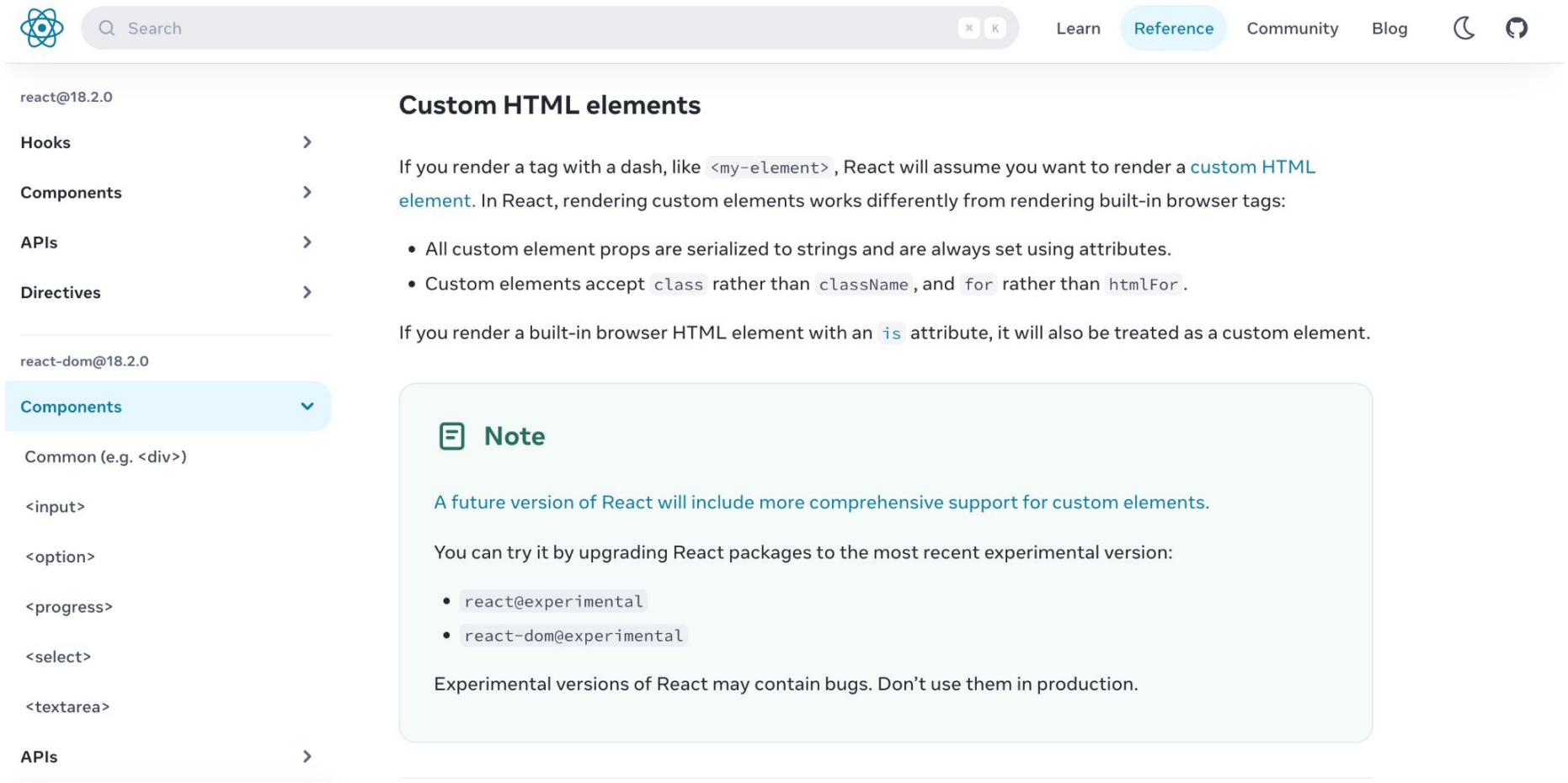
Angular Elements

Vue can generate web components



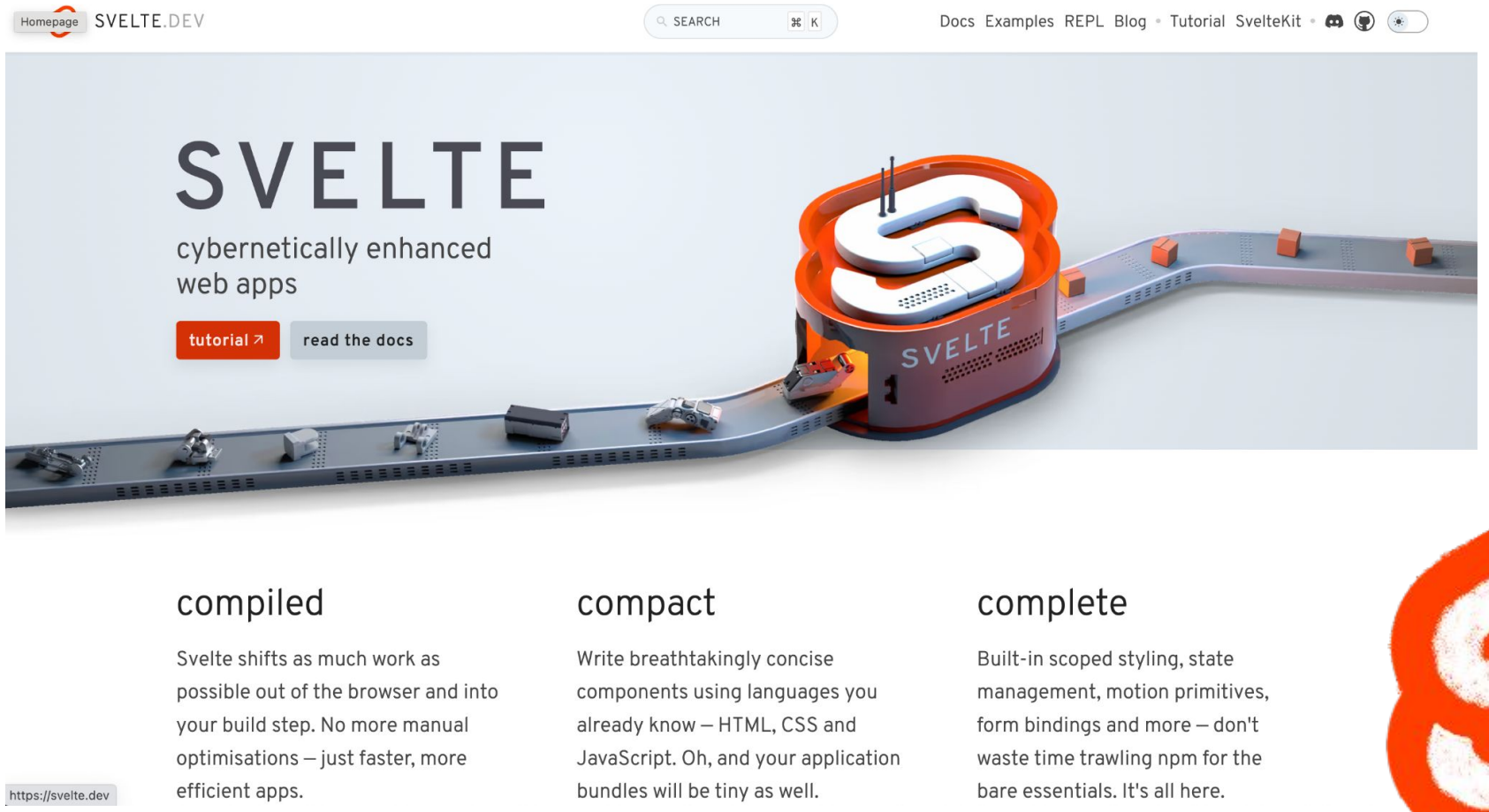
With `defineCustomElement()`

React can generate web components



But it can generate them too

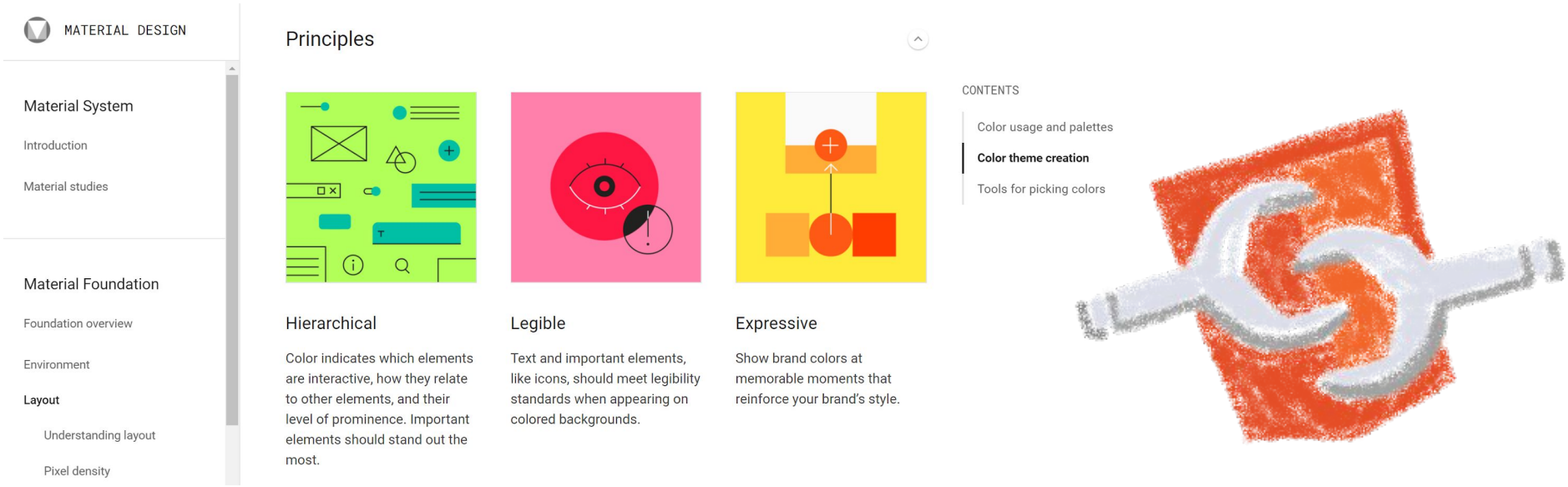
What about Svelte?



Let's look in detail one case

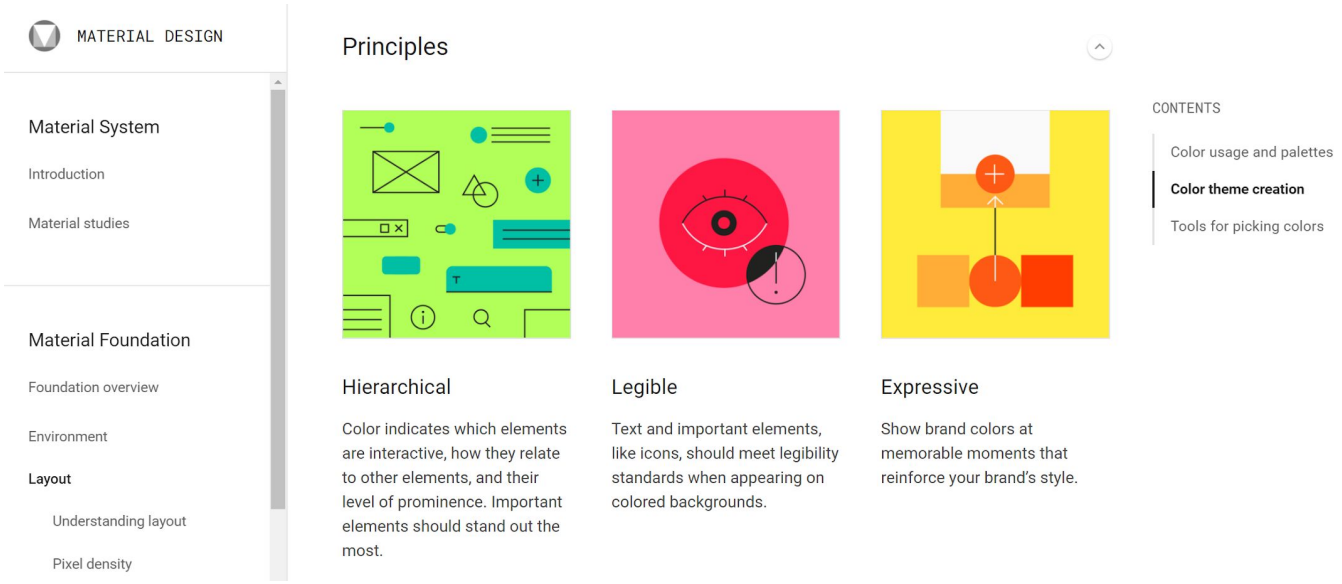
Web Components & Design Systems

One of the best cases for Web Components

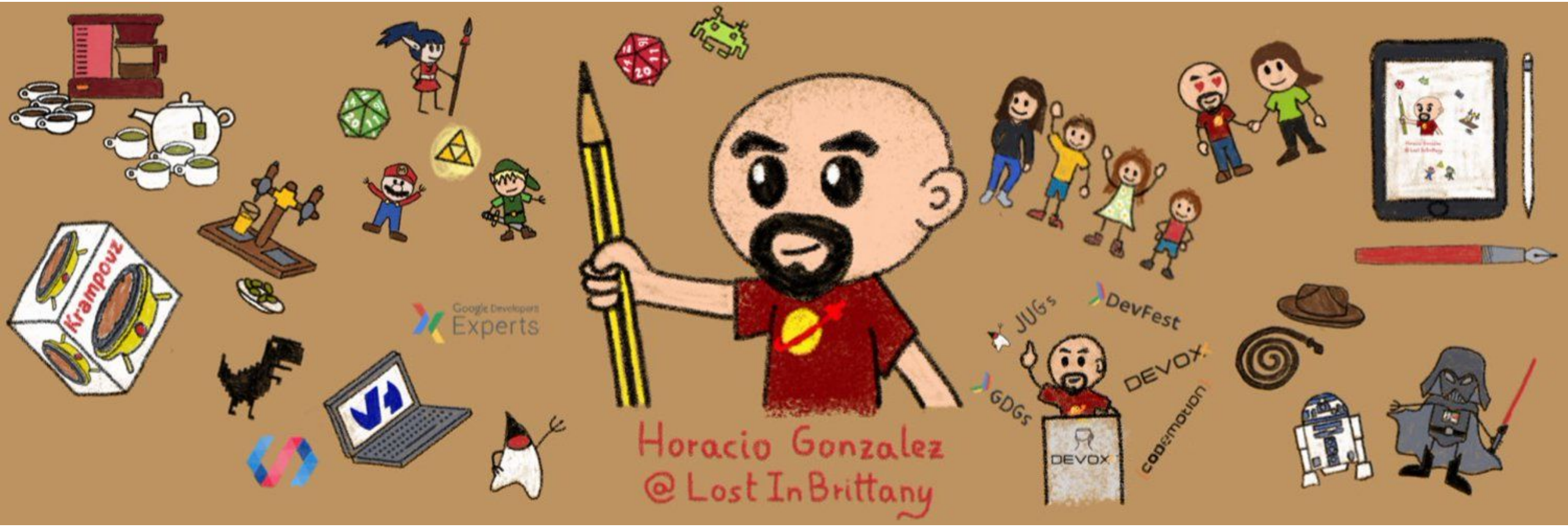


So, what are Design Systems?

And why should I look at them?



A talk for devs by a dev



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

The same or different?

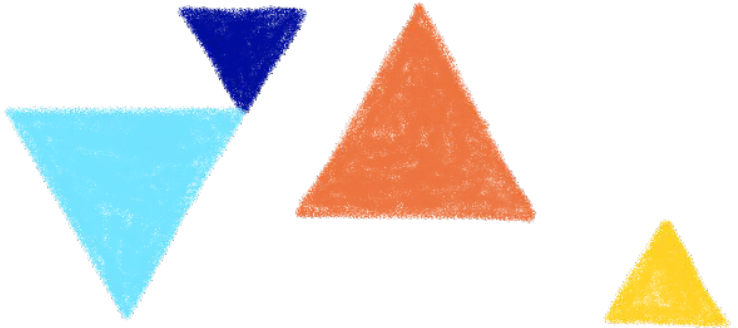
Design System

Component Catalog

Style Guide

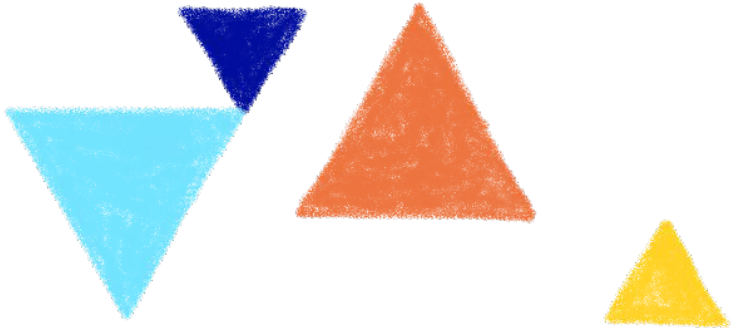
Style Guides

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

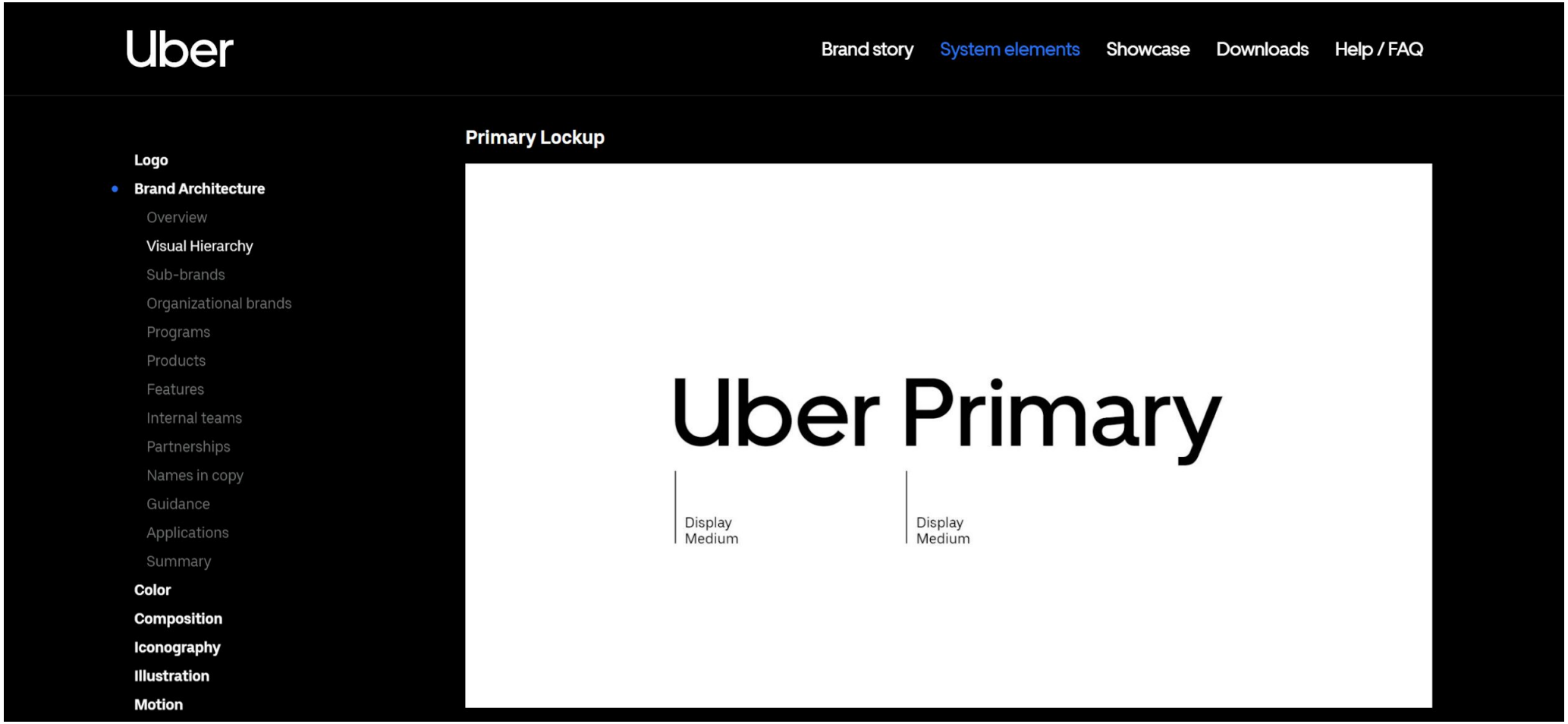


Style Guides

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

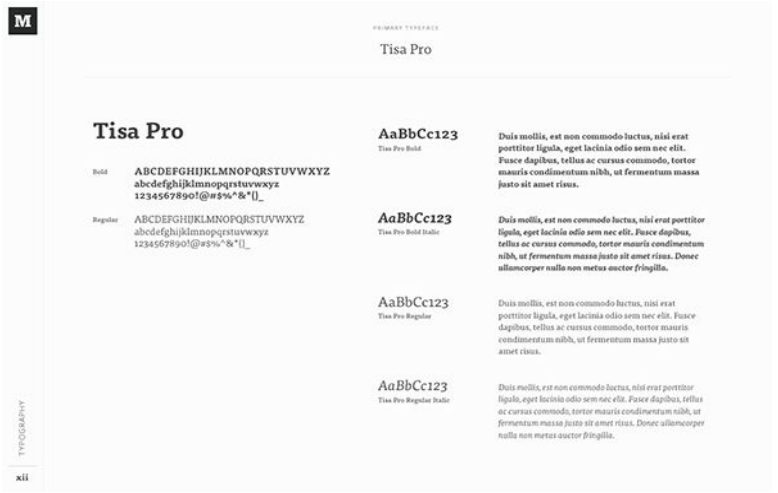


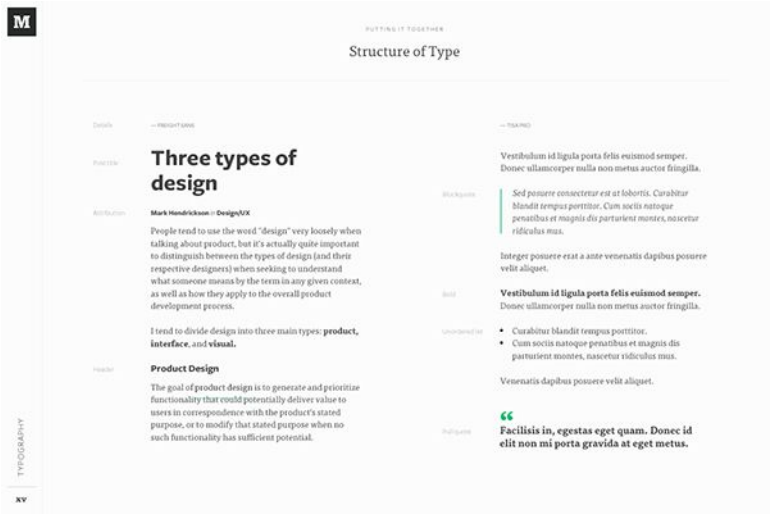
Style Guide Example: Uber

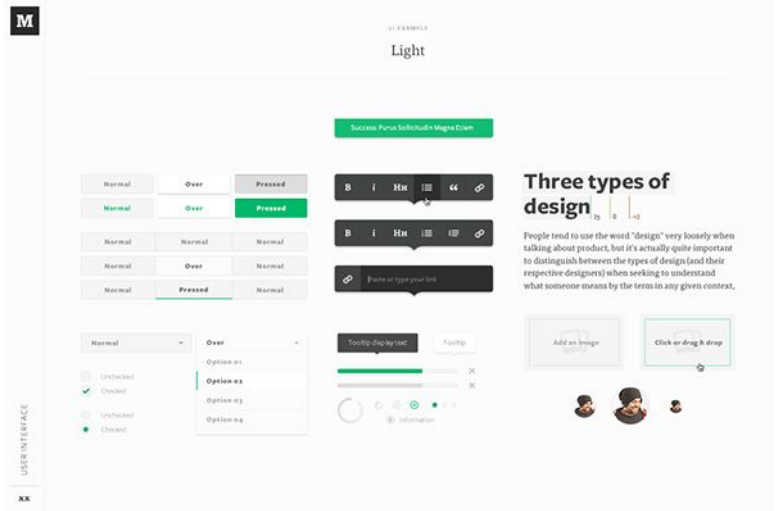


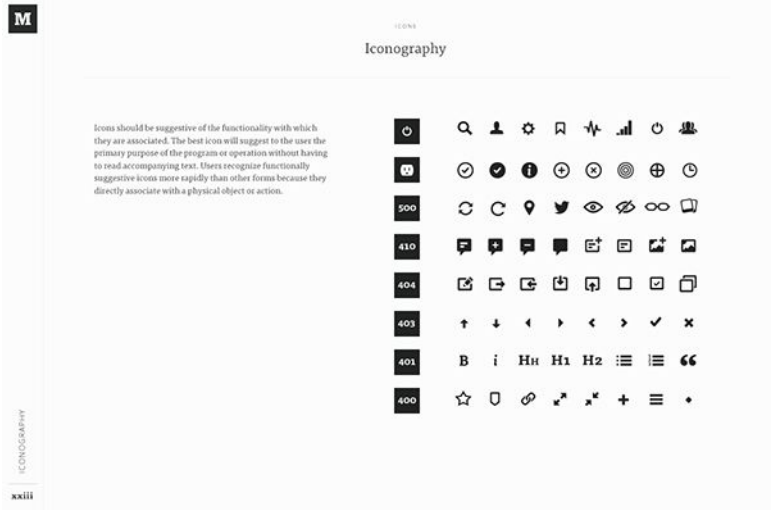
https://brand.uber.com/

Style Guide Example: Medium



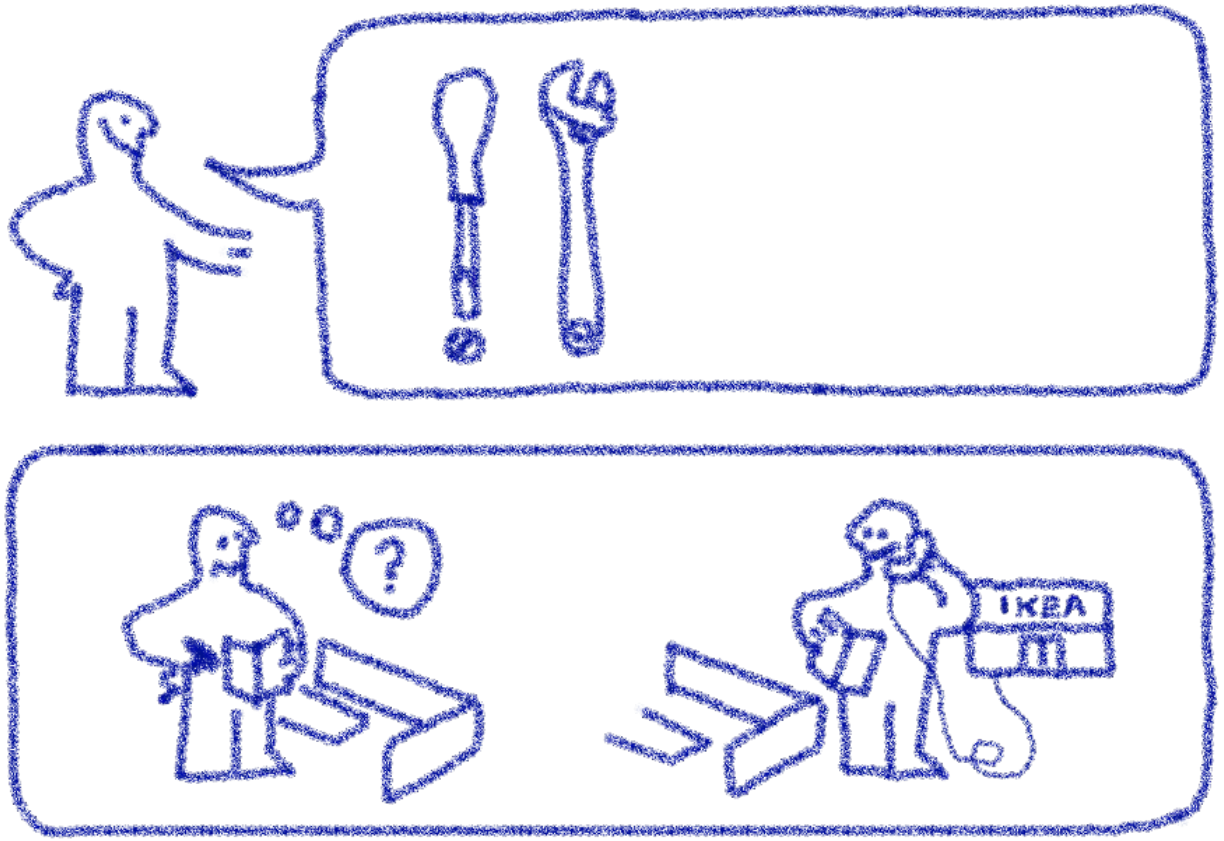






https://www.behance.net/gallery/7226653/Medium-Brand-Development

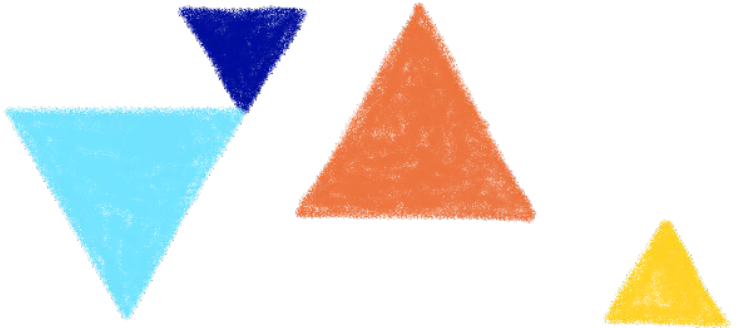
Style Guides alone are ambiguous



Interpretation needed to adapt
the preconisation to the use case

Component Catalogs

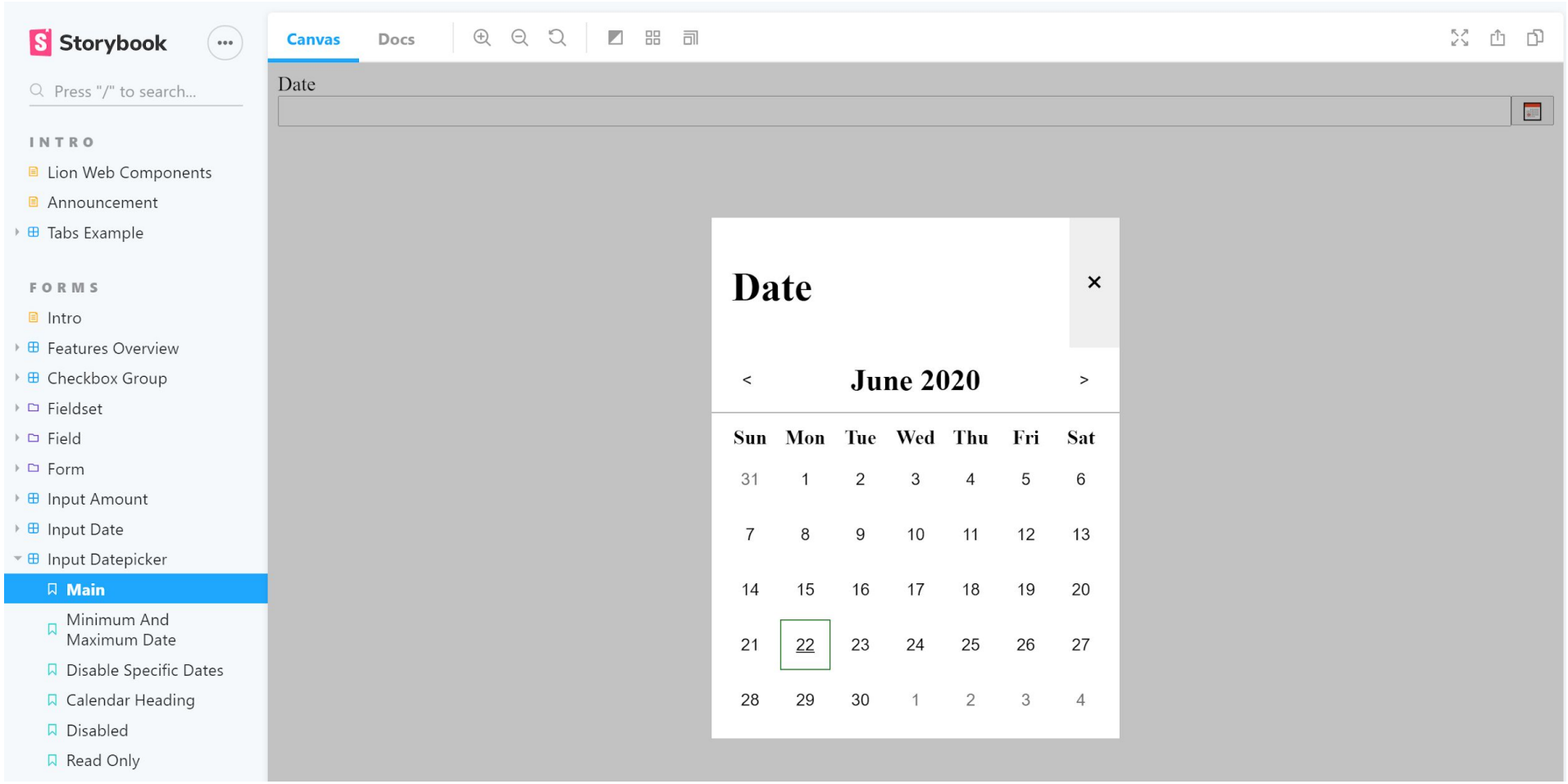
A **component catalog** is a **repository** of components, with one or several **implementations**, code **examples** and **technical documentation**



Component Catalog example: Bootstrap

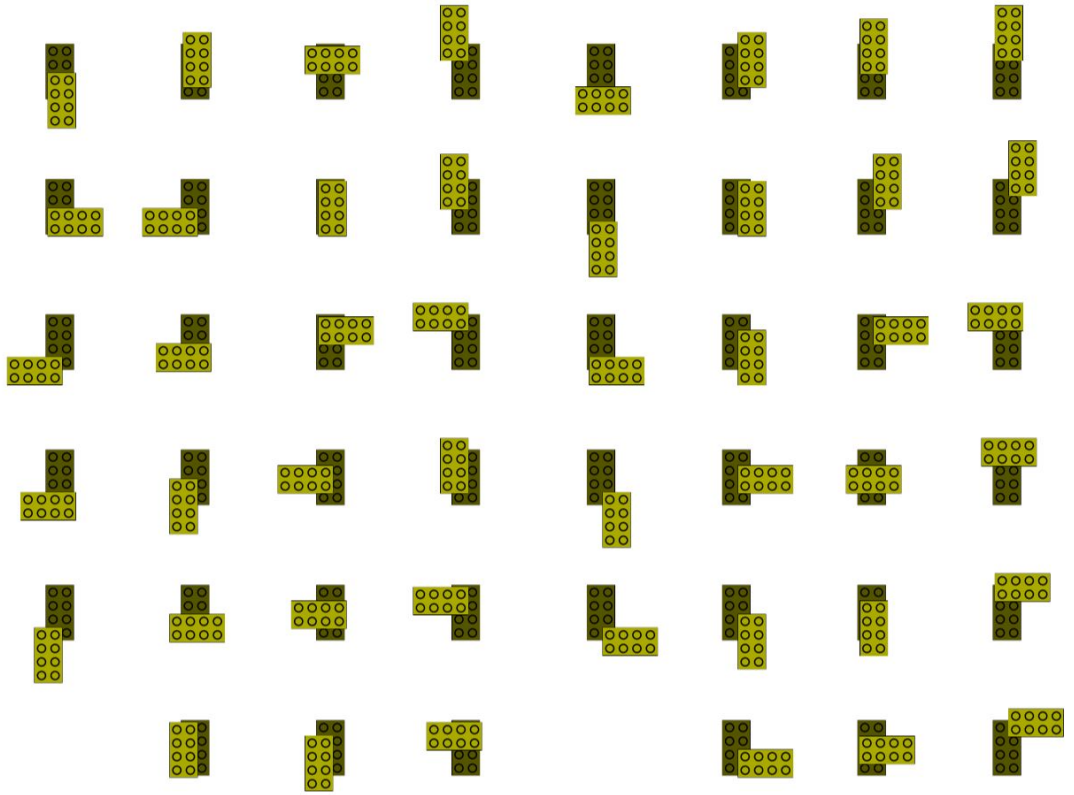
https://getbootstrap.com/

Component Catalog Example: ING's Lion



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

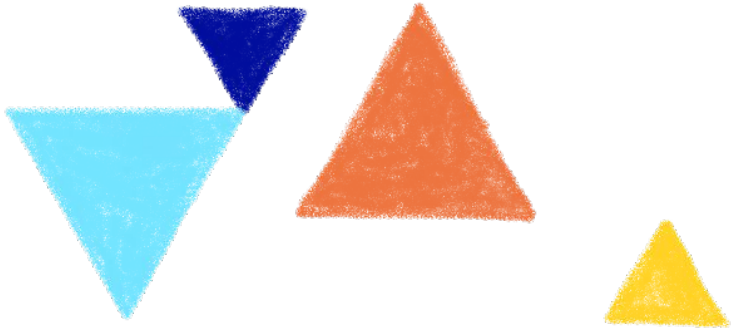
Catalogs alone create inconsistency



Like using the same LEGO bricks
to create very different objects

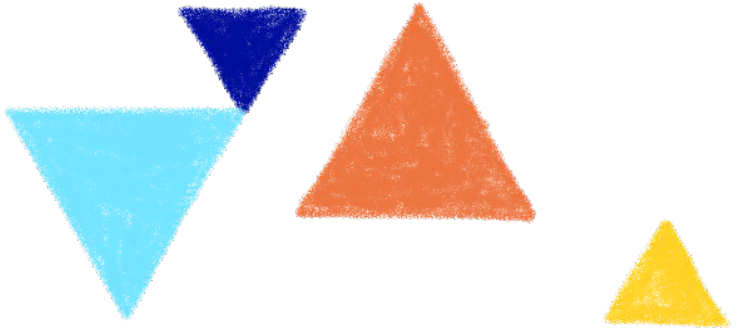
Design Systems

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

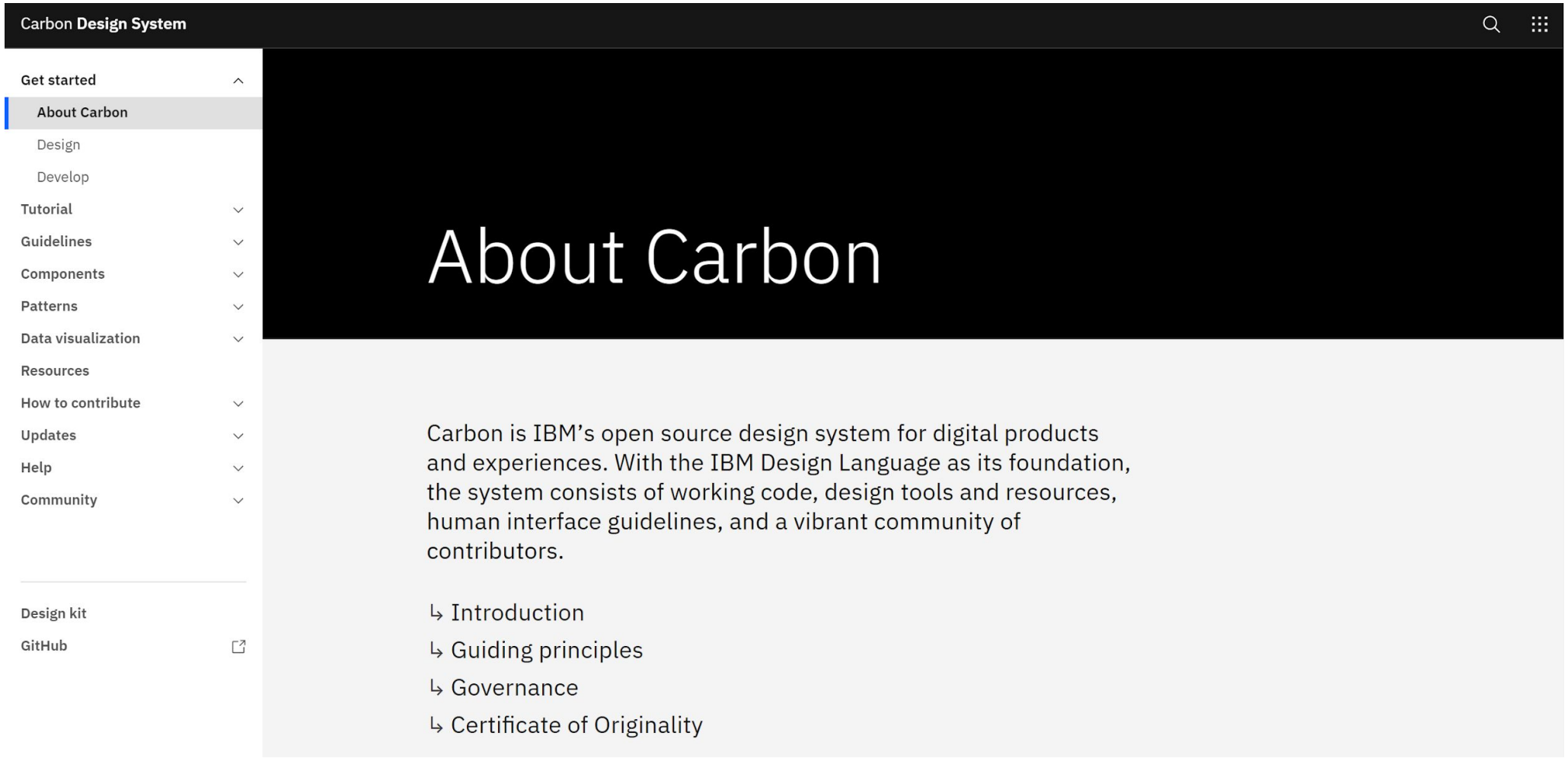


Design systems

Design System ≈

Style Guide + Component Catalog

Example: Carbon Design System



https://www.carbondesignsystem.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

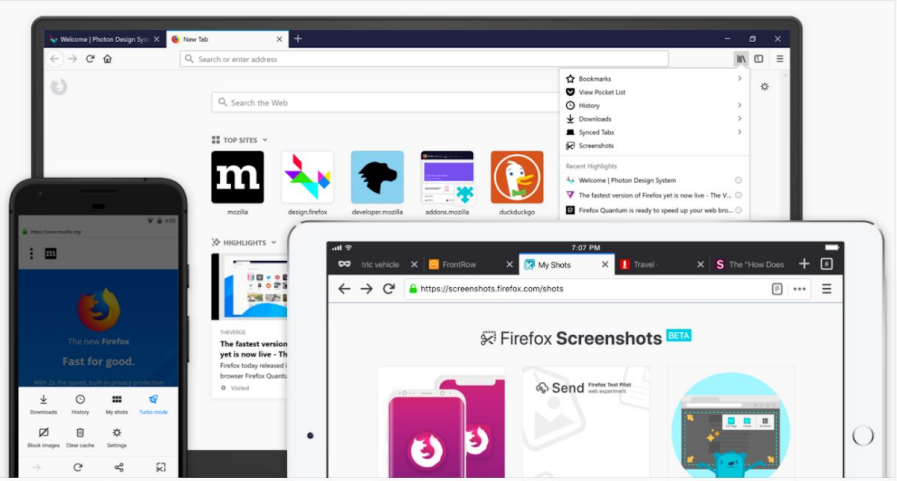
Patterns

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.

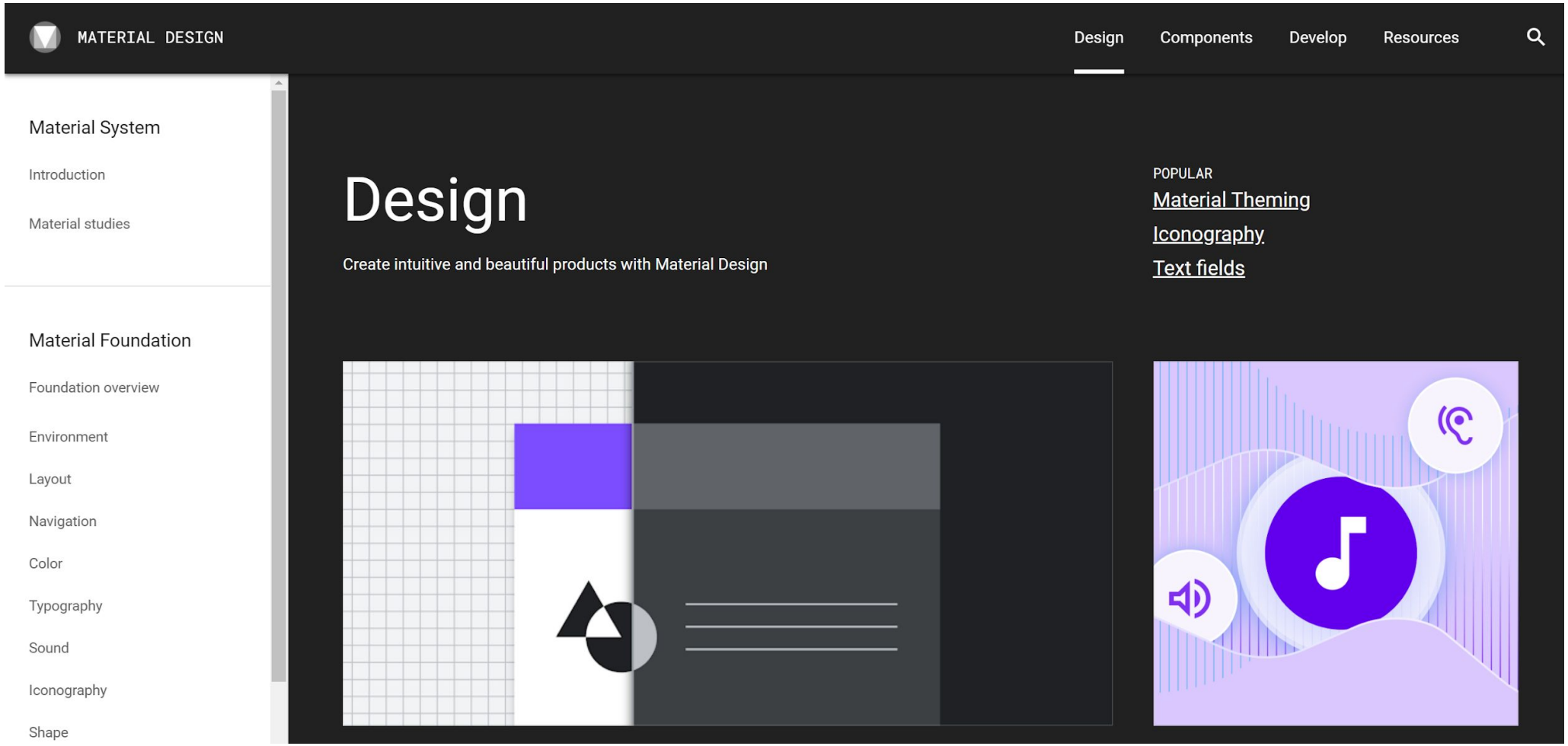
The Photon Design System houses guidelines, reusable UI components, templates, and other resources to help you create products for Firefox users. It is flexible and always evolving to serve the best Firefox experience for every situation.

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

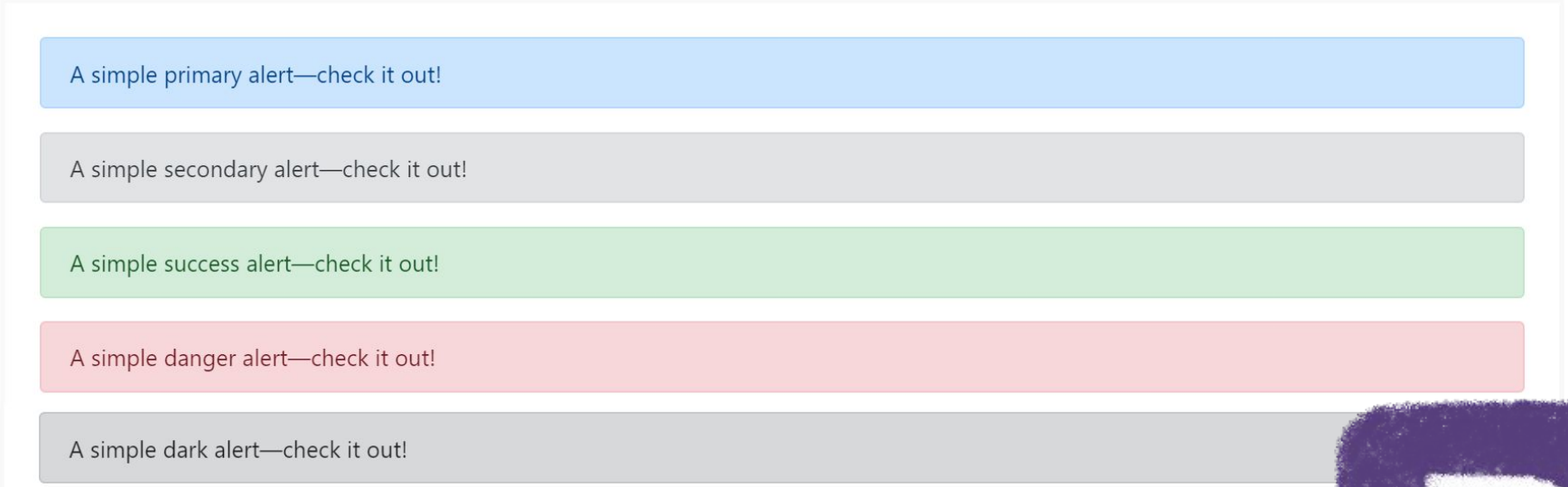


https://material.io/

The component catalog

The poor relative of the Design System family

Let's choose a simple example




```
<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

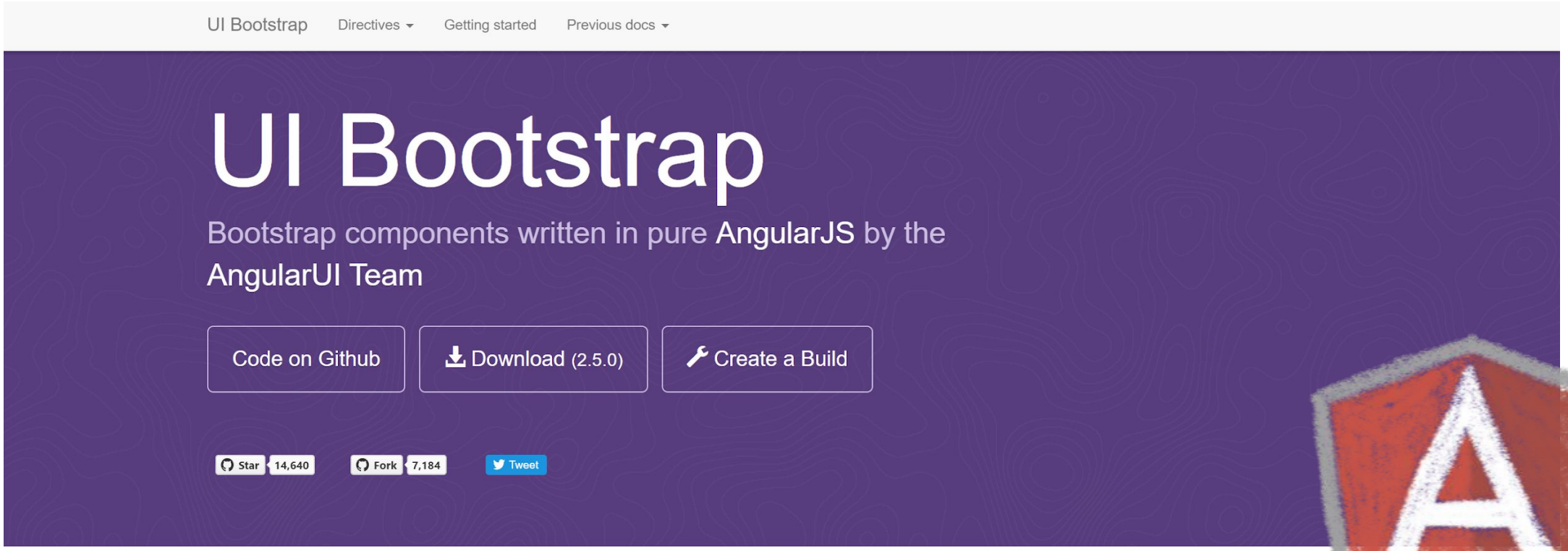
Default buttons

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
Default	`btn`	Standard gray button with gradient
Primary	`btn btn-primary`	Provides extra visual weight and identifies the primary action in a set of buttons
Info	`btn btn-info`	Used as an alternative to the default styles
Success	`btn btn-success`	Indicates a successful or positive action
Warning	`btn btn-warning`	Indicates caution should be taken with this action
Danger	`btn btn-danger`	Indicates a dangerous or potentially negative action
Inverse	`btn btn-inverse`	Alternate dark gray button, not tied to a semantic action or use
Link	`btn btn-link`	Deemphasize a button by making it look like a link while maintaining button behavior

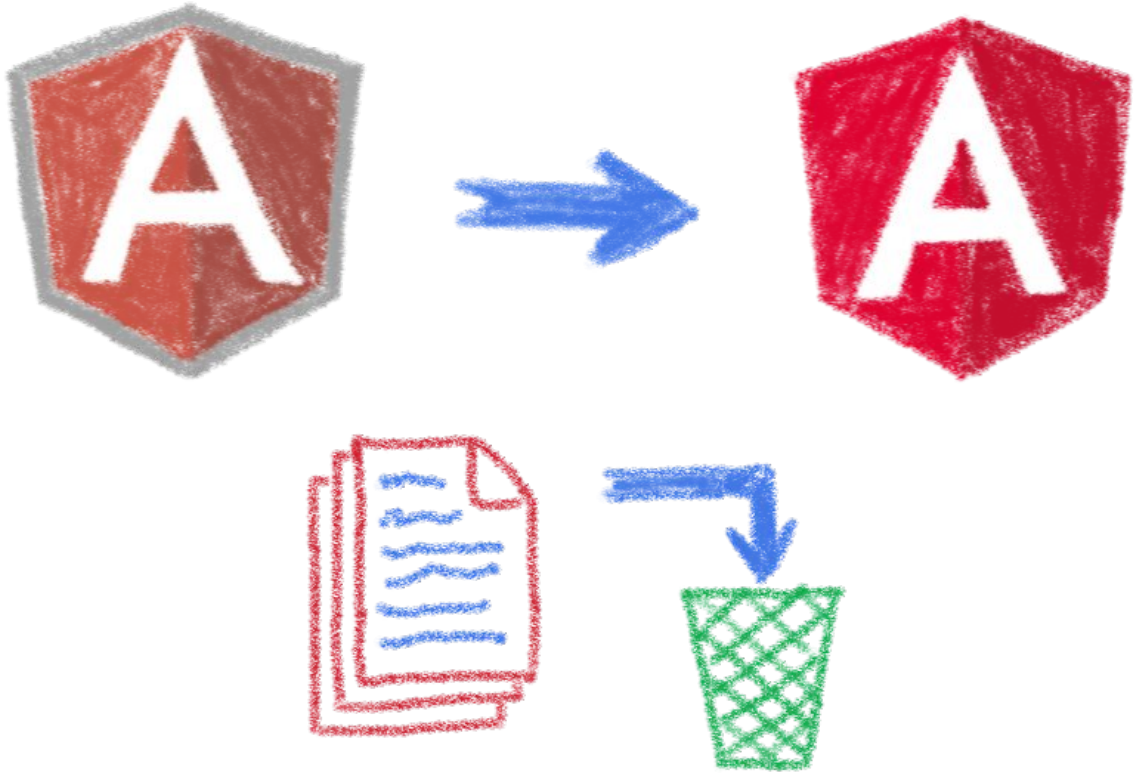
Components defined in HTML, CSS and some jQuery

Then it was AngularJS time...



And new reference implementations were needed

But you know the sad story...

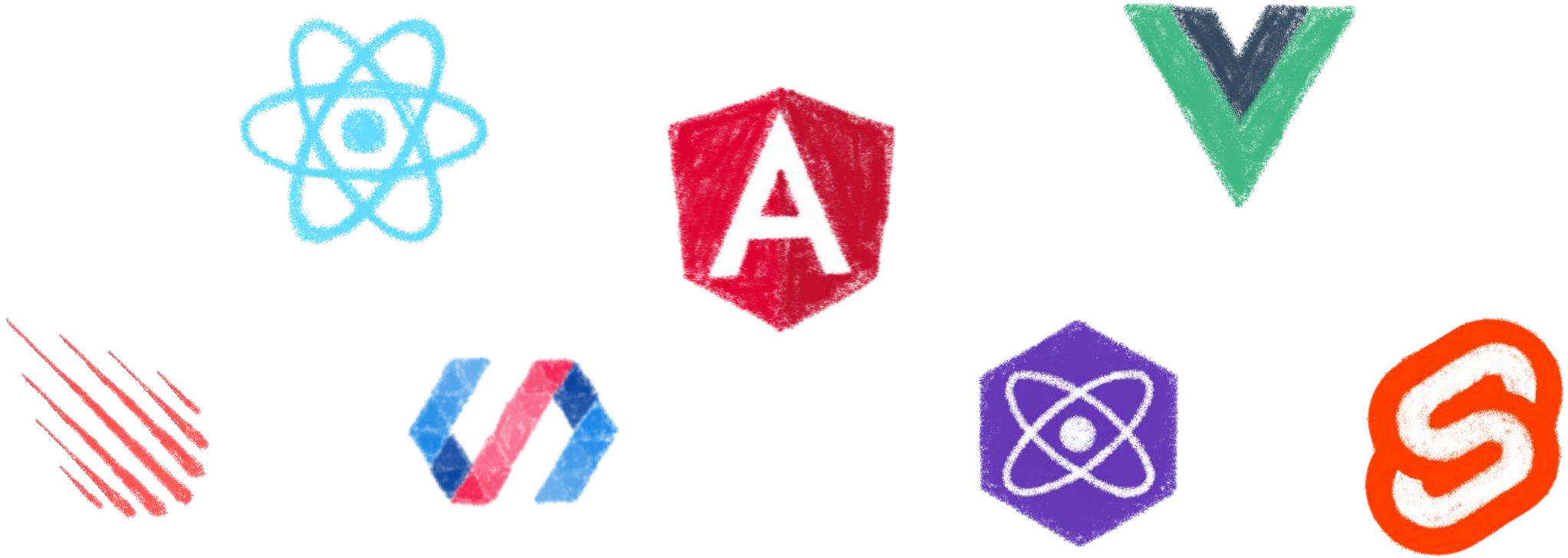


All UI Bootstrap based catalogs woke up with an obsolete implementation

Worry no more, let's do Angular!

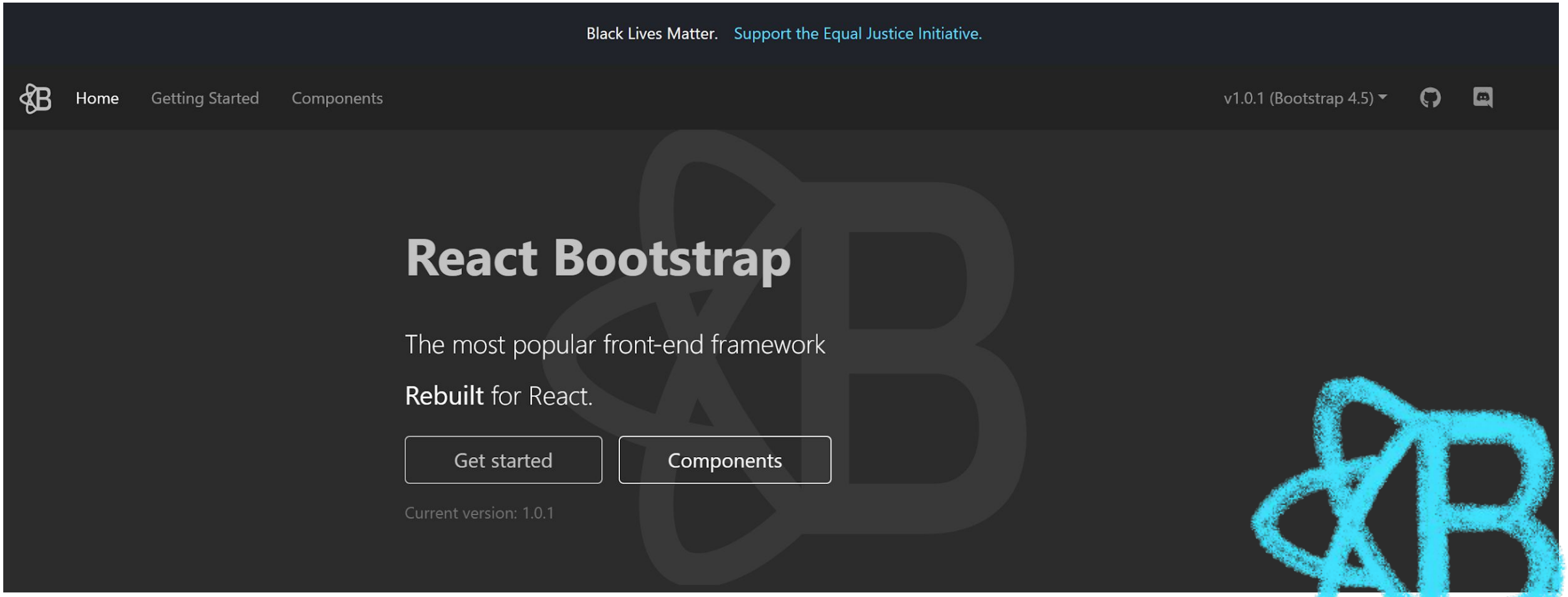
ng-bootstrap to the rescue

But times had changed...



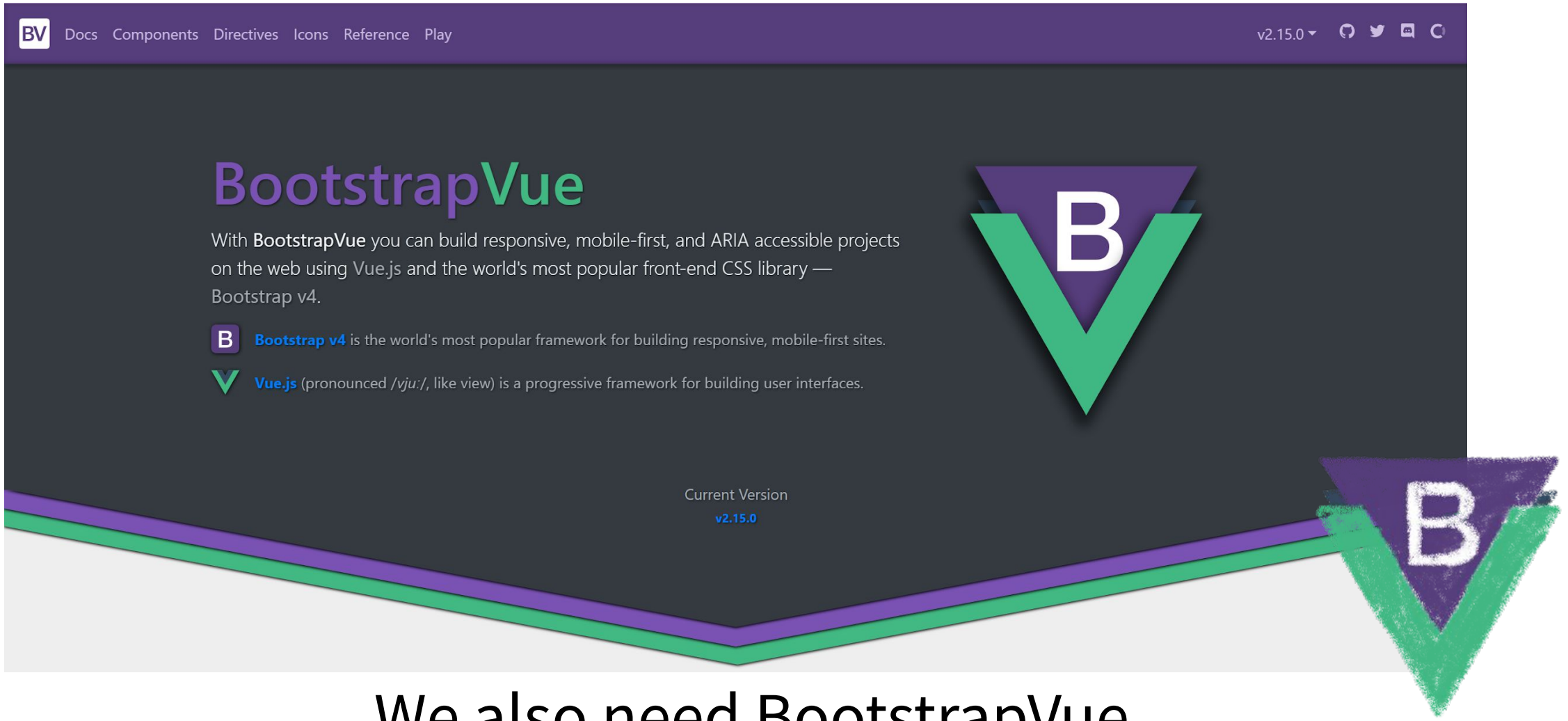
In 2017 Angular is only one more in the clique

React is the new Big Thing™



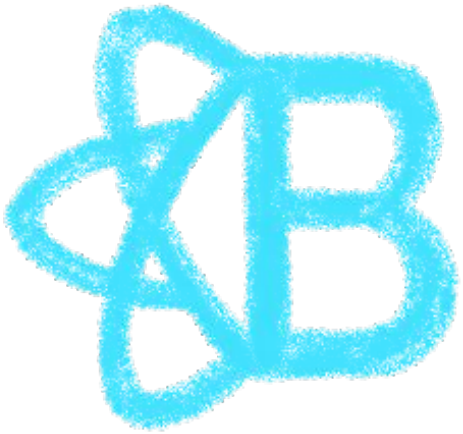
So let's build React Bootstrap...

Wait, what about Vue?

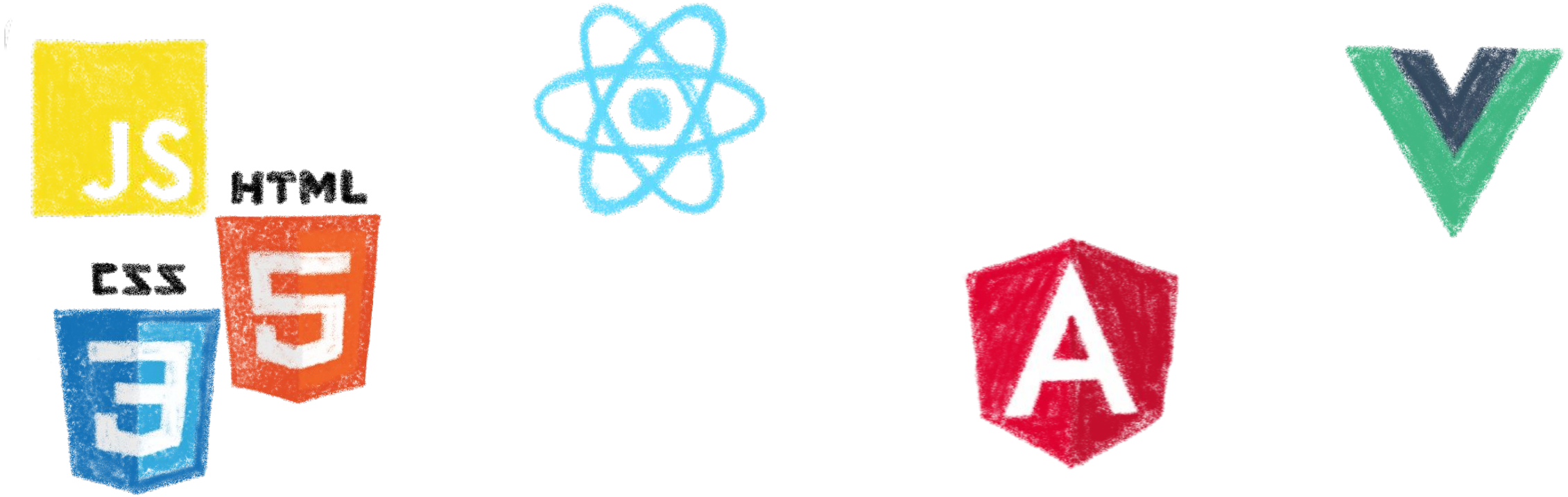


We also need BootstrapVue

OK, I think you see my point...

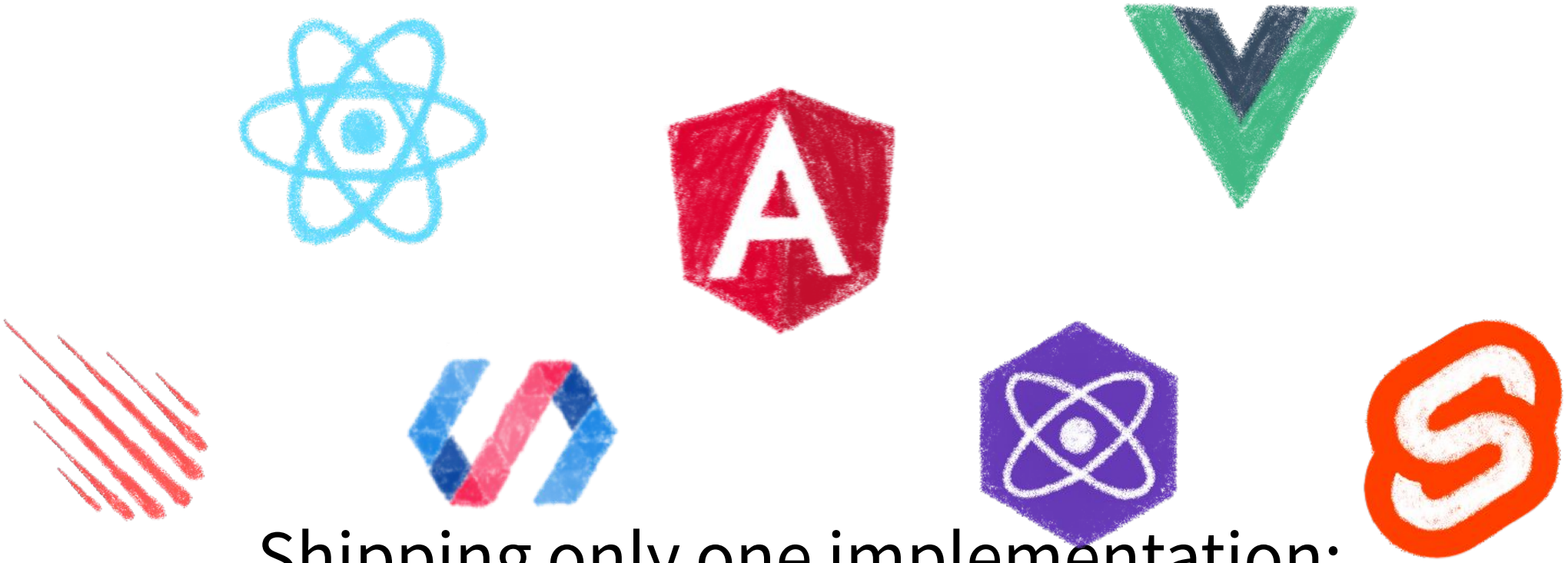


Most Design System do a choice



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

Both choices are problematic



Shipping only one implementation:

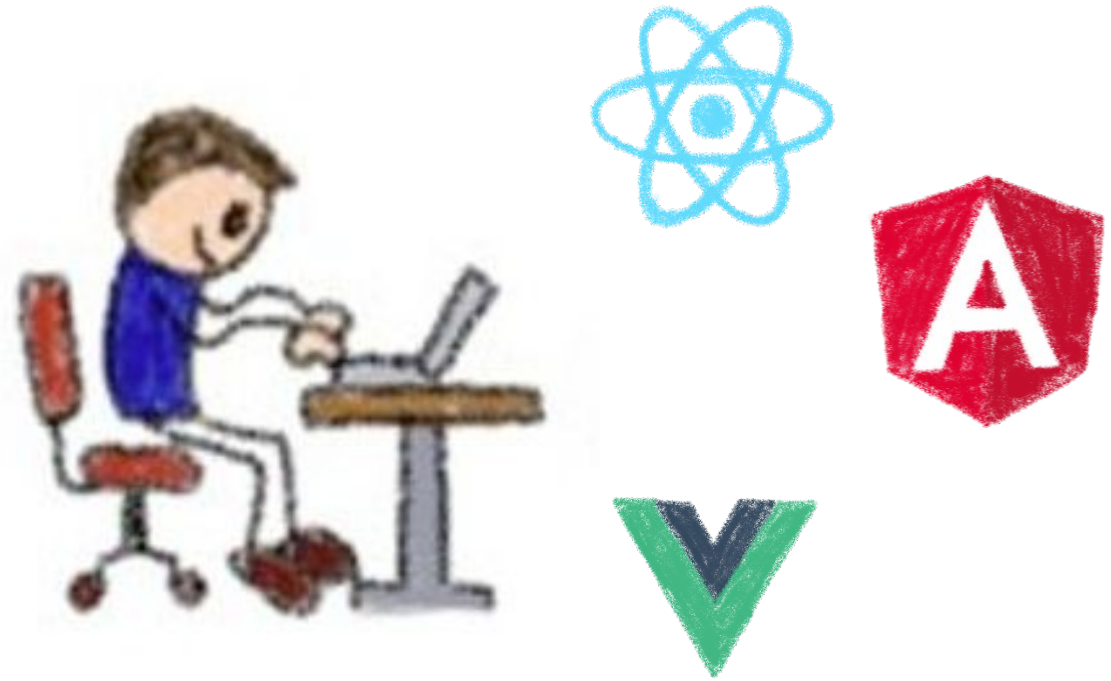
Web dev ecosystem changes quickly and
almost nobody keeps the same framework for years...

Both choices are problematic

Shipping several implementations:

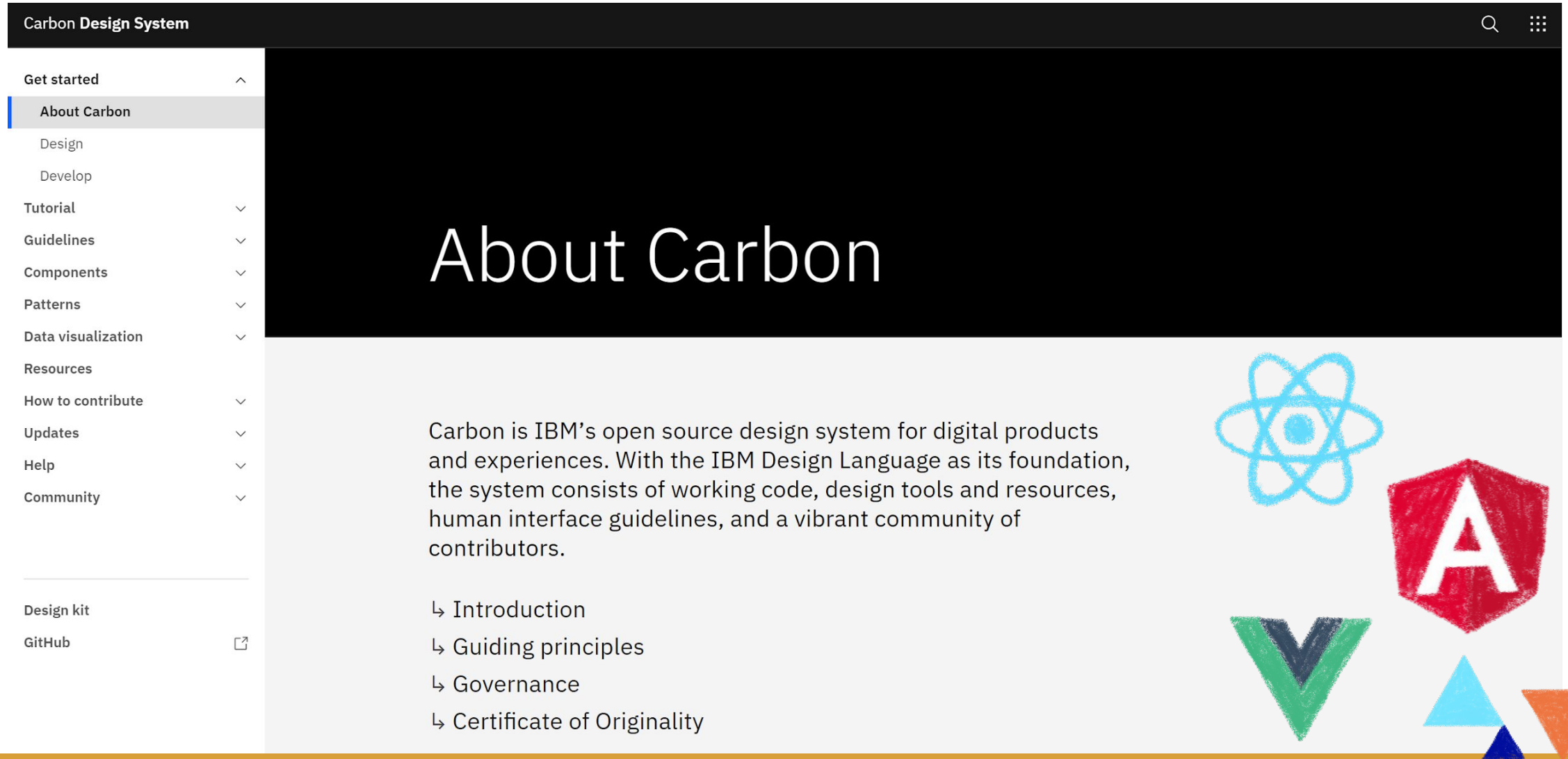
You need to maintain all the implementation…
and you still miss some others

Incomplete catalogs are problematic



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

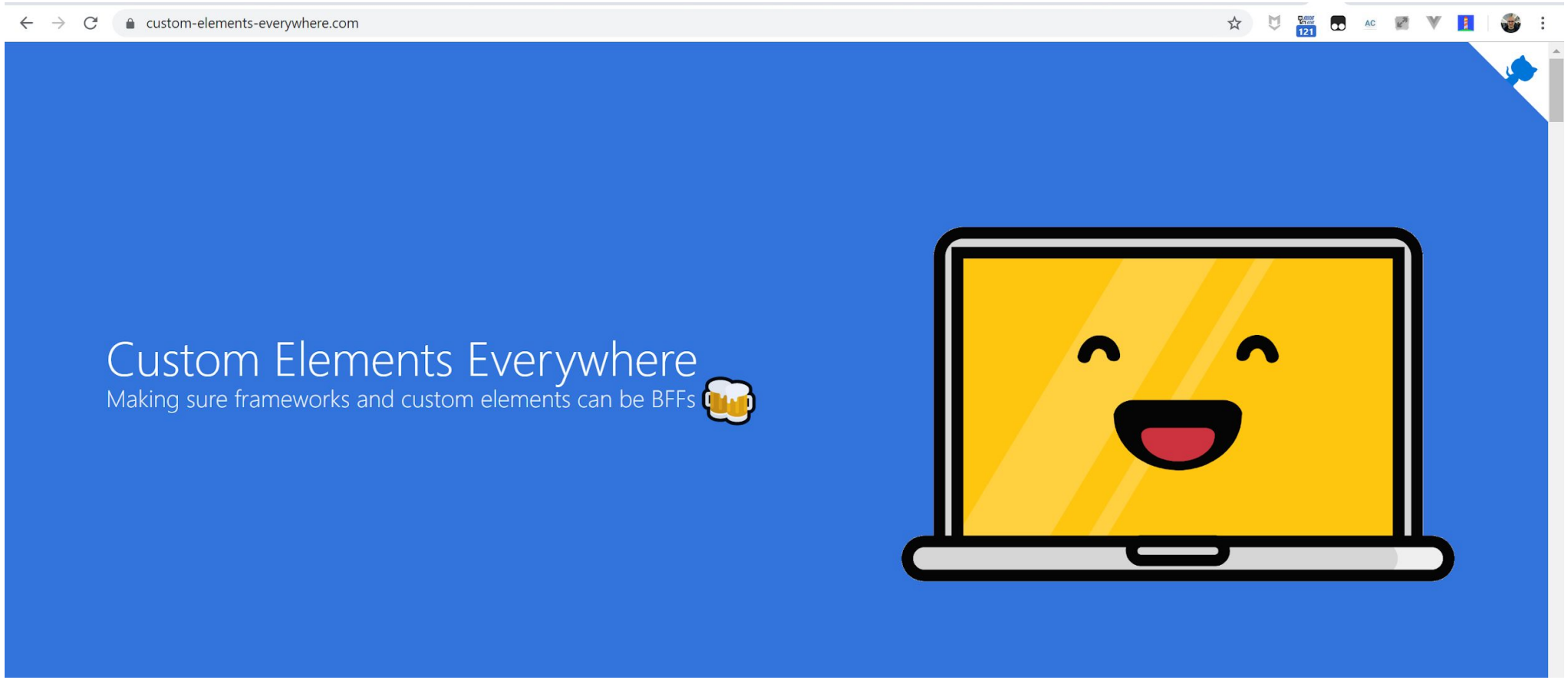
Example: Carbon Design System



Web Components & Design Systems

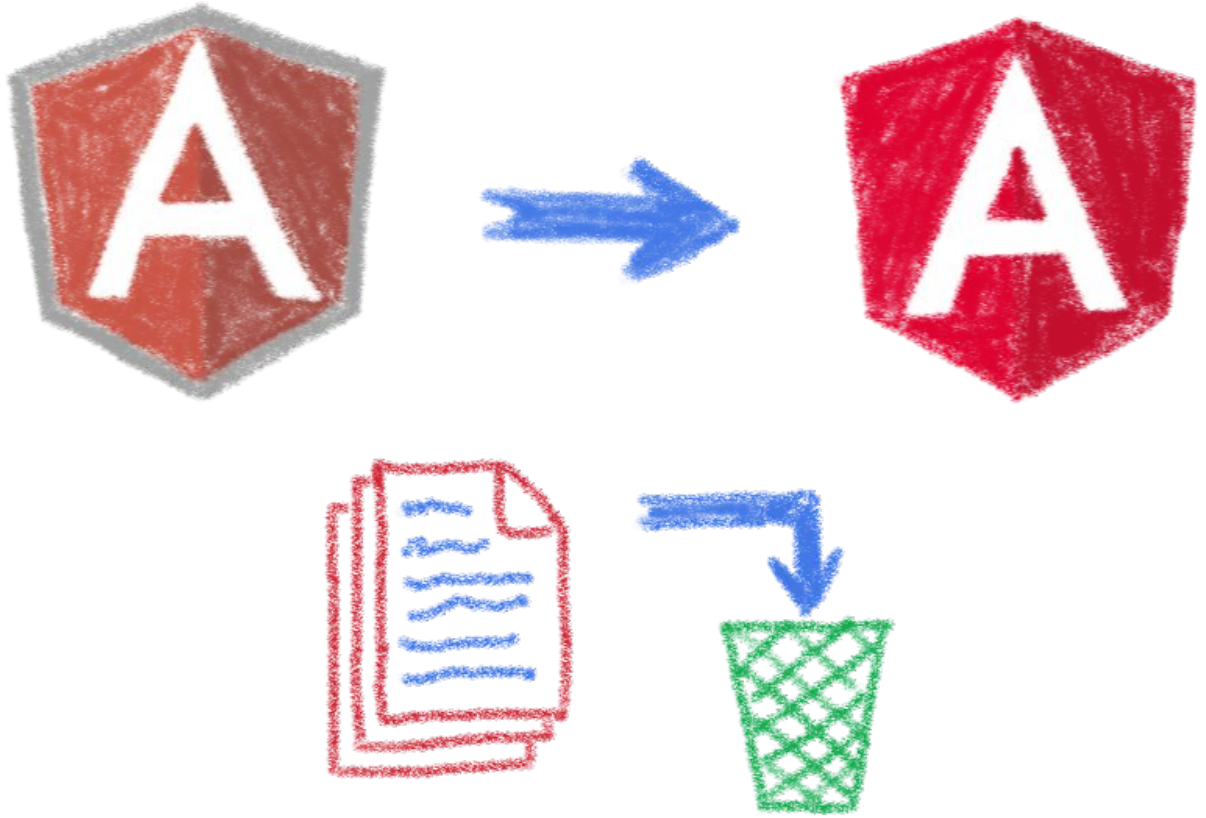
A match made in heaven

Compatibility is on Web Components side



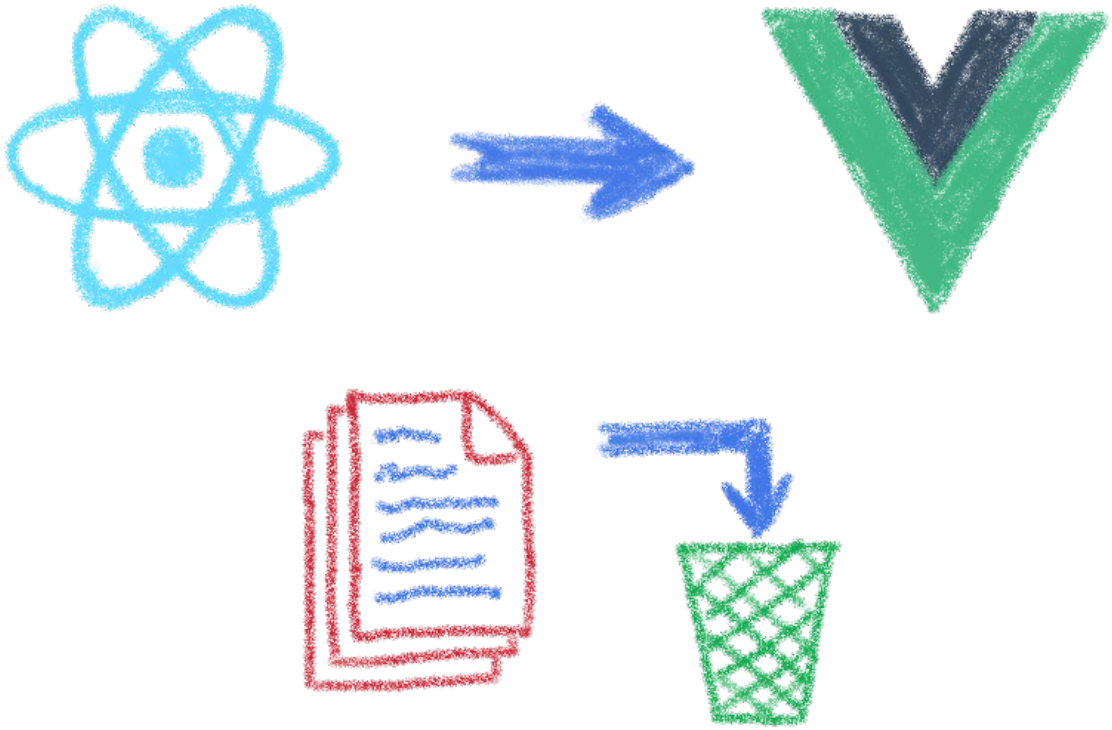
Web Components everywhere, baby!

Do you remember AngularJS?



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

The pain of switching frameworks?

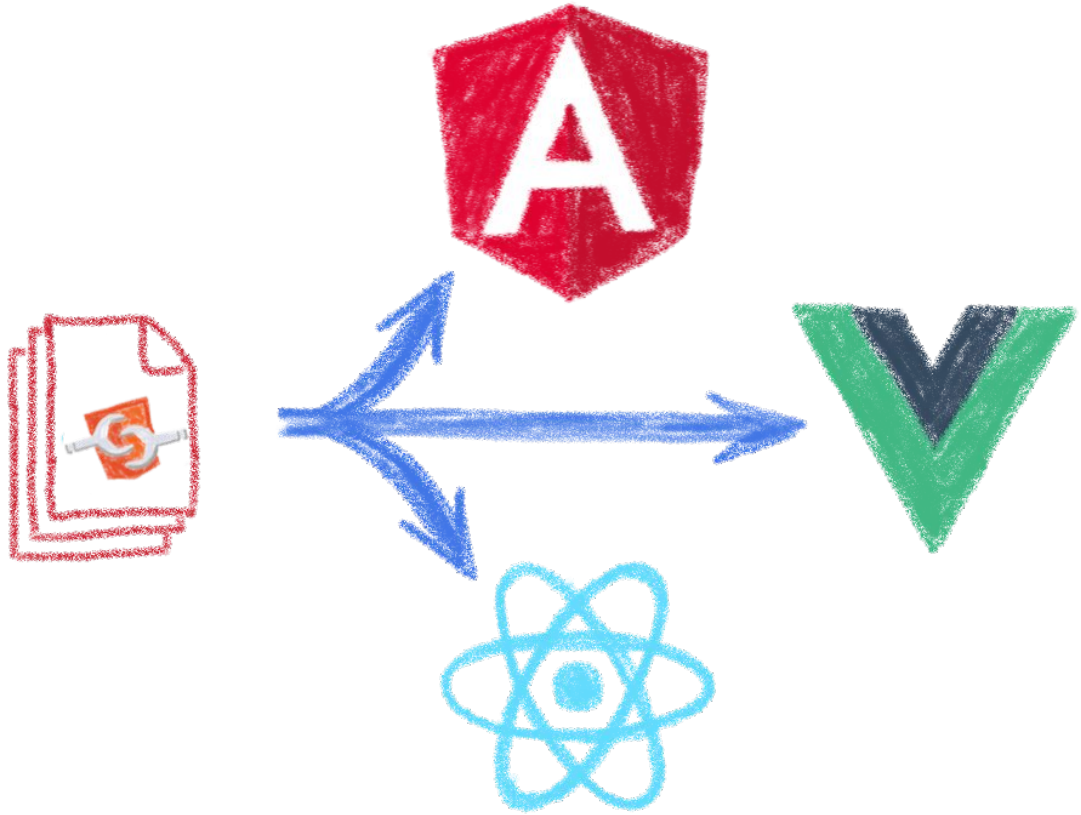


Rewriting once again your code...

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 Official SpaceX 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

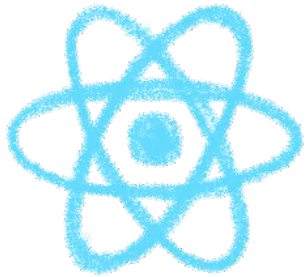
We use a reactive programming library that we developed in house. - Sofian

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 spaaaaaaaace 🚀

You can have a single implementation



And it simply works everywhere

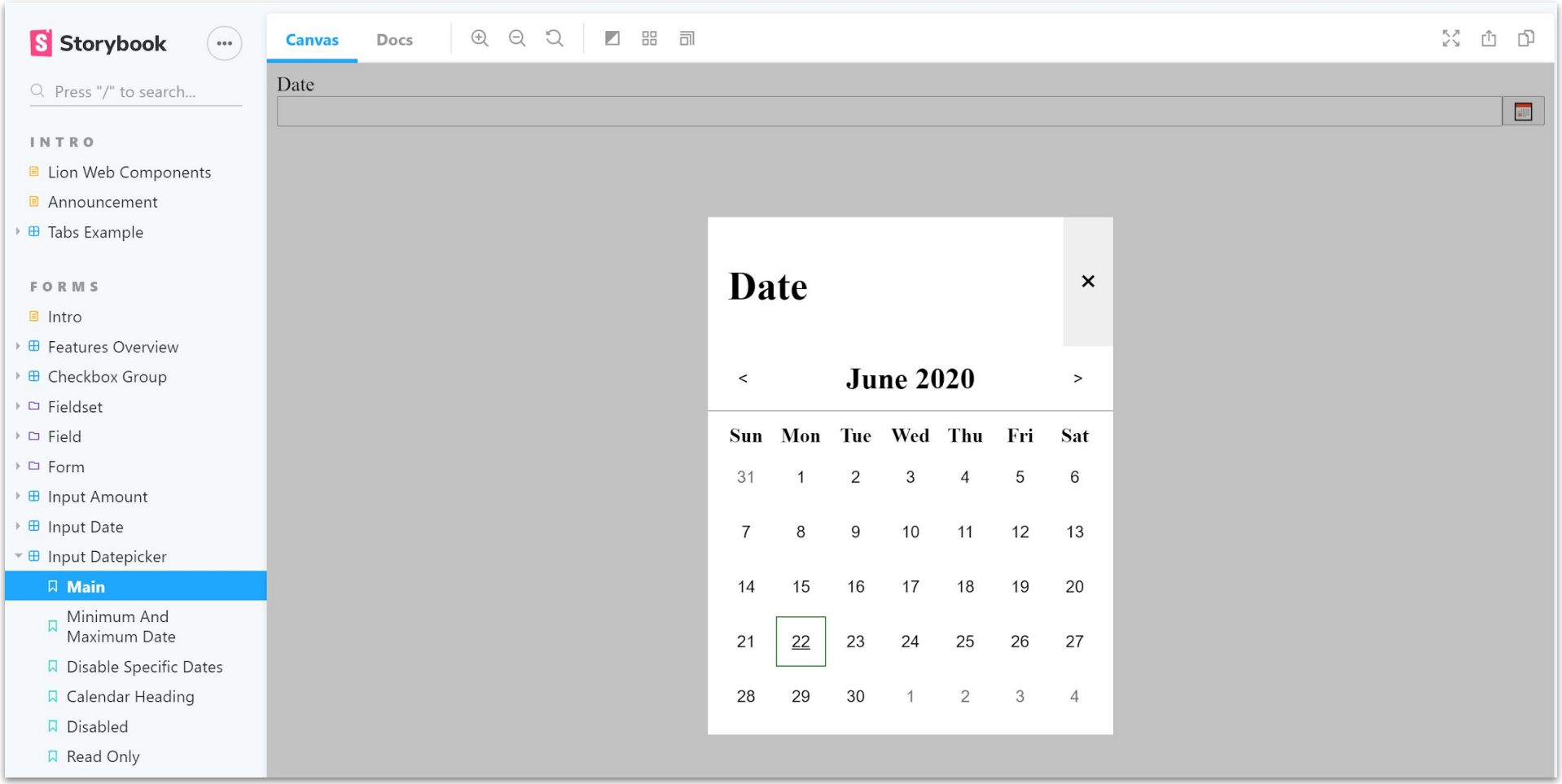
When you need interoperability

Nothing beats the standard

But how to do it?

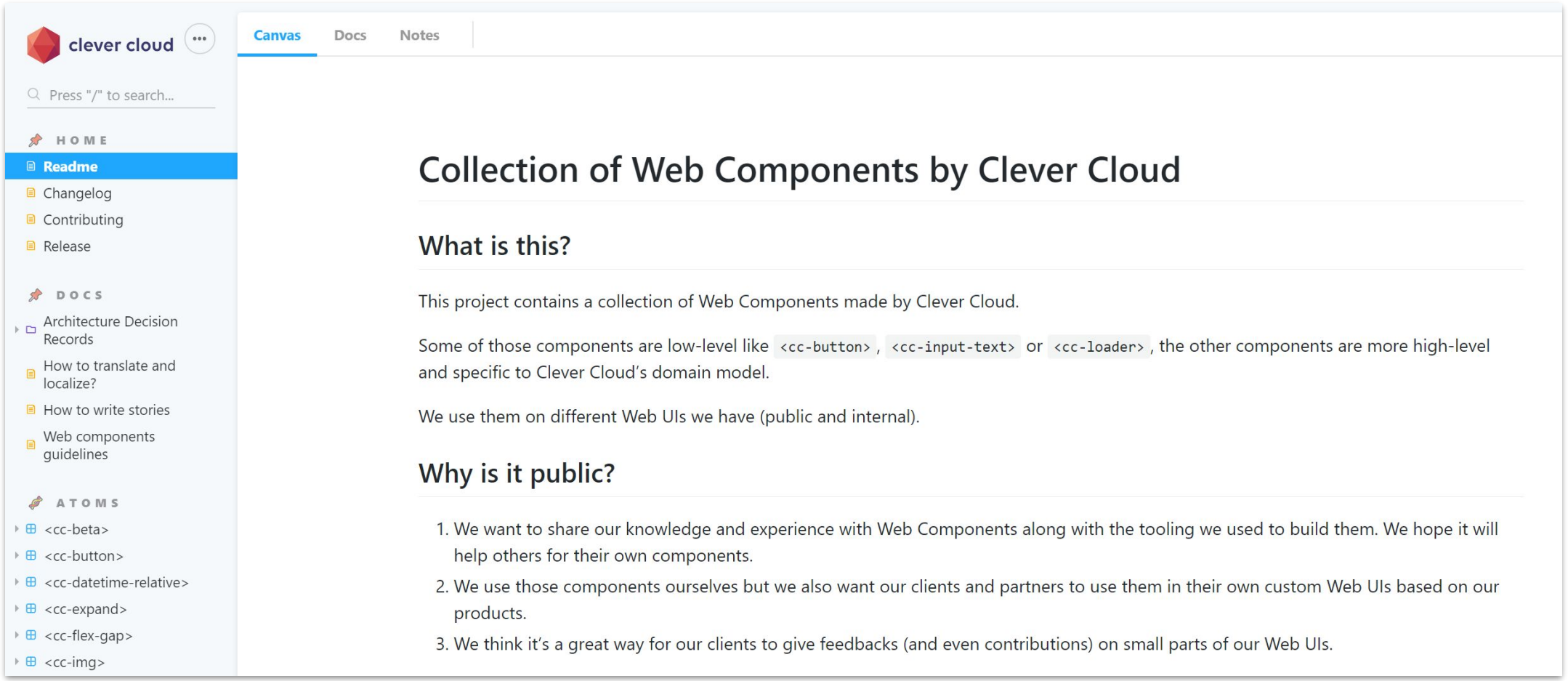
Designing, developing and managing a catalog of Web Components

Learning from the best



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

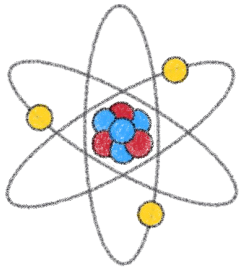
Learning from the best

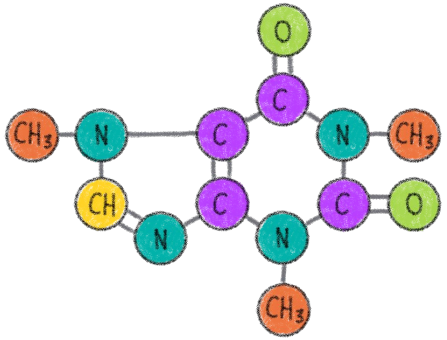


https://github.com/CleverCloud/clever-components

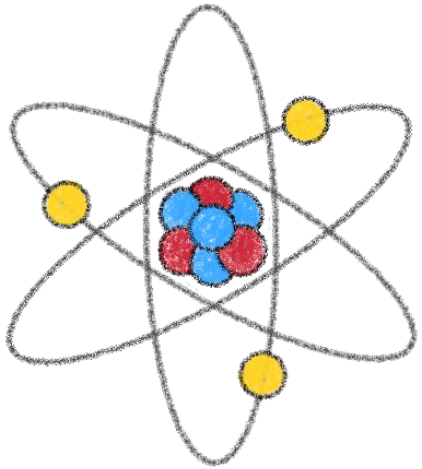
What kind of components?

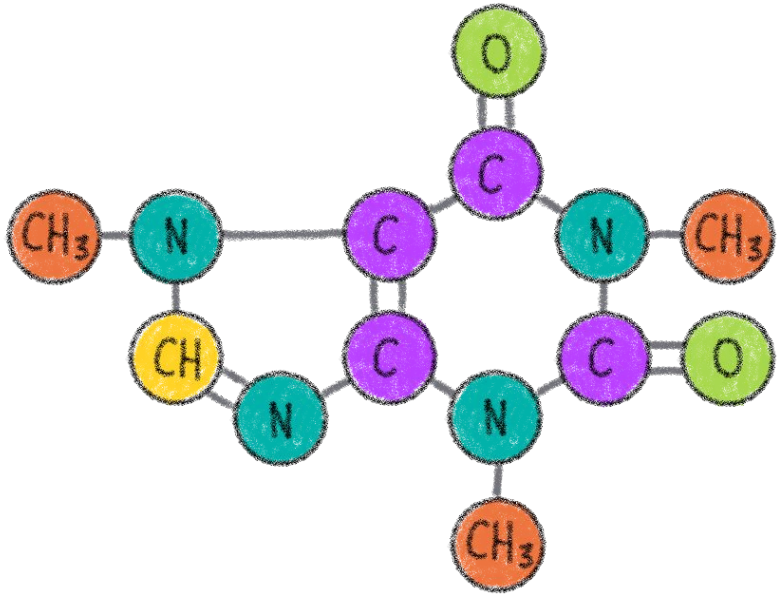
From little atomic blocs to big smart components, and everything in between





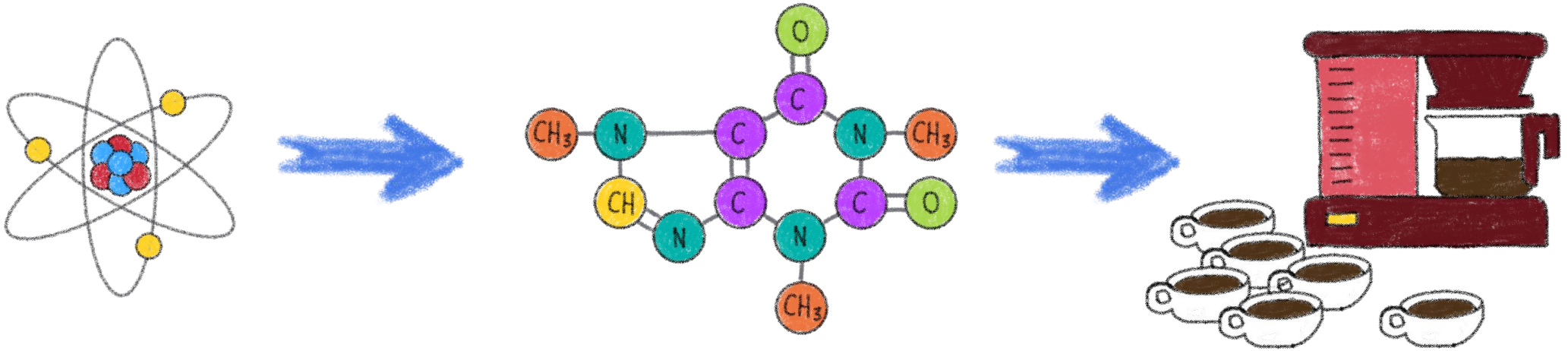
A matter of size and complexity

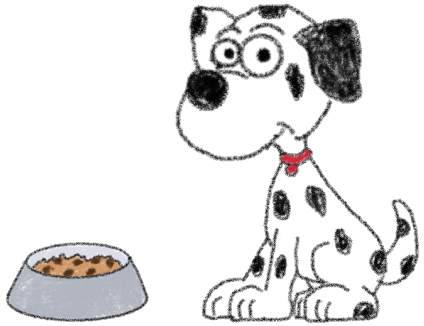




What kind(s) of components you want to build

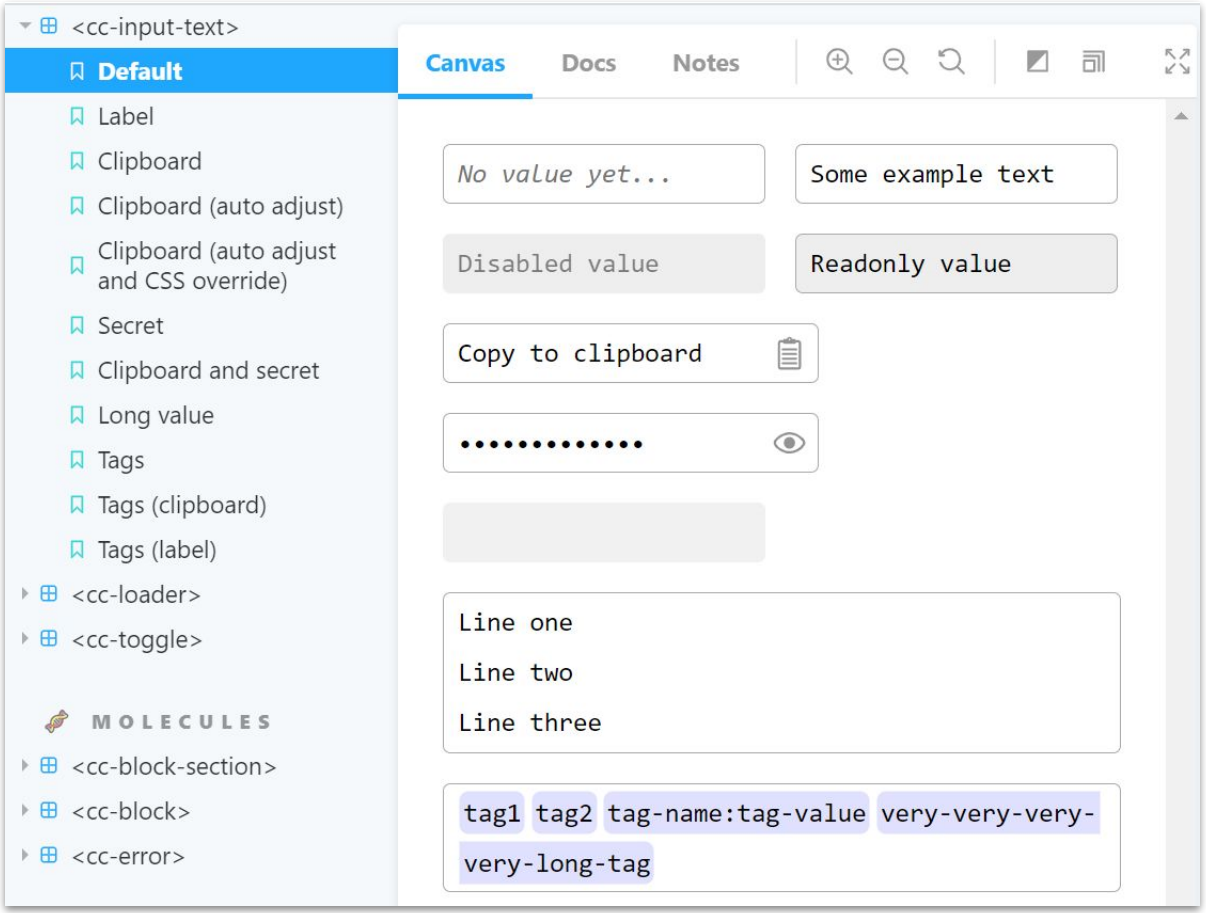
Build from the bottom and go up

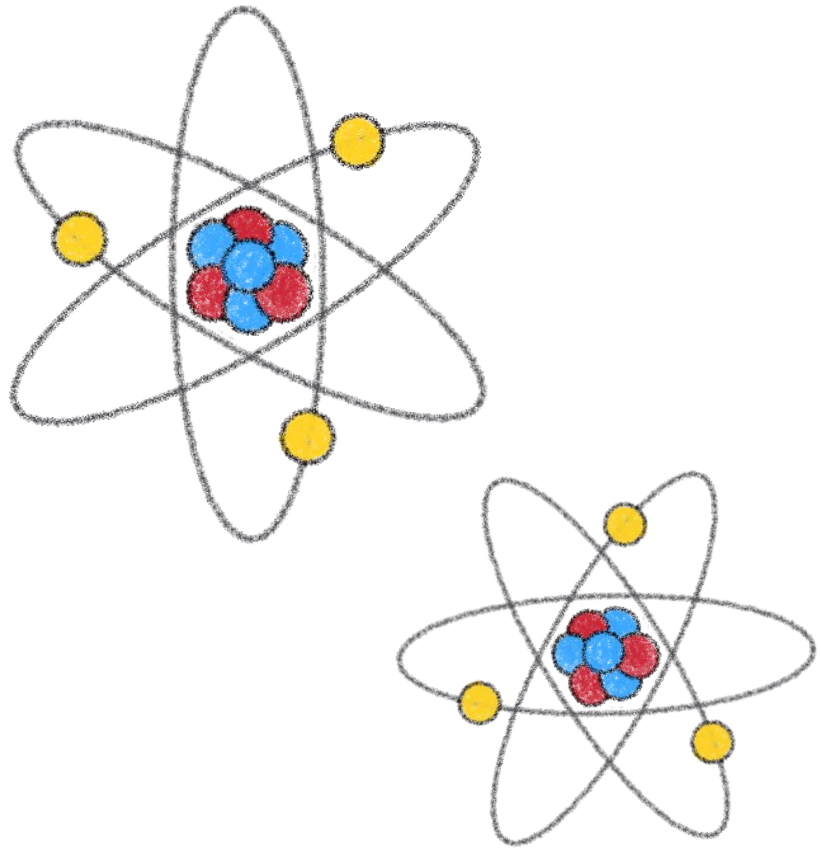




Eat your own dog food

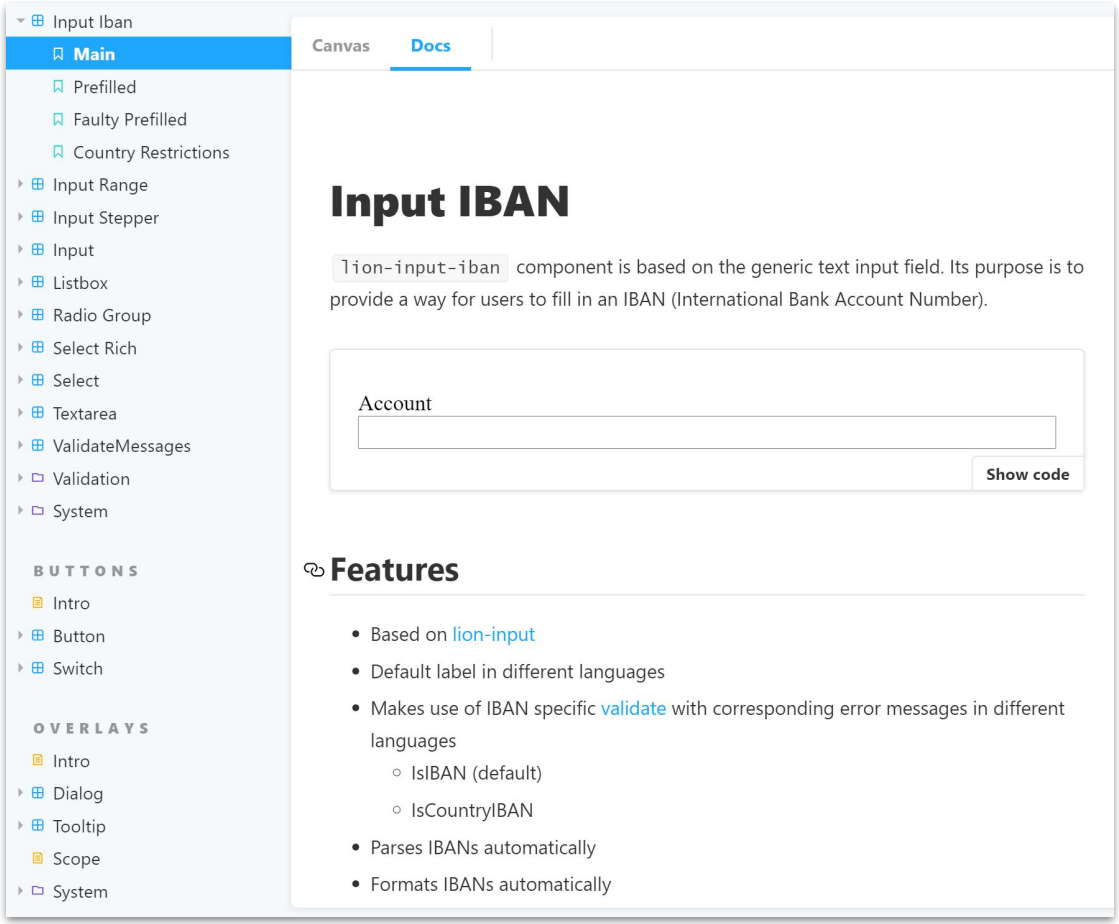
And how to choose the atoms?

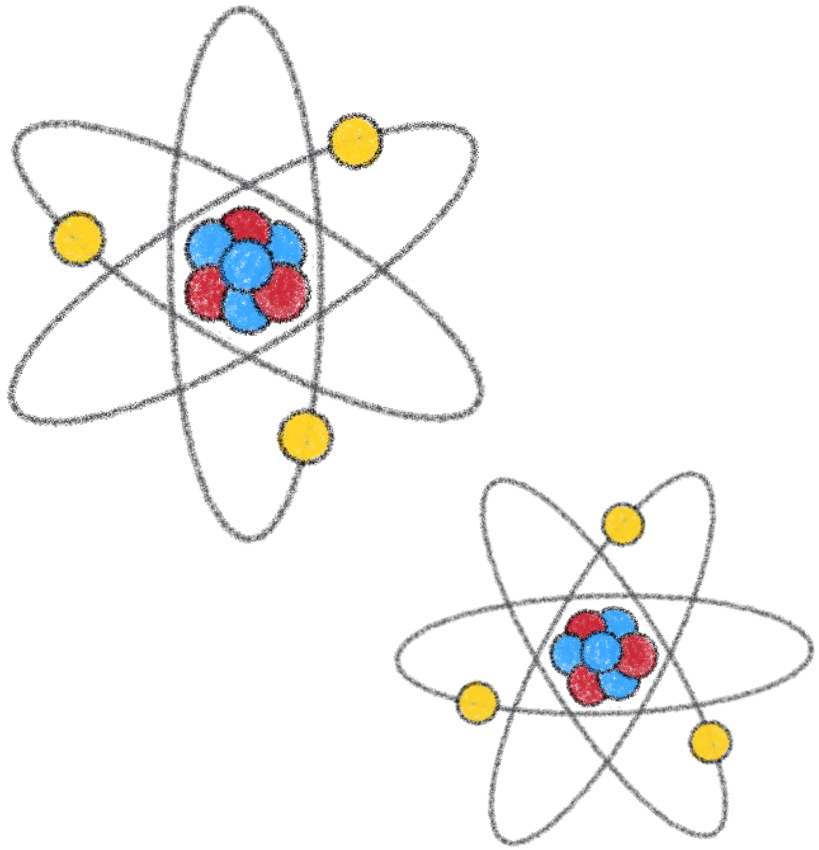




Flexibility and configurability are key

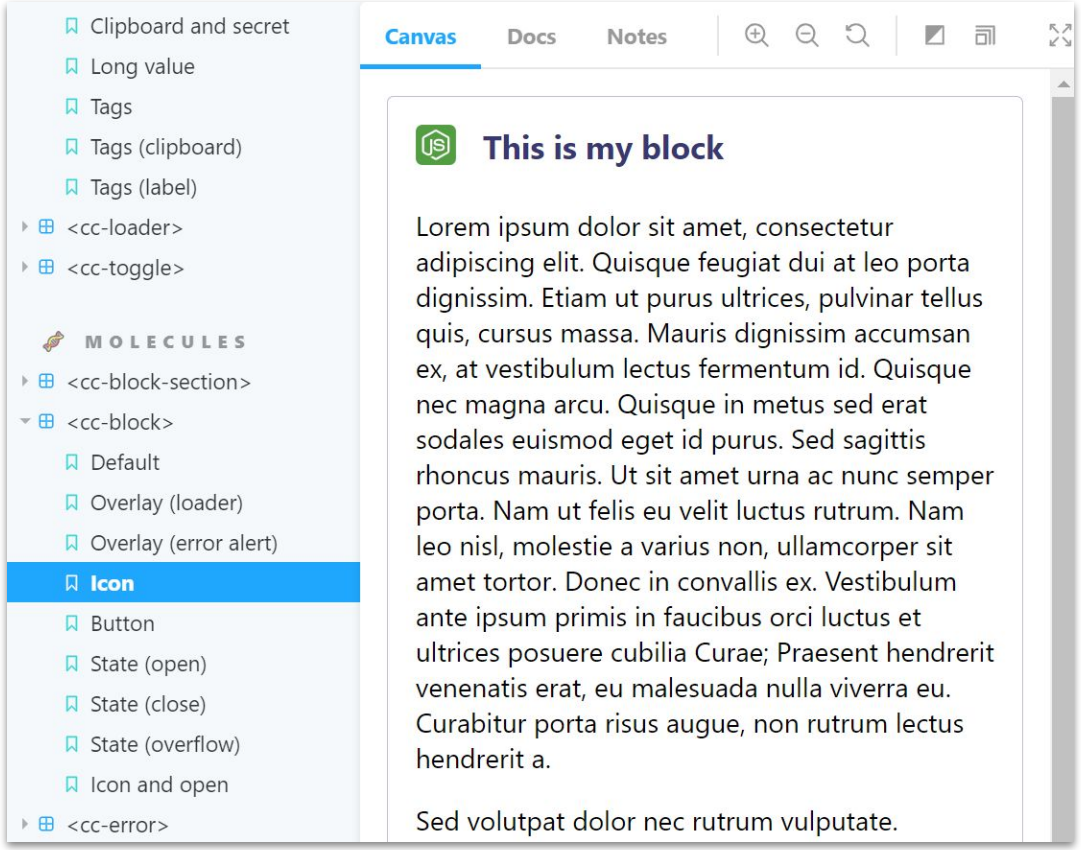
And how to choose the atoms?

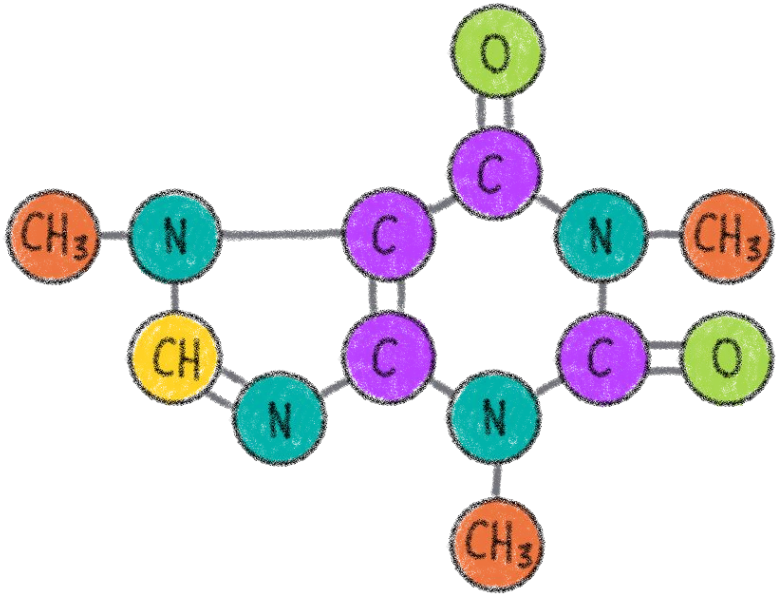




Encode often used patterns

And what about the molecules?

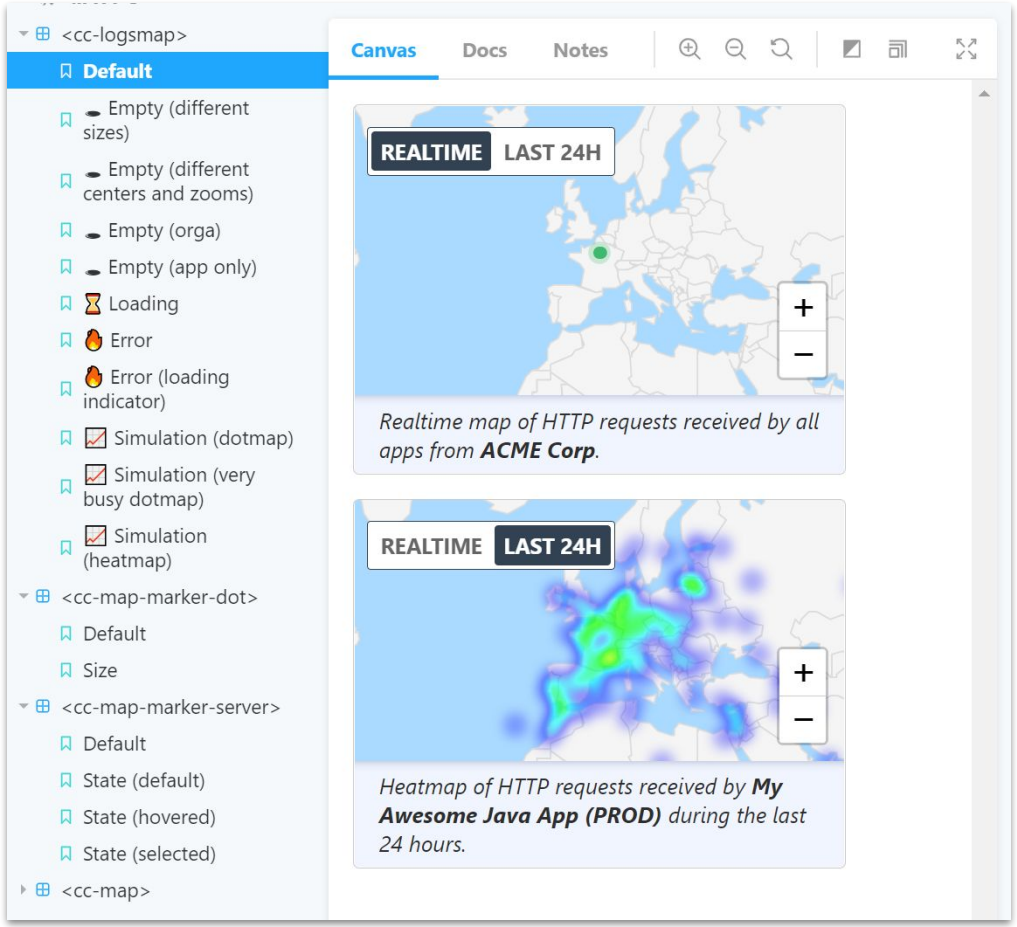




Capitalize on your atoms

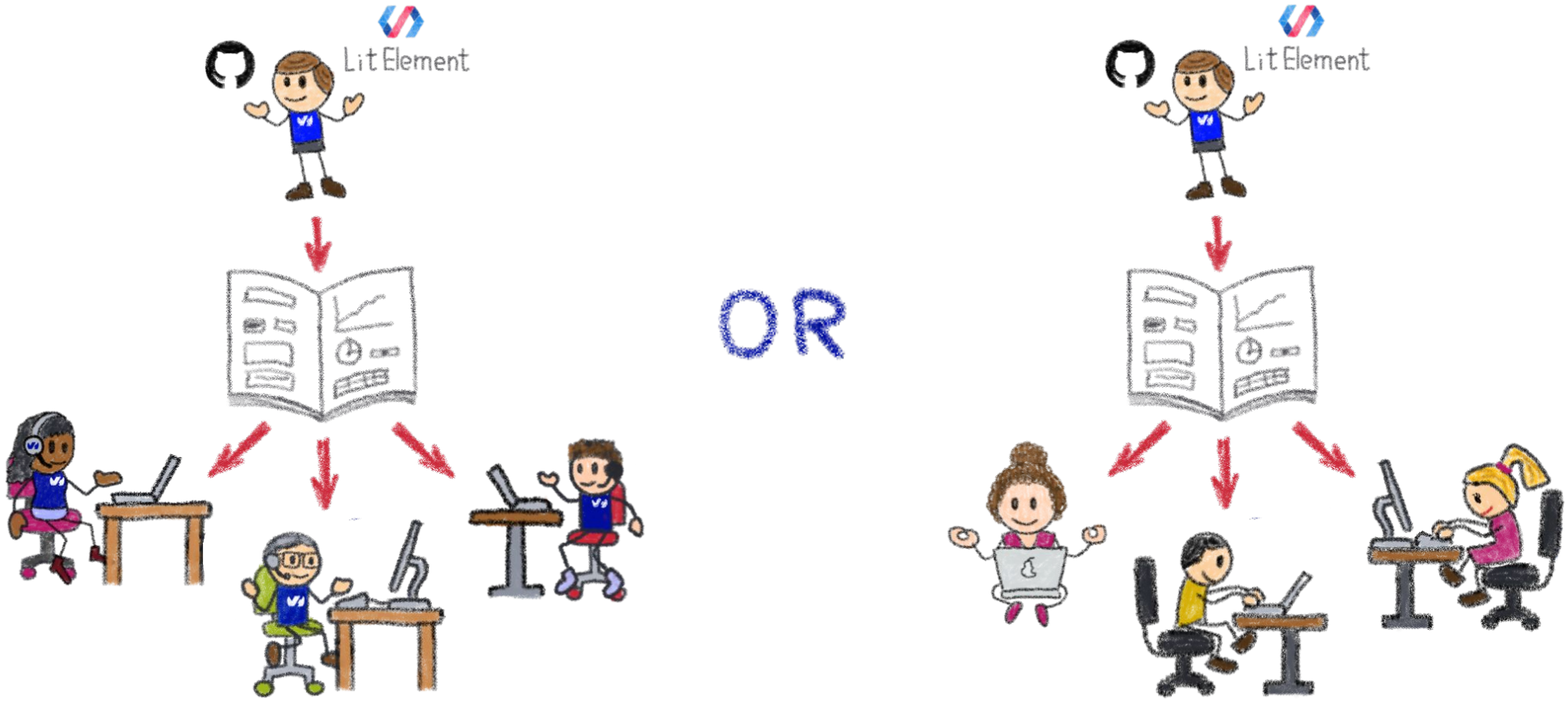
Keep the flexibility and configurability

Big smart business components



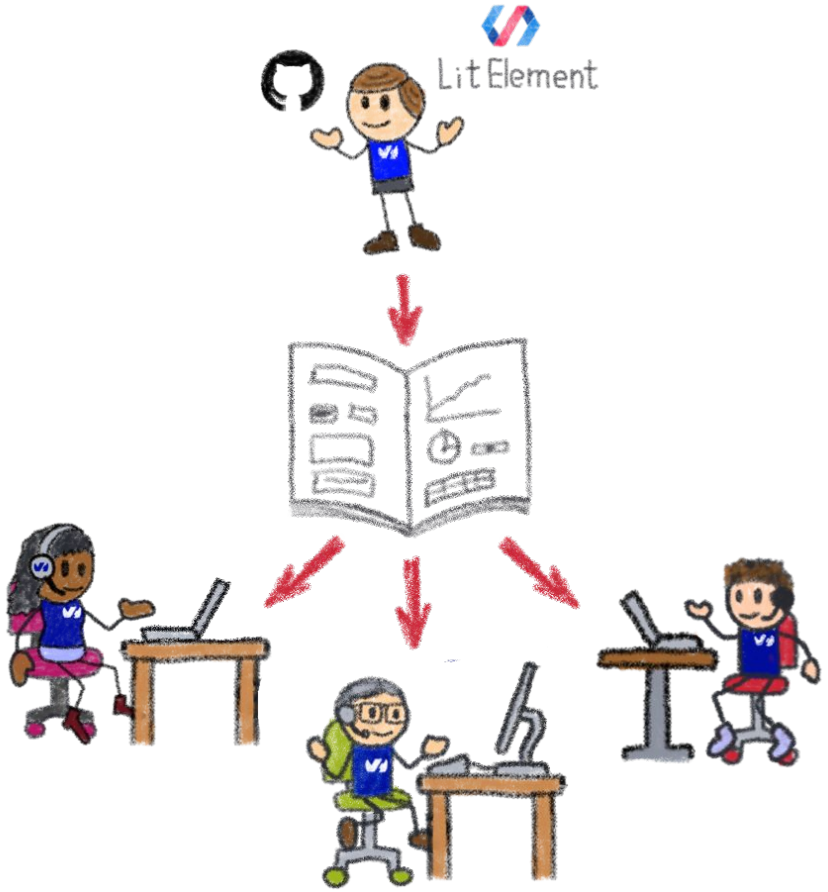
Encoding your business logic

Internal or external customers?



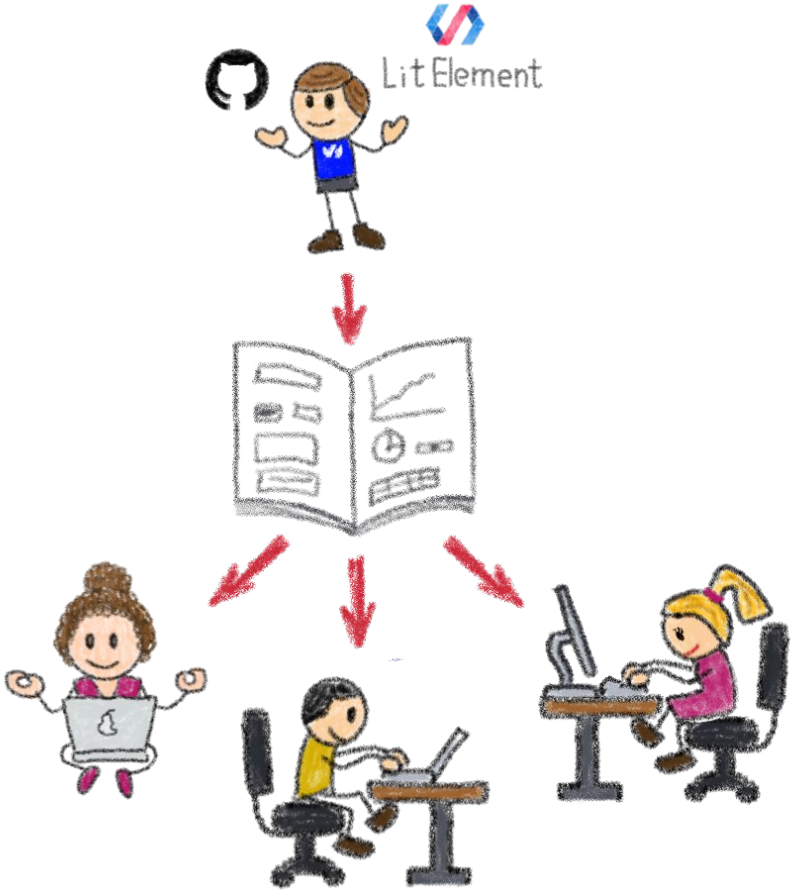
Who are your target users?

Internal customers need off-the-shelf components



A well defined and coherent look-and-feel

External customers need to be able to tweak

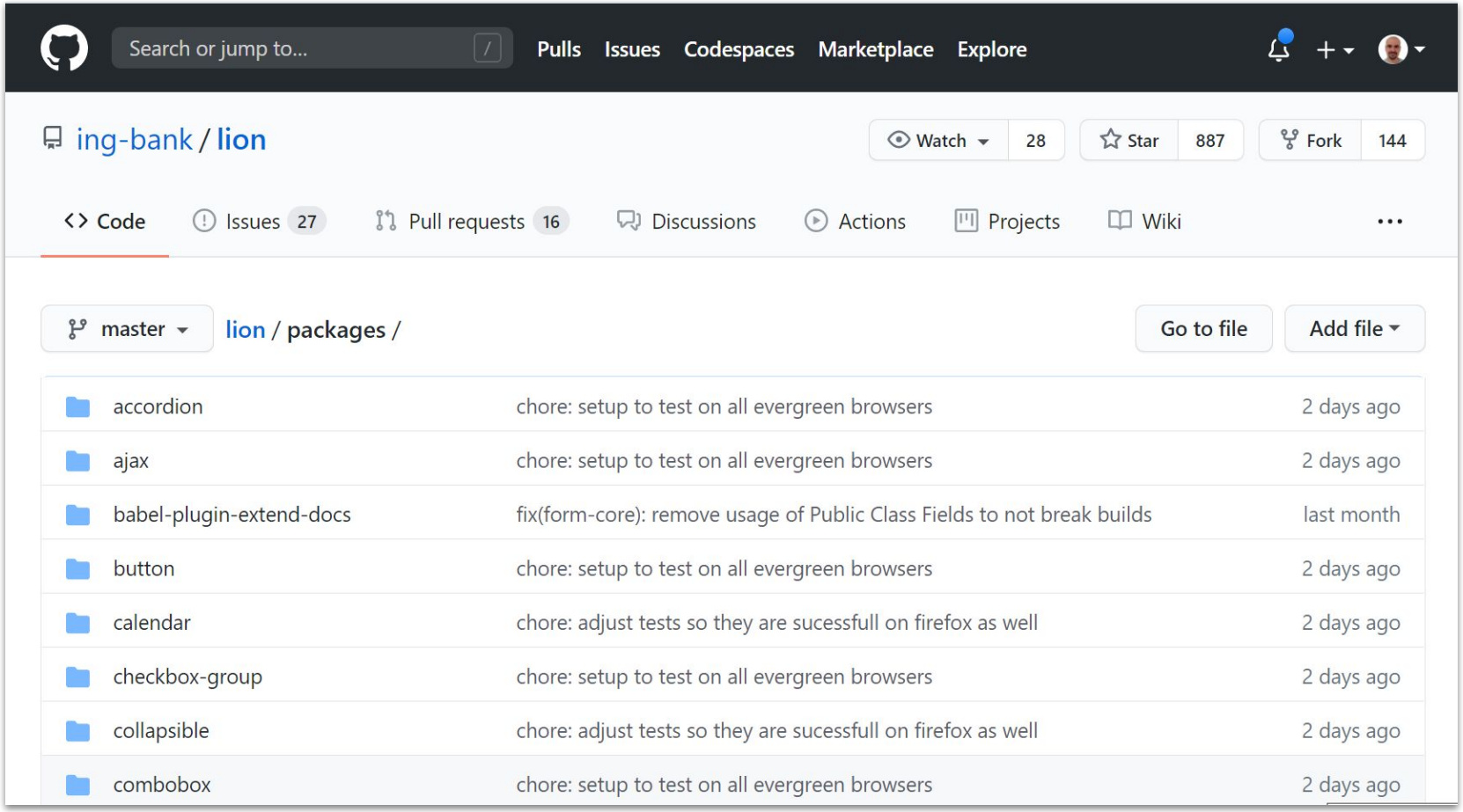


Theming and customizing components

How to organize the catalog

Packages, imports and pragmatism

A single repository



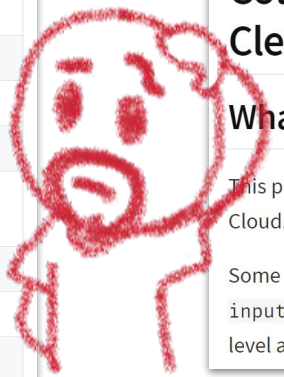
Single source of truth for the catalog

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 v0.10.1	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 v0.17.0	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

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 v0.17.0	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

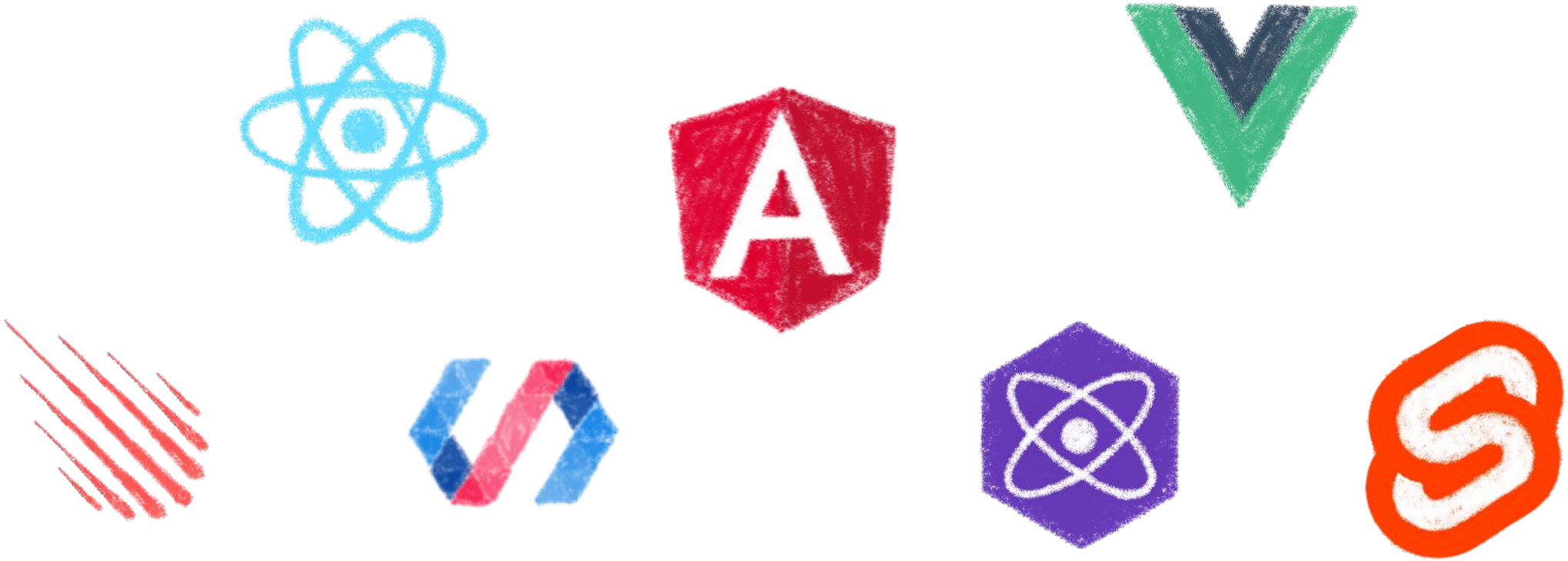
Unpacked Size	Total Files
1.31 MB	145

Individual versioning vs global one

Driving-up adoption

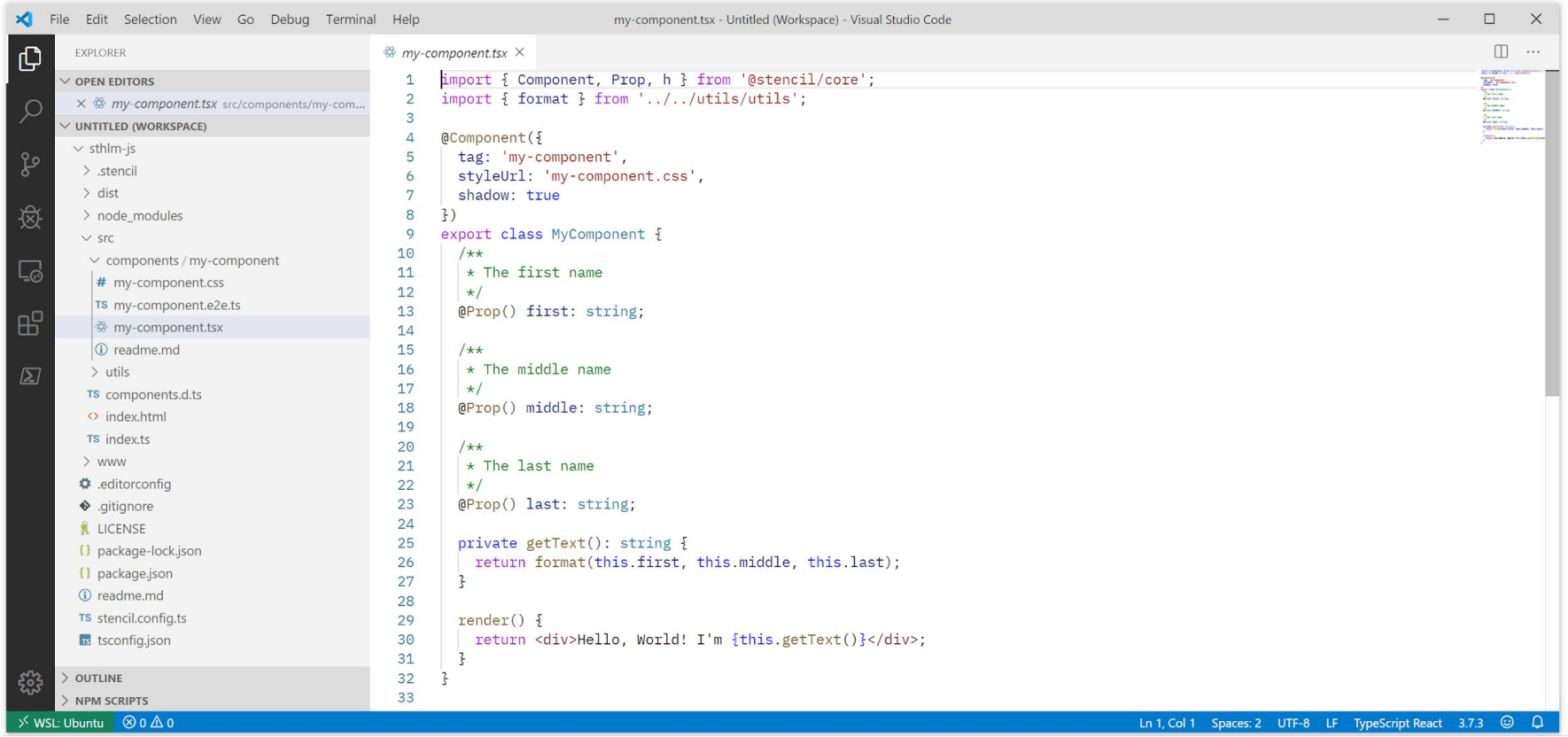
Making devs use your components

Think who are your target users



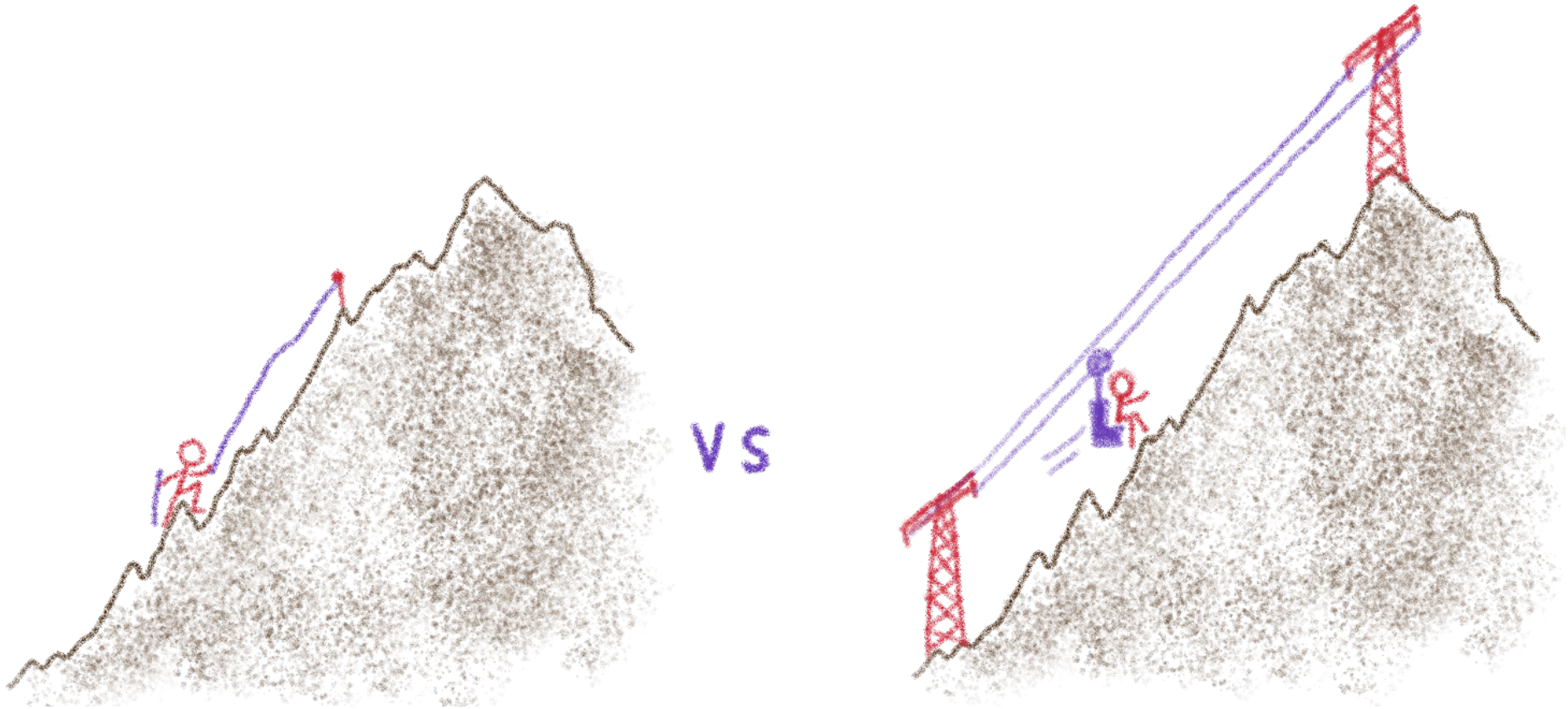
Users of any framework current or future...

They aren't used to your library



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

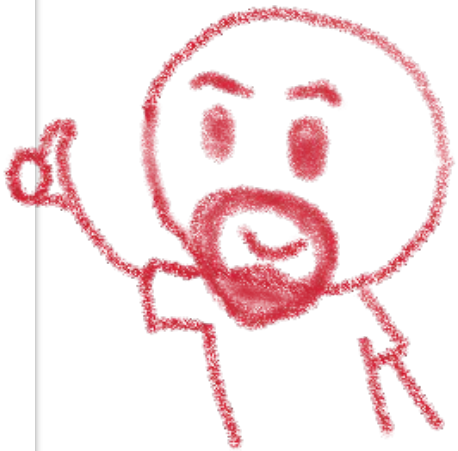
How to install

```
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

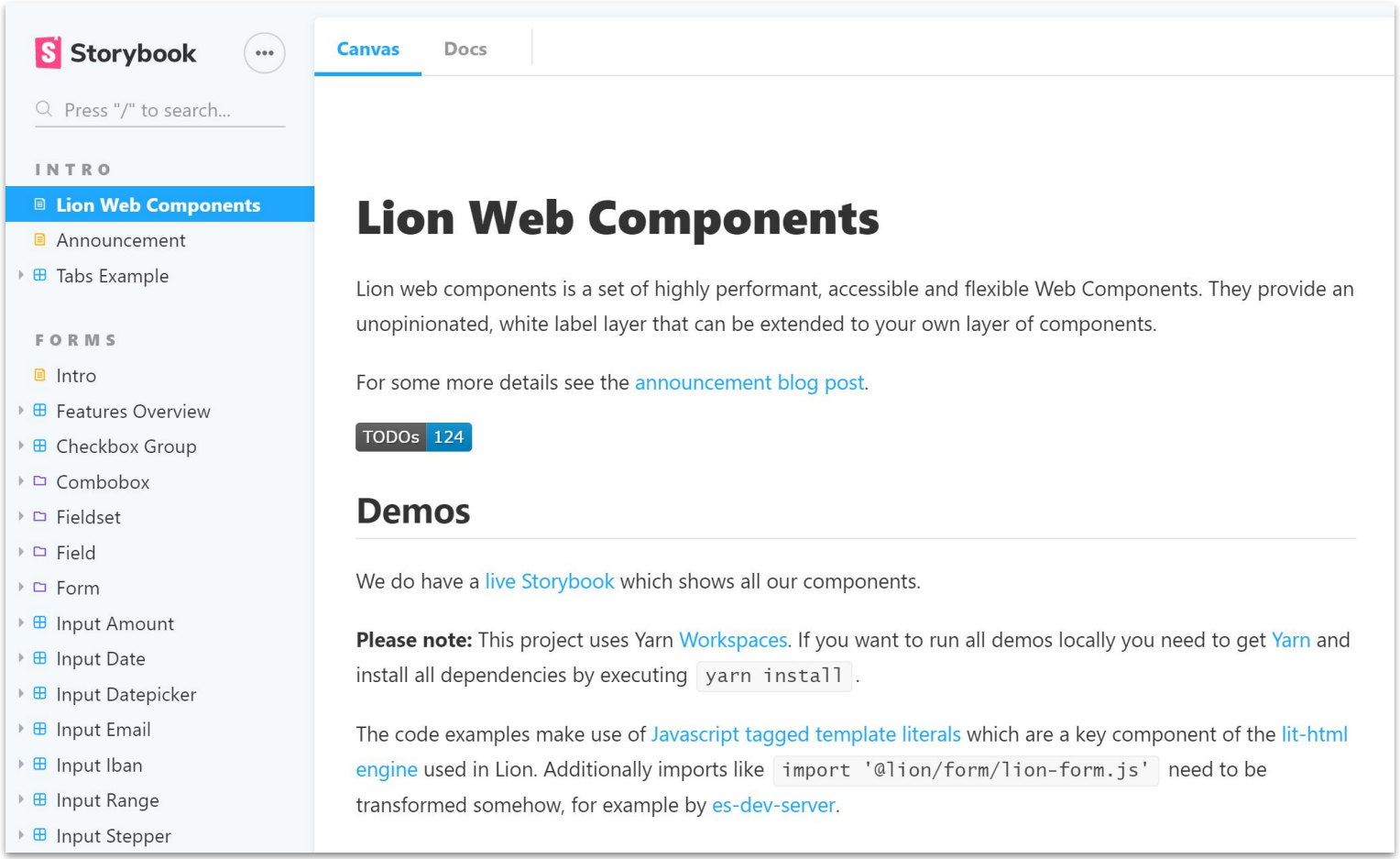
`lion-input-iban` component is based on the generic text input field. Its purpose is to provide a way for users to fill in an IBAN (International Bank Account Number).

```
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> `;
};
```

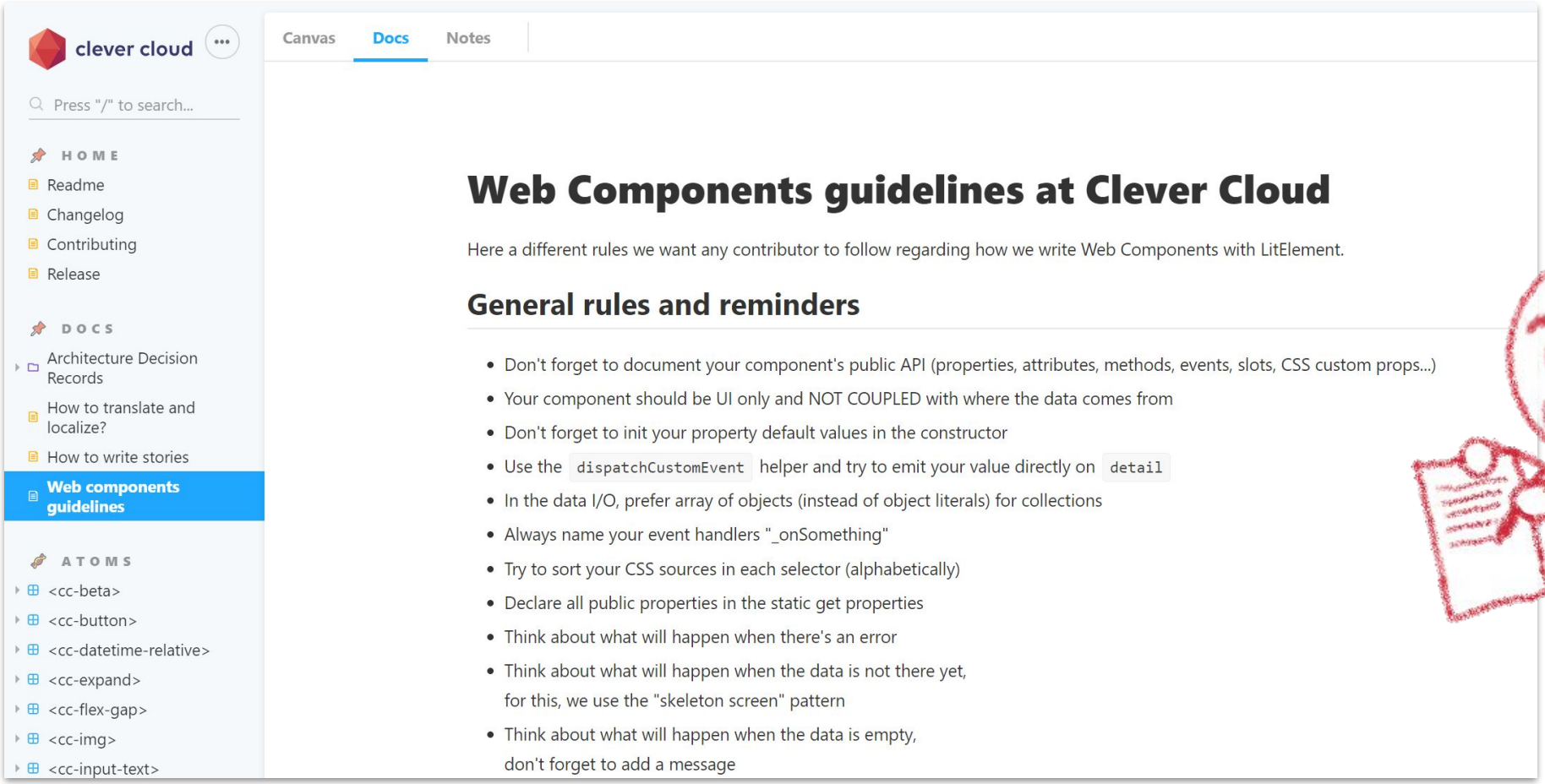
How to use, inputs/outputs, examples...

Documentation isn't enough



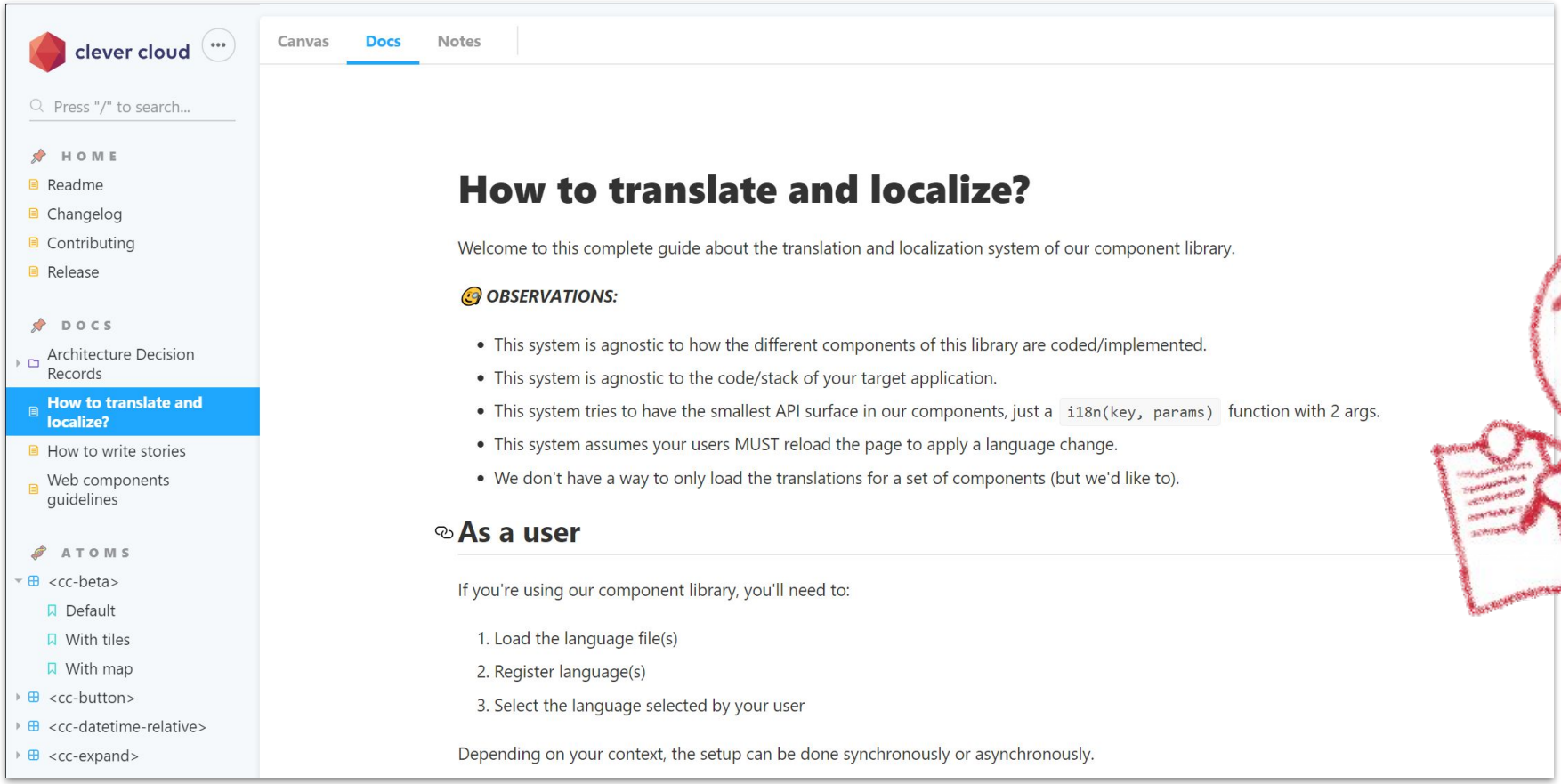
Storybook make adoption easy

Keeping a coherent writing style



Write down your guidelines

I18n shouldn't be an afterthought



Prepare everything for internationalization

That's all, folks!

Thank you all!