

ROME - APRIL 13/14 2018

{codemotion}

{ The Web Components Interoperability Challenge }

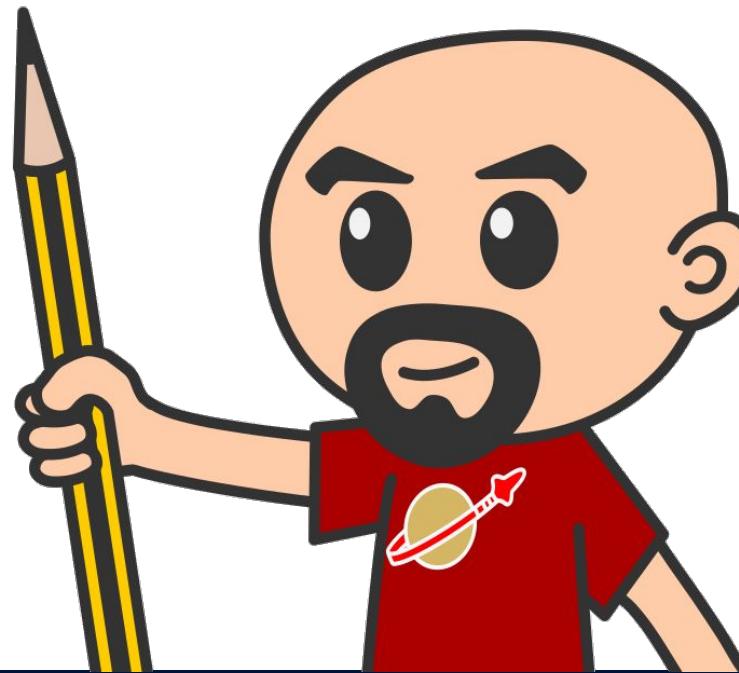
Horacio Gonzalez (@LostInBrittany)



Horacio Gonzalez

@LostInBrittany

Spaniard lost in Brittany, developer, dreamer
and all-around geek



We want the code!

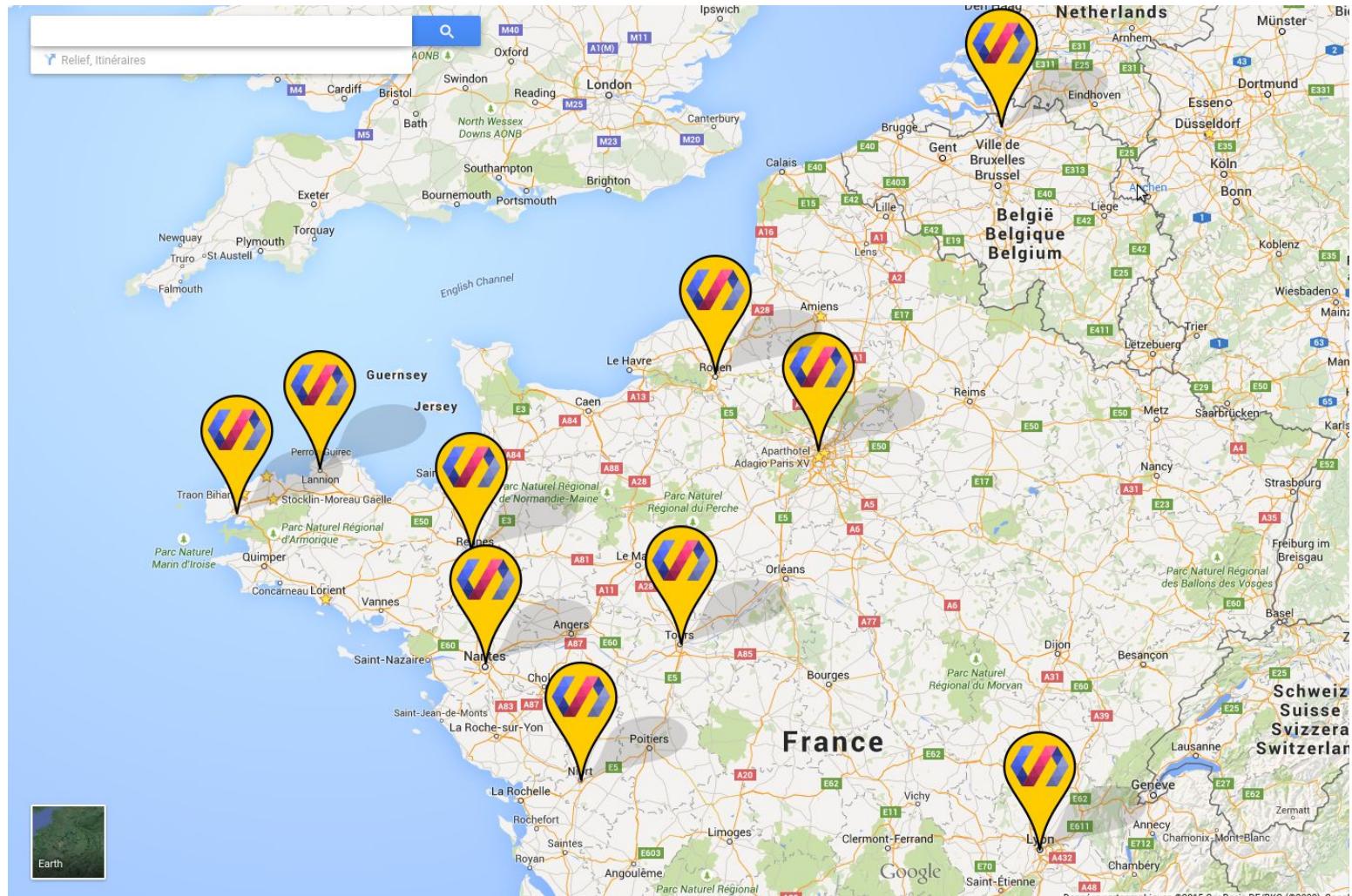
The screenshot shows a GitHub repository page for 'a-world-outside-polymer'. The repository is private and has 3 commits, 1 branch, 0 releases, and 1 contributor. The latest commit was made 8 hours ago. The repository contains files like 'Slim.js', 'step-01', 'step-02', 'step-03', 'step-04', 'step-05', and 'README.md'. The README file contains the text 'a-world-outside-polymer'.

File	Commit Message	Time Ago
Slim.js	Added Slim.js	Latest commit 7e4458a 8 hours ago
step-01	Initial commit	11 hours ago
step-02	Initial commit	11 hours ago
step-03	Initial commit	11 hours ago
step-04	Added Slim.js	8 hours ago
step-05	Added Slim.js	8 hours ago
README.md	first commit	11 hours ago

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



Polymer tour 2014-2018



Web components == Revolution



bu.edu



Build a world brick by brick



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



Variations of the same questions



ideas.lego.com

Does it work in real life?



Web Components



A very basic web component

```
class MyElement extends HTMLElement {  
  
    // This gets called when the HTML parser sees your tag  
    constructor() {  
        super(); // always call super() first in the ctor.  
        this.msg = 'Hello, Codemotion!';  
    }  
  
    // 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('my-element', MyElement);
```



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 MyElementWithShadowDom extends HTMLElement {  
  
    // This gets called when the HTML parser sees your tag  
    constructor() {  
        super(); // always call super() first in the ctor.  
        this.msg = 'Hello from inside the ShadowDOM, Codemotion!';  
        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('my-element-with-shadowdom', MyElementWithShadowDom);
```



Adding ShadowDOM

The screenshot shows a browser window titled "Testing basic web components" with the URL "localhost:8000/step-01/". The page content displays two pieces of text: "Hello, Codemotion!" and "Hello from inside the ShadowDOM, Codemotion!". The browser's developer tools are open, specifically the Elements tab, which shows the DOM structure. The root node is an `<html>` element. Inside it is a `<body>` element. The `<body>` contains a `<my-element>` element, which in turn contains a `<p>` element with the text "Hello, Codemotion!". Below the `<my-element>` is another element, `<my-element-with-shadowdom>`. This element has its own `#shadow-root` (open), which contains a `<p>` element with the text "Hello from inside the ShadowDOM, Codemotion!". Below this shadow-root is a `<script>` element with the source "my-element.js" and another `<script>` element with the source "my-element-with-shadowdom.js". The CSS panel at the bottom shows the styles for the `body` element, including `display: block;` and `margin: 8px;`. The right side of the developer tools interface shows a detailed view of the element's bounding box, with dimensions of 367.200 x 52.800 and margins of 8px on all sides.

```
<!DOCTYPE html>
<html lang="en">
  >#shadow-root (open)
  ><head>...</head>
  ...<body> == $0
    ><my-element>
      <p>Hello, Codemotion!</p>
    </my-element>
    ><my-element-with-shadowdom>
      >#shadow-root (open)
        > <p>Hello from inside the ShadowDOM, Codemotion!</p>
    </my-element-with-shadowdom>
    <script src="my-element.js"></script>
    <script src="my-element-with-shadowdom.js"></script>
  </body>
</html>
```

html	body	my-element-with-shadowdom	#shadow-root	p
------	------	---------------------------	--------------	---

Styles Event Listeners DOM Breakpoints Properties Accessibility

Filter :hov .cls +

```
element.style { }
body {
  display: block;
  margin: 8px;
}
```

user agent stylesheet

margin	8
border	-
padding	-
367.200 × 52.800	
-	-



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.  
    }  
    // Tells the element which attributes to observe for changes  
    // This is a feature added by Custom Elements  
    static get observedAttributes() {  
        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  
    }  
}
```



A simple click counter

As a *vanilla* Web Component



my-counter custom element

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



my-counter custom element

```
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;  
}
```



my-counter custom element

```
render() {
  let button = document.createElement('button');
  button.innerHTML = '+';
  button.addEventListener('click', this.increment.bind(this));
  this.shadowRoot.appendChild(button);

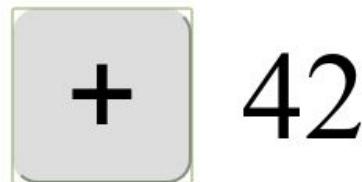
  this.output = document.createElement('span');
  this.shadowRoot.appendChild(this.output);

  this.style.display = 'block';
  this.style.fontSize = '5rem';
  button.style.fontSize = '5rem';
  button.style.borderRadius = '1rem';
  button.style.padding = '0.5rem 2rem';
  this.output.style.marginLeft = '2rem';
}

display() {
  this.output.innerHTML = `${this.counter}`;
}
```



my-counter custom element



Polymer

**Adding syntactic sugar
to the standard**



Everything is better with sugar

```
<link rel="import" href="./bower_components/polymer/polymer.html">

<dom-module id="my-polymer-counter">
  <template>
    <style>
      :host {
        font-size: 5rem;
      }
      button {
        font-size: 5rem;
        border-radius: 1rem;
        padding: 0.5rem 2rem;
      }
    </style>
    <button on-click="increment">+</button>
    <span>[[counter]]</span>
  </template>
```



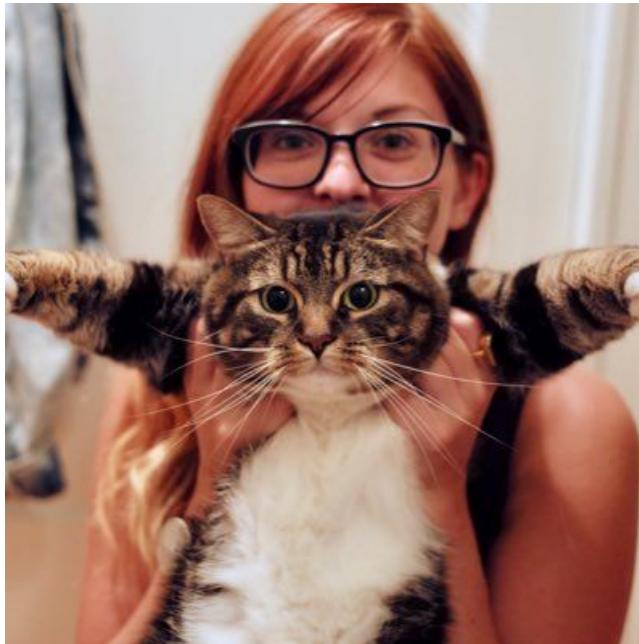
Everything is better with {codemotion} sugar

```
<script>
  class MyPolymerCounter extends Polymer.Element {
    static get is() { return 'my-polymer-counter'; }
    static get properties() {
      return {
        counter: { type: Number, reflectToAttribute:true, value: 0 }
      }
    }
    increment() {
      this.counter = Number.parseInt(this.counter) + 1;
    }
  }

  customElements.define('my-polymer-counter', MyPolymerCounter);
</script>
</dom-module>
```



Everything is better with sugar



Polymer is like jQuery for Web components
@notwaldorf



But they are still custom elements



5



5

Shared value: 5

100% interoperables



Interoperation pattern

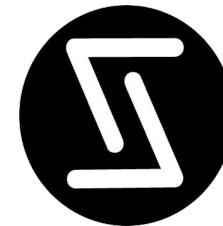
```
<div class="container">
  <my-polymer-counter
    counter="[[value]]"
    on-counter-changed="_onCounterChanged"></my-polymer-counter>
  <my-counter
    counter="[[value]]"
    on-counter-changed="_onCounterChanged"></my-counter>
</div>
```

Attributes for data in
Events for data out

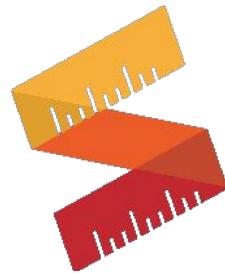


To infinity and beyond!

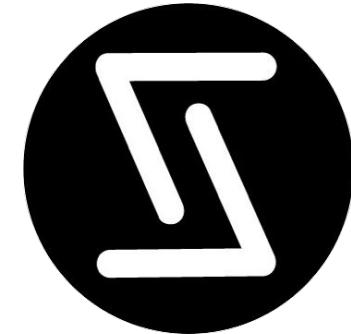
There is a world outside Polymer



Lots of web components libraries



slim.js



STENCIL



For different need and sensibilities



Lots of web components libraries



Angular Elements

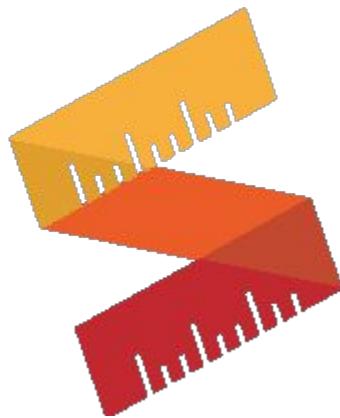


Vue Web Component
Wrapper

And the frameworks work on it too!



Slim.js



Slim.js

[Getting started](#)[Creating an element](#)[Lifecycle](#)[Accessing children](#)[Conditional rendering](#)[Shadow-DOM](#)[Data binding](#)[Repeating elements](#)[Plugins](#)[Extending Slim & Directives](#)

Getting started with Slim.js



Slim.js is a lightweight web component authoring library that provides extended capabilities for components (such as data binding) using es6 native class inheritance. This library is focused on providing developers the ability to write robust web components quickly and without the hassle of unnecessary dependencies or an overhead of a framework.

Installation

```
npm install slim-js  
yarn add slim-js  
bower install slimjs
```

Web-Components Spec. & Polyfills

Slim.js is based on the browser's native DOM API's web-components spec. For browsers that do not support this natively, a polyfill is required.

You can load the polyfill directly from the html file

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/webcomponentsjs/1.0.17/webcomponents-lite.js"></script>
```

[VIEW ON GITHUB](#)

Slim.js

- Lightweight web component library
- Extended capabilities for components
 - data binding
- Using es6 native class inheritance
- Optional Shadow DOM

Like a lighter and lesser-featured Polymer

Fast & Robust Front-End Micro-framework based on modern standards



Slim.js

```
Slim.tag('my-slim-counter', `

<style> [...] </style>
<div class="container">
  <div class="button" s:id="button">  </div>
  <div class="value" bind> {{counter}} </div>
</div>`,
  class extends Slim {
    onCreated() {
      if (this.counter == undefined) {
        this.counter = Number.parseInt(this.getAttribute('counter'))||'0';
      }
      this.button.onclick = () => {
        this.counter++;
        this.dispatchEvent(new CustomEvent('counter-changed', {detail: {counter: this.counter}}));
      }
    }
    static get observedAttributes() { return [ 'counter' ] }

    attributeChangedCallback(attr, oldVal, newVal) {
      if (oldVal !== newVal) { this[attr] = newVal; }
    }
  });
});
```



Bram.js



Bram.js



Bram

A simple 3kB web components library

Home API Guides GitHub

Install the latest:

```
npm install bram --save
```

Or download a [release](#).

Examples

Todo app

HTML JavaScript

```
<template id="todo-template">
  <form on-submit="addTodo">
    <input type="text" name="todo"
      placeholder="What to do?">
    <button type="submit">Add</button>
  </form>

  <ul>
    <template each="{{todos}}">
      <li>{{item}}</li>
    </template>
  </ul>
</template>

<todo-list></todo-list>
```

What to do? Add



Bram.js

- Lightweight web component library
- Extended capabilities for components
 - data binding
- Using es6 native class inheritance
- With optional Shadow DOM

Like a lighter and lesser-featured Polymer

A simple 4kB web components library



Bram.js

```
let template=`
<style> [...] </style>
<div class="container">
  <div class="button" on-click="increase">  </div>
  <div class="value" > {{counter}} </div>
</div>`;

class MyBramCounter extends Bram(HTMLElement) {
  static get template() {
    let t = document.createElement('template');
    t.innerHTML = template;
    return t;
  }
  static get events() { return ['counter-changed']; }
  constructor() {
    super();
    this.model.counter = this.getAttribute('counter') || 0;
  }

  static get observedAttributes() { return [ 'counter' ] }

  attributeChangedCallback(attr, oldVal, newVal) {
    if (oldVal !== newVal) { this[attr] = newVal; }
  }

  increase() {
    this.model.counter++;
    this.dispatchEvent(new CustomEvent('counter-changed', {detail: {counter: this.model.counter}}));
  }
}
```



Skatejs



Skatejs



SkateJS

Effortless custom elements for modern view libraries.

Code HTML Result

```
// @jsx h

import { props, withComponent } from 'skatejs';
import withPreact from '@skatejs/renderer-preact';
import { h } from 'preact';

class WithPreact extends withComponent(withPreact()) {
  static get props() {
    return {
      name: props.string
    };
  }
  render({ name }) {
    return <span>Hello, {name}!</span>;
  }
}
```



Skatejs

- Lightweight web component library
- Abstracts away attribute / property semantics
- Very very fast
- Can use many renderers
 - Basic innerHTML (default)
 - preact
 - lit-html

Nice if you dislike declarative syntax and DOM...



Skatejs

```
import { props, withComponent } from '/node_modules/skatejs/dist/esnext/index.js';

class MySkateCounter extends withComponent() {

    constructor() {
        super();
        this.counter = this.counter || 0;
        this.addEventListener('click', e => this.increment());
    }

    static get props() {
        return {
            // By declaring the property an attribute, we can now pass an initial value for the count as part of the HTML.
            counter: props.string({ attribute: true })
        };
    }

    render({ counter }) {
        return `${this.style()}`
        <div class="container">
            <div class="button"></div>
            <div class="value">${counter}</div>
        </div>;
    }

    increment() {
        this.counter = Number.parseInt(this.counter) + 1;
        this.dispatchEvent(new CustomEvent('counter-changed', {detail: {counter: this.counter}}));
    }

    style() { return `...` };
}
}
```





A new breed of Web Components



Next generation Ionic



Ionic 4 will be fully based on web components
using a new toolkit: Stencil



New kid on the block

Up next AUTOPLAY

- Stencil | Getting Started**
Academind 6.8K views
- Intro to Web Components with StencilJS**
Madness Labs 701 views
- Faster Web Apps Using Stencil**
Paul Halliday 3.3K views
- What You See is What You Deserve: Simple Visual Tools**
Google Chrome Developers 12K views
- Polymer Summit 2017**
Google Chrome Developers
- Practical lessons from a year of building web components -**
Google Chrome Developers 47K views
- ES6 Modules in the Real World (Polymer Summit 2017)**
Google Chrome Developers 8.4K views

Announced during Polymer Summit



Not another library



The magical, reusable web component compiler



Simple

With intentionally small tooling, a tiny API, zero configuration, and TypeScript support, you're set.



Performant

6kb min+gzip runtime, server side rendering, and the raw power of native Web Components.



Future proof

Build versatile apps and components based 100% on web standards. Break free of Framework Churn.

A Web Component compiler



A build time tool



To generate standard web components



Fully featured

- Virtual DOM
- Async rendering
- Reactive data-binding
- TypeScript
- JSX



And the cherry on the cake

SSR

Server-Side Rendering



Hands on Stencil

Clone the starter project

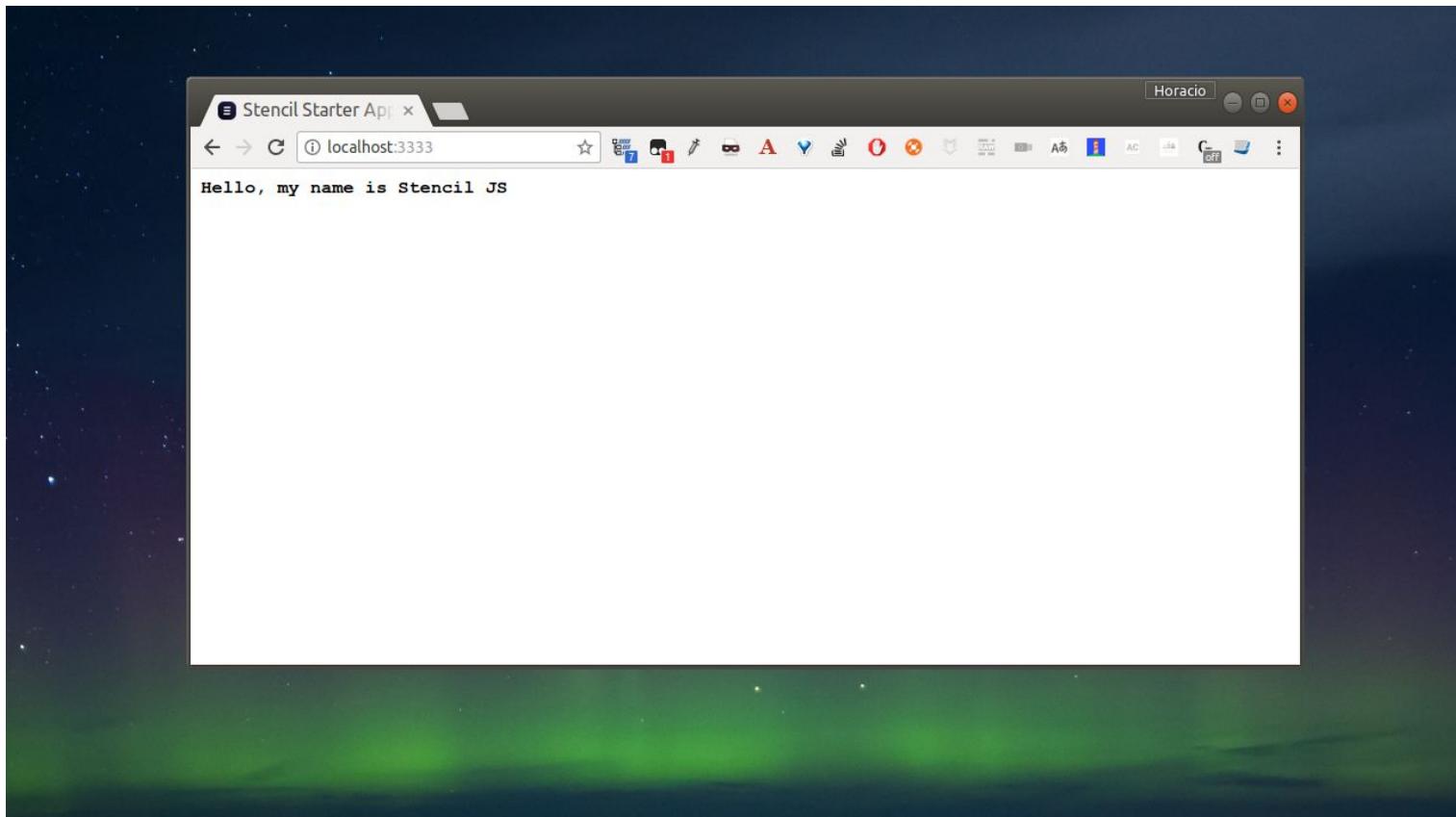
```
git clone https://github.com/ionic-team/stencil-app-starter my-app  
cd my-app  
git remote rm origin  
npm install
```

Start a live-reload server

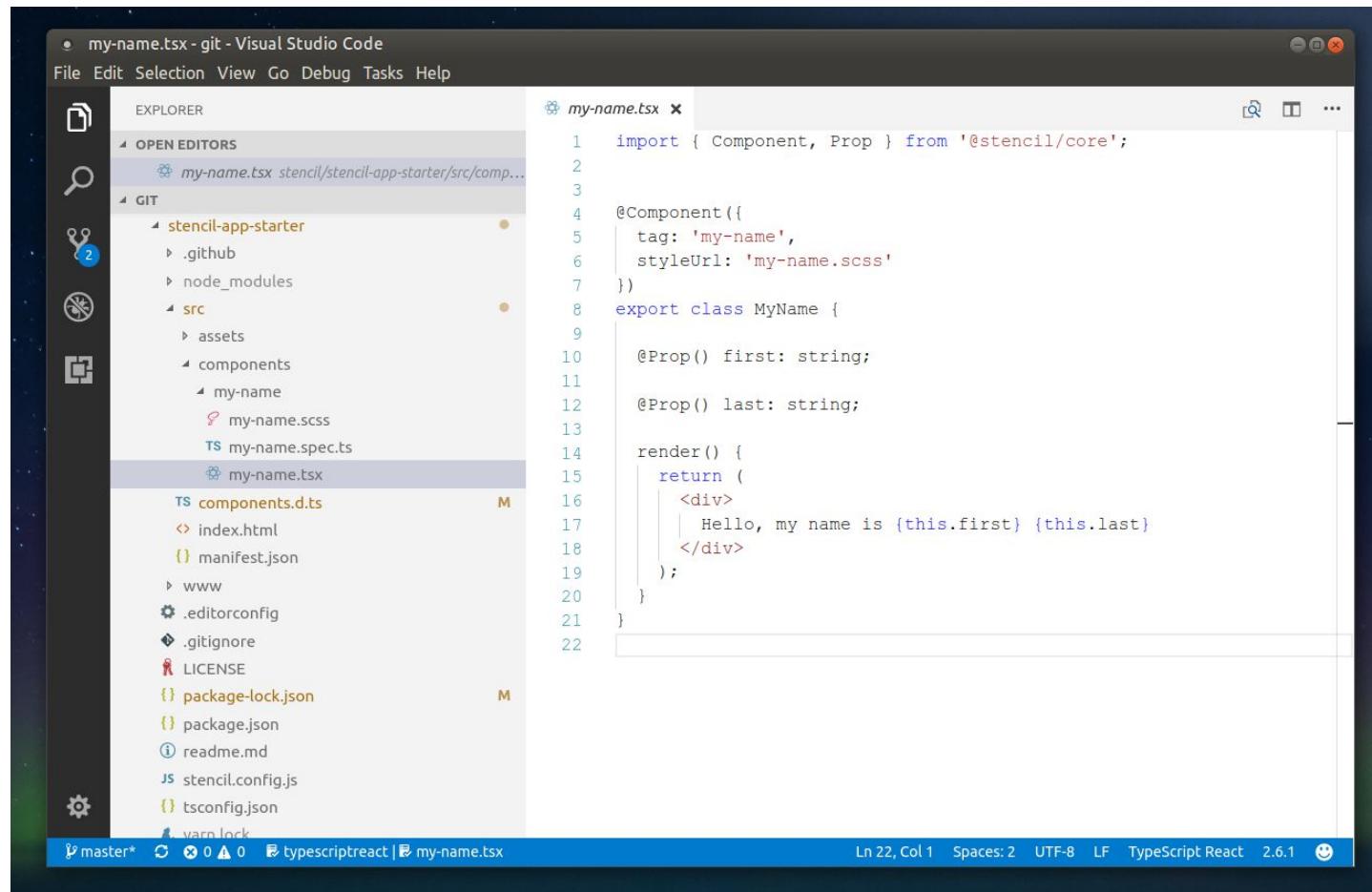
```
npm start
```



Hands on Stencil



Hands on Stencil



The screenshot shows a Visual Studio Code interface with the following details:

- Title Bar:** my-name.tsx - git - Visual Studio Code
- File Menu:** File Edit Selection View Go Debug Tasks Help
- Explorer View:** Shows the project structure:
 - OPEN EDITORS: my-name.tsx
 - GIT: stencil-app-starter
 - .github
 - node_modules
 - src
 - assets
 - components
 - my-name
 - my-name.scss
 - my-name.spec.ts
 - my-name.tsx
 - Components.d.ts
 - index.html
 - manifest.json
 - www
 - .editorconfig
 - .gitignore
 - LICENSE
 - package-lock.json
 - package.json
 - readme.md
 - stencil.config.js
 - tsconfig.json
 - var.lock
- Code Editor:** my-name.tsx (active tab)

```
1 import { Component, Prop } from '@stencil/core';
2
3
4 @Component({
5   tag: 'my-name',
6   styleUrl: 'my-name.scss'
7 })
8 export class MyName {
9
10   @Prop() first: string;
11
12   @Prop() last: string;
13
14   render() {
15     return (
16       <div>
17         | Hello, my name is {this.first} {this.last}
18       </div>
19     );
20   }
21 }
22 }
```
- Status Bar:** master* 0 0 0 typescriptreact | my-name.tsx Ln 22, Col 1 Spaces: 2 UTF-8 LF TypeScript React 2.6.1



Some concepts

```
render() {  
  return (  
    <div>Hello {this.name}</div>  
  )  
}
```

```
render() {  
  return (  
    <div>{this.name ? <p>Hello {this.name}</p> : <p>Hello World</p>}</div>  
  );  
}
```

JSX declarative template syntax



Some concepts

```
import { Component } from '@stencil/core';

@Component({
  tag: 'todo-list',
  styleUrl: 'todo-list.scss'
})
export class TodoList {
  @Prop() color: string;
  @Prop() favoriteNumber: number;
  @Prop() isSelected: boolean;
  @Prop() myHttpService: MyHttpService;
}
```

Decorators



Some concepts

```
import { Event, EventEmitter } from '@stencil/core';

...
export class TodoList {

  @Event() todoCompleted: EventEmitter;

  someAction(todo: Todo) {
    this.todoCompleted.emit(todo);
  }

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

Events



Some concepts

```
@Component({  
  tag: 'shadow-component',  
  styleUrl: 'shadow-component.scss',  
  shadow: true  
})  
export class ShadowComponent {  
}
```

Optional Shadow DOM



Some concepts

stencil.config.js

```
exports.config = {  
  namespace: 'myname',  
  generateDistribution: true,  
  generateWWW: false,  
  ...  
};
```

Generate distribution



Stencil

```
import { Component, Prop, PropWillChange, State, Event, EventEmitter } from '@stencil/core';

@Component({
  tag: 'stencil-counter',
  styleUrl: 'stencil-counter.scss',
  shadow: true
})
export class StencilCounter {
  @Prop() counter: number;
  @State() currentCount: number;
  @Event() currentCountChanged: EventEmitter;

  @Watch('counter')
  counterChanged(newValue: number) {
    this.currentCount = newValue;
  }

  componentDidLoad() {
    this.currentCount = this.counter;
  }

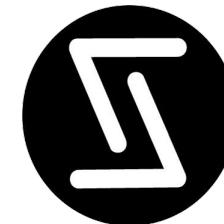
  increase() {
    this.currentCount++;
    this.currentCountChanged.emit({ counter: this.currentCount });
  }

  render() {
    return (
      <div class="container">
        <div class="button" onClick={() => this.increase()}>  </div>
        <div class="value" > {this.currentCount} </div>
      </div>
    );
  }
}
```



Working together

They are all web components



Inside a simple Vue app

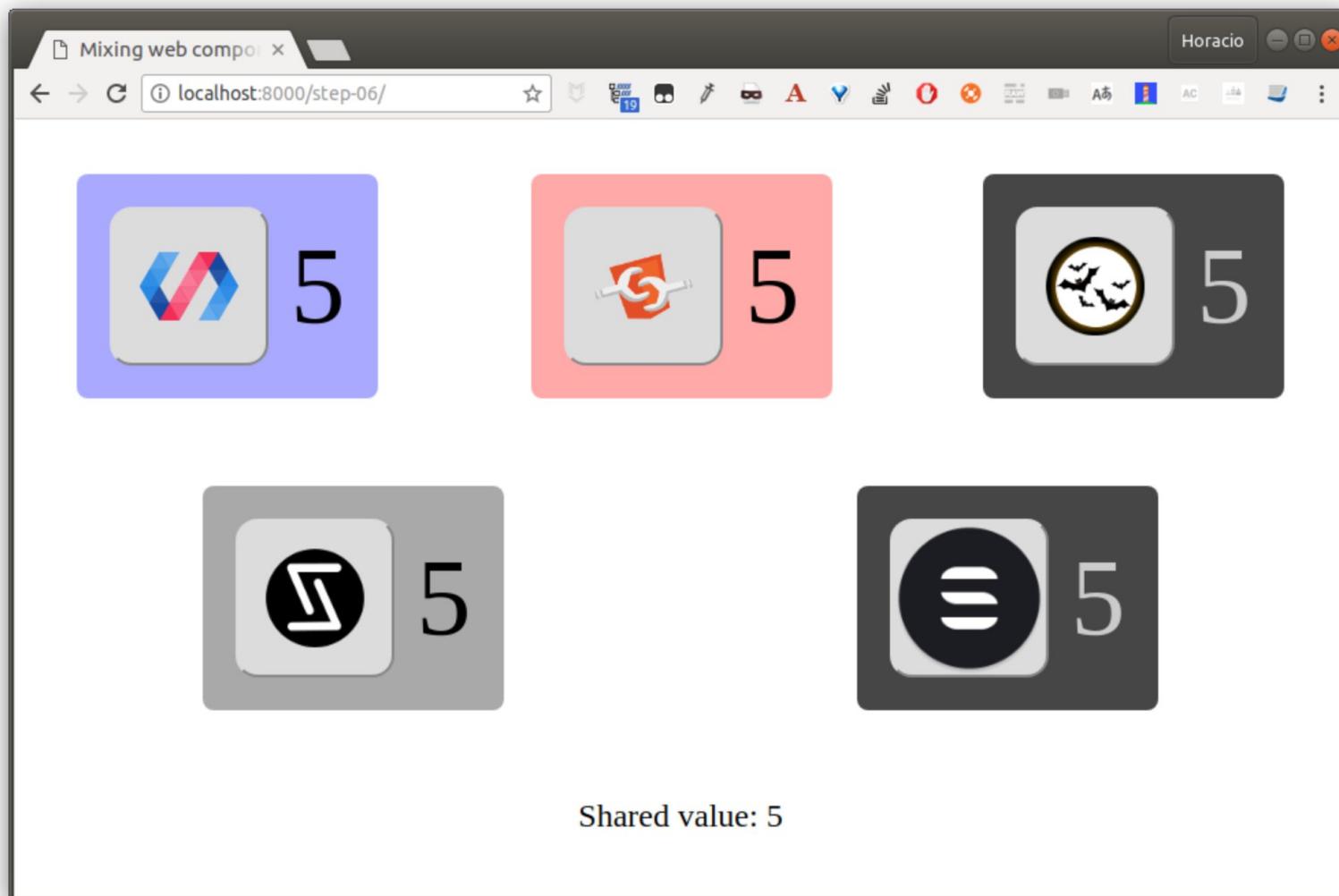


Using the standard pattern

Attributes (or properties) in
Events out



All of them working together



Conclusion

That's all folks!



Thank you!

