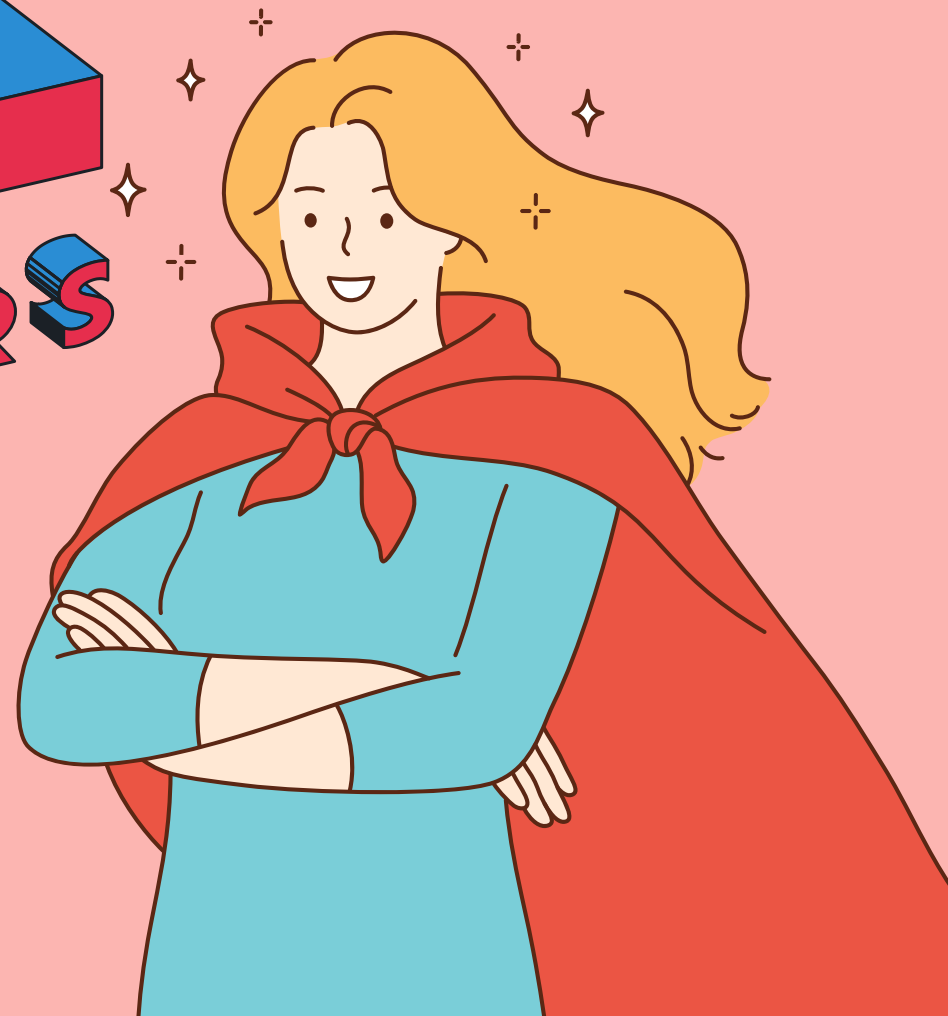


HTML

WITH

SUPERPOWERS



Howdy, I'm Dave Rupert

Work at Paravel in Austin, TX

Co-host ShopTalk podcast with Chris Coyier

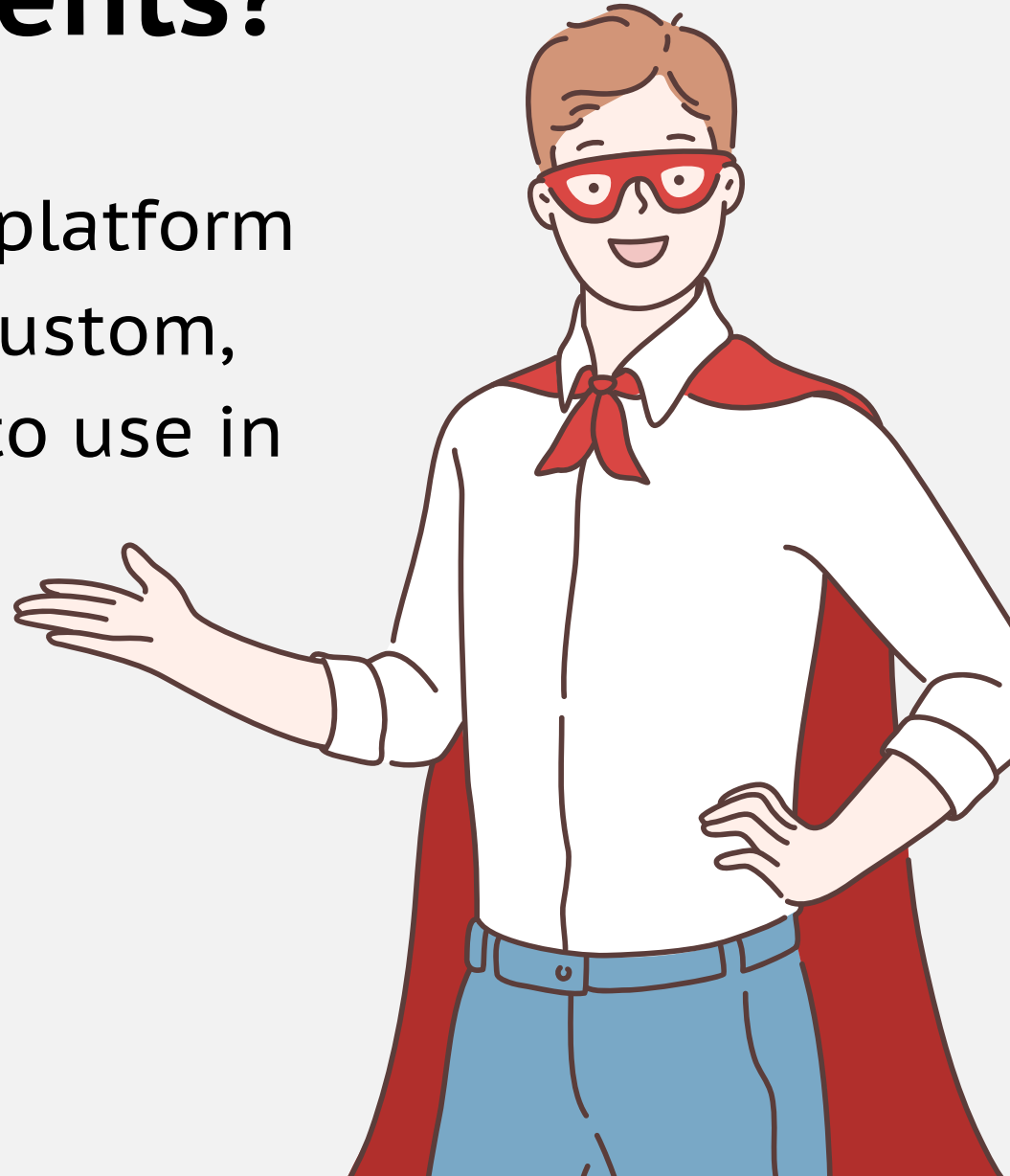
W3C Web Components Community Group (member)



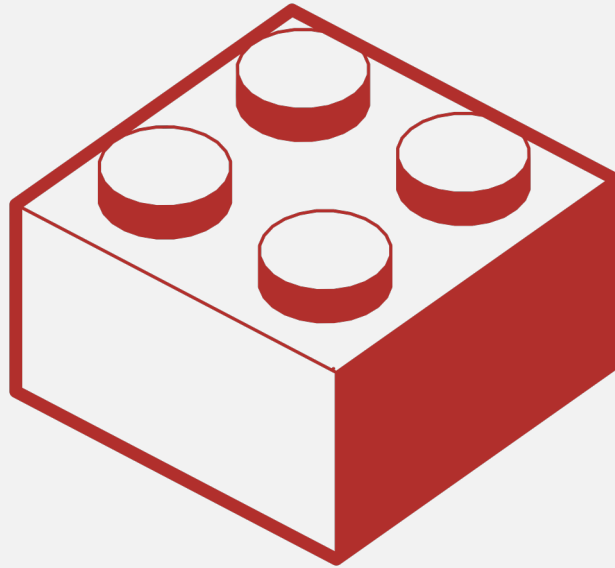
What are Web Components?

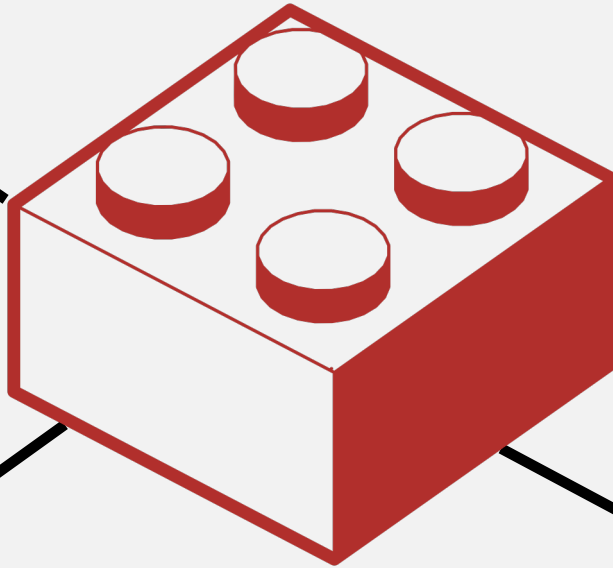
“Web components are a set of web platform APIs that allow you to create new custom, reusable, **encapsulated** HTML tags to use in web pages and web apps.”

—webcomponents.org

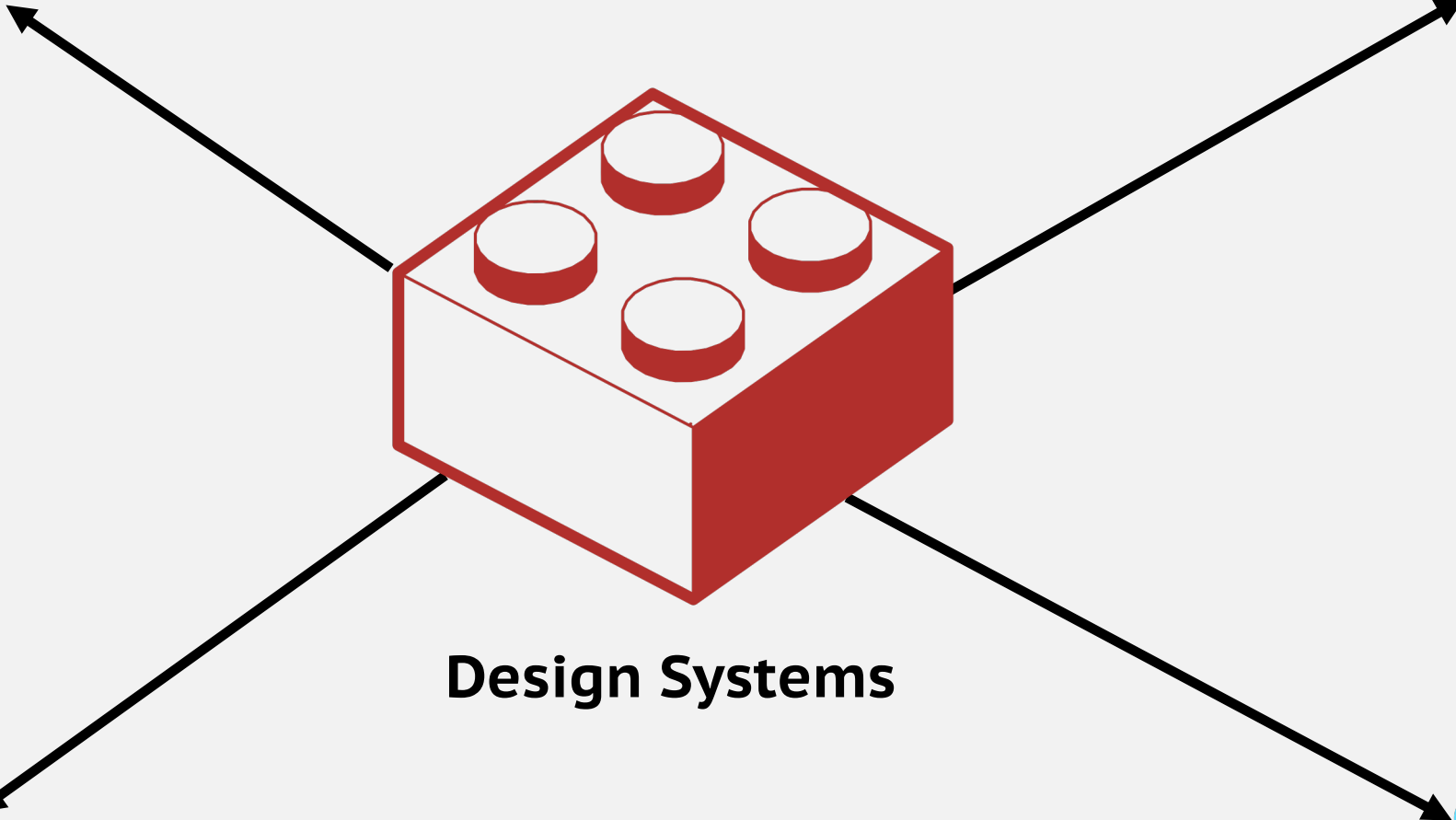
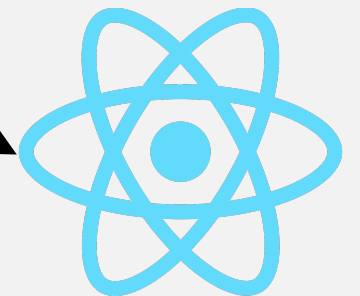


What are Web Components?

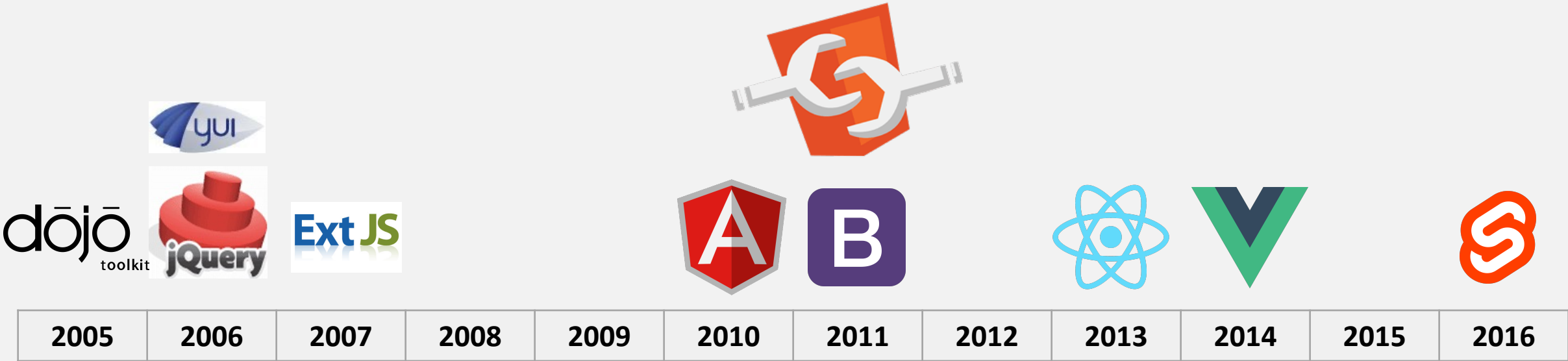




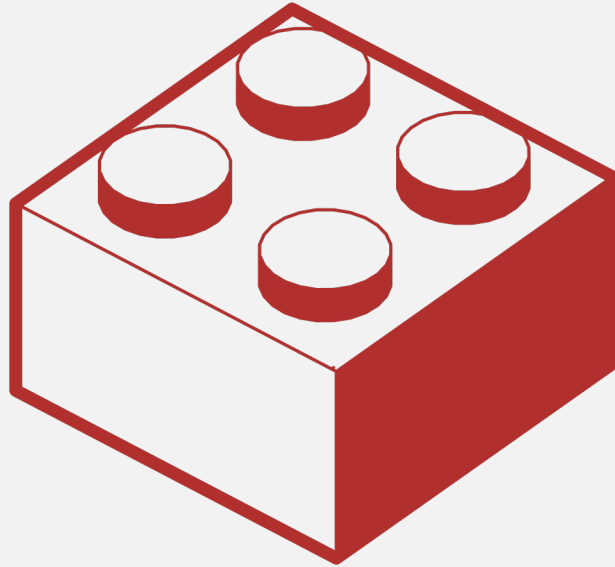
Design Systems



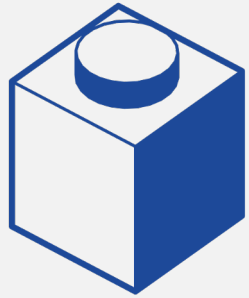
A Brief, Incomplete History of Web-based Component Systems



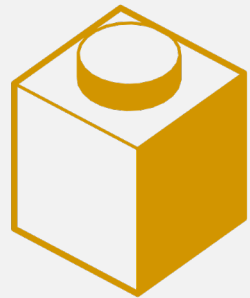
What makes Web Components so unique?



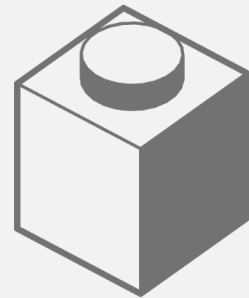
4 native web technologies



HTML <template>



Custom Elements
<custom-element>



Shadow DOM
"Encapsulation"

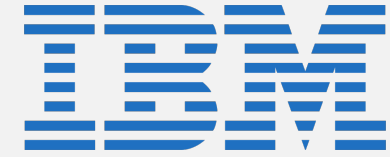


HTML Imports
ES Modules

Great. But...

Are any companies actually using this?

[social validation slide]



According to Google...

12% of pages loaded in Chrome using Web Components

If they're so great...

Why am I not using them?!?



Perfectly valid question!

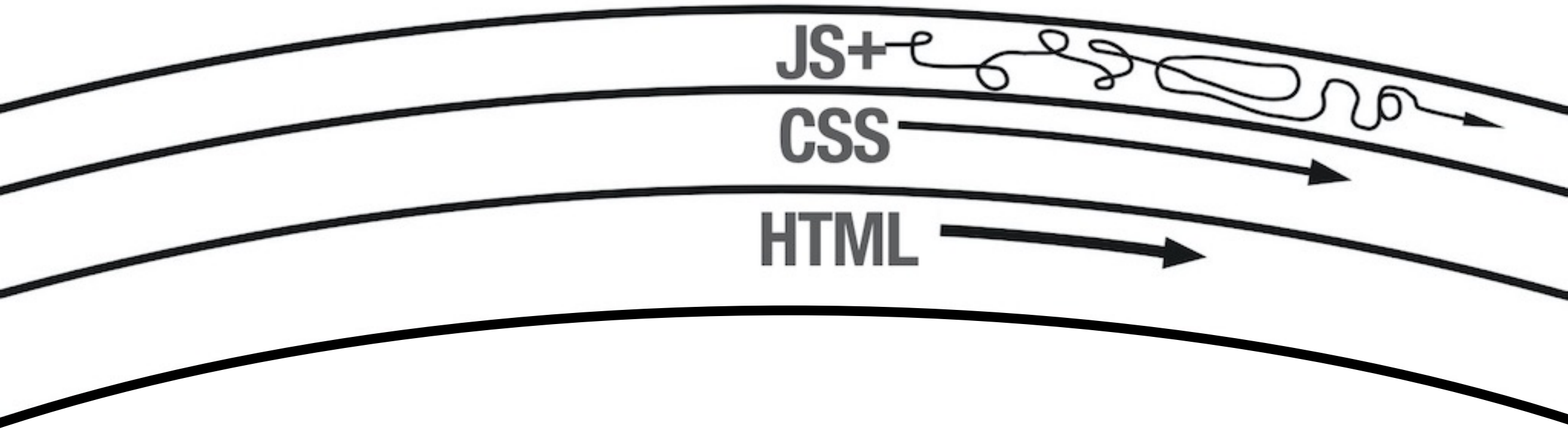
1. Too low-level, designed for framework authors
2. Marketing problem from heavy-handed advocacy
3. Hard to invest in something unknown
4. Not truly supported until very recently

Slow like brisket...

Web Components will always be slower than HotNewFramework.js because they are Web Standards, but that slowness is what can make them a stable foundation to build on.

BUILDING



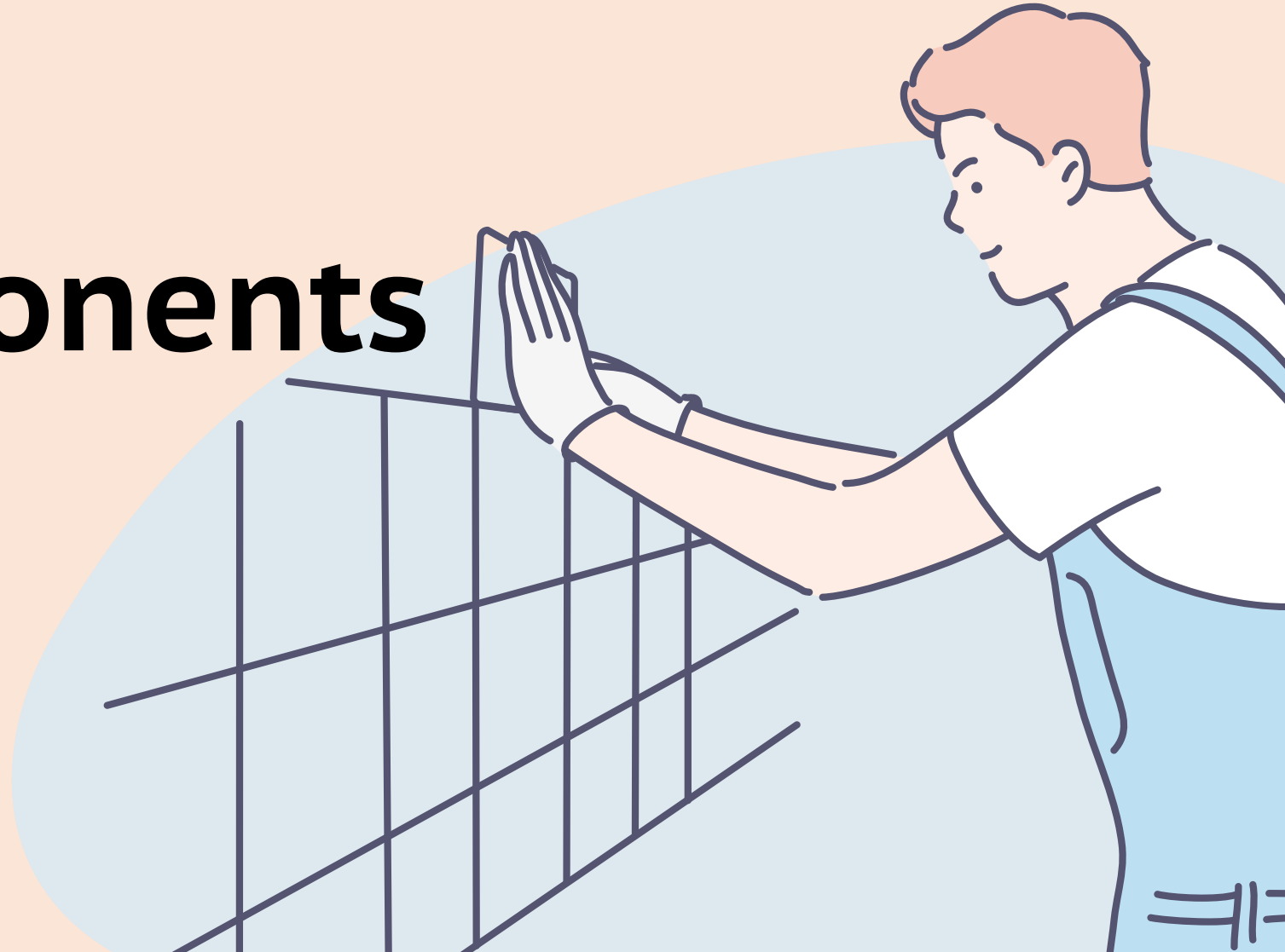


JS+

CSS

HTML

Using Web Components with HTML





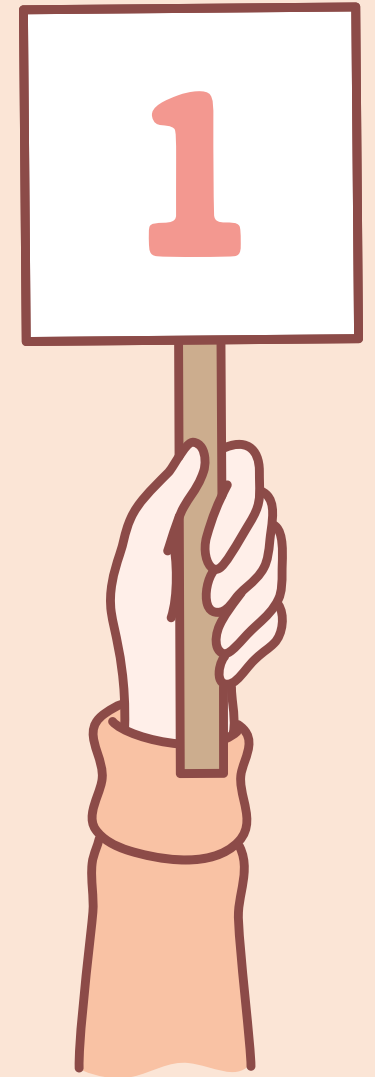


JavaScript

People who just want
to write HTML and CSS

Assertion #1

Web Components bring us one step closer to writing HTML again.





Contribute → Subscribe →

Guardian

News Opinion Sport Culture Lifestyle More ▾

World ▶ Europe US Americas Asia Australia Middle East Africa Inequality Global development

Photography then and now
Japan disaster

Japan's 2011 tsunami, then and now - in pictures

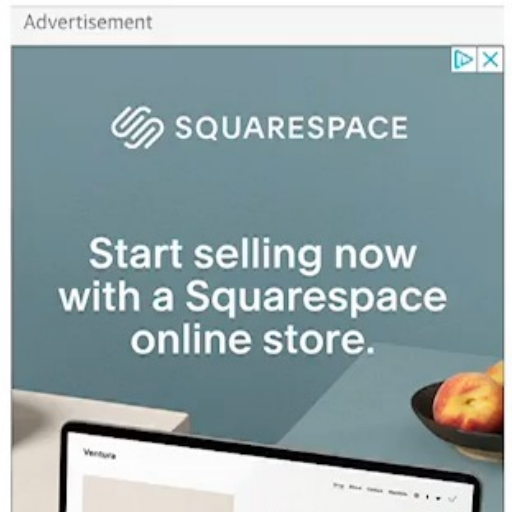
Then and now photographs show the extent of the destruction of the 2011 tsunami in Japan, and the enormity of the reconstruction work

Ten years ago, one of the most powerful earthquakes on record triggered a devastating tsunami in [Japan](#), killing more than 18,000 people and triggering catastrophic meltdowns at the Fukushima Daiichi nuclear plant. Kazuhiro Nogi's photographs compare the destruction of 2011 with the same locations following their reconstruction

Kazuhiro Nogi/AFP/Getty Images

Wed 10 Mar 2021 01:00 EST

f t e 1,099



<two-up>

The screenshot shows the GitHub repository page for `GoogleChromeLabs/two-up`. The browser address bar shows the URL `https://github.com/GoogleChromeLabs/two-up`. The repository name is `GoogleChromeLabs / two-up`. The page includes navigation links for `Code`, `Issues` (2), `Pull requests` (2), `Actions`, `Projects`, `Security`, and `Insights`. The current branch is `master`, with 3 branches and 1 tag. A pull request by `jakearchibald` is highlighted, titled "Merge pull request #3 from GoogleChromeLabs/add-demo", with commit hash `c5be728` and dated Feb 13, 2020. The file list includes `dist`, `lib`, `.gitignore`, `LICENSE`, `README.md`, `package-lock.json`, `package.json`, `rollup.config.js`, `tsconfig.json`, and `tslint.json`. The right sidebar shows the `About` section with no description, a `Readme` link, and an `Apache-2.0 License`. The `Releases` section shows 1 tag, and the `Packages` section shows no published packages. The `Contributors` section shows 3 contributors.

GitHub - GoogleChromeLabs/two-up

https://github.com/GoogleChromeLabs/two-up

Why GitHub? Team Enterprise Explore Marketplace Pricing Search Sign in Sign up

GoogleChromeLabs / two-up

Notifications Star 129 Fork 6

<> Code Issues 2 Pull requests 2 Actions Projects Security Insights

master 3 branches 1 tag Go to file Code

jakearchibald Merge pull request #3 from GoogleChromeLabs/add-demo c5be728 on Feb 13, 2020 7 commits

dist	Initial commit	2 years ago
lib	Initial commit	2 years ago
.gitignore	Initial commit	2 years ago
LICENSE	Initial commit	3 years ago
README.md	Add demo	14 months ago
package-lock.json	1.0.1	2 years ago
package.json	1.0.1	2 years ago
rollup.config.js	Initial commit	2 years ago
tsconfig.json	Initial commit	2 years ago
tslint.json	Initial commit	2 years ago

About

No description, website, or topics provided.

Readme

Apache-2.0 License

Releases

1 tags

Packages

No packages published

Contributors 3

<two-up>

```
<two-up>  
  <div></div>  
  <div></div>  
</two-up>  
  
<script src="path/to/two-up.js"></script>
```



A basic example

```
<div class="alert">  
    
  <p>Alert message goes here.</p>  
  <button>Close</button>  
</div>
```

A basic example



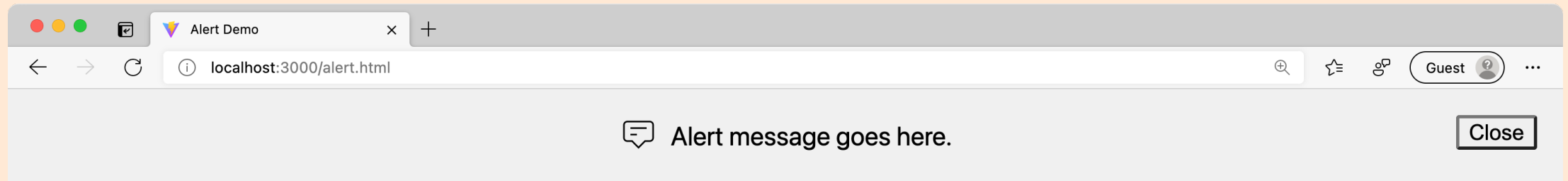
```
<custom-alert icon="note">
```

```
  <p>Alert message goes here.</p>
```

```
</custom-alert>
```

```
<script src="path/to/custom-alert.js"></script>
```

A basic example



A basic example

```
<custom-alert icon="note">
```

```
<p>Alert message goes here.</p>
```

```

```

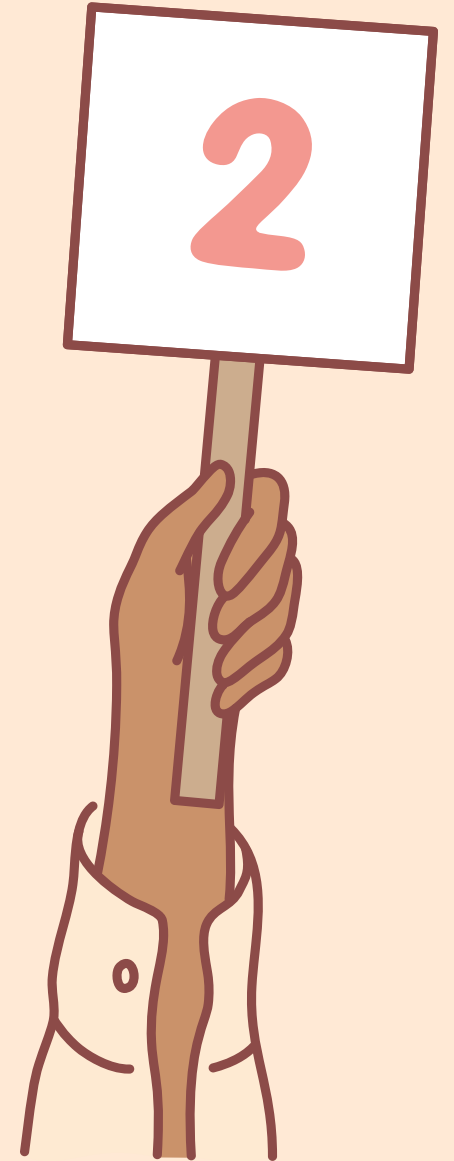
```
<slot></slot>
```

```
<button>Close</button>
```

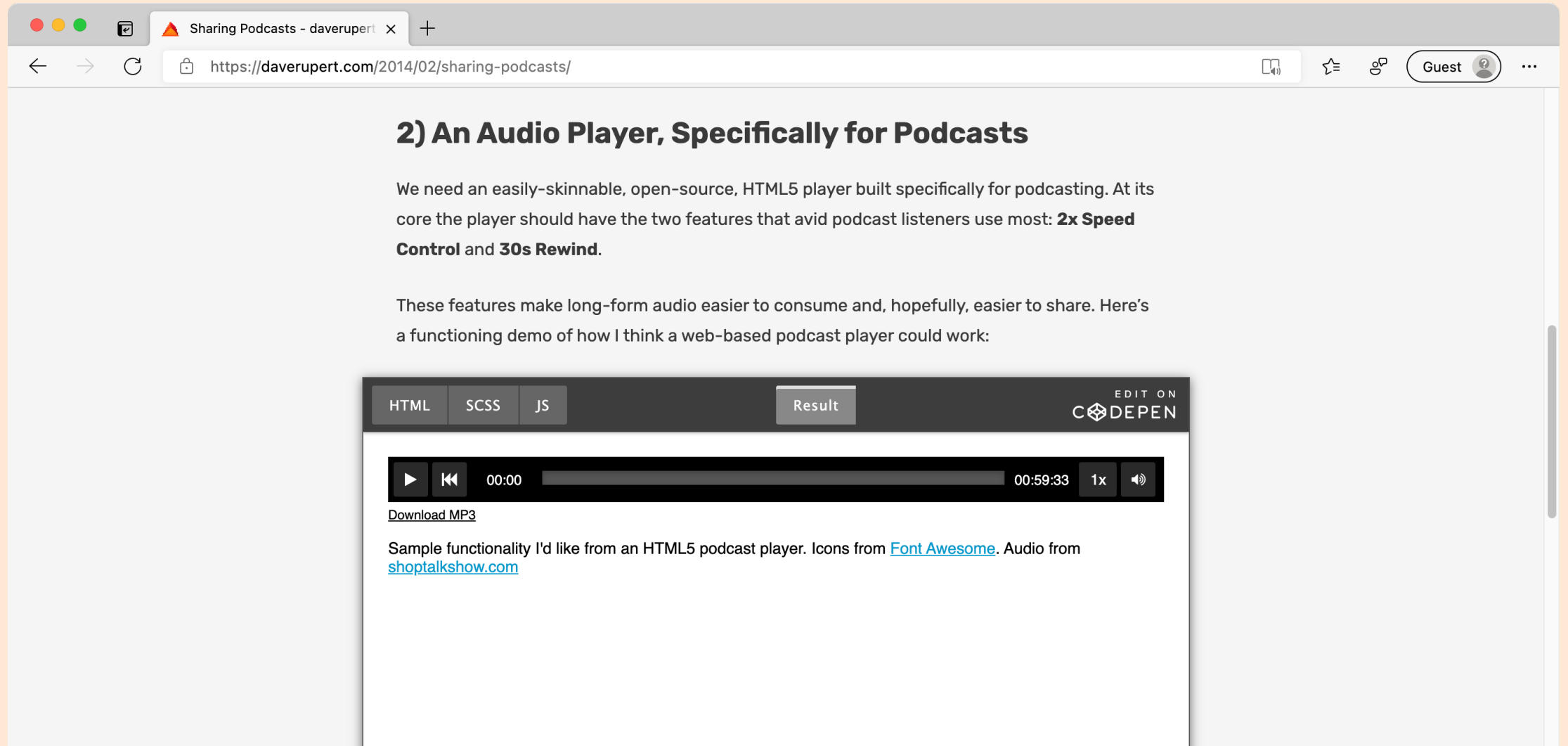
```
</custom-alert>
```


Assertion #2

Web Components are great for
Progressive Enhancement



I have strong opinions about podcast players



The screenshot shows a web browser window with the address bar displaying `https://daverupert.com/2014/02/sharing-podcasts/`. The page content includes a section header, two paragraphs of text, and a code editor demo.

2) An Audio Player, Specifically for Podcasts

We need an easily-skinnable, open-source, HTML5 player built specifically for podcasting. At its core the player should have the two features that avid podcast listeners use most: **2x Speed Control** and **30s Rewind**.

These features make long-form audio easier to consume and, hopefully, easier to share. Here's a functioning demo of how I think a web-based podcast player could work:

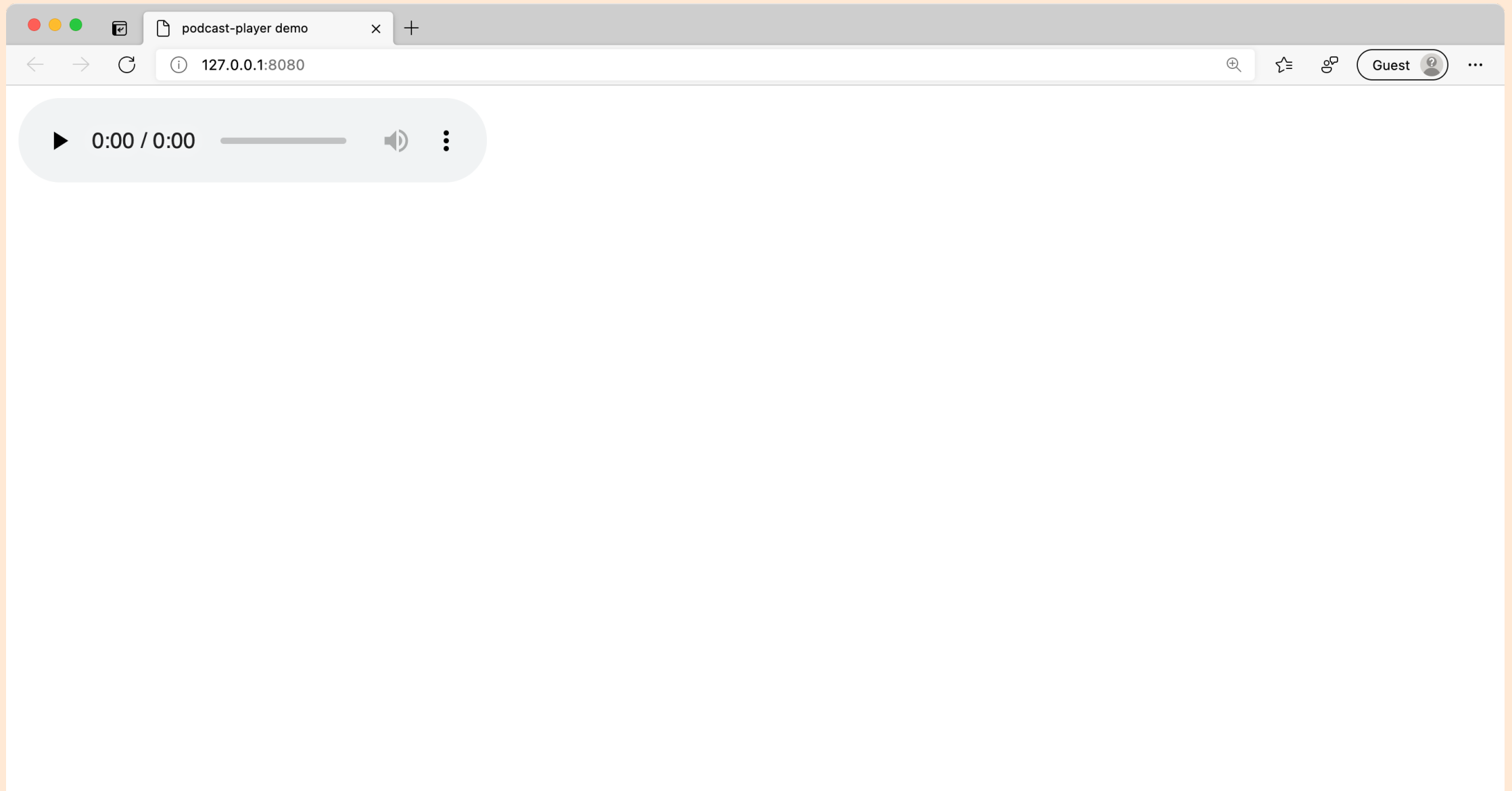
HTML SCSS JS Result EDIT ON CODEPEN

▶ ⏪ 00:00 00:59:33 1x 🔊

[Download MP3](#)

Sample functionality I'd like from an HTML5 podcast player. Icons from [Font Awesome](#). Audio from [shoptalkshow.com](#)

The native `<audio>` player



<podcast-player>

```
<podcast-player>
```

```
  <audio src="path/to/my.mp3" controls></audio>
```

```
</podcast-player>
```

```
<script type="module" src="path/to/podcast-player.js"></script>
```

<podcast-player>



<podcast-player>

The image shows a web browser window displaying a podcast player interface. The browser's address bar shows the URL `127.0.0.1:8080`. The player interface includes a play button, a progress bar set to `00:00`, and buttons for 30-second rewind and fast forward. Below the browser, the Chrome DevTools 'Elements' panel is open, showing the component's shadow DOM structure. The component is a `<podcast-player>` which contains a `<slot>` for an audio player, an `<svg>` with icons for play, pause, rewind, and speaker, and a `<div class="podcast-player">` containing buttons for rewind, play, fast forward, progress, duration, speed, and mute.

```
<!DOCTYPE html>
<html lang="en-us">
  <head>...</head>
  <body>
    <podcast-player>
      <!-->
      <slot>
        <audio> reveal
      </slot>
      <svg style="display: none;">
        <symbol id="icon-play" viewBox="0 0 30.406 46.843">...</symbol>
        <symbol id="icon-pause" viewBox="0 0 32 32">...</symbol>
        <symbol id="icon-rewind" viewBox="0 0 32 32">...</symbol>
        <symbol id="icon-speaker-unmuted" viewBox="0 0 32 32">...</symbol>
        <symbol id="icon-speaker-muted" viewBox="0 0 32 32">...</symbol>
      </svg>
      <div class="podcast-player">
        <button class="button-rewind" aria-label="Rewind 30 seconds" style="grid-area: rewind">...</button>
        <button class="button-play" aria-label="Play" style="grid-area: play">...</button>
        <button class="button-ff" aria-label="Fast Forward 30 seconds" style="grid-area: ff">...</button>
        <div style="grid-area: currenttime">...</div>
        <input type="range" class="progress" style="grid-area: progress" value="0" max="0">
        <div style="grid-area: duration">...</div>
        <button class="button-speed" style="grid-area: speed">...</button>
        <button class="button-mute" aria-label="Mute" style="grid-area: mute">...</button>
      </div>
    </podcast-player>
  </body>
</html>
```

<podcast-player>

The screenshot shows a web browser window with the URL <https://shoptalkshow.com/456/>. The page features the 'SHOPTALK SHOW' logo, a description of the podcast, and navigation links. A large audio player is centered on the page, showing a progress bar at 00:00 and a volume icon set to 1x. Below the player, the episode title '456: WordPress Block Editor, ElementInternals, Writing Code or Leading a Team, and Container Queries' is displayed, along with the date 'MARCH 28TH 2021' and a short description. To the right, a 'GUESTS' section shows a photo of two people holding signs that say 'SHAWP TALK SHOW' and 'DOT:COM'.

456: WordPress Block Editor, ElementInternals, Writing Code or Leading a Team, and Container Queries

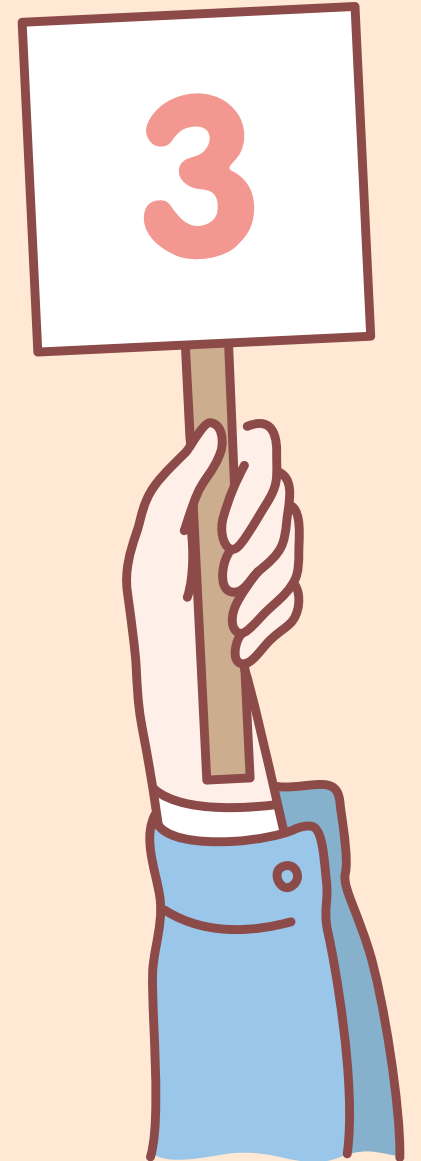
MARCH 28TH 2021

On this episode: Updates from Dave's shed, camera options for streaming and Zooming, Discord chats, Headless WordPress block editor musings, custom form controls with ElementInternals, going from a coder to a team leader, and container query updates.

GUESTS

Assertion #3

Web Components will let us make simpler, more accessible, and more maintainable websites.



Wait. Wait. Hold up. Wait.

Are you saying Web Components would allow us to create and share **accessible prebuilt components** *without the need for specific JavaScript libraries and overly complicated build processes?*!

Yes. I am.

That's the vision.

Generic Components

generic-components

https://genericcomponents.netlify.app/generic-tabs/demo/index.html

generic

[showcase app](#)

[generic-accordion](#)

[generic-alert](#)

[generic-dialog](#)

[generic-disclosure](#)

[generic-listbox](#)

[generic-radio](#)

[generic-skiplink](#)

[generic-switch](#)

[generic-tabs](#)

[generic-visually-hidden](#)

[Github](#)

Default:

► Show code

About Contact

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Selected:

► Show code

About Contact

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

<generic-tabs>

```
<generic-tabs>  
  <button slot="tab">About</button>  
  <button slot="tab">Contact</button>  
  
  <div slot="panel">  
    <p>Lorem ipsum dolor sit amet...</p>  
  </div>  
  <div slot="panel">  
    <p>Sed ut perspiciatis unde omnis...</p>  
  </div>  
</generic-tabs>
```

<generic-tabs>

```
<generic-tabs>
  <h2>About</h2>
  <div>
    <p>Sed ut perspiciatis unde ...</p>
  </div>

  <h2>Contact</h2>
  <div>
    <p>Ut enim ad minima veniam...</p>
  </div>
</generic-tabs>
```

[whatwg] Suggestion: Implementation of Tabbed Forms

This message: [[Message body](#)] [[Respond](#)] [[More options](#)]

Related messages: [[Next message](#)] [[Previous message](#)] [[In reply to](#)] [[Next in thread](#)] [[Replies](#)]

From: Ian Hickson <ian@hixie.ch>

Date: Tue, 17 Aug 2004 15:05:47 +0000 (UTC)

Message-ID: <Pine.LNX.4.61.0408171501020.16813@dhalsim.dreamhost.com>

On Tue, 17 Aug 2004, Matthew Raymond wrote:

```
>
> Ian Hickson wrote:
> > At the moment the spec says it would be:
> >
> > <switch>
> > <section>
> > <h1>Exclusive Section 1</h1>
> > </section>
> > <section>
> > <h1>Exclusive Section 2</h1>
> > </section>
> > </switch>
```

> I presume there is a mechanism for selecting a specific section. If so, > would there be markup that would serve a function similar to a VB > tabstrip? Will there be a markup solution to selecting a section?

The above is not a tabbed system, it's just a list of mutually exclusive sections. Tabs are not mutually exclusive, they are just an unordered group of related sections that are usually shown such that only one is visible at any one time.

Anyway, the spec currently has a separate section for tabs, which says that the tabbed version of the above would be:

```
<tabbox>
  <section>
    <h1>Section 1</h1>
  </section>
  <section>
    <h1>Section 2</h1>
  </section>
</tabbox>
```

The name <tabbox> will probably be changed to <group>. See the spec for better examples. (Graceful degradation as your example had is possible too, since non-section elements in the tabbox are ignored.)



Assertion #4

Web Components are a gateway to a new dimension.



<model-viewer>

Easily display interactive 3D models on the web & in AR

Quick Start

```
<!-- Import the component -->  
<script type="module" src="https://unpkg.com/@google/model-viewer/dist/model-viewer.min.js"></script>  
  
<!-- Use it like any other HTML element -->  
<model-viewer src="shared-assets/models/Astronaut.glb" alt="A 3D model of an astronaut" auto-rotate camera-controls></model-viewer>
```

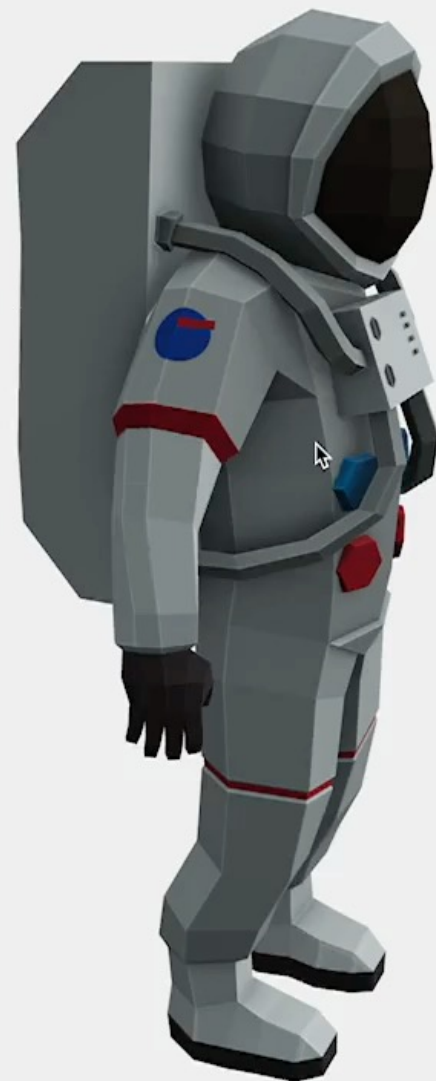
minzipped size 202.6 KB release v1.6.0
Follow @modelviewer 1.7k Star 2.7k

Documentation

API [documentation](#) & [examples](#)

Model Editor

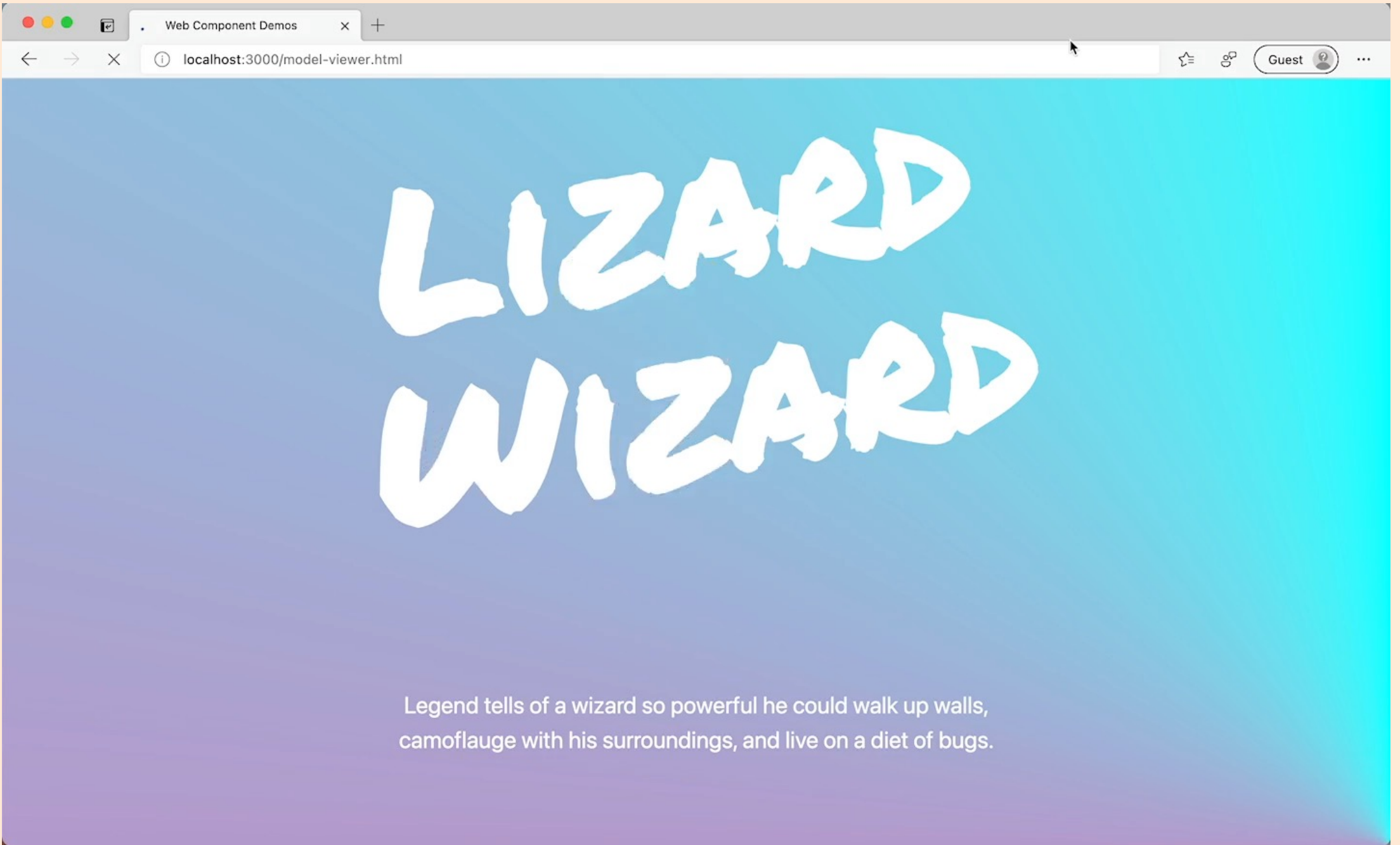
Use the [Model Editor](#) to prepare your GLB models for the web
[Interactive Viewer](#) (Legacy)



<model-viewer>

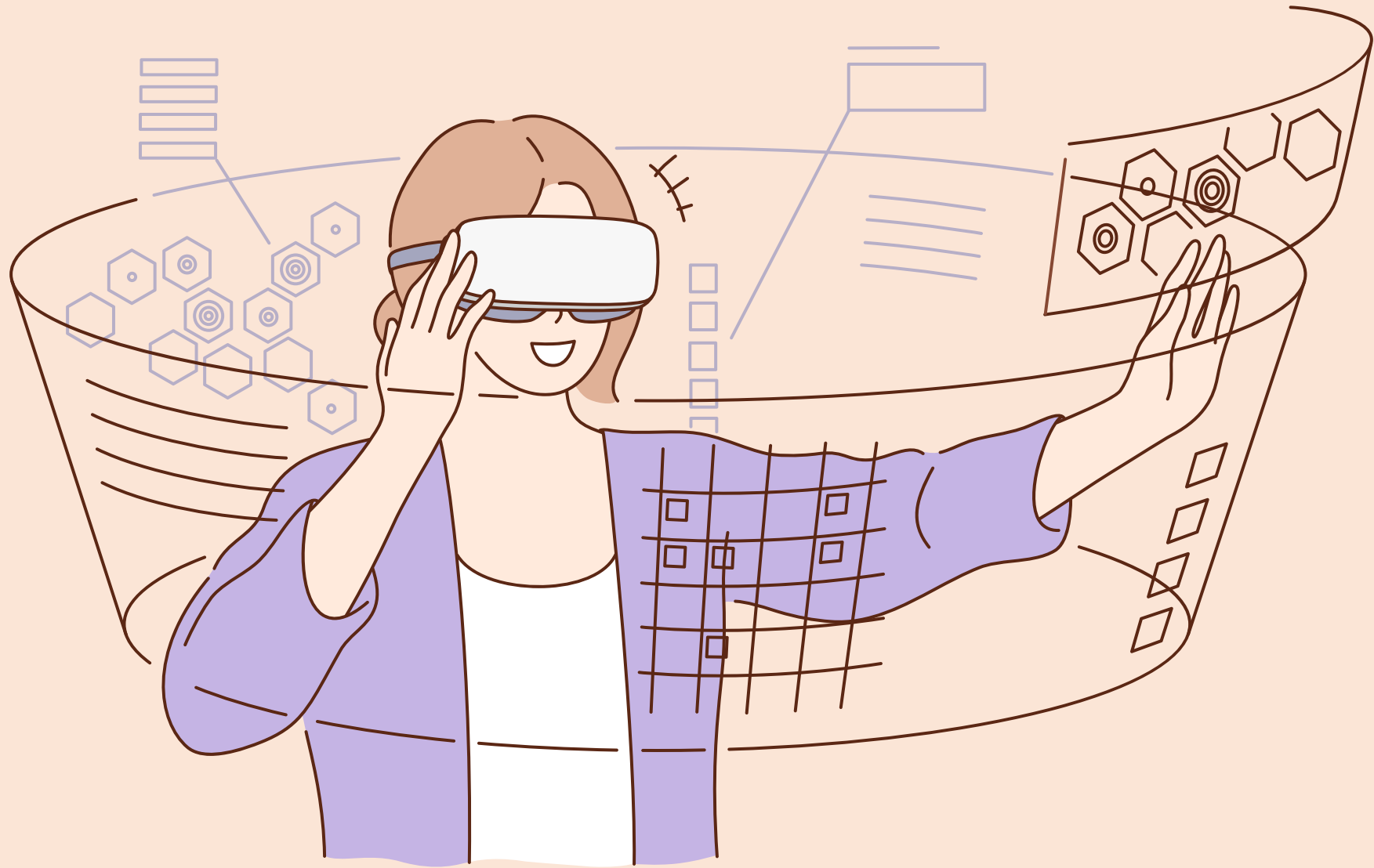
```
<script type="module" src="path/to/model-viewer.min.js"></script>
```

```
<model-viewer src="path/to/Astronaut.glb"  
  alt="A 3D model of an astronaut"  
  auto-rotate  
  camera-controls>  
</model-viewer>
```



LIZARD WIZARD

Legend tells of a wizard so powerful he could walk up walls, camoflauge with his surroundings, and live on a diet of bugs.



A-FRAME

Blog

A-Frame 1.1.0 - AR, Quest 2 ...

Examples

Hello WebVR

Model Viewer

Hand Tracking

Responsive UI

360° Image

360 Video

Anime UI

BeatSaver Viewer

Moon Rider

Gunters of OASIS 🐱

Supercraft 🐱

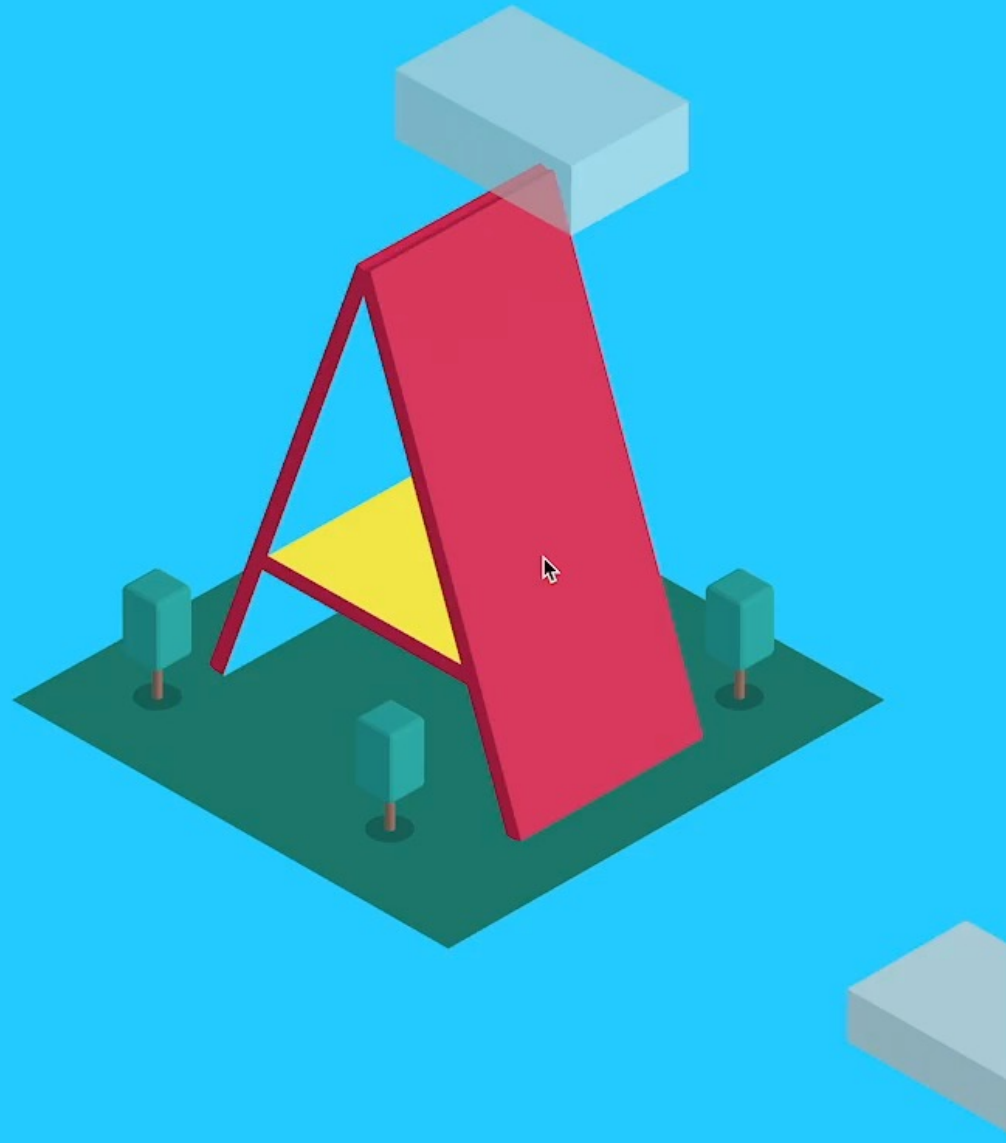
Super Says 🐱

Towermax Fitness 🐱

A-Blast 🐱

A-Painter 🐱

A Saturday Night 🐱



DOCS FAQ BLOG COMMUNITY SHOWCASE

A web framework for building 3D/AR/VR experiences

Make 3D worlds with HTML and Entity-Component
For Quest, Rift, WMR, SteamVR, mobile, desktop

GET STARTED

<a-frame>

```
<script src="path/to/aframe.min.js"></script>
```

```
<a-scene>
```

```
  <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
```

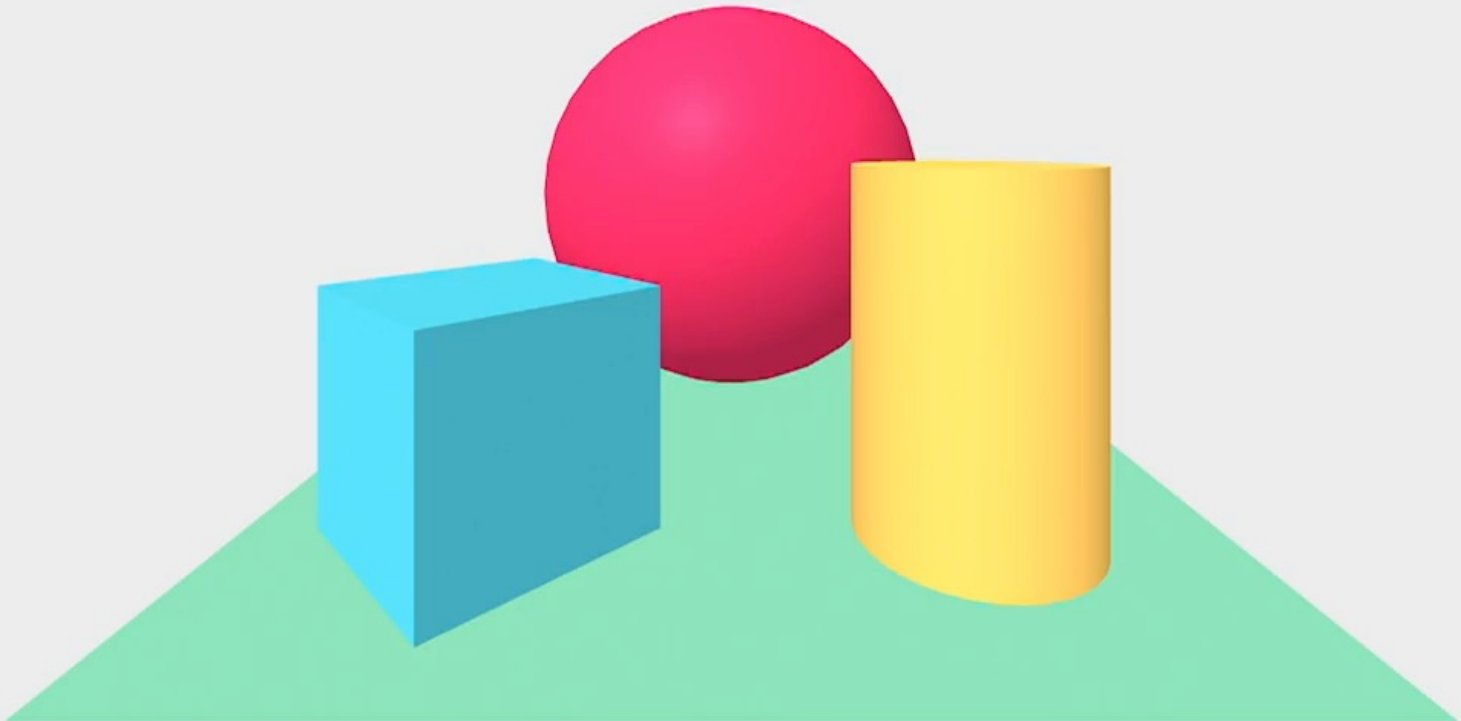
```
  <a-box position="-1 0.5 -3" rotation="0 45 0" width="1" height="1" depth="1" color="#4CC3D9"></a-box>
```

```
  <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>
```

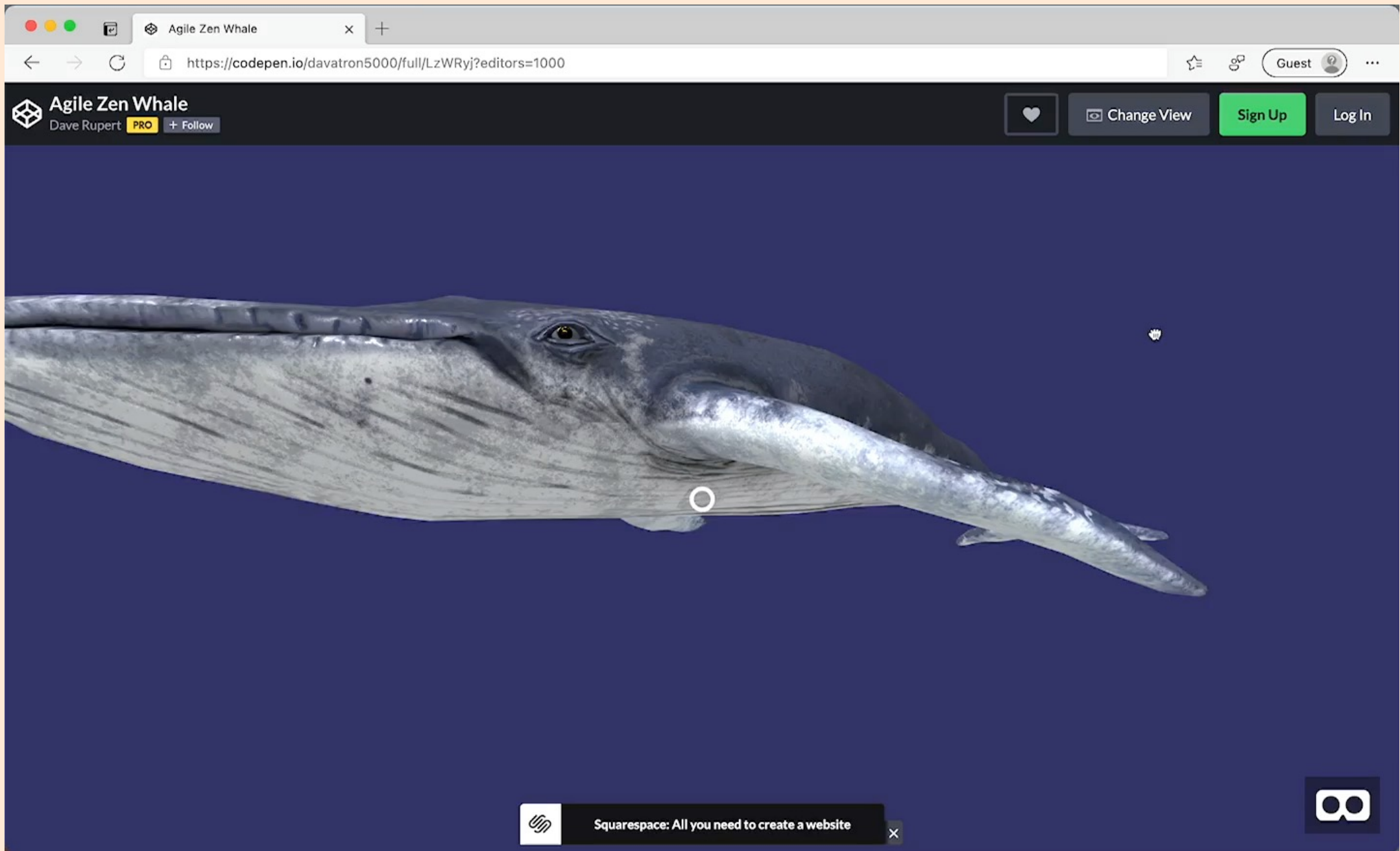
```
  <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a-plane>
```

```
  <a-sky color="#ECECEC"></a-sky>
```

```
</a-scene>
```



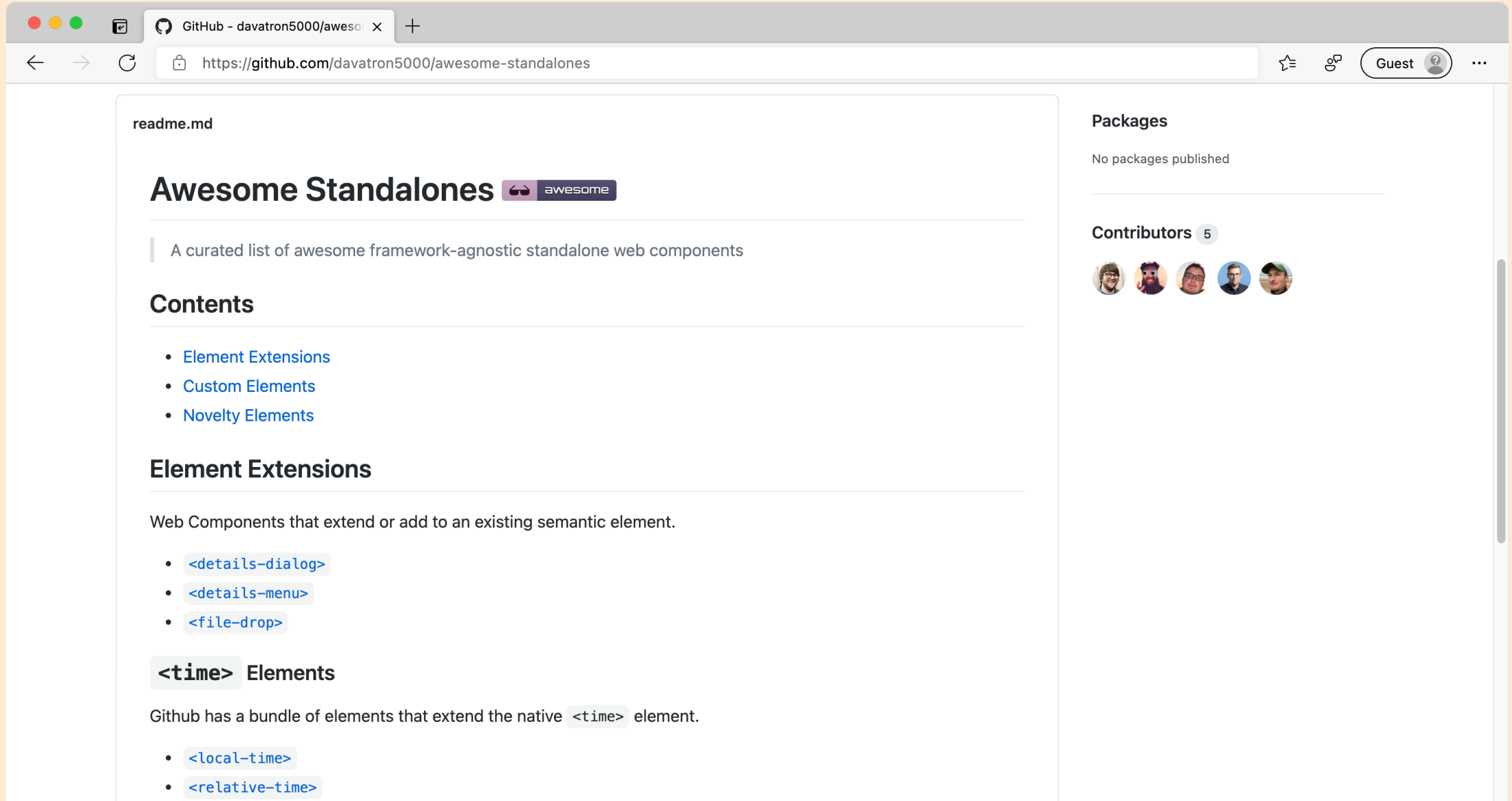
VR



How do you find cool Web Components?

Me, personally? Poor work/life balance. You?

Awesome Standalones



The screenshot shows a web browser window displaying the GitHub repository page for 'davatron5000/awesome-standalones'. The browser's address bar shows the URL 'https://github.com/davatron5000/awesome-standalones'. The page content includes a 'readme.md' file with the following structure:

- ## Awesome Standalones awesome
- A curated list of awesome framework-agnostic standalone web components
- ### Contents

 - [Element Extensions](#)
 - [Custom Elements](#)
 - [Novelty Elements](#)
- ### Element Extensions

Web Components that extend or add to an existing semantic element.

 - `<details-dialog>`
 - `<details-menu>`
 - `<file-drop>`
- ### `<time>` Elements

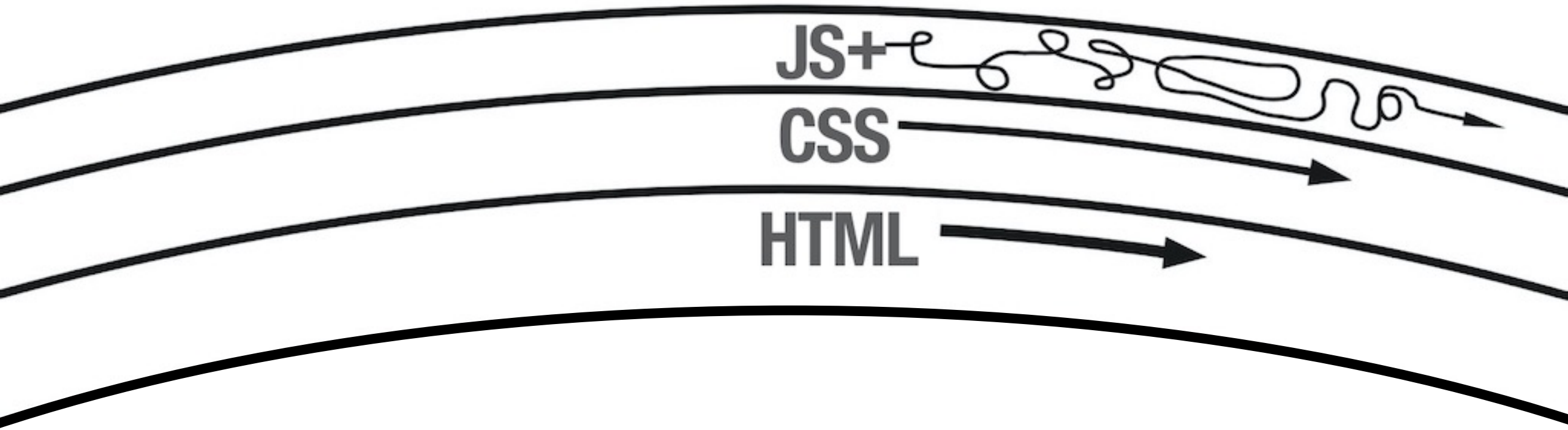
Github has a bundle of elements that extend the native `<time>` element.

 - `<local-time>`
 - `<relative-time>`

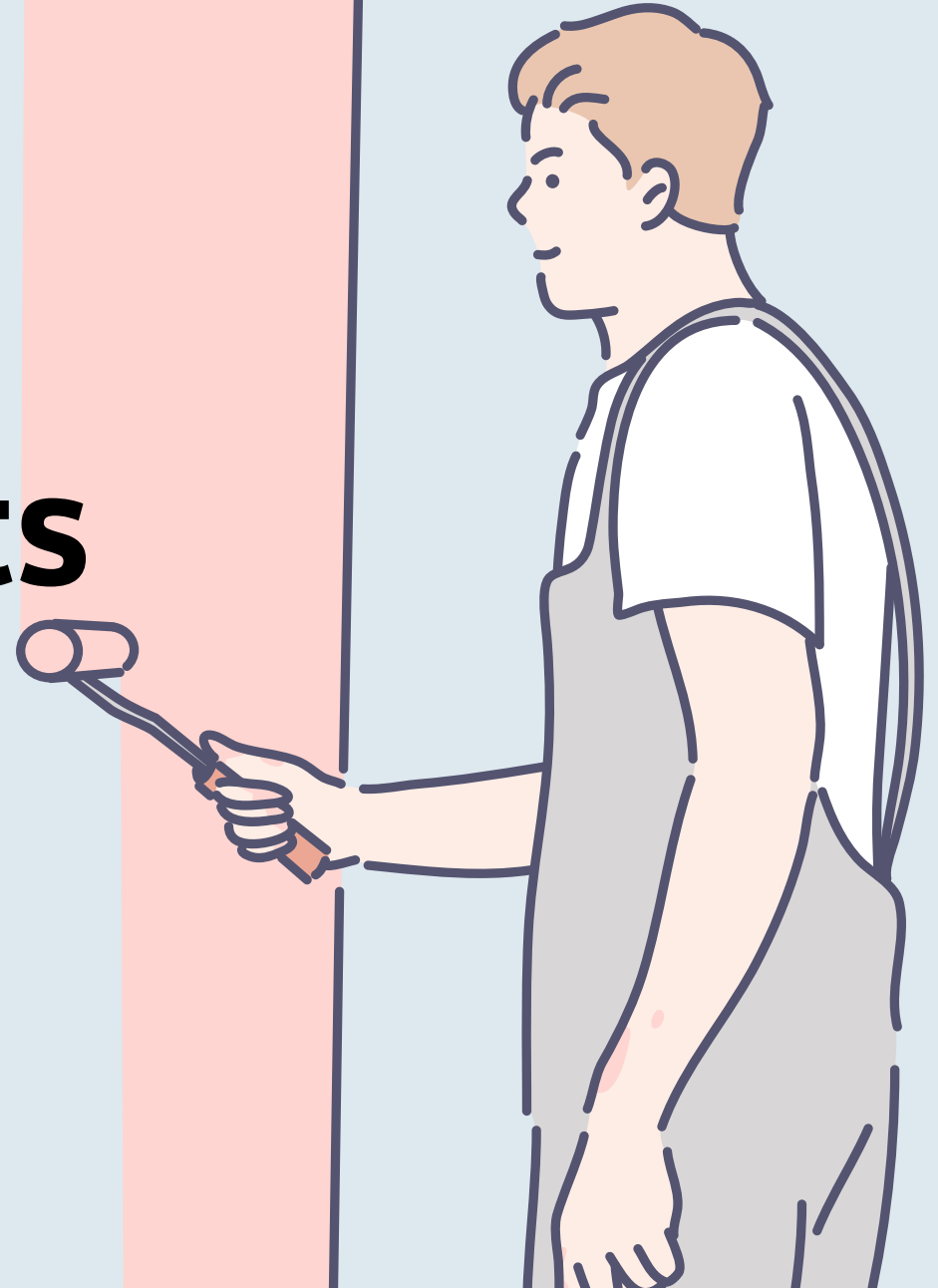
On the right side of the page, there are sections for 'Packages' (No packages published) and 'Contributors' (5 contributors, with 5 profile pictures shown).

Summary of what's good about Web Components

- ✓ Easy as HTML
- ✓ Great reusability
- ✓ Better experiences
- ✓ Progressively enhanced
- ✓ No build tools
- ✓ “Encapsulated”



Styling Web Components with CSS



Truth be told...

This part isn't very fun.



A dark, atmospheric illustration of a creature with glowing blue energy and a bright light source in a cavernous setting. The creature has a menacing appearance with glowing blue energy emanating from its body. The background is a dark, cavernous space with a bright light source on the right, creating a dramatic contrast. The overall tone is mysterious and intense.

PIERCING THE SHADOW DOM

What pierces the Shadow DOM?

```
<custom-alert>  
  <p>Light DOM goes here</p>  
</custom-alert>
```

What pierces the Shadow DOM?

```
<custom-alert>  
  <p>Light DOM goes here</p>  
    
  <slot></slot>  
  <button>Close</button>  
</custom-alert>
```


What pierces the Shadow DOM?

```
body p {  
  transform: rotate(-10deg);  
}
```

```
custom-alert button {  
  background: pink;  
}
```

```
button {  
  color: pink;  
}
```

```
custom-alert {  
  --bg: red;  
  --text: white;  
}
```



What pierces the Shadow DOM?

```
custom-alert button {  
  background: pink;  
}
```

Shadow Boundary

```
body p {  
  transform: rotate(-10deg);  
}
```

```
button {  
  color: pink;  
}
```

```
custom-alert {  
  --bg: red;  
  --text: white;  
}
```

A loss of control


When you're comfortable with the global, cascading nature of CSS it can be quite frustration when it quits working.



Alert Demo × +

localhost:3000/alert.html

Guest ? ...

 This is my alert message with some very important information.

Close

Styling Light DOM?

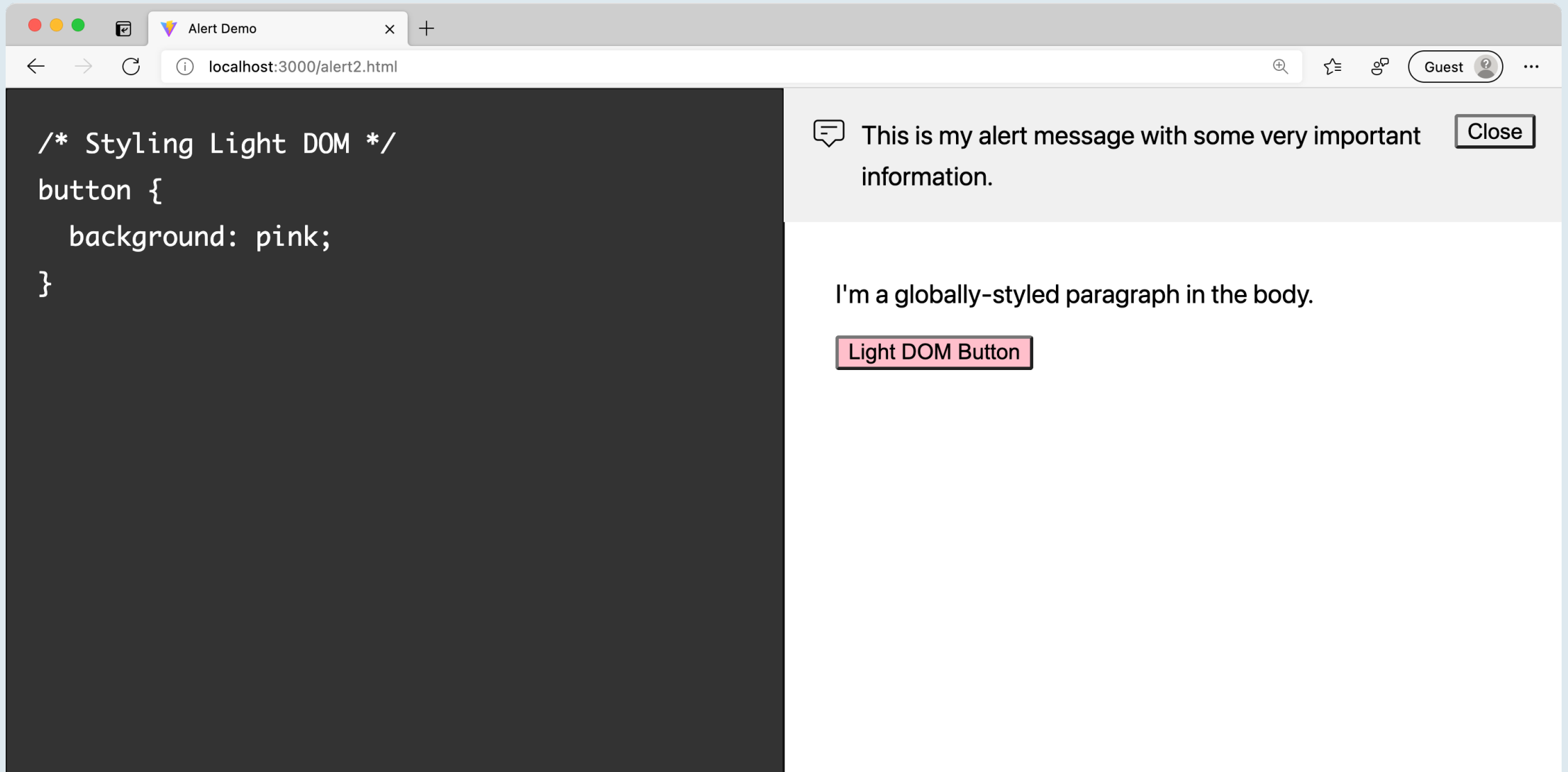
The screenshot shows a web browser window with a single tab titled "Alert Demo". The address bar displays "localhost:3000/alert2.html". The browser interface includes navigation buttons (back, forward, refresh), search, star, and share icons, and a user profile labeled "Guest".

On the left side, a dark-themed code editor displays the following CSS code:

```
/* Styling Light DOM */  
p {  
  transform: rotate(-2deg);  
}
```

On the right side, the rendered page shows a light gray alert message box at the top with a speech bubble icon, the text "This is my alert message with some very important information.", and a "Close" button. Below the alert, the text "I'm a globally-styled paragraph in the body." is displayed. At the bottom, there is a button labeled "Light DOM Button".

What about that <button>?



The screenshot shows a web browser window with a tab titled "Alert Demo" and the address bar displaying "localhost:3000/alert2.html". The browser interface includes navigation buttons, search, and user profile information.

On the left side, a code editor displays the following CSS code:

```
/* Styling Light DOM */  
button {  
  background: pink;  
}
```

On the right side, the rendered page shows an alert message box with a speech bubble icon, the text "This is my alert message with some very important information.", and a "Close" button. Below the alert, there is a paragraph of text: "I'm a globally-styled paragraph in the body." and a button labeled "Light DOM Button" with a pink background.

Deep Styling Shadow Elements?



The image shows a browser window with a code editor on the left and a rendered page on the right. The code editor contains the following CSS:

```
/* Deep-styling Shadow DOM */  
custom-alert button {  
  background: pink;  
}
```

The rendered page shows a modal alert dialog with a pink background and a 'Close' button. Below the dialog, there is a paragraph of text and a button labeled 'Button'. The browser's address bar shows 'localhost:3000/alert2.html' and the user is logged in as 'Guest'.

**But if this wasn't difficult
enough...**

Some styles DO bleed through the shadow boundary!

Inheritable Styles

- border-collapse
- border-spacing
- caption-side
- color
- cursor
- direction
- empty-cells
- font-family
- font-size
- font-style
- font-variant
- font-weight
- font-size-adjust
- font-stretch
- font
- letter-spacing
- line-height
- list-style-image
- list-style-position
- list-style-type
- list-style
- orphans
- quotes
- tab-size
- text-align
- text-align-last
- text-decoration-color
- text-indent
- text-justify
- text-shadow
- text-transform
- visibility
- white-space
- widows
- word-break
- word-spacing
- word-wrap

Inheritable Styles

- border-collapse
- border-spacing
- caption-side
- **color**
- **cursor**
- direction
- empty-cells
- **font-family**
- **font-size**
- **font-style**
- **font-variant**
- **font-weight**
- font-size-adjust
- font-stretch
- **font**
- **letter-spacing**
- **line-height**
- list-style-image
- list-style-position
- list-style-type
- list-style
- orphans
- quotes
- tab-size
- **text-align**
- text-align-last
- text-decoration-color
- text-indent
- text-justify
- text-shadow
- **text-transform**
- visibility
- white-space
- widows
- word-break
- word-spacing
- word-wrap

Inheritable Styles?

The screenshot shows a web browser window with the following content:

```
/* Inheritable styles */  
body {  
  font-family: fantasy;  
  color: red;  
}
```

The browser view displays:

- An alert message: "This is my alert message with some very important information." (text is red, button is "Close")
- A paragraph: "I'm a globally-styled paragraph in the body." (text is red)
- A button: "Light DOM Button" (text is light gray)

CSS Variables?

```
/* CSS Variables */
custom-alert {
  --bg: rebeccapurple;
  --text: white;
}
```

This is my alert message with some very important information. Close

I'm a globally-styled paragraph in the body.

Light DOM Button

<two-up> styling API

```
/* Two Up Styling API */  
two-up {  
  --accent-color: #777;  
  --track-color: #777;  
  --thumb-color: #777;  
  --thumb-background: #fff;  
  --thumb-size: 62px;  
  --bar-size: 6px;  
  --bar-touch-size: 30px;  
}
```

CSS Variables are a great way to add a styling API


They offer the right amount of flexibility for most minor customizations.

More ways to offer styling APIs

A few “hooks” to let their components be customized.

Custom themes or classes

```
<custom-alert theme="warn">  
  <p>Light DOM goes here</p>  
</custom-alert>
```


 This is my alert message with some very important information.

Close

 **Oh god.**

Someone gave them a WYSIWYG and now they're dropping headings in here. I'm very worried about the success of the site now.

Close

 **Welcome to the danger zone.**

Close

I'm still kinda mad

About that button.



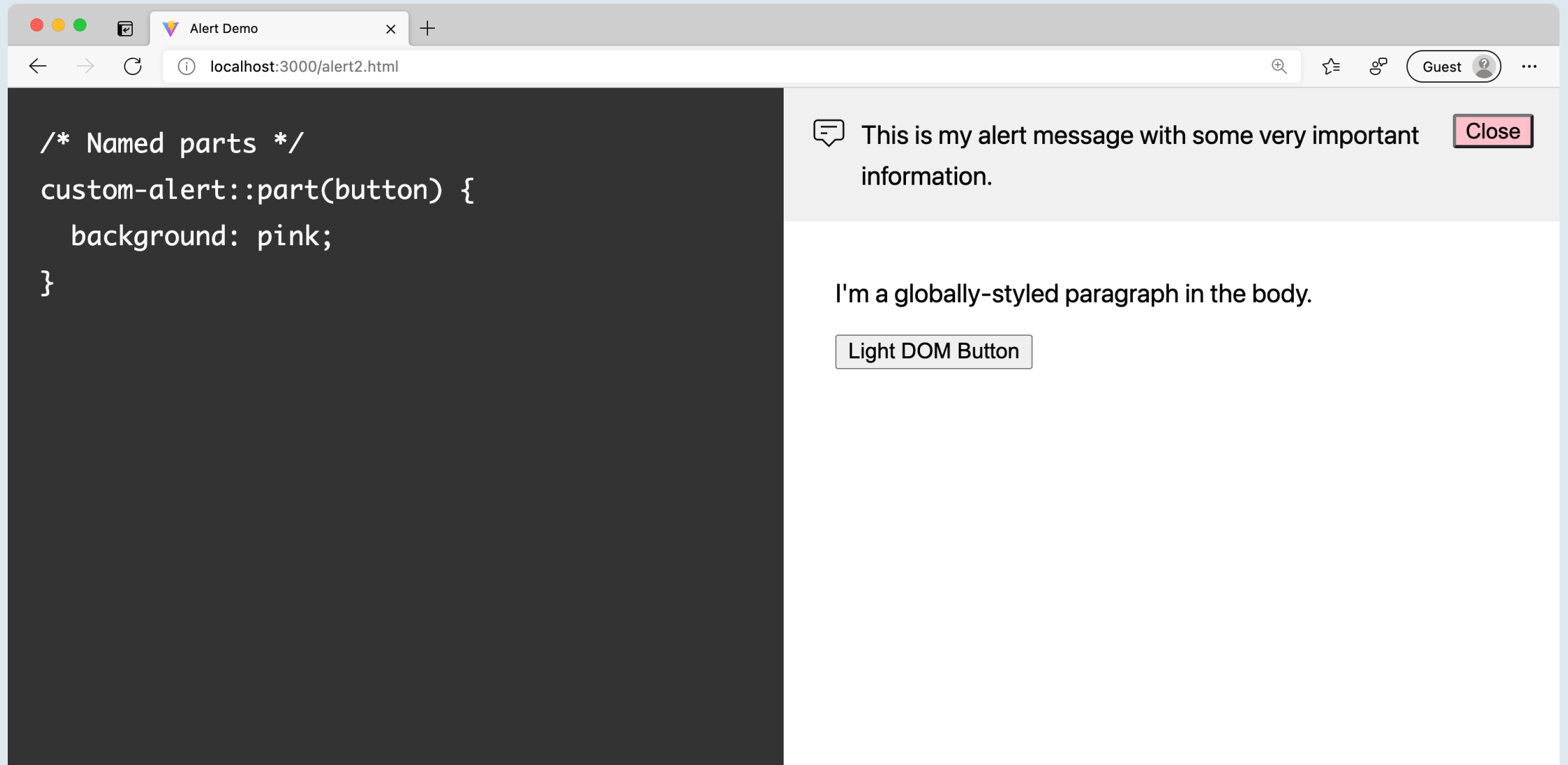
Named Shadow Parts

```
<custom-alert>  
  <p>Light DOM goes here</p>  
</custom-alert>
```

Named Shadow Parts

```
<custom-alert>  
  <p>Light DOM goes here</p>  
    
  <slot></slot>  
  <button name="button">Close</button>  
</custom-alert>
```

Named Shadow Parts



The screenshot shows a web browser window with the title "Alert Demo" and the URL "localhost:3000/alert2.html". The browser's address bar shows navigation icons, a search icon, and a user profile labeled "Guest".

On the left side of the browser, a dark grey code editor displays the following CSS code:

```
/* Named parts */
custom-alert::part(button) {
  background: pink;
}
```

On the right side, the browser's content area shows a custom alert dialog box. The dialog has a light beige background and a pink border. It contains a speech bubble icon, the text "This is my alert message with some very important information.", and a pink "Close" button.

Below the alert dialog, the main page content is visible. It features a paragraph of text: "I'm a globally-styled paragraph in the body." Below this paragraph is a button with the text "Light DOM Button".

Crossing the Shadow Boundary

Styling Light DOM

Deep Styling Shadow DOM

Inheritable Styles

CSS Variables

Theming

Named Shadow Parts



Crossing the Shadow Boundary

Deep Styling Shadow DOM



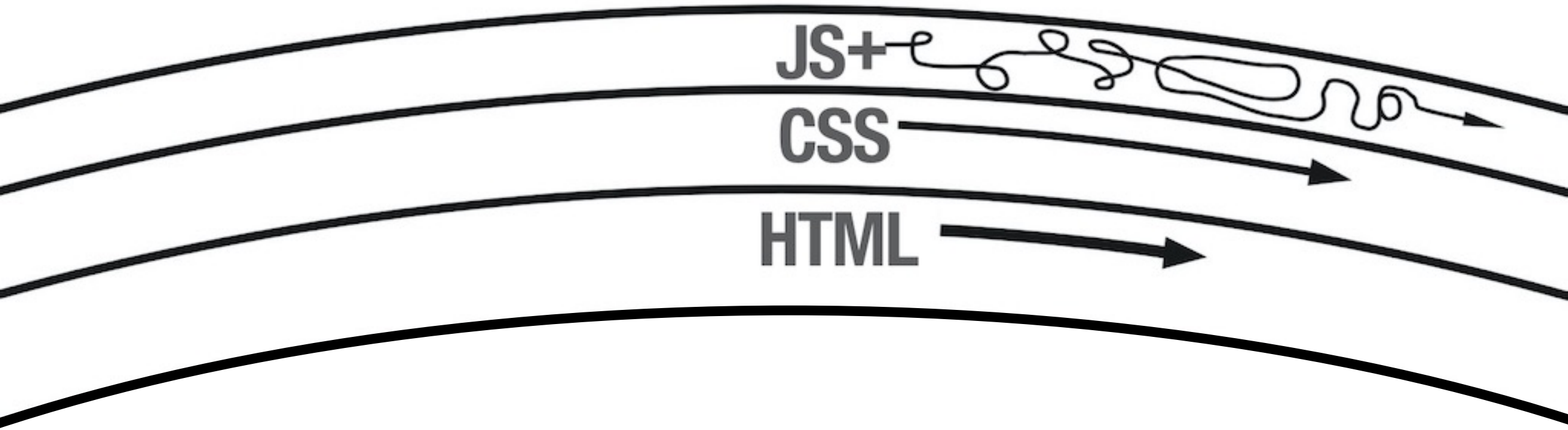
Styling Light DOM

Inheritable Styles

CSS Variables

Theming

Named Shadow Parts



Writing Web Components with JS



Web Component Lifecycle

```
class CustomAlert extends HTMLElement {
  static get observedAttributes() {
    return ['icon', 'theme'];
  }

  constructor() {
    super();
  }

  connectedCallback() {} // componentDidMount
  disconnectedCallback() {} // componentWillUnmount
  attributeChangedCallback(name, oldValue, newValue) {}
}

customElements.define('custom-alert', CustomAlert);
```

Small libraries (5~10kb) exist to make writing Web Components nicer.

The screenshot shows a web browser displaying the Stencil.js documentation page for "My First Component". The browser's address bar shows the URL `https://stenciljs.com/docs/my-first-component`. The page features a dark header with the Ionic logo and navigation links for "Docs", "Resources", and "Blog". A left sidebar contains a navigation menu with categories like "GETTING STARTED", "BASICS", and "CORE FEATURES". The main content area is titled "My First Component" and includes an introduction, a code example, and a "CONTENTS" sidebar. The code example shows the Stencil component structure:

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

@Component({
  tag: 'my-first-component',
})
export class MyComponent {

  // Indicate that name should be a public property on the component
  @Prop() name: string;

  render() {
    return (
      <p>
        My name is {this.name}
      </p>
    );
  }
}
```

A basic example in lit

```
<custom-alert icon="note" theme="warn">  
  <p>Alert message goes here.</p>  
</custom-alert>
```

A basic example

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

export class CustomAlert extends LitElement {
  static get styles() {
    return css`
      :host {
        background: var(--bg, #f0f0f0);
        color: var(--text);
        display: grid;
        gap: 0.5rem;
        grid-template-columns: 1fr 24px auto 1fr;
        align-items: start;
        padding: 1rem 0.5rem;
      }
      :host([hidden]) {
        display: none;
      }
      :host([theme="danger"]) {
```

```
:host([theme="danger"]) {
  --bg: red!important;
  --text: white!important;
}
:host([theme="warn"]) {
  --bg: gold!important;
  --text: black!important;
}
svg {
  grid-column: 2 / 3;
}
button {
  grid-column: 4;
  grid-row: 1;
  justify-self: end;
  margin-right: 8px;
}
::slotted(*) {
  grid-column: 3 / 4;
  margin: 0;
  max-width: 66ch;
  line-height: 1.6;
}
```

```
}
```

```
static get properties() {
  return {
    icon: { type: String },
    theme: { type: String },
  }
}
```

```
constructor() {
  super()
  this.setAttribute('role', 'alert')
  this.setAttribute('aria-live', 'polite')
}
```

```
_close() {
  this.hidden = true;
}
```

```
render() {
  return html`
    <svg xmlns="http://www.w3.org/2000/svg" style="display: none;" >
      <symbol id="note" viewBox="0 0 24 24" fill="none" stroke="currentColor">
        <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="..." />
      </symbol>
      <symbol id="warn" viewBox="0 0 24 24" fill="none" stroke="currentColor">
```

```

    this.innerHTML += tree;
  }

  render() {
    return html`
      <svg xmlns="http://www.w3.org/2000/svg" style="display: none;" >
        <symbol id="note" viewBox="0 0 24 24" fill="none" stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="..." />
        </symbol>
        <symbol id="warn" viewBox="0 0 24 24" fill="none" stroke="currentColor">
          <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="..." />
        </symbol>
      </svg>

      <svg part="icon" height="24" width="24"><use href="#${this.icon}"></use></svg>
      <slot></slot>
      <button part="button" @click="${this._close}">Close</button>
    `;
  }
}

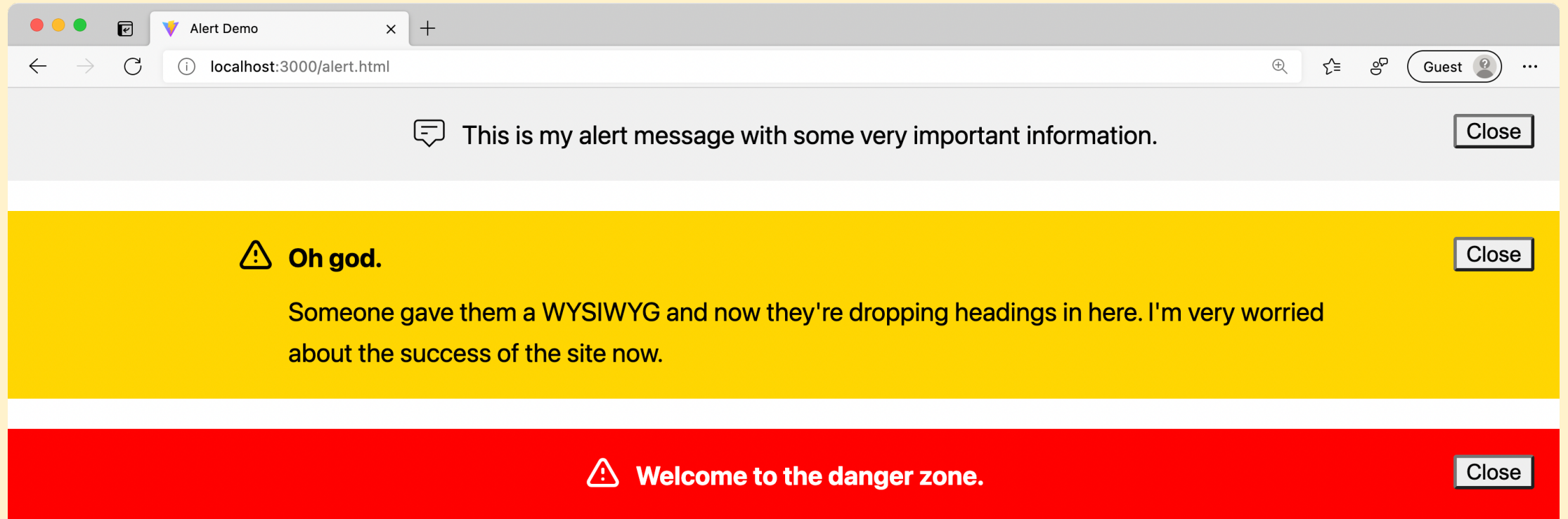
```

```

window.customElements.define('custom-alert', CustomAlert)

```

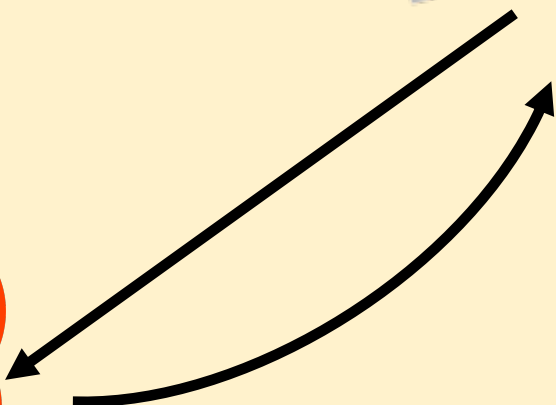
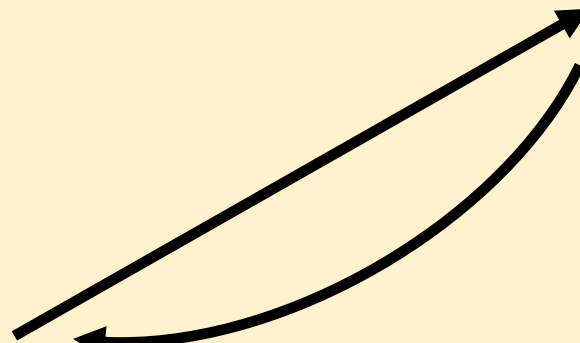
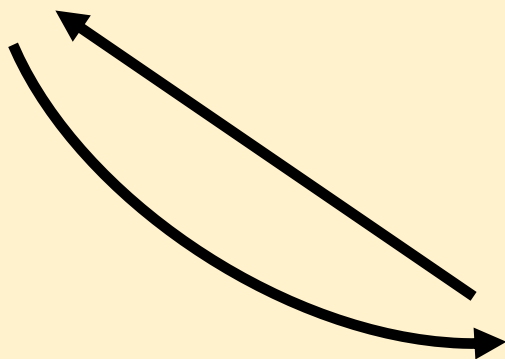
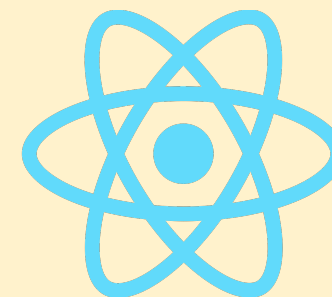

A basic example



The screenshot shows a web browser window with the title "Alert Demo" and the address bar displaying "localhost:3000/alert.html". The browser interface includes navigation buttons (back, forward, refresh), search, star, and share icons, and a user profile labeled "Guest".

Three alert messages are displayed in a vertical stack:

- Alert 1 (Light Gray):** Contains a speech bubble icon, the text "This is my alert message with some very important information.", and a "Close" button.
- Alert 2 (Yellow):** Contains a warning triangle icon, the text "Oh god.", and a "Close" button. The main text reads: "Someone gave them a WYSIWYG and now they're dropping headings in here. I'm very worried about the success of the site now."
- Alert 3 (Red):** Contains a warning triangle icon, the text "Welcome to the danger zone.", and a "Close" button.



Custom Elements Everywhere

Making sure frameworks and custom elements can be BFFs 🍺

LIBRARY

Angular 8.2.14

SCORE

100%

BASIC TESTS

16/16

ADVANCED TESTS

14/14

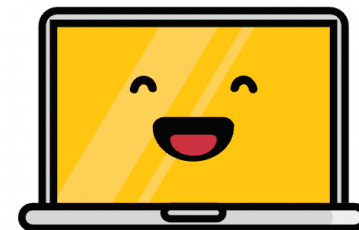
Handling data

Angular's default binding syntax will always set properties on an element. This works well for rich data, like objects and arrays, and also works well for primitive values so long as the Custom Element author has mapped any exposed attributes to corresponding properties.

Angular also provides binding syntax specifically for setting an attribute, if a developer would prefer to communicate with an element that way.

Handling events

Related Issues



Google

 Microsoft

IBM

 YouTube

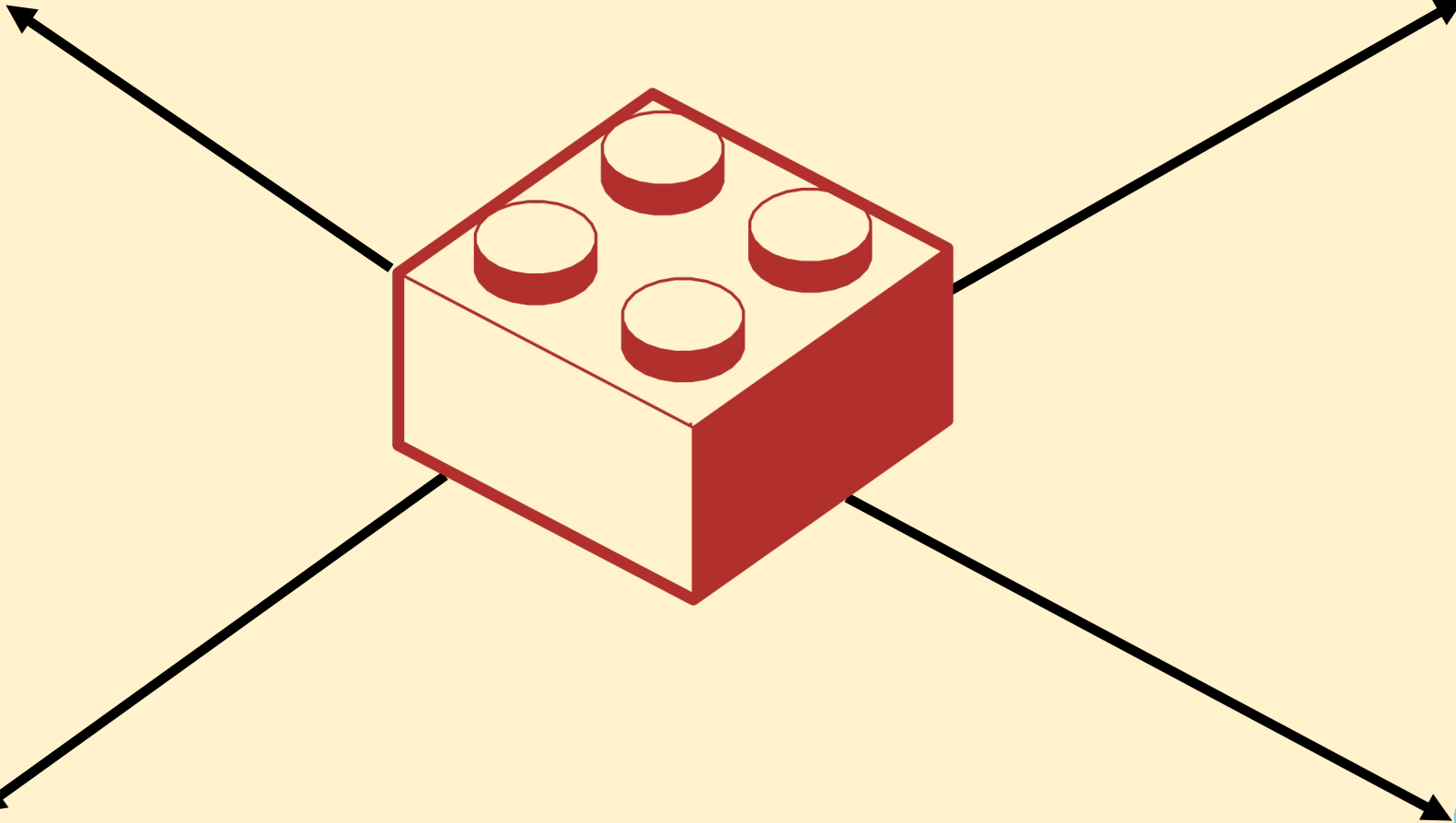
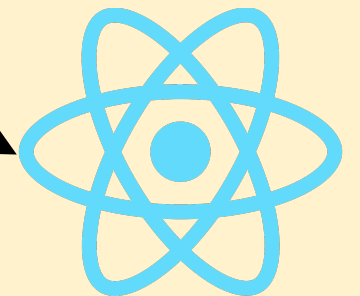
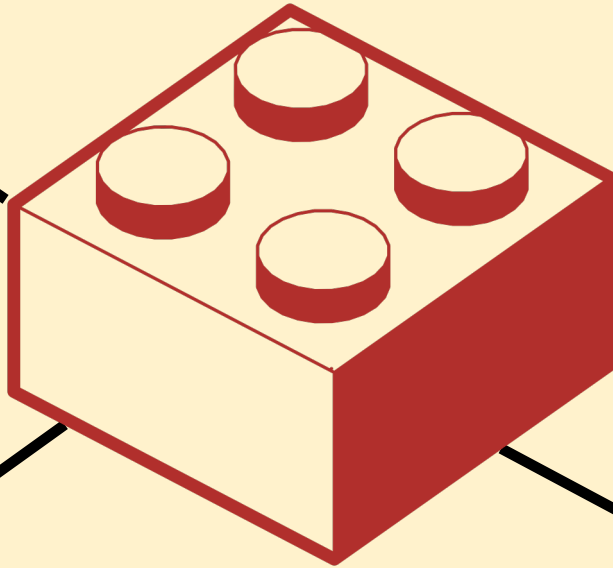
 GitHub

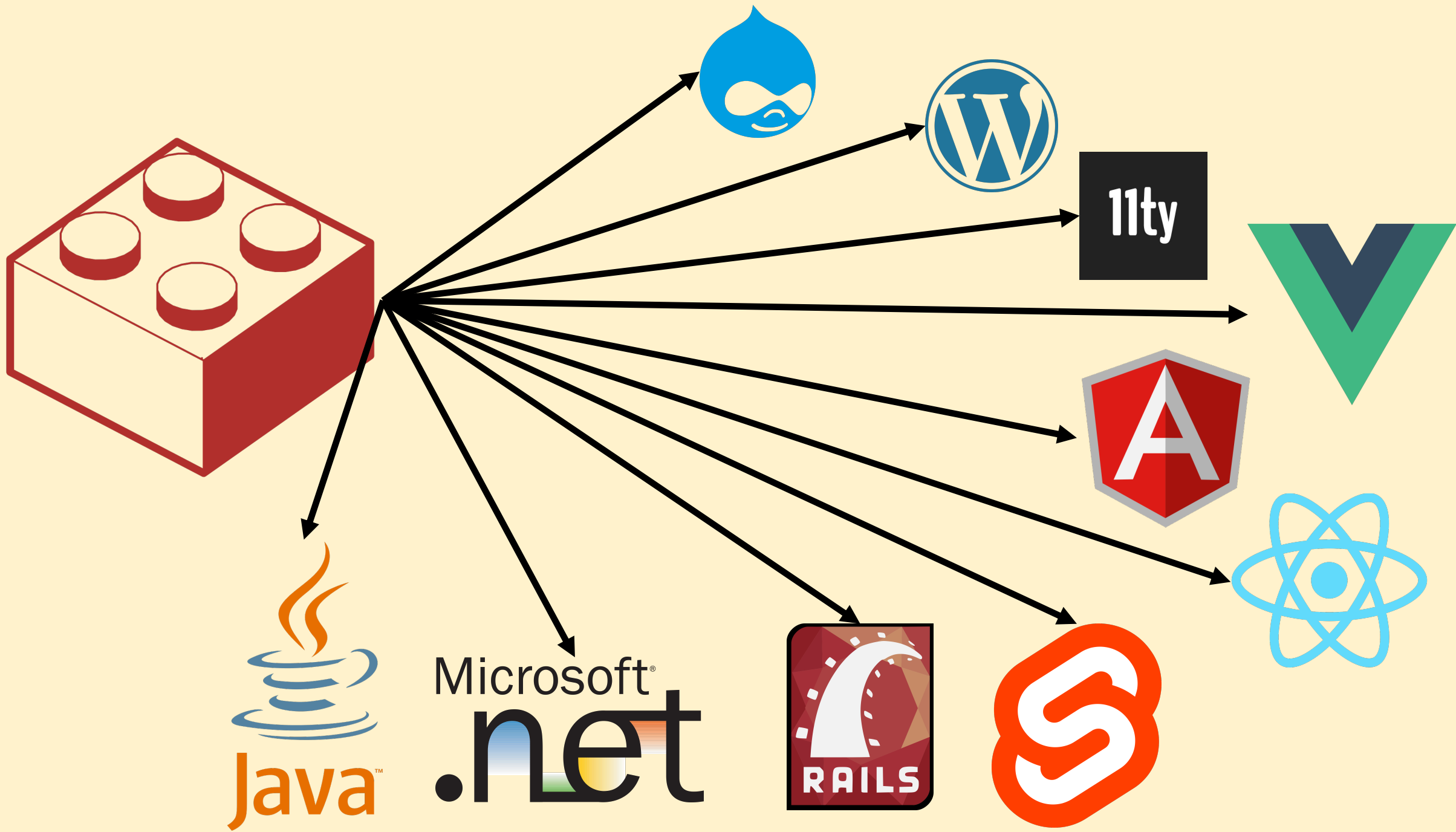
 Red Hat

 salesforce

ING 

 Adobe







FAQ

IE11?

GitHub - webcomponents/poly x +

https://github.com/webcomponents/polyfills

Why GitHub? Team Enterprise Explore Marketplace Pricing Search Sign in Sign up

webcomponents / polyfills Notifications Star 686 Fork 123

<> Code Issues 162 Pull requests 24 Actions Projects Security Insights

master 32 branches 50 tags Go to file Code

aomarks Merge pull request #434 from dpschen/patch-1 ✓ ce3c62f on Feb 23 3,845 commits

.github	Add format step to CI	2 months ago
packages	fix(shadydom): shadydom package url in README	last month
.editorconfig	Backport relevant parts of @webcomponents/platform	2 years ago
.eslintignore	Setup ignore-sync	2 months ago
.eslintignore-sync	Setup ignore-sync	2 months ago
.eslintrc.json	Merge branch 'master' into format	2 months ago
.gitattributes	Add @webcomponents/webcomponentsjs	2 years ago
.gitignore	Merge branch 'master' into custom-elements-scoped	4 months ago

About
Web Components Polyfills
Readme
BSD-3-Clause License

Releases
50 tags

Packages
No packages published

SEO?

The screenshot shows a web browser window with the address bar displaying `https://developers.google.com/search/docs/guides/javascript-seo-basics#web-components`. The page title is "Understand the JavaScript SEO". The browser's address bar also shows "Guest" and a search icon. The page content is from Google Search Central, with a navigation menu including "Documentation", "Support", "Blog", "What's new", "Events", and "Case studies". A search bar and "Sign in" link are also visible.

The main content area is titled "Follow best practices for web components". It explains that Googlebot supports web components and that when rendering a page, it flattens the shadow DOM and light DOM content. This means Googlebot can only see content visible in the rendered HTML. To ensure content is still visible after rendering, users should use the [Mobile-Friendly Test](#) or the [URL Inspection Tool](#) to check the rendered HTML.

If content is not visible in the rendered HTML, Googlebot won't be able to index it. The text provides an example of a web component that displays its light DOM content inside its shadow DOM. One way to ensure both light DOM and shadow DOM content is displayed in the rendered HTML is to use a [Slot](#) element.

```
<script>
  class MyComponent extends HTMLElement {
    constructor() {
      super();
      this.attachShadow({ mode: 'open' });
    }

    connectedCallback() {
      let p = document.createElement('p');
      p.innerHTML = 'Hello World, this is shadow DOM content. Here comes the light DOM: <slot></slot>';
      this.shadowRoot.appendChild(p);
    }
  }

```

Get started
Developer's guide to Search
Advanced guide to Search Console

Guidelines ▾

Control crawling and indexing ▲

- Overview of crawling and indexing topics
- Introduction to indexing
- Create a list of URLs
- Submit URLs to Google
- ▶ Manage your sitemaps
- ▶ Block access to your content
- Remove information from Google
- Pause your business online
- Consolidate duplicate URLs
- Create custom 404 pages
- Soft 404 errors
- ▶ Transfer, move, or migrate your site
- ▶ Manage international and multilingual sites
- Special tags that Google understands

Follow best practices for web components

Googlebot supports web components. When Googlebot renders a page, it flattens the shadow DOM and light DOM content. This means Googlebot can only see content that's visible in the rendered HTML. To make sure that Googlebot can still see your content after it's rendered, use the [Mobile-Friendly Test](#) or the [URL Inspection Tool](#) and look at the rendered HTML.

If the content isn't visible in the rendered HTML, Googlebot won't be able to index it.

The following example creates a web component that displays its light DOM content inside its shadow DOM. One way to make sure both light DOM and shadow DOM content is displayed in the rendered HTML is to use a [Slot](#) element.

```
<script>
  class MyComponent extends HTMLElement {
    constructor() {
      super();
      this.attachShadow({ mode: 'open' });
    }

    connectedCallback() {
      let p = document.createElement('p');
      p.innerHTML = 'Hello World, this is shadow DOM content. Here comes the light DOM: <slot></slot>';
      this.shadowRoot.appendChild(p);
    }
  }

```

Table of contents

- How Googlebot processes JavaScript
- Describe your page with unique titles and snippets
- Write compatible code
- Use meaningful HTTP status codes
 - Avoid soft 404 errors in single-page apps
- Use the History API instead of fragments
- Use meta robots tags carefully
- Use long-lived caching
- Use structured data
- [Follow best practices for web components](#)
- Fix images and lazy-loaded content
- Design for accessibility

A11y?

The screenshot shows a web browser with two tabs. The active tab is titled "Managing focus in the shadow" and the address bar shows the URL `https://nolanlawson.com/2021/02/13/managing-focus-in-the-shadow-dom/`. The browser interface includes navigation buttons, a search bar, and a user profile icon labeled "Guest".

The blog page has a header with the title "Read the Tea Leaves" and the subtitle "SOFTWARE AND OTHER DARK ARTS, BY NOLAN LAWSON". A search bar is located on the right side of the header.

The main content area features a dark green navigation bar with links for "home", "apps", "code", "talks", and "about". Below the navigation bar, the article title "Managing focus in the shadow DOM" is displayed in a large, bold font. To the left of the title is a red badge with the text "13 FEB". To the right of the title are two navigation links: "« Options for styling web components" and "JavaScript performance beyond bundle size »". Below the title, the post date and author information are shown: "Posted February 13, 2021 by Nolan Lawson in accessibility, Web. 26 Comments".

The article text begins with: "One of the trickiest things about the shadow DOM is that it subverts web developers' expectations about how the DOM works. In the normal rules of the game, `document.querySelectorAll('*')` grabs all the elements in the DOM. With the shadow DOM, though, it doesn't work that way: shadow elements are **encapsulated**."

The next paragraph states: "Other classic DOM APIs, such as `element.children` and `element.parentElement`, are similarly unable to traverse shadow boundaries. Instead, you have to use more esoteric APIs like `element.shadowRoot` and `getRootNode()`, which didn't exist before shadow DOM came onto the scene."

The final paragraph reads: "In practice, this means that a lot of JavaScript libraries designed for the pre-shadow DOM era might not work well if you're using web components. This comes up more often than you might think."

On the right side of the page, there is a "Recent Posts" section with a list of five items:

- JavaScript performance beyond bundle size
- Managing focus in the shadow DOM
- Options for styling web components
- 2020 book review
- Programmers are bad at managing state

Below the "Recent Posts" section is an "About Me" section featuring a photo of Nolan Lawson, a man with a beard and a black beanie, making a peace sign.

tl;dr

- Custom Elements around form controls have some gotchas.
 - Getting better with `ElementInternals` and `formData`
- Focus management is tricky, like always.
 - `inert` may be a hero here.

What about my favorite things?

- State management?
- Server-side rendering?
- Props?
- Hot Module Replacement?

**A few big ideas
before we go...**

**Web Components build
into the Web Platform**

**If we have good, accessible
Custom Elements...**

We have a pathway forward for more native elements

<tabs>, <accordion>, <model>, <popup>, etc.

An easier Web lowers barriers

Less complexity, less gatekeeping, a web for everyone.

Thanks

Dave Rupert @davatron5000

