back

November 4, 2024 – @hawkticehurst

CSS Web Components for marketing sites

Hot take: I think “regular” web components (the ones with Shadow DOM and friends) are a terrible solution for marketing website design systems.

It has always left a bad taste in my mouth when I run across a web component for a swimlane, banner, card, and so on. Why? Because these are components that (unless you’re doing something mighty fancy) should never require JavaScript as a dependency.

But, in the world of web components you are locked into JavaScript from the very start. To even register a web component with the browser you need JavaScript.

But what if… we didn’t do that?

HTML Web Components

I’ve spent a good chunk of the last year focused on marketing site design systems at work. A regular topic of discussion is the need to build marketing sites that are accessible to folks with lower powered devices and poor internet connections. How do you achieve that? In short, use less JavaScript and ideally build UI with progressive enhancement in mind.

There are many ways to achieve these goals, but the method I’ve been focused on is how an HTML Web Component archictecture might be applied to implement a marketing site design system.

As a quick reminder/intro, HTML Web Components is a method of building web components where you write HTML as you would normally and then wrap the parts you want to be interactive using a custom element.

For example, if you wanted to create a counter button it would look like this:

```
<counter-button>
  <button>Count is <span>0</span></button>
</counter-button>
```

```
<style>
  counter-button button {
    /* Counter button styles */
  }
</style>

<script>
  class Counter extends HTMLElement {
    // Counter button behavior
  }
  customElements.define("counter-button", Counter);
</script>
```

The markup in an HTML web component is parsed, rendered, and styled as normal HTML. That HTML will then be seamlessly hydrated once the JavaScript associated with the custom element tag is executed.

> In contrast, the markup of a "regular" web component (that uses Shadow DOM) is dynamically generated at runtime using JavaScript -- kind of like an SPA.

This component architecture is a really strong candidate for a marketing design system (and, as a bonus, avoids some of the big gotchas that come with regular web components).

- It is a perfect implementation of progressively enhanced UI
- It uses minimal and self-contained JavaScript — HTML Web Components can be thought of as islands
- You still get the power of custom element APIs to implement stuff like design system component variants
- The component markup is fully SSR-able
- The component markup can be styled like regular HTML
- Common accessibility practices can be applied without issue

But for all these benefits we're still left with the original problem. HTML Web Components require JavaScript.

CSS Web Components

So here's the question: What would happen if we took the ideas of HTML Web Components and skipped all the JavaScript?

You get CSS Web Components.

Note: I've never seen anyone talk about or name this concept before, so I'm using "CSS Web Components" to describe the idea. But please let me know if someone has already written about and named this!

How do they work? The exact same as HTML Web Components but you just take advantage of the powers of CSS to implement key functionality.

As an example let's implement that swimlane component:

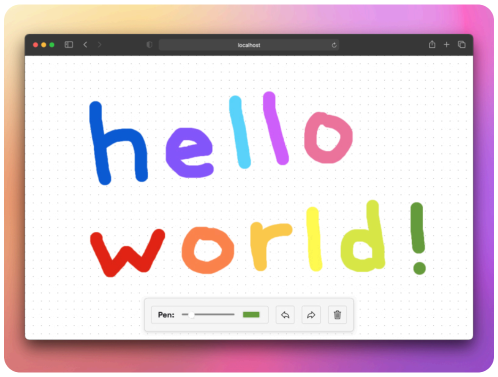
```
<swim-lane>
  <section>
    <h2>Creativity unleashed</h2>
    <p>A brand new way of illustrating for the web.</p>
    <a href="/product">Learn more</a>
  </section>
  <img src="product.jpg" alt="Product image" />
</swim-lane>

<style>
  swim-lane {
    display: flex;
    align-items: center;
    gap: 2rem;
    color: white;
    background: black;
    padding: 1rem;
    border-radius: 16px;
  }
  swim-lane h2 {
    /* Swimlane title styles */
  }
  swim-lane p {
    /* Swimlane description styles */
  }
  swim-lane a {
    /* Swimlane link styles */
  }
  @media (max-width: 650px) {
    /* Mobile responsive styles */
  }
</style>
```

Creativity unleashed

A brand new way of illustrating for the web.

Learn more



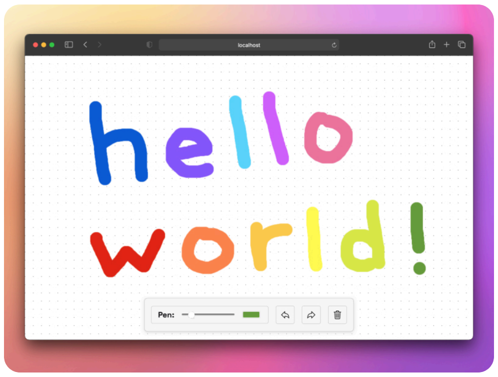
Okay great, we styled some HTML nested inside a custom element. There's nothing too novel about that. But what about adding some functionality? Say, a component variant that lets you reverse the layout of the swimlane?

It's possible using only CSS! Specifically, CSS attribute selectors.

```html
<swim-lane layout="reverse">
  <section>
    <h2>Creativity unleashed</h2>
    <p>A brand new way of illustrating for the web.</p>
    <a href="/product">Learn more</a>
  </section>
  <img src="product.jpg" alt="Product image" />
</swim-lane>

<style>
  /* Other swimlane styles */
  swim-lane[layout="reverse"] {
    flex-direction: row-reverse;
  }
  @media (max-width: 650px) {
    swim-lane[layout="reverse"] {
      flex-direction: column-reverse;
    }
  }
</style>
```

back



Creativity unleashed

A brand new way of illustrating for the web.

Learn more

Another really cool perk of this is that because you're defining an attribute on a custom element you don't have to worry about naming collisions with HTML attributes. No need to add `data-` to the beginning of attributes like you would/should on normal HTML elements.

How far does this go?

In theory, I believe this method of building design systems can go quite far. If you think about it, the vast majority of basic components you might need in a marketing design system are just vanilla HTML elements with specific style variations.

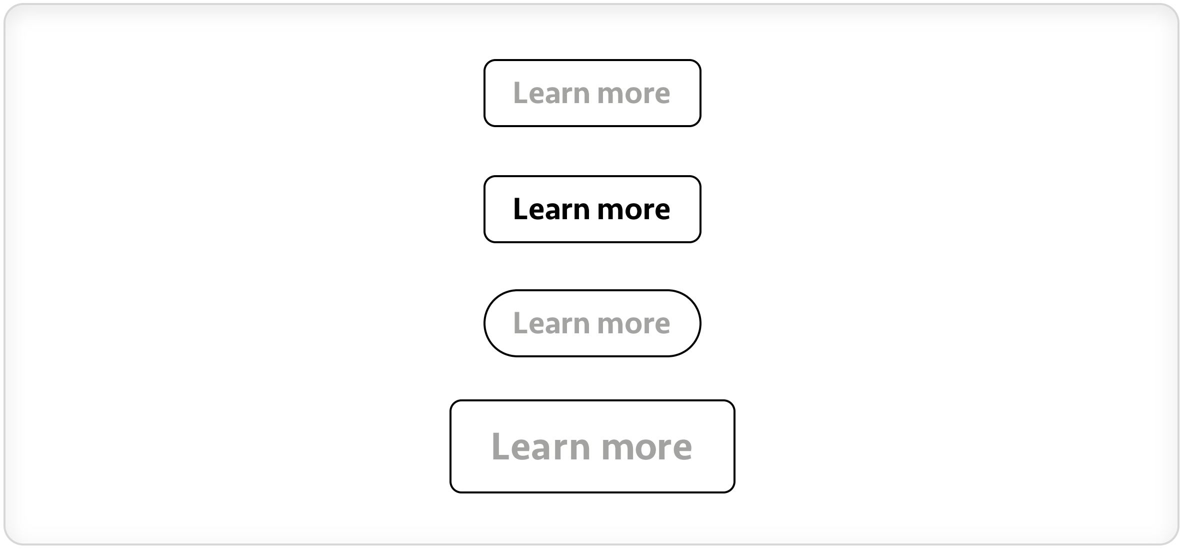
A marketing website button is just an anchor tag wrapped in a `<link-button>` custom element and styled using custom attribute selectors.

```
<link-button>
  <a href="">Learn more</a>
</link-button>
<link-button variant="secondary">
  <a href="">Learn more</a>
</link-button>
<link-button pill>
  <a href="">Learn more</a>
</link-button>
<link-button size="large">
  <a href="">Learn more</a>
</link-button>

<style>
  link-button a {
    /* Default link button styles */
  }
  link-button[variant="secondary"] a {
```

back

```
    background: transparent;
    color: white;
  }
  link-button[pill] a {
    border-radius: 50px;
  }
  link-button[size="large"] a {
    padding: 10px 20px;
    font-size: 1.25rem;
  }
</style>
```



From here, imagine incorporating all the other powers that CSS (and HTML) bring to the table:

- Cascade layers for better control of how styles get applied
- Container queries for conditional style variants based on a parent container
- :has(), :is(), and :where() to simplify complex selectors
- CSS variables for theming
- @property rule for even more powerful CSS variables
- light-dark() for system aware theming
- Popover API for menus, toggletips, and dialogs without JS
- Exclusive accordions for FAQ sections

The possibilities are quite large.

What do you think?