Out of the Shadows and into the Light

Making custom Web Components reusable, Front-end developer friendly, SEO friendly and Accessible

Lisa Harrison Ridley
DrupalCamp Florida 2025

Who am I?

Lisa Ridley, Chattanooga Tennessee

18 yrs web development
32 yrs project management
Acquia Certified Developer (D7)
Certified Scrum Master
Certified Scrum Product Owner
Certified in Human Centered Design

Smack-talking SEC Football Fanatic
Tennessee Vols fan
NASCAR Fanatic
Gardener and beginning homesteader

What are Web Components?

Web Components

Web Components are a standard way to create reusable, encapsulated, modular HTML elements without using a JavaScript framework.

Web Components are a Set of Standards

- Custom HTML Elements
- HTML Templates
- Shadow DOM
- ES Modules

Custom HTML Elements

An HTML element (a tag) whose behavior is defined by the developer using a combination of javascript, HTML Templates and styling, and/or Shadow DOM, all encapsulated as an ES Module.

HTML Templates

A mechanism for holding HTML fragments which can be used:

- Later via Javascript, or
- Immediately in the Shadow DOM

HTML Templates are extremely useful for generating **reusable** structured markup.

Shadow DOM

A hidden DOM tree that can be attached to elements in the regular DOM tree (Shadow Host).

- Shadow Host – The regular DOM node where the Shadow DOM is attached. (This is also referred to as the **Light DOM**)
- Shadow Tree – the DOM tree inside the Shadow DOM
- Shadow Root – the root node of the Shadow tree
- Shadow Boundary – the place where the Shadow DOM ends and the regular DOM begins

Shadow DOM: Friend or Foe?

Shadow DOM

Can be a double-edged sword:

- Shadow DOM is encapsulated, which means the HTML element can’t be touched or influenced by whatever is outside of the element

What does this mean, practically speaking?

Shadow DOM: Encapsulation

HTML Elements existing in the Shadow DOM can’t be accidentally styled, and HTML element styling won’t mess with your global styles.

So…

You can’t style elements in the Shadow DOM from your global stylesheets (without proper planning and structuring).

This will driver your front end developer team insane.

Shadow DOM: JS only vs Declarative

JS only: template and styling resides completely within the JS for the HTML element. This is entirely rendered client side.

Declarative: template and styling is part of the Light DOM markup, i.e., can be rendered server side.

- **Pro**: Declarative markup styling is more accessible to front end developers
- **Con**: Markup suddenly gets alot messier, IE doesn't support Declarative Shadow DOM

Shadow DOM: Declarative Shadow DOM

Can be used to build a functional base element that can be progressively enhanced.

- Base element functions without the use of Javascript
  - Server side rendered
- Progressively enhanced functionality is applied with the javascript of the web component
  - Client side rendered
- Build basic accessibility into your Declarative Shadow DOM

Shadow DOM encapsulation can be penetrated – styling

Instead of hard-coding text styling (colors, font-styles, etc) into your component, leverage CSS variables for styling, and apply default values inside the web component's style rules.

Global stylesheets can override web component styles by changing the value of the variables at the global level, and the CSS variable values will be inherited inside the custom HTML element.

Shadow DOM: Accessibility

Same standard web rules apply:

- If your custom web element is an interactive control, make sure it's focusable, keyboard accessible, and properly leverages styles for `focus`, `hover` and `active`.

Shadow DOM: Don’t forget your ARIA

Creating custom web components, especially control elements, means you have to be aware of the steps you need to take to make the markup accessible, i.e.:

- Use accessibility principles (aria attributes, generally accessible HTML)
- Build accessibility components into your Shadow DOM templates
- Explicitly leverage aria states and properties (`aria-expanded`, `aria-multiline`, etc) on web controls.

Shadow DOM ::part pseudo-class

By leveraging the “part” pseudo-class attribute of your element you can utilize that to target styling elements within the Shadow DOM of your web components.

- Best used for dynamically applied styling that occurs on events (click, tab, etc)

Why use Web Components?

Why would I want to use web components?

- If you work on a single site or multiple sites on a single platform, web components may add a layer of complexity that you may not need.
- If you work on multiple platforms (Drupal, Wordpress, AEM, mobile apps, etc) and need to maintain branding, consistency in UI behaviors, etc. between platforms, web components can solve that issue effectively.

For more information

References

Mozilla Web Docs:
https://developer.mozilla.org/en-US/docs/Web/API/Web_components
Steve Griffith Youtube series (good starter series on fundamentals):
https://www.youtube.com/playlist?list=PLyuRouwmQCjnENQk6NJlckZRXOfQP0x5B
Chris Ferdinandi Youtube series (provides context on how to build components differently):
https://www.youtube.com/playlist?list=PLYDkABV2cq82GTLIH2g93w5uie-IyOmHv
Lion (white label open source web components library maintained by ING Bank):
https://lion.js.org/

References (cont.)

Lit Framework:
https://lit.dev/, https://github.com/lit/lit/, https://www.youtube.com/@buildWithLit
SkillBakery Studio:
https://www.youtube.com/playlist?list=PLnIJ02CcZiYBrMTWjDrgblDHL9chUhNtN

Contact Information for Lisa

Drupal Slack: @Lisa Ridley, soon to be @Lisa Rae
Drupal.org: drupal.org/u/lisa.rae (formerly drupal.org/u/lhridley)
Github: lhridley
Email: lisa@codementality.com