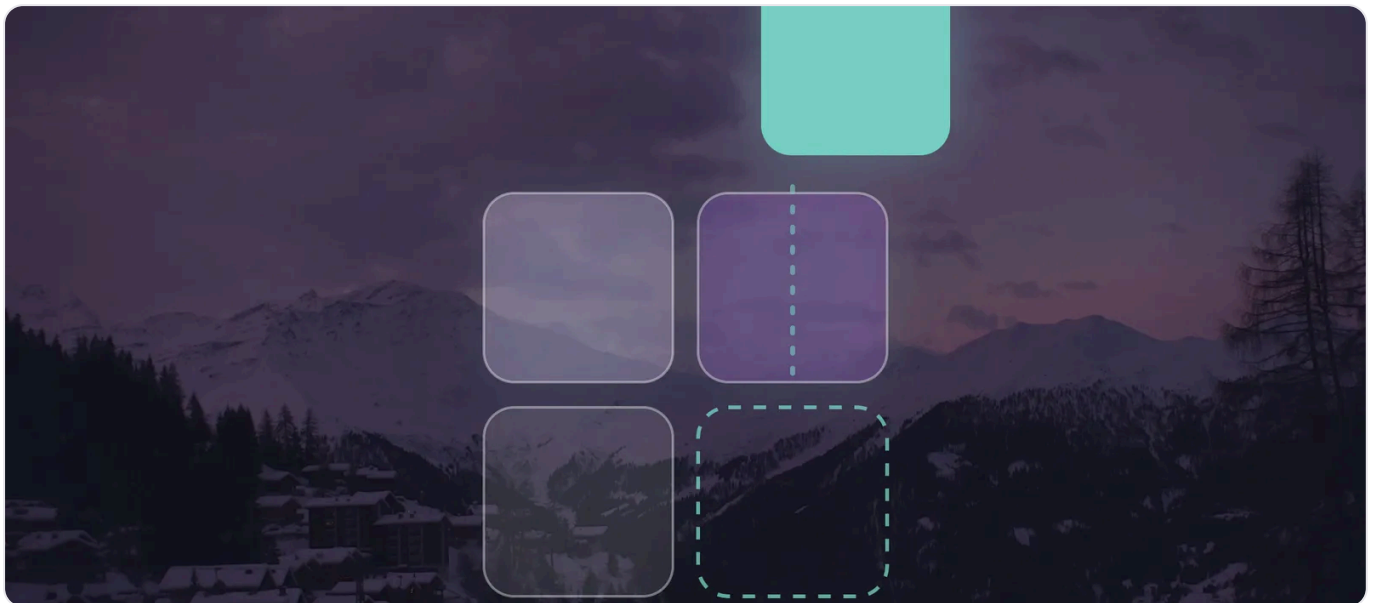


BLOG

Our Journey with Web Components

The story of our decision to build our frontend on Web Components and how we got it done.

Sensei Labs August 4, 2020



Our platform, [Conductor](#), originated from a group of engineers using their skills and spare time to make the lives of their colleagues better. This has created a culture where we value technology as a means of solving problems and where we aren't afraid to try new things if we believe it will provide the best experience for our customers or give us a technological advantage over competitors. A few years ago, we bet on an emerging browser technology called Web Components that has since gained more adoption and huge growth in browser and developer's community support.

What are web components?

Since the advent of JavaScript, many new ways of building a web page have been created in the form of abstraction layers called frameworks. All these abstraction layers were created in an attempt to improve developers' experience and increase performance. Web components are a set of web standards that aim to provide better browser capabilities to possibly replace the need for a framework and allow developers to create reusable encapsulated building blocks known as components. Those components then can be combined to build web pages and applications.

The four pillars for web components are:

Custom Elements

Web standards have a set of elements that we use to create our web pages. Those elements usually provide minimal functionality and it is up to developers to compose them to get something meaningful. Examples of these elements are: div, span, form, input, button, table, etc.

Custom elements allow us to declare a new element and define a custom behaviour for that element.

Shadow DOM

One important aspect of building components is that they should be able to provide functionality without depending or interfering with their surroundings, and Shadow DOM provides exactly that. It allows developers to encapsulate some styles and events within the element so the component becomes isolated from its context. Since the initial Shadow DOM specification, lots of improvements have been made around theming to allow consumers of these components to customize them.

Template

Template elements can be used to create components' markup efficiently. All the contents of a template tag are not rendered by the browser, but are available at run time to be used in new components.

ES Modules

This web standard defines how these components can be reused across different locations. It essentially defines a package that can be imported by other scripts that may need the component.

Why we chose web components

Back in early 2018 we started to develop our design system components and we had to choose a tool to build them. We considered many different popular options, such as React, Vue, and Angular. But over the years our team of experienced engineers have seen the rise and fall of many different frameworks and we were concerned about introducing another framework into our platform. We were also looking for the best developer experience with the smallest footprint possible.

Although all these frameworks provided good developer experience, good community support, and most of them were at their peak of popularity, they wouldn't integrate easily with our current stack. That led us to look for a native solution that would be framework agnostic. Web components not only provided the seamless integration that we were looking for, but also gave us a major technological advantage in terms of performance and life span of this solution. Since then, many other big companies have adopted web components as their primary solution for developing components.

Developer experience was also fundamental in our decision, we wanted our developers to feel comfortable in learning and adapting to use this technology, and since then the feedback has been really positive in regards developing web components. Here is what one of our talented engineers has to say about his experience using web components:

“Transitioning to working with web components and LitElement specifically was fairly enjoyable, as they function a lot like other modern component-based frameworks but without the hurdles of learning many library-specific quirks (e.g. being forced into doing things “the angular way” or having to relearn a fundamentally new concept like React Hooks after years of working with lifecycle methods). I also find it satisfying to work with vanilla JS and to not rely too much on library methods, and working with web components you get a bit of both.”

Josh Coles, Frontend Engineer

Lessons learned and future developments

Our initial development of web components began with vanilla javascript, we managed and maintained a lot of boilerplate code and developed a basic rendering engine. We discovered a library called LitElement which improved many of the features that we were maintaining ourselves, so we decided to start building new components using this library. That decision boosted the development of new components not only as part of our design system, but also in other parts of our application.

After over 2 years of working with web components, we are confident that it was the right decision for our business needs and we continue to refactor older parts of our code base with web components. In some pages where we were able to completely refactor using web components, we saw performance gains of up to 90% in initial load/render times.

Resources to learn more

- **Intro to Web Components** — <https://blog.usejournal.com/web-components-will-replace-your-frontend-framework-3b17a580831c>
- **LitElement** — <https://lit-element.polymer-project.org/guide>
- **Components Library** — <https://www.webcomponents.org/introduction>

READ ONLINE

Read this article on our website

getconductor.com/resources/our-journey-with-web-components



GET STARTED

Talk to sales

Walk through an example with our team and see Conductor on your own programs.

Talk to Sales →



Scan to book