1. Web components (WC)
2. WC - why?
3. About my WC library

1. Where did the WC come from? As we mentioned yesterday WCs have actually been around for quite a while, they were first proposed in 2011. At that time everyone was writing JQuery plugins, nobody cared about components. So it was quite a while before they really gained proper browser support it was only last year. That Firefox and our support. EDGE has actually only just getting support as they switch to using blink as their engine. But so what is changed since 2011? Mostly these days when we are writing code we write it with components. We use frameworks and frameworks are built out of components. So everybody is used to add you have components now, all think that there are a pretty good idea. The question though is if we have got these frameworks then why do

we need native components in the browser. We have already got components in our frameworks or to put it another way in framework components and WCs with a fight. Who would win?

2. A few arguments on both sides. Framework components give you a bunch of stuff you do not get from WCs for instance. They give you an easy way to bind your JS model to your HTML view through like JSX templates and Angular that kind of thing. A lot of them give you change detection for instance. Agular gives you automatic change detection through zone JS and Vue does something similar you native JS get as incentives. They give you things above the level of component like state management and routing. On the other side WCs are a lot more lightweight right all of this stuff that your framework or library. I guess in the case of React gives you it does not come for free there is a cost to having all of that managed

for you. Were is WCs you are just shipping the JS that you are actually using for your component. The other thing is WCs are flexible. You can mix and match them. You can use them with your frameworks or not or with whatever. You can use WCs with your CMS. If you do not have a framework but you still want to have reusable components so you can use them there.

The answer to who would win is that - we do not need to fight! What kind of situation you want to use WCs. In the first is if you do not have a framework. Maybe you are building a simple site that just does not require a framework but you still want to have reusable components. Maybe you are got multiple site that you want to share components between? They are written in different frameworks. Maybe you have got a React out and an Angular out by they are still under the same brand and feel but you do not want to replicate all that code. WCs is good for that.

Let us have a look at WCs are actually separate a specs. The first one is ES modules. Second is HTML templates give you a way to create like reusable fragments of HTML. Custom elements and Shadow DOM. That is a way to create new HTML elements and encapsulate that DOM.

There are a bunch of libraries available to you.

3. Now I want to talk about Stencil.

Stencil compiler produces vanilla JavaScript, without any dependencies.

Stencil is a simple compiler for generating Web Components and progressive web apps. Stencil was built by the Ionic Framework team for its next generation of performant mobile and desktop Web Components.

Stencil combines the best concepts of the most popular frontend frameworks into a compile-time rather than run-time tool. It takes TypeScript, JSX, a tiny virtual DOM layer, efficient one-way data binding, an asynchronous rendering pipeline (similar to React Fiber), and lazy-loading out of the box, and generates 100% standards-based Web Components that run on both modern browsers and legacy browsers back to Internet Explorer 11.

Stencil components run directly in the browser through script include just like normal Custom Elements (because they are just that!), and run by using the tag just like any other HTML component.

I hope you enjoyed this article!