Documenting what we support

In reading Jeremy Wagner's book Responsible Javascript from A Book Apart, I came across the concept of a technology statement: What's your technology stack, what browsers you support and what technologies you don't and, particularly in the technologies you choose not to support, provide a clear explanation of why you choose not to support them...

In the book Jeremy points out the A11Y project and their technology statement as a model to follow, specifically (emphasis mine):

> The A11Y Project is a comprehensive resource on web accessibility, and this excerpt of their technology statement doesn't mince words about their technology preferences:
>
> > This is a deliberate choice intended to balance:
> >
> > > - Accessibility and interoperability. Ease of setup.
> > > - Approachability for beginners.
> > > - Cross-Operating System and browser support.
> > > - Local and end-user performance.
> >
> > When working on the website, please be sure to utilize these technology choices first, and stay with them if at all possible. Honoring these technology choices helps to keep the site easier to maintain. (http://bkaprt.com/rjs39/02-09)
>
> **Their statement lists not only the technologies used but also ones to be avoided, and explains why those technologies are incompatible "with the A11Y Project's mission to provide an accessible and fast resource on web accessibility. This gives prospective contributors a roadmap to making contributions that align with the**

> **project's mission.**
>
> Responsible Javascript by Jeremy Wagner — A Book Apart

But when you I went to the website, I found the following statements that lists what they won't support but doesn't really tell you why. So I thought I'd point out my disagreement and areas of concern. All the quotes below are taken from the A11Y Project's Technology page:

> # Avoid
>
> These technologies were evaluated and purposefully not utilized. We are not interested in incorporating them into the project.
>
> ## Babel, webpack, etc
>
> These module bundlers are complicated to set up and maintain and not a good fit for a site of this size and complexity.

Yes, they are complicated to set up but you only do it once per version of your project. There is no need for each individual contributor to set the technologies up, particularly if you use Gulp as your Task Runner and build system.

I get it that they may not want to support it but using complexity is not really a good excuse when the result are fewer requests sent to the server and, potentially, smaller payloads. This is important even with HTTP/2. See Browser module loading - can we stop bundling yet?

I'll address their not wanting to support Babel when I talk about ES6 below.

> ## CSS Custom Properties
>
> We use Sass to control CSS-related variables to to maximize compatibility with older and non-standard browsers.

Custom properties serve a completely different purpose than SASS and other pre-processor variables.

They are live so they don't require recompilation whenever you make changes and make it easier to code sliders and other tools that change the colors or layout on a page.

Because of browser support, custom properties are prime candidates to progressive enhancement. If you test for support then you can get the best of both worlds, if you're willing to put the effort.

Furthermore the latest version of the SASS reference implementation supports CSS Custom properties alongside their traditional variables (https://sass-lang.com/documentation/style-rules/declarations#custom-properties)

While I can understand the need to support older browsers, I wish they would list what non-standard browsers the need to support and what other concessions they are making to support them.

> ## CSS-in-JS
>
> We consider this approach to CSS to be an industry antipattern.

It may be true that it's an anti-pattern but if it's widely used in the industry, shouldn't you consider looking at it from an accessibility standpoint?

> ## Docker and other containers
>
> We intentionally use a relatively limited set of technologies, so our need for codifying our environment to the degree a container grants is less of a concern. Additionally, the computational resources needed to run a container may make working on the site difficult for more lower-power devices.

Yes, you keep your stack small but it's still not trivial to set up the infrastructure to make it run. That's where Docker comes in.

It's not an either/or solution. If a developer doesn't have a powerful enough machine they don't have to use Docker but using that as an excuse to not even explore the option is troublesome to me.

> ## ECMAScript 6
>
> We use pre-ES6 JavaScript to maximize compatibility with older and non-standard browsers, as well as niche assistive technology.

This is the part that worries me the most as it reflects a level of laissez-faire that can be dangerous.

What non-standard browser doesn't support at least ES2015? What browser supports only a 10-year-old version of the ECMAScript specification? Because that's the best-case scenario of what the statement says. Heaven help the users if they need to go further back to ES3.

Same thing with niche accessibility technology. What tool needs ES5 to run?

As far as configuration it's a matter of coming to an agreement on what to support and how to do it. I have no doubt you'd be able to pick up a volunteer to run the pain of configuring Babel and using the configuration as part of a Gulp task.

You can use browserslists to tell Babel what plugins and transpilations to run for a given version of a browser. But you have to be willing to put the work to understand what the benefits are, not just the drawbacks.

I think this is an issue confusing ES6 with all the versions of Javascript that have been released since. It is still an issue of who supports non-evergreen browsers and what technology has stagnated to the point of depending on 10 year old software. We could come up with an agreement of supporting ES2017 and earlier and polyfilling any newer Javascript features

It is also compounded by the project's refusal to support babel, a tool that would transpile current 2021 code into something that older browsers could read.

> ## PostCSS
>
> We use Sass in favor of PostCSS not only to lessen dependencies, but to also streamline the amount of learning someone needs to do to work with our site's CSS.

One hard syntax is enough? You're already installing Gulp so the number of dependencies shouldn't be the driving factor.

If you take the time to configure it, PostCSS can work just as SASS does and do things like running tools like Autoprefixer that SASS on its own cannot do.

> ## React, Vue, and other Single Page Applications
>
> Our content needs do not require the benefits of a Single Page Application approach. In addition, these technology choices would artificially inflate the level of complexity to work on the site, as well as introduce significant barriers for assistive technology users and low-power devices.

So you shouldn't work to provide accessibility know-how and tools for these frameworks? and how would you provide accessibility information and best practices for frameworks if you don't use them and discourage people to do so?

How do they know that there are barriers for assisitve technology users and those on low-power devices? Have they tested the impact of SSR or using a framework for static site creation? Have they tested these sites in low-power devices?

I'm not one for supporting frameworks. THis is not what this criticism is about. It's about making claims that may or may not be supported by facts.

> ## YAML
>
> We previously used to use YAML for storing data, but now prefer JSON. The exception to this is for Eleventy and Markdown frontmatter.

Is there any reason for the change of preference that makes contributors have to remain aware of the two languages that you're foisting on them? Or is it because JSON is easier to lint than Yaml?

If the goal is to provide a good exeprience for first-time contributors this doesn't strike me as the way to do it.

So, what would a technical requirement document look like?

In addition to the technology stack, I would include a list of specific browser versions that our stack supports.

Be explicit. If you say that you don't support all the versions of Ecmascript then say why and go beyond the excuse that you do it for browser compatibility or for user experience.

Final Notes

Whenever I see restrictions like these in a project I question the reasoning, particularly when I don't see a clear rationale for them.

I don't necessarily disagree with some of those terms like CSS-in-JS but if you work with accessibility, you owe it to your users to at least talk about the reasons