Web Native - Building the resilient web - What ever happened to progressive enhancement?

Written by Seth Corker on Benevolent Bytes

The web is a remarkable platform because the *core building blocks* of the web are resilient. You'll learn this with your first line of HTML, if you get it wrong the browser takes its best guess. Misspell a CSS property, the browser will ignore it like nothing happened. Forget to declare a variable in JS, it won't bring the whole webpage crashing to a halt. We take this for granted, but not many languages are like this. If you're a programmer using a compiled language, you're expected to be precise and the compiler will let you know if and when you've done something wrong. Why is this important? We're breaking the resilience of the web unwittingly.

The web is resilient by default. As web developers, we can sometimes forget this. I work a lot with React in my job, and in almost all of my frontend side projects I rely on React. It makes working with components a breeze and gives the flexibility I want from a framework. The one issue is that React devours all, and it becomes a crutch to lean on. You may not realise until it's too late!

> I used to write CSS in separate files. I find myself writing CSS in React.
>
> We used to write HTML, I now write JSX.

This abstraction has benefits, but it often comes at the expense of resilience. What we lose is the ability to embrace the web as a platform and employ progressive enhancement.

What's progressive enhancement?

At its core, progressive enhancement asks us to start with HTML first. This means that if a user has disabled JavaScript (or we introduce a bug that breaks a script), they still get a nicely formatted document. They can still read the content – everything else is icing on the cake. You can then come in and add CSS for some better aesthetics and sprinkle on some JavaScript to breathe life into the webpage. Progressive enhancement is about embracing the variety of devices and browsers that a website can be viewed on. It's accepting that change is inevitable.

If a new feature comes out that you want to make use of, you can. Not everyone will see it, but that's okay. The beauty of the web is that you can use new CSS features and older browsers won't break, they'll just ignore it. You can use custom HTML elements, and older browsers that don't know what to do will simply render the body of the tag. This simple feature of the web is flexible and gives you the power to experiment.

How are we breaking the web?

If progressive enhancement asks us to start with HTML as our foundation, single-page applications (SPA) that only run on the client with JavaScript, are in direct competition with this principle. If JavaScript is disabled, you'll fallback to whatever the developer decided to include on the HTML page, which is usually nothing more than a way to bootstrap React or their framework of choice.

I use React all the time. It's become part of my toolset, a tool I reach for too often. When it comes to delivering powerful and rich interactive experiences, React handles it. Looking at progressively enhancement depending on the user's browser, that's a little more challenging. I'm not against React, it isn't really the problem, the real problem is the reliance on a single tool to do the job. When I started using frameworks like React, it was to add complex interactivity to existing pages. It was to add little islands of interactivity that involved drag and drop interactions, dynamic fetching of data and complex user flows. These were just that, islands in an otherwise static (or server rendered) page. As time has progressed, it began to take the place of HTML pages to the point where SPAs were all the rage. We gained the ability to add rich functionality everywhere, but it came at a cost.

Is React to blame?

Not entirely. React is a great library that helps developers build dynamic UIs. There are, however, a few key concerns for resilience when building a React app. Using a solution like Create React App will give you an app that is entirely bootstrapped in the browser. It relies on JavaScript running on the client. In fact, your app won't do anything useful until React and your code is downloaded, parsed and executed. This has become the norm. The use of SPAs for websites has increased, and thus JavaScript has become a de facto requirement.

Another problem that isn't just tied to React but can be found in React apps is the disregard for semantic elements, creating a div soup. There are semantic elements for a lot of use cases in HTML5. You can describe where navigation is, what content is important, where an article begins, etc. There's even a disclosure element

that shows a summary which can be expanded and contracted, all without JavaScript. By using these elements in your websites and apps, you leverage what the browser does by default.

Issues arise when developers, maybe for easier styling, or they just don't know, decide to use div and span for everything. This especially affects people who rely on accessibility affordances to browse and interact with the web. By not using elements like buttons, links, etc. properly, you're removing those built-in features everyone relies on.

> I've seen divs with `onClick` handlers that open a URL. The browser doesn't handle this case very well. A user clicking on the element will be navigated to a new page, but what about trying to open in a new tab? Nope. How about copying the link? Not supported either.

Oversights like this breaks core features of the web. So, what can we do about it?

How do I build a resilient website?

HTML **Start with the core**. The foundation of the web is where it all started, HTML. Ensure your markup is semantic and uses the elements that make sense in the context. If an element is clickable, use a button. If it navigates to a new page, use a link. It's tempting to want to avoid styling the elements that browser styles for you and instead opt for divs and spans, as they have minimal browser styles. **Don't do it**. Everything can be styled the way you want with CSS with very few exceptions.

Use semantic HTML wherever you can. This will provide the experience users expect of the web. If a user is on an older browser or an underpowered device, they'll still be able to see the content. If a user relies on navigation with keyboard, it should just work. Once you have a solid foundation to build on, you can move on to styling.

CSS CSS is as powerful as it is scary for some web developers. It provides flexibility to design an experience that looks good and offers affordances to users. Embrace responsive design and make use of sizes that adapt to the person's screen size. Use media queries to change the layout to adapt to different sizes, support light and dark mode and even reduce animations which would be uncomfortable to a person with vestibular motion disorder. The best part about both HTML and CSS is that you can layer on newer features whenever you like, and older browsers will simply ignore it. You're not going to completely break the website for older browsers, you're simply enhancing the experience for newer ones. The core of the site is still accessible. What do we do about the elephant in the room – what about JS?

JavaScript JavaScript is the most relied upon part of the web stack, and the least reliable. If you're using a UI library or framework like React, Vue, Svelte, etc. It's easy to write great apps that look great and feel snappy, but you should consider them in the context of progressive enhancement.

What will a person see if they have JavaScript disabled? But, who would have JavaScript disabled in 2022? If that seems a bit excessive, what about someone on slow public WiFi or a spotty 3G connection. How long will it take them to load your single page app? Some experiences heavily rely on JavaScript and that's enviable, you don't have to try to move mountains to provide a video editing experience on the web without JS, that's not feasible. If your SPA could easily be a website with small islands of interactivity, consider some ways of embracing progressive enhancement.

React, Vue and Svelte are great frameworks, they also have some very popular frameworks that offer server-side rendering (SSR) and static site generation (SSG) too Next.js, Nuxt.js and SvelteKit respectively. These frameworks will lay the foundation of HTML and CSS and enhance it with JS afterwards. You don't need to forgo modern tooling to make a resilient website, just consider building on a solid foundation.

Where can I learn more?

One of the best resources I recommend for insight into how we got here and how we should help build a more robust web is Resilient web design by Jeremy Keith. Another surprising resource is from the UK Government Service Manual – Building a resilient frontend using progressive enhancement, it details a practical approach to progressive enhancement.

Conclusion

Progressive enhancement is a key strategy in ensuring the web remains resilient and accessible to everyone. It's the foundation that allows people with the fastest internet connections and newest devices to benefit just as much as someone trying to browse your website on a 10-year-old tablet with a 3G connection. The key takeaway

is this. Starting with the most resilient layers of the web and build rich experiences on top of them as necessary. A website fully reliant on JavaScript can mean a more fragile experience that caters to a smaller audience.

Cover photo by Cinthia Aguilar on Unsplash