Artesian Web Development - How it feels to write a web page from scratch in 2020 - Shedding the burden of modern tooling

Written by Seth Corker on Benevolent Bytes

In a world of build tools and web frameworks, it's not often you just sit down and code a webpage from scratch. No CSS framework. No auto-prefixing. No webpack config or dependency manager. No intermediary steps involved. Not even a line of JavaScript. Just me, my editor, my browser and the raw materials of the modern web. It's been a serene experience that feels unfamiliar in 2020.

I'm a software engineer living and working in London. In my daily work my primary tools for the web are React, Next.js, GraphQL and Emotion for CSS. In my personal projects, I often play around with different technologies like Gatsby, Svelte and whatever else looks interesting at the time. I don't often get the chance to write plain HTML and CSS. It's not that I avoid it but it just never seems to be part of my workflow like it once was. I decided to take a step back and start a project with the core building blocks of the web, it's been a novel experience.

When you get used to something like React, it becomes comfortable, it becomes the de facto tool for the job. Need a new blog? Use Gatsby. Want a marketing site? I'll use Next.js. Need styling for a quick admin interface? Take your pick of CSS frameworks and/or component libraries. There's a plethora of choices, a tool for every job that all fits into the React ecosystem and more broadly, the ecosystem of the web. To ignore that for a moment, however brief, is a breath of fresh air. To forget all of it and go back to basics, to code like the early web, it's liberating.

So, why did I do it? I wanted to try out variable fonts and I didn't need any reactivity, I just wanted to tinker with a static webpage. That's why I avoided Create React App. I avoided all of the tools I've relied on over the past 5 years and just did it. This is what, I discovered along the way and a few reasons why you might want to give it a go every now and then.

Figure 1: Blue CSS3 logo beside thumbs up

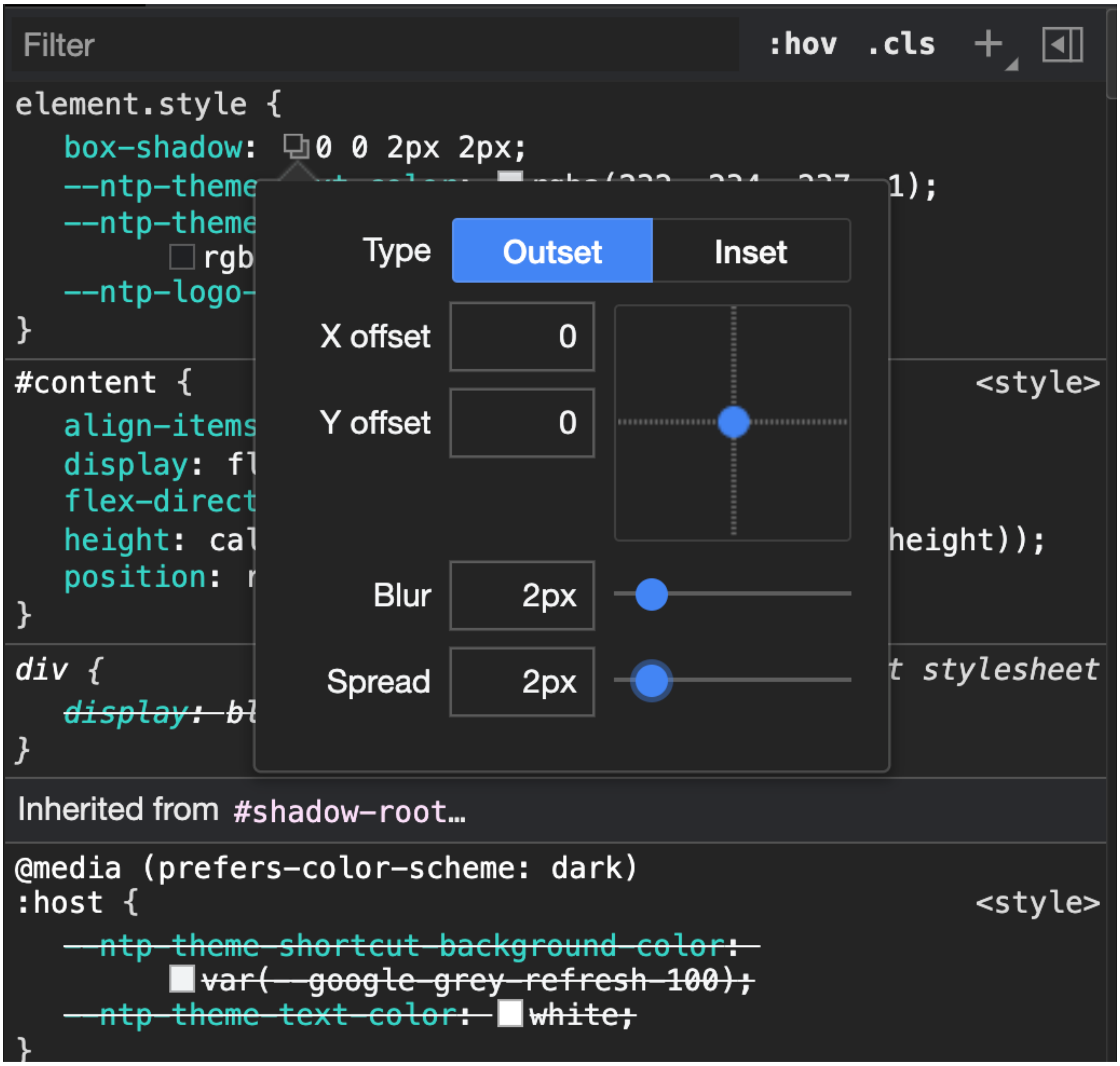
Hand crafted CSS

Before the explosion of JavaScript into the behemoth it is today, there were few things more contentious than CSS. A language that is loved and hated with equal measure. There have been many systems created to make CSS easier to wield like BEM, SMACSS and OOCSS. There are preprocessors that aim to add features and fix the flaws like LESS and SASS. Now with the rise of React, there are countless ways of including CSS in your app from simple imports to CSS-in-JS solutions.

I left this all behind to hand-craft some CSS. Here's what I learned.

Dev tools are wondrous

Working in the browser's dev tools is great. The developer tools from Firefox, Chrome and Safari are excellent. The best part is that they just come with browser. There's nothing to find, add and configure unless you really want to. CSS is one of the primitives of the web and dev tools can make working with it a dream. My favourite feature is the ability to live edit your CSS with workspaces. Changes made in dev tools are synced with your project on your filesystem. It's almost like an evolution of Dreamweaver for the modern era. You can inspect gradients, create box shadows without trying to remember which order values should be in and all from within



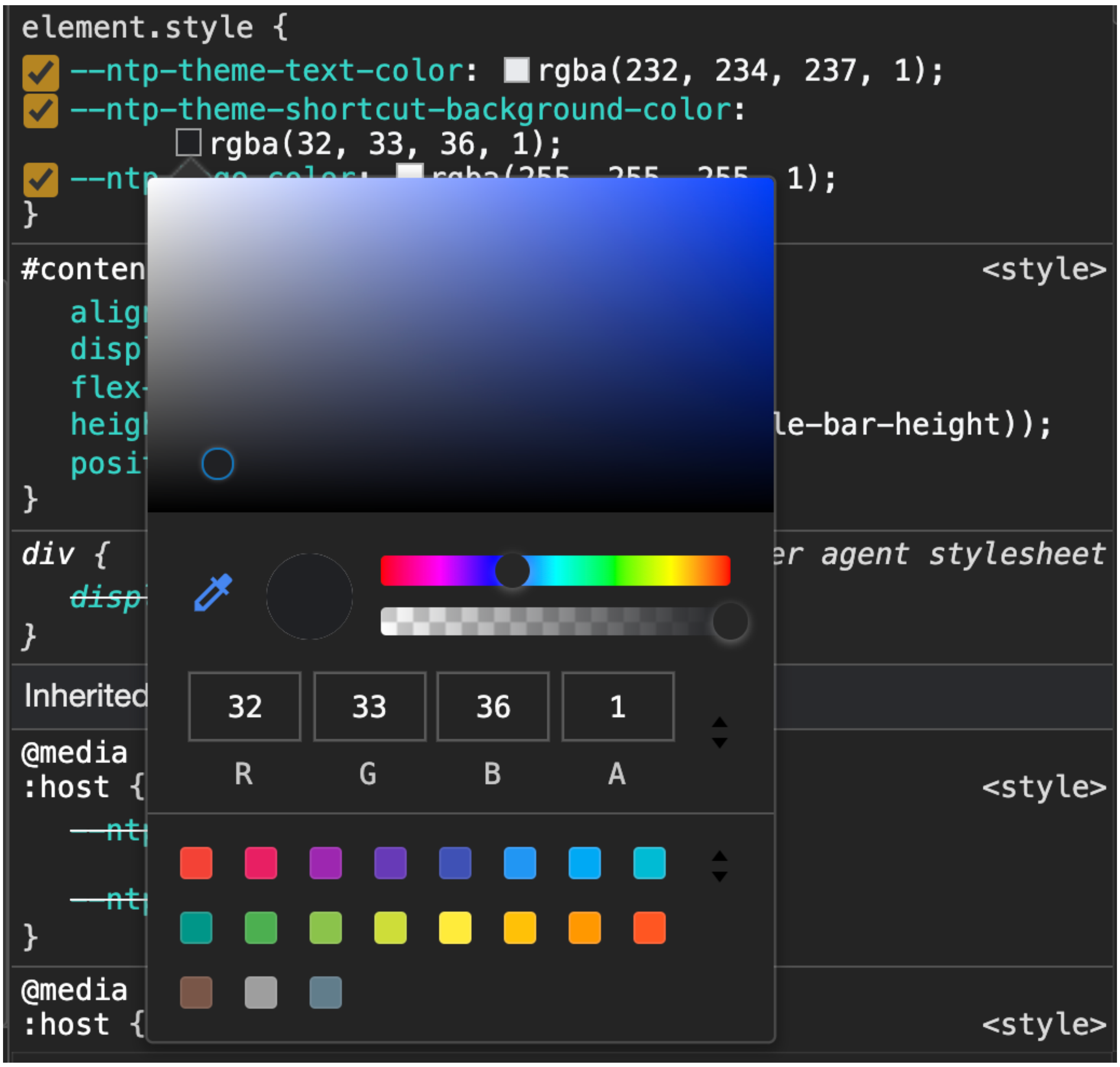
```
element.style {
    box-shadow: 0 0 2px 2px;
    --ntp-theme          1);
    --ntp-theme
        rgb
    --ntp-logo-
}
#content {                                           <style>
    align-items
    display: fl
    flex-direct
    height: cal                               height));
    position: r
}
div {                                      t stylesheet
    ~~display: bl~~
}
```

Type	[✓ Outset]	Inset
X offset	0	
Y offset	0	
Blur	2px	
Spread	2px	

Inherited from #shadow-root…

```
@media (prefers-color-scheme: dark)
:host {                                              <style>
    ~~--ntp-theme-shortcut-background-color:~~
        ~~var(--google-grey-refresh-100);~~
    ~~--ntp-theme-text-color: white;~~
}
```

Figure 2: screenshot of editing box shadow in dev tools



```
element.style {
  --ntp-theme-text-color: rgba(232, 234, 237, 1);
  --ntp-theme-shortcut-background-color:
        rgba(32, 33, 36, 1);
  --ntp                                          1);
}
#conten                                            <style>
    align
    disp
    flex
    heig                                  le-bar-height));
    posi
}
div {                                 er agent stylesheet
    disp
}
Inherited
@media
:host {                                            <style>
    --nt

    --nt
}
@media
:host {                                            <style>
```

```
32    33    36    1
R     G     B     A
```

Figure 3: screenshot of colour picker in dev tools

the browser. It's the ultimate WYSIWYG, there's no guessing what your page will look like - you're editing it live, zero delay.

Vendor prefixes are becoming less relevant

One downside of editing your CSS by hand is the lack of auto-prefixing. With every year and every new release of browsers, this is becoming less relevant. Browser support in a much better place than it was a few years back and the majority of what you need to do is supported. Jumping through hoops to get a nice developer experience with modern tooling feel antiquated when a great solution already exists. It might not meet all the needs of browser compatibility out of the box, but the experience is great and the feedback, instantaneous.

Figure 4: Orange HTML5 logo beside hand gesturing small

Artisanal HTML

I've been writing React components for over 5 years now. Before that, I used to write a lot of HTML, a lot of CSS and a lot *less* JavaScript. In the past, there was often a templating language that generated HTML. Templates with Ruby on Rails, ASP.NET or Pug. There was never a need to write HTML by hand for the majority of products I've worked on. So what was it like?

Wait, how do I do this again?

It feels unfamiliar writing plain HTML again. Years of React has made me forget some of the differences between JSX and HTML. Writing `class` instead of `className` feels out of the ordinary. I had to delve into the recesses of my mind to remember the proper way to link a stylesheet; when you're doing React, you're often used to a transpile step and more likely to add CSS to the build chain. These little differences throw you off for a while as you re-acclimatise to that estranged markup language.

Separation of concerns

The most refreshing thing about writing HTML from scratch and CSS from scratch is the separation of concerns. I miss it. Writing markup in one file and how it looks in another, loosely coupled with class names is great. The product I work on professionally uses Emotion, which means styling is added directly to the element and is pulled into the head of the document on build. It's useful in some circumstances and isn't as noticeable when hidden away in reusable components but it's always present. When parsing code, looking at the semantic structure isn't as easy when styles are essentially inlined for the majority of elements. Your markup, styles and logic are smooshed into a Frankenstien's monster that we call a React component. There's also a gap between what you write and what's produced, a class name is automatically generated for each element so tracing a style from a production build back to the source can be challenging.

A separate CSS file and HTML file make it easier to parse both files. It's just more readable. It's not necessarily the best method for a component based frontend but a lot of webpages, it works so well. There is some back and forward when inspecting classes in HTML and the implementation in CSS but it's nothing that can't be eased by using dev tools or a split editor.

Zero JS

I often don't get a chance to just write a static web page, and if I do I often include JavaScript to add little enhancements here and there. As a daily React developer, my mind is constantly aware of where interactivity

Figure 5: Yellow JS logo beside hand pointing

could be added. There's often a need for interactivity and JS is the goto tool. For my experiment, I didn't write a line of JavaScript. The most complex thing on the page is some stateless animations which are achieved with CSS. It's useful to remember that CSS is a really powerful language. It may be difficult to master but once you have a good understanding of it, you can make web pages more interactive and feature rich. You can go a long way before JS is necessary.

The most fun from not using CSS is the constraints it adds. Constraints make the most interesting decisions, they force you to think with the tools you have and push the boundaries further than you normally would. Setting myself an artificial constraint of not using JS was fun because it forced me to use CSS for animations and focus on the look and feel rather than functionality. You're not tempted to add additional libraries because you can't. You're not attempting to manage state within a component because there is little to no state. Constraints give you a framework to work within, to the push the limits of and to embrace.

You should try it

I don't often get the chance to write HTML, CSS or plain JS in day to day life and I don't think I'm alone. In my job, HTML is rarely written by hand because external requirements often mean that markup is generated by data. It's generated by a CMS or piped into React components so it can be made interactive. There's an expectation of compatibility because you're serving a large audience with different devices and it has to work consistently across all of them. This results in CSS being generated rather than hand crafted. It's safer. It's more compatible. It's more scalable. Your styles are auto-prefixed, to work with multiple browsers, deduped and minified for faster loading. JavaScript is often the result of a pipeline that starts with TypeScript and/or JSX. The source code makes its way towards something safer, more compatible and easier to maintain . It's very rare to write JavaScript from scratch with no changes between the source we write and the bundle the client receives.

Should you stop using modern tooling, throw caution to the wind and code like it's 1999? *Probably* not.

Modern tooling gives us superpowers for manipulating the raw elements that make up the web but it's a layer of abstraction. That's not a bad thing. The modern approach to web development takes some of the tedium out of frontend development. It's allowed us to move faster than ever before and create experiences which are feature rich and delightful to use on more devices than ever before. Having said this, sometimes it's nice to have a break. Everything is a tradeoff and sometimes it's nice to revaluate. It's a breath of fresh air to just sit down and whip up a simple webpage without ever waiting for a build step. Never installing a package via npm. Having full control over the output.

It takes away certain decisions like which package manager to use, how to bundle assets, which CSS-in-JS method is the best. It adds constraints. You're not thinking of what package could solve a problem, you're tinkering directly with the web and working with its quirks and limitations. It's not bad or good, it's just different. It harks back to times where we had different problems and different concerns.

I challenge you to go out of your comfort zone and give it a go. See what it's like to code from scratch in 2021, you might learn something new or gain an appreciation for how far we've come.