Nathan McMillan A11700628

Homework 3 Writeup

Vanilla https://ember-dex.firebaseapp.com/

Lines of code: 871 total
Hours taken: approximately 26 hours

Load times:

Mobile / default.css: 84ms	Desktop / default.css: 58ms
Mobile / Optimus Princeps: 325ms	Desktop / Optimus Princeps: 38ms
Mobile / Optimus Princeps Bold: 391ms	Desktop / Optimus Princeps Bold: 41ms
Mobile / Font Awesome: 378ms	Desktop / Font Awesome: 69ms
Mobile / index: 0.80~1.37s	Desktop / index: 0.24~1.00s
Mobile / home: 1.81~1.86s	Desktop / home: 0.18~0.62s
Mobile / view character: 1.25~1.58s	Desktop / view character: 0.21~0.26s

Byte count:

- Index: 131.36 kb
- Home: 285.52 kb
- View Character: 219.21 kb

Using vanilla CSS was enjoyable and provided full control over my design. However it was a challenge that often involved spending hours stuck on what would turn out to be very simple solutions to problems. When I believed I had finally finished, I only then found out that it looked terrible on mobile and still needed a lot more work. This added an additional half of a day to understand and implement all of the HTML and CSS rules that I needed to allow the website to look good depending on device and screen resolution. Nonetheless, I overall preferred writing vanilla significantly more than working with a framework. CSS requires a lot of knowledge already, so learning all the necessary information of a framework and then fighting to make it do what I want did not feel worth it for me. Looking at the byte count and load times, I'm satisfied with the job I have done to make it look aesthetically pleasing while remaining relatively quick to load. On another note, to improve loading speeds, I am considering removing the bold version of the font I am using, although I believe the custom font and icons are too important to remove entirely.

Framework https://ember-dex-framework.firebaseapp.com/

Using: bootstrap from http://getbootstrap.com/
Lines of code: 340 total
Hours taken: approximately 7 hours

Load times:

Mobile / bootstrap.css: 103~292ms	Desktop / bootstrap.css: 23ms
Mobile / add.css: 144~233ms	Desktop / add.css: 46ms
Mobile / index: 1.54~1.58s	Desktop / index: 0.21~0.31s
Mobile / home: 1.26~1.78s	Desktop / home: 0.21~0.38s
Mobile / view character: 1.66~2.19s	Desktop / view character: 0.16~0.25s

Byte count:

- Index: 22.92 kb
- Home: 273.16 kb
- View Character: 110.78 kb

Initially I attempted to use the pure framework as my choice. But after not too long, I realized there was not enough documentation for me to understand everything pure had to offer, so I switched to bootstrap which seemed to have much better documentation and examples. Bootstrap proved to be relatively nice with their “mobile first” and grid based design. Using their classes offered a simple approach for creating everything I needed without having to go back and add special rules for mobile use. Because of this it took me significantly less time to finish compared to vanilla CSS. My biggest complaint after using bootstrap was their navbar. A lot of their documentation relied on special dropdown “hamburger” menus, and would otherwise look bad when on multiple lines. Nonetheless, the navbar in vanilla CSS was also by far the hardest part for me to write as well.

Discussion

Writing a website with vanilla CSS and then going back and remaking it with the bootstrap framework was both challenging and informative. After working with both, there are clearly many benefits as well as disadvantages of each method.

There are many great benefits of working with frameworks, especially if one is just beginning to learn CSS then it can help lay down a strong foundation without being too difficult. This also ties into the importance of building good habits such as encouraging grid based design which can be very important for a good, readable and responsive website. Additionally, frameworks take away a lot of the problems with building a cross-browser website. This can be a major problem that takes up a lot of time, but frameworks generally take this into account and make common problems associated with it significantly more manageable. Taking these advantages into account, they also contribute to a negative side. Using a framework means taking time to learn all of it, so becoming truly proficient will require using the same framework many times in order to maximize the benefit. Similar to the last point, the frameworks all have their own custom preferred way of how to write HTML and CSS, which often leads to being forced to write in a specific manner that is not always what the developer is used to knows to be good practice. For instance many frameworks suffer from "div-itus" which we've learned we should avoid. Finally, frameworks are always full of features that one will probably never use, and therefore have quite a lot of bloat attached to them that does nothing but slow down loading times for anyone on the website.

With all of that said, we can delve into why vanilla CSS also has so many pros and cons. Vanilla has the great advantage of being fully customizable, have no restrictions, and do exactly what one tells it to do. One can custom design a series of CSS rules tailored to the website or app's exact needs, and there will be to next to no bloat with unused rules contributing to poor byte counts and performance. Vanilla also forces the developer to thoroughly understand what they are doing if their goal is to make a responsive website that works with a multitude of devices. CSS will be around for many years to come, so having the knowledge of everything it offers and what it can do is incredibly important. Moving to different jobs or projects, the CSS framework might change, but the underlying CSS will always be the same. Thoroughly understanding these fundamentals offers a huge performance improvement for the developer for when the inevitable time comes that they can no longer use their preferred framework. That said, vanilla CSS is not always the right choice. The web is always changing with new rules added for CSS and browsers being updated. Different browsers always have their own set of quirks and special prefixes. This makes it difficult to keep track of everything and fix problems that only appear on specific browser versions. Using vanilla forces the developer to take time to solve these problems, which can be very difficult and stressful when deadlines are in place. Similar to this, a developer might for instance need to build a quick prototype for a website. Without using a framework, they will need to build everything from the ground up, including very simple things like the look of buttons. Whereas a framework will standardize and come prepacked with relatively good looking buttons and other common elements.

In conclusion, we've looked at the pros and cons of each and seen that they are not better or worse than each other. Without having the time or resources to test my website on all the most common devices, I am much more convinced my website appears correct with my framework version. I am sure there are many small things I do not know about that I missed with my vanilla implementation that bootstrap already fixes. Bootstrap also took significantly less time to implement, but suffers from looking similar to any other page using default bootstrap. Because of these reasons, I would recommend using a framework whenever their is a short deadline and not enough time to worry about details compared to making sure their is a working, reliable website. If this is not the case, then there is plenty of reason to take time to develop a custom tailored site from the ground up with vanilla CSS. With proper testing, it will have all the benefits of being responsive and reliable as frameworks, in addition to the benefits of being lightweight and unique.