Implementing CSS Grid for Website Layout

By Emily Hennes

Overview

I initially began building the layout for my website using CSS Flexbox, as most likely my website is not going to be a complex 2D design that requires columns and rows. However, I kept running into situations where I needed more specific placement for certain elements, and using Flexbox was becoming unnecessarily complicated. I came to the conclusion that even though this was a simple layout, using Grid would be more straightforward and more flexible for future changes; I wanted to test that thought out for my homepage.

Approach

To begin, I wrapped all HTML in my `<body>` in a `<div class="wrapper">`. This served as the grid system for my homepage (and potentially my entire webpage); it is important to note that grid specifications only extend to direct children, so only the children of the `.wrapper` class are included in the grid. Then, I specified my grid with the following CSS:

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-template-areas:
    "header header header header"
    ". main main .";
}
```

Following the `grid-template-areas`, the first section to complete is the website header. Because this section only includes a logo and navigation bar, it is very simple to use Flexbox and call it a day, specifically using the `space-between` and `inline-flex` properties. However, I found it was even more intuitive to use Grid, even though it required a nested grid to reach the grandchild `.header` class. To do this, I applied the following styles to the `.header` class:

```
.header {
  background-color: $header-color;
  display: grid;
  grid-template-columns: 65% 35%;
  grid-template-rows: 1fr;
  grid-column: 1 / span 4;
```

Although, in this case, Flexbox worked almost as well, if not just as well, as Grid, I will most likely use the latter for future navigations because it was a lot easier to place each section exactly where I wanted it (less tinkering required).

The next section listed in `grid-template-areas` is the main content for my homepage. To complete this, I added the following CSS to my `.intro-container` class:

```
.intro-container {
  grid-area: main;
  justify-self: center;
}
```

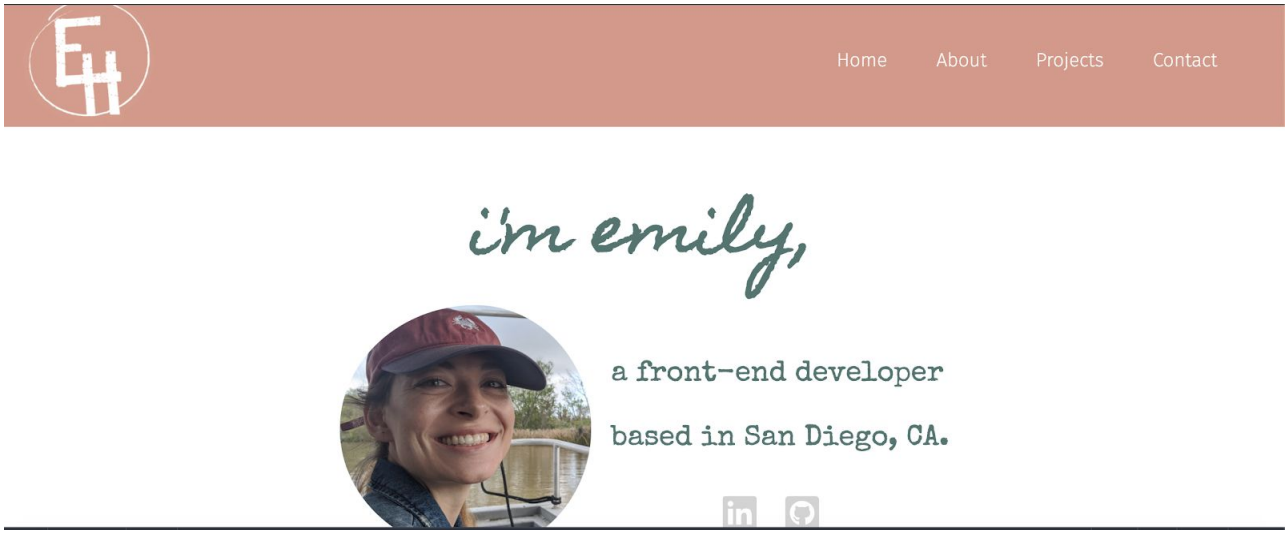
Using Flexbox, I found myself adding `display: flex` to multiple classes inside `.intro-container`, as well as using a lot of `padding` and `text-align` to get the result I wanted. As you can see, using Grid made everything more succinct. However, I did find myself needing Flexbox's optional wrapping capability in order to get my text to sit next to my image. In order to do this, the following was all that was required:

```
.subheading {
  align-items: center;
  display: flex;
}
```

While Grid is excellent at precisely placing elements on a page, I believe Flexbox will always shine in how flexible it can be, and if you naturally need items to fall into place, Flexbox will usually be the solution for that. For this reason, I moved away from my original plan to convert everything to a grid-based layout, and ended up using a combination of Grid and Flexbox.

Result

The resulting homepage is identical to the original in terms of what the user sees. However, in terms of the codebase, implementing a combination of CSS Grid and Flexbox resulted in cleaner code with less repetition, and the ability and flexibility to handle updates and changes more readily, without greatly disturbing all of the surrounding elements. After this initial test on my homepage, I will most likely continue to use this combination for the entire site.



View on Codepen