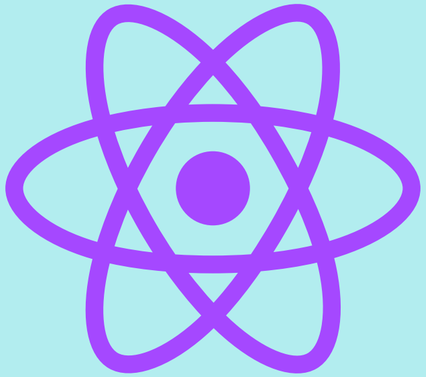
COMP6080

Single Page Apps



Presented by
Christian Scott

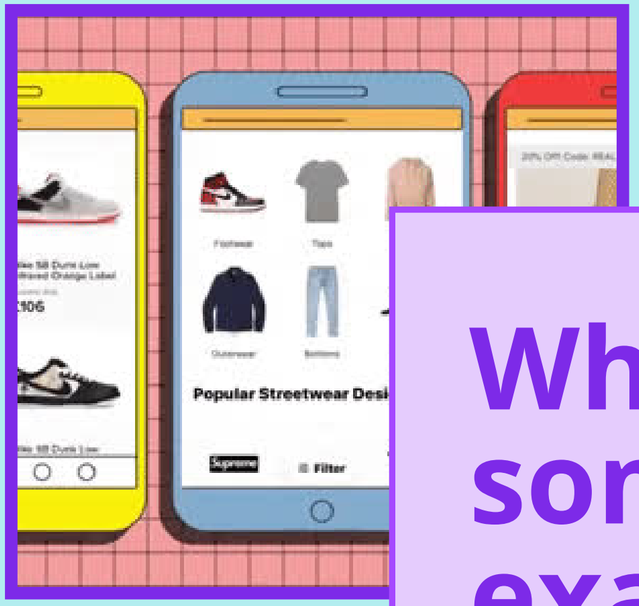
What are SPAs?

Single page applications (SPAs) are a kind of web application. They get their name from the fact that they only download a *single* HTML document.

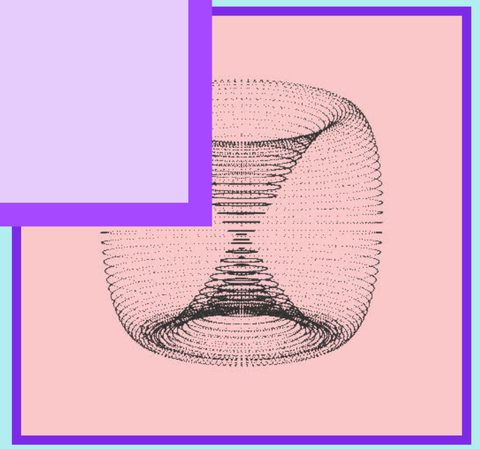
When you interact with an SPA, the content on the screen is updated via JavaScript.

This is in contrast with a traditional, multi-page website, where the content will be updated by downloading a new HTML document every time you navigate to a new page.

In an SPA, new content is downloaded over the network using JavaScript APIs like XMLHttpRequest and fetch.



What are some examples?



Single

The new versions of Reddit and Facebook are both SPAs

Multi

Sites like Wikipedia and Amazon are not SPAs

Why an SPA?

SPAs offer faster "post-load" performance that can make them feel more like a native application.

That is, after the page is loaded updates to the page are much faster since you don't need to download a new document every time you navigate to a new page.

Instead, these applications load small amounts of data from the server and use that to update the page.

By pre-fetching data or just having all the data available on first load, you can even do near-instantaneous transitions between pages.

Why *not* an SPA?

SPAs are not a silver bullet. In fact, they're quite hard to get right.

There are a few big challenges that you'll run into when building an SPA:

- Search engine optimisation (SEO)
- Performance
- Browser support

SEO

For many websites, getting a good ranking on sites like google and bing is critical for their success.

SEO is a key part of this. In short, SEO means making sure that search engine crawlers are able to *understand the content of your website.* People often pull some dirty tricks to try and boost their ranking, but most of the time it's just about making sure your content is good & that it's structured in a way that makes sense.

There are a few ways that SPAs and crawlers don't mix:

- By default, the HTML document for an SPA is empty
- The "pages" of your website aren't real
- *TODO: expand on these points*

Performance

SPAs are a double edged sword when it comes to performance.

On one hand, **once your application is loaded it will usually be a lot snappier than a traditional multi-page website**. This is because the content on the page is updated using JavaScript that you've already downloaded, rather than waiting for a new HTML document.

On the other hand, the **performance from the perspective of the initial load is usually much worse with an SPA**. This is due to two factors:

1. Downloading the JS bundle required to run an SPA is expensive
2. Not only do you need to wait for the JS bundle to download, you also need to wait for it to be parsed, compiled (yes, compiled!) and run

Performance (cont)

All this means that while your application might be snappy while it's running, your users will have to wait a bit longer to use it than they would if you had built a multi-page website instead.

However, that advantage is assuming that your users are on fast devices too! Your website might be a delight on your brand new macbook, but what's the experience like on a 5 year old, mid-range Android phone?

Not only will the initial performance be worse due to the fact that older devices will take longer to parse, compile and run your app, the performance as your app is running will be worse too – it might be so heavyweight that it slows to a halt.

Browser Support

Browsers are not designed to support SPAs out of the box. Support for SPAs is getting better all the time – but often you end up needing to rebuild basic browser functionality inside JavaScript. This re-built functionality is often incorrect and slower than the version that the browser includes.

Take navigation for example. Having a history of all the pages you have visited is very useful. Imagine if you couldn't use the back button while navigating Facebook!

How are SPAs built?

You can build an SPA using any JavaScript framework – or even without a framework at all if you feel like getting stuck in to it!

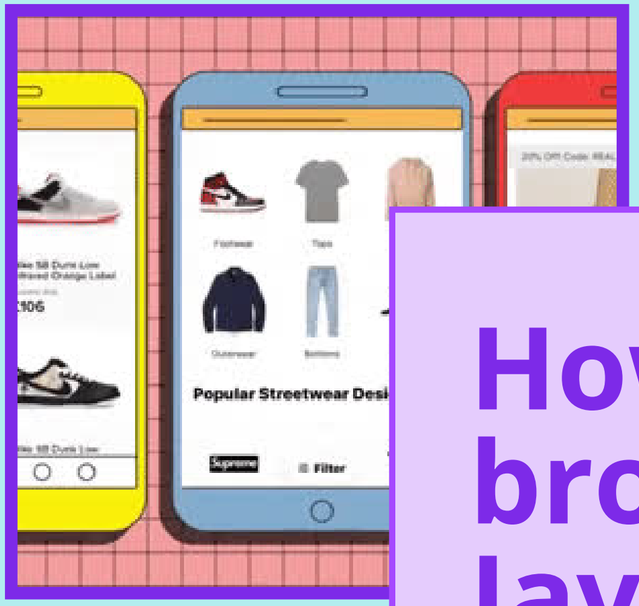
The most popular JS framework at the moment is (surprise surprise) React.JS.

For routing, the most popular library is react-router.

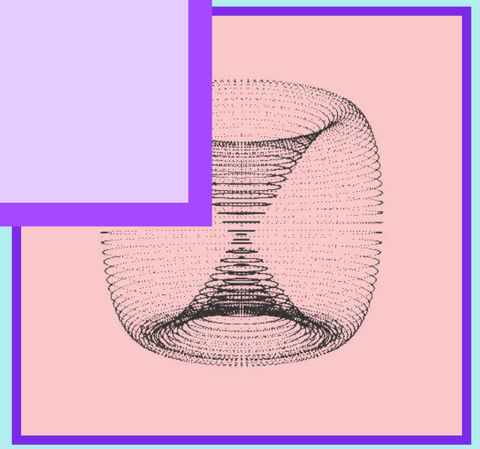
How does routing work?

TODO: finish this page

The end!



How does the browser run JavaScript?



Will the heading change?

```
<body>
  <script>
    const heading = document.querySelector('h1');
    if (heading !== null) {
      heading.textContent = 'I changed!';
    }
  </script>
  <h1>I did not change</h1>
</body>
```

Will the heading change?

No.

```
<body>
  <script>
    const heading = document.querySelector('h1');
    if (heading !== null) {
      heading.textContent = 'I changed!';
    }
  </script>
  <h1>I did not change</h1>
</body>
```

Will the heading change?

```
<body>
  <h1>I did not change</h1>
  <script>
    const heading = document.querySelector('h1');
    if (heading !== null) {
      heading.textContent = 'I changed!';
    }
  </script>
</body>
```

Will the heading change?

Yes!

```
<body>
  <h1>I did not change</h1>
  <script>
    const heading = document.querySelector('h1');
    if (heading !== null) {
      heading.textContent = 'I changed!';
    }
  </script>
</body>
```

Why?

JS is executed by the browser when it is encountered. This means that in the first case, the heading element has not even been created yet!

Will the heading change?

```
<body>
  <script src="change-heading.js"></script>
  <h1>I did not change</h1>
</body>
```

Will the heading change?

No.

```
<body>
  <script src="change-heading.js"></script>
  <h1>I did not change</h1>
</body>
```

Will the heading change?

```
<body>
  <h1>I did not change</h1>
  <script src="change-heading.js"></script>
</body>
```

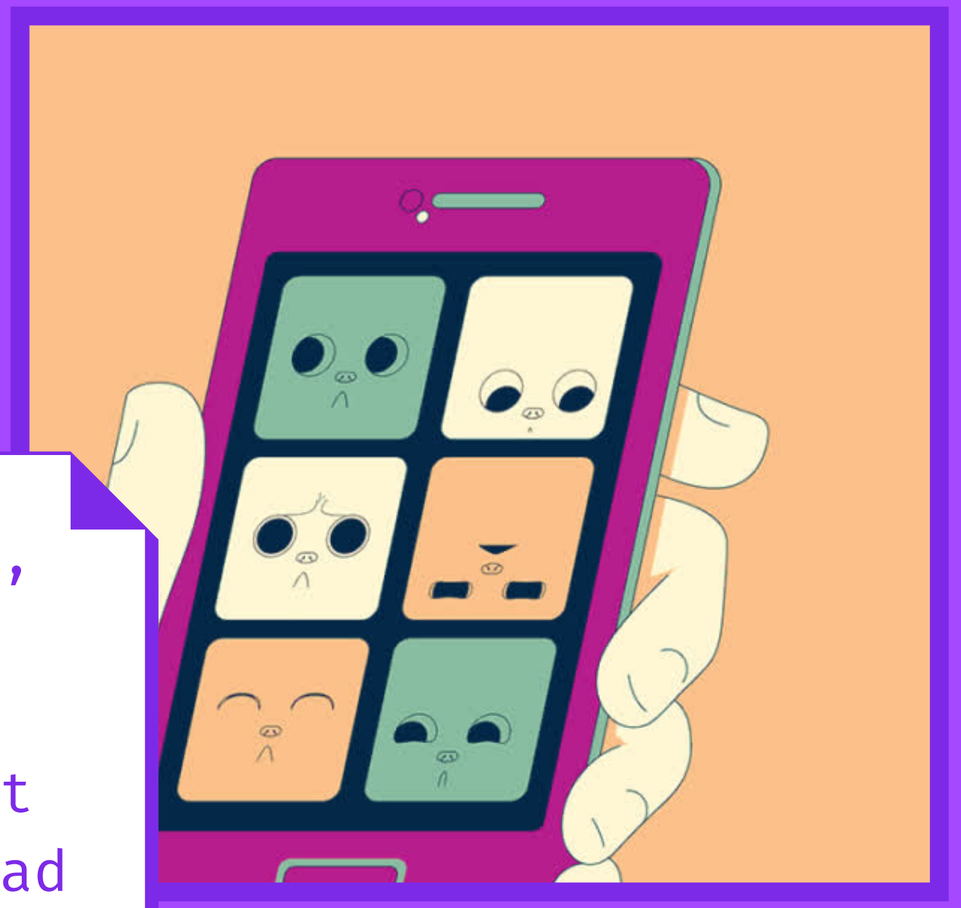
Will the heading change?

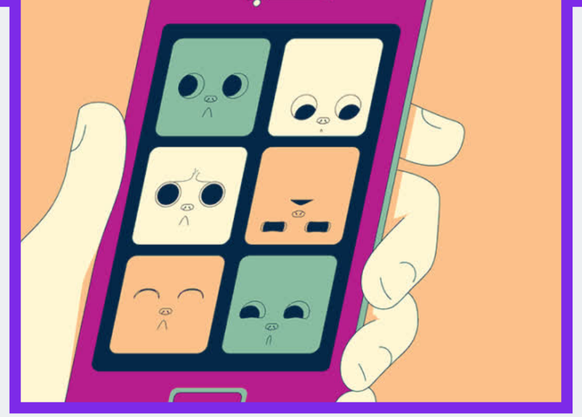
Yes!

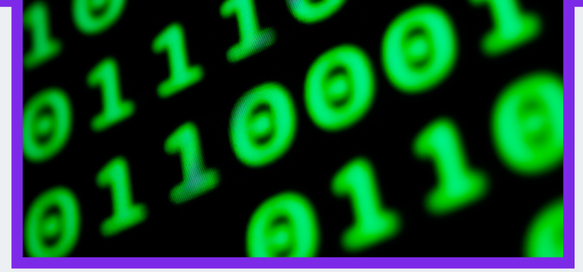
```
<body>
  <h1>I did not change</h1>
  <script src="change-heading.js"></script>
</body>
```

Why?

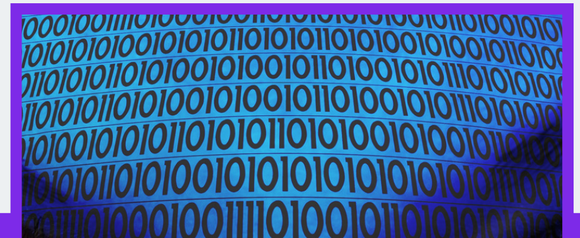
Same as before! By default, the browser will execute JavaScript when it encounters it, even if that means waiting for it to load over the internet







What if the internet is really slow?



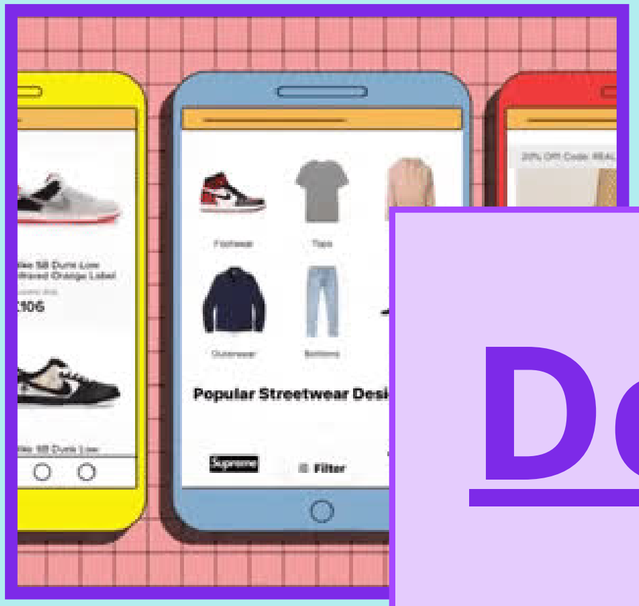
WTF is render blocking?

Before the browser can render the HTML document on your screen, it needs to *parse* the whole document. As mentioned previously, the parser will stop & execute a script whenever it encounters one.

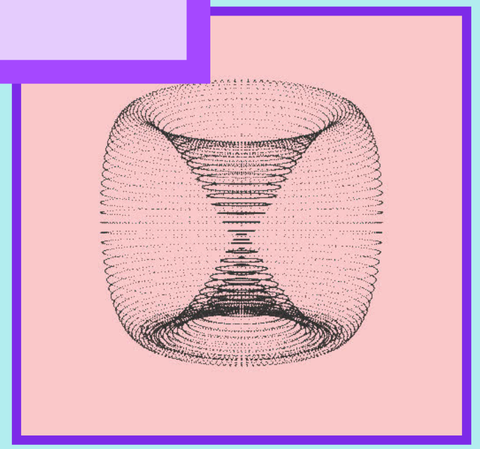
If the script is external (i.e not embedded in the document), this means that the parser will need wait for the request to complete before it can continue.

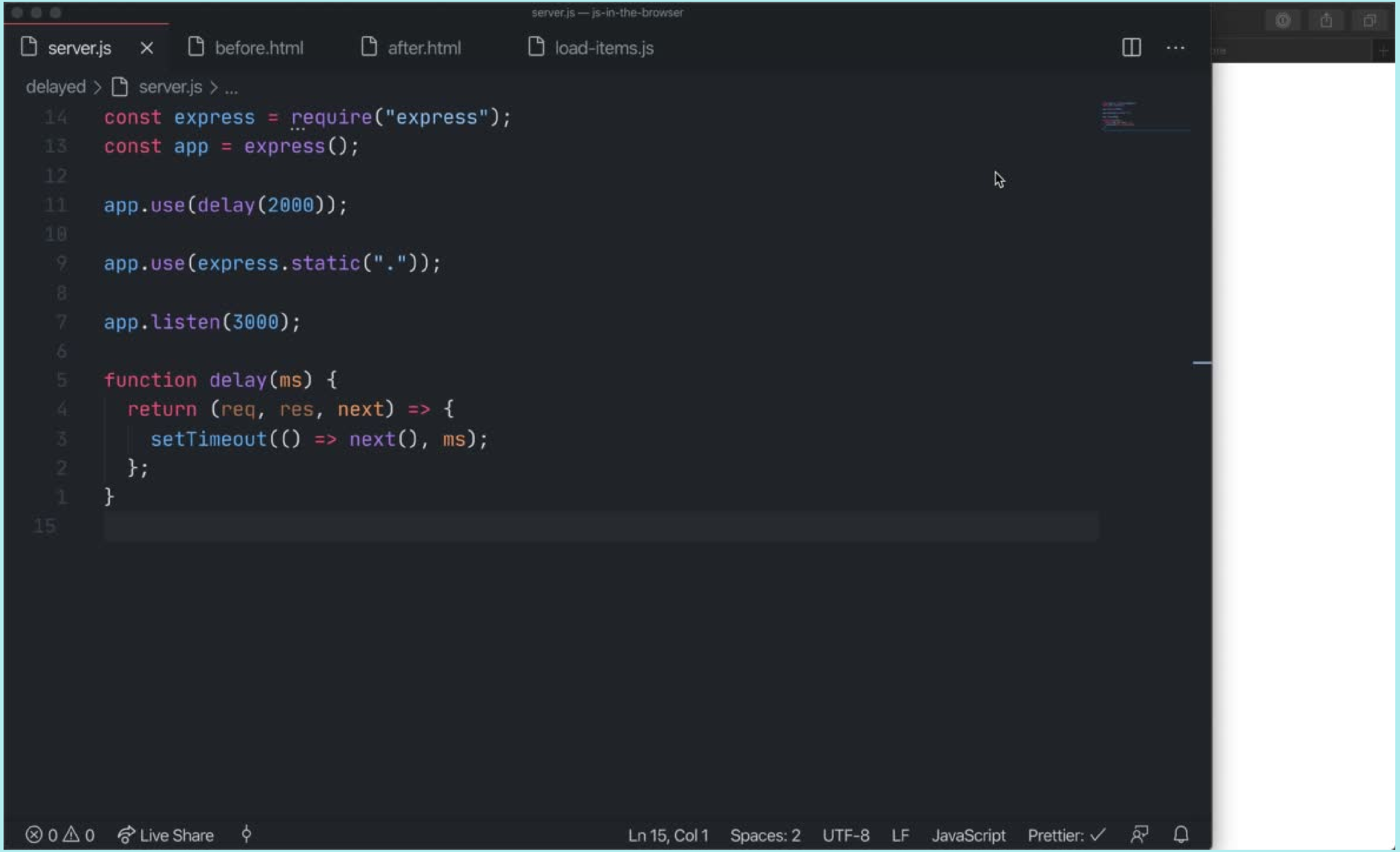
The effect of this is that sometimes you will need to wait for several seconds to see anything on the page, even once the initial request has completed! This also applies to CSS.

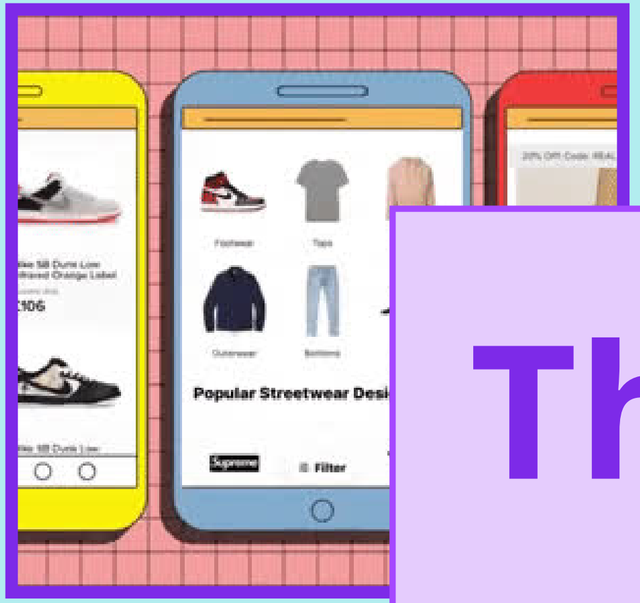
Google Web Fundamentals - Adding Interactivity With JavaScript



Demo!




server.js — js-in-the-browser
server.js
before.html
after.html
load-items.js
delayed > server.js > ...
14 const express = require("express");
13 const app = express();
12
11 app.use(delay(2000));
10
9 app.use(express.static("."));
8
7 app.listen(3000);
6
5 function delay(ms) {
4 return (req, res, next) => {
3 setTimeout(() => next(), ms);
2 };
1 }
15
⊗ 0 ⚠ 0
Live Share
Ln 15, Col 1
Spaces: 2
UTF-8
LF
JavaScript
Prettier: ✓



Thanks!