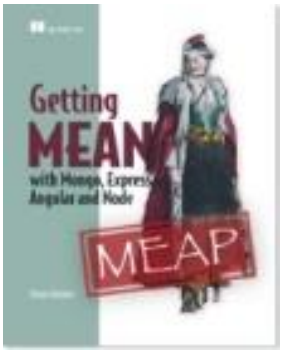
The pros and cons of SPAs

By Simon Holmes, author of Getting MEAN with Mongo, Express, Angular, and Node



Coding in SPAs (Single Page Applications) is most likely a vast improvement on what you've been doing before, but it may not always be the best solution. Here's a brief look at some things to bear in mind about SPAs when designing a solution, and how to decide whether a full SPA is right for your project.

Coding an SPA in Angular is like driving a Porsche along a coastal road with the roof down. Both are amazing. They're fun, fast, sexy, agile and very, very capable. And it's most likely that both are a vast improvement on what you've been doing before.

But sometimes they're not appropriate. If you want to pack up the surfboards and take your family away for the week you're going to struggle with the sports car. As amazing as your car may be, in this case you're going to want to use something different. It's the same story with Single Page Applications. Yes, building them in Angular is amazing, but sometimes it's not the best solution to your problem.

Let's take a brief look at some things to bear in mind about SPAs when designing a solution and decide whether a full SPA is right for your project or not. This article is not intended to be in any way "anti-SPA." SPAs generally offer a fantastic user experience whilst reducing the load on your servers, and therefore also your hosting costs.

Hard to crawl

JavaScript applications are very hard for search engines to crawl and index. Most search engines look at the HTML content on a page but do not download or execute much JavaScript. For those that do, the actual crawling of JavaScript-created content is nowhere near as good as content delivered by the server. If all of your content is served via a JavaScript application, then you can't be sure how much of it will be indexed.

A related downside is that automatic previews from social sharing sites – such as Facebook, LinkedIn and Pinterest – do not work very well. This is also because they look at the HTML of the page you're linking to and try to extract some relevant text and images. Like search engines, they don't run JavaScript on the page, so content served by JavaScript won't be seen.

Making an SPA crawlable

There are a couple of workarounds to make it look as though your site is crawlable. Both involve creating separate HTML pages that mirror the content of your SPA. You can have your server create a HTML-based version of your site and deliver that to crawlers, or you can use a headless browser such as PhantomJS to run your JavaScript application and output the resulting HTML.

Both of these do require quite a bit of effort, and can end up giving a maintenance headache if you have a large complex site. There are also potential SEO pitfalls. If your server-generated HTML is deemed to be too different from the SPA content, then your site will be penalized. Running PhantomJS to output the HTML can slow down the response speed of your pages, which is something for which search engines – Google in particular – downgrades you.

Does it matter?

Whether this matters or not depends on what you want to build. If the main growth plan for whatever you're building is through search engine traffic or social sharing then this is something you want to give a great deal of thought to. If you're creating something small that will stay small, then managing the workarounds is achievable, whereas at a larger scale you'll struggle.

On the other hand, if you're building an application that doesn't need much SEO – or indeed if you *want* your site to be harder to scrape – then this isn't an issue you need to be concerned about. It could even be an advantage.

Analytics and browser history

Analytics tools such as Google Analytics rely heavily upon entire new pages loading in the browser, initiated by a URL change. SPAs don't work this way. There's a reason they're called *Single Page* Applications!

After the first page load, all subsequent page and content changes are handled internally by the application. So the browser never triggers a new page load, nothing gets added to the browser history, and your analytics package has no idea who's doing what on your site.

Adding page loads to an SPA

You can add page load events to an SPA using the HTML5 history API; this will help you integrate analytics. The difficulty comes in managing this and ensuring that everything is being tracked accurately – this involves checking for missing reports and double entries.

The good news is that you don't have to build everything from the ground up. There are several open source analytics integrations for Angular available online, addressing most of the major analytics providers. You still have to integrate them into your application and make sure that everything is working correctly, but you don't have to do everything from scratch.

Is it a major problem?

The extent to which this is a problem depends on your need for undeniably accurate analytics. If you want to monitor trends in visitor flows and actions then you're probably going to find it easy to integrate. The more detail and definite accuracy you need, the more work it is to develop and test. Whilst it is arguably much easier to just include your analytics code on every page of a server-generated site, analytics integration is not likely to be the sole reason that you choose a non-SPA route.

Speed of initial load

Single Page Applications have a slower first page load than server-based applications. This is because the first load has to bring down the framework and the application code before rendering the required view as HTML in the browser. A server-based application just has to push out the required HTML to the browser, reducing the latency and download time.

Speeding up the page load

There are some ways of speeding up the initial load of an SPA, such as a heavy approach to caching and lazy-loading modules when you need them. But you'll never get away from the fact that it needs to download the framework, at least some of the application code, and will most likely hit an API for data before displaying something in the browser.

Do you care about speed?

The answer to whether you should care about the speed of the initial page load is, once again "it depends." It depends on what you're building, and how people are going to interact with it.

Think about Gmail. Gmail is an SPA and takes quite a while to load. Granted this is only normally a couple of seconds, but everyone online is impatient these days and expects immediacy. But people don't mind waiting for Gmail to load, as it is snappy and responsive once you are in. And once you are in, you often stay in for a while.

However, if you have a blog pulling in traffic from search engines and other external links you don't want the first page load to take a few seconds. People will assume your site is down or running slowly and hit the back button before you've had the chance to show them content. I'm willing to bet that you know this happens as you've done it yourself!

To SPA or not to SPA?

Just a reminder that this wasn't an exercise in SPA-bashing, we're just taking a moment to think about some things that often get pushed to one side until it's too late. The three points about crawlability, analytics integration and page load speed are not designed to give clear-cut definitions about when to create an SPA and when to do something else. They are there to give a framework for consideration.

It might be the case that none of those things is an issue for your project, and that an SPA is definitely the right way to go. If you find that each point makes you pause and think, and it looks like you need to add in workarounds for all three, then an SPA probably isn't the way to go.

If you're somewhere in between then it's a judgment call about what is most important, and crucially what is the best solution for the project. As a rule of thumb, if your solution includes a load of workarounds at the outset then you probably need to rethink it.