JavaScript rendering and the problems for SEO in 2020

30-second summary:

- Anyone working in enterprise SEO in 2020 will have encountered this web architecture scenario with a client at some point. Frameworks like React, Vue, and Angular make web development more simply expedited.
- There are tons of case studies but one business Croud encountered migrated to a hybrid Shopify / JS framework with internal links and content rendered via JS. They proceeded to lose traffic worth an estimated $8,000 per day over the next 6 months… about $1.5m USD.
- The experienced readers amongst us will soon start to get the feeling that they're encountering familiar territory.
- Croud's VP Strategic Partnerships, Anthony Lavall discusses JavaScript frameworks that deal with the most critical SEO elements.

While running the SEO team at Croud in New York over the last three years, 60% of our clients have been through some form of migration. Another ~30% have either moved from or to a SPA (Single Page Application) often utilizing an AJAX (Asynchronous Javascript and XML) framework to varying degrees.

Anyone working in enterprise SEO in 2020 will have encountered this web architecture scenario with a client at some point. Frameworks like React, Vue, and Angular make web development more simply expedited. This is especially true when creating dynamic web applications which offer relatively quick new request interactivity (once the initial libraries powering them have loaded – Gmail is a good example) by utilizing the

power of the modern browser to render the client-side code (the JavaScript). Then using web workers to offer network request functionality that doesn't require a traditional server-based URL call.

With the increased functionality and deployment capabilities comes a cost – the question of SEO performance. I doubt any SEO reading this is a stranger to that question. However, you may be still in the dark regarding an answer.

Why is it a problem?

Revenue, in the form of lost organic traffic via lost organic rankings. It's as simple as this. Web developers who recommended JavaScript (JS) frameworks are not typically directly responsible for long-term commercial performance. One of the main reasons SEOs exist in 2020 should be to mitigate strategic mistakes that could arise from this. Organic traffic is often taken as a given and not considered as important (or controllable), and this is where massive problems take place. There are tons of case studies but one business we encountered migrated to a hybrid Shopify / JS framework with internal links and content rendered via JS. They proceeded to lose traffic worth an estimated $8,000 per day over the next 6 months… about $1.5m USD.

What's the problem?

There are many problems. SEOs are already trying to deal with a huge number of signals from the most heavily invested commercial algorithm ever created (Google… just in case). Moving away from a traditional server-rendered website (think Wikipedia) to a contemporary framework is potentially riddled with SEO challenges. Some of which are:

- Search engine bot crawling, rendering, and indexing – search engine crawlers like Googlebot have adapted their

crawling process to include the rendering of JavaScript (starting as far back as 2010) in order to be able to fully comprehend the code on AJAX web pages. We know Google is getting better at understanding complex JavaScript. Other search crawlers might not be. But this isn't simply a question of comprehension. Crawling the entire web is no simple task and even Google's resources are limited. They have to decide if a site is worth crawling and rendering based on assumptions that take place long before JS may have been encountered and rendered (metrics such as an estimated number of total pages, domain history, WhoIs data, domain authority, etc.).

Google's Crawling and Rendering Process – The 2nd Render / Indexing Phase (announced at Google I/O 2018)

- Speed – one of the biggest hurdles for AJAX applications. Google crawls web pages un-cached so those cumbersome first loads of single page applications can be problematic. Speed can be defined in a number of ways, but in this instance, we're talking about the length of time it takes to execute and critically render all the resources on a JavaScript heavy page compared to a less resource intensive HTML page.
- Resources and rendering – with traditional server-side code, the DOM (Document Object Model) is essentially rendered once the CSSOM (CSS Object Model) is formed or to put it more simply, the DOM doesn't require too much further manipulation following the fetch of the source code. There are caveats to this but it is safe to say that client-side code (and the multiple libraries/resources that code might be derived from) adds increased complexity to the finalized DOM which

means more CPU resources required by both search crawlers and client devices. This is one of the most significant reasons why a complex JS framework would not be preferred. However, it is so frequently overlooked.

Now, everything prior to this sentence has made the assumption that these AJAX pages have been built with no consideration for SEO. This is slightly unfair to the modern web design agency or in-house developer. There is usually some type of consideration to mitigate the negative impact on SEO (we will be looking at these in more detail). The experienced readers amongst us will now start to get the feeling that they are encountering familiar territory. A territory which has resulted in many an email discussion between the client, development, design, and SEO teams related to whether or not said migration is going to tank organic rankings (sadly, it often does).

The problem is that solutions to creating AJAX applications that work more like server-based HTML for SEO purposes are themselves mired in contention; primarily related to their efficacy. How do we test the efficacy of anything for SEO? We have to deploy and analyze SERP changes. And the results for migrations to JavaScript frameworks are repeatedly associated with drops in traffic. Take a look at the weekly stories pouring into the "JS sites in search working group" hosted by John Mueller if you want some proof.

Let's take a look at some of the most common mitigation tactics for SEO in relation to AJAX.

The different solutions for AJAX SEO mitigation

1. Universal/Isomorphic JS

Isomorphic JavaScript, AKA Universal JavaScript, describes JS

applications which run both on the client and the server, as in, the client or server can execute the <script> and other code delivered, not just the client (or server). Typically, complex JavaScript applications would only be ready to execute on the client (typically a browser). Isomorphic Javascript mitigates this. One of the best explanations I've seen (specifically related to Angular JS) is from Andres Rutnik on Medium:

1. The client makes a request for a particular URL to your application server.
2. The server proxies the request to a rendering service which is your Angular application running in a Node.js container. This service could be (but is not necessarily) on the same machine as the application server.
3. The server version of the application renders the complete HTML and CSS for the path and query requested, including <script> tags to download the client Angular application.
4. The browser receives the page and can show the content immediately. The client application loads asynchronously and once ready, re-renders the current page and replaces the static HTML with the server rendered. Now the web site behaves like an SPA for any interaction moving forwards. This process should be seamless to a user browsing the site.

Source: *Medium*

To reiterate, following the request, the server renders the JS and the full DOM/CSSOM is formed and served to the client. This means that Googlebot and users have been served a pre-rendered version of the page. The difference for users is that the HTML and CSS just served is then re-rendered to replace it with the dynamic JS so it can behave like the SPA it was always intended to be.

The problems with building isomorphic web pages/applications appear to be just that… actually building the thing isn't easy. There's a decent series here from Matheus Marsiglio who documents his experience.

2. Dynamic rendering

Dynamic rendering is a more simple concept to understand; it is the process of detecting the user-agent making the server request and routing the correct response code based on that request being from a validated bot or a user.

This is Google's recommended method of handling JavaScript for search. It is well illustrated here:

The Dynamic Rendering Process explained by Google

The output is a pre-rendered iteration of your code for search crawlers and the same AJAX that would have always been served to users. Google recommends a solution such as prerender.io to achieve this. It's a reverse proxy service that pre-renders and caches your pages. There are some pitfalls with dynamic rendering, however, that must be understood:

- **Cloaking** – In a world wide web dominated primarily by HTML and CSS, cloaking was a huge negative as far as Google was concerned. There was little reason for detecting and serving different code to Googlebot aside from trying to game search results. This is not the case in the world of JavaScript. Google's dynamic rendering process is a direct recommendation for cloaking. They are explicitly saying, "serve users one thing and serve us another". Why is this a problem? Google says, "As long as your dynamic rendering produces similar content,

Googlebot won't view dynamic rendering as cloaking." But what is similar? How easy could it be to inject more content to Googlebot than is shown to users or using JS with a delay to remove text for users or manipulate the page in another way that Googlebot is unlikely to see (because it is delayed in the DOM for example).

- **Caching** – For sites that change frequently such as large news publishers who require their content to be indexed as quickly as possible, a pre-render solution may just not cut it. Constantly adding and changing pages need to be almost immediately pre-rendered in order to be immediate and effective. The minimum caching time on prerender.io is in days, not minutes.
- **Frameworks vary massively** – Every tech stack is different, every library adds new complexity, and every CMS will handle this all differently. Pre-render solutions such as prerender.io are not a one-stop solution for optimal SEO performance.

3. CDNs yield additional complexities… (or any reverse proxy for that matter)

Content delivery networks (such as Cloudflare) can create additional testing complexities by adding another layer to the reverse proxy network. Testing a dynamic rendering solution can be difficult as Cloudflare blocks non-validated Googlebot requests via reverse DNS lookup. Troubleshooting dynamic rendering issues therefore takes time. Time for Googlebot to re-crawl the page and then a combination of Google's cache and a buggy new Search Console to be able to interpret those changes. The mobile-friendly testing tool from Google is a decent stop-gap but you can only analyze a page at a time.

This is a minefield! So what do I

do for optimal SEO performance?

Think smart and plan effectively. Luckily only a relative handful of design elements are critical for SEO when considering the arena of web design and many of these are elements in the <head> and/or metadata. They are:

- Anything in the <head> – <link> tags and <meta> tags
- Header tags, e.g. <h1>, <h2>, etc.
- <p> tags and all other copy / text
- <table>, <ul>, <ol>, and all other crawl-able HTML elements
- Links (must be <a> tags with href attributes)
- Images

Every element above should be served without any JS rendering required by the client. As soon as you require JS to be rendered to yield one of the above elements you put search performance in jeopardy. JavaScript can, and should be used to enhance the user experience on your site. But if it's used to inject the above elements into the DOM then you have got a problem that needs mitigating.

Internal links often provide the biggest SEO issues within Javascript frameworks. This is because onclick events are sometimes used in place of <a> tags, so it's not only an issue of Googlebot rendering the JS to form the links in the DOM. Even after the JS is rendered there is still no <a> tag to crawl because it's not used at all – the onclick event is used instead.

Every internal link needs to be the <a> tag with an href attribute containing the value of the link destination in order to be considered valid. This was confirmed at Google's I/O event last year.

To conclude

Be wary of the statement, “we can use React / Angular because we’ve got next.js / Angular Universal so there’s no problem”. Everything needs to be tested and that testing process can be tricky in itself. Factors are again myriad. To give an extreme example, what if the client is moving from a simple HTML website to an AJAX framework? The additional processing and possible issues with client-side rendering critical elements could cause huge SEO problems. What if that same website currently generates $10m per month in organic revenue? Even the smallest drop in crawling, indexing, and performance capability could result in the loss of significant revenues.

There is no avoiding modern JS frameworks and that shouldn’t be the goal – the time saved in development hours could be worth thousands in itself – but as SEOs, it’s our responsibility to vehemently protect the most critical SEO elements and ensure they are always server-side rendered in one form or another. Make Googlebot do as little leg-work as possible in order to comprehend your content. That should be the goal.

Anthony Lavall is VP Strategic Partnerships at digital agency Croud. He can be found on Twitter @AnthonyLavall.

More about:

JavaScript rendering and the problems for SEO in 2020