

PWA: How to avoid partial rendering issues with service workers

In preparation for my upcoming SMX Advanced session about The New Renaissance of JavaScript, I decided to code a progressive web app and try to optimize it for SEO. In particular, I was interested in reviewing all key rendering options (client side, server side, hybrid and dynamic) from a development/implementation perspective.

I learned six interesting insights that I will share during my talk. One of the insights addresses a painful problem that I see happening so often that I thought it was important to share it as soon as possible. So, here we go.

How partial rendering kills SEO performance

When you need to render JavaScript server side, there is a chance that you won't get the full page content fully rendered. Let's review a concrete example.

The category view all page from the AngularJs site above hasn't finished loading all product images after 20 seconds. In my tests, it took about 40 seconds to load fully.

Here is the problem with that. Rendering services won't wait forever for a page to finish loading. For example, Google's dynamic rendering service, Rendertron by default won't wait more than 10 seconds.

View-all pages are generally preferred by both users and search engines when they load fast. But, how do you load a page with over 400 product images fast?

Service workers to the rescue

Before I explain the solution, let's review service workers and how they are applicable in this context. Detlev Johnson, who will be moderating our panel, wrote a great article on the topic.

When I think about service workers, I think about them as a content delivery network running in your web browser. A CDN helps speed up your site by offloading some of the website functionality to the network. One key functionality is caching, but most modern CDNs can do a lot more than that, like resizing/compressing images, blocking attacks, etc.

A mini-CDN in your browser is similarly powerful. It can intercept and programmatically cache the content from a PWA. One practical use case is that this allows the app to work offline. But what caught my attention was that as service worker operates separate from the main browser thread, it could also be used to offload the processes that slows the page loading (and rendering process) down.

So, here is the idea:

1. Make an XHR request to get the initial list of products that return fast (for example page 1 in the full set)
2. Register a service worker that intercepts this request, caches it, passes it through and makes subsequent requests in the background for the rest of the pages in the set. It should cache them all as well.
3. Once all the results are loaded and cached, notify the page so that it gets updated.

The first time the page is rendered, it won't get all the results, but it will get them on subsequent ones. Here is some code you can adapt to get this started.

I checked the page to see if they were doing something similar, but sadly they aren't.

This approach will prevent the typical timeouts and errors from disrupting the page rendering at the cost of maybe some missing content during the initial page load. Subsequent page loads should have the latest information and loaded faster from the browser cache.

I checked Rendertron, to see if this idea would be supported, and I found a pull request merged into their codebase that confirms support for the required feature.

However, as Google removed Googlebot from the list of bots supported in Renderton by default, you need to add it back to get this to work.

Service workers limitations

When working with service workers and moving background work to them, you need to consider some constraints:

1. Service workers require HTTPS
2. Service workers intercept requests at the “directory level” they are installed in. For example, `/test/ng-sw.js` would only intersect requests under `/test/*` and `/ng-sw.js` would intercept requests for the whole site.
3. The background work shouldn't require DOM access. Also there is no window, document or parent objects access.

Some example tasks that could run in the background using a service worker are data manipulation or traversal, like sorting or searching – also loading data and data generation.

More potential rendering issues

In a more generalized way, when using Hybrid or server-side rendering (using NodeJs), some of the issues can include:

1. XHR/Ajax requests timing out.
2. Server overloaded (memory/CPU).
3. Third party scripts down.

When using Dynamic rendering (using Chrome), in addition to the issues above, some additional issues can include:

1. The browser failed to load.
2. Images take long to download and render.
3. Longer latency

Bottom line is that when you are rendering pages server side and there are issues preventing full, correct rendering, the rendered content can have important discrepancies with the content shown to end users (or search bots).

There are three potential problems with this: 1) important

content not getting indexed 2) accidental cloaking and 3) compliance issues.

We haven't seen any client affected by accidental cloaking, but it could be a risk. However, we see compliance issues often. One example of compliance issue is the one affecting sites selling on Google Shopping. The information in the product feed needs to match the information on the website. Google uses the same Googlebot for organic search and Google Shopping, so something as simple as missing product images can cause ads to get disapproved.

Additional resources

Please note that this is just one example of the insights I will be sharing during my session. Make sure to stop by so you don't miss out on the rest.

I found the inspiration for my idea in this article. I also found other useful resources while researching for my presentation that I list below. I hope you find them helpful.

Developing Progressive Web Apps (PWAs) Course
JavaScript Concurrency
The Service Worker Lifecycle
Service Worker Demo

Opinions expressed in this article are those of the guest author and not necessarily Search Engine Land. Staff authors are listed here.

About The Author

Hamlet Batista is CEO and founder of RankSense, an agile SEO platform for online retailers and manufacturers. He holds U.S. patents on innovative SEO technologies, started doing SEO as a successful affiliate marketer back in 2002, and believes great SEO results should not take 6 months.

<http://feeds.searchengineland.com/~r/searchengineland/~3/eyJZOG8MwSOY/pwa-how-to-avoid-partial-rendering-issues-with-service-workers-317631>