Single Page Apps - Handling SEO

## Overview

One of challenges of creating a production Single Page Application (SPA) is that with most SPA frameworks, by design very little HTML is provided on the initial request, then the page is populated by client side JavaScript once loaded. While this allows for very quick response times and no server rendering of views (reducing server load), during search engine crawling and social posting no page content is served initally and therefore most indexing bots will not work correctly.

There are several methods used to mitigate this problem for single page applications. With the most common used being:

* Server Side Rendering (SSR)
* Pre-rendering
* Dynamic Rendering

# Terminology

## Rendering

* SSR: Server-Side Rendering - rendering a client-side or universal app to HTML on the server.
* CSR: Client-Side Rendering - rendering an app in a browser, generally using the DOM.
* Rehydration: “booting up” JavaScript views on the client such that they reuse the server-rendered HTML’s DOM tree and data.
* Prerendering: running a client-side application at build time to capture its initial state as static HTML.

## Performance

* TTFB: Time to First Byte - seen as the time between clicking a link and the first bit of content coming in.
* FP: First Paint - the first time any pixel gets becomes visible to the user.
* FCP: First Contentful Paint - the time when requested content (article body, etc) becomes visible.
* TTI: Time To Interactive - the time at which a page becomes interactive (events wired up, etc).

## Web

* SEO - General term of Search Engine Optimisations. For good SEO, the first response body generally require plain HTML of all page content. SPAs such as React, rely on javascript to fill the page content which can lead to issues on non-javascript crawler bots (which is most)
* SMO - General term for Social Medial Optimisation. Social media bots generally rely on meta tags in the response head. Therefore, for a page to be shared sucessfully on social media, the correct meta tags must be available in the first response head. Currently most social media bots do not use javascript to render a page and therefore will not work with standard SPA setups.

# Typical SSR

The standard style of SSR for modern SPAs is, for the first page visited by a client, to render the initial HTML markup (excluding media) on the server using JSDOM. This initial render is served to the client and a method such as React rehydrate is used to allow the client to load in the application javascript without causing a double render. Many librarys such as Preact, have the standard style of SSR built into them.

When applications include API calls to fetch data, the client application will be designed to accept filling of the data from the server.

## Pros and Cons

There are various advantages and disadvantages to typical SSR.

Advantages:

* Quicker time to FP
* Metadata and page content available on response body (good for SEO SMO)
* Can be cached to speed up response times
* Works well with fixed applications (e.g. company site)

Disadvantages:

* Requires application server to serve client bundle, since information will need loading
* Higher server load due to rendering initial page visit
* Can become very complex when combined with SPA routes, since each entry point will require an SSR method.
* Slower response times for human clients

# Headless Chrome SSR

Headless Chrome SSR uses a headless chrome controller, such as puppeteer, running on the server to perform SSR, instead of modifying the client application. This type of SSR comes with the advantage both of not requiring fundamental changes to the current client application, and very little setup on the application server.

Puppeteer loads the application using headless chrome just like a client would, although since it is based locally, it generally performs much faster. The resulting HTML snapshot is then returned to the client as the first response. Similar to typical SSR, responses can be cached in memory to allow for much quicker responses after pages have requested for the first time.

## Pros and Cons

Advantages:

* Metadata and page content available on response body (good for SEO SMO)
* Can be cached to speed up response times
* Tends to be slightly slower than typical SSR on first run, since there are delays on booting headless chrome
* Very little configuration on application server
* Very little change to client side application
* Works well with dynamic applications (e.g. marketplace, social media)

Disadvantages:

* Requires application server to serve client bundle, since information will need loading
* Higher server load due to rendering initial page visit
* Slower response times for human clients

# Dynamic SSR

Since SSR is commonly only used to provide SEO and SMO, particuarly in smaller applications, dynamic SSR can be used to only perform SSR bots and crawlers.

A diagram that shows how dynamic rendering works. The diagram shows the server serving
              initial HTML and JavaScript content directly to the browser. In contrast, the diagram
              shows the server serving initial HTML and JavaScript to a renderer, which converts the
              initial HTML and JavaScript to static HTML. Once the content is converted, the
              renderer serves static HTML to the crawler.

Dynamic SSR normally combines Headless Chrome SSR with a user agent filter to provide both minimal client application changes and SSR setup, while allowing for quick response times and low server load for human clients. The user agent filter can be performed on either the reverse proxy or application server depending on setup, allowing the option to have dedicated servers for bot and crawlers.

The most common applications to provider dynamic rendering are an application server running Puppeteer, or Google's Rendertron.

## Pros and Cons

Advantages:

* Metadata and page content available on response body (good for SEO SMO)
* Can be cached to speed up response times
* Tends to be slightly slower than typical SSR on first run, since there are delays on booting headless chrome
* Very little configuration on application server
* Very little change to client side application
* No changes for human clients
* Options for separation of concerns with crawlers and bots by using different servers for their requests

Disadvantages:

* Higher server load due to rendering initial page visit (can be moved to different servers)

# Resources

* <https://developers.google.com/search/docs/guides/dynamic-rendering>
* <https://developers.google.com/web/tools/puppeteer/articles/ssr>
* <https://medium.com/@shotap/server-side-rendering-and-dynamic-rendering-with-headless-chrome-f23cdabfae48>
* <https://stackoverflow.com/questions/51011222/react-ssr-nextjs-vs-chrome-headless-prerendering>
* <https://huckabuy.com/seo-knowledge/dynamic-rendering/>
* <https://developers.google.com/web/tools/puppeteer>
* <https://reactjs.org/docs/react-dom-server.html>
* <https://www.freecodecamp.org/news/demystifying-reacts-server-side-render-de335d408fe4/>