APRIL 24, 2023 / #JAVASCRIPT

Server Side Rendering in JavaScript – SSR vs CSR Explained

Scott Gary

The concept of Server Side Rendering (SSR) is often misunderstood. So my aim in this article is to bring clarity to this process and how it works.

Here's what we'll cover in this guide:

2. How do you know if a site is rendered using SSR?
3. How to use SSR, and things to keep in mind when choosing an SSR framework.
4. How to leverage SSR to improve performance.

We'll cover each of these items in depth, so that by the end of this tutorial you have a firm grasp of server side rendering and how it fits into the ever-changing world of web development.

We should also look at two key terms we'll be using throughout, and what they mean:

1. HTML – Hyper Text Markup Language. HTML isn't technically code, it's simply a markup language that structures your content on a web page.
2. DOM – Document Object Model. The DOM is an actual model of your HTML, made up of objects. It has an API interface which allows you to modify it, and in-turn modify the HTML.

It's important to understand the difference between HTML and the DOM. When you're reading through documentation it's easy to become confused and misinterpret the two.

What is Server Side Rendering (SSR)?

SSR is when you render your website's HTML on the server. This is as opposed to Client Side Rendering (CSR) when your website

How to Check For SSR

There are certain times when you'll want to check whether a site is using server side rendering. For instance, both developers and SEO professionals often need this information to help troubleshoot and optimize technical SEO issues.

We'll discuss a few techniques commonly used to determine this.

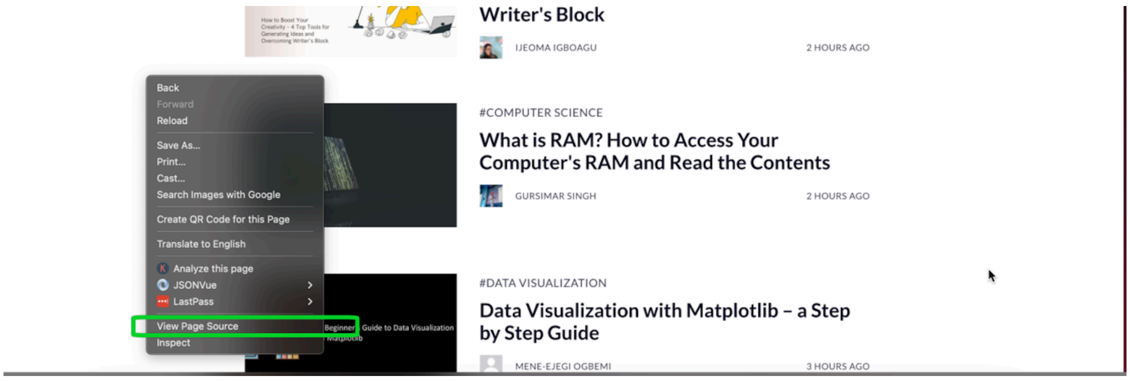
Check the page source

An easy way to determine if a site is using SSR is to view the page source.

If the HTML code is complete with all the content, including the main body, images, text, and so on, the site is likely using SSR.

On the other hand, if the HTML code is bare-bones, then it requires JavaScript to render the content. In this case, it's probably not using server side rendering.

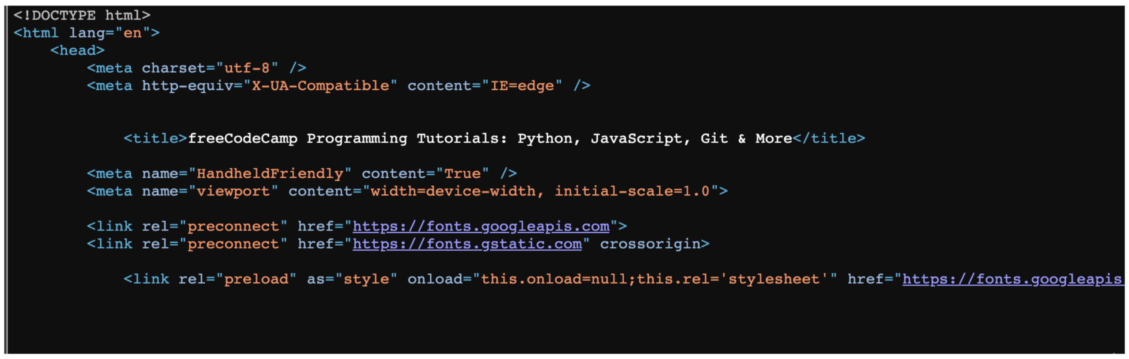
The first step is to right click in Chrome or your favorite web browser:


Writer's Block
IJEOMA IGBOAGU 2 HOURS AGO
Back
Forward
Reload
Save As...
Print...
Cast...
Search Images with Google
Create QR Code for this Page
Translate to English
Analyze this page
JSONVue
LastPass
View Page Source
Inspect
#COMPUTER SCIENCE
What is RAM? How to Access Your Computer's RAM and Read the Contents
GURSIMAR SINGH 2 HOURS AGO
#DATA VISUALIZATION
Data Visualization with Matplotlib – a Step by Step Guide
MENE-EJEGI OGBEMI 3 HOURS AGO

Right click 'view page source'

Once you are viewing the source code, you'll be able to easily search for content elements, for example you'll bee able to find `<p>` , `<h1>` , and so on.

If you can see them here, more then likely they are rendering server side:



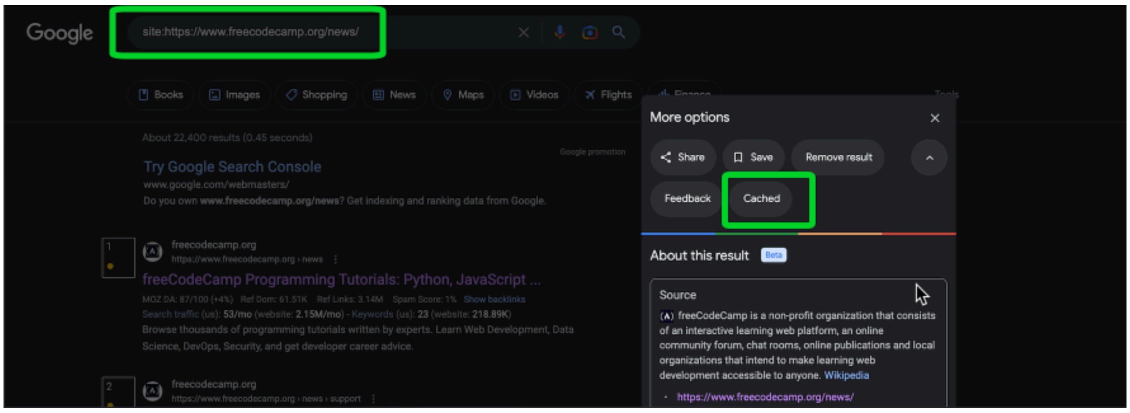
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />


            <title>freeCodeCamp Programming Tutorials: Python, JavaScript, Git & More</title>

        <meta name="HandheldFriendly" content="True" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0">

        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

            <link rel="preload" as="style" onload="this.onload=null;this.rel='stylesheet'" href="https://fonts.googleapis
```

Anything you see here is rendering sever side

Check Google Cache

An easy way to determine if your content is rendering server side is to check the Google Cache.

For example below I typed in `site:https://www.freecodecamp.org/news/` then selected '*Cached*':


site:https://www.freecodecamp.org/news/

Books · Images · Shopping · News · Maps · Videos · Flights

About 22,400 results (0.45 seconds)

Google promotion

Try Google Search Console
www.google.com/webmasters/
Do you own **www.freecodecamp.org/news**? Get indexing and ranking data from Google.

1 freecodecamp.org
https://www.freecodecamp.org › news
freeCodeCamp Programming Tutorials: Python, JavaScript ...
MOZ DA: 87/100 (+4%) Ref Dom: 61.51K Ref Links: 3.14M Spam Score: 1% Show backlinks
Search traffic (us): 53/mo (website: 2.15M/mo) - Keywords (us): 23 (website: 218.89K)
Browse thousands of programming tutorials written by experts. Learn Web Development, Data Science, DevOps, Security, and get developer career advice.

2 freecodecamp.org
https://www.freecodecamp.org › news › support

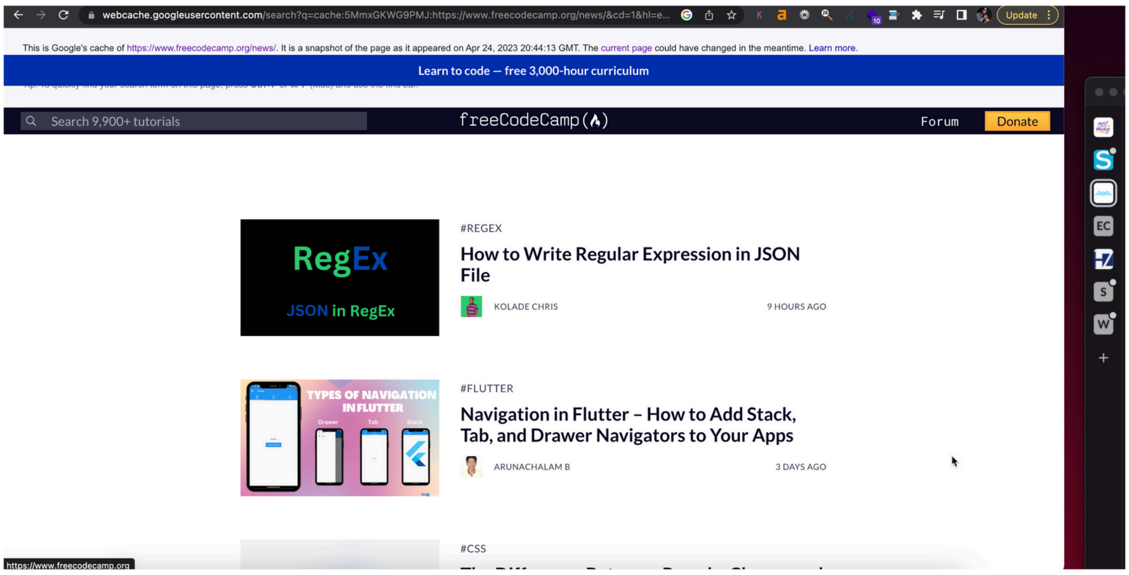
More options
Share · Save · Remove result · Feedback · Cached

About this result Beta

Source
freeCodeCamp is a non-profit organization that consists of an interactive learning web platform, an online community forum, chat rooms, online publications and local organizations that intend to make learning web development accessible to anyone. Wikipedia
• https://www.freecodecamp.org/news/

type in your URL into Google, then select 'Cached'

Generally speaking, anything you can visually see is server side rendering. If it's rendering with Javascript, more than likely you won't be able to see it:


webcache.googleusercontent.com/search?q=cache:5MmxGKWG9PMJ:https://www.freecodecamp.org/news/&cd=1&hl=e...

This is Google's cache of https://www.freecodecamp.org/news/. It is a snapshot of the page as it appeared on Apr 24, 2023 20:44:13 GMT. The current page could have changed in the meantime. Learn more.

Learn to code — free 3,000-hour curriculum

Search 9,900+ tutorials · freeCodeCamp · Forum · Donate

#REGEX
How to Write Regular Expression in JSON File
KOLADE CHRIS · 9 HOURS AGO

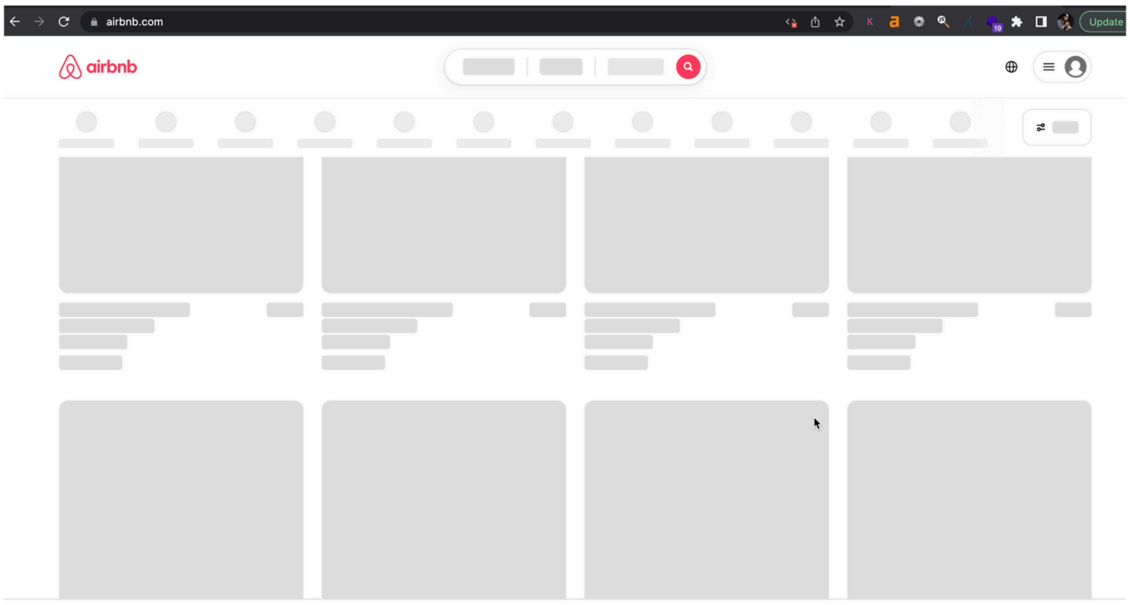
#FLUTTER
Navigation in Flutter – How to Add Stack, Tab, and Drawer Navigators to Your Apps
ARUNACHALAM B · 3 DAYS AGO

#CSS

https://www.freecodecamp.org

Anything you see here most likely renders Serverside browser. If the website's content is still visible without JavaScript, it is likely using SSR. If the website appears blank, it is not using SSR.

In this example here, we can clearly see Airbnb is not leveraging server side rendering on their homepage:



And here's a pretty good visual representation if you don't quite get the concept yet:

SSR vs CSR

Let's break down the different processes in client side and server side rendering, followed by the advantages and disadvantages of each one.

How SSR Works

1. An HTTP request is made to the server.
2. The server receives the request, and processes all (or most of) the necessary code right then and there.
3. The end result is a fully formed and easily consumed HTML page that can be sent to the client's browser via the server's response.

This is a fairly simple concept on the surface, but things can get pretty complicated when considering how to include interactive components on the client that require JavaScript. We'll touch on that later – first let's check out what happens when you request a CSR web app.

2. The server receives the request, and responds by sending an empty HTML shell to the client along with a bunch of bundled JavaScript.
3. The client receives the empty HTML shell, and proceeds to process all of the JavaScript.
4. The JavaScript modifies the DOM extensively, which renders the final HTML for the end user.

To put this in simplified language, client side rendering is when your website or web app renders into HTML in the browser by processing JavaScript, rather than on the server using whatever the backend framework of choice is.

Next we'll take a look at the strengths and weaknesses of each style of rendering.

Benefits of SSR

The main benefit of server side rendering is page load speed. Page load speed is an important metric for user experience, and subsequently an important aspect of technical SEO. Google wants to consume pages fast, too.

When a page is rendered into HTML on the server, all of the heavy-lifting is taken care of. For this reason, when the response makes it to the client's browser, there isn't much work left for the browser to display the page. It's ready to go upon delivery.

Drawbacks of SSR

Designers want pages to be interactive, and when a page is rendered into pure HTML for the client, it leaves a pretty bland user experience. So how do we make these pages interactive while preserving all the great benefits of server side rendering?

The answer is an added layer of complexity that goes by many names, but it's most commonly known as code-splitting and hydration.

Code Splitting and Hydration

In order for a page to be interactive, we need JavaScript to be sent to the client. SSR frameworks such as Next.js and Astro allow us to build an HTML only page on the server that can be sent to the client fast, while allowing for specific bundles of JavaScript to be sent to the client after the initial HTML has loaded.

In the React world, this process is known as hydration. The code is split into manageable chunks that can then be requested on an as needed basis and injected, or *hydrated*, into the client page to add interactivity and functionality.

You may be wondering why this is a "drawback." Well, the idea itself isn't the drawback, it's the technical challenge that comes along with it. Isomorphic React and other technologies that are used to accomplish this goal are notoriously complex, and it takes intimate knowledge of a framework to program these sites efficiently.

Benefits of CSR

JavaScript on the client. In fact, the entire CSR framework is often sent to the client in a purely client side rendering environment.

For this reason, once the page is initially loaded everything is very responsive for the end user. That's because everything, including the code for all other pages, is loaded along with the initial page load.

From a developer's standpoint, client side rendering is a great experience. The complexity of sharing the workload with the server is non-existent, and we can focus on building reusable interactive components that make for a streamlined development process.

Drawbacks of CSR

Soon after the explosion of CSR frameworks, SEO specialists started to realize that Google and other search engines don't do a good job indexing these pages. The synopsis was clear: pure CSR sucks for SEO.

Initial page load speed is the main drawback here. When using CSR, the page is initially sent to the client as an empty HTML shell with no content. This empty husk is often what Google and other search engines see, which isn't desirable for obvious reasons.

The JavaScript will build the page pretty quickly, but in practice most search engines still have trouble indexing the content after the DOM manipulation has completed and the HTML is rendered.

In the worst case scenario, the load time of a poorly built client side rendering app can even begin to negatively affect user experience, which is the ultimate blunder.

Based on what we've learned so far, it should be no surprise that server side rendering is a great choice when initial page load is a priority and technical SEO is important. But that's not the only driving factor behind this consideration.

When a site has a ton of dynamic and frequently changing data, server side rendering allows developers to share the workload of retrieving content.

When using a purely client side rendering app for data intensive sites, it requires many calls from the client to the server to fetch the data. This can lead to pages becoming bogged down and slow to load, which can result in a poor user experience.

Server side rendering addresses this issue by allowing the server to pre-fetch and pre-render the necessary data before sending it to the client.

Remember, most server side rendering implementations aren't *purely* SSR, they just get the heavy stuff out of the way. Developers still have the option to send specific, small bundles of JavaScript after the fact that add interactivity and even data fetching, in essence sharing the data fetching load with the server.

How to Leverage SSR for Your Project

Unless you're trying to build a framework of your own, SSR isn't something you want to implement from scratch. Luckily, there are many frameworks and libraries that can help you leverage SSR in your project.

other performance optimizations.

When it comes to leveraging server side rendering in a way that maximizes its performance benefits, you should be mindful of the impact of your application's data fetching distribution. Heavy data loads can slow down the SSR process and impact the performance of your application, which is one reason developers fetch data from the client as well.

When it comes to data fetching and processing on the server side, you can also start to incur some pretty hefty fees from your hosting provider. Keep a close eye on this if your project demands include an abundance of external data.

Overview and Concluding Thoughts

Server-side rendering (SSR) can be a powerful tool for improving the performance and user experience of web applications. By rendering HTML on the server before sending it to the client, SSR can significantly reduce the time required to display a web page, resulting in faster load times and a better user experience.

When used correctly, the technical perks of SSR usually translate to better SEO as well, as it provides search engines with easily crawlable HTML documents. If you are interested in learning more about server side rendering, check out OhMyCrawl to learn more.