Performance & UX May 7, 2024 • 9 min read

Client-side Rendering (CSR) vs. Server-side Rendering (SSR)

By Nefe Emadamerho-Atori 5909

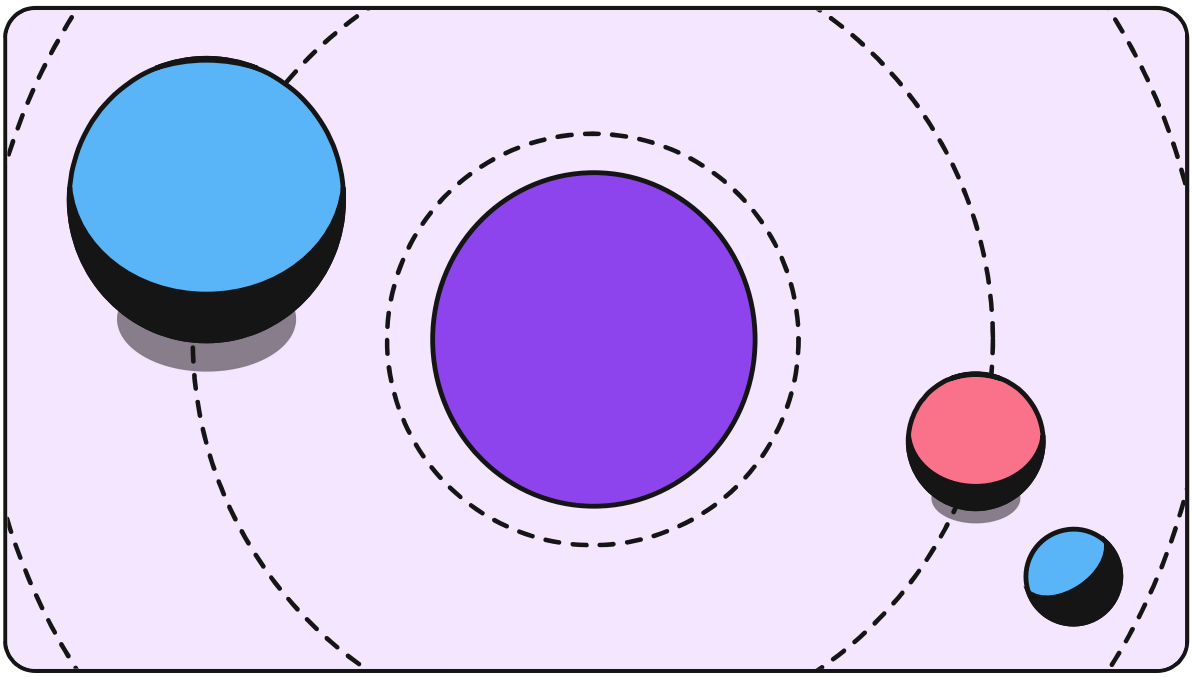


Table of contents

Share article

73

Building a new website or web application from scratch can be tricky, particularly because of the various decisions developers have to make, like “Which CSS framework should I use?” “Should I have a light mode and dark mode switcher?” and so on.

Another important decision that can *truly* affect several aspects of a web project, including its performance, SEO, page load speed, and user experience, is the rendering method you choose. Client-side rendering (CSR) and server-side rendering (SSR) are the most popular rendering strategies for web development.

The question on everyone's mind is, "***CSR vs. SSR: which should I use?***" This guide aims to answer that question.

In this article, we will explore what CSR and SSR are, how they work, their advantages and disadvantages, and most importantly, their differences. In the end, you’ll be able to make an informed decision about which rendering method to use for future projects.

Without further ado, let’s dive in!

What is client-side rendering?

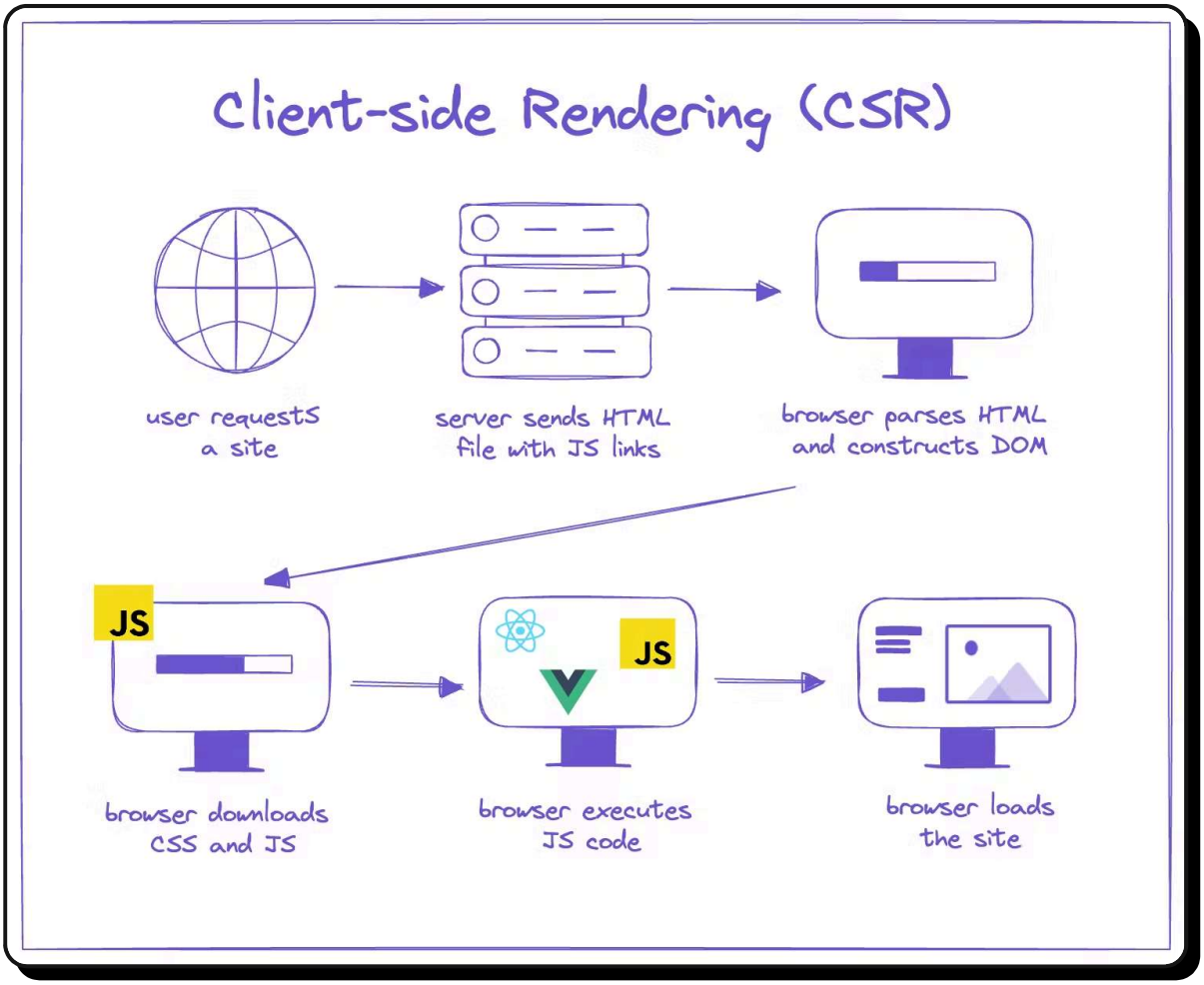
Client-side rendering (CSR) is a JavaScript rendering technique where the final HTML content and user interface (UI) components are generated on the client's browser using JavaScript.

In the CSR approach:

- The server sends an initial HTML file with minimal content
- The client-side JavaScript code fetches data from the server
- The JavaScript code renders the complete UI on the browser

A major advantage of CSR is its ability to create highly interactive and dynamic web applications. This makes it a good fit for applications like chat apps and social media platforms but less of a fit for static websites. For websites though, it's well-suited for single-page applications (SPAs), where content (ex: dynamic prices, live interactions with your content) is frequently updated without requiring a full page reload.

JavaScript frameworks like React, Vue.js, Angular, Svelte, Backbone.js, and Ember.js are commonly used to implement client-side rendering.



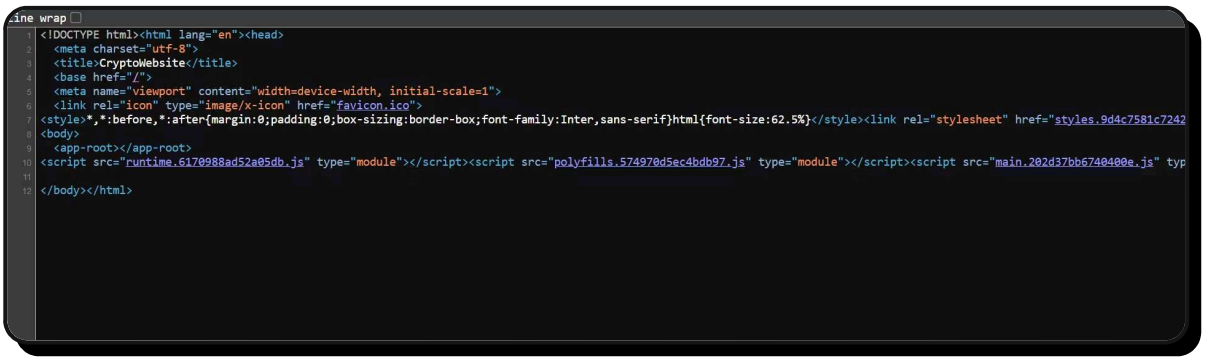
The client-side rendering process typically follows these core steps:

1. A server sends a blank HTML page with links to CSS and JavaScript files to the browser.
2. The browser parses the HTML and constructs the page's Document Object Model (DOM) tree.
3. The browser downloads the CSS and JavaScript resources. After that, it renders the web page and includes necessary elements like text, images, buttons, and styles.
4. The browser executes the JavaScript code to add interactivity and dynamic content like animations, form validations, and data fetched from an API.
5. The browser re-renders and updates parts of the web page based on user interactions like button clicks and form submissions.

Want to know more about client-side rendering?

Explore our in-depth guide to learn more about the client-side rendering process in detail.

Here's the source code of a website built with Angular and rendered on the client. It contains a minimal HTML file, links to the necessary CSS and JavaScript files, and the main <app-root> element where the web page will be loaded.



Advantages of client-side rendering

Provides faster website interactivity

Client-side rendering doesn't stop at rendering a web page on the browser. It also handles updating the page's content on the browser. This means that users can get near-instant updates and feedback when they perform interactions like button clicks and form submissions without requiring a full page reload.

CSR allows us to build fluid and responsive UIs. This makes it a great choice for applications like online games, chat apps, and social media platforms that require frequent content updates without needing a full page reload.

UX tips for developers

The user experience is an important aspect of creating websites and applications. Learn about effective UX tips you can implement as a developer: UX tips for developers.

Reduced server load

With SSR, the server has to fetch the necessary data, styles, and content for a page and render the page. This requires considerable server resources, especially during high-traffic periods.

CSR reduces the server load by shifting most of the rendering to the browser. With CSR, the server's only responsibility is to send the HTML file to the browser, which then handles the rendering. This helps reduce the server's workload significantly.

Great for websites that require dynamic updates

CSR is highly compatible with applications like as real-time data dashboards, collaborative tools, and online games that require frequent and dynamic content updates. CSR ensures that their content stays updated without requiring a full page reload.

Disadvantages of client-side rendering

SEO limitations

During the initial page load, web pages rendered on the client consist of empty HTML files with links to JavaScript resources. This makes it difficult for bots and web crawlers to understand the pages' content and properly index the pages. Essentially, CSR affects a page's visibility, which can lead to poor search engine rankings.

CSR's SEO limitations make it the wrong approach for websites that require high search engine visibility. However, it is a good fit for applications like chat apps, dashboards, and social apps, where SEO is not the priority.

How Next.js can boost SEO

Learn how Next.js can help boost your website's SEO with the built-in SEO-friendly features it offers.

Longer initial loading time

Client-side rendering can lead to longer page load times than server-side rendering because the browser has to download and execute all the necessary JavaScript files before rendering the complete webpage. The user often encounters a blank page or a loading screen while the page loads.

CSR's slow initial page load is especially true for slow network connections and larger applications with extensive JavaScript dependencies. We can use loading code splitting and lazy-loading strategies to reduce the number of assets the browser needs to download and improve the initial page load time.

Caching is not possible until the page is fully loaded

With CSR, the entire page cannot be cached until the JavaScript code has finished rendering the UI. This can impact performance, especially for users with slower devices or network connections.

Users will see a blank page if JavaScript is disabled

If a user has JavaScript disabled in their browser, they will only see the initial HTML file, which may contain minimal or no content, resulting in a poor user experience.

mail@example.com Subscribe

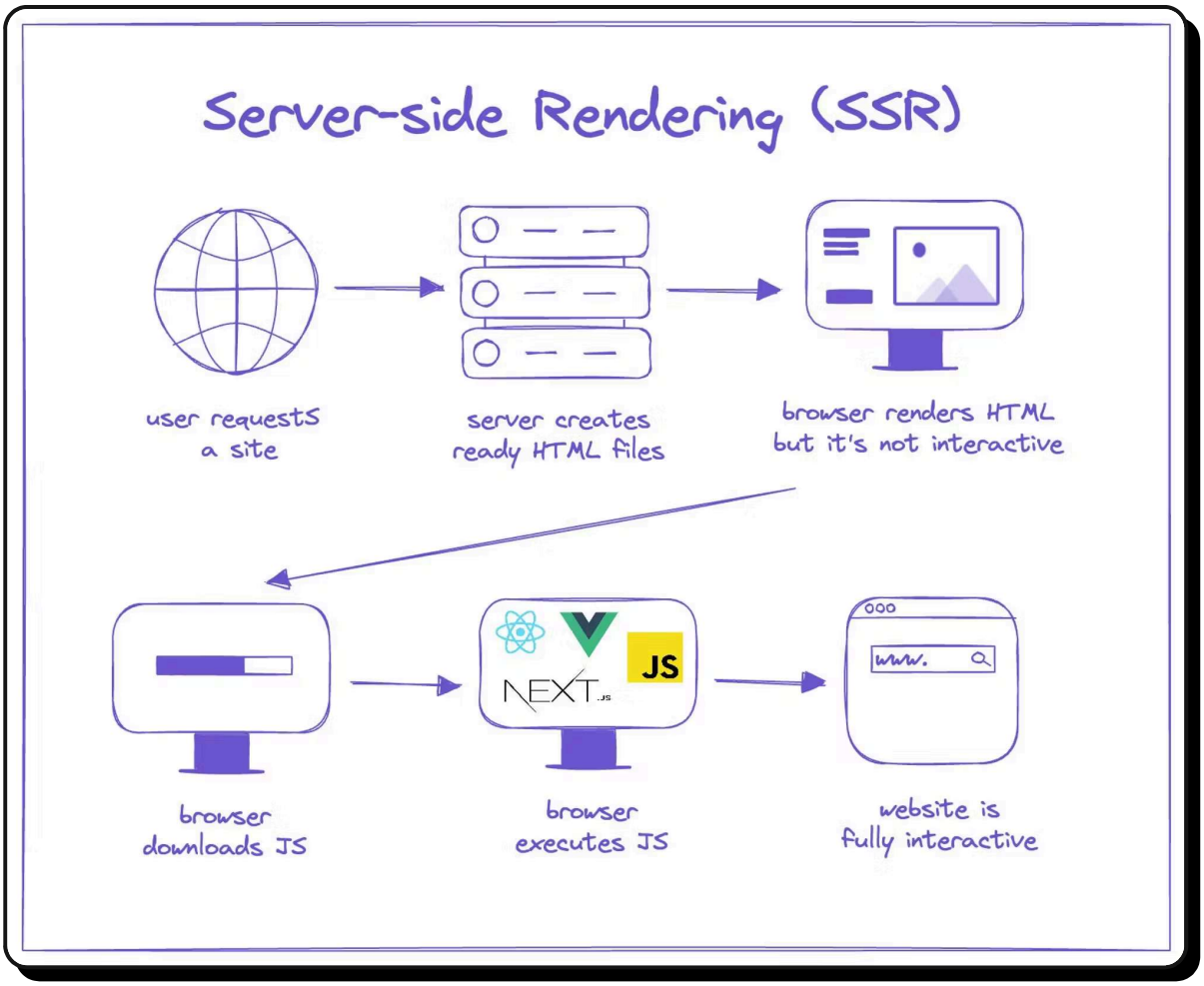
What is server-side rendering?

Server-side rendering (SSR) is a rendering approach where the rendering process occurs on the server.

In this approach, the server generates the complete HTML page with the rendered UI and sends it to the client's browser. The browser then displays the fully rendered page without running any client-side JavaScript code.

SSR is the traditional method of rendering web pages and has been widely used for many years. It is well-suited for applications like ecommerce platforms and landing pages where search engine optimization (SEO) and initial page loads are important.

JavaScript meta-frameworks like Next.js, Nuxt.js, SvelteKit, Angular Universal, Remix, Astro, and Qwik are commonly used to implement client-side rendering.



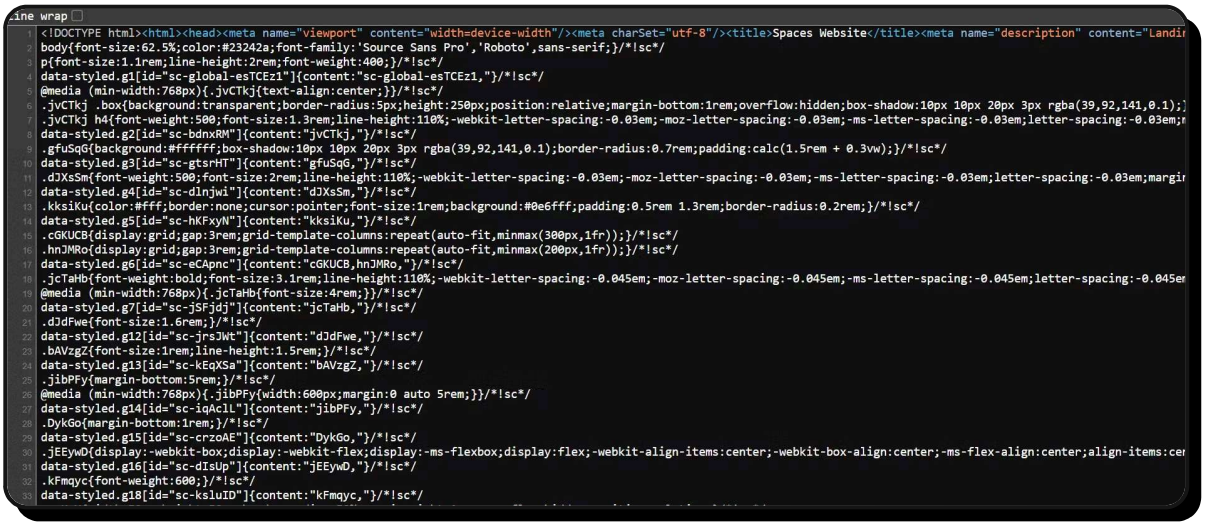
The server-side rendering process typically follows these steps:

1. A server receives a request for a web page, retrieves the necessary data for that page, and populates that data into an HTML template.
2. The server generates the HTML markup for the page, renders its content, and applies the necessary styles.
3. After rendering the page, the server sends the fully rendered page to the browser to display. With SSR, the browser doesn't have to execute any JavaScript code since the initial page load does not require JavaScript.
4. For subsequent user interactions and updates, the client-side JavaScript code takes over and handles the rendering and data fetching.

Want to know more about server-side rendering?

Explore our in-depth guide to learn more about the server-side rendering process in detail.

Below, you'll find the source code of this SSR application built with Next.js. The HTML file was rendered on the server, so it contains all the web page's content, HTML elements, and styles.



Tips for viewing source code

Here are some ways you can view a website's source code: you can use the **Ctrl + U** shortcut (**option+Command+U** for iOS devices) on your web browser or add the "**view-source:**" **prefix to the website's URL.** For example, "view-source: https://prismic.io/"

Advantages of server-side rendering

Improves SEO

According to Search Engine Journal, organic search has the best ROI of any marketing channel, and 1.88 billion websites are fighting for the top position on search engine result pages (SERPs).

Server-side rendering ensures that search engines can easily crawl and index the content of a web page. Since the HTML is fully rendered on the server and sent to the client, search engines can accurately understand, analyze, and rank the page, improving its visibility in search results. This leads to better SEO and higher SERP rankings.

Looking to improve your website's SEO?

Learn how Pallyy used Prismic and Nuxt.js to grow its SEO, which now contributes 90% of its total traffic. Working with Prismic also helped Pally enhance its blog and empower its content writers.

Faster loading times

According to WebFX, 83% of people expect websites they visit to load in 3 seconds or less, and 40% of people will abandon a website that takes over 3 seconds to load. These stats show that failure to meet users' expectations will result in high abandonment.

How does SSR come in? It improves website loading times by moving the rendering process to the server. This eliminates the need for additional JavaScript downloads and client-side rendering processes, resulting in faster initial page loads.

Besides improving the user experience, optimizing loading times is also good for SEO, as the average page load speed of sites ranking on Google's first page is 1.65 seconds.

Looking to improve your website's loading time?

Then, explore the following resources:
- 5 Key Strategies for Reducing LCP Issues
- Core Web Vitals: What They Are and How to Improve Them
- Serving Images in Next-Gen Formats
- 8 Ways to Minimize Main Thread Work to Improve Core Web Vitals
- A Guide to Next.js Image Optimization Using next/image

Great for static websites

SSR is an efficient approach for websites like landing pages, documentation, and blogs with static content. The server can generate and cache the rendered HTML pages, thereby reducing the server.

Build a Next.js landing page

This YouTube tutorial teaches you how to build a simple landing page with Next.js and Prismic. Check it out.

Eliminates loading screens

Since the rendering occurs on the server instead of the browser, SSR eliminates the need for loading indicators and UI skeletons. This is because the initial page load already includes the fully rendered HTML content. The user doesn't have to wait for JavaScript to download, execute, and render the content dynamically.

Eliminating loading indicators helps offer web visitors a better-perceived loading experience since they can quickly view the desired web page without delay. This is a major reason why businesses favor SSR for their

landing pages. However, if needed, we can add loading screens to SSR applications.

Disadvantages of server-side rendering

Increased server load

With SSR, the server is responsible for rendering the entire UI for each request, which can increase the server's workload, especially for applications with high traffic or a many concurrent users.

The SSR approach can be resource-intensive on the server and can put a strain on the server's resources.

Delayed website interactivity

SSR may not provide the same level of interactivity and dynamic user experience as CSR.

As the rendering occurs on the server, any updates or changes in the UI require a round-trip communication between the client and server, potentially causing delays and irresponsive UI. This is one reason why SSR is better suited for websites like blogs and landing pages that focus on displaying static content to users.

Added development complexity

Developing server-side rendered applications can be more complex than client-side ones, and implementing SSR can introduce additional complexity to the development process. It requires understanding how SSR frameworks work, ensuring proper data synchronization between the server and client, and managing the rendering logic on the server.

Cost implications

SSR consumes more server resources since the rendering process takes place on the server. This can lead to increased CPU and memory usage, particularly for high-traffic applications. The increased traffic may require more server resources, which are often costly.

Differences between client-side and server-side rendering

Rendering process

The major difference between CSR and SSR is that CSR's rendering process occurs on the client while SSR's occurs on the server.

With CSR, the server's only responsibility is to serve the blank HTML file. After that, the browser handles the remaining rendering process. However, with SSR, the server renders the page and sends the complete HTML file to the browser.

SEO

Since SSR sends a fully rendered HTML file to the client, web crawlers can properly analyze the page's content, index it, and rank it

appropriately. However, crawlers struggle to properly index web pages rendered on the client since they encounter a minimal page on the first pass.

Initial page load time

CSR has a slower initial page load time because the browser has to first download the required JavaScript code to render the page. On the other hand, SSR offers faster initial page loads since the fully rendered HTML is sent to the client.

Interactivity

CSR offers highly interactive and dynamic user experiences, since the rendering occurs directly in the browser. However, SSR has some interactivity limitations, as any UI changes or updates require a round-trip communication between the client and server.

User experience

Users will likely encounter an empty page or a loading indicator when visiting a client-side rendered page before the JavaScript code fetches the data and renders the UI. However, server-side rendered pages display a fully-rendered HTML page on the first load, which provides an immediate and better visual experience.

Server load

CSR minimizes the server load, as the server's primary responsibility is to serve the initial HTML file. However, SSR can increase server load, as the server needs to render the complete UI for every request.

Number of HTTP requests

With CSR, fewer HTTP requests are made to the server, as the client-side JavaScript code handles most of the UI updates and data fetching. However, with SSR, every page is rendered from scratch, leading to more HTTP requests.

JavaScript dependency

JavaScript must be enabled on the browser for CSR to work. If JavaScript is disabled, the user may see a blank. On the other hand, SSR can still provide a basic user experience, even with JavaScript disabled on the client.

	Client-side rendering	Server-side rendering
Rendering process	Rendering process occurs on the browser using JavaScript	Rendering process occurs on the server
SEO	Harder for search engines to crawl and index content	Easier for SEO as search engines can crawl rendered HTML
Initial page load	Initially loads an HTML shell, then JavaScript bundle is fetched and executed to render the UI	Initially loads a fully rendered HTML page from the server
Initial page load experience	User sees a blank page or loading spinner until the JavaScript bundle is downloaded and executed	User sees the rendered content immediately upon page load
Type of application	Ideal for SPAs, highly interactive web apps like social media platforms and chat apps, and internal apps user dashboards	Ideal for content-heavy websites and apps with limited interactivity requirements like landing pages, ecommerce apps, documentation, and media publications

Framework that support it	Frameworks include React, Angular, Vue, Svelte, Backbone.js, and Ember.js	Frameworks include Next.js, Nuxt.js, Remix, SvelteKit, Angular Universal, Astro, and Qwik. Note that some of these frameworks also support CSR.
Interactivity	Highly interactive and responsive after the initial load, as subsequent interactions are handled client-side without requiring full page refreshes	Initial interactivity is limited to the pre-rendered content; subsequent interactions may require full page refreshes or client-side rendering
Loading speed	Slower initial load time due to fetching and parsing JavaScript files	Faster initial load time as HTML is pre-rendered
Caching	Difficult to cache rendered pages	Easier to cache rendered HTML pages on servers or CDNs
Data fetching	Data is fetched via API calls after the initial load	Data is fetched on the server
Server load	Reduced server load since rendering occurs on the browser	Increased server load since rendering occurs on the server
HTTP requests	Makes fewer HTTP requests to the server	Requires more HTTP requests to the server
JavaScript dependency	Depends heavily on JavaScript	Minimal JavaScript dependency

Conclusion

Client-side rendering and server-side rendering are major rendering techniques used in web development, and they both have pros and cons. Understanding the differences between both approaches is crucial for making informed decisions when building websites and web applications.