JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 38 (2026) 1-5
Received: 9 May 2025
Accepted: 12 November 2025

Comparative analysis of Next.js and Astro frameworks

Patryk Gieda*, Marek Miłosz

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This article presents an analysis of differences in web performance metrics reported by Lighthouse between applications built with Astro and Next.js meta-frameworks, using three rendering strategies: Static Site Generation (SSG), Server-side Rendering (SSR) and Client-side Rendering (CSR). Analysis of the tests showed that the Astro framework performed better on the Total Blocking Time, while Next.js had an advantage on the Largest Contentful Paint. Although there were small differences in the Speed Index, they were noticeable - in favor of Next.js and the SSG rendering strategy in general. The results suggest that there is no single optimal solution for every case. The choice of framework and rendering strategy should be tailored to specific project needs and performance priorities.

Keywords: Astro; Next.js; meta-framework; render strategy

*Corresponding author

Email address: **patryk.gieda@pollub.edu.pl** (P. Gieda)

1. Introduction

Web development has come a long way since the first static HTML pages. Initially, they provided static content with a limited layer of interaction. With the development of JavaScript, websites have become more and more interactive. The Multi Page Application (MPA) model made it possible to build more complex sites. Unfortunately, due to its specificity, this model forces the page to reload during navigation and carries certain limitations in terms of smoothness and performance. Then came the Single Page Application (SPA) model, providing dynamic content updates. As the demand for application development increased, new JavaScript libraries and frameworks were introduced, including React. This ensured efficient and unified application development.

However, Client-side Rendering (CSR), characteristic of SPA sites, had issues with loading the entire application, or Search Engine Optimization (SEO). To improve this while maintaining the advantages of the SPA model, attention was paid to new content rendering strategies: Static Site Generation (SSG), where content is pre-rendered and served as static files; and Server-side Rendering (SSR), where content is rendered on the server side in response to each client request. Due to this, and the increasing complexity of SPA applications, meta-frameworks (higher-level tools that extend the functionality of another framework or library) have been created.

In the State of JavaScript 2024 [1] Next.js achieved the highest “Usage” score in the meta-framework category - 54% and the second Astro reached 23%. It represents the percentage of users using a given technology, among users who indicated at least one meta-framework. In the “Used at Work” ranking, Next.js again took first place with 45%. Astro also deserves a mention, with the highest “Interest” rate, at 67%. It also stands out with the highest “Retention” (the percentage of developers, who still want to work with a given technology), reaching 94%. In terms of the “Positivity” index, it also achieved the highest index - 73% - the percentage of positive ratings compared to neutral and negative ones. Although Next.js has a clear advantage (31 percentage points) over Astro in the "Usage" index, a slowdown in the growth dynamics of its popularity can be observed. In the previous study, it achieved a result of 2 percentage points higher. Astro, on the other hand, recorded an increase of this score by 4 percentage points during over same period. A situation where one technology is a clear market leader, and the other has a group of loyal users, dynamically gaining market share, provides a solid basis for conducting a comparative analysis of both solutions.

2. Purpose and scope of work

The purpose of the study is to determine whether there are significant differences in key performance metrics, between different rendering strategies (SSG, SSR, CSR) in applications built on Astro and Next.js. The scope of the work includes conducting a literature review, designing a research methodology, conducting the study, presenting and analyzing the results, and formulating conclusions. Therefore, the key research question guiding this study is: “Are there significant differences in key performance metrics between Astro and Next.js?”.

3. Literature review

In their study [2], the authors analyzed SSR and CSR rendering strategies in the context of performance and SEO. They point out the advantage of server-side rendering over SEO, while client-side rendering features better accessibility. Similar conclusions were also reached by the authors of another publication [3], who also pointed out the potentially better compatibility of the SSR approach with older versions of web browsers and devices. However, it may result in reduced interactivity and increased server load. The authors of the study [4] discussed the key aspects of SPA application development for SEO, allowing to achieve results equivalent to MPA applications. In the paper describing the rendering strategies used by frameworks such as React, or Svelte [5], we can also read, about the significant performance benefits of limiting website content updates to only those components that actually need to be updated. Paper [6] discusses the possibility of reducing the amount of JavaScript code downloaded by the client, through the use of meta-frameworks and overall improvement in performance results, while also noting the significant trade-offs associated with the use of modern rendering strategies. The importance of minimizing the amount of browser-side JavaScript code is also discussed in another article [7], in the context of the future of web development Also discussed are key architectural factors that developers consider in the context of meta-frameworks, such as extensibility or ease of migration [8].

4. Research methodology

To conduct the study, two equivalent prototype applications were implemented: based on Astro and Next.js. Each with three subpages. Each of the subpages in both applications was built from possibly identical React components. The applications were implemented based on their latest stable versions. The latest possible versions of the dependencies have been used (Table 1).

Table 1: Specification of framework versions and packages

Dependency	Astro	Next.js
Framework	5.1.5	15.1.4
React	19.0.0	19.0.0
React DOM	19.0.0	19.0.0
Node.js	22.13.0	22.13.0
TypeScript	5.7.3	5.7.3
Supabase Client	2.47.12	2.47.12
TailwindCSS	3.4.17	3.4.17
React Integration	@astrojs/react@4.1.3	built-in
Vercel Integration	@astrojs/vercel@8.0.2	built-in

A common database was created to ensure comparable testing conditions. Both applications use common resources provided by the Supabase platform as a data source. As a result, each subpage of the application is identical both visually and in terms of its content. Both applications have been published on the Vercel platform.

4.1. Structure of applications

Both applications have three subpages that share the same user interface, functionality, and content. The only difference is the rendering strategy used:

- /ssg - using the SSG method,
- /ssr - using the SSR method,
- /csr - using the CSR method.

The implementations of these methods follow the official documentation of both frameworks [9, 10]. The differentiating factors in the context of this study are:

- application level: the meta-framework used,
- subpage level: the rendering methods.

The top navigation bar and footer are common to the entire application. Each subpage has the same page header containing a photo, title, and subtitle. The main content contains a grid of 27 articles, each with a photo, title, and short description.

The test page was moderately complex, generating a total of 471 Document Object Model (DOM) elements in the document tree. Each subpage loaded an identical set of static resources, including a 167 kB Cascading Style Sheets (CSS) stylesheet and 28 images totaling 2.58 MB. The key difference between the analyzed applications was the size of the JavaScript packages, resulting from the nature of the meta-frameworks used. The application built with Astro had a total script size of 111 kB, and the one built with Next.js measured 270 kB.

The test application used a database based on the Supabase platform, which uses PostgreSQL. The database schema was deliberately simplified and consisted of two tables: articles (storing articles) and pages (storing subpages), reflecting the structure typical of a simple Content Management System (CMS). For the study, each subpage used the same record from the pages table.

4.2. Testing Environment

All research activities were performed on the test bench whose specifications are listed in Table 2. The tests were performed using the Google Chrome browser and the Lighthouse Command Line Interface (CLI) [11] - a tool that allows you to measure many metrics that evaluate a website's performance from various aspects.

Table 2: Summary of Test Bench Specification

Specification	Value
Model Name	MacBook Pro (14-inch, 2021)
Operating System	macOS 15.2
CPU	Apple M1 Pro
RAM	LPDDR5 16 GB
Storage	SSD 512 GB
Network Connectivity	Wi-Fi 6 (802.11ax)
Web browser	Google Chrome 132.0.6834.72
Lighthouse CLI	12.3.0
Node.js	22.13.0
Python	3.13

4.3. Metrics

The metrics used in this study were obtained via the Lighthouse tool. The measured metrics also include Core Web Vitals [12]. This is a set of metrics developed by Google that are critical to the user experience. These metrics are not fixed and change over time [13]. This study analyzed the following five metrics, which are components of the Lighthouse tool's overall performance score:

- Total Blocking Time (TBT) ms - the total time the main thread is blocked by tasks longer than 50ms,
- Largest Contentful Paint (LCP) ms - measures the time it takes to render the largest piece of content in the viewport,
- Cumulative Layout Shift (CLS) no unit - a measure of the movement of visible page elements during rendering,
- First Contentful Paint (FCP) ms - determines when the first page element is displayed,
- Speed Index (SI) ms - a general measure of rendering speed.

4.4. Test procedure

To obtain reproducible results, data was collected using a Python script running on the test bench (Table 2.). To average the results, 100 iterations were made over the array of target Uniform Resource Locator (URL) addresses. During each iteration, a query was sent to each address in a random order (using Lighthouse CLI). This avoids any possible impact of the test order on the results. To minimize the impact of hosting performance, a one-second pause was taken after each query, and an additional second after each full iteration.

All of the JavaScript Object Notation (JSON) files were saved with the full Lighthouse reports, to allow for further analysis if needed. Next, the values of the tested metrics were saved to a separate file, along with the test identifiers (URL, framework tested, rendering type, iteration number, timestamp). A set of this data was saved to a Comma-separated Values (CSV) formatted file, which provided the basis for further statistical analysis.

5. Results

The following section presents the results of the tests performed.

5.1. Total Blocking Time

Figure 1 shows the distribution of TBT values for all six variants (Astro and Next.js in the SSG, SSR, and CSR strategies). Astro is much lower in most cases, while Next.js (especially in SSG and SSR) tends to show more variation in the upper registers.

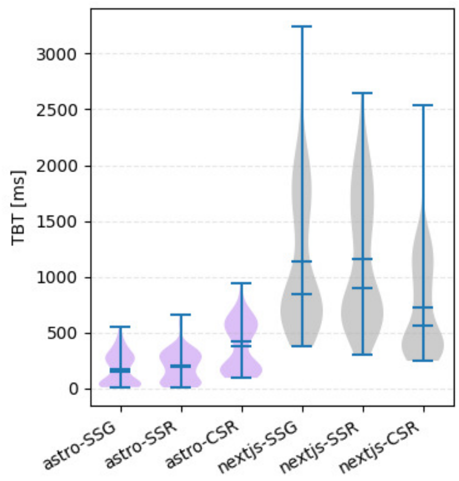

Figure 1: Graphical distribution of Total Blocking Time values.

Detailed statistics describing TBT are presented in Table 3. In turn For Astro-SSR, the mean is 198 ms with a median of 204 ms, while for CSR strategy the mean reaches 382 ms with a median of 420 ms. The Next.js-SSR variant, on the other hand, has a mean of 1162 ms and a median of 898 ms, with a standard deviation (std) of 586 ms.

Table 3: Total Blocking Time results for all variants

app	render	mean (ms)	med (ms)	std (ms)	q75 (ms)	p95 (ms)
Astro	SSG	173	152	142	286	416
	SSR	198	204	139	302	389
	CSR	382	420	221	578	694
Next.js	SSG	1137	843	617	1657	2077
	SSR	1162	898	586	1686	2107
	CSR	724	568	416	1095	1357

5.2. Largest Contentful Paint

LCP results are presented in Figure 2. We can see that much higher values are observed for Astro for each rendering strategy.

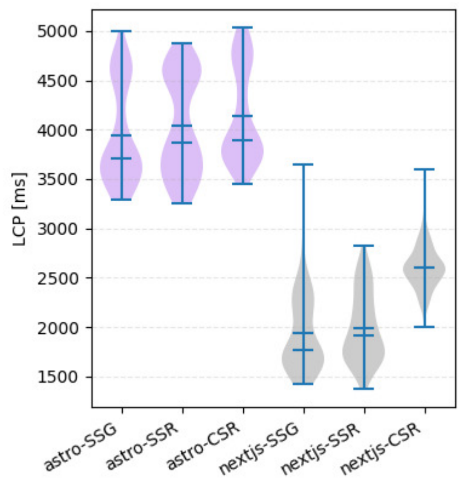

Figure 2: Graphical distribution of Largest Contentful Paint values.

For Astro-SSR, the mean is 4037 ms, with a median of 3872 ms and a 95th percentile (p95) of 4710 ms. The Next.js-CSR variant has a mean of 2602 ms and a median of 2600 ms; in this case, the p95 reaches 2938 ms. The statistics are summarized in Table 4.

Table 4: Largest Contentful Paint results for all variants

app	render	mean (ms)	med (ms)	std (ms)	q75 (ms)	p95 (ms)
Astro	SSG	3939	3710	495	4554	4729
	SSR	4037	3872	502	4577	4710
	CSR	4137	3896	474	4683	4901
Next.js	SSG	1938	1763	410	2242	2499
	SSR	1993	1919	349	2228	2636
	CSR	2602	2600	221	2716	2938

5.3. Cumulative Layout Shift

Due to the small scatter in the results of the CLS metric, this study is limited to Table 5. For the CSR rendering strategy of both frameworks, the mean value is 0.181, which is the same as the median and p95 values, while values of 0 are observed for the other variants.

Table 5: Cumulative Layout Shift results for all variants

app	render	mean	med	std	q75	p95
Astro	SSG	0	0	0	0	0
	SSR	0	0	0	0	0.
	CSR	0.181	0.181	0	0.181	0.181
Next.js	SSG	0	0	0	0	0
	SSR	0	0	0	0	0
	CSR	0.181	0.181	0	0.181	0.181

5.4. First Contentful Paint

Figure 3 shows the FCP distribution of FCP metric.

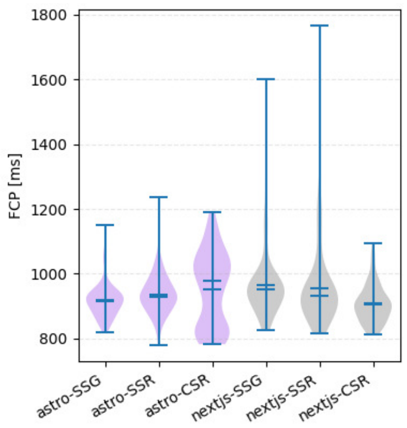

Figure 3: Graphical distribution of First Contentful Paint values.

From Table 6, we can see that the mean for Astro-CSR is 952 ms with a median of 978 ms, and for Next.js-CSR the mean is 907 ms, with a median of 904 ms. In Astro-SSG, p95 reaches 1006 ms, while in Next.js-SSG it is 1166 ms.

Table 6: First Contentful Paint results for all variants

app	render	mean (ms)	med (ms)	std (ms)	q75 (ms)	p95 (ms)
Astro	SSG	916	918	52	939	1006
	SSR	935	930	61	971	1029
	CSR	952	978	114	1038	1134
Next.js	SSG	965	950	104	976	1166
	SSR	956	930	124	969	1191
	CSR	907	904	59	934	1024

5.5. Speed Index

The distribution of SI values is shown in Figure 4.

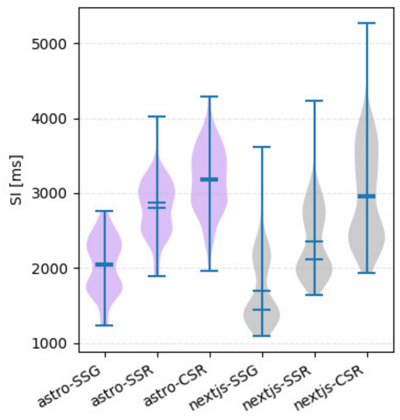

Figure 4: Graphical distribution of Speed Index values.

For Astro-SSR, the mean SI is 2800 ms, the median is 2876 ms, and p95 reaches 3255 ms. In contrast, the Next.js-CSR variant has a mean of 2966 ms, the median reaches 2936 ms, and the 95th percentile reaches 3874 ms. The detailed results are shown in Table 7.

Table 7: Speed Index results for all variants

app	render	mean (ms)	med (ms)	std (ms)	q75 (ms)	p95 (ms)
Astro	SSG	2028	2056	376	2353	2598
	SSR	2800	2876	384	3073	3255
	CSR	3170	3198	435	3490	3821
Next.js	SSG	1698	1445	470	2113	2390
	SSR	2358	2116	478	2732	3063
	CSR	2966	2936	643	3516	3874

6. Conclusions

Based on the results, it can be seen that the choice of rendering strategy (SSG, SSR, CSR) and the choice of framework (Astro or Next.js) have an impact on key performance metrics. In most cases, Astro shows lower Total Blocking Time values, which can be beneficial from the perspective of responsiveness to user actions. However, the Largest Contentful Paint metric showed significantly higher values in Astro, indicating increased display time for the largest element on the page. The analysis of Cumulative Layout Shift showed no significant differences between the variants; only in the CSR strategy did a measurable value appear for both frameworks. About First Contentful Paint, the results are quite similar for all variants. On the other hand, based on the Speed Index, we can see an overall advantage for the SSG rendering strategy. The worst-performing strategy was CSR. In addition, the Next.js framework shows an advantage over Astro for both SSG and SSR strategies.

The previously posed research question: "Are there significant differences in key performance metrics between Astro and Next.js?" should therefore be answered in the affirmative. The collected data suggests that there is no single, universally optimal solution. Instead, the choice of a specific strategy and framework should be determined by specific project priorities. In practice, this means adjusting the approach accordingly: when responsiveness is paramount, Astro may be the better option. In turn, in cases where fast rendering of the most critical content is crucial, using Next.js may turn out to be more beneficial.

References

[1] State of JavaScript 2024 – meta-frameworks, https://2024.stateofjs.com/en-US/libraries/meta-frameworks, [25.12.2024].

[2] T. F. Iskandar, M. Lubis, T. F. Kusumasari, A. R. Lubis, Comparison between client-side and server-side rendering in the web development, IOP Conference Series: Materials Science and Engineering 801(1) (2020) 012136, https://doi.org/10.1088/1757-899X/801/1/012136.

[3] D. Verma, P. Aland, A comparative review of server rendering and client side rendering in web development, International Journal of Scientific Research & Engineering Trends 10(2) (2024) 521–530.

[4] K. Kowalczyk, T. Szandala, Enhancing SEO in single-page web applications in contrast with multi-page applications, IEEE Access 12(2024) 11597–11614, https://doi.org/10.1109/access.2024.3355740.

[5] R. Ollila, N. Mäkitalo, T. Mikkonen, Modern web frameworks: A comparison of rendering performance,

Journal of Web Engineering 21(3) (2022) 789–813, https://doi.org/10.13052/jwe1540-9589.21311.

[6] T. Lonka, Improving the initial rendering performance of React applications through contemporary rendering approaches, Master thesis, Aalto University, Espoo, 2023.

[7] J. Vepsäläinen, A. Hellas, P. Vuorimaa, The rise of disappearing frameworks in web development, In International Conference on Web Engineering (2023) 319–326, https://doi.org/10.1007/978-3-031-34444-2_23.

[8] J. H. Noor, The effects of architectural design decisions on framework adoption: A comparative evaluation of meta-frameworks in modern web development, Master thesis, Aalto University, Espoo, 2024, http://doi.org/10.13140/RG.2.2.10552.97287.

[9] Astro framework documentation, https://docs.astro.build, [25.12.2024].

[10] Next.js framework documentation, https://nextjs.org/docs, [25.12.2024].

[11] Introduction to Lighthouse – an open-source audit tool, https://developer.chrome.com/docs/lighthouse/overview, [25.12.2024].

[12] Core Web Vitals, https://web.dev/articles/vitals, [25.12.2024].

[13] Lighthouse tool performance scoring overview, https://developer.chrome.com/docs/lighthouse/performance/performance-scoring, [25.12.2024].