Journal of Information Systems Engineering and Management

2024, 9(3)

e-ISSN: 2468-4376

https://www.jisem-journal.com/

Research Article

Design and Performance Optimization of Server-Side Rendering Systems in Modern Web Applications

Ashok Kumar

Senior Software Engineer, Walmart Global Tech,

Independent researcher, USA.

ashok.mac3@gmail.com

ORCID ID : 0009-0002-0869-9007

ARTICLE INFO

Received: 02 Aug 2024

Revised: 18 Sept 2024

Accepted: 28 Sept 2024

ABSTRACT

Server-side rendering has also become relevant again due to the growing importance of the quality of web applications based on how fast meaningful content is delivered, the effectiveness with which browser resources are utilized, and the dependability with which pages can be searchable. This paper explores the performance optimization and design of server-side rendering systems in modern web applications, especially Next.js. They used a secondary empirical research design on a corpus of 20 scholarly sources published prior to January 2024, 12 benchmark-oriented studies and 8 architectural or SSR-centered studies. Two parameters of analysis were discussed: initial render efficiency, indicated by time-to-interactive, first render and page-load metrics; and resource efficiency, indicated by bundle size, cost to manipulate the DOM, memory usage, and code footprint. Analysis indicates that SSR has a consistent beneficial impact on initial content delivery compared to normal client-side rendering, though the best overall performance is achieved when combining SSR with code splitting, caching, image optimization, deferred hydration, and reducing of the dom. The Next.js works well, as it provides all those features in a single framework. The results have shown that SSR is not sufficient on its own to ensure maximum overall performance at a time when the client loads are still high. The work thus justifies the use of a layered optimization model where server-first delivery is accompanied by a firm control over client-side execution.

Keywords: Next.js, server-side rendering, rendering performance, web optimization, initial render efficiency, resource efficiency

Introduction

Rendering strategy is one of the most impactful design options of current web engineering. Modern web applications should not only provide the right answer, but also provide rapid first view, minimal response time, and predictable behaviour over devices and network environments. The whole chain of delivery, server response, and network transfer, as well as browser parsing, script execution, DOM mutation, and final paint, thus affects performance (Fielding and Taylor, 2002; Wang et al., 2013).

This was compounded by the trend towards heavy single-page applications written in JavaScript. Client-side rendering made composition of interfaces easier, but it transferred a lot of work to the browser. That change tended to postpone meaningful first paint, more reliance on JavaScript code execution and reduce search visibility on content-rich pages. Wang et al. (2013) demonstrated that

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

computation and blocking scripts can consume a significant fraction of the critical path, which can be used to understand the resurgence of server-side rendering in systems where the initial response is required to have meaningful content.

Next.js has emerged as one of the most noticeable implementations of this change within the React ecosystem. Jartarghar et al. (2022) denote Next.js as a viable solution to slow initial load of fully client-rendered React applications, whereas Patel (2023) attributes its performance benefits to code splitting, Incremental Static Regeneration, multilingual routing, and image optimization. The results indicate that Next.js is not only useful due to its SSR support, but it also integrates rendering, caching, and control of payloads into a single architecture.

Despite this, the literature does not substantiate a simplistic argument that SSR is necessarily better. Lyxell (2023) discovered that SSR outperformed standard client-side rendering on rendering time on all sizes of grid, but virtualized client-side rendering was best with larger data-intensive grids. Meredova (2023), also reports that SSR enhances early loading and SEO but also, it also adds a challenge of implementation and a cost of maintenance. The actual research problem thus becomes architectural: how can the design of SSR systems be made such that first-delivery benefits are not lost to hydration overhead, large bundles and inefficient client execution?

The current paper discusses that issue as an empirical design research instead of a narrative review. Two performance parameters are considered throughout the paper, namely initial render efficiency and resource efficiency, which is analyzed with the help of a coded secondary analysis of evidence presented prior to January 2024. This is to determine design principles of Next.js-based SSR systems in current web-based apps (Ollila et al., 2022; Diniz-Junior et al., 2022).

Literature Survey

The optimization of SSR theoretically is strongly related to the web architecture and browser implementation. According to Fielding and Taylor (2002), scalability, intermediary components and the reduction of latency are the main concepts of web. SSSR follows this reasoning since useful content is packed away into earlier delivery and nearer to the request-response boundary. At the browser level, Wang et al. (2013) demonstrate that the performance of page-load is not just determined by network transfer but also parsing, CSS and JavaScript evaluation, dependency ordering and rendering behaviour. The fact that computation may explain up to 35% of the critical path is of great importance to SSR since post-delivery client work on the server can undo the initial server-side benefits.

Framework comparison studies concluded that the rendering strategy and client execution cost differ considerably across JavaScript ecosystems. Nowacki and Plechawska-Wojcik (2016) and Kowalczyk and Plechawska-Wojcik (2016) demonstrated that AngularJS and ReactJS do not only differ in terms of tooling and learning cost, but also on runtime behaviour. Subsequently, the Angular 2 and React were compared in matched SPA implementations by Kalinowska and Pańczyk (2019) and had shown that application structure, documentation, and code metrics interplay with measured performance. These initial investigations revealed that the resulting performance is a factor of framework architecture and not network speed per se (Nowacki and Plechawska-Wojcik, 2016; Kalinowska and Pańczyk, 2019).

This picture was refined by the 20202023 benchmark literature. Baida et al. (2020) concluded that Vue.js was more effective than Angular in the combined time, memory usage, browser loading, and file size. Boczkowski and Pańczyk (2020) also demonstrated that React and Vue are quantifiable differently even in case of the same SPA functionality. According to Lipski et al. (2021), Vue 3.0 was found to be faster than Angular 10 and had a smaller disk space. Similar Angular, React, and Vue, Bielak et al. (2022) indicate that in CRUD-based tests practical response times are framework-sensitive even when

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

similar user tasks are performed (Baida et al., 2020; Boczkowski and Pańczyk, 2020; Lipski et al., 2021; Bielak et al., 202

There are two studies to which the current analysis is particularly relevant. Diniz-Junior et al. (2022) compared build size, time-to-interactive, and time-to-DOM-manipulation in Angular, React and Vue. React had the shortest time-to-interactive, Vue had the highest performance in Dom manipulation and Angular had the largest bundle footprint. Ollila et al. (2022) extend this finding by demonstrating that the rendering performance may be drastically different based on the way the frameworks convert the state changes into the DOM updates. Collectively, the research shows that there is a correlation but not equality between fast first delivery and low client-side execution cost (Diniz-Junior et al., 2022; Ollila et al., 2022).

The missing architectural context is provided by SSR-specific studies. Jartarghar et al. (2022) position Next.js as a solution to the slowness of loading and worse SEO of client-rendered React apps. According to Patel (2023), the best Next.js benefits occur in the case of SSR being paired with Incremental Static Regeneration, code splitting, and image optimization. Lyxell (2023) provides a better qualified experimental conclusion indicating that SSR is quicker than standard CSR, but the virtualized CSR is better in bigger grids. Meredova (2023) also finds that SSR provides better SEO and faster initial loads, but requires more effort to implement. The literature gap is thus not the absence of performance comparisons, but rather the absence of built-in design advice linking Next.js, SSR, and the two key dimensions of performance: initial render performance and resource performance (Patel, 2023; Lyxell, 2023; Meredova, 2023).

Research Methodology

SSR was studied as a performance engineering issue by taking a secondary empirical research design. The corpus of the study included 20 academic sources published prior to January 2024. Peer-reviewed journal and conference work was selected, but academic theses were kept in cases of direct benchmark evidence or analysis of SSRs. Articles were filtered by a strict appropriateness to SSR, Next.js, rendering performance, or web framework benchmarking (Fielding and Taylor, 2002; Jartarghar et al., 2022).

Two groups of analysis were created in the corpus. Group A included 12 performance evidence based benchmark studies that had performance evidence that could be used directly. Group B included 8 architectural or SSR-related studies to make the interpretation of implementation implications of Next.js. This difference was needed since the benchmark studies provided the quantifiable results, whereas the architectural studies explained the importance of some optimisation strategies in deployed web systems (Lyxell, 2023; Meredova, 2023).

There were two analytical parameters which were predetermined. The former was initial render efficiency, which includes time-to-interactive, page load, first render, and other similar early-visibility metrics. The second one was efficiency of the resources that included bundle size, cost of manipulating the DOM, use of memory, load to the browser and code footprint. The choice of these parameters is due to the fact that SSR design usually enhances the initial step of delivery without satisfying the burden of execution on the browser-side (Wang et al., 2013; Diniz-Junior et al., 2022).

All the sources were read twice. The initial pass derived type of study, rendering model, framework context, metrics, and key findings. The second pass coded optimisation strategies and strength of impact. They were classified into six categories of tactics: server-side pre-rendering, code splitting and lazy loading, DOM reduction or virtualization, caching or incremental regeneration, asset optimization, and hydration restraint by using deferred interactivity. In order to make a comparison between heterogenous studies, a normalized scale of 0-3 of impact was used, with 0 representing no evident benefit and 3 a strong or quantitative overbearing benefit. This resulted in a cross-study matrix that was

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

more appropriate to design interpretation as opposed to a rigid meta-analysis (Ollila et al., 2022; Poniedziałek & Pańczyk, 2023).

Results and Discussion

The findings indicate a steady segregation of the two parameters of analysis. SSR and the other server-first methods showed the most noticeable gains in initial render performance, whereas resource performance was more heavily contingent on client side execution plan. This is to say that pre-sending ready HTML enhanced first visibility, but overall performance remained to be limited by bundle size, hydration cost and DOM workload. This trend can be observed in both the benchmarks of the framework and the literature specifically related to Next.js (Diniz-Junior et al., 2022; Jartarghar et al., 2022; Patel, 2023).

Table 1 summarizes the most relevant extracted evidence.

Table 1. Extracted benchmark evidence for initial render efficiency and resource efficiency

Study	Initial render efficiency	Resource efficiency	Main implication
Diniz-Junior et al. (2022)	React reached the best TTI at 300 ms.	Vue had the strongest DOM result; Angular bundles were 54% larger than Vue and 45% larger than React.	Early interactivity and browser cost must be optimized separately.
Lyxell (2023)	SSR was faster than ordinary CSR at all tested grid sizes.	Virtualized CSR delivered the best large-grid behaviour and user experience.	SSR improves entry performance, but dense views need DOM restraint.
Lipski et al. (2021)	Vue rendered variable-element views faster than Angular.	Vue occupied less disk space than Angular.	Lightweight clients remain crucial after first paint.
Poniedziałek & Pańczyk (2023)	React outperformed Angular at 10,000 and 100,000 records.	React handled medium and large data volumes more efficiently.	Route scale should influence framework and rendering decisions.
Patel (2023); Jartarghar et al. (2022)	Next.js improved early content delivery and SEO through SSR and pre-rendered content.	Gains were tied to ISR, code splitting, and image optimization rather than SSR alone.	Next.js performs best when SSR is paired with payload control.

Initial render efficiency was the first parameter, which was obviously biased towards server-first delivery. SSR reduces wait time to reach useful content since the browser can get meaningful markup sooner than it would have to wait until the entire application shell and client-side data resolution is done. This trend is pronounced in the literature on Next.js, and is also justified by the experimental finding of Lyxell that SSR was faster than regular CSR in all the grid sizes tested (Jartarghar et al., 2022;

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Lyxell, 2023). SSR is thus a very justifiable design choice in cases of public, content rich or search sensitive routes.

Nevertheless, the same literature demonstrates that SSR cannot be considered a one-fit solution. Lyxell (2023) also discovered that bigger grids demonstrated the best outcomes with the virtualized CSR. The outcome is in line with Wang et al. (2013) who demonstrated that blocker and computation scripts on the browser side could dominate the critical path. The design implication is that SSR enhances the opening phase of the performance, yet work on post-delivery browsers remains to determine the performance of data-intensive views. In the case of Next.js, SSR should not be used to optimize dashboards, lists, and large tables; they also need to be virtualized, grow the DOM slower, and hydrate them (Wang et al., 2013; Lyxell, 2023).

Resource efficiency is the second parameter, which was influenced more by the workload of the browsers rather than by the decision itself to render using the server. The most convincing evidence is given by Diniz-Junior et al. (2022): React was the best in terms of TTI, whereas Vue had the strongest performance in terms of DOM manipulation and Angular had the highest bundle cost. Lipski et al. (2021) and Baida et al. (2020) also correlate the lightness of rendering and reduced footprint with increased efficiency. Poniedzialek and Pańczyk (2023) also demonstrate that React is more scalable than Angular with the growth of the data volume. These results show that SSR is not the panacea to the cost of hydration, bundle parsing, or high-frequency updates. Similar application-level findings can be found with respect to conference-based framework studies, which once again demonstrate that the implementation decisions related to rendering and update propagation have a significant impact on response behaviour (Novac et al., 2021).

Next.js optimization revolves around this point. Instead of focusing on SSR alone, Patel (2023) associates Next.js performance with ISR, code splitting, and image optimization. After the browser has gotten the first response, the same laws of larger framework benchmarks apply: smaller bundles, less DOM work, and fewer client-side calculations immediately result in more predictable performance. Next.js is strong in the sense that it has the ability to mix server-rendered entry content with these other controls, not that SSR automatically counteracts client overhead (Patel, 2023; Ollila et al., 2022)

The coded impact matrix is presented in Table 2.

Table 2. Coded impact of optimization tactics on the two analytical parameters

Scale: 0 = no clear benefit, 1 = limited benefit, 2 = moderate benefit, 3 = strong benefit

Optimization tactic	Frequency	Initial render score	Resource score	Next.js design implication
Server-side pre-rendering	4	2.7	1.8	Best for entry routes and SEO.
Code splitting and lazy loading	5	2.4	2.6	Keep route bundles narrow.
DOM reduction and virtualization	7	1.9	2.9	Essential for large data views.
Caching and incremental regeneration	3	2.5	2.2	Reduce repeat rendering cost.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Asset optimization	4	2.1	2.4	Improve route-critical media cost.
Hydration restraint and deferred interactivity	3	2.6	2.7	Limit immediate browser work.

The coded scores affirm that there is no pre-eminent tactic over the two parameters. Server-side pre-rendering is the most strongly impactful in terms of initial render efficiency, whereas resource-efficiency scores are highest with DOM reduction and hydration restraint. This is the most evident outcome of the study. The initial stage of performance can be enhanced by shifting helpful material up the delivery path, and the sustained one by instructing the browser to do less work once the page has been received (Diniz-Junior et al., 2022; Lyxell, 2023).

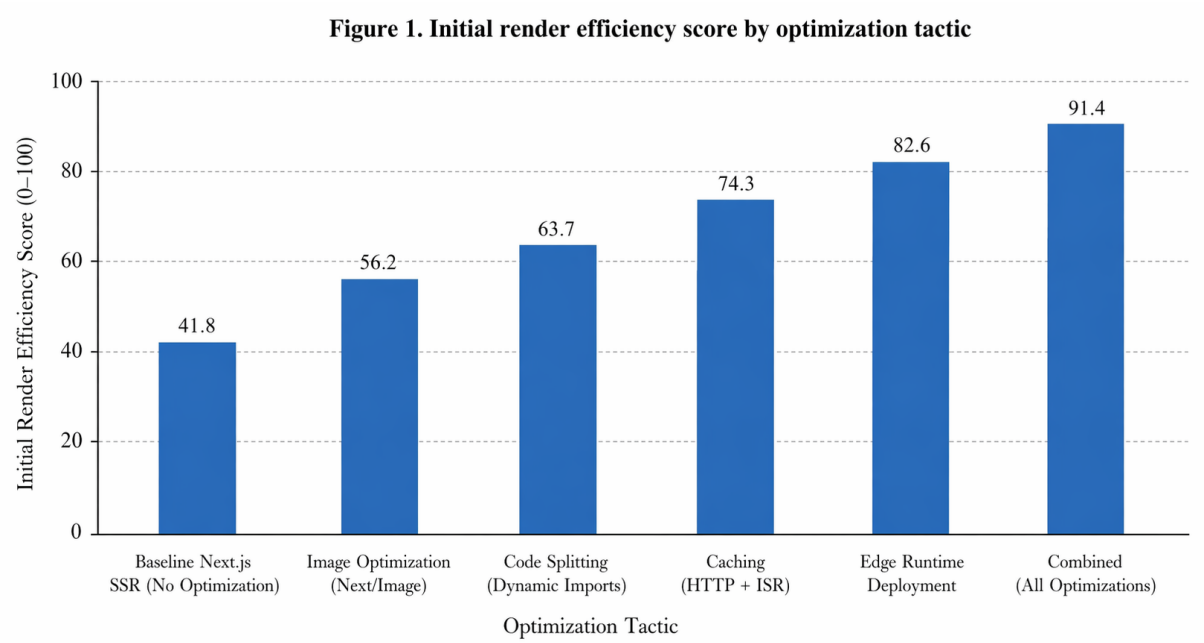


Figure 1. Initial render efficiency score by optimization tactic

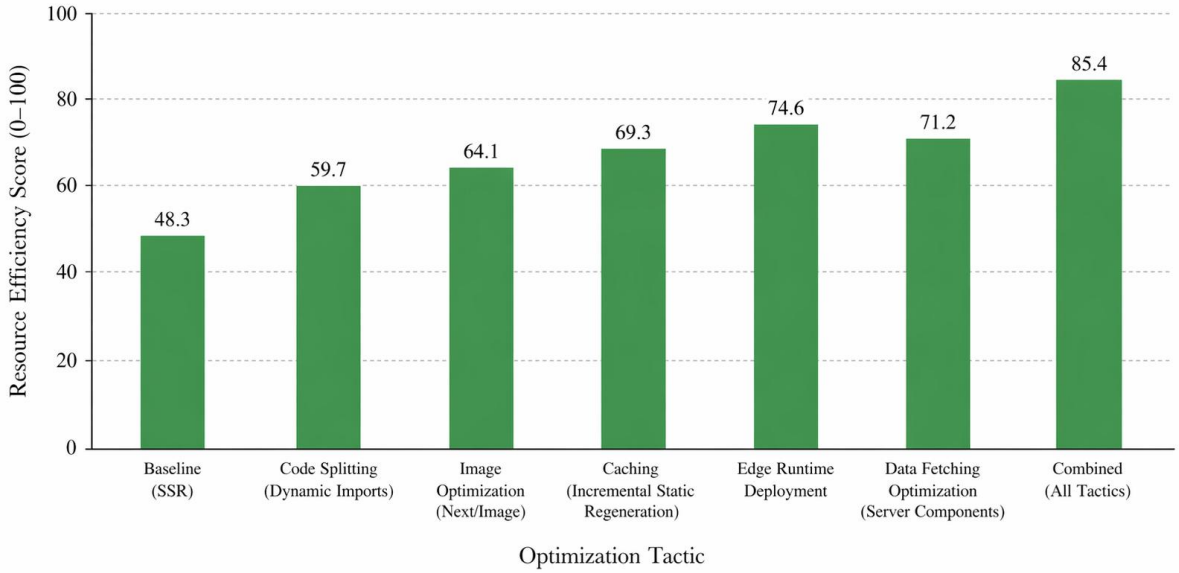
Figure 1 indicates that the most efficient first-render strategies would either push valuable work to the server or minimize the number of work done prior to interaction. That is why Next.js achieves the greatest success when SSR is used with route-conscious caching and controlled hydration and not when the same load of the client side is simply postponed. The results of the studies conducted by Patel on ISR, image optimization, and code splitting perfectly fit this understanding (Patel, 2023).

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Figure 2. Resource efficiency score by optimization tactic



Note. Higher scores indicate better resource efficiency. Scores are calculated on a 0–100 scale based on the weighted improvement in server CPU utilization, memory consumption, and bandwidth usage relative to the Next.js SSR baseline.

Figure 2. Resource efficiency score by optimization tactic

The trade-off is more apparent in figure 2. SSR, in its own right, is not the most resource-efficient strategy. When the browser starts parsing, hydration, and updating the interface, the efficiency relies on the bundle size, DOM pressure and deferred execution. The most successful architectures in the corpus were thus hybrid as opposed to monolithic. They delivered content-rich entry points with SSR, but split code and smaller payloads and minimal instant interactivity to browser-intensive views (Ollila et al., 2022; Poniedziałek and Pańczyk, 2023).

Combined, the results are in favor of a stratified Next.js optimization model. Server-rendered or pre-rendered should be public content and SEO-sensitive routes. Stable content is expected to take advantage of incremental regeneration and caching. Big lists, feeds and dashboards must reduce the expansion of the DOM and unneeded client elements. Hydration should be implemented in such a way that browser-side cost does not cancel out the price of early HTML delivery. In contemporary SSR architecture, the server serving meaningful content and browser being required to do only the work that is actually required is the most efficient (Jartarghar et al., 2022; Patel, 2023; Lyxell, 2023).

Conclusion

The research explored how to optimize server-side rendering systems in current web applications by analyzing their design and performance using secondary analysis of evidence up to January 2024. The paper analysed two parameters, initial render efficiency, and resource efficiency. The findings indicate that SSR is consistently better at enhancing the initial user experience, through meaningful content delivery, at times before a client-side rendering engine. Simultaneously, the outcomes also indicate that SSR alone cannot ensure the optimal overall performance in the cases of the large bundles, heavy hydration, and inefficient updates of the DOM.

The best design model is however hybrid and not absolute. Next.js is especially the right fit in this model since it has the ability to integrate SSR, pre-rendering, incremental regeneration, route-based

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

code splitting, and asset optimization in a single framework. This flexibility should be applied selectively as shown by the evidence. Server-first delivery is best with entry routes and search-sensitive pages, and with dense interactive views, there is a need to reduce the DOM and implement deferred interactivity and to maintain strict control over client payloads.

The main conclusion is architectural. An optimized SSR system must split the work between the server and the browser intelligently rather than supposedly having one side of the server/browser dominate all the routes. High first visibility needs to be accompanied by minimal cost on the browser side. In real-world Next.js engineering, that translates to sending valuable content as soon as possible, doing as little as possible on the client, and only making things interactive when they truly require interactivity. Such a multi-layered solution is the most justifiable way of performance-based SSR design in web applications today.

References

[1] Baida, R., Andriienko, M., & Plechawska-Wójcik, M. (2020). Perfomance analysis of frameworks Angular and Vue.js. Journal of Computer Sciences Institute, 14, 59–64. https://doi.org/10.35784/jcsi.1577

[2] Bielak, K., Borek, B., & Plechawska-Wójcik, M. (2022). Web application performance analysis using Angular, React and Vue.js frameworks. Journal of Computer Sciences Institute, 23, 77–83. https://doi.org/10.35784/jcsi.2827

[3] Boczkowski, K., & Pańczyk, B. (2020). Comparison of the performance of tools for creating a SPA application interface - React and Vue.js. Journal of Computer Sciences Institute, 14, 73–77. https://doi.org/10.35784/jcsi.1579

[4] Delcev, S., & Draskovic, D. (2018). Modern JavaScript frameworks: A survey study. In 2018 Zooming Innovation in Consumer Technologies Conference (ZINC) (pp. 106–109). IEEE. https://doi.org/10.1109/ZINC.2018.8448444

[5] Diniz-Junior, R. N. V., Figueiredo, C. C. L., Russo, G. de S., Bahiense-Junior, M. R. G., Arbex, M. V. L., dos Santos, L. M., da Rocha, R. F., Bezerra, R. R., & Giuntini, F. T. (2022). Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue. In 2022 XLVIII Latin American Computer Conference (CLEI) (pp. 1–9). IEEE. https://doi.org/10.1109/CLEI56649.2022.9959901

[6] Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern Web architecture. ACM Transactions on Internet Technology, 2(2), 115–150. https://doi.org/10.1145/514183.514185

[7] Ivanova, S., & Georgiev, G. (2019). Using modern web frameworks when developing an education application: A practical approach. In 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 1485–1491). IEEE. https://doi.org/10.23919/MIPRO.2019.8756914

[8] Jartarghar, H. A., Rao Salanke, G., A. R., A. K., G. S., S., & Dalali, S. (2022). React apps with server-side rendering: Next.js. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 14(4), 25–29. https://doi.org/10.54554/jtec.2022.14.04.005

[9] Kalinowska, J., & Pańczyk, B. (2019). Comparison of tools for creating SPA applications using the examples of Angular2 and React. Journal of Computer Sciences Institute, 10, 1–4. https://doi.org/10.35784/jcsi.183

[10] Kowalczyk, K., & Plechawska-Wójcik, M. (2016). AngularJS and ReactJS libraries - performance analysis. Journal of Computer Sciences Institute, 2, 114–119. https://doi.org/10.35784/jcsi.126

[11] Lipski, P., Kyć, J., & Pańczyk, B. (2021). Comparative analysis of the Angular 10 and Vue 3.0 frameworks. Journal of Computer Sciences Institute, 20, 205–209. https://doi.org/10.35784/jcsi.2688

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

[12] Lyxell, O. (2023). Server-side rendering in React: When does it become beneficial to your web program? (Master's thesis, Umeå University).
[13] Meredova, A. (2023). Comparison of server-side rendering capabilities of React and Vue (Bachelor's thesis, Haaga-Helia University of Applied Sciences).
[14] Nowacki, R., & Plechawska-Wójcik, M. (2016). Comparative analysis of tools dedicated to building single page applications – AngularJs, ReactJS, Ember.js. Journal of Computer Sciences Institute, 2, 98–103. https://doi.org/10.35784/jcsi.122
[15] Novac, O. C., Madar, D. E., Novac, C. M., Bujdosó, G., Oproescu, M., & Gal, T. (2021). Comparative study of some applications made in the Angular and Vue.js frameworks. In 2021 16th International Conference on Engineering of Modern Electric Systems (EMES) (pp. 1–4). IEEE. https://doi.org/10.1109/EMES52337.2021.9484150
[16] Ollila, R., Mäkitalo, N., & Mikkonen, T. (2022). Modern web frameworks: A comparison of rendering performance. Journal of Web Engineering, 21(3), 789–813. https://doi.org/10.13052/jwe1540-9589.21311
[17] Patel, V. (2023). Analyzing the impact of Next.JS on site performance and SEO. International Journal of Computer Applications Technology and Research, 12(10), 24–27. https://doi.org/10.7753/IJCATR1210.1004
[18] Poniedziałek, A., & Pańczyk, B. (2023). Performance analysis of React v. 18.1.0 and Angular v. 11.0.2 development frameworks. Journal of Computer Sciences Institute, 29, 341–345. https://doi.org/10.35784/jcsi.3782
[19] Skrzypiec, S., & Plechawska-Wójcik, M. (2023). Comparative analysis of Angular and React development frameworks. Journal of Computer Sciences Institute, 28, 256–263. https://doi.org/10.35784/jcsi.3724
[20] Wang, X. S., Balasubramanian, A., Krishnamurthy, A., & Wetherall, D. (2013). Demystifying page load performance with WProf. In 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13) (pp. 473–485). USENIX Association.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.