Comparison of Web Page Rendering Methods Based on Next.js Framework Using Page Loading Time Test

Roy Hanafi[1], Abd Haq[2], Ninik Agustin[3*]

[1,2,3]Informatics Study Program, Faculty of Mathematic and Computer Science, Universitas Nahdlatul Ulama Al Ghazali, Cilacap, Central Java, Indonesia
Email: [1]hanafiroy032@gmail.com, [2]abdulhaq@unugha.id, [3*]ninik.agustin@unugha.id

(Received: 26 Jan 2024, revised: 1 Mar 2024, accepted: 7 Mar 2024)

Abstract

In the rapidly developing digital age, websites have become indispensable for interaction, information dissemination and transaction. To improve the performance of web applications, choosing the right rendering technology is critical. Next.js is a framework designed to overcome React's limitations in server-side rendering. This study investigates the effectiveness of Client-side Rendering (CSR), Server-side Rendering (SSR), and Static Site Generation (SSG) on the Next.js-based Filmku website using the loading time method. The study concentrates on page loading speed, complete page rendering speed, and user experience. The authentication page takes 422 ms to complete the CSR process, which is 57.41% slower than the SSG finish time of 180 ms and 34.88% slower than SSR, which completes the authentication page in 274 ms. On the Profile page, SSG completes the page rendering process much faster, taking only 524 ms, which is 25.79% faster than SSR's completion time of 706 ms and even 13.75% faster than CSR's completion time of 608 ms. The SSG rendering method completed in 1,135 ms on the main page, which is 15.93% faster than the CSR completion time of 1,350 ms and 25.57% faster than the SSR completion time of 1,525 ms. It is evident that SSG has a faster rendering speed compared to the other methods. However, it should be noted that CSR may result in slower initial page load times. SSR can provide stable rendering times, but it can also burden the server as every client request is fully processed on the server.

Keywords: Client-Side Rendering, Next.Js, Server-Side Generation, Server-Side Rendering.

I. INTRODUCTION

In the current digital landscape, websites play a critical role in global communication, information distribution, and business transactions. A website comprises static documents created using Hypertext Markup Language (HTML), which allows convenient sharing of information provided one is connected to the internet [1], [2]. Several factors are considered when determining the quality of a website, including access speed, easy-to-read content, and a consistent layout or design [3]. Website complexity poses challenges for developers in terms of website performance. The greater the amount of data or content displayed, the more significant the impact on the browser's rendering speed and page loading times [2]. A website designer should prioritize not only the design layout but also the enhancement of website performance and speed. The website's speed and responsiveness have a significant impact on user retention and conversion rates, and brand image [4].

To enhance web application performance, the selection of a rendering technique is of utmost importance [5]. Next.js is a novel framework aimed at surpassing the limitations of React in Server-side Rendering. It is based on the React Framework, which leverages the benefits of React while incorporating additional features [6]. Next.js is a lightweight JavaScript library for creating static and server-rendered applications. It offers a minimal and flexible configuration that can be tailored to the specific needs of the application [7]. Next.js is a full-stack framework, serving as both a front-end tool for building web interfaces and a back-end solution for rendering and managing databases [8]. Next.js utilizes folder directories to route web pages. This enables automatic routing for pages by using page directories [7]. Moreover, Next.js boasts a plethora of features, comprising Server-side rendering (SSR), automatic code splitting, Static site generation (SSG), Client-side routing (CSR), CSS and Sass support, plugins, and integration [6], [9].

By default, Next.js pre-renders each page and generates HTML for each page first, so that not everything is done by JavaScript on the client side [10]. Next.js offers SSR (Server-side Rendering) and SSG (Static Site Generation) pre-rendering methods, which optimize performance and SEO

[11]. Additionally, Next.js provides a CSR (Client-side Rendering) alternative. Basically, Next.js enables the utilization of multiple rendering methods on a single page, simplifying the selection of an efficient and suitable method. Determining the most suitable method (SSR, CSR, or SSG) depends on the application's purpose, page intricacy, and interactivity requirements. SSR is beneficial for SEO but has an impact on server performance [12], while CSR may slow down the initial load time but enables faster subsequent page loads without straining the server [13]. On the other hand, SSG is highly dependable in terms of speed, but may not be the optimal choice for databases requiring real-time updates [14]. Based on the attributes of these rendering methods, it is crucial to compare the efficiency of Client-side Rendering, Server-side Rendering, and Static Site Generation when used on a website built on the Next.js platform. The Page Loading Time test technique is used to compare the rendering methods. Page Loading Time refers to the duration taken to download and present all webpage content on the browser [15]. This research project aims to investigate the speed of loading webpages and completing page rendering (finish) [16]. A comparative analysis of rendering techniques in the Next.js framework will be conducted to provide an overview of their performance. This study will be useful for developers and researchers in selecting the appropriate rendering technique for web application development.

II. METHODOLOGY

The research device is a laptop running Windows 11 with 11th generation Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42GHz specifications, 8GB DDR4 RAM and 512GB SSD. Node.js, Visual Studio Code, and Developer Tools on Google Chrome, Mozilla Firefox, and Opera browsers are the software tools used. The research subject is the “Filmku” web system source code in the form of a TypeScript file based on the Next.js framework. The “Filmku” web system has a web structure as shown in the Figure 1. One of page of this web system (My List page) is shown in the Figure 2.

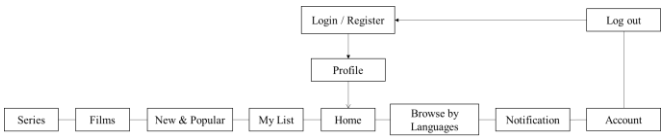


Figure 1. Web Structure of the "Filmku" Website

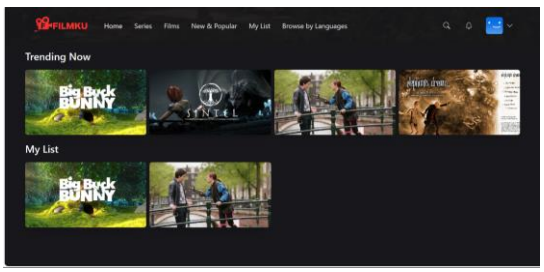


Figure 2. My List Page on "Filmku" Web System

This study examines the performance comparison of page load speeds, with a particular emphasis on the front end of websites. The research procedure is shown in Figure 3 below.

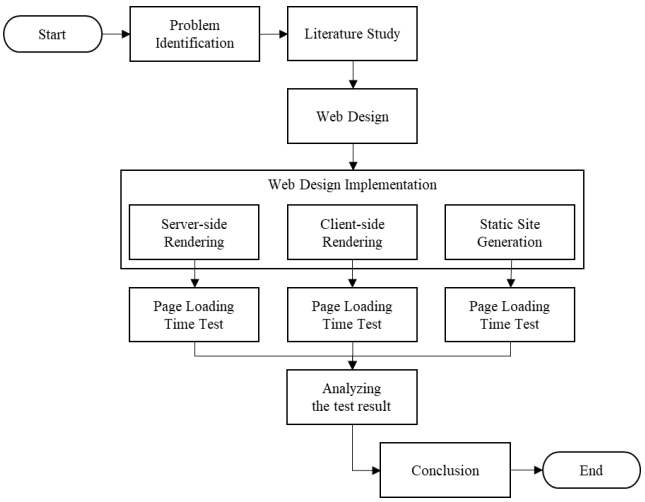


Figure 3. Research Procedure

A. Server-side Rendering

Server-side rendering (SSR) is a technique for rendering web pages on the server and sending the fully rendered HTML to the client [17]. This technique offers various advantages over client-side rendering (CSR), for example, faster initial page loading times, improved SEO [11], and better accessibility for users with slow internet connections. Figure 4 illustrates the SSR process, which starts with an HTTP request to the server that is generated when a user enters a URL in a browser or clicks a link to a website. When a request is made, the server retrieves the necessary data from either the database or third-party API, activating SSR. This demonstrates server-side pre-rendering whereby the server compiles the JavaScript into a static HTML file [18]. During build time, Next.js generates HTML pages and serves pre-rendered pages from the server to the browser, utilizing minimal JavaScript code.

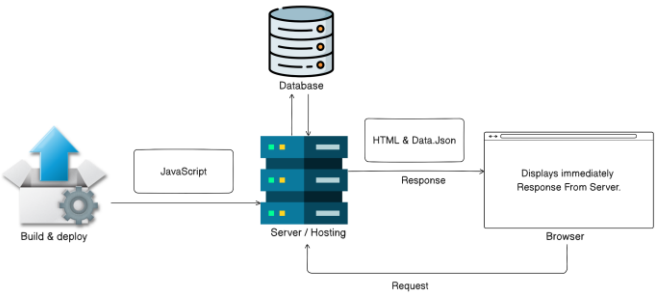


Figure 4. SSR Stages from Browser Perspective (Modified From Lazuardy and Anggraini [10])

The use of SSR on a page is done by exporting the getServerSideProps function. The SSR implementation code comes from the Index.jsx file script. This function will retrieve existing data on the server. The data retrieved is adjusted to the request from the user, this is done to display in real-time. When

the data on the server changes, the data seen by the user will automatically change.

B. Client-side Rendering

Client-side rendering (CSR) refers to the direct rendering of all processes in the browser using JS. This process includes logic, fetching, routing, and templating [18]. In Figure 5, Client refers to the device used, such as a smartphone or computer, which sends requests to the server and presents an interface for interaction [8]. React.js, Angular, and Vue.js utilize client-side rendering that runs in the browser [19]. CSR involves the utilization of JavaScript to showcase content on the web browser. Subsequently, during the initial load, only the essential HTML document, which contains JavaScript files, is received [11]. This approach replaces the need to retrieve all content directly from the HTML document.

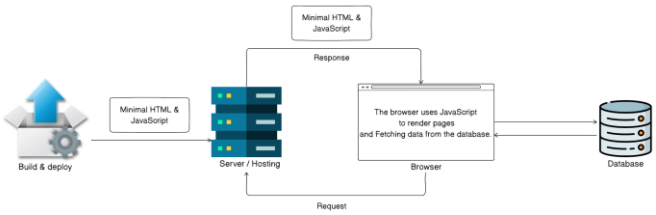


Figure 5. CSR Stages from Browser Perspective (modified from Lazuardy and Anggraini [10])

The application of CSR in Next.js involves utilizing React Hooks within the page like useEffect(). The other approach involves employing data retrieval libraries such as SWR or TanStack. The implementation code for CSR, which uses the useEffect() function, originates from the Index.jsx file script. This function serves to present and recover data in CSR. The data retrieval occurs in the browser, and to display the latest data, a trigger is essential.

C. Static Site Generation

Static Site Generation (SSG) is a technique that converts web pages embedded with client-side JavaScript into static files of HTML, CSS, and JavaScript. Essentially, SSG comprises diverse software tools that can generate static pages [14]. Data retrieval takes place during the build. In Figure 6, Static HTML and JSON files are created and deployed to the server. JSON is a format for sharing data. JSON, originating from the JavaScript programming language is now widely supported in other languages such as Python, Ruby, PHP, and Java [20]. When a client initiates a request, a static page is generated instantly without requiring any retrieval or creation of data either on the server or client side. Nevertheless, it is crucial to bear in mind that if there is a modification in the backend data, it will not be reflected on the page as there is no mechanism to trigger the generation and retrieval of data. During the build process, the static site generator generates HTML files and a data.json file, which is then deployed to the server. The user makes a request, the server sends the HTML and data.json files to the client, and the browser displays the data sent by the server. It's important to note that any changes made to the database will not be reflected on the web page or server until the build and deploy process is repeated.

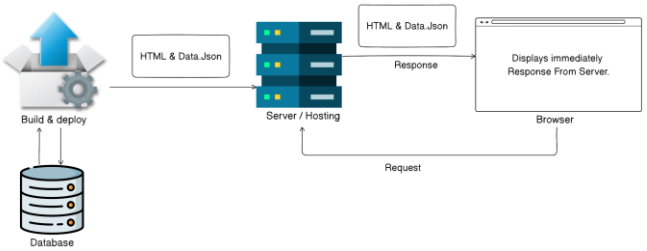


Figure 6. SSG Stages from Browser Perspective

The getStaticProps or getStaticPaths function retrieves data in SSG [10]. The SSG implementation code using the getStaticProps function is in the Index.jsx script. The function that displays and retrieves data in SSG is getStaticProps. This function retrieves pre-existing data during build and deployment. If the requested data does not exist during build and deployment, then the data will not be found.

D. Page Loading Time Test

This research uses the page loading time test method. The selection of this test method refers to user experience quality. Therefore, the stability factor is essential to speed up the loading process of a website. The performance in starting a website is one of the factors that will make users comfortable in using the service [21].

This test was conducted on three pages and three different browsers. The pages tested are the Authentication, Profile, and Home pages. The browsers used are Opera version 119.0, Chrome version 118.0.5993.90, and Mozilla Firefox version 119.0. The page loading time test was performed using 2.5 Mbps [16] internet speed and 300 ms latency on each page.

Page loading time (PLT) testing of web pages utilizes the network features menu in the DevTools of each browser. The process involves testing each web page and comparing the loading time between SSR, CSR, and SSG. Table 1 displays the measurement metrics used in this research.

Table 1. Measurement Variables [15]

No	Measurement variables	Description
1	Request	The number of requests on the rendered page
2	Resource	The amount of data on the rendered page
3	DOM Content Loaded	DOM (Document Object Model) Ready or DOM Content Loaded is an event that will be triggered when it has successfully read all DOM elements, namely from the beginning to the end of the web page.
4	Load	The time it takes for the page to Load the page only

No	Measurement variables	Description
5	Finish	The time it takes for the page to complete all rendering, starting from the page, content, video, and various other information.

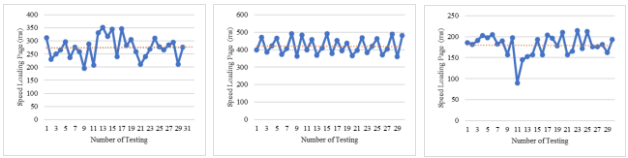
The measurement variables refer to the response and data of all transactions performed. Its objective is to simplify the identification of traffic or bottlenecks that impact the tested system's performance. Additionally, thus variables are utilized to gauge the website's effectiveness and speed in interacting with users. The variables display comprehensive data to users.

III. RESULT AND DISCUSSION

After developing and implementing each page rendering method, the "Filmku" website underwent testing. The speed of page load time was tested on three pages: Auth, Profile, and Main. Each page was tested using SSR, CSR, or SSG on the same device and with the same tools. Each page was tested using SSR, CSR, or SSG on the same device and with the same tools. Each page was tested using SSR, CSR, or SSG on the same device and with the same tools. The videos utilized in the "Filmku" web system are in MP4 format and range in size from 160 to 200 Mb.

A. Authentication Page Comparison

The Auth page is designed for user login and registration on the Filmku website. The graph below illustrates the three rendering methods used on this page.



(a) (b) (c)

Figure 7. Test Results for (a) "Finish CSR", (b) "Finish SSR", (c) "Finish SSG" on the Authentication Page

Figures 8 to 10 show the results of testing the five variables for each rendering method.

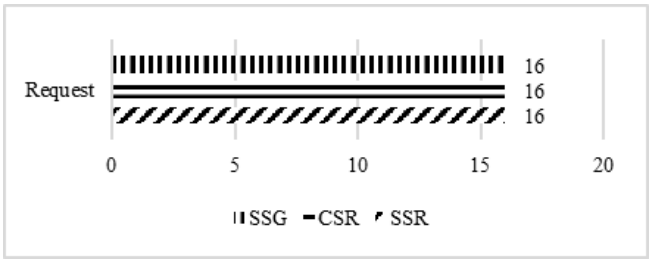


Figure 8. Request Comparison on Authentication Page Testing

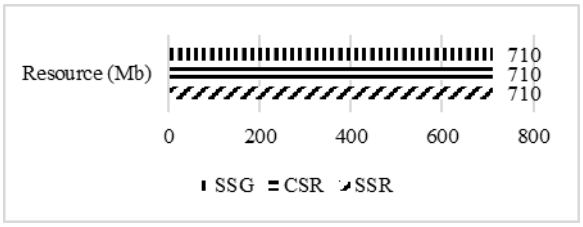


Figure 9. Resource Comparison on Authentication Page Testing

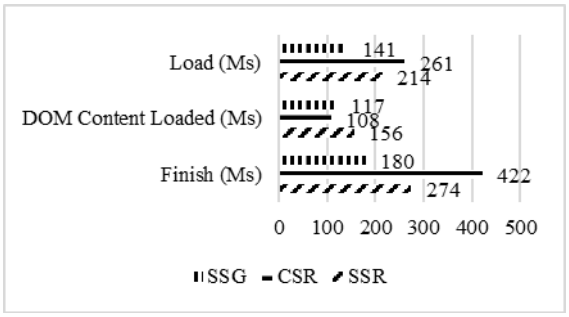
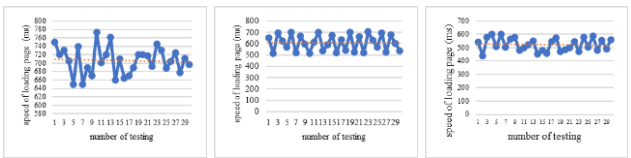


Figure 10. Authentication Page Rendering Performance

The test results above show differences in the performance of each rendering method. When considering the number of requests and resources, SSR, CSR, and SSG exhibit similar figures. However, when it comes to page load time, CSR outperforms the others with its DOM Content Loaded time. On the other hand, SSG has a shorter time than SSR and CSR in terms of Load and Finish. Furthermore, during the initial rendering of a web page, Client-Side Rendering (CSR) is slower compared to Server-Side Rendering (SSR) and Static Site Generation (SSG). This is because CSR retrieves and renders all data from the website server as a whole on the client side. According to the data, the finish time of CSR is 422 ms, which is 57.41% slower than the finish time of SSG (180 ms) and 34.88% slower than SSR with a finish time of 274 ms. However, SSG is 34.60% faster than SSR.

B. Profile Page Comparison

The profile page is a page that shows the user's picture and name for confirmation. The following is a graphical comparison of the SSR, CSR, and SSG methods on the profile page.



(a) (b) (c)

Figure 11. Test results for (a) "Finish CSR", (b) "Finish SSR", (c) "Finish SSG" on the Profile page

Figures 12 to 14 show the results of testing the five variables for each rendering method on the Profile page.

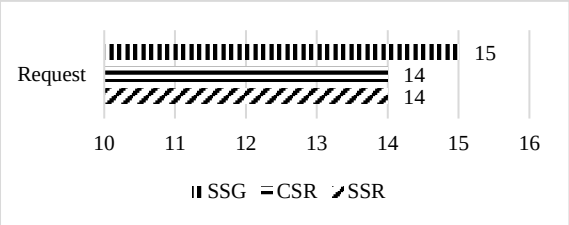


Figure 12. Request Comparison on Profile Page Testing

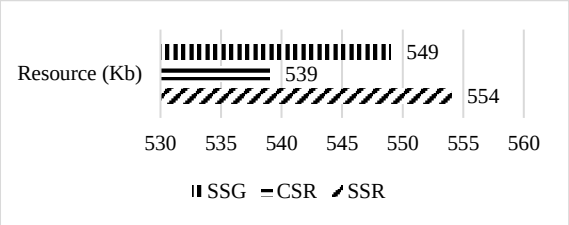


Figure 13. Resource Comparison on Profile Page Testing

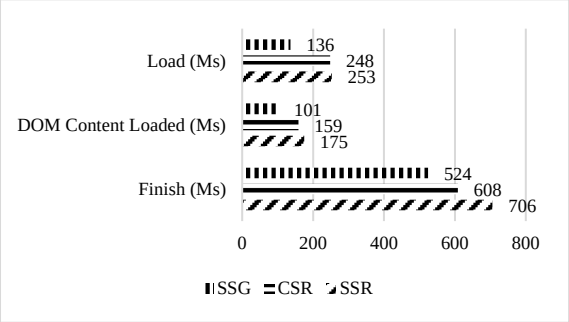
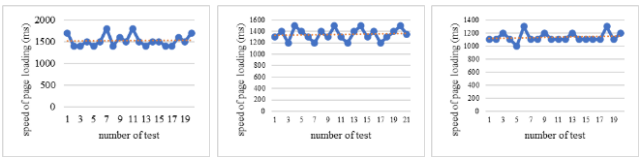


Figure 14. Profile Page Rendering Performance

The Profile SSG page completes the page rendering process in a very short time of only 524 ms, which is 13.75% faster than the CSR finish time (608 ms) and even 25.79% faster than the SSR finish time (706 ms), based on the values of the five variables from the three figures above. In terms of resources, SSR has the largest amount. CSR begins to render the second page once all website data has been retrieved and prepared during the rendering of the first page. On the other hand, SSR performs well with efficient resource size, despite having a slightly slower finish time than CSR.

C. Home Page Comparison

This Home Page is the page that has the most content, and components ranging from navbars, dashboards, trending films now, and films in the User Playlist. The figure below shows a graph testing SSR, CSR, and SSG methods.



(a) (b) (c)

Figure 15. Test results for (a) "Finish CSR", (b) "Finish SSR", (c) "Finish SSG" on the Home page

Figures 16 to 18 show the results of testing the five variables for each rendering method on the Profile page.

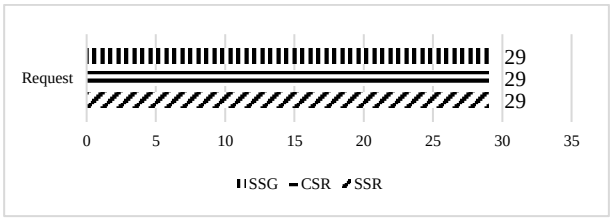


Figure 16. Request Comparison on Home Page Testing

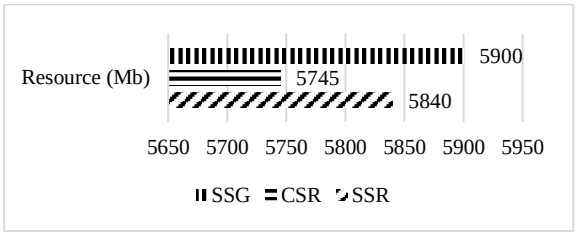


Figure 17. Resource Comparison on Home Page Testing

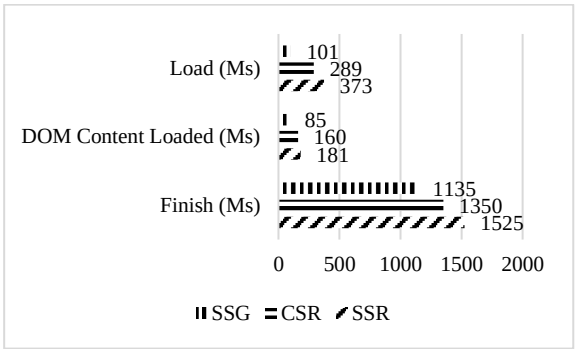


Figure 18. Home Page Rendering Performance

Figures 16 to 18 show that the Client-Side Rendering (CSR) test results differ significantly from those of Server-Side Rendering (SSR) and Static Site Generation (SSG) in terms of the number of requests, source size, and page rendering time. It is important to note that these results are based on objective data and not subjective evaluations. Although CSR's finish time is 15.93% slower than SSG's finish time (1,135 ms), it is 11.48% faster than SSR's finish time (1,525 ms). Although Server-Side Rendering (SSR) performed well in some aspects, it showed a slight increase in finish time compared to CSR and SSG. SSR has lower performance in terms of larger resource sizes and a much slower finish time, making it the least performant among the three.

D. Discussion

Among the tests conducted on the Authentication, Profile, and Home pages, SSG proved to be the fastest in completing rendering at 1,839 ms and has a relatively large resource (7,159 kb). This indicates that SSG is more reliable in terms of rendering speed. On the other hand, CSR has a longer rendering time on the first page load, but it can be loaded quickly and has stable resources. This is demonstrated by the time it takes CSR to render all pages in 2,379 ms, as well as

having the smallest resource (6,994 kb). Of the three rendering methods compared, SSR is the slowest, taking 2,506 ms to complete. In terms of resources, SSR uses 7,104 kb, which is smaller than SSG but larger than CSR.

Page Loading Time (PLT) testing on the three pages of the "Filmku" web system shows that there are advantages in each rendering technique measured based on the measurement variables. PLT testing on the Authentication page shows that SSG is superior to CSR and SSR from all parameters. In PLT testing on the Profile page, SSG is lighter than Request and Resource parameters, but CSR is faster than SSR and SSG on DOM Content Loaded, Load, and Finish parameters. Based on PLT testing on the Homepage, SSR is the lightest of the Request and Resource parameters, but CSR is the fastest on the DOM Content Loaded, Load, and Finish parameters.

The initial loading of the home page using the CSR technique is comparatively slower than other methods due to the retrieval of all the minimal HTML and JavaScript files from the server for client-side rendering [22]. Then, after all the files are rendered, the next page will be faster to display. CSR is the best method if used on the web with real-time data usage and does not require SEO (Search Engine Optimization) [13]. This is different from what happens with SSR. SSR has a stable rendering time but will be burdensome from the server side because every request from the client will be processed on the server completely. However, by rendering it on the server, the database displayed is always real-time. Despite various rendering page loading tests, it is crucial to consider website usability when using this rendering method. When real-time data and good SEO are required, SSR is the best method.

The SSG rendering method is faster than other rendering methods, this is because SSG does not require real-time data retrieval, the data in SSG has been created JSON files during build and deployment, and this data will not change before rebuilding and deploying. So, SSG is the best method used for website rendering if it does not require real-time data but requires SEO [14].

IV. CONCLUSION

Based on the conducted tests, it can be concluded that the SSG rendering method is faster than the other methods, with a completion time of 1839 ms. This is because SSG does not require real-time data retrieval, as data is created in JSON files during build and deploy, and will not change until build and deploy are performed again. The initial page load time for CSR is slower than other methods because it requires rendering of all HTML and JavaScript files on the client side, at least from the server. However, subsequent page loads are faster. CSR received a total turnaround time of 2379 ms. In contrast, SSR has a longer rendering time of 2506 ms. Among the compared rendering methods, server-side rendering (SSR) received the lowest score due to its slower performance caused by the need for each request to return to the server for processing. However, this method guarantees real-time data, which can put a strain on the server as each client request is processed entirely on the server.

In addition to considering rendering speed, it is also important to consider the usability of the website. Optimization of performance and functionality is necessary. If real-time data and good SEO are required, SSR is the optimal method. For real-time database needs without SEO issues, CSR is more suitable. However, if real-time data is not required or existing data does not change frequently but SEO is a priority, then SSG is the recommended method.

REFERENCE

[1] A. Suprapto and D. Sasongko, "Evaluasi Performa Website Berdasarkan Pengujian Beban Dan Stress Menggunakan Loadimpact (Studi Kasus Website Iain Salatiga)," *Netw. Eng. Res. Oper.*, vol. 6, no. 1, p. 31, 2021, doi: 10.21107/nero.v6i1.198.

[2] M. F. Santoso, "Teknik Single Page Application (SPA) Layout Web Dengan menggunakan React Js Dan Bootstrap," *J. Khatulistiwa Inform.*, vol. 9, no. 2, pp. 107–114, 2021, doi: 10.31294/jki.v9i2.11357.

[3] Suliman, "Analisis Performa Website Universitas Teuku Umar Dan Universitas Samudera Menggunakan Pingdom Tools Dan Gtmetrix," *J. Sist. Inf. dan Sist. Komput.*, vol. 5, no. 1, pp. 24–32, 2020, doi: 10.51717/simkom.v5i1.47.

[4] A. Yusuf F, I. Nuryasin, and Z. Sari, "Optimasi Kecepatan Loading Time Web Template Dengan Implementasi Teknik Front-End," *J. Repos.*, vol. 2, no. 11, p. 1456, 2020, doi: 10.22219/repositor.v2i11.746.

[5] R. Ollila, N. Mäkitalo, and T. Mikkonen, "Modern Web Frameworks: A Comparison of Rendering Performance," *J. Web Eng.*, vol. 21, no. 3, pp. 789–813, 2022, doi: https://doi.org/10.13052/jwe1540-9589.21311.

[6] D. Bui, "Next.Js for Front-End and Compatible Backend Solutions," South-Eastern Finland, University of Applied Science, 2023.

[7] J. Johansson, "Create React App vs NextJS," 2021.

[8] A. Hadjin, *The Ultimate Next.js Ebook*. JS Mastery, 2023.

[9] M. Riva, *Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js, the React framework for production*. Packt Publishing, 2022.

[10] M. F. S. Lazuardy and D. Anggraini, "Modern Front End Web Architectures with React.Js and Next.Js," *Int. Res. J. Adv. Eng. Sci.*, vol. 7, no. 1, pp. 132–141, 2022.

[11] H. A. Jartarghar, G. R. Salanke, A. K. A.R, S. G.S, and S. Dalali, "React Apps with Server-Side Rendering: Next.js," *J. Telecommun. Electron. Comput. Eng.*, vol. 14, no. 4, pp. 25–29, 2022.

[12] V. Patel, "Analyzing the Impact of Next.JS on Site Performance and SEO," *Int. J. Comput. Appl. Technol. Res.*, no. November, 2023, doi: 10.7753/ijcatr1210.1004.

[13] T. Fadhilah Iskandar, M. Lubis, T. Fabrianti Kusumasari, and A. Ridho Lubis, "Comparison between client-side and server-side rendering in the web development," *IOP*

Conf. Ser. Mater. Sci. Eng., vol. 801, no. 1, 2020, doi: 10.1088/1757-899X/801/1/012136.

[14] A. A. Yusuf, "Analisis Static Site Generator Pada Web portal Berita," *Ind. High. Educ.*, vol. 3, no. 1, pp. 1–68, 2021.

[15] E. Budiman, N. Puspitasari, S. N. Alam, T. M. A. Akbar, Haeruddin, and D. Indra, "Performance analysis of the resource loading time for borneo biodiversity information system," in *Proceedings of the 3rd International Conference on Informatics and Computing, ICIC 2018*, 2018, pp. 1–5, doi: 10.1109/IAC.2018.8780515.

[16] R. Oktrifianto, D. Adhipta, and W. Najib, "Page Load Time Speed Increase on Disease Outbreak Investigation Information System Website," *IJITEE (International J. Inf. Technol. Electr. Eng.*, vol. 2, no. 4, p. 114, 2019, doi: 10.22146/ijitee.46599.

[17] A. Meredova, "Comparison of Server-Side Rendering Capabilities of React and Vue," Haaga-Helia University of Applied Sciences, 2023.

[18] O. Lyxell, "Server-Side Rendering in React : When Does It Become Beneficial to Your Web Program ?," 2023.

[19] R. N. V. Diniz-Junior *et al.*, "Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue," in *2022 XVLIII Latin American Computer Conference (CLEI)*, 2022, pp. 1–9, doi: https://doi.org/10.1109/CLEI56649.2022.9959901.

[20] H. Kurniawan and Eka Puji Widianto, "Analisis Peningkatan Performa Akses website Dengan Web Server Stress Tools," *Jatisi*, vol. 2, no. 2, pp. 108–119, 2019.

[21] I. M. E. Listartha, "Pengujian Performa dan Tingkat Stress pada Website Legalisir Ijasah Online Universitas Pendidikan Ganesha," *Electro Luceat*, vol. 6, no. 1, pp. 66–73, 2020, doi: 10.32531/jelekn.v6i1.182.

[22] M. C. Weigle, M. L. Nelson, S. Alam, and M. Graham, "Right HTML, Wrong JSON: Challenges in Replaying Archived Webpages Built with Client-Side Rendering," 2023, doi: 10.1109/JCDL57899.2023.00022.