# Client-Side vs Server-Side Rendering: Impacts on Performance and User Experience in Web Applications

Joey Clapton
Department of Software Engineering, Pontifical Catholic University of Minas Gerais (PUC Minas)
Belo Horizonte, Minas Gerais, Brazil
joeyclapton42@gmail.com

Hugo Bastos
Department of Software Engineering, Pontifical Catholic University of Minas Gerais (PUC Minas)
Belo Horizonte, Minas Gerais, Brazil
hugo@pucminas.br

João Pedro Oliveira Batisteli
Department of Software Engineering, Pontifical Catholic University of Minas Gerais (PUC Minas)
Belo Horizonte, Minas Gerais, Brazil
joao.batisteli@sga.pucminas.br

Danilo Boechat Seufitelli
Federal Center for Technological Education of Minas Gerais (CEFET-MG)
Belo Horizonte, Minas Gerais, Brazil
danilobs@cefetmg.br

Cleiton Tavares
Department of Software Engineering, Pontifical Catholic University of Minas Gerais (PUC Minas)
Belo Horizonte, Minas Gerais, Brazil
cleitontavares@pucminas.br

## ABSTRACT

The increasing complexity of web applications necessitates careful selection of rendering strategies to ensure optimal performance and user experience. This study presents a comparative analysis of Client-Side Rendering (CSR) and Server-Side Rendering (SSR) techniques, evaluating their impact on web application quality based on Google's Core Web Vitals metrics (CWV). In this work, photo gallery web applications were implemented using both rendering approaches to allow a homogeneous environment for comparable results. Through controlled experiments across five data sizes (ranging from 498 kB to 26 MB), measurements were taken for key performance indicators, including First Contentful Paint (FCP), Speed Index (SI), Time to Interactive (TTI), Largest Contentful Paint (LCP), and JavaScript Heap Used. Statistical analysis using the Kruskal-Wallis test revealed significant performance differences ($p \leq 0.05$) in all metrics. Results show that SSR achieves superior performance in TTI (mean reduction of 62.3%) and LCP (mean reduction of 58.7%), particularly under increased data loads, while CSR exhibits better FCP and SI performance for smaller datasets. The results suggest that the choice between CSR and SSR should consider the application's specific objectives and usage environment. These findings contribute to evidence-based decision-making frameworks for web application architecture, supporting software quality engineering practices in modern development environments.

## KEYWORDS

Client-Side Rendering, Server-Side Rendering, Performance Analysis, User Experience, Web Applications

## 1 Introduction

With the expansion of digital platforms, there is an increase in the volume of data generated and processed as a result of the digitization of daily activities and the growing need to collect and process large datasets [15]. In this context, the JavaScript Object Notation (JSON) format has emerged as a popular choice for representing and exchanging data in web applications due to its flexibility and ease of use. However, with its popularity, significant challenges have arisen, such as data parsing [7], performance issues [1], and concerns about the quality and efficiency of the application [10]. The increase in data volume implies higher computational costs. For this reason, it is crucial that developers understand the impact of their technological choices and coding practices impact computational resource usage and the energy consumption of applications [3].

Previous studies compare data interaction formats in applications, addressing performance, computational costs, and environmental concerns. For instance, Gil and Trezentos [1] compare processing and energy consumption costs in mobile applications using XML, JSON, and Protocol Buffers. Tusa and Clayman [12] show that the choice of data encoding technology has a significant impact on performance and resource efficiency. Additionally, Nurseitov et al. [11] compare the performance of JSON and XML in a client-server environment, measuring transmission speed and computing resource usage. However, other factors such as the different processing approaches of data formats, both on the client and server sides, and their impact on energy consumption, performance, and user experience in data exchange for web applications must be considered.

Processing large volumes of JSON data has become a bottleneck in terms of performance and cost [7], which highlights the importance of understanding and analyzing the cost of data interaction. Efficiently addressing this issue not only seeks to reduce the costs of handling data [7], but also contributes to alleviating environmental concerns by reducing energy consumption and Carbon Dioxide ($CO_2$) emissions. According to Guamán et al. [2], in the year 2020, the Information and Communication Technology (ICT) sector was responsible for 1.43 billion tons of $CO_2$, representing 2.5% of global greenhouse gas emissions.

In this scenario, the goal of this paper is to compare the efficiency and performance between Client-Side Rendering (CSR) and Server-Side Rendering (SSR) in web applications. To achieve this goal, the following specific objectives were proposed: (i) analyze performance aspects during data transactions; (ii) identify and compare the computational costs involved in data exchanges; and (iii) investigate and measure application performance and interactivity based on Google's Core Web Vitals (CWV) metrics.

To accomplish these objectives, a systematic methodology was adopted. A back-end application, implemented using Node.js, was made responsible for handling two parameters: data size and rendering type. A front-end application, developed in JavaScript, was designed to request the data, render the content, and display the results. Performance evaluation was automated through a custom script. This script leveraged the Lighthouse library for comprehensive auditing, `chrome-launcher` for initiating clean Chrome browser instances, and Puppeteer for collecting profiling and computational cost data.

The results demonstrate statistically significant differences between CSR and SSR across various metrics. It was observed that SSR rendering performed better in metrics such as Time to Interactive (TTI) and Largest Contentful Paint (LCP), while the CSR approach excelled in First Contentful Paint (FCP), and Speed Index (SI). Based on this data, it is expected that software engineers will be able to determine the most advantageous scenarios for each approach, taking into account response time, memory consumption, and CWV indicators. Thus, this study provides a guide to assist in choosing the most appropriate rendering method, contributing to the development of more efficient and user-friendly web applications that optimize performance and minimize computational costs.

This study is structured as follows: Section 2 presents the theoretical background related to data interaction in web applications. Section 3 discusses related work. Section 4 details the materials and methods used in this research. Sections 5 and 6 present and discuss the results. Threats to validity are addressed in Section 7. Finally, Section 8 presents the conclusions.

## 2 Background

This section presents the key concepts related to the study. Section 2.1 describes the SSR method. Section 2.2 discusses CSR method. Section 2.3 addresses software performance aspects, and Section 2.4 covers quality indicators in web applications based on Core Web Vitals (CWV) metrics.

### 2.1 Server-Side Rendering (SSR)

Server-Side Rendering (SSR) is a web application development methodology in which pages are rendered on the server and sent to the browser [4]. When a request is made, the server processes it and returns the page already rendered in HTML to the user's browser.

In applications that use SSR, the rendering process is handled on the server, enabling the page to be fully loaded when accessed by the user Iskandar et al. [4]. This approach improves the user experience, particularly on low-performance devices, as the server performs the more computationally intensive tasks. Moreover, SSR provides advantages in search engine optimization (SEO), as it facilitates the indexing of page content by search engines.

### 2.2 Client-Side Rendering (CSR)

Client-Side Rendering is a web page rendering mechanism in which the rendering process takes place in the user's browser [4]. The server sends a basic HTML structure with JavaScript responsible for dynamically rendering the content on the client side.

CSR enables the development of more dynamic and interactive applications, as both the rendering logic and Document Object Model (DOM) manipulation are executed directly within the user's browser [4]. This approach can deliver a smoother and more responsive user experience, particularly in highly interactive scenarios such as Single Page Applications (SPAs), where navigation between views occurs without the need for full page reloads.

### 2.3 Software Performance Aspects

A software performance aspect refers to a non-functional characteristic that describes the operational efficiency of a software system under various execution conditions [9]. These aspects are essential to ensure that the software meets the expected quality and efficiency demanded by users. Key performance metrics include response time, which measures how long the system takes to respond to a request, and resource usage, which evaluates the memory and CPU consumption during application operations.

Performance aspects are critical to user experience, user retention, and the overall success of a web application. Pages that load slowly or respond poorly can frustrate users and cause them to abandon the site. Good performance is also important for Search Engine Optimization (SEO), as search engines tend to favor fast-loading websites [4]. As a result, optimizing these performance aspects has become a crucial part of developing and maintaining websites, requiring an integrated approach that considers both operational efficiency and end-user experience [10].

### 2.4 Quality Indicators in Web Applications Based on Core Web Vitals Metrics

CWV is a set of performance metrics introduced by Google[1] to help optimize and evaluate the user experience of web pages, guiding improvements in the user experience. The CWV includes:

**Largest Contentful Paint (LCP)**: measures the loading time of the largest content element visible to the user.

**First Input Delay (FID)**: measures the time from the user's first interaction to the browser's response.

**Cumulative Layout Shift (CLS)**: evaluates layout shifts during page loading.

**First Contentful Paint (FCP)**: measures how quickly content is rendered to the screen, ensuring users see something useful sooner.

**Speed Index (SI)**: measures how quickly visible content is displayed during page loading, influencing user experience and search engine ranking.

**Time to Interactive (TTI)**: evaluates page usability. A faster TTI means users can interact with the page sooner.

These metrics are essential not only for ensuring a good user experience but also have significant implications for search engine rankings, as Google has started to consider user experience as a ranking factor for search results [10].

## 3 Related Work

The related works presented in this section discuss studies connected to the context of quality indicators in web applications

[1]https://developers.google.com/web/tools/lighthouse

based on Core Web Vitals (CWV) metrics, evaluation and comparison of performance aspects, and identification and comparison of computational costs in web applications.

Langdale and Lemire [6] presents *simdjson* tool, a high- performance solution for processing large volumes of data in JSON. The study emphasizes that, with the growing exchange of structured data in web applications, the performance of traditional parsers has become a bottleneck. *Simdjson* outperforms reference analyzers such as RapidJSON by employing optimized techniques for detecting JSON structures. Thus, the study conceptually contributes to the present study by highlighting how the interpretation and manipulation of JSON data can directly impact overall application performance, particularly in contexts where different rendering strategies are compared.

van Riet et al. [13] analyzed an industrial case study in an agricultural technology company aiming to improve the performance of a large-scale web dashboard. The authors developed a performance engineering plan using a benchmarking tool based on Google Lighthouse to assess the impact on web performance metrics. The results demonstrated a significantly faster browsing experience, with pages loading 98.37% faster on desktop devices and 97.56% faster on mobile. Content rendering also became more efficient, with improvements of 48.25% and 19.85% in rendering speed on desktop and mobile devices, respectively. This study is relevant to the current research as it provides data on the impact of different interventions affecting performance metrics using Google Lighthouse. However, there remains a gap for analyzing performance during data transactions between JSON and HTML formats using the same tool.

Korobeinikova et al. [5] explored mechanisms to ensure the resilience and performance of web applications under high-demand scenarios and potential failures. The study investigated different scaling approaches for applications and databases, such as horizontal and vertical scaling, database replication, and sharding, focusing on achieving fault tolerance and autoscaling capabilities. The authors emphasizes the importance of making applications stateless by using external solutions for session storage and user files to facilitate horizontal scaling. The study concludes that to build a fault-tolerant web application, scalability is essential, and autoscaling can significantly reduce manual configuration time and improve responsiveness to traffic spikes. The research offers insights into how different scaling strategies can influence the performance and resilience of web applications, critical aspects also considered in the comparison between CSR and SSR.

Iskandar et al. [4] investigated the performance of web applications built using CSR and SSR rendering techniques. The study developed a web application named Show Up using both approaches and compared their performance based on FCP, Speed Index, TTI, and FMP metrics. The results showed that SSR rendering outperformed CSR across all evaluated metrics, delivering a faster and smoother user experience. In addition, SSR scored higher on Google audit metrics for performance, accessibility, and best practices. The authors provide a discussion on different rendering strategies and their implications for application performance. This enables further investigation into other contributing factors, such as the impact of JSON and HTML data formats on user experience and web application efficiency, analyzed in the present study.

## 4 Design Study

This study conducts a quantitative experimental research by investigating and comparing the performance, computational cost, and CWV metrics analysis between SSR and CSR for data presentation in web applications. The following subsections describe the development environment, which consists of a web application. Subsequently, the materials and methods used in the experiment are detailed, including performance measurement tools, computational cost metrics, and the different data sizes analyzed.

### 4.1 Study Steps

The study was divided into three steps, which are described as follows.

In the *first step*, the data were defined and created in JSON format. The datasets were generated in five different sizes: 498 kB, 2 MB, 7.9 MB, 16.8 MB, and 26 MB. Each database was composed by combining 13 different images, resulting in: 370 images (498 kB), 1,500 images (2 MB), 6,000 images (7.9 MB), 12,000 images (16.8 MB), and 19,000 images (26 MB). These sizes were selected based on the study by Nurseitov et al. [11], which analyzed the performance of data interchange formats in various scenarios. For the experiment, a data structure was designed to simulate a photo gallery system using JSON (Figure 1). The choice of this data structure aims to represent a real-world use case where user information is frequently exchanged between systems.

```json
{
  "user_id": 101,
  "username": "user6",
  "avatar_url": "https://example.com/profiles/user6.jpg",
  "posts": [
    {
      "post_id": 2,
      "image_url": "https://example.com/photos/photo2.jpg",
      "caption": "Sunset vibes",
      "timestamp": "2023-10-02T18:30:00Z",
      "likes": [
        {
          "user_id": 107,
          "username": "user7"
        },
        {
          "user_id": 108,
          "username": "user8"
        }
      ]
    }
  ]
}
```

**Figure 1: JSON data structure used in the application**

Furthermore, a back-end application was developed using the NodeJS platform. An Application Programming Interface (API) was created to handle two parameters: data size and rendering type (SSR or CSR). Based on these parameters, the API returns either the HTML rendered from the JSON or the raw JSON itself to the client. The data is stored locally.

The *second step* consisted of developing a front-end application in JavaScript. As illustrated in Figure 2, this application enables users to select both the size and format of the data to be requested. Once the request is sent to the back-end, the application displays the response, which can either be the fully rendered HTML returned by the server or the raw JSON used to dynamically build the web page.

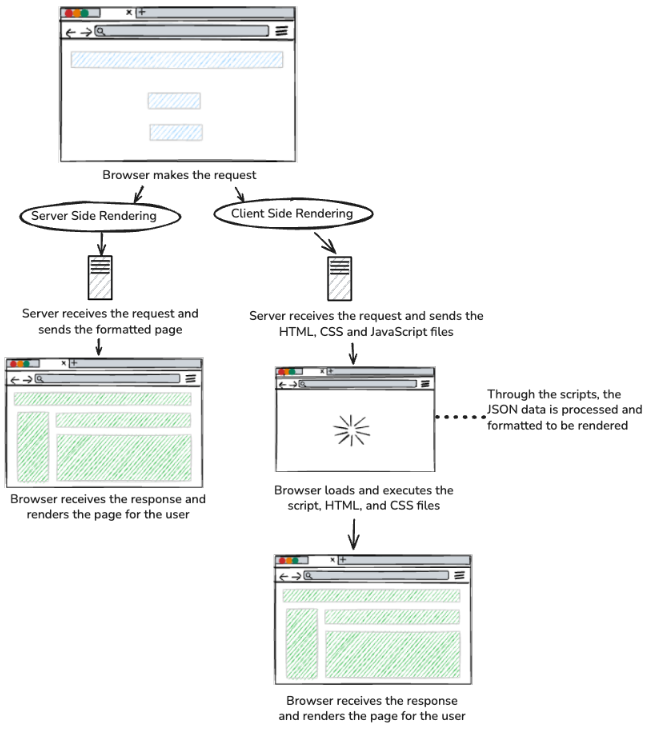

Browser makes the request
Server Side Rendering
Client Side Rendering
Server receives the request and sends the formatted page
Server receives the request and sends the HTML, CSS and JavaScript files
Through the scripts, the JSON data is processed and formatted to be rendered
Browser receives the response and renders the page for the user
Browser loads and executes the script, HTML, and CSS files
Browser receives the response and renders the page for the user

**Figure 2: Application operation flow**

Subsequently, a JavaScript script was developed to evaluate application performance. Leveraging the Lighthouse library, the script automatically computes key performance metrics. To ensure consistent testing conditions, the `chrome-launcher` library was used to start a clean instance of the Chrome browser. In addition, the Puppeteer tool[2] was used to collect profiling data and computational cost metrics. Each configuration (JSON and HTML) was tested 10 times, with the average execution time calculated to provide more accurate and representative results. The following metrics were analyzed in this study:

(1) **First Contentful Paint**
(2) **Speed Index**
(3) **Time to Interactive**
(4) **Largest Contentful Paint**
(5) ***JS Heap* Size**

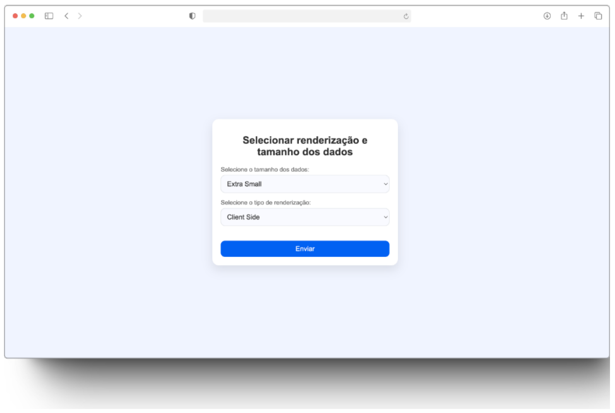
In the *third step*, the Kruskal-Wallis statistical test[3] was applied to validate the significance of the results across the different data formats tested. Core Web Vitals metrics were evaluated to understand the impact of data formats on user experience.

## 4.2 Development Environment

All experiments were conducted in a controlled environment to ensure consistency and reproducibility of results. The system's development and execution were performed on a MacBook Air M2, featuring an 8-core *3.5 GHz* CPU, running macOS Sequoia 15.0, equipped with 16GB of RAM and a 256GB SSD.

## 4.3 Photo Application

Two photo gallery applications were developed: one focused on server-side rendering, and the other on client-side rendering, along with a selection page where the user can choose which rendering method will be displayed. Both applications were built using the same HTML structure, the same CSS styles, and without inserting any scripts, in order to prevent side effects on test results. Figure 3 shows the initial selection page, while Figure 4 shows the application running with client-side rendering using the smallest data size.



Selecionar renderização e tamanho dos dados

Selecione o tamanho dos dados: Extra Small

Selecione o tipo de renderização: Client Side

**Figure 3: Data size and rendering type selection**

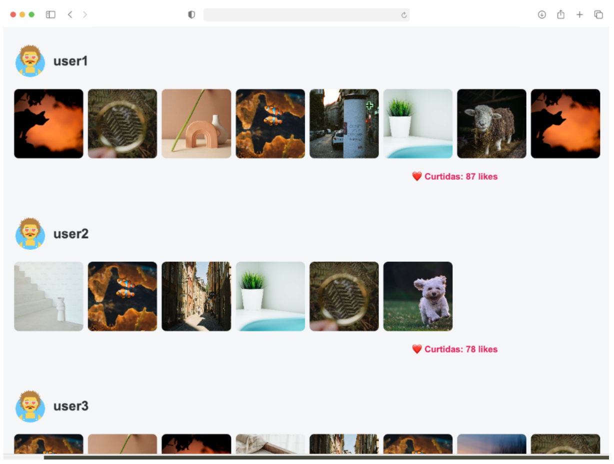


**Figure 4: Application using extra small data and CSR**

## 5 Results

After conducting the experiment, the average, median, and standard deviation values were calculated for each metric and rendering

[2]https://pptr.dev/
[3]Kruskal-Wallis is a non-parametric test used for comparing three or more independent samples.

Table 1: Analyzed Metrics

| Metric | Measures | Client Small | Server Small | Client XS | Server XS | Client M | Server M | Client L | Server L | Client XL | Server XL |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **LCP (ms)** | Average | 14,716 | 4,520 | 17,382 | 5,270 | 28,841 | 10,456 | 49,067 | 20,245 | 68,644 | 31,369 |
| | Median | 14,716 | 3,837 | 17,779 | 2,180 | 28,839 | 10,456 | 49,051 | 20,240 | 69,325 | 30,342 |
| | Std. Dev. | 1 | 2,048 | 920 | 6,180 | 5 | 1 | 31 | 8 | 2,185 | 3,078 |
| **FCP (ms)** | Average | 608 | 2,980 | 614 | 1,241 | 608 | 9,668 | 608 | 19,424 | 608 | 29,548 |
| | Median | 608 | 2,979 | 609 | 1,240 | 608 | 9,668 | 608 | 19,433 | 608 | 29,542 |
| | Std. Dev. | 0.6 | 3.6 | 8.8 | 3.4 | 1.3 | 3.2 | 0.9 | 31 | 0.6 | 19 |
| **SI (ms)** | Average | 608 | 2,980 | 614 | 1,241 | 792 | 9,668 | 1,250 | 19,424 | 1,796 | 29,548 |
| | Median | 608 | 2,979 | 609 | 1,240 | 785 | 9,668 | 1,249 | 19,433 | 1,793 | 29,542 |
| | Std. Dev. | 0.6 | 3.6 | 8.8 | 3.4 | 23 | 3.2 | 27 | 31 | 27 | 19 |
| **TTI (ms)** | Average | 15,646 | 4,593 | 17,794 | 5,370 | 29,816 | 10,509 | 50,035 | 20,493 | 69,452 | 32,220 |
| | Median | 15,646 | 3,890 | 18,154 | 2,232 | 29,814 | 10,509 | 50,019 | 20,481 | 70,199 | 31,120 |
| | Std. Dev. | 1 | 2,108 | 832 | 6,276 | 5 | 1 | 32 | 26 | 2,404 | 2,867 |
| ***JS Heap* Used (KB)** | Average | 202,997 | 205,631 | 120,867 | 108,542 | 440,358 | 436,163 | 525,017 | 491,820 | 613,318 | 560,978 |
| | Median | 205,728 | 207,186 | 121,014 | 104,708 | 446,453 | 443,045 | 528,830 | 485,718 | 612,958 | 552,913 |
| | Std. Dev. | 12,000 | 10,105 | 3,696 | 10,222 | 26,620 | 15,233 | 31,953 | 27,080 | 32,861 | 33,344 |

* Client = Client Side, Server = Server Side, XS = Extra Small, M = Medium, L = Large, XL = Extra Large, Std. Dev. = Standard Deviation.

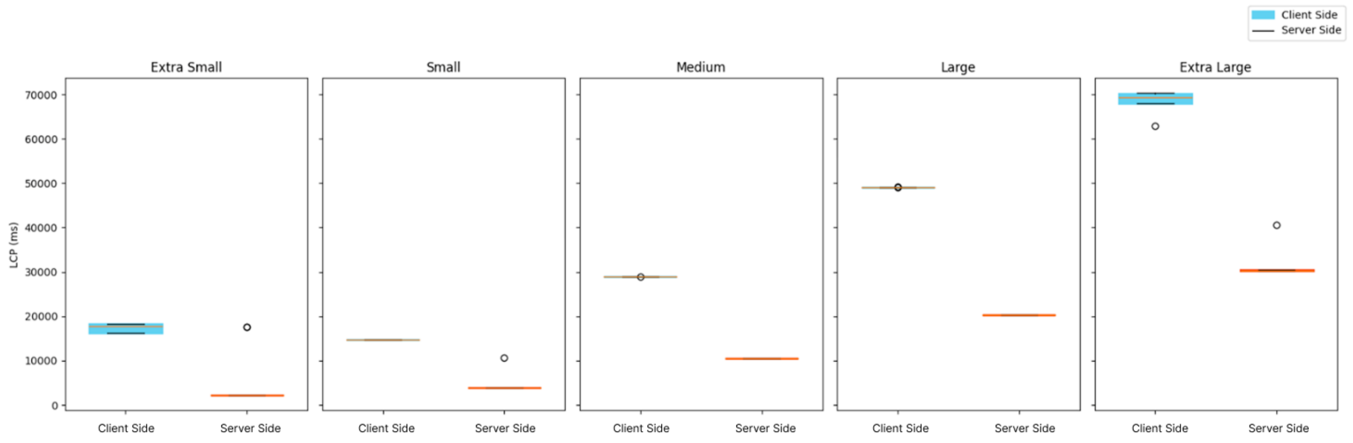


Figure 5: LCP values

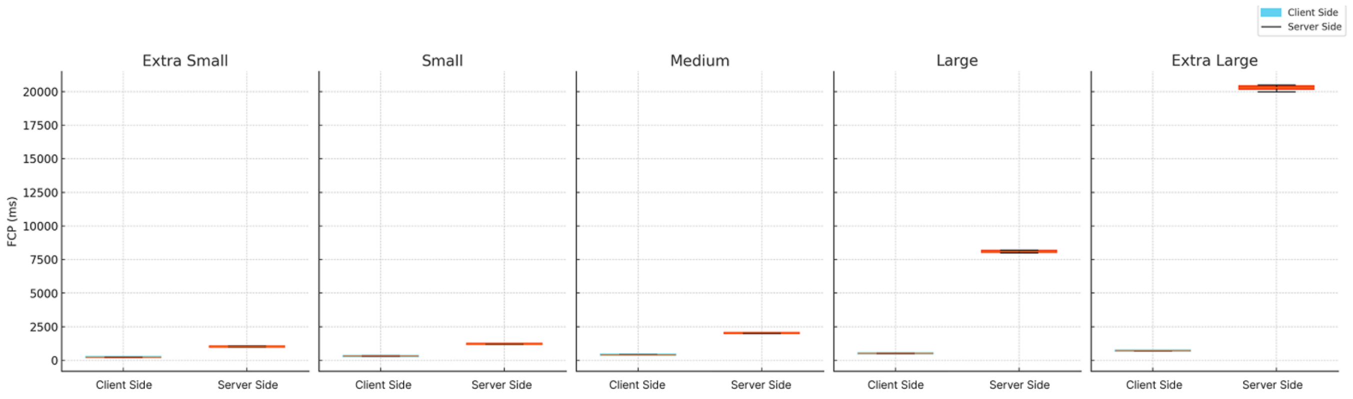


Figure 6: FCP values

method. It was observed that, for smaller data sizes, the metrics showed lower and more consistent values, especially on the server side. On the other hand, as the data size increased, the standard deviation also increased in some measurements, indicating greater variability in the results. Table 1 presents the computed results.

***Largest Contentful Paint.*** The LCP metric measures the loading time of the largest element on the page, which may be an image, text section, or video. A value up to 2.5s indicates a good user experience; between 2.5s and 4.0s indicates room for improvement; and above 4.0s indicates a poor experience. Figure 5 shows the boxplot with the results. SSR rendering achieved better performance across all data sizes compared to CSR, which performed worse, especially with larger datasets.

***First Contentful Paint.*** The FCP metric refers to the time between when the user navigates to the page and when the first piece of content is rendered on the screen. A good experience occurs when this value is up to 1.8s, while values above 3s are considered poor. Figure 6 presents the boxplot with the results. For smaller datasets, both server-side and client-side rendering provide fast user experiences. As data volume increases, SSR shows a significant increase in loading time, while CSR maintains consistently good performance compared to SSR.

***Speed Index.*** The SI metric measures the speed at which content is displayed during page loading. Lighthouse classifies times up to 3.4s as fast, between 3.4s and 5.8s as moderate, and above 5.8s as slow. Figure 7 shows the boxplot with the results. For smaller data sizes, SI followed the same trend as the LCP and FCP metrics, with both rendering methods showing comparable loading times. However, with larger data volumes, there was a significant performance degradation in server-side rendering, resulting in worse outcomes.

***Time to Interactive.*** The TTI metric measures how long it takes for the page to become fully interactive. Lighthouse considers times up to 3.4s as fast, between 3.4s and 5.8s as moderate, and above 5.8s as slow. Figure 8 shows the boxplot with the results. SSR showed shorter loading times compared to CSR across all data sizes.

***JS Heap Size.*** The *JS Heap Size* metric refers to the memory space allocated for storing JavaScript objects. Although there is no standardized quality threshold, it was observed that CSR and SSR had similar memory usage for smaller data sizes. However, memory consumption increased significantly for CSR with larger datasets. Figure 9 shows the boxplot with the results.

## 5.1 Statistical Analysis

All results discussed in the previous sections were considered for the statistical analysis. The Shapiro-Wilk test was performed to assess data normality. Next, non-parametric variance analyses (Kruskal-Wallis) were conducted to compare the results. A $p$-value below 0.05 was considered statistically significant, indicating a 95% confidence level.

Table 2 presents the results of the statistical tests for each metric, highlighting the impact of the rendering method factor on the analyzed metrics. These results suggest statistically significant differences ($p$-value $\leq 0.05$) across all variables based on the rendering method factor.

**Table 2: Client Side vs Server Side (Kruskal-Wallis) by data size**

| **Size** | **Metric** | **$p$-value** |
|---|---|---|
| **Extra Small** | LCP | 1.50e-03 |
| | FCP | 1.57e-04 |
| | SI | 1.57e-04 |
| | TTI | 1.50e-03 |
| | *JS Heap* | 2.33e-02 |
| | Heap Used | 2.33e-02 |
| **Small** | LCP | 1.57e-04 |
| | FCP | 1.57e-04 |
| | SI | 1.57e-04 |
| | TTI | 1.57e-04 |
| | *JS Heap* | 9.40e-01 |
| | Heap Used | 9.40e-01 |
| **Medium** | LCP | 1.57e-04 |
| | FCP | 1.57e-04 |
| | SI | 1.57e-04 |
| | TTI | 1.57e-04 |
| | *JS Heap* | 7.62e-01 |
| | Heap Used | 7.62e-01 |
| **Large** | LCP | 1.57e-04 |
| | FCP | 1.57e-04 |
| | SI | 1.57e-04 |
| | TTI | 1.57e-04 |
| | *JS Heap* | 3.43e-02 |
| | Heap Used | 3.43e-02 |
| **Extra Large** | LCP | 1.57e-04 |
| | FCP | 1.57e-04 |
| | SI | 1.57e-04 |
| | TTI | 1.57e-04 |
| | *JS Heap* | 6.50e-03 |
| | Heap Used | 6.50e-03 |

## 6 Discussion

While both rendering strategies demonstrated good performance with smaller datasets, significant differences emerged as data volume increased, directly impacting application efficiency. Specifically, SSR consistently yielded better results for LCP and TTI , likely due to the server delivering a fully rendered HTML page, allowing the browser to display core content and become interactive more rapidly without extensive client-side JavaScript processing. Conversely, CSR exhibited superior performance in FCP and SI, especially with larger datasets where its ability to dynamically render content once initial scripts are loaded minimized the time to render visible elements. This distinction suggests that for content-heavy applications where initial load and SEO are paramount, SSR may be more advantageous, whereas highly interactive applications might benefit from CSR's dynamic rendering capabilities after initial asset loading. Regarding *JS Heap* memory usage, there were no statistically significant differences, as shown in Table 2.

This study supports the findings of Meredova [8], which analyzed server-side rendering (SSR) capabilities using the React and Vue front-end libraries. According to the study, the SSR approach enhances the user experience by reducing the initial page load time. This advantage makes SSR suitable for devices with slower internet connections and for applications where fast content loading is essential.

## 7 Threats to Validity

This section addresses the potential threats to validity across different dimensions and the strategies adopted to mitigate them [14].

**Internal Validity.** For internal validity, factors such as hardware and software influence on the stability of the results were

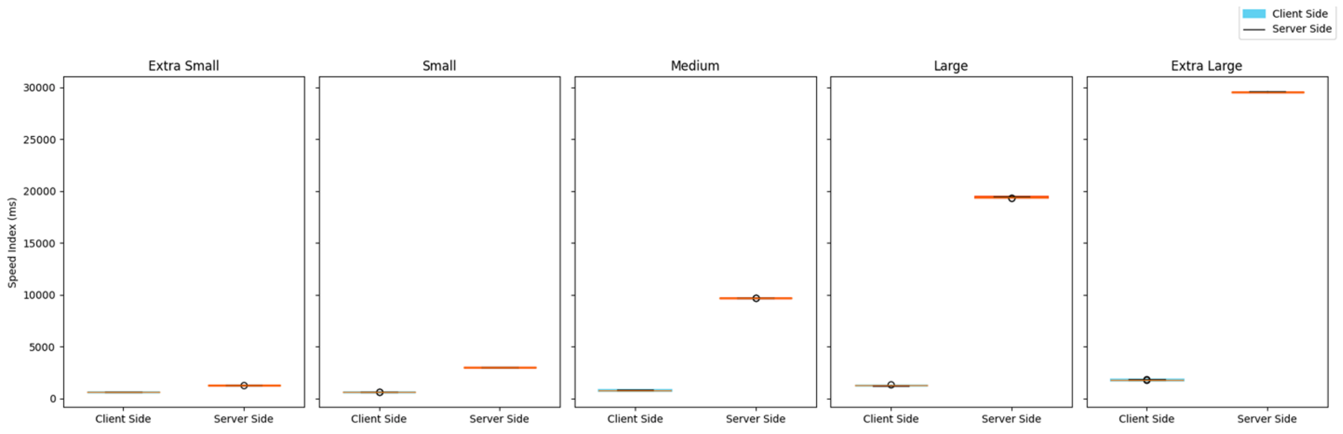


**Figure 7: SI values**

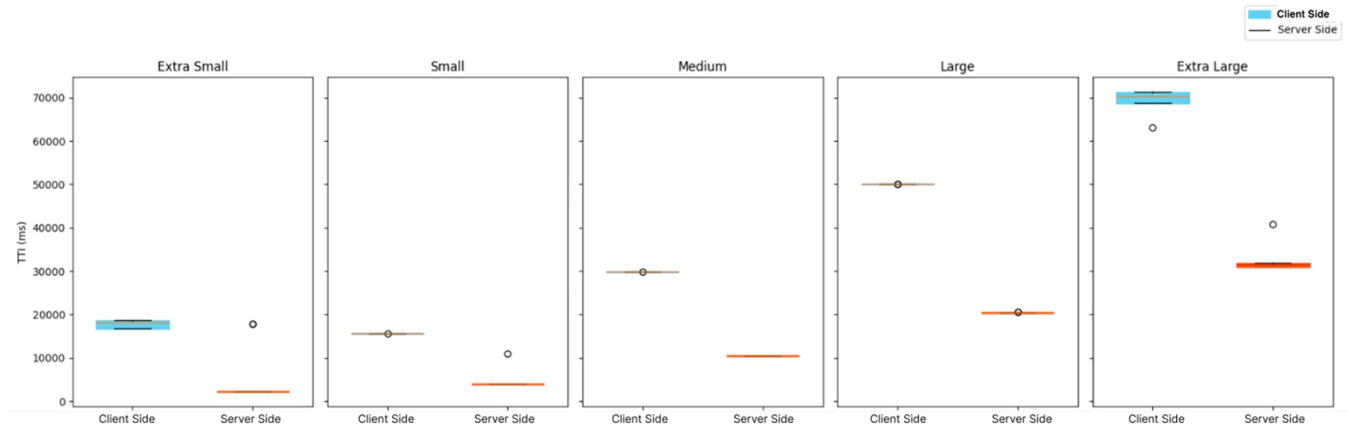


**Figure 8: TTI values**

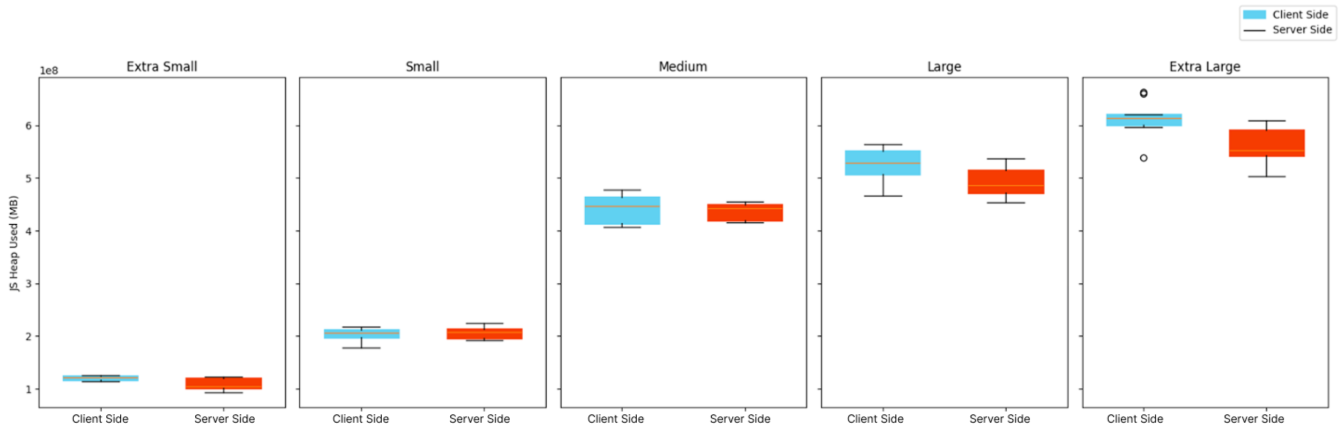


**Figure 9: *JS Heap* memory usage**

considered. To mitigate these interferences, all tests were executed in a local environment, on the same machine and operating system. Additionally, each experiment was run ten times, ensuring that the observed results are consistent and not merely due to random chance or transient system variations. This repetition enhances the reliability of the findings by providing more robust and representative data averages.

**External Validity.** Regarding external validity, a recognized limitation of this study is that the experiments were conducted within a single execution environment. All experiments were executed on a single hardware configuration (a MacBook Air M2) and within a single browser environment (Chromium). This restricted setup constrains the generalizability of the findings, as performance and behavior may vary across distinct hardware platforms, operating systems, and browsers employing different rendering engines. However, with the environment used, relevant insights were found for the study proposal, which can be expanded in future work.

**Conclusion Validity.** With respect to conclusion validity, several techniques exist to improve application performance, such as data minification and compression, asynchronous loading (lazy loading), the use of Content Delivery Networks (CDNs), image preloading, and data caching. However, the objective of this study was to analyze the impact of the rendering approach (SSR and CSR) and data size on the user experience. To ensure that the evaluation focus remained on rendering performance, the application used only HTML, CSS, and JavaScript without any external libraries.

## 8 Conclusion

This study compared the performance between two types of rendering in web applications, using Google's Core Web Vitals metrics as a quality assessment approach. For the analysis, both CSR and SSR rendering strategies were evaluated. A photo gallery application was developed for each rendering approach, and aspects such as performance, user experience, and memory usage were compared.

For data collection, a Node.js script was created using the Lighthouse performance measurement tool. The Kruskal-Wallis statistical test was applied to compare the results and verify the presence of statistically significant differences. The tests indicated relevant differences. One hypothesis raised is that CSR rendering provides better performance in metrics related to the loading of visible content, such as FCP and SI, while SSR rendering presents better results in metrics related to interactivity and loading of the largest page content, such as TTI and LCP. The results suggest that the choice between CSR and SSR should consider the specific objectives of the application and its usage environment.

As future work, we suggest incorporating performance optimization techniques to evaluate whether, in combination with these techniques, there are significant performance differences between CSR and SSR. Moreover, analyzing other rendering approaches in web applications, such as Incremental Static Regeneration (ISR) and isomorphic architecture, would be of interest. It is also recommended to perform tests on mobile devices and under varying bandwidth conditions, given the relevance of such access scenarios to ensure an optimized and consistent experience across different platforms and contexts. Additionally, future research should include a broader range of application encompassing more interactive and diverse scenarios such as e-commerce platforms, dashboards, and complex single-page applications, in order to strengthen the generalizability of the findings.

## ARTIFACT AVAILABILITY

The *scripts* and *dataset* are available at: https://github.com/CleitonSilvaT/cs-vs-ssr-sbqs-2025

## REFERENCES

[1] Bruno Gil and Paulo Trezentos. 2011. Impacts of data interchange formats on energy consumption and performance in smartphones. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication* (Lisboa, Portugal) *(OSDOC '11)*. Association for Computing Machinery, New York, NY, USA, 1–6. doi:10.1145/2016716.2016718

[2] D. Guamán, J. Pérez, and P. Valdiviezo-Diaz. 2023. Estimating the energy consumption of model-view-controller applications. *Journal of Supercomputing* 79 (2023), 13766–13793. doi:10.1007/s11227-023-05202-6

[3] Stefan Huber, Lukas Demetz, and Michael Felderer. 2022. A comparative study on the energy consumption of Progressive Web Apps. *Information Systems* 108 (2022), 102017. doi:10.1016/j.is.2022.102017

[4] Taufan Fadhilah Iskandar, Muharman Lubis, Tien Fabrianti Kusumasari, and Arif Ridho Lubis. 2020. Comparison between client-side and server-side rendering in the web development. In *IOP Conference Series: Materials Science and Engineering*, Vol. 801. IOP Publishing, 012136. doi:10.1088/1757-899X/801/1/012136

[5] Tetiana Korobeinikova, Volodymyr Maidaniuk, Olexandr Romanyuk, Roman Chekhmestruk, Oksana Romanyuk, and Sergey Romanyuk. 2022. Web-applications fault tolerance and autoscaling provided by the combined method of databases scaling. In *2022 12th International Conference on Advanced Computer Information Technologies (ACIT)*. IEEE, 27–32. doi:10.1109/ACIT54803.2022.9913098

[6] G. Langdale and D. Lemire. 2019. Parsing gigabytes of JSON per second. *The VLDB Journal* 28 (2019), 941–960. doi:10.1007/s00778-019-00578-5

[7] Yinan Li, Nikos R. Katsipoulakis, Badrish Chandramouli, Jonathan Goldstein, and Donald Kossmann. 2017. Mison: a fast JSON parser for data analytics. *Proc. VLDB Endow.* 10, 10 (jun 2017), 1118–1129. doi:10.14778/3115404.3115416

[8] Aylar Meredova. 2023. Comparison of Server-Side Rendering Capabilities of React and Vue. Bachelor's Thesis, Haaga-Helia University of Applied Sciences, Helsinki, Finland.

[9] Mahshid Helali et al. MOGHADAM. 2023. Machine Learning to Guide Performance Testing: An Autonomous Test Framework. *Research Institutes of Sweden (RISE) SICS, Sweden; Mälardalen University, Västerås, Sweden* (2023).

[10] Larisse NONJAH, Fabiana; KUPSKI. 2023. Page Experience: Primeiras Percepções Sobre o Novo Algoritmo do Google no SEO de Empresas com Atuação no Mercado Brasileiro. *Não especificado* (2023).

[11] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. 2009. Comparison of JSON and XML data interchange formats: a case study. *Caine* 9 (2009), 157–162.

[12] F. Tusa and S. Clayman. 2021. The Impact of Encoding and Transport for Massive Real-time IoT Data on Edge Resource Consumption. *Journal of Grid Computing* 19, 32 (2021). doi:10.1007/s10723-021-09577-9

[13] Jasper van Riet, Ivano Malavolta, and Taher A Ghaleb. 2023. Optimize along the way: An industrial case study on web performance. *Journal of Systems and Software* 198 (2023), 111593. doi:10.1016/j.jss.2022.111593

[14] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. 2012. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

[15] Feng Zhu, Lijie Xu, Gang Ma, Shuping Ji, Jie Wang, Gang Wang, Hongyi Zhang, Kun Wan, Mingming Wang, Xingchao Zhang, Yuming Wang, and Jingpin Li. 2022. An empirical study on quality issues of eBay's big data SQL analytics platform. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice* (Pittsburgh, Pennsylvania) *(ICSE-SEIP '22)*. Association for Computing Machinery, New York, NY, USA, 33–42. doi:10.1145/3510457.3513034