JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 28 (2023) 197–203
Received: 24 May 2023
Accepted: 21 June 2023

Performance optimization of web applications using Qwik

Optymalizacja wydajności aplikacji internetowych z wykorzystaniem Qwik

Adam Lipiński*, Beata Pańczyk

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

Abstract
This article analyzes the performance of three frameworks - React.js, Next.js and Qwik - that offer different methods of rendering application views. The purpose of the study was to show whether the new Qwik framework allows for better application load times compared to the other frameworks. The study was conducted using 3 applications representing the same research content, referring to cases occurring in production environments. In order to assess the performance, the Google Lighthouse tool was used, thanks to which it was proved that it is impossible to unequivocally say that Qwik allows for better optimization of the application compared to other frameworks.

Keywords: optimization; React; Next; Qwik

Streszczenie
W niniejszym artykule przeprowadzono analizę wydajnościową trzech szkieletów programistycznych - React.js, Next.js oraz Qwik - oferujących różne metody renderowania widoków aplikacji. Celem badania było wykazanie, czy nowy szkielet Qwik pozwala na uzyskanie lepszych wyników czasów ładowania aplikacji, w porównaniu z pozostałymi szkieletami. Badanie przeprowadzono z wykorzystaniem 3 aplikacji reprezentujących tę samą treść badawczą, nawiązującą do przypadków występujących w środowiskach produkcyjnych. W celu oceny wydajności wykorzystano narzędzie Google Lighthouse, dzięki czemu dowiedziono, że nie da się jednoznacznie stwierdzić, aby Qwik pozwalał na lepszą optymalizację aplikacji w porównaniu z pozostałymi szkieletami.

Słowa kluczowe optymalizacja; React; Next; Qwik

*Corresponding author
Email address: adam.lipinski@pollub.edu.pl (A. Lipiński)

©Published under Creative Common License (CC BY-SA v4.0)

1. Introduction

Along with the development and popularization of modern technologies used to create the client part of websites and web applications, programmers were encouraged to transfer some operations to browsers. Increasingly, MPA (Multi Page Applications) were abandoned in favor of SPA (Single Page Applications) which allowed for increased smoothness of operation, as well as avoiding constant reloading of the page with each user's action. Initially, the idea was to upload a nearly empty HTML file and fill it with content using code that is executed when the page loads. This method was named CSR (Client Side Rendering) because all views were rendered on the client's browser side.

The increased amount of often computationally demanding code contributed to the search for new solutions in the field of rendering application views, as well as optimization in terms of their loading speed. This brought many changes in this area which made the SSR (Server Side Rendering) method more and more often used in comparison to CSR. This method differed significantly from Client Side Rendering because the initial view of the application sent in response to the client's request was rendered on the server. As a result, the application loading speed was significantly improved and the user could see its content faster. However, this is not without some losses - the user does see the page faster, but there is no possibility to interact with it as long as the hydration process is not completed.

Hydration is the process by which application code is loaded in the background when a view is displayed. In this process, all listeners for any events are attached to the static page and the code required when the application is run for the first time is executed. As it has already been mentioned, applications very often began to grow very large so the time between the first page view and the moment when the user could start using it was getting longer and longer.

Therefore, many developers began to look for a way to reduce the impact of hydration on page loading. There have been many approaches to this problem but one of the last ones that started to get community acknowledgement was the use of the resumability method. It was proposed by Misko Hevery, the creator of one of the currently most popular programming frameworks used to create web applications - Angular. This method assumes that operations on the server are suspended and the process is resumed in the same place but on the client side, without the need to re-create and download the entire application logic. It has been implemented in the constantly developed new Qwik framework which, according to people working on it, is to change the quality of web applications and solve the problem that the Internet has to face due to the hydration process [1].

Due to the fact that this technology is very young, there are practically no scientific sources on it. To fill this gap this article will examine how the Qwik framework handles the day-to-day cases CSR and SSR applications face. The collected results will concern application rendering times and delays. They will be compared with the results for the widely used React.js and Next.js frameworks which will make it possible to determine whether Qwik can really become a new direction in the development of web applications in the future.

2. Purpose and scope of the paper

The aim of the paper is to examine and compare three programming frameworks: Qwik, React.js, and Next.js. Frameworks will be compared based on application rendering times and delays.

The scope of the paper includes: analysis of literature on research oscillating in the subject of optimization of web applications, selection of research methods, development of research scenarios, preparation of applications in each of the analyzed frameworks, research implementation, data analysis, and formulation of conclusions.

For the purposes of the work, the following research hypotheses have been set which will be tested thanks to the performed analysis:

H1. The Qwik framework allows to reduce the loading time of the web application compared to the React.js and Next.js frameworks.
H2. The Qwik framework allows for faster rendering of applications with a different number of elements compared to the React.js and Next.js frameworks.
H3. The Qwik framework allows for more efficient loading of a large number of larger-sized resources compared to the React.js and Next.js frameworks.
H4. The Qwik framework allows for more efficient execution of more demanding code compared to the React.js and Next.js frameworks.

3. Literature review

Although the technology analyzed in this article is completely new, comparing the optimality and speed of programming frameworks is a frequent topic of scientific papers and publications.

In the research conducted in articles [2, 3] the authors tried to determine which of the analyzed frameworks - React.js, Next.js, and Vue.js - is better for tasks such as rendering a different number of elements in the DOM (Document Object Model) tree. By comparing several of the same applications written in different programming frameworks, the authors concluded that in most cases React.js performed the best in both studies.

In addition, the frameworks compared in publications [4-7] offered different methods of rendering application views. On this basis, the researchers wanted to indicate which of the methods allows for the best results in terms of response time for a specific test case. In most scenarios, rendering views on the server offered better performance results, compared to rendering on the client side.

Particularly noteworthy are the articles [8, 9] in which the researchers, more than on the frameworks themselves, focused on various techniques for optimizing web applications. In these papers, one application was profiled in terms of performance and then subjected to speed optimization methods. Thanks to that it was possible to indicate that selecting the appropriate rendering method can significantly affect the speed of the application.

A frequent topic of consideration was also the development of tools and test frameworks allowing for easier and more accurate profiling of applications as well as identifying areas where improvements could be made to increase the speed of their operation which was analyzed by researchers in articles [10-17]. In the majority of instances, the prepared test environments were characterized by high accuracy of results, compared to other widely used testing software. Moreover, the sheer multitude of publications on this subject proves the general interest in the subject of performance optimization.

Yet another approach to the topic of application optimization was adopted by researchers in a publication [18] in which they tried to check how adequate the results proposed by automatic performance testing tools are to the real, empirical feelings of users from using web applications. Their task was also to indicate which metrics best reflect the experience of using the application. The obtained results showed that the FCP (First Contentful Paint) time, which is an integral metric accessible within the Google Lighthouse performance assessment, very clearly translated into the actual perception of the website's performance.

Among the newer sources, the subject of application performance is also very often mentioned which can be read in the article [19] prepared by researchers from Google and Shopify. In this article, they described the advantages and disadvantages of various application rendering methods and also explained how the choice of technology, and thus the rendering method, affects performance as well as SEO (Search Engine Optimization). An important part of the article is a fragment devoted to the problem of hydration occurring in modern programming frameworks prepared for developing client parts of web applications that offer the possibility of rendering views on the server.

The last of the analyzed articles focused entirely on considerations very close to the subject of this thesis. In the paper [20] it was explained what the resumability method utilized by the Qwik framework is and how it works. The author emphasized the problems and limitations caused by the hydration process and conducted a discussion on how the mentioned method can affect their solution. The challenges faced by the use of this framework, and thus this method in production applications, were also identified.

4. Research method

For this study, three identical applications with a minimalist graphical interface were prepared in each tested

programming framework. The research is aimed at checking application rendering times for each of these technologies in various test cases. The analyzed times are: FCP (First Contentful Paint), TBT (Total Blocking Time), LCP (Largest Contentful Paint), and SI (Speed Index). The results were obtained through the utilization of data gathered from the Google Lighthouse tool. In order to ensure consistency and accuracy, caching of data in the browser was disabled for each of the conducted tests. Furthermore, 50 trials were undertaken for every test to increase reliability and validity of the results.

The prepared test cases performed in this study are as follows:

1. Rendering applications with a large number of elements in the DOM tree:
   a) 100 elements,
   b) 1000 elements,
   c) 10000 elements.
2. Rendering applications with a large number of resources of large sizes - images with a size of several MB:
   a) 100 images,
   b) 1000 images,
   c) 10000 images.
3. Rendering applications that perform expensive operations and processes data - downloading a large amount of external data then processing it and displaying it on the screen.

4.1. Testing platform

For the purpose of conducting performance audits, the following testing platform was used:

- web browser: Google Chrome 109.0.5414.75,
- OS: MacOS Monterey 13.1,
- CPU: Intel Core i5 1.4GHz quad-core,
- RAM: 16GB 2133 MHz LPDDR3,
- GPU: Intel Iris Plus Graphics 645 1536 MB,
- model: Macbook Pro 13 2020.

4.2. Description of the experiment

To achieve the research objective an experiment was designed for which the research scenarios presented in Tables 1-3 were determined**.**

Table 1: A research scenario for examining the rendering times of an application with a large number of elements in the DOM tree

Name: Rendering an application with a large number of elements in the DOM tree		
Aim: Verifying how a large number of elements in the DOM affects the rendering speed of the application		
Initial conditions: The application is ready to load, the number of elements in the tree is set to the initial value - 100		
Final conditions: App load time data has been saved		
Next steps		
No.	Description of activities	Expected result
1.	Loading the application	The application has loaded
2.	Saving the value of the loading times	The data has been saved
3.	The app is loaded another 49 times	The application has loaded and the results have been saved
4.	Changing the number of rendered items to 1000	The number of items has been changed
5.	Repeating points 1-3 for a new number of elements	The application has loaded and the results have been saved
6.	Changing the number of rendered items to 10000	The number of items has been changed
7.	Repeating points 1-3 for a new number of elements	The application has loaded and the results have been saved

Table 2: Research scenario for examining the rendering times of an application with a large number of displayed photos

Name: Rendering an application which loads large numbers of images of significant size		
Aim: Verifying how loading and displaying a large number of photos of significant sizes affects application rendering times		
Initial conditions: The application is ready to load, the number of photos to download is set to the initial value - 10		
Final conditions: App load time data has been saved		
Next steps		
No.	Description of activities	Expected result
1.	Loading the application	The application has loaded
2.	Saving the value of the loading times	The data has been saved
3.	The app is loaded another 49 times	The application has loaded and the results have been saved
4.	Changing the number of photos displayed to 100	The number of photos has been changed
5.	Repeating points 1-3 for a new number of photos	The application has loaded and the results have been saved
6.	Changing the number of photos displayed to 1000	The number of photos has been changed
7.	Repeating points 1-3 for a new number of photos	The application has loaded and the results have been saved

Table 3: Research scenario for testing the rendering times of an application that downloads and processes a large amount of data

Name: Rendering an application which performs expensive operations and data processing - downloading a large amount of external statistical data, then processing, sorting and displaying it on the screen in the form of a graph
Aim: Verifying how downloading and processing large amounts of external data affects application rendering speed
Initial conditions: The application is ready to load
Final conditions: App load time data has been saved

Next steps		
No.	Description of activities	Expected result
1.	Loading the application	The application has loaded
2.	Saving the value of the loading times	The data has been saved
3.	The app is loaded another 49 times	The application has loaded and the results have been saved

5. Results

The results of the audits conducted for the three prepared tests are presented in Tables 4-11. Additionally, Figures 1-7 showcase graphs that display the collected results, categorized into tests and frameworks.

It is important to mention that the LCP times were not measurable using the Google Lighthouse tool for the third test, and hence, they were not considered in the analysis of the application's performance for this particular test. The absence of outcomes concerning this metric could potentially be attributed to the particularity of this test.

Table 4 Mean metric values for test one - 100 elements

Metric	React	Next	Qwik
FCP [s]	1.40±0.02	1.11±0.04	0.80±0.00
LCP [s]	1.40±0.02	1.83±0.06	0.80±0.00
TBT [ms]	0.00±0.00	0.01±2.40	0.00±0.00
SI [s]	1.40±0.02	1.11±0.04	0.80±0.00

Table 5: Mean metric values for test one - 1000 elements

Metric	React	Next	Qwik
FCP [s]	1.40±0.00	1.11±0.03	0.90±0.00
LCP [s]	1.40±0.00	1.86±0.05	0.90±0.00
TBT [ms]	11.20±7.18	9.40±2.40	0.00±0.00
SI [s]	1.40±0.00	1.11±0.03	0.90±0.00

Table 6: Mean metric values for test one - 10000 elements

Metric	React	Next	Qwik
FCP [s]	1.40±0.00	1.78±0.05	1.31±0.06
LCP [s]	1.40±0.00	1.96±0.24	1.45±0.06
TBT [ms]	177.80±7.90	158.20±38.42	309.80±64.29
SI [s]	1.40±0.00	1.82±0.07	1.31±0.06

Table 7 Mean metric values for test two - 10 photos

Metric	React	Next	Qwik
FCP [s]	1.40±0.00	1.10±0.02	0.80±0.00
LCP [s]	8.25±0.78	5.32±0.74	7.61±0.76
TBT [ms]	449.99±346.34	22.80±29.97	176.98±336.80
SI [s]	2.26±0.16	1.12±0.05	2.12±0.50

Table 8 Mean metric values for test two - 100 photos

Metric	React	Next	Qwik
FCP [s]	1.40±0.00	1.10±0.01	0.80±0.00
LCP [s]	7.97±0.56	5.56±0.34	7.89±0.67
TBT [ms]	384.73±313.05	23.60±34.74	62.08±213.68
SI [s]	2.37±0.18	1.12±0.09	2.14±0.27

Table 9: Mean metric values for test two - 1000 photos

Metric	React	Next	Qwik
FCP [s]	1.40±0.00	1.40±0.00	0.90±0.01
LCP [s]	10.39±0.56	5.07±1.49	27.11±4.09
TBT [ms]	2.24±0.44	19.80±34.26	1.95±0.46
SI [s]	5.27±0.72	1.46±0.15	5.42±0.85

Table 10: Mean metric values for test three

Metric	React	Next	Qwik
FCP [s]	1.40±0.00	1.10±0.00	1.42±0.07
LCP [s]	N/A	N/A	N/A
TBT [ms]	109.35±6.80	130.80±12.09	89.40±28.24
SI [s]	5.88±1.48	1.43±0.05	12.15±4.76

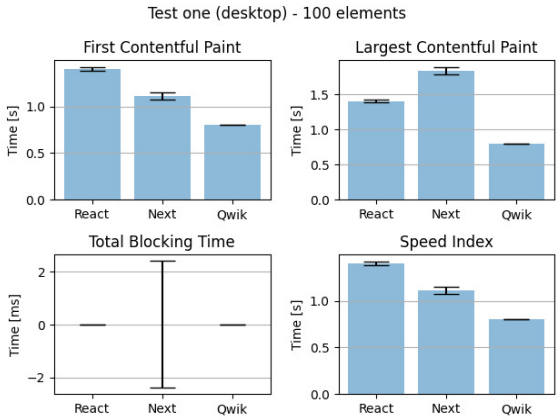

Figure 1: Mean metric values in graphical form obtained in test one for 100 elements.

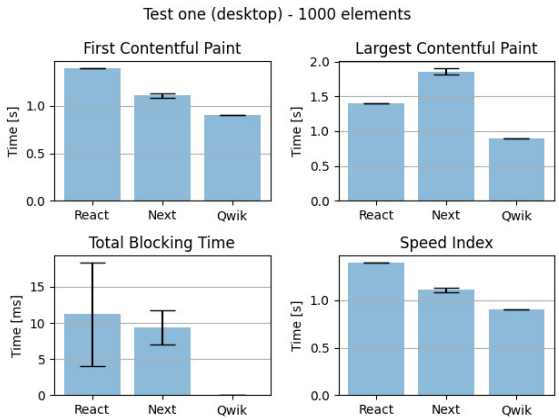

Figure 2: Mean metric values in graphical form obtained in test one for 1000 elements.

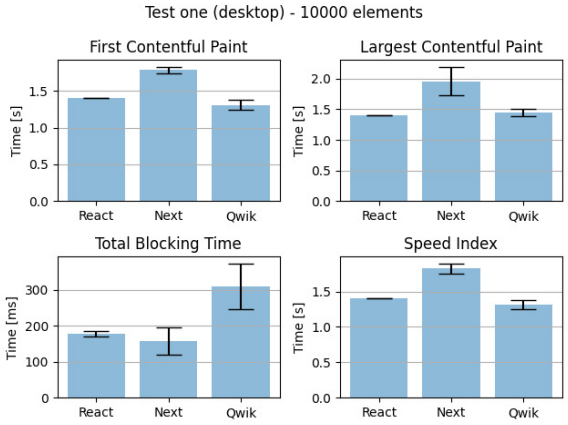

Figure 3: Mean metric values in graphical form obtained in test one for 10000 elements.

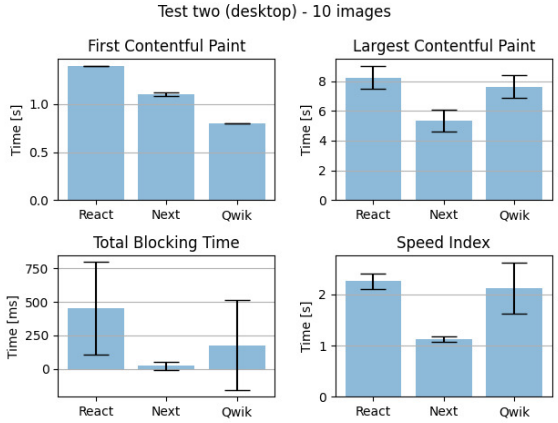

Figure 4: Mean metric values in graphical form obtained in test two for 10 photos.

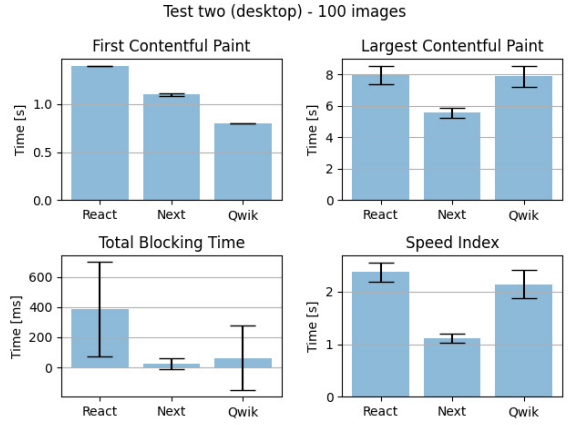

Figure 5: Mean metric values in graphical form obtained in test two for 100 photos.

Upon analyzing the results obtained from the research, it has become apparent that the metric values in individual tests differed significantly from one another. Furthermore, it is worth noting that in most cases, within a given study, analyzed technologies maintained the same trends, thereby providing consistent results, and the frameworks' ranking remained relatively stable throughout the study.

For test one Qwik had the best FCP and LCP times for each test case. Visually the content was also visible the fastest in the case of Qwik. For the time of blocking user interaction the greatest discrepancies can be observed - this time was the largest for Next for a small number of elements, for React.js for the average number of elements, and for Qwik it was the largest for the largest number of elements. Overall, Qwik is the best scorer according to Lighthouse's score calculation [21] but it's not a landslide victory.

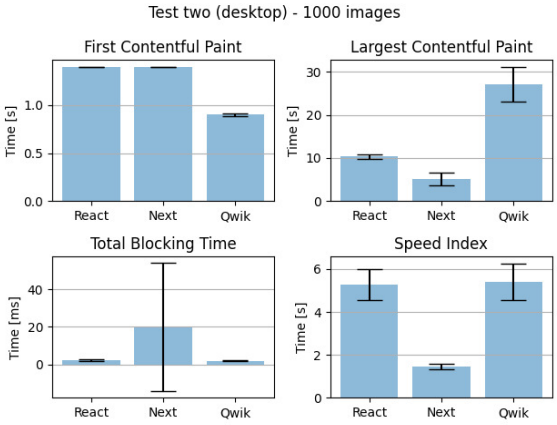

Figure 6: Mean metric values in graphical form obtained in test two for 1000 photos.

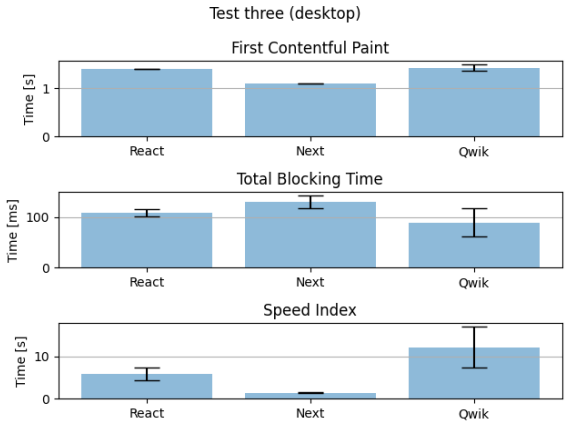

Figure 7: Mean metric values in graphical form obtained in test three.

In the second test once again the lowest FCP time was achieved by Qwik. It can be seen, however, that the situation has changed for LCP and the lowest values were obtained by Next.js while Qwik was classified at the end of the ranking. Next.js also had the best TBT results but was overtaken by Qwik and React.js for the large number of images displayed. For the SI metric, the best results were achieved by Next.js. Qwik and React have kept these times on a similar level with Qwik leading slightly. To sum up this study, and considering that Lighthouse's TBT and LCP [21] times are the most important for performance, Next.js performed significantly better than Qwik in this test while React.js performed the worst.

In the case of the third test, Qwik obtained the best values for the TBT metric but it also had the largest fluctuations of values as evidenced by the high value of

the standard deviation. However, it performed the worst in other metrics where Next.js took the lead and once again the standard deviation for the results obtained by Qwik was the largest.

6. Conclusions

The obtained test results show that each of the tests performed was a different challenge for the analyzed frameworks. The first test went the best, as evidenced by the fact that each of the frameworks obtained satisfactory results allowing for passing the Lighthouse audit with a high score of application optimization. As expected, the increase in the number of rendered elements resulted in an increase in application load times. The biggest surprise in this case, however, was the sudden jump in interaction blocking time for the Qwik framework for the largest number of elements which nevertheless did not stop it from showing that it was the best among the analyzed frameworks at the task of displaying a large number of elements in the DOM tree, thus confirming hypothesis H2.

In general, the results were the worst for the second test which was also characterized by the lowest convergence between the values from subsequent trials. In this case, Next.js showed the best of the frameworks, which may be the result of the use of the Image component which allows optimizing the displayed images [22]. Therefore, Qwik did not obtain results allowing to conclude that it has a chance to improve the performance of an application loading a large number of resources with increased sizes, thus refuting hypothesis H3.

The third test assumed some disadvantages in favor of the Qwik framework - the library used to display the graphs is a React library so Next.js and React.js can use it directly and Qwik must additionally convert the component used from it. Despite the apparent injustice, it must be borne in mind that this framework is new and if we want to use it in a production environment we cannot count on developers to create from scratch every library that has already been created in React.js, especially having the possibility to use it in an application written in Qwik. Therefore, this study made perfect sense as this is a typical case that can occur in a production environment. This probably influenced the result of the study which showed that the application created in Qwik performed worse than other applications, and thus caused hypothesis H4 to be rejected.

Summing up all tests, the application written in Qwik received better metrics results only in the first test so it was not possible to fully demonstrate that this framework allows for better application loading times which was the subject of hypothesis H1.

References

[1] Presentation by M. Hevery on “Qwik + Partytown: How to remove 99% of JavaScript from main thread” at WeAreDevelopers World Congress 2022, https://www.youtube.com/watch?v=0dC11DMR3fU, [13.06.2023].

[2] C. M. Novac, O. C. Novac, R. M. Sferle, M. I. Gordan, G. Bujdosó, C. M. Dindelegan, Comparative study of some applications made in the Vue.js and React.js frameworks, 16th International Conference on Engineering of Modern Electric Systems (EMES), (2021) 1-4, https://doi.org/10.1109/EMES52337.2021.9484149.

[3] Z. Dinku, React.js vs. Next.js, Metropolia University of Applied Sciences, (2022), https://www.theseus.fi/bitstream/handle/10024/750122/Dinku_Zerihun.pdf.

[4] A. Świątkowski, K. Ścibior, Comparative analysis of React, Next and Gatsby programming frameworks for creating SPA applications, Journal of Computer Sciences Institute, 24 (2022) 224-227, https://doi.org/10.35784/jcsi.2972.

[5] T. Fadhilah Iskandar, M. Lubis, T. Fabrianti Kusumasari, A. Ridho Lubis, Comparison between client-side and server-side rendering in the web development, IOP Conference Series: Materials Science and Engineering, 801 (2020) 1-6, https://doi.org/10.1088/1757-899X/801/1/012136.

[6] M. Hakim, Speed index and critical path rendering performance for isomorphic single page applications, Proceedings of the 16th Winona Computer Science Undergraduate Research Seminar, (2016) 41-46, https://cs.winona.edu/cs-website/current_students/Projects/CSConference/2016conference.pdf.

[7] N. K. SG, P. K. Madugundu, J. Bose, S. C. S. Mogali, A Hybrid Web Rendering Framework on Cloud, 2016 IEEE International Conference on Web Services (ICWS), IEEE, (2016) 602-608, https://doi.org/10.1109/ICWS.2016.83.

[8] F. Pavić, L. Brkić, Methods of Improving and Optimizing React Web-applications, 44th International Convention on Information, Communication and Electronic Technology (MIPRO), (2021) 1753-1758, https://doi.org/10.23919/MIPRO52101.2021.9596762.

[9] J. Väyrynen, Ensuring Availability of a Server-Side Rendered React Application: A Case Study, Aalto University, (2019), http://urn.fi/URN:NBN:fi:aalto-201905122998.

[10] A. M. Aladwani, An empirical test of the link between website quality and forward enterprise integration with web consumers, Business Process Management Journal, Emerald Publishing, 12 (2) (2006) 178-190, https://doi.org/10.1108/14637150610657521.

[11] A. M. Aladwani, P. C. Palvia, Developing and validating an instrument for measuring user-perceived web quality, Information & Management, Elsevier, 39 (6) (2002) 467-476, https://doi.org/10.1016/S0378-7206(01)00113-6.

[12] F. Almeida, J. Monteiro, The role of responsive design in web development, Webology, Webology Center, 14 (2) (2017) 48-651, http://www.webology.org/2017/v14n2/a157.pdf.

[13] G. Richards, A. Gal, B. Eich, J. Vitek, Automated construction of JavaScript benchmarks, ACM SIGPLAN, 46 (10) (2011) 677-693, https://doi.org/10.1145/2076021.2048119.

[14] H. Findel, J. Navon, A Test Environment for Web Single Page Applications (SPA), In Proceedings of the 11th International Conference on Web Information Systems and Technologies - WEBIST, (2015) 47-54, https://doi.org/10.5220/0005428000470054.

[15] H. Golestani, S. Mahlke, S. Narayanasamy, Characterization of Unnecessary Computations in Web Applications, IEEE International Symposium on Performance Analysis of Systems and Software, (2019) 11-21, https://doi.org/10.1109/ISPASS.2019.00010.

[16] K. Kiyokawa, Q. Jin, A Front-End Framework Selection Assistance System with Customizable Quantification Indicators Based on Analysis of Repository and Community Data, Big-Data-Analytics in Astronomy, Science, and Engineering, Lecture Notes in Computer Science, Springer, 13167 (2022) 41-55, https://doi.org/10.1007/978-3-030-96600-3_4.

[17] M. Kaluža, K. Troskot, B. Vukelić, Comparison of Front-End frameworks for web applications development, Journal of the Polytechnic of Rijeka, 6 (1) (2018) 261-282, https://doi.org/10.31784/zvr.6.1.19.

[18] L. Borzemski, M. Kędras, Measured vs. Perceived Web Performance, Information Systems Architecture and Technology: Proceedings of 40th Anniversary International Conference on Information Systems Architecture and Technology – ISAT 2019, ISAT 2019, Advances in Intelligent Systems and Computing, Springer, 1050 (2019) 285-301, https://doi.org/10.1007/978-3-030-30440-9_27.

[19] J. Miller, A. Osmani, Rendering on the Web, Google Developers, web.dev, (2019), https://web.dev/rendering-on-the-web/, [20.03.2023].

[20] R. Carniato, Resumable JavaScript with Qwik, DEV Community, (2022), https://dev.to/this-is-learning/resumable-javascript-with-qwik-2i29, [20.03.2023].

[21] How the Performance score is weighted - Lighthouse 10, https://developer.chrome.com/docs/lighthouse/performance/performance-scoring/#lighthouse-10, [25.04.2023].

[22] Next.js Documentation - Image Component and Image Optimization, https://nextjs.org/docs/basic-features/image-optimization, [25.04.2023].