Journal of Emerging Information System and Business Intelligence
ISSN: 2774-3993
Journal homepage: https://ejournal.unesa.ac.id/index.php/JEISBI/

Comparison Of Web Load Speed Between Babel Transpiler and SWC On Website-Based Applications

Rangga Prasetya[1], Bambang Sujatmiko[2]
[1]*Sistem Informasi, Universitas Surabaya*
rangga.190566@mhs.unesa.ac.id, fuukarine@gmail.com
[2] *Sistem Informasi, Universitas Surabaya*
bambangsujatmiko@unesa.ac.id

ABSTRACT

Transpilers play an important role in software development by translating code from one programming language to another, allowing developers to take advantage of modern features and capabilities without having to change the entire project. For example, by using a transpiler like Babel, developers can write the latest JavaScript code that remains compatible with older browsers, similar to translating a book into multiple languages to make it accessible to a wider audience. Babel is the most commonly used transpiler. On the other hand, there is SWC which is a new transpiler that is claimed to be faster than Babel. This study aims to determine the difference in the speed of the Babel and SWC transpilers. The data for this study were taken from several pages using Google Lighthouse. The data were analyzed using a parametric test, namely the paired sample t-test. The results of the study showed that SWC had a significant difference in speed compared to Babel in the First Contenfulpaint (FCP) and Speed Index (SI) indicators. Babel is superior in Total Blocking Time (TBT). While in the Largest Contenfulpaint (LCP), Babel is significantly superior to SWC. This shows that SWC is faster than Babel transpiler in web load speed because in the speed indicators, namely FCP and SI, SWC is significantly superior.

Keyword: Transpiler, SWC, Babel, Google Lighthouse, CRUD.

Article Info:
Article history:
Received December 3, 2024
Revised February 8, 2025
Accepted March 27, 2025

Corresponding Author
Rangga Prasetya
Sistem Infotmatika Universitas Surabaya, Surabaya, and Indonesia
rangga.190566@mhs.unesa.ac.id

1. INTRODUCTION

The importance of transpilers in web development cannot be ignored, especially in this ever-evolving era. With programming languages and web technologies constantly evolving, transpilers have become invaluable tools for web developers. Transpilers convert code from one programming language to another with the aim of simplifying development, increasing efficiency, and expanding the scope of the project [1]. In the dynamic world of the web, transpilers allow developers to exploit the power of different programming languages and frameworks. For example, transpilers can convert modern code written in programming languages such as TypeScript or ECMAScript 6+ into JavaScript code that is more compatible with various browsers. This allows developers to use the latest features in web development without having to worry about cross-platform compatibility [2]. Additionally, transpilers allow for smoother migration in web development. In large projects or legacy code, transpilers can help convert old code to newer and more relevant technologies. This avoids the need for a complete rewrite of the code, which can be very time-consuming and costly. Developers can

switch from one framework or programming language to another by optimizing the existing code [3].

Not only that, transpilers can also help in code optimization. Some transpilers are capable of producing more efficient and compact code, which results in better website performance and faster load times. Developers can write code in a more declarative and expressive style, while transpilers will produce code that is optimized for speed and efficiency. In increasingly complex web development, transpilers serve as a bridge between different languages and technologies, allowing developers to achieve the desired results faster and better. From increasing development speed to expanding migration and optimization opportunities, transpilers are essential tools that support modern and innovative web development [4]. In the context of web development, tools such as Babel and SWC play an important role as transpilers that provide concrete solutions for developers in overcoming challenges related to compatibility, optimization, and efficiency in JavaScript code. Babel has become one of the most famous transpilers in the web development community. It allows developers to write code with the latest features of ECMAScript and convert it into code that is compatible with various older browsers. In modern web development, where JavaScript language standards and features are constantly evolving, Babel helps maintain cross-platform compatibility. Developers can leverage the power of more advanced programming languages without having to worry about whether the code will run correctly on different devices [5].

SWC (Super-fast Web Compiler) is a relatively new transpiler but is gaining recognition in the web development community [4]. SWC aims to provide better performance than other traditional transpilers. Compared to Babel, SWC has a higher compilation speed, which can have a positive impact on development time and website load time. This makes it an attractive option for larger projects or larger scale applications. Both Babel and SWC help developers optimize and maintain code. Apart from its function as a transpilation tool, Babel can also be used to perform additional optimizations, such as unused code removal and compression. While SWC focuses on high performance, it also produces more optimized code and can help in improving the overall performance of a web application [6]. In the web development community, the performance comparison between Babel and SWC transpilers has been an interesting and relevant topic of discussion. Although there are no scientific studies that provide a very detailed direct comparison, some practical experiences and information from the development community provide insight into the performance differences between the two transpilers [7].

Creating a website with CRUD (Create, Read, Update, Delete) operations is one of the important aspects in developing modern web applications. In this context, the speed of the transpiler is a key factor that can affect the performance and responsiveness of the website. A transpiler, or translator compiler, functions to convert source code from one programming language to another, for example from a high-level programming language to JavaScript that can be run in a browser. The importance of transpiler speed in developing a CRUD website lies in the efficiency of converting source code to a language that can be executed on the client-side. With a fast transpiler, the website development process can be accelerated, resulting in a better user experience. A reliable transpiler is able to optimize code efficiently, reduce page load times, and improve overall application performance. Applications with CRUD themselves have higher complexity and are more complicated than applications without CRUD. PT Andromedia is currently developing a CRUD-based application that is aimed at the point of sales category. Where the application is still under development has two versions of the transpiler, namely Babel and SWC, where Babel is used by senior workers while SWC is worked on by new workers at Andromedia. Babel, as an established transpiler, has a long history of helping developers overcome cross-platform compatibility challenges. However, in

some cases, its performance can be less than optimal, especially when dealing with complex and large codes. This is what makes some developers look for alternatives, and this is where SWC comes in.

Phelan indirectly revealed that SWC has faster performance than Babel. However, measuring performance differences scientifically can be challenging, because results can vary depending on the context of the code, project size, and testing environment. In-depth empirical studies will require careful and repeated trials to collect accurate and comprehensive data [8]. Although there has been no scientific research that produces a universally accepted direct comparison, feedback from the development community suggests that SWC tends to have better compilation speeds in some situations. However, there are also other factors to consider when choosing a transpiler, such as the features provided, the plugin ecosystem, and the ability to generate optimized code [9]. In practice, a faster transpiler can result in shorter development times and better web application load times. However, the decision to choose a transpiler should be based on the needs of the project, the context of the code, and development priorities. As part of the evolution of web development, Babel and SWC show that performance improvements are something that developers value, while still considering other relevant factors in the software development process [8].

2. METHODS

The following is the research flow that will be carried out by researchers to test the performance of each compiler through the rendering process:

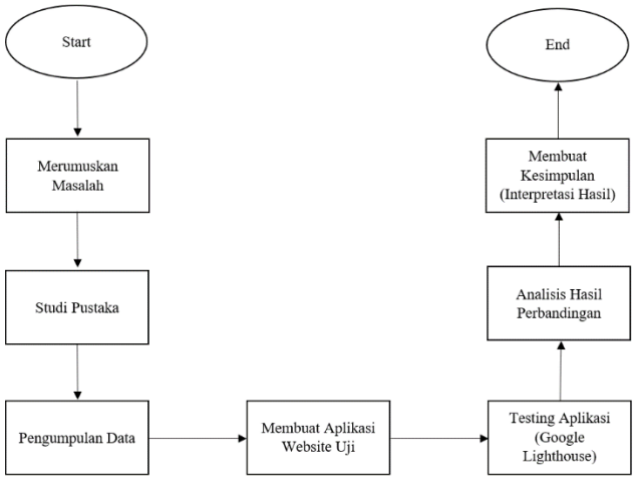

Fig. 1 Flowchart of Research Flow.

The following is an explanation of the flow above:

a. First, the formulation of the problem is in accordance with the description in the background.
b. Second, the author conducts a literature study with the aim of finding references from previous research and references from the theories used and will be tested in this study.
c. The third stage, based on the findings of the previous literature study, data is collected to create a dummy application or simulation that will be tested later.
d. The fourth stage, where the researcher creates a dummy application or simulation application that will be tested using the Babel and SWC transpilers.
e. The fifth stage, the researcher conducts testing with Google Lighthouse on the designed application. Testing will later be carried out 5 times or more to get maximum results.
f. The sixth stage or core stage, where the testing results from the two transpilers come out, then the results are analyzed and compared with each other.

g. Finally, conclusions or interpretation of the results are drawn as the author's final study.

Testing applications with Google Lighthouse is an approach to measure and optimize the performance, accessibility, appearance, and best practices of a website or web application using a tool called "Lighthouse" developed by Google [10]. Lighthouse is an open-source tool that can be used to audit websites or web applications and provide recommendations to improve user experience and performance [11]. After both applications have been developed, the next step is to test the rendering performance with the help of the Google Lighthouse tool. In the testing process, Google Lighthouse uses the parameters in the model below with 1 additional parameter, namely the Speed Index. Reported from the official Google Lighthouse documentation page, the following are the parameters measured along with the categorization of rendering time on Google Lighthouse:

Table 1. Parameter Speed Test

Parameter	Kategori	Waktu *Rendering*
First Contentful Paint	Fast (Hijau)	0 - 1.8 *seconds*
	Average (Orange)	1.9 - 3 *seconds*
	Slow (Merah)	> 3 *seconds*
Cumulative Layout Shift	Fast (Hijau)	0 - 0.1 %
	Average (Orange)	0.2 - 0.25 %
	Slow (Merah)	> 0.25 %
Total Blocking Time	Fast (Hijau)	0 - 200 *milliseconds*
	Average (Orange)	201 - 600 *milliseconds*
	Slow (Merah)	> 600 *milliseconds*
Largest Contentful Paint	Fast (Hijau)	0 - 2.5 *seconds*
	Average (Orange)	2.6 - 4 *seconds*
	Slow (Merah)	> 4 *seconds*
Speed Index	Fast (Hijau)	0 - 3.4 *seconds*
	Average (Orange)	3.5 - 5.8 *seconds*
	Slow (Merah)	> 5.8 *seconds*
Final Score	Fast (Hijau)	90 - 100
	Average (Orange)	50 - 89
	Slow (Merah)	0 - 49

Table 2. Proportion of Speed Test

Parameter	Bobot
First Contentful Paint	10 %
Cumulative Layout Shift	15 %
Total Blocking Time	10 %
Largest Contentful Paint	30 %
Speed Index	25 %
Total	**100%**

Data collection is the process of collecting relevant information or facts from various sources or respondents for the purpose of analysis, research, decision making, or other purposes [12]. On the other hand, data collection is also the initial step in the process of further information processing, the data collected can be in the form of numbers, facts, opinions, or other information relevant to a particular purpose [13]. Transpiler analysis involves converting code from one programming language to another programming language [14]. Collecting the data required for transpiler analysis is essential to ensure that the conversion process runs smoothly and the results are accurate.

In this study, there will be 2 similar applications, but the Transpiler used is different. The application being tested will render all previously collected data at once. The Point of Sales (POS) application "Kopi Racik" is a prototype software application specifically designed to facilitate the management of sales transactions, inventory management, and customer service at the "Kopi Racik" coffee shop. This application is designed with the main CRUD (Create, Read, Update, Delete) feature that allows users to perform various operations, including

creating orders, viewing order status, browsing the menu, and managing product and customer information.

The CRUD feature in this application plays a key role in allowing "Kopi Racik" users to easily manage and update data. With the Create feature, users can create new orders by adding menu items to the shopping cart. Order information such as type of drink, quantity, and additional preferences can be adjusted according to customer wishes. After an order is created, the Read feature allows users to view the order status in real-time, including the preparation, completion, and delivery stages. In addition, the "Kopi Racik" application is also equipped with an Update feature that allows users to change order details, such as adding or reducing the number of items or changing additional preferences. This feature is very useful in adjusting customer orders that may experience changes or adjustments. In addition, the Delete feature allows users to delete unnecessary or incorrect orders. The "Kopi Racik" application also provides easy access to the coffee shop's complete menu, allowing customers to explore the various choices of drinks and food available. Detailed information such as product descriptions, prices, and availability can also be accessed quickly through this application. In addition, "Kopi Racik" users can also manage product and customer information through the CRUD feature, including adding, editing, or deleting products from the menu, as well as managing customer lists and their preferences.

In modern software development, selecting the right tools and technologies is critical to ensure optimal performance and efficiency. One important aspect of web application development is the use of JavaScript transpilers, which convert JavaScript code written in a newer version to an older version that is compatible with older browsers. In the context of the "Kopi Racik" application, transpilers are used to ensure optimal performance and cross-platform compatibility. The "Kopi Racik" application was developed using two leading transpilers: SWC and Babel. A comparison of the speed of the two is key in determining which transpiler is best suited to the needs of this application.

The comparison of variables in this study is to test the differences between two variables, and in this study it refers to two Babel and SWC transpiler data, namely as follows:

Table 3. Independent Sample t-Test

Group Statistics

	Jenis Database	N	Mean	Std. Deviation	Std. Error Mean
Response Time Database	Database MySQL	42	7.1905	2.83048	.43675
	Database MongoDB	42	34.0714	64.10182	9.89113

Independent Samples Test

		Levene's Test for Equality of Variances				
		F	Sig.	t	df	Sig. (2-tailed)
Response Time Database	Equal variances assumed	13.197	.000	-2.715	82	.008
	Equal variances not assumed			-2.715	41.160	.010

The independent sample t-test aims to compare the means of two unpaired or unrelated groups. The Polled Variance formula is used for the t-test for the same variance.:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

Fig. 5 Equal variance t-test formula

This leads to the hypothesis testing in this study, namely:

H0: there is no difference in the mean between one transpiler and another.

H1: there is a difference in the mean between one transpiler and another.

3. RESULTS AND DISCUSSION

The comparison of variables in this study is to test the differences between two variables, and in this study it refers to two Babel and SWC transpiler data, namely as follows:

Table 4. Paired Sample T Test First Contentfulpaint (FCP)

Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
					95% Confidence Interval of the Difference				
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper			
Pair 1	Babel FCP - SWC FCP	791.50000	311.35983	98.46062	568.76659	1014.23341	8.039	9	.000

Based on the test results in the table above, Paired Samples Test, it can be seen that the average value when the two transpilers are combined is 568.76659. This indicates that at the lowest speed value, Babel is 568.76659 longer. While the Upper value shows a value of 1014.23341. This indicates that Babel is 1014.23341 milliseconds longer on the FCP indicator page than SWC. While the resulting t-statistic value is 8.039 with a significance value of 0.000. A significance value of less than 0.05 indicates that there is a significant difference between the Babel and SWC transpilers on the First Contentfulpaint (FCP) indicator with SWC being faster than Babel.

Table 5. Paired Sample T Test Largest Contentfulpaint (LCP)

Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
					95% Confidence Interval of the Difference				
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper			
Pair 1	Babel LCP - SWC LCP	25.20000	340.42908	107.65313	-218.32829	268.72829	.234	9	.820

Based on the test results in the LCP table, Paired Samples Test, it can be seen that the average value when two transpilers are combined is 25.2 milliseconds. The lower value shows that the lowest value of the Babel transpiler than the SWC transpiler is -218.32829. This indicates that at the highest speed value, Babel is faster by 218.32829. While the Upper value shows a value of 268.72829. This indicates that Babel is 268.72829 milliseconds longer on the LCP indicator than SWC. While the resulting t-statistic value is 0.234 with a significance value of 0.820. A significance value of more than 0.05 indicates that there is no significant difference between the Babel and SWC transpilers on the LCP indicator.

Table 6. Paired Sample T Test Total Blocking Time (TBT)

Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
					95% Confidence Interval of the Difference				
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper			
Pair 1	Babel TBT - SWC TBT	-163.70000	40.01680	12.65442	-192.32630	-135.07370	-12.936	9	.000

Based on the test results on the landing page table, Paired Samples Test, it can be seen that the average value when the two transpilers are combined is 163.7 milliseconds. The lower value shows that the lowest value of the Babel transpiler than the SWC transpiler is -190.326. This indicates that at the lowest speed value, Babel is faster by 190.326. While the Upper value shows a value of -135.0747. This indicates that Babel is 135.0747 milliseconds faster on the TBT indicator than SWC. While the resulting t-statistic value is 12.936 with a significance value of 0.000. A significance value of less than 0.05 indicates that there is a significant

difference between the Babel and SWC transpilers on the TBT indicator. A negative t-value indicates that Babel is faster than SWC for the TBT indicator.

Table 7. Paired Sample T Test Speed Index (SI)

Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
					95% Confidence Interval of the Difference				
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper			
Pair 1	Babel SI - SWC SI	2891.30000	2176.07613	688.13569	1334.62891	4447.97109	4.202	9	.002

Based on the test results in the overall total table, Paired Samples Test, it can be seen that the average value when the two transpilers are combined is 2891.3 milliseconds. The lower value shows that the lowest value of the Babel transpiler than the SWC transpiler is 1334.629. This indicates that at the lowest speed value, SWC is 1334.629 faster. While the Upper value shows a value of 1683.7. This indicates that Babel is 4447.971 milliseconds longer on the speed index indicator than SWC. While the resulting t-statistic value is 4.202 with a significance value of 0.002. A significance value of less than 0.05 indicates that there is a significant difference between the Babel and SWC transpilers on the speed index (SI) indicator on the website or application used, with SWC being faster than Babel on the SI indicator.

Based on the results of the analysis, it is known that the SWC transpiler is significantly faster than the Babel transpiler. Especially in the FCP and Speed Index indicators. Statistically, it is proven that there is a difference between the speed values produced by the Babel and SWC Transpilers with a significance value of less than 0.05 with Babel being faster only in the TBT indicator. Then in LCP there is no significant difference in both transpilers. Meanwhile, based on descriptive statistics, it is known that the SWC speed value is higher than Babel with a millisecond speed produced for the entire page except TBT. This could be because SWC is written in the Rust programming language, which is known for its high performance and efficiency. Rust allows SWC to manage memory very effectively and run computational operations at higher speeds. Rust also provides strong memory security without sacrificing performance, which means that code written in Rust tends to be faster and safer than code written in JavaScript or even in other languages such as C++.

The results of this study are in line with the statement put forward by Nguyen, who in his writing stated that SWC is used in Next JS because it is more effective for webload and takes into account the number of components used [15]. In his research, he stated that SWC uses a very efficient architecture in the parsing and compilation process. SWC has a parsing algorithm that is optimized for speed, allowing faster source code parsing. After parsing, SWC also performs code transformations faster thanks to optimizations performed at a very low level. For example, SWC is able to manage AST (Abstract Syntax Tree) in a very efficient manner, reducing the overhead that often occurs in code transformation.

The results of this study also support the results of research conducted by Sasikumar et al., where SWC, which is a new transpiler, is the most efficient transpiler than other transpilers used for webload, especially those that lead to faster rendering [16]. This could be because SWC fully supports multi-threading, which means it can utilize all available CPU cores to speed up the compilation process. This is very different from Babel which generally runs in a single-threaded environment due to the inherent nature of JavaScript. By utilizing multi-threading capabilities, SWC can process multiple files in parallel, which significantly reduces the overall compilation time, especially on large projects with many files.

This study is also supported by research conducted by Park et al. which states that the SWC transpiler is indeed faster than Babel [17]. This is due to the nature of the SWC ecosystem and architecture which is more modern than Babel. SWC was designed with performance in mind from the start, while Babel, which is older, initially focused on flexibility and ease of use.

While Babel has made numerous optimizations over the years, its underlying architecture still has some performance limitations that are difficult to overcome without major changes.

SWC's claim that it is faster than other transpilers, especially Babel, is supported by a number of independent studies and benchmarks. In various tests, SWC performs significantly faster than Babel. The study conducted in this study showed that SWC can compile JavaScript code up to five times faster than Babel. The tests covered a range of project sizes, from small to large, and in almost all cases, SWC consistently showed shorter compile times. This speedup is not only visible in the initial compile time but also in watch mode, where incremental changes are applied in real-time.

CONCLUSION

Based on the research results, it was found that the SWC transpiler is superior to SWC in the First Contentfulpaint (FCP) and Speed Index (SI) indicators only, while in the Total Blocking Time (TBT) indicator Babel is superior to SWC. Then in the Largest Contentfulpaint (LCP) indicator there is no significant difference between the Babel and SWC transpilers. Overall, SWC is superior in terms of speed, namely in FCP and SI which better represent web load speed. SWC, written in Rust, has advantages in memory efficiency and computing speed. Rust allows SWC to run operations faster and safer than JavaScript or C++. This study supports Nguyen's findings that SWC is effective for webload and has an efficient architecture in parsing and compiling. SWC optimizes the Abstract Syntax Tree (AST) efficiently and supports multi-threading, enabling parallel processing that reduces compilation time. Other studies by Sasikumar et al. and Park et al. also confirm SWC's superiority over Babel, especially in large projects. Independent benchmarks show that SWC can compile JavaScript code up to five times faster than Babel, in both early compilation and watch mode.

REFERENCES

[1] A. R. Sofyan and S. D. Y. Kusuma, "Implementasi Load Balancing Web Server menggunakan Haproxy pada Virtual Server Direktorat SMK Kemendikbudristek," *J. Pendidik. Tambusai*, vol. 6, no. 2, pp. 9669–9682, 2022.

[2] A. A. Yusuf, "Analisis Static Site Generator pada Web Responsif Portal Berita= Static Site Generator Analysis on Responsive Web News Portal," 2022, *Universitas Hasanuddin*.

[3] H. M. Elmatsani, "Desain Metode PrefetchCache untuk Peningkatan Kinerja Aplikasi Web," *Techno. Com*, vol. 19, no. 2, pp. 147–156, 2020.

[4] I. Nuryasin and Z. Sari, "Optimasi Kecepatan Loading Time Web Template Dengan Implementasi Teknik Front-End," *J. Repos.*, vol. 2, no. 11, pp. 1456–1463, 2020.

[5] V. L. S. Putra, "Optimasi Load Time Halaman Website Produk Kesehatan Dengan Plug-In Datatables," 2023.

[6] S. D. Riskiono and D. Darwis, "Peran Load Balancing Dalam Meningkatkan Kinerja Web Server Di Lingkungan Cloud," *Krea-TIF J. Tek. Inform.*, vol. 8, no. 2, pp. 1–8, 2020.

[7] A. P. D. G. Andini, D. Wahyuningsih, and M. Yunus, "Analisis Dan Peningkatan Performa Aplikasi Berbasis Website Menggunakan Stress Tools Gtmetrix," *Temat. J. Teknol. Inf. Komun.*, vol. 9, no. 2, pp. 191–201, 2022.

[8] R. Phelan, "An Evaluation on the Performance of Code Generated with WebAssembly Compilers," 2021.

[9] K. Juan and S. Budi, "Pengembangan Menu Digital Menggunakan ReactJs Implementasi Hasil Belajar Studi Independen di Frontend Engineering Program

Ruangguru CAMP (Career Acceleration Bootcamp)," *J. Strateg. Maranatha*, vol. 5, no. 1, pp. 130–142, 2023.

[10] T. Heričko, B. Šumak, and S. Brdnik, "Towards Representative Web Performance Measurements with Google Lighthouse," in *Proceedings of the 2021 7th Student Computer Science Research Conference*, 2021, p. 39.

[11] T. McGill, O. Bamgboye, X. Liu, and C. S. Kalutharage, "Towards Improving Accessibility of Web Auditing with Google Lighthouse," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, IEEE, 2023, pp. 1594–1599.

[12] E. Surahman, A. Satrio, and H. Sofyan, "Kajian Teori Dalam Penelitian," *JKTP J. Kaji. Teknol. Pendidik.*, vol. 3, no. 1, pp. 49–58, 2020, doi: 10.17977/um038v3i12019p049.

[13] M. R. Fadli, "Memahami desain metode penelitian kualitatif," *Humanika*, vol. 21, no. 1, pp. 33–54, 2021, doi: 10.21831/hum.v21i1.38075.

[14] D. D. Tambunan, "Perbandingan Analisis Aplikasi Database NoSQL Redis dan SQL MySQL," 2016, *Universitas Widyatama*.

[15] A. Nguyen, "Building an E-commerce Website Using Next Js, Mantine, and Strapi," 2022.

[16] S. Sasikumar, S. Prabha, and C. Mohan, "Improving Performance Of Next. Js App And Testing It While Building A Badminton Based Web App," *Js App Test. It While Build. A Badmint. Based Web App (May 27, 2022)*, 2022.

[17] J. Park, D. Youn, K. Lee, and S. Ryu, "Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations," *Proc. ACM Program. Lang.*, vol. 7, no. PLDI, pp. 493–515, 2023.