Stress Testing on PalmViews Application: Case Study of Daily Production Monitoring Application at PT Perkebunan Nusantara IV Regional II

Rendio P. Simamora[1], Dino A. Harahap[2], Brema A. Siregar[3]

[1,2,3]*Department of Computer and Informatics Engineering, Politeknik Negeri Medan, Medan, Indonesia*

E-mail: srinovida@polmed.ac.id

Abstract. Performance testing is a crucial aspect in ensuring software quality, especially in systems with a high number of users. This research analyzes the performance of the PalmViews application through the stress testing method using K6, an open-source script-based test tool. The test was conducted by simulating a gradual spike in user load to evaluate response time, throughput, and error rate. Results showed a performance degradation when the number of virtual users exceeded the service capacity, indicating the system's tolerance limit. These findings form the basis for future application performance improvement and optimization.

Keyword— stress testing, K6, system performance, software testing, user load

1. Introduction

System performance is one of the crucial aspects in software development, especially for applications that are used on a large organizational scale and involve simultaneous access by many users. Performance testing is crucial in software development to ensure system reliability and optimal functionality under various load conditions [1]. When the system is unable to handle load spikes, the consequences can be in the form of decreased quality of service, increased response time, and system failure. This not only impacts the user experience, but can also disrupt business processes and decision-making that depend on system reliability. Therefore, performance testing is an important part of the software development cycle to ensure that applications are able to operate optimally under various load conditions, including extreme conditions.

PalmViews is a daily palm oil production monitoring application used in PT Perkebunan Nusantara IV Regional II. This application is used by various work units to record and monitor production data in real-time. Given its crucial function in supporting daily operations and managerial decision-making, the PalmViews application must be able to maintain reliable performance despite being accessed by many users simultaneously, especially during peak hours such as shift end or daily recapitulation time.

One approach used to measure system resilience to extreme loads is stress testing. This test aims to determine the maximum load limit that can still be tolerated by the system by simulating usage scenarios that exceed normal capacity. Stress testing helps identify potential points of failure, such as bottlenecks

in data processing or delays in server response [5]. The results of these tests can be an important basis for planning capacity increases and system optimization.

In addition, to ensure the quality of testing and the efficiency of system performance objectively, an evaluation approach based on ISO/IEC 25010 can also be applied, as described in research by Kurniasari and Rochimah. In this research, testing was conducted on various web transactions using several frameworks and evaluated based on three main performance sub-characteristics: resource utilization, time behavior, and capacity. By adopting these principles, stress testing on the PalmViews application can be tailored to provide a more comprehensive picture of the performance and efficiency limitations of the system in a complex operational environment [14]

To perform stress testing, this research uses K6, an open-source JavaScript-based performance testing tool. K6 allows simulation of a large number of loads with flexible configurations and supports integration with CI/CD pipelines. In addition, test results from K6 can be visualized in real-time through Grafana, making it easier to analyze overall system performance (Grafana Labs, 2025). Previous studies by Hartono and Ardiansyah (2021) showed that K6 has advantages in terms of scalability, efficient use of resources, and ease of writing and maintaining test scenarios.

The purpose of this research is to evaluate the performance of the PalmViews application through the stress testing method using the K6 tool. This evaluation was conducted by developing test scenarios involving a gradual increase in the number of virtual users to measure performance metrics such as response time, number of requests per second (throughput), and error rate[6].

2. Literature Review

2.1. System Performance Testing

System performance testing is a process that aims to evaluate the extent to which an application-such as an Application Programming Interface (API) or web application-can run properly under various load conditions [2]. The focus of this test is to measure important metrics, including response time, concurrent user capacity, and number of requests per second. In the research conducted by [3], performance testing focused on a RESTful API for a diet application that uses image recognition technology to identify foods and provide nutritional information.

Recent studies have explored different methodologies for performance testing. For example, research by Huerta-Guevara et al. introduced dynamic workload adaptation to enhance efficiency in performance testing. This approach reduces trial-and-error cycles by adjusting workloads in real-time based on application behavior, ensuring more accurate identification of performance bottlenecks [10].

This testing process uses the K6 appliance to simulate high loads and evaluate how well the system can handle large HTTP requests. The measured metrics, such as http request duration and number of requests, provide a clear picture of the system's performance, so that its reliability and efficiency can be assessed [4]. This approach is especially important for testing applications such as PalmViews, which need to evaluate system stability when active users vary. evaluate system stability when active users vary.

In addition, a similar approach was also seen in Indrianto's (2023) research, which used Apache JMeter and BlazeMeter to test the performance of a web-based teacher information system. The study shows that although the average response time remains stable for 50 to 100 users, an increase in the number of users has a significant impact on throughput and standard deviation. This emphasizes the importance of conducting performance testing in stages to identify the maximum capacity of the system and prevent a decline in service quality when user load increases [15].

2.2. Stress Testing

Stress testing is a type of non-functional testing that aims to assess system performance when faced with very heavy loads, such as when many users access simultaneously or get a large number of HTTP requests. Stress testing takes performance evaluation further by pushing the application to its maximum limits or even beyond its capacity. By applying extreme loads, stress testing shows how the system

behaves when resources are limited, database connections are saturated, or hardware components are under pressure [9]. A good system must have robust API quality, and to ensure the quality of the API in the system, testing such as stress testing is required [12]. The purpose of this test is to find the limit of system capacity, response time, and failure rate that may occur. In research conducted by [3], stress testing using K6 on the dietary API, with two scenarios: one with 10 virtual users sending 1000 requests, and the other with 15 virtual users making 750 requests. The results showed an average response time of 1 second, with no failures in the first scenario, while the second scenario recorded a failure rate of 27%. This method can also be applied to PalmViews to identify weak points that appear when the system is under stress. In addition, a similar approach was also used in research by Li et al., who developed a lightweight simulation framework called IoTECS to efficiently perform stress testing on IoT cloud systems. By leveraging symbolic representations of IoT devices and grouping edge devices into simulation nodes, this method has proven capable of accurately identifying system capacity limits. This concept can serve as a reference for stress testing the PalmViews application, particularly to ensure the stability and reliability of the daily production monitoring system at PT Perkebunan Nusantara IV Regional II when facing simultaneous high access loads [13].

2.3. K6 as Testing Tool

K6 is an open-source performance testing tool developed by Grafana Labs. It is designed to test APIs, microservices, and websites through virtual user simulations. By using JavaScript to construct test scenarios, K6 is able to provide various important metrics such as response time (http_req_duration), number of failed requests (http_req_failed), as well as throughput (http_reqs). In the research by [3], K6 was used to perform stress testing on the diet API, which resulted in metrics such as a response time of 1 second and throughput between 7.26 to 8.96 requests per second. K6 is well suited for PalmViews testing due to its flexible ability to simulate high loads.

2.4. Related Research

Research on stress testing using K6 shows how effective this method is in assessing system reliability. Risqulla and colleagues (2024) tested the dietary API with K6, and they obtained throughput between 7.26 and 8.96 requests per second, and noted failure variations that depended on the number of virtual users. In addition, [2] tested a website called Classes and found that performance remained stable for 50 users, with response times of around 1 to 1.3 seconds. However, they also noted a significant number of failures, such as 11,852 failures that occurred on the admin page. These two studies highlight how important it is to customize test parameters to match real-world scenarios [8], which is especially relevant for PalmViews testing using the K6.

3. Research Methodology

3.1. Research Object

The object of this research is the PalmViews application, a web-based fullstack information system used by PT Perkebunan Nusantara IV Regional II to monitor daily palm oil production. This application integrates the user interface and backend logic in a single system. Data communication is done through AJAX to access internal endpoints, such as production data retrieval, report recap, and daily input. AJAX technology enables asynchronous data exchange between the client and server, reducing latency and improving user experience. According to research by Dewan et al., AJAX enhances real-time monitoring capabilities by optimizing data transfer efficiency and minimizing network congestion [11]. Because it is used regularly by many users from various work units, the stability of application performance when receiving many requests simultaneously is crucial to ensure smooth operations [7]. Tests were conducted directly on the AJAX endpoints used by the main features in the application.

3.2. Tools

In this research, stress testing was conducted using a set of integrated tools to evaluate the performance of the PalmViews application. The primary tool used was K6, a JavaScript-based open-source performance testing tool designed to simulate user load on the PalmViews API. To store the test result metrics generated by K6, InfluxDB, a time-series database, was utilized. For visualization purposes, Grafana was employed to connect with InfluxDB and present key performance indicators such as response time and error rate in an intuitive dashboard format. The system used for testing had the following specifications: an Intel Core i3 processor, 4 GB of RAM, and ran on Ubuntu 24.04 LTS operating system. These specifications provided a realistic environment for conducting stress tests representative of standard hardware setups.

3.3. Testing Scenario

The test was conducted by developing a stress testing scenario using the K6 script to access the API endpoint from PalmViews gradually with increasing load. The test scenario consists of several stages as follows:

Table 1. The test scenario

Stages	Number of Virtual Users (VU)	Testing Duration	Objective
Stage 1	10 VU	2 Minute	Light load simulation
Stage 2	50 VU	2 Minute	Medium load simulation
Stage 3	100 VU	2 Minute	Heavy load simulation
Stage 4	200 VU	2 Minute	Extreme load simulation

Endpoints tested included:

- GET /dashboard/produksi
  Display daily production data.
- POST /dashboard/ajax/dashboard/data
  Display production dashboard data.

Metrics observed include:

- Response time (avg, p95): Average and 95th percentile time in responding to requests.
- Throughput (requests per second): The number of requests that can be processed per second.
- Error rate: Demand failure percentage.

3.4. Testing Architecture

Tests were conducted in a local environment using the following architecture:

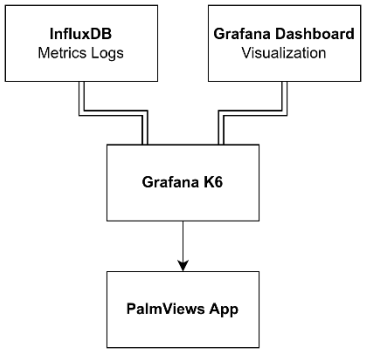

Figure 1. Local Stress Testing Architecture

K6 is run via CLI and configured to send metrics to InfluxDB. InfluxDB holds real-time performance data which is then visualized in the Grafana Dashboard. This visualization helps researchers in analyzing performance changes as the load increases.

4. Results and Discussion

4.1. Test design

Testing was conducted on two main endpoints in the PalmViews application, namely:

- GET /dashboard/production - Display daily palm oil production data in full page form.
- POST /dashboard/ajax/dashboard/data - Displays production dashboard data through AJAX requests.

Testing was conducted using K6 for ±8 minutes, with load scenarios simulating up to 200 virtual users (VU). The results of the tests were analyzed to determine system performance in terms of response speed, stability, and resilience to traffic spikes.

4.2. Test Results per Endpoint

4.2.1. Endpoint Test Results GET /dashboard/production

After testing with a maximum configuration of 200 virtual users and a duration of 8 minutes, this endpoint showed a relatively average performance but started to show heavy load with some requests failing (timeout).

- Total requests: 10.856
- Average response time: 2.88 seconds
- Median (p50): 1.84 seconds
- p90: 6.60 seconds
- p95: 8.54 seconds
- Max response time: 30.59 seconds
- Error rate: 0.04% (5 failed/timed out requests)

The total number of iterations reached 31,235 with only a few extreme variations at the maximum value. This shows that the endpoint is relatively lightweight and able to handle a large number of requests in a stable manner.

4.2.2. Endpoint Test Results POST /dashboard/ajax/dashboard/data

Unlike the GET endpoint, the POST endpoint that handles AJAX requests shows a heavier performance:

- Total requests: 3.008
- Average response time: 10.24 seconds
- Median (p50): 8.77 seconds
- p90: 22.17 seconds
- p95: 24.59 seconds
- Max response time: 60.0 seconds
- Error rate: 0.03% (1 failed request)

Although the number of requests is less, the response time is much higher. This shows that the backend takes longer to process AJAX requests that load dynamic data and possibly perform many complex queries.

4.3. Comparison Discussion

The following table summarizes the performance comparison of the two endpoints:

Table 2. Performance Comparison of Two Endpoints

Metric	GET /dashboard/produksi	POST /ajax/dashboard/data
Total Request	10.856	3.008
Average response time	2.88 s	10.24 s
Median (p50)	1.84 s	8.77 s

p90	6.60 s	22.17 s
p95	8.54 s	24.59 s
Max response time	30.59 s	60.0 s
Error rate	0.04%	0.03%

From the data, it can be concluded that the GET endpoint is much more efficient and stable, even though the number of requests is much larger. This is likely due to the fact that the GET endpoint only serves HTML pages and more static data, while the POST endpoint requires dynamic data processing and composing complex JSON.

4.4. Visualization Through Grafana

This test was also integrated with Grafana using data from InfluxDB. Visualization of performance metrics such as:

- Average response time of both endpoints
- Number of requests per second (RPS)
- Number of active virtual users
- Error rate and response time distribution

Graphs from Grafana show an increasing trend in response time as virtual users grow, especially on the POST endpoint. Spikes are also evident in the p95 response time graph.

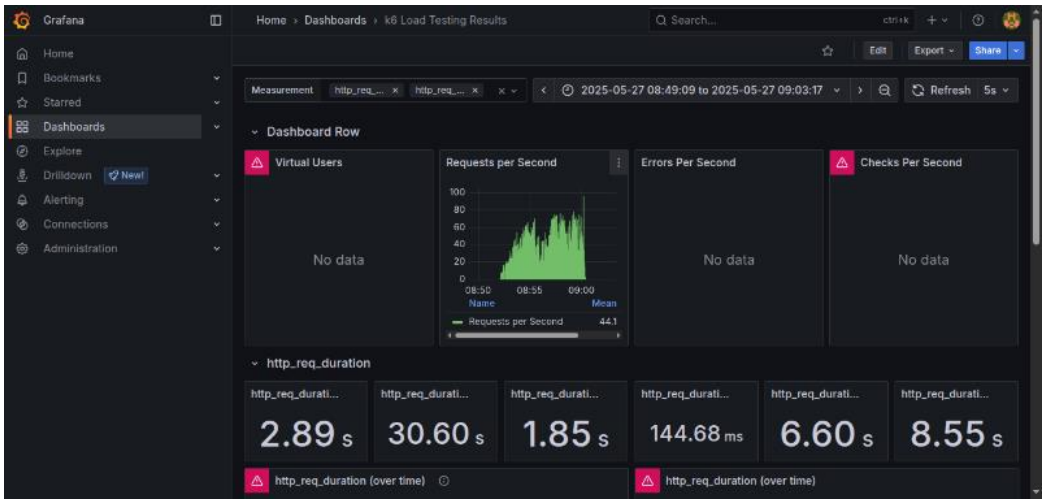

Figure 2. Graphic Grafana GET /dashboard/produksi

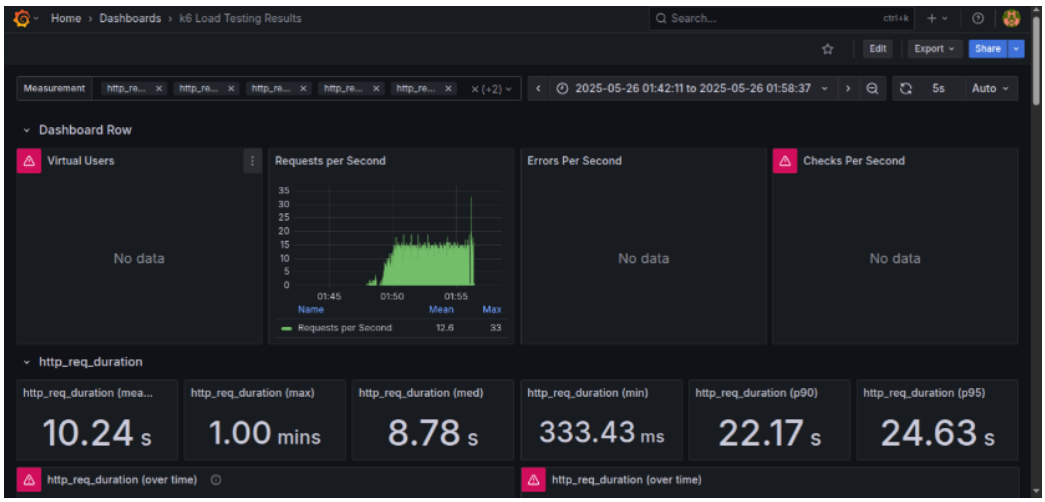

Figure 3. Graphic Grafana POST /ajax/dashboard/data

4.5. System Evaluation

The results of stress testing the PalmViews application show that the system performs quite well in handling a large number of requests, but starts to experience performance degradation at certain points, especially when the number of virtual users (VU) approaches the maximum limit. The evaluation was conducted based on three main aspects, namely response time, error rate, and system stability.

4.5.1. Response Time

Response time is the main indicator to measure the application's ability to respond to user requests. Based on the data:

- The GET /dashboard/production endpoint has an average response time of 2.88 seconds with a p95 of 8.54 seconds, and a maximum value of up to 30.59 seconds. This shows that under high load conditions, some users will experience delays in accessing data.
- The POST /dashboard/ajax/dashboard/data endpoint shows a much higher response time, with an average of 10.24 seconds and a p95 of 24.59 seconds, even reaching a maximum of 60 seconds. This indicates that the endpoint is much heavier and prone to bottlenecks.

In addition to the average response time, this test also considers the percentile value as an indicator of service quality:

- p50 (median) indicates the response time experienced by half of all requests. This value represents the average user.
- p90 indicates that 90% of requests complete faster than this value, meaning that only 10% of users experience slower times.
- p95 is the worst response time still experienced by 95% of users, and is used to identify the worst user experience in the majority group.

The high p90 and p95 values, as seen in the POST /dashboard/ajax/dashboard/data endpoint which reached a p95 above 24 seconds, indicate that some users will experience serious delays in responding to requests, especially at high loads. While on the GET /dashboard/production endpoint, the p95 value is still below 10 seconds, which is considered within acceptable limits, but still requires attention for performance improvement.

4.5.2. Error Rate and Stability

During testing, the error rate (http_req_failed) remained in the low range:

- GET /dashboard/production: 0.04% (5 out of 10,856 requests)

- POST /dashboard/ajax/dashboard/data: 0.03% (1 out of 3,008 requests)

Although these values are still within acceptable limits, the presence of request timeouts especially on the GET endpoint indicates that the system starts to become unresponsive when it reaches maximum load. This error appears consistently during peak load (200 VUs), indicating a weak point in terms of server capacity, backend logic, or database query efficiency.

4.5.3. System Stability Analysis from Grafana

From the visualization in Grafana Dashboard, we can see a pattern of drastic spikes in response time and resource usage when the scenario reaches the final phase (peak load). This pattern supports the K6 data, where response time continues to increase as the load rises. CPU and memory usage (if monitored) also show significant spikes. This indicates the need for system optimization, such as:

- More efficient database queries
- Implementation of cache on dashboard data
- Rate limiting or throttling for extreme loads
- Server-side logic optimization for dashboard data processing

4.5.4. Recommendation

Based on the evaluation of the test results, some recommended corrective measures to improve the performance of the PalmViews application are:

1. Refactoring the POST endpoint, considering its response time is much higher than GET.
2. Implementation of cache (e.g. Redis) for frequently accessed data on the dashboard.
3. Increase infrastructure capacity or implement auto-scaling, if high loads often occur at certain times.

5. Conclusion

Based on the results of performance testing using the stress testing method with the K6 tool and visualization integration through Grafana Dashboard, it can be concluded that the PalmViews application, as a daily palm oil production monitoring system at PT Perkebunan Nusantara IV Regional II, has a fairly stable performance under normal load conditions, but shows signs of degradation when faced with extreme loads.

Tests were conducted on two main endpoints, namely GET /dashboard/production and POST /dashboard/ajax/dashboard/data. The test results show that the GET endpoint has a lower average response time than the POST endpoint. However, both endpoints experienced a significant increase in response time when the number of virtual users approached the maximum (200 VUs), even resulting in a request timeout in the peak scenario. This shows that the system has limitations in handling consistently high loads, especially on endpoints with complex processing loads.

The overall error rate is relatively low (<0.05%), but it remains an important indicator to consider the long-term stability of the system. Response times exceeding 8 seconds for 5% of users (p95) can negatively impact the user experience, especially if they occur on an ongoing basis.

As such, this stress test successfully identified performance tipping points and provided a comprehensive overview of the system's tolerance limits to extreme loads. These results are expected to be the basis for decision-making in making system optimizations, such as increasing backend efficiency, implementing caches, or scaling infrastructure to support larger-scale use.

References:

[1] M. Hendayun, A. Ginanjar, and Y. Ihsan, "ANALYSIS OF APPLICATION PERFORMANCE TESTING USING LOAD TESTING AND STRESS TESTING METHODS IN API SERVICE," *JURNAL SISFOTEK GLOBAL*, vol. 13, no. 1, p. 28, Mar. 2023, doi: 10.38101/sisfotek.v13i1.2656.

[2] V. Hosal, H. Angriani, A. Muawwal, P. Studi, S. Informasi, and S. Kharisma Makassar, "IMPLEMENTASI SOFTWARE TESTING DALAM QUALITY ASSURANCE PADA LEARNING MANAGEMENT SYSTEM WEBSITE CLASSES," 2021. [Online]. Available: https://tech.kharisma.ac.id

[3] F. Risqulla, "Evaluating the Performance of RESTful APIs Under Large HTTP Requests with K6," 2024, [Online]. Available: https://api.dietary.cloud/food/predict=.

[4] F. Waheed, F. Azam, M. W. Anwar, and Y. Rasheed, "Model Driven Approach for Automatic Script Generation in Stress Testing of Web Applications," *ACM International Conference Proceeding Series*, pp. 46–50, 2020, doi: 10.1145/3397125.3397137.

[5] D. Manishkumar Dave Amit, "Performance Testing: Methodology for Determining Scalability of Web Systems," *International Journal of Science and Research (IJSR)*, vol. 13, no. 1, pp. 1254–1261, 2024, doi: 10.21275/sr24121010827.

[6] L. Trinh, Q. Luu, T. M. Nguyen, and H. L. Vu, "A novel framework for adaptive stress testing of autonomous vehicles in multi-lane roads," vol. XX, no. X, pp. 1–12.

[7] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, *An autonomous performance testing framework using self-adaptive fuzzy reinforcement learning*, vol. 30, no. 1. Springer US, 2022. doi: 10.1007/s11219-020-09532-z.

[8] T. M. Rezende, A. Enrique, M. Medri, J. T. Santos, M. A. Alves, and F. D. C. Zottino, "Benchmark review and case study of stress test for a gaming computer applied in CPU".

[9] I. Hamidah, I. Haromain, and I. M. Drehem, "Evaluasi Pengujian Kinerja Menggunakan JMeter untuk Menunjang Stabilitas Aplikasi Layanan Perbankan pada PT Bank Rakyat Indonesia Tbk," Journal of Digital Business and Technology Innovation (DBESTI), vol. 2, no. 1, pp. 114–126, May 2025. [Online]. Available: https://journal.nurulfikri.ac.id/index.php/DBESTI

[10] O. Huerta-Guevara, V. Ayala-Rivera, L. Murphy, and A. O. Portillo-Dominguez, "Towards an efficient performance testing through dynamic workload adaptation," in Proc. 31st IFIP Int. Conf. on Testing Software and Systems (ICTSS), Paris, France, Oct. 2019, pp. 215–233, doi: 10.1007/978-3-030-31280-0_13.

[11] H. Dewan, Y. Gupta, and A. Leekha, "Real-time monitoring using AJAX and WebSockets," Journal of Statistics and Management Systems, vol. 23, no. 1, pp. 1–10, Jan. 2020, doi: 10.1080/09720510.2020.1714154.

[12] N. L. A. S. Ginasari, K. S. Wibawa, dan N. K. A. Wirdiani, "Pengujian Stress Testing API Sistem Pelayanan dengan Apache JMeter," JITTER - Jurnal Ilmiah Teknologi dan Komputer, vol. 2, no. 3, pp. 1–10, Desember 2021.

[13] J. Li, B. Moeini, S. Nejati, M. Sabetzadeh, and M. McCallen, "A Lean Simulation Framework for Stress Testing IoT Cloud Systems," IEEE Transactions on Software Engineering, vol. PP, no. 99, pp. 1–24, Jun. 2024. [Online]. Available: https://arxiv.org/abs/2404.11542

[14] D. T. Kurniasari and S. Rochimah, "An evaluation model of website testing framework based on ISO 25010 performance efficiency," Indonesian Journal of Electrical Engineering and Computer Science, vol. 37, no. 2, pp. 1130–1139, Feb. 2025, doi: 10.11591/ijeecs.v37.i2.pp1130-1139.

[15] Indrianto, "Performance testing on web information system using Apache JMeter and BlazeMeter," Jurnal Ilmiah Ilmu Terapan Universitas Jambi, vol. 7, no. 2, pp. 138–149, Dec. 2023, doi: 10.22437/jiituj.v7i2.28440.