Comparative Load Testing of Laravel Academic Information System using JMeter and Locust

[1]Ahmad Reynaldi*, [2]Heny Pratiwi, [3]Muhammad Fahmi
[1]Program Studi Teknik Informatika, STMIK Widya Cipta Dharma, Samarinda, Indonesia
[2,3]Program Studi Sistem Informasi, STMIK Widya Cipta Dharma, Samarinda, Indonesia
[1,2,3]Jl. M. Yamin No.25, Gn. Kelua, Kec. Samarinda Ulu, Kota Samarinda, Kalimantan Timur 75123
*e-mail: 2343030@wicida.ac.id

(***received*****:** 30 August 2026, ***revised*****:** 6 October 2026, ***accepted*****:** 11 October 2026)

Abstract

Academic information systems require performance evaluation to determine how responsiveness changes under concurrent access. This study compares measurements obtained using Apache JMeter and Locust for a Laravel-based academic information system testbed under equivalent authenticated workloads. Experiments involved 10, 50, and 100 virtual users, with three replications per tool and load level, yielding 18 runs. Each virtual user authenticated once using a randomly selected student account and subsequently accessed Dashboard, KRS, Nilai, and KHS services. The workload used an approximately 10-second ramp-up, a 300-second duration, and a 1–5-second think time. Response time, percentile latency, throughput, error rate, and between-run variability were analyzed. At 100 virtual users, mean response time reached 1,205.33 ms for JMeter and 1,745.74 ms for Locust, while mean P95 reached 1,343.33 ms and 2,833.33 ms, respectively. Between 50 and 100 virtual users, throughput increased from 15.507 to 23.445 requests/s for JMeter and from 15.444 to 19.934 requests/s for Locust, indicating diminishing gains relative to the increase in concurrency. All runs recorded a 0% error rate, although responsiveness deteriorated substantially at the highest tested load. Locust exhibited greater between-run variability at 100 virtual users. The study contributes a controlled comparison of authenticated academic workloads combining endpoint-level analysis, percentile latency, and repeated measurements. These findings characterize the tested environment and do not establish production capacity, identify a specific server-side bottleneck, or demonstrate universal tool superiority.

Keywords: JMeter, Laravel, load testing, Locust, REST API

1 Introduction

Academic information systems support recurring services such as user authentication, course registration, academic dashboards, grade access, and semester-result retrieval. These services are commonly accessed by multiple users concurrently, particularly during periods of registration and result publication. Consequently, functional correctness alone does not establish that an academic information system can maintain acceptable responsiveness under concurrent demand.

Performance testing provides a means to characterize system behavior under controlled workload conditions. Response time and throughput are common performance indicators, while percentile-based latency can provide additional information about slower requests that is not captured by the arithmetic mean. For RESTful APIs, the testing process can also involve dependencies between requests, application state, and authentication. Ehsan et al. identified request dependencies, state management, and authentication among the challenges that need to be considered in RESTful API testing [1].

Previous studies have applied JMeter to compare web frameworks, evaluate REST APIs, and assess performance efficiency in academic information systems [2], [3]. Locust provides programmable user behavior and has also been used to evaluate API robustness under high workloads [4]. These characteristics make both tools suitable for generating concurrent user activity in web-based systems.

Performance evaluation in academic information systems has been reported in previous studies. Dewi et al. evaluated performance and efficiency characteristics of an Academic Information System

 ISSN:2302-8149
e-ISSN:2540-9719

using performance testing [2]. In the context of STMIK Widya Cipta Dharma, Purniawan, Fahmi, and Salmon examined the information architecture of the institution's Academic Information System and discussed academic services including course registration [3]. Their work focused on information architecture rather than backend scalability. Harianto et al. similarly examined information-system quality in Samarinda using the DeLone and McLean model [4].

Several studies have also examined Laravel REST API performance. Siahaan and Wijaya compared Laravel and Express.js using Apache JMeter [5], while Yulianto et al. compared Spring Boot and Laravel through API web services [6]. Valentina et al. evaluated Laravel 11 and CodeIgniter 4 through repeated experimental runs [7]. These studies establish the relevance of performance evaluation for Laravel-based APIs but primarily emphasize framework or implementation comparisons.

A study by Sabella et al. is particularly relevant because it directly compared JMeter and Locust using stress testing and reported average response time, percentile latency, throughput, and error rate [8]. Accordingly, comparing these two tools alone does not establish the novelty of the present research. The specific question addressed here is how consistently the tools characterize an authenticated Laravel academic REST API when the logical workload is explicitly controlled and measurements are repeated. This focus is relevant because authentication introduces dependencies between requests, while differences in workload execution and measurement variability can complicate cross-tool interpretation.

The present study contributes an empirical comparison that combines one-time authentication, token reuse, randomized student-account selection, and a common distribution of protected academic-service requests. Three replications at each concurrency level support a descriptive assessment of between-run variability alongside response time, percentile latency, throughput, and error rate. The objectives are to characterize performance changes at 10, 50, and 100 virtual users; compare measurements obtained using JMeter and Locust; and examine their consistency across repeated runs. The contribution is bounded to the experimental testbed and does not seek to establish a universal ranking of the tools or the maximum capacity of a production academic information system.

2 Literature Review

Performance testing evaluates the behavior of a software system under specified workload conditions. In REST API environments, the testing process should account for request dependencies and state because an application may generate information in one request that is required by a subsequent request [1], [9]. This consideration is particularly relevant when authentication produces a token that is then used to access protected services. [10], [11]

JMeter has been used in several studies to evaluate API and web-application performance. Husufa and Prihandi examined JMeter configuration for performance testing [12]. Siahaan and Wijaya used JMeter to compare Laravel and Express.js REST API performance [5], while Yulianto et al. investigated Laravel and Spring Boot through API web services [6]. Such studies demonstrate the applicability of JMeter to controlled API workload experiments.

Locust is designed around programmable user behavior and has been used to evaluate robustness under high loads [13]. This characteristic is useful when the workload is represented as a sequence of user actions instead of a set of independent HTTP requests. The present study applies this characteristic to an authenticated academic workflow in which login is followed by protected API requests.

Academic information systems also require performance evaluation. Dewi et al. used performance testing to analyze time behavior, resource utilization, and capacity in an academic information system [2]. This confirms the relevance of performance metrics for academic applications rather than treating functional verification as the sole criterion.

Comparative performance studies have commonly focused on frameworks or architectures. Valentina et al. compared Laravel 11 and CodeIgniter 4 using repeated experimental runs [7], while Faisal and Nuryana used statistical analysis to compare several REST API implementations [14]. Such work supports the use of repeated measurements and statistical summaries in performance research.

Sabella et al. compared JMeter and Locust directly and reported multiple latency percentiles and throughput [8]. This prior work prevents the present study from claiming that a JMeter-versus-Locust

 ISSN:2302-8149
e-ISSN:2540-9719

comparison is itself novel. Instead, the present work focuses on an equivalent authenticated workload, randomized student accounts, explicit request dependency through authentication, three independent replications, and an analysis of measurement variability in addition to conventional performance metrics.

Studies focused on API and system testing also emphasize the need to evaluate performance under changing workloads and realistic usage conditions. Latah and Toker used gradually increased workloads for load and stress testing of an API and distinguished ordinary load testing from sustained high-workload stress testing [11]. Annisa et al. similarly showed that increasing virtual users can reveal a transition from stable response behavior to sharply increased response time and emerging request errors [15].

Previous studies also highlight the importance of workload realism and systematic performance testing. Di Meglio et al. emphasized workload-generation challenges involving data dependencies between user interactions [9]. Cazzola et al. presented a dedicated performance-testing framework for concurrent and distributed systems, illustrating that performance testing itself requires systematic experimental support [10].

For API security and authenticated workloads, Dwiyanto et al. evaluated a REST API using JWT authentication and tested the system with 100 concurrent users while maintaining successful data delivery [16]. Poth et al. further showed that REST API testing tools and approaches need systematic evaluation when integrated into real software-engineering workflows [17]. These studies support the use of explicit authentication and workload controls in API experiments, while not addressing the specific JMeter-Locust comparison considered here.

Comparative studies using different interfaces and architectures have reported that workload conditions influence the observed performance of REST APIs and distributed applications. Śliwa and Pańczyk compared REST, GraphQL, and gRPC using equivalent functionality and performance tests [18]. Ain et al. examined REST and gRPC under different traffic conditions and data sizes using JMeter and Wireshark [19]. Wahyudin et al. compared monolithic and microservice architectures through load, soak, and stress testing using response time, throughput, and error rate [20]. These findings reinforce the importance of maintaining comparable workload conditions when interpreting comparative results.

The reviewed studies establish complementary foundations for this experiment: framework comparisons examine implementation-dependent performance, academic information-system studies motivate application-specific evaluation, and API-testing research highlights authentication and request dependencies. The direct comparison by Sabella et al.[8] further establishes that cross-tool evaluation is an existing research approach. Building on these foundations, the present study examines their intersection through a controlled authenticated workload on a Laravel academic testbed. Its contribution lies in combining cross-tool, endpoint-level, and between-run analyses within the same experimental setting. This positioning supports a context-specific empirical contribution without claiming that previous research has not compared JMeter and Locust.

3 Research Method

The research used a controlled experimental design. The system under test was a Laravel-based Academic Information System developed as an experimental testbed. The testbed is not claimed to be the operational academic information system of STMIK Widya Cipta Dharma; the two WICIDA references are used only to establish institutional and academic-system context [3], [4].

The experiments were conducted in a local Windows 64-bit environment using a computer equipped with an 11th Gen Intel Core i3-1115G4 processor at 3.00 GHz and 12 GB of RAM, of which 11.8 GB was usable. The application environment consisted of Apache 2.4.58, PHP 8.3.29, Laravel Framework 13.26.1, and MariaDB 10.4.32. Apache JMeter 5.6.3 and Locust 2.46.3 were used to generate the test workloads. Table 1 summarizes the experimental environment.

The Laravel application, MariaDB database, and load-testing tool were hosted on the same computer. JMeter and Locust tests were conducted separately, with only one load-testing tool running at a time. Consequently, the application, database, and active load generator shared the available computing resources during each test. This configuration should be considered when interpreting the results because load generation could compete with the application and database for CPU and

 ISSN:2302-8149
e-ISSN:2540-9719

memory resources. However, resource utilization was not measured, so the extent of any such contention cannot be determined from the collected data.

Table 1 Experimental hardware and software specifications

Component	Specification
Processor	11th Gen Intel Core i3-1115G4 at 3.00 GHz
Processor architecture	x64
Memory	12 GB RAM; 11.8 GB usable
Total storage capacity	477 GB
Operating system	Windows, 64-bit
Web server	Apache 2.4.58 (Win64)
Programming language	PHP 8.3.29
Application framework	Laravel Framework 13.26.1
Database	MariaDB 10.4.32 (Win64)
Load-testing tools	Apache JMeter 5.6.3 and Locust 2.46.3

a. System and Authentication

The testbed exposes REST API services and uses Laravel Sanctum for authentication. The database contains approximately 10,000 student accounts. Each virtual user selects a student identifier randomly, submits the credentials to /api/login, and stores the returned access token. The token is then attached to requests for Dashboard, KRS, Nilai, and KHS. Login is executed once per virtual user.

b. Tested Endpoints and Workload

Five API services were included in the workload. The authenticated read services use relative task weights of 3:2:2:1 for Dashboard, KRS, Nilai, and KHS, respectively. This produces approximate proportions of 37.5%, 25%, 25%, and 12.5% for the four protected read operations.

Table 2 API endpoints and workload distribution

Endpoint	Method	Function	Workload Weight
/api/login	POST	Authentication	1 per VU
/api/dashboard	GET	Academic dashboard	3
/api/krs	GET	Course registration	2
/api/nilai	GET	Grade information	2
/api/khs	GET	Semester results	1

c. Experimental Scenario

Apache JMeter version 5.6.3 and Locust version 2.46.3 were used. Three load levels were tested with three independent replications for each tool. The detailed experimental configuration for each load scenario is presented in Table 3. The same logical workload was applied to both tools, including random student accounts, one-time authentication, protected requests, 1–5-second think time, approximately 10-second ramp-up, and 300-second execution duration.

Table 3 Experimental configuration

Load Level	Virtual Users	Ramp-Up	Duration	Think Time	Replications
Low	10	10 s	300 s	1–5 s	3
Medium	50	10 s	300 s	1–5 s	3
High	100	10 s	300 s	1–5 s	3

Workload equivalence in this study refers to the configured logical behavior: account selection, one-time authentication, token reuse, protected-service weights, think-time range, ramp-up, and execution duration. It does not imply identical request arrival times or identical numbers of completed requests between tools. Consequently, the configured virtual-user count should be interpreted together with the observed throughput when comparing results. Differences between tools are assessed as outcomes of the complete testing configuration rather than isolated measurements of application performance.

 ISSN:2302-8149
e-ISSN:2540-9719

The experiments provide a controlled comparison within the specified concurrency range. They do not reproduce the full range of traffic patterns, deployment conditions, or infrastructure characteristics of an operational academic information system.

d. Performance Metrics and Analysis

Average response time was calculated using the arithmetic mean formula as shown in Equation (1). The variability among experimental replications was measured using sample standard deviation as expressed in Equation (2). To compare variability relative to the mean value, coefficient of variation (CV) was calculated using Equation (3). The percentage change between two load levels was calculated using Equation (4).

The mean response time across replications was calculated using:

$$\bar{x} = (\Sigma x_i) / n \quad (1)$$

The sample standard deviation was calculated using:

$$s = \sqrt{[\Sigma(x_i - \bar{x})^2 / (n - 1)]} \quad (2)$$

Coefficient of variation was calculated as:

$$CV = (s / \bar{x}) \times 100\% \quad (3)$$

To quantify scalability degradation between two load levels, the percentage change was calculated as:

$$\Delta = [(x_new - x_old) / x_old] \times 100\% \quad (4)$$

For Locust, throughput values used in the analysis were taken from the Aggregated Requests/s field in the CSV results. For JMeter, the TOTAL throughput from the Aggregate Report CSV was used. Percentile values in the summary tables represent the arithmetic mean of the percentile values obtained from the three replications, rather than a pooled percentile over all individual requests.

The resulting design contains 18 main experiments, calculated as 2 tools × 3 load levels × 3 replications. The primary comparisons were made at the tool-by-load level, followed by endpoint-level and repeatability analysis.

The analysis is descriptive. Standard deviation and coefficient of variation summarize the dispersion of the three run-level mean response times for each tool and load condition; they do not describe the variation of individual request latencies within a run. The reported P95 and P99 summaries are arithmetic means of the corresponding run-level percentiles, rather than percentiles calculated from pooled request records. With three replications per condition, these statistics provide an initial indication of repeatability and should not be interpreted as establishing statistically significant differences between tools.

4. Results and Discussion

The 18 final experimental runs produced zero request failures. Table 4 summarizes the performance results for the six tool-by-load conditions. Average response time increased with concurrency for both tools. The increase was especially pronounced at 100 virtual users.

Table 4 Overall performance across load levels

Tool	VU	Mean Response Time ± SD (ms)	CV (%)	Mean P95 (ms)	Mean P99 (ms)	Throughput (req/s)	Error
JMeter	10	83.33 ± 1.15	1.39	122.33	359.67	3.271	0.00%
JMeter	50	202.00 ± 3.46	1.71	222.33	5090.00	15.507	0.00%
JMeter	100	1205.33 ± 27.21	2.26	1343.33	17256.00	23.445	0.00%
Locust	10	68.51 ± 7.76	11.32	98.67	320.00	3.309	0.00%
Locust	50	233.50 ± 3.57	1.53	233.33	6033.33	15.444	0.00%
Locust	100	1745.74 ± 476.62	27.30	2833.33	20666.67	19.934	0.00%

 ISSN:2302-8149
e-ISSN:2540-9719

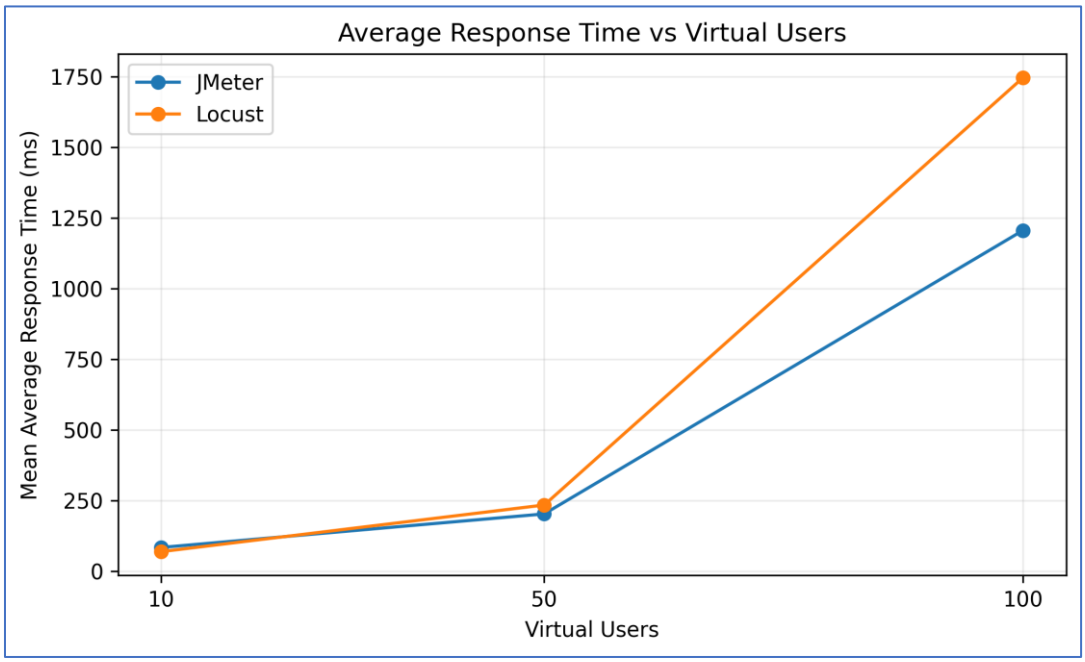


Figure 1 Mean Response Time versus Virtual Users

Figure 1 shows that both tools followed the same upward response-time trend. JMeter increased from 83.33 ms at 10 VU to 202.00 ms at 50 VU and 1,205.33 ms at 100 VU. Locust increased from 68.51 ms to 233.50 ms and 1,745.74 ms, respectively. The common direction of change indicates agreement in the overall scalability pattern, while the magnitude differs at high load.

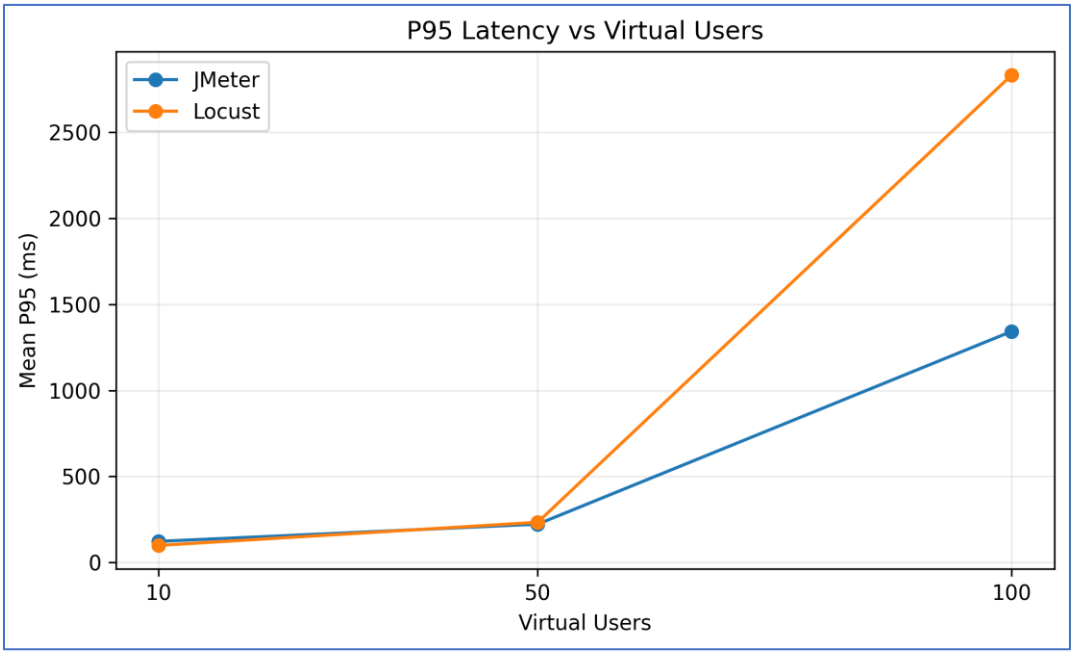


Figure 2 Mean P95 latency versus virtual users

The percentile latency behavior is shown in Figure 2, where the increase in P95 latency becomes more apparent at 100 virtual users. Mean P95 increased substantially between 50 and 100 virtual users, reaching 1,343.33 ms for JMeter and 2,833.33 ms for Locust at the highest tested load. These results indicate that slower requests experienced substantial delays alongside the increase in average response time. However, the relative change in P95 should be interpreted separately for each tool and alongside its average response time. The reported values represent averages of run-level percentiles and therefore do not describe a single pooled response-time distribution.

 ISSN:2302-8149
e-ISSN:2540-9719

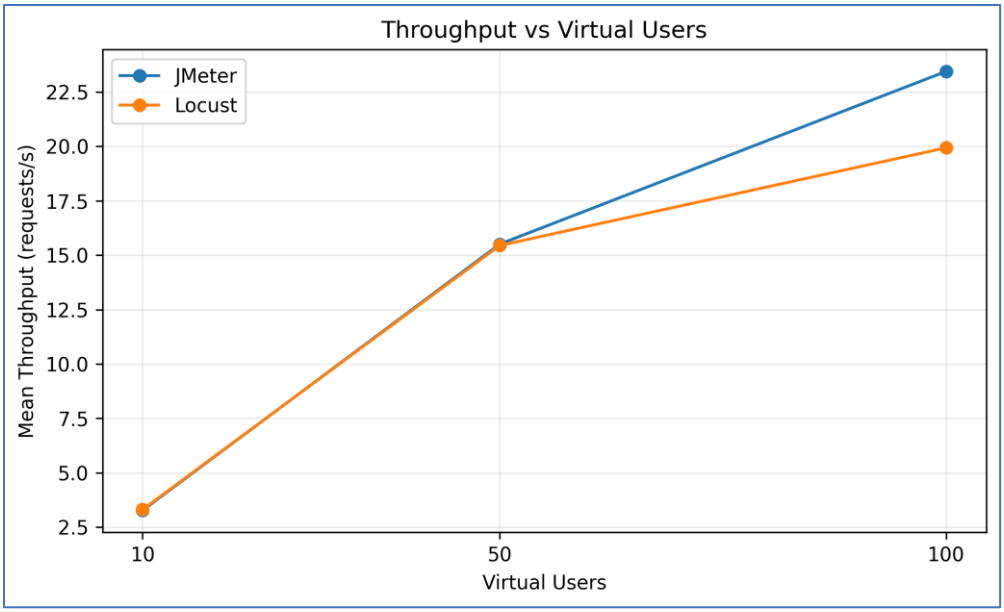


Figure 3 Mean throughput versus virtual users

The throughput trend for both tools is illustrated in Figure 3. The figure indicates that throughput continued to increase, although the improvement became smaller at higher concurrency. Throughput increased when load increased, but the gain diminished at 100 VU. From 50 to 100 VU, JMeter throughput increased from 15.507 to 23.445 requests/s, a 51.19% increase, whereas Locust increased from 15.444 to 19.934 requests/s, a 29.07% increase. Because virtual users doubled from 50 to 100 while throughput increased by substantially less than 100%, the results indicate a high-load region with diminishing throughput gains.

Average response time reinforces this interpretation. From 50 to 100 VU, JMeter response time increased by 496.70% and Locust by 647.65%, even though throughput increased only 51.19% and 29.07%, respectively. The combination of sharply increasing latency and limited throughput gain is consistent with high-load degradation, although the experiment does not establish the absolute saturation point.

a. Cross-Tool Comparison

At 10 VU, Locust recorded a lower mean response time than JMeter (68.51 versus 83.33 ms), while throughput was nearly identical (3.309 versus 3.271 requests/s). At 50 VU, the mean response time was lower for JMeter (202.00 versus 233.50 ms), while throughput again remained very close (15.507 versus 15.444 requests/s). At 100 VU, the divergence was larger: JMeter recorded 1,205.33 ms compared with 1,745.74 ms for Locust, while throughput was 23.445 and 19.934 requests/s, respectively.

These results should not be interpreted as proof that one load-testing tool is universally faster. The measured values represent the interaction between the load generator, the system under test, and the experimental environment. The more defensible finding is that both tools identified the same performance trend but differed in measurement magnitude at high load.

b. Endpoint-Level Analysis

Further analysis was conducted at the endpoint level to examine the response time behavior of each API service. The mean response time of each endpoint across the three experimental replications is presented in Table 5.

Table 5 Mean endpoint response time across three replications (ms)

Tool	VU	Login	Dashboard	KRS	Nilai	KHS
JMeter	10	422.00	78.67	80.33	79.67	79.67
JMeter	50	4233.67	161.67	140.00	164.67	172.33
JMeter	100	12965.00	1037.00	1047.00	1054.33	976.67
Locust	10	366.80	64.64	67.45	64.78	65.38
Locust	50	4812.13	172.82	189.11	181.51	209.57
Locust	100	13243.10	1569.50	1410.56	1588.11	1662.50

 ISSN:2302-8149
e-ISSN:2540-9719

Login had the highest mean response time among the measured endpoints for both tools. At 100 virtual users, its mean response time reached 12,965.00 ms for JMeter and 13,243.10 ms for Locust. However, Login was executed once per virtual user, whereas protected read requests were repeated throughout the workload. These operations therefore differ in execution frequency and temporal distribution. The comparison identifies authentication as an operation requiring further investigation, but it does not establish Login as the principal cause of the overall performance degradation.

The increase in latency was also observed across the protected read endpoints, including Dashboard, KRS, Nilai, and KHS. As presented in Table 5, these endpoints experienced increased response times under higher concurrency levels. At 100 virtual users, their mean response times ranged from 976.67 to 1,054.33 ms for JMeter and from 1,410.56 to 1,662.50 ms for Locust. This indicates that performance degradation was not confined to authentication.

c. Repeatability and Variability

The coefficient of variation provides a normalized measure of run-to-run variability. JMeter remained relatively stable across the three load levels, with CV values of 1.39%, 1.71%, and 2.26% at 10, 50, and 100 VU. Locust recorded 11.32%, 1.53%, and 27.30%, respectively. The largest variability therefore occurred for Locust at 100 VU.

The three Locust 100-VU mean response-time values were 1,455.00 ms, 1,486.42 ms, and 2,295.79 ms. The third replication was substantially slower than the first two, which explains the higher CV. The run was retained in the analysis because it satisfied the predefined workload conditions and recorded zero request failures. Retaining the observation avoids selectively removing an unfavorable result.

The lower coefficient of variation observed for JMeter at 10 and 100 virtual users indicates more consistent run-level mean response times within those experimental conditions. At 50 virtual users, Locust recorded a slightly lower CV than JMeter, at 1.53% and 1.71%, respectively. Consequently, neither tool exhibited lower variability at every load level. Repeatability also does not establish measurement accuracy because no independent reference measurement was available. The higher variability observed for Locust at 100 virtual users cannot be attributed to the tool alone; shared-host execution and unmeasured resource behavior remain possible influences that the present design cannot separate.

d. Reliability and Performance

All 18 experiments recorded an error rate of 0%. This indicates that the configured tests reported no request failures under the specified conditions; it does not demonstrate that response times would satisfy operational requirements. At 100 virtual users, mean P99 reached 17,256.00 ms for JMeter and 20,666.67 ms for Locust. Successful request completion can therefore coexist with substantial latency. Performance evaluation should consider responsiveness alongside the reported error rate.

e. Discussion of Findings and Limitations

The principal finding is a substantial deterioration in responsiveness between 50 and 100 virtual users, accompanied by continued but diminishing throughput gains. JMeter throughput increased from 15.507 to 23.445 requests/s, while Locust throughput increased from 15.444 to 19.934 requests/s. Over the same interval, mean response time increased from 202.00 to 1,205.33 ms for JMeter and from 233.50 to 1,745.74 ms for Locust. Thus, doubling the configured concurrency did not produce a proportional increase in completed requests per second. The observed pattern identifies a region of reduced performance efficiency within the tested range, without establishing an exact saturation threshold.

Both tools captured this overall pattern despite differences in measurement magnitude. This agreement supports the conclusion that performance deterioration was observable under both testing configurations. However, matching the logical workload does not guarantee identical request timing or resource demand from the load generators. Because the application, database, and active tool shared one computer, the high-load differences should be interpreted in relation to the application,

 ISSN:2302-8149
e-ISSN:2540-9719

workload execution, and local environment together. The experiment does not isolate the contribution of each factor.

In relation to the earlier JMeter–Locust comparison discussed in the literature review [8], the present results add an explicitly authenticated academic-service workload and a descriptive assessment of repeated measurements. The comparison should remain methodological rather than numerical: differences in application and experimental conditions prevent direct equivalence between performance values across studies. The contribution is therefore evidence about behavior and measurement variability within this controlled testbed.

The endpoint results help narrow the scope of further investigation. Login exhibited the largest mean delay, while all protected read services also slowed at 100 virtual users. These findings justify examining both authentication and shared processing resources in subsequent experiments. CPU utilization, memory pressure, database connection availability, and query execution time are relevant diagnostic measurements, but none was collected as an outcome in this study. Accordingly, these components remain candidates for investigation rather than demonstrated causes of the observed delays.

Several limitations restrict interpretation. First, the local, shared-host environment does not reproduce production deployment conditions and may permit resource contention between the application, database, and load generator. Second, the absence of server-side profiling prevents causal attribution to CPU, memory, database, or application-query behavior. Third, three replications per condition provide limited evidence about repeatability, particularly where one run differs substantially from the others. Fourth, the tested range ends at 100 virtual users, so neither maximum capacity nor behavior beyond this range can be inferred. Fifth, JMeter was executed in GUI mode during the recorded runs, introducing a potential source of load-generator overhead. Finally, the controlled workload was not derived from measured production traffic.

Future research should extend concurrency beyond the tested range, use deployment conditions closer to production, and derive workload distributions from anonymized operational traffic. Separating load generation from application hosting and collecting synchronized CPU, memory, database, and query-level measurements would help distinguish client-side limitations from server-side constraints. Additional replications would provide a stronger basis for assessing variability. These extensions are proposed future investigations and are not part of the experimental evidence reported here.

5. Conclusion

This study compared JMeter and Locust using equivalent authenticated workloads on a Laravel-based academic information system testbed. Across 10, 50, and 100 virtual users, both tools identified a substantial increase in response time, with the strongest deterioration occurring between 50 and 100 virtual users. At 100 virtual users, mean response time reached 1,205.33 ms for JMeter and 1,745.74 ms for Locust, while mean P95 reached 1,343.33 ms and 2,833.33 ms, respectively. Throughput continued to increase, although its gains were limited relative to the increase in concurrency. The tools agreed on the overall performance trend but differed in measurement magnitude and between-run variability. JMeter recorded lower CV values at 10 and 100 virtual users, whereas Locust recorded a slightly lower CV at 50 virtual users. Locust exhibited its highest variability at 100 virtual users. All experiments recorded a 0% error rate, demonstrating that the absence of reported request failures did not prevent substantial deterioration in responsiveness. Endpoint analysis identified high Login latency and broader delays across protected read services, without establishing a specific internal bottleneck. The study contributes a controlled, repeated comparison that integrates authenticated academic workflows, endpoint behavior, percentile latency, and measurement variability. Its conclusions are restricted to the local, shared-host testbed, three replications per condition, and the tested concurrency range. Production-oriented testing at higher loads, supported by operational traffic patterns and synchronized infrastructure profiling, is required to investigate capacity limits and explain the causes of performance degradation.

References

[1] A. Ehsan, M. A. M. E. Abuhaliqa, C. Catal, and D. Mishra, "*RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions*," *Appl. SCI.*, Vol. 12, No. 9, p.

 ISSN:2302-8149
e-ISSN:2540-9719

4369, 2022, DOI: 10.3390/app12094369.

[2] U. K. Dewi, R. L. Wicaksono, M. D. Purbolaksono, and V. Satria, "*Performance and Efficiency Testing Analysis of Database Systems in Academic Information Systems*," *Nuansa Inform.*, Vol. 19, No. 2, pp. 115–122, 2025, DOI: 10.25134/ilkom.v19i2.438.

[3] T. Purniawan, M. Fahmi, and Salmon, "Penerapan Information *Architecture* untuk Optimalisasi *Website* Sistem Informasi Akademik (SIAK) STMIK Widya Cipta Dharma," *J. Nas. Komputasi dan Teknol. Inf.*, Vol. 8, No. 3, pp. 1174–1184, 2025, DOI: 10.32672/jnkti.v8i3.9005.

[4] K. Harianto, Azahari, and F. A. Azhar, "*Evaluation Information System Success of the PPDB Information System in SMP Negeri 16 Samarinda using DeLone and McLean's Model,*" *TEPIAN*, Vol. 4, No. 1, pp. 18–23, 2023, DOI: 10.51967/tepian.v4i1.2217.

[5] M. Siahaan and R. Wijaya, "*Performance Comparison between Laravel and ExpressJs Framework using Apache JMeter*," *J. Informatics Telecommun. Eng.*, Vol. 7, No. 2, pp. 545–554, 2024, DOI: 10.31289/jite.v7i2.10571.

[6] R. Yulianto, Mardiana, R. A. Pradipta, and G. F. Nama, "*Performance Comparison Analysis of SpringBoot and Laravel Frameworks using API Web Service*," *J. Inform. dan Tek. Elektro Terap.*, Vol. 12, No. 2, pp. 1145–1153, 2024, DOI: 10.23960/jitet.v12i2.4141.

[7] P. E. Valentina, D. Handayani, S. R. M. Ibrahim, N. Nanang, and F. M. K. Putra, "Komparasi *Performa REST API Laravel 11* dan *CodeIgniter 4* menggunakan Metode Eksperimental," *J. Fasilkom*, Vol. 16, No. 1, pp. 167–174, 2026, DOI: 10.37859/jf.v16i1.11274.

[8] B. Sabella, R. R. Tina, F. Achmad, A. Lestari, and Fathurrahmani, "Pengujian Aplikasi SIHARAPAN menggunakan Metode *Stress Testing*," *J. Teknol. Inf. dan Terap.*, Vol. 11, No. 1, pp. 7–14, 2024, DOI: 10.25047/jtit.v11i1.359.

[9] S. Di Meglio, L. L. L. Starace, and S. Di Martino, "*Web App Performance Testing in Industrial Contexts: Supporting Workload Generation with E2E-Loader++*," *J. Syst. Softw.*, Vol. 232, p. 112684, 2026, DOI: 10.1016/j.jss.2025.112684.

[10] W. Cazzola, F. Cesarini, and L. Tansini, "*PerformERL: A Performance Testing Framework for Erlang*," *Distrib. Comput.*, Vol. 35, No. 5, pp. 439–454, 2022, DOI: 10.1007/s00446-022-00429-7.

[11] A. Latah and L. Toker, "*Load and Stress Testing for SDN's Northbound API*," *SN Appl. SCI.*, Vol. 2, 2020, DOI: 10.1007/s42452-019-1917-y.

[12] N. Husufa and I. Prihandi, "*Optimizing JMeter on Performance Testing using the Bulk Data Method*," *J. Inf. Syst. Informatics*, Vol. 4, No. 2, pp. 205–215, 2022, DOI: 10.51519/journalisi.v4i2.244.

[13] R. G. Zatserkovnyi, V. I. Babych, M. I. Plesha, L. I. Khmilyarchuk, and O. M. Shvets, "*A System to Evaluate the Robustness of an API under High Loads*," *Syst. Technol.*, Vol. 66, No. 2, pp. 43–49, 2023, DOI: 10.32782/2521-6643-2023.2-66.5.

[14] M. Faisal and I. K. D. Nuryana, "*Comparative Analysis of Laravel, Lumen, Guzzle, Leaf, and Slim Framework Performance on REST API using One Way ANOVA*," *J. Emerg. Inf. Syst. Bus. Intell.*, Vol. 6, No. 2, pp. 77–83, 2025, DOI: 10.26740/jeisbi.v6i2.66146.

[15] A. R. Annisa, M. Ardiana, P. N. Rahayu, T. Brian, and M. A. Jami'in, "*Performance Evaluation of Web Applications using JMeter Load Testing for Server Capacity and Response Efficiency*," *J. Appl. Informatics Comput.*, Vol. 10, No. 1, pp. 585–590, 2026, DOI: 10.30871/jaic.v10i1.11840.

[16] R. A. Dwiyanto, G. A. Mutiara, and M. I. Sari, "*Performance Analysis of REST API in a Real-Time IoT-based Vehicle Monitoring System*," *Int. J. Reconfigurable Embed. Syst.*, Vol. 14, No. 3, pp. 766–784, 2025, DOI: 10.11591/ijres.v14.i3.pp766-784.

[17] A. Poth, O. Rrjolli, and A. Arcuri, "*Technology Adoption Performance Evaluation Applied to Testing Industrial REST APIs*," *Autom. Softw. Eng.*, Vol. 32, p. 5, 2025, DOI: 10.1007/s10515-024-00477-2.

[18] M. Śliwa and B. Pańczyk, "*Performance Comparison of Programming Interfaces on the Example of REST API, GraphQL and gRPC,*" *J. Comput. SCI. Inst.*, Vol. 21, pp. 356–361, 2021, DOI: 10.35784/jcsi.2744.

[19] M. Z. Ain, R. Ardiansyah, S. A. Pratama, M. Akbar, and N. T. Lapatta, "*Comparative Performance Analysis of gRPC and REST API under Various Traffic Conditions and Data*

 ISSN:2302-8149
e-ISSN:2540-9719

Sizes using a Quantitative Approach," *J. Appl. Informatics Comput.*, Vol. 9, No. 2, pp. 450–457, 2025, DOI: 10.30871/jaic.v9i2.9276.

[20] A. Wahyudin, H. Siregar, A. Anisyah, S. M. Kusumawardani, and Erlangga, "*Migrating Monolithic to Microservices: Comparative Performance Testing Evaluation between Architectures*," *SCI. J. Informatics*, Vol. 12, No. 4, pp. 797–816, 2026, DOI: 10.15294/sji.v12i4.28499.