

Comparative analysis of k6, JMeter and Postman test tools for performance API CRUD testing

Sergii Khlamov^{1,*†}, Mariia Mendieliieva^{1,†}, Oleksandr Vovk^{1,†} and Tetiana Trunova^{1,†}

¹ Kharkiv National University of Radio Electronics, Nauky Ave. 14, 61166 Kharkiv, Ukraine

Abstract

Load testing tools play an important role in evaluating the performance of modern REST API systems. This paper presents an experimental comparison of k6, JMeter and Postman for API performance testing using CLI mode. Experiments were performed on a locally deployed CRUD Node.js application using Ramp, Spike, Peak and Fixed workload profiles. Average response time, p95 response time, CPU and RAM consumption were evaluated using a two-way ANOVA model. The results demonstrate that both load testing tool and workload profile significantly affect performance characteristics. The results indicate that performance and resource efficiency are not always aligned, especially for complex read operation. JMeter achieves the best response time, k6 provides higher resource efficiency with increased memory usage, and Postman can be suitable for lightweight API performance and stability testing. The study provides practical insights for selecting load testing tools for local performance analysis and CI/CD integration.

Keywords

Performance testing, load testing, Postman, Jmeter, response time, workload, virtual users, delay

1. Introduction

Modern software systems increasingly rely on APIs (Application Programming Interfaces) as the primary mechanism for communication between distributed components [1]. The widespread adoption of microservice architectures, cloud-based platforms [2], and DevOps (Development and Operations) practices has significantly increased the number and complexity of API interactions within modern applications. As APIs often represent critical communication layers between services, ensuring their reliability, scalability, and performance under different workload conditions has become an essential aspect of software engineering [3]. API performance testing is essential for evaluating response time, throughput, and resource consumption under different workloads [4]. However, the increasing complexity of distributed systems makes performance evaluation more challenging, as results depend not only on the application architecture and workload but also on the selected testing tool and its execution mechanisms [5, 6]. This challenge is particularly relevant for on-premise environments used in domains with sensitive data, such as healthcare, banking, and astronomy [7] or those that run their application behind a firewall. Numerous API load testing tools differ in workload generation, scripting, resource consumption, reporting, and CI/CD integration, making objective tool selection essential [8]. Although previous studies compared performance testing tools, most considered different tool combinations or application scenarios. A systematic comparison of k6, Apache JMeter, and Postman under a unified experimental setup remains limited.

ADT-ISMDDC 2026: Advanced Digital Technologies: Intelligent Systems, Modeling, and Data-Driven Control, September 22–24, 2026, Odesa, Ukraine

* Corresponding author.

† These authors contributed equally.

✉ sergii.khlamov@gmail.com (S. Khlamov); mariia.mendieliieva@nure.ua (M. Mendieliieva); oleksandr.vovk@nure.ua (O. Vovk); tetiana.trunova@nure.ua (T. Trunova)

ORCID 0000-0001-9434-1081 (S. Khlamov); 0009-0002-4282-3147 (M. Mendieliieva); 0000-0001-9072-1634 (O. Vovk); 0000-0003-2689-2679 (T. Trunova)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

This study addresses this gap by evaluating the three tools using identical REST API workloads and load profiles.

The study aims to compare the performance of k6, Apache JMeter, and Postman for REST API load testing by addressing the following research questions:

RQ1. Does the performance of testing tools differ?

RQ2. Does the load profile affect tool performance?

RQ3. Which HTTP (Hypertext Transfer Protocol) endpoints are most sensitive to the choice of load testing tool?

The main contributions of this study include the unified experimental scenario for comparing k6, Apache JMeter, and Postman, evaluation under multiple load profiles, and statistical analysis of the effects of tool selection and workload on API performance metrics.

2. Literature review

A previous study compared JMeter and Postman for load testing of five public API services [9]. The evaluation was limited to average response time under four load profiles. It considered only public APIs, without assessing CPU and RAM usage or statistically validating the observed differences. Traditional web application architectures face scalability challenges, making performance testing essential for evaluating system behavior under different workloads [10, 11]. Because of its time-consuming nature, performance testing is commonly integrated into DevOps CI/CD pipelines [12]. Grafana k6 is an open-source load testing tool written in Go that supports JavaScript scripting [13] and efficient performance evaluation [14] under different load conditions [15, 16]. Apache JMeter is an open-source tool supporting multiple protocols and workload scenarios [17, 18] but is limited by high memory consumption [19], restricted real-time monitoring, and complex distributed configuration [20, 21]. Postman [22] supports API performance testing [23] using predefined load profiles and reusable API collections but is constrained by resource-dependent scalability, support for only one data file per test scenario, limited collaboration in the free plan, and the absence of think time between requests [24]. Study [13] evaluated the integration of Apache JMeter, Grafana k6, Gatling, Locust, and Artillery into a Jenkins-based CI/CD pipeline. Based on response time and CPU/memory utilization under multiple load profiles, Grafana k6 demonstrated the highest resource efficiency. A comparative study of JMeter, k6, Gatling, and Locust showed that tool performance depends on workload characteristics: k6 performed best for write-intensive workloads, JMeter for read-intensive workloads, Gatling provided efficient memory utilization, and Locust was suitable for resource-constrained environments [25]. The results [26] showed that Postman outperformed JMeter and Robot Framework in overall performance, while JMeter exhibited the longest execution time in comparisons with Karate DSL, REST Assured, k6, and Postman/Newman [8]. Unlike these studies, the present work compares k6, JMeter, and Postman using the same API scenario, multiple load profiles, and statistical analysis of the results.

3. Methodology

This section describes the experimental methodology used to compare the performance of three load testing tools under different workload patterns. First, the experimental environment is presented, followed by the evaluated tools, workload design, and the metrics and statistical methods used to analyze the collected results.

3.1. Experimental environment

The experimental environment was deployed on a local physical server to minimize the influence of network latency and external infrastructure variability. All experiments were executed under identical hardware and software conditions to ensure reproducibility and fair comparison between the evaluated load testing tools. Experiments were conducted on a workstation equipped with Windows 11 x64 25H2, equipped with an Intel Core i9-11900H 8-core CPU and 16 GB RAM.

The application under test was implemented as a RESTful API using Node.js with the Express framework. Data persistence was provided by the Better-SQLite3 embedded database. Synthetic test data were generated using the Faker library to create realistic datasets while maintaining identical input conditions across experimental runs (see Figure 1a).

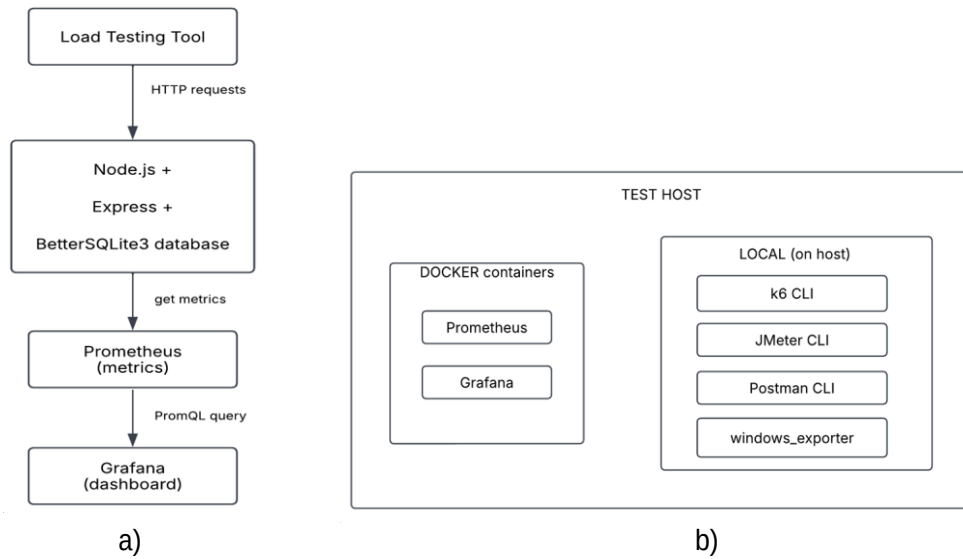


Figure 1: Experimental environment: a) architecture; b) supporting services (source: own elaboration).

The supporting services were executed in Docker containers (see Figure 1b). Resource utilization were collected using Prometheus, while Grafana was used for visualization and inspection of CPU and memory (RAM) consumption throughout each experiment. To ensure consistency, the same application version, database state, workload configuration, and monitoring setup were used for every experimental run. Each Tool × Load combination was executed 10 times under identical conditions: Node.js – v24.16.0; Express – v5.2.1; Better-SQLite3 – v12.11.1; Faker – v5.5.3; Docker – v4.81.0; Prometheus – v3.12.0; Grafana – v13.0.2. The application uses a relational database consisting of four entities: Users, Products, Orders, and Order_Items, as shown in entity-relationship (ER) diagram in Figure 2.

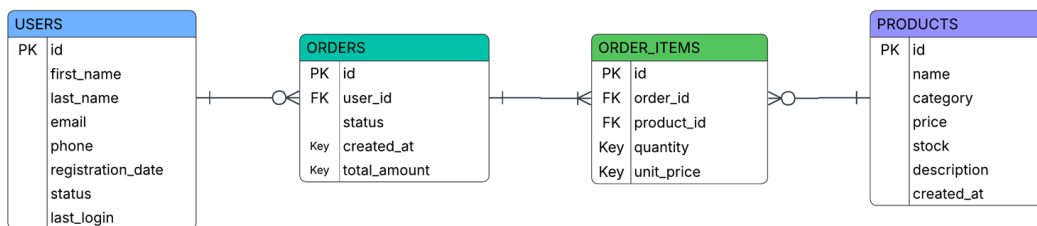


Figure 2: ER diagram of the experimental database schema (source: own elaboration).

Each user could create multiple orders, and each order contained one or more order items linked to products, representing a simplified e-commerce workflow for CRUD performance evaluation. A Better-SQLite3 database was populated with 1000 users, 3000 products, 10000 orders, and 30000 order items. Before each experiment, the database was restored from a predefined dump and the local application server was restarted to ensure identical initial conditions.

3.2. Performance test tools

Three load testing tools were evaluated: k6, Apache JMeter, and Postman. They were selected as representative solutions with different workload generation approaches. To ensure a fair comparison, equivalent test scenarios and workload configurations were implemented under identical experimental conditions using CLI execution. k6 uses JavaScript-based virtual user scenarios, JMeter relies on XML test plans and thread-based load generation, and Postman executes JavaScript collections through CLI. Despite differences in workload mechanisms, equivalent load profiles were configured across tools while keeping the number of virtual users, load level, test duration, and request sequence constant. Experiments were conducted on a local workstation with k6 v2.0.0-rc1, Apache Jmeter 5.6.3, Postman 12.19.3 installed.

3.3. Workload design

Four workload profiles were designed to represent common performance testing scenarios: Ramp, Spike, Peak, and Fixed. Each workload profile was executed independently using all three load testing tools. For every Tool $\times$ Load combination, 10 independent experimental runs were performed to reduce the influence of random variation. Each load profile defines a different pattern of virtual user generation over time. The workload patterns used in this study are illustrated in Figure 3.



Figure 3: Load type visualization: a) Ramp; b) Spike; c) Peak; d) Fixed (source: own elaboration).

To ensure comparability between the evaluated load testing tools, an identical logical test scenario was implemented in k6, Apache JMeter, and Postman. The scenario used the same API endpoint sequence, predefined test conditions, and iteration data selection strategy, as shown in Figure 4. Test data (identifiers) and initial conditions were established prior to test execution. During each iteration, the identifiers (userId, productId, and orderId) were deterministically selected from predefined datasets according to the iteration index. To avoid repeatedly accessing the same database records [27], each identifier was cyclically chosen from a predefined set of five values, ensuring that consecutive requests targeted different records and providing a more balanced distribution of database accesses. Each virtual user executed the same sequence of REST API operations during every iteration as shown in Table 1, while the implementation details were adapted according to the execution model of each tool. After completing the sequence, the virtual user immediately started the next iteration until the end of the test.

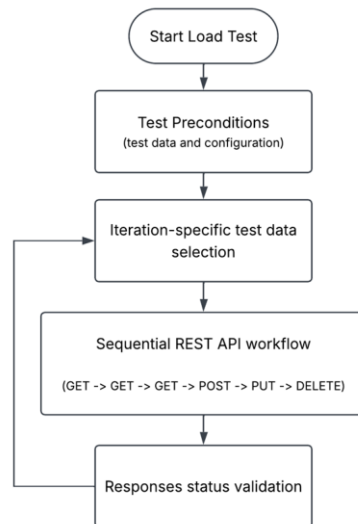


Figure 4: Generic workflow of the load-testing scenario implemented in k6, Apache JMeter, and Postman (source: own elaboration).

Table 1
HTTP Methods used in test scenario

HTTP Method	Endpoint	Description
GET	/products	Retrieve available products
GET	/orders	Retrieve user orders
GET	/orders/{id}/details	Retrieve order details
POST	/orders	Create a new order
PUT	/orders/{id}	Update an existing order
DELETE	/orders/{id}	Delete an order

The workload included CRUD operations, multi-table database joins, and business logic involving data aggregation to represent realistic application behavior. A think time was applied after each request to avoid unrealistically bursty traffic. Equivalent delay mechanisms were configured in each tool: Constant Timer in JMeter, Delay endpoint (GET <https://postman-echo.com/delay/X>, where X is the number of seconds in a pause) in Postman, and the `sleep()` function in k6. To ensure a fair comparison across all evaluated tools, the workload profiles were designed according to the load generation strategies supported by all platforms. In particular, the four workload profiles (Ramp, Spike, Peak, and Fixed) were adopted because they are natively supported by Postman and can be equivalently implemented in k6 and JMeter, as shown in Table 2.

Table 2
Performance Test Scenarios

TC #	Load	Configured VUs	Think Time (sec)	Test Duration (min)
1	Ramp	125 - 500	3	10
2	Spike	80 – 800 - 80	3	10
3	Peak	300 – 1500 - 300	3	10
4	Fixed	500	3	15

This enabled identical testing scenarios to be executed across all tools:

- Ramp load – 125 virtual users run for 2:30 minutes, ramp up to 500 for 2:30 minutes, then maintain 500 for 5 minutes, each executing all requests sequentially.
- Spike load – 80 virtual users run for 4 minutes, spikes to 800 over 1 minute, drops to 80 over 1 minute, maintains 80 for 4 minutes, each executing all requests sequentially.
- Peak load – 300 virtual users run for 2 minutes, ramp up to 1500 over 2 minutes, maintains 1500 for 2 minutes, decrease to 300 over 2 minutes, maintains 300 for 2 minutes, each executing all requests sequentially.
- Fixed load – 500 virtual users run for 15 minutes, each executing all requests sequentially.

3.4. Metrics and statistical analysis

During each experimental run, performance metrics (average response time, 95th percentile (p95) response time, CPU utilization, and RAM consumption) together with workload execution metrics (total requests and completed iterations) and error metrics (overall error rate, HTTP errors, connection errors, and request timeouts) were recorded. The analysis was based on 10 independent runs for every Tool $\times$ Load combination. Response time metrics characterize both the typical request latency and its variability under different workload conditions, while resource utilization metrics reflect the computational overhead introduced by each tool. To ensure consistent comparison across the evaluated tools, Postman JSONError events were classified as HTTP errors because they represent failures in client-side processing of successfully received HTTP responses rather than connection failures or request timeouts. Since each experimental run included multiple REST API endpoints with different response times, a single performance value was computed for every Tool $\times$ Load $\times$ Run combination. Specifically, the Metric Mean Response Time (M) was calculated as the arithmetic mean of the average response times measured for all evaluated endpoints:

$$M = \frac{1}{N} \sum_{i=1}^N R_i, \quad (1)$$

where R_i denotes the average metric value of the i -th endpoint during a single experimental run; N is the total number of evaluated endpoints.

Average response time (Avg) was used to evaluate typical system latency, while the p95 metric was analyzed to capture tail latency and identify performance degradation under load.

CPU and RAM utilization were monitored on the local host using Prometheus time-series data in Grafana while executing the load tests. CPU and RAM utilization reported in this study represent only the resource consumption of the load-testing tool processes (k6, Apache JMeter, and Postman). Host-wide CPU and memory utilization were intentionally excluded from the analysis. Process-level metrics were collected using *windows_exporter* and retrieved from Prometheus via *PromQL* queries filtered by the process name (process="k6", process="java", process=~"postman"). During the experiments, no other Java processes except Apache JMeter were running on the load-generation host. For both CPU and memory metrics, values were aggregated by summing all matching process instances. CPU usage was calculated from the *windows_process_cpu_time_total* metric using the *PromQL* function *rate()*, while memory consumption was obtained from *windows_process_working_set_bytes*. The collected metrics served as dependent variables in the subsequent statistical analysis [28]. Each experimental run provided raw response time and resource

utilization measurements. These metrics were collected during a test run and aggregated into a single value before ANOVA. For statistical analysis, each row in resulting dataset represented an aggregated observation. For overall performance metrics, an observation corresponded to a Tool-Load-Run combination, while endpoint-level observations additionally included the API endpoint identifier. Average response time, p95 response time and errors were obtained directly from the summary reports exported by k6, JMeter, and Postman for each experimental run. Endpoint-level response time metrics were extracted from the corresponding endpoint-level reports and analyzed separately by applying the same statistical procedure to each endpoint individually. For k6 endpoint-level performance metrics were obtained by post-processing the exported JSON test report with a custom Python script, which aggregated HTTP request measurements by endpoint using only *http_req_duration* records and calculated the corresponding response time (avg, p95) statistics. CPU and RAM utilization were obtained from Prometheus time-series data in Grafana by first aggregating all matching process instances and then calculating the arithmetic mean of the sampled values over the test duration using the Grafana Mean function on Stat panel of the created test dashboard.

ANOVA was performed to evaluate the effects of the load testing tool, workload profile, and their interaction on the measured response time. The model included the factors *Tool*, *Load*, and the interaction term *Tool* $\times$ *Load*.

$$Metric = Tool * Load, \quad (2)$$

where *Metric* is the observed performance metric (e.g., Mean Avg Response Time, P95, CPU or RAM usage); *Tool* represents the evaluated load testing tool; *Load* represents the load profile;

The interaction term was included to determine whether the effect of the testing tool depends on the workload profile. For each analyzed metric, the following hypotheses were evaluated:

H1: The testing tool has a statistically significant effect on the metric;

H2: The load profile has a statistically significant effect on the metric;

H3: There is a statistically significant interaction between the testing tool and load profile.

Before applying ANOVA, the assumptions of normality and homogeneity of variances were evaluated. Normality of residuals was assessed using the Shapiro-Wilk test, while homogeneity of variances was verified using Levene's test. Since the CPU and RAM metrics results did not satisfy the normality assumption, a natural logarithmic transformation was applied prior to statistical analysis. The transformed data were subsequently used for testing ANOVA assumptions and fitting the two-way ANOVA models with factors *Tool* and *Load*. The residuals obtained from the fitted ANOVA model were used for the normality assessment. A p-value greater than 0.05 indicated that the corresponding assumption was not violated. When statistically significant differences were identified [29], Tukey's Honestly Significant Difference (HSD) post hoc test was applied to determine which pairs of tools differed significantly.

For each factor and their interaction, the F statistic was used to test the null hypothesis, the associated p-value was used to assess statistical significance, and partial eta squared (η^2_p) was reported as a measure of the proportion of variance explained by each effect in order to evaluate the practical importance of the observed differences. Experimental results were exported after each execution and processed using Python. Data preparation, statistical analysis, effect size calculation, and visualization were performed using pandas, SciPy, statsmodels, Pingouin, Matplotlib, and Seaborn. The workflow was fully automated to ensure reproducibility.

4. Results

The experimental results compare the performance of the evaluated testing tools under different workload levels. The analysis includes descriptive statistics, statistical significance testing using two-way ANOVA with Tukey's HSD post hoc analysis, endpoint-level performance evaluation, and resource utilization analysis (CPU and memory).

4.1. Performance metrics

To provide an initial comparison of the evaluated performance testing tools, the average response time (avg) for each workload level and then standard deviation, and 95% confidence interval (95% CI) were calculated. These descriptive statistics summarize the overall behavior of each tool before applying statistical analysis (see Table 3).

Table 3
Aggregated Avg and p95 response time performance metrics

Tool	Load	Metric	Mean	Standard Deviation	95% CI lower	95% CI upper
k6	Ramp	Avg	305.338	18.502	292.1027	318.573
k6	Spike	Avg	312.15	48.403	277.524	346.776
k6	Peak	Avg	2090.722	32.580	2067.416	2114.028
k6	Fixed	Avg	794.197	63.273	748.934	839.460
JMeter	Ramp	Avg	194.6	45.998	161.695	227.505
JMeter	Spike	Avg	132.3	8.460	126.248	138.352
JMeter	Peak	Avg	1022.1	38.869	994.295	1049.905
JMeter	Fixed	Avg	575.6	60.959	531.992	619.208
Postman	Ramp	Avg	400.6	89.641	336.475	464.726
Postman	Spike	Avg	510.383	79.221	453.712	567.054
Postman	Peak	Avg	1984.733	62.897	1939.74	2029.727
Postman	Fixed	Avg	637.4	166.239	518.480	756.321

The results in Table 3 and Figure 5a presents the aggregated average response time (avg) for the evaluated performance testing tools under different load profiles. Error bars correspond to the 95% confidence interval. The results show that k6 demonstrated the highest average response time across Peak and Fixed load, while JMeter had the lowest response time across all tested load types. Postman showed higher response time for Ramp and Spike load. Figure 5b demonstrates the aggregated p95 response time for evaluated performance testing tools under 4 workload profiles. For the Ramp and Spike workload profiles, JMeter tool consistently demonstrates lower p95 values than the other tools. The differences between p95 values of k6 and JMeter are relatively small for the Fixed profile, but become more pronounced under Peak loads for Avg metric. Postman p95 value increases more than the others for the Spike and Peak profiles, but for the Fixed profile, it is the smallest of all test tools.

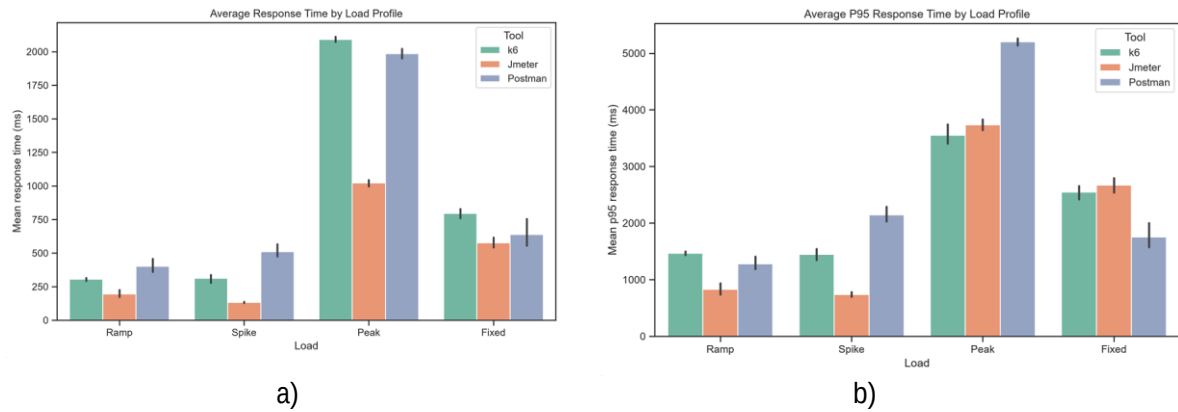


Figure 5: Mean response time: a) Average; b) p95 response time (source: own elaboration).

For each experimental run, both configured and achieved workload characteristics were recorded (Table 4). The reported values represent the mean across repeated runs for each Tool × Load combination. Total Requests in Table 4 represents the total number of HTTP requests generated during a test execution, while Iterations represents the number of completed workload scenario repetitions. The Table 4 and observed differences in total request counts shows variations in workload generation behavior among the evaluated tools.

Table 4

Mean values of Actual generated workload by performance testing tools

Tool	Load	Configured VUs	Test Duration	Total Requests	Iterations	Error rate (%)	Http Errors	Connection Errors	Timeouts
k6	Ramp	125 - 500	10	75112.5	12324.3	0	0	0	0
JMeter	Ramp	125 - 500	10	67169.8	11195.2	0	0	0	0
Postman	Ramp	125 - 500	10	58797.8	9799.2	0.0302	0	17.6	0
k6	Spike	80 - 800 - 80	10	29714.2	4659.4	0.0003	0	0.1	0
JMeter	Spike	80 - 800 - 80	10	28683.2	4780.5	0.017	31.7	450	0
Postman	Spike	80 - 800 - 80	10	26303.3	4383.5	0.4052	14	91.3	0
k6	Peak	300 - 1500 - 300	10	100611.9	17323.8	15.229	0	15326.8	0
JMeter	Peak	300 - 1500 - 300	10	115883.2	19313.9	14.5583	650.6	16216.8	0
Postman	Peak	300 - 1500 - 300	10	76738.2	12789.2	13.4913	3452	6942.4	0
k6	Fixed	500	15	121673.9	20088.2	0.246	0	299.8	0
JMeter	Fixed	500	15	125947	20991.2	0.7538	0	942.5	0
Postman	Fixed	500	15	110892.3	18481.6	1.3898	90	853.2	0

K6 tool generated a higher number of Total requests compared with other tools, indicating differences in workload execution mechanisms. JMeter produced a slightly lower request volume than k6 across most workload profiles, while exceeding other tools under Peak and Fixed workload profile. Postman generated fewer requests compared with JMeter and reached the lowest request volume for Peak workload profile.

4.2. Resources consumption

K6 tool demonstrated lower CPU utilization but higher memory consumption compared with the other tools, as shown in Figure 6a. This indicates a different resource utilization pattern during workload generation by tools. In addition to higher RAM consumption (Figure 6b), k6 had a higher average response time than JMeter. This suggests that JMeter was a more resource-efficient under the evaluated conditions. Postman had the highest CPU utilization during all load types in CLI mode, but lower RAM consumption than JMeter and k6.

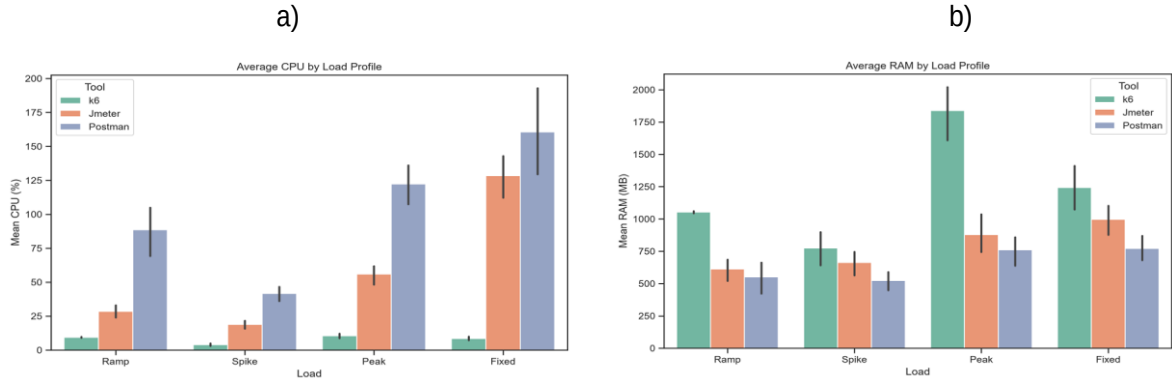


Figure 6: Average resources consumption by Load profile: a) CPU; b) RAM (source: own elaboration).

4.3. Statistical analysis

To determine whether the observed differences between the evaluated load-testing tools and workload profiles were statistically significant, a two-way ANOVA was performed for each analyzed metric using the model $Metric = Tool \times Load$ (see Formula 2). Two-way ANOVA revealed statistically significant main effects of both *Tool* and *Load*, as well as a significant *Tool* $\times$ *Load* interaction ($p < 0,001$) for all metrics. Avg and p95 metrics effects exhibited large effect sizes (Partial $\eta^2 > 0,6$), as shown in Table 5.

Table 5

Summarized ANOVA results for mean Avg, p95, CPU and RAM metrics

Metric	Factor	F	p-value	Partial η^2
Avg	Tool	415.750	< 0.001	0.885
	Load	2547.053	< 0.001	0.986
	Tool x Load	127.178	< 0.001	0.876
p95	Tool	88.901	< 0.001	0.622
	Load	1322.396	< 0.001	0.973
	Tool x Load	105.927	< 0.001	0.855
CPU	Tool	766.224	< 0.001	0.784
	Load	119.793	< 0.001	0.665
	Tool x Load	14.002	< 0.001	0.509
RAM	Tool	51.430	< 0.001	0.609
	Load	24.523	< 0.001	0.514
	Tool x Load	3.011	< 0.001	0.365

Overall, obtained ANOVA results indicate that the choice of testing *Tool* had a statistically significant effect on metrics, with highest value for CPU and RAM, supporting H1. The *Load* profile also demonstrated a significant influence on metrics, with highest value for *Avg* and *p95*, supporting H2. Furthermore, a significant interaction between the testing *Tool* and *Load* profile was observed, supporting H3.

This indicates that the performance differences between tools depend on the applied load conditions. Since significant main effects of the testing tools were identified, Tukey HSD post-hoc analysis was performed. JMeter achieved lower response time than Postman for all load types for *Avg* metric, as well Ramp, Spike, and Fixed load for *p95* metric, which was statistically significant (Tukey HSD, $p < 0.01$).

K6 achieved lower response time than Postman only for Ramp and Spike load for *Avg* metric, as well as Ramp, Spike load for *p95* metric. Regarding resource consumption, k6 showed significantly lower CPU utilization than JMeter under Spike and Peak workloads, while JMeter outperformed Postman for all workload profiles except Fixed.

K6 also consumed less CPU than Postman across all profiles. For RAM, the largest differences were observed in the k6-Postman and k6-JMeter comparisons: Postman consistently used less RAM than k6 across all workload profiles, whereas JMeter outperformed k6 under Ramp and Peak workloads.

4.4. Endpoint-level performance analysis

Figure 7 and Figure 8 present average and p95 response times across endpoints and workload profiles. Colors denote evaluated tools, and error bars represent 95% confidence intervals. JMeter achieved the lowest average and p95 response times across most endpoints, whereas k6 showed higher latency. The difference between tools increased with workload intensity, particularly for GET requests under Peak load. POST, PUT, and DELETE operations remained stable across profiles, while Postman demonstrated higher latency variability and k6 the highest p95 latency for high-load GET requests.

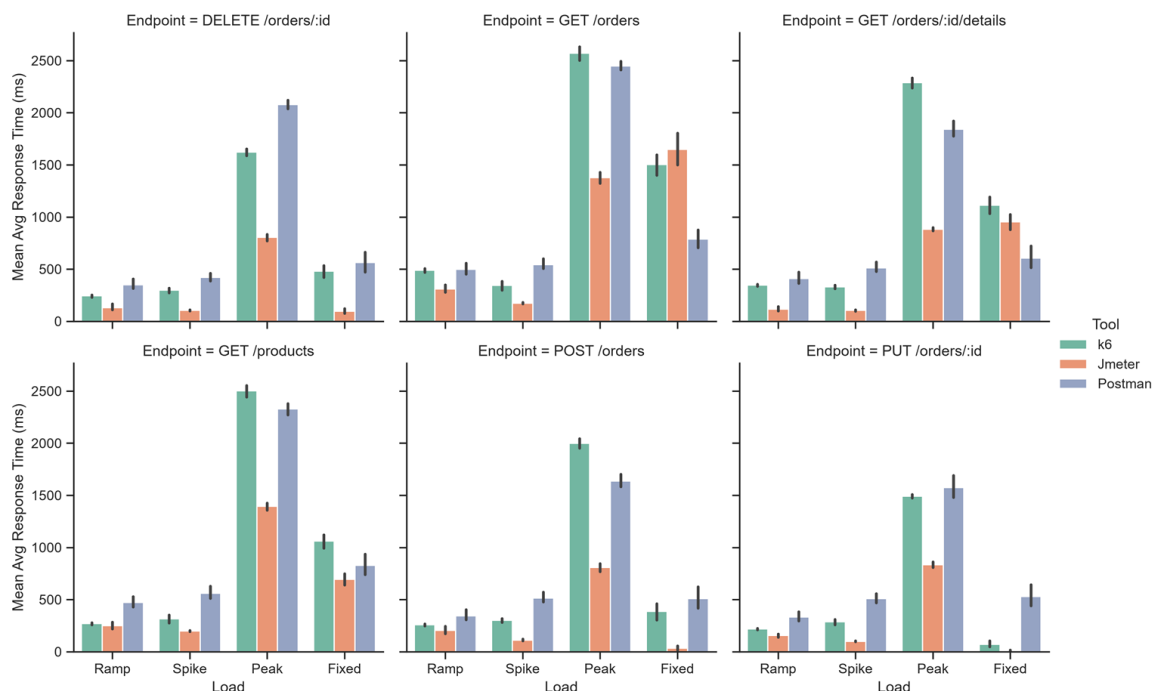


Figure 7: Mean Avg response time across endpoints (source: own elaboration).

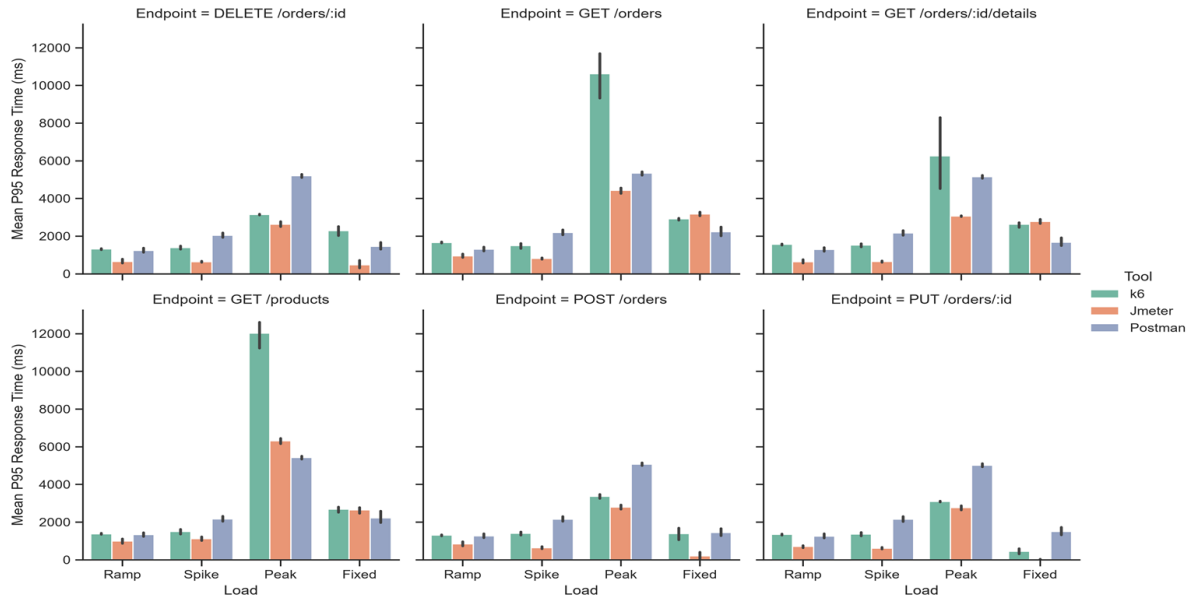


Figure 8: Mean p95 response time across endpoints (source: own elaboration).

5. Discussion

The obtained performance results were analyzed using a two-way ANOVA model with *Tool*, *Load*, and the *Tool* $\times$ *Load* interaction as factors. The analysis was conducted on a locally deployed Node.js application using Better-SQLite3 embedded database, which implements a simple set of CRUD operations. Performance testing was performed using k6, JMeter, and Postman testing tools for a case when four types of load as Ramp, Spike, Peak, and Fixed were applied, for identical workload scenarios. The experimental results obtained in Table 5 demonstrate that the selected *Load* and *Tool* has a statistically significant impact on several performance metrics, including average and p95 response time ($p < 0,001$ and Partial $\eta^2 > 0,6$). In addition, workload intensity significantly influenced response-time metrics, confirming that both the selected Tool and Load profile affect the observed performance characteristics. Tukey HSD analysis showed that differences between tools (k6-JMeter, k6-Postman, and JMeter-Postman) increased with workload intensity, indicating that tool performance is workload-dependent. Overall, JMeter demonstrated the best response-time characteristics across most workload profiles, while k6 and Postman showed higher latency under specific high-load conditions. The p95 results in Figure 5 revealed performance variability not captured by average response times, particularly for k6 under Ramp and Spike loads and Postman under Peak load. The resource consumption analysis in Figure 6 indicated different execution strategies among the evaluated tools. Lower CPU consumption of k6 was not consistently associated with lower memory usage, suggesting differences in workload generation and resource management approaches. Postman generally demonstrated the highest CPU utilization across workload profiles. The endpoint-level analysis from Figure 7 and Figure 8 confirmed that tool performance depends on request type and workload profile. JMeter provided the most stable performance for both read and write operations, while k6 showed increased latency for complex GET requests under Peak load. Postman achieved competitive results only for specific retrieval scenarios under Fixed workload. The results in Table 5 support RQ1 and RQ2, confirming statistically significant performance differences between the evaluated tools and their dependency on workload profiles. The endpoint-level analysis from Figure 7 and Figure 8 showed that tool impact increases with read-operation complexity, with k6 demonstrating higher latency for complex GET operations. The results suggest that load testing tool selection should consider response time, resource utilization, and workload characteristics rather than response time alone. This can support more informed tool selection for local performance evaluation and CI/CD integration. Limitations of the study are the evaluation of only three testing tools and experiments were conducted using a single application and a predefined workload scenarios. Different application architectures or more complex user interactions may produce

different performance characteristics. Finally, the experiments were executed within a controlled environment, which does not fully represent production deployments. Evaluation of additional load-testing tools, incorporating distributed load generation, and analyzing more complex workload scenarios, like multi-threading image processing [30] with the further decision-making [31] may be considered in the future.

6. Conclusions

An analysis of performance metrics for k6, JMeter, and Postman load testing tools was conducted. As a result average response time, p95 response time, errors, CPU and RAM utilization were selected for usage in performance tests on a locally deployed Node.js application. That made it possible to enable identification of difference between performance of test tools under 4 load types. Additionally, response time between different HTTP requests types was analyzed to check which endpoints are more sensitive to the choice of performance test tool. Unlike some publications, which focus only on response time comparison, this study focuses on response time metrics and resource consumption metrics analysis for well-known performance tools k6 and JMeter and popular API testing tool Postman, all of which provide CLI performance testing capabilities, which is less researched. The statistical analysis confirmed that both testing tool selection and workload intensity significantly affect response time and resource utilization metrics. No single tool consistently outperformed the others across all evaluated scenarios. JMeter demonstrated the best overall response-time performance, while k6 showed advantages in CPU efficiency with higher memory consumption. Postman tool exhibited higher resource consumption but may be suitable for specific lightweight testing scenarios. Consequently, no single tool consistently outperformed the others across all evaluated HTTP endpoints and performance metrics. Therefore, load testing tool selection should consider the evaluation objective, workload characteristics, and the trade-off between performance and resource efficiency.

Acknowledgements

The research was supported by the Ukrainian project of fundamental scientific research “Development of computational methods for detecting objects with near-zero and locally constant motion by optical-electronic devices” #0124U000259 in 2024-2026 years.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly Premium tool for the grammar improving and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

References

- [1] S. Khlamov, V. Savanevych, O. Briukhovetskyi, T. Trunova, Big data analysis in astronomy by the Lemur software, Proceedings of the IEEE UKRMICO (2023): 5-8. doi: 10.1109/UkrMiCo61577.2023.10380398.
- [2] V. Akhmetov et al., Cloud computing analysis of Indian ASAT test on March 27, 2019, Proceedings of the IEEE International Scientific and Practical Conference "PICST" (2019): 315-318. doi: 10.1109/PICST47496.2019.9061243.
- [3] V. Kozub, V. Druzhynin, D. Trufanova, P. Ihnatenko, K. Kolos, Using artificial intelligence in software development processes: achievements and challenges, Sustainable Engineering and Innovation 7.2 (2025): 463–476. doi: 10.37868/sei.v7i2.id526.

- [4] N. Li, J. Wang, C. Chen, H. & Hu, Application of API automation testing based on microservice mode in industry software, *Proceedings of the International Conference on Algorithms, Software Engineering, and Network Security* (2024): 460-464. doi: 10.1145/3677182.3677264.
- [5] H. Padmanaban et al., Using Java to create and analyze models of parallel computing system, *Parallel Computing* 125 (2025): 103146. doi: 10.1016/j.parco.2025.103146.
- [6] A. S. Abdelfattah, T. Cerny, J. Yero, E. Song, D. Taibi, Test coverage in microservice systems: An automated approach to e2e and API test coverage metrics, *Electronics* 13.10 (2024): 1913. doi: 10.3390/electronics13101913.
- [7] V. Troianskyi et al., Optical observations of the potentially hazardous asteroid (4660) Nereus at opposition 2021, *Icarus* 420 (2024): 116146. doi: 10.1016/j.icarus.2024.116146.
- [8] Y. R Shah, Comparative Analysis of API Test Automation Frameworks: An Experimental Study on Performance, Reliability, and CI/CD (Computer science), Ph.D. thesis, Parul University, Vadodara, Gujarat, India, 2026.
- [9] S. Khlamov, M. Mendieliava, O. Vovk, Zh. Deineko, Comparative analysis of JMeter and Postman for API-based performance testing, *CEUR Workshop Proceedings* 4048 (2025): 426-440. doi: 10.55056/ceur-ws.org/Vol-4048/paper34.pdf.
- [10] P. Buñay-Guisñan, N. Estrada, A. Flores, J. Erazo, Performance and Load Testing of Serverless Microservices, *Lecture Notes in Networks and Systems* 1532 (2024), Springer, Cham. doi: 10.1007/978-3-031-98890-5_44.
- [11] S. Khlamov et al., Machine Vision for Astronomical Images using The Modern Image Processing Algorithms Implemented in the CoLiTec Software, *Measurements and Instrumentation for Machine Vision* (2024): 269-310. doi: 10.1201/9781003343783-12.
- [12] I. W. R. Pinastawa, M. G. Pradana and N. Maulana, Web Server Performance Evaluation: Comparing Nginx and Apache Using K6 Testing Methods for Load, Spike, Soak, and Performance, *Proceedings of the International Conference on Informatics, Multimedia, Cyber and Information System* (2024): 1002-1007. doi: 10.1109/ICIMCIS63449.2024.10956402.
- [13] M. Pinyagin, A. Sadovykh, Automating Performance Testing in CI/CD – Tools Evaluation, *Lecture Notes in Computer Science* 16107 (2025), Springer, Cham. doi: 10.1007/978-3-032-05188-2_13.
- [14] G. I. Hegyi, M. Maliosz, C. Simon, Performance measurements of function as a service platforms, *Proceedings of the 3rd International Symposium on Computer Science and Intelligent Control* (2019): 1-6. doi: 10.1145/3386164.3390516.
- [15] E. Hadzhyiev et al., Application of Docker Compose for constructing the infocommunication system for online processing of astronomical images, *Proceedings of the 15th International Conference on Advanced Computer Information Technologies* (2025): 629-633. doi: 10.1109/ACIT65614.2025.11185586.
- [16] M. S. Prawira, F. Abdussalam, E. H. Yossy, Database Implementation using Citus on Telecommunication Data, *Proceedings of the 9th International Conference on Computing, Engineering and Design* (2023): 1-6. doi: 10.1109/ICCED60214.2023.10425036.

- [17] Y. Kravchenko et al., Optimizing and Improvement a Web Application Using Open Source Tools, CEUR Workshop Proceedings 3624 (2023): 368-379.
- [18] D. Kondepati, S. Mallidi, Performance Testing and Assessment of Various Network-Based Applications, Master's thesis, Blekinge Institute of Technology, Karlskrona, Sweden, 2019.
- [19] S. Siddhant, S. B. Prapulla, Comprehensive review of load testing tools, International Research Journal of Engineering and Technology 7.5 (2020): 3392-3395. <https://www.irjet.net/archives/V7/i5/IRJET-V7I5651.pdf>
- [20] AutomateNow, Advantages and Disadvantages of using JMeter, 2026. URL: <https://autmatenow.io/advantages-and-disadvantages-of-using-jmeter>
- [21] TestSigma, JMeter vs Postman, 2026. URL: <https://testsigma.com/blog/jmeter-vs-postman>
- [22] Postman, Performance Test Configuration, 2026. URL: <https://learning.postman.com/docs/tests-and-scripts/performance-testing/performance-test-configuration>
- [23] L. Wei, Best Postman Alternatives: A Complete Analysis, Article, University of Ottawa, Canada, 2024. URL: https://www.researchgate.net/publication/383623317_Best_Postman_Alternatives_A_Complete_Analysis
- [24] V. S. Susan Rini, When Postman Goes That Extra Mile to Deliver Performance to APIs, Software Testing Magazine, 2026. URL: <https://www.softwaretestingmagazine.com/tools/when-postman-goes-that-extra-mile-to-deliver-performance-to-apis>
- [25] D. Park, J. H. Kim, H. Sung, Performance and Resource Efficiency Analysis of Load Testing Tools for Web Game Workloads. Journal of Computer Games and Contents 38.7 (2025): 16-26.
- [26] C.-H. Hsieh et al., Evaluation System for Software Testing Tools in Complex Data Environment, Proceedings of the 4th International Conference on Information Communication and Signal Processing (2021): 604-609, doi: 10.1109/ICICSP54369.2021.9611846.
- [27] S. Di Meglio, L. L. L. Starace, V. Pontillo et al., A Taxonomy of Bad Practices in Software Performance Testing: Insights from Gray Literature and Practitioner Validation, ACM Transactions on Software Engineering and Methodology (2026). doi: 10.1145/3821538.
- [28] S. Khlamov, V. Savanevych, I. Tabakova, T. Trunova, The astronomical object recognition and its near-zero motion detection in series of images by in situ modeling, Proceedings of the 29th IEEE IWSSIP (2022): 1-4. doi: 10.1109/IWSSIP55020.2022.9854475.
- [29] V. O. Shvedun et al., Statistical modeling for determination of perspective number of advertising legislation violations, Actual Problems of Economics 184.10 (2016): 389-396.
- [30] S. Khlamov et al., Development of the matched filtration of a blurred digital image using its typical form, Eastern-European Journal of Enterprise Technologies 1.9-121 (2023): 62-71. doi: 10.15587/1729-4061.2023.273674.
- [31] Y. Romanenkov, V. Mukhin, V. Kosenko, D. Revenko, O. Lobach, N. Kosenko, A. Yakovleva, Criterion for Ranking Interval Alternatives in a Decision-Making Task, International Journal of Modern Education and Computer Science 16.2 (2024): 72-82. doi: 10.5815/ijmeecs.2024.02.06.