International Journal of Management Science and Information Technology
IJMSIT

E-ISSN: 2774-5694 P-ISSN: 2776-7388 OPEN ACCESS

Volume 6 (1), January-June 2026, 978-990
DOI: https://doi.org/10.35870/ijmsit.v6i1.7215

Performance Analysis of the SimInvest Application Programming Interface Using Load Testing

Cut Dahri Fajrina El Qahar [1*], Dewi Agushinta Rahayu [2], Lana Sularto [3]
[1*,2] Information Systems Management Study Program, Faculty of Computer Science and Information Technology, Universitas Gunadarma, Depok City, West Java Province, Indonesia
[3] Accounting Study Program, Faculty of Economics, Universitas Gunadarma, Depok City, West Java Province, Indonesia

Email: its.dahrina@gmail.com [1*], dewiar@staff.gunadarma.ac.id [2], lana@staff.gunadarma.ac.id [3]

Article history:
Received May 15, 2026
Revised June 19, 2026
Accepted June 22, 2026

Abstract
This study evaluates the performance capacity of the SimInvest application programming interface (API), with emphasis on stock transaction and portfolio services that are frequently accessed during peak market activity. The study used an applied performance-testing design through Apache JMeter by simulating concurrent user loads of 100, 500, 1,000, 2,000, and 4,000 users. The observed indicators were response time, transactions per second, and error rate. The findings show that the application remained usable under 100 and 500 concurrent users, with low aggregate error rates of 4.80% and 4.48%, although several history-related endpoints already showed failed requests. When the load reached 1,000 users, the total error rate increased to 53.17%, indicating a clear decline in service reliability. Under 2,000 and 4,000 users, the system recorded error rates of 49.04% and 65.32%, with repeated failures in Account Cash, Cash Withdrawal, Order List, Portfolio, RDN History, RDN Info, and Trade List services. These findings indicate that the present infrastructure requires API optimization, query tuning, server-capacity improvement, load balancing, and real-time monitoring to maintain reliable fintech services during traffic spikes.

Keywords:
Apache JMeter; Application programming interface; Load testing; Performance testing; SimInvest.

1. INTRODUCTION

Digital investment applications rely on stable transaction services because users expect market information, order placement, cash positions, and portfolio records to be available with minimal delay. In stock-trading applications, performance degradation is not only a technical inconvenience. Slow response, failed transactions, and delayed portfolio data can influence user trust, investment timing, and operational confidence. SimInvest, developed by Sinarmas Sekuritas, provides digital access to stock and mutual-fund investment services and supports activities such as buying, selling, monitoring orders, and managing portfolios (SimInvest Luncurkan Fitur Investasi Reksa Dana, 2023; Sinarmas Sekuritas, 2024).

The application depends on application programming interface (API) services that connect mobile interfaces with backend services. In this context, an API refers to a communication mechanism that allows the mobile application to send requests and receive data from server-side services. These services are particularly critical during the opening of trading sessions and other high-traffic moments, when many users' access transaction and portfolio feature concurrently. A stock-trading system is expected to manage order submission, price observation, account validation, and settlement-related information in a safe and transparent manner (Brzeszczyński, J., & Ibrahim, B. M., 2019). A portfolio feature also supports investment decisions by allowing users to monitor ownership, value changes, and potential gain or loss (Ou, 2023). Transaction processing systems support routine business activities by collecting, processing, and providing operational data for users and decision makers (Karolinda, 2021; Kurniawan., et al., 2019).

Software testing is a quality-control activity used to identify errors, verify system behavior, and ensure that software components operate according to defined requirements. In API-based applications, testing is important because API services are responsible for managing data exchange between users, application

interfaces, gateways, and backend systems. Recent studies emphasize that RESTful API testing remains essential because modern applications depend heavily on reliable API communication, authentication flows, data processing, and backend integration (Golmohammadi et al., 2023; Liu et al., 2022). In financial technology, testing cannot be limited to functional correctness because users interact with services under fluctuating demand. Non-functional testing is therefore required to measure speed, stability, scalability, and reliability under defined workloads.

Performance testing examines how an application responds to workload pressure and helps teams identify bottlenecks before production failures occur (Abdeen et al., 2023; ICreativeLabs, 2024). Load testing is one of the most relevant performance-testing approaches for this context because it evaluates system behavior under expected and increasing workloads. In this study, load testing was used to simulate multiple concurrent users accessing the SimInvest API at the same time. The main metrics consisted of response time, transactions per second, and error rate because these indicators represent service responsiveness, processing capacity, and reliability. Stress testing is useful for finding the breaking point of a system, whereas endurance testing evaluates longer duration stability. These approaches are complementary, but load testing is appropriate when the research objective is to understand service behavior across staged user volumes (Firstian, 2023; GRCI, 2024; Proxsis Consulting, 2024). Apache JMeter is widely used for this purpose because it can simulate HTTP requests, measure response-time distributions, and present throughput and error statistics for web, API, and mobile-service testing (Apache Software Foundation, n.d.; ICreativeLabs, 2024; Nosuke, 2023).

Previous studies have used load-testing tools in various application contexts. Ismail et al. (2023) tested an online examination system using JMeter and found that hardware upgrades alone did not fully address errors at high user loads. Tejaya et al. (2023) evaluated the Invitees website and showed that response time could exceed the intended threshold under stress conditions. Madhani et al. (2023) tested a core banking system using load and stress testing, finding that transacting features required longer processing than non-transacting features. Ginasari et al. (2021) examined API stress testing with JMeter and observed error messages when the sample size increased. Permatasari (2020) also showed that JMeter could verify whether an application met response-time and memory targets. These studies show that performance evaluation is strongly context-sensitive; every system must be tested using its own workload, features, and infrastructure conditions.

Based on this context, this study aims to evaluate the performance of SimInvest API services for stock transactions and portfolio management using load testing. The study focuses on three indicators: response time, transactions per second (TPS), and error rate. This research contributes practical evidence for identifying API services that require optimization and provides a structured performance-testing report that can support fintech application development and operational planning.

2. RESEARCH METHOD

This study used an applied quantitative performance-evaluation design. The object was the SimInvest API for stock transaction and portfolio services. Data were collected through document review, business-process analysis, application-architecture observation, endpoint identification, and load-testing execution using Apache JMeter. The research procedure followed the Software Testing Life Cycle logic, which begins with requirement analysis, continues with planning and test-case configuration, and ends with execution, reporting, and evaluation (Testim, 2023). The first stage defined the object, service scope, and workload assumptions for the SimInvest API performance test.

Table 1. Performance testing activities

Performance Testing
Test planning
Test-scenario design
Test-environment setup
Testing-tool implementation
Test execution
Data collection
Data analysis, reporting, and retesting

Table 1 shows the seven activities used in the performance-testing process, from test planning to data analysis, reporting, and retesting. These activities ensured that the load test was conducted as a structured procedure rather than as a standalone technical execution. The workload basis was derived from SimInvest user data and active-user information.

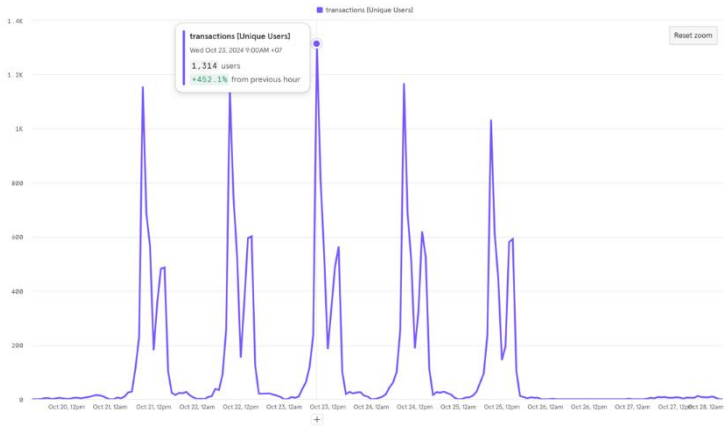

Figure 1. Daily active users trend

Figure 1 shows the daily active-user trend, in which user activity rises sharply during specific trading periods. This pattern indicates that SimInvest needs to maintain API stability during peak access windows.

Figure 2. Total SimInvest users

Figure 2 shows 83.94K total SimInvest users. This user base supports the relevance of testing concurrent access because only a portion of total users can still create substantial load during market activity peaks.

Table 2. Data collection results

No.	Object	Specification
1	Total SimInvest users	83,940
2	Daily Active Users (DAU) during peak trading hours	1,200-2,000 users
3	API services	HTTP RESTful API: POST Login User ID; POST Login Trading PIN; GET Stock Quote; GET Order Book; POST Order Buy; POST Order Sell; GET Order List; GET Trade List; GET Order History; POST Cash Withdrawal; GET RDN History; GET RDN Info; GET Account Cash; GET Portfolio

Table 2 shows the empirical basis for defining the test scope. The total number of SimInvest users and the peak-hour daily active users justify the staged simulation, while the listed RESTful API endpoints identify the services evaluated in the test. The research flow was organized to connect requirement analysis, scenario design, test execution, result analysis, and recommendation formulation.

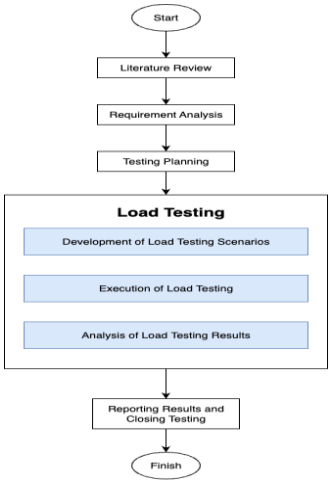

Figure 3. Research flow

Figure 3 shows the research flow, beginning with literature review and requirement analysis, followed by testing planning, load-testing scenario development, execution, result analysis, reporting, and completion. This sequence aligns the technical test with the research objective. The technical environment was documented before the test was executed.

Table 3. Hardware specifications

No.	Device	Quantity	Specification
1	Laptop	1	13-inch Apple M1, 8 GB RAM, macOS Sequoia 15.0, 245 GB storage
2	Mobile phone / smartphone	1	256 GB, iOS 18.1.1

Table 3 shows the hardware used during the test, consisting of a laptop and a smartphone. These devices served as the local testing environment for configuring JMeter and accessing the SimInvest application.

Table 4. Software specifications

No.	Software	Version
1	Operating system	macOS Sequoia version 15.0
2	SimInvest application	Version 1.120.1
3	JMeter	Version 5.6.3

Table 4 shows the software environment, including macOS Sequoia, SimInvest version 1.120.1, and Apache JMeter version 5.6.3. These details are important because performance-test results are influenced by application version, operating system, and testing-tool configuration. The tested business processes consisted of stock transactions and portfolio monitoring.

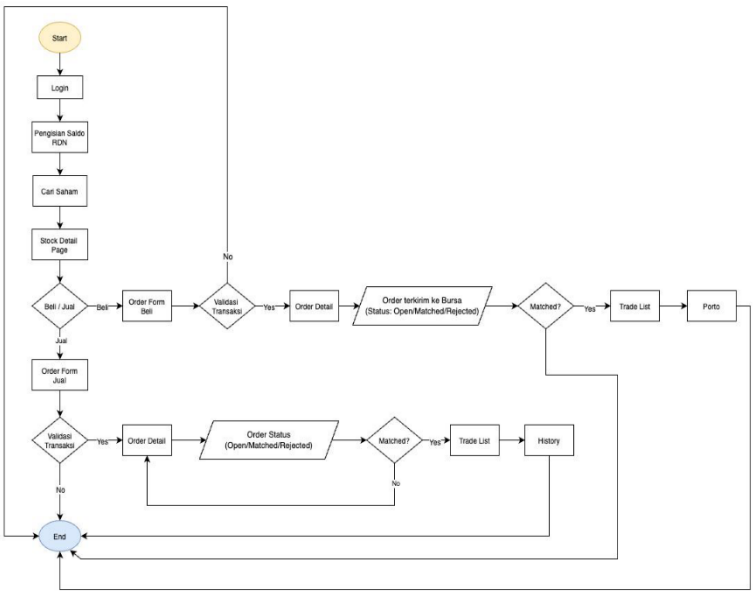

Figure 4. Stock transaction process

Figure 4 shows the stock transaction process from login and account verification to buy or sell order submission and trade confirmation. This process explains why order-related endpoints were included in the load test.

Figure 5. Portfolio management process

Figure 5 shows the portfolio management process, in which users access portfolio information after authentication. This process supports the inclusion of the portfolio endpoint as one of the tested API services. The SimInvest API architecture was reviewed to understand the system layers involved in the load test.

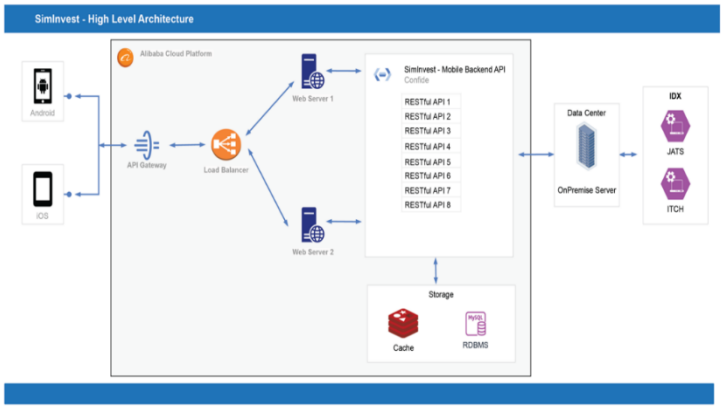

Figure 6. Application architecture

Figure 6 shows the application architecture connecting user devices, gateway, load balancer, web servers, API services, cache, Redis, database, and data-center components. The architecture clarifies that API performance depends on server capacity, service routing, and backend data access.

Table 5. Application server specifications

No.	Server	Domain	CPU (Core)	Memory (GB)	Disk (GB)	OS
1	Web Server 1	srv1.xxx.co.id	4	16	124	Ubuntu
2	Web Server 2	srv2.xxx.co.id	4	16	124	Ubuntu

Table 5 shows two Ubuntu-based web servers with four CPU cores, 16 GB memory, and 124 GB disk capacity each. This configuration became the infrastructure context for interpreting system response under increasing concurrent-user loads. The measurement parameters and tested business features were defined before test execution.

Table 6. Object testing parameters

No.	Parameter
1	Error rate
2	Transactions per second (TPS)
3	Response time

Table 6 shows the three testing parameters: error rate, transactions per second, and response time. These parameters were used to evaluate reliability, processing capacity, and service responsiveness.

The performance indicators were defined to support consistent interpretation of the test results. Response time refers to the time required by the server to process a request and return a response to the client. Transactions per second (TPS) refers to the number of requests or transactions successfully processed by the system within one second. Error rate refers to the percentage of failed requests compared with the total number of requests executed during the test. In addition, 400 Bad Request indicates that the server could not process the request because of invalid request syntax or parameters, 401 Unauthorized indicates an authentication or authorization problem, and 502 Bad Gateway indicates that the gateway or proxy server did not receive a valid response from the upstream server. These definitions were used to interpret whether the SimInvest API remained responsive, stable, and reliable under increasing concurrent-user loads.

Table 7. SimInvest business process

Feature ID	Feature	Description
001	Login User ID	Allows users to access the SimInvest application.
002	Login Trading PIN	Provides transaction verification and an additional security layer before financial activities.
003	Stock Quote	Provides current stock-price information, including opening price, last price, and average price.
004	Order Book	Displays queue information for bid and ask prices in the stock market.
005	Order Buy	Allows users to submit buy orders by defining quantity and target price.
006	Order Sell	Allows users to submit sell orders by defining quantity and selling price.
007	Order List	Displays the complete list of buy and sell orders created during the current trading day.
008	Trade List	Displays stock transactions that have been successfully executed.

009	Order History	Records previous stock orders, including successful and failed orders.
0010	Cash Withdrawal	Allows users to withdraw funds from the customer fund account to a personal bank account.
0011	RDN History	Displays the transaction history of the customer fund account.
0012	RDN Info	Shows balance and other key information related to the customer fund account.
0013	Account Cash	Displays available cash balance for investment transactions.
0014	Portfolio	Displays asset information and supports stock investment monitoring.

Table 7 shows the SimInvest business processes represented by 14 API features, including authentication, stock quotation, order execution, cash services, RDN services, and portfolio access. These features reflect critical user activities in stock trading and investment monitoring. The performance objectives and JMeter scenario were used to structure the load-testing execution.

Table 8. Performance test objectives

No.	Parameter	Objective
1	Error rate	Lower is better
2	Transactions per second (TPS)	Higher is better
3	Response time	< 2.5 seconds

Table 8 shows the performance objectives used in the study. A lower error rate, higher TPS, and response time below 2.5 seconds were used as benchmarks for evaluating whether the API services remained acceptable under load.

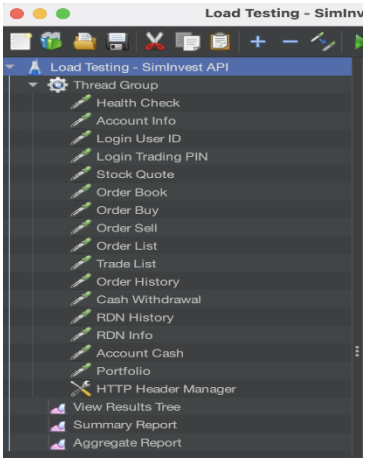

Figure 7. JMeter testing scenario

Figure 7 shows the JMeter testing scenario containing the API request sequence, including health check, login, stock quote, order, RDN, account cash, portfolio, and reporting components. This setup enabled endpoint-level observation during load testing. The JMeter configuration was prepared after the API scope and request sequence had been finalized.

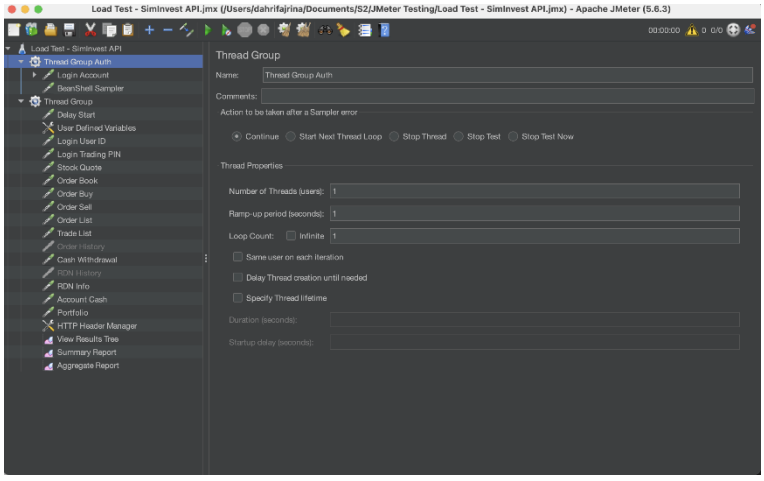

Figure 8. Thread group configuration in Apache JMeter

Figure 8 shows the thread group configuration in Apache JMeter. The scenario used a one-second ramp-up period and one loop count, allowing each user-load condition to be executed consistently across the tested endpoints.

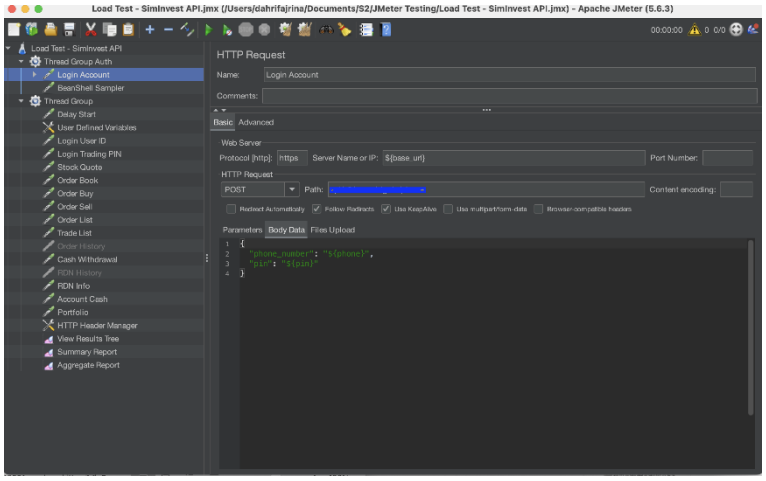

Figure 9. Login account request configuration in Apache JMeter

Figure 9 shows the login account request configuration in Apache JMeter. Parameterized credentials were used to complete authentication before transaction, RDN, account, and portfolio requests were executed. The authentication output was configured for reuse across endpoint calls.

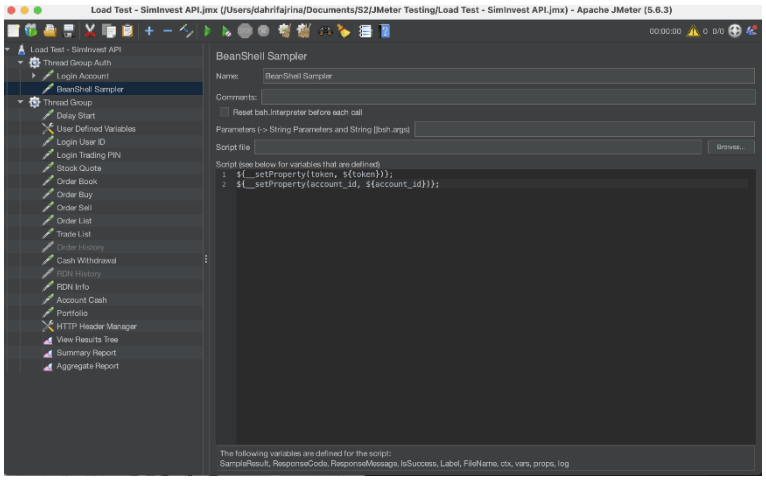

Figure 10. Token and account ID storage configuration in Apache JMeter

Figure 10 shows the BeanShell sampler configuration used to store the authentication token and account ID. This mechanism allowed subsequent API requests to reuse valid authentication data during the test scenario.

3. RESULTS AND DISCUSSION

3.1. Load Testing Execution and Performance Overview

The load-testing results are presented by user scenario to show how API performance changed as concurrent users increased.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**1500**	**72**	**4.80%**	**1093.71**	**118**	**8662**	**358.00**	**5158.00**	**5219.95**	**7594.64**	**76.50**	**644.20**	**29.53**
Account Cash	100	0	0.00%	385.62	285	698	359.50	419.60	664.75	697.95	13.00	19.21	4.46
Cash Withdrawal	100	0	0.00%	5193.00	5148	5490	5187.00	5224.90	5250.55	5487.73	15.11	19.61	6.39
Health Check	100	0	0.00%	1461.74	954	1788	1488.00	1734.80	1750.90	1787.96	55.28	62.06	18.25
Login Trading PIN	100	0	0.00%	129.16	118	168	127.50	135.90	141.95	167.99	192.68	227.77	80.72
Login User ID	100	0	0.00%	249.67	132	513	239.00	290.00	318.25	511.21	189.04	217.86	80.49
Order Book	100	0	0.00%	139.20	120	433	136.00	147.90	175.95	430.71	156.01	196.24	53.47
Order Buy	100	0	0.00%	307.63	199	675	295.50	368.60	573.15	674.59	121.80	173.06	63.88
Order History	100	62	62.00%	340.79	168	778	305.50	503.20	601.35	777.23	55.71	78.26	20.18
Order List	100	0	0.00%	896.30	645	1794	764.00	1429.00	1484.60	1793.42	50.18	4469.13	17.25
Order Sell	100	0	0.00%	224.20	179	514	209.50	249.90	297.65	513.92	121.21	163.72	63.45
Portfolio	100	0	0.00%	356.81	301	653	354.00	387.90	393.95	650.43	13.04	32.00	4.48
RDN History	100	10	10.00%	5607.42	1635	8662	5815.00	7968.80	8128.45	8661.50	11.10	209.85	4.34
RDN Info	100	0	0.00%	487.29	358	740	477.50	571.90	612.40	739.88	12.85	21.98	4.34
Stock Quote	100	0	0.00%	205.67	122	495	201.00	210.90	215.95	494.93	165.84	222.74	56.03
Trade List	100	0	0.00%	421.21	337	958	399.00	480.80	502.60	957.04	58.96	69.50	20.27

Figure 11. Load testing result for 100 concurrent users

Figure 11 shows the result for 100 concurrent users. The test produced 1,500 samples with an average response time of 1,093.71 ms, 76.50 transactions per second, and a 4.80% total error rate. Most endpoints recorded no error, but Order History and RDN History already showed failures, indicating early sensitivity in history-related services.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**7500**	**336**	**4.48%**	**2637.29**	**60**	**25885**	**463.00**	**7736.50**	**11400.85**	**22983.99**	**145.88**	**4831.97**	**56.32**
Account Cash	500	0	0.00%	321.87	98	9006	143.00	371.00	864.90	7037.61	15.51	21.89	5.32
Cash Withdrawal	500	0	0.00%	5760.89	5098	11063	5435.50	6951.80	7603.75	9195.84	22.10	28.82	9.35
Health Check	500	0	0.00%	2767.06	165	4535	2844.50	4036.40	4222.80	4490.99	104.04	120.79	34.34
Login Trading PIN	500	0	0.00%	129.61	62	535	91.50	272.90	344.00	433.98	193.72	219.72	81.16
Login User ID	500	0	0.00%	159.82	64	973	106.00	316.80	504.00	886.90	195.47	225.80	83.23
Order Book	500	0	0.00%	244.29	63	4767	135.50	494.40	655.80	2136.09	65.42	97.04	22.42
Order Buy	500	0	0.00%	817.39	137	6232	513.50	1584.00	2693.05	5404.73	47.43	67.57	24.87
Order History	500	270	54.00%	1367.57	124	12323	622.50	3484.80	6408.60	9587.22	32.02	50.84	11.60
Order List	500	0	0.00%	7484.00	484	14654	8458.00	11457.40	11926.80	13012.65	30.61	12169.98	10.52
Order Sell	500	0	0.00%	1271.86	116	7566	833.00	3021.90	4182.75	5681.33	33.04	45.15	17.30
Portfolio	500	0	0.00%	245.72	176	3957	205.00	254.80	415.60	1520.80	15.82	18.27	5.44
RDN History	500	66	13.20%	14723.17	462	25885	16337.50	23448.10	24632.45	25733.28	13.41	1096.60	5.24
RDN Info	500	0	0.00%	896.73	112	9169	190.50	3040.10	5202.75	8198.51	14.46	25.21	4.89
Stock Quote	500	0	0.00%	168.09	60	1217	119.00	322.90	409.90	725.61	156.69	208.89	52.94
Trade List	500	0	0.00%	3201.25	193	10823	2669.50	6700.50	7738.10	10236.59	30.96	35.70	10.64

Figure 12. Load testing result for 500 concurrent users

Figure 12 shows the result for 500 concurrent users. The scenario generated 7,500 samples, increased the average response time to 2,637.29 ms, and recorded 145.88 transactions per second with a 4.48% error rate. Order History and RDN History remained the main problematic endpoints, suggesting a consistent weakness in historical data retrieval.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**15000**	**7975**	**53.17%**	**3232.02**	**35**	**41539**	**347.00**	**5912.90**	**33032.75**	**40130.93**	**267.97**	**751.87**	**103.05**
Account Cash	1000	1000	100.00%	275.27	36	1261	264.00	409.90	526.90	593.96	279.80	290.72	95.91
Cash Withdrawal	1000	1000	100.00%	606.20	40	32664	389.00	1276.90	1315.00	1365.98	27.46	28.24	11.61
Health Check	1000	0	0.00%	2844.80	83	7298	1864.50	6654.80	6862.90	7111.99	92.81	107.56	30.63
Login Trading PIN	1000	0	0.00%	211.50	74	957	142.00	450.90	736.00	895.00	357.53	410.50	149.78
Login User ID	1000	0	0.00%	285.50	74	1268	228.00	547.00	679.90	810.99	197.47	226.60	84.08
Order Book	1000	0	0.00%	420.18	83	5660	320.50	735.80	992.40	2046.70	104.60	153.15	35.85
Order Buy	1000	0	0.00%	1442.22	207	9595	932.00	3212.50	4283.15	7078.70	83.82	118.54	43.96
Order History	1000	998	99.80%	928.53	39	32867	391.00	1312.90	1357.00	18746.09	24.10	25.00	8.73
Order List	1000	971	97.10%	35598.36	191	41539	38128.50	40479.70	40847.70	41256.99	23.65	611.60	7.59
Order Sell	1000	18	1.80%	3576.92	260	32651	2448.00	6859.60	7636.00	31804.05	23.63	32.10	12.37
Portfolio	1000	1000	100.00%	194.93	35	1304	189.00	296.90	318.90	546.95	271.15	278.97	93.21
RDN History	1000	1000	100.00%	388.69	40	1423	335.00	552.00	591.95	1311.96	251.76	257.72	98.34
RDN Info	1000	1000	100.00%	353.98	39	1424	311.00	548.90	579.00	1214.78	275.63	283.63	93.13
Stock Quote	1000	0	0.00%	355.71	84	1837	308.00	647.60	779.95	1269.90	245.58	324.77	82.98
Trade List	1000	988	98.80%	997.56	38	31346	275.50	1330.90	1365.00	17594.90	24.13	24.85	8.29

Figure 13. Load testing result for 1,000 concurrent users

Figure 13 shows the result for 1,000 concurrent users. The test produced 15,000 samples, 3,232.02 ms average response time, 267.97 transactions per second, and a 53.17% error rate. Reliability declined sharply because several account, cash, RDN, portfolio, and trade-list services recorded near-total or complete failure.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**30000**	**14711**	**49.04%**	**6523.06**	**35**	**95669**	**467.00**	**32275.60**	**48993.75**	**68722.84**	**268.83**	**7156.50**	**103.57**
Account Cash	2000	1968	98.40%	1388.64	35	59463	243.00	610.60	2051.95	42329.00	29.96	31.48	10.27
Cash Withdrawal	2000	1718	85.90%	5526.52	37	60044	322.50	18726.00	39475.30	56781.26	19.39	21.57	8.20
Health Check	2000	0	0.00%	2059.85	92	12536	1149.00	4082.40	5806.55	11340.98	79.36	91.08	26.20
Login Trading PIN	2000	0	0.00%	2393.28	51	9190	2485.00	5055.00	5753.80	7063.99	50.60	57.96	21.20
Login User ID	2000	0	0.00%	2314.15	54	10389	1724.00	5212.70	6649.90	9065.73	62.69	71.98	26.69
Order Book	2000	1	0.05%	4256.16	54	27559	3991.00	8891.90	10006.80	13068.98	31.35	46.11	10.75
Order Buy	2000	34	1.70%	7264.55	133	38373	8081.00	13293.30	14810.70	18051.85	19.59	26.95	10.27
Order History	2000	1789	89.45%	2523.88	38	49593	341.00	5055.50	12532.30	42669.91	19.16	24.28	6.94
Order List	2000	1546	77.30%	40181.39	87	95669	45525.50	68721.40	79949.30	87560.81	18.72	7164.59	6.21
Order Sell	2000	254	12.70%	7654.07	102	60049	5116.50	15586.80	18702.70	48343.99	19.02	25.80	9.96
Portfolio	2000	1955	97.75%	978.97	35	57573	229.00	463.90	1172.80	30206.37	30.02	31.06	10.32
RDN History	2000	1999	99.95%	9218.58	38	67270	324.50	45723.50	51713.55	60419.76	21.04	22.95	8.22
RDN Info	2000	1952	97.60%	1268.26	36	60367	289.00	2766.10	4065.85	20820.33	29.79	31.57	10.07
Stock Quote	2000	0	0.00%	5396.11	52	33537	6119.50	10428.60	11363.90	13975.90	36.55	47.93	12.35
Trade List	2000	1495	74.75%	5421.55	37	63148	2788.50	13514.00	18981.55	46810.89	18.97	20.54	6.52

Figure 14. Load testing result for 2,000 concurrent users

Figure 14 shows the result for 2,000 concurrent users. The scenario produced 30,000 samples, an average response time of 6,523.06 ms, and a 49.04% error rate, with 14,711 failed requests. The high number of failed requests indicates backend saturation and unstable service routing under heavy demand.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**60000**	**39193**	**65.32%**	**924.04**	**35**	**6015**	**1337.00**	**1821.00**	**1919.00**	**2115.00**	**1391.21**	**1639.24**	**537.10**
Account Cash	4000	4000	100.00%	1101.64	86	2673	1203.50	1771.90	1870.00	2082.00	103.04	120.40	35.32
Cash Withdrawal	4000	4000	100.00%	1171.32	46	2516	1290.00	1792.00	1895.95	2153.91	108.99	127.76	46.09
Health Check	4000	0	0.00%	541.05	82	6015	165.00	1541.20	2790.40	5427.88	129.07	150.01	42.60
Login Trading PIN	4000	125	3.13%	110.52	35	1073	63.00	232.00	311.95	726.99	138.36	160.26	57.96
Login User ID	4000	132	3.30%	124.23	38	1268	72.00	262.00	348.00	745.99	139.34	161.01	59.33
Order Book	4000	182	4.55%	1031.73	38	2332	1124.00	1713.00	1815.00	2090.98	126.63	146.07	43.40
Order Buy	4000	4000	100.00%	1145.34	81	2491	1282.00	1784.00	1893.00	2118.99	122.15	144.30	64.06
Order History	4000	2545	63.63%	680.49	49	2913	322.00	1805.90	2004.00	2265.99	109.19	134.27	39.56
Order List	4000	4000	100.00%	1193.21	49	2613	1328.50	1811.90	1907.95	2161.98	114.83	133.38	39.47
Order Sell	4000	4000	100.00%	1178.98	46	2642	1334.50	1796.00	1895.80	2134.98	117.78	139.06	61.65
Portfolio	4000	4000	100.00%	1049.08	85	2504	1153.00	1748.90	1859.00	2059.93	103.49	120.52	35.58
RDN History	4000	4000	100.00%	1171.18	82	2528	1288.00	1789.90	1907.00	2101.99	108.32	126.35	42.31
RDN Info	4000	4000	100.00%	1155.15	84	3647	1267.00	1792.00	1900.95	2099.00	103.05	120.45	34.82
Stock Quote	4000	209	5.22%	1009.14	37	2450	1069.50	1689.00	1777.95	2003.97	131.79	169.39	44.53
Trade List	4000	4000	100.00%	1197.62	56	2484	1315.00	1807.00	1915.85	2119.00	112.00	130.54	38.50

Figure 15. Load testing result for 4,000 concurrent users

Figure 15 shows the result for 4,000 concurrent users. Although transactions per second reached 1,391.21, the error rate increased to 65.32%, with 39,193 failed requests. The lower aggregate average response time was influenced by fast failed responses, not by healthy processing.

3.2. Error Distribution Across Load Scenarios

The error distribution was analyzed to identify API endpoints that were most sensitive to concurrent-user pressure.

Table 9. Error results for 100 users

No.	API	Error	Number of Errors
1	Order History	400/Bad Request	57
2	RDN History	400/Bad Request	4
Total	Total	Total	61

Table 9 shows that under 100 users, errors were limited to Order History and RDN History, with 61 total failed requests. This indicates that historical data retrieval became unstable earlier than most other services.

Table 10. Error results for 500 users

No.	API	Error	Number of Errors
1	Order History	400/Bad Request	270
2	RDN History	400/Bad Request	66
Total	Total	Total	336

Table 10 shows that the same two endpoints failed under 500 users, with total errors increasing to 336. The concentration of failures in Order History and RDN History suggests a persistent issue in history-based API access.

Table 11. Error results for 1,000 users

No.	API	Error	Number of Errors
1	Account Cash	502/Bad Gateway	1,000
2	Cash Withdrawal	502/Bad Gateway	1,000
3	Order History	502/Bad Gateway	998
4	Order List	502/Bad Gateway	905
5	Order Sell	502/Bad Gateway	18
6	Portfolio	502/Bad Gateway	1,000
7	RDN History	502/Bad Gateway	1,000
8	RDN Info	502/Bad Gateway	1,000
9	Trade List	502/Bad Gateway	988
Total	Total	Total	7,909

Table 11 shows that at 1,000 users, the error pattern expanded to 7,909 failed requests across nine API services. The dominance of 502 Bad Gateway responses indicates that gateway or upstream services could not consistently handle this load.

Table 12. Error results for 2,000 users

No.	API	Error	Number of Errors
1	Account Cash	502/Bad Gateway	1,968
2	Cash Withdrawal	502/Bad Gateway	1,718
3	Order Book	400/Bad Request	1
4	Order Buy	401/Unauthorized	34
5	Order History	502/Bad Gateway	1,789
6	Order List	502/Bad Gateway	1,546
7	Order Sell	400/Bad Request	254
8	Portfolio	502/Bad Gateway	1,955
9	RDN History	502/Bad Gateway	1,999
10	RDN Info	502/Bad Gateway	1,952
11	Trade List	502/Bad Gateway	1,495
Total	Total	Total	14,711

Table 12 shows 14,711 failed requests at 2,000 users. The largest failures occurred in RDN History, Account Cash, Portfolio, RDN Info, Order History, Cash Withdrawal, Order List, and Trade List, indicating broader service instability.

Table 13. Error results for 4,000 users

No.	API	Error	Number of Errors
1	Account Cash	502/Bad Gateway	4,000
2	Cash Withdrawal	502/Bad Gateway	4,000
3	Login Trading PIN	502/Bad Gateway	125
4	Login User ID	502/Bad Gateway	132
5	Order Book	502/Bad Gateway	182
6	Order Buy	502/Bad Gateway	4,000
7	Order History	502/Bad Gateway	2,545
8	Order List	502/Bad Gateway	4,000
9	Order Sell	502/Bad Gateway	4,000
10	Portfolio	502/Bad Gateway	4,000
11	RDN History	502/Bad Gateway	4,000
12	RDN Info	502/Bad Gateway	4,000
13	Stock Quote	502/Bad Gateway	209
14	Trade List	502/Bad Gateway	4,000
Total	Total	Total	39,193

Table 13 shows 39,193 failed requests at 4,000 users, the highest error count in the study. Several services reached full request failure, confirming that transactional, account, RDN, and portfolio services were the most vulnerable at extreme load.

3.3. Interpretation and Improvement Priorities

The results show that SimInvest API services were able to process low to moderate simulated traffic, but reliability weakened when concurrent users reached 1,000 and above. The first two scenarios still produced low aggregate error rates, although Order History and RDN History already showed repeated failures. Once the workload reached 1,000 users, the system moved from limited endpoint instability to broad service failure. The appearance of 502 Bad Gateway errors indicates that the gateway or upstream services failed to return valid responses under load. This finding is consistent with prior performance-testing evidence showing that server scaling without architectural optimization may not remove bottlenecks (Ismail et al., 2023; Madhani et al., 2023).

The tested endpoints can be grouped into three performance categories. The first category consists of relatively resilient services, such as Health Check, Login User ID, Login Trading PIN, Stock Quote, and Order Book. These services either had low error rates or remained partially available under heavy load. The second category includes services that became unstable from early scenarios, especially Order History and RDN History. The third category consists of services that collapsed under higher load, including Account Cash, Cash Withdrawal, Portfolio, RDN Info, Order List, Order Buy, Order Sell, and Trade List. This grouping is useful for prioritizing technical improvements.

The practical implication is that API optimization should begin with history, cash, RDN, and portfolio endpoints. These services likely require query optimization, indexing, caching for frequently requested data, connection-pool tuning, and review of timeout configuration. Order-related endpoints also require attention because a failure in buy or sell services can directly affect user trust and transaction continuity. Since two web servers were used in the tested architecture, load balancing should be reviewed to ensure traffic is distributed properly and unhealthy upstream services are detected quickly. Database replication or read-write separation can also reduce the pressure on backend services that retrieve historical records.

The study also shows the importance of continuous performance monitoring. A high TPS value is not sufficient evidence of healthy performance when error rates are also high. In the 4,000-user scenario, throughput increased but service reliability dropped severely. For this reason, future performance dashboards should combine TPS with error rate, response-time percentiles, gateway errors, resource utilization, and endpoint-specific failure patterns. Periodic regression testing after application updates is also needed so that new releases do not reduce performance capacity.

3.4. Research Limitations and Future Research Directions

Although this study provides empirical evidence on the performance of the SimInvest API under staged concurrent-user loads, several limitations should be acknowledged. First, the testing was conducted in a controlled environment using Apache JMeter, so the results may not fully represent all dynamic conditions in a real production environment, such as fluctuating network quality, user device diversity, background services, and unpredictable user behavior during actual market activity. Second, this study focused on load testing with specific concurrent-user scenarios of 100, 500, 1,000, 2,000, and 4,000 users. Therefore, the

maximum breaking point of the system has not been examined in more detail through extended stress testing. Third, the analysis was limited to response time, transactions per second, and error rate, while server-side resource utilization, such as CPU usage, memory consumption, database load, connection-pool behavior, and network latency, was not measured directly. Fourth, the tested API services were limited to selected stock transaction and portfolio-related features, so the results may not fully describe the performance of all SimInvest services.

Future studies should extend the evaluation by conducting stress testing, endurance testing, and spike testing in a production-like environment. Further research should also include server-side monitoring to capture CPU, memory, database, cache, and network utilization during load execution. In addition, future studies may compare different load-balancing strategies, caching mechanisms, database indexing approaches, and API optimization techniques to identify the most effective solution for improving service reliability. These extensions would provide a more comprehensive understanding of API performance capacity and support more accurate technical recommendations for digital investment applications.

4. CONCLUSION

This study evaluated the performance of the SimInvest API for stock transaction and portfolio services using Apache JMeter load testing. The findings show that the application performed acceptably under 100 and 500 concurrent users, with aggregate error rates of 4.80% and 4.48%. However, the system became unreliable when the load reached 1,000 users, as the total error rate rose to 53.17% and several account, RDN, portfolio, and order-related services failed. Under 2,000 and 4,000 users, the system showed persistent instability, with 14,711 and 39,193 failed requests respectively. The most affected services were Account Cash, Cash Withdrawal, Order List, Order Buy, Order Sell, Portfolio, RDN History, RDN Info, and Trade List. These findings answer the research objective by showing the capacity boundary and identifying the API services that require technical improvement. The recommended actions are API optimization, database-query tuning, caching, load balancing, server-capacity improvement, database replication, and real-time performance monitoring. Future studies should extend this work through stress testing, endurance testing, and resource-utilization analysis in a production-like environment so that long-duration stability and maximum failure thresholds can be examined more precisely.

REFERENCES

Abdeen, W., Chen, X., & Unterkalmsteiner, M. (2023). An approach for performance requirements verification and test environments generation. Requirements Engineering, 28(1), 117-144. https://doi.org/10.1007/s00766-022-00379-3

Amid Golmohammadi, Man Zhang, and Andrea Arcuri. 2023. Testing RESTful APIs: A Survey. ACM Trans. Softw. Eng. Methodol. 33, 1, Article 27 (January 2024), 41 pages. https://doi.org/10.1145/3617175

Apache Software Foundation. (n.d.). Apache JMeter. Apache Software Foundation.

Brzeszczyński, J., & Ibrahim, B. M. (2019). A stock market trading system based on foreign and domestic information. Expert systems with applications, 118, 381-399. https://doi.org/10.1016/j.eswa.2018.08.005

Firstian. (2023). Stress testing: Mengenal definisi, tipe dan tools stress testing. https://www.wowrack.com/id-id/blog/security-id/mengenal-definisi-tipe-dan-tools-stress-testing/

Ginasari, N. L. A. S., Wibawa, K. S., & Wirdiani, N. K. A. (2021). Pengujian stress testing API sistem pelayanan dengan Apache JMeter. Jurnal Ilmiah Teknologi dan Komputer, 2(2), 2.

GRCI. (2024). Mengenal stress test: Pengertian, tujuan, dan fungsinya bagi korporasi. https://grc-indonesia.com/mengenal-stress-test-pengertian-tujuan-dan-fungsinya-bagi-korporasi/

ICreativeLabs. (2024). Apa itu performance testing? Dan apa saja toolsnya? https://icreativelabs.com/blog/apa-itu-performance-testing-dan-apa-saja-toolsnya

Ismail, A., Ananta, A. Y., Arief, S. N., & Hamdana, E. N. (2023). Performance testing sistem ujian online menggunakan JMeter pada lingkungan virtual. Jurnal Informatika Polinema, 9(2), 159-164. https://doi.org/10.33795/jip.v9i2.1190

Karolinda, B. (2021). Implementasi pembelajaran sistem informasi manajemen di perusahaan So Good Food. EduPsyCouns: Journal of Education, Psychology and Counseling, 3(2), 63-78. https://ummaspul.e-journal.id/Edupsycouns/article/view/2370

Kurniawan, T. A., Wisjhnuadji, T. W., & Riandono, F. (2019). Implementasi Transaction Processing System Berbasis Web Dan Mobile. Jurnal Ilmiah Fakultas Teknik LIMIT'S Vol, 15(1).

Madhani, D., Darwiyanto, E., & Gandhi, A. (2023). Performance Testing Menggunakan Metode Load Testing dan Stress Testing pada Sistem Core Banking PT. XYZ. e-Proceeding of Engineering, 10(6), 5431-5441.

Nosuke. (2023). JMeter Apache: Aplikasi Java untuk menguji website. https://appkey.id/pembuatan-website/backend/jmeter-apache/

Ou, S. (2023). Portfolio optimization and analysis using modern portfolio theory. Finance & Economics, 1(3). https://doi.org/10.61173/7q7mp960

Permatasari, D. I. (2020). Pengujian aplikasi menggunakan metode load testing dengan Apache JMeter pada sistem informasi pertanian. Jurnal Sistem dan Teknologi Informasi (JUSTIN), 8(1), 135. https://doi.org/10.26418/justin.v8i1.34452

Proxsis Consulting. (2024). Kenali 5 jenis stress test yang sering digunakan dalam IT. https://strategy.proxsisgroup.com/insight/kenali-5-jenis-stress-test-yang-sering-digunakan-dalam-it/

Shariff, S. M., Li, H., Bezemer, C. P., Hassan, A. E., Nguyen, T. H. D., & Flora, P. (2019). Improving the testing efficiency of Selenium-based load tests. Proceedings - 2019 IEEE/ACM 14th International Workshop on Automation of Software Test, AST 2019, 14-20. https://doi.org/10.1109/AST.2019.00008

SimInvest Luncurkan Fitur Investasi Reksa Dana. (2023). Sinarmas. https://www.sinarmas.com/blog/?p=3747

Sinarmas Sekuritas. (2024). SimInvest.

Tejaya, W., Rahman, S., & Munir, A. (2023). Pengujian website Invitees menggunakan metode load testing dengan Apache JMeter. KHARISMA Tech, 18(1), 99-112. https://doi.org/10.55645/kharismatech.v18i1.305

Testim. (2023). What is the software testing life cycle? A complete guide. https://www.testim.io/blog/software-testing-life-cycle/

Yi Liu, Yuekang Li, Gelei Deng, Yang Liu, Ruiyuan Wan, Runchao Wu, Dandan Ji, Shiheng Xu, and Minli Bao. 2022. Morest: model-based RESTful API testing with execution feedback. In Proceedings of the 44th International Conference on Software Engineering (ICSE '22). Association for Computing Machinery, New York, NY, USA, 1406–1417. https://doi.org/10.1145/3510003.3510133