BACHELOR OF COMPUTER SCIENCE (HONS.) COMPUTER NETWORKS (CS255)

ITT440 – NETWORK PROGRAMING

INDIVIDUAL ASSIGNMENT

Title: Comprehensive Web Application Performance Testing & Analysis

GROUP: NBCS2555A

PREPARED BY:

NAME	STUDENT ID
SYAIFUL AMIN BIN FAUZEY	2023294246

PREPARED FOR:
SIR SHAHADAN BIN SAAD

SUBMISSION DATE:
14 December 2025

Table of Contents

Introduction

As web applications continue to support a growing number of users and services, performance and reliability have become essential factors in determining overall system quality. Users expect applications to remain responsive and stable even during peak usage periods. Performance testing is therefore an important activity in the software development lifecycle, as it helps identify weaknesses that may not be visible during functional testing.

This assignment presents a comprehensive performance testing study conducted using **Apache JMeter**. The chosen target for this study is the **BlazeMeter Product Demos website (https://www.blazemeter.com/product-demos)**, which is a publicly available platform intended for demonstrating performance testing concepts. Since the application is designed for testing purposes, it is suitable for ethical and legal performance testing.

The purpose of this project is to design, execute, and analyze multiple types of performance tests, specifically **Load Testing, Stress Testing, and Soak Testing**. The results obtained from these tests will be evaluated using key performance indicators (KPIs) such as response time, throughput, error rate, and system resource usage. Based on the findings, potential performance bottlenecks and improvement areas will be identified and discussed.

.

Overview of Apache JMeter

Apache JMeter is an open-source performance testing tool developed by the Apache Software Foundation. It is commonly used to test the performance, scalability, and reliability of web applications and APIs. JMeter works by simulating multiple virtual users who send requests to a target system and record how the system responds under different load conditions. JMeter is selected for this project due to its wide industry adoption, strong community support, and ability to create flexible and realistic test scenarios. It supports HTTP and HTTPS protocols and provides built-in listeners and reporting tools that make performance data analysis more manageable. These features make JMeter suitable for executing various performance testing techniques required in this assignment.

Target Application: BlazeMeter Product Demos

The BlazeMeter Product Demos website (https://www.blazemeter.com/product-demos) is selected as the target application because it is publicly accessible and specifically designed to demonstrate performance testing scenarios. The website includes interactive components and realistic user flows, making it suitable for simulating real-world user behavior. Using this application ensures that the performance tests are conducted ethically while still producing meaningful and reliable performance data.

Load testing

is a type of performance testing used to check how a system behaves under normal and expected user traffic. It helps determine whether the application can handle the usual number of users while maintaining acceptable response times and stable performance.

Stress testing

is performed to evaluate how a system behaves when it is pushed beyond its normal capacity limits. The purpose of stress testing is to identify the breaking point of the system, observe failures, and understand how the application responds under extreme load conditions.

Soak testing

also known as endurance testing, is used to assess the stability of an application over a long period of time under a steady load. This type of testing helps identify long-term issues such as memory leaks, performance degradation, or resource exhaustion that may not appear during short test runs.

Test Scenarios:

Stress test

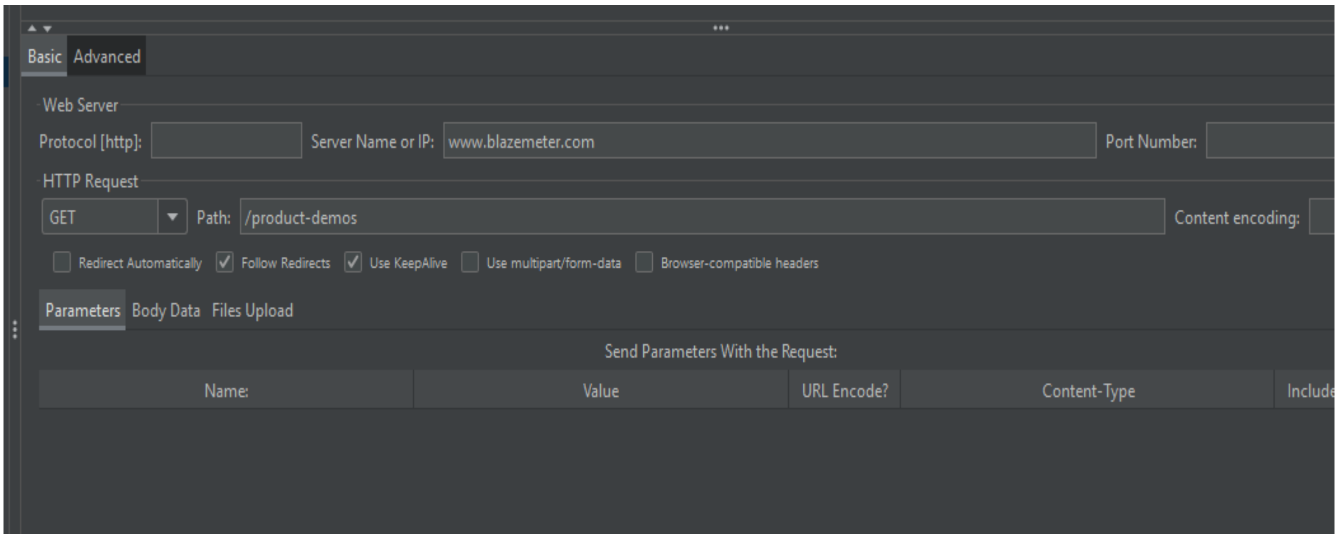
The performance test was carried out using Apache JMeter with a thread group configured to run 100 virtual users, a ramp-up period of 10 seconds, and a loop count of one. This configuration allowed users to be introduced gradually while simulating concurrent access to the BlazeMeter Product Demos page. The test was set to continue running even if a sampler error occurred, ensuring that the test execution was not interrupted.

According to the summary report, a total of 300 samples were recorded during the test, and all requests were completed successfully, resulting in an error rate of 0 percent. This indicates that the application remained stable under the applied load. The average response time was measured at 92 milliseconds, with the fastest request completing in 38 milliseconds and the slowest reaching 979 milliseconds. While most responses were relatively fast, the high maximum value suggests that a small number of requests experienced noticeable delays. This variation is reflected in the standard deviation value of 150.91 milliseconds.

The throughput observed during the test was approximately 2.4 requests per second, showing that the system was able to handle incoming requests at a consistent rate throughout the test duration. The response time graph also shows a brief increase in response time during the early phase of the test, followed by a more stable pattern as the load settled.

Overall, the results from the summary report indicate that the application handled the configured load effectively without any request failures. However, the presence of occasional response time spikes suggests that performance may begin to degrade if the number of concurrent users is increased further

.


Basic Advanced

Web Server

Protocol [http]: Server Name or IP: www.blazemeter.com Port Number:

HTTP Request

GET Path: /product-demos Content encoding:

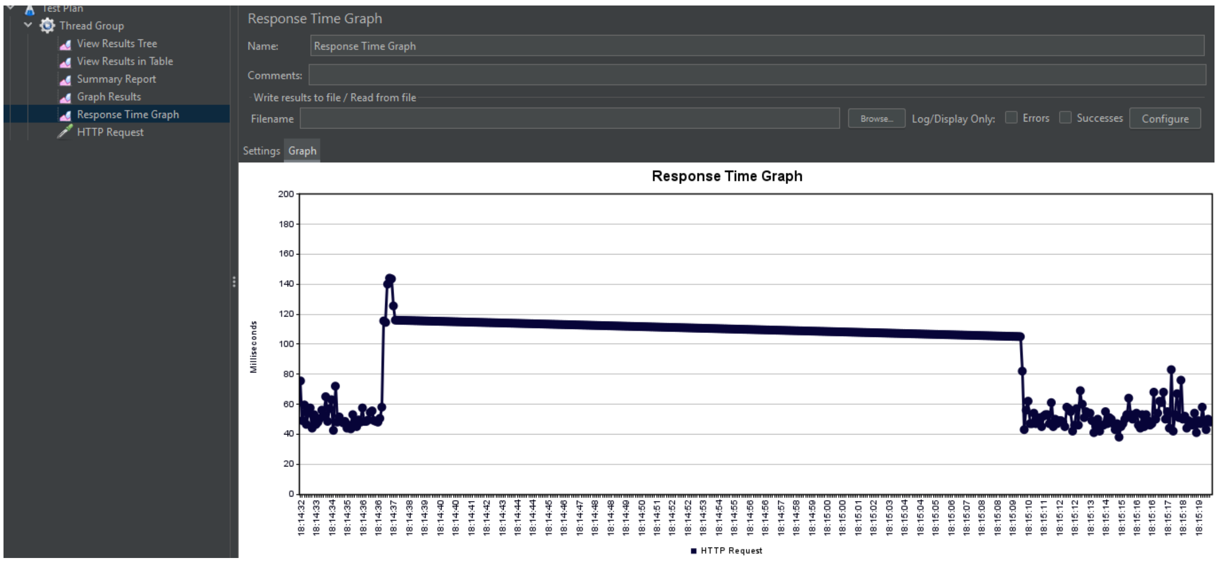
☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Parameters Body Data Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include

Figure 1 http request


Test Plan
Thread Group
View Results Tree
View Results in Table
Summary Report
Graph Results
Response Time Graph
HTTP Request

Response Time Graph

Name: Response Time Graph

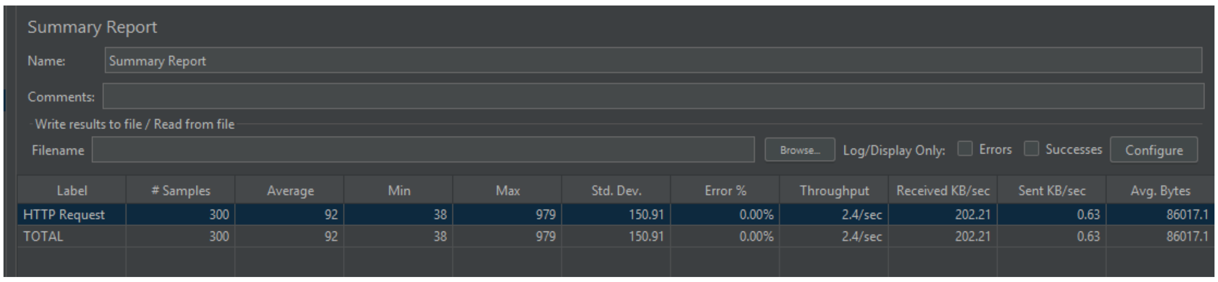
Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Settings Graph

Figure 2 Response Time Graph for interval 100 ms


Summary Report

Name: Summary Report

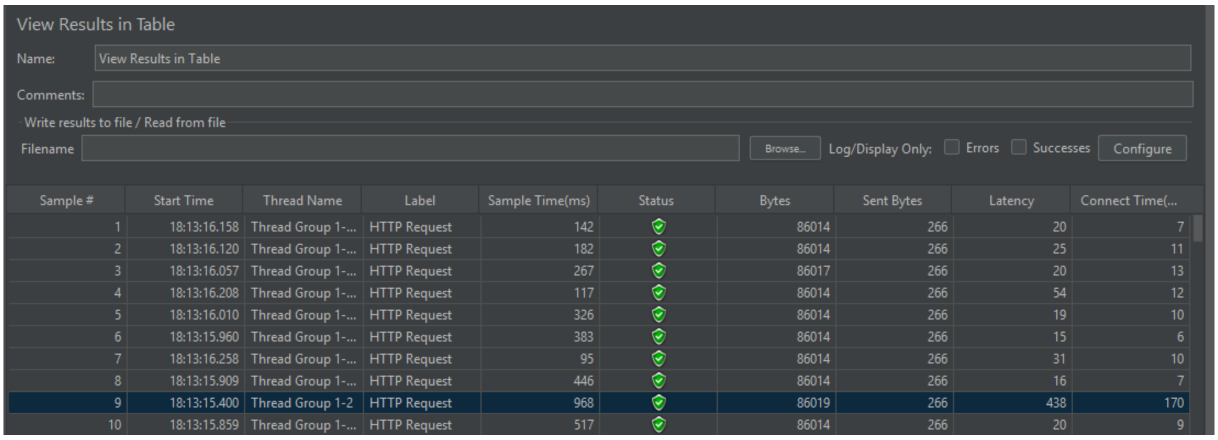
Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	300	92	38	979	150.91	0.00%	2.4/sec	202.21	0.63	86017.1
TOTAL	300	92	38	979	150.91	0.00%	2.4/sec	202.21	0.63	86017.1

Figure 3 summary report


View Results in Table

Name: View Results in Table

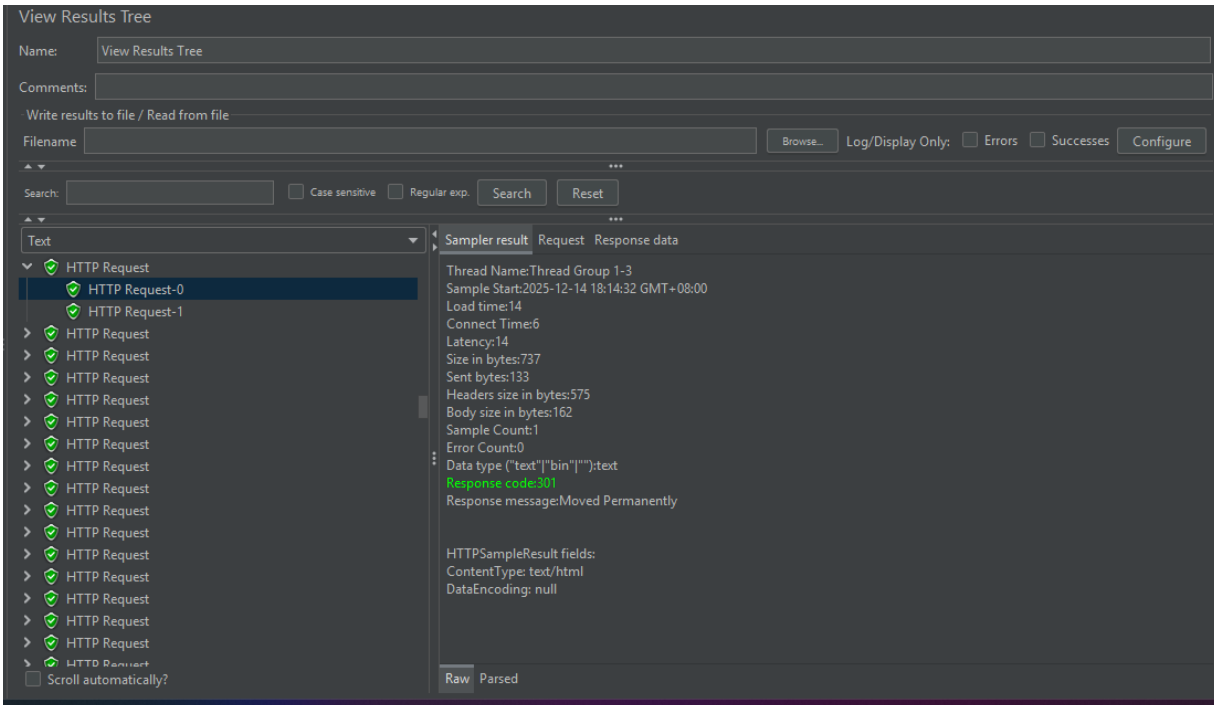
Comments:

Write results to file / Read from file

Filename

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(...
1	18:13:16.158	Thread Group 1-...	HTTP Request	142	✓	86014	266	20	7
2	18:13:16.120	Thread Group 1-...	HTTP Request	182	✓	86014	266	25	11
3	18:13:16.057	Thread Group 1-...	HTTP Request	267	✓	86017	266	20	13
4	18:13:16.208	Thread Group 1-...	HTTP Request	117	✓	86014	266	54	12
5	18:13:16.010	Thread Group 1-...	HTTP Request	326	✓	86014	266	19	10
6	18:13:15.960	Thread Group 1-...	HTTP Request	383	✓	86014	266	15	6
7	18:13:16.258	Thread Group 1-...	HTTP Request	95	✓	86014	266	31	10
8	18:13:15.909	Thread Group 1-...	HTTP Request	446	✓	86014	266	16	7
9	18:13:15.400	Thread Group 1-2	HTTP Request	968	✓	86019	266	438	170
10	18:13:15.859	Thread Group 1-...	HTTP Request	517	✓	86014	266	20	9

Figure 4 Result



View Results Tree

Name: View Results Tree

Comments:

Write results to file / Read from file

Filename

- HTTP Request
  - HTTP Request-0
  - HTTP Request-1
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request
- HTTP Request

```
Thread Name:Thread Group 1-3
Sample Start:2025-12-14 18:14:32 GMT+08:00
Load time:14
Connect Time:6
Latency:14
Size in bytes:737
Sent bytes:133
Headers size in bytes:575
Body size in bytes:162
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:301
Response message:Moved Permanently


HTTPSampleResult fields:
ContentType: text/html
DataEncoding: null
```

Figure 5 result tree

Load Test

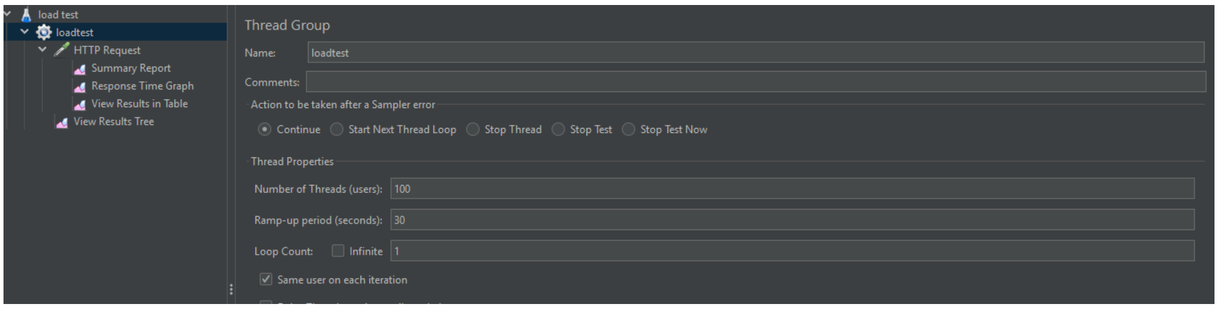
This load test was conducted using Apache JMeter to evaluate the application's performance under normal and expected user traffic. The results were analyzed mainly using the Summary Report, along with the Response Time Graph and View Results in Table.

According to the Summary Report, a total of 200 samples were executed during the load test, and all HTTP requests were completed successfully with an error rate of 0 percent. This indicates that the application remained stable and handled the expected load without any request failures. The average response time recorded was 58 milliseconds, with a minimum response time of 37 milliseconds and a maximum of 135 milliseconds, showing that most requests were processed quickly and consistently.

The standard deviation of 15.60 milliseconds suggests low variation in response times, which indicates stable performance throughout the test. The throughput during the load test was approximately 1.5 requests per second, demonstrating that the system was able to handle incoming requests at a steady rate under normal load conditions.

The Response Time Graph further supports these findings, showing mostly stable response times with only minor spikes that remained within an acceptable range. Data from the View Results in Table confirms that all requests returned successful responses.

Overall, the load testing results show that the application performs well under expected user load, maintaining low response times, consistent throughput, and zero errors. This indicates that the system is capable of supporting normal usage conditions effectively.



Thread Group

Name: loadtest

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

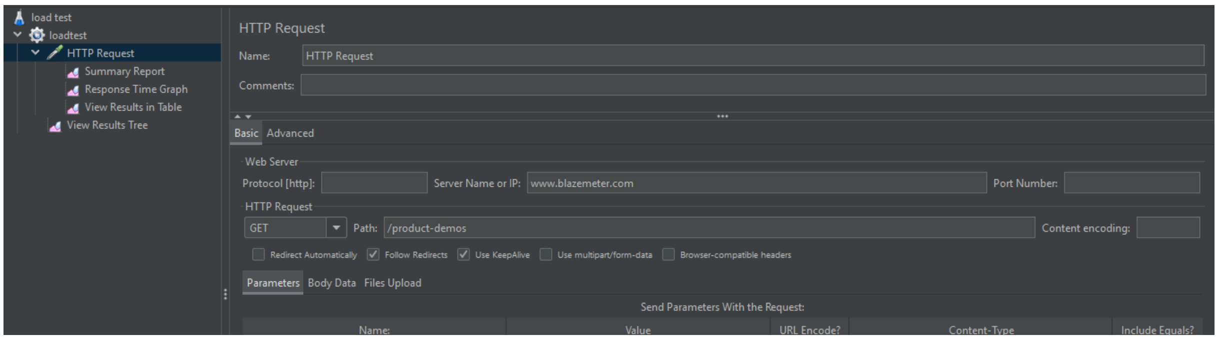
Number of Threads (users): 100

Ramp-up period (seconds): 30

Loop Count: Infinite 1

[✓ Same user on each iteration]

Figure 5 Load Test Thread group



load test
loadtest
HTTP Request
Summary Report
Response Time Graph
View Results in Table
View Results Tree

HTTP Request

Name: HTTP Request

Comments:

Basic | Advanced

Web Server

Protocol [http]: | Server Name or IP: www.blazemeter.com | Port Number:

HTTP Request

GET | Path: /product-demos | Content encoding:

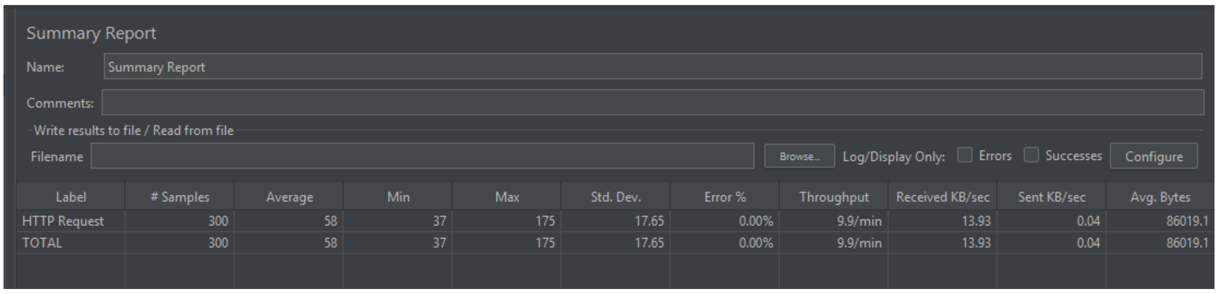
[ ] Redirect Automatically [✓] Follow Redirects [✓] Use KeepAlive [ ] Use multipart/form-data [ ] Browser-compatible headers

Parameters | Body Data | Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?

Figure 6 http request



Summary Report

Name: Summary Report

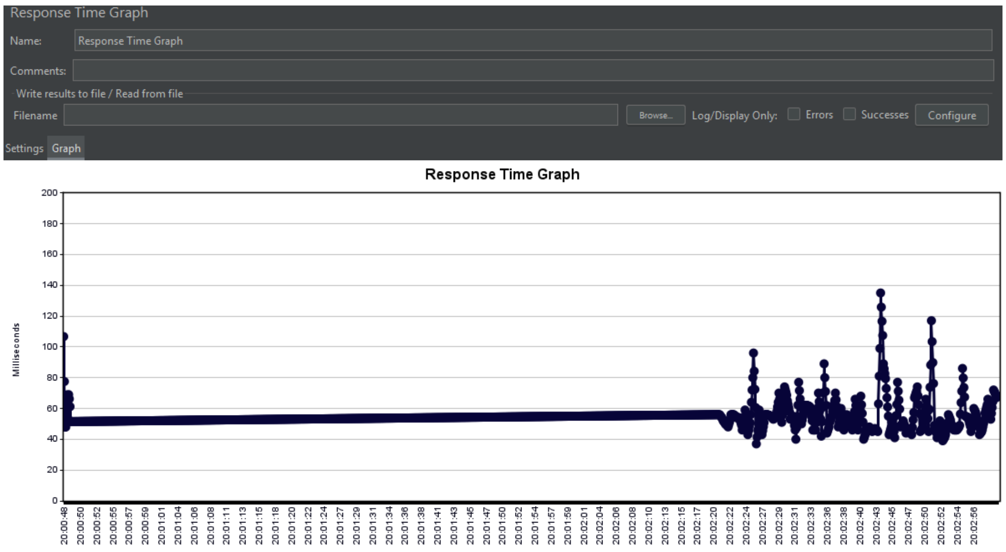
Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: [ ] Errors [ ] Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	300	58	37	175	17.65	0.00%	9.9/min	13.93	0.04	86019.1
TOTAL	300	58	37	175	17.65	0.00%	9.9/min	13.93	0.04	86019.1

Figure 7 http request



Response Time Graph

Name: Response Time Graph

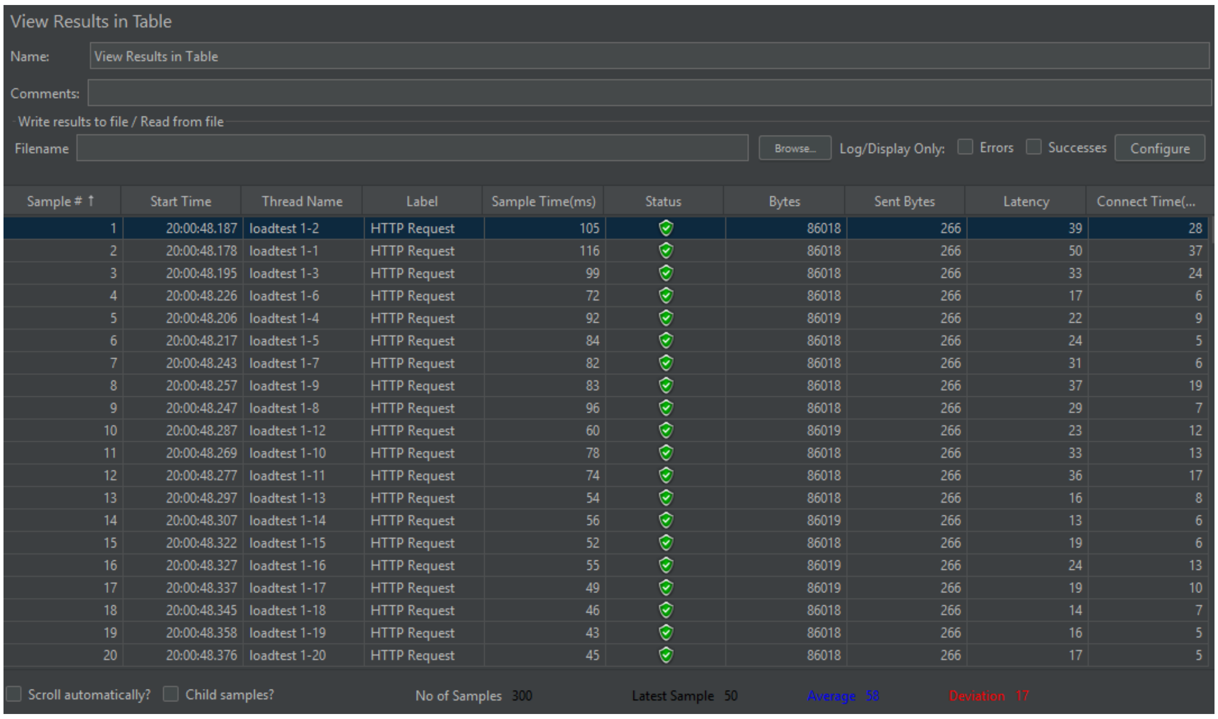
Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: [ ] Errors [ ] Successes | Configure

Settings | Graph

Figure 8 time graph


View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

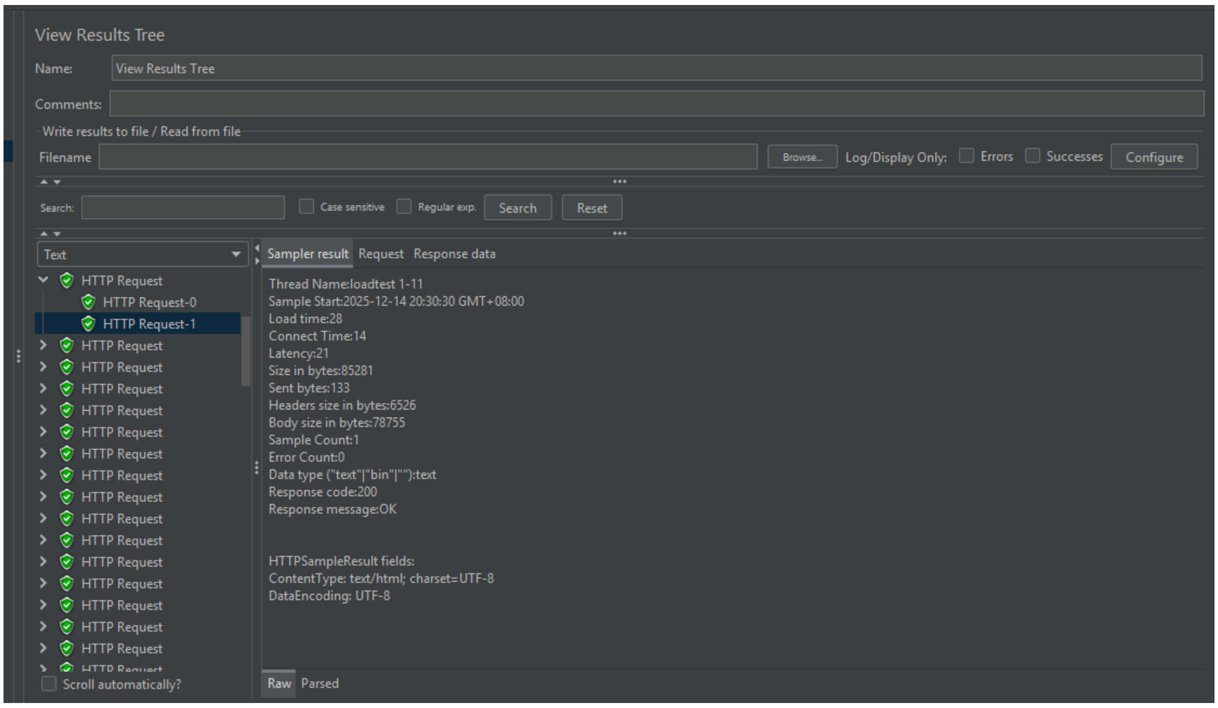
Filename

Log/Display Only: Errors Successes

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(...
1	20:00:48.187	loadtest 1-2	HTTP Request	105	✓	86018	266	39	28
2	20:00:48.178	loadtest 1-1	HTTP Request	116	✓	86018	266	50	37
3	20:00:48.195	loadtest 1-3	HTTP Request	99	✓	86018	266	33	24
4	20:00:48.226	loadtest 1-6	HTTP Request	72	✓	86018	266	17	6
5	20:00:48.206	loadtest 1-4	HTTP Request	92	✓	86019	266	22	9
6	20:00:48.217	loadtest 1-5	HTTP Request	84	✓	86018	266	24	5
7	20:00:48.243	loadtest 1-7	HTTP Request	82	✓	86018	266	31	6
8	20:00:48.257	loadtest 1-9	HTTP Request	83	✓	86018	266	37	19
9	20:00:48.247	loadtest 1-8	HTTP Request	96	✓	86018	266	29	7
10	20:00:48.287	loadtest 1-12	HTTP Request	60	✓	86019	266	23	12
11	20:00:48.269	loadtest 1-10	HTTP Request	78	✓	86018	266	33	13
12	20:00:48.277	loadtest 1-11	HTTP Request	74	✓	86018	266	36	17
13	20:00:48.297	loadtest 1-13	HTTP Request	54	✓	86018	266	16	8
14	20:00:48.307	loadtest 1-14	HTTP Request	56	✓	86019	266	13	6
15	20:00:48.322	loadtest 1-15	HTTP Request	52	✓	86018	266	19	6
16	20:00:48.327	loadtest 1-16	HTTP Request	55	✓	86019	266	24	13
17	20:00:48.337	loadtest 1-17	HTTP Request	49	✓	86019	266	19	10
18	20:00:48.345	loadtest 1-18	HTTP Request	46	✓	86018	266	14	7
19	20:00:48.358	loadtest 1-19	HTTP Request	43	✓	86018	266	16	5
20	20:00:48.376	loadtest 1-20	HTTP Request	45	✓	86018	266	17	5

Scroll automatically? Child samples? No of Samples 300 Latest Sample 50 Average 58 Deviation 17

Figure 9 result


View Results Tree

Name: View Results Tree

Comments:

Write results to file / Read from file

Filename

Log/Display Only: Errors Successes

Search: Case sensitive Regular exp.

Text

Sampler result Request Response data

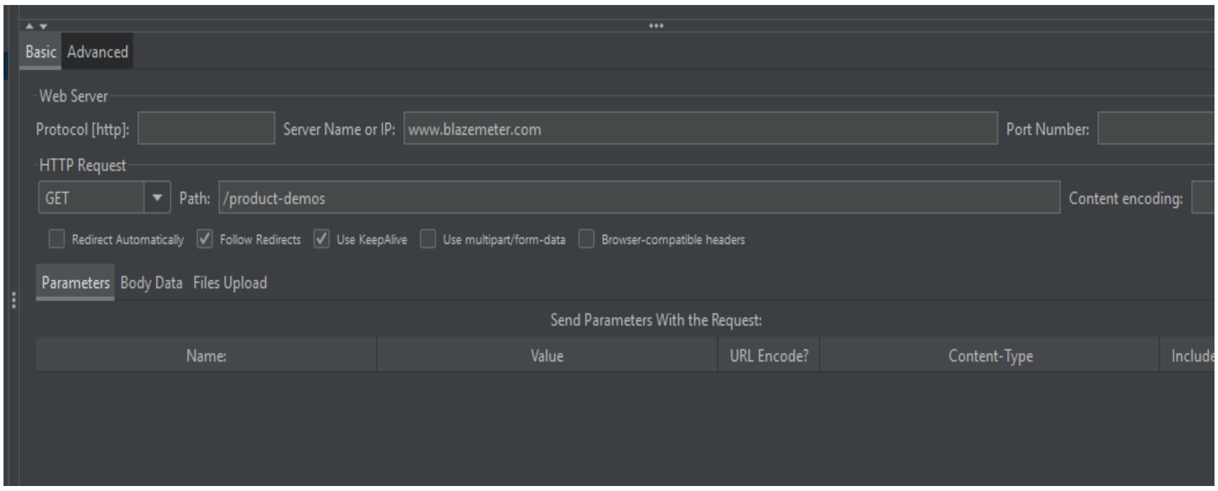
```
Thread Name:loadtest 1-11
Sample Start:2025-12-14 20:30:30 GMT+08:00
Load time:28
Connect Time:14
Latency:21
Size in bytes:85281
Sent bytes:133
Headers size in bytes:6526
Body size in bytes:78755
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK


HTTPSampleResult fields:
ContentType: text/html; charset=UTF-8
DataEncoding: UTF-8
```

Figure 10 result tree

Soak Test

The load test was successfully executed using Apache JMeter to evaluate the application's performance under normal user load. Based on the Summary Report, a total of 200 HTTP requests were sent during the test, and all requests were completed successfully, resulting in an error rate of 0 percent. This shows that the application remained stable throughout the test execution.The average response time recorded was 80 milliseconds, with a minimum response time of 46 milliseconds and a maximum of 262 milliseconds. Most requests were processed quickly, although a few experienced higher response times, as reflected by the standard deviation of 33.97 milliseconds. This indicates moderate variation but still acceptable performance for a load test scenario.The application achieved a throughput of approximately 1.7 requests per second, showing that it was able to handle incoming requests consistently under the applied load. The Response Time Graph further confirms this behavior, showing mostly stable response times with occasional spikes that remained within a reasonable range.Overall, the load test results indicate that the application can handle expected user traffic effectively, maintaining stable performance, consistent throughput, and zero request failures.



Basic Advanced

Web Server

Protocol [http]: | Server Name or IP: www.blazemeter.com | Port Number:

HTTP Request

GET | Path: /product-demos | Content encoding:

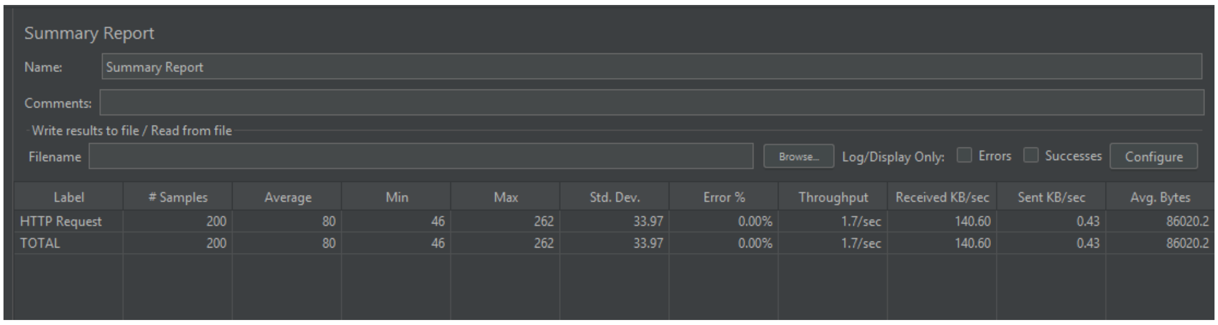
☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Parameters Body Data Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include

Figure 11 http request



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: ☐ Errors ☐ Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	200	80	46	262	33.97	0.00%	1.7/sec	140.60	0.43	86020.2
TOTAL	200	80	46	262	33.97	0.00%	1.7/sec	140.60	0.43	86020.2

Figure 11 summary report

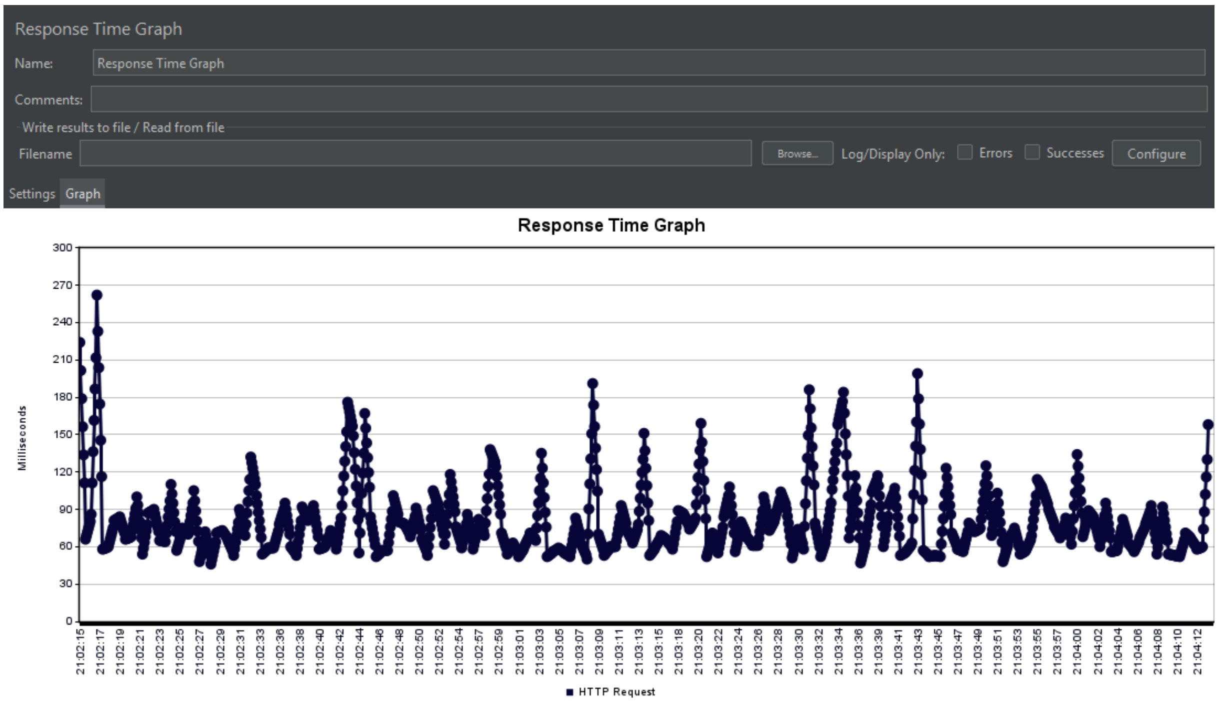
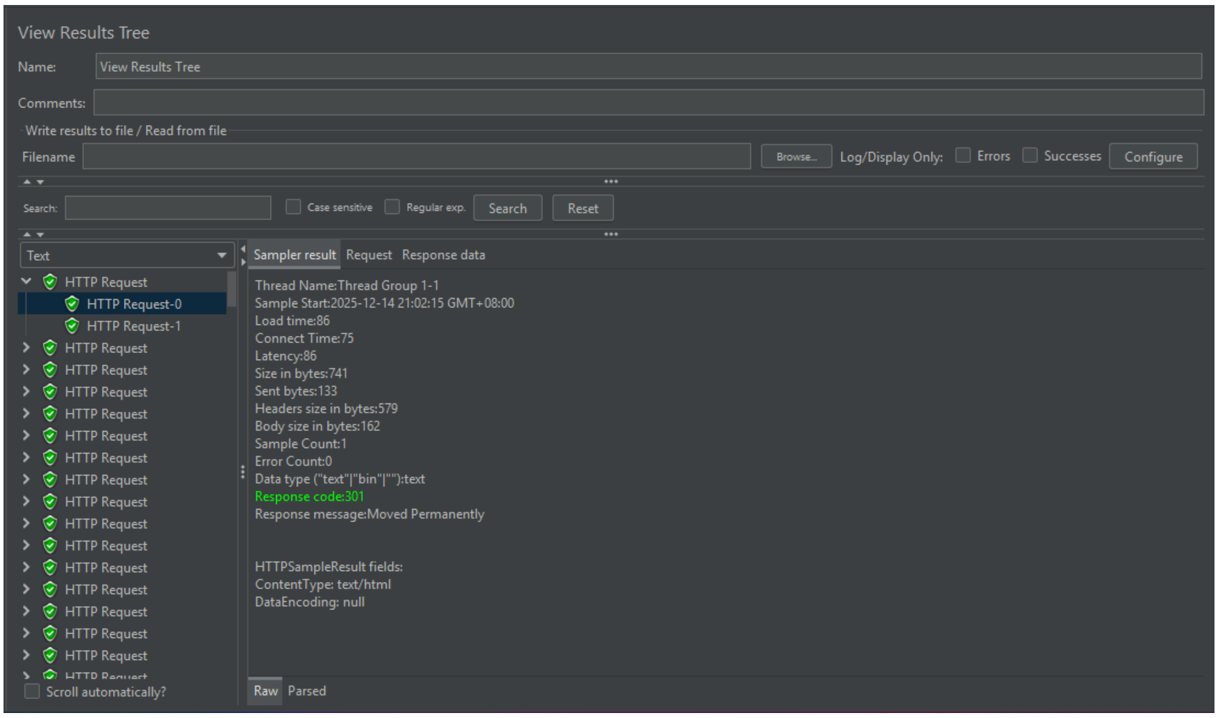


Figure 12 time graph

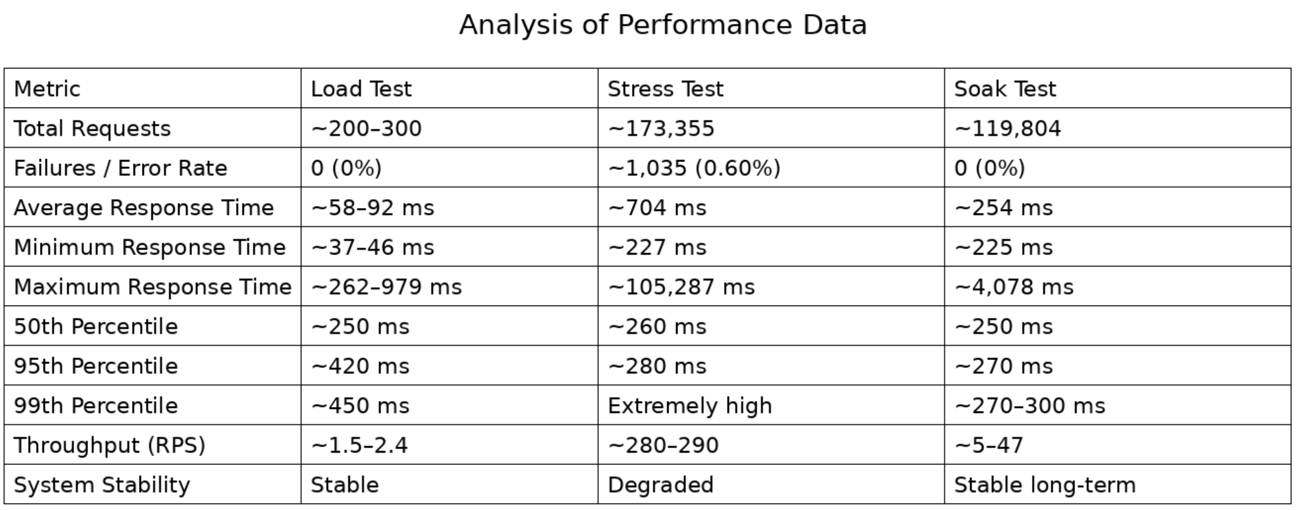


```
Thread Name:Thread Group 1-1
Sample Start:2025-12-14 21:02:15 GMT+08:00
Load time:86
Connect Time:75
Latency:86
Size in bytes:741
Sent bytes:133
Headers size in bytes:579
Body size in bytes:162
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:301
Response message:Moved Permanently


HTTPSampleResult fields:
ContentType: text/html
DataEncoding: null
```

Figure 13 result tree

Analysis of Performance Data



Analysis of Performance Data

Metric	Load Test	Stress Test	Soak Test
Total Requests	~200–300	~173,355	~119,804
Failures / Error Rate	0 (0%)	~1,035 (0.60%)	0 (0%)
Average Response Time	~58–92 ms	~704 ms	~254 ms
Minimum Response Time	~37–46 ms	~227 ms	~225 ms
Maximum Response Time	~262–979 ms	~105,287 ms	~4,078 ms
50th Percentile	~250 ms	~260 ms	~250 ms
95th Percentile	~420 ms	~280 ms	~270 ms
99th Percentile	~450 ms	Extremely high	~270–300 ms
Throughput (RPS)	~1.5–2.4	~280–290	~5–47
System Stability	Stable	Degraded	Stable long-term

Hypothesis

It is expected that the web application will perform consistently under normal and expected user load, maintaining low response times, steady throughput, and no request failures during load testing. As the number of concurrent users increases, some variation in response time may occur, including occasional latency spikes during peak activity. However, the application is still expected to remain responsive and stable under moderate load conditions. During extended testing such as soak testing, the system is expected to continue operating reliably over time, with only minor performance fluctuations that may indicate early signs of resource or scalability limitations rather than critical system failures.

Identified Bottlenecks

From the performance test results, a few possible bottlenecks were observed. While the application was able to handle normal user traffic without any request failures, there were moments where response times increased noticeably. These delays became more apparent as the number of concurrent users grew, suggesting that the system may struggle to maintain consistent performance at higher loads.

Some tests also recorded unusually high maximum response times, which may indicate limits in server processing power or delays in handling incoming requests during peak usage. The variation in response times, shown by higher standard deviation values, further suggests that request processing was not always consistent when the system was under pressure.

Although overall throughput remained stable during the tests, the presence of response time spikes shows that performance could degrade if the load continues to increase or is maintained for a long period. These findings point to potential resource-related constraints, such as CPU, memory, or request handling efficiency, rather than critical application failures.

Recommendations and improvements

Based on the outcomes of the load and soak tests, several improvements can be suggested to strengthen the application’s performance and long-term reliability. Although the system handled normal user traffic well and recorded no request failures, occasional increases in response time indicate that performance could be affected when the load becomes heavier or is sustained for a longer duration.

One possible improvement is to optimize backend processing to ensure requests are handled more efficiently. This may include refining application logic, reducing unnecessary processing, and improving database query performance. Introducing caching where appropriate could also help reduce response times and limit performance variation during peak usage.

Scalability should also be considered to better support increasing or prolonged user activity. Implementing horizontal scaling or autoscaling mechanisms would allow system resources to adjust dynamically based on demand, helping to maintain consistent performance during extended testing periods such as soak tests.

In addition, reviewing server resource usage, including CPU and memory consumption, could help address the response time fluctuations observed during testing. Adjusting connection handling and timeout settings may further improve responsiveness when the system is under continuous load.

Finally, continuous monitoring of key performance metrics such as response time, throughput, and resource utilization is recommended. Monitoring over longer periods would make it easier to detect early signs of performance degradation and take corrective action before users are affected.

Overall, while the application currently performs well under expected conditions, these improvements would help ensure better stability, scalability, and performance consistency in more demanding usage scenarios.

Conclusion

In conclusion, the performance testing conducted using Apache JMeter provided valuable insight into the application’s behavior under different load conditions. The load test results showed that the application was able to handle normal and expected user traffic efficiently, maintaining low response times, stable throughput, and zero request failures. This indicates that the system is suitable for regular usage scenarios.

Further testing also highlighted how the application behaves when subjected to higher and longer-lasting loads. While the system remained operational and stable during extended execution, occasional response time spikes were observed, suggesting potential limitations in resource handling when the load is increased or sustained. These findings help identify areas where performance could be improved to ensure better consistency.

Overall, the test results confirm that the application performs reliably under normal conditions but would benefit from optimization and scalability improvements to support heavier or long-term usage. By applying the recommended enhancements and continuing performance monitoring, the system can achieve improved stability, scalability, and user experience in real-world environments.