Documentation for JMeter Load Testing

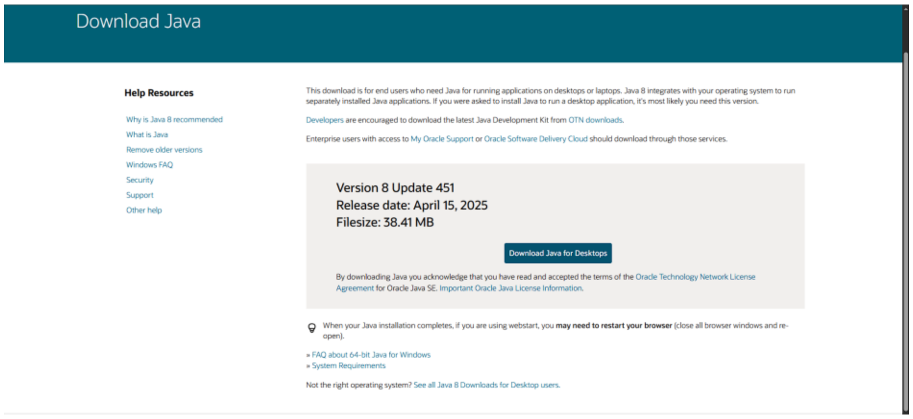
JMeter is a free application, used for load testing an application or a web site, so it can benchmark the limits while there is a big traffic at a time visiting/using the app or website. While load testing is performed, we simulate loads of traffic and typical user activities on the app/website.

There are 2 things the tester gets as feedback:

1. How the system works under loads of visitors/users
2. The number of resources used and needed

How do you set up the app, to get it ready for testing:

1. On the device you will execute the tests, you need to download Java Virtual Machine (JVM download link)



Download Java

Help Resources

- Why is Java 8 recommended
- What is Java
- Remove older versions
- Windows FAQ
- Security
- Support
- Other help

This download is for end users who need Java for running applications on desktops or laptops. Java 8 integrates with your operating system to run separately installed Java applications. If you were asked to install Java to run a desktop application, it's most likely you need this version.

Developers are encouraged to download the latest Java Development Kit from OTN downloads.

Enterprise users with access to My Oracle Support or Oracle Software Delivery Cloud should download through those services.

Version 8 Update 451
Release date: April 15, 2025
Filesize: 38.41 MB

Download Java for Desktops

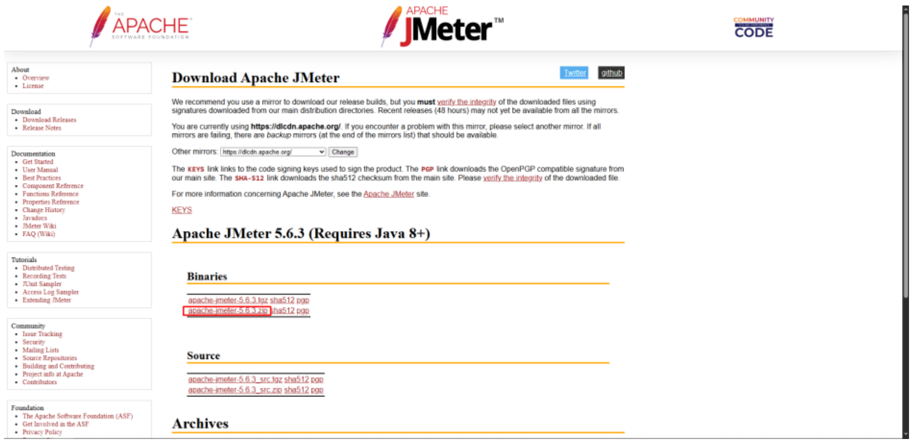
By downloading Java you acknowledge that you have read and accepted the terms of the Oracle Technology Network License Agreement for Oracle Java SE. Important Oracle Java License Information.

When your Java installation completes, if you are using webstart, you **may need to restart your browser** (close all browser windows and re-open).

» FAQ about 64-bit Java for Windows
» System Requirements

Not the right operating system? See all Java 8 Downloads for Desktop users.

2. Then you download the JMeter application from the following link (JMeter)



About
- Overview
- License

Download
- Download Releases
- Release Notes

Documentation
- Get Started
- User Manual
- Best Practices
- Component Reference
- Functions Reference
- Properties Reference
- Change History
- Javadocs
- JMeter Wiki
- FAQ (Wiki)

Tutorials
- Distributed Testing
- Recording Tests
- JUnit Sampler
- Access Log Sampler
- Extending JMeter

Community
- Issue Tracking
- Security
- Mailing Lists
- Source Repositories
- Building and Contributing
- Project info at Apache
- Contributors

Foundation
- The Apache Software Foundation (ASF)
- Get Involved in the ASF
- Privacy Policy

Download Apache JMeter

We recommend you use a mirror to download our release builds, but you **must** verify the integrity of the downloaded files using signatures downloaded from our main distribution directories. Recent releases (48 hours) may not yet be available from all the mirrors.

You are currently using **https://dlcdn.apache.org/**. If you encounter a problem with this mirror, please select another mirror. If all mirrors are failing, there are *backup* mirrors (at the end of the mirrors list) that should be available.

Other mirrors: https://dlcdn.apache.org/ Change

The KEYS link links to the code signing keys used to sign the product. The PGP link downloads the OpenPGP compatible signature from our main site. The SHA-512 link downloads the sha512 checksum from the main site. Please verify the integrity of the downloaded file.

For more information concerning Apache JMeter, see the Apache JMeter site.

KEYS

Apache JMeter 5.6.3 (Requires Java 8+)

Binaries

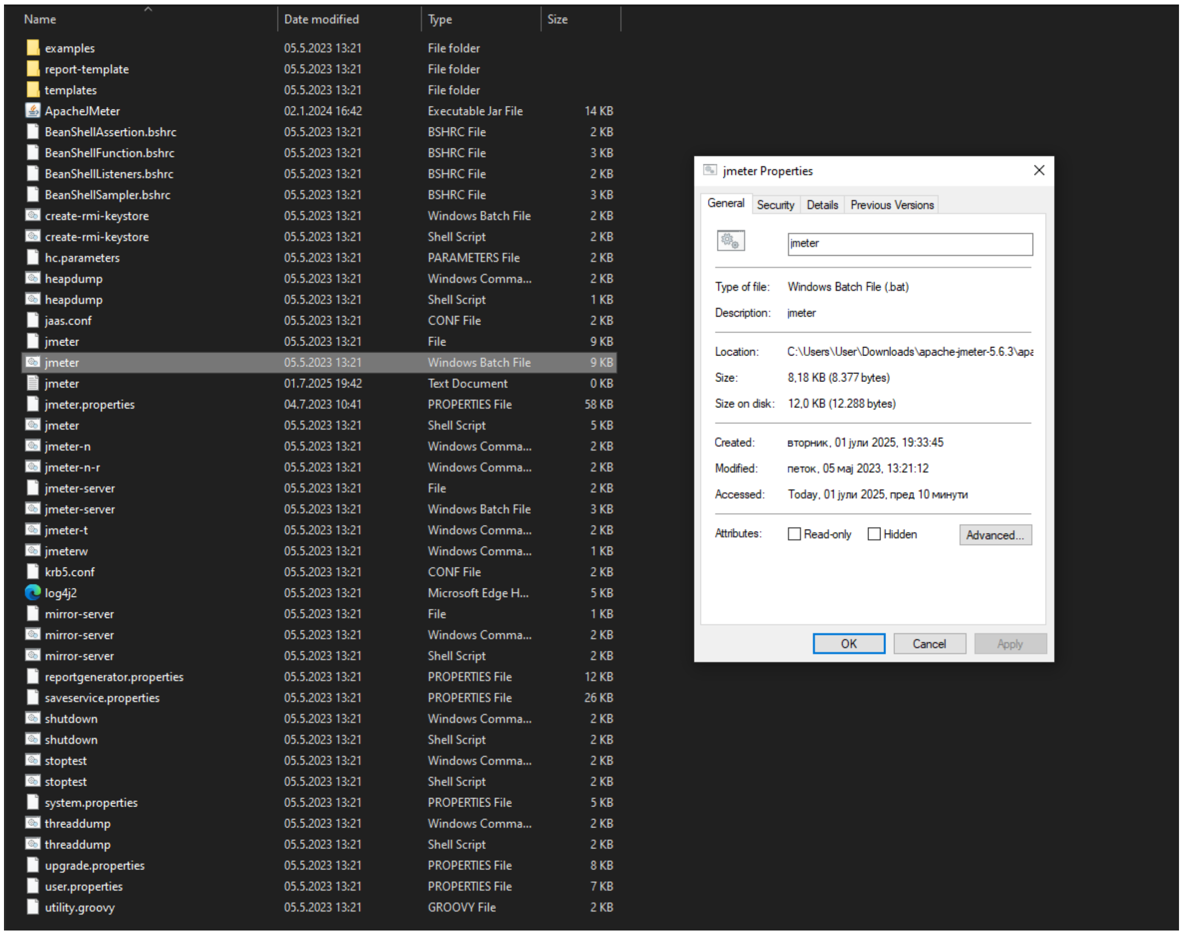
apache-jmeter-5.6.3.tgz sha512 pgp
apache-jmeter-5.6.3.zip sha512 pgp

Source

apache-jmeter-5.6.3_src.tgz sha512 pgp
apache-jmeter-5.6.3_src.zip sha512 pgp

Archives

3. After downloading the .zip file, you need to extract it first, then go to bin and we look for jmeter.bat. If the extension is not written, you can do it by clicking right click to the files and check the extension in their properties.



Name	Date modified	Type	Size
examples	05.5.2023 13:21	File folder	
report-template	05.5.2023 13:21	File folder	
templates	05.5.2023 13:21	File folder	
ApacheJMeter	02.1.2024 16:42	Executable Jar File	14 KB
BeanShellAssertion.bshrc	05.5.2023 13:21	BSHRC File	2 KB
BeanShellFunction.bshrc	05.5.2023 13:21	BSHRC File	3 KB
BeanShellListeners.bshrc	05.5.2023 13:21	BSHRC File	2 KB
BeanShellSampler.bshrc	05.5.2023 13:21	BSHRC File	3 KB
create-rmi-keystore	05.5.2023 13:21	Windows Batch File	2 KB
create-rmi-keystore	05.5.2023 13:21	Shell Script	2 KB
hc.parameters	05.5.2023 13:21	PARAMETERS File	2 KB
heapdump	05.5.2023 13:21	Windows Comma...	2 KB
heapdump	05.5.2023 13:21	Shell Script	1 KB
jaas.conf	05.5.2023 13:21	CONF File	2 KB
jmeter	05.5.2023 13:21	File	9 KB
jmeter	05.5.2023 13:21	Windows Batch File	9 KB
jmeter	01.7.2025 19:42	Text Document	0 KB
jmeter.properties	04.7.2023 10:41	PROPERTIES File	58 KB
jmeter	05.5.2023 13:21	Shell Script	5 KB
jmeter-n	05.5.2023 13:21	Windows Comma...	2 KB
jmeter-n-r	05.5.2023 13:21	Windows Comma...	2 KB
jmeter-server	05.5.2023 13:21	File	2 KB
jmeter-server	05.5.2023 13:21	Windows Batch File	3 KB
jmeter-t	05.5.2023 13:21	Windows Comma...	2 KB
jmeterw	05.5.2023 13:21	Windows Comma...	1 KB
krb5.conf	05.5.2023 13:21	CONF File	2 KB
log4j2	05.5.2023 13:21	Microsoft Edge H...	5 KB
mirror-server	05.5.2023 13:21	File	1 KB
mirror-server	05.5.2023 13:21	Windows Comma...	2 KB
mirror-server	05.5.2023 13:21	Shell Script	2 KB
reportgenerator.properties	05.5.2023 13:21	PROPERTIES File	12 KB
saveservice.properties	05.5.2023 13:21	PROPERTIES File	26 KB
shutdown	05.5.2023 13:21	Windows Comma...	2 KB
shutdown	05.5.2023 13:21	Shell Script	2 KB
stoptest	05.5.2023 13:21	Windows Comma...	2 KB
stoptest	05.5.2023 13:21	Shell Script	2 KB
system.properties	05.5.2023 13:21	PROPERTIES File	5 KB
threaddump	05.5.2023 13:21	Windows Comma...	2 KB
threaddump	05.5.2023 13:21	Shell Script	2 KB
upgrade.properties	05.5.2023 13:21	PROPERTIES File	8 KB
user.properties	05.5.2023 13:21	PROPERTIES File	7 KB
utility.groovy	05.5.2023 13:21	GROOVY File	2 KB

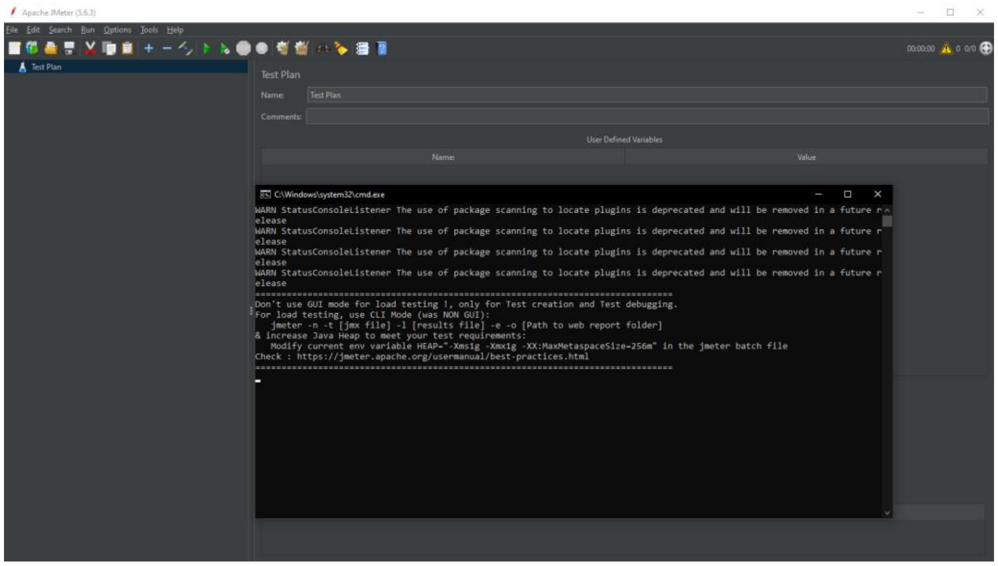
jmeter Properties

General | Security | Details | Previous Versions

jmeter

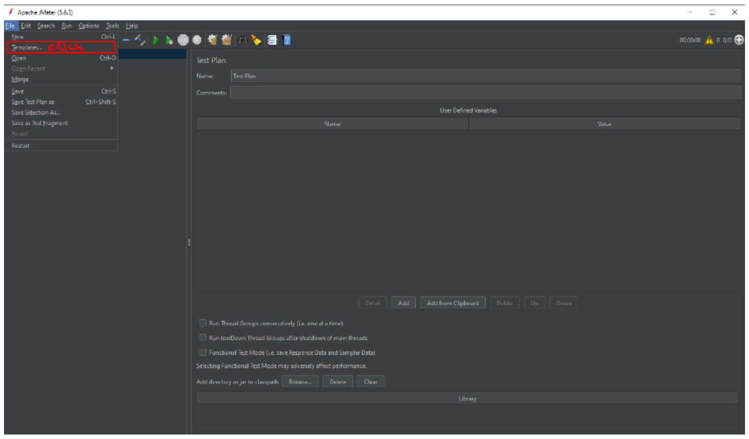
Type of file:	Windows Batch File (.bat)
Description:	jmeter
Location:	C:\Users\User\Downloads\apache-jmeter-5.6.3\apa
Size:	8,18 KB (8.377 bytes)
Size on disk:	12,0 KB (12.288 bytes)
Created:	вторник, 01 јули 2025, 19:33:45
Modified:	петок, 05 мај 2023, 13:21:12
Accessed:	Today, 01 јули 2025, пред 10 минути
Attributes:	☐ Read-only ☐ Hidden

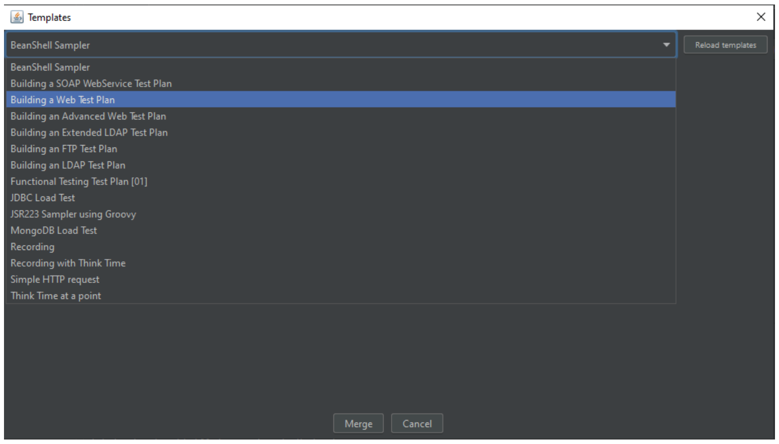
4. After the file is found, double click it and you are going to see a terminal in one window, and in another window the app GUI.



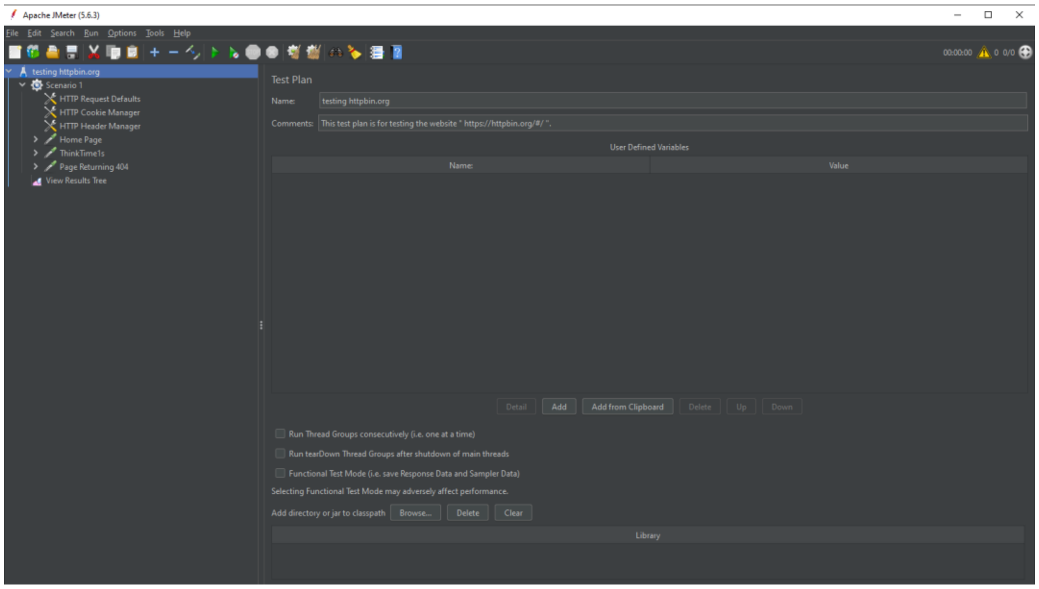
```
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
================================================================================
Don't use GUI mode for load testing !, only for Test creation and Test debugging.
For load testing, use CLI Mode (was NON GUI):
   jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report folder]
& increase Java Heap to meet your test requirements:
   Modify current env variable HEAP="-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m" in the jmeter batch file
Check : https://jmeter.apache.org/usermanual/best-practices.html
================================================================================
```

5. Now the application is runed, and the testing can start preparing. The first thing after the app is runed, need to be the 'Test Plan Building'




Templates
BeanShell Sampler
Reload templates
BeanShell Sampler
Building a SOAP WebService Test Plan
Building a Web Test Plan
Building an Advanced Web Test Plan
Building an Extended LDAP Test Plan
Building an FTP Test Plan
Building an LDAP Test Plan
Functional Testing Test Plan [01]
JDBC Load Test
JSR223 Sampler using Groovy
MongoDB Load Test
Recording
Recording with Think Time
Simple HTTP request
Think Time at a point
Merge
Cancel

6. Now the test plan is created, named with a descriptive name and the scenarios writing can start being written.



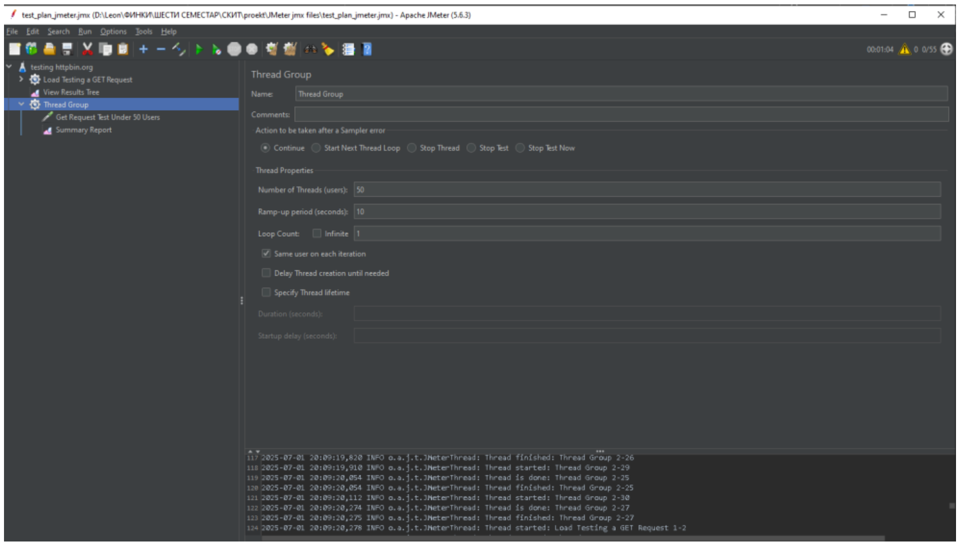
Name: testing httpbin.org

Comments: This test plan is for testing the website " https://httpbin.org/#/ ".

Test scenarios describe specific actions or use cases that need to be simulated, so it can give feedback about the system load performance. They help understanding how the application behaves under certain conditions.

In the context of JMeter, a test scenario includes details like the number of virtual users, the type of HTTP request (GET, POST, etc.), the target URL or endpoint, data input, and expected results. Scenarios are created using Thread Groups, HTTP Samplers, etc....

Scenario1: Load testing a /get request


test_plan_jmeter.jmx (D:\Leon\ФИНКИ\ШЕСТИ СЕМЕСТАР\СКИТ\proekt\JMeter jmx files\test_plan_jmeter.jmx) - Apache JMeter (5.6.3)
File Edit Search Run Options Tools Help
00:01:04 0 0/55
testing httpbin.org
Load Testing a GET Request
View Results Tree
Thread Group
Get Request Test Under 50 Users
Summary Report

Thread Group

Name: Thread Group

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): 50

Ramp-up period (seconds): 10

Loop Count: Infinite ☐ 1

☑ Same user on each iteration

☐ Delay Thread creation until needed

☐ Specify Thread lifetime

Duration (seconds):

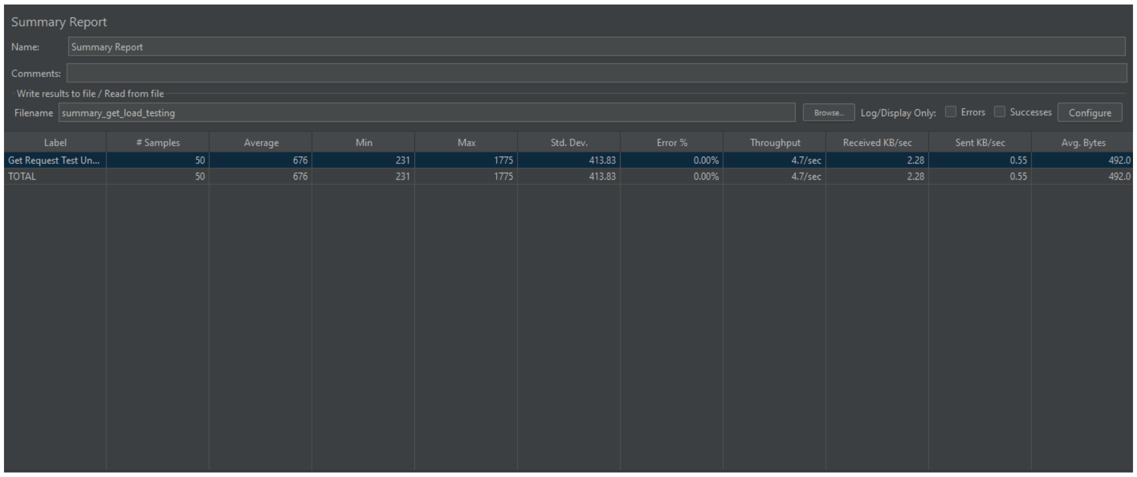
Startup delay (seconds):

```
2025-07-01 20:09:19,820 INFO o.a.j.t.JMeterThread: Thread finished: Thread Group 2-26
2025-07-01 20:09:19,910 INFO o.a.j.t.JMeterThread: Thread started: Thread Group 2-29
2025-07-01 20:09:20,054 INFO o.a.j.t.JMeterThread: Thread is done: Thread Group 2-25
2025-07-01 20:09:20,054 INFO o.a.j.t.JMeterThread: Thread finished: Thread Group 2-25
2025-07-01 20:09:20,112 INFO o.a.j.t.JMeterThread: Thread started: Thread Group 2-30
2025-07-01 20:09:20,274 INFO o.a.j.t.JMeterThread: Thread is done: Thread Group 2-27
2025-07-01 20:09:20,275 INFO o.a.j.t.JMeterThread: Thread finished: Thread Group 2-27
2025-07-01 20:09:20,278 INFO o.a.j.t.JMeterThread: Thread started: Load Testing a GET Request 1-2
```

This Thread Group will be used for the next //TO_FIX number of scenarios.

Explanation of the setup:

So, JMeter has 50 virtual users (Number of Threads), then 50 users, over 10 seconds (Ramp-Up), it means that 50/10 -> the test will start 5 users per second. And there is no infinite loop, besides there is 1 request per user, not more.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename summary_get_load_testing

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get Request Test Un...	50	676	231	1775	413.83	0.00%	4.7/sec	2.28	0.55	492.0
TOTAL	50	676	231	1775	413.83	0.00%	4.7/sec	2.28	0.55	492.0

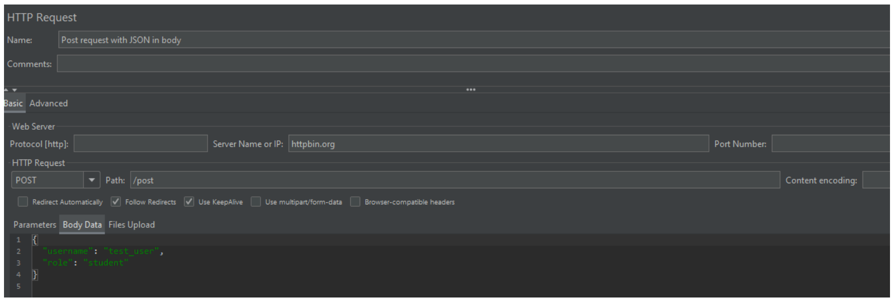
Explanation of the test result:

Server handled 50 /get requests with no errors. The average response time is 676ms, which is decent. The response time range is [231,1775] ms, meaning that some spikes occurred, likely due to simulated concurrency. The throughput of 4.7/sec is normal given your ramp-up of 10 seconds for 50 users.

This is a basic load test to measure server performance under light concurrent traffic.

Scenario2: /post request with JSON body

This scenario, we create a post request, with a body of a JSON string.



HTTP Request

Name: Post request with JSON in body

Comments:

Basic | Advanced

Web Server

Protocol [http]: | Server Name or IP: httpbin.org | Port Number:

HTTP Request

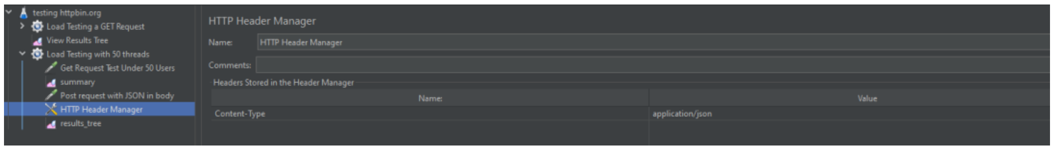
POST | Path: /post | Content encoding:

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

Parameters | Body Data | Files Upload

```
{
  "username": "test_user",
  "role": "student"
}
```

Then we add Http Header, because we need to specify the content we we are putting in the body.


testing httpbin.org
Load Testing a GET Request
View Results Tree
Load Testing with 50 threads
Get Request Test Under 50 Users
summary
Post request with JSON in body
HTTP Header Manager
results_tree

HTTP Header Manager

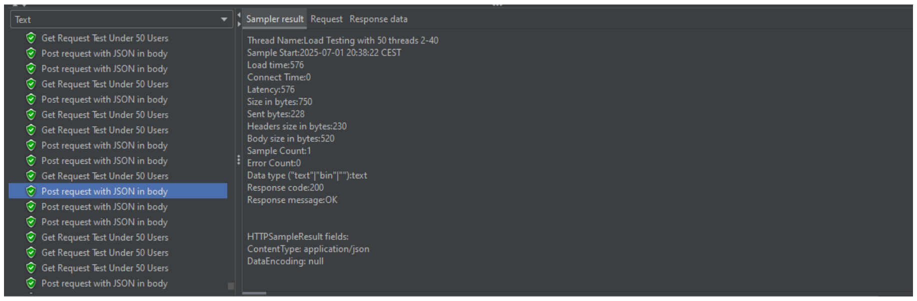
Name: HTTP Header Manager

Comments:

Headers Stored in the Header Manager

Name:	Value
Content-Type	application/json

We add a tree result, where we can see for each thread the request it is making, how the response is.

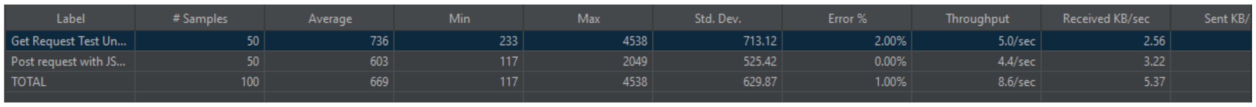

Text
Get Request Test Under 50 Users
Post request with JSON in body
Post request with JSON in body
Get Request Test Under 50 Users
Post request with JSON in body
Get Request Test Under 50 Users
Get Request Test Under 50 Users
Post request with JSON in body
Post request with JSON in body
Get Request Test Under 50 Users
Post request with JSON in body
Post request with JSON in body
Post request with JSON in body
Get Request Test Under 50 Users
Get Request Test Under 50 Users
Get Request Test Under 50 Users
Post request with JSON in body
Sampler result
Request
Response data

```
Thread Name:Load Testing with 50 threads 2-40
Sample Start:2025-07-01 20:38:22 CEST
Load time:576
Connect Time:0
Latency:576
Size in bytes:750
Sent bytes:228
Headers size in bytes:230
Body size in bytes:520
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK


HTTPSampleResult fields:
ContentType: application/json
DataEncoding: null
```

Now the testing is started again, under the same thread group configuration as before.

Here there are the results:



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/
Get Request Test Un...	50	736	233	4538	713.12	2.00%	5.0/sec	2.56	
Post request with JS...	50	603	117	2049	525.42	0.00%	4.4/sec	3.22	
TOTAL	100	669	117	4538	629.87	1.00%	8.6/sec	5.37	

Explanation of the test result:

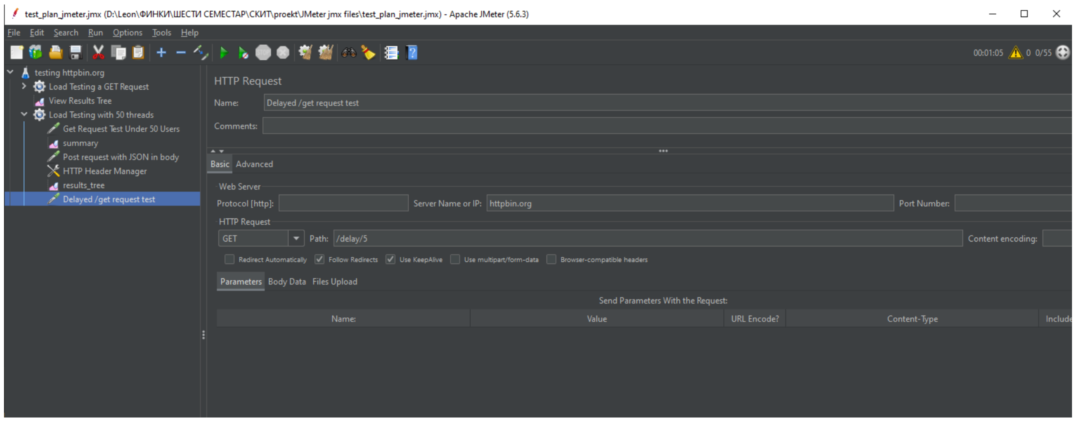
The POST request with the provided JSON string in the body sent 50 post requests with to the /post endpoint of httpbin.org. The goal was to check if the server received and echoed the JSON correctly, and to observe performance under a light load. The average response time was 603ms, which was slightly faster than the previous GET request test This test validated the functional correctness and stability of POST request handling under light load.

Scenario3: simulating slow API response

The third scenario is made to test slow API response, using a delay of 5 seconds for a /get request. We set the test to https://httpbin.org/delay/5, and we need to check if there is going to be a delay of 5 seconds, and then the response.

Simply, it allows us to evaluate how well our system can tolerate such latency and still function correctly.

Here is the setup:



test_plan_jmeter.jmx (D:\Leon\ФИНКИ\ШЕСТИ СЕМЕСТАР\СКИТ\proekt\JMeter jmx files\test_plan_jmeter.jmx) - Apache JMeter (5.6.3)

- testing httpbin.org
  - Load Testing a GET Request
  - View Results Tree
  - Load Testing with 50 threads
    - Get Request Test Under 50 Users
    - summary
    - Post request with JSON in body
    - HTTP Header Manager
    - results_tree
    - Delayed /get request test

HTTP Request

Name: Delayed /get request test

Comments:

Web Server

Protocol [http]:

Server Name or IP: httpbin.org

Port Number:

HTTP Request

GET

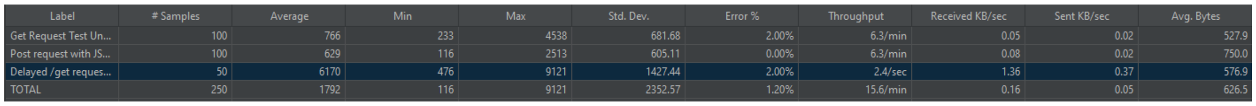
Path: /delay/5

Content encoding:

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

Send Parameters With the Request:

Here is the result:



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get Request Test Un...	100	766	233	4538	681.68	2.00%	6.3/min	0.05	0.02	527.9
Post request with JS...	100	629	116	2513	605.11	0.00%	6.3/min	0.08	0.02	750.0
Delayed /get reques...	50	6170	476	9121	1427.44	2.00%	2.4/sec	1.36	0.37	576.9
TOTAL	250	1792	116	9121	2352.57	1.20%	15.6/min	0.16	0.05	626.5

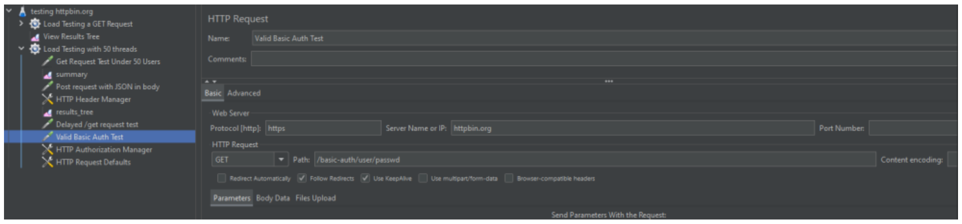
Explanation of the test result:

We sent 50 requests with a moderate number of users. The average response time was 6170 ms, which aligns with the expected delay. The system experienced 2% of failed requests, which could be caused by timeouts or temporary overload. The test helps simulate real-world latency and observe how the system performs under delayed API responses. The throughput dropped to 2.4/sec, demonstrating that slow endpoints reduce overall performance.

Scenario4: Basic authentication test

This scenario is meant to see if the login is successful/unsuccessful. It means that, if the login is successful, it will return status code 200 OK, otherwise 401 Unauthorized.

We need to set a new request, where we set the path.



testing httpbin.org
- Load Testing a GET Request
- View Results Tree
- Load Testing with 50 threads
  - Get Request Test Under 50 Users
  - summary
  - Post request with JSON in body
  - HTTP Header Manager
  - results_tree
  - Delayed /get request test
  - Valid Basic Auth Test
  - HTTP Authorization Manager
  - HTTP Request Defaults

HTTP Request

Name: Valid Basic Auth Test

Comments:

Basic | Advanced

Web Server — Protocol [http]: https — Server Name or IP: httpbin.org — Port Number:

HTTP Request — GET — Path: /basic-auth/user/passwd — Content encoding:

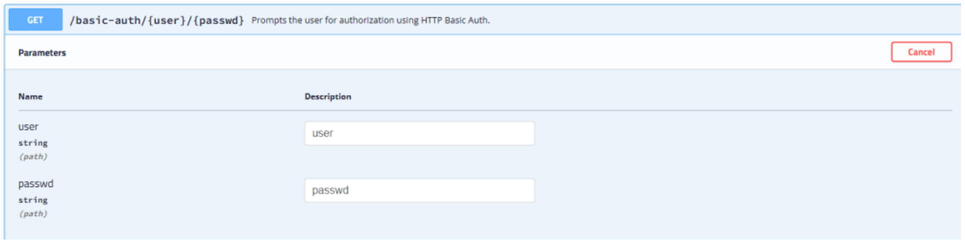
[ ] Redirect Automatically [✓] Follow Redirects [✓] Use KeepAlive [ ] Use multipart/form-data [ ] Browser-compatible headers

Parameters | Body Data | Files Upload

Send Parameters With the Request:

We set a Http Request Defaults, so it can use the https protocol, for more safety, and I set the domain httpbin.org, so it can be the same for all requests.

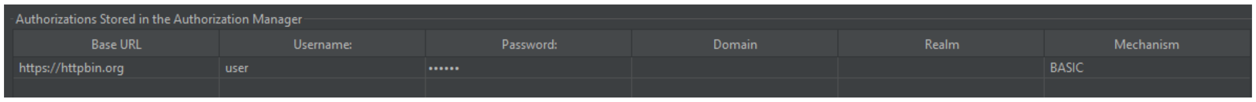
Then we set a new file, called Http Authentication Manager, where we set the password and the username that need to be provided to the searched domain/path, so it can be logged in.



GET /basic-auth/{user}/{passwd} Prompts the user for authorization using HTTP Basic Auth.

Parameters

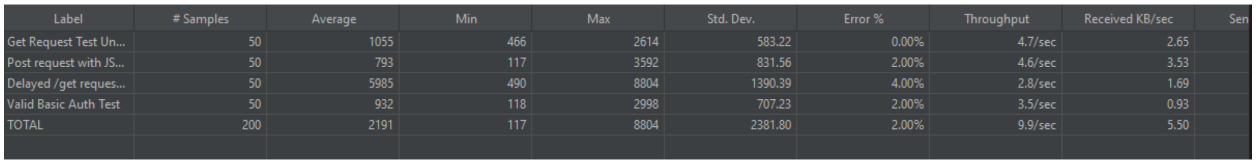
Name	Description
user string *(path)*	user
passwd string *(path)*	passwd



Authorizations Stored in the Authorization Manager

Base URL	Username:	Password:	Domain	Realm	Mechanism
https://httpbin.org	user	••••••			BASIC

And after testing, we got the following results:



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sen
Get Request Test Un...	50	1055	466	2614	583.22	0.00%	4.7/sec	2.65	
Post request with JS...	50	793	117	3592	831.56	2.00%	4.6/sec	3.53	
Delayed /get reques...	50	5985	490	8804	1390.39	4.00%	2.8/sec	1.69	
Valid Basic Auth Test	50	932	118	2998	707.23	2.00%	3.5/sec	0.93	
TOTAL	200	2191	117	8804	2381.80	2.00%	9.9/sec	5.50	

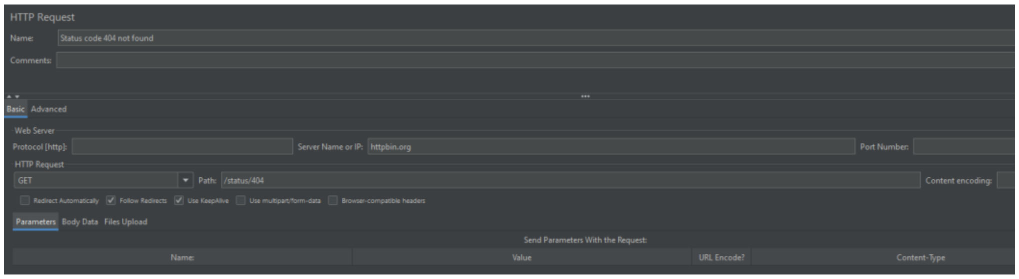
Explanation of the test result:

In this scenario, we tested HTTP Basic Authentication by sending a GET request to /basic-auth/user/passwd with valid credentials. We used JMeter's HTTP Authorization Manager to provide the username and password. Out of 50 requests, 49 completed successfully (98%), and 1 request failed (2%) *likely due to temporary connection issues*. The test successfully demonstrated correct server-side authentication handling.

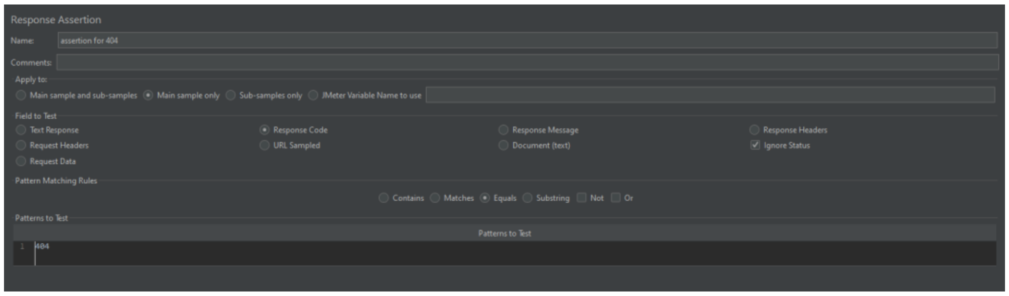
Scenario5: Testing different http status codes

Here in this scenario, we made testing on some status codes. First, and most important thing, we create the http requests for all of them separate. We choose to test 200, 404, 500 status codes. Then on the request we create an assertion that is going to make a test check if the test code we got, as the one we defined (the code we expect).

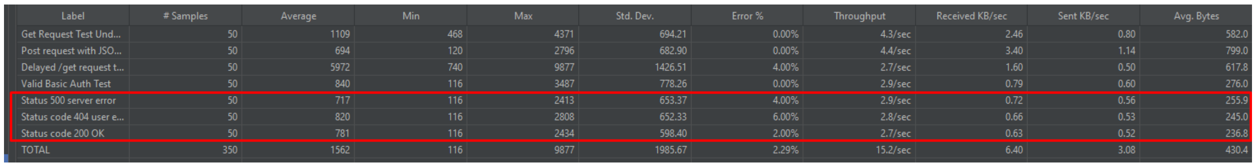
Here we will show just one request and assertion, for example let it be 404 Not Found (the others are the same, but with different code)


HTTP Request
Name: Status code 404 not found
Comments:
Basic | Advanced
Web Server
Protocol [http]: | Server Name or IP: httpbin.org | Port Number:
HTTP Request
GET | Path: /status/404 | 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


Response Assertion
Name: assertion for 404
Comments:
Apply to:
Main sample and sub-samples [✓ Main sample only] Sub-samples only JMeter Variable Name to use
Field to Test
Text Response [✓ Response Code] Response Message Response Headers
Request Headers URL Sampled Document (text) [✓ Ignore Status]
Request Data
Pattern Matching Rules
Contains Matches [✓ Equals] Substring Not Or
Patterns to Test
Patterns to Test
404

Note: For the test to be evaluated as true or false, we need to check Ignore Status in the assertion, otherwise we will always get error when we set error codes (the point of the scenario is to check if the status can be accessed).



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get Request Test Und...	50	1109	468	4371	694.21	0.00%	4.3/sec	2.46	0.80	582.0
Post request with JSO...	50	694	120	2796	682.90	0.00%	4.4/sec	3.40	1.14	799.0
Delayed /get request t...	50	5972	740	9877	1426.51	4.00%	2.7/sec	1.60	0.50	617.8
Valid Basic Auth Test	50	840	116	3487	778.26	0.00%	2.9/sec	0.79	0.60	276.0
Status 500 server error	50	717	116	2413	653.37	4.00%	2.9/sec	0.72	0.56	255.9
Status code 404 user e...	50	820	116	2808	652.33	6.00%	2.8/sec	0.66	0.53	245.0
Status code 200 OK	50	781	116	2434	598.40	2.00%	2.7/sec	0.63	0.52	236.8
TOTAL	350	1562	116	9877	1985.67	2.29%	15.2/sec	6.40	3.08	430.4

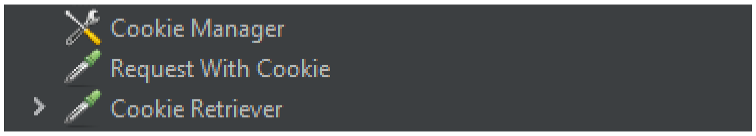
Explanation of the test result:

This scenario confirms that the application consistently handles various HTTP response codes as expected, which is essential for robust error handling, client-side logic, and backend monitoring. The minor error percentages observed may be attributed to differences between the expected and actual status codes, caused by transient server or client-side issues during high-load conditions.

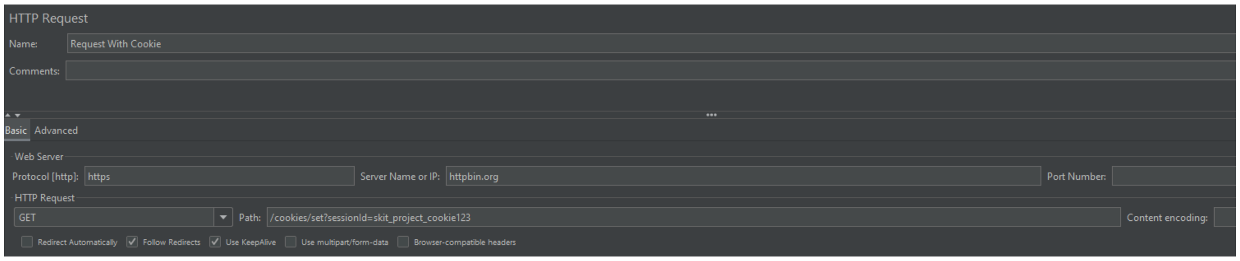
Scenario6: Cookie handling test

This test is going to check if a set cookie, can be recognized and retrieved.

We create 3 files now:



The cookie manager allows storing and sending cookies (the communication between them)



HTTP Request

Name: Request With Cookie

Comments:

Basic | Advanced

Web Server

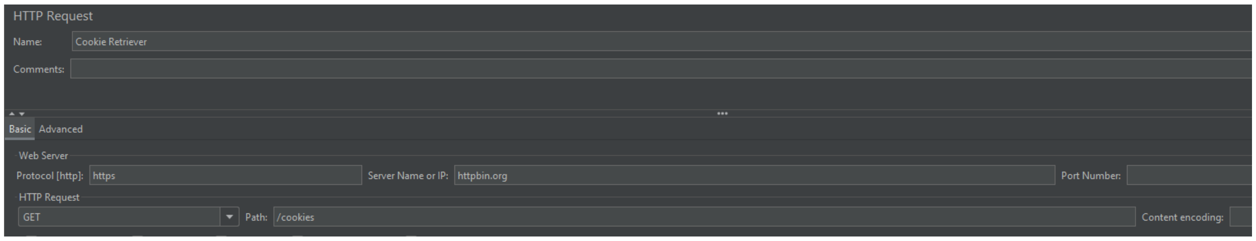
Protocol [http]: https   Server Name or IP: httpbin.org   Port Number:

HTTP Request

GET   Path: /cookies/set?sessionId=skit_project_cookie123   Content encoding:

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

Then we got the request with cookie, here we are creating a simple get request where we set?sessionId=skit_project_cookie123, so it is a trigger to the server for setting a cookie.



HTTP Request

Name: Cookie Retriever

Comments:

Basic | Advanced

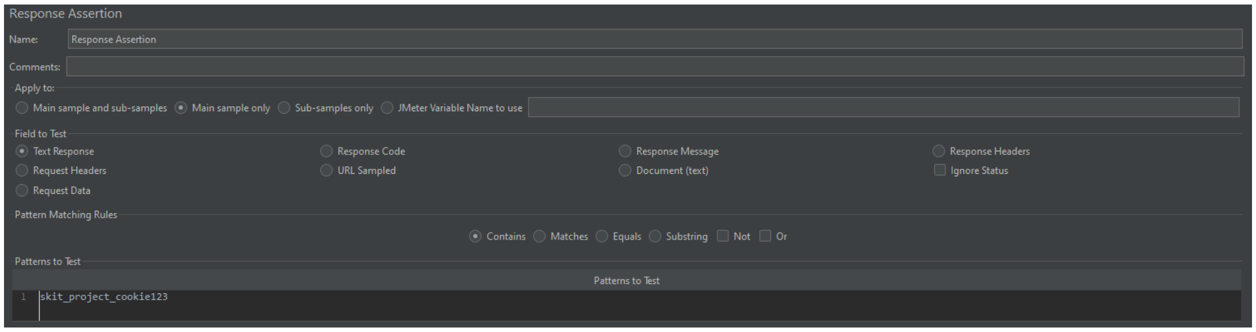
Web Server

Protocol [http]: https   Server Name or IP: httpbin.org   Port Number:

HTTP Request

GET   Path: /cookies   Content encoding:

And the cookie retriever is a request that is trying to retrieve the cookies that are already set. So in the first request we set, and here we see if the cookie exists. Additionally we set an assertion to the retriever request, where we see if the cookie is set correctly.



Response Assertion

Name: Response Assertion

Comments:

Apply to:

Main sample and sub-samples [✓ Main sample only] Sub-samples only JMeter Variable Name to use

Field to Test

[✓ Text Response] Response Code Response Message Response Headers

Request Headers URL Sampled Document (text) [ ] Ignore Status

Request Data

Pattern Matching Rules

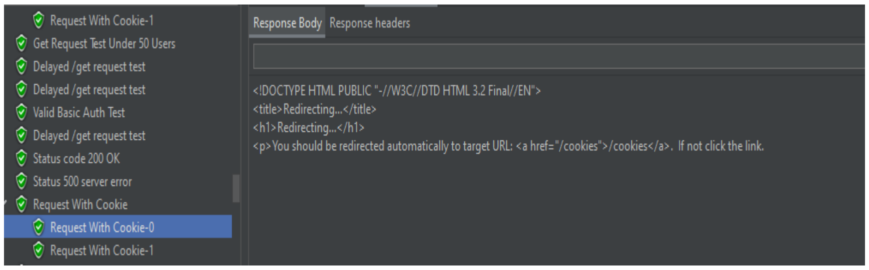
[✓ Contains] Matches Equals Substring [ ] Not [ ] Or

Patterns to Test

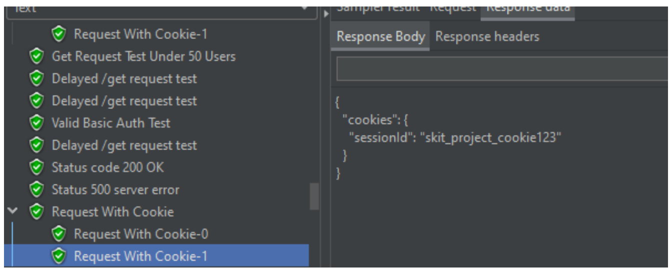
Patterns to Test

```
skit_project_cookie123
```

On the screenshot below, we can see that the body of the request is giving that the cookie exists and gives its sessionId.



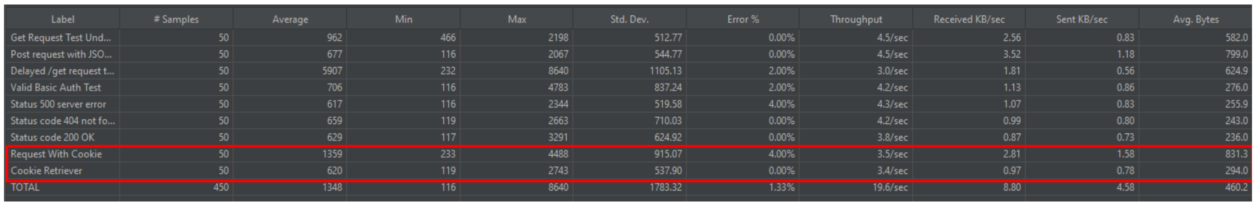
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="/cookies">/cookies</a>.  If not click the link.
```



```
{
  "cookies": {
    "sessionId": "skit_project_cookie123"
  }
}
```

Request with cookie-0 - (/cookies/set) sets the cookie and then redirects to /cookies.

Request with cookie-1 - (/cookies) is explicitly added to verify the cookie value.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get Request Test Und...	50	962	466	2198	512.77	0.00%	4.5/sec	2.56	0.83	582.0
Post request with JSO...	50	677	116	2067	544.77	0.00%	4.5/sec	3.52	1.18	799.0
Delayed /get request t...	50	5907	232	8640	1105.13	2.00%	3.0/sec	1.81	0.56	624.9
Valid Basic Auth Test	50	706	116	4783	837.24	2.00%	4.2/sec	1.13	0.86	276.0
Status 500 server error	50	617	116	2344	519.58	4.00%	4.3/sec	1.07	0.83	255.9
Status code 404 not fo...	50	659	119	2663	710.03	0.00%	4.2/sec	0.99	0.80	243.0
Status code 200 OK	50	629	117	3291	624.92	0.00%	3.8/sec	0.87	0.73	236.0
Request With Cookie	50	1359	233	4488	915.07	4.00%	3.5/sec	2.81	1.58	831.3
Cookie Retriever	50	620	119	2743	537.90	0.00%	3.4/sec	0.97	0.78	294.0
TOTAL	450	1348	116	8640	1783.32	1.33%	19.6/sec	8.80	4.58	460.2

Explanation of the test result:

The test confirms successful cookie management between requests. The redirect in the first request increases the average response time and accounts for some variability. The second request validates that the cookie is correctly stored and reused, with stable performance and zero errors. This test is essential for validating session persistence and client tracking, effectively introducing stateful behavior into an otherwise stateless HTTP protocol.

Load Test with 10 users at a time:

Then we create a new Thread Manager, with 10 users at a time, all of them will have access on a period of 5/10 seconds and will have to access it 2 times.

Scenario 1 – Http Put

Here we test the method put, where on /put we insert a json that updates update=true. We check it by adding response body assertion that checks if the json is contained in the updated object.

Scenario 2 – Redirect Request

Here we want to achieve redirecting to another page /redirect-to?url=https://httpbin.org/get. And we check if the page url is the same with the url provided for redirection.

Scenario 3 – Ip Detection Request

Here we try to detect if on /ip, it will return an ip address.

Scenario 4 – Check Host Header

Here it checks the host header mapped on /headers.

Scenario 5 – Base64 – Echo

It goes to an endpoint /base64/encoded string}, where the encoded string is decoded and in the assertion, we check if the meaning of the encoded string is the one we like to be returned.

Load Testing Fake Store Application (ReqRes app)

Description of the testing:

This thread manager is going to test a fake application that is created for api testing, and I will load test it (app_link). I will use 100 users, every of them will start on a period of 10/100 seconds.

Scenario 1 - Get Users Page = 2

This is going to fetch all the users from the second page and check if there is a user and to get his email. If this is done, it means that there is a list of users. We will set to check from the data[1], so it can check if there is a list returned. This will check if users are successfully fetched.

Scenario 2 – Get just one user (single user)

This will load tests by fetching a single user (in our case with id=2). The check is going to be made with Json assertion, to check if the fetched object has id, that means that there is a data (user) returned.

Scenario 3 – Get non-existing user

This scenario validates the system properly handles cases where a user does not exist. When querying /api/users/404, the server returned a 404 response code, which was verified using a Response Assertion. This is important for verifying error handling and client feedback mechanisms under heavy load.

Scenario 4 – Creating user using POST method

In this scenario, we simulated creating a new user using the /api/users POST endpoint. We validated that the server returned a 201 Created status and that the response JSON by checking if there is a generated ID inside, and a generated ID.

Scenario 5 – Update Existing User, with id=2

Here we are updating the user with id=2, by putting new name and job to it. We make the check with json assertion, where we set the new expected job label to be updated.

Scenario 6 - Delete User

Here we test deleting the user with id = 2. The application simulates deleting the same user, that is going to prove the concept of 'what if many users delete at same time, will it perform good'. We check if the code 204 for successful delete is returned.

Scenario 7 - Delayed response handling

We check here the delay of 3 seconds, if the delay expires and the response code is 200, it means that the get method is successful and the test passed. Also we added a check, for the duration not to be over 4200 milliseconds.

Scenario 8 - New user registration

Here we tested successful user registration via /api/register by submitting valid credentials. The server responded with HTTP 200 and returned a JSON token. Assertions ensured both the token field and response code were valid.

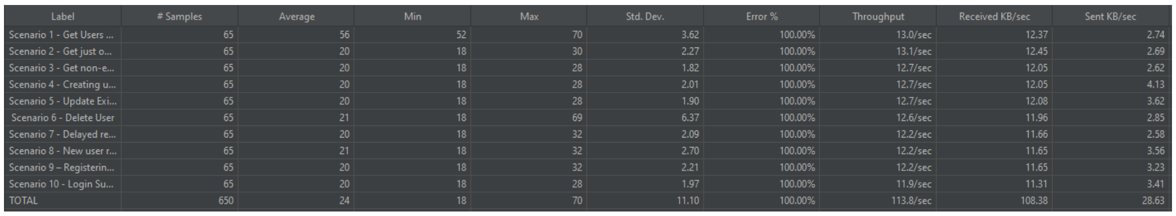
Scenario 9 – Registering fails (missing password)

Opposite of scenario 8, this is a negative test case that checks how the API handles incomplete user input during registration. When the password is missing, the /api/register endpoint returns a 400 Bad Request with an appropriate error message. We catch the error by checking if the code is 400 and the register is failed with error message.

Scenario 10 - Login Successfully Done

This scenario tested the login functionality via the /api/login endpoint using valid credentials. The server responded with a 200 OK and returned a valid authentication token. Assertions verified the presence of the token field and correct status code, ensuring the login flow is functioning reliably under simulated user load.

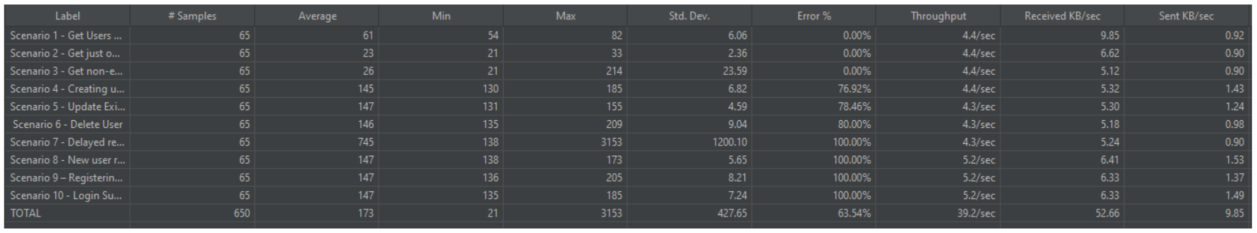
Now we will provide some images of the load testing with different Thead Manager Configurations

1. Number of Threads (users): 65
   Ramp-up Period (seconds): 5
   Loop Count: 1
   

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec
Scenario 1 - Get Users ...	65	56	52	70	3.62	100.00%	13.0/sec	12.37	2.74
Scenario 2 - Get just o...	65	20	18	30	2.27	100.00%	13.1/sec	12.45	2.69
Scenario 3 - Get non-e...	65	20	18	28	1.82	100.00%	12.7/sec	12.05	2.62
Scenario 4 - Creating u...	65	20	18	28	2.01	100.00%	12.7/sec	12.05	4.13
Scenario 5 - Update Exi...	65	20	18	28	1.90	100.00%	12.7/sec	12.08	3.62
Scenario 6 - Delete User	65	21	18	69	6.37	100.00%	12.6/sec	11.96	2.85
Scenario 7 - Delayed re...	65	20	18	32	2.09	100.00%	12.2/sec	11.66	2.58
Scenario 8 - New user r...	65	21	18	32	2.70	100.00%	12.2/sec	11.65	3.56
Scenario 9 – Registerin...	65	20	18	32	2.21	100.00%	12.2/sec	11.65	3.23
Scenario 10 - Login Su...	65	20	18	28	1.97	100.00%	11.9/sec	11.31	3.41
TOTAL	650	24	18	70	11.10	100.00%	113.8/sec	108.38	28.63

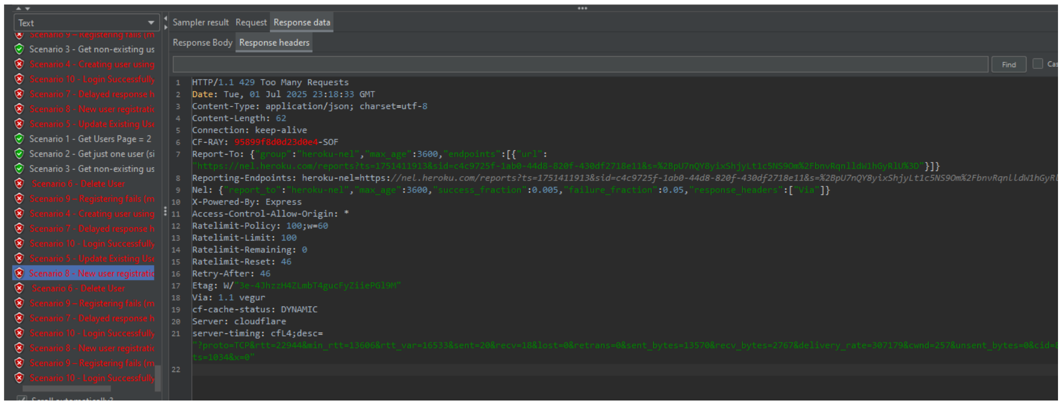
   Catastrophically output, because the load of 65 users, each user differentiated by a period of 5/65 seconds with one try each is not going well. It gives so bad testing results

2. Number of Threads (users): 65
   Ramp-up Period (seconds): 15
   Loop Count: 1



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec
Scenario 1 - Get Users ...	65	61	54	82	6.06	0.00%	4.4/sec	9.85	0.92
Scenario 2 - Get just o...	65	23	21	33	2.36	0.00%	4.4/sec	6.62	0.90
Scenario 3 - Get non-e...	65	26	21	214	23.59	0.00%	4.4/sec	5.12	0.90
Scenario 4 - Creating u...	65	145	130	185	6.82	76.92%	4.4/sec	5.32	1.43
Scenario 5 - Update Exi...	65	147	131	155	4.59	78.46%	4.3/sec	5.30	1.24
Scenario 6 - Delete User	65	146	135	209	9.04	80.00%	4.3/sec	5.18	0.98
Scenario 7 - Delayed re...	65	745	138	3153	1200.10	100.00%	4.3/sec	5.24	0.90
Scenario 8 - New user r...	65	147	138	173	5.65	100.00%	5.2/sec	6.41	1.53
Scenario 9 – Registerin...	65	147	136	205	8.21	100.00%	5.2/sec	6.33	1.37
Scenario 10 - Login Su...	65	147	135	185	7.24	100.00%	5.2/sec	6.33	1.49
TOTAL	650	173	21	3153	427.65	63.54%	39.2/sec	52.66	9.85

   Now the output is a bit better, still not as good as we need. Especially, when we log in, when we register, the most critical parts are shown as bad while big traffic visit .

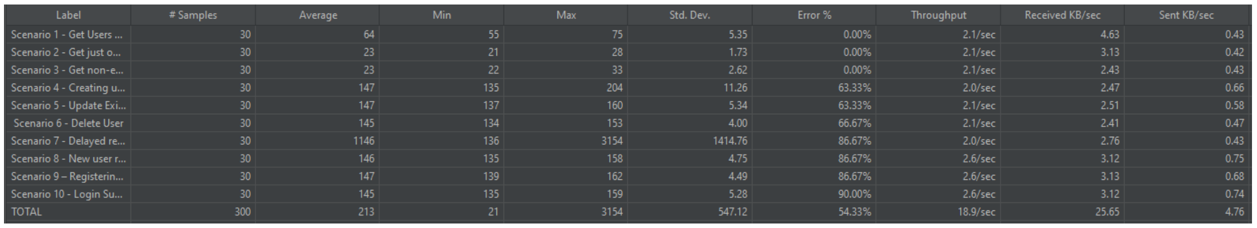


```
Scenario 3 - Get non-existing us
Scenario 4 - Creating user using
Scenario 10 - Login Successfully
Scenario 7 - Delayed response h
Scenario 8 - New user registratic
Scenario 5 - Update Existing Use
Scenario 1 - Get Users Page = 2
Scenario 2 - Get just one user (si
Scenario 3 - Get non-existing us
Scenario 6 - Delete User
Scenario 9 – Registering fails (m
Scenario 4 - Creating user using
Scenario 7 - Delayed response h
Scenario 10 - Login Successfully
Scenario 5 - Update Existing Use
Scenario 8 - New user registratic
Scenario 6 - Delete User
Scenario 9 – Registering fails (m
Scenario 7 - Delayed response h
Scenario 10 - Login Successfully
Scenario 8 - New user registratic
Scenario 9 – Registering fails (m
Scenario 10 - Login Successfully
```

```
HTTP/1.1 429 Too Many Requests
Date: Tue, 01 Jul 2025 23:18:33 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 62
Connection: keep-alive
CF-RAY: 95899f8d0d23d0e4-SOF
Report-To: {"group":"heroku-nel","max_age":3600,"endpoints":[{"url":
"https://nel.heroku.com/reports?ts=1751411913&sid=c4c9725f-1ab0-44d8-820f-430df2718e11&s=%2BpU7nQY8yixShjyLt1c5NS9Om%2FbnvRqnlldW1hGyRlU%3D"}]}
Reporting-Endpoints: heroku-nel=https://nel.heroku.com/reports?ts=1751411913&sid=c4c9725f-1ab0-44d8-820f-430df2718e11&s=%2BpU7nQY8yixShjyLt1c5NS9Om%2FbnvRqnlldW1hGyRl
Nel: {"report_to":"heroku-nel","max_age":3600,"success_fraction":0.005,"failure_fraction":0.05,"response_headers":["Via"]}
X-Powered-By: Express
Access-Control-Allow-Origin: *
Ratelimit-Policy: 100;w=60
Ratelimit-Limit: 100
Ratelimit-Remaining: 0
Ratelimit-Reset: 46
Retry-After: 46
Etag: W/"3e-4JhzzH4ZLmbT4gucFyZiiePGl9M"
Via: 1.1 vegur
cf-cache-status: DYNAMIC
Server: cloudflare
server-timing: cfL4;desc=
"?proto=TCP&rtt=22944&min_rtt=13606&rtt_var=16533&sent=20&recv=18&lost=0&retrans=0&sent_bytes=13570&recv_bytes=2767&delivery_rate=307179&cwnd=257&unsent_bytes=0&cid=
ts=1034&x=0"
```

   All the errors are Too Many Requests

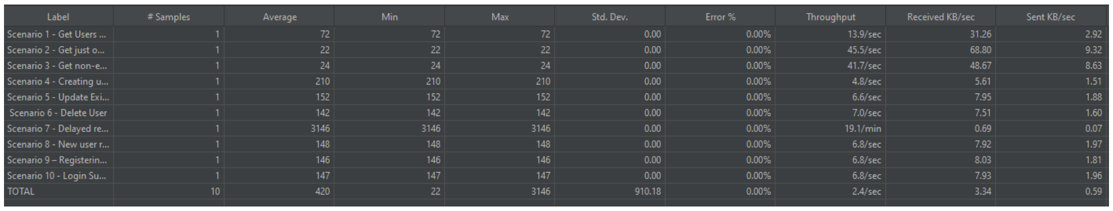
3. Number of Threads (users): 30
   Ramp-up Period (seconds): 15
   Loop Count: 1



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec
Scenario 1 - Get Users ...	30	64	55	75	5.35	0.00%	2.1/sec	4.63	0.43
Scenario 2 - Get just o...	30	23	21	28	1.73	0.00%	2.1/sec	3.13	0.42
Scenario 3 - Get non-e...	30	23	22	33	2.62	0.00%	2.1/sec	2.43	0.43
Scenario 4 - Creating u...	30	147	135	204	11.26	63.33%	2.0/sec	2.47	0.66
Scenario 5 - Update Exi...	30	147	137	160	5.34	63.33%	2.1/sec	2.51	0.58
Scenario 6 - Delete User	30	145	134	153	4.00	66.67%	2.1/sec	2.41	0.47
Scenario 7 - Delayed re...	30	1146	136	3154	1414.76	86.67%	2.0/sec	2.76	0.43
Scenario 8 - New user r...	30	146	135	158	4.75	86.67%	2.6/sec	3.12	0.75
Scenario 9 – Registerin...	30	147	139	162	4.49	86.67%	2.6/sec	3.13	0.68
Scenario 10 - Login Su...	30	145	135	159	5.28	90.00%	2.6/sec	3.12	0.74
TOTAL	300	213	21	3154	547.12	54.33%	18.9/sec	25.65	4.76

   Slightly better under less traffic.

4. The results when we have 1 user only are good, but it does mean that the application is meant to be used by only 1 or 2 users at a time. Otherwise, it will have bad results, because the server will be overheated with so many requests and only 429 – Too Many Requests will show.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec
Scenario 1 - Get Users ...	1	72	72	72	0.00	0.00%	13.9/sec	31.26	2.92
Scenario 2 - Get just o...	1	22	22	22	0.00	0.00%	45.5/sec	68.80	9.32
Scenario 3 - Get non-e...	1	24	24	24	0.00	0.00%	41.7/sec	48.67	8.63
Scenario 4 - Creating u...	1	210	210	210	0.00	0.00%	4.8/sec	5.61	1.51
Scenario 5 - Update Exi...	1	152	152	152	0.00	0.00%	6.6/sec	7.95	1.88
Scenario 6 - Delete User	1	142	142	142	0.00	0.00%	7.0/sec	7.51	1.60
Scenario 7 - Delayed re...	1	3146	3146	3146	0.00	0.00%	19.1/min	0.69	0.07
Scenario 8 - New user r...	1	148	148	148	0.00	0.00%	6.8/sec	7.92	1.97
Scenario 9 – Registerin...	1	146	146	146	0.00	0.00%	6.8/sec	8.03	1.81
Scenario 10 - Login Su...	1	147	147	147	0.00	0.00%	6.8/sec	7.93	1.96
TOTAL	10	420	22	3146	910.18	0.00%	2.4/sec	3.34	0.59

Conclusion:

This project demonstrates the power and flexibility of Apache JMeter for both load and functional testing. Through multiple test scenarios using public APIs like httpbin.org and reqres.in, we were able to validate:

- System behavior under concurrent user load.
- Correct handling of different HTTP methods (GET, POST, PUT, DELETE).
- Response verification through assertions (status codes, JSON responses, redirects, cookies, headers).

The ReqRes application specifically revealed limitations under high load (many 429 “Too Many Requests” errors), highlighting its unsuitability for high-traffic environments without proper throttling or load balancing.

By creating structured thread groups, adding assertions, and analyzing metrics like response time, throughput, and error percentage, we demonstrated how JMeter can be used not just for automation but as a powerful tool for simulating production-like traffic, improving reliability, performance, and scalability of web APIs and applications.

JMeter, when used correctly, is an essential asset in a QA or DevOps pipeline for identifying bottlenecks and ensuring readiness of the application for the real world usage.