Introduction to Performance Testing

Task 1: Basic Test Plan and Purpose of each component

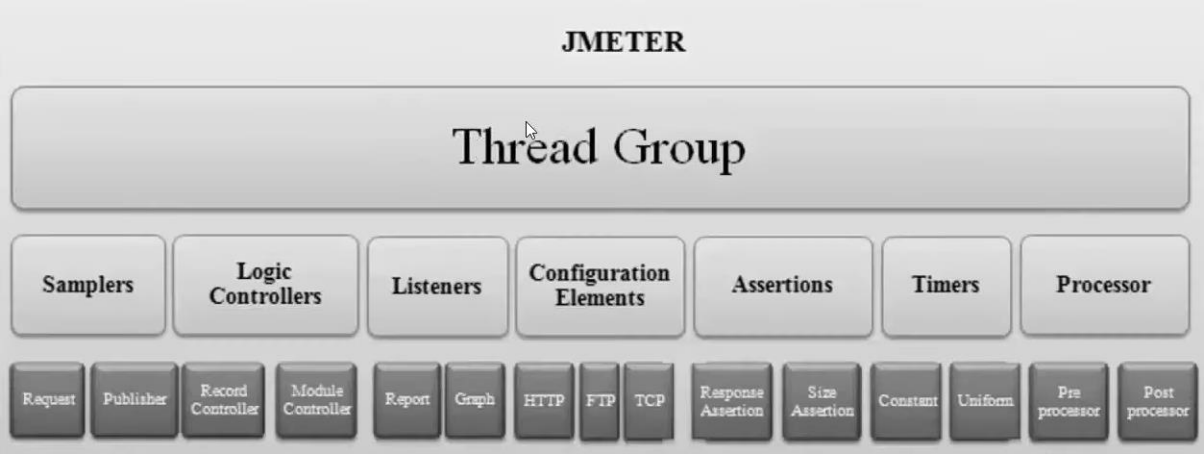
Apache JMeter is open-source Java application designed to measure performance and load test applications. It can measure performance and load test static and dynamic web applications.

It can also be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyse overall performance under different load types.

Test Plan: A test plan is a container for running performance test, it includes elements, settings and configuration to simulate user traffic and measure the performance of web application and server.

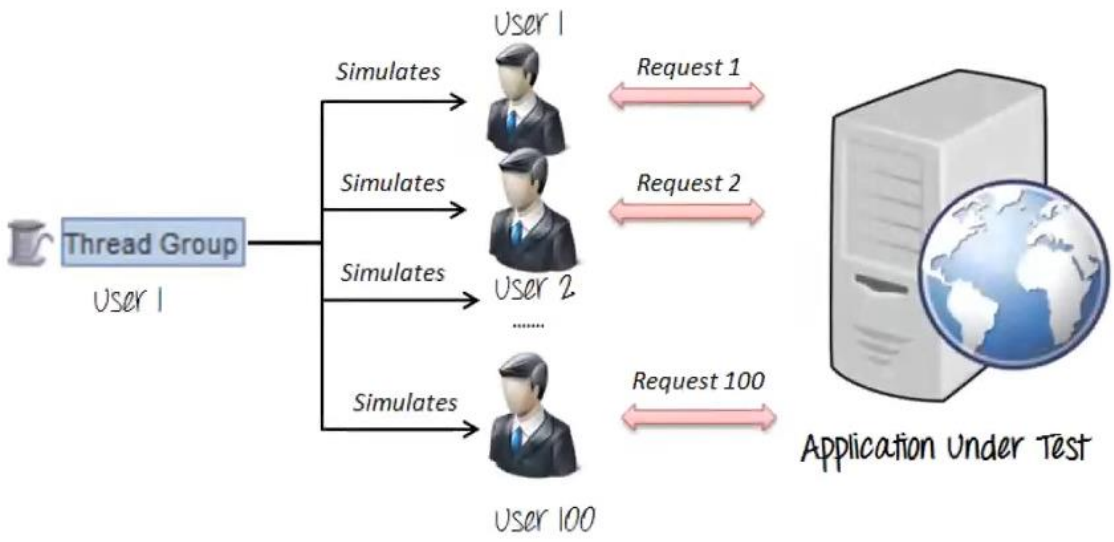
Components: The different components of JMeter are called Elements. Each element is designed for a specific purpose.

1. Thread Group
2. Samplers
3. Listeners
4. Configuration
5. Assertion
6. Timers
7. Processor



1. Thread Group: It is a collection of threads, each thread simulates one real user request to the server means here thread represents one user using the application under test.

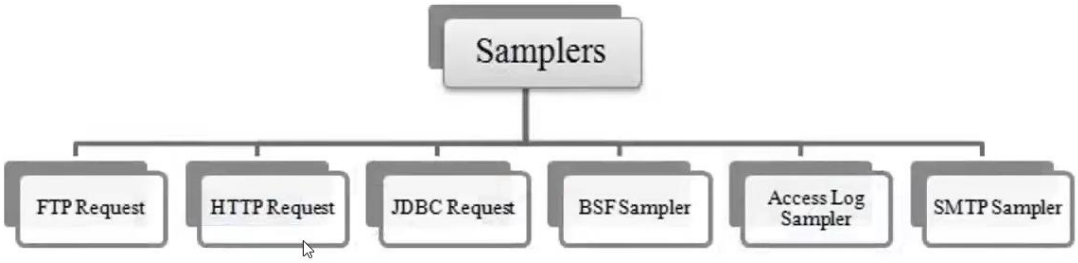
For example: If we set the number of threads as 100, JMeter will create and simulate 100 user requests to the server under test.



2. Samplers: Samplers are used to sending request to the server, it simulates the user requests for a web page, REST APIs, database etc.

There are various samplers that is common in JMeter -

1. HTTP Request
2. FTP Request
3. JDBC Request
4. TCP Request
5. SMTP Request
6. BeanShell Sampler
7. Debug Sampler etc.

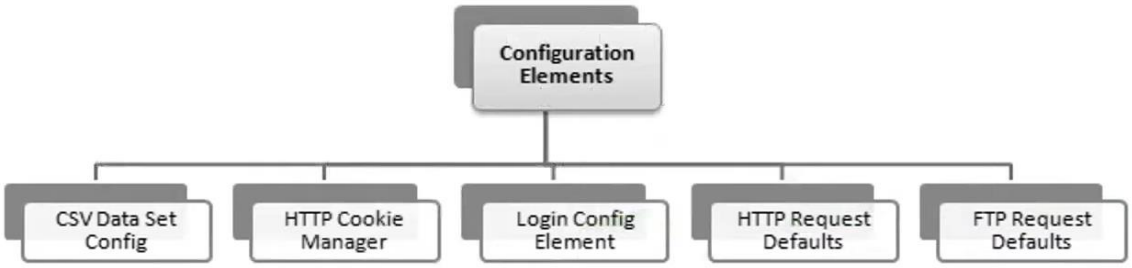


3. Listeners: Listeners is used to represent the test execution result, It shows the result in a different format such as Table format, Graph format, View Result Tree, Log etc.

4. Configuration Elements: It is used to configure the behaviour of samplers, controller and other elements within the test plan.

There are some common configure elements are:

1. CSV Data Set Config
2. HTTP Cookie Manager
3. Login Config Elements
4. HTTP Request Defaults
5. FTP Request Defaults
6. HTTP Header Manager
7. User Defined Variables etc.



5. Assertion: Assertion is used to validate the responses we received from the server we get after the test execution; it allows to check that server meets our certain criteria or conditions.

JMeter includes some common assertions:

1. Response Assertions
2. Duration Assertions

3. XML Assertions
4. Size Assertions
5. HTML Assertions
6. JSON Assertion etc.

6. Timers: Timer is used to introduce delays between requests sent by threads in a thread group.

Timer element can be added in a test plan to apply wait between each sampler/request.

Various types of timers in JMeter:

- Constant Timer
- Uniform Random Timer
- Gaussian Random Timer
- BeanShell Timer
- BSF Timer
- JSR223 Timer

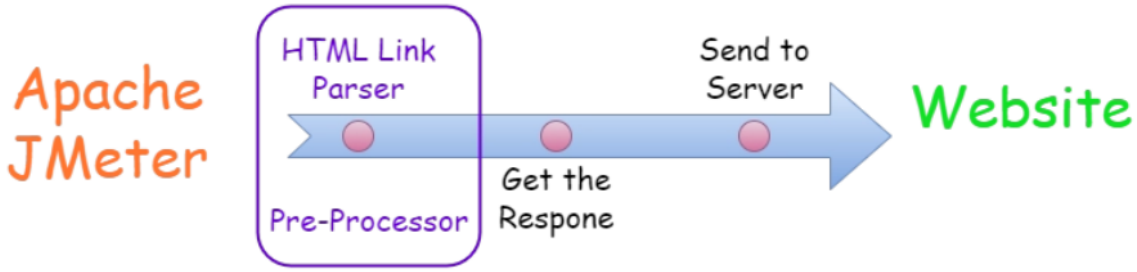
7. Processor: Processor is used to modify and change the samplers in their respective scopes.

Processors have a two types:

1. Pre-Processor
2. Post-Processor

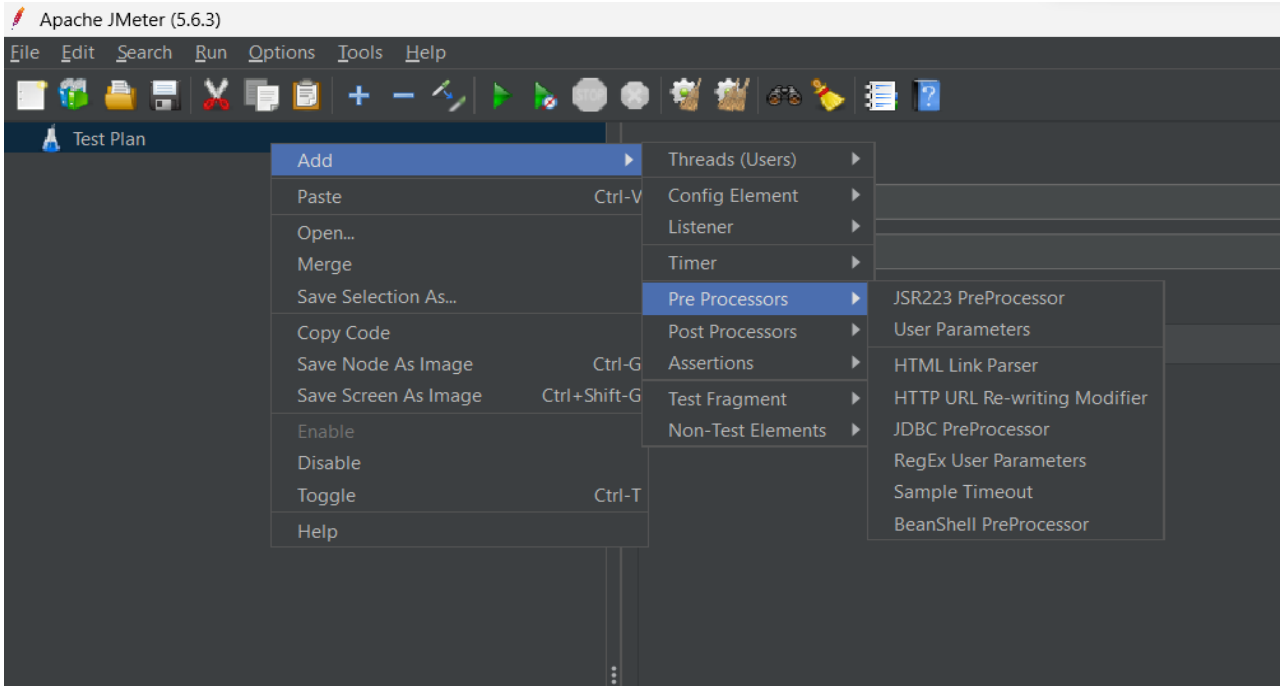
Pre-Processor Elements in JMeters:

A Pre-Processor element is a type of elements that executes some action, but this action is executed before making the sampler request.



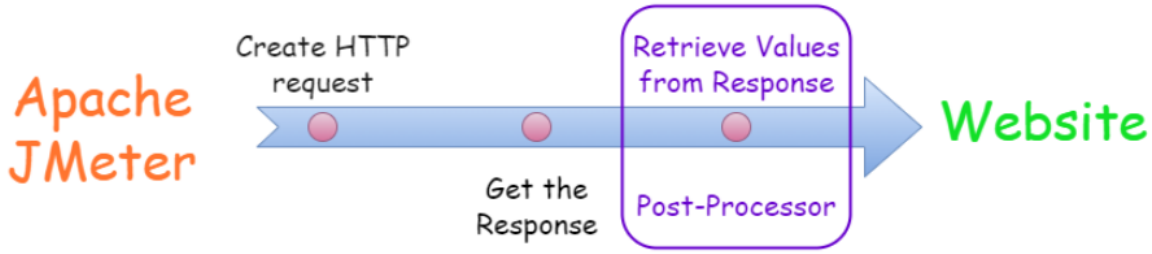
Following are all the Pre-processor elements that are already present in JMeter:

- **JSR223 Pre-processor,**
- **JDBC Pre-processor,**
- **RegEx User Parameters,**
- **BSF Pre-processor,**
- **BeanShell Pre-processor,**
- **HTML Link Parser,**
- **HTTP URL Re-writing Modifier,**
- **User Parameters, and**
- **HTTP User Parameter Modifier.**



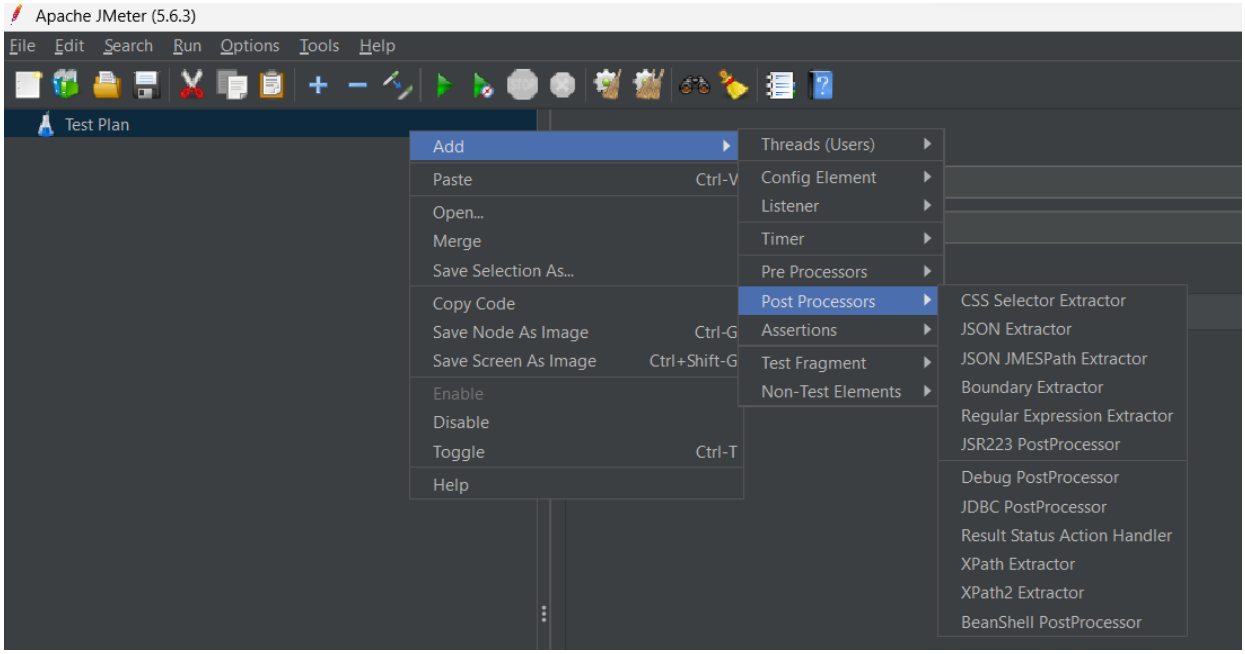
Post-Processor Elements is JMeter:

A Post-Processor element is also a type of element that executes some action, but here the action is executed just after making sampler request.



Following are all the Post-processor elements that are already present in JMeter:

- **CSS/JQuery Extractor,**
- **JSR223 Post-processor,**
- **BeanShell Post-processor,**
- **JDBC Post-processor,**
- **Regular Expression Extractor,**
- **Debug Post-processor,**
- **Result Status Action Handler,**
- **XPath Extractor, and**
- **BSF Post-processor.**



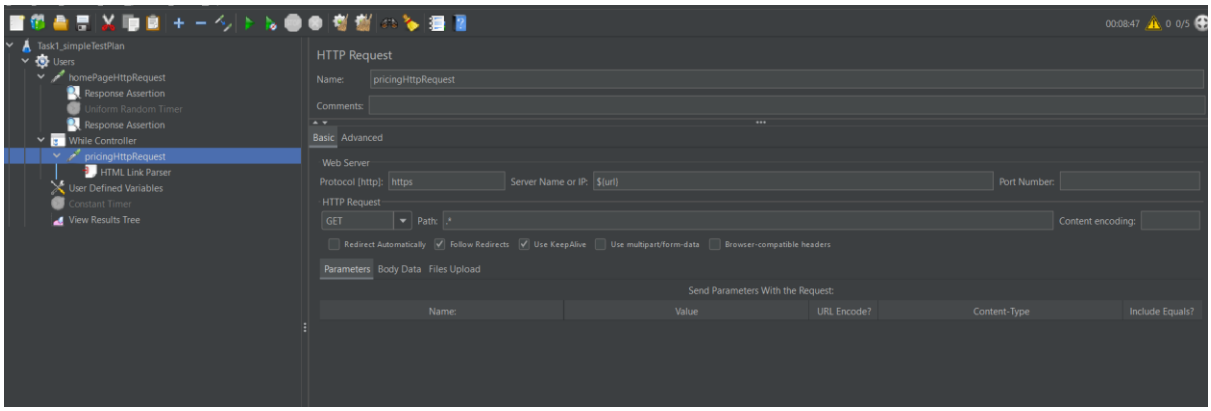
Basis Test Plan:

Inside this basic test plan I created a thread group names as “User” and here I declare:

- Number of Threads: 10
- Ramp-up period: 2
- Loop Count: 1

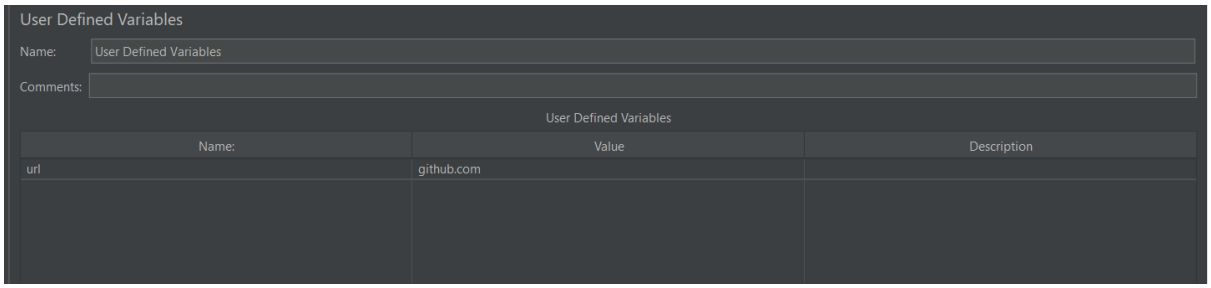
After that I include a “HTTP SAMPLER” and there I give the Server name “google.com”

After that I describe all this in the form of table listener as shown in figure:

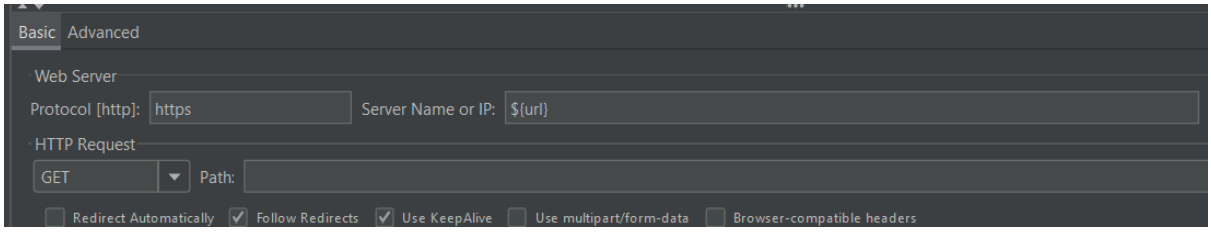


Best Practices: (I follow entire test plan)

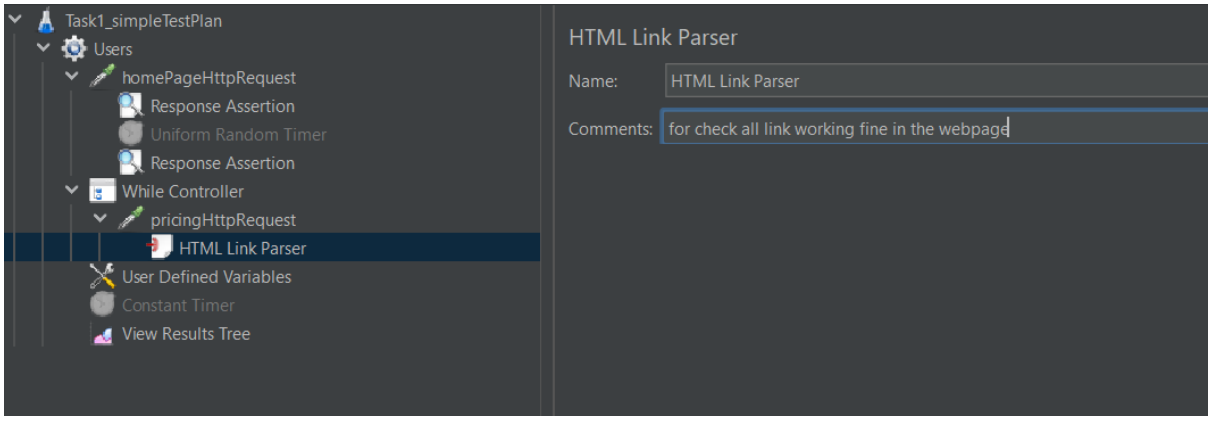
1 User defined variable



And use this url inside the https request sampler: ${url}



2. Uses Pre-processors like (**HTML Link Parser)**



3. Assertions

4. Different types of Listeners (for better analysis)

5. Various delays (timers)

6. various config elements(CSV, HTTPs Header manager, User defined Variables)

Task 2: Record a simple browsing scenario on a demo website.

HTTP(S) Test Script Recorder is used in JMeter to record the interactions between a web browser and a web application.

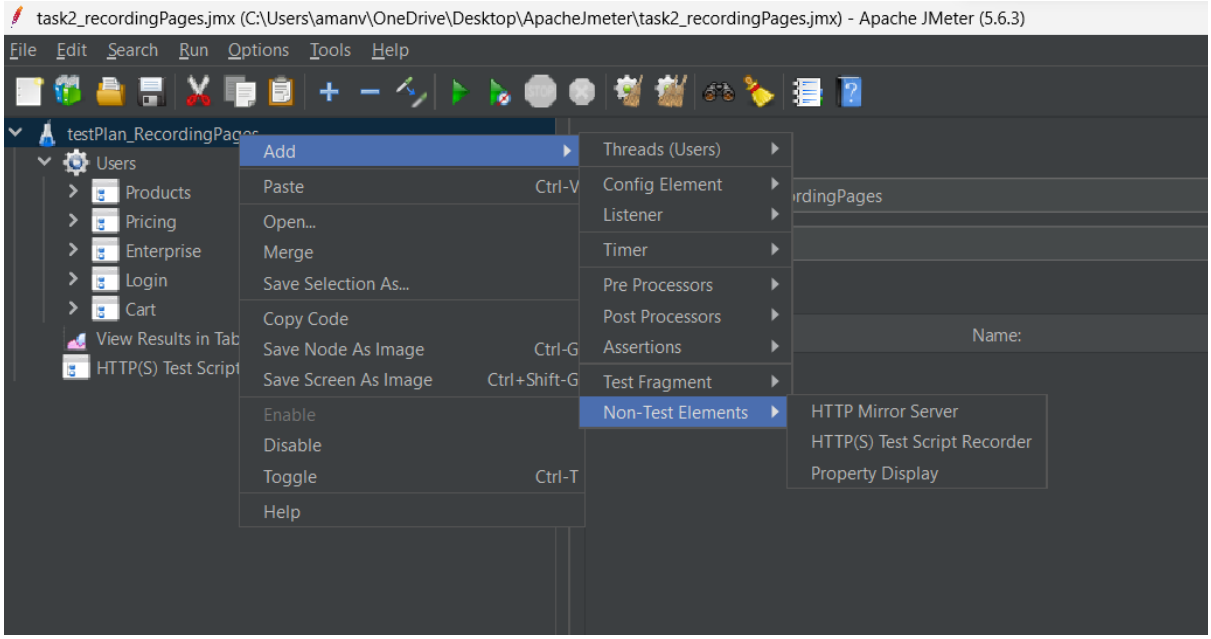
Some key reason for using HTTPs test script recorder:

- Ease to use
- Reusability
- Time Saving

How To use?(> = right click)

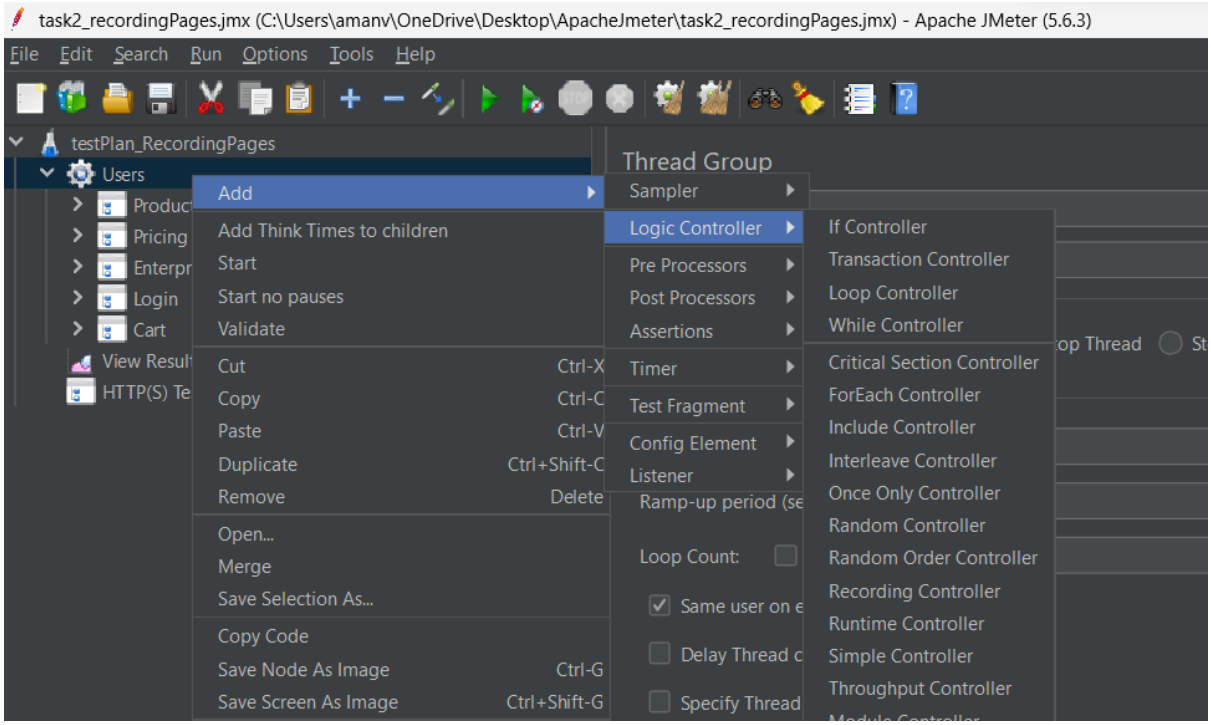
1. Go to Jmeter > Test Plan > Add > Non-Test Element

2. Select HTTPS Test Script Recorder.



3. I created a Thread Group (users)

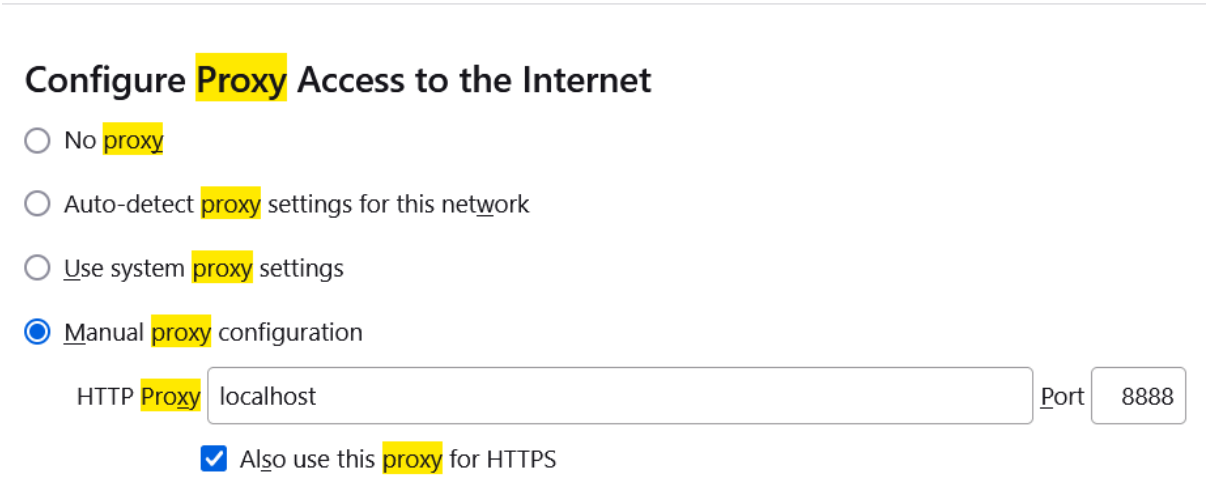
4. Inside this I created a Logic Controller (Recording Controller).

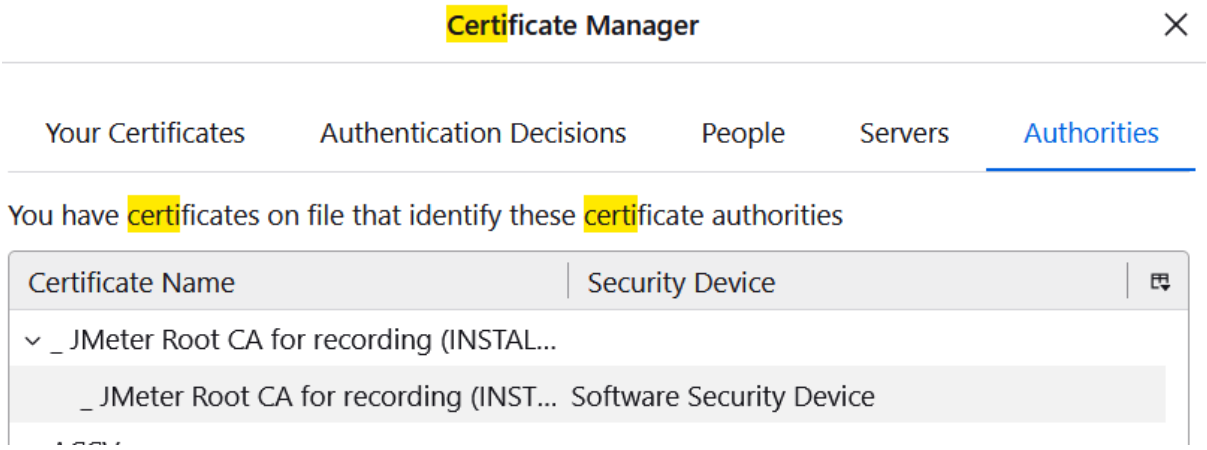


As here I named the Recording Controller as (Products, Pricing, Enterprise, Login, Cart).

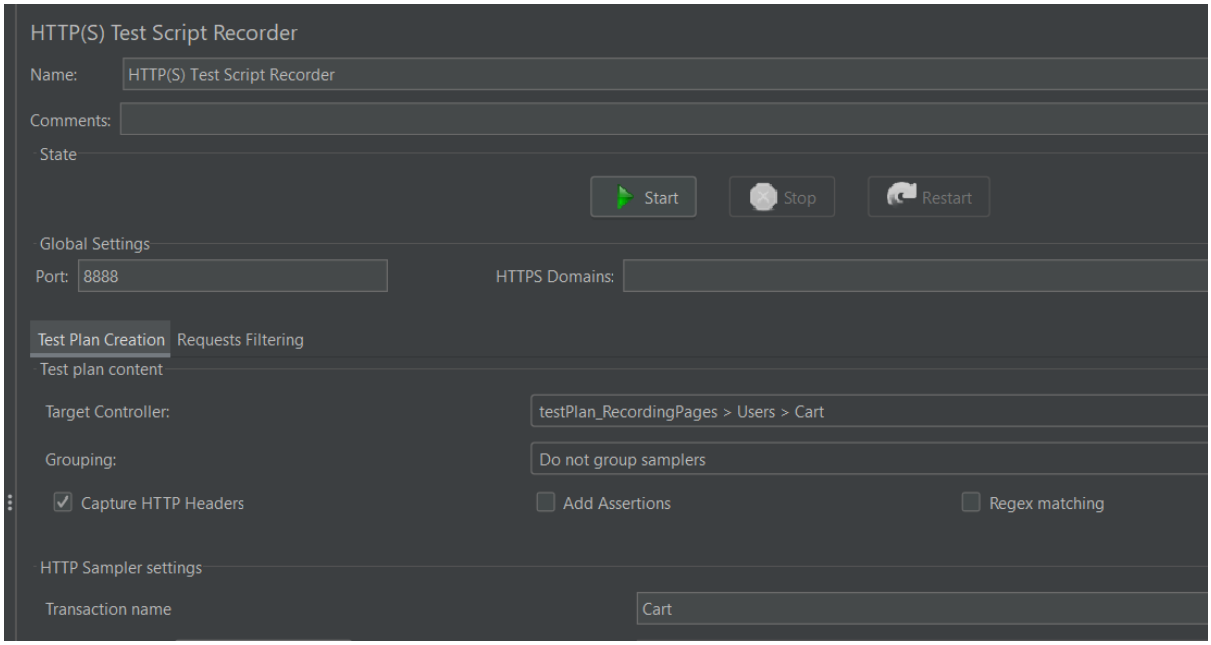
5. Configure Browser proxy:

For that I use firefox here I set up a manual proxy and certificate that is present in the jmeter

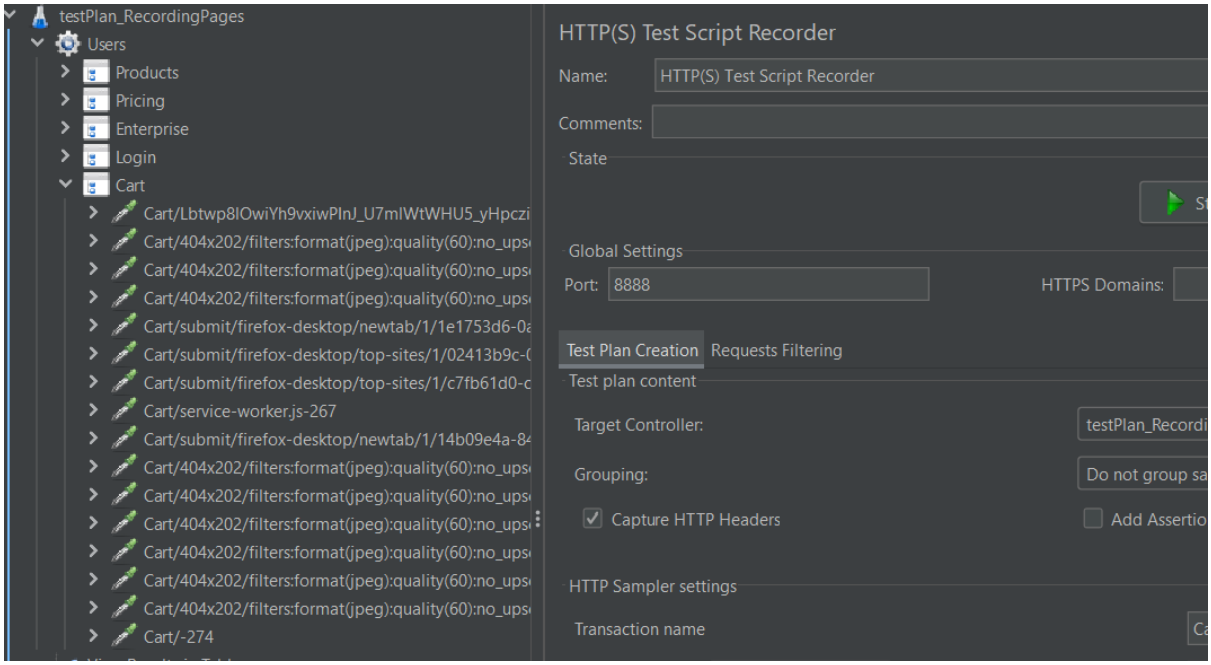




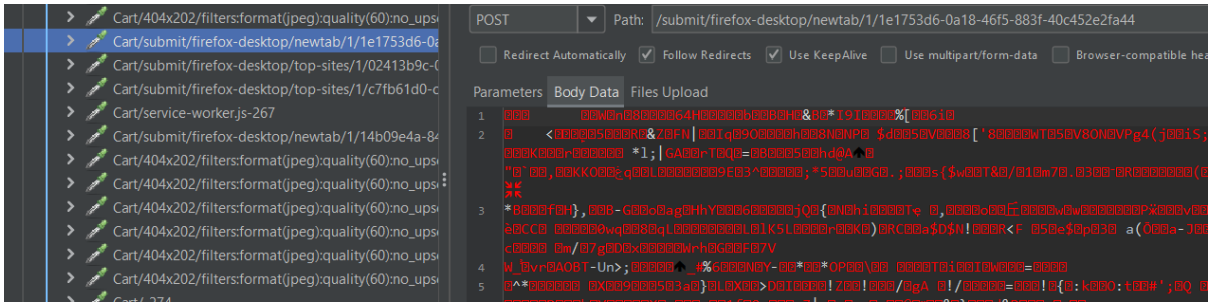
6.Start Recording

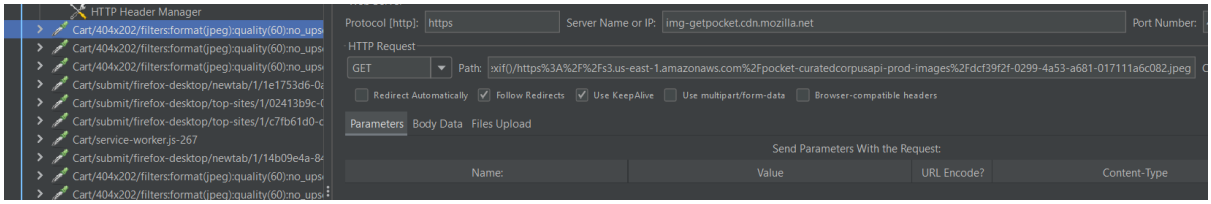


Here, I give the Target controller and also the Transaction name as Cart by this when the results occur in the controller we can easily configure.



Here I can get various results.



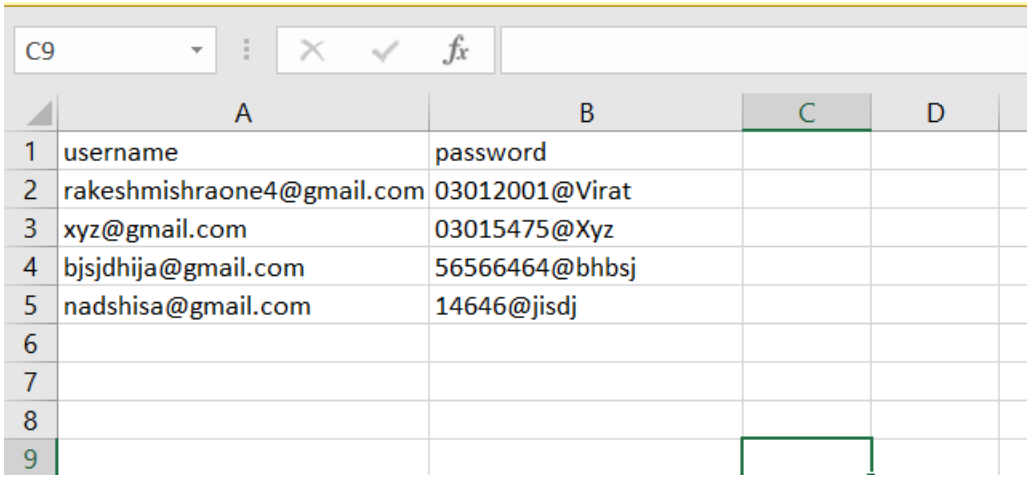


Task 3: How to use external data for testing using multiple user credentials (CSV)

We can use CSV data for providing the input data for our test script in jmeter,we can mainly use this particularly parameterized our different set of data.

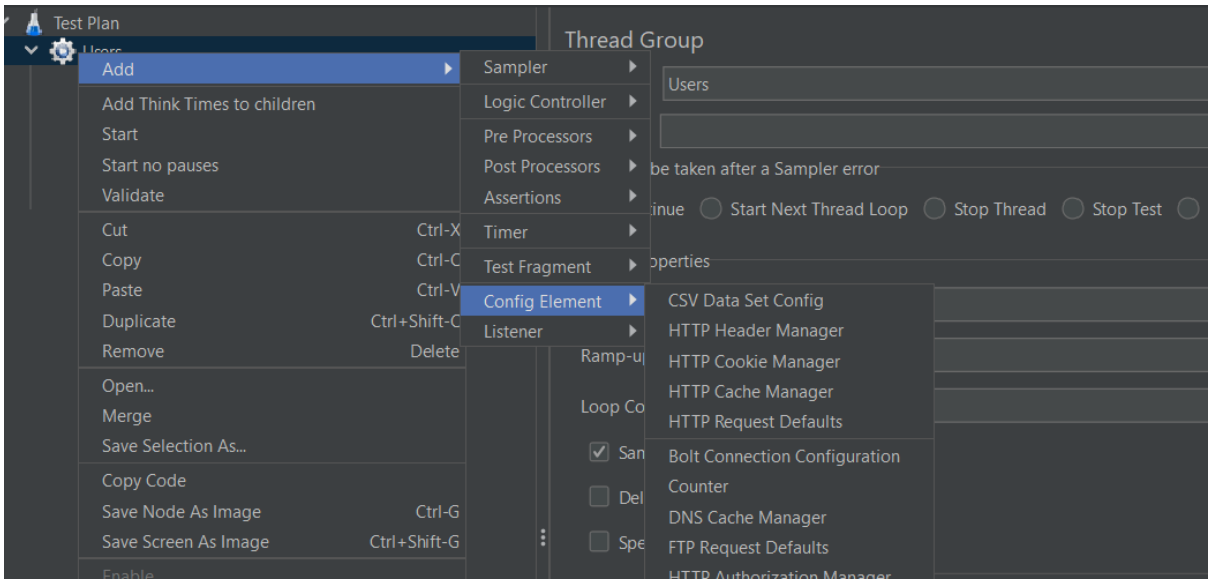
How?

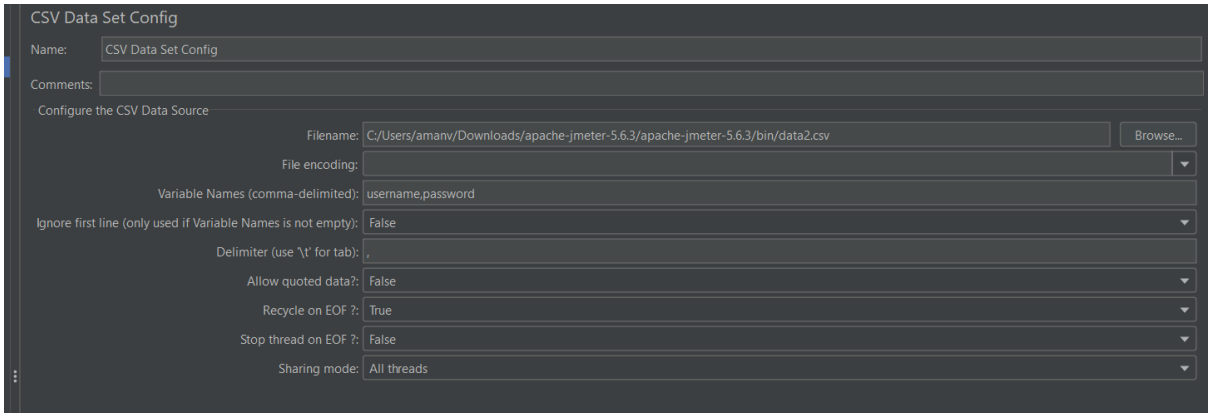
1. Create a csv file



	A	B	C	D
1	username	password		
2	rakeshmishraone4@gmail.com	03012001@Virat		
3	xyz@gmail.com	03015475@Xyz		
4	bjsjdhija@gmail.com	56566464@bhbsj		
5	nadshisa@gmail.com	14646@jisdj		
6				
7				
8				
9				

2. Configuring the Jmeter to use CSV file:

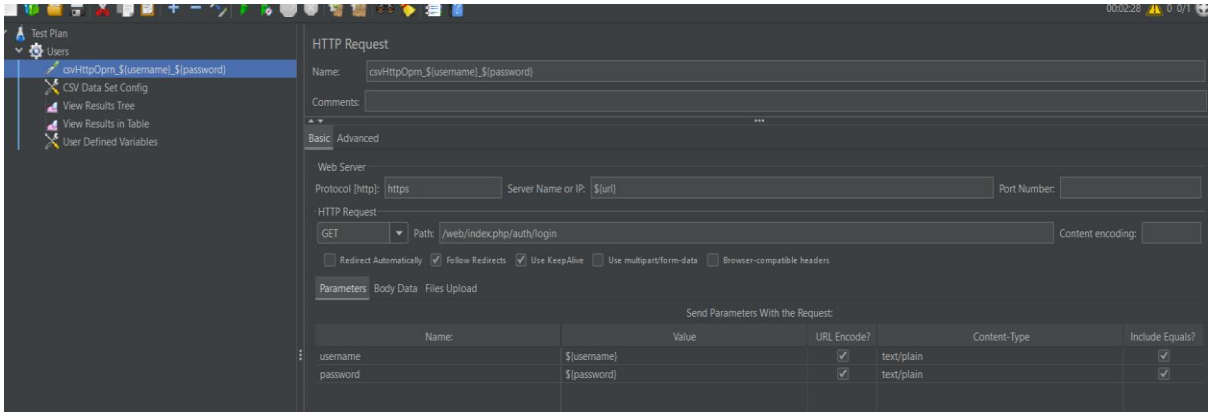




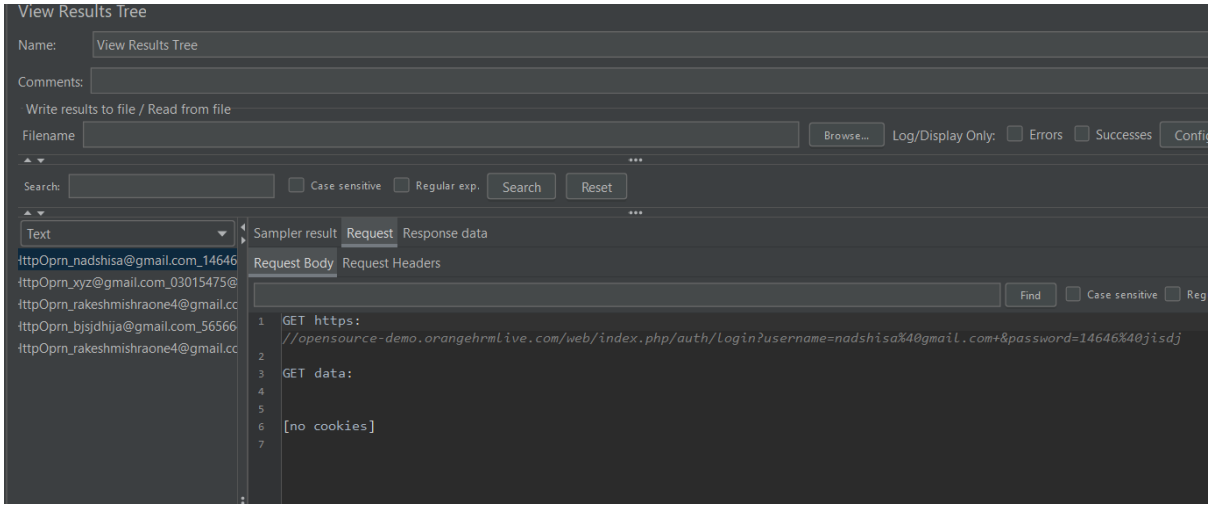
Set the file name and give the variable names and if you use the variables names in the header of the csv file than set it to “true”.

3. Create a HTTPS request sampler

For the parameter we use ${variable_name} like here I use ${username} and ${password}

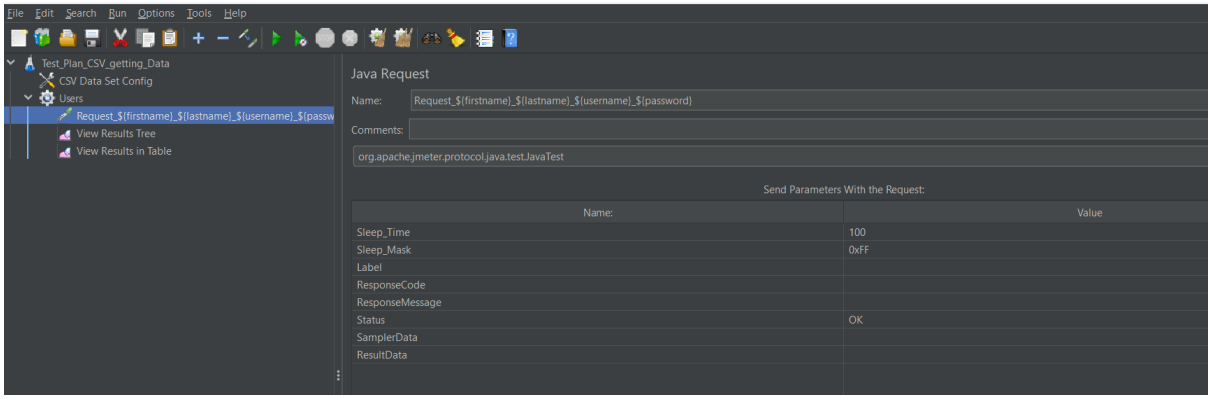


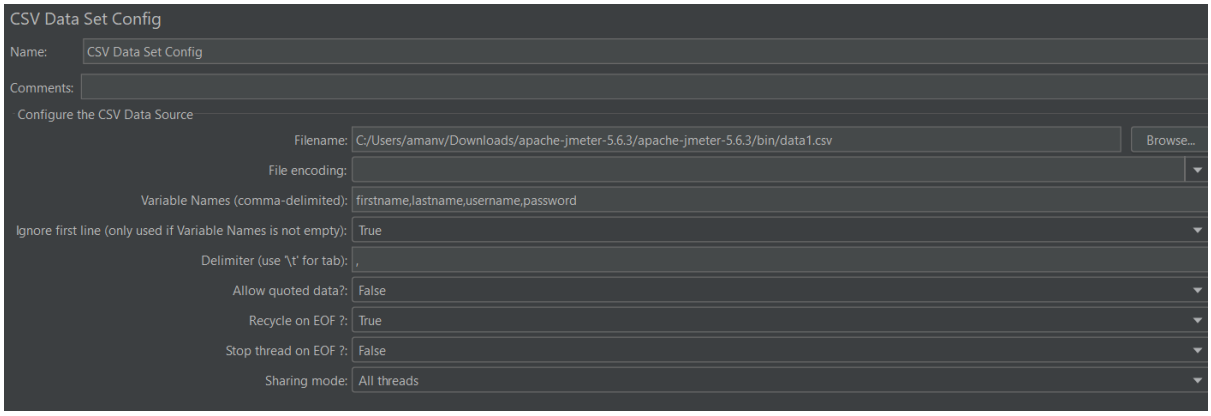
And after run the script we can get the data.



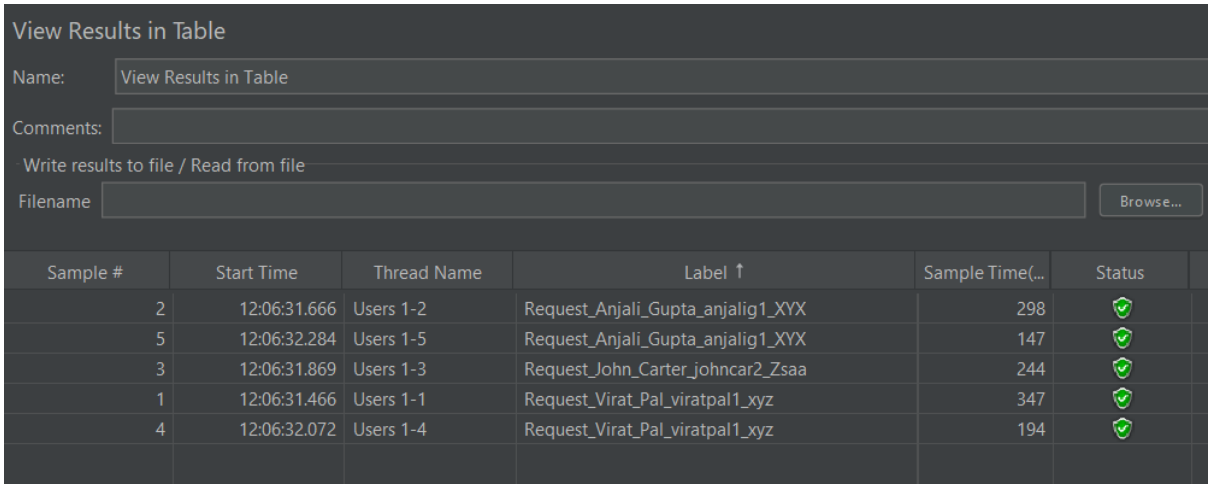
Here I also created one more file for the “getting data from csv” shown in below:

In this file I use java request sampler and I am performing the getting data also use parameterization.





Here we getting data from CSV



Sample #	Start Time	Thread Name	Label	Sample Time(...	Status
2	12:06:31.666	Users 1-2	Request_Anjali_Gupta_anjalig1_XYX	298	
5	12:06:32.284	Users 1-5	Request_Anjali_Gupta_anjalig1_XYX	147	
3	12:06:31.869	Users 1-3	Request_John_Carter_johncar2_Zsaa	244	
1	12:06:31.466	Users 1-1	Request_Virat_Pal_viratpal1_xyz	347	
4	12:06:32.072	Users 1-4	Request_Virat_Pal_viratpal1_xyz	194	

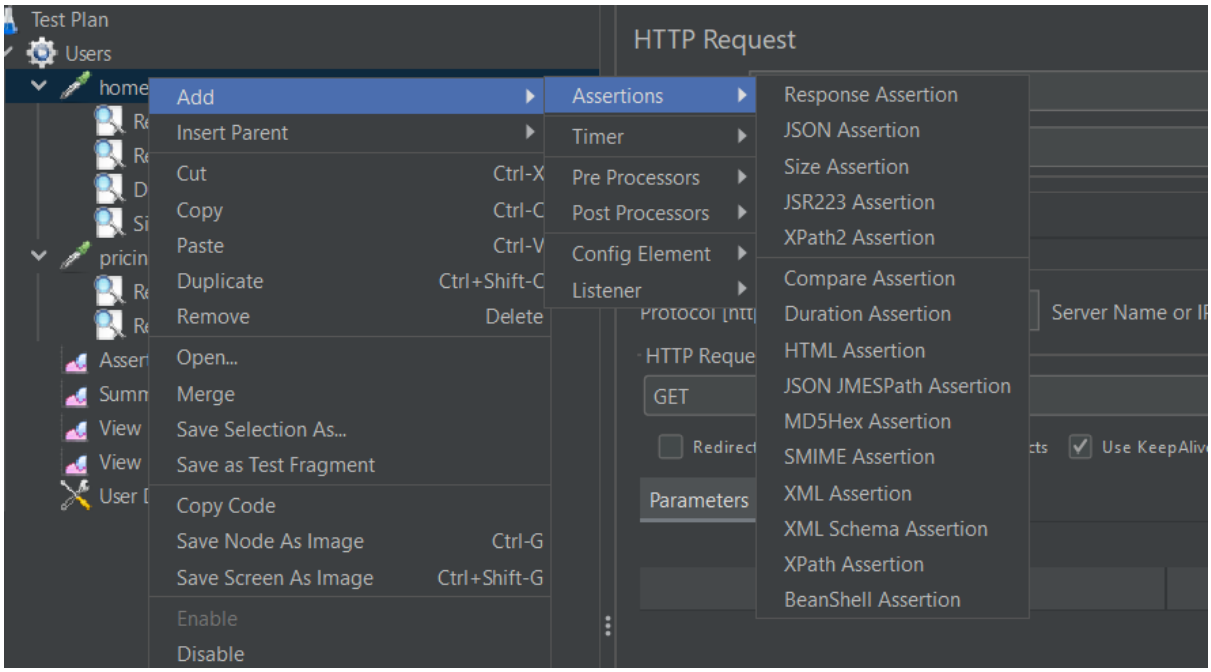
Task 4: Add assertions to a test plan to validate the presence of specific texts in the response of a web requests

How?

1. Create a thread group(users)

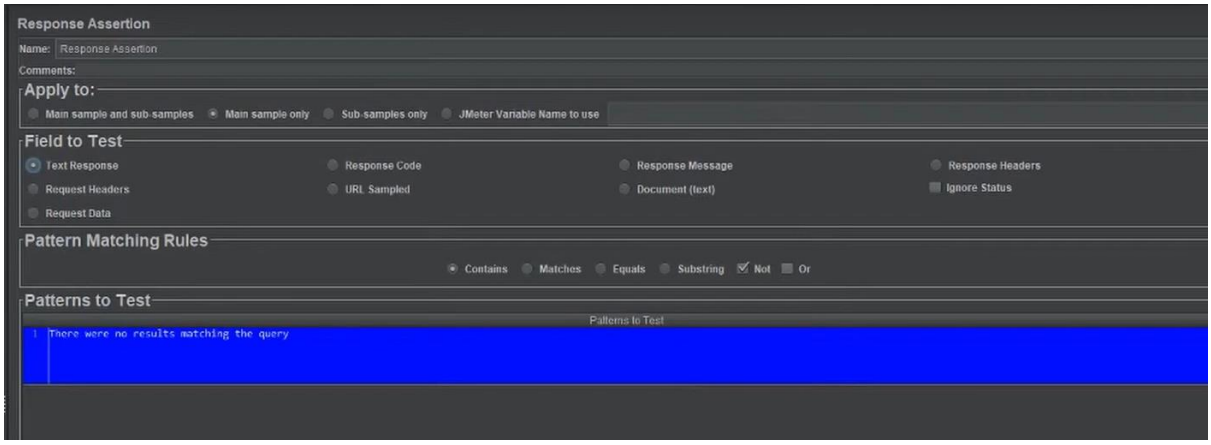
2. Inside this create a HTTPS SAMPLER

3. Use different assertions

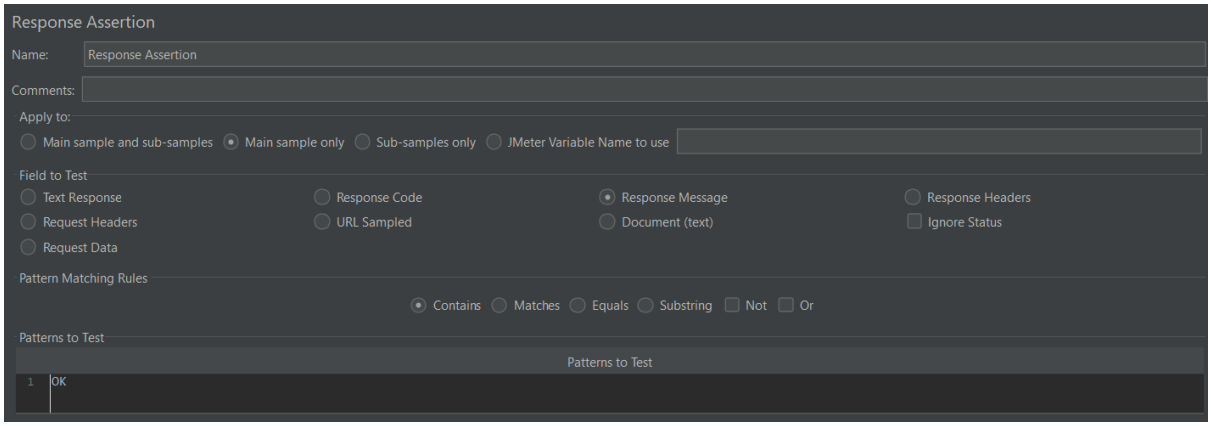


I use different assertions like response, duration, size.

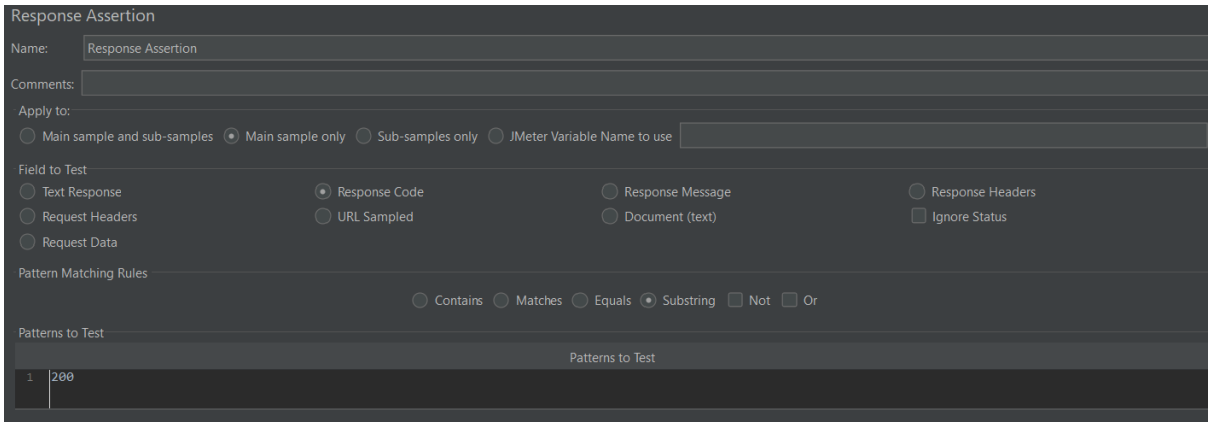
a. Response Assertion (for the specific text)



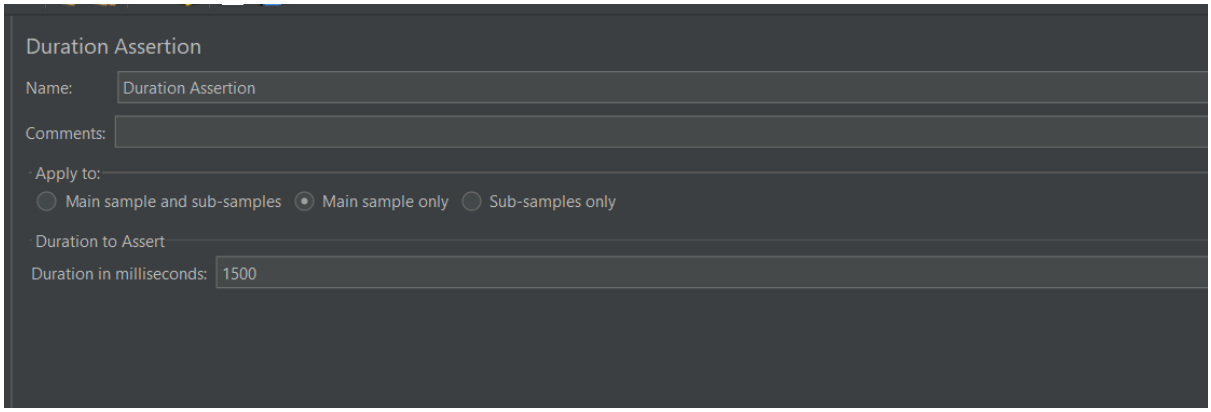
b. Response Assertion(for the response text)



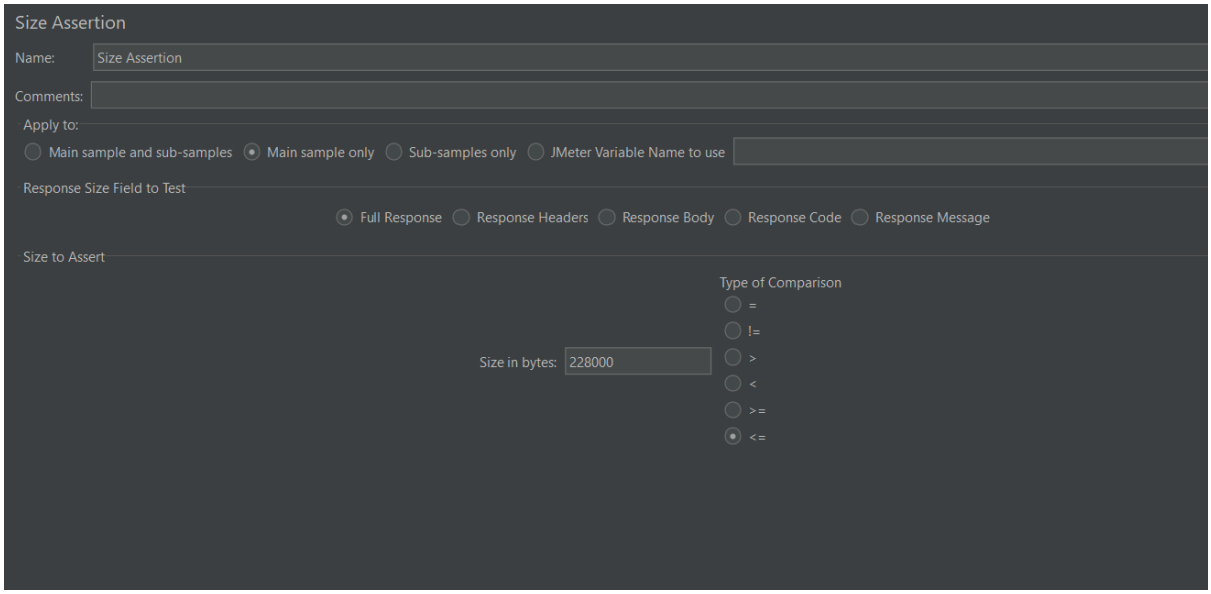
c. Response Assertion (for the status code)



d. Duration Assertion: if any request give the response me after the set time inside the duration assertion It will fail

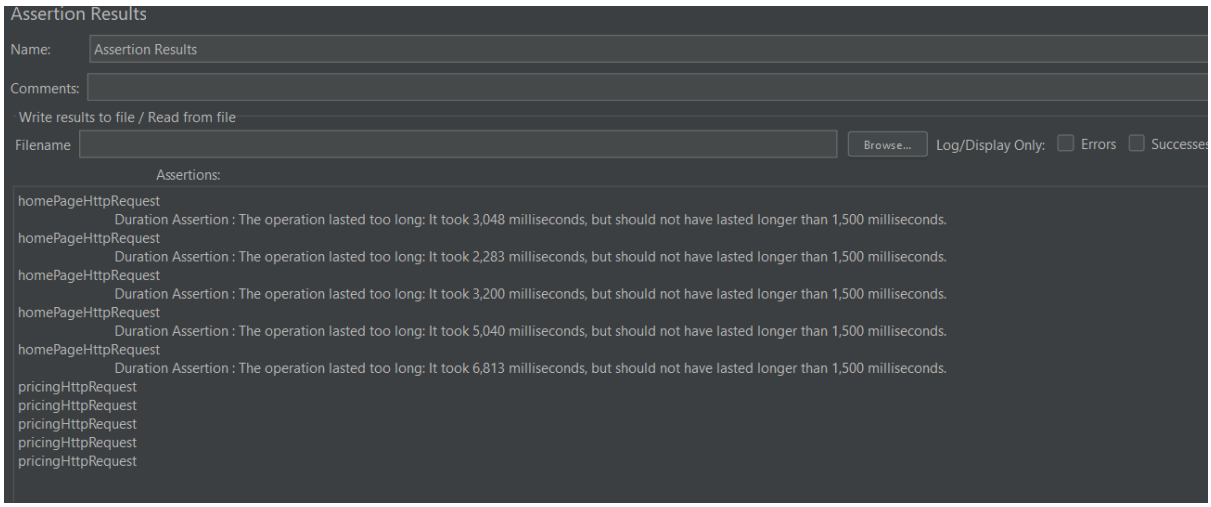


e. Size Assertion: if any request go to beyond the set size it will fail.

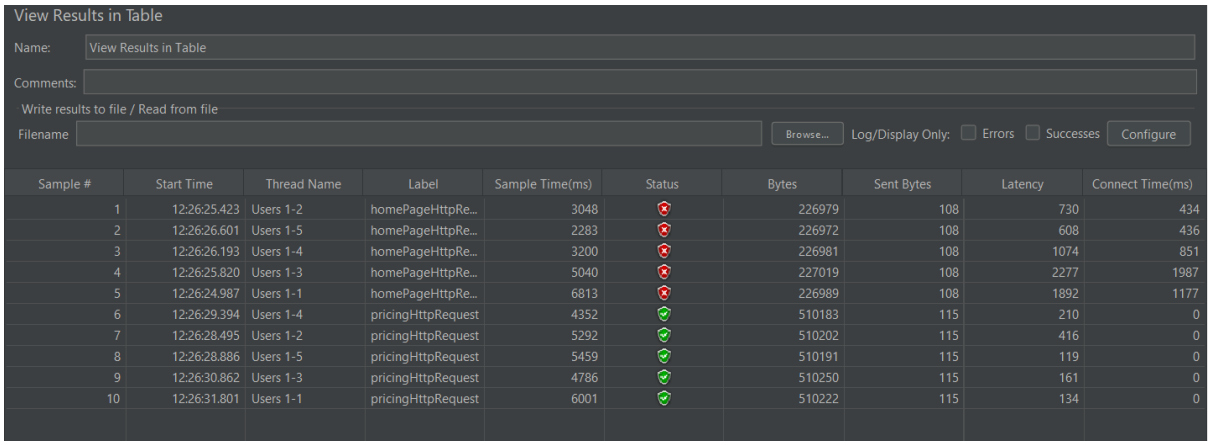


Results:

You can see here that some requests are fail due to duration assertion because it takes more that 1500 ms.

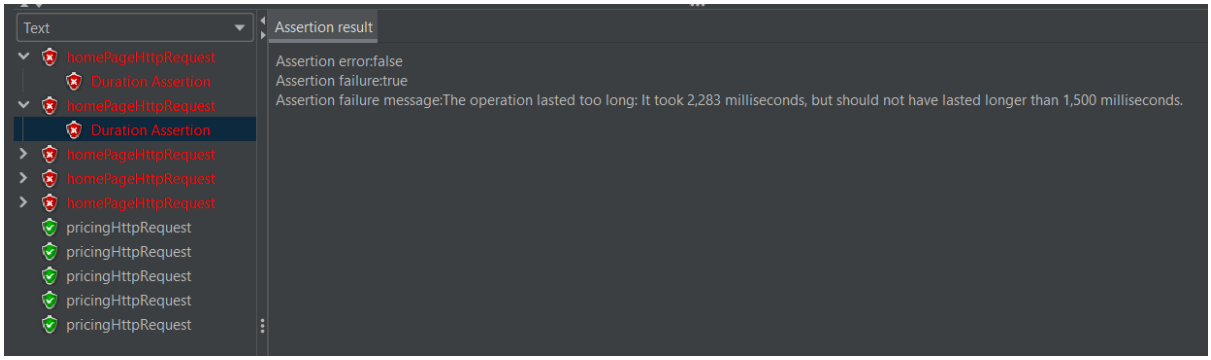


You can see here in the view results listener

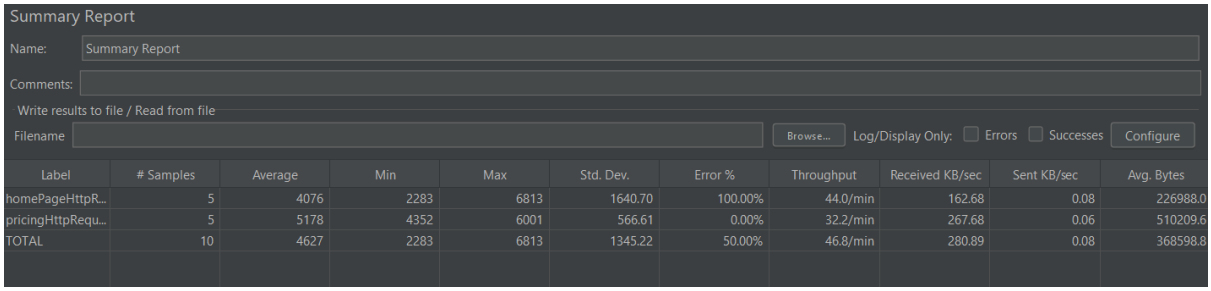


Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	12:26:25.423	Users 1-2	homePageHttpRe...	3048	✗	226979	108	730	434
2	12:26:26.601	Users 1-5	homePageHttpRe...	2283	✗	226972	108	608	436
3	12:26:26.193	Users 1-4	homePageHttpRe...	3200	✗	226981	108	1074	851
4	12:26:25.820	Users 1-3	homePageHttpRe...	5040	✗	227019	108	2277	1987
5	12:26:24.987	Users 1-1	homePageHttpRe...	6813	✗	226989	108	1892	1177
6	12:26:29.394	Users 1-4	pricingHttpRequest	4352	✓	510183	115	210	0
7	12:26:28.495	Users 1-2	pricingHttpRequest	5292	✓	510202	115	416	0
8	12:26:28.886	Users 1-5	pricingHttpRequest	5459	✓	510191	115	119	0
9	12:26:30.862	Users 1-3	pricingHttpRequest	4786	✓	510250	115	161	0
10	12:26:31.801	Users 1-1	pricingHttpRequest	6001	✓	510222	115	134	0

Here we get the message also in the view results in tree listener



There is a summary report:

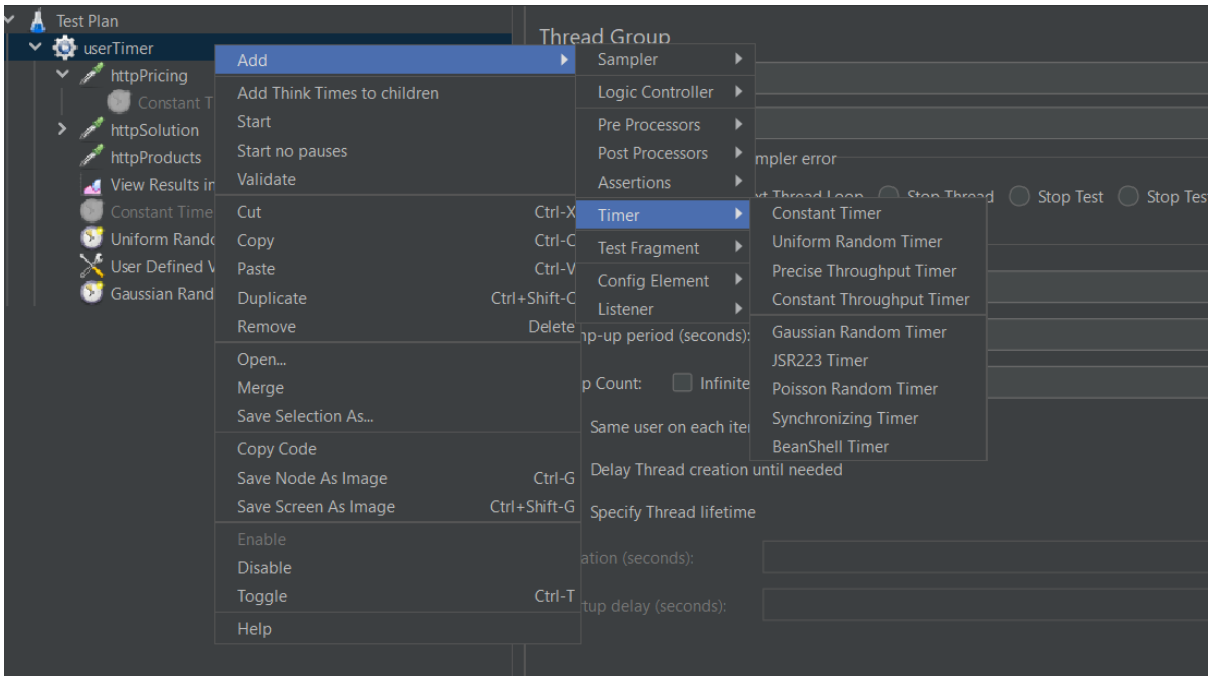


Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
homePageHttpR...	5	4076	2283	6813	1640.70	100.00%	44.0/min	162.68	0.08	226988.0
pricingHttpRequ...	5	5178	4352	6001	566.61	0.00%	32.2/min	267.68	0.06	510209.6
TOTAL	10	4627	2283	6813	1345.22	50.00%	46.8/min	280.89	0.08	368598.8

Task 5: Insert a Constant Timer to add a delay between requests in a test plan and observe the impact on the load test results.

How?

1. **Create a thread group**
2. **Add the timers**



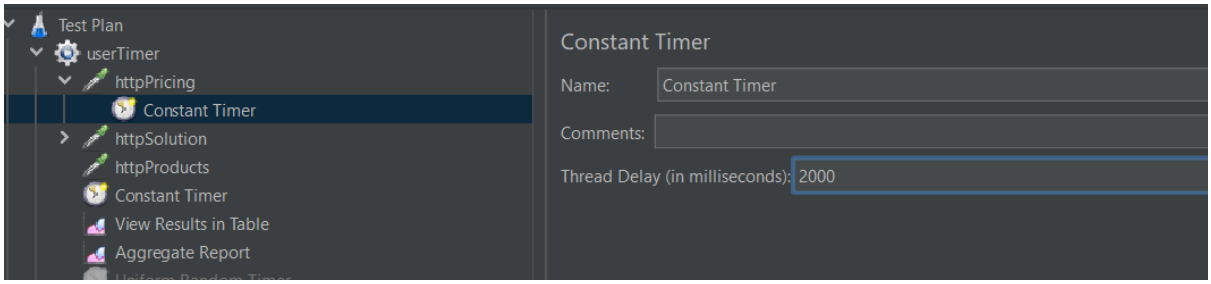
Here I use various timers like constant, uniform, and Gaussian timer etc.

a. **Contant timer:** As I examine it is used for the constant time of delays between the user request.

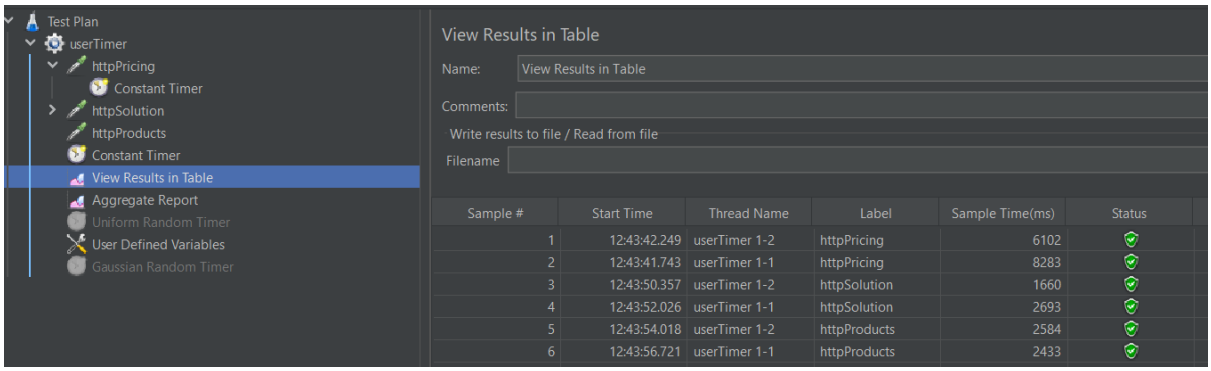
As you can see I use two constant timer here first one is inside the https request sampler and second one is inside thread.

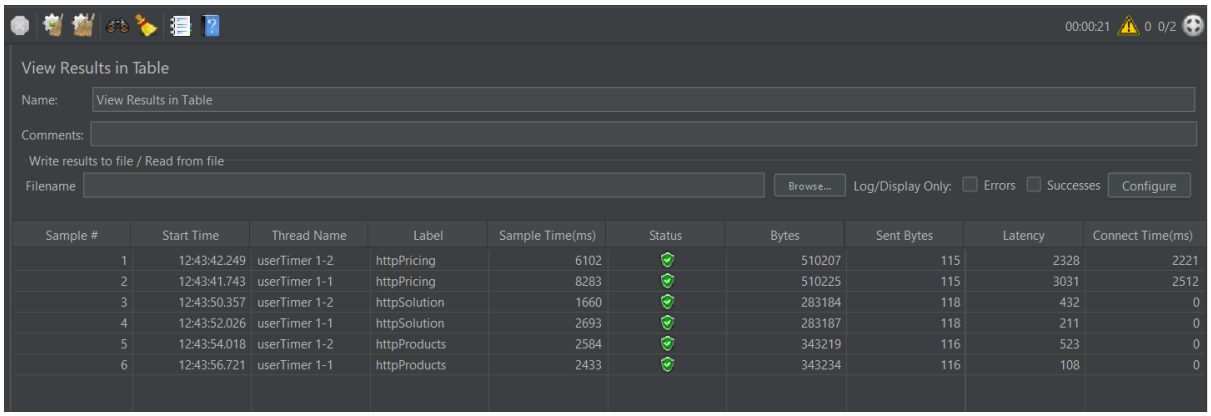
It means first constant timer is used for the http request and second work for whole thread means when I run script assume both have a 2s delay so for the first http request it takes 4 sec.

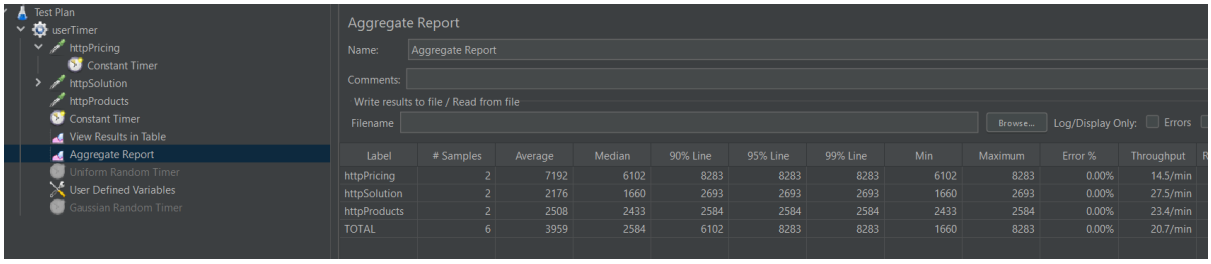
How: first delay + whole delay = 2s + 2s = 4s => first request occurs.



Result:







b. Uniform Random Timer: As we know in constant timer delay it takes contant time for the multiple user delay but in practicle we know there is no contant set of time so for that we use uniform random timer here it take random delays.

- Uniform Random Timer
  Random Delay Max
  Constant Delay Offset
- Formula:
  - 0.X * Random Delay Max + Constant Delay Offset
- X : 0-9
- Example:
  0.X * 100 + 0
  0 - 99 milli sec

Here constant delay means It takes 200 ms minimum for the request delay and random delay means it will take any random time delay between the 200 – 2000 ms.

Results: As you can see it takes less time delays here

View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

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

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	12:50:54.630	userTimer 1-2	httpPricing	5631		510157	115	825	645
2	12:50:53.674	userTimer 1-1	httpPricing	6593		510222	115	1200	965
3	12:51:02.079	userTimer 1-1	httpSolution	2187		283184	118	455	0
4	12:51:02.478	userTimer 1-2	httpSolution	2439		283134	118	95	0
5	12:51:05.176	userTimer 1-1	httpProducts	2248		343241	116	461	0
6	12:51:06.747	userTimer 1-2	httpProducts	2243		343214	116	110	0

00:00:16 0 0/2

Aggregate Report

Name: Aggregate Report

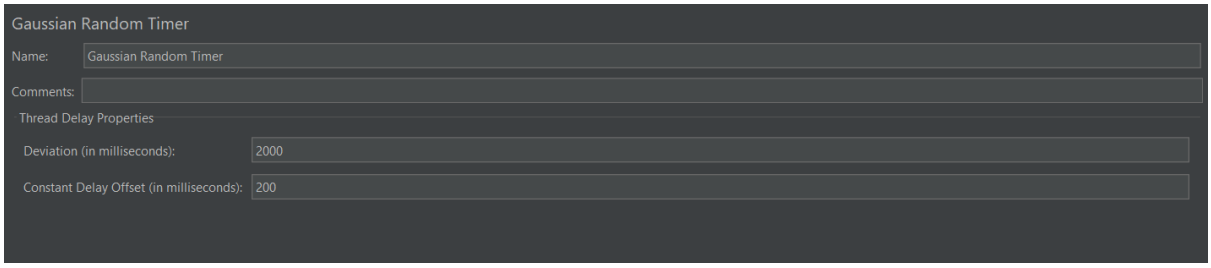
Comments:

Write results to file / Read from file

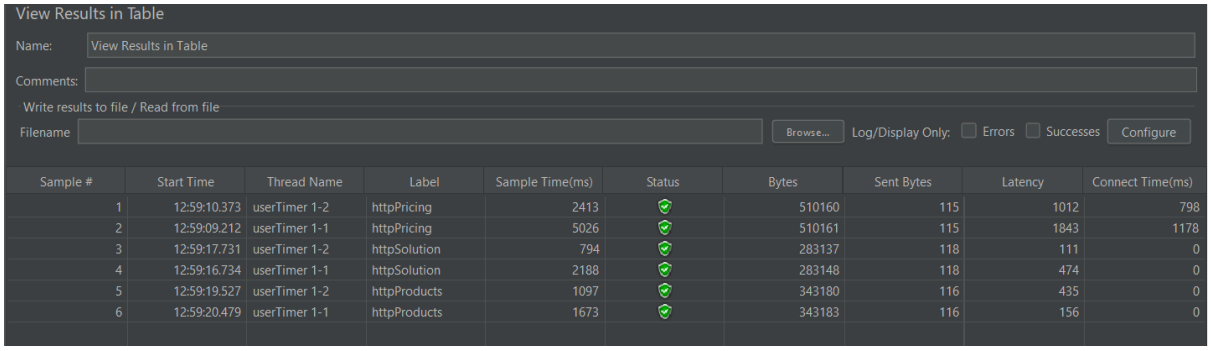
Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
httpPricing	2	6112	5631	6593	6593	6593	5631	6593	0.00%	18.2/min	151.14	0.03
httpSolution	2	2313	2187	2439	2439	2439	2187	2439	0.00%	42.3/min	194.87	0.08
httpProducts	2	2245	2243	2248	2248	2248	2243	2248	0.00%	31.5/min	175.76	0.06
TOTAL	6	3556	2248	5631	6593	6593	2187	6593	0.00%	23.5/min	144.94	0.04

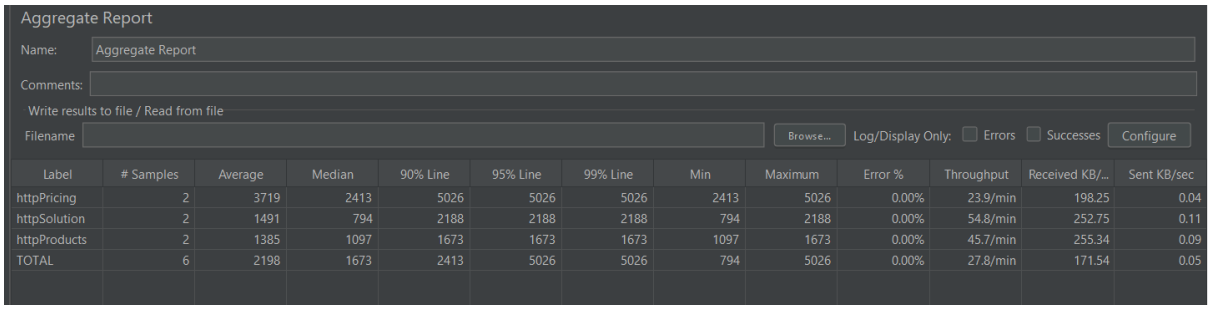
c. Gaussian timer: Delays are distributed according to a Gaussian (Normal) distribution, suitable for simulating realistic user behavior with clustering around a mean value.



Results: Here we can see it takes less time than other ones.



Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	12:59:10.373	userTimer 1-2	httpPricing	2413		510160	115	1012	798
2	12:59:09.212	userTimer 1-1	httpPricing	5026		510161	115	1843	1178
3	12:59:17.731	userTimer 1-2	httpSolution	794		283137	118	111	0
4	12:59:16.734	userTimer 1-1	httpSolution	2188		283148	118	474	0
5	12:59:19.527	userTimer 1-2	httpProducts	1097		343180	116	435	0
6	12:59:20.479	userTimer 1-1	httpProducts	1673		343183	116	156	0



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
httpPricing	2	3719	2413	5026	5026	5026	2413	5026	0.00%	23.9/min	198.25	0.04
httpSolution	2	1491	794	2188	2188	2188	794	2188	0.00%	54.8/min	252.75	0.11
httpProducts	2	1385	1097	1673	1673	1673	1097	1673	0.00%	45.7/min	255.34	0.09
TOTAL	6	2198	1673	2413	5026	5026	794	5026	0.00%	27.8/min	171.54	0.05

Based on this results here's an examination of the performance metrics:

I examined the performance data from three timers: Gaussian, Uniform Random, and Constant. Here's what I observed:

1. Response Times:

- **Gaussian Timer:** I observed that the httpPricing request took around 3010 ms on average to complete, while httpSolution took about 1043 ms, and httpProducts took around 1291 ms.
- **Uniform Random Timer:** In this case, the httpPricing request had an average response time of about 4531 ms, httpSolution took around 2174 ms, and httpProducts took around 1770 ms.
- **Constant Timer:** The httpPricing request had an average response time of about 8470 ms, httpSolution took around 6044 ms, and httpProducts took around 5026 ms.

2. Throughput:

- **Gaussian Timer:** The total throughput for all requests was approximately 171.07 requests/second.
- **Uniform Random Timer:** The total throughput for all requests was approximately 158.84 requests/second.
- **Constant Timer:** The total throughput for all requests was approximately 86.37 requests/second.

3. Error Rates:

- I observed that all timers reported a 0% error rate, indicating that all requests were successful.

4. Observations:

- The Constant Timer seemed to introduce a higher delay in the requests, resulting in longer response times compared to the other timers.
- The Gaussian Timer showed more consistent response times compared to the Uniform Random Timer, which exhibited more variability.

5. Conclusion:

- The choice of timer can significantly impact the performance characteristics observed during the load test.
- The Gaussian Timer and Uniform Random Timer provide different distributions of delays, affecting the perceived performance of the application under test.
- The Constant Timer, with its fixed delay, can be used to simulate a more predictable load on the system.

In summary, the choice of timer should be based on the specific testing requirements and the desired behavior of the load test.

6. Design a load test plan for a web page with increasing users (thread) over time and analyze the results.

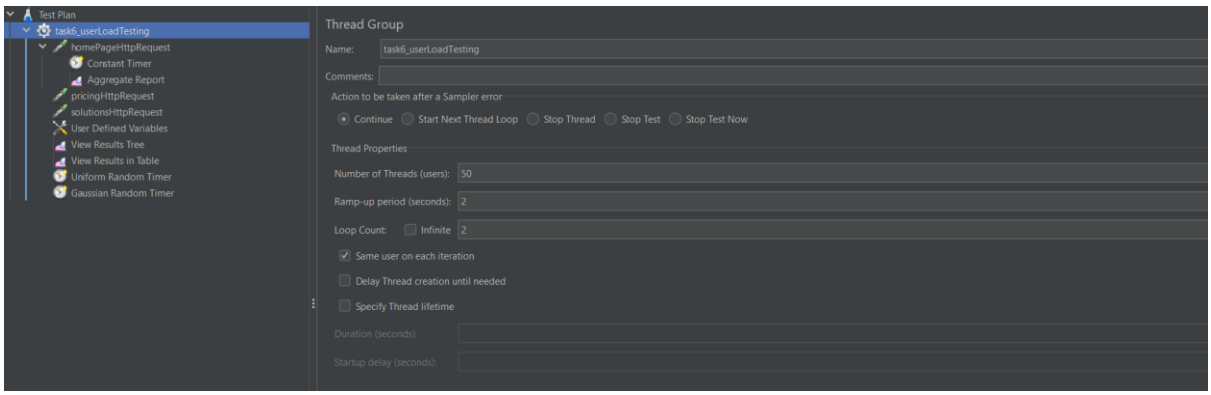
Load testing is conducted to examined how a system performs under normal and peak load conditions. It helps identify the maximum operating capacity of an application as well as any bottlenecks and potential issues that may arise when multiple users access the application simultaneously.

(In GUI Mode)

How?

- Create a threads group
    1. Set the No. of users (from 10 – 100 and so on)
    2. Increase the ramp up time
    3. Also increase the Interations
- Create a multiple http sampler request
- and inside it create some delays

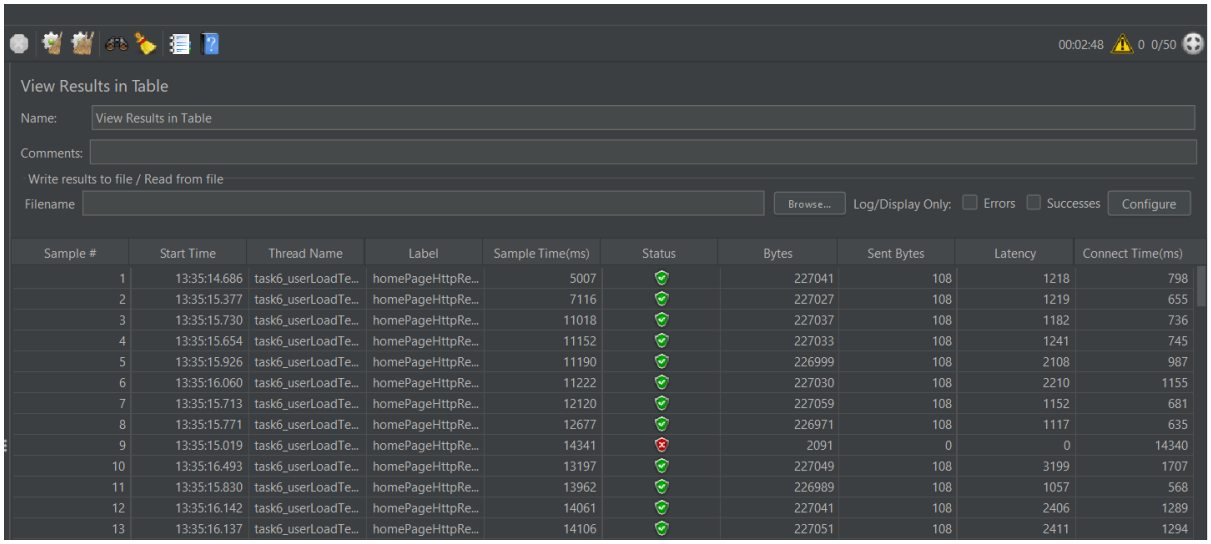
For 50 users:



For 100 users:

After ensuring these steps it will create load.

Results: (for 50 users)



00:02:48 0 0/50

View Results in Table

Name: View Results in Table

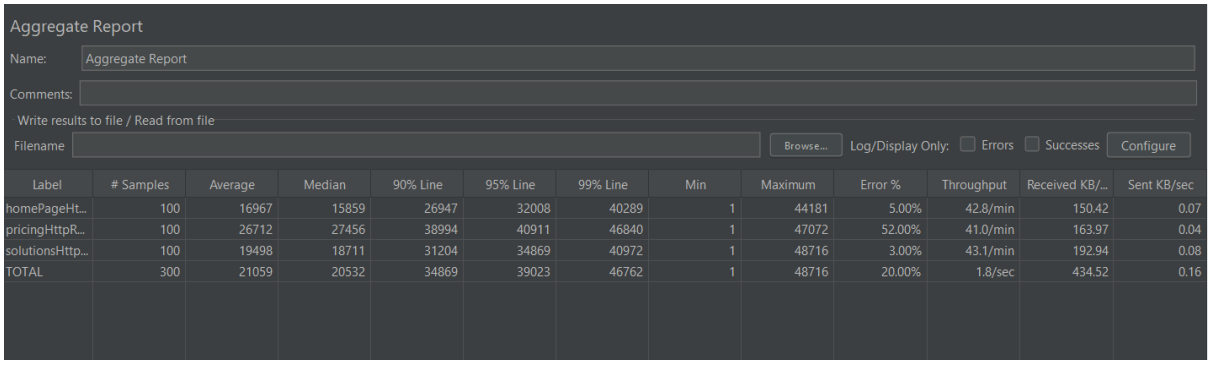
Comments:

Write results to file / Read from file

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

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	13:35:14.686	task6_userLoadTe...	homePageHttpRe...	5007		227041	108	1218	798
2	13:35:15.377	task6_userLoadTe...	homePageHttpRe...	7116		227027	108	1219	655
3	13:35:15.730	task6_userLoadTe...	homePageHttpRe...	11018		227037	108	1182	736
4	13:35:15.654	task6_userLoadTe...	homePageHttpRe...	11152		227033	108	1241	745
5	13:35:15.926	task6_userLoadTe...	homePageHttpRe...	11190		226999	108	2108	987
6	13:35:16.060	task6_userLoadTe...	homePageHttpRe...	11222		227030	108	2210	1155
7	13:35:15.713	task6_userLoadTe...	homePageHttpRe...	12120		227059	108	1152	681
8	13:35:15.771	task6_userLoadTe...	homePageHttpRe...	12677		226971	108	1117	635
9	13:35:15.019	task6_userLoadTe...	homePageHttpRe...	14341		2091	0	0	14340
10	13:35:16.493	task6_userLoadTe...	homePageHttpRe...	13197		227049	108	3199	1707
11	13:35:15.830	task6_userLoadTe...	homePageHttpRe...	13962		226989	108	1057	568
12	13:35:16.142	task6_userLoadTe...	homePageHttpRe...	14061		227041	108	2406	1289
13	13:35:16.137	task6_userLoadTe...	homePageHttpRe...	14106		227051	108	2411	1294

Aggregate report:



Aggregate Report

Name: Aggregate Report

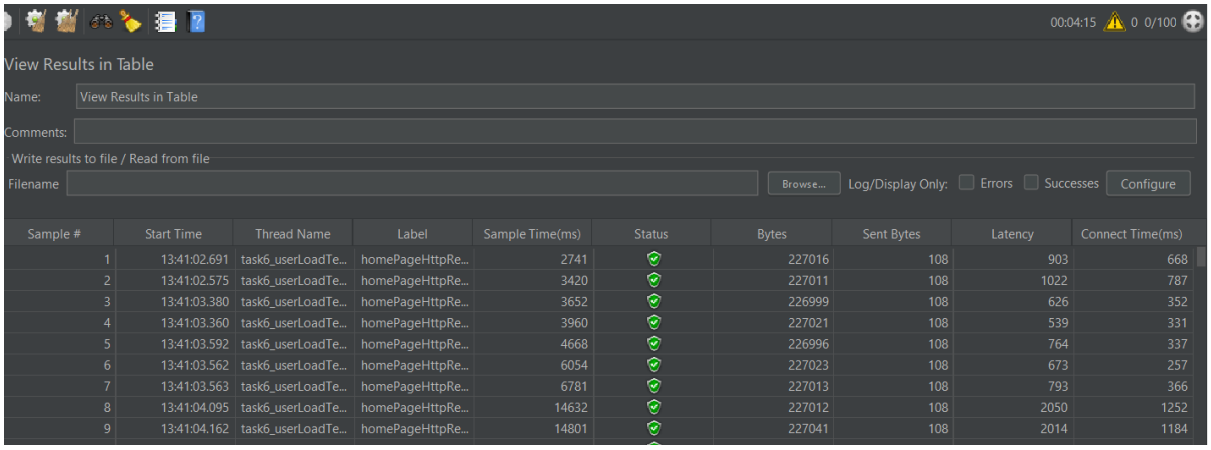
Comments:

Write results to file / Read from file

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

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
homePageHt...	100	16967	15859	26947	32008	40289	1	44181	5.00%	42.8/min	150.42	0.07
pricingHttpR...	100	26712	27456	38994	40911	46840	1	47072	52.00%	41.0/min	163.97	0.04
solutionsHttp...	100	19498	18711	31204	34869	40972	1	48716	3.00%	43.1/min	192.94	0.08
TOTAL	300	21059	20532	34869	39023	46762	1	48716	20.00%	1.8/sec	434.52	0.16

For 100 users:



View Results in Table

Name: View Results in Table

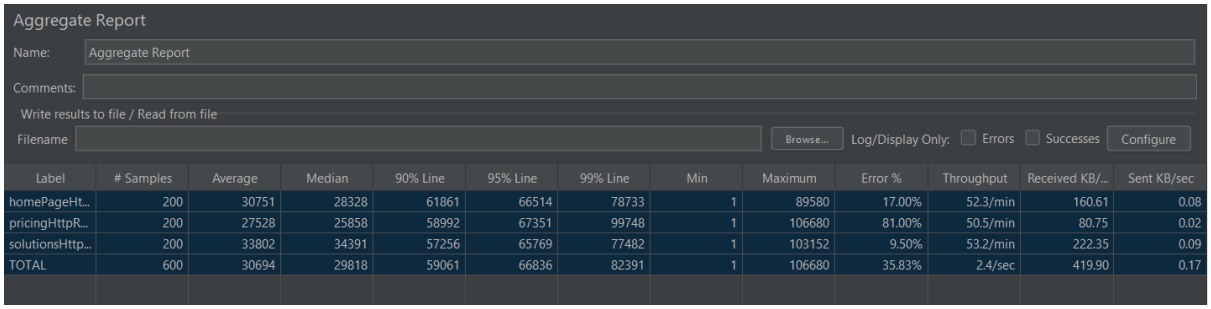
Comments:

Write results to file / Read from file

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

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	13:41:02.691	task6_userLoadTe...	homePageHttpRe...	2741		227016	108	903	668
2	13:41:02.575	task6_userLoadTe...	homePageHttpRe...	3420		227011	108	1022	787
3	13:41:03.380	task6_userLoadTe...	homePageHttpRe...	3652		226999	108	626	352
4	13:41:03.360	task6_userLoadTe...	homePageHttpRe...	3960		227021	108	539	331
5	13:41:03.592	task6_userLoadTe...	homePageHttpRe...	4668		226996	108	764	337
6	13:41:03.562	task6_userLoadTe...	homePageHttpRe...	6054		227023	108	673	257
7	13:41:03.563	task6_userLoadTe...	homePageHttpRe...	6781		227013	108	793	366
8	13:41:04.095	task6_userLoadTe...	homePageHttpRe...	14632		227012	108	2050	1252
9	13:41:04.162	task6_userLoadTe...	homePageHttpRe...	14801		227041	108	2014	1184

Aggregate report (100 users)



Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

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

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
homePageHt...	200	30751	28328	61861	66514	78733	1	89580	17.00%	52.3/min	160.61	0.08
pricingHttpR...	200	27528	25858	58992	67351	99748	1	106680	81.00%	50.5/min	80.75	0.02
solutionsHttp...	200	33802	34391	57256	65769	77482	1	103152	9.50%	53.2/min	222.35	0.09
TOTAL	600	30694	29818	59061	66836	82391	1	106680	35.83%	2.4/sec	419.90	0.17

My Observation:

I conducted load tests with 50 and 100 users to evaluate our system's performance. Here's what I observed:

1. **Response Times**:

    - With 50 users, the home page loaded in approximately 158.59 ms, the pricing page in 274.56 ms, and the solutions page in 187.11 ms on average.
    - As the number of users increased to 100, the response times also increased. The home page took around 283.28 ms, the pricing page took about 258.58 ms, and the solutions page took approximately 343.91 ms to load.

2. **Throughput**:

    - The throughput for 50 users ranged from 0.683 to 0.719 requests/second, while for 100 users, it increased to about 0.840 to 0.887 requests/second.

3. **Error Rates**:

    - The error rates for all requests were low, indicating that the system handled the load well without significant errors.

4. **Conclusion**:

- Overall, our system performed reasonably well under the tested load conditions, with response times and throughput remaining within acceptable limits for most requests.
- However, there may be areas for improvement, particularly for the solutions page, which showed slightly slower performance with increased user load.

These findings provide valuable insights into our system's performance and can help us make informed decisions to optimize its performance further.

(In non-GUI)

In non-GUI or using cmd is more efficient in many ways:

1. **Efficiency:** CMD is more efficient for running large scale because it doesn't have any further load or overhead like GUI, also we can consume few resources and handle more threads and iterations.

2. **Headless Execution:** Command-line mode allows you to run JMeter tests on servers or machines without a graphical environment.

3. **Performance:** Command-line mode can potentially provide better performance for the same test plan compared to GUI mode, especially for tests with high loads or long durations.

How we use non-GUI(cmd) for the Load test?

1. **Go to cmd**
2. **Navigate to the Jmeter bin directory:** cd /path/to/your/apache-jmeter/bin
3. **Run the Jmeter test:**

   jmeter -n -t /path/to/your/test_plan.jmx -l /path/to/your/results.jtl

- -n: This option tells JMeter to run the test in non-GUI mode.
- -t: Specify the path to your JMX test plan file.
- -l: Specify the path to save the results in JTL format.

```
C:\Users\amanv\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin> jmeter -n -t "C:\Users\amanv\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\tas
k6_userLoadTesting.jmx" -l "C:\Users\amanv\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\results1.jtl"
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
```

Result:

```
Creating summariser <summary>
Created the tree successfully using C:\Users\amanv\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\task6_userLoadTesting.jmx
Starting standalone test @ 2024 Mar 27 14:00:48 IST (1711528248014)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary +      5 in 00:00:12 =    0.4/s Avg:  4535 Min:  2411 Max:  6903 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary +     47 in 00:00:30 =    1.6/s Avg: 15935 Min:  1989 Max: 32448 Err:     1 (2.13%) Active: 50 Started: 50 Finished: 0
summary =     52 in 00:00:42 =    1.2/s Avg: 14839 Min:  1989 Max: 32448 Err:     1 (1.92%)
summary +     77 in 00:00:30 =    2.5/s Avg: 17868 Min:     1 Max: 61948 Err:    23 (29.87%) Active: 50 Started: 50 Finished: 0
summary =    129 in 00:01:12 =    1.8/s Avg: 16647 Min:     1 Max: 61948 Err:    24 (18.60%)
summary +     59 in 00:00:30 =    2.0/s Avg: 22596 Min:  4610 Max: 67293 Err:    12 (20.34%) Active: 47 Started: 50 Finished: 3
summary =    188 in 00:01:42 =    1.8/s Avg: 18514 Min:     1 Max: 67293 Err:    36 (19.15%)
summary +     72 in 00:00:30 =    2.4/s Avg: 19047 Min:     1 Max: 41969 Err:    17 (23.61%) Active: 24 Started: 50 Finished: 26
summary =    260 in 00:02:12 =    2.0/s Avg: 18662 Min:     1 Max: 67293 Err:    53 (20.38%)
summary +     40 in 00:00:22 =    1.8/s Avg: 10809 Min:   768 Max: 26211 Err:     1 (2.50%) Active: 0 Started: 50 Finished: 50
summary =    300 in 00:02:34 =    1.9/s Avg: 17615 Min:     1 Max: 67293 Err:    54 (18.00%)
Tidying up ...    @ 2024 Mar 27 14:03:22 IST (1711528402112)
... end of run
```

For generate the report:

- Command: jmeter -g /path/to/your/results.jtl -o /path/to/your/report-folder

```
C:\Users\amanv\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin>jmeter -g  "C:\Users\amanv\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\result
s1.jtl" -o  "C:\Users\amanv\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\report-folder1"
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
```

This we get in the report folder-

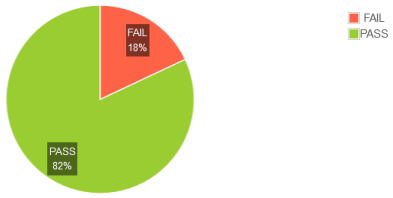
Test and Report information

Source file	"results1.jtl"
Start Time	"3/27/24, 2:00 PM"
End Time	"3/27/24, 2:03 PM"
Filter for display	""

APDEX (Application Performance Index)

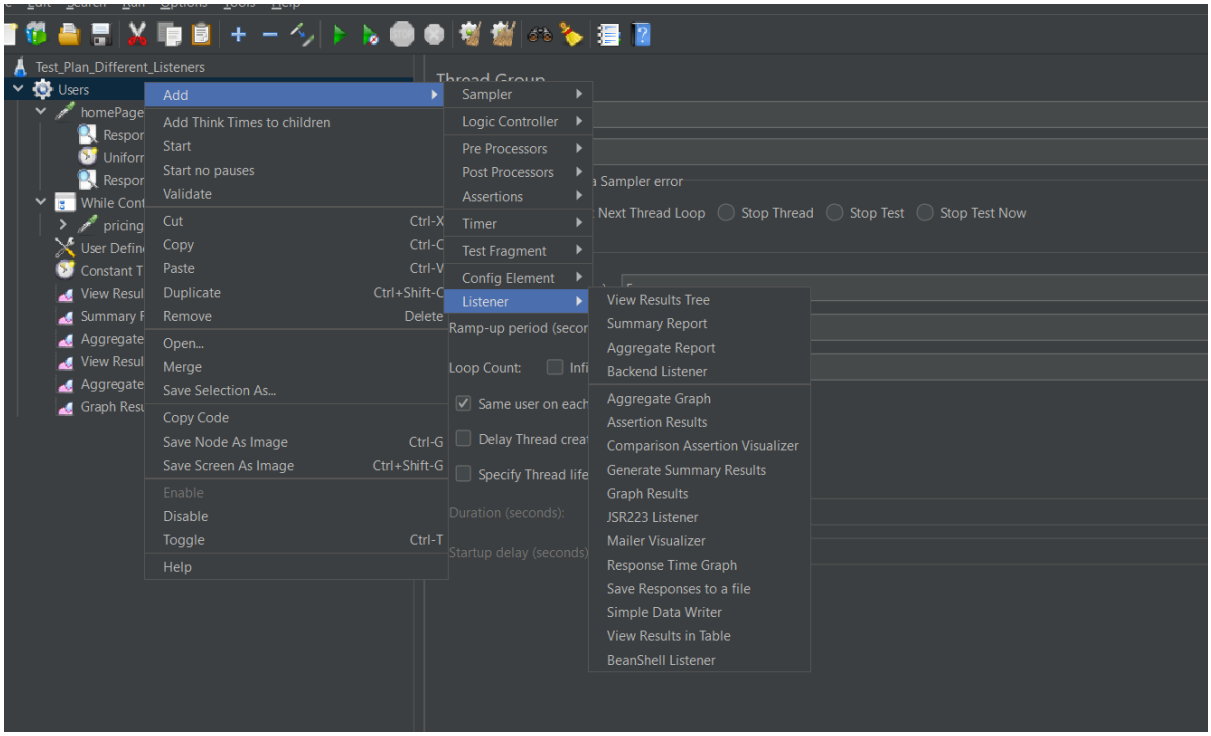
Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.003	**500 ms**	**1 sec 500 ms**	**Total**
0.000	500 ms	1 sec 500 ms	homePageHttpRequest
0.000	500 ms	1 sec 500 ms	pricingHttpRequest
0.010	500 ms	1 sec 500 ms	solutionsHttpRequest

Requests Summary



7. Run a test and use different Listeners (e.g., View Results Tree, Aggregate Report) to analyze and interpret the data collected during the test

How to create a listener?



I have used different listeners for the better result optimization or analysis:

1. View result tree

2. Summary Report

3. Aggregate Report

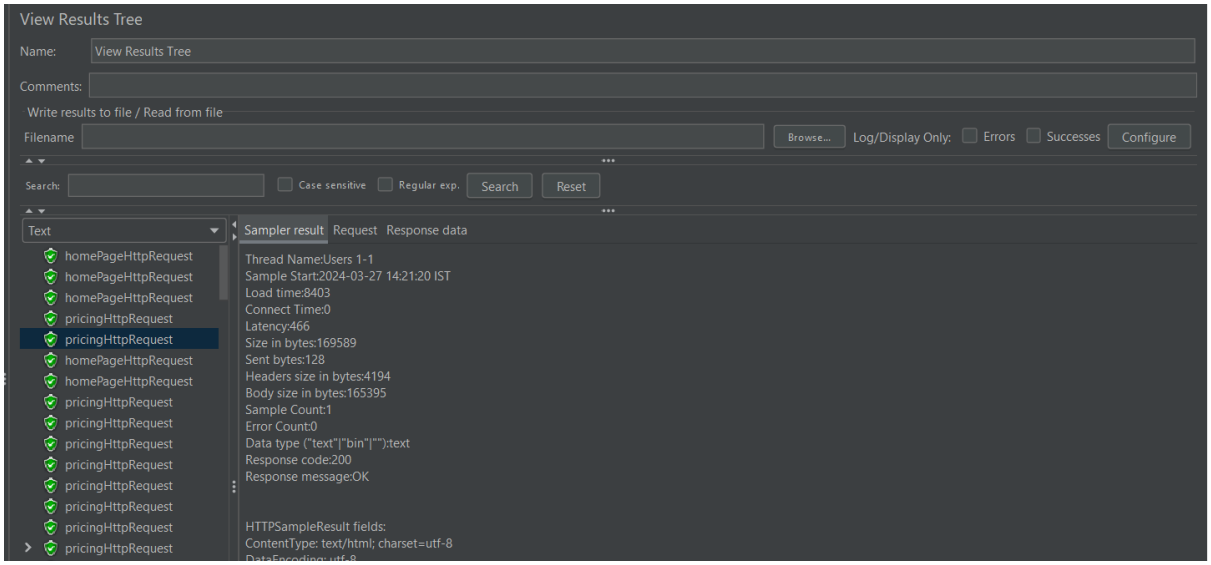
4. View Result in Table

5. Aggregate Result

6. Graph Result

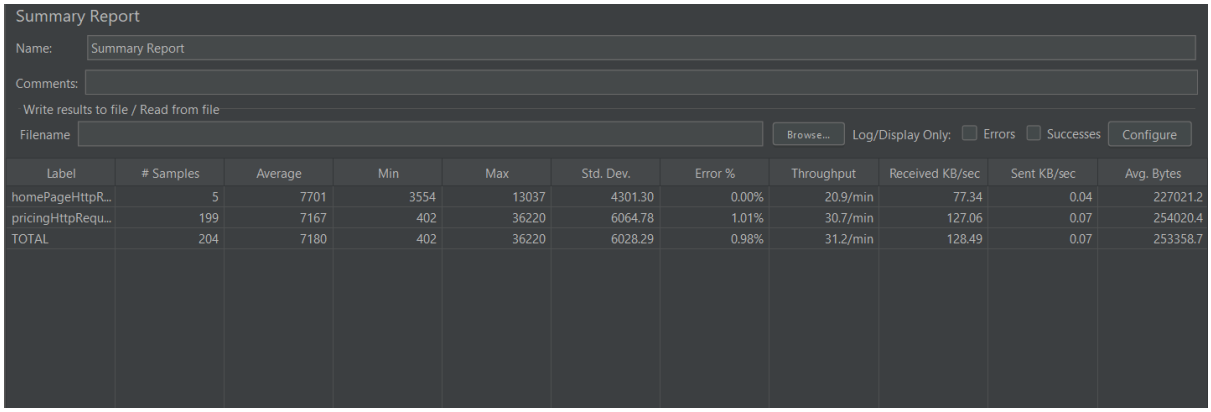
1. **View Result Tree:**

- I observe detailed information about each request and response in a tree-like structure, allowing me to examine the data exchanged between the client and server.



2. **Summary Report:**

- I observe a summary of the test results, including key metrics like average response time and error count, which I examine to understand the overall performance of the test.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
homePageHttpR...	5	7701	3554	13037	4301.30	0.00%	20.9/min	77.34	0.04	227021.2
pricingHttpRequ...	199	7167	402	36220	6064.78	1.01%	30.7/min	127.06	0.07	254020.4
TOTAL	204	7180	402	36220	6028.29	0.98%	31.2/min	128.49	0.07	253358.7

3. **Aggregate Report:**

- I observe aggregated data from all the samples, such as average response time and throughput, which I examine to get a comprehensive view of the test performance.

Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

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

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
homePageHt...	5	7701	4851	12852	13037	13037	3554	13037	0.00%	20.9/min	77.34	0.04
pricingHttpR...	209	7111	5698	14336	18213	26124	402	36220	0.96%	30.5/min	128.81	0.07
TOTAL	214	7125	5679	14336	17863	26124	402	36220	0.93%	30.9/min	130.15	0.07

4. **View Result in Table:**

- I observe the test results in a tabular format, which I examine to analyze each sample's details like request URL, response code, and response time.

View Results in Table

Name: View Results in Table

Comments:

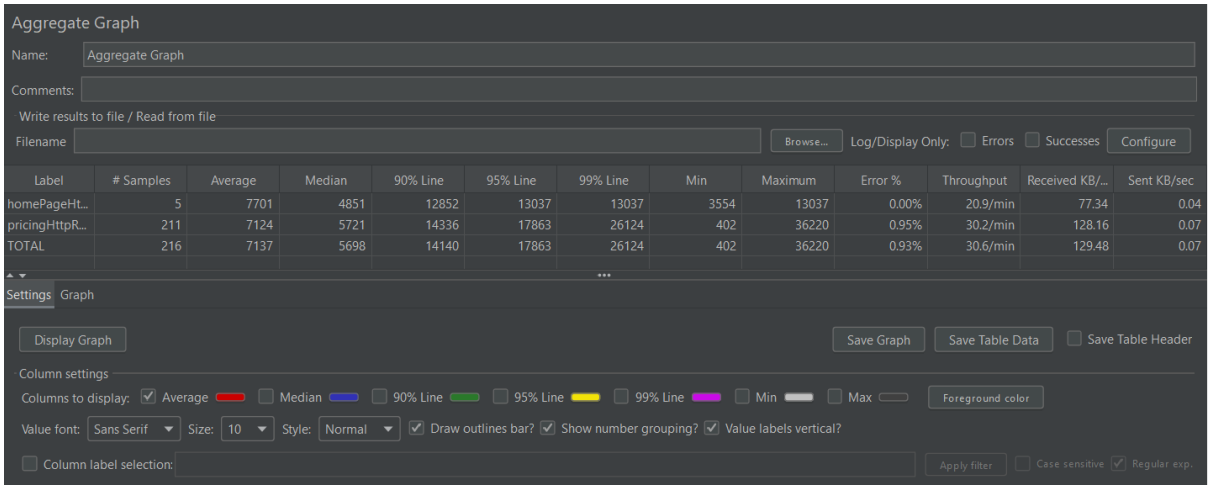
Write results to file / Read from file

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

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	14:21:16.189	Users 1-1	homePageHttpRe...	3554		227000	108	544	371
2	14:21:16.330	Users 1-4	homePageHttpRe...	4211		226981	108	463	334
3	14:21:15.729	Users 1-3	homePageHttpRe...	4851		227037	108	972	832
4	14:21:20.819	Users 1-3	pricingHttpReque...	6349		54780	114	2231	0
5	14:21:20.017	Users 1-1	pricingHttpReque...	8403		169589	128	466	0
6	14:21:16.944	Users 1-2	homePageHttpRe...	12852		227043	108	2856	2203
7	14:21:17.025	Users 1-5	homePageHttpRe...	13037		227045	108	2119	1119
8	14:21:27.388	Users 1-3	pricingHttpReque...	3585		227024	108	111	0
9	14:21:20.788	Users 1-4	pricingHttpReque...	12877		189380	114	791	0
10	14:21:30.314	Users 1-5	pricingHttpReque...	3511		132508	119	508	0
11	14:21:30.046	Users 1-2	pricingHttpReque...	6741		227039	108	115	0
12	14:21:31.225	Users 1-3	pricingHttpReque...	8182		283187	118	864	0
13	14:21:37.052	Users 1-2	pricingHttpReque...	6523		187496	125	404	0

5. **Aggregate Graph:**

- observe a graphical representation of the aggregated results over time, which I examine to identify trends and patterns in the test performance.



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
homePageHt...	5	7701	4851	12852	13037	13037	3554	13037	0.00%	20.9/min	77.34	0.04
pricingHttpR...	211	7124	5721	14336	17863	26124	402	36220	0.95%	30.2/min	128.16	0.07
TOTAL	216	7137	5698	14140	17863	26124	402	36220	0.93%	30.6/min	129.48	0.07

6. **Graph Results:**

- I observe a graph showing the response times of each sample over time, which I examine to pinpoint any anomalies or areas of concern in the test execution.