Performance Testing

- **What is Performance Testing?**

Performance testing is a type of testing done to check how well a system performs under various conditions. It is like checking how fast a car goes or how smoothly it runs on different roads.
We do performance testing to make sure that our software or system can handle its workload effectively. It helps us understand if our system can handle many users or a large amount of data without slowing down or crashing.

Types of Performance Testing:

1. **Load Testing**: Checks how the system behaves under expected loads. It's like checking how many people can use a website at once without it getting slow.
2. **Stress Testing**: Pushes the system beyond its limits to see when it breaks. It's like testing how much weight a bridge can hold before collapsing.
3. **Endurance Testing**: Tests how the system performs over a long period. It's like checking if a marathon runner can keep going without getting tired.
4. **Scalability Testing**: Measures the system's ability to handle growing amounts of work. It's like seeing if a business can handle more customers without losing quality.
5. **Volume Testing**: Checks how the system handles large amounts of data. It's like testing if a storage container can hold all the items it's supposed to.

- **What is JMeter?**

The Apache JMeter is an open-source, purely Java-based software. The software is used to perform performance testing, functional testing, and load testing of web applications.

It helps check how well a website or application performs under different conditions, like lots of users or heavy traffic.

We use JMeter because it's easy to use, widely used in the industry, and can simulate real-life scenarios. It helps us find problems like slow response times or crashes before they happen in the real world. Plus, it's free, so anyone can use it to make sure their software runs smoothly.

- **Why use Jmeter:**

1. **Free to Use**: JMeter doesn't cost anything, and it's easy to use. We can automate our work with it.
2. **Tests Many Types of Applications**: JMeter can test both simple and complex applications, like websites with dynamic content.
3. **Loads and Stresses Testing**: It helps check how well our software handles lots of users at once. This way, we can find out how many users our server can handle before it slows down or crashes.
4. **Works with Multiple Threads**: JMeter's framework allows different parts of a test to happen at the same time. This helps simulate real-world situations better.
5. **Shows Results in Graphs**: JMeter presents test results in easy-to-understand graphs and tables. This makes it simpler to analyze how well your software performed.
6. **Works on Any Platform**: Because JMeter is built in Java, it can run on any type of computer or server that supports Java.

TASK-1:

- **TEST PLAN:**

In JMeter, the Test Plan is like a blueprint for our testing. It is where we outline what we want to test and how we want to do it that is where we lay out our testing strategy and organize all the elements of our test.

What is it?

The Test Plan is the main container where we organize all the elements of our test. As the cover page of a book tells what's inside.

Where is it?

We can find the Test Plan in the left pane of the JMeter GUI. It is the first thing we see when we open JMeter below the icon section.

How to Access It?

To create or access a Test Plan, we simply open JMeter and start a new project. The default Test Plan is automatically created for us.

What Can We Do with It?

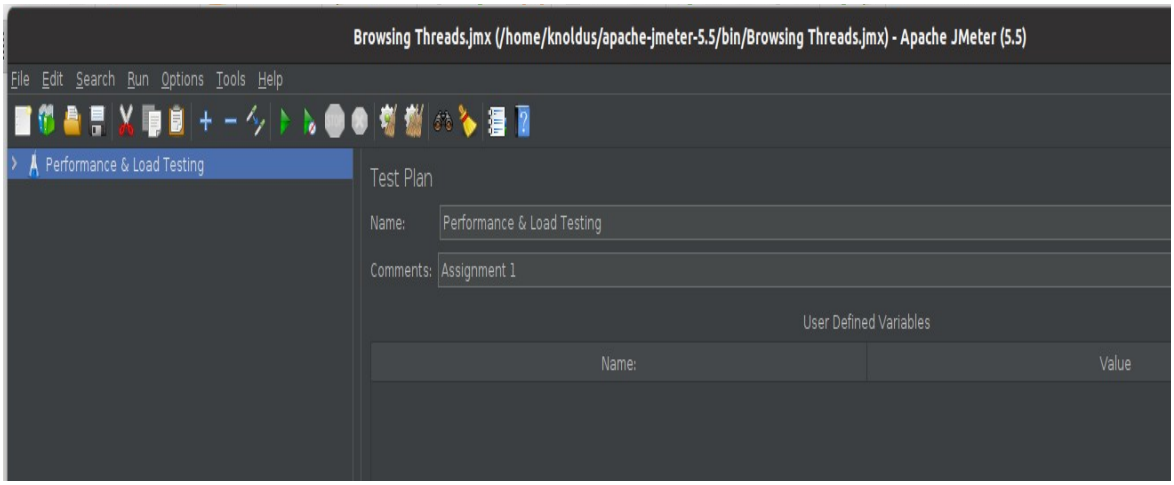
In the Test Plan, we can add different elements like Thread Groups (to simulate users), Samplers (to send requests to our website or app), and Listeners (to see the results of our tests).

Organizing Your Tests

We can organize our tests within the Test Plan by adding folders and labels. This helps keep everything neat and tidy.

Running the Tests

Once we've set up our Test Plan, we can run our tests by clicking the "Run" button/icon. JMeter will then execute all the elements within our Test Plan and give us the results.



- **THREAD GROUP:**

Thread Group is like a group of people (or threads) that will perform the actions in the test plan.Thread Group is a fundamental component in JMeter that helps us simulate user behavior and control the execution of our performance tests. It's where we define the parameters for our test scenarios, such as the number of users etc

What is it?

The Thread Group is where we define how many users (threads) will be simulated and how they will behave during the test.

How to Use It/Find it?

To use the Thread Group, we simply add it to our Test Plan by right-clicking on the Test Plan, choosing "Add", then "Thread (Users)", and finally "Thread Group".

Options:

Within the Thread Group, we can set options like the number of users (threads), ramp-up time (in how much time users are added), and loop count (how many times the test will run). These options help simulate real-world usage of our application.

Simulating Users:

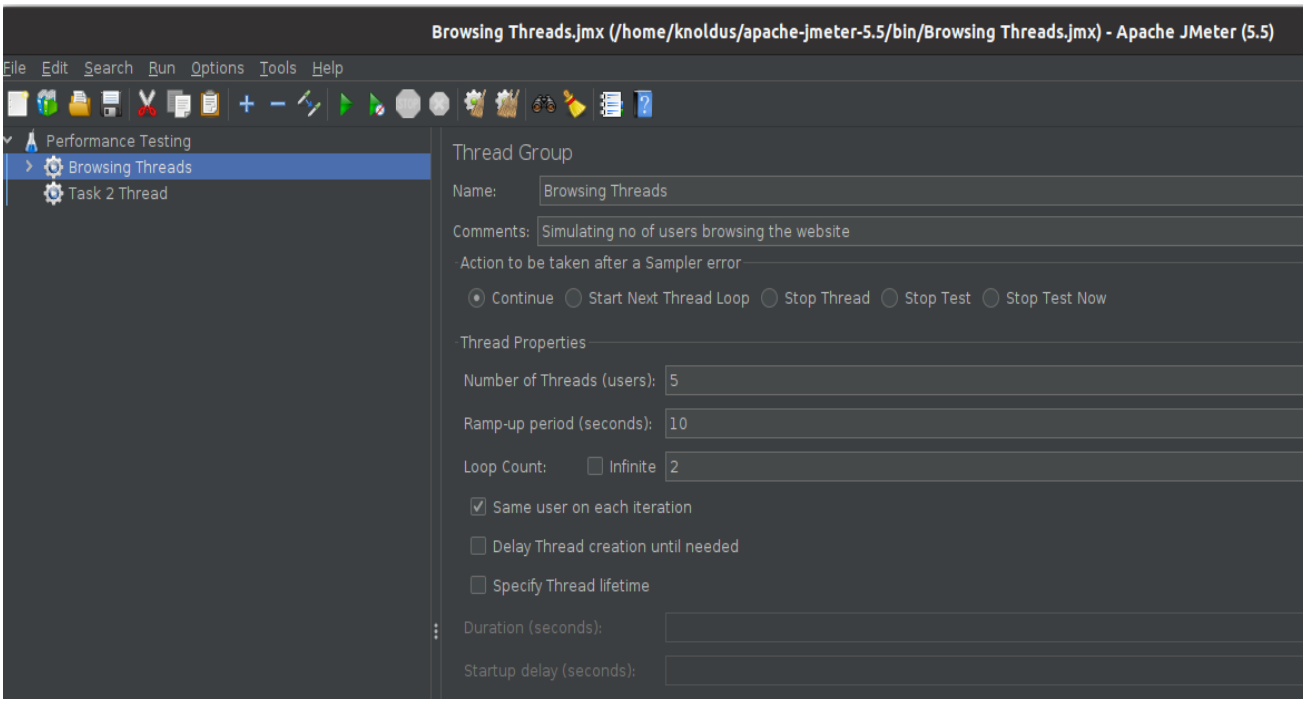
The Thread Group allows us to simulate multiple users accessing our application concurrently. Each thread represents a user performing actions like clicking links or submitting forms, etc.

Controlling Test Execution:

With the Thread Group, we can control how our test runs, such as starting all users at once or gradually adding them over time. This helps simulate different scenarios and analyze how our application performs under various loads.

Analyzing the Results:

- After running the test, you'd use Listeners in JMeter to analyze the results.
- You'd look at metrics like response time, throughput, and error rates to see how well your website handles adding products to the shopping cart under different loads.



- **SAMPLERS:**

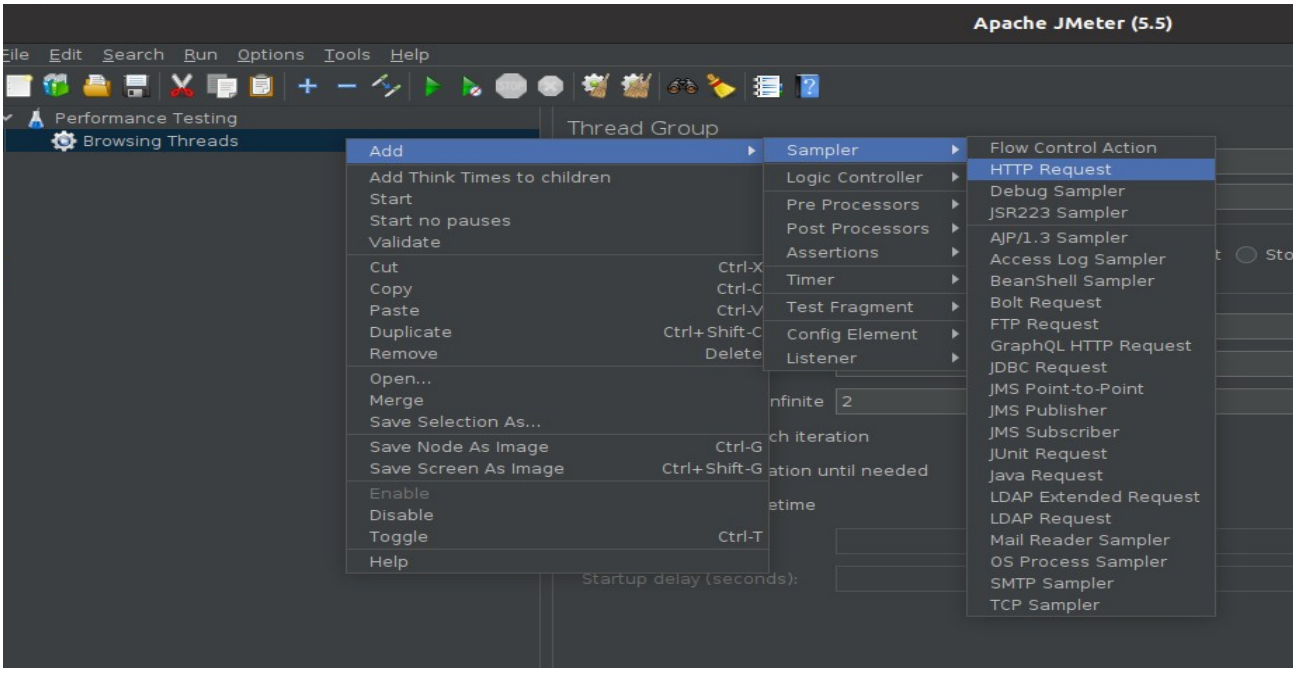
  Samplers are like messengers that deliver requests to your website or app. They help us test how well our system responds to different actions/requests.They allow us to test how our system responds to different actions and identify any performance issues.

  What are they?

  Samplers are the tools we use to send requests to our website or app during testing.It is used to test performce via hitting different types of requests.

  How to Use Them/Find them:

  We can find Samplers after we have created the Thread Group.To use a Sampler, we simply add it to our Test Plan by right-clicking on the Thread Group or other element, then selecting "Add" -> "Sampler" -> choosing the type of Sampler we want to use (Here, HTTP Request)

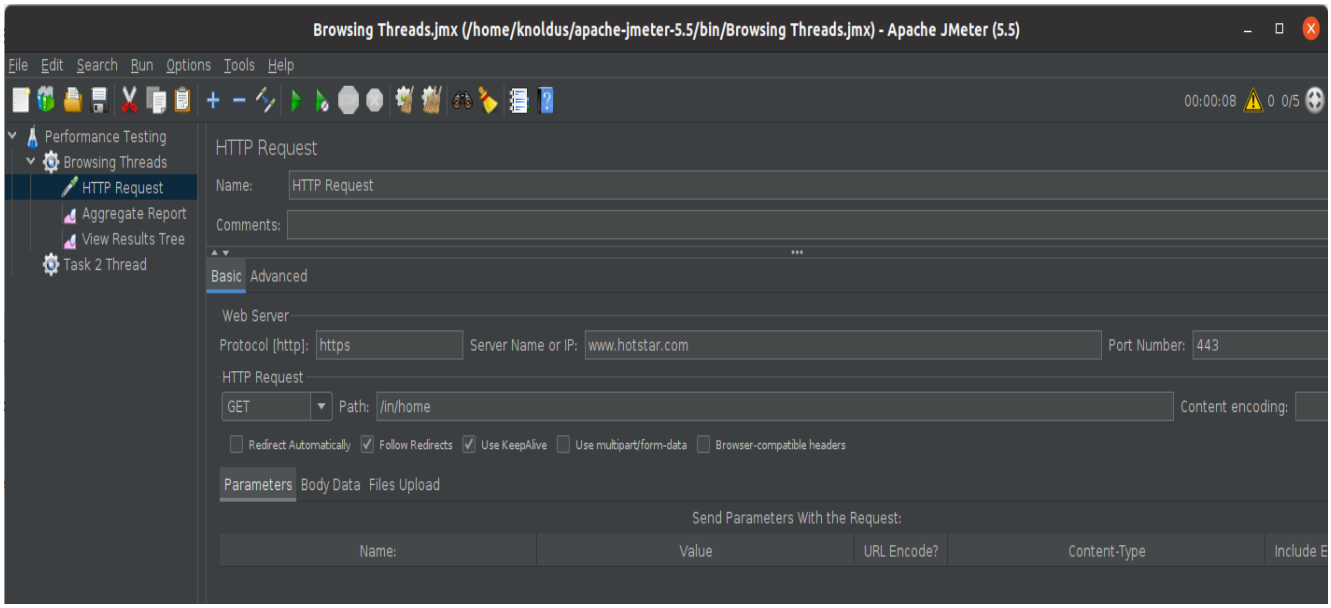


What Can We Do with Them?

Samplers come in different types, like HTTP Request, FTP Request, JDBC Request, etc. Each type is used to send a specific type of request to our website or app for testing the same.

Options:

- Protocol (HTTP): Specifies the communication protocol, such as HTTP or HTTPS, to use for the request.
- Server Name or IP: Identifies the server where the request will be sent, either by domain name or IP address.
- Port Number: Specifies the port on the server to which the request will be sent, typically port 80 for HTTP and port 443 for HTTPS.
- HTTP Requests (GET, POST, PUT, PATCH, DELETE): Defines the type of HTTP request to send, such as GET to retrieve data or POST to submit data.
- Path: Specifies the path or endpoint on the server to which the request will be sent.
- Content Encoding: Determines how the content of the request body is encoded, such as UTF-8 or gzip.
- Parameters: Includes additional parameters to be sent with the request, such as query parameters or form data.



Testing Different Actions:

With Samplers, you can test various actions like loading a webpage, submitting a form, downloading a file, or interacting with an API.

Real-World Example:

Let us consider in E-commerce website, if we want to test how well it handles adding products to the shopping cart.

Adding a Product to Cart: We will have to use the HTTP Request Sampler in JMeter to simulate a user clicking the "Add to Cart" button. In the sampler options, we will specify the server name (our website's domain), the path to the product page, and the method (usually POST) along with any necessary parameters like the product ID.

- **LISTENERS:**

Listeners in JMeter are tools that show us the results of our tests. They're like reporters, telling you what happened during our test. We use them to see how well our software performed and if there are any problems. Listeners provide various formats for analyzing test results.

What are Listeners?

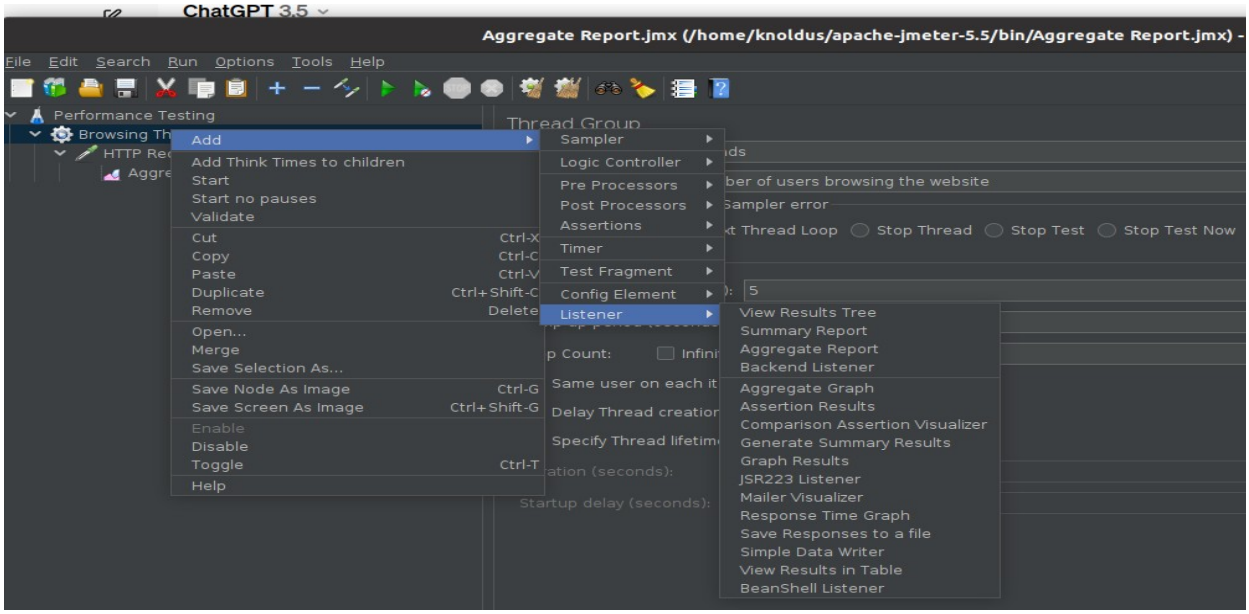
Listeners in JMeter are like reporters. They show us what happened during our test.

Where to Find Listeners?

We can find Listeners in the left pane of JMeter, just below the Test Plan.

How to Use Listeners?

To use Listeners, we simply add them to our Test Plan, and it'll automatically show us the results of our test when we run it.



Types of Listeners:

- View Results Tree: Shows detailed information about each request and response during the test.
- Aggregate Report: Summarizes the results of our test in a table format, showing averages, medians, and percentiles.
- Graph Results: Displays the results of our test in graphical format, making it easy to visualize performance trends.
- Summary Report: Provides a summary of the test results, including the number of samples, average response time, and throughput.
- Response Assertion: Checks if the response from the server contains specific text or matches a pattern.

Real-World Example:

Let us imagine we are part of a team responsible for a popular e-commerce website. Our task is to ensure that the website can handle a large number of users during peak shopping seasons, like Big Billion Day Sale.

- View Results Tree: We add the View Results Tree Listener to our Test Plan. During testing, this Listener shows detailed information about each user's interaction with the website. For example, we can see the response time for each page load, any errors encountered, and the content of the server response.

- Aggregate Report: To get an overall view of website performance, we include the Aggregate Report Listener. After running the test, this Listener provides key performance metrics such as average response time, minimum and maximum response times, and throughput (number of requests per second). With this data, we can identify performance bottlenecks and areas for improvement.

  Aggregate Report listener in JMeter provides a tabular view of various performance metrics collected during the test execution. Its components include:

  Label:
- Represents the name of the sampler or the transaction being tested. It identifies the specific request or action within our test plan.

  Samples:
- Indicates the number of samples (requests) sent for the corresponding label.

  Average:
- Represents the average response time of all samples for the corresponding label. It's calculated by summing up the response times of all samples and dividing by the total number of samples.

  Median:
- Refers to the median response time of all samples for the corresponding label. It's the value below which 50% of the samples fall.

  90% Line:
- Represents the response time below which 90% of the samples fall. It's a percentile value indicating the response time for most of the samples.

95% Line:

- Similar to the 90% Line, but represents the response time below which 95% of the samples fall.

99% Line:

- Similar to the 90% and 95% lines, but represents the response time below which 99% of the samples fall.

Min:

- Indicates the minimum response time observed among all samples for the corresponding label.

Max:

- Indicates the maximum response time observed among all samples for the corresponding label.

Error%:

- Represents the percentage of samples that resulted in errors (e.g., HTTP errors, assertion failures) for the corresponding label.

Throughput:

- Refers to the number of requests processed per unit of time (requests/second) for the corresponding label. It indicates the workload the server is handling.

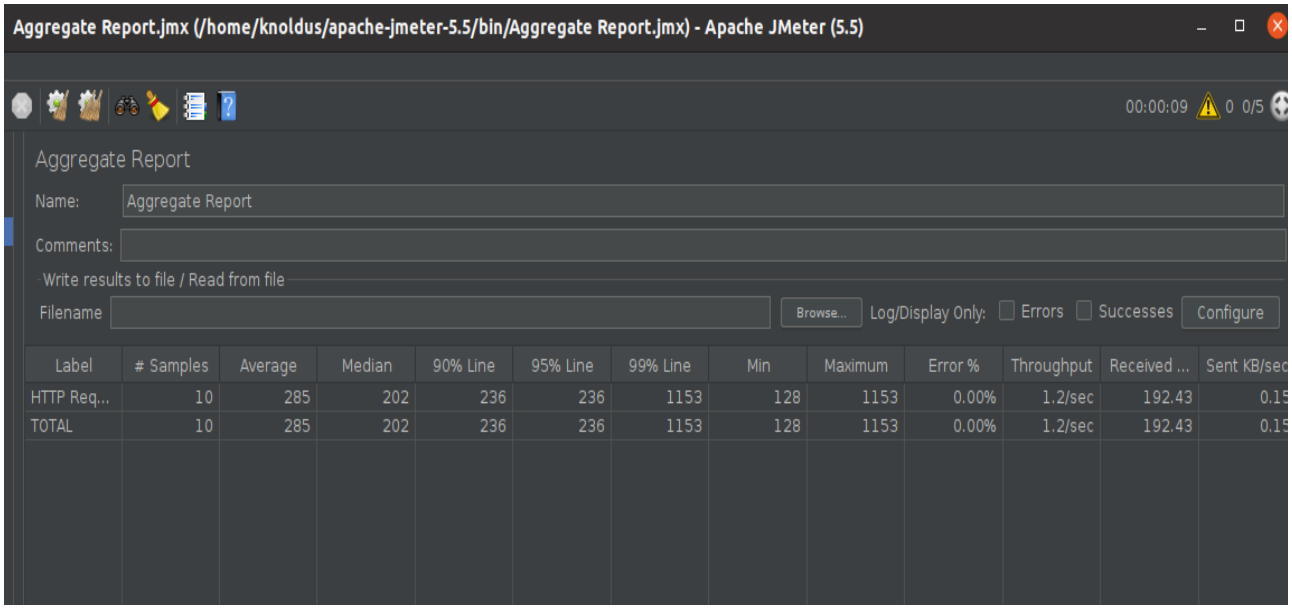
Received:

- Indicates the number of bytes received in response to the samples for the corresponding label.

Sent:

- Indicates the number of bytes sent as part of the samples for the corresponding label.

These components collectively provide insights into the performance and behavior of our application under test. By analyzing the Aggregate Report, we can identify performance bottlenecks, errors, and overall system health during load testing.



The ***View Results Tree*** is a listener in JMeter that displays detailed information about the samples (requests) sent during a test.

The main components of the View Results Tree include:

- Sampler Result:

  This section provides details about each sample (request) sent during the test. It includes information such as:

  - Sample Label: Name of the sampler (HTTP request, JDBC request, etc.).
  - Sample Time: Time taken for the sample to execute.
  - Success: Indicates whether the sample was successful or not.
  - Response Code: HTTP response code returned by the server.
  - Response Message: Brief description of the response.
  - Bytes: Size of the response in bytes.
  - Assertion Results: Results of any assertions applied to the sample.

- Request:

  This section displays the details of the request sent to the server. It includes:

  - Method: HTTP method used for the request (GET, POST, etc.).
  - URL: URL of the request.
  - Parameters: Parameters sent with the request.
  - Headers: Request headers.

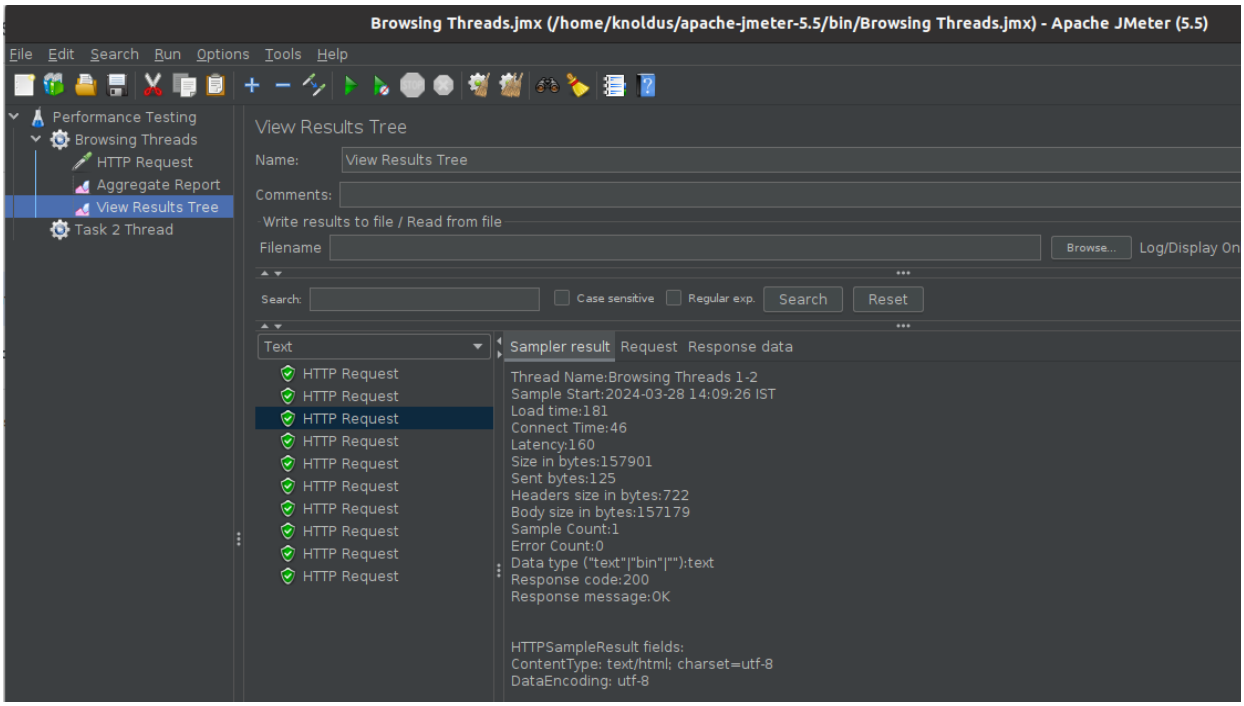
- Response:

  This section shows the response received from the server. It includes:

  - Response Code: HTTP response code returned by the server.
  - Response Message: Brief description of the response.
  - Headers: Response headers.
  - Body: Response body, which may contain HTML, JSON, XML, etc.

- Response Data:

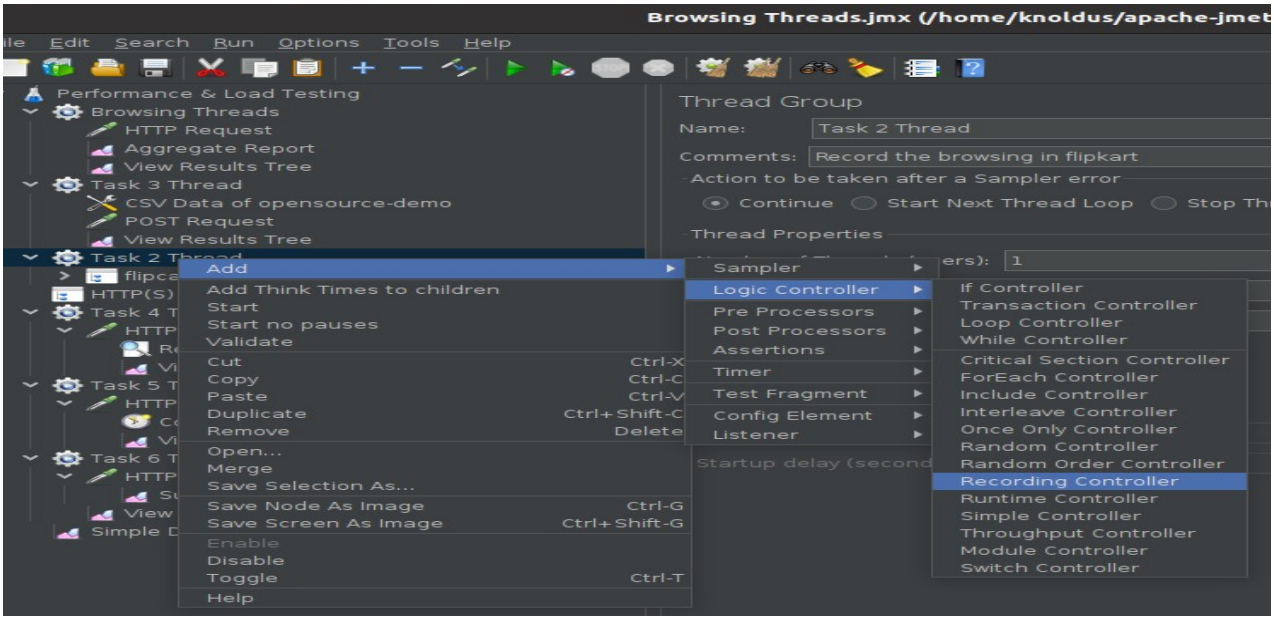
  If the sample includes response data (e.g., HTML content), it will be displayed in this section. This helps us analyze the content of the response in detail.



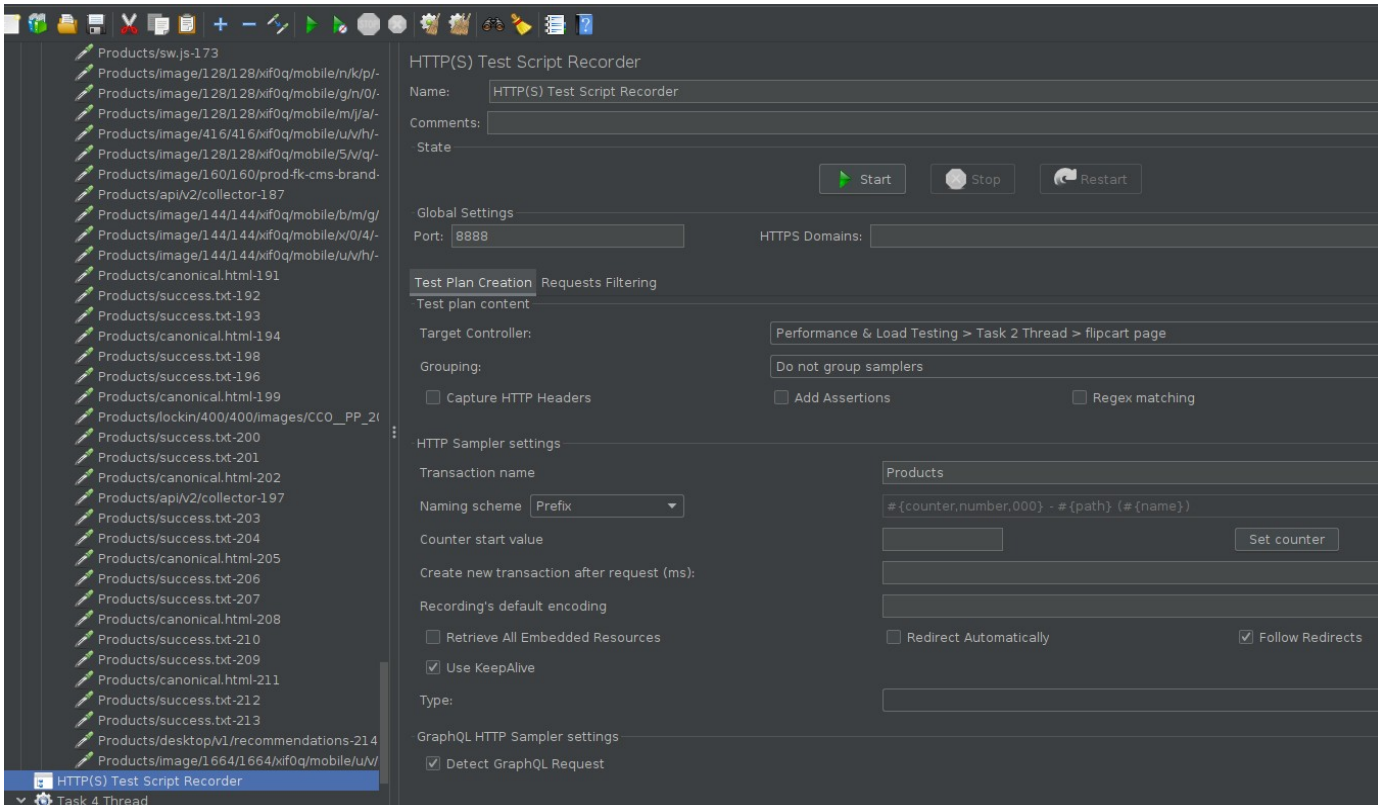
TASK-2:

- Steps for configuration

➔ Right-click on "Test Plan" then select "Add" -> "Logic Controller" -> "Recording Controller".



➔ Right-click on "Test Plan" again and select "Add" -> "Non-Test Elements" -> "HTTP(S) Test Script Recorder".



- Configure the HTTP(S) Test Script Recorder:

➔ Set the "Port" to an available port number (default is 8888).

➔ Check the box next to "Redirect Automatically".

➔ Optionally, we can specify any URL patterns we want to exclude from recording in the "URL Patterns to Exclude" field.

- Configure Browser Proxy Settings

➔ Open your web browser (e.g., Chrome, Firefox).

➔ Go to browser settings or preferences.

➔ Search for "Proxy" settings.

➔ Configure the proxy settings to use JMeter's HTTP(S) Test Script Recorder.

➔ Set the proxy hostname to "localhost".

➔ Set the proxy port to the port number specified in JMeter (default is 8888).

- Start Recording

  In JMeter, click the "Start" button on the HTTP(S) Test Script Recorder.

  Starting browse the website we want to record.

  Performing actions like navigating through pages, searching product, etc.

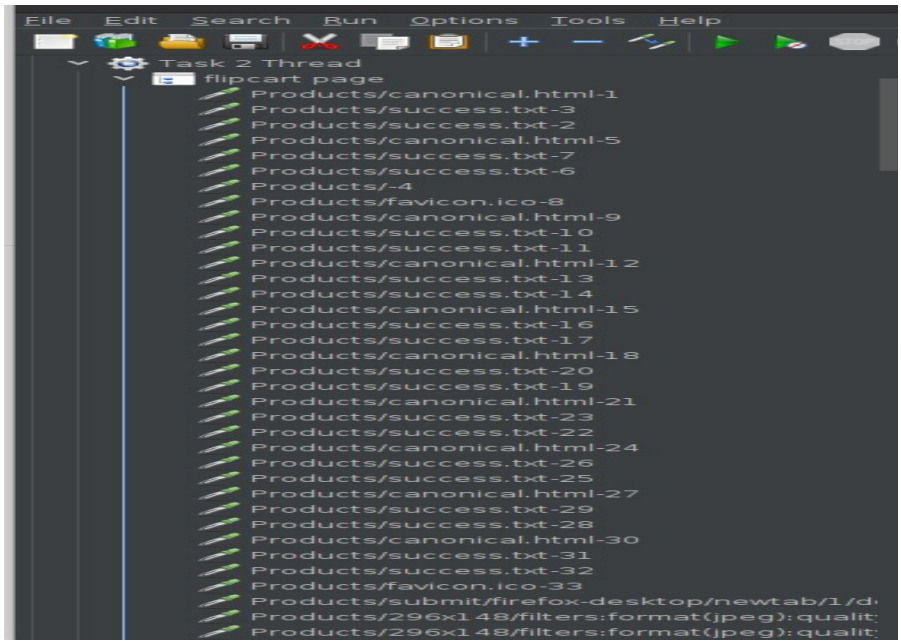
  It will capture the HTTP requests and responses generated by our browser.

- Stop Recording

  Once we've finished recording, go back to JMeter.

  Click the "Stop" button on the HTTP(S) Test Script Recorder.

  JMeter will stop capturing requests, and we'll see them listed under the "Recording Controller" in the Test Plan tree.



TASK-3

- **CONFIG ELEMENT:**

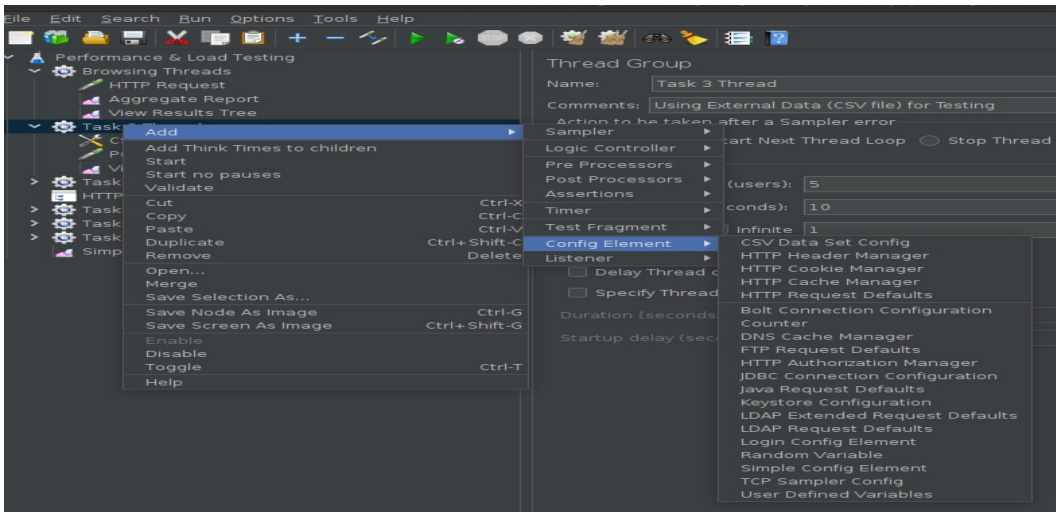
Config Element is used to configure settings or variables that are used across different parts of your test plan. It's like setting up the environment for your tests.

What is it?

A Config Element is a component in JMeter that helps you set up configurations or variables that your test plan needs.

Where is it?

You can find Config Elements in the left pane of the JMeter GUI, alongside other elements like Samplers and Listeners.



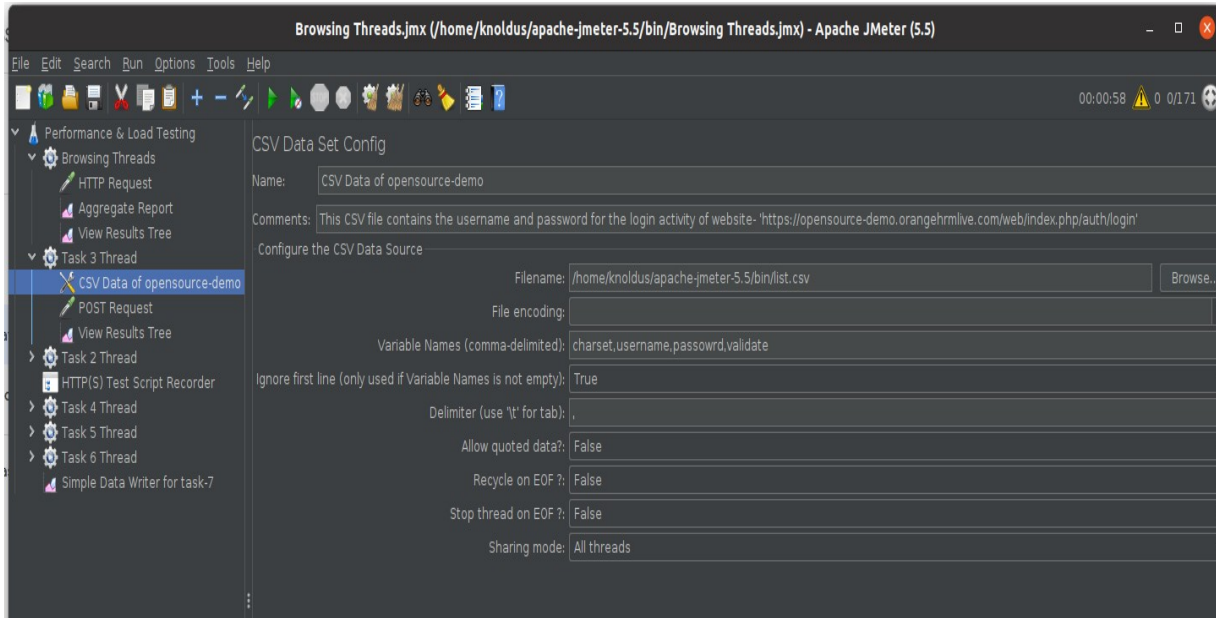
Common Config Elements:

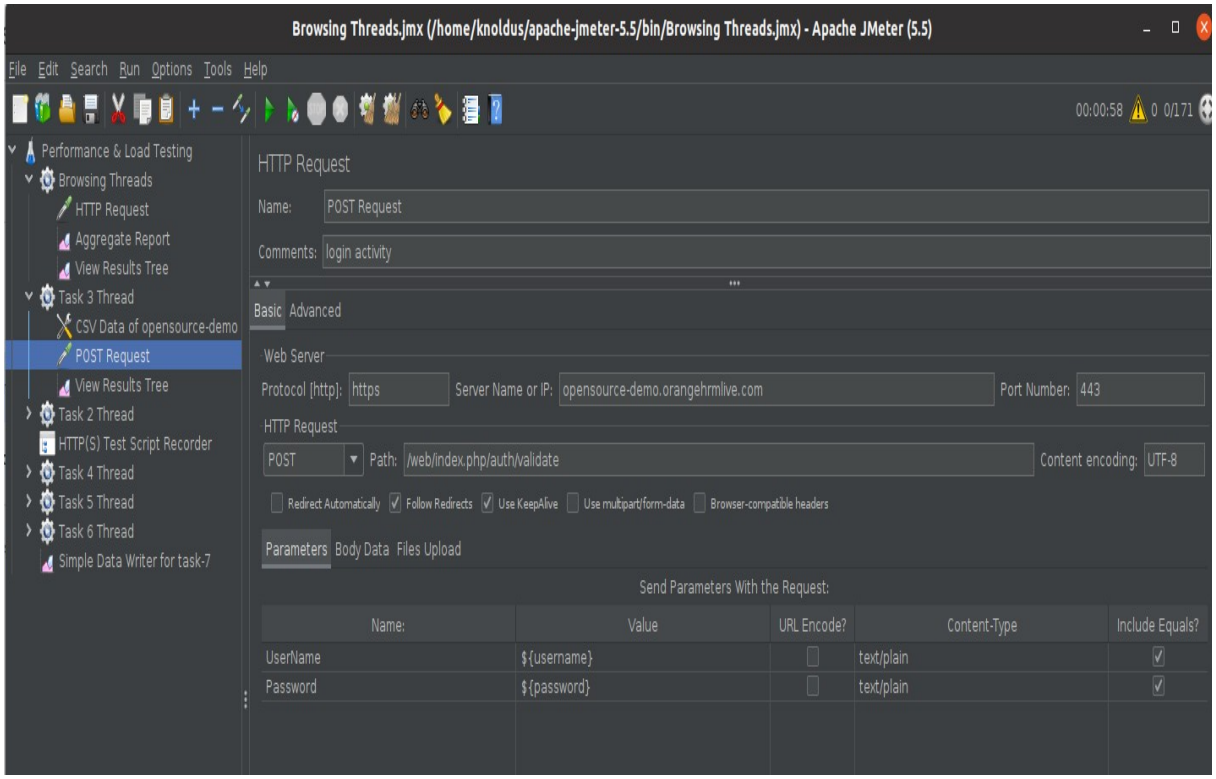
- CSV Data Set Config: Used to read data from a CSV file and use it in your test.
- HTTP Request Defaults: Sets default values for HTTP requests, like server name and port number.
- HTTP Cookie Manager: Manages cookies sent by the server during HTTP requests.
- User Defined Variables: Allows you to define custom variables that can be used throughout your test plan.

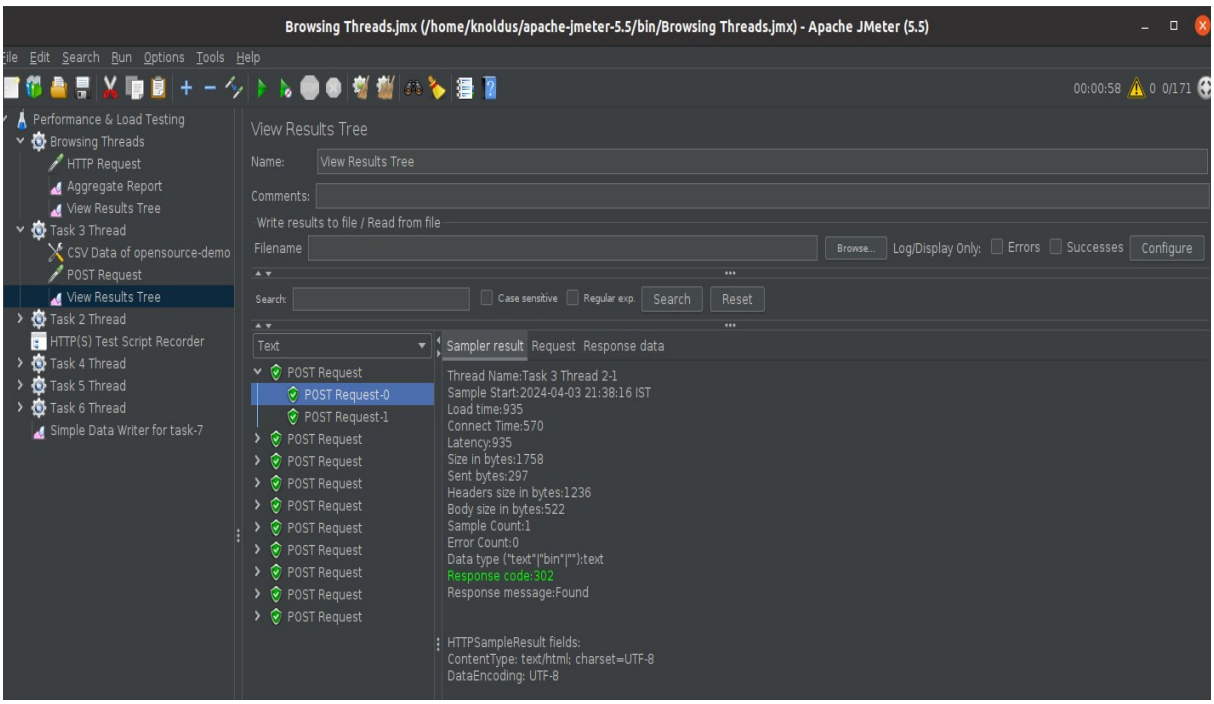
Real-World Example:

Let us consider we're testing a demo website. We might use the CSV Data Set Config to load a list of product names or login credentials from a CSV file. Then, we can use these variables in our test plan to simulate different users adding items to their carts or loging at

same time or making purchases.







TASK-4:

- **ASSERTIONS:**

Assertions in JMeter are like checkpoints that verify if certain conditions are met during testing. They help ensure that our responses from the server or website are correct.

Where to Find Assertions:

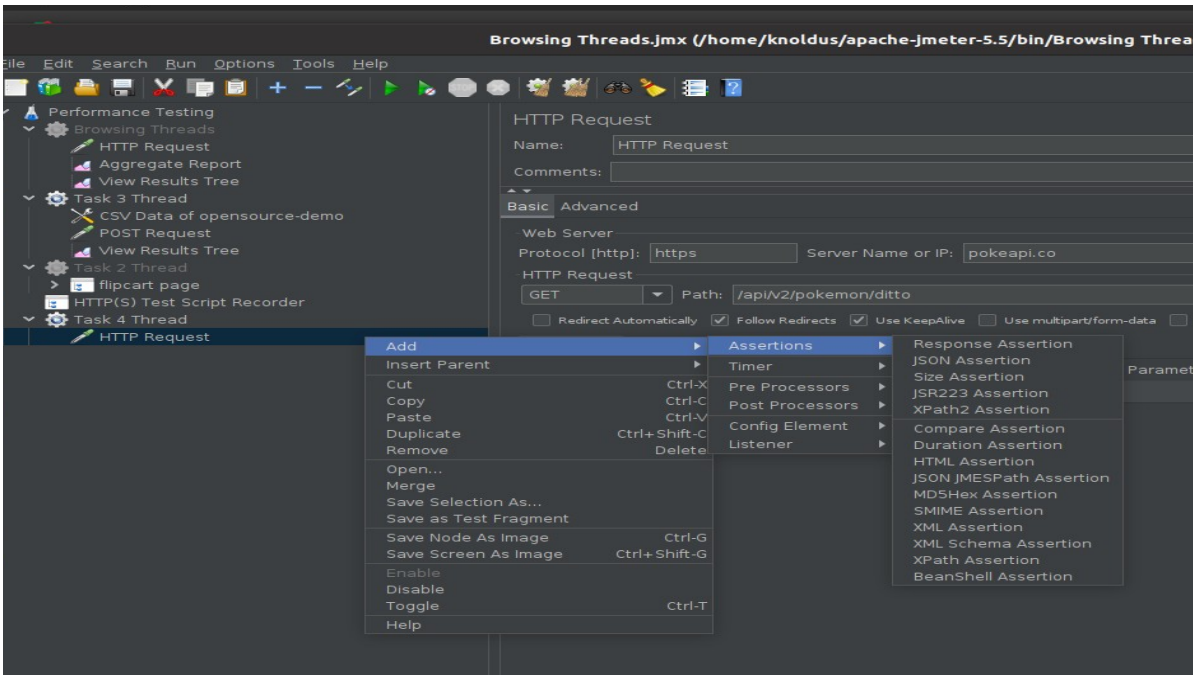
We can add Assertions to specific samplers in our Test Plan. They're located in the same menu where we find Thread Groups and other elements.

How to Use Assertions:

Add an Assertion to a sampler by right-clicking on the sampler, then go to Add -> Assertions.

Choose the type of Assertion you want to use, such as Response Assertion or Duration Assertion.

Configure the Assertion settings to define what conditions should be checked.



Commonly Used Assertions:

- Response Assertion: Checks if the server response contains certain text or meets specific criteria.
- Duration Assertion: Verifies if the response time of a request falls within a specified range.
- Size Assertion: Validates if the response size is within certain limits.
- XPath Assertion: Checks if the XML response matches an XPath expression.

The Response Assertion is used to add criteria to test if a response meets certain conditions. The components of the Response Assertion include:

Applies to :

This option allows you to choose whether the assertion applies to the main sample only or to both the main sample and its sub-samples.

- Main Sample: Asserts against the main request/response.
- Main sample and sub-samples: Asserts against all requests/responses, including sub-samples.

Field to Test:

This option specifies which part of the response to test against the assertion criteria.

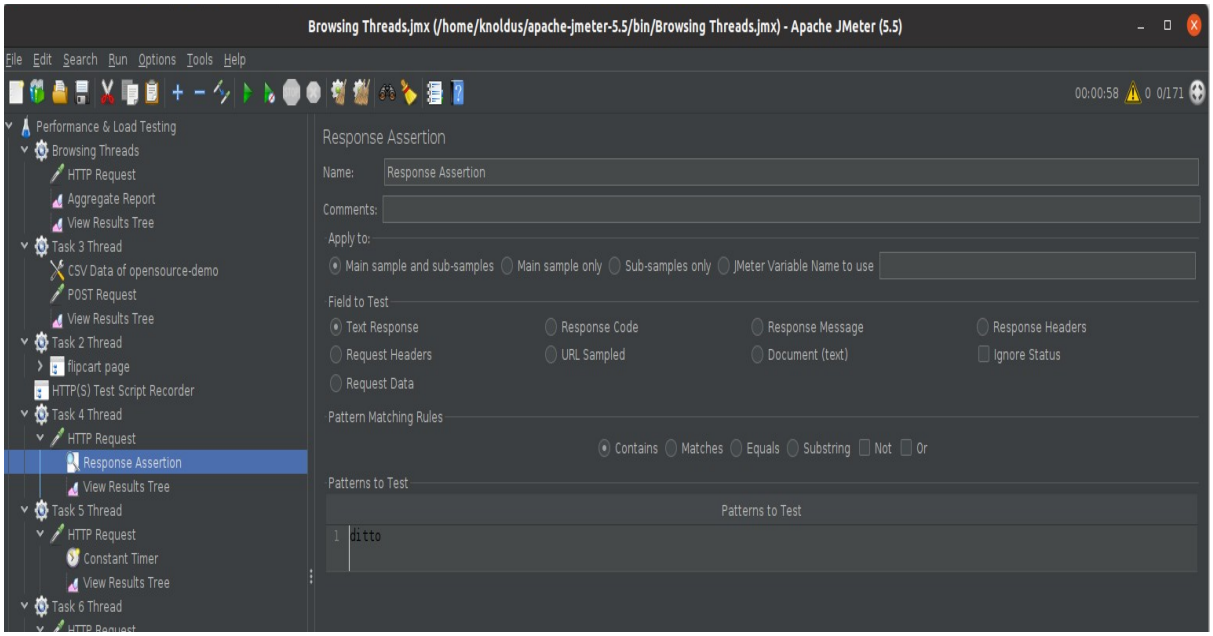
- Examples include Response Code, Response Message, Response Data (Body), Response Headers, etc.

  Pattern Matching Rules:

  These rules define how the pattern specified in the "Patterns to Test" field will be matched against the response.
- Equals: The entire response must exactly match the specified pattern.
- Contains: The response must contain the specified pattern.
- Matches: The response must match the regular expression pattern specified.
- Substring: The response must contain the specified substring.

  Patterns to Test:

  This field contains the pattern or value against which the response will be tested. Depending on the selected pattern matching rule, you can enter a simple string, regular expression, or substring to test against the response.



Real-World Example:

Let us assume we are testing an e-commerce website. We can use Assertions to ensure that when a user adds a product to their cart, the response from the server confirms that the product was successfully added if not then assertion fails in our script. This helps ensure that the website functions correctly and provides a seamless shopping experience for customers.

TASK-5:

- **TIMER:**

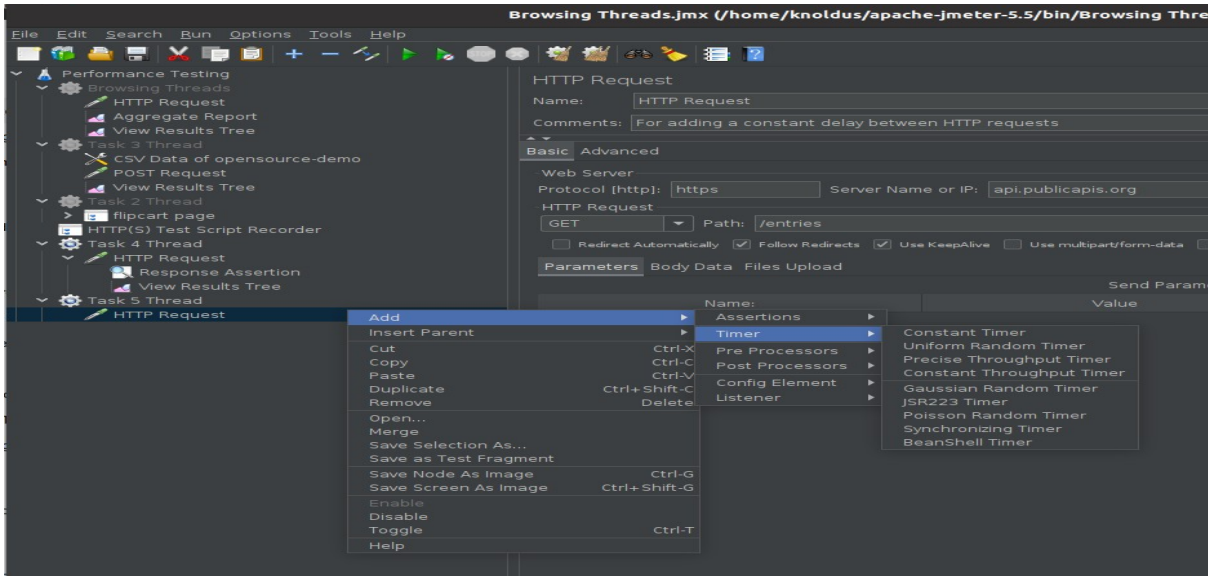
A Timer in JMeter is like a clock that helps control the timing of requests in your test plan. It adds delays between requests to simulate real user behavior and prevent overwhelming the server with too many requests at once.

What is it?

A Timer in JMeter is like a clock that helps control the timing of requests in our test plan.

Where is it?

We can find the Timer element in the JMeter GUI. To add it to our test plan, right-click on a Thread Group, go to "Add", then "Timer", and choose the type of Timer we want to use.

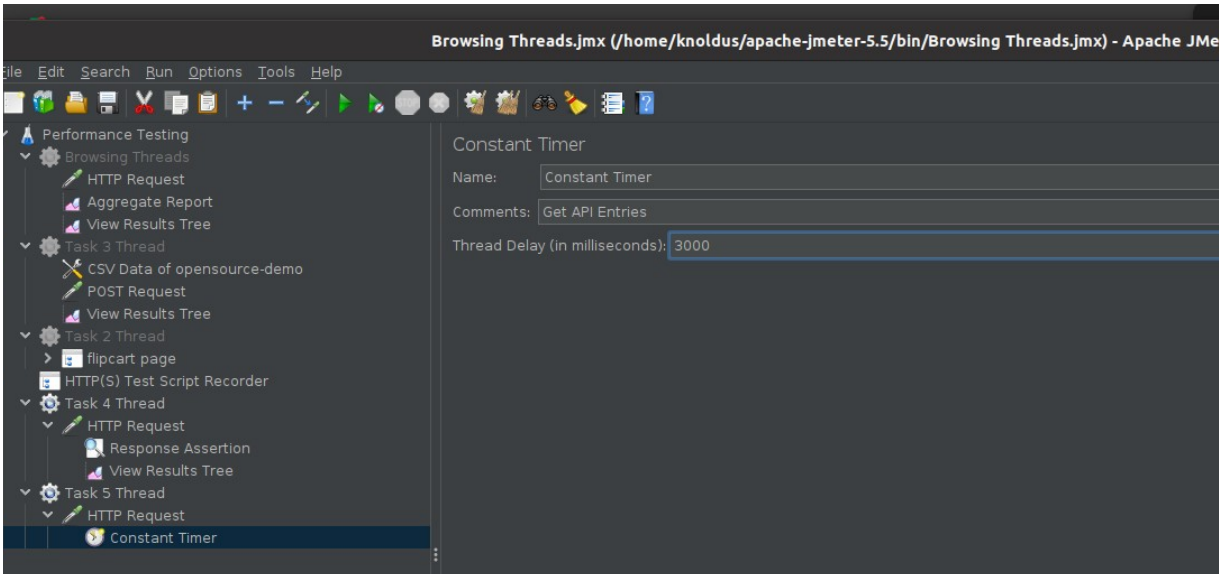


How to Use It:

Once added to our test plan, we can configure the Timer to introduce delays between requests. This helps simulate real user behavior and prevents flooding the server with too many requests at once.

Some commonly used Timers:

- Constant Timer: Adds a constant delay between requests.
- Gaussian Random Timer: Adds a delay with a Gaussian distribution.
- Uniform Random Timer: Adds a delay with a uniform distribution.



Real-World Example:

In e-commerce websiteif we want to simulate how real users behave when browsing products. We can use a Timer to add delays between actions like clicking on different products or adding items to the cart. This way, our test closely resembles how actual users interact with the website.

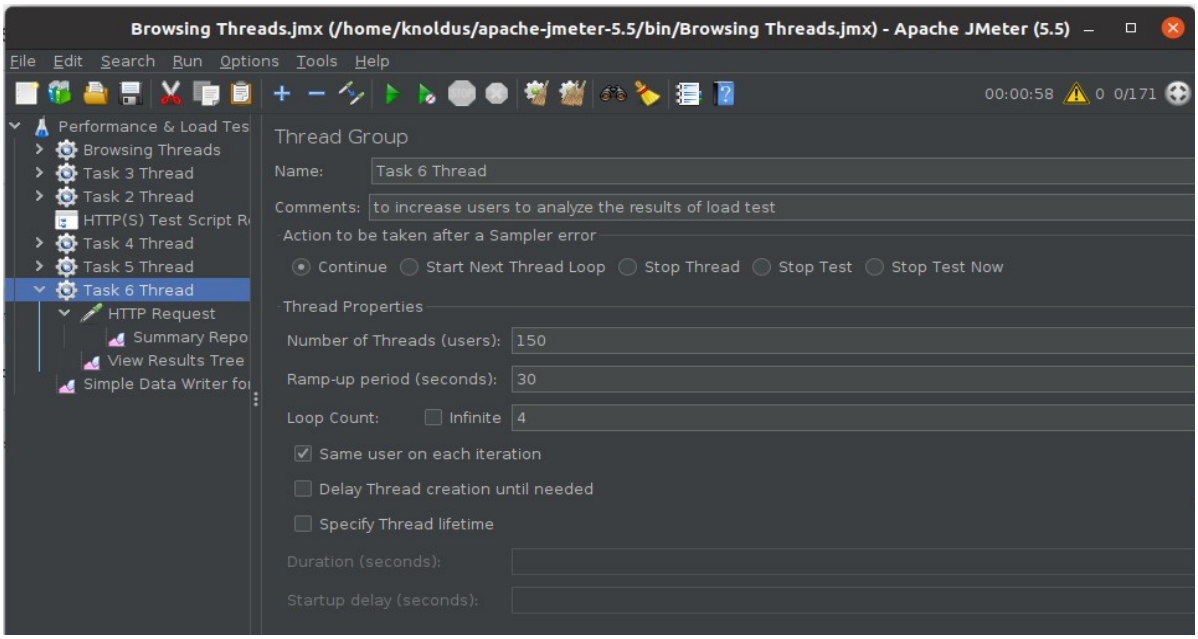
TASK-7:

Increasing Load

To design a load test plan I are using the website- "https://api.publicapis.org/entries" with increasing users over time by following these steps:

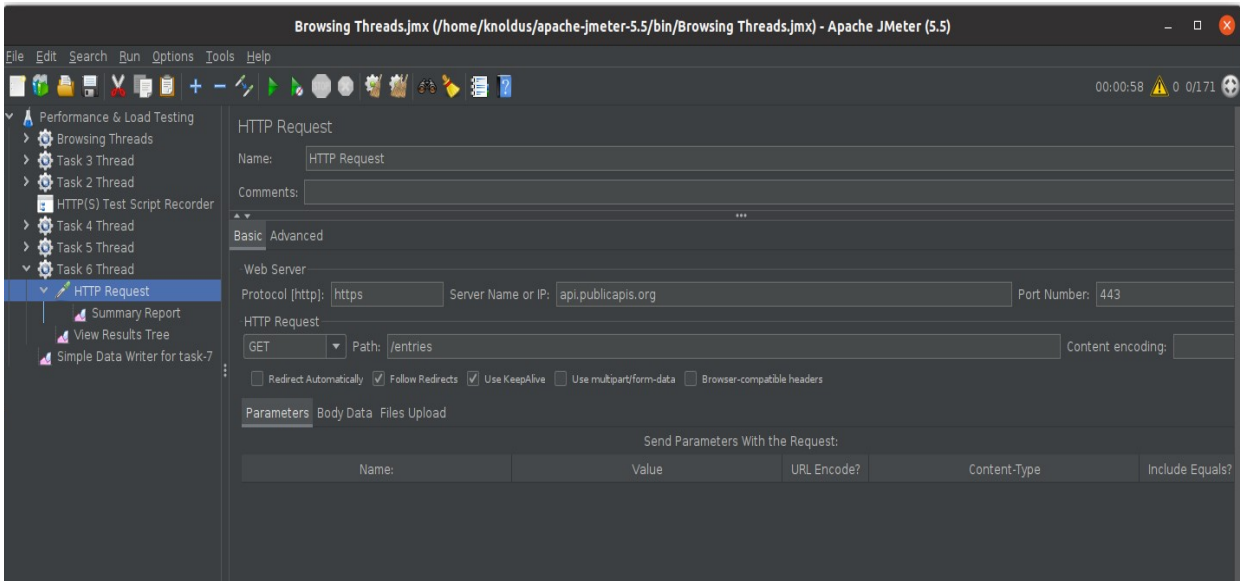
Right-click on "Test Plan" in the left pane and select "Add" -> "Threads (Users)" -> "Thread Group".

- Setting the number of threads (users) to start with (here, 150 - load).
- Setting the "Loop Count" to 4 (number of iterations).
- Setting the "Ramp-Up Period" to gradually increase the number of users over time (here, 30 seconds).

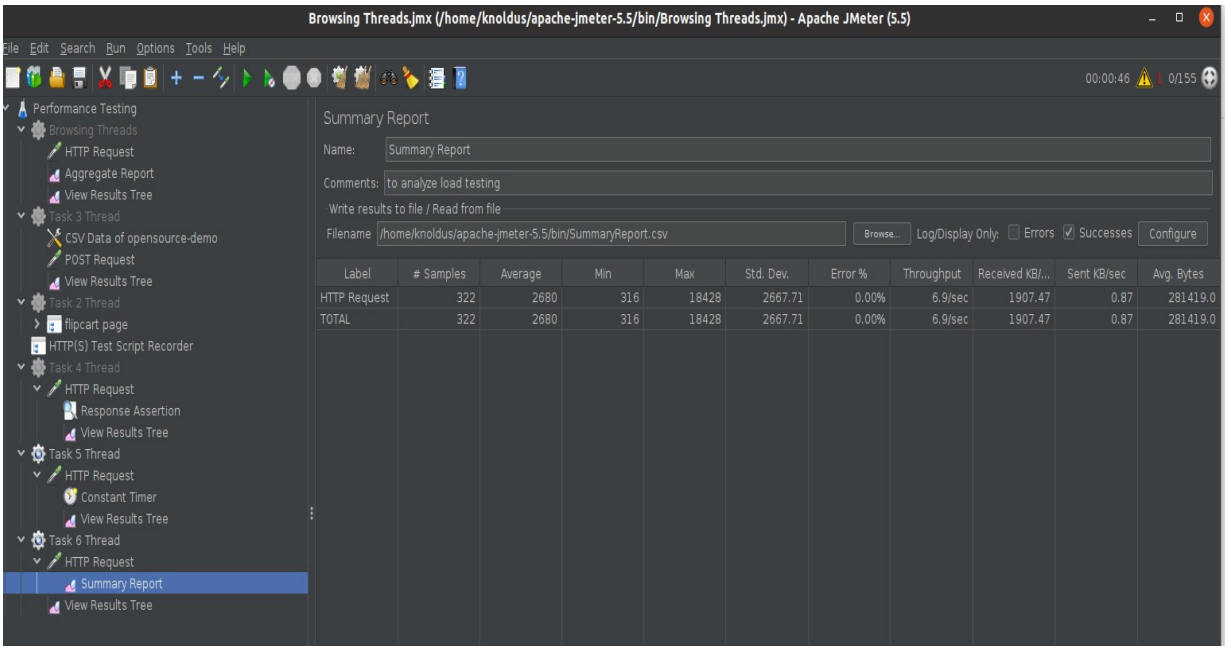


Then adding HTTP Request by right clicking on the Thread Group -> "Add" -> "Sampler" -> "HTTP Request".

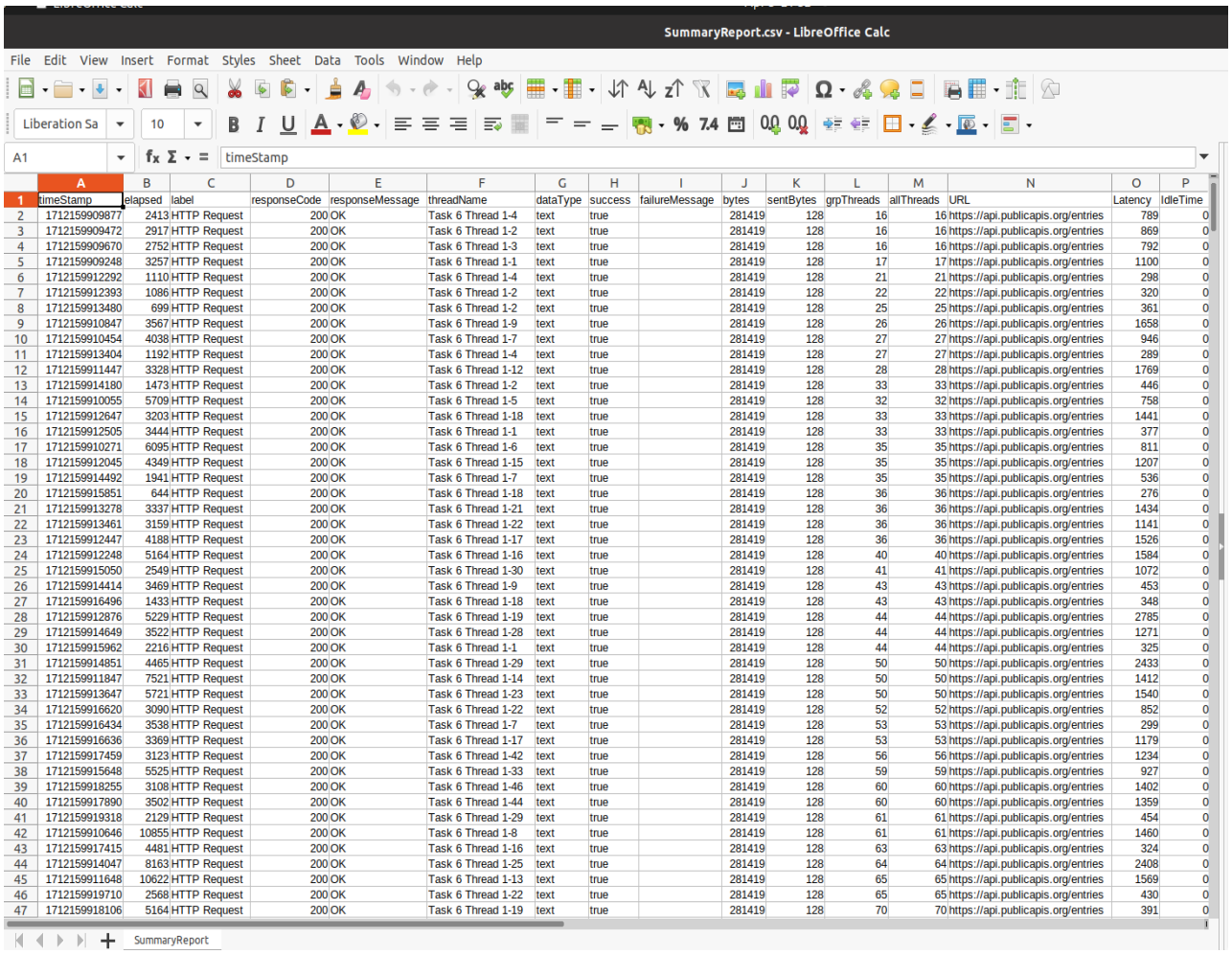
- Setting the following details:
- Protocol: HTTPS
- Server Name or IP: api.publicapis.org
- Path: /entries



Then adding the Listener by right clicking on the HTTP Request sampler -> "Add" -> "Listener" -> "Summary Report".



Runing the Test and reviewing the report:



	timeStamp	elapsed	label	responseCode	responseMessage	threadName	dataType	success	failureMessage	bytes	sentBytes	grpThreads	allThreads	URL	Latency	IdleTime
2	1712159909877	2413	HTTP Request	200	OK	Task 6 Thread 1-4	text	true		281419	128	16	16	https://api.publicapis.org/entries	789	0
3	1712159909472	2917	HTTP Request	200	OK	Task 6 Thread 1-2	text	true		281419	128	16	16	https://api.publicapis.org/entries	869	0
4	1712159909670	2752	HTTP Request	200	OK	Task 6 Thread 1-3	text	true		281419	128	16	16	https://api.publicapis.org/entries	792	0
5	1712159909248	3257	HTTP Request	200	OK	Task 6 Thread 1-1	text	true		281419	128	17	17	https://api.publicapis.org/entries	1100	0
6	1712159912292	1110	HTTP Request	200	OK	Task 6 Thread 1-4	text	true		281419	128	21	21	https://api.publicapis.org/entries	298	0
7	1712159912393	1086	HTTP Request	200	OK	Task 6 Thread 1-2	text	true		281419	128	22	22	https://api.publicapis.org/entries	320	0
8	1712159913480	699	HTTP Request	200	OK	Task 6 Thread 1-2	text	true		281419	128	25	25	https://api.publicapis.org/entries	361	0
9	1712159910847	3567	HTTP Request	200	OK	Task 6 Thread 1-9	text	true		281419	128	26	26	https://api.publicapis.org/entries	1658	0
10	1712159910454	4038	HTTP Request	200	OK	Task 6 Thread 1-7	text	true		281419	128	27	27	https://api.publicapis.org/entries	946	0
11	1712159913404	1192	HTTP Request	200	OK	Task 6 Thread 1-4	text	true		281419	128	27	27	https://api.publicapis.org/entries	289	0
12	1712159911447	3328	HTTP Request	200	OK	Task 6 Thread 1-12	text	true		281419	128	28	28	https://api.publicapis.org/entries	1769	0
13	1712159914180	1473	HTTP Request	200	OK	Task 6 Thread 1-2	text	true		281419	128	33	33	https://api.publicapis.org/entries	446	0
14	1712159910055	5709	HTTP Request	200	OK	Task 6 Thread 1-5	text	true		281419	128	32	32	https://api.publicapis.org/entries	758	0
15	1712159912647	3203	HTTP Request	200	OK	Task 6 Thread 1-18	text	true		281419	128	33	33	https://api.publicapis.org/entries	1441	0
16	1712159912505	3444	HTTP Request	200	OK	Task 6 Thread 1-1	text	true		281419	128	33	33	https://api.publicapis.org/entries	377	0
17	1712159910271	6095	HTTP Request	200	OK	Task 6 Thread 1-6	text	true		281419	128	35	35	https://api.publicapis.org/entries	811	0
18	1712159912045	4349	HTTP Request	200	OK	Task 6 Thread 1-15	text	true		281419	128	35	35	https://api.publicapis.org/entries	1207	0
19	1712159914492	1941	HTTP Request	200	OK	Task 6 Thread 1-7	text	true		281419	128	35	35	https://api.publicapis.org/entries	536	0
20	1712159915851	644	HTTP Request	200	OK	Task 6 Thread 1-18	text	true		281419	128	36	36	https://api.publicapis.org/entries	276	0
21	1712159913278	3337	HTTP Request	200	OK	Task 6 Thread 1-21	text	true		281419	128	36	36	https://api.publicapis.org/entries	1434	0
22	1712159913461	3159	HTTP Request	200	OK	Task 6 Thread 1-22	text	true		281419	128	36	36	https://api.publicapis.org/entries	1141	0
23	1712159912447	4188	HTTP Request	200	OK	Task 6 Thread 1-17	text	true		281419	128	36	36	https://api.publicapis.org/entries	1526	0
24	1712159912248	5164	HTTP Request	200	OK	Task 6 Thread 1-16	text	true		281419	128	40	40	https://api.publicapis.org/entries	1584	0
25	1712159915050	2549	HTTP Request	200	OK	Task 6 Thread 1-30	text	true		281419	128	41	41	https://api.publicapis.org/entries	1072	0
26	1712159914414	3469	HTTP Request	200	OK	Task 6 Thread 1-9	text	true		281419	128	43	43	https://api.publicapis.org/entries	453	0
27	1712159916496	1433	HTTP Request	200	OK	Task 6 Thread 1-18	text	true		281419	128	43	43	https://api.publicapis.org/entries	348	0
28	1712159912876	5229	HTTP Request	200	OK	Task 6 Thread 1-19	text	true		281419	128	44	44	https://api.publicapis.org/entries	2785	0
29	1712159914649	3522	HTTP Request	200	OK	Task 6 Thread 1-28	text	true		281419	128	44	44	https://api.publicapis.org/entries	1271	0
30	1712159915962	2216	HTTP Request	200	OK	Task 6 Thread 1-1	text	true		281419	128	44	44	https://api.publicapis.org/entries	325	0
31	1712159914851	4465	HTTP Request	200	OK	Task 6 Thread 1-29	text	true		281419	128	50	50	https://api.publicapis.org/entries	2433	0
32	1712159911847	7521	HTTP Request	200	OK	Task 6 Thread 1-14	text	true		281419	128	50	50	https://api.publicapis.org/entries	1412	0
33	1712159913647	5721	HTTP Request	200	OK	Task 6 Thread 1-23	text	true		281419	128	50	50	https://api.publicapis.org/entries	1540	0
34	1712159916620	3090	HTTP Request	200	OK	Task 6 Thread 1-22	text	true		281419	128	52	52	https://api.publicapis.org/entries	852	0
35	1712159916434	3538	HTTP Request	200	OK	Task 6 Thread 1-7	text	true		281419	128	53	53	https://api.publicapis.org/entries	299	0
36	1712159916636	3369	HTTP Request	200	OK	Task 6 Thread 1-17	text	true		281419	128	53	53	https://api.publicapis.org/entries	1179	0
37	1712159917459	3123	HTTP Request	200	OK	Task 6 Thread 1-42	text	true		281419	128	56	56	https://api.publicapis.org/entries	1234	0
38	1712159915648	5525	HTTP Request	200	OK	Task 6 Thread 1-33	text	true		281419	128	59	59	https://api.publicapis.org/entries	927	0
39	1712159918255	3108	HTTP Request	200	OK	Task 6 Thread 1-46	text	true		281419	128	60	60	https://api.publicapis.org/entries	1402	0
40	1712159917890	3502	HTTP Request	200	OK	Task 6 Thread 1-44	text	true		281419	128	60	60	https://api.publicapis.org/entries	1359	0
41	1712159919318	2129	HTTP Request	200	OK	Task 6 Thread 1-29	text	true		281419	128	61	61	https://api.publicapis.org/entries	454	0
42	1712159910646	10855	HTTP Request	200	OK	Task 6 Thread 1-8	text	true		281419	128	61	61	https://api.publicapis.org/entries	1460	0
43	1712159917415	4481	HTTP Request	200	OK	Task 6 Thread 1-16	text	true		281419	128	63	63	https://api.publicapis.org/entries	324	0
44	1712159914047	8163	HTTP Request	200	OK	Task 6 Thread 1-25	text	true		281419	128	64	64	https://api.publicapis.org/entries	2408	0
45	1712159911648	10622	HTTP Request	200	OK	Task 6 Thread 1-13	text	true		281419	128	65	65	https://api.publicapis.org/entries	1569	0
46	1712159919710	2568	HTTP Request	200	OK	Task 6 Thread 1-22	text	true		281419	128	65	65	https://api.publicapis.org/entries	430	0
47	1712159918106	5164	HTTP Request	200	OK	Task 6 Thread 1-19	text	true		281419	128	70	70	https://api.publicapis.org/entries	391	0

➔ Created and used different Listeners (e.g., View Results Tree, Aggregate Report) to analyze and interpret the data collected during the test in above Test Plans. My overall project structure is-