IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

API TESTING

[1]Adarsh Dubey, [2]Aayush kanthe, [3]Nidhi Brahmgotia
[1]Student,[2]Student,[3]Student
[1]Department of Electronic and Computer Science,
[1]Shree L.R Tiwari College of Engineering, Thane, India.

Abstract: — This paper deals with different methods to test various API using different techniques. This technique empowers testers, developers, and quality assurance teams to automate testing procedures, schedule test runs, and perform data-driven testing, thereby enhancing software quality and reducing testing time and costs. It includes essential security testing features to uncover vulnerabilities and ensures that APIs are not only functional but also secure. Rigorous testing is essential to ensure it is reliable, efficient, and functional. Automated API testing has become an effective way to improve and implement APIs. Moreover, the framework integrates essential security testing features to uncover vulnerabilities and ensure that APIs adhere to robust security standards. Detailed reporting capabilities generate comprehensive logs, metrics, and analytics, aiding in issue identification and performance tracking over time. The paper emphasizes the importance of API testing in the software development lifecycle and highlights the efficiency of automated testing in mitigating bugs and discrepancies.

***Keyword* - API (Application Programming Interface)**

I.INTODUCTION

In today's complex software landscape, APIs play a crucial role in facilitating communication and data exchange between different software components. However, ensuring the reliability and functionality of APIs presents significant challenges, given the intricacies of modern software architectures. Manual testing of APIs can be time-consuming and error-prone, underscoring the need for a systematic approach to API testing.
API testing involves validating various aspects of API functionality, including endpoint behaviour, request-response handling, error management, and security vulnerabilities. Flaws in APIs can have far-reaching consequences, leading to system failures, security breaches, and application inefficiencies. Therefore, comprehensive API testing is essential to identify and address potential issues before they impact the overall software ecosystem.
One of the key benefits of API testing is its ability to execute multiple test cases simultaneously, reducing testing time while maintaining accuracy. Automated testing frameworks like Apache JMeter enable testers to simulate diverse user scenarios and execute numerous test cases concurrently. This parallel execution capability accelerates the testing process and ensures comprehensive coverage of API functionalities.
Moreover, API testing allows testers to evaluate the scalability limits of websites or applications by assessing performance under different user load conditions. By gradually increasing the number of concurrent users or requests, testers can determine the maximum threshold of users that the system can handle without compromising performance. Testers can define specific criteria, such as expected response times or valid data formats, to generate reports that highlight areas of improvement and potential vulnerabilities. For instance, if an API request fails due to passing irrelevant or incorrect data, the testing framework can flag this as an assertion failure, indicating a discrepancy between expected and actual outcomes.
This paper aims to provide a comprehensive guide to API testing with Apache JMeter, covering various aspects of API testing methodologies, best practices, and advanced techniques using JMeter.

II. API TESTING TOOL

Apache JMeter Apache JMeter is an open-source software that allows load testing, functional testing and measuring performance. Although it was originally designed to test Web Applications, it has since expanded and now offers a wider range of functionalities. All tests in Apache JMeter must have a test plan, a thread group and at least one sampler. A test plan is the basis of any test and it describes a sequence of steps that JMeter will execute once it is run. . This tool offers various assertions among which the most popular are Response and JSON assertions. The HTTP methods that Apache JMeter supports are: GET, POST, HEAD, PUT, OPTIONS, TRACE, DELETE, PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK, REPORT, MKCALENDAR and SEARCH. For load testing, there is a CLI Mode which allows loading tests from any Java compatible OS. As mentioned earlier, it supports multithreading, and it offers CI support as well. The thread group offers control over the number of threads/users that will be used when executing the tests, the ramp–up period (in what time frame should all threads be executed) and the number of times the test should be executed. Samplers simply send requests to a server and wait for a response.

III. METHODOLOGY

In today's fast-paced world, end-users gravitate towards applications that offer seamless interaction, efficiency, and swift responsiveness. A sluggish software application, burdened with excessive data consumption and prolonged loading times, often fails to captivate users' attention and engagement. To address these challenges and optimize the performance of websites, various testing methodologies and tools are employed, among which Apache JMeter stands out as a versatile solution. Leveraging JMeter, developers and testers can assess the responsiveness of web pages and APIs under diverse conditions, such as varying request frequencies and response times. API testing, a core feature of JMeter, allows for direct evaluation of request-response pairs, enabling thorough analysis of factors like response time, latency, and load times.

The workflow begins with inputting website URLs or specific web page endpoints into JMeter, initiating the process of performance evaluation. Through API testing, JMeter captures performance statistics for individual requests, facilitating the validation of response times and load capacities through assertions. JMeter further enriches the analysis with its graphical output capabilities, presenting performance data in user-friendly formats like tables and graphs. The generated performance statistics serve as crucial inputs for performance metrics analysis, where methodologies are applied to gauge the software application's reliability and effectiveness. By consolidating multiple API requests into sessions, testers can assess various performance metrics such as software availability, mean time to failure (MTTF), mean time between failures (MTBF), and mean time to repair (MTTR). These metrics serve as key indicators of software quality and user experience, guiding efforts to enhance application performance and responsiveness.

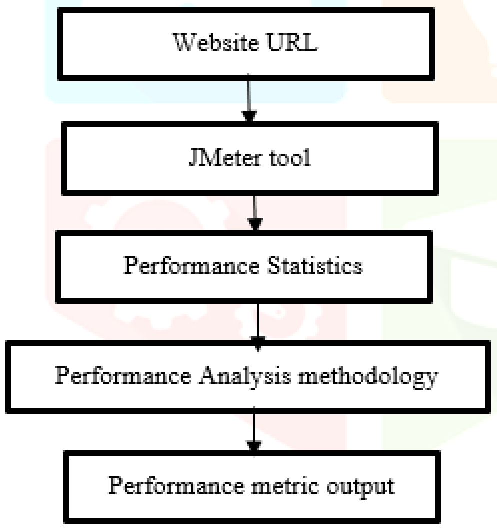


Figure 1: Performance Analysis Process

IV. PROPOSED WORK

Adding Thread Group: Initially, create new Thread Groups in Apache JMeter to simulate concurrent users accessing the API. Specify the number of threads (users) and ramp-up period to simulate realistic user behavior.

Adding HTTP Request Defaults: Now, create an HTTP request under this Thread Group and give a suitable name to this request . Add the protocol for this web connection (like "https" in this scenario). Set server name or domain name of the webpage (like "www.weatherview.com"). We can also give the relative path for the web page.

Adding HTTP Sampler: Configure HTTP samplers to send various types of HTTP requests (e.g., GET, POST, PUT, DELETE) to the API endpoints. In these we are using excel sheets to provide data inputs to the websites by parameterizing the file name/path using CSV(Comma-separated values) Data Set Config.

Adding listener and Visualizers: Add any listener to listen the process of JMeter and show output to the user. Here for this example let us add "View Results Tree" listener to capture the output of the web request. Graph Result, Summary Report, etc. are some of the listeners.

Running Test: Execute the test plan in Apache JMeter to simulate user interactions and API calls. Monitor system performance metrics such as response time, latency, throughput, and error rate during test execution.

Analyzing Results: Analyses test results to identify performance bottlenecks, scalability issues, and errors in API responses. Using built-in listeners and visualizers in Apache JMeter we can view and interpret test results graphically. These also helps us to calculate MTTR (Mean Time to Response), MTTF (Mean Time to Failure).

Iteration and Refinement: Refine the test plan iteratively based on insights gained from test execution. These helps us to provide multiple test case to analyses the behavior of website performance.

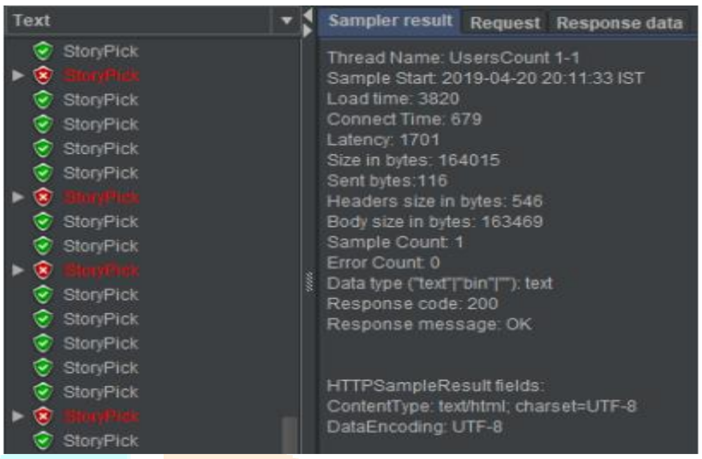
Scripting Dynamic User Scenarios: Utilize scripting languages such as Beanshell or JavaScript to simulate dynamic user scenarios. Script user authentication, session management, and data correlation to mimic real-world API interactions.

Generate Reports: Configure Apache JMeter to generate comprehensive test reports based on predefined conditions. Customize report formats, metrics, and thresholds to meet specific reporting requirements. For example., In weatherview.com website we can provide condition like generated test output for cities where temperatures is above 35°C.The report for these condition will be generated.

Assertions Elements: Assertions in Apache JMeter validate whether the returned values match the expected outcomes. When an assertion fails, it indicates a discrepancy and marks the corresponding sampler as failed in the test results displayed in listeners. For instance, providing unrelated data, like a non-existent city name, will trigger an assertion failure. This ensures that only valid and relevant data is used for testing, enhancing the accuracy and reliability of the test results.

Timers: Timers in Apache JMeter regulate the timing between requests, simulating user behavior during performance testing. By adding timers, delays can be introduced between requests to mimic realistic user interactions. For example, after querying weather data for a city, a timer can simulate the time a user takes to read the information before making another request. Timers help replicate real-world scenarios, ensuring more accurate performance testing results.

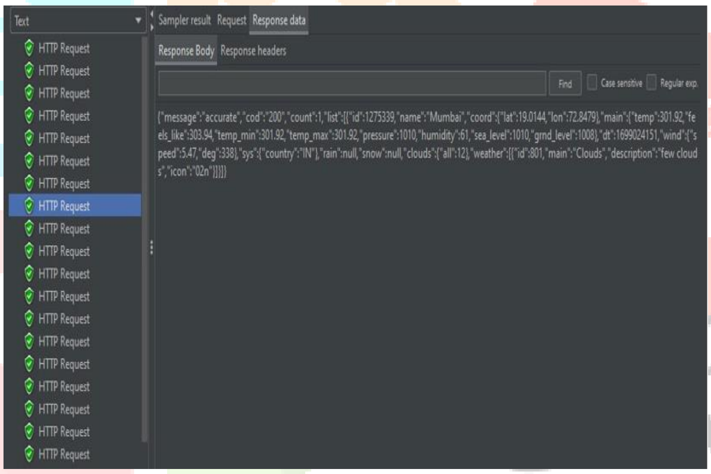
Performance Results: We can evaluate web application response time (captured by JMeter) as following:



```
Thread Name: UsersCount 1-1
Sample Start: 2019-04-20 20:11:33 IST
Load time: 3820
Connect Time: 679
Latency: 1701
Size in bytes: 164015
Sent bytes:116
Headers size in bytes: 546
Body size in bytes: 163469
Sample Count: 1
Error Count: 0
Data type ("text"|"bin"|""): text
Response code: 200
Response message: OK


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

Figure 2: Sample Response time



```
{"message":"accurate","cod":"200","count":1,"list":[{"id":1275339,"name":"Mumbai","coord":{"lat":19.0144,"lon":72.8479},"main":{"temp":301.92,"fe
els_like":303.94,"temp_min":301.92,"temp_max":301.92,"pressure":1010,"humidity":61,"sea_level":1010,"grnd_level":1008},"dt":1699024151,"wind":{"s
peed":5.47,"deg":338},"sys":{"country":"IN"},"rain":null,"snow":null,"clouds":{"all":12},"weather":[{"id":801,"main":"Clouds","description":"few cloud
s","icon":"02n"}]}]}
```

Figure 3: Sample HTTP request data

V. RESULT DISCUSSION

Manual Response Time:

Action: Navigate to the weather website and search for weather information for New York City.
Manual Response Time: 6.5 seconds (time taken from initiating the search until weather information is displayed).

JMeter Response Time:

Thread Group: Configured with 10 users and 100 loops. (means each user will execute 100 test cases like Mumbai, New York, Moscow, London…).
HTTP Request: Simulated search action for weather information for New York City.
JMeter Results: Average Response Time: 4.2 seconds (recorded by JMeter across all iterations).
These showcase that JMeter allows you to generate results for multiple inputs simultaneously (here we implement 10 users * 100 loops = 1000 requests) whereas manual testing typically involves testing one input at a time.
We can also evaluate various types of software testing to ensure more system performance concepts of these analysed performance metrics using JMeter.

JMeter Response data: Figure 3, which displays the HTTP response output as an example. The current HTTP request demonstrates the comprehensive information provided by the website. For instance, when querying weather data for Mumbai, the response includes details such as temperature, minimum temperature, maximum temperature, humidity, sea level, wind speed, and other relevant parameters. This illustrates the extensive range of data accessible through the API endpoint, showcasing the depth of information available for analysis and testing purposes.

We can also evaluate various types of software testing to ensure more system performance concepts of these analysed performance metrics using JMeter.

Scalability testing: The objective of scalability testing is to determine the software application's effectiveness in "scaling up" to support an increase in user load. It helps plan capacity in addition to your software system.

Load testing: It checks the application's ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.

Stress testing: It involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.

Volume testing: Under Volume Testing large no. of. Data is populated in a database and the overall software system's behaviors is monitored. The objective is to check software application's performance under varying databases.

Spike testing: It tests the software's reaction to sudden large spikes in the load generated by users.

Comparison with Earlier research: This research paper focused on improvement of two aspects which are following:

-More accurate
- Better report generation / representation
- Better performance management
- Increase user Traffic

We can differentiate earlier research with this research as per following table:

Table 1: Research work Comparison

Earlier Research	**Proposed Work**
Only one input can be provided	Multiple Input can be provided
Accuracy was around 85% while working with more than on thread.	JMeter accuracy is around 90% because response time also include certificate installation time to ensure real time behaviour.
Sites crash when multiple users log in at a same time.	Defines the traffic limit the website can handle.

VI. CONCLUSION

This research paper delves into the realm of API testing methodologies, emphasizing the significance of rigorous testing to ensure the reliability, security, and performance of software applications. Through the exploration of different techniques and tools, such as Apache JMeter, testers and developers can streamline the testing process, automate procedures, and enhance software quality while reducing testing time and costs.

By leveraging Apache JMeter, testers can conduct comprehensive API testing, validate response times, identify performance bottlenecks, and simulate real-world user interactions. The integration of essential security testing features in JMeter enhances the overall testing process, ensuring that APIs not only function as intended but also adhere to robust security standards. Moreover, detailed reporting capabilities provided by JMeter enable stakeholders to gain insights into performance metrics, track issues, and make informed decisions for software optimization and improvement.

In summary, this research contributes to the understanding of automated API testing methodologies and highlights the efficiency and effectiveness of tools like Apache JMeter in ensuring software reliability and functionality.

VII. REFERENCE

[1] Amid, Man Zhang, Andrea Arcuri, “Testing RESTful APIs”, Association for Computing Machinery, November 2023

[2] Adeel, Cagatay Catal, Deepti Mishra, Mohammad Ahmad, “RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions”, MDPI , Volume :33, Issue 1, April 2022

[3] Sherine George, Nagaraja G.S,” Automated Framework for API Testing”, IRJET, Volume: 07 Issue:05, May 2020

[4] Prerna Sanjay Wavhule, Rutuja Vasant Nagarkar, Bhushan Bharat Johare “Automated API Testing: Technologies, Challenges, and Future Directions”, IJARIIE, 2023

[5] Alberto Martin-Lopez, “AI-driven web API testing”, International Conference on Software Engineering(ICSE), June 2020

[6] Goshu, Y.Y. and Daniel Kitaw, D. (2017) „Performance measurement and its recent challenge: a literature review‟, International Journal of Business Performance Management, May 2016.

[7] David Maplesden, Ewan Tempero, John Hosking, John C. Grundy, “Performance Analysis for Object-Oriented Software: A Systematic Mapping”, Volume: 41, Issue: 7 , January 2015.

[8] Isha Arora, Vikram Bali, “A Brief Survey on Web Application Performance Testing Tools Literature Review”, International Journal of Latest Trends in Engineering and Technology, May 2015.

[9] Pooja Ahlawat and Sanjay Tyagi, “A Comparative Analysis of Load Testing Tools Using Optimal Response Rate”, Volume 3, Issue: 5, May 2013.

[10] Eljona Proko and Ilia Ninka, “Analyzing and Testing Web Application Performance”, Dept. of Computer Science, Vol.3, Issue: 10, October 2013.

[11] Muhammad Abid Jamil, Muhammad Arif, Normi Sham Awang Abubakar, Akhlaq Ahmad, “Software Testing Techniques: A Literature Review”, 6th International Conference on Information and Communication Technology for The Muslim World, 2016.

[12] Sukhdev Singh Ghuman, "Software Testing Techniques", International Journal of Computer Science and Mobile Computing, Vol. 3, Issue: 10 ,October 2014.

[13] Reenu Bhatia, Anita Ganpati, “Performance Evaluation of JMeter, LoadComplete and WAPT”, International Journal of Scientific & Engineering Research, Volume 7, Issue 12, December 2016.

[14] Shikha Dhiman, Pratibha Sharma, “Performance Testing: A Comparative Study and Analysis of Web Service Testing Tools”, IJCSMC, Vol. 5, Issue. 6, pg.507 – 512, June 2016.

[15] Avritzer and E.J. Weyuker, "Deriving Workloads for Performance Testing" Software-Practice and Experience, vol. 26, no. 6, pp. 613-633, June 1996.