EJSIT
European Journal of Science, Innovation and Technology
ISSN: 2786-4936
www.ejsit-journal.com
Volume 4 | Number 2 | 2024

Software Testing using Multi-Collaboration Platforms

Sana Rizwan[1], Mati ul Rehman[2], Ayesha Tahir [3]
[1]Assistant Professor, Department of Computer Science, COMSATS University Islamabad, Lahore Campus, Pakistan
[2,3]Student of BSSE, Department of Computer Science, COMSATS University Islamabad, Lahore Campus, Pakistan

ABSTRACT

Software applications become increasingly complex and competitive pressures intensify; the importance of quality assurance in software development cannot be overstated. Software testing plays a crucial role in the Software Development Lifecycle, necessitating the adoption of enhanced and efficient methodologies and techniques to ensure superior quality. This paper aims to explain and explore existing testing techniques to enhance quality assurance in software development processes by testing means. Various testing tools for backend and frontend applications are available to figure out the outcomes, but selection of the appropriate tool to find the concerned outcomes in software testing aspect is imperative. Loading, implementation, fetching results, extraction of the suitable fallouts is the major aim to the study.

Keywords: Unit Testing Approach, System Testing Approach, Integration Testing Features, Acceptance Testing Capabilities, Testing Lifecycle Procedures, Critical Production in Software Testing, Risk Assesment, UAT, HyperText Transfer Protocol, Integrated and Continous Deployment Pipelines

INTRODUCTION

Software testing serves several crucial purposes in the software development life cycle. Software testing is an integral part of the software development process that contributes to the delivery of reliable, high-quality, and user-friendly software while reducing risks and ensuring compliance with requirements (Felderer et al., 2014). Testing helps in uncovering bugs, errors, and defects in the software. Detecting and fixing these issues early in the development process can prevent more significant problems and reduce the cost of fixing defects later on. The primary goal of software testing is to ensure that the software meets specified requirements and functions as intended. Quality assurance through testing helps deliver a reliable and high-quality product to end-users (Quadri & Farooq, 2010). Thorough testing helps in identifying issues that may affect the user experience, such as poor performance, usability issues, or inconsistencies. Addressing these problems contributes to a positive user experience. It helps validate that the software performs the functions it was designed to do. Performance testing assesses the responsiveness, speed, and stability of the software under different conditions. This type of testing helps optimize the software's performance and scalability. Testing helps identify and mitigate risks associated with software development (Awotar & Sungkur, 2018). By thoroughly testing various aspects of the software, developers and stakeholders can make informed decisions and reduce the likelihood of critical issues occurring in production. Security testing identifies vulnerabilities and weaknesses in the software that could be exploited by attackers. It helps in making the software more secure and protects sensitive data. Before deploying updates or changes to software, testing is crucial to ensure that new features do not introduce new issues and that existing functionality remains intact. In certain industries, there are regulatory standards and compliance requirements that software must adhere to. Testing helps ensure that the software complies with these standards (Reid, 2012). High-

quality software that is free from critical issues contributes to customer satisfaction. Satisfied users are more likely to continue using the software and recommend it to others.

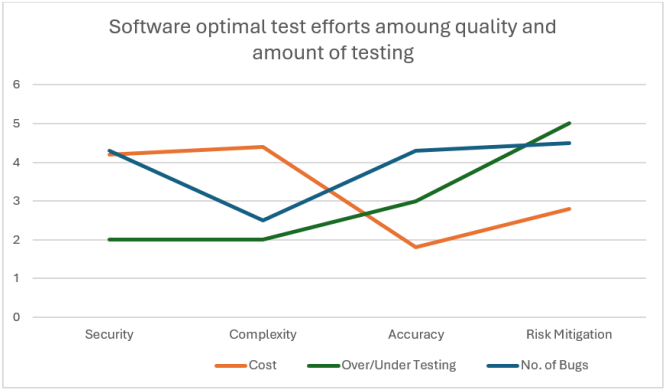


Figure 1: Software optimal test efforts amoung quality and amount of testing

Figure 1 illustrates the relationship between testing cost and the number of errors found during software testing. As depicted, the cost of testing increases significantly as both functional and non-functional testing activities are conducted. This relationship highlights the potential trade-offs involved in decision-making regarding the extent of testing to be performed. Opting to reduce testing efforts may result in overlooking numerous bugs, which could compromise software quality.

Figure 1 underscores the importance of software testing as a fundamental component of software quality assurance. Particularly in the context of life-critical software, where the stakes are high, rigorous testing is essential. Failure to adequately test such software can lead to severe consequences, including schedule delays, cost overruns, or even project cancellation. The goal of effective testing is to strike a balance by conducting an optimal amount of tests to minimize the need for excessive testing efforts. This ensures that critical bugs are identified and addressed without unnecessarily inflating testing costs. By employing a strategic approach to testing, organizations can enhance software quality while managing resources efficiently.

Existing Testing Methods

In the Software Development Life Cycle (SDLC), different types of testing are conducted at various stages to ensure the quality and reliability of the software product. Breakdown of testing types commonly associated with different SDLC phases. In Requirement Analysis and Review phase work involves analyzing and reviewing the software requirements to ensure they are clear, complete, and feasible. Testing activities in this phase may include requirements validation, traceability matrix creation, and user story validation. In design phase, architectural testing surely works and focuses on evaluating the software architecture and design to ensure it aligns with the specified requirements and is scalable, maintainable, and robust.

In development phase, unit testing integrated by default and developers perform unit testing to validate individual units or components of the software. It ensures that each unit functions correctly as per the design and requirements. Nexly, integration testing verifies the interaction between integrated components/modules to ensure they work together seamlessly as intended. Afterwards, component testing or module testing validates the functionality of individual software components or modules. Though not strictly a testing type, code reviews are conducted during the development phase to identify defects, improve code quality, and ensure adherence to coding standards.

In Testing Phase, system testing evaluates the entire software system's functionality, performance, and behavior in a simulated environment like the production environment. While acceptance testing (UAT), end-users validate the software's compliance with business requirements and determine its readiness for deployment. In regression testing, new changes or modifications to the software do not adversely affect existing functionalities. It involves retesting previously tested functionalities. In alpha/beta testing, software is tested by a selected group of users (alpha) or a broader audience (beta) to gather feedback and identify any remaining issues.

Working in the deployment phase, smoke testing verifies whether the critical functionalities of the software are working properly after deployment. In compatibility Testing software functions correctly across different platforms, browsers, devices, and environments.

SDLC last phase is maintenance phase, regression testing ensure that modifications or enhancements do not introduce new defects or impact existing functionalities. To conduct performance testing, monitor and optimize the software's performance over time. Each testing type plays a crucial role in ensuring the software meets quality standards and fulfills user requirements throughout the SDLC.

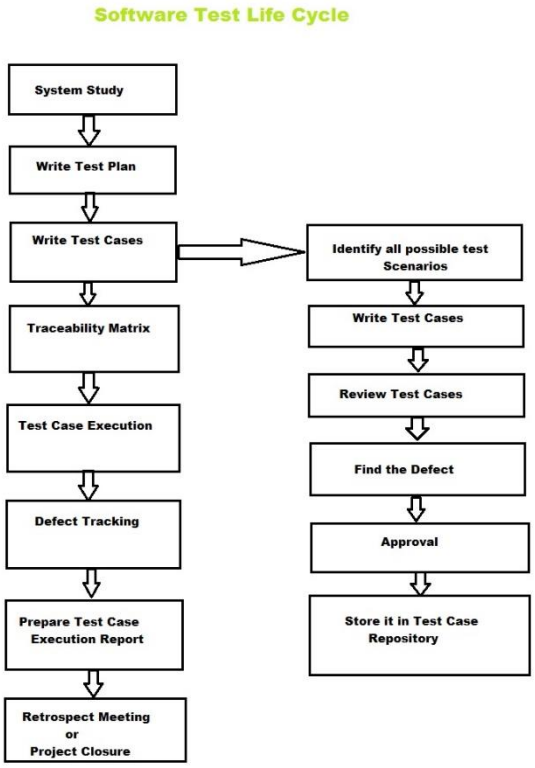


Figure 2: Software Testing Life Cycle

The Software Testing Life Cycle (STLC) has indeed evolved significantly to adapt to the changing landscape of software development from the system study to project closure by implementing test cases, review them, traceability aspects, defect tracking and execution report. Figure 2 shows an overview of the evolution of STLC, and some key tactics employed by organizations to maximize testing efficiency and effectiveness.

METHODOLOGY

Backend Testing using Postman

Postman is a popular collaboration platform for API development. It provides tools for designing, testing, and managing APIs (Application Programming Interfaces). Postman is widely used by developers, testers, and other stakeholders involved in the API development process.

Reasons to use Postman

Postman is a crucial tool for API development and testing, offering a user-friendly interface for designing, testing, and managing APIs. It streamlines the process of creating and executing API requests, automates testing with scripting capabilities, facilitates collaboration through shared collections and workspaces, and provides features like monitoring. With its versatility and ease of use, Postman accelerates the API development lifecycle, ensuring the reliability and efficiency of APIs while fostering collaboration among development teams. Postman is user-friendly interface that provides an intuitive and easy-to-use interface, making it accessible to both developers and non-developers. The interface allows users to quickly create, test, and manage API requests without a steep learning curve. Request and response handling can easily create different types of HTTP requests (GET, POST, PUT, DELETE, etc.) and customize headers, parameters, and authentication methods. Postman provides a clear view of API responses, making it easy to inspect and validate the data returned from API calls. Postman supports automated testing with scripting capabilities using JavaScript. Test scripts can be written to validate API responses, ensuring that the API behaves as expected and meets specified criteria. Collections allow users to organize group-related requests, making it easier to manage and execute multiple API calls. Collections facilitate the creation of test suites, representing different scenarios or stages of API development. Postman allows the use of environment variables, making it easy to manage and switch between different sets of parameters for testing in different environments (e.g., development, staging, production).

Steps to Perform Postman

- Create a new request by requiring the request method (e.g., GET, POST, PUT) and the target API endpoint.
- Write and execute test scripts using JavaScript to validate the API response.
- Collections can be used to represent different parts of an API, test scenarios, or various stages of development.
- Share the collections with team members or other stakeholders.
- Collaborators can view, edit, and execute requests within shared collections, fostering collaboration and communication.
- Postman also provides features for commenting, discussing issues, and documenting APIs.

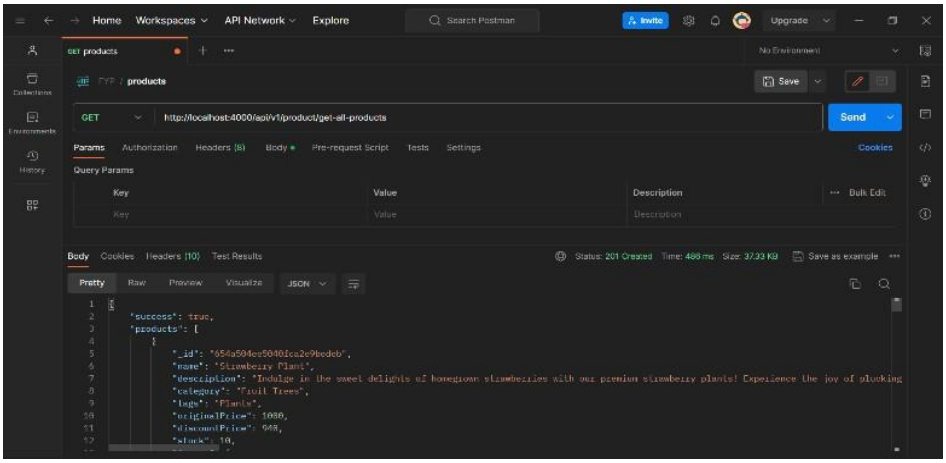
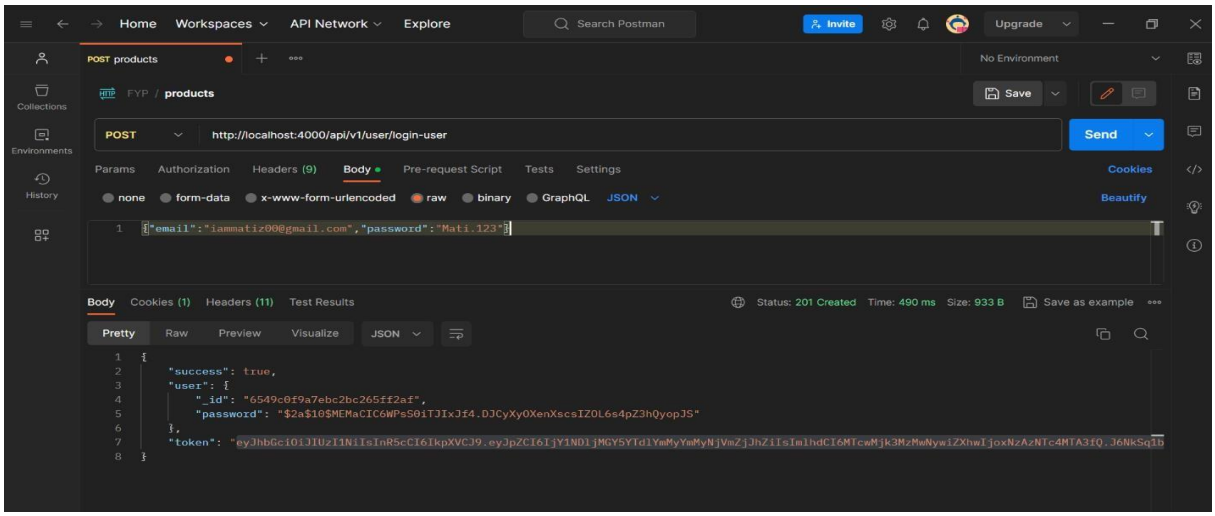


Figure 3: Get All Product Function

Testing is performed on "Get All Product" functionality, with various parameters and values, the loading procedures are defined in Figure 3. Similarly, login user functionality loaded in Figure 4(a) using POST method with username, id and password parameters, token value has been passed to it by postman to bind it. GET process will manipulate to bind the token with login function in the Query Params as shown in Figure 4(b).



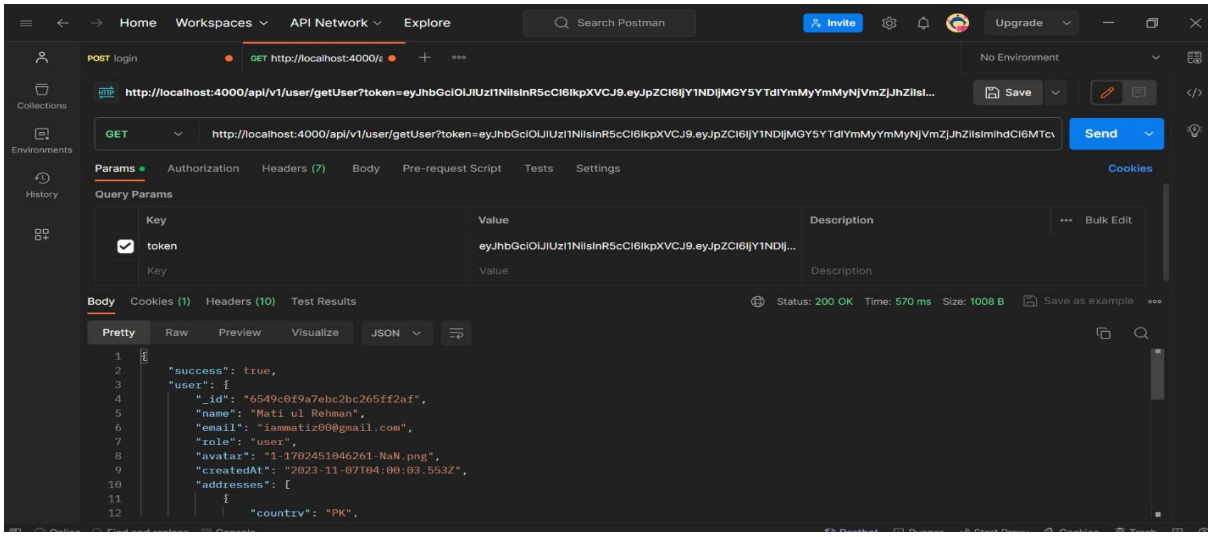


Figure 4 (a,b) : POST and GET method

Frontend Testing using JMeter and BlazeMeter

JMeter

Apache JMeter is an open-source software designed to measure and analyze the performance of web applications. It can be used to simulate a heavy load on a server, group of servers, network, or object to test its strength or to analyze overall performance under different load types. JMeter supports protocols such as HTTP, HTTPS, FTP, JDBC, LDAP, and more. User sends request for server connectivity; simulation actively work over the request process. Server response and JMeter will save the request to perform. For procedure checking and binding, JMeter collects data and manipulates statistical Information about user request. After satisfying, the request process will stop, and a report will be generated about software performance as shown in Figure 5.

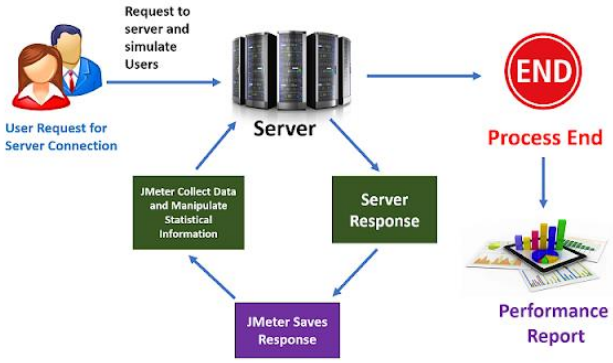


Figure 5: JMeter Internal Working

BlazeMeter

BlazeMeter is a commercial, cloud-based performance testing platform that is built on JMeter. It allows users to run JMeter tests in the cloud, enabling scalability and flexibility. Commands can send to on-premises load generator to handle and load requests on the application server or database server. Besides it, request of the user can divert to cloud load generator to handle the load for the usage of application server. BlazeMeter provides features like real-time reporting, analytics, and collaboration tools for distributed teams. It simplifies the process of running large-scale tests from different geographic locations shown in Figure 6.

BlazeMeter employs a centralized architecture with a Cloud-based BlazeMeter controller overseeing the testing process. The controller manages the test execution, and depending on the chosen configuration, virtual users are generated either by Cloud resources or On-Premises Load Generator machines. The Load Generator machines simulate realistic mobile device users by emulating various devices and mimicking diverse network conditions. Requests generated by these virtual users are directed towards the application server, which in turn communicates with the database server. BlazeMeter effectively gathers data from the application server and presents the results in graphical formats. Additionally, if a monitoring tool is in use, it collects performance counters and transmits them to BlazeMeter. The integration of monitoring data with test results allows for a comprehensive analysis, aiding in the identification of server states during performance degradation and facilitating the resolution of performance issues. BlazeMeter can handle a high volume of virtual users, scaling up to one million. The tool offers web-based reporting accessible from anywhere, providing a comprehensive overview of test results. It integrates with New Relic, allowing for server monitoring. A New Relic account or license is required for this feature. Test execution is carried out on Amazon cloud servers, providing flexibility and scalability. It supports plug-ins for various tools, including Google Analytics, Apache JMeter, and Drupal. The tool seamlessly integrates with Google Analytics,

automatically creating tests based on best practices and historical data. Users can run tests locally using the JMeter plug-in and export the execution data to BlazeMeter for analysis and reporting. Load Generator machines can be assigned from different geographical locations, allowing for realistic load testing. It supports integration with Continuous Integration tools such as TeamCity, Jenkins, and JetBrains, making it suitable for Agile methodologies. The tool provides data security by masking sensitive information, especially when using on-premises load generation. It utilizes JMeter as its scripting component, offering a powerful scripting engine for test customization.

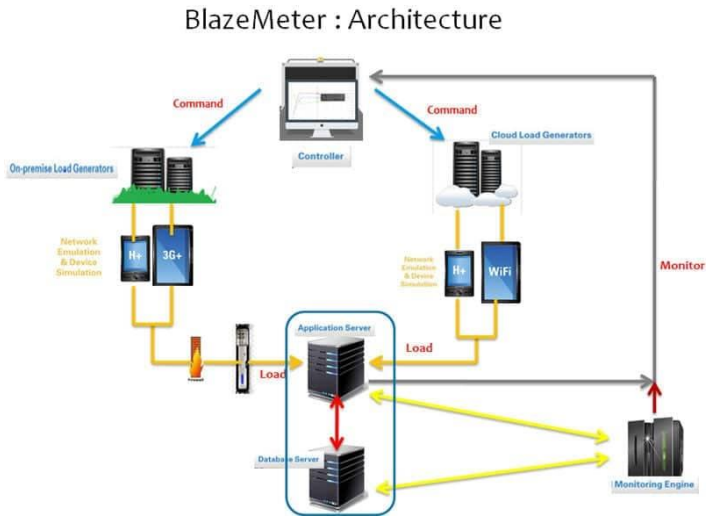


Figure 6: BlazeMeter Internal Working

RESULTS AND DISCUSSION

Basically, its use is to implement load and unit testing for software. Save the results of BlazeMeter in a file and then pass the result to JMeter and it will automatically detect all individual requests and show the results in detail as shown in Figure 7. JMeter will demonstrate discrete results of every request. BlazeMeter displays overall architectural result whereas JMeter shows result of every endpoint and its call in Figure 8.

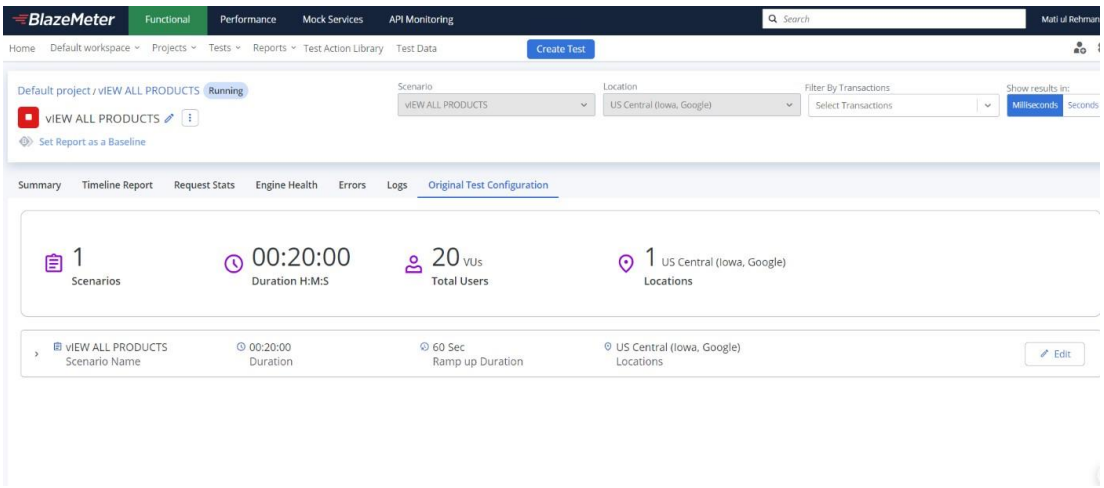


Figure 7: Loading Summary Results BlazeMeter

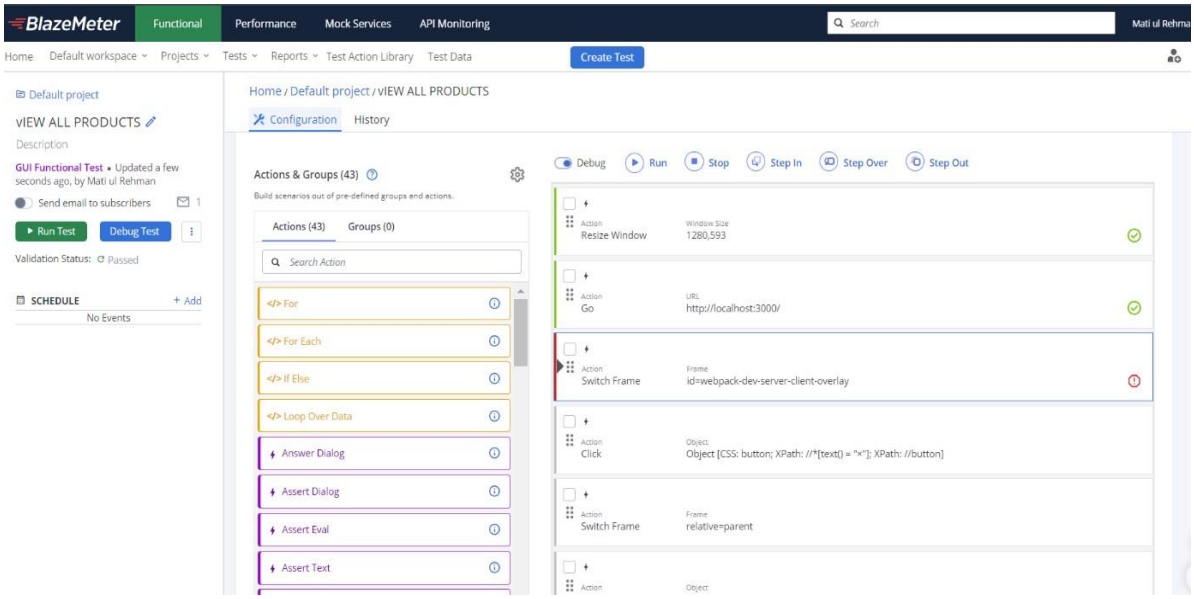


Figure 8: Individual Endpoint Results using BlazeMeter

These are different endpoints for API, by which request can be sent for data gathering, so that data can display according to demand on website and render the results accordingly. Case study software has 20 to 30 API endpoints and results are extracted from 6 of them due to limitations. These include endpoints for getting data of the product, endpoint for getting data for most famous events, endpoint for payment methods, endpoint for getting data of all the products, endpoint for getting data of all the users and all the shops in Figure 9, 10, 11.

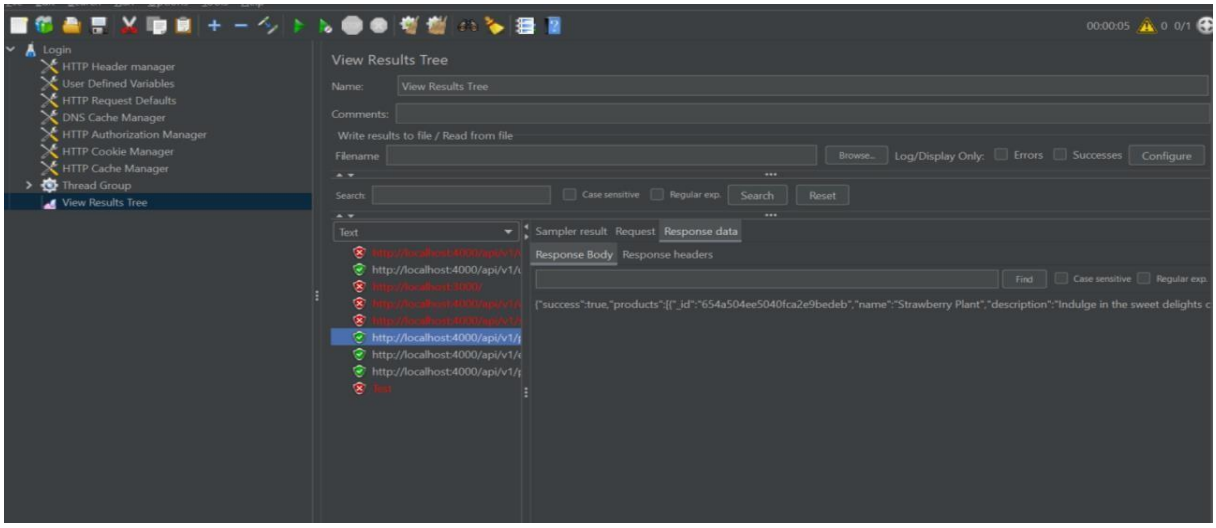


Figure 9: Product Result Tree

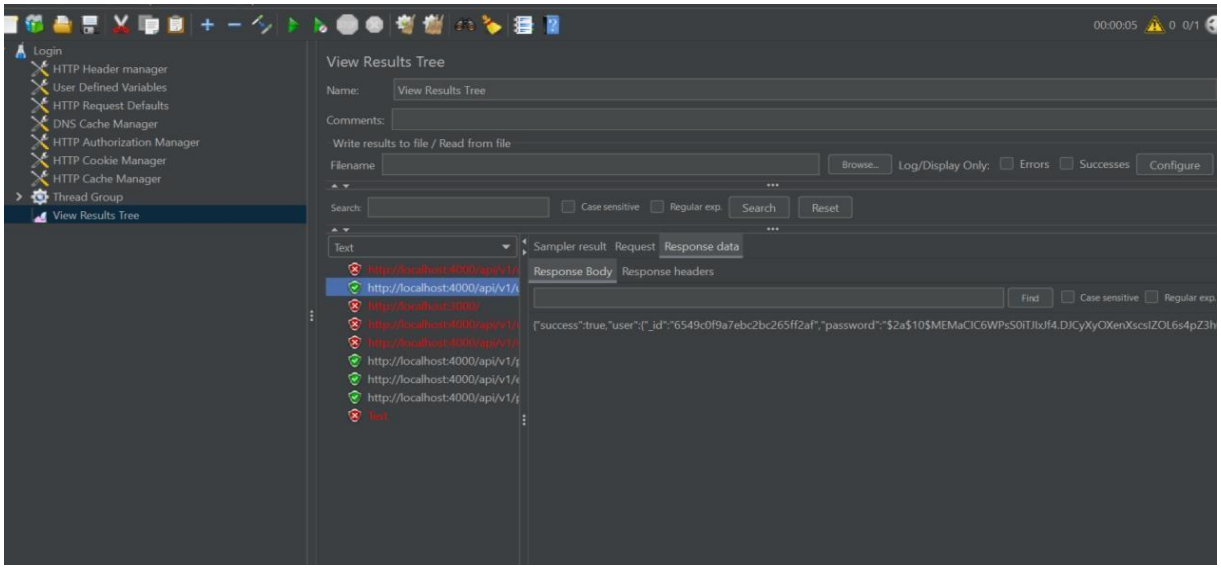


Figure 10: Users Result Tree

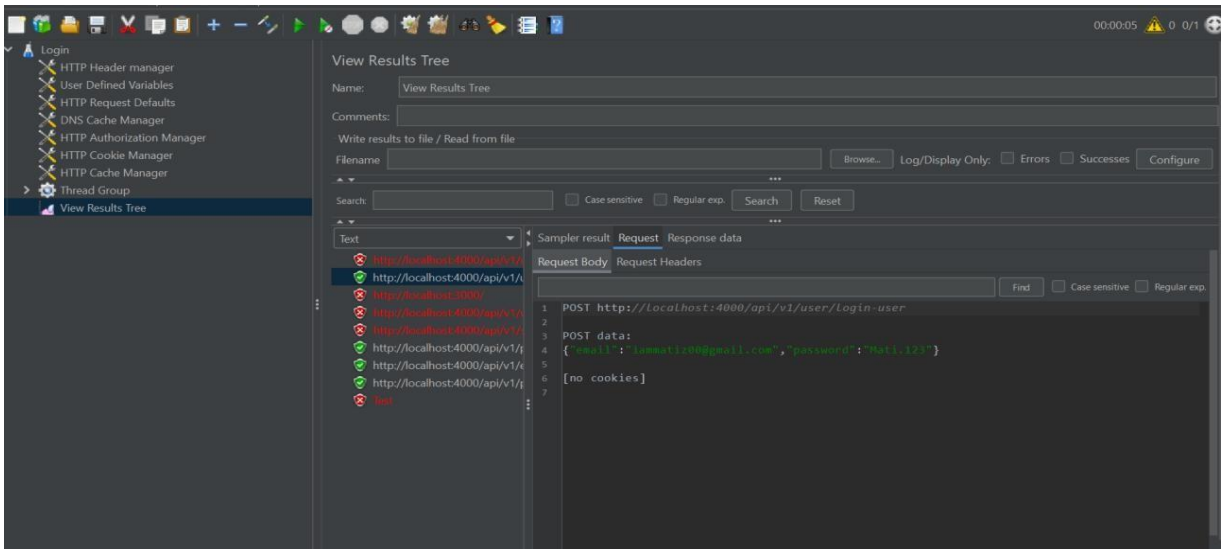


Figure 11: Login (Username and Password) Result Tree

In a single-threading environment like Node.js, a single thread is responsible for processing events and managing I/O operations. When a request is made, instead of waiting for it to be completed, Node.js can continue processing other tasks. This asynchronous behavior is reflected in our graph, where the curve indicates the handling of multiple requests over time as shown in Figure 12.

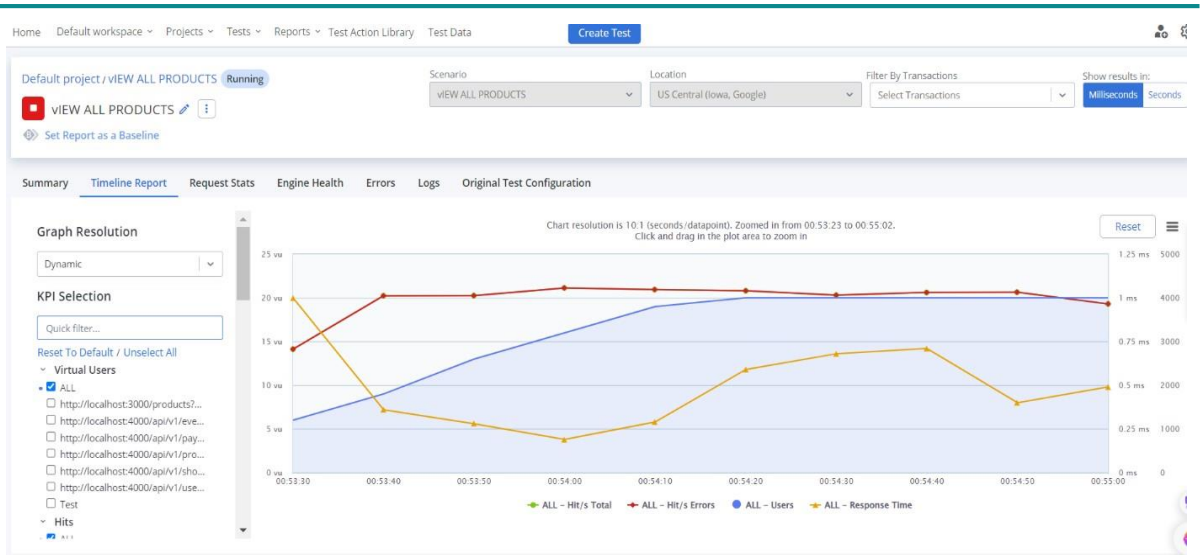


Figure 12: Graph Resolution of View All Products

Graph displays the asynchronous behavior so only one thread is handling all the requests, that is why graph is displaying such a curve. Detailed performance metrics and reports will generate, allow to identify, and analyze performance bottlenecks in the application. This information is crucial for optimizing code, database queries, and other components that may impact performance. Automation results for performance tests allows for repeated and consistent testing, making it easier to identify performance issues early in the development process. It also facilitates integration with continuous integration/continuous deployment (CI/CD) pipelines.

CONCLUSIONS

Asynchronous behavior of all request handling procedure graph presents the curve . It is obvious that multi users increasing status using login functionalities response and execution time is appropriate. Related dependencies are working smoothly. Error fetching scenario is directly proportional to user joining capacity. Discrepancies are aligned with view product details, product updating requests and inventory updating modules. GET and POST queries in Postman integrated with JMeter initiate along with BlazeMeter unit strength to show individual endpoint results in details and summary report format.

ACKNOWLEDGMENT

The Authors wish to thank Department of Computer Science COMSATS University Islamabad, Lahore Campus, Pakistan for support and help in learning Software Testing Concepts and Implementation, Software Quality Assurance, Software Requirement Engineering, Concepts of Formal Methods and Perceptions of Object Orientation Software Engineering etc.

REFERENCES

Awotar, M. & Sungkur, R. K. (2018). Optimization of Software Testing. International Conference on Computational Intelligence and Data Science (ICCIDS 2018). *Procedia Computer Science, 132*(2018), 1804–1814.

Felderer, M., Haisjackl, C., Pekar, V., & Breu, R. (2014). A Risk Assessment Framework for Software Testing. In T. Margaria & B. Steffen (Eds.), *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*. ISoLA 2014. Lecture Notes in Computer Science, vol 8803. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-45231-8_21

Quadri, S.M.K. & Farooq, S. U. (2010). Software Testing – Goals, Principles, and Limitations. *International Journal of Computer Applications, 6*(9), 7-10.

Reid, S. (2012). The New Software Testing Standard. In C. Dale & T. Anderson (Eds.), *Achieving Systems Safety*. Springer, London. https://doi.org/10.1007/978-1-4471-2494-8_17