Haseeb Ahmed(21f-9651)  
Aima Rashid(21f-9511)  
Aiman Zahid(21f-9485)  
Summary Report

**Phase 3**

In phase 3 our team zzlook-tech choose the project of Java (Sprintboot) because Java Spring Boot is a powerful framework that simplifies the development of robust, scalable, and production-ready applications in Java.

**Task 1:**

In this task, the integration of static analysis tools aimed to enhance code quality and maintain consistent coding standards across multiple languages within the project**. Checkstyle**, applied specifically to Java, enforced coding conventions and stylistic rules. Custom configurations were introduced: `no-single-char-variable`, which alerted developers to avoid single-character variable names, and `opening-brace-same-line`, prompting the placement of opening braces on the same line as the preceding code. Both configurations were set to generate warnings to prompt necessary adjustments.

**The Semgrep** configuration primarily focused on whitespace consistency with the `WhitespaceAround` module, ensuring uniformity in whitespace usage across different code constructs.

Throughout the integration process, identified issues flagged by Checkstyle and Semgrep were promptly addressed in the codebase. This comprehensive approach not only enforced coding standards but also detected potential issues related to variable naming, brace placement, and whitespace inconsistencies. The implementation resulted in improved code quality, minimized coding discrepancies, and facilitated consistent formatting practices, thereby contributing to a more maintainable and standardized codebase across various programming languages used in the project.

**Task 2:**

In the Java project's source folder, a test directory was established, further dividing it by creating a performance folder. JMeter, chosen among other tools, facilitated performance testing. The process commenced by creating a thread group within JMeter, specifying the number of users and defining requests in milliseconds to simulate concurrent users and their interactions with the system.

The selection of the HTTP request type allowed the configuration of specific requests for testing against the designated system. Post-execution, listeners were employed to analyze results. These listeners encompassed graphical outputs providing visual representations, tabular formats for comprehensive data insights, and the ability to save test results to a file for further analysis.

This structured approach within JMeter enabled comprehensive performance testing, allowing simulation and evaluation of system behavior under varying loads and scenarios. The use of listeners empowered detailed result analysis, providing graphical representations and tabulated data for an in-depth understanding of system performance.

All the performance output from Jmeter will be stored in performance folder in java project.

**Task 3:**

The task necessitates defining basic, testable non-functional requirements (NFRs) for a generic web API, covering four quality characteristics: Performance, Reliability, Security, and Developer Guidelines, aligned with ISO 25010 sub-attributes and recognized references like OWASP cheatsheets.

For Performance, the API should respond to 95% of requests within 500 milliseconds under a load of 1000 concurrent users and handle a maximum of 10,000 concurrent requests within 750 milliseconds. Reference points include ISO 25010's Time Behavior.

Regarding Reliability, the API must maintain an uptime of 99.9% under standard conditions, and it should be available 99.99% of the time under a maximum load of 10 million concurrent users. Additionally, it should handle single-node failures seamlessly, recovering within 5 minutes to its last consistent state. ISO 25010 attributes like Maturity, Availability, Fault Tolerance, and Recoverability guide these requirements.

Security demands encryption for sensitive data transmission, checksums or hashing mechanisms for data integrity, immutable transaction logs, traceable user actions, and robust authentication mechanisms like OAuth 2.0. References include OWASP guidelines on Transport Layer Protection, Data Integrity, Non-repudiation, Accountability, and Authentication, aligning with ISO 25010 Security sub-attributes.

Regarding Developer Guidelines, it's crucial to adopt contributor guidelines akin to established open-source projects like Kubernetes, TensorFlow, or Node.js. This involves structuring the codebase following the Model-View-Controller (MVC) architectural pattern, ensuring clear separation of concerns across models, views, and controllers. The documentation within the repository should reflect these guidelines.

Exploring different Contributor Guidelines documents from diverse open-source projects aids in understanding ideal project structures, particularly those employing the MVC framework. Analyzing the provided codebase helps in elucidating its structure, emphasizing the adherence to MVC principles and the clear segregation of components. The repository's `docs` folder should contain comprehensive files outlining Performance, Reliability, Security, and Developer Guidelines in line with these requirements and references.

This meticulous adherence to industry-standard references and recognized guidelines ensures the establishment of robust, testable NFRs encompassing ISO 25010 sub-attributes for each quality characteristic. These guidelines serve as a solid foundation for the development and assessment of a resilient and secure generic web API.