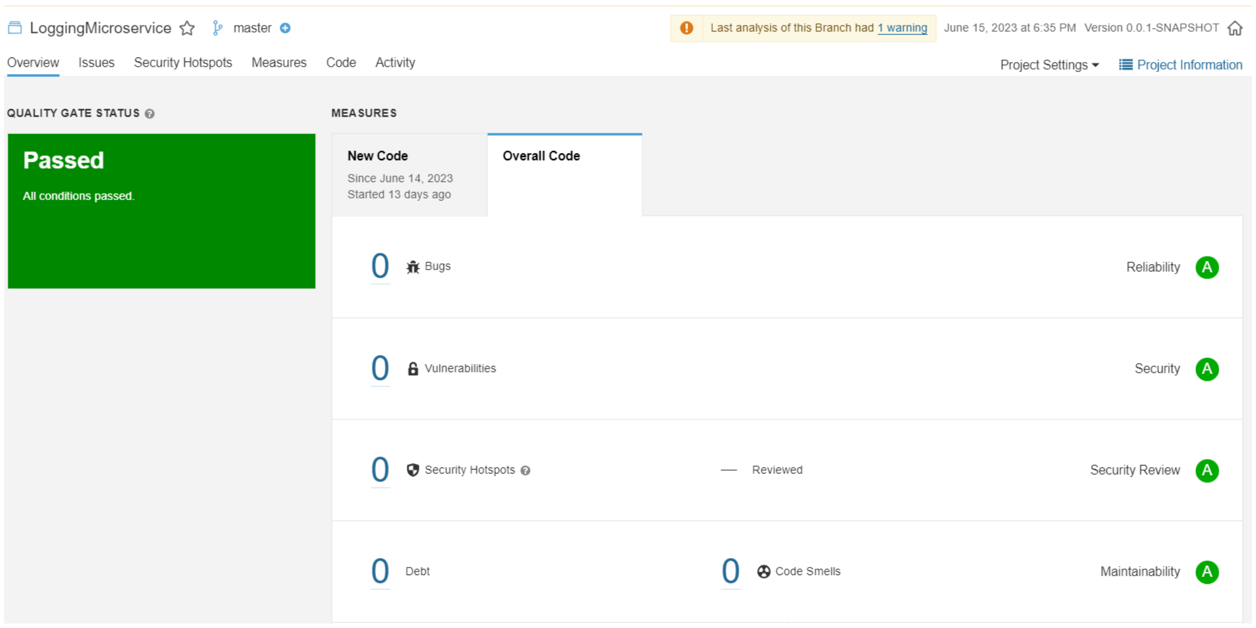
CTS CAFE

Logging Microservice using SpringBoot

Quality Assurance Report


LoggingMicroservice master

Last analysis of this Branch had 1 warning June 15, 2023 at 6:35 PM Version 0.0.1-SNAPSHOT

Overview Issues Security Hotspots Measures Code Activity Project Settings Project Information

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code
Since June 14, 2023
Started 13 days ago

Overall Code

0 Bugs		Reliability	A
0 Vulnerabilities		Security	A
0 Security Hotspots	— Reviewed	Security Review	A
0 Debt	0 Code Smells	Maintainability	A

Introduction

The purpose of this Quality Assurance (QA) documentation is to establish a systematic approach to testing, defect management, and quality control for the CTS CAFE project. It provides guidelines and instructions for QA engineers, developers, testers, and other stakeholders involved in the software development process

Scope

It entails the development of quality standards and best practices, as well as test planning, design, execution, defect management, and reporting. In order to guarantee consistency, traceability, and adherence to quality objectives, the documentation defines strategies, methods, and guidelines. It covers topics such as test case design, execution methods, defect tracking and detection, test reporting formats, and the use of testing resources and tools. The goal of software QA documentation is to give clear instructions and documentation to assist in the delivery of high-quality software products. The breadth of the documentation is suited to the project's demands.

Test Strategy

Objectives

- Verify that the program satisfies the functional specifications listed in the CAFE requirements document.
- Make that software carries out the desired features and functionalities precisely, effectively, and dependably.
- Find any differences or inconsistencies between the application's actual behavior and its intended behavior.
- Verify how well the program communicates with external systems and databases.

Environment

Test Tools:
SonarQube, IntelliJ

Softwares:
SpringBoot, SonarQube

Test Cases for API Logger:

TC_001: Registering a new user
Input: Invoke the API logger to register a new user.
Expected Output: User entry created with details of the user registration.

TC_002: Registering a user with an existing username
Input: Invoke the API logger to register a user with a username that already exists.
Expected Output: An error message indicating the username is already taken.

TC_003: Invoking the API logger with no parameters
Input: Invoke the API logger without providing any parameters.
Expected Output: An error message indicating missing parameters.

TC_004: Validating API key
Input: Validate an API key using the API logger.
Expected Output: A message saying “Invalid API Key or App ID”.

TC_005: Validating App ID
Input: Validate an App ID using the API logger.
Expected Output: A message saying “Invalid API Key or App ID”.

TC_006: Providing all details to the postLog function
Input: Invoke the postLog function of the API logger with all required details.
Expected Output: Log entry created with the provided details.

TC_007: Omitting log level in postLog function
Input: Invoke the postLog function of the API logger without specifying the log level.
Expected Output: Log entry created with a default log level = info assigned.

TC_008: Validating API key and App ID in getLog
Input: Invoke the getLog function of the API logger with API key and App ID validation.
Expected Output: Log entries fetched based on the API key and App ID validation.

TC_009: Not giving the date in getLog
Input: Invoke the getLog function of the API logger without specifying a date.
Expected Output: Log entries fetched without any date filtering.

TC_010: Giving the date in getLog
Input: Invoke the getLog function of the API logger with a specific date.
Expected Output: Log entries fetched for the specified date.

TC_011: Giving an input with malicious code
Input: Provide an input to the API logger that contains malicious code.
Expected Output: The filtered input with all the malicious code stripped.

TC_012: Giving an input above 1000 characters
Input: Provide a long input (more than 1000 characters) to the API logger.
Expected Output: The first 1000 characters of the log message, after the remaining has been filtered.

Process

Planning:

- **During the planning phase:**
    1. Created test cases and prepared relevant test data.
    2. Installed and set up SonarQube, imported the necessary dependencies in the application such as Jacoco Plugin and Spring Boot Starter Test.

- **Execution of test cases**:
    1. Ran edge cases through API Testing tools such as Postman.
    2. Generated a SonarQube report by setting up a connection.

- **Documentation:**
    1. Documented the test cases, data, and execution steps for future reference.

- **Benefits of planning:**
    1. Ensured comprehensive testing with well-defined test cases and accurate data.
    2. Were able to rectify any vulnerabilities, defects and deliver code adhering to the industry standards.

Execution:

- **Testing on Postman:**
    1. We tested the application on Postman by hosting it on localhost, sending requests to Postman as per our edge cases.
    2. We took notes on several outputs, testers ensured that all cases passed, and incase of failure, they would be rectified.

- **SonarQube Report:**
    1. SonarQube was set up on a machine with all dependencies installed and it was executed.
    2. The reports highlighted all vulnerabilities, such as regex patterns, code smells, etc and they were patched,
    3. Testers ensured that all relevant messages were recorded for analysis and debugging purposes.

Analysis:

- During the analysis phase, we encountered the following errors:
- Regex Pattern: The regex pattern used for filtering the log messages were at the risk of causing a stack overflow.
- Null Username: Users were able to register without passing any arguments.
- Lack of PUT Response: The put response was not returning an authentication for the users request,

Test Case	Input	Expected Output	Output	Remarks - Pass/Fail

TC_001	Registering a new user	JSON containing the apiKey, appID and username.	{
			"appName": "Testing",
			"appId": "3eda2750-574f-4c69-a6ab-903e4e6f950f",
			"apiKey": "a7b47158edbf407bbdde791ae58dc152"
			}
				Pass

TC_002	Registering a user with an existing username	An error message indicating the username is already taken	"timestamp": "2023-06-28T00:41:31.933+00:00", "status": 500, "error": "Internal Server Error", "trace": "java.lang.IllegalArgumentException: Name has already "path": "/register" }	Pass
TC_003	Invoking the API logger with no parameters	An error message indicating missing parameters	"timestamp": "2023-06-28T00:44:10.269+00:00", "status": 500, "error": "Internal Server Error", "trace": "java.lang.IllegalArgumentException: The name parame	Pass
TC_004	Validating API key	A message saying "Invalid API Key or App ID".	"timestamp": "2023-06-28T00:44:45.881+00:00", "status": 500, "error": "Internal Server Error", "trace": "java.lang.IllegalArgumentException: Invalid API Key or	Pass
TC_005	Validating App ID	A message saying "Invalid API Key or App ID".	"timestamp": "2023-06-28T00:44:45.881+00:00", "status": 500, "error": "Internal Server Error", "trace": "java.lang.IllegalArgumentException: Invalid API Key or	Pass
TC_006	Providing all details to the postLog function	Log entry created with the provided details	Logs successfully stored in Testing	Pass
TC_007	Omitting log level in postLog function	Log entry created with a default log level assigned	Logs successfully stored in Testing	Pass
TC_008	Validating API key and App ID in getLog	Log entries fetched based on the API key and App ID validation	Invalid API Key or App ID	Pass

TC_009	Not giving the date in getLog	Log entries fetched without any date filtering	[
			{
			"date": "2023-06-26",
			"logLevel": "info",
			"ClassName": "Module Within from where it''s called.",
			"time": "19:55:18.1216255",
			"message": "This has the content"
			}
				Pass

TC_010	Giving the date in getLog	Log entries fetched for the specified date: 15/4/2023	[]	Pass
TC_011	Giving an input with malicious code	The filtered input with all the malicious code stripped.	Logs successfully stored in Testing	Pass
TC_012	Giving an input above 1000 characters	The first 1000 characters of the log message, after the remaining	Logs successfully stored in Testing	Pass

Conclusion:

The QA documentation and testing process has successfully established a systematic approach to ensure the quality and reliability of the software. It covers various aspects of testing, defect management, and quality control, providing clear guidelines for stakeholders involved in software development. The test strategy incorporates appropriate tools such as SonarQube, Jacoco, and IntelliJ, and defines specific objectives to enhance the comprehensiveness and accuracy of the testing process. The execution of careful test cases focuses on critical elements like user registration, log storage, handling malicious input, managing duplicate entries, and data filtration.

The results of the test cases have provided valuable insights for improving the logging microservice. Additionally, the inclusion of a tabular column facilitates easy visualization of the expected output, actual output, and remarks associated with each test case, enabling a clear overview of the test results.