Martin Eskaros

261031885

ECSE429 Software Validation

1. Introduction

The purpose of this project is to validate the functionalities and robustness of the REST API Todo List Manager, an application that allows users to manage todo items via a RESTful API. As part of the project, I conducted two main testing activities: **Exploratory Testing** and a **Unit Test Suite.**

In the exploratory testing phase, I aimed to identify both documented and undocumented capabilities of the application, focusing on areas related to **todos** as outlined by the project guidelines. This testing method enabled me to explore the application's behavior and find potential areas of instability. The unit testing phase involved implementing a suite of tests to ensure that each API endpoint behaves as expected.

The entire project has been conducted within a controlled environment, where each test session was timeboxed to 45 minutes to ensure focused and consistent testing efforts. This report summarizes the key deliverables, findings from the exploratory testing, the structure of the unit test suite, and results from the unit test executions.

2. Summary of Deliverables

The deliverables for this project consist of several artifacts generated through the testing process:

- **Session Notes**: These notes document the exploratory testing sessions conducted on the REST API Todo List Manager. I tested each API endpoint using **curl** commands to verify if the actual behavior aligned with the expected one. My session notes include detailed observations, outputs, and screenshots for reference.
- **Unit Test Suite**: The suite was developed using **JUnit** and **OkHttp** libraries in Java to facilitate HTTP request testing. The suite includes comprehensive tests covering both documented and undocumented API functionalities. I ensured that each test is capable of running independently and assesses API behavior for correctness, side effects, and proper error handling.
- **Video of Unit Tests**: A video demonstration is included to showcase the execution of unit test.

- **Source Code Repository**: The source code and testing artifacts have been uploaded to a GitHub repository, organized with clear directory structures for ease of navigation. This repository serves as a central location for accessing the project code, unit tests, and other associated files.

The deliverables collectively provide a comprehensive overview of the testing efforts and outcomes, forming the foundation for validating the API's stability and functionality.

During the exploratory testing sessions, I engaged with the REST API Todo List Manager's functionality to identify both its documented capabilities and potential areas of instability. The testing was conducted using **curl** commands, and each session was timeboxed to 45 minutes. My findings are summarized below:

3.1 Overview of Testing Approach

The testing focused on **todos** capabilities, as outlined in the project requirements for a single-member team. By using **Charter Driven Session-Based Exploratory Testing**, I identified how well the application aligns with its intended use and documented behavior. I aimed to uncover both functional strengths and limitations, allowing for a thorough examination of the API.

3.2 Session Findings

- **API Functionality**: Overall, the documented API functions performed as expected. I observed that most of the endpoints worked correctly, with intuitive error messages that helped diagnose incorrect requests or malformed payloads.
- **Error Handling**: The application handled errors reasonably well, although certain error messages could be more descriptive, particularly for complex curl commands. Notably, when XML payloads were used to update objects, I encountered some inconsistencies compared to JSON handling.
- **Unexpected Behaviors**: I found that the POST method accepts specific IDs, which is typically associated with PUT requests. This behavior may not be immediately intuitive to users familiar with REST API conventions and should be documented to avoid confusion.
- **Security Concerns**: There is a shutdown endpoint within the API, which allows clients to terminate the server. While this may serve certain use cases, it could pose a security risk if exposed in a production environment.

- **XML Payload Handling**: The API shows inconsistent behavior with XML payloads, indicating that XML may not be as well-supported as JSON. This could impact users who rely on XML for data interchange.

3.3 Concerns and New Testing Ideas

- **Concerns**:
    - The shutdown endpoint could allow unauthorized clients to halt the server, which might expose vulnerabilities.
    - The POST method's acceptance of specific IDs could lead to unexpected results if not properly documented.
- **New Testing Ideas**:
    - Test the API with null parameters to evaluate error handling in these cases.
    - Explore less common HTTP methods like PATCH and COPY to assess how the API responds to unsupported commands.
    - Test inputs with HTML payloads to observe behavior and potential vulnerabilities.
    - Run tests on different operating systems to check for cross-platform compatibility.

4. Structure of Unit Test Suite

The unit test suite was developed using **JUnit** and **OkHttp**, both of which provide a flexible foundation for testing REST APIs in Java. The suite is designed to ensure that each API endpoint functions as expected and that errors are handled gracefully. Below is an outline of the suite's structure and key considerations:

4.1 Overview

The unit test suite contains multiple test modules, each targeting a specific API endpoint. These modules were designed based on findings from the exploratory testing phase, with separate tests for both documented and undocumented functionalities. The suite covers:

- **Confirmation of Expected Behavior**: Tests verify that the API meets documented specifications and performs expected operations.
- **Error Handling**: Tests assess how the API responds to malformed inputs, such as incorrect JSON or XML payloads.

- **Side Effects**: Each module includes tests to confirm that API actions affect only the intended data and do not produce unexpected side effects.
- **Return Codes**: Tests verify that each endpoint returns appropriate HTTP status codes, such as 200 for successful requests or 404 for non-existent resources.

4.2 Key Test Cases

- **Initialization and Teardown**: Each test module begins by setting up the required preconditions, saving the system state, and cleaning up any residual data afterward. This ensures that tests are isolated and repeatable.
- **Malformed Payloads**: Specific tests send malformed JSON and XML payloads to the API, confirming that errors are properly flagged and that the system remains stable.
- **Invalid Operations**: Tests include attempts to delete already-deleted objects and other operations that should produce predictable failures.

4.3 Code Quality

The test suite follows **Clean Code** principles, as recommended by Bob Martin. Each test module is written with clarity, simplicity, and proper structuring in mind, which facilitates maintenance and further development. The use of descriptive method names and consistent formatting aids readability and helps others understand the test logic.

5. Description of Source Code Repository

The project code and associated test files are stored in a GitHub repository, ensuring easy access and version control. Below is an overview of how the repository is structured and managed:

5.1 Repository Structure

The tests will be in the following directory

- **test/java/ca/mcgill/ecse429/todos**: The test directory holds the unit test suite. Each API endpoint has a dedicated test module, organized by functionality.

6. Findings of Unit Test Suite Execution

The unit test suite was executed in the development environment using **JUnit**, and the tests were run in random order to verify their independence. Below is a summary of the findings from this phase:

6.1 Overview of Execution Results

- **Overall Success Rate**: All unit tests passed, confirming that the API's functionality aligns with the expected behavior. The tests confirmed that typical operations, such as creating and deleting todos, performed correctly without unintended side effects.

6.2 Key Learnings and Recommendations

- **Improve XML Support**: Given the issues encountered with XML payloads, it may be beneficial for the API to either enhance support for XML or clearly document any limitations. This would improve user experience for those requiring XML data formats.
- **Clarify POST/PUT Usage**: To align with standard REST practices, it is recommended to either adjust the API to restrict POST from accepting specific IDs or document this behavior as an intentional design choice.
- **Secure the Shutdown Endpoint**: The shutdown endpoint could be a security vulnerability, especially if the API is exposed beyond a controlled environment. Limiting access or requiring authentication for this endpoint would be prudent steps to mitigate this risk.

The findings from the unit test suite execution provide a well-rounded view of the API's current state and highlight areas where further refinement may be beneficial. Overall, the API exhibits solid performance but could benefit from minor adjustments to enhance usability and security.

7.0 Final Thoughts

This project has emphasized the importance of both exploratory and unit testing in validating an API's functionality and resilience. While the REST API Todo List Manager is largely reliable, these targeted improvements could further solidify its position as a robust and user-friendly tool. Overall, this testing experience highlights the critical role of thorough validation in uncovering not only functional defects but also opportunities for enhancing security and adherence to standards.