Timothy Johnson  
December 12th, 2024  
CS 320  
Professor Tuft

Project Two: Summary & Reflection

In this project, I wrote unit tests for three services: Contact, Task, and Appointment. Aiming to ensure their functionality and meet the project requirements. I designed these tests to check the services' behavior under various scenarios, covering valid inputs, invalid inputs, and edge cases. This report discusses my unit testing approach, the challenges I faced, and my reflections on the process.

For each service, I focused on testing essential operations such as adding, updating, and deleting data. In the Contact Service, I tested fields like phone numbers and names. For example, I wrote tests to ensure phone numbers were exactly 10 digits long and names didn't exceed 10 characters. Invalid inputs, like non-numeric phone numbers, were tested to ensure the system threw appropriate exceptions.

In the Task Service, I ensured that tasks could only be created with valid IDs, names, and descriptions. I tested for tasks with IDs exceeding 10 characters and descriptions longer than 50 characters, ensuring the system rejected them. A key test confirmed that tasks with invalid IDs were not added.

For the Appointment Service, I verified that appointments could only be scheduled with future dates and valid descriptions. Tests ensured the system rejected past dates and overly long descriptions, providing reliable error handling.

My approach focused on handling both common and edge cases. For example, in the Contact Service, I checked that invalid data like null values or incorrectly formatted phone numbers would trigger exceptions. By testing these edge cases, I confirmed the services could handle unexpected scenarios effectively.

The quality of my tests was high, as they covered a range of possible inputs and ensured the system behaved correctly in both expected and unexpected scenarios. Although I didn’t directly measure test coverage, I made sure that each critical functionality was tested. As the JUnit 5 User Guide states, “Good tests cover edge cases and ensure the application behaves as expected under all conditions” (JUnit 5 User Guide). My tests adhered to this principle, ensuring thoroughness.

I implemented several key JUnit assertions to verify the correctness of the functionality. For example, I used assertEquals(expected, actual) to check that the phone number of a contact matched the expected value after an update. The assertTrue(condition) assertion was useful for confirming that a phone number matched the required 10-digit format in the Contact Service. Additionally, I used assertFalse(condition) to ensure that invalid data, like an ID longer than the allowed limit in the Task Service, was rejected. Assertions like assertNull(object) and assertNotNull(object) helped confirm that invalid or missing data returned the expected null values, while valid inputs returned the correct results.

These assertions ensured that the system handled both valid inputs and error conditions effectively. By focusing on each scenario—valid data, invalid data, and edge cases—I ensured that the system would perform correctly in a variety of conditions.

Through this process, I learned the importance of testing edge cases and invalid inputs. Initially, I found it challenging to organize my tests effectively, but I gradually learned to write simple, focused tests that each checked one specific functionality. For example, I wrote a test to verify that the system rejected appointments with past dates in the Appointment Service. This approach made the tests easier to maintain and understand.

I also prioritized efficiency by reusing helper methods to generate test data. This helped avoid redundancy and kept my code clean and maintainable. For example, I reused the method to generate valid task IDs across multiple tests, reducing duplication.

Unit testing was the primary technique I used for this project. It allowed me to test individual methods in isolation, helping me catch small bugs early in development. While I didn’t use integration or system testing in this project, these techniques are important for larger systems. Integration testing ensures that different components work together, while system testing checks the entire system's behavior. In future projects, I plan to integrate these testing methods for comprehensive coverage.

Throughout this project, I adopted a cautious mindset, being careful to test not only expected behaviors but also error conditions. As noted in the JUnit 5 User Guide, “Testing edge cases is critical to ensuring robustness” (JUnit 5 User Guide). I focused on testing scenarios where the code might fail, such as handling null values or invalid inputs. In the Task Service, for instance, I wrote tests to ensure the system rejected tasks with invalid IDs, even though these situations were rare.

Maintaining discipline was essential for ensuring the quality of my tests. I avoided cutting corners by thoroughly testing edge cases like past dates and overly long descriptions. By being disciplined in my approach, I ensured that my code was reliable and avoided potential issues down the line.

This project reinforced the importance of unit testing in software development. By writing comprehensive tests for the Contact, Task, and Appointment services, I ensured that each component worked correctly under various conditions. Moving forward, I plan to continue applying these principles to ensure high-quality, reliable software.

References

Baeldung Unit Testing Guide. Retrieved from <https://www.baeldung.com/junit-5>

JUnit 5 User Guide. Retrieved from <https://junit.org/junit5/docs/current/user-guide/>