Project Two

Jacob Englund

CS 320

Professor Springer

August 17, 2024

For the Contact Service, my unit testing approach focused on making sure that contacts were created, updated, and deleted properly, following all the rules. For example, I tested creating contacts with both valid and invalid IDs, names, and phone numbers. For the Task Service, I also tested adding, deleting, and updating tasks while making sure the task ID, name, and description met the rules. My tests confirmed that tasks could not be created with missing or incorrect data. When working on the Appointment Service, I made sure appointments were added and deleted correctly, and that the date field could not be set to a past date. The tests also checked that the appointment ID and description followed the length limits.

My unit testing approach closely followed the software requirements for each service. For example, in the Contact Service, I made sure the test cases checked all the required fields, like contact ID and phone number, based on the specifications. Similarly, the Task and Appointment Services had tests that matched the rules for field lengths, null checks, and date validation. For example, in the Contact Service, the JUnit tests made sure that the phone number field had exactly 10 digits, as required.

Overall, I believe my JUnit tests were effective because they covered both positive and negative scenarios, making sure the system behaved as expected in different situations. The high-test coverage percentages showed that most, if not all, of the code paths were tested. For example, the test coverage report indicated that all important methods in the Task Service were covered. Writing JUnit tests was a learning experience because it made me think carefully about potential edge cases and failures in the system. I gained a better understanding of how to write code that can be tested and behaves correctly. To make sure my code was technically sound, I wrote tests that handled both expected and unexpected inputs. For example, in the Appointment Service, I tested for invalid dates, like past dates, to make sure the code was robust. My code was efficient because I focused on writing clear and concise test cases that covered multiple scenarios without repeating code. For instance, in the Task Service, I used parameterized tests to check different task name lengths in one method.

Testing interactions between components was not needed in this project because each service was separate with no external dependencies. System testing, which checks the whole system’s compliance with requirements, was not necessary since we focused on individual services without a full system setup. As a software tester, I approached the project carefully, especially when dealing with complex, interconnected code. For example, when testing the Appointment Service, I was cautious to ensure that the date validation logic did not accidentally allow invalid dates because of time zone differences.

To avoid bias, I looked at the code from an outsider’s perspective, questioning assumptions and thinking about how the code might fail. For example, when testing the Contact Service, I pretended not to know the code, which helped me catch common mistakes that a developer might miss. Being disciplined in maintaining quality was important throughout the project. Cutting corners in testing could lead to undetected bugs that might cause problems later. To avoid technical debt, I plan to always write thorough tests and refactor code when needed, instead of taking shortcuts. For example, making sure the Contact Service has proper validation helps prevent future issues with data integrity.

To sum it all up, my unit testing for the Contact, Task, and Appointment Services made sure that each part worked the way it was supposed to. I tested different situations, like both valid and invalid inputs, to make sure the system followed all the rules. Writing JUnit tests helped me improve my testing skills, especially when it came to thinking about unusual cases and making sure the code was solid. Even though I did not need to test the complete system, I tried to stay focused on doing things the right way and not cut corners. By looking at the code carefully and avoiding bias, I made sure that each service was dependable, efficient, and did not have any common mistakes.