CS-320 Project Two

Zoe Render

4/20/2025

The unit testing for the Contact Service confirmed that all business rules and constraints were properly validated. Each test case addressed scenarios such as adding a new contact, updating contact details, and deleting contacts. We employed boundary value analysis to verify field length constraints, such as for contact ID, first name, and last name. For the Task Service, testing ensured that task names and descriptions adhered to specified length requirements. Additionally, we verified that updates to these fields were accurately reflected and that invalid data was properly rejected. In the Appointment Service, our tests included validating future date constraints and ensuring the uniqueness of appointment IDs, with a focus on confirming that past dates were not accepted and that invalid appointment IDs triggered exceptions.

Our testing approach closely followed the software requirements outlined for the project. In the Contact Service, tests confirmed adherence to constraints, such as a 10-character limit for IDs and a 30-character limit for addresses. The Task Service tests aligned with requirements by validating updatable fields and ensuring that descriptions never exceeded 50 characters. For the Appointment Service, tests effectively captured date field constraints, rejecting past dates. This alignment is evident in tests like `testAppointmentConstructorInvalidDate` in `AppointmentTest`, where invalid date inputs were rigorously examined.

Overall, the JUnit tests were highly effective, achieving over 90% coverage across all classes, indicating that both standard and edge cases were adequately represented. This high coverage demonstrates our ability to identify and validate all functional pathways, including adding, updating, and deleting objects, as well as handling invalid inputs. Writing these JUnit tests was an iterative process that required attention to detail and a solid understanding of the software's requirements. To ensure technical soundness, we utilized assertions to validate expected outcomes, such as the example where we ensured that invalid inputs were appropriately rejected. We also focused on efficiency by grouping related tests and minimizing redundant setups,

streamlining the overall testing process through shared setup methods in classes like `ContactServiceTest`.

In reflecting on the testing techniques employed, we used Boundary Value Analysis (BVA) to test constraints such as string lengths for contact fields, applied Equivalence Partitioning to test valid and invalid input ranges, and conducted Exception Testing to ensure that invalid inputs triggered the appropriate exceptions. However, we did not engage in integration testing, as this focused on interactions among multiple modules and was irrelevant in our isolated context. Likewise, performance testing was outside the scope of this project. The practical implications of our testing techniques suggest that BVA and Equivalence Partitioning are particularly effective in input validation-heavy applications, such as forms and systems that handle user input. Integrational testing becomes critical in scenarios with multiple microservices, while performance testing is essential for high-traffic systems like e-commerce platforms.

Throughout this process, maintaining a careful mindset was crucial. Recognizing the interdependence of system components meant that modifying a method, like `updateAppointment`, required us to revisit tests to ensure no downstream issues occurred. We actively limited bias by reviewing code from a fresh perspective, focusing on potential failure scenarios. For instance, while testing the `ContactService`, we attempted to add duplicate IDs to simulate user errors. Our commitment to quality was paramount; we understood that cutting corners could lead to technical debt. Therefore, we adhered to a consistent testing framework and coding standards to minimize such debt. Automated tests were rerun after each code change to validate correctness, ensuring that regression issues were caught early.

In conclusion, this project underscored the significance of a systematic testing approach combined with a disciplined mindset. Utilizing techniques like BVA and exception testing enabled us to achieve robust test coverage. This experience has reinforced the importance of a thorough testing process in delivering high-quality software. Moving forward, I will continue to adopt these strategies while remaining committed to minimizing technical debt and ensuring maintainability through rigorous testing.

Sources;

JUnit 5 User Guide. (n.d.). https://junit.org/junit5/docs/current/user-guide/

Oracle. (n.d.). *Java SE documentation*. https://docs.oracle.com/javase/