Nikki Yanez

7-2 Project Two Submission

10/19/2025

**Summary**

My unit testing approach for the Contact, Task, and Appointment services was directly aligned with the project requirements. Each requirement was translated into a test case. For example, the Contact ID had to be unique and under 10 characters, which I verified by attempting to add duplicate IDs in ContactServiceTest. For the Task service, I confirmed that names over 20 characters and descriptions over 50 characters triggered exceptions. This alignment ensured my code enforced all constraints accurately.

My JUnit tests were effective because they achieved high coverage and tested both valid and invalid inputs. I used assertions such as assertEquals, assertThrows, and assertNotNull to validate that methods worked as intended and handled errors correctly. Each test isolated one function, which made it easier to identify and fix issues. The consistent use of boundary and exception tests confirmed the strength and completeness of the test suite.

To ensure technical soundness, I used @BeforeEach in my tests to create fresh instances before every run:

**@BeforeEach**

**void setUp() { taskService = new TaskService(); }**

This prevented data from one test affecting another. I also kept my tests efficient by combining related checks in single methods,for example, updating both task name and description in one test to reduce redundancy and speed execution.

**Reflection**

I mainly used unit testing and boundary testing. Unit testing focused on verifying each method independently, while boundary testing checked input limits, like ensuring that a name of exactly 20 characters passed and one of 21 failed. These techniques helped confirm that each class functioned correctly and handled edge cases.

I did not use integration, system, or user acceptance testing, which are used later in development to verify how components interact and meet user needs. My focus was ensuring internal logic and validation within each class, not full-system behavior.

Unit and boundary testing are ideal for early stage development, ensuring small pieces of code are reliable. Integration and system testing are more useful in complex systems with multiple interacting components, while user acceptance testing confirms readiness for deployment.

**Mindset**

I approached this project carefully and tested beyond my expectations of what should work. For instance, I intentionally tried invalid inputs like null IDs or past dates in AppointmentTest to challenge my assumptions. I limited bias by verifying each result against the written requirements instead of my intended design.

Being disciplined about testing prevents technical debt and ensures long-term reliability. Cutting corners would lead to bugs that are harder to fix later. I plan to maintain quality by keeping tests automated, consistent, and part of the development process for every future project.

This project strengthened my understanding of unit and boundary testing as essential foundations of software quality. Through disciplined, unbiased testing, I ensured each service met its requirements and built a reliable foundation for future software development.