**Summary**

When I worked on Project One I mainly used unit testing with JUnit to check if the services I built worked the way they were supposed to. For the contact service, I tested creating new contacts, making sure IDs, names, and phone numbers followed the rules, and I also tested updates to confirm nothing broke when data changed. In the task service, I checked adding, updating, and deleting tasks, and I made sure the name and description stayed within the limits. For the appointment service, I tested that appointments had valid future dates, duplicates weren’t allowed, deleting by ID worked, and that IDs and descriptions followed the requirements. Each of these tests tied directly to what the requirements asked for, so I felt like my approach lined up well with what was expected.

I think my JUnit tests were effective because they didn’t just cover the simple cases but also edge cases, like invalid IDs or descriptions that were too long. The coverage report showed that I was hitting almost all the methods, which made me more confident that the tests were actually useful. Writing these tests also helped me figure out how to make my code stronger and cleaner. I used assertions like assertEquals to check the output and assertThrows to confirm exceptions, which made the tests solid. To keep things efficient, I used @BeforeEach to set up test objects so I didn’t have to rewrite the same code every time. That made the test files shorter and easier to follow.

**Reflection**

The main testing techniques I used were unit testing and boundary testing. Unit tests let me check small parts of the code like single methods, while boundary tests let me see what happened when values hit the limits, like a name being too short or too long. I didn’t do integration, system, or acceptance testing for this project, but I know those play a bigger role in larger projects. Integration testing would make sure different parts of the program talk to each other correctly, system testing would check the whole app from start to finish, and acceptance testing would make sure the client is happy with the final product.

Each technique has its own use. Unit and boundary tests are great during development because they catch little problems before they become bigger ones. Integration testing is more useful once multiple modules are connected, while system and acceptance testing are better for later stages when the product is almost done.

Working on this project made me careful about testing. Instead of just checking the “good” cases, I made sure to test the bad ones too, like invalid dates in the appointment service or task descriptions that didn’t fit the limits. It showed me how important it is to think about how one small change can affect other parts of the program. I also made an effort to avoid bias. Since I was the one who wrote the code, it would have been easy to assume it was correct, but I tried to think like a tester and find mistakes instead.

I also realized how important it is to stay disciplined. It can be tempting to skip certain tests to save time, but that just builds up technical debt and creates bigger problems later. If I hadn’t tested edge cases, the bugs might have slipped through and been harder to fix after the fact. Going forward, I plan to always write thorough tests and keep them updated as the code changes. That way I can keep the quality high and avoid problems down the road.