For Project One, my unit testing approach for each feature was focused on ensuring that all software requirements were met and that the functionality of the mobile application was reliable. For the Contact Service, I created tests that specifically validated the rules for each field in the contact object. For example, I tested that the contact ID was unique, ensuring that no two contacts could have the same ID. I also wrote tests to confirm that fields like the first name, last name, and phone number were not only non-null but also within the allowed character limits. Each of these tests helped me ensure that the data integrity of the contact service was maintained as per the requirements. My testing approach aligned well with the software requirements, as all my test cases reflected the exact constraints set by the client. For instance, I knew that the maximum length for a contact's first name was 10 characters, so I created a test specifically to handle inputs longer than this limit.

When it came to the Task Service, my approach was similar in that I focused on validating the ID, name, and description fields. The task ID had to be unique and non-updatable, which I tested by attempting to create duplicate tasks and ensuring that the service rejected them. I also verified that task names and descriptions met the size constraints and could not be null. This approach ensured that every task added to the system adhered to the requirements and that any invalid tasks were properly caught during testing.

For the Appointment Service, my unit testing was centered around making sure that the appointment ID was unique and that appointment dates were valid. Since the dates couldn’t be in the past, I wrote specific tests to check that invalid dates (like those set before the current date) were rejected. I also ensured that descriptions were no longer than 50 characters and were never null. This attention to detail helped me ensure that the appointment service would only allow valid and future appointments to be added, directly reflecting the client's needs.

The quality of my JUnit tests was solid, as I aimed for coverage of at least 80% for all my testable code. This high coverage gave me confidence that the tests were effective. For example, my test cases for validating contact fields covered all the edge cases, such as ensuring the phone number contained exactly 10 digits and checking for null values across different fields. The coverage report showed that most of the critical paths in my code were exercised by my tests.

Writing the JUnit tests was a learning experience, but it helped me ensure that my code was both technically sound and efficient. I made sure that each test method focused on one specific behavior, like testing the validity of appointment dates. This way, if a test failed, I knew exactly which part of the code was the issue. For instance, in my testInvalidAppointmentDate method, I used a past date to confirm that the service correctly rejected it, which aligned with the requirements. I also kept my tests simple and efficient by avoiding redundant code and focusing on the core functionality.

In this project, I primarily used unit testing as my core software testing technique. Unit testing focuses on verifying the smallest parts of the application, in this case, individual methods and classes. This technique was effective because I could isolate each function and ensure it was working as expected. Unit testing is especially useful for projects where I need to validate specific rules or constraints, like ensuring that appointment dates are always in the future. This method allowed me to quickly identify and resolve issues at the method level before they affected the rest of the system.

There are other software testing techniques I did not use in this project, such as integration testing and system testing. Integration testing focuses on verifying that different modules or services work well together, while system testing checks the entire application in a complete environment. These techniques would be practical for larger projects or when multiple services need to communicate with each other. For example, if the contact, task, and appointment services interacted with each other, integration testing would be critical to ensure that adding an appointment doesn’t break contact management. Since my focus was on isolated unit tests, I didn’t need to employ these more advanced techniques, but they have their place in more complex projects.

When working on this project, I adopted a mindset of caution, knowing that small mistakes could lead to larger issues later on. As a software tester, it was important for me to appreciate the complexity of the code and how different parts of the system interrelate. For example, while writing tests for the ContactService, I realized that if I didn’t carefully validate the uniqueness of contact IDs, it could lead to duplicate entries, causing data integrity issues. Understanding this complexity helped me focus on writing thorough tests that would catch potential problems early.

Limiting bias was also important while reviewing the code. I made sure not to assume that my code was correct just because I had written it. By approaching the tests as if someone else had written the code, I could be more objective and catch issues I might have otherwise overlooked. If I had assumed that my code was flawless, I might have missed critical errors, like accidentally allowing duplicate IDs in the task service. Testing your own code can introduce bias, so it’s essential to be as objective as possible.

Finally, being disciplined in my commitment to quality was critical throughout this project. It would have been tempting to cut corners, especially when writing tests for every possible edge case, but I knew that doing so could lead to technical debt in the long run. By thoroughly testing every requirement and edge case, I avoided issues that could come back to haunt me later. For example, testing date validation in the appointment service might seem tedious, but failing to do so could lead to users scheduling appointments in the past, which would break the functionality of the application. As I continue to grow in this field, I plan to avoid technical debt by always committing to quality, even when it’s tempting to take shortcuts.