**Project Two**

**Mohamed Elmarzougui**

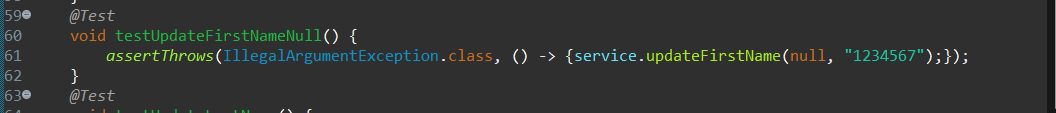
**CS-320 Software Test, Automation QA**

**10/16/2024**

* **Summary**

In Project One, the unit testing approach I adopted was designed to ensure that each component was thoroughly tested using the JUnit framework. My primary goal was to validate that the services met the requirements for creating, updating, and deleting objects, while also ensuring that constraints were enforced. For instance, in the contact service, tests focused on validating that unique identifiers were maintained, and that first and last names conformed to length and format requirements.

The testing approach I employed was closely aligned with the project’s software requirements. Each unit test was directly tied to a functional requirement of the system. For example, in the task service, the requirement that a task description could not exceed a certain character limit was translated into a JUnit test to confirm this constraint. Similarly, in the appointment service, validation tests ensured that appointments were created with valid start and end times.

The overall quality of the JUnit tests can be defended by analyzing the coverage percentage, which was 88%. This high-test coverage rate demonstrates near-complete coverage of all methods. The testing consisted of both positive tests, where inputs conformed to expectations, and negative test cases to test scenarios with invalid inputs. For example, in the contact service test method testUpdateFirstNameNull(), I tested a case where a null first name was passed. The test was designed to check whether an IllegalArgumentException was thrown, ensuring that the program would handle invalid input correctly:

Writing JUnit tests for this project was a positive experience that improved both the technical soundness and efficiency of the code. To ensure technical soundness, I consistently checked boundary cases and handled invalid inputs, which required writing tests for a wide range of scenarios. For instance, in the contact service, I wrote tests to cover both valid and invalid inputs for each attribute:



To ensure efficiency, I focused on writing concise and reusable tests while minimizing code duplication. This approach not only reduced redundancy but also made the testing framework more maintainable.

* **Reflection**

The primary testing technique I employed throughout Project One was unit testing, which involved testing individual components of the system in isolation. This approach allowed me to verify that each service functioned as expected without dependencies on other components. Unit testing is particularly valuable in the early stages of development because it provides almost immediate feedback, helping to catch issues early and ensuring that functionality is intact.

One technique I did not utilize in this project was integration testing. While unit tests focus on verifying the behavior of individual components, integration testing ensures that these components work together as expected when combined. Integration testing is essential in larger systems to ensure the interaction between modules is smooth and reliable.

The mindset I adopted during this project was cautious and methodical. Recognizing the complexity of the system and the relationships between its different parts, I made it a priority to anticipate potential issues before they could affect functionality. For example, unit testing ensured that a duplicate contact could not be added, safeguarding the uniqueness of the system's data. Here’s a simpler example from the code, illustrating how I used unit testing to confirm that duplicate contacts were not allowed.A computer screen with colorful text

Description automatically generated

Limiting bias in my review of the code was another important consideration. Since I both wrote the code and the tests, I made a conscious effort to review them objectively. To achieve this, I asked a friend to review key sections of my tests, providing an external perspective. For example, after writing the validation tests for the task service, I relied on feedback from my professor to ensure that the code logic was sound and that I wasn’t overlooking any edge cases or making assumptions about how the code should behave.

Discipline and commitment to quality are essential traits for any software engineer. Cutting corners, such as skipping edge case tests or ignoring performance considerations, can lead to the accumulation of technical debt, which complicates future updates or bug fixes. In this project, I made it a point to write comprehensive tests for all key functionalities and resisted the temptation to skip scenarios where potential bugs could arise.