7-2 Project Two

Tyten Perez

CS 320: Software Test Automation & QA

Ramsey Kraya

December 10, 2023

**Summary**

For the **Contacts** feature, I focused on validating the integrity of contact information. The ContactTest class, for instance, used assertThrows to confirm that invalid contact IDs, such as null or excessively long strings, would correctly trigger exceptions, as demonstrated in testContactIDInvalidNull. Similarly, tests like testContactIDValid1 ensured that valid IDs were accepted. In the ContactServiceTest, the testAddContact method was pivotal in confirming the addition of contacts, aligning with the fundamental requirement of contact management.

The **Task** component demanded an equally rigorous testing methodology. In the TaskTest, tests such as testTaskIDValid and testTaskIDInvalidNull checked for the correct implementation of task IDs. The TaskServiceTest suite, particularly through testAddTask and testDeleteTask, scrutinized the addition and deletion functionalities, ensuring the service's effectiveness in managing tasks.

For the **Appointment** code, I employed a similar strategy. The AppointmentTest suite included tests like testAppointment to test date limits for appointments, important for verifying an appointment was actually in a valid future date. The AppointmentServiceTest extended this by testing the addition and deletion of appointments. The testAddNewAppointment method also would confirm the successful addition of an appointment and its attributes.

**Testing Techniques**

Throughout the development process, I employed both positive and negative testing techniques. Positive testing was evident in scenarios where I tested the system's response to valid inputs, such as in testContactIDValid1. Negative testing was crucial for handling erroneous inputs, as seen in testTaskIDInvalidNull, where I expected the system to throw an exception.

There are several techniques that were not utilized in this project due to its size and stage of development. Stress or performance testing, which are generally more suited to scenarios demanding high scalability and performance under load were not used as the relevant infrastructure has not been developed. Integration testing was also not employed but would have tested the entire applications systems and how they interacted with each other. Since the Contact, Task, and Appointment objects in this project do not interact with each other there is little need for this test strategy. Automated testing was also skipped due to the project’s current size. In the event that the scope widens, Automated testing would be crucial for a continuous integration pipeline where every update needs to repeatedly retest the codebase.

**Mindset**

My mindset throughout was one of caution and continuous improvement. One way I did this was by examining implied specifications of the project. For each object there was to be no null fields, implying that empty strings were also not acceptable and therefore should be accounted for. My continuous improvement is evident in the more detailed and isolated testing of null object fields and character limits as initially they were combined in early iterations of the project. Furthermore, I had to carefully manage how each object interacted with each other, such as how ContactService.java managed Contact.java, was essential. One way this was achieved is by limiting how IDs were assigned between the two, ensuring that they could not be modified after initialization. This approach was critical in crafting comprehensive tests that covered a wide range of potential scenarios.

To limit bias, particularly the kind that can arise when testing one's code, I reminded myself to analyze my tests to ensure they were checking the right aspects of the code. In order to ensure I was properly testing code I needed to be specific. For example, checking that an exact exception is thrown in a test method. This can be seen in AppointmentTest.java with the testInvalidIDNull test method which checks if a null ID input throws the expected “Invalid ID” exception message.

In order to limit technical debt, a continuous process of improvement is necessary. This means consistent attention to detail and periodic reassessment. Without this companies are at risk of costly missteps where entire projects need to be rewritten for a better foundation that is fit for their vision. Furthermore, with the increasing reliance of technology with sensitive information, this process of self-reflection and improvement is not only important for a product’s long-term functionality but for protecting user’s sensitive data from hacking as well. In this project’s case, it would be imperative to closely monitor possible exploits for contact information to be leaked.

In conclusion, the development of this mobile application required a thorough and detailed approach to unit testing, ensuring that each object in the project functioned as intended and adhered to the customer’s requirements. During this development, JUnit tests proved invaluable, serving as a benchmark for my code quality. It didn’t just help confirm that the intended functionality is present, but also rigorously ensure that it operates without unintended side effects.