**Project 2**

Edward Garcia

CS320-R4845-Software Test Automation & QA

Professor Kraya

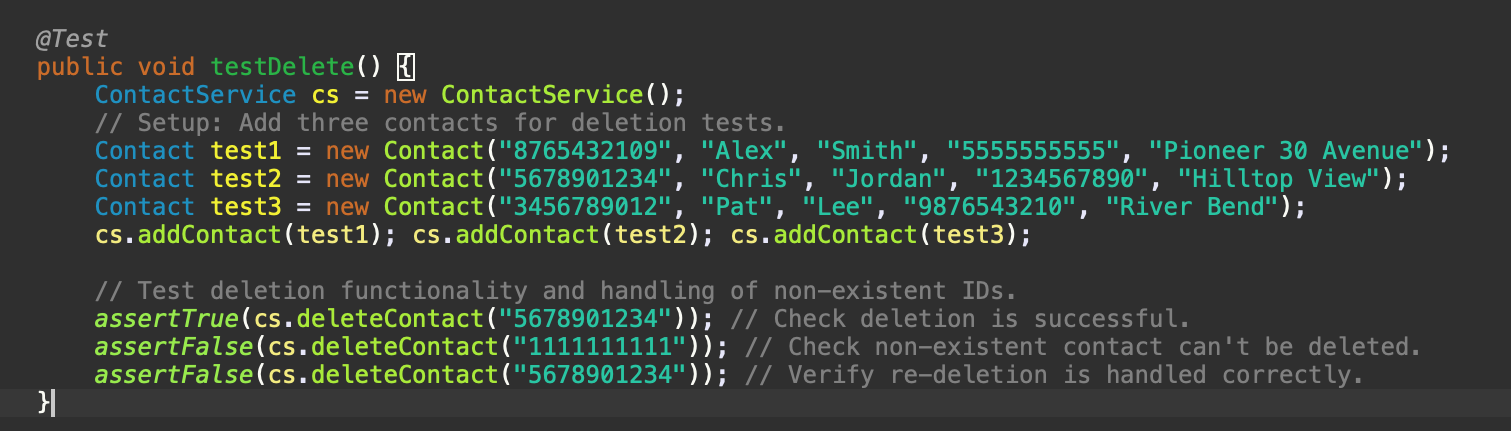
April 20, 2024

**Project 2**

**Summary**

A screenshot of a computer

Description automatically generatedDuring my development process of the mobile application for the customer, I followed an effective unit testing approach. I made sure to structure around the functionalities of the contact, task, and appointment services. Within the Contact Service, I designed tests to validate the creation, deletion, and updating processes within the application following specific constraints such as ID length and phone number formatting. My tests within the Contact Service included scenarios to handle edge cases like maximum field lengths and invalid input formats. The Appointment Service tests I designed on ensuring that all of the appointments could be accurately created with valid IDs, future dates, and descriptions adhering to length and unique id requirements. I also included tests within the appointment service that tested against past dates and duplicate IDs. Within the Task Service class I employed tests to determined if tasks maintained valid IDs, names, and descriptions, while at the same time verifying if updates and deletions were handled correctly. I used a combination of black box and white box testing to make sure my tests functioned correctly. Entering data and making sure the right output was generated was part of the Black box testing. Examining the code and making sure it is logical was part of White box testing. In my attached coverage screenshot below, I was able to attain 90% coverage which confirms that the testing methods were effective and efficient.

I made sure that my testing approach aligned with the software requirements by structuring each test to confirm that each specific service stayed within their boundaries. For example, in the contact service I implicated a test to make sure that the contact IDs were exactly ten characters long and in the appointment service I made sure the appointments were not set in the past. In my provided code example below, I give an example of my implementations within the contact service. The test verifies the delete functionality and handling of non-existent IDs.

A screen shot of a computer program

Description automatically generatedI was able to keep my JUnit tests effective by analyzing high coverage metrics within the scope of the program and making sure that each of the test cases had sufficient parameters for testing. Tests such as validating the non-acceptance of past dates in appointments and enforcing ID length in contacts are prime examples of my tests' alignment and quality. In the code example below I created a test named “testAppointmentDetailsMatchUponCreation” within the AppointmentServiceTest class. This test verifies that the details of each appointment are set correctly and that the data and description fields match what is specified.

A screenshot of a computer program

Description automatically generatedWriting the JUnit tests was exciting and highly informational for me. I gained much incite on how significant testing can be in the creation of efficient software. I found the Junit tests to be somewhat challenging, which allowed me to have a deeper understanding of the application’s functionality and edge cases. For me to ensure technical soundness within my code, I utilized assertions to compare expected outcomes with actual outcomes in order to confirm the code was able to function under various conditions. In my given code example below I created a test to verify that the UUIDs generated for the task service are unique. My efficiency in testing was achieved by creating each test case independently and allowing them to be repeatable. This strategy helped to eliminate errors, increase my test coverage, and align with the specified requirements.

**Reflection**

In this project I used several software testing techniques which consisted primarily of black-box testing. Black-box testing’s practical use is used across various types of software, from web applications to mobile apps and backend services (Hambling, 2019). It's highly effective for validating user interfaces, APIs, and other system endpoints. Black box testing processes focus on input and output validations to test the functionality of the code, without considering the internal code structure. I made sure to exercise caution thoroughly by reviewing test cases to verify the comprehensive coverage is efficient for all possible scenarios. For example, I tested edge cases to identify any potential bugs that could be triggered under certain conditions. I mitigated bias by involving multiple test scenarios with different perspectives. A consistent approach I feel requires discipline in both testing and within the code development. I was able to demonstrate discipline by following defined code standards and keeping a strict testing schedule. I also used regression testing to ensure that new changes did not disrupt existing functionalities. Regression testing is particularly important in continuous integration environments where new code commits are integrated regularly. Regression testing helps to ensure that new changes do not disrupt the existing functionality of the application. Some testing techniques that I did not use were performance and stress testing, as the project scope primarily focused on functionality.

**Mindset**

Adopting a cautious mindset for me was a vital aspect to have during every step of my code structure and implementation. This mindset helped me to anticipate how changes in one part of the system could affect others. For instance, when updates were made to the Contact Service, such as adding a new field for contact type, I had to consider the wider implications on other services. This change required the implementation of integration testing to ensure that the new contact types interacted smoothly with the Appointment and Task Services. When software developers are tasked with testing their own code, there will always remain a risk of bias. This risk can significantly affect the efficiency and thoroughness of the testing process. To minimize bias in testing, I included negative scenarios that aimed to break the system rather than just confirm it worked under normal circumstances. When planning ahead, implementing a continuous integration/continuous deployment (CI/CD) pipeline that includes automated testing will be an important step for any software developer. This strategy will ensure that your tests are consistently executed while maintaining code quality and reliability throughout the development process. Remaining disciplined in my commitment to quality code as a software engineer is required for several reasons. This discipline will impact my code’s overall functionality and reliability in the software I create.

**References**

Hambling, B., & Morgan, P. (2019.). Software testing: An ISTQB-BCS certified tester foundation guide (4th ed.). Retrieved from <https://ebookcentral-proquest-com.ezproxy.snhu.edu/lib/snhu-ebooks/detail.action?docID=5837074>