Hannah Hendrix

10/17/2024

CS-320

Project Two

During developing the contact, task, and appointment services for the mobile application project, I applied a structured unit testing approach for each feature to ensure compliance with the software requirements. The unit tests for the ContactService and TaskService were focused on verifying that all fields like ID, first name, last name, and phone number met specified constraints and rejected invalid inputs. For example, I wrote tests to ensure that IDs longer than 10 characters or phone numbers that did not have exactly 10 digits would result in an IllegalkAregumentException. Similarly, for the TaskService, I verified that tasks could only be created if they met the required constraints for ID, name, and description fields, and tested functionality such as adding, deleting, and updating tasks. In the AppointmentService, unit tests were conducted to verify unique appointment IDs, dates not set in the past, and descriptions that met length requirements using SimpleDateFormat.

My app[roach to unit testing was closely aligned with the software requirements, as seen by particular tests such as testInvalidLength() in the ContactService and testAddTask() in the TaskSewrvice, which represented the limits outlined in the project specification. The JUnit tests’ quality was ensured by attaining high test coverage, which included thoroughly testing each feature for both positive and negative cases. For example, the testDeleteContact() and testUpdateTask() functions ensured that the system appropriately handled invalid or missing data. Writing the JUnit tests was simple, but it required careful attention to edge cases, such as handling null inputs or testing past dates in the AppointmentService. I ensured that my code was technically sound by using unambiguous assertions like assertThrows(IllegalArgumentException.class) to validate exceptions. Efficiency was improved by combining related test cases and developing modular, focused tests, which decreased redundancy and simplified debugging.

The primary testing technique I used in this project was unit testing, which is required to invalidate the behavior of individual components in isolation. (AWS, 2024). Unit tests, such as those written for the Contact, Task, and Appointment services, are quick and efficient at detecting bugs early in the development process. In addition to unit testing, I performed integration testing to ensure that the components worked properly, especially when adding and changing tasks and appointments. However, I did not use regression testing, which would be important in larger projects where constant updates could cause new defects, nor did I undertake performance testing, which evaluates how well the system scales under load.

Unit testing is ideal for any project, especially those that use an agile methodology, because it allows for frequent changes while retaining code stability. Integration testing is required for systems with numerous interdependent modules, whereas regression testing is essential in environments that get regular changes. Performance testing, while not used here, is critical for applications that must manage a large number of users or transactions. (Pittet, 2019).

In terms of mindset, I approached this project with caution, knowing that even little code changes could cause unexpected complications. For example, in the AppointmentService, I ensured that invalid date inputs, like past dates, were properly handled to avoid future issues. I used a test-driven development strategy (TDD) to reduce bias when testing my code. By writing tests first, I avoided subconsciously disregarding implementation issues. If I had tested after writing the code, I might have been inclined to believe it was correct. For example, the testNullTaskName() test ensured that I approached validation honestly, catching any null inputs before implementing the logic.

Maintaining high code quality standards was important throughout this project., I resisted the temptation to cut corners in my tests, knowing that doing so would result in technical debt. By thoroughly testing edge cases and negative scenarios, I ensured that the application was robust and manageable. To avoid technical debt in future projects, I intend to implement continuous testing and refactoring techniques that ensure the code is not only functional but also clean and efficient.

References:

AWS. “What Is Unit Testing? - Unit Testing Explained - AWS.” *Amazon Web Services, Inc.*, 2024, [aws.amazon.com/what-is/unit-testing/](http://aws.amazon.com/what-is/unit-testing/).

Pittet, Sten. “The Different Types of Testing in Software.” *Atlassian*, 2019, www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing.