7-2 Project Two

Jordan Schutte

10/15/2023

For the approach I used for my unit tests, I employed both positive and negative testing to validate input and output. Ensured that contacts were created, read, updated, and deleted correctly and that invalid operations were adequately handled. I also wanted to ensure all contact data (ID, name, etc.) adhered to specified formats and lengths, correlating directly with the requirements stipulated for data integrity and user experience. For my TaskService tests I implemented boundary, exception, and functionality testing to confirm reliable task management (creation, modification, deletion). Task ID, name, and description validations were intensely executed, safeguarding against illegal arguments and ensuring adherence to task data specifications. For my AppointmentService tests I emphasized ensuring that appointments could not overlap, were scheduled appropriately, and that reminder functionalities were operative. I also maintained strict validation of appointment timings and user notifications, aligning with the necessity for reliable scheduling.

I believe the effectiveness of my JUnit tests can be seen from achieving comprehensive coverage by considering varied scenarios, from typical usage to potential misuse, such as invalid inputs and boundary cases. My tests ensuring a task cannot be created with an invalid ID (using assertThrows), ensured system robustness by protecting against erroneous data infiltration. I also ensured null, empty, and boundary condition checks, e.g., validating that the task name adheres to specified length limitations. I also have efficient code which can be seen by leveraging @BeforeEach to initialize shared test components, although further optimization is warranted to consistently utilize this throughout all test methods.

My testing ensured variables operated correctly at their limits aswell as verifying that the system handled exceptions gracefully and predictably. Some techniques that were not employed was performance testing like evaluating the system’s operational performance under emphasized or peak loads was not notably addressed. While unit tests were thorough, testing in tandem operation of multiple units or systems wasn’t explicitly covered.

Since this was my first experience with JUnit testing, I adopted a cautious, detail-oriented mindset was imperative. Recognizing the intricate web of dependencies, for instance, understanding how an invalid task creation could cascade and influence subsequent operations, informed a meticulous testing approach. I also made sure to limit any potential bias to ensure impartiality, especially when developing and testing within the same context, was crucial. A conscious effort was made to test not just for expected and desired outcomes but for potential misuse and unexpected user inputs.

I believe I expressed discipline and commitment to quality through the recognition that quality assurance is not merely a stage but an ongoing, embedded process is crucial. Avoiding “quick fixes” and ensuring that every code push adhered to established standards negated the accrual of technical debt. Avoiding technical debt will involve maintaining this stringent adherence to comprehensive testing and rectifying issues at their root, as opposed to applying surface-level fixes. A sustainable, future-proof development trajectory necessitates this discipline.

Through the application and further reflection on the employed testing techniques, there’s a prominent recognition of the balance required between thorough, robust testing and maintaining development efficiency and momentum. Unit testing, while foundational, forms just a component of a holistic testing strategy. As I move forward, embedding these practices within a wider framework that encompasses integration, system, and acceptance testing, among others, will fortify our software development and maintenance lifecycle, ensuring not just conformance to requirements but enhanced user satisfaction, reliability, and system longevity.