**Project Two**

Beth Campbell

Southern New Hampshire University

CS 320: Software Test Automation& QA

Doctor Ramsey Kraya

08/13/2023

In the Contact Service Test, I thoroughly evaluated contact management. My method encompassed all vital contact operations, spanning creation, reading, updating, and deletion. Test cases aligned with effective contact management requirements. I confirmed accurate addition, reading, updating, and deletion by focusing on these core operations. I checked added contacts for correctness and unique IDs. Updates and deletions were validated in complex scenarios to prevent errors. One of the tests I designed within Contact Service Test focused on verifying the accuracy of contact creation. In this test, I used the addContact method to add a new contact with specific values for the first name, last name, phone and address. I then used assertion to validate that the contact was correctly added to the contactList and its attributes matched the values provided.   
 The Task Service Test thoroughly checked task management. I focused on critical functions—task creation, deletion, and updates—to ensure accuracy. Tasks were created accurately, deleted seamlessly, and updated as intended. This strategy aligned with robust task management requirements. I ensured proper attributes and unique IDs. Additionally, accurate task removal was validated upon deletion. For the Task Service Test, I crafted a test to ensure that task creation adhered to the requirements, including the unique ID constraint. In this test, I created two tasks with the same ID and attempted to add them to Task Service. I then used assertions to verify that only the first task was added successfully, and the second task was not added due to the duplication.

 The Appointment Service Test evaluated the app's ability to schedule, modify, and cancel appointments, aligning with robust scheduling requirements. I concentrated on vital scenarios to ensure proper system operation, confirming accurate scheduling attributes, and subsequent error-free modifications and cancellations. The objective was to bolster scheduling confidence. In the Appointment Service Test, I verified the scheduling mechanism by adding and retrieving appointments via the getAppointment method for validation.

Each of these testing approaches was rooted in the commitment to not just fulfilling software requirements, but also to elevate the quality and reliability of the application. By meticulously designing and implementing these test suites, I aimed to ensure that the contact, task, and appointment services performed seamlessly, without errors, and in alignment with the project's objectives. The quality of my JUnit tests are supported by the comprehensive coverage achieved during testing. I ensured that my tests covered a wide range of scenarios, including edge cases, boundary conditions, and normal use cases. This approach is reflected in the coverage percentage.

I used code coverage tools to analyze test coverage and identify executed code areas. This guided me in enhancing test suite gaps. In the Contact Service Test, I covered all operations, achieving high branch coverage for diverse code paths. In the Task Service Test, scenarios like duplicate IDs and exceeding task limits targeted various code branches. The Appointment Service Test ensured proper scheduling and retrieval, achieving both statement and branch coverage. My effective JUnit tests unveiled bugs during development, aiding in rapid regression checks and maintaining functionality.

Maintaining code quality and adherence to best practices were constant priorities during testing. In the Contact Service Test, I meticulously examined the contact addition method to ensure accuracy. For instance, the line "assertTrue(ContactService.addContact("Alice", "Johnson", "1234567890", "123 Main St"));" validated successful addition. Likewise, in the Task Service Test, I verified proper task creation using "assertTrue(TaskService.addTask(new Task("T123456789", "Sample Task", "Task Description")));". This affirmed adherence to requirements. Efficiency was paramount. In the Appointment Service Test, I validated the efficiency of the scheduling efficiency through careful timing analysis. Take the following code as an example:

“long startTime = System.currentTimeMillis();

AppointmentService.scheduleAppointment("2023-08-15", "09:00 AM");

long endTime = System.currentTimeMillis();

long executionTime = endTime - startTime;

assertTrue(executionTime < 100);”  
This code demonstrates how I measured the execution time of the appointment scheduling process and ensured it met the efficiency requirements. These practices collectively enhanced the robustness and optimization of the codebase, bolstering the reliability of the backend services.

In this project, I employed diverse software testing techniques to ensure the backend services' quality and functionality. One of the techniques I utilized was unit testing, where I examined individual components or functions in isolation. For instance, in the Contact Service Test, I employed unit tests to validate contact creation, updates, and deletions. This approach proved invaluable in catching bugs early in development and swiftly offering feedback on code accuracy.

Integration testing played a vital role analyzing component interactions. Boundary testing was key in both the Task Service and Appointment Service Tests. I tested adding exactly 10 tasks in the Task Service, assessing critical boundaries. This technique validated software behavior at limits, revealing potential boundary-related errors.

While these methods were used, other testing approaches weren't applied. Despite its significance in extensive projects, regression testing—re-running tests after code changes to prevent regressions—was omitted. Performance testing, evaluating system responsiveness and scalability, and security testing, identifying vulnerabilities, were not explicitly employed, although crucial for high-demand situations. In practice, these techniques address distinct aspects: unit testing for component accuracy, integration testing for system collaboration, and boundary testing for robustness. Technique choice hinges on project complexity, requirements, and available testing resources.

I was careful in the project, aware of the impact of minor oversights. In the Task Service Test, I rigorously assessed over 10 tasks, acknowledging their closeness to system limits. This caution came from knowing small errors can ripple and affect reliability. In the Contact Service Test, I checked method interactions for smooth data flow, spotting issues, and validating services. Recognizing code interconnection was vital for testing components and system interactions.

To reduce bias in my code review, I maintained objectivity, emphasizing expected behavior as per requirements. In the Appointment Service Test, I verified precise scheduling based on specified inputs. Testing my own code as a developer could introduce bias due to familiarity and assumptions about its behavior.

Upholding quality is crucial in software engineering. Taking shortcuts in coding or testing invites unforeseen issues and technical debt. Hurriedly written code might patch an issue initially but lead to maintenance challenges. In the Appointment Service Test, I diligently tackled scheduling conflicts, incorporating additional test cases. This method detected problems early, sidestepping expensive fixes. To prevent technical debt, I'll prioritize reviews, thorough testing, and documentation. This dedication ensures enduring software stability and maintainability.