Marcus Moore

CS-320

Project Two

My testing approach for the contact, task and appointment services consisted of using unit tests that correlated with the requirements that were given for each feature of the application. The unit tests that were used were developed by using the Junit framework to make sure that every class and method functioned correctly. For the contact service my tests focused on validating specific restraints like string length limits and I created tests for adding, updating, or deleting contacts, as well as edge cases. When creating tests for the appointment, I made sure to make sure they covered validating the appointment IDs and dates. I also used exception handling for invalid inputs. While developing the task service, my tests checked for the unique ID requirements and constraints for the name and description length of the tasks.

My testing approach was closely aligned with the requirements, this is shown by the tests that are following each requirement for IDs and appointments. Each unit test was made to validate a specific requirement. For example, the AppointmentTest.Java file has a test that ensures that dates that are in the past will be invalid:

**assertThrows(IllegalArgumentException.class, () -> { new Appointment("12345", pastDate, "Valid Description"); });**

I know my Junit tests were effective because there was a test that addressed each requirement, so the code has high coverage of the app’s functionality. I made an effort to make sure that the tests provide regression prevention so that if new code is added it wouldn’t mess with the current functionality. I also included validation for expected success and fail scenarios so the code handles edge cases properly.

My experience with writing JUnit tests has been very interesting and I’ve learned a lot of valuable things. I made sure my code was technically sound by verifying the methods I used were correlated to the requirements of the project and by testing edge cases, like in ContactServiceTest when I created a test so that duplicate IDs cannot be added. Here is an example of that test:

**assertThrows(IllegalArgumentException.class, () -> {**

**service.addContact(new Contact("12345", "Jane", "Doe", "1234567890", "123 Street"));**

**service.addContact(new Contact("12345", "John", "Smith", "9876543210", "456 Avenue"));**

**});**

To be sure that my code was efficient, I designed tests that do not have excessive computations and are focused on the functionality that I am building. This is demonstrated in my TaskServiceTest where the test only updates relevant fields:

**service.updateTask("12345", "Updated Name", null);**

**Task updatedTask = service.getTask("12345");**

**assertEquals("Updated Name", updatedTask.getName());**

**assertEquals("Original Description", updatedTask.getDescription());**

For these services I used multiple type of testing techniques. A technique that was used frequently is unit testing, which focuses on testing individual units by themselves like a single method or class. The practical use for these unit tests is to verify that the code is correct in smaller modules. I also utilized boundary testing to test edges of valid inputs and make sure these inputs are handled correctly. The practical use for these kinds of tests is to prevent failures that can be caused by certain inputs.

Some techniques that were not used in this project, but I hope to use in the future include system testing, performance testing, and integration testing. System testing tests the entire application to make sure that it fits with the overall requirements. Its practical use is for complete validation of an application before releasing it to the public. Performance testing verifies how the application acts in specific conditions like when there are limited resources, and its practical use is to make sure that the application is responsive and scalable. Integration testing focuses on hoe two or more modules work together, the practical use for integration testing is for complex systems with multiple components.

My mindset was very open minded and excited to learn about quality assurance and testing when working on this project. I employed caution very often throughout the testing to be sure that there were minimal mistakes in the tests so that the application would function as intended by the requirements. An example of this was making sure that updating one field in task service did not overwrite any of the other fields on accident which required cautious testing and development. I limited bias in the review by testing the code from a neutral perspective and focusing on the requirements and reviewing it as if someone else wrote it. Being disciplined is always one of the most important qualities to have not just when software engineering but in life in general. Discipline was used during this project so that shortcuts were avoided, and writing was clear with reusable tests. An example of this is using parameterized tests instead of hardcoding the test inputs.

Citations

GeeksforGeeks. (2024, October 22). *Types of software testing*. https://www.geeksforgeeks.org/types-software-testing/

Kinsbruner, E. (n.d.). *The Complete Guide to different types of testing | Perfecto by perforce*. Perfecto.io. https://www.perfecto.io/resources/types-of-testing