Gavin Bliss

CS 320

7-2 Project Two

April 9, 2024

Within the mobile application we created the contact service, task service, and finally the appointment service. Creating the contact service involved validating functionality such as adding, updating, and deleting contacts. The tests were designed specifically to ensure requirements were sufficiently met, such as the required length and uniqueness of an ID. For instance, ‘testAddContact’ and ‘testAddDuplicateContact’ in the ‘ContactServiceTest’ class ensures that contacts cannot contain two of the same IDs.

Task Service is quite similar, yet naturally tailored towards tasks. The focus here is on validating the add, update, and delete operations while making sure that task names, descriptions, and IDs meet the set requirements. The ‘testAddAndDeleteTask’ method is an example of testing both the adding and deleting of functionalities.

In the appointment service, testing is centered around date validations, ensuring descriptions and IDs meet requirements as well. The ‘testAppointmentWithPastDate’ method directly addresses the requirements that appointments cannot be set in the past.

We can see and example of ensuring our alignment to the software requirements in the unit test within the ‘ContactTest’ class, in the method ‘testContactIDTooLong()’:  
***@Test(expected = IllegalArgumentException.class)***

***public void testContactIDTooLong() {***

***new Contact("ID12345678901", "Mike", "Tyson", "1234567890", "123 MSG St");***

***}***

This tests the requirement that a contact ID must not go over the ten-character limit.

JUnit tests were implemented extensively and can be seen in instances such as in the ‘testDuplicateAppointment’ method within the ‘AppointmentServiceTest’.

***@Test(expected = IllegalArgumentException.class)***

***public void testDuplicateAppointment() {***

***service.addAppointment("456", futureDate, "Team Collaboration");***

***service.addAppointment("456", futureDate, "Team Collaboration Part Two");***

***}***

This method specifically tests the application’s ability to handle duplicate appointment entries, which helps us maintain data integrity. This displays depth in testing strategy through a focus on this potential logical vulnerability, and ensures the applications functions as expected, making it more reliable.

Writing such JUnit tests involved considering a wide set of scenarios, which includes both valid cases and boundary conditions. Tests like ‘testAppointmentWithNullDescription()’ were beneficial in ensuring that technical soundness was maintained.

***@Test(expected = IllegalArgumentException.class)***

***public void testAppointmentWithNullDescription() {***

***new Appointment("123", new Date(), null);***

***}***

The test case above demonstrates the need for negative testing and ensures the system is able to handle an invalid input correctly. Putting these tests together demanded an understanding of the requirements of the application.

We were able to ensure code efficiency by minimizing redundancy and applying an emphasis on code reuse, as we can see below.

***@Before***

***public void etup() {***

***service = new TaskService();***

***}***

This approach reduces the use of boilerplate code, which refers to code that is reused in many places with minimal alteration in each of its instances. (Zaveri, 2018) Through avoiding overuse of boilerplate, the tests are simpler to understand and maintain.

A testing technique employed throughout the application was unit testing which allowed us to verify the functionality of individual components of the program in isolation. This is the direct purpose of unit testing, as it aims to evaluate the units, or individual parts of a program so that potential issues can be identified prior to the program being brought together. (GeeksforGeeks, 2023) Here is an example of a unit test implemented:  
***@Test***

***public void testValidAppointment() {***

***Date futureDate = new Date(System.currentTimeMillis() + 100000);***

***Appointment appointment = new Appointment("1234567890", futureDate, "Coffee Meeting");***

***}***

Among testing methods that were not used, integration testing was not used. Integration testing is unlike unit testing in that it involves bringing the components together. This form of testing will verify that the interactions between the now integrated components act as expected. (GeeksforGeeks, 2023) We also did not use stress testing, which helps assess the performance of the application under outlier or unexpected conditions.

Mindset plays a significant role in everything we do in life, and software development and testing are no different. It is always important to adopt the mindset of a user along with the developer and consider every possible issue that may occur. It is easy to become biased as you work on a project and become increasingly familiar with a system as you spend more time developing it. However, we have to consider what those who are entirely unfamiliar with the system or application, and how they may react when presented with each component. It is important not to consider any aspect not worth testing. For example, we will look at a seemingly simple functionality tested in the testValidAppointment(): method.

***@Test***

***public void testValidAppointment() {***

***Date futureDate = new Date(System.currentTimeMillis() + 100000);***

***Appointment appointment = new Appointment("1234567890", futureDate, "Coffee Meeting");***

***}***

Within the project, there is a display of discipline and commitment to quality, which helps maintain the application is reliable and scalable. The code below shows a commitment to reliable software with a comprehensive testing strategy.

***@Test(expected = IllegalArgumentException.class)***

***public void testAppointmentWithNullDescription() {***

***new Appointment("123", new Date(), null);***

***}***

***@Before***

***public void setUp() {***

***service = new TaskService();***

***}***

***@Test***

***public void testValidAppointment() {***

***Date futureDate = new Date(System.currentTimeMillis() + 100000);***

***Appointment appointment = new Appointment("1234567890", futureDate, "Coffee Meeting");***

***}***

# References

GeeksforGeeks. (2023, November 21). *Integration Testing – Software Engineering*. Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/software-engineering-integration-testing/

GeeksforGeeks. (2023, November 16). *Unit Testing – Software Testing*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/unit-testing-software-testing/

Zaveri, M. (2018, January 02). *What is boilerplate and why do we use it? Necessity of coding style guide*. Retrieved from freeCodeCamp: https://www.freecodecamp.org/news/whats-boilerplate-and-why-do-we-use-it-let-s-check-out-the-coding-style-guide-ac2b6c814ee7/