Michael Clerico

CS320 Automation and Testing

Professor Phillips

While developing the mobile application for project one. I adopted a rigorous testing approach for Contact, Task, and Appointment. Having a reliable and robust testing platform is vital for successful deployment of our application. For this I implemented Junit tests to ensure our application will meet the customers requirements. For Contact Service, the unit tests verified that the contact objects met all constraints for valid inputs. We used ContactTest.testInvalidContactId() to ensure that contact IDs and null IDs triggered an IllegalArgumentException. We used in the same argument, ContactTest.testInvalidFirstName() and ContactTest.testInvalidLastName() to ensure that names were not null or met the length requirement. If they did not, we ensured that they returned with an IllegalArgumentException. For task Service we compiled field requirements for yard service tasks. We checked for null and abnormally long task names with ContactTest.testInvalidContactId(). If they did not meet requirements then the object would be rejected. For tasks we also evaluated for updates made within the system with TaskServiceTest.testUpdateTask() to ensure the updated task name and description were being processed properly with updateTask(). Lastly for the Appointment Service I focused on making sure that appointments were scheduled in the future and not the past. AppointmentTest.testInvalidAppointmentDate() would evaluate for invalid past dates. I also evaluated descriptions using AppointmentTest.testInvalidDescription() ensuring descriptions did not exceed 50 characters and null descriptions (Parasoft, n.d.).

I utilized a structured approach when attempting to evaluate in a reliable and methodical fashion. In test methods such as AppointmentTest.testInvalidAppointmentDate(), I used clear descriptive names to communicate the purpose of the test. This ensures debugging and review can be done easily. My Junit tests consisted of edge and negative testing. Examples of negative testing happen during AppointmentTest.testInvalidAppointmentDate() when I cover invalid dates that are one second in the past. I also put the program through edge or boundary testing with TaskServiceTest.testTaskDescriptionBoundry() when I input a task with exactly fifty characters wrong to ensure it is being accepted. I then create a task with fifty-one characters to ensure that an IllegalArgumentException is created with a correct message. I also instituted the same type of test in AppointmentTest.testAppointmentDateBoundary() when I scheduled an appointment one day in the future to see if it was accepted and evaluated if a date scheduled on the current date was rejected. I minimized redundancy by combining related tasks where appropriate. In TaskServiceTest.testUpdateTask() I combined update description and task name in the same assertion. In the end writing these Junit tests made it easier to understand the behavior of the application under certain conditions (TestProject, n.d.).

When designing the Junit Tess I ensured high coverage of all the testing requirements. I looked at critical paths, error handling situations, and edge cases. I utilized all requirements as test cases. These include ID lengths and field formats. In ContactTest.testInvalidContactId() we ensured that IDs exceeding the 10-character limit were rejected. We also evaluate across TaskTest.testInvalidTaskId() and AppointmentTest.testInvalidAppointmentId() to ensure the same requirements are being met across all services. I also covered the requirements with field format validation. Contact names where evaluated to ensure they were non-null and were held to maximum length constraints with ContactTest.testInvalidName(). Phone numbers were also validated for format and length with ContactTest.testInvalidPhoneNumber(). Then lastly appointment descriptions were evaluated to ensure 50 characters or less in AppointmentTest.testInvalidDescription(). Some techniques I did not use are integration testing where we evaluated the interactions between different classes. Because the classes were independent of each other it was not used. System testing was also not used because of no true back-end service. The software does not have a GUI interface so that further negated the need. Then lastly, I left out performance testing because the requirements were focused on implementation and validating functionality rather than efficiency and high volumes. To avoid overlooking edge cases, I used a critical mindset and tried to minimize assumptions when writing tests. I made sure I was managing all requirements by not just seeing if the dates were in the past but also included the current day. Making sure my tests were understandable and clear was especially important. This would allow for future testing if the application were to grow or new requirements are needed to be met (Baeldung, n.d.).

In the end by adopting the use of Junit testing with combined discipline and systematic testing. I was able to evaluate the boundaries of the application and its reliability within its scope. I created a robust testing back-end service that meets the customer and corporations needs (TestProject, n.d.).

**References**

**Parasoft. (n.d.). *JUnit tutorial: Setting up, writing, and running Java unit tests*. Retrieved from https://www.parasoft.com/blog/junit-tutorial-setting-up-writing-and-running-java-unit-tests**

**Baeldung. (n.d.). *Guide to JUnit 5*. Retrieved from https://www.baeldung.com/junit-5**

**TestProject. (n.d.). *JUnit testing – Learn how to test your code in Java*. Retrieved from https://blog.testproject.io/2021/01/11/junit-testing-learn-how-to-test-your-code-in-java/**