Jerod Goode

CS320 – Project 2: Backend Project Review

Professor Gudivada

06/18/2025

Over the course of this term, I have built three core services – ContactService, AppointmentService, and TaskService – that operate as the main logic of my backend project. Additionally, I built Juint tests for each java file to ensure that they operate and meet the performance expectations given to me. In this review, I will examine the core services individually and reflect on the mindset I had for my Junit testing and the code design.

**CONTACTSERVICE**

ContactService handles the contact information. In this package, Contact.java handled storing the first name, last name, id, address, and phone number for a contact, while ContactService.java handled the creation, updating, and deleting of a contact. First thing I did was to make the ID immutable. Originally, I did not have the ID as immutable, but since I didn’t include a setter method for the ID string, I thought it could not be altered. I learned however, since it’s not declared as **final**, the ID still isn’t truly immutable. Declaring the ID string as **private final** is a safer approach that helps enforce data integrity and prevents accidental changes. In the Junit tests on ContactServiceTest.java, to keep code efficient and avoid redundant code, I used **@BeforeEach** to set up a new instance of ContactService before each test, for example:

**class ContactServiceTest {**

**private ContactService contactService;**

**//creates new CustomerService instance before each test  
@BeforeEach**

**void setUp() {**

**contactService = new ContactService();**

**}**

I also ran tests to throw errors if any of the input fields are left null or empty. For example:

**assertThrows(IllegalArgumentException.class () -> {  
 contactService.deleteContact(null);  
});**

So, if the user tried to enter in nothing for the input for first name, last name, address, or phone number, the program would throw an error to the user saying that the input is invalid.

**APPOINTMENT SERVICE**

AppointmentService handles the logic for appointments. Appointment.java stored the information for ID, date, description of appointments, while AppointmentService handled the methods for creating, updating, and deleting appointment information. Like ContactService, the ID was made immutable. The junit conducted tests for the successful addition of a valid appointment, successful deletion of an appointment by ID, successfully updates an appointment, rejection of a duplicate ID, and rejection of an appointment made with a past date. To test for that, I set the test to throw an error if the user inputs a past date:

**@Test  
void testAppointmentDateIsInPast() {**

**Calendar cal = Calendar.getInstance();  
cal.set(1988, Calendar.JUNE, 6);  
Date pastDate = cal.getTime();**

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

**new Appointment("1111", pastDate, "Meeting with Tom.");  
}):**

**}**

**TASK SERVICE**

TaskService handles the tasks associated with the contact and appointment data. Similar to AppointmentService, TaskService also stores the information for an immutable ID, name, and description. The junit tests for TaskService were broken down into validating the input from the user, failing if there is an invalid input. The tests also checked for successful addition of name, ID, and description and if those three were in the parameters of the set limitations in Task.java. Since the unit testing focuses on isolated components, I was able to determine pretty quickly when a function met its requirements and when it failed. This was helpful in troubleshooting particular areas of the code that was not being accessed by the tests.

Some testing techniques that I did not use but should have considered using were Regression testing. Regression testing is a type of software testing technique that re-runs functional and non-functional tests to ensure that a software application works as intended after any code changes, updates, revisions, improvements, or optimizations. The slightest code modification can cause a domino effect that can alter a product’s primary functionalities; thus, regression testing plays a key role in investigating the product’s architecture, which is crucial to determine the root cause of both product success and failure. (Kanade, 2022) These unit tests help catch bugs early on in the software development lifecycle. Unit testing is perfect for early-stage development when software is being built and tested frequently. While unit testing provides a solid foundation to ensure that code operates as it should, incorporating other testing techniques will always strengthen the security of the application. Each technique offers unique benefits, and using a combination of testing methods helps ensure the code is thoroughly validated and reliable.

In conclusion, my primary technique was Junit testing that checked the functionality of program and ensured that it meets the expectations set. I treated each edge case – null inputs, duplicate ID’s, past dates – as a potential fault source and wrote the tests to make sure they covered the potential vulnerabilities. These tests were based on the requirements for each service, and I believe that their coverage was sufficient enough to catch most vulnerabilities. Going forward, I plan to integrate continuous integration coding principles to my work to ensure my test qualities continue to improve.

Sources:

Kanade, V. (2022, October 6). Understanding the role of regression testing - Spiceworks Inc.

Spiceworks Inc. https://www.spiceworks.com/tech/devops/articles/what-is-regression-testing/