Christopher Rossi

Karl Lewis

CS-320

For the mobile application I developed in Project One, I focused on writing unit tests for the contact, task, and appointment services. For the Contact Service, I tested the creation, updating, and deletion of contacts. This included validating that contact information like name, phone number, and email were properly handled. I also checked for edge cases such as invalid formats or empty fields. For the Task Service, my tests focused on adding, updating, and deleting tasks. I made sure tasks were correctly marked as complete and that due dates were not set in the past, and that task priority was handled properly. In the Appointment Service, I wrote tests to make sure that appointments could be created with valid dates and times, rescheduled, and canceled. I also tested edge cases, such as handling appointment conflicts and making sure the app did not let users schedule appointments in the past.

My approach was matched the software requirements. For example, the customer needed the app to handle different contact formats and for tasks to have valid due dates which I made sure to address in my tests. The coverage of the tests was also as needed. This meant that most of the important functions were well tested.

Writing the tests was an important part of the project. For example, when testing task creation, I wrote tests to make sure the system would reject invalid dates and only allow tasks to be created with proper due dates. To make sure the code was good, I checked that tasks could not be updated with a past due date. I also made sure the tests were efficient by using parameterized tests for different task priorities to avoid repeating similar code. This allowed me to test multiple scenarios without writing redundant tests.

For this project, the main testing techniques I used were unit testing and mocking. Unit testing focused on verifying that each individual part of the system worked as expected. Mocking helped me simulate parts of the system that were not implemented or were not needed for the specific tests. For example, in the Appointment Service, I mocked the database calls so I did not have to interact with a real database during the tests which saved time and made the tests more reliable. Other techniques I used were integration testing and system testing. These were used in this project but could be important for larger projects. Integration testing checks if different parts of the system work together while system testing makes sure the entire system works as intended. These techniques would be useful in a situation where more components interact with each other.

Throughout the project, I took a cautious approach since I knew that skipping important tests could lead to bugs. It was important to think about how different pieces of the code worked together because small mistakes could lead to big issues in the end. For example, when I modified the task creation logic, I made sure the changes did not affect other parts of the task functionality, like priority or completion status.

To limit bias, I made sure to my friend to review my tests. Since I wrote the code myself, I could have missed something, and an outside view helped visualize areas that might need more improvement. This was very important to make sure the tests covered everything needed and did not overlook any potential issues.

Being disciplined in writing tests is important to making good code. Skipping tests or rushing through them can lead to problems later on when the code is harder to change or fix. I have learned that it's important to take the time to write thorough tests and avoid cutting corners. In conclusion, I plan to continue following best practices and writing tests for all important scenarios, including edge cases, to make sure my code is reliable.