1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

My testing approach looked for input validation as well as making sure there was as much coverage in the JUnit tests as possible.

* + 1. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

The more coverage my tests have, the more sure I am that my classes are well-tested and secure. Although it is difficult to get 100% coverage, seeing that all of my tests passed without errors or failures and at least 80% coverage makes it clear that the tests were effective.

* 1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

To ensure my code was technically sound, I made sure to have getters and setters for any attributes I planned to alter in later sections of the code. One example is the getTask() method located in TaskService.

* + 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

One way I learned to make my code efficient was to use a map/hashmap (in line 9 of TaskService) instead of an arraylist. The library for HashMap provides methods that make it much easier to find a certain key/element, add to a list, etc.

1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

I performed manual testing as the author of the code, constantly checking each new line of code to make sure it has proper syntax, no errors, and neat formatting. Unit/white box testing was also performed via JUnit tests written out for each class. I also performed functional testing by checking to see my code met all of the system requirements.

* + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.

Some other testing methods are maintenance testing and nonfunctional testing. The former consists of testing code after the system has already been published for use by the public. Since this is not a completed system, maintenance testing was not possible. Nonfunctional testing checks usability, performance, efficiency, security, etc. While factors like security can be tested, there is no “main” method, so our system had no interface to test the performance and usability of.

* + 1. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.

Functional testing is done for any system - it makes sure a system is built to the standards written out beforehand. This can be done in any software development project. White box testing can be done for any system that is built in phases, as new components are added. Nonfunctional testing is done in conjunction with clients and users, the ones who created the business requirements. Maintenance testing, as previously stated, can be done for any system after completion and user testing.

* 1. Mindset
     1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

I employed caution in the sense that I carefully proofread and tested my code after writing every few lines. This was especially important because the three features, contacts, appointments, and tasks have such similar functionalities such as updating contacts and tasks, so it was essential that there were no minor typos or errors that could lead to the whole program falling apart.

* + 1. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

There is absolutely a bias if you are testing your own code, because anyone that reviews their own work is likely to miss mistakes, whether it is intentional or otherwise. I tried to limit bias by following the testing methods outlined in the reading, along with other methods I researched myself. By creating more of a structure to the testing process, it is easier to self-test and eliminate as much bias as possible.

* + 1. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

Discipline is important to remember and maintain when you recognize how much can be changed (for better or worse) with seemingly the smallest of details. When I am a practitioner in the field, I plan to avoid technical debt by keeping a checklist of testing methods, such as functional/nonfunctional testing and every aspect of each one, and following it as closely as possible throughout each project.