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.    
         1. My approach aligned to the software requirements by using Junit5 testing to test the ability to add a single contact/task/appointment or multiple contacts/tasks/appointments with a unique ID, testing that a contact/task/appointment with duplicate ID is handled correctly by returning false, tests that verify that contacts/tasks/appointments can be added, retrieved, or deleted correctly, and tests that the contact/task/appointment can be updated based on what fields each can update, such as first name, last name, phone number, or address.
      2. 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?
         1. I know my Junit tests were effective based on the coverage percentage by testing at least 80 percent of my code. I also made sure to cover all CRUD operations which are create, read, update, and delete which helps have a high coverage. I also made sure to create tests that were logical sound, to include checking for null.
   2. 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.
         1. To ensure my code is technically sound I made well thought out tests to verify all functionality was operating as intended. The code snippet below is an example of this. This tests checks if a single task can be added successfully, validating the creation process and the enforcement of the required fields.

@Test

void testAddTask() {

assertTrue(taskService.addTask(new Task("task1", "Task One", "Description of Task One")));

}

Furthermore, I also made sure that when I deleted an object it did indeed get deleted. For example:

contactService.deleteContact("1");

Assertions.assertNull(contactService.getContact("1"));

This test verified that after the contact is deleted, attempting to retrieve it returns null, so it is confirmed that the delete function works as intended and is technically sound.

* + 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.
       1. I ensured my code was efficient by using HashMap. The use of This minimizes the time it takes to find where to store or retrieve an object and makes it efficient. For example in my test, appointmentService.updateAppointment("1", new Date(), "Updated Description"); I am updating an appointment by searching for it by ID which has an O(1) time complexity, then modifying the relevant fields. This is efficiently handled by HashMap without the need for additional traversal.

1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.
         1. The software testing techniques that I employed in this project are Unit Testing, boundary testing, and error handling. For unit testing I created Junit5 tests to validate each of my methods behaviors under different conditions like adding, updating, or deleting entities. For boundary testing I made sure that names or descriptions had a min or max length and ensured phone numbers had ten digits. And for error handling I made sure to check how my application handled unexpected inputs, for example I included tests that attempted to add duplicate ID’s to check how my application handled that error since duplicate ID’s are not allowed.
      2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
         1. There are several other software testing techniques that I did not use for this project that include performance testing, security testing, and usability testing. Performance testing tests the responsiveness of an application by applying stress and seeing how the application runs. Security testing includes risk assessment and vulnerability scanning using tools like Maven to check for dependencies. And usability testing tests that the user experience is intuitive, and that real people can use the application without issue.
      3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.
         1. Unit testing is ideal for early stages of development to ensure that individual methods function correctly independently of each other. Boundary value testing is useful in applications where input ranges are restricted. Error handling Testing is important for all systems as all systems can have unexpected errors. Performance testing ensures that the software meets the needs and conditions to run successfully under the normal stress load. Security testing ensures that sensitive information is not breached. And usability testing is essential for clients because the customer is going to need to sue the product and it should be easy for them to use.
   2. 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.
         1. Adopting the right mindset when working on this project was important for making sure the software not only met its requirements but also was of high quality. Employing caution can mitigate risks, which reduces the likelihood of bugs. One way I employed caution was in my contactservicetest, I tested an edge case by adding contacts with the maximum field length and attempted to insert duplicate IDs:

Assertions.assertFalse(contactService.addContact(new Contact("1", "John", "Doe", "1234567890", "123 Baker St")));

That line shows caution by ensuring that the application properly handles the duplicate ID’s.

* + 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.
       1. Limiting bias in the testing of code is a challenge especially when I am also the one writing the code I am testing. Bias can lead to overlooked errors and making assumptions about functionality I created. The ways I tried to limit bias is by attempting to cover a wide range of use cases in my testing. I included common and not so common cases. For example, in the contactservicetest class I had the following code:

Assertions.assertFalse(contactService.addContact(new Contact("ID12345678", "Johnathan", "DoeLongName", "0123456789", "123 Elm Street Very Long Address")));

This test checks for maximum lengths in the contact fields which ensures the system enforces limits correctly rather than just testing with shorter names or addresses.

* + 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.
       1. It is important to be disciplined in my commitment to quality as a software engineering professional and not cut corners because users are relying on me as a developer to create safe tested products. High quality code is more reliable and usually results in less failures. This reliability builders user trust which is critical for any software application. Furthermore, well written and thoroughly tested code is easier to maintain. Cutting corners may save me time or costs upfront but I’ll most likely pay for it later. And lastly security culnerablities often come from overlooked flaws and rushed code so by being disciplined and not cutting corners when it comes to software testing I can avoid some of those vulnerabilities.