My unit testing approach for all three features were meant to align with every software requirement. Each requirement for each feature was a testable item and could have one or more unit tests written to check them. Ways that I knew I was aligning with the requirements were writing tests to check if errors were thrown when trying to add null parameters into constructors or setter methods. A specific example of this is within my AppointmentTest class where I have three different unit tests checking to see that an error is thrown when each different parameter is null. In this class I had also checked to see if the description was to long, the appointment date was before today, not a hard coded today but a rolling today, or testing to see if the appointment ID was to long. Having all of these unit tests written made sure that every requirement of that class was tested individually.

I know that every unit test I wrote for all three features was of a very high quality. I know this because each test class has a one hundred percent coverage score on its respective class. This was something I made sure was the case, that way every single line of code was tested for every requirement that was given to each of the three features. This also ensures that my JUnit tests were very effective in testing their respective classes.

I made sure that my code was technically sound by following best practices and standards. I used getter and setter methods, I made sure that variable names were clear in their function, and I made sure that method names were also clear in their function. My code was also error free and ran perfectly, even if constructors and methods were provided with invalid inputs. Some lines that really show this are the full Constructor of the Contact class. This constructor had if statements first checking if each parameter was null or not, and then checking the length of each parameter. This order of checking makes sure that if a parameter was null, the program wouldn’t error out due to checking the length of a null string. I did have this issue once, but once the order was changed, I made sure to carry this mentality through to the rest of the three features.

I know that my code is efficient because it is very modular and there is not code that is repeated multiple times. Another level of efficiency can be shown in my AppointmentService class. This class uses some if statements in the addAppointment and removeAppointment methods to check if an appointment does exist or does not exist before following through with a line of code to add or remove an Appointment. This helps make sure that in the addAppointment case we are not overriding an appointment ID that is already in the system and in the removeAppointment case make sure that trying to remove an appointment that does not exist causing some odd, possible unintended behavior.

The only testing technique that I used in this project with all three features was the unit testing technique. This testing technique checks small pieces of code as it is added into a larger project, so that way each part of the code can be checked for requirements and errors. This method is the only one that made sense for this project given the feature requirements and overall simplicity of the project. Some types of testing techniques that were not used in this project are integration testing, performance testing, security testing, or compatibility testing. All of these methods would be used on my complex projects that are being released to the public or to a client. Integration testing is something that would be done if there are many different small parts of a large project being added together. This would be done to make sure each smaller part integrates well with the whole, hence the name. Performance testing is done to check the performance of a project. This can test the ability of the program to put up with large loads of data or use, check how it runs over a long period of time, or checks to see what happens if it is overloaded to the point of breaking down. Security testing is done to make sure the program is secure. This is done to find potential unauthorized points of entry, or ways to hack into the data of the program itself and mess with intended behavior. Lastly compatibility testing is done to see how the program would run on different hardware or within different software.

A practical use of integration testing might come from game developers putting together a game. If one team was working on world creation, one team on the character creation and another on physics, an integration test can be ran to make sure each of these different parts work together. If each was tested without the other and worked great, what might happen when they are all thrown in the same project running at the same time. A practical use of performance testing could be testing a website. Testers would check what happens when a large amount of traffic comes into the site, or what happens when it runs for a longer period of time under normal traffic. This could unearth some unintended behaviors and allow for these issues to be fixed before production. Security testing is done on any bit of software that goes out into the world. This is done to prevent hackers from getting into the program and data from being stolen. This might be the most important non-functional testing because of the personal data that could be at stake. Lastly, compatibility testing can be done to check how a website responds on different web browsers or on slower machines. This allows the software to be used on many different devices, if that is how the client or developers want this to be used.

The mindset that I adapted was one of testing each requirement of each feature. In this project I did not see a need for caution, I just needed to test each requirement independently. One time where I was able to appreciate the complexity and interrelationships in the code was when I was modifying the task names or descriptions in the TaskService class. This made me think about where I needed to be testing the requirements of the project to make sure everything was perfect. I don’t feel like I considered how much I was biased on my own code. I was just writing the code and tests to meet the requirements of each feature, and that was it. I could imagine bias being a concern for larger projects with more complex requirements. This might cloud the testing process if you had it in your mind that the way you are doing this was correct and that was it. It is very important to stay disciplined and just follow project requirements. This ensures that no corners are cut and that the code is quality. By testing early and often, to writing tests as I am writing features, I can avoid technical debt. This helps bugs be caught as they are happening and prevents them from compounding exponentially as the development process moves along.