While working on the mobile application project for Grand Stand Systems, I focused on using Junit tests to test the program's back-end services. These tests were used to ensure that the program checked for the correct user input and that inputs would be tested and adopted appropriately throughout the program. This static testing technique is more useful the more early on when it is applied in the code. It is able to test aspects without having to deploy the entire project. By testing these smaller items individually and ensuring they are working correctly, focusing on other aspects of the project down the road can take more priority. Since the Junit tests can stay active in the code while the rest of the project is being completed, there is less concern about focusing on those specific requirements since the Junit tests can be run at any time to confirm they are still working appropriately.

I ensured the Junit tests aligned with the software requirements by ensuring I covered all bases I could think of to validate input. I checked that each test case accounted for possible edge cases. Throughout the project, all input was tested and validated to ensure that the input was within the requirements. While working through the project I made sure to keep in mind how all the different parts would work together to form the larger program. I kept bias to a minimum and did not assume that my code was correct and implemented regular checks while I coded through. In the Contact program, all first and last names had to be less than or equal to ten characters. Addresses needed to be no more than thirty characters. Phone numbers had to be exactly ten characters, which threw me for a loop as I was originally adding the country code into the phone numbers I was testing, which resulted in errors when I ran the Junit Tests. After some very vigorous math (counting to ten), I was able to determine where I had made my mistake. That error caused me to be more diligent and thoughtful and my inputs for test cases. I needed to make sure that the test cases would be accurate and represent the project requirements. Aside from validating the length of strings entered, I also tested to ensure that all contact information could be updated, as well as tested that the delete, duplicate, and nonexistent contact elements would respond appropriately.

Moving to the Task part of the project, it was very similar to the Contact aspect. Ensuring that task names did not exceed twenty characters, description length did not exceed fifty characters, and the ID length needed to be ten characters or lower. Just as with the Contact codes, I also tested that tasks could be added, updated, and deleted as well as making sure non-existent tasks did not end up causing any issues. I found that The Contact and Taks aspects of the project were very similar in what items were tested and set up.

The Appointment class and test were a bit more difficult than the other because we needed to link in the appointments with the current date and time to ensure that any created appointments would not be created in the past and tested those aspects appropriately with the Junit tests. Aside from that all layout and tests were very similar to the Contact and Task code.

I believe that my testing approach aligned very well with the software requirements. By ensuring that appropriate and thorough testing of different possibilities was included. All tests and outlaying cases thoroughly passed the Junit tests. Using the “asserts throw” method ensured that validation would throw errors for any invalid inputs.

It was very important throughout this project to ensure that I stayed focused on implementing testing for any outlying cases that could be replicated in the real world. By making sure I implemented as many edge cases as possible, I hoped to limit any additional errors that could arise as the program continues to move through the process when it eventually receives an user interface where people will be able to directly input strings into the program. Once the program receives an interface, system testing will be used to validate and ensure the program responds and works appropriately. It is important to ensure that the program is fully vetted before it can be implemented into a device.

Maintaining a focused mind to pay attention and fully understand the code is important to a successful project. Not assuming everything is correct and running regular checks is vital to a healthy programming mindset. Ideally, all code would be peer-reviewed, and I would receive feedback on things that could be improved on to increase efficiency. Everything can always be improved, and it is important that as a programmer you are able to receive criticism and grow from it. It’s important to be able to recognize your own bias as you are working on a project. You must be able to remain objective and think thoroughly throughout the process. You want to make sure you are following best coding practices so you can produce high-quality code for your clients.

Implement Junit testing and not having an interface has been an enjoyable experience. It has been a nice change. Typically, when running code, I would validate inputs manually by having an interface or simple menu and running the code as an application. This process can get tiring and unreliable when you are running through so many possible test cases. As humans we make mistakes and sometimes it is more efficient and reliable to hard code the tests in to be done by a machine.