Waitless Testing & Inspection Summary

Maaz Ahmed, Zaid Patel, Bryan Gutierrez, Ammar Idrees

Our project is an application designed for waiters within a restaurant setting, to help optimize and change the dynamic of a restaurant setting. The app helps waiters utilize which tables are currently in use, gives customers the ability to order, displays urgent tasks, and even showcases different statistics within the restaurant. The project itself was completed utilizing Java, and Java Fx for the GUI aspect of the app. The current prototype assumes that the restaurant has 6 available tables, and displays to the waiter the most optimal route to serve the customers. Furthermore, comparing our project to the description report we had the same goal which was to increase the overall efficiency, and production within the restaurant. Within our testing, we opted to utilize JUnit tests as the entire project ran with Java.

In regards to testing the StatsScreen class, we had opted to test if the correct revenue was generated, wait time was correct, and different statistics regarding the availability of tables were updating. Overall, for the environmental needs utilizing Maven was an efficient tool to help with testing. The stats screen had depencies on the menuDisplay and the tableDisplay packages, as it utilized data from both. Furthermore, the tests had passed that we had checked for, as different assert statements properly passed. In regards to inspecting the stats screen, we checked if there were any time constraints and if the class would work on a larger scale. In the inspection we realized that the time and functionality was the same with more tables, and the tables properly update based on the change of states with the tables. These tests overall give us a greater understanding regarding the code, and convey that though there may be a few bugs the overall functionality is working.

As for testing Menu Screen, the most important part of the testing involved ensuring that an item was added or removed to a table's order. The menuDisplay included all the items in the restaurant's order, implemented as buttons, a textView box that was used to enter the tables number, so that it would be entered into the correct table, a delete option, a list of items ordered, and a "View Orders" screen. We tested adding an item to multiple tables at a time and checking that all the items were being added to the multiple tables. The delete option used the textView for the table number and another textView for writing the item to be deleted. These actions would be then displayed in the list("Item added to table#" or "Item cancelled from table#"). All of these changes were finally reflected in 6 different lists in the "View Orders" screen. One list for each table, ofcourse. We made two major observations for improving the Menu Screen. The first was to simply improve the graphical user interface of the screen to make it look more appealing. The second improvement was involved in the code. Instead of making many if statements to check for each and every button of the menu, we could have put all this in a loop in order to facilitate adding items to the menu if the restaurant ever wanted. Overall, the menu did its job correctly.

For testing the table display, the tests given revolve more around if the basic gui features work. The first test is about displaying the six tables once logging in. The next test was if the user were to press the table, the application should be able to indicate if the table is occupied or not. Red would indicate it is occupied while white would indicate it is not. The third, fourth, and

fifth tests ensure that the application can correctly open the menu, task list, and stats screen respectively. The fifth test is to ensure the program properly displays the optimal route for a waiter to know what tables are highest priority. This test is particularly important because the main theme of the project is the traveling knapsack. In this particular program, we implement this theme by allowing the application to generate a route that takes into consideration each table order's cost, wait time, and quantity. The test requires the user to input two table orders and select the update button, and the test will pass if both tables appear. To ensure the algorithm works, it can be run multiple times with different orders and the user will notice the order won't always be the same each time.

The main criteria that was assessed when inspecting the table display was the same as the other portions of code: the code should limit bugs, be readable, and be organized. For the display specifically, the code should be clear to any developer that this particular feature would be the main screen of the application and is the part of the application that would be used the most, so this feature needed maximum reliability. The code for the most part effectively completes each requirement for the inspection.

In regards to testing the Tasks List, the main thing we wanted to test was to ensure the basic functionality was working and being updated correctly on the GUI. The first test case was making sure that the simple task of adding a task to the task list was being done correctly. To ensure this, we used JUnit to make sure that the table id and the string user entered were matching correctly on what the variables were storing. After this, the next test case focused on deleting an item from the task list. We added multiple tasks to the tasks list and then picked one we wanted to remove. The user could select which tasks they wanted to remove. For the testing, we picked one and then made sure that the task selected was deleted and wasn't appearing on the GUI. Lastly, we tested the update button on the task list. This was testing two things at once. First, it's making sure that the timer is running correctly and then also that the update button works and displays the reminder on the tasks list page. This involved using table display code as well as we to first make the tables occupied.

When inspecting the tasks List code, the first thing we all noticed was that it had good readability. One critique we had though was that there weren't that many comments. Using more comments would be helpful in explaining what each function or confusing lines are intending to do. We did notice a minor bug which was noted in the open issue sections of the report. Overall, the code was well organized and was effective in getting the task intended done.

As for open issues, there were only a couple that we weren't able to fully debug before the final demonstration. For one, we found an error with the task list that it wouldn't open again after being closed once. We couldn't determine what exactly was causing this issue but as a workaround we implemented multithreading so the tasks list can be running at all times. Furthermore, we found a small bug with how the menu gui was set up because it would only allow you to order 28 items before being full. This can be easily fixed by clearing the orders after a table has placed their order.

For future iterations of the application, one of the things that we would hope to achieve is to implement some sort of database. This would allow the application to more effectively manage tasks, orders, and would allow the app to maintain multiple employees to use the application at once. This would be made possible through using something like Microsoft SQL server or teradata. Furthermore, another feature that could be implemented would be to create

some sort of client that customers can use to input their orders rather than the waiters inputting them manually.