# SELF ASSESSMENT

| CRITERIA | PERCENTAGE | REASON |
|---|---|---|
| Add a passenger to the train queue<br>Weight<br>15.00% | 70.00 to 100.00 % | 90%, The data saved in CW-01 is loaded into CW-02 without any issues. Once done, each seat number is checked whether they have arrived, following this checking they are added in the waiting room. The waiting room passenger details are then displayed. Further, the 6-sided die that generates the number of passengers to be added into the queue is implemented as well. No need of entering any sort of passenger data, simply entering of A will add the passengers onto the queue in ascending order based on seat number. The queue is assumed to be of size 42 so the error message for full will not be shown until all 42 passengers have joined, however, the code has been implemented. Finally, the queue in the GUI is arranged accordingly |
| View the train queue<br>Weight<br>10.00% | 70.00 to 100.00 % | 90%, The train queue GUI displayed shows the passenger name beside the slot, if the passenger hasn't joined the queue it labels that slot as empty, furthermore, once joined the border is colored red. Switching between the Train and queue GUI is as simple as clicking on the button. The Train GUI has the name beside each seat number if that passenger has arrived into the waiting room, if not is labelled empty, also is bordered red when the passenger boards the train. The waiting room GUI is updated accordingly as well. |

| | | |
|---|---|---|
| Delete passenger from the train queue<br>Weight<br>15.00% | 70.00 to 100.00 % | 90%, Passenger details is requested: the seat number, NIC and name (since 1 name can book multiple seats), if valid (found in train queue), the details are outputted and they are removed from the train queue. Furthermore, the queue is re-ordered correctly. As the queue moves forward from the point at which the customer was deleted |
| Store train queue data<br>Weight<br>10.00% | 70.00 to 100.00 % | 90%, All the required information to be stored to keep the GUI updated the next time load is called, are saved without any issues. Furthermore, NoSQL was used |
| Load train queue data<br>Weight<br>10.00% | 70.00 to 100.00 % | 90%, All the required information is loaded back without any issues, and the GUI returns back to how it was before program was stopped. |
| Run the simulation and produce report<br>Weight<br>30.00% | 70.00 to 100.00 % | 90%, Simulation has been implemented. 3 6-sided dice have been used to generate the processing time for each passenger, whom are removed from the queue and added into the train. Pop-up GUI contains all the details of the passenger who is to be added, along with the max length, max, min and avg times of all passengers in the queue. Furthermore, the GUI which is opened contains details of all the passengers who have boarded the train. Finally, the details of the passengers are saved in a text file, who have boarded the train. |

| | | |
|---|---|---|
| Code quality, demonstration and self-assessment form<br>Weight<br>10.00% | 70.00 to 100.00 % | 80%, Naming conventions have been followed, variable names have meaning; methods are verbs and the class is a noun. Indentation is maintained throughout the code. Comments are present throughout the code explaining what each method does. |

## ASSUMPTIONS

- Took the max size of the queue as 42.
- Names of passengers are between 5 and 15; otherwise the label layouts start getting out of alignment.
- By the sentence “Have visualized the train queue with 42 slots and passenger names against the seat if the passenger arrived.” I implemented the train queue which initially has 42 slots and decreases upon each passenger being added into the train/ deleted. And by the section it says, “passenger names against seat,” I did so for the Train view itself, as in the 42 seats in the train itself has the name against the seat of the train if the passenger has arrived into the waiting room. Passenger names and their corresponding seat number/s are displayed on the train queue as well.