Project Memo

To: Professor Bistriceanu

From: Amer Alsabbagh, Alrick Sayasavanh

Date: 22 November 2017

Re: “Thalia Tix” – Ticketing Software for Theatres

**Purpose**

We would like to propose our ticket-managing application known as Thalia Tix. The purpose of this software is to provide theatres a tool that will assist in managing functionalities that are essential to them. We will cover each specification within the application and provide a clear understand on how the application functions.

**Process**

The development of the application composed of both of us, Amer and Alrick. We planned days accordingly to work on the assignment using pair programming. Before given the API to base the project off, we first met up to brainstorm ideas on how we wanted to go about the application. We illustrated how the theatre might be generated and how we would go about creating each entity within the theatre (such as sections, rows, and seats). Once we had a game plan for creating the show, we started to paired programming process and had a skeleton for the project. Once the API had been released, we started to meet up on set days to work on the project. We, as a team, benefit from pair programming because we were able to catch out mistakes a lot more frequently, learn new techniques from the other peer, and keep a steady workflow throughout the design of the application. We had varying ideas on how we wanted to go about the algorithms when designing certain aspects, which is good, and later decided to choose the ones that would be in the best interest of design.

**Challenges**

There were many obstacles that we overcame throughout the course of the assignment. Before we get into the problems we had when developing the application, we’ll discuss the challenges in pair programming. The main concept of pair programming is to be working on one computer only. That means we had to find times that worked well with both of our schedules, which can be a bit difficult. This is due to one of us being a commuter and had to travel the distance to meet up to work. It was a great opportunity to utilize pair programming, which helped us both learn new things from each other. As for the application, one of the challenges we came across was the implementation of the donation system. Having to donate a ticket and request a ticket was a bit of an issue because we initially did not have a well enough grasp on how we should do it. We could have either made the system operate so that patrons would only be given one ticket per donation to prevent a single user from taking all the tickets from a given show. The other option we had was to utilize a first-come-first-serve basis in which users who requested a donation with a certain amount of tickets would get that many before the next patron. We also had the issue of requesting the proper seating when given the count and starting seat ID. We originally had the seat request function check the row in which the starting seat was requested, then needed to figure out how to handle the request when it was invalid. We had to understand three scenarios: If the seat is already taken, If the row does not have enough contiguous seats, and if there are not enough seats in the last row. These scenarios go hand-in-hand because we had to make sure that the seat is available, then if the row that the starting seat was requested does not have enough seats, the search mechanism must go through the remaining rows to find a fit. The way we solved this issue was to iterate through each row, check each seats’ availability, and keep a tally mark of how many seats are available in a row. If a seat is found to be taken, then the tally gets reset and the process continues afterwards. Once all the seats in the section have been traversed without the correct amount of seats, then it will fail the search.

After we developed the ticketing software, we stumbled upon many challenges within Maven. Due to our lack of knowledge of both Maven and Dropwizard, we took quite the time to understand how to make it function properly. We figured out how to configure the pom.xml file, utilize the configuration.yml POJO, and create a Dropwizard service to handle the project in a local server through the sources provided in the course.

**Statistics and Results**

Over the course of the development of the application, we came across the following statistics:

* Lines of code: 7500
* Lines of code in unit tests: 1849
* Unit test coverage using Jacoco: 24%/72%
* We have accumulated the total number of instructions run through the application and have two percentages to present. 24% of the entire application is covered through tests, but this is due to the overwhelming amount of adapters and API implementations involved. Given a total number of 19,220 instructions, 10678 being from adapters and the API, that leaves us with a total of 8542 instructions within the application. With a total of 14,588 instructions untested and subtracting the 10678 from the adapters and the API, we are left with 3910 untested instructions due to functions such as getters, setters, and hash codes that are used to complete the object. This turns out to show that we essentially have 72% coverage from unit tests.
* Number of hours to get code working: ~ 110 hours
* Number of hours to get prepare submission: ~ 5 hours
* Cyclomatic Complexity: 5
* Link to Repository: <https://github.com/amersa64/Theatre-REST-App.git> (NOTE: This repository was merged from a previous repository due switching to Maven)
* Status: Complete – Meets all requirements for the project

References

[https://www.agilealliance.org/glossary/pairing/#q=~(filters~(postType~(~'page~'post~'aa\_book~'aa\_event\_session~'aa\_experience\_report~'aa\_glossary~'aa\_research\_paper~'aa\_video)~tags~(~'pair\*20programming))~searchTerm~'~sort~false~sortDirection~'asc~page~1)](https://www.agilealliance.org/glossary/pairing/#q=~(filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'pair*20programming))~searchTerm~'~sort~false~sortDirection~'asc~page~1))

<http://wiki.c2.com/?PairProgrammingBenefits>

<https://www.versionone.com/agile-101/agile-software-programming-best-practices/pair-programming/>