**Report**

**Design Decisions**

The primary actors in the system include a customer, an operator, and a station manager. The requirements specified the customer to be able to use a ticket machine to generate a ticket and use that ticket to enter and exit a station via a ticket gate. The station manager was required to manage stations and ticket machines via CRUD operations, as well as incorporate a specify a pricing schedule for the system. The operator would be required to set up a ticket machine during install by inserting the ticket machine’s serial number (UUID) to request its configuration.

The use cases produced were derived from these requirements and provide full coverage of uses for each actor.

The robustness diagram sets out the objects interacting to deliver the functioning system for each of the 3 actors. Each actor has a set of Java Server Pages (JSPs) that allow them to interface to the system. The diagram outlines these JSPs, the actions they allow the actors to perform and the background model that performs these actions.

**Implementation**

The program full implements all functionality as specified in the use cases. The station control can create and update stations and create ticket machines for those stations. He can also set a pricing schedule. The operator can then set up a ticket machine using its serial number (UUID) and the ticket machine will automatically download its configuration at regular intervals. At this stage, a ticket machine is configured. A customer can then use a ticket machine to create a ticket. He will then enter a departure station using the entry gate and exit a destination station using the exit gate.

For this implementation, the customer can specify the departure station and destination station at the ticket gates. This was included as a method of testing failure cases of tickets being used at incorrect station gates. In the real system, a gate would be hard coded to a station. The customer can also enter the current time at the ticket machine, entry gate or exit gate. This was also included to test failure cases of a ticket being out-of-date. In the real system, the time would be set by specific machine clock, or managed by a time server.

Any user input is validated before the model attempts to perform any methods, to ensure correct values have been entered.

I added an additional attribute to the ticket – the number of zones the ticket allows to be travelled. This ensures customers are traveling to stations within the zone limit paid for. A customer can travel to any station using their ticket, provided they do not exceed this zone limit.

Each ticket is bound by a time frame of 24 hours, enforced by the ticket gates, which use the ticket issue date attribute to verify if the ticket is in-date.

Other gate error checking includes verification that the ticket is issued by a valid authority. When a ticket is created, it is assigned an encrypted hash. The ticket gates decrypt this using the public key and assess if the ticket has been altered.

**Test Strategy Rationale**

The manual test plan is written to thoroughly demonstrate system functionality. Each test case is derived from its associated use case and includes the action and expected result to ensure correct functionality. The tests are written to demonstrate correct function only, from the perspective of a user – failure cases are beyond the scope of this test plan due to the nature of the test event timescale being limited.

The test plan contains a set of pre-requisite conditions that must be met as a minimum before proceeding with tests. This allows for the test cases to be executed in any order as desired. Individual tests may also have specific pre-conditions for that test to be met before starting.

**Project Evaluation**

The program meets the specified use cases and design fully as intended.

Unfortunately, I did not incorporate the logging of ticket issues or payments into this assignment. In the real system, this would be useful data to obtain for sales and analytics and could be used to allow station managers to optimise the system.

Another area of improvement I would have focussed on with more time is to add more rigorous error checking to the pricingScheduleManager JSP in order to catch any invalid inputs by a Station Manager and throw an error.