Underground Train Ticket System Assignment

Radu Lefter

I have created my Ticketing System based on the model that was provided for us at the course GitHub address <https://github.com/dtriboan/solent2Public>. I have created use case and robustness diagrams that reflect the class diagrams already provided in the UMLmodel file.

The project has two web applications, one for the API, named project-web, and another one for the client, called project-web-client. The API allows the user to perform crud operations for the stations and the machines. The client application lets the user to purchase tickets by selecting the departure and destination stations.

I focused on adding, updating, and deleting ticket machines to every station. This can be made in two ways: by selecting every station individually or by accessing the API endpoint.

When doing it through the jsp interface, the user has more control over his actions, as he can add, update, and delete each ticket machine in turn. This is easily done by setting a station for the ticket machine and then saving the ticket machine on the ticketMAchineDAO.

When creating the machines through the API, they can only be created or deleted in bulk on all the station and in a random number by accessing the <http://localhost:8080/projectfacadeweb/rest/stationService/generateTicketMachineForEachStation> endpoint and the other endpoint that are created in the TicketMachineRestService class. The user retains the possibility to modify the machine created this way through the jsp interface. The API also allows the user to access a ticket machine by its uuid or by its station, in the latter case returning the first machine if there are several. The details about the machine thus returned can then be used by the interface in the client web project.

The price should be calculated using the PriceCalculatorDAO and the Rate, PriceBand and PricingDetails classes in the model. However, I encountered problems implementing this part, due to my unfamiliarity with the JPA API. I managed to create the xml file containing the pricing details and implemented the code to extract the code into the DAO, but I wasn’t able to save it. Consequently, I decided to hard code the pricing details in the getTicketMachineConfig endpoint and then calculate the price on the client side.

The project-web-client simulates the steps a user would take to buy and validate a ticket. In the SelectStations jsp the user can select the starting and the destination stations. These values along with the zones into which the stations are found are passed as parameters through a form to the generateTicket jsp that will make an API call to the endpoints in the TicketMachineRestService file in the project-web. Based on the data retrieved from that endpoint, namely the pricing details for the ticket machines, a price will e calculated and displayed on the screen. The price depends on the time the ticket was purchased at and if it falls in a PEAK or OFFPEAK band from one of three price bands hard coded into the PricingDetails class.

The ticket created this way will be encoded in the AssymetricCryptography class and displayed in a field on the page in xml form. This code can be copied and pasted in the openGate jsp that simulated a ticket gate. Validating the code in this page will display Open Gate only if the code was decoded correctly.

There is much to improve on the project. Although I managed to implement the crud for stations and ticket machines, it would have been much better if every ticket machine would have had its own schedule, but that was made very difficult by the complexity of the PricingDetailsDAO. However, sending data from the API to the client accomplished one of the most important parts of the system, as manipulating data cannot take place without having access to that data in the first place.

In order to handle the data returned by the api endpoints I used the gson library, which i installed in the pom.xml file of the project-web. I also used the log4j library to log information to throughout the project.

My test plan focused on the main crud operations that a station manager has to be able to do and on the most likely scenario that a ticket buyer will encounter when using the system.

Test plan

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Expected | Actual | Action |
| Project-client-rest |  |  |  |
| RestClientDAOServiceFacadeTest | That a station can be saved, deleted and retrieved from the DAO | As expected | Perform crud operation son a station in project-web |
| ConfigurationPollerTest and RestClientServiceFacadeTest | That a ticket machine be configurated | As expected | Perform crud operations on a ticket machine in project-web |
| Project-dao-jpa |  |  |  |
| StationDAOJpaImplTest | That a station machine can be saved and retrieved from the database | As expected | Perform crud operation son a station in project-web |
| TicketMachineDAOJpaImplTest | That a ticket machine can be saved and retrieved from the database | As expected | Perform crud operations on a ticket machine in project-web |
| Project-model |  |  |  |
| TicketEncoderImplTest, TicketMachineConfigReplyMessageJaxbTest, TicketModelJaxbTest | That a ticket class can be transcribed into xml with JAXB | As expected | Select start and destination station from the project-web-client and an xml ticket will be generated |
| Project-service |  |  |  |
| ServiceFacadeImplTest | Sets up all the configuration classes so that the DAO can be accessed | As expected | Starting the system in order to perform any action |
|  |  |  |  |