# 

# Solent University

# in Partnership with QA (QAHE)

Object Oriented Design and Development (AE2)

QHO543

**Academic Year:** 2021-2022

**Module Leader :** Tendai Mhlanga

**Hand in date :** 17th June 2022 before 4 PM (local UK time)

Daniel Grabinski

University ID: 15078949



Table of Content

[**Introduction** 4](#_Toc106312152)

[**Database & Rest server** 4](#_Toc106312153)

[**Central Controller Service** 5](#_Toc106312154)

[**Ticket Machine** 6](#_Toc106312155)

[**Gate service** 8](#_Toc106312156)

[Entry Gate 8](#_Toc106312157)

[Exit Gate 10](#_Toc106312158)

[**Use cases diagrams** 11](#_Toc106312159)

[**Class diagram** 14](#_Toc106312161)

[**Data Flow Diagram** 14](#_Toc106312162)

[**Model** 15](#_Toc106312163)

[0-level DFD 15](#_Toc106312164)

[1-level DFD 16](#_Toc106312165)

[2- Level DFD 16](#_Toc106312166)

[**Testing (Junit)** 17](#_Toc106312167)

[Entry Gate Testing 18](#_Toc106312168)

[Exit gate testing 19](#_Toc106312169)

[**Javadoc** 20](#_Toc106312170)

[**Reflection** 22](#_Toc106312171)

# **Introduction**

The application includes two projects.“RestServerUnderground” (backend – Glassfish Server) includes Rest Web services and connection to the database. The “ClientUnderground” (frontend – Tomcat server) includes retrieving data front the database, controllers, classes, methods and web pages for the application (JSPs). For this Project I used Apache Netbeans IDE (v.12.6), Apache Tomcat(v. 10.0.16 running on PORT 8081) and Glassfish Server (5.1.0, running on PORT 8080). Customers can buy the encoded ticket using a ticket machine system. The tickets are validated at station gates to allow entrance or exit from the station. The machine will display the information for the customers and calculate the price of the ticket bases on the number of zones travelled. As a confirmation customer will have to make a payment for the ticket using a button. After simulating the payment, the ticket will be displayed for the user in XML format including all details that will be used to enter and exit the station. The entry gate will be open if the ticket validation is successful and display the ticket to exit the station. At the exit gate, the user will be prompted to choose the station he would like to exit at, if he decided to exit at earlier stations. The customer will paste his entry ticket. After successful validation, the gate will open and display the name of the station he exited at.

# **Database & Rest server**

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedFor the Rest Service I used Glassfish server connected to the database on the localhost (PORT 8080). I created connection Pool in and resource in Glassfish. After successful connection I created Restful Web services that allows the connection between the database and Rest Server.

A screenshot of a computer

Description automatically generated with medium confidenceGraphical user interface, text, application

Description automatically generatedIn the database I created 3 tables where I store information about the Stations, Schedule and Prices. On the picture below we can notice relationship between the Stations tables and Schedules where “stationid” is a primary key for tables Stations. Stationid is created automatically (autoincrement) when we populate the table with new stations. The table includes 12 stations (2 stations for each zone.

Figure 1 Database Design SQL Figure 2 stations table

# **Central Controller Service**

We merely obtain the stations from the rest server, and then we add them to the list in Rest Client providing the Web Target path from the localhost. The manager provides XML ticket with name of stations, zones, from when ticket is valid and when the ticket expires.

Text

Description automatically generated with medium confidence

Figure 3 RestClient.java

Subsequently, in JSP, we will create a simple loop to retrieve the stations in the dropdown menu. Station managers who could control the stations are known as controllers.

Text

Description automatically generated Figure 4 Stations in XML format

# **Ticket Machine**

Customers can generate encoded tickets using a ticket system, and these tickets are then validated at "gates" to allow entry or exit from the station. When a passenger creates a ticket, they will choose their information from the following fields on a TickenMachineGenerator.jsp page.

* **Departure Station** – refers to the zone from which ticket will be valid
* **End Station** – refers to the zone where the ticket is valid to.
* **From Date** – the date is generated automatically as a current date
* **To Date** – the date when the ticket expires

Graphical user interface, text, application

Description automatically generated

Figure 5 TickeMachineGenerator.jsp

Once a user will choose the information from the following fields , the system will generate for the customer information about the ticket he would like to buy , and the price for the ticket. The button “Make a payment” simulates the user will pay for the ticket. After successful payment, the machine will generate the ticket in XML format for the user that will be used to enter the gate on the entry station.

Graphical user interface, text, application, email

Description automatically generated

Figure 6 Generating Ticket process

Graphical user interface, text, application

Description automatically generated

Figure 7 Ticket.java class

# **Gate service**

An EntryGate.jsp file is given for users to use to imitate the entry gate reader. It has a field that accepts ticket data that can be pasted into it. On a system that is currently operational, the JSP can be accessed at <http://localhost:8081/ClientUnderground/EntryGate.jsp>.

## Entry Gate

It is possible to mimic putting the ticket through the reader by pasting the generated Ticket XML into the text field and then pressing the "Enter the station" button. The user will choose the station he would like to enter from. When the button is clicked, the gate will open if the following conditions are met:

* The zones listed on the encoded ticket are the same as or in the same range as the station use will enter the gate from
* The ticket is not expired ( new date is before valid to date)
* Ticket has not been used before to enter the station
* The format of the XML ticket is correct

Graphical user interface, text, application

Description automatically generated

Figure 8 EntryGate.jsp page

On the picture below the user can choose the entry station from and including zone 1 to zone 2. If the conditions to perform validation of the ticket were successful the gate will open. If not the message will be displayed “Gate not opened”).

Text, email

Description automatically generated

Figure 9 Successful gate validation for the ticket

The gate will not open if the user will try to enter the station at zone greater than 2 ( zone valid 1-2)

Graphical user interface, text, application, email

Description automatically generated

Figure 10 Unsuccessful gate validation

## Exit Gate

I applied the same logic for the exitGate.jsp. User will paste his new generated ticket with new Boolean value “ EnteredStation = true “ into the text field on the exit gate page. The user will choose the exit station he would like to exit at. When the button “Exit from the station” is clicked, the gate will open if the following conditions are met:

* The zones listed on the encoded ticket are the same as or in the same range as the station use will exit the gate from
* The ticket is not expired ( new date is before valid to date)
* Ticket has not been used before to exit the station
* The format of the XML ticket is correct

On the example below the customer can exit the station between zone 1 and zone 2. In this scenario the customer will exit the station at “Canada Water” station and the gate will open, because the ticket validation at the gate was successful.

Graphical user interface, text, application, email

Description automatically generated

Figure 11 Exit Gate.jsp page

If user choose the station which is not valid for the purchased ticket (zone greater then zone 2), the gate will not open.

**Graphical user interface, text, application, email

Description automatically generated**

Figure 12 Unsuccessful validation at the exit gate

# **Use cases diagrams**

The use case and model diagram, both of which are contained in the system, include all the classes in the system. This helps illustrate the links between the classes.

Diagram

Description automatically generated

Figure 13 Underground Ticket system Use case

|  |  |
| --- | --- |
| Use case | underground ticket system |
| Actors | User |
| Pre-Condition | The server must be running |
| Post-Condition | The user entered the station in correct zone, The user exited the station in the correct zone |
| Scenario | **Serial No. Step** |
| Users | 1. Create Ticket |
|  | 1. Make payment |
|  | 1. Copy the ticket to the entry gate |
|  | 1. Choose entry station |
|  | 1. Open gate and copy the ticket |
|  | 1. Paste the ticket to exit gate |
|  | 1. Exit station |

# Diagram Description automatically generated

Figure 14 Creating Ticket use case

|  |  |
| --- | --- |
| Use case | Creating ticket machine |
| Actors | users |
| Pre-Condition | The ticket machine should be running |
| Post-Condition | The user has entered all the required information |
| Scenario | **Serial No. Step** |
| Users | 1. Select start station, zone |
|  | 1. Select end station, zone |
|  | 1. Choose the expiry of your ticket |
|  | 1. Click on the button buy a ticket |
|  | 1. Make the payment |
|  | 1. Your ticket in the XML form |

Diagram

Description automatically generated

Figure 15 Entry gate Use case

|  |  |
| --- | --- |
| Use case | EntryGate |
| Actors | Users |
| Pre-Condition | The user should have a ticket |
| Post-Condition | The ticket should be validated |
| Scenario | **Serial No. Step** |
| users | 1. Select entry station |
|  | 1. Paste your ticket |
|  | 1. Click on Enter the station for ticket validation |
|  | 1. Gate is opened |
|  | 1. Paste the ticket again |
|  | 1. Got to Exit gate |

# **Diagram Description automatically generatedClass diagram**

# **Data Flow Diagram**

Diagram

Description automatically generated

Figure 17 Data Flow Diagram

# **Model**

A model is a documentation of the fundamental data types and interfaces that need to be implemented for each component. Within this project, we have these models shown below in the diagram, and all the attributes are confidential due to the nature of the data being both public and private.

Diagram

Description automatically generated

Figure 16 Model documentation

## 0-level DFD

The controller will handle the stations and set the rates according to peak and off-peak time, so Controllers are the managers who deal with the stations and all the stations has the Ticket machines

Diagram

Description automatically generated

## 1-level DFD

It is up to the controllers to oversee stations and determine peak and off-peak prices, hence Controllers are station managers, and all stations have Ticket machines to process the tickets. Ticket information is validated from the database and provides a response, if the ticket validation is successful then user can enter the gate.

Diagram

Description automatically generated

## 2- Level DFD

It is the responsibility of the controllers to monitor the stations and all station information is stored in station DB. They establish the pricing for the peak and off-peak periods; hence, controllers serve in the capacity of station managers, and each station is equipped with ticket machines and all the information of the ticket is stored in ticket DB to handle the processing of tickets. The information on the tickets is checked against the database and provides a response to validate the ticket; if the ticket validation is successful, then you will be allowed to enter and exit through the gate of your selected station.

Diagram

Description automatically generated

# **Testing (Junit)**

I created various number of automated tests using framework Junit. By creating test cases for controller and models I ensure that the program logic works as expected. The Junit package contains different classes and interfaces like @Before, @After, @Assert , which I demonstrate during my testing.

In test packages we can find 2 folders for models and controllers. In the controller I perform testing on TicketCreator.java where I test If the ticket can be generated, and marshalled to the XML or Un-marshalled into the Ticket class. I compare departure station values from two tickets if they are equal.

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

## Entry Gate Testing

In the test package “Models” I perform testing on Gate.java class. In the first testing, I am checking If the gate will open , if the entry zone are in the correct range.

Text

Description automatically generated

In the next test case , I am checking I the customer can Entry gate twice using the same ticket. If the Boolean value “EnteredStation = true” the gate will remain closed. I am using asserTrue from Junit to check if the ticket validation was unsuccessful.

Graphical user interface, text

Description automatically generated

The next testing, checks if the gate will be closed , if the user choose enter station in the wrong zone range.

A picture containing application

Description automatically generated

In another tasting I am checking if the ticket expired and user will try to enter gate with invalid ticket. To simulate invalid date I set “valid\_to” date before the new Date .

Graphical user interface, text, application

Description automatically generated

## Exit gate testing

The next three testing I perform for the exited method from the class Gate.java. I applied the same logic as for the entry gate. Because the exited method returns Boolean value not a valid ticket , I assigned the new Boolean value from ticket validation into a variable , and compare if the result is true or false depending on the testing case.

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

The all test cases were performed successfully.

Graphical user interface, text, application

Description automatically generated

# **Javadoc**

For all classes and the methods for the program I created Javadoc documentation. Below I attach two screenshots. More documents can be found in the root folder of my project under (“Javadoc- screenshots folder).

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

# **Reflection**

I applied most of the functionality in the program. I was not able to fully retrieve the station's schedule and Marshall it within a ticket in XML format. I declare prices for the zones within a TicketGeneratorMachine.JSP and use a multidimensional array. I added the functionality which allowed the customer to enter and exit the station within the valid zone range for the ticket, not only the station the ticket was bought from. The code does not match the design where I retrieve the schedule and prices from the database. I was not able to achieve that because I couldn't find the right way to Marshall the price and schedule within a ticket. I tried unsuccessfully to create UI to modify the central schedules.