SAI Final Assignment: Travel Agency

Table of Contents

[1. Introduction 1](#_Toc26800535)

[2. Start-up Code 5](#_Toc26800536)

[3. Screen Shots 5](#_Toc26800537)

[4. Assignment 6](#_Toc26800538)

[4. Grading Criteria 7](#_Toc26800539)

[5. Submission and Deadlines 8](#_Toc26800540)

# 1. Introduction

In this assignment you will integrate a system of several applications for booking airplane tickets for business trips of employees. This system is used to gather booking offers from several travel agencies when employees travel abroad for their work. One example scenario for the Business Trip Booking system is shown in Figure 1.

The **Booking Client** application sends a *ClientBookingRequest* which has the following data: (1) the name of the origin international airport, (2) the name of the destination international airport, (3) the number of employees who will travel, (4) date and (5) client id. I the client is null, then the client is not a “registered client”. If the client id is not null, that this is a “registered client”.

First, if the client is registered (i.e., *ClientBookingRequest* clientID is not null), the **Administration** service is called to get the data of the client based on client id. The service will return one of the following client/account types:

* STANDARD: the client does not get any discount.
* SILVER: the client gets 10% discount,
* GOLD: the client gets 20% discount, or
* PREMIUM: the client gets 30% discount

The *ClientBookingRequest* is then forwarded to the **Travel Agency** applications as an *AgencyRequest*. The *AgencyRequest* contains the same airport names from the original *ClientBookingRequest*. Field isRegisteredClient in *AgencyRequest* is set to true if and only if both clientID *in ClientBookingRequest* and account type (from Administration service) are not null. Otherwise, isRegisteredClient is false.

Each *AgencyRequest* is forwarded to the three travel agencies applications according to following rules (Figure 2 shows flow example for unregistered client):

|  |  |
| --- | --- |
| travel agency application | processes AgencyRequest |
| Cheap Tickets | Number of travellers is higher than 2. |
| Business Tickets | Only if the client is registered, i.e., in *ClientBookingRequest* clientID is not null. |
| Easy Tickets | Always (every AgencyRequest). |

Each travel agency application sends back *AgencyReply* containing the name of the agency and the offered price for the requested ticked per person/employee. For each *ClientBookingRequest*, the Booking Client application finally receives a *ClientBookingReply,* which contains the offer of the agency which offered the lowest total price. The total price of the offer is calculated as (agency\_reply\_price \* nr\_passangers) and, if the client is registered, the appropriate discount is applied: (agency\_reply\_price \* nr\_passangers) \* (1 – discount/100).



Figure 1. The Travel Agency integration system – registered client



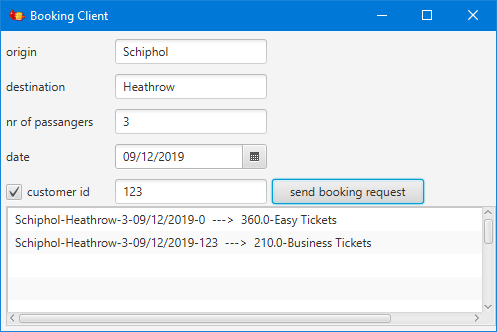
Figure 2. The Travel Agency integration system – unregistered client

# 2. Start-up Code

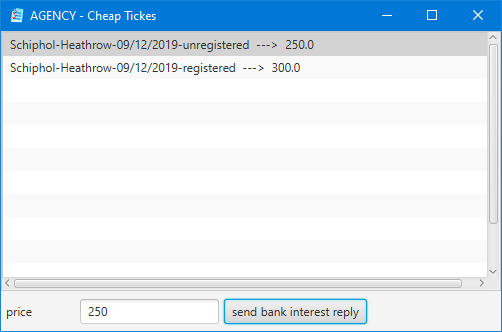
1. **“sai-travel-agency”** is a GRADLE project with **booking-client** and **travel-agency** sub-projects modules. Note that the whole project is a GRADLE project, and **booking-client** and **travel-agency** are GRADLE sub-projects (see settings.gradle in the project root)[[1]](#footnote-1). Note that the GUI is created in JavaFX, but, if you want, you may re-create the GUI in Java Swing (see the next section which shows the GUI).
2. **“administration-service”** contains:
   * administration.war file of a RESTful service which you should deploy on a web Server (e.g., Apache Tomcat). The service can be accessed at <http://localhost:8080/administration/>, where you can read a description about how this service works.
   * administrationModel.jar contains domain classes which can be used to execute method(s) of the service.

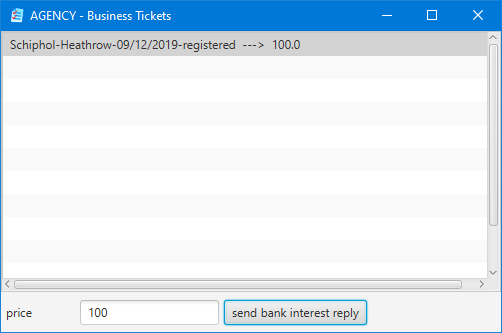
# 3. Screen Shots

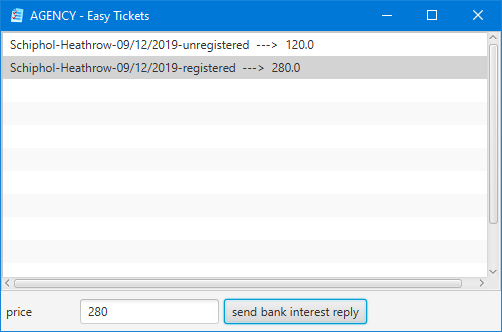
**Booking Client application:**



**Travel Agency applications:**







# 4. Assignment

Implement described integration system as described in this document. You should make use of the following integration patterns:

* Message Broker
* Correlation Identifier (for asynchronous request-reply communication with JMS),
* Return Address (for asynchronous request-reply communication with JMS)
* Messaging Gateway
* Chained Gateways
* Content-Based Router,
* Content Enricher,
* Recipient List,
* Aggregator, and
* Scatter-Gather.

# 4. Grading Criteria

This assignment is INDIVIDUAL, i.e., it is not allowed to work in groups with other students. The grade you get for this assignment is between 1 and 10, and this will be your grade for the Software Applications Integration (SAI) course.

IntelliJ project(s) including full source code and all necessary libraries (gradle, maven or .jar) must be submitted. All submitted projects must compile and run correctly on the computer of the teacher. If the teacher does not have your full source code or cannot run your project(s) due to compiling errors, missing libraries or exceptions, then your SAI grade will be 1. Otherwise, SAI grades will be determined based on implemented Application Integration Patterns in the following way:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | SAI grade | | | | |
|  | 6 | 7 | 8 | 9 | 10 |
| The system works correctly with one travel agency application | x | x | x | x | x |
| Message Broker | x | x | x | x | x |
| Correlation Identifier | x | x | x | x | x |
| Return Address | x | x | x | x | x |
| Messaging Gateway | x | x | x | x | x |
| Chained Gateways | x | x | x | x | x |
| Content-Based Router |  | x | x | x | x |
| Content Enricher |  | x | x | x | x |
| The system works correctly with three travel agency applications |  |  | x | x | x |
| Recipient List |  |  | x | x | x |
| Aggregator |  |  | x | x | x |
| Scatter-Gather |  |  | x | x | x |
| Use of Jeval (or similar) instead of  hard-coded travel agency routing rules. |  |  |  | x | x |
| Well organized code, with comments, proper variable and method names, no redundant code. |  |  |  |  | x |

# 

# 5. Submission and Deadlines

**Submission of the source code**

The IntelliJ project(s) with full source code and all necessary libraries (via gradle, maven or \*.jar) must be submitted via Canvas. The deadline for submission is set in this Canvas assignment. It is not possible to submit after this deadline. If you do not submit your source code before the deadline, you will not receive a SAI grade in this block (i.e., you will not pass the SAI course in this block).

Only students who submitted their source code via Canvas before the deadline specified in Canvas will be invited for this exam. You will receive this invitation with your specific time slot from you teacher several days before the exam. In this invitation it will be specified at which time you should be present for this exam. Each student will have his/her own time slot, and you should and can be present only during your own time slot.

**Defense of your assignment**

In week 8 or 9 SAI exam is scheduled (see class schedules). During this exam you will speak in person to the teacher about your assignment: you will be asked to explain your code, suggest ideas for improvement, etc. If you are not present during this exam, then you will not get a grade for SAI. It is not possible to mote your exam at another time.

1. If you do not want to work with GRADLE, create a new IntelliJ project (without gradle) and copy-paste the source code of all sub-projects. [↑](#footnote-ref-1)