Dragon rides booking backend

1. Submitting your exercise

You would need to reply to the email where you received this with the source code of the maven project with all the files inside.

Do not upload your solution to any public repository.

2. The exercise

The world of game of thrones has evolved and now all the warriors can fly from Winterfell to King's landing or to any other part of the realm in Dragons.

Several companies offer their dragons and dragon landing ports through the world.

We want to create a backend application that recreates this.

On a high level detail this app should be able to:

- Save/update users
- Save dragon flights
- Save bookings
- Search origin/destination dragon flights with and without pagination
- Manage flight offers

Here you can find the detail of the project:

a. Design a REST API using Maven and Java that contains all the needed operations.
b. Using only a JAX-RS library for the development of web services.
c. For the data that we want to store in a database, it can be used any relational or non relational schema where you feel more comfortable. In this database we want to store users, dragon flights and bookings.
d. A method to create/update/get/delete users has to be implemented
e. Also is mandatory to have methods to save bookings, search dragon flights and manage new offers.
f. As a source of dragon flights, it will be provided an API that returns a list of dragon flights (see section 3).
g. During the deploying process of the web application (it would be worthier to use JBoss as application server), it has to load in the database all the dragon flights in their corresponding table.
h. It is recommended to take special care to the architecture and the design patterns during the implementation of this exercise.

Entities that are necessary (at least) in this application:

- User
- Corporate User
- Dragon Flight (for each one way trip)
- Booking

Then, a short description about each transaction of the RESTful layer.

Save / update user

To define a user entity with the fields of your choice (name, address, gender...) and to develop their corresponding transaction. This user entity, has to be persisted in the database. Besides of this, it has to be defined another kind of user to manage the bookings that the corporate people can make. It is possible to store this kind of user in the same table but is necessary to identify them in some way, and store the role that these people have inside the company.

Save flights

During the startup of the application services, we have to retrieve the dragon flights (see section 3) and store them in database. This entity can be made with the fields that the developer consider necessary. Taking to account that we need al least information about flight origin and destination, dates of departure and arrival, price and a flag to show if that trip is an offer or if it has a regular fare. At the beginning every flight has a regular fare.

Save bookings

Transaction to store the relation between users and flights after the purchase process.

Search origin/destination dragon flights with and without pagination

In the mobile app, once we retrieve a list of dragon flights with an origin and destination given, we have the chance to sort this list by price (service response has flights with different currencies) or trip duration. This sort type will be passed as a parameter of this method. Besides, we will have another parameter to indicate if we want the entire list or we want just a number of results in order to do some kind of pagination.

Manage flight offers

Periodically, we want to launch offers for some destinations. In order to develop this functionality, it will be necessary to schedule a robot that retrieves these destinations from another web service (see section 3.3) and updates the price of all the flights with this destination applying the percentage value retrieved from the service in their corresponding table and activate the flag which indicates this trip is an offer.

3. Web services

3.1. Get available flights service

In the following URL you will find a rest service.

http://odigeo-testbackend.herokuapp.com

This service will return the list of flights availability.

Keep in mind it is a mock service, the data inside the flights could not make sense.

It is an non parameter call to the URL of the service. The reply will be a list of flights, defining the flight as following:

a. Segments (inBound, outBound)

1. Airline: String
2. AirlineImage: String
3. ArrivalDate: String mm/dd/yyyy
4. ArrivalTime: String24H hh:mm
5. Origin: String
6. Destination: String
7. DepartureDate: String mm/dd/yyyy
8. DepartureTime: String24H hh:mm

b. Price

1. Price: Double
2. Currency: String

3.2 Currency converter service

It will receive the currency code (USD, EUR, GBP, JPY) to convert from, and the exchange rate of the other currencies. Notice that we don't warranty to have all currency conversions. These exchange rate will change once a day.

http://jarvisai.herokuapp.com/currency?from=xxx&to=yyy

where xxx and yyy is one of the codes above.

3.3 Deals service

It will be a GET operation which retrieves a list of discounts. This is the URL:

http://odigeo-testbackend.herokuapp.com/discount

a. A pair of destination-discount like this

1. Destination: String
2. Discount: Number

4. Appendix

4.1. Flight Availability Example

Request: http://odigeo-testbackend.herokuapp.com


```
{
        "inbound":{
                "airline":"Balerion",
                "airlineImage":"http://dragonimages.net/images/gallery/dragon-images-by-unkn
                own-246.jpg",
                "arrivalDate":"7/20/2065",
                "arrivalTime":"11:35",
                "departureDate":"6/10/2047",
                "departureTime":"6:29",
                "destination":"Vaes Dothrak",
                "origin":"Lys"
        },
        "outbound":{
                "airline":"Vermithrax",
                "airlineImage":"http://dragonimages.net/images/gallery/dragon-images-by-unkn
                own-236.jpg",
                "arrivalDate":"1/27/2018",
                "arrivalTime":"19:55",
                "departureDate":"6/3/2070",
                "departureTime":"14:55",
                "destination":"Lys",
                "origin":"Vaes Dothrak"
        },
        "price":4438.92,
        "currency":"JPY"
}
```


4.2. Currency Service Example

Request: http://jarvisai.herokuapp.com/currency?from=USD&to=EUR


```
{
"jarvis":{
        "from":"USD",
        "to":"EUR",
        "rate":0.907311
        }
}
```

4.3. Deals Service Example

Request: http://odigeo-testbackend.herokuapp.com/discount


```
{
        "city":"Chroyane",
        "discount":10
}
```