Homework 3 **Kim Hammar**
ID2208 **Mallu**
Programming Web-Services Due Date: 15 February 2017

Problem Statement

This report covers the work done in an assignment on programming WebServices. The given task was to implement flight-ticket reservation services as a RESTful web-service.

Main problems and solutions

- *Designing the RESTful API* - An important parameter for your RESTful service is which resources you have and how they're accessed.
- *Implenting RESTful service with the jersey libarary in Java [1]* - A java library for developing RESTful services.

Designing the RESTful API

When designing the API we have followed the general design principles of REST to promote interoperability with other services and ease of use for clients.

The application is made available through a API with logical separation into *resources* which can be manioulated by the regular HTTP methods like GET, POST, PUT, DELETE etc. The media-type of the HTTP-responses are based on the accept-header in the HTTP-request, an alternative would have been to have the media-type explicit in the URI e.g: "/resource.xml". When a error occur the API utilizes the HTTP status-codes.

There are different opinions on the details of the API, we have chosen to structure our resources as following. The base-resource is a verb, e.g "`/tickets`", individual specific resources are accessed through a nested URI as follows: "`/tickets/1`". To keep the base resource simple, functions like filtering, sorting, searching etc are doing through query-parameters on top of the base URI, e.g: "`/rest/itineraries?departmentCity=Stockholm&destinationCity=Mumbai‘`". Further best practices that are out of scope of this homework is to add versioning to the API resources as well as HATEOAS principles to the HTTP-responses (essentially provide links and metadata such that a client can consume the API through exploring and using the responses of some resource to find another resource).

The resources available to for the flight-ticket reservation service are:

- `/login`: Resource for logging in to the flight reservation service using username and password in order to generate access token. Client can login through HTTP-POST request. All resources listed below require the secret token in-order to enable access
- `/itineraries`: Resource for itineraries. Client can retrieve and search itineraries through HTTP-GET request with query parameters

- `/pricelist`: Resource for pricelist. Client can retrieve pricelist with through HTTP-GET request.
- `/flights`: Resource for flights. Client can add flights with HTTP-PUT, delete flights with HTTP-DELETE or retrieve flights with HTTP-GET.
- `/tickets`: Resource for tickets. Client can add tickets HTTP-PUT, delete tickets with HTTP-DELETE or retrieve tickets with HTTP-GET
- `/bookticket`: Resource for booking tickets. Client can book ticket with HTTP-POST providing list of tickets to book as well as creditcard-number.
- `/purchasedtickets`: Resource for purchased tickets. Client can issue its purchased tickets with HTTP-GET by providing receiptid in query parameter.

Implementing RESTful service in Java

Steps Followed:

- Define resource class as plain java object.
- Added annotations to all the resources we wanted to build. E.g: Resource class.

```
@Path("/flights")
public class Flights {
```

URI-endpoint of resource

```
  @PUT
  @Consumes({MediaType.APPLICATION_JSON,
     MediaType.APPLICATION_XML})
  @Produces({MediaType.APPLICATION_JSON,
     MediaType.APPLICATION_XML})
  public ArrayList<Flight> putFlights(@QueryParam("token")
     String token, FlightsPutRequest flightsPutRequest){
```

- Create client side class and using web resource class access the resources using uri. E.g:

```
  Itinerary itinerary = webResource.path("/itineraries/1")
  .queryParam("token", SECRET_TOKEN)
  .accept(MediaType.APPLICATION_JSON)
  .get(Itinerary.class);
```

- Start server first
- Run client side main program to fetch resources through defined http methods.

Conclusions

We can say building restful web services is quite simple and interesting as we treat everything as resources also the technology is quite flexible, as resources can be added or removed or

modified. As everything is a resource in restful world it is easy to split complicated system into small resources which are build and then accessed.

Attachments

Documented source code with server and test-client can be found in the attached zipfile. See README.MD in the root directory for instructions how to execute and build the program, as well as instructions how to test the service.

References

[1] Oracle. Jersey - restful web services in java. `https://jersey.java.net/`, 2017. [Online; accessed 13-Feb-2017].