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 the flight-ticket reservation business 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.
- *Implementing RESTful service with the jersey libarary in Java [1]* - A java library for developing RESTful services was used for development.

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.

Resources

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", but this approach was not taken for this assignment.

Status codes

HTTP-status codes are an essential part of RESTful API's, when you return a response to the client you not only return the data that's necessary, you also return a HTTP status code that indicates the outcome of the request. Even if a request does not require to send any data back to the client it should return a status code indicating if everything went ok (code 200), if there were some error (500) or some other indication. The HTTP-status code is very important because it tells the client alot of useful information, for instance if there was an error with the request, depending on what status code the server return the client can know if it should retry the request at a later time, if the request was not properly formulated, if the resource was moved etc. Imagine the inconvenience for clients if API-designers did'nt use the HTTP status codes but rather every API used they're own error-codes and standards.

Our API

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 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 with 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. A receipt is returned.
- `/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 use the jersey client library to test the API and access the resources, e.g:

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

Conclusions

REST is a more minimalistic solution to implementing a webservice, it has its own guidelines and best-practics as compared to SOAP/WSDL based web-services. The big pro of REST is probably that it's so simple, you just publish your resources as resources identified with HTTP URI's and let client access it through regular HTTP methods. As compared to the typical SOAP based web-service where you have alot of choices to make about encoding/decoding, the communication structure, underlying transport protocol, what types to use etc.

However a con of REST is that for interoperability between services to be possible it relies alot on pre-established assumptions between the requestor and provider in how the service should be used, this is partly implied by the REST standards but still there are some uncertainty. In SOAP/WSDL webservice all information necessary for invokation is explicitly stated in machine-readable format and published in WSDL files. Also SOAP/WSDL provide webservices that have alot more flexibility than REST have, but that also means increased complexity.

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].