RESTful Hypermedia Services in Java

Part 1. CRUD applications

Server side

In this part of the tutorial, you are going to create a CRUD REST application using the tools provided by NetBeans.

- Create a Web application.
- Add to your application a RESTful Web Service from Database
    - Select one of the tables you used last lab (Sakila database), the table actor can be a good start.
    - It will automatically create the Entity class of your table.
    - Open the file *<Table>*FacadeREST.java. In this file you can find the defined methods for GET, POST, PUT and DELETE.
    - In AbstractFacade.java you can find the actual methods for insert, delete, modify and find.
- To test your application:
    - If you are using your own computer:
        - Right click over the application, select the Test RESTful Web Services.
        - In this module you can try the different methods through the console provided by this tool.
        - Send a request to GET("application/xml"). In the results retrieved, copy all elements corresponding to one register in the DB.
        - Send a POST(application/xml) and in the field Content, paste the code you copied in previous step and modify the data.
        - If everything went right, then your record has to appear next time you execute a GET.
    - Otherwise
        - Deploy your application in the server ats.
        - Send a GET request to you application writing in a console the following:

          curl -i -H "Content-type: application/xml" http://ats.cs.ut.ee:8080/application/resources/entity.actor

          note: GET method can also be tested in the browser

        - In the results retrieved, copy all elements corresponding to one record in the DB.
        - Send a POST(application/xml) and after the –d, paste the code you copied in previous step and modify some of the parameters.

          curl -i -H "Content-type: application/xml" -d "
          <actor><actorId>195</actorId><firstName>Silvester</firstN

ame><lastName>Stallone</lastName><lastUpdate>2011-02-15T04:34:33+02:00</lastUpdate></actor>"
http://ats.cs.ut.ee:8080/application/resources/entity.actor

- If everything went right, then your record has to appear next time you execute a GET.

- As you could observe the status response was a "204 No Content" in the POST request. To add the status codes you have to return a Response object in the method of the FacadeRest.java file. In the method create:
  - Drop the @Override tag of the method.
  - Change the name of your method if necessary.
  - Add the following code after you executed the super.create(entity) command.

    return Response.status(Response.Status.OK).build()

    note: be careful selecting the correct import for the Response object (import javax.ws.rs.core.Response;).

  - As you know you can add the location of the resource in the response. Add the following lines to your code:

    UriBuilder builder = uriInfo.getRequestUriBuilder();
    URI uri = builder.path(entity.getActorId().toString()).build(null);

    return Response.status(Response.Status.OK).location(uri).build();

  - If you send a POST request then you will find both elements the hyperlink and the URL.
  - In some cases, the entity object is not updated (wrong id is displayed in the URL returned in the Response). To fix this issue, you have to add the following line in AbstractFacade.java:

    getEntityManager().flush();

  - Complete the codes missing for each of the methods.

Client side

Download the package: client.zip. In this example we are using the library httpClient (http://hc.apache.org/httpclient-3.x/), XStream (http://xstream.codehaus.org/) and joda-time (http://joda-time.sourceforge.net/index.html). HttpClient is used to send the requests to the server. XStream is a library for transforming objects into XML and vice versa. Finally, joda-time is used for the management of dates in the serialization. In total, there are 6 jar files in the package.

Analyze of the code and try it with your application. If you are using another table different than actor, then it is important you create the right class.

Part 2. Hypermedia REST applications

In this second part we will show an alternative way for adding hyperlinks in your XML files. There exist some other libraries that you can check out and that offer some tools for adding such hyperlinks (e.g., http://restfulie.caelum.com.br/).

In this package: hyperlinks.zip, we defined one class that can add the hyperlinks to an existing object. In this example, we are using JAXB instead of XStream to let you try both approaches. Look carefully all the @Xml annotations in all classes. You can add as many links as you need. You will need to adapt the object you receive in the client side to be able to handle the hyperlinks added.