Télécom Saint-Étienne
Web Services
TP n°4 - 3h00

Syed Gillani

Remarks

– You need to form groups of two students (binôme).
– A binôme should write a report containing the answers of questions and exercises listed in this document and send them to syed.gillani@univ-st-etienne.fr.
– DEADLINE: 25/10/2016

In TD3, we learned how to develop a client for the twitter RESTful API. In this TD we will develop a simple RESTful service.

Objectives

– Introduction to **Jersey**: Java for RESTful services.
– Develop a RESTful service that answers GET requests.

Tools Needed

– Eclipse EE (Enterprise Edition) Luna, Kepler etc.
– Glassfish
– REST-client. Postman for Chrome or RESTClient for Firefox.

Jersey

Jersey RESTful Web Services framework is an open source, production quality, framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS Reference Implementation (We'll use the Jersey Jar files for this purpose).

JAX-RS: Java API for RESTful Web Services (JAX-RS) is a Java programming language API that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern.

1 Create a Web Service Application In Eclipse

1. Go to `File` ►`New` ►`Other` ►`Web` ►`Dynamic Web Project`.
2. Call the project `TD4`.
3. Call the package `com.yourlastName.RESTfulService`.
4. Add an HTML page called `index.html`. Add something in the body of this page (e.g. <h1> Hello </h1>)
5. Add a new `Glassfish` to Eclipse. If you do not know how to do this then check TD2.
6. Create a new class called `CurrencyConverterApplication`. This class should extend (heritage) the following class `javax.ws.rs.core.Application`. Do not forget to make the necessary imports.

7. Above the class declaration, add the following annotation:
   `@ApplicationPath("v1/converterApp")`

The class will look like:

```
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Application;

@ApplicationPath("v1/converterApp")
public class CurrencyConverterApplication extends Application{
}
```

The annotation `@ApplicationPath("v1/converterApp")` is very important. It marks the main link (entrance) to your web service. From now on, resource URI (i.e. link) in your web service should be preceded by v1/converterApp.
Note that your project now includes two subprojects:

1. A very small web-site containing your index.html
2. A web service server.

To test **the web site**, run the project on the server. You will get the index.html page displayed (In case you're unable to deploy your project from Eclipse, then export the WAR file from your project, and then manually deploy it on glassfish server using the browser).

2 RESTful Currency Converter

1. Create a new package called `com.yourlastName.model`. Usually, you put your own classes in this package.
2. Create a copy of the class `Currency` created in TD2 to this package.
3. Add an attribute `int ID` to the class. Do not forget to update the constructor to create setters and getters to this attribute.
4. In the other package `com.yourlastName.RESTfulService`, create a new class and call it `CurrencyConverter`.
5. Add the following annotation before the class declaration `@Path("currencyConverter")`. Now all the **resources** in this class will be preceded by `currencyConverter`. If you do not understand, we will have an example soon.
6. Add a String to this class. The string name is `version` (global variable). It indicates the current version of your API. version = "1.0";
7. Add the following method:

```
/*------------------------Code Block Number 1-----------------------*/
        @GET
        @Path("version")
        public String version()
        {
                return "The current version is " +version;

        }
```

Here is an explanation of the annotations:

– `@GET`: specifies the method `version()` is invoked when the rest-client calls an *"HTTP GET"* on the resource URI.

– `@Path` specifies the resource of this method.

After adding this method, do not forget to re-run the project on the GlassFish server. Now let us test the results. Run the following link in your web browser:
http://localhost:8080/TD4/v1/converterApp/currencyConverter/version/

You should get an output indicating the version message.

2.1 Using The RestClient Tester

1. In Firefox go to `Outil ►RESTClient` (you can also use Postman RESTful client for Chrome).
2. Put the method to be `GET` copy the URL above in the URL field and click on send.
3. Check the results and go to the different available tabs i.e. `Response Headers, Response Body (Raw), Response Body (highlights)`.

Questions:

Question 1: Retrieve Data About a Resource (a currency)

version() method did not take arguments. This excercise shows how to write a Java method with an argument.

Our conversion agency converts three currencies (Dollar, Euro and Yen). Therefore, create a list which will contain this information:

```
        private static List<Currency> currencyList = new ArrayList<Currency>();
```

Then create a method that initialize this list:

```
        private  static void initializeCurrencies()
        {
                currencyList.add(new Currency("USA", "Dollar", 1800,1));
                currencyList.add(new Currency("EU","Euro",2000,2));
                currencyList.add(new Currency("Japan", "Yen",1945,3));

        }
```

Since your project does not, up until now, contain a Database to ensure data persistence, remember to call this method whenever you re-deploy the project. Otherwise the list `currencyList` will be null .

Question 1: Write a method that allows to retrieve a currency name when the user provides the ID e.g.:
http://localhost:8080/TD4/v1/converterApp/currencyConverter/currency/1

This example should gives you "dollar" as output since the currency "dollar" is associated with ID = 1. If the ID indicated by the user is not found, return an error message (you can simply use a HashMap to store ID and currency names/types).

Hints

– You should search for how you can add an ID to the path. **@Path("currency/identifier: [0-9]*")**

– Now to path the argument from the path to the method, check the "@PathParam" annotation.

Question 2: The Conversion Method

Question 2.1: Rewrite the method convert of TD2 in REST(Jersey).

Please specify the path of the new method as `@Path("conversion/source/destination/amount")`. For instance, the following link should lead to the result of converting 1000 dollars to Yens:

http://localhost:8080/TD4/v1/converterApp/currencyConverter/conversion/D/Y/1000

Question 2.2: Compare the method you write, with the method of TD2. Please explain the difference between RESTful Services and SOAP services.

Hints

– To pass multiple arguments to a method see "@PathParam" annotation.

Question 3: Producing Complex Data Types

In the previous two questions, the return type of the method was a simple String. In this question, we want to return a group of objects of a complex class (i.e. currency). Usually when a RESTful service returns an object. It either uses **XML** or **JSON** (as we have seen in TD3) to encode the response. In this question you should write two methods one produces the list of currencies encoded in XML and the other produces the list of currencies encoded in JSON.
The first method should have `getCurrenciesXML()` as name, and the second `getCurrenciesJSON()` as name.

Question 3.1: Write a method that returns the list of currencies encoded in XML.

The path should of this method should be `@Path("currencies")`.

Hints

– To specify the type produced by the method, use the annotation `@Produces(MediaType.TEXT_XML)` before the method declaration

– You should tell Jersey that your class `currency` is convertible to XML. This is done by using the annotation `@XmlRootElement(name="currency")` and `@XmlAccessorType(XmlAccessType.FIELD)`.

– You need to provide a default constructor (i.e. constructor with no arguments and that does nothing) to your class `Currency`

Test the result by using the following URL in the browser or in the REST Client:
http://localhost:8080/TD4/v1/converterApp/currencyConverter/currencies/

Question 3.2: Write another method that returns the list of currencies encoded in JSON.

The `@Path` should be exactly the same path i.e. `@Path("currencies")`.

Hints

– To specify the type produced by the method, use the following annotation before the method declaration:
`@Produces(MediaType.APPLICATION_JSON)`

Test the same URL, **what do you get as a result of it?**
Now you have two methods invoked when calling the `GET HTTP-Method` the same resource (URL). To choose the type of the result, you can use the REST-Client as follows:

– in `RESTClient` (Firefox), go to `Headers` ▶`Custom Header`.

– in the field put `Accept`. In the field value put `text/xml` and send the request. This means that you specify in the header of the request the type of the response that you want.
– Repeat the process with `application/json` as value.
– take a photo of the result XML and JSON and put it in the report.

Using Query Parameters

In this question, we want the methods `getCurrenciesXML()` and `getCurrenciesJSON()` to return the list of currencies sorted by the name of the currency. You should add a queryParameter to the method. This query parameter, let us call it `sortedYN`, is a String. When sortedYN="y", this means that the user wants the results sorted. Otherwise, no need to sort the results.
For instance the following link should display a list of currencies sorted by the currency name: http://localhost:8080/TD4/v1/converterApp/currencyConverter/currencies?sortedYN=y

Question 4: Add a query parameter to the methods getCurrenciesXML() and getCurrenciesJSON() so that the results are sorted .

Hints

– Note how the query parameter is added to the URL preceded by `'?'`.

Remarks

– Compress the project as a .zip file, attach it with the answers (pdf file) and send it to the email address mentioned above. Please put "WS-TD" in the subject (title) of the email.