Télécom Saint-Étienne
TD n°5 - 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: 03/11/2016

Objectives

- RESTful web service with persistence data source.
- Develop a RESTful web service that connects to the DB to answers GET & POST requests.

Tools Needed

- Eclipse EE (Enterprise Edition) Luna, Kepler etc.
- Glassfish
- REST-client: Postman for Chrome or RESTClient for Firefox.
- MySQL: If you don't have it you can download it from download page (Link for installation guide on Windows).
- JDBC Connector: connects MySQL to Java (Download link)
- Helper Classes (Downloadable from here)

1 DB Creation & Connection with GlassFish

In this TD, we will consider that our currency converter has offices around the world.

- Open `mysql` from the terminal.
- Create a Database called `WS_TD5`.
- type `USE WS_TD5`.
- Create table called `office` as follows:

```
CREATE TABLE office
(ID int NOT NULL AUTO_INCREMENT,
city varchar(255) NOT NULL,
manager_name varchar(255),
email varchar(255),
year_founded int,
PRIMARY KEY (ID)
);
```

- using `INSERT INTO` add 3 or 4 offices with cities including (Paris, London, NY, Hong Kong) with dates of construction (1998, 2001, 2010, 2015 etc.). Repeat Paris two times. Example:

```
INSERT INTO office (city,manager_name,email, year_founded)
VALUES ('paris','Jean Doe','paris@rest.fr',1998);
```

- Now you have to set the connection between `GlassFish` and `MySQL`. For that, follow this guide (link).
- Add a new package called: `com.rest.DB` to the `TD4` project.
- Add a new class called `DBClass` to your package. And connect to the mysql database from Java.
- Copy & Paste the file called "`DBClass.java`" (from helper classes). This class provides two methods one is private and the other is public called `return Connection`. This method returns a connection to the DB that can be then used to execute queries to the DB. Please go through the file, read the comments and understands what it does.
- Some details about the content of the class are provided over here.

2 Retrieving The Office List from The DB

In this part, the client of our RESTful service will ask to retrieve list of our offices around the world. The response will be in JSON format. The plan of what you need to do is:

1. Retrieve offices from the DB (using a select query).
2. Convert SQL data to JSON array.
3. Return the JSON array in the response.

The conversion is done as follows:

- Create a new package in your project. Call it `com.rest.util`.
- Add a new class to this package. Call the class `ToJSON`. Copy & Paste the content of the `ToJSON.java` file you got in the code folder of helper classes (link).
- This class contains a method `toJSONArray` that converts the results retrieved from the DB[1] to JSON objects contained in a JSON array. Please go through this class, read the comments and understand what it does.

JSONObject & JSONArray

The JSON Object and JSON array used in the class `ToJSON` are from the library `org.codehaus.jettison`. You can download it from here. Then you have to add it to your project:

- Go to eclipse, right click on the project name.
- `Properties` ►`Java Build Path` ►`Libraries Add External JARS` ►add the Jar file you downloaded.
- Do not forget to import this library when needed:
  - import org.codehaus.jettison.json.JSONArray;
  - import org.codehaus.jettison.json.JSONObject;

2.1 Question 1

In your class `CurrencyConverter`, created in TD5, add a new method called `getOffices()`. This method has the following header and annotations:

```
        @GET
        @Path("offices")
```

1. In JDBC, the results retrieved from the DB are received in an object of the type `ResultSet`.

```
        @Produces(MediaType.APPLICATION_JSON)
        public String getOffices()
```

This method:

1. Gets a connection using the method `returnConnection` mentioned above.
2. Creates a prepared SQL statement that reads all the content of the table.
3. Executes the statement and receives the results in a variable of the type `ResultSet`.
4. Converts this variable to JSONarray using the method `toJSONArray`.
5. Returns the JSON array as an output of the method. Since the method returns a variable of String type, you need to call `jsonArray.toString()`.

2.2 Question 2: A Method with QueryParameter

Using the technique of passing query parameters, write the following method:

```
        @GET
        @Path("office")
        @Produces(MediaType.APPLICATION_JSON)
        public String getOffice(@QueryParam ("city")String city)
```

It takes the city as a query parameter and displays offices in this city.

2.3 Question 3: Using The Response Class

Change the return type of the method above to Response instead of String.

```
        @GET
        @Path("office")
        @Produces(MediaType.APPLICATION_JSON)
        public Response getOffice(@QueryParam ("city")String city)
```

Remember that Response from the package: `import javax.ws.rs.core.Response`. Your method should return a response with success code (`code 200`) when there is an office in that city, otherwise return a failure code (`code 400`). Click here to see all the HTTP response codes. Check this code for an example.

3 Adding an Office to our DB

Remember that, up till now, our RESTful API responded to requests with `GET` method. `GET` Request is both **read-only** and idempotent. It is important to try to use other methods. In this section we will use the `POST` method. `POST` is neither read-only nor it is idempotent, since it allows the client to add a new resource to the DB.

3.1 Making a Post Request Using Form Data

In web programming (e.g. PHP and JSP) usually we use `POST` methods with forms (i.e. formulaires). A form usually contains a submit button, which when created will call the Post method and send a `Post request` that is going to be handled by a page. When the program is a web project (i.e. web site), this post request is handled by a PHP/HTML page. When it is a web service, we write a method and use the `@POST` annotation to handle the request coming from the form.

3.1.1 Question 4

To answer this question, you should use an already-made form (provided in the code folder of helper classes). This form is simply an HTML file (named **form.html**). Please take a look at the source code of this form. Using this form, you can submit data that allows you to create a new office. Then your task is to complete the following method:`addOffice`:

```
@POST
@Path("offices")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public String addOffice(...)
{
        \\complete the code here.
}
```

The annotation `@Consumes(MediaType.APPLICATION_FORM_URLENCODED)` indicates that this method receives data coming from the form. The method takes arguments corresponding to data filled in the form; such data should be handled by the method. Please take a look at the example to see how to connect the form and the method. Then, after receiving the data from the form, the method should add them (offices) to the DB using `INSERT INTO` query. Here is an example of this query with JDBC. Remember that you can always retrieve an object containing the connection using the `return Connection()` method in your DBClass.

If the insertion worked, return a an OK response (code 200), otherwise return an error response (e.g. 400)

3.2 Using a REST Client

In the previous subsection, we used a form to add an office. Although this is a human-friendly manner to add offices, this is not how restful API's usually work. For instance, in TD3 we wrote a client that accessed the twitter RESTful API. Now we will use the same methodology over here.

3.2.1 Question 5

Write a new version of the method `addOffice()` that receives data in JSON object instead of form data. This method has the following header and annotations:

```
@POST
@Path("offices_v2")
@Consumes(MediaType.APPLICATION_JSON)
public Response addOffice_v2(String incomingData) throws JSONException
{
        \\Complete Code Here
}
```

This method has to do the following:

1. Parse the string argument (`incomingData`) into JSONObject. To do so use:
   `JSONObject jsonObject = new JSONObject(incomingData);`
2. Retrieve the elements inside this jsonObject. For instance, to get the city:
   `String city = jsonObject.getString("city");`
3. Then, after getting all the elements from the jsonObject, insert them into the DB by calling the `InsertOfficeIntoDataBase(Office office1)` method that first connect to the database and then add a new office in the database. This method takes as argument an object of the type Office (A POJO Office class containing the geters and seters, similarly

to the Currency class and can be found with the helper classes). This method returns the http_code. If insertion worked, it will return 200. Otherwise it returns 400.

4. Depending on returned code, your method `addOffice_v2` should either returns an `OK` response or en error as follows:

```
int http_code = InsertOfficeIntoTheDataBase(office);
if (http_code==200) {
        return Response.ok().build();

}
else
{
        return Response.serverError().build();
}
```

In order to test your new method, you need to write a client:

– Create a new project called T4Client which a class called `Main` that contains a main() method.
– Convert it to maven project (`Configure ►Convert to Maven`).
– Use example as a guideline.
– This client uses Jersey classes.In order to import them into your project, you need to add them as dependencies into your `pom.xml` file.

```
<dependencies>
        <dependency>
              <groupId>com.sun.jersey</groupId>
              <artifactId>jersey-client</artifactId>
              <version>1.9.1</version>
      </dependency>
</dependencies>
```

– In the example, they use a string called `input` that contains the JSON object to be sent by the post method. You have to make a similar String that contains the office you want to add:

```
String input = "{\"city\":\"Lyon\",\"manager_name\":\"manager3\",\"email\":
\"Lyon@rest.fr\",\"year_founded\":\"2007\"}";
```

– Do not forget to change the `resource` value so that it points to your resource:
`http://localhost:8080/TD4/v1/converterApp/currencyConverter/offices_v2`
– To test, first republish the server and then run the client. Then check if a new office was added to the DB.

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.