

Developing RESTful Web Services with JAX-RS (Jersey) – Lab 1

Jersey is the reference library implementation of the JAX-RS standard. In this lab you will set up a sample Jersey project and implement some simple RESTful resources.

1. Download and unzip Tomcat 7:

Windows x64:

<http://mirrors.whoishostingthis.com/apache/tomcat/tomcat-7/v7.0.91/bin/apache-tomcat-7.0.91-windows-x64.zip>

or pick the correct version for your machine from: <https://tomcat.apache.org/download-70.cgi>

2. Setup an Eclipse EE Workspace

Open Eclipse EE (not standard edition!) and create a fresh workspace.

3. Add Tomcat server to your Eclipse Project

Go to the “Servers” tab in Eclipse, and click the link to create a new server.

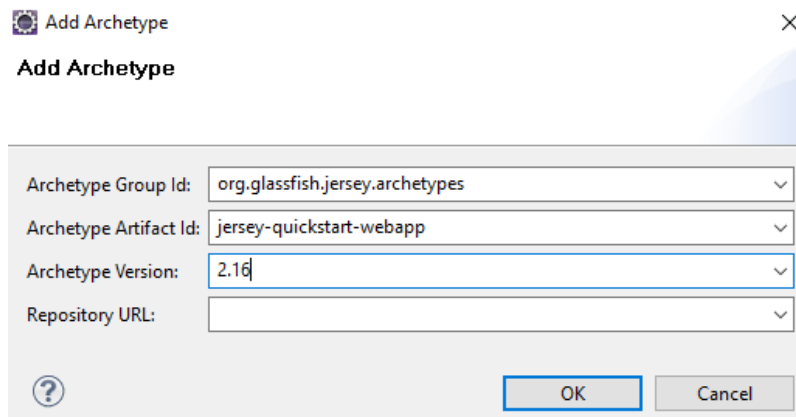
Select “Tomcat v7.0 Server” and click Next.

Browse to the Tomcat directory from Step 1 and then click Finish – “Tomcat v7.0 Server at localhost” should be displayed on the “Servers” tab.

4. Setup a Jersey sample project

Right click in the Project Explorer in Eclipse, select “New > Project...>Maven Project” and click Next and Next again.

In the Select an Archetype pane, click “Add Archetype...” and enter the following values:



Click OK, then select the archetype that was just created and click Next.

Enter ie.gmit.sw as the Group Id (package name)

Enter REST_Lab as the Artifact Id

Click Finish, and the project REST_Lab should be created in your Project Explorer. This will import and set up all required libraries, configuration etc.

(To remove the HttpServlet library not found error, right click on the REST_Lab project, select “Properties>Project Facets>Runtimes” and make sure that Apache Tomcat v7.0 is checked)

5. Run the Jersey sample project

Right click on the REST_Lab project in the Project Explorer, and select “Run As > Run on Server”. Click Next and then Finish.

You should see the landing page (generated by index.jsp) for the sample Jersey RESTful Web Application.

Click on the “Jersey resource” link to see the text “Got it!” displayed – this is the output of the getIt() method in the class MyResource. Note that this resource is accessed at the following URL: http://localhost:8080/REST_Lab/webapi/myresource

6. Examine the structure of the sample project

Open and inspect the following files:

src>main>webapp>WEB-INF>web.xml

pom.xml

src>main>webapp>index.jsp

ie.gmit.sw.REST_Lab>MyResource.java

7. Add a HelloWorld Resource which uses the @PathParam annotation

Copy the file HelloName.java from the dsREST_Lab.zip into the ie.gmit.sw.REST_Lab package in your Eclipse EE project.

Note that this class has two methods with the @GET annotation. One may be accessed using the URL: http://localhost:8080/REST_Lab/webapi/hello. The output of this method is the string “Hello World!”.

The second method uses the @PathParam annotation to accept a name as an input parameter, and may be accessed at the URL: http://localhost:8080/REST_Lab/webapi/hello/name/Patrick. The output of the method is the string “Hello Patrick!”. Note that “Patrick” is the input parameter which is passed via the URL. Edit the URL to display a different name.

8. Add a HelloWorld Resource which is a singleton

Copy the file HelloSingleton.java from the dsREST_Lab.zip into the ie.gmit.sw.REST_Lab package in your Eclipse EE project. By default, Jersey creates a new instance of the class which handles a given @Path for each new request which is received. Note that the class HelloSingleton makes use of the @Singleton annotation in order to implement a singleton design pattern (i.e. there is only ever a single instance of HelloSingleton created). The HelloSingleton Resource may be accessed at the URL: http://localhost:8080/REST_Lab/webapi/hellosingleton. Refresh the browser window, and you should see that the counter variable is incremented with each new request which is handled by the resource.

Comment out the @Singleton annotation and refresh the page. You should see that the counter now only ever displays 1 – the default instantiation behaviour for Jersey has been restored. You may need to stop and restart Tomcat for the change to take effect.

(In a few weeks we will see the amount of code which must be written to create a singleton with Apache Axis for XML-RPC – much more complex than with Jersey).

9. Exercise: Use PuTTY to retrieve different responses based on the client's Accept: type

One of the most flexible features of a RESTful architecture is the ability for servers to service requests in the client's preferred format. This exercise will demonstrate how this may be achieved using Jersey with PuTTY as the client.

Copy the file `HelloAcceptTypes.java` from `dsREST_Lab.zip` into the `ie.gmit.sw.REST_Lab` package in your Eclipse EE project. Note that in this example there are two methods which have the `@GET` annotation, and which can handle requests to the `@Path("/helloaccepttypes")`, but that they have different `@Produces()` annotations. Adding different `@Produces` annotations for the same `@Path()` allows different implementations based on the `Accept: type` (e.g. `Accept: text/html`) which is specified by the client.

Open the following URL in a web browser to check that the example is working:
http://localhost:8080/REST_Lab/webapi/helloaccepttypes

Download the PuTTY binary for Windows x64 from:
<https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe>

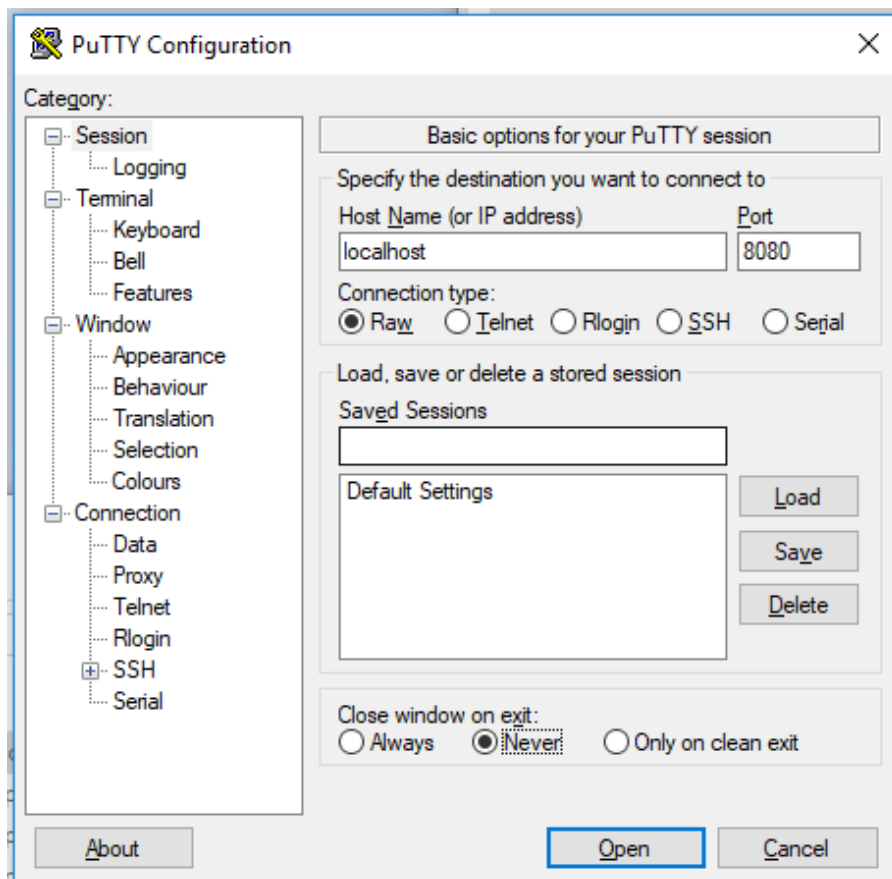
Open PuTTY and input the following settings, then click Open:

Host: localhost

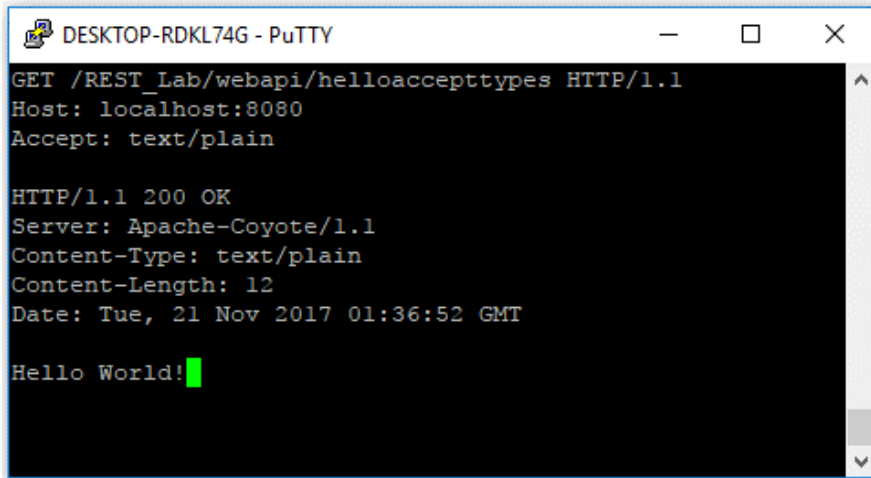
Port: 8080

Connection type: Raw

Close window on exit: Never



Copy the contents of the Putty_Request.txt file into the PuTTY console, and hit enter. You should see a response like the following:

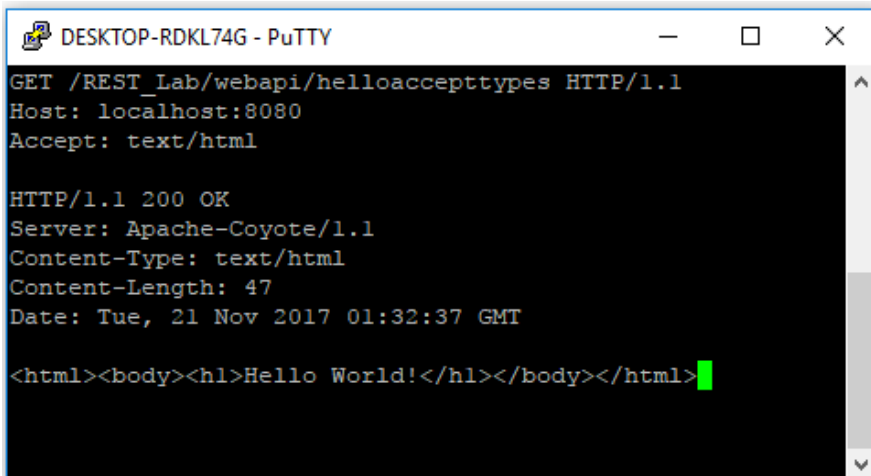


```
DESKTOP-RDKL74G - PuTTY
GET /REST_Lab/webapi/helloaccepttypes HTTP/1.1
Host: localhost:8080
Accept: text/plain

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/plain
Content-Length: 12
Date: Tue, 21 Nov 2017 01:36:52 GMT

Hello World!
```

Change the Accept: type to “Accept: text/html” (without quotes), then copy the modified GET request back into PuTTY. You should see a response like the following:

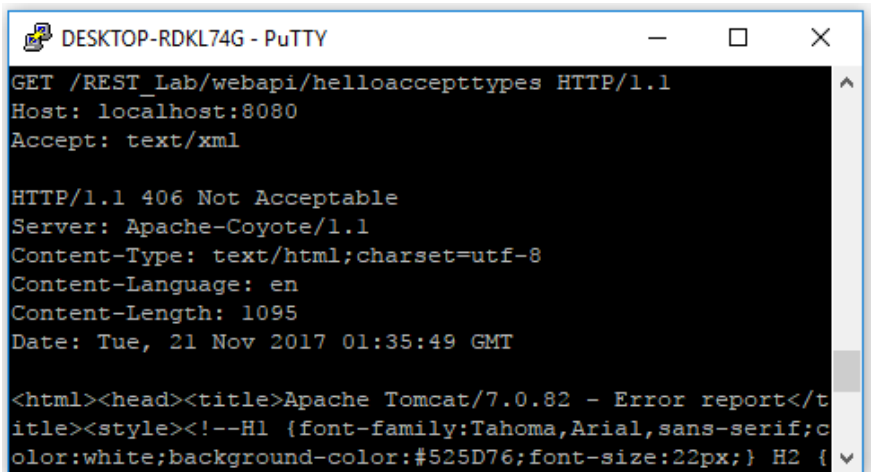


```
DESKTOP-RDKL74G - PuTTY
GET /REST_Lab/webapi/helloaccepttypes HTTP/1.1
Host: localhost:8080
Accept: text/html

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html
Content-Length: 47
Date: Tue, 21 Nov 2017 01:32:37 GMT

<html><body><h1>Hello World!</h1></body></html>
```

Change the Accept: type to “Accept: text/xml” (or any other Accept: type which does not have an @Produces() annotation to handle it), then copy the modified GET request back into PuTTY. You should see a response like the following:



```
DESKTOP-RDKL74G - PuTTY
GET /REST_Lab/webapi/helloaccepttypes HTTP/1.1
Host: localhost:8080
Accept: text/xml

HTTP/1.1 406 Not Acceptable
Server: Apache-Coyote/1.1
Content-Type: text/html; charset=utf-8
Content-Language: en
Content-Length: 1095
Date: Tue, 21 Nov 2017 01:35:49 GMT

<html><head><title>Apache Tomcat/7.0.82 - Error report</t
itle><style><!--H1 {font-family:Tahoma,Arial,sans-serif;c
olor:white;background-color:#525D76;font-size:22px;} H2 {
```

10. Add a Calculate Resource

Copy the file CalculateResource.java from dsREST_Lab.zip into the ie.gmit.sw.REST_Lab package in your Eclipse EE project.

Note that this example makes use of the javax.ws.rs.core.Response class to build the HTTP response to the caller (can use this to generate response codes other than 200 OK).

More on javax.ws.rs.core.Response:

<https://docs.oracle.com/javaee/7/api/javax/ws/rs/core/Response.html>

Test out the methods which are made available by the Calculate resource by building the appropriate URLs, e.g.

http://localhost:8080/REST_Lab/webapi/calculate/add/1/1

http://localhost:8080/REST_Lab/webapi/calculate/subtract/1/1

http://localhost:8080/REST_Lab/webapi/squareroot/1

11. Exercise: Create an interface-based version of the Calculator Resource

Create two new .java files in the ie.gmit.sw.REST_Lab package: CalculateInt.java and CalculateImpl.java

In CalculateInt.java, create an interface which has the same method signatures and annotations as the CalculateResource.java file above. Do not include the outer @Path() annotation on the interface, but do include the @Path() annotation on each method declaration.

In CalculateImpl.java, create the implementation for the methods specified in CalculateInt.java by copying them from CalculateResource.java

Apply the @Path("/calculateimpl") annotation to the CalculateImpl class, NOT to the CalculateInt interface (otherwise Jersey will not be able to find your implementation class and will return a 404).

Note that by using an interface for (almost) all of the JAX-RS/Jersey annotations, it is possible to separate out business logic (the implementation) from the annotations, resulting in cleaner code.