HU Extension Assignment 08 E-90 Cloud Computing

Handed out: 10/23/2015 Due by 11:59 PM on Friday, 10/30/2015

FINAL

Place all of your narratives and illustrations in a single Word or PDF document named E90_LastNameFirstNameHW08.docx [.pdf]. Use this assignment as the initial template. Please implement code solutions as a separate class in one (Java, C#, Ruby, Python,..) project. Add your steps and your code below problem statements used for that problem. Upload your homework file and your working code (e.g., filename.java) into your Assignment 8 folder. Do not include executables. Please also do not zip your files.

The following requires installation and configuration of Apache Tomcat on your Windows or Mac server as well as in Eclipse or IDE which are necessary configurations for this problem set.
Download Tomcat's Binary distribution from http://tomcat.apache.org/download-80.cgi
Select zip file for your system (Windows or MAC). Install Tomcat by expanding the zip file to: c:\ e.g. `c:\apache_tomcat_8.0.28`. Create the environmental variable `CATALINA_HOME` with that value. You must have environmental variable `JAVA_HOME` set to the installation of your Java6/7/8 JDK not JRE. You start Tomcat by running `startup.bat` file in `Tomcat bin` directory. You stop Tomcat by running `shutdown.bat`.

Go to http://jersey.java.net/download.html and download file called `jersey-archive1.19.jar` file with Oracle implementation of JAX-RS. The name of the hyper link is: `Jersey 1.19 JAR bundle`. Once you start building your Eclipse Dynamic Web Projects, you will add that `jar` to your Build Path. You will also copy that `jar` file (Jersey 1.19 bundle) to your `WebContent\WEB-INF\lib` directory under your Eclipse project. If you decide to modify and compile class `CustomerResource` or class `CustomerResourceClient` from the command line, you will need to have some of those `jars` in your `CLASSPATH`, as well. Your client class is contained in the file `CustomerResourceClient.java`.
Note: File `jax_rs_code.zip` contains java sources of classes discussed in the lecture on RESTFul technology. Files from this zip (src folder, web.xml) will need to be moved to your project.

If you are a C# programmer and you are familiar with WCF, you are welcome to run this exercise in C#. In that case, you will not use Java Jersey API but rather WCF classes. You are similarly welcome to do this assignment in any other language of your choice.

Problem 1:
Open Eclipse in a new workspace. Go to File -> New -> Dynamic Web Project. Call it "rest". This name is arbitrary, however, please do not change it or if you want to change it then modify all URL-s in the client code. Those URLs point to an application called "rest" on the localhost and port 8080. Expand attached ZIP file jax_rs_code.zip. Copy

packages `com.rest.client` and `com.rest.ws` from directory `rest` of the expanded ZIP file to the `src` directory under `Java Resources` folder of your project. Similarly, move file `web.xml` to the `WebContet\WEB-INF` directory of your Eclipse project. If your `WEB_INF` directory has a `lib` subdirectory, populate it with the Jersey bundled jar mentioned above (OR if you downloaded the Jersey zip then copy all the libraries). Build your project and fix any errors you might encounter. Run your project on your Tomcat server. You run the project by right clicking on the project and selecting `Run As -> Run on Server`. If your Eclipse is not aware of your Tomcat, please add new server. If your Tomcat is version 8, select Tomcat-8 as the server type. Subsequently, run your client class `CustomerResourceClient` as a Java application. Show your output. Submit working version of all classes, even if you did not modify a thing.
Points: 30

Problem 2:
Modify your `CustomerResource` class so that it could support a request that one particular `Customer` gets deleted. Also, add necessary methods that would allow your resource class to return a specified number of customers (2 or 3 is fine), starting with a particular customer id. To test new resource class you will have to modify your original client class `CustomerResourceClient` so that it creates 3 or 4 new customers before you could start requesting multiples of them back or before you could delete any of them. Code for creating 4 customers could be just four copy-pasted segments of existing code. Change customer names and perhaps their addresses. Run your code. Show your output. Submit the code you modified.
Points: 30

Problem 3:
Capture HTTP requests and responses for your `GET, PUT, POST` and `DELETE` method calls from Problem 2. Use TCPMon or some other HTTP monitoring tool to convince yourself that network traffic is indeed as advertised, meaning that you are sending HTTP messages with expected content and receiving HTTP messages with a similar content. Show your HTTP output `(GET, PUT, POST, DELETE)`.
Points: 10

Problem 4:
Launch 2 AWS EC2 instances (Windows or Linux) in the same availability zone using Amazon Console or AWS CLI. Install Java JDK and Tomcat on one AWS Instance and startup Tomcat. Make sure you have security group set up (SSH, HTTP, etc.) and inbound rules. This is your Server and serves as your host of your RESTful web service. You can also use an AWS EC2 Instance that has Tomcat pre-installed (make sure Java JDK is installed and Tomcat is running. On your Server check that you can reach Tomcat in a browser: http://localhost:8080/). Assign a private IP address to your Server by creating an Elastic IP. You can do this in AWS Console or AWS CLI. Export your project (CustomerResource class) from problem 2 in Eclipse (as rest.war file) and copy it to your Server's Tomcat \webapps directory. Make sure it has proper executable permissions. Restart your Tomcat server.

Launch a second AWS EC2 instance (Windows or Linux similar to your first EC2 instance with security group then install Java JDK.
On your laptop in Eclipse - add the private IP address of your Server to your CustomerResourceClient class for post, get, delete customer and recompile.
Copy your .jar folder to similar folder structure as you used in your package to your client (2nd AWS EC2 Instance). . Run your class: java … You should see customers being created, queried, … Show your output. Submit the code you have modified.
Points: 30