SJSU CMPE 282 HW RESTful Web Svc on NoSQL FALL 2017

REMINDER: Each homework is individual. "Every single byte must come from you." Cut&paste from others is not allowed.

Theme: Java-based REST server + NoSQL on your laptop

Description

REST server side: please create Java-based RESTful Web Service server on your laptop. The server retrieves data from, and persists data to NoSQL according to the functional matrix specified later. To facilitate debugging, please add appropriate high-level logging from your REST server by using existing packages (e.g., Log2J, etc.).

- NoSQL: choose either one of two most popular NoSQLs, Cassandra or MongoDB
- On your laptop, Install an instance of NoSQL for development purpose without relying on container (we will work on NoSQL on container in the next homework)

You may build your REST server in any way, such as using Servlet directly (and deploy to say Tomcat), or leverage Jax-RS (and then deploy to say Tomcat), or even use Spring (standalone or deploy to Tomcat), etc. See Design options later.

REST client side: you can use any REST client, such as RESTClient (Firefox), soapUI, curl, etc.

Data objects

There are two types of objects:

Employee
- **int** id // user-generated unique value across all employees
- **String** firstName
- **String** lastName

Project
- **int** id // user-generated unique value across all projects
- **String** name
- **float** budget

NOTE: the id property is a **user-defined** property. Do **not** use the internal built-in identifier (or key name) as the id.

Format of HTTP/REST Request and Response

You can choose either XML or JSON. No need to have both. For example, XML representations of these objects are as follows

```
<employee>
  <id>1</id>
  <firstName>John</firstName>
  <lastName>Doe</lastName>
</employee>
<project>
  <id>10</id>
  <name>projectX</name>
  <budget>1234.56</budget>
</project>
<employeeList>
  <employee>…</employee>
  <employee>…</employee>
  …
</employeeList>
```

```
<projectList>
  <project>…</project>
  <project>…</project>
  …
</projectList>
```

Environment

Install a Linux VM on your laptop

- To install a Linux VM, first install the free VMware Workstation Player (Windows, Linux) or free Oracle VirtualBox (Mac) on your laptop, and then install a recent Linux such as Ubuntu desktop as guest OS.
- Linux VM is recommended since we will work on Linux VM only in the next homework. However, it is certainly possible to develop the REST server on Windows in this homework, and then move the code to Linux VM in the next homework.

Additional requirements

- REST resource (URI) must start with /cmpe282<YourName><L3SID>/rest/.. . For example, the entire URL would be http://hostname[:port]/cmpe282Demo123/rest/… The screenshots (explained later) must include the proper resource (URI).
- The database name (i.e., keyspace in Cassandra) of NoSQL must be cmpe282<YourName><L3SID>. The screenshots (explained later) must include the proper database name.
- Externalize the NoSQL configuration information, such as NoSQL IP, port, DB name, etc., as initial parameters in the web.xml. This would become useful in HW2.

Questions

Q1.

a. List technologies, softwares (including version), and platforms for dev tools, REST client, REST server, and NoSQL database.
b. How did you build your REST server?
c. Is the format of the data object based on XML or JSON?
d. A sample HTTP request URI and request body for POST to create a new `employee` based on the XML format or JSON format (depending on your answer to c). Also indicate if there is any additional setup (e.g., HTTP header, etc.).
e. If the NoSQL needs to create DB objects beforehand, such as Cassandra, include the script to create DB objects.

For example,

a.
Dev tools: Eclipse Java EE Neon.3 on Windows 10
REST Client: soapUI 5.3.0 on Windows 10
REST Server: direct Servlet, JAXB, Tomcat 8.5.16 on Windows 10
NoSQL database: Cassandra 3.9.0 (from Datastax) on Windows 10

b. REST Server: use HttpServlet directly (without relying on JAX-RS and Spring). Then deploy to Tomcat.

c. Based on XML

d.
Request URI: `POST /cmpe282Demo123/rest/employee`
Request body:

```
<employee>
```

```
    <id>1</id>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
</employee>
```

No additional setup

e. Cassandra script: …

Q2.

Functionality matrix of the REST server: (URI "/…/rest" means "/cmpe282Demo123/rest")

HTTP method + URI	Description	HTTP status - successful	HTTP status –err, in addition to 400, 500
GET /…/rest/employee/m	Retrieve employee with id m. Resp body: XML or JSON	OK (200)	Not found (404)
GET /…/rest/project/n	Retrieve project with id n. Resp body: XML or JSON	OK (200)	Not found (404)
POST /…/rest/employee	Create a new employee. Req body: XML or JSON Resp header: include Location header	Created (201)	Conflict (409)
POST /…/rest/project	Create a new project. Req body: XML or JSON Resp header: include Location header	Created (201)	Conflict (409)
PUT /…/rest/employee/m	Update employee with id m. Req body: XML or JSON (allow partial properties, except id, to be updated)	OK (200)	Not found (404)
PUT /…/rest/project/n	Update project with id n. Req body: XML or JSON (allow partial properties, except id, to be updated)	OK (200)	Not found (404)
DELETE /…/rest/employee/m	Delete employee with id m	OK (200)	Not found (404)
DELETE /…/rest/project/n	Delete project with id n	OK (200)	Not found (404)
GET /…/rest/employee	Retrieve all employees. Resp body: XML or JSON (<employeeList>…</employeeList>)	OK (200)	Not found (404)
GET /…/rest/project	Retrieve all projects. Resp body: XML or JSON (<projectList>…</projectList>)	OK (200)	Not found (404)

For each of the function matrix,

- add the **status** column and specify its implementation status (i.e., **done**, **partially done**, **not implemented**)
- If it's done

    - include before screenshot of data objects/rows on NoSQL
    - error case: include screenshot of REST request and response (method, URI, HTTP status code, HTTP headers)
    - success case: include screenshot of REST request and response (method, URI, HTTP status code, HTTP headers)
        - POST: need to show HTTP response headers which include the **Location** header
        - PUT: need to demo **partial** property update (i.e., HTTP request only includes a single property)
    - include after screenshot of data objects/rows on NoSQL, if it's PUT, POST, or DELETE
- if it's partially done, indicate exactly which portion is done and which is not

You should also include the following info

- List known issues if any
- Any additional unique design or features you are proud of

Note: You do NOT need an interface like http://thomas-bayer.com/restgate/index.do which has "built-in" REST client. The homework asks for the core of REST server (i.e., be able to handle GET, PUT, POST, and DELETE). You can choose any REST client to communicate with your REST server.

Submission

Submit the followings as *separate* files to Canvas

- CMPE282_HW1_<YourName>_<L3SID> (.pdf, .doc, or .docx): the report consists of answers and screenshots to questions specified in **Question**.

  Note:
    - You receive no credit if your report is not .pdf, .doc, or .docx.
    - REST resource (URI) must start with /cmpe282<YourName><L3SID>/rest/.. . The screenshots must include the proper resource (URI).
    - If a screenshot is unreadable, it will be treated as if you did not turn in that screenshot.
    - You will not receive credit if you do not follow requirements (including naming conventions).
- CMPE282_HW1_<YourName>_<L3SID>.zip: zip the project directory to include .java sources, web.xml, and, if any, config file. Do *not* include .class, .jar, .war files.
- cmpe282<YourName><L3SID> (.war or .jar): a *deployable* .war file, or (mainly for Spring) a standalone .jar file

The ISA and/or instructor leave feedback to your homework as comments and/or annotated comment. To access annotated comment, click "view feedback" button. For details, see the following URL: http://guides.instructure.com/m/4212/l/106690-how-do-i-use-the-submission-details-page-for-an-assignment

Additional Info

- Get started early

- Design options
    - war vs ear+war
    - persistence: either Cassandra or MongoDB
    - REST: direct Servlet vs Jersey (JAX-RS)
        - http://www.tutorialspoint.com/servlets/
        - https://jersey.github.io/
    - XML: JAXB vs XPath vs DOM/SAX

        - JAXB: http://www.vogella.com/tutorials/JAXB/article.html
        - XML: http://www.vogella.com/tutorials/JavaXML/article.html

- Servlet: be able to handle GET, POST, PUT, and DELETE
    - http://www.tutorialspoint.com/servlets/
    - http://www.vogella.com/tutorials/EclipseWTP/article.html
    - In web.xml, url-pattern (within servlet-mapping) should be similar to /rest/*
    - req.getPathInfo() returns string after /rest. E.g., it returns /employee/20 if URI is /rest/employee/20

- JAX-RS
    - http://www.vogella.com/tutorials/REST/article.html
    - http://crunchify.com/how-to-build-restful-service-with-java-using-jax-rs-and-jersey/
    - http://crunchify.com/create-very-simple-jersey-rest-service-and-send-json-data-from-java-client/
- Spring
    - http://www.dineshonjava.com/2017/03/spring-restful-web-services-json-crud-example.html
    - http://www.dineshonjava.com/2017/03/spring-restful-web-services-xml-crud-example.html
- How to support both XML and JSON format in REST?
    - The client sends in the proper HTTP **Accept** header ("application/xml", “application/json”) in request, or the client sends in the proper HTTP **Content-Type** header in POST or PUT request. The server sets the proper HTTP **Content-Type** header ("application/xml", “application/json”) in response. One should also choose a default content type if the request does not have the proper Accept header.
    - Use different URI to distinguish between XML and JSON.
    - Jersey (JAX-RS)
        - https://www.tutorialspoint.com/restful/restful_quick_guide.htm
        - http://www.benchresources.net/jersey-2-x-web-service-using-both-json-xml-example/
        - https://www.javainterviewpoint.com/jax-rs-rest-consumes-both-xml-and-json-example/
        - https://www.javainterviewpoint.com/jax-rs-rest-produces-both-xml-and-json-example/
        - Spring: http://www.journaldev.com/8934/spring-rest-xml-and-json-example