Assignment 4: REST API – POST, PUT, DELETE

Release Date: October 19, 2016
Due Date: November 6, 2016 (11:59 pm Central Time)
(This is an individual assignment)

Important Note:

There are two important dates for this assignment: November 4, 2016 and November 6, 2016

Midnight of November 4: Submit first version of the assignment by this date. The assignment does not have to work completely, but it has to be at the level where all startup related issues have been resolved (Eclipse, Tomcat, etc.). This submission will be graded for 3 points. This means if your first commit is on or before November 4th , you will receive points out of 100. If you only have one submission (after November 4th , you will receive points out of 97). The reason to have this deadline is to encourage you to get started on the assignment well ahead of the actual deadline.

Midnight of November 6: Deadline by which you should have completed your assignment.

Objective:
In this assignment we will learn how to implement POST, PUT, and DELETE REST API calls using RESTEasy and JDBC.

Introduction:
You are going to add following calls to the projects REST resource that you implemented in the previous assignment.

- Create a project (POST)
- Update a project (PUT)
- Delete a project (DELETE)
- Read a project (GET)

Note that you don't need to interact with the eavesdrop website (http://eavesdrop.openstack.org/) in this assignment. The project resource that you will create using above calls would be stored in a MySQL database. For interacting with the database through your Java code you need to use JDBC.
Specifically, you will support following resources and actions on them. The exact requests and responses are colored in grey below.

1) Create project example:

```
POST http://localhost:8080/assignment4/myeavesdrop/projects/
<project>
  <name>solum</name>
  <description>Project representing solum</description>
</project>
```

This should create project named solum in your system. You should create a unique projectId to represent a created project. The projectId will be used in subsequent calls. You should return the resource URL representing the newly created project as part of the location response header (check ex03_1 from the OReilly book for how to do this.)
Response:
If the request is successful:
  Response status: 201 Created
  Location header: <Full path to the newly created resource>
  Response body: Empty
If the request is unsuccessful based on validation requirements:

Response status: 400 Bad Request
Response body: Empty

2) Update project example:

PUT http://localhost:8080/assignment4/myeavesdrop/projects/<projectId>

Body:

```
<project>
  <name>solum</name>
  <description>Updated description of solum</description>
</project>
```

This should change description of project represented by project with id "projectId" that is passed in the PUT request.
"Project representing solum" to
"Updated description of solum"

Response:
If the request is successful:
Response status: 204 No Content or 200 OK
Response body: Empty
If the request is unsuccessful based on validation requirements:
Response status: 400 Bad Request
Response body: Empty

To get immediate feedback on whether the action was successful or not in case of POST and PUT on projects, you can return the created/updated resource in the response body. It is not required though.

3) Get project details:

GET http://localhost:8080/assignment4/myeavesdrop/projects/<projectId>

Response XML:

```
<project id=projectId>
  <name>solum</name>
  <description>Project representing solum</description>
</project>
```

Response details:
If the request is successful:
Response status: 200 OK
Response body: XML representation of the requested resource (Response XML shown above)
If the request is unsuccessful based on validation requirements:
Response status: 404 Not Found
Response body: Empty

4) DELETE http://localhost:8080/assignment4/myeavesdrop/projects/<projectId>

Response details:
If the request is successful:
Response status: 200 OK
Response body: Empty
If the request is unsuccessful based on validation requirements:
Response status: 404 Not Found
Response body: Empty

Design Details:

You need to use RESTEasy framework for this assignment. You should organize your code into two layers, resource layer and a service layer. You can divide the responsibilities between these layers as follows. Use the resource layer to perform input validation and response generation. Use the service layer to implement database interactions using JDBC.

Input validation:

Perform following checks on the input data:
•In the call to create a project (POST /projects), ensure that name and description are not empty (“”).
•In the call to update a project (PUT /projects), ensure that name and description are not empty.
If the input fails validation in any of the above cases, return HTTP 400 Bad Request response.
•In the calls to GET a project, or DELETE a project, if a project name that does not exist in your system is passed, return HTTP 404 Not Found response.

Database design:
You will need one table to represent projects.
For details on what columns to define, what should be the primary key, etc. refer to the database example of students and courses tables from the class notes (Database-basics.pptx).
We will have to use JDBC and MySQL for this assignment.
You will need to install and run MySQL server locally on your machine. You can find instructions on installing MySQL in the README of the JDBC folder of class github repository (https://github.com/devdattakulkarni/ModernWebApps.git).

We are working with the system staff to get you all database access for this assignment. Once this is ready, we will announce it on Canvas and Piazza.

NOTE:
You are not required to implement either functional tests or unit tests for this assignment. However, for the service layer, implementing test methods, which execute the service methods that perform database calls would be helpful when you develop your JDBC code.
Also, writing functional tests for the resource layer will be helpful when designing and debugging REST calls (specifically for response generation, content marshalling, etc.)
Another tool that you may find helpful is Mozilla add-on called RESTClient (https://addons.mozilla.org/en-Us/firefox/addon/restclient/). Using this you will be able to make REST calls against your REST service. I have uploaded a screen shot in the Assignments-addendum folder under Files section showing how to do POST using RESTClient (post-with-restclient.png)
You might also find curl (http://curl.haxx.se/) useful while developing/debugging your REST API. I have uploaded a screen shot showing how to use curl for POST/GET/PUT actions (post-with-curl.png).

Submission Details:
Create the following folder structure:

```
assignment4/src/main/java/assign/domain/<Domain class>
assignment4/src/main/java/assign/resources/<Resource class>
assignment4/src/main/java/assign/services/<Service classes>
assignment4/src/resources/schema.ddl
assignment4/README.yaml
assignment4/WebContent/WEB-INF/web.xml
assignment4/pom.xml
```

assignment4/src/resources/schema.ddl should contain table creation statements. Refer to schema.ddl in the JDBC example folder for example of this.
Use assignment4/WebContent/WEB-INF/web.xml to define database credentials/parameters

Instructions for web.xml and schema.ddl

1. Include following statements in schema.ddl
   create <database-name>;
   use <database-name>;
   create <table-name> (check the syntax for the create table SQL statement).

2. Modify web.xml to read the database name using DBNAME variable instead of reading the complete DBURL.

(i.e. instead of
https://github.com/devdattakulkarni/ModernWebApps/blob/master/assignment-templates/assignment4/WebContent/WEB-INF/web.xml#L13
read DBNAME.

The value of DBNAME should be the same as that specified in the 'create <database-name>' statement in schema.ddl

3. Additionally, read DBHOST from web.xml as well.

4. Construct the database url by using the values of DBHOST and DBNAME variables. (You can use the default MySQL port of 3306).

5. Except for the DBNAME variable, keep other variables empty (DBHOST, DBUSERNAME, DBPASSWORD) (i.e. <param-value> should be left empty/blank) when submitting your assignment.

When you are *developing* the assignment, you should put values for these variables in web.xml.

When you are *about to submit* the assignment, just remove the values for DBHOST, DBUSERNAME, DBPASSWORD and then submit. The reason to do this is that it is not a good software engineering practice to store these variables in a source code repository. For grading, we will add our values for DBHOST, DBUSERNAME, DBPASSWORD to your web.xml before running your code. Make sure you include schema.ddl as mentioned in step 1. We will run it against the DBHOST to setup the database and tables that are required by your application.

Use following format for README.yaml

```
---
name: SUBSTITUTE_WITH_YOUR_NAME
eid: SUBSTITUTE_WITH_YOUR_EID
notes: |
  Add notes (if you have any).
```

Push the entire assignment4 folder to your bitbucket account.
Grant “read” access to Eddy and Brandon
  - Usernames: eddy_hudson, brandonhollowell
We will use the latest commit ID for grading.

Helpful Material:
a) Lecture Notes: REST API, Database-basics, JDBC
b) Book Chapters: Chapters 1, 2, 3, 4, 5, 6, 17, 18, 19 of RESTful Java with JAX-RS 2.0 book.
c) Github (https://github.com/devdattakulkarni/ModernWebApps)
   Projects: REST/REST-JDBC, JDBC, REST
d) The ex03_1 example code from RESTful Java book
   https://github.com/oreillymedia/restful_java_jax-rs_2_0
e) Starter project:
   https://github.com/devdattakulkarni/ModernWebApps/tree/master/assignment-templates/assignment4

Grading criteria:

1) POST works as expected
- An entry is created in db table
- Location header is sent back
- Appropriate HTTP response is sent back

2) PUT works as expected
- Correct entry is updated in the db table
- Appropriate HTTP response is sent back

3) DELETE works as expected
- Correct entry is deleted from the db table
- Appropriate HTTP response is sent back

4) GET works as expected
- Correct project XML is sent back in response
- Appropriate HTTP response is sent back

5) Error checking/input validation
- Appropriate responses are sent back corresponding validation requirements listed earlier

Hints on getting started:

1) Read the lecture notes on MySQL.
2) Download and try the REST-JDBC example from course github page (mentioned in Helpful Material section above). Your goal should be to get this example working as the first step (either in Eclipse or Intellij or directly from command line (using mvn)).
3) Convert the tables in the REST-JDBC example to the tables that are required for this assignment.
4) Make sure that you are able to insert, query, delete items from the database using plain SQL.
5) Modify the test methods available in TestCourseStudentServiceImpl class according to this assignment's requirements.
6) Exercise the new methods that you have created.
7) Modify the service layer to use the new methods that you have created.
8) Convert the domain objects from REST-JDBC example to the domain objects required for this assignment.
9) Test end-to-end.