Assignment 5: Hibernate

Release Date: November 8, 2016
Due Date: November 20, 2016 (11:59 pm Central Time)
(This is an individual assignment)

There are two important dates for this assignment: November 18, 2016 and November 20, 2016.

Midnight of November 18: 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, MySQL, Hibernate, etc.). This submission will be graded for 3 points. This means if your first commit is on or before November 18th , you will receive points out of 100. If you only have one submission (after November 18th , 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 20: Deadline by which you should have completed your assignment.

1) Objective:
In this assignment we will learn how to use Hibernate to interact with database.

2) High-level Description:
You are going to add following calls to the projects REST resource that you implemented in the previous assignment.

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

A project may have one or more meetings associated with it. You will implement a meetings sub-resource for a project with a create (POST) action. Note that you don't need to interact with the eavesdrop website (http://eavesdrop.openstack.org/) in this assignment. The project and the meeting resources 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 *Hibernate*.
Specifically, you will support following resources and actions on them. The exact requests and responses are colored in gray below.

1) Create project (same as your assignment 4)
POST http://localhost:8080/assignment5/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.)

2) Create a meeting for a project:
POST http://localhost:8080/assignment5/myeavesdrop/projects/<projectId>/meetings
Body:
```
<meeting>
  <name>m1</name>
  <year>2014</year>
</meeting>
```
This should create meeting named m1 and it should be associated with project with id “projectId” that is passed in the POST call.
A meeting can be associated with only one project. You should create a unique meetingId to represent a created meeting.

Response:
If the request is successful:
   Response status: 201 Created
   Response body: Empty
If the request is unsuccessful based on validation requirements:
   Response status: 400 Bad Request
   Response body: Empty

3) Get project details:
GET http://localhost:8080/assignment5/myeavesdrop/projects/<projectId>
Response XML:
```
<project id=``projectId">
   <name>cs378</name>
   <description>Project representing cs378</description>
   <meetings>
     <meeting id=``m1">
        <name>m1</name>
         <year>2014</year>
    </meeting>
     <meeting id=``m2">
         <name>m2</name>
         <year>2015</year>
   </meeting>
</meetings>
</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) Update meeting:
PUT http://localhost:8080/assignment5/myeavesdrop/projects/<projectId>/meetings/<m1>
Body:
```
<meeting>
   <name>m1-name-changed</name>
   <year>m1-description-changed</year>
</meeting>
```
This should change name and description of the meeting represented by meeting with id “m1” for project with id “projectId” that is passed in the PUT request.
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
If there is no project with id “projectId” or if there is no meeting with id “m1” then you should return a 404 Not Found response.
   Response status: 404 Not Found
   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. If you return the updated resource then the response code should be 200 OK.

5) DELETE http://localhost:8080/assignment5/myeavesdrop/projects/<projectId>
This should delete the project with id "projectId" and all the meetings associated with it.
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

3) Design Details:
You need to use RESTEasy framework for building the API layer and ***Hibernate*** framework to interact with the database. 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 *Hibernate*.

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 ("") or does not contain only spaces.
•In the call to create a meeting (POST /projects/../meetings), ensure that name and year are not empty ("") or does not contain only spaces.
•In the call to update a meeting (PUT /projects/../meetings), ensure that name and year are not empty ("") or does not contain only spaces.
If the input data fails validation in any of the above cases, return HTTP 400 Bad Request response.
•In the calls to GET, PUT, or DELETE, if a project id or meeting id that does not exist in your system is passed, return HTTP 404 Not Found response.

Database design:
You will need two tables to represent projects and meetings. For details on what columns to define, what should be the primary key, etc. refer to the project *Hibernate* on class github page.
You can install and run MySQL locally or you can use the one deployed for you by the CS department.

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 Hibernate 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.)
Here are two tools that you can use for interacting with your REST service – RESTClient and curl. RESTClient is a Mozilla add-on (https://addons.mozilla.org/en-Us/firefox/addon/restclient/). Curl is a command line tool (http://curl.haxx.se/). I have uploaded screen shots on Canvas in the Assignments-addendum folder under Files section showing how to do POST using RESTClient (post-with-restclient.png) and using curl (post-with-curl.png).

4) Submission Details:
Create the following folder structure:
assignment5/src/main/java/assign/domain/<Domain class>
assignment5/src/main/java/assign/resources/<Resource class>
assignment5/src/main/java/assign/services/<Service classes>
assignment5/src/main/resources/hibernate.cfg.xml

```
assignment5/README.yaml
assignment5/WebContent/WEB-INF/web.xml
assignment5/pom.xml
```

Use assignment5/WebContent/WEB-INF/web.xml to define database credentials/parameters

Use following format for README.yaml

```
---
name: SUBSTITUTE_WITH_YOUR_NAME
eid: SUBSTITUTE_WITH_YOUR_EID
notes: |
  Add some notes here.
```

Push the entire assignment5 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.

5) Helpful Material:

a) Lecture Notes: REST API, Database-basics, ORM
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: Hibernate, REST, REST-JDBC
d) The ex03_1 example code from RESTful Java book
https://github.com/oreillymedia/restful_java_jax-rs_2_0

6) Grading criteria:

1) POST works as expected
   - Project and meeting entries are created in the database
   - Location header is sent back
   - Appropriate HTTP response is sent back
2) PUT works as expected
   - Correct entries are updated in the database
   - Appropriate HTTP response is sent back
3) DELETE works as expected
   - When a project is deleted all of its meetings are deleted
   - Appropriate HTTP response is sent back
4) GET works as expected
   - Correct project XML with all the meetings is sent back in response
   - Appropriate HTTP response is sent back
5) Error checking/input validation
   - Appropriate responses are sent back following the validation requirements listed earlier

7) Hints on getting started:

1) Read the lecture notes on ORM.
2) Download and try the Hibernate example from course github page. 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 Hibernate example to the tables that are required for this assignment.
4) Convert the domain objects from Hibernate example to the domain objects required for this assignment.
5) Make sure that you are able to insert, query, delete items from the database using the test methods provided in that project.
6) Add RESTEasy specific things (resource definitions, marshaling tools, web.xml, etc.).
7) Modify the DBLoader to create new methods that you need for this assignment.
8) Modify the resource layer to use the new methods that you have created.
9) Test end-to-end.