Assignment 2: Spring framework and Unit testing

Release Date: September 22, 2016
Due Date: Midnight October 2, 2016 (Sunday, 11:59 pm)
(This is an individual assignment)

Important Note:
There are two important dates for this assignment: September 30, 2016 and October 2, 2016
Midnight of September 30: 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 September 30th, you will receive points out of 100. If you only have one submission (after September 30th, 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 October 2: Deadline by which you should have completed your assignment.

Objective:
In this assignment you will learn about Spring framework and unit testing.

Introduction:
In this assignment we will build on the previous assignment and calculate the total number of files generated corresponding to meetings that happened for a particular project in a particular year. Similar to assignment 1, you will query the eavesdrop website (http://eavesdrop.openstack.org/meetings) for obtaining the meeting data.

You *do not* need to implement sessions in this assignment.

Details:
You will implement a Spring controller that will handle two query parameters:
- project
- year

The project and the year parameters will be used for determining what data should be shown in response to a request.

The allowed values for the query parameters are as follows:

Query parameter	*Allowed Values*	*Case sensitivity, other restrictions*
project	Names of projects from eavesdrop site	Should not be treated as case sensitive (solum, Solum, soLuM, etc. should be treated as same)
year	Year numbers from eavesdrop site	Only digits are allowed (2016 is correct, twentysixteen is incorrect)

Setup:
You should run your Spring controller to respond to following URL pattern: /openstackmeetings
You should set the context root of your web app to: /assignment2

With above settings, your web app will be accessible at:
http://localhost:8080/assignment2/openstackmeetings

Here are some sample interactions:

http://localhost:8080/assignment2/openstackmeetings	Welcome to OpenStack meeting statistics calculation page. Please provide project and year as query parameters.
http://localhost:8080/assignment2/openstackmeetings?project=solum&year=2015	Number of meeting files: 4
http://localhost:8080/assignment2/openstackmeetings?project=solum&year=2014	Number of meeting files: 4

Other constraints on input and output
1. The query parameters themselves are case-sensitive. You are required to support query parameters that are only lower case ("project", "year").
2. The "project" and "year" parameters are required to be passed in together. If either one is missing you should respond back with following message: "Required parameter <name> missing".
4. Output format should be plain text.
5. If an invalid value is provided for project or year parameters you should respond with following error messages as applicable: "Project with <name> not found". "Invalid year <year-number> for project <project-name>". If in a single request both the project name and the year are invalid then your response should be "Project with <name> not found".

Design details and getting started:
You need to use Spring framework for this assignment. Break down your code into logical layers consisting of a controller layer and a service layer. The controller layer will contain your controller class and the service layer would contain one or more service classes. The controller should perform input validation and output generation. The service classes should implement the logic of querying the eavesdrop website, parsing the data, generating the statistics, etc.

For dependency injection, you are free to use either setter injection or constructor injection. Define the beans and dependency wiring in *servletContext.xml* file.

Implement JUnit and Mockito based unit tests for the controller class and service classes.

You need to provide *at least five unit tests* between controller and service classes. This means you could have 5 tests for the controller and 0 for the service class, or 4 for controller class and 1 for service class, or 0 for controller and 5 for service class, or some other combination. Note that unit tests that don't test anything will not count. Also, realize that unit tests are going to dependent on your code. If you make your code modular with separate classes and separate methods for different concerns, you will have better chance of writing good unit tests.

Use pom.xml with dependencies for mockito, junit, spring-webmvc.

Start with setting up your development environment (Eclipse/IntelliJ/other IDE or text editor with command line). Try out sample servlet projects from the class github repository (listed under Useful material section below). Divide the implementation into parts (as mentioned above).

Grading criteria:

- First version was submitted by September 30th
- Correct count output when project and year are provided valid values
- Correct count output when project and year are provided invalid values
- Correct count output when one of project or year is missing
- Correct output when neither project nor year are provided
- Unit tests for Controller class
  - Unit test present
  - Tests pass
  - Appropriate mocks created
  - Appropriate expectations set
  - Appropriate asserts and verifys defined
- Unit tests for Service class(es)
  - Same as above

Submission Instructions:

Name your submission folder as “assignment2”
Name your Controller class “OpenStackMeetingsController.java”
Structure of assignment2 directory should be as follows:
assignment2/src/main/java/<Controller class>
assignment2/src/main/java/<Service interface>
assignment2/src/main/java/<Service class>
assignment2/src/main/java/<Any other Java class(es)>
assignment2/src/test/java/<Controller test class>
assignment2/src/test/java/<Service test class>
assignment2/WebContent/WEB-INF/web.xml
assignment2/WebContent/WEB-INF/servletContext.xml
assignment2/pom.xml
assignment2/README
Structure the README in yaml format as shown below:

```
   Name: Your name
   EID: UT EID
   Notes: Discussion/Notes about the assignment (if you have any)
```

Push the entire assignment2 folder to your bitbucket account.
Grant “read” access to me, Eddy, and Brandon
- Usernames: devdattakulkarni, eddy_hudson, brandonhollowell

You don’t need to submit anything on canvas.

Summary of requirements:

- Use Spring
- Create Controller and Service classes
- Use constructor or setter injection to wire controller and service classes together

- Define beans in servletContext.xml
- Write at least 5 unit tests between controller and service classes
- Include README as part of your submission

Useful material:
Lecture Notes: Spring_framework, ServletDetails, Unit_testing
Book Chapters: Chapter 12 from “Java for Web Applications”
Github: https://github.com/devdattakulkarni/ModernWebApps/tree/master/Servlets
Following example projects: spring-email-service, test-load-on-startup, mockito-example, servlet-unit-test