SER422 Spring 2018 ONLINE Session B Lab 1
Due 3/19/18 at 11:59:00pm via submission to Blackboard

In this lab you are required to complete 3 tasks. The first asks you to code a simple Java servlet-based application. The second asks you to add stateful behaviors to that application. You must complete this lab individually.

Submission:

For submission, you should submit to Blackboard a single zipfile named lab1_<asurite>.zip that has within it:

- A warfile named lab1-asurite.war. WAR files should not have any source (.java) files in them.
- Your soure tree, meaning the directory (and subdirectory structure) with all of your source (.java) files.
- Other non-code files as needed, including your index.html, web.xml, and build scripts (next). No .class files please! No IDE artifacts or .git directories please!
- You should provide an Ant or Gradle build/deploy file. No matter which you use, I expect a build.xml or build.gradle to be provided with targets “compile”, “build”, “dist”, “clean” and “deploy” (at a minimum). If you use Ant you should be able to customize mine quite easily.
- Do NOT give us a format other than .zip. No .rar, .7z, or .tgz formats. The Java jar utility produces zipfiles (it puts a .jar extension on them that you can rename to .zip, as it uses the zipfile compression format anyway).

Task 1: Simple To-Do list functionality (40%)

Write a complete web application that does the following:

1. (8 pts) Provide a web form at the root of the application context root (http:/localhost[:port]/lab1-asurite/index.html) that asks a user to input:
    a. A name of a "to-do" task
    b. A description of the task
    c. Days of the week the task should be done
    d. An estimate of how long the task will take
    e. at least one additional custom descriptive attribute you decide to add.

    Step c must be multivalued, but you may choose the widget. The other steps must be single line textboxes.

2. (16 pts) The web form must POST to a URL http:/localhost[:port]/lab1-asurite/tasks. The POST must be handled by a servlet, which will receive the information store it in an in-memory data structure (that you design). The browser should display a message stating the operations was successful (or not) followed by a count of the number of tasks currently in the data store, and a hyperlink back to the web form page.
    a. You decide the data structure for storing tasks.
    b. The hyperlink back to the web form (landing) page should not be hardcoded. In other words, do not hardcode (http:/localhost[:port]/lab1-asurite/index.html), you have to figure it out programmatically.
    c. If the task name matches an existing entry in the data store, then you should *replace* the entry in the date store with the updated information.
3. (16 pts) A GET request on URL /tasks lists all the entries in the data store. The GET should take one or more query parameters that filter results by a substring of the description field, substring of your custom attribute, and/or by days of the week.

    Example: GET http:/localhost[:port]/lab1-asurite/tasks?description="happy"&custom="elated"&days=13

    Would return all tasks where the description field had a subset including "happy", AND a custom attribute with a substring "elated", AND happens on Monday OR Wednesday (days are entered 1-7, with Monday=1 to Sunday=7).

    Note that the parameters are AND’d together, while within the days attribute it is an OR if multiple days are specified. Note that if no parameters are given to the GET then the entire set of tasks should be returned. Make certain that the results of GETs are not cached by the browser.

    As in #2, at the bottom of your page should be a hyperlink back to the homepage (that again is not hardcoded).

Task 2: Persist the To-Do list to a file (20%)

Task 1 used an in-memory store of your choice, meaning the list of tasks will start over every time you restart Tomcat. Make it so your To-Do list is persisted to the file system after each new task is added. There is no change to the GET/POST functionality of Task 1, but now you must ensure the data file you use is written after every successful POST, and that the file you use is specified as an init-param inside your web.xml. The file format is up to you. Please name it lab1data.XXX where XXX is a propr extension.

Task 3: Provide custom rendering based on HTTP request headers (30%)

1. Detect if the client (browser) is based on AppleWebKit, and if so change the background color of your web page to gray.
2. Detect if the client (browser) wants plain text as the preferred response. If yes, modify the response payload and approprate response headers to return a plain text rendering of the To-Do list on a GET request

Task 4: Implement appropriate error-handling and provide an appropriate HTTP response (10%)

1. Your servlets should never allow a stacktrace to appear on the screen. You should validate inputs in case of a formatting error and return the appropriate response.
2. Implement proper exception amd error handling of server-side errors and return the appropriate response in such cases.

Extra Credit (15 points)

Provide a second servlet in the same application context that provides the following features:

1. (5) A GET request to http:/localhost[:port]/lab1-asurite/taskadmin returns the number of tasks in the data store as plain text
2. (10) A DELETE request to URL http:/localhost[:port]/lab1-asurite/taskadmin?task=**name** removes the task from the data store if there is a task with the given **name**. Note you do not have to provide an HTML web form for this. You must return a 200-level status code if the task has been removed, and a 404 if there is no such task.

   Note: We mentioned DELETE in the HTTP lecture, but haven't gone over it in code. You will need to look at the servlet API to understand how to do this (though the method you are looking for whould be obvious to you).

Global Constraints:

1. The context name should be /lab1-asurite where asurite is your ASURITE id. You must not hardcode URLs to your server (e.g. localhost) anywhere in the code or HTML, as you can get all of this information via the servlet API.
2. The HTML generated does not have to be fancy, just readable on the browser. This lab is not about aesthetics, it is about learning servlet structure. No Javascript for this lab. You may beautify with CSS if you really want to, but it has no bearing on your grade. No functionality may be implemented on the browser side via style elements, Javscript, or CSS.
3. For this task you are *not* required to show it works on 2 separate Tomcats running behind an httpd server as per the class environment setup. Demonstrating on one Tomcat instance directly is fine. You are required (and always will be) to demonstrate thread-safe coding practices within that single Tomcat instance.