SER598 Spring 2016 Lab5
Assigned 4/11/16, due 4/20/16 at 11:59:00pm via submission to Blackboard

In this lab you are required to complete 2 tasks. The first asks you to add simple stateful behavior to Lab 4. The 2nd asks you to incorporate persistence using the fs module. Extra credit is offered for a 2nd implementation of Task 2 using MongoDB.

Submission:

For submission, you should submit to Blackboard a single zipfile named lab5_<asurite1>.zip (or .jar) that has within it 2 (or 3) zip files (one for each task).

- Please have your web server process listen on port 8081

Task 1: Simple state functionality (70%)

Using your previous lab (Lab 4 in Node), implement the same features from Task 2 of Lab 2 using Node:

Add 3 new features to Lab 4:

1. Create a landing page that remembers who you are (10)

Create a new landing (home) page that recognizes the user if s/he has previously been to the site. If the user has not been to the site then *redirect* the user to a page asking for her/his first and last name, and remember it the next time s/he visits the site. If the user did previously register, then the landing page should display a sorted list of the top 3 matches that are closest to her/his preferences for each of the attributes as described in Lab 4.

2. Make a multi-page web form (25 pts)

Split the webform from Lab 4 #1 into 5 distinct screens – the first 4 asks a distinct step in the Lab 4 workflow, and the last page lists the complete set of answered questions for review with Submit and Cancel buttons. Give the user Prev and Next buttons to navigate backward and forward through the workflow, and pre-populate any HTML form widgets with previously entered values. Note Prev is required on the last page with your Submit and Cancel buttons. If the user Submits or Cancels on the last page make sure the "remembered" widget values are cleared. Upon Submit or Cancel the user should return to the landing page. All forms should use POST.

3. Return the proper response error codes (15)

Verify valid error responses are returned depending on the request. For this step, modify your code to return the proper response code for all negative test cases. For example, what if the GET request has improper query parameters? What if a route that has only the GET verb defined receives a POST request? Note this is NOT the same thing as writing an error message to the response!

Constraints for Task 1:

1. UNLIKE LAB 2, for this task you do not have to persist entries to a file or database (see next task and extra credit). However you still need an in-memory data structure, and should test by using multiple browsers.
2. You must use a combination of browser cookies and sessions to complete this functionality, but it is up to you to decide where each make the most sense.

3. This task also must use multiple routes. It is up to you to determine the best design. That is, you do not have to conform to any specific URLs (no *post_coder* or *get_coders* if you do not want).
4. Constraints 1-3 from Task 1 still apply, except change your servlet context to /ser422/lab2task2. But Task 1, constraint 4 does not. This means you have to handle a) concurrency between Tomcat instances, b) a shared resource (the XML file) between Tomcat instances, and c) ensure you enable "stickiness" so a conversation happens with the proper Tomcat instance.

Task 2: Persist entries to the flat file system (30%):

For this task persist the in-memory data structure from Task 1 to a flat filesystem. Lab 2 asked you to use XML, but I suggest for this lab you use JSON.

Unlike Lab 2, you may hardcode a filesystem location for your storage, though you must tell is in a README.txt file exactly what to change in your code to change the filesystem location.

You still have to worry about concurrent and properly ordered access. That is, consider you synchronous versus asynchronous calls, chaining to preserve read/write ordering, error-handling, file-locking, etc.

EXTRA CREDIT TASK (+35%): Persist entries to MongoDB

Unfortunately we do not have time to cover MongoDB. However it is pretty straightforward as far as CRUD operations are concerned, and it handles JSON easily. For this extra credit task, implement Task 2 using MongoDB.

NOTE: IF YOU CHOOSE TO DO THIS TASK YOU DO **NOT** HAVE TO TO TASK 2 USING THE fs MODULE!

Constraints for the entire lab:

1. Absolutely no Javascript in the front-end or CSS for this lab. This includes AJAX. All functionality must be implemented via HTML and NodeJS (with associated modules).
2. No 3rd party frameworks outside of Express and either Jade or EJS.
3. I have a few slides on MongoDB I will post, but if you choose to do the extra credit you will have to do your own tutorial. You may choose to do Task 2 and the extra credit, but the max credit on this lab is 135% no matter if you do both or just the extra credit.