**SER421 Fall 2021 Lab3 NodeJS Express**

**Objectives**:

1. Gain proficiency in NodeJS/Express
2. Implement the MVC pattern.
3. Apply templating to realize a View layer

**Overview**:

For this lab you will construct a Model-View-Controller application. This application should be a *near-exact replica* of the web application available at *https://swent1linux.asu.edu/simplemvcex*. This application is a simple survey application where a user answers questions on a multi-page survey and can check for closest matches other users have on the survey.

**Constraints on all parts of the application:**

1. What is meant by near-exact is that you should replicate every screen exactly as shown in this lab, and only deviate in those cases below where a deviation is required.
2. This application must be written using the NodeJS/Express framework. Unlike other labs, there are no limitations on external libraries you may use, but there are some constraints on what libraries you MUST use (namely Express, Express-session middleware, and 1 of the 2 templating frameworks as discussed in Part 2).
3. NO Javascript and NO CSS may be used in this lab. This includes inline styling elements in the HTML output.
4. You are required to implement robust error handling. By this I mean you must catch all 4xx and 5xx errors (despite whether this sample application catches them). **This is a 10-point requirement**.
5. Your submission must be in a zipfile named lab3.zip. In the zipfile you may have subdirectories for each part if you want them graded separately, or you may choose to submit a single solution for all parts (the risk with the latter is if something breaks in a one part, we may not be able to fully grade the other parts). For this lab you are free to package your solution as you like, such as just submitting one more .js files and telling us what to npm install and how to run, or by giving us a complete and correct package.json (if you do not know what this is do not do it!). Name your main programs task1.js, task2.js, taskec.js respectively (Please do not put spaces in your file/folder names!)
6. IMPORTANT: Do ***not*** upload your *node\_modules* subdirectory. This is where npm will place all of your 3rd party libraries for express and its dependencies. This will create very large file uploads that could fail. Instead, tell us what to npm install or give us the package.json. As always, a complete Readme is how you tell us!

**Task 1: Replicate the key features of the application in Express (65 points)**

1. (5) The landing page must be at URL http://<server>:3000/landing and replicate the index.jsp page (it does not have to replicate index.jsp as the URL; you may use any URL you like as the landing page, put it in your Readme).
   1. Note this asks for a username but there is no “login” functionality (no password to check).
   2. Do not allow empty usernames or usernames with spaces in them (only allow a combination of uppercase and lowercase letters). You may implement this via HTML5 or the server-side, your choice (NOTE: this is different than the sample app).
2. (45) The survey button takes you to a multi-page delivery of a multi-question survey (one question per page). Note the following:
   1. (5) Each page has a “prev” or “next” button, or just “next” if on the 1st page.
   2. (10) Each page asks a multiple-choice single answer question (MCSA). If the user has visited the page already and selected an answer in this instance (e.g. hits “prev” to go back to the previous page, or hits the “next” button to proceed forward to a page s/he has already visited and answered), then the question should be pre-populated with the previous answer
   3. (10) On each survey question page there is a hyperlink to a page to “Set your rendering preferences”. If the user clicks on this page, then:
      1. Display the preferences page as shown in the sample app.
      2. If the rendering is changed from horizontal to vertical or vice-versa, ensure the manner in which the MCSA answer options are displayed is changed. *This change is in effect for any user on THAT browser!*
      3. After clicking on “OK” return to the same question the user was on but rendered according to the new preference.
      4. If no preference is known for the browser, render the questions horizontally.
   4. (10) The number of questions is not pre-specified; it is determined by a global “lab3survey” variable. We will give you test examples just like we did in lab 1. For this requirement you must ensure your code does not hardcode the number of pages.
   5. (10) Upon completion you get a completion page with a link back to the landing page, pre-populated with the username. Constraint: the user cannot bookmark, or in any way, “back into” a submitted survey. In other words, once a survey has been completed it is immutable (forever completed).
3. (15) If selecting “match”, print out a list of other users and the number of questions matched in sorted order from top to bottom. Note: this means you have to maintain a data structure that stores individual user answers.

**Task 2: Render your pages using a templating engine (25 points)**

Render *all* of your content using one of the EJS or Pug templating engines. This is just refactoring the View layer in your app to use templates. Part of the rubric here will be examining if your code is cleanly structured such that the View layer could be rendered with templates, without templates, or by choosing another template easily.

**Extra Credit (15 points each)**

1. Modify the app to use a persistent store (filesystem) for both the survey, and for user answers to the survey. The default name of the survey file should be survey.json and reside in the root directory of the application. You may organize the user answers in any way you design except they cannot be put *in* *the same survey.json file*. When the application starts back up it should initialize from the appropriate file(s). Think of what this means - survey answers by a user must be associated with the proper survey. For example, suppose there are 2 surveys, “A” and “B”. A has 3 questions, B has 5 questions, and none of the questions appear on both surveys. Now suppose Jim and Sue have both completed both surveys A and B. When the server starts up using survey “A”, then the responses for Jim and Sue to survey “A” should be read in, but the responses to survey “B” are not.
2. Provide an “intelligent assistant” (IA) for completing a survey. The IA should recommend to the user which selection s/he should make for that survey question (e.g. “Psst, you should pick option 2”). The IA may use some intelligence to do so based on user behavior, but I am also fine with you just implementing a random choice function (bounded by the number of available options for that question). Further, upon completion of the survey, your application should print out how many times the user selected a choice the IA recommended on the survey completion page. The constraint here is that you must implement this entirely using your *own middleware*. Depending on your design there may even be more than one middleware. Your middleware should be in a separate file named ia.js and integrated into your application using package management 🡪 require (“ia”).