# SER421Online Lab 4 – Node, MVC & Express Due: 11:59:00pm Thursday, November 10

## LAB 4 may be completed with a single partner, but not one you have partnered with before!

Your assignment is to be submitted via the class website before 11:59:00pm on Thursday the 10th. You should submit a zipfile that includes the source tree for the solution for the functionality described below.

## Assignment Goal: To demonstrate an example of MVC

## e421Match.com: Find your next SER421 lab partner online!

You are asked to develop a web application that determines who should be your next SER421 lab partner based on a series of questions. The site works like this:

1. A user comes and enters her/his name on the site’s landing page in a simple login form.
   1. The login form should be pre-populated with the name of the user who previously took the survey on that browser (if that exists).
   2. We will use the same "fake" authentication logic as lab 3 – if a password is entered the same as the username then the user is authenticated.
2. If the user has previously taken a survey, then after login s/he is taken to a page that displays the list of best partner matches, and is provided a link to go revise her/his survey.
3. If the user is new, or if the pre-existing user clicks the link from step 2, then s/he takes a survey about their 421 preferences.
   1. These questions are provided in a survey.txt file.
   2. The web app should deliver the questions one per each screen.
   3. If the user had already taken a survey, then her/his previous answers should pre-populate the survey question forms.
4. At the conclusion of the survey:
   1. S/he receives a ranked list of users most compatible by comparing her/his answers to the answers of all other user surveys (similar to step 2 for existing users).
   2. The survey results should be persisted in the user store.
5. At any screen the user should be able to logout. If s/he was in the middle of a survey then those survey results are lost. The user is returned to the login page.
6. The app should support a special user "admin". Instead of taking a survey, the admin is provided a different set of tools:
   1. On login, the admin is taken to a page (/tools) that indicates how many users/surveys are known to the system.
   2. A link should be provided for each user to view that user's set of survey responses
   3. A link should be provided to delete that user and her/his survey responses from the system

**Specific assumptions for your implementation**:

1. Use Express routes to provide endpoint behaviors for:
   1. "/login", "/logout"
   2. "/survey/<question number>" where the <question number> is (obviously) the question number. Here you have to check if it is valid that the user be on that survey page. This endpoint should support both GET and POST.
      1. If accessed via GET, it should return the rendered survey question (pre-populated with prior answers as in 3.c if an existing user).
      2. If accessed via POST, it is expected that this represents the submission of a survey question answer from the user, and should be persisted in conversational state.
   3. "/matches" is the endpoint that renders the best partner matches based on survey results (4a).
   4. "/tools" is the endpoint for the admin functionality (6a) and should only be available to the admin.
   5. The additional admin functions should be at URLs and routes you define.
2. For all endpoints you should consider what the appropriate verbs appicable on that endpoint, what request headers should be expected, and what parameters should be there. Some of this is up to your design, others I have defined for you here. No matter the case, the application should provide a suitable error message with proper HTTP response codes at the 4xx level for that error. This includes a catch-all case of a 404 (user gives a URL that is not servicable).
3. Use Expression middleware for sessions to handle conversational state. By the problem description above you should understand where the conversational state is in this application and design accordingly.
4. Implement a session timeout of 2 minutes. That is, if the user does not complete the survey in 2 minutes expire the session and revert the user back to the landing page. There is more than one way to do this!
5. The user store and prior survey answers should be persisted on the server using the fs module. However the format and structure and file organization is entirely up to your design. Likewise, we provide the survey.txt file with the survey questions but you may convert this file to whatever format you wish *as long as it is read in from the filesystem when the program starts*
6. You should use jade or ejs to render all pages except the login (initial) page. You decide which template engine to use.
7. Please use port 3000.
8. You may whatever algorithm you choose to find the best matches, however that algorithm must look at all stored surveys in the persistent store and consider all questions in each stored survey instance.

**Submission instructions**:

1. Submit your solution in a zipfile named <asurite1>[<asurite2>.lab4.zip. This should be your complete source tree.
2. Do not add extra modules beyond those we have discussed in class for MVC with express and associated middleware.