SER421 Lab 3 – Node and Express **Due: 11:59pm, Thursday, October 11**

Activity 1: An article reviewer system (35 points)

In this lab you would be developing a web app that has an article displayed and you would either add comments or delete those comments. For this activity you will be using Express to build your application.
I strongly advise you to go to the node express tutorial here https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/skeleton_website and go through this tutorial before you start this activity. I want you to use pug and I want you to use routes. https://expressjs.com/en/starter/generator.html on this site it explains how you can setup the skeleton for your app already, which will greatly help you in getting started.

R1. The main UI of your application (/) is a page that displays:
   a. A welcome message followed by a section that displays the article content read from a file named article.txt using fs module
   b. You would have another section called "comments" below the article section that displays the comments if they exist (see also later).
   c. A simple web form that contains buttons named "Add Comment ", "Delete Comment", "Undo", "View User Activity" and "Reset"

R2. As inferred by 1, the system should support 4 options: "Add Comment", "Delete Comment", "View User Activity"," Undo"
   a. "Add Comment" button should be part of a web form that takes in the Comment Id and a comment about the article from the user as inputs. It must be implemented as '/add' and should support only POST. Use a persistent store named "comments.json", to store the comments so that it is easy to add and delete comments.
   b. "Delete Comment" button should be part of a web form that takes in the Comment Id from the user as input and deletes the corresponding comment with that id from the comments section. It must be implemented under /delete and must support only POST. If the id does not exists give the user a good response and error message.
   c. "View User Activity" must be implemented using your own version of an underlying stack. Whenever the user presses the add or delete comment button the corresponding activity needs to be tracked and stored in the stack. It must be implemented as /view and must support only GET and it should take you to a page which renders the list of user activities retrieved from the stack. Also see R7 for what you need to store.
   d. "Undo" must be implemented as /undo and support GET or POST. To implement the "undo" operation you have to pop the top operation off the stack and update the comments section.
   e. The response of each of these actions should be a message stating success (or an error) and a link back to the landing page (/) except for "View User Activity" page alone. The "View User Activity" page would display the list of existing user activities from the stack in the success page and must have a link back to the landing page (/).

R3. Implement a 5th option "Reset" that supports only GET at URL endpoint /reset. This operations resets the entire comments section to be empty and empties the stack. Again, the response should be a success or error.

R4. Push is inferred; you are to automatically push when an "add" or" delete" button is clicked; however you do not have an URL to expose Push as an operation to be invoked from the UI.

R5. You should ensure that pages are not cached. Each request to the server should be "new". (Hint: Try learning about the HTTP response headers)

R6. You must validate the URLs, methods and inputs your program receives. If an invalid URL is given, ensure the proper HTTP error code is returned. If an invalid method is given to the URL, ensure the proper HTTP error code is returned. If the input typed into the form is invalid, ensure the proper HTTP error code is returned.

R7. The stack implementation is a history of operations carried out by the user and each row in a stack must include the operation, the operand (if applicable – eg comment), the IP address and the User-agent of each request.

Constraints:

C1. You must use one of Pug or EJS to render each page (will be one of your dependencies).
C2. You have to use express, which should make your life easier of course.
C3. No Javascript or CSS in the browser at all, same as in assign 2.

Note: The article reviewer app is shared; that is there is no requirement based on different users/browsers. All users of the app share the same app state (stack).

Activity 2: Roommate Finder - Find your next roommate online (60 points)

For your activity 2, you would be developing a web application that helps you to find a roommate based on a series of questions. The landing page of the app contains a user input where the user enters his name and a button named "Match" is present. The user after entering the username clicks the "Match" button and then answers a series of questions. After all the questions have been answered, the user finally receives a list of users most compatible by comparing their answers to the answers of all other users. Users may return later and change their answers to get a new list of matched users.

You are required to use Node and Express to develop the application with the following requirements:

R1. The landing page should contain a text box to allow the user to enter a username and then a "match" button which then takes to a list of questions.
R2. Each page after the landing page displays a question and a list of answers with a "Next" and "Prev" button. Each page must be implemented using /question/:id, where id is the question number.
R3. The last page after all the questions would display a list of matched users, where the users are matched based on the number of same answers with other users.
R4. When the user clicks the "prev" button, the answer to the previous question should be preselected and similarly for the "next" button the answer should be preselected if it had been already answered i.e conversational state should be maintained when you move back and forth between the questions.
R5. Include an option in each question page either as a button or hyperlink to select the answer rendering preferences either as vertical list or horizontal list (based on the user preference the answers might be listed either vertically or horizontally).
R6. Implement a one-shot timer that terminates the survey immediately if the user has not completed answering all the questions within 30 seconds.
R7. Delegate the code into separate modules as folders and your main routine file should be named server.js. Also, you should provide a package.json file that specifies all the external dependencies with the targets to install and run the application.
R8. Be sure to implement error handling appropriate to your application (as in Activity 1 R6 above).
R9. You should ensure that the pages are not cached and each request to the server should be "new"

Constraints:

C1. The answers to each question must be a radio button and the answers must be saved to a file store called answers.json using fs module. The questions must be read from a file questions.json (you need to come up with a question file and format).
C2. Use Pug or EJS to render the application
C3. You must use session middleware to accomplish the conversational state features of this app. Do not use cookies, hidden form fields, or URL rewriting for *conversational* state.
C4. No Javascript or CSS in the browser
C5. You should pre-populate survey responses from any browser when re-logging in for a particular user.
C6. The number of questions in the survey should be variable – do NOT hardcode the set of questions within the webapp. They should go with the file, and if the set of questions in the file changes, then the questions in the survey should change.

Activity 3: Adding admin functionality (15 points)

Create admin functionality for activity 2 so that an admin can alter the survey questions (thus changing the question.json file).

R1. Have a button or link on the main page, where the admin can go to an admin login page.
R2. On the login page the admin is asked for a username and password (we again only check if username is equal to password)
R3. A valid login leads the admin to a page where the admin can add questions to the survey.
R4. The admin should have the option to logout on this homepage.

C1. You cannot reach the page to add questions without being logged in as an admin.

C2. You have to come up with a way to check that the user is authenticated to add questions (do not just use the simple cookies we did in lab 2).
C3. Have appropriate error handling as in activity 1 and 2.
C4. The admin cannot take the survey as the regular users can when logged in.

Submission instructions:

1. Submit your solution in a zipfile named asurite1_lab3.zip. In that file should be 2 source trees in 2 subdirectories, /activity1 and /activity2 (activity 2 will include activity 3).
2. Make sure all files are included, view, json files, js files etc
3. Each source tree should have its own readme telling us how to run your application (or anything else you might want to tell us).
4. Please listen on port 8088 in your http servers.
5. Do not add extra modules beyond those we have discussed in class or the videos (e.g. url, querystring, fs, express, middleware) unless you get permission first.
6. No Javascript and CSS in the browser at all in this lab.
7. DO NOT include the “nodes_modules” folder in your submission (we will deduct for that). Only include the package.json with your dependencies which we will use to npm install the dependencies.

Extra Credit for Activity 2 (make sure that everything for the activity work, you will not get extra credit if the main functionality does not work):

EC1. (10) Use async/await to perform I/O asynchronously in Activity 2.
EC2. (10) Use a database as the persistent store for survey answers instead of a flat file. You may use MongoDB. Do not use a ORM framework such as sequelize. If you decide to do this EC then you do not have to implement file persistence for the survey answers part of Activity 2 if you do not wish to (but then you have to get this EC correct to get full credit for Activity 2!). Note that the survey questions must remain file-based.