SER421 Lab 2 Fall 2019 due Sunday 9/23 11:59:00PM

THIS LAB IS TO BE COMPLETED INDIVIDUALLY

The goal of Lab 2 is to get you working in NodeJS, understanding the event loop, the file APIs, and the low-level HTTP module.

Activity 1 (30 points): File I/O

For this activity you will implement a QA service in NodeJS JavaScript code using the "fs" package.
I advice you to work with Git here and if you want GitHub of course. Make sure to keep track of your different versions.
The FAQService is a Q&A management system for a university. Create an FAQ00.js file which you will use for this activity.

Java Script FAQ object:

Start of with implementing an FAQ class. The FAQ class should provide the following features:

- R1. The ability to write a Q&A to the persistent store (sample JSON of the Q&A is provided)
- R2. The ability to update the content (answer text) of an existing Q&A from the existing persistent store.
- R3. The ability to update the tags for a Q&A from the existing persistent store.
- R4. The ability to delete a Q&A from an existing persistent store.
- R5. The ability to return a collection of Q&As based on a filter, where the filter checks for one or more of the criteria such as:
    a. Author
    b. Date Range (start, end)
    c. Tags

Constraints on the Q&A service:

- C1. The class should prevent concurrent read/write problems (e.g. lost updates).
- C2. A Q&A is described by the following attributes: question, answer, tags, author, date and id. You can decide how you want to create a unique id (they can look different than the one in the example)
- C3. The persistent store for the Q&A service must be a JSON file named QA.json and in the same directory as your top-level executable code. We will provide an example for you, and you are free to generate and share additional test cases.

Overall Constraints:

- C4. Put your code in 1 file, FAQ00.js.
- C5. You do not have to write a user interface, but you do have to specify the API for your object types in your README.txt, and provide example starting files and a sample test case of each service. That is, you must provide examples of how to instantiate your service object, and how to invoke it so we can test it manually.
- C6. Your code must be clear and well-written.
- C7. Use the synchronous file I/O features in NodeJS (it makes C1 MUCH easier).

This is for now just a normal Java Script like you have done before with the calculator. I advise you to first get this running, be able to read and write and change the JSON file. Create methods and test them throughout.

Save this as FAQ00.js create a simple Readme telling us how to run R1-R5. Please include test for your class so we can test it as well.

I want this exact version submitted to see that this works and we can grade you on this part alone.

Activity 2 (30 points): Implement a simple FAQ service

For this activity you will implement a web application leveraging your solution from Activity 1. In theory, you should be able to import the solution from the first Activity, for now I would strongly advise you to not import it into your new JavaScript and do that later.
Start a new JavaScript called FAQService00.js.

Getting Started:

In here you should start setting up a server on Port 3000. This server should be able to receive requests from the specified port. I advise you to play around with some POST and GET requests here to make sure you understand what is happening when and how to read the requests and how to respond. EG. get a request, print a simple response, etc.
When you figured this out then continue (do not try to do everything at once if you are not familiar with all this yet).

Background (will be needed later):

There are 2 roles in the system: "Instructor" and "Student". Here is what each can do:

a. "Instructor" can post/create a Q&A, edit and delete it (based on Activity 1). Also, the instructor should be provided with a feature to search for Q&A based on the tags.
b. "Student" can search for the Q&A based on the tags, author and date range.

Login/Logout (use HTTP POST):

Start with the login and use this to get to know how to handle POST and GET requests, change the displayed content on the page etc.

R1. On the login page (create one), provide an HTML form asking for a username, a password, and a radio button that selects the Role (default to Student). Your login logic:
   a. We will fudge authentication by simple testing that the username and passwords match (obviously not how we really want to do it!). If the login fails, display a page indicating failure with a link back to the landing page (or just display an error message on the login page – see my video).
   b. If login is successful take the user to the "View Q&A" (home) page with a welcome message.

R2. All screens for a logged in user should indicate the username and role, and provide a "Logout" link that logs the user out and returns the user to the login page with a good bye message and the user could login again right away (you can also have a separate logout page with a link back to the login).

R3. When the user returns to the landing page of the application in their browser, the application should remember what username was last used on that browser, and their role. The landing page should say "Welcome [Instructor/Student] <username>, please enter your password", and have the username pre-filled in the textbox for entering a username. (In my video I did not put the "Welcome back" message, I do not want to re-record for that though).

R4. The user should not be able to go to your home page without being logged in. If the user tries to enter a URL manually then redirect to the login (for non logged in users) or to the home page (for logged in users).

I want this as FAQService00.js in your submission (this should work without Activity 1).

Activity 3 (50 points):

Now we are getting to the features and putting it all together.
Use the two previous scripts as basis to create a FAQ.js and FAQService.js script (copies from your 00 versions). FAQ.js should be included in your FAQService (check out export and require). In theory, you should not need to change anything in the Activity 1 script but maybe you do, that is why I want you to submit these separate (just to be able to give partial credit if things do not work out). So, you do not touch the 00 versions anymore.

In case you needed to change the FAQ.js to make this activity 3 work, I want you to mark the changes and explain why the change was needed.

Features (use HTTP GET or POST as you feel is best):

R5. A View Q&A (home) page will display all questions and answers (and tags, author, date) for all users (this is the home page users get to see after logging in). This page should do the following:
   a. If the user is the instructor then the questions should be a hyperlink that allows the instructor to edit the answer and tags (the question should just be displayed). When clicked then R7 from below should happen. Also Add and Delete options should be displayed (button or links) (see R6)
   b. If the user in her/his role is not allowed to create/edit/delete a Q&A ("Student"), then only display the Q&As (not hyperlinked) no buttons/links.

R6. If the Role is Instructor then the home Q&A page should have a link at the top for Creating a new question and answer (Q&A). If clicked, an HTML form should be displayed allowing the Instructor to create a new Q&A with all of the fields (question, answer, tags – author and date will be provided from the server) with "Save" and "Cancel" options to the form. Upon "Save", persist the Q&A via the Q&AService.
   a. If the save fails, stay on the Create Q&A page and provide an informative error message.
   b. If the save succeeds, then return the user to an updated View Q&A page (updated means it should contain the new QA that was added)

R7. Editing a question after the question link has been clicked. Render the corresponding Q&A that was hyperlinked from the Q&A page in an HTML-friendly format that you design. We do not grade on aesthetics, and no Javascript is allowed. It just has to be complete (all fields that can be edited) and readable. The original answer and tags should be pre-filled. Each Q&A structure is defined in Activity 1 Constraint C2. Provide a save and cancel button. In case the update fails, provide an error message and go back to the landing page.

R8. If the user is the Instructor, then provide a link or button that allows the Instructor to Delete the corresponding Q&A. If Delete, then invoke the delete functionality from Activity 2 and return to the landing page. You could also choose to give an Edit and Delete option after the user clicked the hyperlink, that is up to you as long as both things can be done.

R9. Any user (who is logged in) should be provided a link at the top for "Search". You should also include fields for the user to put the author, tags and dates in so you can filter based on them when the user clicks "search". Upon clicking of the search button the page should display the search results on the same page.

R10. A user should not be able to go to any page directly without logging in first. That is, you should not allow "bookmarkable" URLs; if the user attempts to go directly to a page via the browser bar then redirect to login page.

R11. Error-checking:
   a. You must decide the most appropriate HTTP method to use for any link or form (GET or POST). Your code should use only the proper method for a given action. If the wrong method is used, you should return the proper HTTP response code (you should look this up!).
   b. Any unknown URL presented to the web server should return the proper response code.

Constraints:

C1. No Javascript in the browser whatsoever. CSS may only be used for embedded styling (no external links) and may not be used to satisfy any functional requirements (i.e. no "hiding" content via CSS to accomplish role differentiation).
C2. No 3rd party packages or libraries unless we have discussed them in class. If you have a question on it then ask before you just go use something!
C3. URLs: the landing page should be at the root URL (/). You must run on port 3000.
C4. End users should never see stacktraces or other “developer” errors. Trap all errors you can think of and present user-friendly error messages and directions for recovering to the user.
C5. You should try to catch as many errors as you can think of, so the program does not crash. Again, I would rather have less functionality that works correctly than all the functionality and they only work half the time.