### SER421 Online: Web Applications and Mobile Systems Fall 2016B

### Lab 3, due Thursday, 11/3/16 at 11:59pm via online submission to Blackboard

***You MAY do this lab in pairs but you are required to participate and understand all parts of the solution.***

The goal of Lab 3 is to get you working in NodeJS, understanding the event queue, the file APIs, and the low-level HTTP module. Submission instructions are at the end, PLEASE FOLLOW SUBMISSION INSTRUCTIONS!

**Activity 1 (60 points): Implement a simple Blog delivery service**

For this activity you will implement a simple Blog delivery service. Requirements:

Roles:

1. For this Activity, there are 2 roles in the system: “Visitor” and “Reviewer”. Here is what each can do:
   1. Visitors can read any public Blog article, and become a Reviewer
   2. Reviewers can read any Blog article, public or private
   3. The application should recognize a user who has created a profile before. A profile consists of a username and a role.

Auth/Quit (use HTTP POST):

1. For this lab we will “fudge” authentication. On any Visitor page, provide a “Auth” link that presents a username and password HTML form. The Auth page should be at /auth. If the 2 fields match, then log that user in as a Reviewer. Obviously this is not how real authentication should work!
2. All screens for a logged in user (by definition, a Reviewer) should indicate the user’s name, the user’s role, and provide a “Quit” link that logs the user out and returns the user to the landing page as a Visitor.
3. When the user first goes to the landing page of the application in their browser, they should be automatically logged in as a Reviewer if s/he had last used the app as that Reviewer. (This is a restatement of requirement 1c).

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

1. A View Blog (the landing) page will display the titles of all Blog articles for all users. This page should do the following:
   1. If the user in her/his present role can View the article, then the title should be a hyperlink to that article.
   2. If the user in her/his present role is not allowed to view the article, then only display the title of the article (not hyperlinked).
2. The system should provide common header and footer text/images for all pages. Be creative. These headers and footers should be stored in their own HTML content files on the server and delivered as part of the server response.
3. Blog articles are aggregations of HTML content fragments stored in individual files under the Blog directory. An article file is a JSON file with the name <id>.art with the following structure:
   1. Title: <title>
   2. Author: <author>
   3. Public: Yes/No
   4. Fragments: [array of .html file references]

You may design your own identification scheme for the <id>s. Example .art files are provided on Blackboard. When the user accesses the .art file, then (if permitted by role), the application should display the aggregated content *header, frag1, frag2, …, footer* – you are basically concatenating the content of the files.

1. The media, and Blog directories under your NodeJS server should all be accessible via a direct URL. However, URLs to /<id>.art must be parsed and handled by the server. And direct access to an article by an unauthorized user role must be handled with an appropriate error message returned (what would be appropriate? – think HTTP Primer module).

Non-functional requirements:

1. The HTML files stored on the server filesystem under a subdirectory named "blog" relative to the root of your application. If an HTML fragment refers to an image or other media those files should be in a subdirectory media.
2. **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).
3. **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! ***This means you must use the low-level HTTP module****, not a module like Express or HapiJS or other!*
4. You may use synchronous file I/O APIs, and you will find they are much easier (even though we talked about how these are "bad" in the NodeJS asynchronous paradigm). See the Extra Credit section for extra points if you use asynchronous I/O.
5. URLs: the landing page should be at the root URL (/). You should run on port 3000. Again you should serve static files from direct URLs (like /blog/frag1.html or /media/redsox.jpg).
6. I expect basic good programming practices – comments, proper indenting, no global variables (unless required), etc. You have learned these things since your 100-level classes and I reserve the right to deduct points, at my discretion, for submissions of poorly constructed code unbefitting a senior-level class.

**Activity 2 (40 points): Provide an Author role**

For this activity you will extend Activity 1 by creating an Author role. Authors Auth/Quit to the system in the same way as Reviewers, though now you will have to provide a new UI widget to indicate whether the user is an Author or a Reviewer. All other aspects of Activity 1 Feature set "Auth/Quit" above are naturally extended.

Additional features for Authors:

1. Authors should have a hyperlink in the page navigation to Create a new article. The link should take them to an Add article page, where the Author should be able to construct a .art file as described in Activity 1, Feature 7. However, the Author field of the .art file should automatically be filled in by the authenticated username.
2. Authors should have a hyperlink on the landing page to create a new HTML article fragment in the /blog directory. The Author should be taken to a screen with a form that allows textarea entry of the HTML fragment and gives a name.
3. The View Blog (landing) page should be modified to allow Authors to see a hyperlinked list of all Public Blog articles (just as the Visitor sees) plus the Private Blog articles s/he has authored (but not all of them like Reviewer).
4. Further, for the Private articles the Author has authored, there should a hyperlink to "delete" the article. If "delete" is chosen the user is asked to confirm (on a separate page) and if confirmed, deletes the article file.

Non-functional requirements:

1. The NFRs from Activity 1 still apply to this Activity.
2. The construction of the article file must be done via a visual UI, not by providing a textarea and hand-entering the JSON (though you may want to start that way for testing purposes).

**EXTRA CREDIT (Note it is possible for an individual on the team to do the extra credit alone)**:

1. (15 points) Activity 2 feature 2.b. provides a delete feature. However the delete does not cascade and remove HTML and media files that are no longer referenced by any article. Add the capability to detect if individual files may be safely deleted, and if so ask the user on a per file basis if s/he wants the file deleted.
2. (35 points) I indicate in Activity 1 that you should use synchronous File I/O, even though asynchronous File I/O is the Node best practice for reasons we discuss in the videos. Convert ALL of your synchronous File I/O to Asynchronous I/O.

**SUBMISSION INSTRUCTIONS (READ CAREFULLY and ASK QUESTIONS!):**

1. Create a zipfile named <asurite1>\_421Lab3.zip where <asurite> is your ASURITE id (use <asurite1>\_<asurite2>\_421Lab3.zip if in a pairing). If you partner, we only need one submission between the 2 students; the only exception is if you did the extra credit yourself. Please use the Readme.txt to indicate who is in the pairing is and if only one person did an extra credit part.
2. The zipfile should have a root folder with the NodeJS code and article files, and subdirectories , media and blog. You may populate whatever test cases you like in these directories with your code, but also realize we will be copying in our own test data.
3. In each folder you are welcome to include a file named README.txt with any information you want us to know. For example, if you only completed some of the parts of a multipart activity, you can indicate that and what is undone (we consider partial credit).
4. I strongly suggest, especially on programming problems, that you get a stable solution to a Part, save it, and then move on. We grade in parts. You can also give us a partially done Part which we will evaluate for partial credit, but your README.txt has to tell us that! Finally, we allow as many submissions as you want to do, we only grade the last one!
5. Remember, if you pair my expectation is that you are pair programming, not divide-and-conquer. If I find evidence to the contrary I reserve the right to give you half credit or an additional quiz or both!

**Hints and Guidance on Grading:**

* The grading criteria I employ awards points for the successful implementation of features (observable and testable functionality), and deduct points for violation of non-functional requirements. Further, if you violate a NFR to implement a testable feature, then I will not award credit for that feature and deduct (typically 10 points) for violation of the NFR. So for example, if you implement authentication (1.2) by using Javascript in the browser, you will not only not earn the points for that feature but you will lose at least 10 points for violating the NFR! So when I say no Javascript in the browser and only use low-level HTTP, I really mean it!
* I strongly advise starting this program by writing a command-line version using console.log to do the file concatenation part first, and work yor way back to .art files and so forth. Thinking of "web as adapter" as discussed in the videos, and foreshadowing proper separation of concerns, I really suggest starting this way. Helps with testing too!
* Functional programming lends itself very very well to "test a little, code a little"!
* I am serious about not giving me poorly constructed code. You are at a level of training and maturity where handing something in hacked at 11:58pm is a poor process. Start early, finish early, and allocate time to refactor and clean. Be proud of the code you submit!
* I hope you have watched the video on course policies regarding academic integrity, ethics (working in pairs), and late policies. I believe I was very clear in this material – written and video. If you haven't yet, no time like the present!