SER421: Web Applications and Mobile Systems Fall 2020 Lab 5

The goal of Lab 5 is to have you "put it all together", creating a server-side API and a client-side (near) SPA that talks with the API via AJAX/Fetch.

IN A NUTSHELL: This assignment is a repeat of Lab 2, except you are creating an HTTP-based API for Lab 2 Activity 1 (the internal API), and changing Lab 2 Activity 2 from a server-driven web application to a client-driven application. For this reason, you will see a lot of the same text from Lab 2, augmented with changes/additions in maroon to highlight what is new (cherry red remains the points awarded as in Lab 2; if you cannot see the colors well please let me know).

Activity 1 (30 points): NewsService

For this activity you will implement a news service in NodeJS Javascript code. *The set of requirements R1-R6 below is exactly the same as Lab 2 Activity 1. However, the difference is this time you will need to use an endpoint (URL) and accept input data via HTTP as specified below in maroon. You will need to decide whether to use GET, POST, PUT, PATCH, or DELETE, and what the format of the input data should be. Good principles of design will be part of the grading criteria for R1-R5 and the new additional requirement.*

NewsService Requirements

The NewsService is an object representing a content management system for a hypothetical news organization. The NewsService should provide these features: *R1-R5 are worth 4 points*

R1. The ability to write a new news story to the persistent store. *You will need the author and the story attributes (C2). This should be at endpoint /create*

R2. The ability to update the headline of an existing news story. *You will need the author and new headline (title) and whatever you consider identifying information to know what headline to change. This should be at endpoint /editTitle*

R3. The ability to change the content of an existing news story. *You will need the author and new content (title) and whatever you consider identifying information to know what story content to change. This should be at endpoint /editContent*

R4. The ability to delete an existing news story. *You will need the identifying information of a story to know what to delete. This should be at endpoint /delete*

R5. The ability to return a collection of NewsStory objects based on a filter, where the filter checks for one or more of:
   a. Substring of the title
   b. Date Range
   c. Author

   You will need to determine how to accept input for each of these query parameters. This should be at endpoint /search

R6. Construct 10 total test cases in a file named NewsService_test.txt or .json that exercises each one of R1-R5. *Guidance:*
   a. For R1, simply write one new story (1 test case)
   b. For R2-R4, perform an update headline, change content, or delete on an existing news item. Then do a 4th one where you are attempting to perform any one of these operations on a non-existing new item. (4 test cases)
   c. For R5, 3 test cases, one per each of a-c. 1 test case for *a* and *c*. 1 test case for *a, b*, and *c*.

For this version of the lab, your test cases should be either cURL requests, or Postman collection exports. Postman is a popular tool for talking directly to web servers; you can construct HTTP requests of all kinds within it. A collection is simply a group of related requests (in this case 10 requests). You may use it and export a v2.1 collection JSON. Alternatively, you can simply give us the cURL command-line commands ready-to-execute for your 10 test cases.

½ point per valid test case for 5 points.

Additional Requirement in Lab 5: Provide two new endpoints: /login and /logout. /login accepts a username and password and checks that they match. If successful, it returns a "secret" (a token or word) that must be sent on every subsequent request to verify the user is authenticated (4 points). /logout simply invalidates the token (1 point). No test cases are required for these endpoints.

Constraints/Non-Functional Requirements (NFRs) on the NewsService:

C1. ~~IMPORTANT: Your "persistence store" here is an internal (in-memory) data structure that you design. You determine the data/object structure to use. There is no File I/O required (but see the Extra Credit 1).~~ *THIS CONSTRAINT IS NO LONGER VALID. YOU MUST HAVE A FILE-BASED PERSISTENCE STORE AS PER EC 1!!!*

C2. A NewsStory is a type of object that represents a news article and is described by the following attributes: author (a username), title, public flag, story content, and date.

Overall Constraints:

C3. Put your code in 1 file, NewsService*API*.js. There is no need for modules, though you may use a module.exports if you wish (and document in the README). Include your test cases as NewsService_test.*json|txt*. If you have additional test cases merely include them in the zipfile and describe in your README.

C4. IMPORTANT: You do not write a user interface, you specify the API your README.txt. *You may optionally include an HTML specification of your API, there are lots of tools out there that can auto-generate this from comments in your code.*

C5. Your code must be clear and well-written.

C6. *(NEW) Your auth token design is up to you, but it must be a hash including a timestamp (non-repeatable) (it cannot be as naïve as the username). The same username should be allowed to acquire tokens on different browsers at the same time.*

Additional information: We will provide a solution for Lab 2 Activity 1 in case you are not comfortable with using your solution. Also, there is NO requirement to use just Node HTTP. You may use Express or any other middleware or 3rd party packages you find on npmjs.org. No special permission is required.

Activity 2 (50 points): Implement a simple News delivery application using AJAX/Fetch

For this activity you will implement a *client-side* web application leveraging your solution from Activity 1.

Background: *The Background is the same as before.*

There are 3 roles in the system: "Author", "Guest", and "Subscriber". Here is what each can do:

a. Guests can read any public news story
b. Subscribers can read any news story, public or private
c. Authors can read any public story, and private stories that s/he has authored

Login/Logout (use HTTP POST): **(6 points)**

R7. On the landing (home) page (/), provide an HTML form asking for a username, a password, and a radio button that selects the Role (default to Guest). *Note: your landing page must be delivered by the server.* Your login logic: *3 points*
   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. *You will do this by accessing the /login endpoint from Activity 1. You will get back a token that you will need to use on all subsequent requests. Do NOT perform the login logic within the Javascript on the browser!*
   b. If login is successful take the user to the "View News" page (R10 with no filter by default). *This is expected to be a true full page load, which you may accomplish in whatever manner you wish.*
   c. If you are not logged in this is the only page you can see! *This is another way of stating that all views except this one require a valid token.*

R8. All screens for a logged in user should indicate the username and role, and provide a "Logout" link that logs the user out *(expires the token using the /logout endpoint in Activity 1)* and returns the user to the landing page. *3 points*

R9. ~~When the user returns to the landing page of the application in their browser, the application should remember what username that was last used *on that browser*, and their role. The landing page should say "Welcome [Guest/Author/Subscriber] <username>, please enter your password", and have the username pre-filled in the textbox for entering a username. *4 points, and this should be done via a specific mechanism (you need to figure this out) discussed in the class. If done any other way we will deduct 2 points.*~~ *R9 is not required for Lab 5.*

Special Notes on Activity 2 Features: The features below are the exact same features from Lab 2 Activity 2, I have not changed anything except for R15. For Lab 5 however, you need to use AJAX and/or Fetch API calls to accomplish tasks, DOM manipulation to change between views (no full page loads other than what is above), and possibly events to know when DOM changes complete.

Features (use HTTP GET or POST as you feel is best): **(44 points)**

R10. A View News ("sub" landing) page will display the titles of all news stories for all users. This page should do the following:
   a. If the user in her/his role can View the story, then the title should be a hyperlink to that story (invokes R12).
   b. If the user in her/his role is not allowed to view the story, then only display the title of the story (not hyperlinked).
   Based on the role-based requirements above. You have to test this requirement, both parts, for all roles:
   - *For a Guest, only Public stories should be hyperlinked and viewable (3 pts), and non-public only title (3 pts)*
   - *For an Author all Public plus stories that Author has authored should be hyperlinked and viewable (3 pts), and non-public/non-authored only title (3 pts)*
   - *For a Subscriber, all stories should be hyperlinked and viewable (3 pts)*

R11. If the Role is Author then the View News page should have a link at the top for Creating a new news story. If clicked, an HTML form should be displayed allowing the Author to create a new news story with all of the fields, with "Save" and "Cancel" options to the form. Upon "Save", persist the story via the NewsService. *(6 points)*
   a. If the save fails, re-display the Create Story page and provide an informative error message
   b. If the save succeeds, then return the user to an updated View News page (R10)

R12. Render news stories that are hyperlinked from the View News 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 of a news story) and readable. News stories are provided by the NewsService. Each story structure is defined in Activity 1 Constraint C2. *(6 pts).*

R13. If the user is the Author of the story being viewed, then provide a link on the page rendering the story that allows the Author to Delete the story *(6 pts)*
   a. If Delete is selected, invoke the delete functionality from Activity 1 R4 and return to the View News page
   b. If the Delete fails for some reason, re-display the story with an error message at the top.

R14. 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 the landing page. *(4 points)*

R15. Error-checking:
   a. You must ~~decide~~ use the ~~most~~ appropriate HTTP method to use for any link or form *as dictated by your Activity 1 API endpoint you are accessing. Note hyperlinks can be modified to invoke a JS function via onclick. 2 pts*
   ~~b. Any unknown URL presented to the web server should return the proper response code. *2 pts*~~
   ~~c.~~ *Your client side code must be able to handle HTTP response code errors from the API in a way that allows the user to recover (nature of error, ability to retry/correct any user errors). 5 points.*

Constraints (points shown are deductions; you do not "earn" points for adhering to constraints:

The Lab 2 constraints no longer make sense for Activity 2 in Lab 5. Constraints for the client-side app:

1. *No 3rd party client-side Javascript frameworks. But you may now use CSS!*
2. *You may use a client-side templating framework if you think it will help. But you still must be performing plain-old DOM manipulation using the APIs from your notes and in the class examples. This is only for the experienced.*

EXTRA CREDIT – these are new to Lab 5 (19 points):

EC1. (10 pts) You will notice Activity 2 still does not require edit features that utilize APIs in Activity 1 R2 and R3. Extend your Activity 2 solution to include features to edit a headline and edit a story (5 points each).

EC2. (9 pts) Note that with our login/logout functionality, if a user never logs out the server does not know – there is no effective timeout or other way to expire the token unilaterally on the server. Implement, in Activity 1, a timeout where if the token is not utilized in a request in the past 3 minutes, then that token expires (is no longer accepted as valid by the server).

SUBMISSION INSTRUCTIONS (READ CAREFULLY and ASK QUESTIONS!):

1. Create a zipfile named <asurite>_421Lab5.zip where <asurite> is your ASURITE id. No RAR or 7zip files!
2. The zipfile should have a root folder with the NodeJS code and related files, plus the client-side app and the test cases. You may organize the directories how you wish, and may or may not use modules and package management (npm). Your README should explain exactly what is required to run your code, and you should test this process!
3. If you do choose to use npm (ask us to do an npm install), please do NOT submit your node_modules directory you generate. This makes your submission very large, and we want to rely on your package(-lock).json.
4. I strongly suggest, especially on programming problems, that you get a stable solution to a part, save it, and then move on. We do give partial credit. We allow as many submissions as you want to do and only grade the latest! For this lab this means you may choose to Activity 1 and then Activity 2, or vice-versa (mocking out the API on the client), or you may choose to do each requirement of Activity 1 and then do the corresponding requirement of Activity 2. In any case, if you turn in something where nothing works due to even one defect, you risk losing ALL the credit! Turn in what is **stable!**
5. If you decide to do either EC, please document in the Readme, and indicate if it is in the regular submission or if you chose to implement it in a new directory. For this lab you could probably just include it in your regular solution, but I leave that up to you.