SER421: Web Applications Fall 2021 Lab 2

The goal of Lab 2 is to get you working in NodeJS, and some basic HTML5.

Important: Do not use any Javascript in the browser in this assignment. You may use CSS if you wish but it cannot be used to implement any of the functionality, it may only be used for aesthetics (we do not grade you on aesthetics however, so to be safe we recommend simply not using any CSS or JS in the browser at all).

Background:
For this lab you will replicate the task 1 server-side program you used with Postman. That program was written in Java using servlet technology, you will replicate, and in specific places enhance, that application. You may assume your solution should work *exactly* like that application except in places explicitly specified below.

Task 1 (30 points): HTML5-ize the application
The index.html page of the grocery list is a simple HTML form. Grab that form (do a simple "view source" and copy/paste it), and do the following:

1. Put the following constraints on the form fields: *(8 points, 4 x 2)*
   a. Product Name must start with a capital letter and must be at least 5 letters long with no numbers of special characters
   b. Product Brand cannot be longer than 10 characters but may have numbers or special characters
   c. Quantity must be a number between 1 and 12
   d. All form fields except Diet must be required
2. Convert aisle to a single-choice dropdown between 2 and 20 *(2 pts)*
3. Convert diet to a checkbox group with options for "Keto", "Mediterranean", "Vegan", "South Beach", "Atkins". Note as a checkbox this now means your input may be multivalued. Also, you should allow for no option to be selected as well (this value is not required). *(2 pts)*
4. Add an optional form field that is named "Delivery date" that is a datetime-local field that allows a value from 9/16/2021 at 9am to 10/12/2021 at 7pm. *(2 pts)*
5. Use a header to replace "Lab 1 Task Solution" with "Lab 2 Solution" *(1 pt)*
6. Use a footer to add your name and ASURITE id at the bottom of the page. *(1 pt)*

1-6 sum to 16 points. 4 points for having the right form parameters (action, method). The total for the 1st form is 20 points

Additionally, add a second form that provides an interface to make the queries to display groceries you made in lab 1. This form should have 2 fields, *Aisle* and *Diet*. Both of these should be displayed the exact same way as the requirements above (2 and 3), but neither is required. When submitting this second form you are making a very similar query as you did to *my_groceries* in lab 1.
The second form is worth 10 points. 4 points for having the right form parameters (action, method). 3 points each for the 2 form fields.

Submission: These HTML forms go on your index.html page that is in your Task 2 submission in this lab.

Task 2 (50 points): Replicate the server-side application in NodeJS
For this task you should consider the Lab 1 Task 1 application I hosted as your test "oracle", with the following changes:

1. Diet is now a multi-valued input so you must process those multiple values and store them (if diet is given)
2. You must process and store the new Delivery date (if one is given)
3. You need to figure out how to display Diet and Delivery Date as part of a search query result (the my_groceries page). I leave that up to you – aesthetics do not matter, but it needs to be readable.
4. You will need to process the second form to make a query to my_groceries. The specification is slightly altered from lab 1 since *diet* is now multivalued. You should consider multiple diet values as *OR'd* together. For example, if the user checks "Keto" and "Atkins" then you search for products that were stored with *either* Keto *or* Atkins *or* both.
5. Error-handling:
   a. The server should know if an unknown URL is given and what response code to give back
   b. The server should know if there is a syntax error on a request and return the proper response code
   c. The server should know if the wrong method is given (GET or POST) and return the proper response code
   d. If the server errors (throws an exception or other program fault) then the proper response code should be given
6. Make sure your application can support the queries from the lab 1 task 1 requirements, modified where appropriate according to these specifications. Specifically, ensure you can accept inputs in multiple formats and displaying output in various formats see Lab 1 Task 1 requirements 3, 5, and 6

For grading lab 2, consider the requests that the server must handle. 1) index.html, 2) adding a grocery, 3) querying groceries. The grade distribution for these is 1) 8 points, 2) 21 points, 3) 21 points. I cannot give you the point-by-point breakdown since that will give away part of the solution that you need to figure out how to implement. But some guidance based on our pattern; we will look for:

- *Is the server handling the request based on the proper combination of URL and method (routing, or step 3)*
- *Is the server handling request query strings and/or payloads properly (steps 1 & 2)*
- *Is the proper payload assembled?*
- *Are request headers set appropriately (status, content-type, etc.)?*
- *Are error cases handled (400, 404, 405 request errors, 500-level errors)*
- *Functionally, we will check against the existing application functions from lab 1 with the modifications to some parameters (and their processing) as required*

Constraints:

- "store" here simply storing them in an in-memory data structure like an array of objects, not reading/writing to a file (see EC)
- Host your Node HTTP server on localhost with port 3000. Use relative URLs everywhere you can.
- You must only use the node http, url, and related base modules discuss in class. Do not use Express or another framework.

Submission: You should zip up your source tree with all files and name it lab2task2.zip. Your node main file should be named "task2server.js" and we should be able to run it by simply doing "node task2server.js" on our terminal command-line

Task 3 (20 points): Add some advanced features

Right now this is not really a multi-user app. As you know from making your queries in Lab 1, everyone is writing to the same grocery list. Let's add a feature so a user can mark her/his favorites among grocery products:

On the *my_groceries* output, you can see it is outputting the list of grocery products in a table (which you will modify as specified in Task 2 #3). Add a new leftmost (first) column that is a checkbox (you will put the table inside a form) and title that column "Favorites". Do the following:

1. If the user selects the product as a favorite, then when submitting the server will set a cookie indicating what was selected and re-render the same my_groceries page being displayed. The cookie should persist for 10 minutes.
2. Add a new query string Boolean parameter for my_groceries named "favorites". If true then only the products previously selected as favorites should be shown.
3. To accommodate #2, you should add a "Favorites only?" toggle box/button/you decide on the index.html page. If selected, then you should execute the query using only favorites.

Points:

- *Modifying the output of Task 2 #3 (4 points)*
- *#1 Proper setting of the cookie with all proper attributes (8 points)*
- *#2 Adding the new query string parameter and processing that as part of the query functionality (6 points)*
- *#3 Adding a "Favorites only?" toggle form option (#3 is related to #2) (2 points)*

Hint: The most common error students make when coding with cookies the first time is not testing their code with/without cookies set. Make sure you clear cookies and run clean tests to ensure your code does not break!

Submission: I do **not** recommend combining task 2 and task 3. Instead create a lab2task3.zip to submit with all source files and run a similar way ("node task3server.js")

EXTRA CREDIT (25 points):

EC1. (5 pts) Store and read the grocery list to a file using File I/O with the 'fs' package. You may choose between synchronous and asynchronous I/O as per your preference (IMHO synchronous is way easier). You may use features we have not discussed yet, namely promises and/or async (if you don't know what they are then I wouldn't try this route), or you may simply deal with callbacks, your choice.

EC2. (10 pts) EC1 as written is naïve; it just says to read and write the grocery list from/to disk. But as a web developer you cannot and should not assume a single user environment – assume it is running out on a server like lab 1 task 1. Will your EC1 scale? Will your EC1 be correct (not suffer from lost updates or other concurrency issues)? Enhance your EC1 solution to:

   a. Work correctly in the face of overlapping read/write requests
   b. Work efficiently in the face of overlapping read/write requests

   This is as much a design question as a programming question. You have tradeoffs. How often should you update the file? What are the overlapping request scenarios like in Node's single-threaded event queue model? Please include a readme.txt that explains how you solved these design problems.

EC3. (10 pts) Consider the Delivery Date value that is new in this lab. When a Delivery Date day and time passes, remove the product from the grocery list. This must be done using Node events.

Grading: We don't usually do point breakdowns for ECs. You either "get it" or you don't. However there will be some partial credit for #2 based on how you address a and b (these are almost rubrics).

Submission: Indicate in a readme.txt which extra credits you have attempted, what you have named the files and how to run them (and any other constraints like where you are reading/writing files). Name the submitted files lab2EC.zip

We want to emphasize that you should provide a Readme.txt or Readme.md (if you prefer markdown) in the root directory of your task submissions that tells us anything we need to know. Perhaps special instruction to run. Or maybe you know you have a bug you could not quite figure out. We will read your Readme before running your code.