SER421: Web Applications and Mobile Systems Fall 2019

Lab 4

The goal of Lab 4 is to get you working in a browser-based application, using various techniques such as DOM manipulation, event handling, and storage. Submission instructions are at the end, PLEASE FOLLOW SUBMISSION INSTRUCTIONS!

Activity 1: DOM expressions (20 points)

Go to www.bing.com and perform a search using 3 distinct words of your choosing. Save the resulting page (save the complete page, not just the "source"). Load the page you just saved locally (Open File…<file you just saved>).

Now, for this activity, write DOM expressions that do the following:

1. (3) Output to the console the <ol> element encompassing the results of the search
2. (4) Output to the console the code for the "onload" event on the <body> element
3. (3) Output to the console the 2nd child node underneath the <body> element
4. (3) Output to the console the number of <h2> tags in the page
5. (3) Output to the console the value in the search bar (you must get this from the search bar not anywhere else on the page)
6. (4) Make the "Make Bing your search engine" text in the upper right corner result go away

Activity 2: Implement a "forum" ON THE CLIENT (35 points)

This activity is about creating a replica of a typical "forum" webpage. The web application would have some article or review about a movie and then the user would be prompted to enter comments or their feedback about the movie. After, the user had entered his/her feedback about the movie reviews, you need to censor all the rude words (and replace) with their antonyms fetched from the dictionary provided. **Note:** Please use Firefox browser while implementing activity 2 and activity 3, since Chrome does not allow you to set cookies locally and allows only from a hosted server.

You will use a dictionary structure provided to you. Requirements:

Create your own web application under the following constraints (both for Activity 2 and Activity 3)

C1. Your application has NO server component whatsoever.
C2. Your application has NO CSS.
C3. Your application must be a "single page application" – that is, it never reloads a page from a local source or does a document.write() to simulate a page refresh. The page is in effect your desktop application GUI.

Functionally your program must:

R1. (5) Greet the user by asking for her/his name on startup in a simple form. The name should be "remembered" and used anywhere a direct naming of the end user is appropriate. The web app should then start the conversation with a question. The question should be a common greeting like "Hello <name> Welcome to movie review System! Please enter your comments about the movie" and then followed by some article content. If the user had already logged in and is returning back greet him with an alert box having a message like "Welcome back <name>" along with the greeting message.
R2. (10) After the greeting is done, now display your own movie-review content about a movie using DOM manipulation of a non-form element (such as<p> or tag) that you have. Provide any input type that allows the user to enter his comments about the movie reviews given by the critics. You should parse the string the user types in and search for matches in the dictionary then censor all the words that are available as "keys" in the dictionary and replace them with their corresponding values.
R3. (5) You should vary the responses to the same keywords (although the number of responses will of course be finite) by introducing some simple randomization so no 2 same words follow the same pattern of responses. For randomization, use the basic Math.random() built into Javascript.
R4. (5) If the user does not complete or respond within 30 seconds after he sees the movie reviews, then the webapp should display a dialog box with a message from one of the values in the "idle" key field of the dictionary provided. There should be a prompt for every 30 second the user has not submitted his review. Again, the prompt should not always be the same. To implement this feature, look up the window.setTimeout API in the browser.
R5. (10) Your program should detect the presence of JSON input *into the user input form*, and have the ability to dynamically incorporate the new JSON into the dictionary by adding (not replacing) its entries to the existing dictionary the webapp is using. When this happens, the webapp should proudly announce using a prompt that : "Word added to the dictionary and the dictionary is smarter" (Note in your server-side implementation we did this with the fs module, here we will do it through user input). If the JSON is not valid (does not conform to a dictionary entry structure), then the webapp should say "Could not find the proper key and the dictionary stays dumb" using a prompt. If the entered JSON is completely invalid then prompt the user with an error message stating "Invalid JSON! Please enter a valid JSON!". Hint: You can treat an input to be a JSON if you find the braces "{","}" present in the input.

NOTES:

The given dictionary can be included as part of the js file by simple copy paste or using the "require".

1. Do not use session or local storage for keeping track of the user's name. You have 2 other alternatives ☺.

2. Note there is no HTTP in this lab. So, the textual input has to be handled on the button event through JavaScript.

Activity 3: Make your "forum" Stateful (35 points)

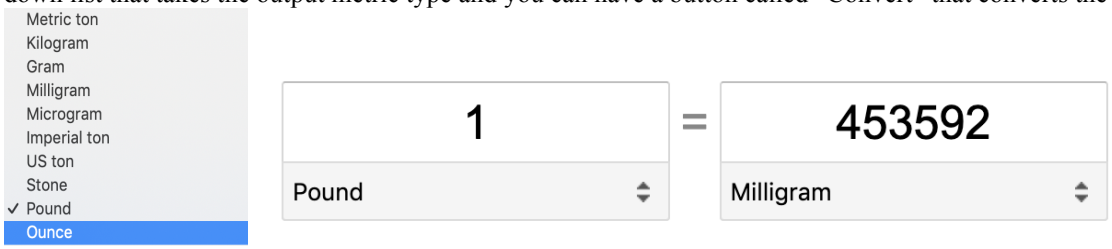
The "forum" web application in Activity 2 remembers a user's name but does not remember the conversation. In Activity 3, you should add stateful behavior – namely, "forum" should "remember" previous comments and censors the user has presented and also applies some randomness to avoid deterministic and/or repeated responses. New requirements:

R1. (5) Make your "forum" stateful by saving the censored responses to <name>. If the browser closes and restarts, and you come back to your "forum", you should be able to restore to the prior state where you left or where the last logged in user left.
R2. (5) Add a special "/clear" operation so that it clears the state of the application for <name>, then returns the app to the start form.
R3. (5) Activity 2, R2 asks you to randomize responses based on a keyword. Extend this functionality so that not only is it random, but you ensure no answer is repeated until all answers are given at least once.
R4. (5) Add a special /search <string> operation that searches the dictionary for any user entered <string>(search in "keys"), and copies the entire answer from the dictionary for that word into the user input area.
R5. (5) Add a special /history operation that lists all of the searches does within that browser session. The history should be displayed below the user input area using a non-from tag(<p> or , you have to manipulate the DOM here). If the browser is closed and re-opened, then the history is automatically cleared. The history is also cleared if the /clear special command is given (R2).
R6. (5) Add a special command "/count" that gives the total count of rude words used in that particular session by the user, onto the user input area. If the browser is closed and re-opened, then the count is automatically reset to zero and also the "/clear" command should set it to zero. (Do not use global variable for the count, marks would be deducted if done so, use one of the appropriate storage mechanisms)
R7. (5) Add a special command "/list" that lists all the reviews that the user had submitted. The content should be displayed again by doing DOM manipulation of non-form tags(such as or <p> tag) and can be displayed below the text area. Also, the review should be tracked only for a particular session and if the tab is closed and reopened the list of reviews should be empty. The list of reviews should contain both the "actual reviews" submitted and the "censored reviews" that were received.

NOTES:
1. You are expected to implement Activity 3 requirements using session or local storage. Using the right storage and managing it properly is considered part of the grading rubric for proper design.
2. Please specify your test input string for each command("/clear","/search","/history") and the expected output in the README.

Activity 4: DOM Manipulation (10 points)

1. (10) Create a simple weight converter application using HTML, JS (and CSS if you want, we don't grade based on the aesthetics though) that converts weight from one scale to another scale. There should be a one-line input that takes the number and a drop-down list that takes the metric type. Similarly, you should have a non-editable one-line output that displays the output and a drop-down list that takes the output metric type and you can have a button called "Convert" that converts the weights on click



Submission Instructions:

Submit your lab as a single zipfile named <asurite>_ser421lab4.zip with the following structure:
1. In a subdirectory "activity1", have the complete web page that is the result of your Bing search. Have a javascript file named activity1.js that has the sequence of expressions used to answer activity 1. Please put comments before each block of code to label it with each step of activity 1. Then save the source of your modified file (after running your activity1.js) as activity1.html (this is not the web page "complete", it is just what you get when you "view source". You can cut-and-paste into an editor and save).
2. Save your answer to activity 2 in the root directory under the folder activity2 with name activity2.js
3. Save your answer to activity 3 in the root directory under the folder activity3 with name activity3.js
4. Save your answer to activity 4 in the root directory the folder activity4 with name activity4.js
5. As always if there is anything you want us to know, put a README.txt in the root directory.
6. And again, you can submit a partial credit if not done in activity 2 or 3. Name it activity2[3]pc.js and explain what we should expect to find in the README.txt. We allow unlimited submissions so there is no reason to be late! Late submissions will not be accepted!