SER421: Web Applications and Mobile Systems Fall 2018

Lab 4, due Monday, 11/5/18 at 11:59:00pm via online submission to Blackboard

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 seach engine" text in the upper right corner result go away

Activity 2: Implement your own Eliza ON THE CLIENT! (45 points)

Since the early days of computing, humans wondered if computers could be made "intelligent". In the AI subfield of natural language processing, the argument was presented that if a computer could communicate like a human, then it possessed human intelligence. To demonstrate this approach, a program named ELIZA was created to emulate a computer talking to a person by responding to queries (https://en.wikipedia.org/wiki/ELIZA, http://psych.fullerton.edu/mbirnbaum/psych101/Eliza.htm). Eliza was Alexa before Alexa was!

The basic approach behind Eliza is to focus on keywords or substrings that a user presents to Eliza, and use it to determine a pseudo-intelligent response from a dictionary of responses. You will use a dictionary structure provided to you on the Blackboard site. Requirements:

Create your own Eliza under the following constraints:

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. Eliza should then start the conversation with a question. The question is not fixed, but should start with common greetings like "<name> how is your day going?" "<name>, is something troubling you?" or "<name> you seem happy, why is that?"
R2. (15) Provide a one-line web form that allows the user to "talk" to Eliza. You should parse the string the user types in and search for matches in the dictionary and select one for a response. Echo the user's input and Eliza's response "above" the web form (that is, the one-line web form should always be at the bottom of the page). *For echoing the input and Eliza's response, you MUST manipulate the DOM of an HTML non-form element – meaning something like a <p> tag or a list element, but not a textarea!*
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 sessions follow the same pattern of responses. For randomization, use the basic Math.random() built into Javascript.
R4. (10) If the user does not respond to an Eliza question within 30 seconds, Eliza should display a dialog box with a message such as "<name>, I'm waiting here!" or "Whatsa matter <name>, cat got your tongue?" or so forth (have some fun with it, but it must include <name>, where <name> is the end user's name). 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 Eliza is using. When this happens, Eliza should proudly announce: "I just got 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 Eliza should say "I don't understand that!'.

NOTES:

1. Eliza should display only the previous user response, Eliza's response, and the next question Eliza asks, followed by the prompt for the next end user input.
2. Do not use session or local storage for keeping track of the user's name. You have 2 other alternatives ☺
3. Note there is no HTTP in this lab. So the textual input has to be handled on the button event through Javascript.
4. You only display the last user response and Eliza response/question pair above the one-line input form.

Activity 3: Make Eliza Stateful (35 points)

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

R1. (5) Make it so Eliza displays a running dialogue of the entire conversation.
R2. (10) Make your Eliza program stateful by saving the responses it gives to <name>. If the browser closes and restarts, and you come back to Eliza, and enter the same <name> as a prior respondent, then you should be able to restore to the prior conversation.
R3. (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.
R4. (10) Activity 2, R3 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.
R5. (5) Add a special /search <string> operation that searches the conversation for any previous user answer containing <string>, and copies that entire previous answer into the one-line user input area.
R6. (5) Add a special /history operation that lists all of the searches does within that browser session. 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 (R3).

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.

EXTRA CREDIT:

1. (30) On the class website is a file ec.jar which is an executable desktop Java application. You can run it by downloading it and typing in java -jar ec.jar on the command-line while in the directory where you downloaded it. For this extra credit, replicate this desktop application as a SPA in the browser. You may use any tricks you like, including CSS, for this EC, but you still may not use a server and HTTP. You are to create the closest approximation of the user experience that you can. Name your solution file ec.js

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 name activity2.js
3. Save your answer to activity 3 in the root directory under the name activity3.js
4. The extra credit should be named ec.js as specified.
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!