University of Arizona, CSc 337
Homework Assignment 9: Chat-It

This assignment is about making a web service and using Ajax to post data to it. You will write a NodeJS service that will save posted chat comments into a file and respond to GET requests with JSON representing all chat comments ever made. You will also write client Javascript code to make requests to this service and inject the responses into a page. Turn in these files:

- **chatit_service.js**, the NodeJS service that will save new comments and return data on all comments
- **chatit.js**, the client Javascript that will send to and request the information from **chatit_service.js** and inject it into **chatit.html**
- **chatit.html** the HTML representing the chatit page
- **chatit.css** the CSS for the chatit page

Web Service Details:

Your **chatit_service.js** service will respond differently to GET and POST requests.

A POST request should contain information about the particular comment to add to the file. This information should include as separate pieces, the commenter's name and the comment. Your code should append these to the end of `messages.txt`. Append them as a new line with the name first and the comment second. Separate the name and comment by 3 colons (`:::`). Send back a message to the client indicating whether or not the request was successful. Only consider it successful if the message was able to be saved correctly.

A GET request should be responded to with the contents of `messages.txt` represented as JSON. For example, if the file contained:

```
Merlin:::meow
Purcy:::meow meow meow
Merlin:::grumble meow
```

The JSON should appear as follows:

```
{"messages" : [{"name" : "Merlin", "comment" : "meow"},
               {"name" : "Purcy", "comment" : "meow meow meow"},
               {"name" : "Merlin", "comment" : "grumble meow"}]}
```

You may assume that `messages.txt` exists and has valid content at all times.

Javascript Client Details:

Your **chatit.js** will use Ajax fetch to send and request data from your NodeJS service and insert it into **chatit.html**. Here is the functionality your page should have:

- When the page loads it should request all of the comments from the web service and insert them into the page. You can decide exactly where you want to insert them into the page. The only constraints are, they need to all be inserted and they need to have the name and comment inserted in different tags (although these tags can be contained in one, for example a the name in a span which is contained in a paragraph which also contains the comment, would be fine).
- The page should refresh the comments every 5 seconds. You will find the `setInterval` function helpful for this. To refresh the comments, all comments should be removed from the page and you should then do exactly the same things as when the page loads.
- When a user clicks the send button the contents of the name box, and the comment box should be sent to the service as a POST request. The service will respond with a success message if the request executed successfully and a failure message if it did not. Some indication of whether the message sent successfully should be added to the page.

HTML and CSS:

The HTML and CSS are up to you. We expect you to follow the style guidelines that we have discussed so far this semester. Make your HTML representative of the structure of the page. However, we leave the exact type, number and placement of tags and all style up to you. To receive full credit you will need to include at least 10 different, non-redundant, styles.

Implementation and Grading:

For full credit, your JS code should follow the rules listed in the **Style Guide** on the class web site. Your page's output must successfully pass the W3C **HTML validator**. (Not the NodeJS service but your **chatit.html** after you have inserted data you have gotten back from the service in it. To validate your page, view the page, then choose View Source in your browser and copy/paste it into the validator.) Your JavaScript code should pass our **JSLint** tool with no errors (an error because it doesn't recognize fetch is fine). Your .js file must run in **JavaScript strict mode** by putting `"use strict";` at the top.

We will also grade the **style of your NodeJS and JS code**. Do not use the `global` keyword. Use indentation/spacing, and avoid long lines over 100 characters. Avoid redundant code, and use parameters and return values properly. Make extra effort to minimize **redundant code**. Capture common operations as functions to keep code size and complexity from growing. For full credit your code must have **at least one function** in each language to remove redundancy and/or capture structure.

Use the **"module pattern"** shown in class to wrap your JS code in an anonymous function. **No global variables or symbols**, nor "module-global" vars, are allowed in JS on this assignment; values should be declared at the most local scope possible. If a constant value is used, you may declare it as a module-global "constant" variable `IN_UPPER_CASE`.

Fetch data using **Ajax fetch**. Be careful to avoid redundancy in your Ajax code; if you do similar Ajax operations in many places, make a helper function(s). Process JSON data using `JSON.parse`.

Another grading focus is **commenting**. Put a descriptive header (name, course, date, description) at the top of your code and comment each non-trivial section of NodeJS and JS code explaining the purpose of that code.

Format your JS and NodeJS code similarly to the examples from class. Properly use whitespace and indentation.

Please do not place a solution to this assignment online on a publicly accessible web site.

Copyright © Allison Obourn, licensed under Creative Commons Attribution 2.5 License. All rights reserved.