Notes on Senior Project

Aaron Danielpour

5/2/16

As I mentioned in the presentation and report, PHP cannot be run by browsers on their own. To get around this, you need to set up a local Web Development Environment. I used Wampserver for this (<http://www.wampserver.com/en/>), as it offers everything that was needed, but as you are on a Mac you'll need to find something for that. About 10 seconds on google yielded a site for an application called Mampserver, which appears to offer similar functionality for Macs. What could also work would be to establish proper web hosting for the app and use something like Filezilla to push updated files when running new tests, but I used Wampserver for testing purposes.

PHP itself can make SQL calls to a server. For reference, check either of the getdata or getsqlline scripts. In both of them, the variable $conn is only defined if a connection can be established with those arguments. A quick check is made to verify that $conn exists and kills the script early if there was a problem with connection. Otherwise, it calls the query defined in $sql. The file getsqlline takes an argument from the php call to allow it to pull up a specific line from the sql, check that file to see how, it's self-explanatory.

For the main web app, the functions used in it can be divided into several broad categories:

-First, there are the php calls. These are governed by getsql() and grabnewline(). In both cases, these make a call to the php files on the server (see those specific functions for an example of how it is done, it's straightforward). The JSON returned by the PHP cannot be directly parsed, as it's got <html> tags surrounding it. Thus, trimResponse removes those tags. It may be possible to remove those tags directly from the PHP to streamline the process, I didn't check before our presentation. Once that is done, the response is passed into the parseResults method, which automatically pushes the data into various arrays set up to store their data.

-Then, there are the Chart functions. These functions handle initial setup for the charts. It should be noted that the vars for the charts are defined at the start of the script section, and redefined when the chart functions are accessed. This is to ensure the variables are in-scope for the entirety of the web app's code, so they can be altered in any section. Each chart function runs similarly - they run their initialize method, putting all of the relevant data from our base arrays into the corresponding chart data arrays. They then run their update methods, updating the chart's datArray variable with data in the proper format. Lastly, it plugs the datArray into the chart's data argument and runs the render() method. The render() method is provided by canvasJS, so we don't need to define it. Because datArray is now fed into the chart's data argument, all future methods to edit the chart's data only need to alter the chart's corresponding datArray and run the render() method.

-The trimlines() function is specific to the line chart. It runs to keep the line chart's data arrays to a specific length, so that the chart stays at a constant resolution and that it does not "zoom out" as the user keeps the page open.

-Each chart also has a "clear" function written to reset its arrays so that data does not pile up (resolving a bug that created multiple instances of the averages in the Pie Chart, creating a 200-segment pie chart). There are further "\_\_\_\_\_\_All" functions that call all of the chart methods. If you add any other charts you should be sure to stick the relevant methods into the "\_\_\_\_All" function.

-Lastly, we have updateAll and the Filter method. These are called by the grabNewLine() method, and deal with the dynamic updating of the charts. Simply put, updateAll clears the chart arrays that need clearing (i.e. the actual arrays - as the averages are ints they do not need clearing), and then if a filter has been applied, filter() is run, while otherwise, the clear, update, and render functions are called in turn, to provide a graph with all data sets included. Filter first clears all the datArrays, and sets the "filtered" boolean to "true", indicating that a filter is active. It then gets all "filterCheckbox" objects, and checks the value of each one. If it is checked, then the corresponding dataset is added to the datArrays. Once all filterCheckboxes have been reviewed, the charts are re-rendered, so the user gets immediate feedback. This also means that updateAll just has to call Filter() in the event filters are active.

-the var updateInterval sets the interval at which new lines are grabbed from the data. As I mentioned in the presentation, this is every 3 seconds for testing purposes, as waiting 10 seconds for each new line is too slow. Simply change the value of "secondsBetweenUpdate" to the desired number of seconds for an update. As the sensor data is taken every 10 seconds, the final value should presumably be 10.

Lastly, some notes:

-There is no check for the database being empty. This is because the sensor data should keep coming in every 10 seconds, ensuring that the database is never empty.

-You'll notice that the functions call each other in a chain. The key parts of this chain are getSQL -> parseResults -> the Chart functions. This ensures that the received SQL data is already stored clientside in the relevant arrays, so that the charts are certain to have data to display. Likewise, grabnewline also calls in a chain, calling parseNewLine and then updateAll, ensuring that the new data is stored before usage attempts. Not doing this leads to a possibility of missing the newly parsed data due to the (very small) response time from the PHP file.

-Adding new charts is as simple as changing the "type" argument in the data for the chart, and making sure that the dataPoints are in the proper format.

-The Line function uses an x variable to ensure that the new data points are added sequentially. Without the x-variable, the new datapoints are all added at the same x location. The label argument in the line datasets overwrites the x-value on the x-axis.

**Notes**:

1. The die() function prints a message and exits the current script. (<http://www.w3schools.com/php/func_misc_die.asp>)
   1. This function is an alias of the [exit()](http://www.w3schools.com/php/func_misc_exit.asp) function.
2. The mysqli\_connect\_error() function returns the error description from the last connection error, if any. (<http://www.w3schools.com/php/func_mysqli_connect_error.asp>)
3. The $\_GET variable is used to get data from a form that is written in HTML. Also in the url $\_GET variable will display the data that was taken by the $\_GET variable. (<http://shodor.org/~kevink/phpTutorial/nileshc_getreqpost.php>)
4. The mysqli\_query() function performs a query against the database. (<http://www.w3schools.com/php/func_mysqli_query.asp>)
5. In getSQL():
   1. AJAX – The onreadystatechange Event (<http://www.w3schools.com/ajax/ajax_xmlhttprequest_onreadystatechange.asp>):
      1. **onreadystatechange**: Stores a function (or the name of a function) to be called automatically each time the readyState property changes.

**Questions**:

1. Why is the password blank?
2. Why is the limit 60 in getdata.php?