Shivam Dave - Unit Tests

Javascript - DateSetter.js & JSON files

The DateSetter.js file not only setups and updates the charts however it also gets the queried date range, parses it, and creates the url that will be called on the api to get data to be fed into the charts. It also displays the charts or a message based on if the data exists for a specified time period.

To test the chart creation and updating I used the tiny.html file that displays the dashboard. Based on the options I set, I checked if the updates affected the site and how they affected the site.

To test the date range parser I used console logs that would log what the url was when coming from the date and time pickers. Then as I parsed the date and time, I used the logs to step-by-step see how the string of the date and time changed. After creating the final parsing of the api querying url, I compared that with the wanted format of :
“day=2015-11-22&start=21h00m00s&end=21h00m20s”

To test the empty data checker I used 2 api url calls on the database: one that contained data and one that did not contain data. Using these 2 data api calls I was able to consistently test my functions against the actual data sets that were empty or filled. Also then after creating the lists of charts that should be shown and hidden, I compared that with the database query using console logs.

PHP - history.php & db_credentials.php (Not in GitHub because possibly sensitive)

The history.php file is used by the api to parse a url string (for accessing the database) and creates a mysql query based on that. In order to test this part, I used phpMyAdmin, which is a GUI used to manage databases on a server, or in our case a server on a virtual machine. The phpMyAdmin page is hosted on the VM so one can access it by going to http://localhost:8080/phpMyAdmin/. This page displays all the rows within a database that has tables with data. To test if my history.php file was pull data consistent with the database I would use a SQL query in the phpMyAdmin page on the database t_data, which holds temperature and frequency data values and timestamps with each of them. By running these queries side by side I could tell if I was querying correctly and if the timestamps matched up with both queries.

ChromePHP is a library I used to log php variables and show them in the Chrome developer console. This aided testing on the php file side to see if the url from the api call was being parsed correctly by the php file.

HTML/CSS - tiny.html

The HTML/CSS portion was tested by hosting the html file and accompanying files on a server VM’s /var/www/html directory. The frontend representation of the dashboard could be accessed through the URL http://localhost:8080/app/tiny.html. As a result all changes to the frontend could be seen through this URL and also the developer console also provided error messages and any logs I had put into the files.