**CIS 695 Documentation**

**Introduction**

Due to many individual OPENXC projects that have been developed, there is a need for integration of all the scattered projects, in which engineers are able to use them through one user interface. The idea is to build a RESTful API for each individual project while the integration program running on another server is able to call their functions and get the results, as well as display them on its own user interface. In this project, the main goal was to build RESTful API for Trace Analyzer (<http://trace-analyzer.openxcplatform.com)>. Trace Analyzer is a browser – based web application that receives a vehicle data file with JSON format and responds with a map, shows the drive trace, graphs show fuel consumed, fuel efficiency, gear position, vehicle speed, engine speed, and other figures like fuel consumed cost, cumulative fuel efficiency.

**Developing Environment and Implementation**

The goal of building RESTful API is to do the less modification as it needs. Trace analyzer is written mostly by JAVASCRIPT. In this project, RESTful API of trace analyzer was written using node.js. Node.js is a light weight asynchronous event driven JAVASCRIPT runtime.

- **Express** web framework is used to build the API.

- Because of functions are called through API instead of through HTML document, a **data object model** needs to be built in the API to support all the functions (Trace analyzer uses a lot of external libraries like jQuery, underscore.js, chart.js, d3.js). They are called based on document nodes (like id, class or a tag name).

1. **var** jsdom = require('jsdom');
2. **const** {JSDOM} = jsdom; //create data model
3. **const** dom = **new** JSDOM('YOUR HTML CODE');

- Import all the **external libraries** that needed to fulfil the original functions. (They can be downloaded from **npm** -- https://www.npmjs.com)

1. **const** express = require('express');
2. **var** request = require('request');
3. **var** \_ = require('underscore');
4. **var** d3 = require('d3');
5. **var** $ = require('jquery')(**new** JSDOM('').window);
6. **var** extend = require('jquery').extend;
7. **var** Chart = require('chart.js');

- **Copy and paste** original functions into the server and make minor modifications.

- API function started by creating a express() object and call HTTP method(I used GET here). It also has two parameters, one is the API pattern, the other is a callback function which contains two parameters (a request and response object) that are used to pass the contents of request and response.

1. **var** app = express();
2. app.get(‘/api/pattern’,**function**(req,res) {
3. //your code
4. });

- API calls **listen()** function to listen to a port number. it will response when there is http request coming through.

1. app.listen(process.env.port || 4000, **function**() {
2. console.log('now listening');
3. });

In this project, we wanted to be able to display fuel information (For example, overall fuel efficiency, total fuel consumed, total fuel cost and average gas price in the area), as well as a graph of fuel consumed and graph of fuel efficiency. We also wanted to be able to display gear position histogram, graph of vehicle speed and average vehicle speed, graph of engine speed and average engine speed, graph of torque and average torque value, and accelerate pedal, as well as an odometer graph. There are 4 functions in this API to support implementing the above functions. They are sendURL, retrieveAndProcessData, getPosition and getResults.

***-sendURL***

The caller sends the URL of vehicle data file through jQuery ajax request. **All the request calls are made through jQuery ajax**. The server receives the URL and stores it in a local variable for further use. The caller will receive confirmation stating that the data has been received.

***-retrieveAndProcessData***

In this function, the caller send request to RESTful API and RESTful API will retrieve the data from URL it just received and parse data into a certain format and finally store the processed data into an array. The caller will receive confirmation stating that the data have been received and processed as well. The body of function is modified and reorganized from original processTrace() and handleMessage() in trace.js file.

***-getPosition***

To get the position (latitude, longitude) of where the drive taken place from the received data. The purpose of getting position is to call another API to get the local gas price. In the body of getPosition, updateGasPrices() and updateFuelSummary() are invoked from original trace analyzer. Some changes have been applied to these two functions.

1. **var** updateGasPrices = **function**(trace) {
2. **var** gasDistance = 5;
3. **var** apiKey = "rfej9napna";
4. **var** recordWithPosition = \_.find(trace.records, **function**(record) {
5. **return** record.latitude && record.longitude;
6. });
8. /\*$.ajax({
9. url: "http://devapi.mygasfeed.com/stations/radius/" +
10. recordWithPosition.latitude + "/" + recordWithPosition.longitude +
11. "/" + gasDistance + "/reg/price/" + apiKey + ".json",
12. dataType: "jsonp",
13. success: function(data) {
14. console.log(data.stations.length);
15. var stations = data.stations;
16. if(stations && stations.length > 0) {
17. var stationsWithPrice = \_.filter(stations, function(station) {
18. return station.reg\_price !== "N/A";
19. });
21. var averagePrice = \_.reduce(stationsWithPrice,
22. function(memo, station) {
23. return memo + parseInt(station.reg\_price, 10);
24. }, 0) / stationsWithPrice.length;
26. $("#total-fuel-cost").text((averagePrice \*
27. trace.fuelConsumedGallons.toFixed(2)).toFixed(2)).parent().show();
28. $("#average-fuel-cost").text(averagePrice.toFixed(2)).parent().show();
29. }
30. }
31. });\*/
32. **return** recordWithPosition;
33. };

I commented this ajax request (getting gas price from position values) and moved it to caller ’s side, as well as added a return statement of position values since the ajax call is asynchronous. When the updateGasPrices() function return a value, the callback function from ajax call has possibly been executed. That’s why instead of executing a ajax call here, I moved it to caller’s side to let it receive the position parameters from API and return gas price.

1. **var** recordsWithFuelConsumed =  **function**(records) {
2. **return** \_.filter(records, **function**(record) {
3. **return** record.fuel\_consumed\_since\_restart;
4. });
5. };
7. **var** fuelConsumedGallons = **function**(a, b) {
8. **var** fuelConsumedLiters = b.fuel\_consumed\_since\_restart -
9. a.fuel\_consumed\_since\_restart;
10. **return** fuelConsumedLiters \* 0.264172;
11. };
13. **var** calculateFuelConsumedGallons = **function**(trace) {
14. **var** fuelRecords = recordsWithFuelConsumed(trace.records);
15. **return** fuelConsumedGallons(\_.first(fuelRecords), \_.last(fuelRecords));
16. };
18. **var** updateFuelEfficiency = **function**(trace) {
19. trace.overallFuelEfficiency = distanceMiles(\_.first(trace.records),
20. \_.last(trace.records)) / trace.fuelConsumedGallons;
21. //$("#fuel-efficiency").text(trace.overallFuelEfficiency.toFixed(2)).parent().show();
22. **return** trace.overallFuelEfficiency.toFixed(2);
23. };
25. **var** updateFuelSummary = **function**(trace) {
26. trace.fuelConsumedGallons = calculateFuelConsumedGallons(trace);
27. fuel\_values.push(trace.fuelConsumedGallons.toFixed(2));
28. //$("#total-fuel-consumed").text(trace.fuelConsumedGallons.toFixed(2)).parent().show();
29. //updateGasPrices(trace);
30. fuel\_values.push(updateFuelEfficiency(trace));
31. };
32. //updateFuelEfficiency function, Intead of assign the value to a node in html code
33. //I let it return a value so it can response back to callers

In both updateFuelEfficiency() and updateFuelSummery, I commented the assignment of value to HTML node. Instead, I push both values in to a local array variable **fuel\_values.**

1. res.jsonp({"recordWithPosition":recordWithPosition,"totalFuelConsumed":fuel\_values[0],"OverallMPG":fuel\_values[1]});

Then all those values collected will be sent to the caller’s application for further use.

***-getResults***

The getResults function calls 3 original functions from the trace analyzer. They are timeseriesHandler.onLoad(), calculateCumulativeFuelEfficiency(), drawGearHistogram()

In calculateCumulativeFuelEfficiency(), I did a little change on function drawTimeseries(), which is used to draw the graph. In original function, it uses d3.js tool to select node from HTML document and append the graph elements with data. However, in our API, we don’t have HTML document from the caller’s application. Instead, we used the document object model we’ve created.

1. //original code
2. **var** graph = d3.select("#" + elementId).append("svg:svg").attr("width", "100%")
3. .attr("height", "100%");
4. //modified code
5. **var** graph = d3.select(dom.window.document.querySelector("#" + elementId)).append("svg:svg").attr("width", "100%")
6. .attr("height", "100%");

We also needed to change some of syntax of d3 function calls because in original function d3.v3.js is applied, but in RESTful API d3.v4.js is applied. For example,

1. //original code
2. **var** x = d3.scale.linear().domain([\_.min(dataX), \_.max(dataX)]).range(
3. [0, dimensions.width]);
4. //modified code
5. **var** x = d3.scaleLinear().domain([\_.min(dataX), \_.max(dataX)]).range(
6. [0, dimensions.width]);

In function drawGearHistogram() , I commented the codes that select the document, get context, and draw histogram using chart.js. Instead, I let it return the **data** that needed to draw the histogram.

At the end of getResults function, the document object model and bar chart data are sent back to caller’s application.

1. res.jsonp({"dom":dom.serialize(),"barChartData":data});

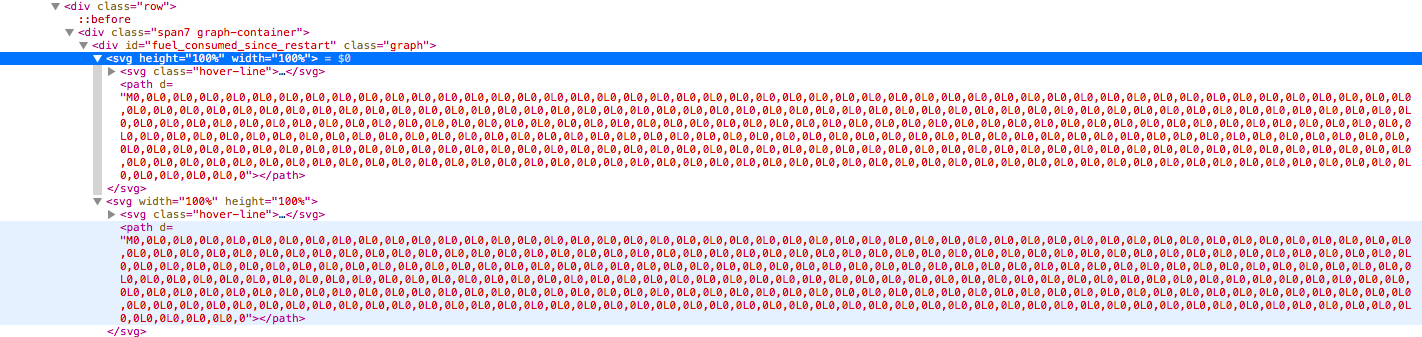
To run the RESTful API on server, we use terminal directing to root directory of Trace analyzer and find our server file ‘index.js’, type ‘node index’ in terminal, the API is officially running when you see ‘now listening’ in the terminal.

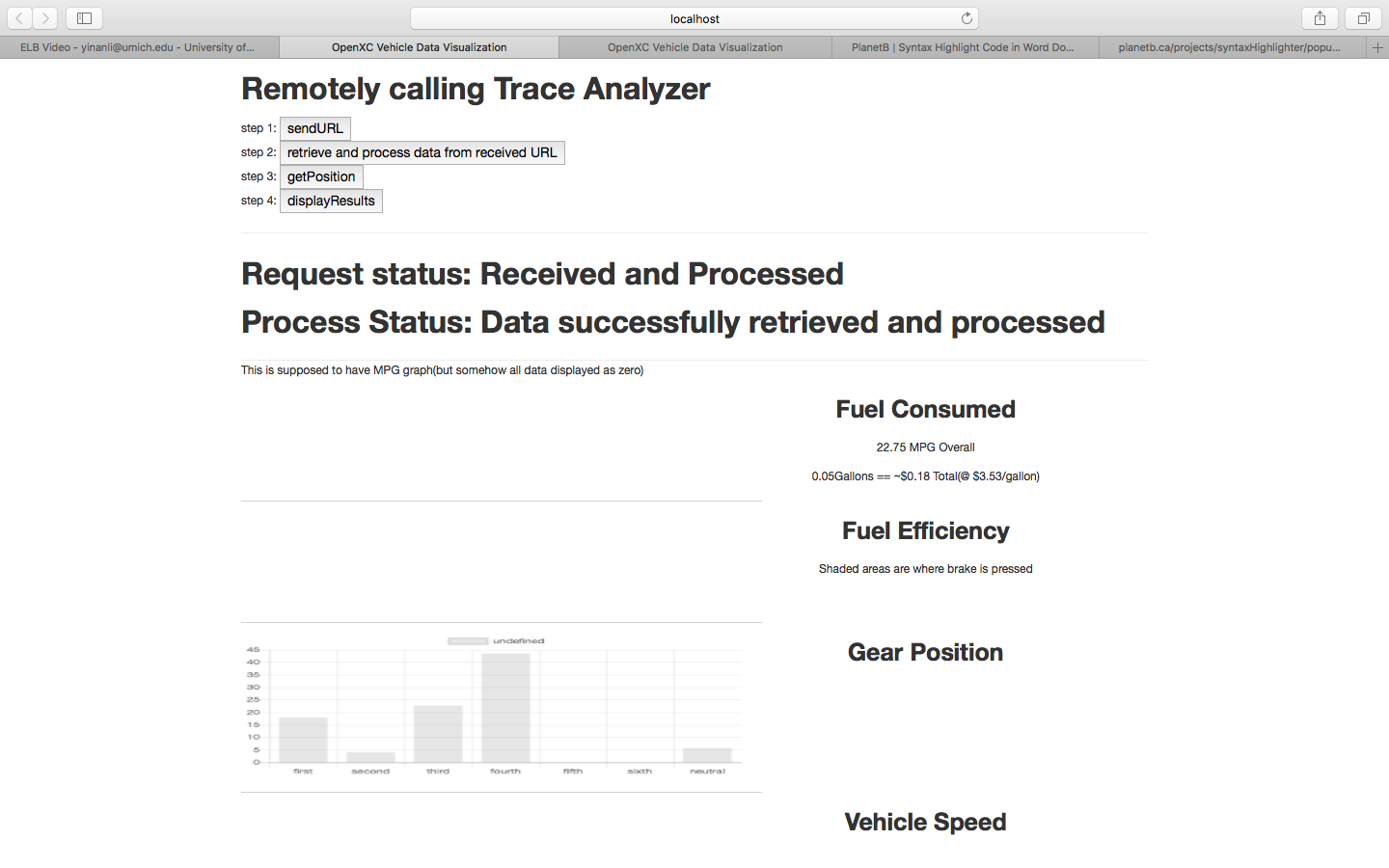
On caller’s side, we use controller.js to make all the calls to consume the RESTful API,

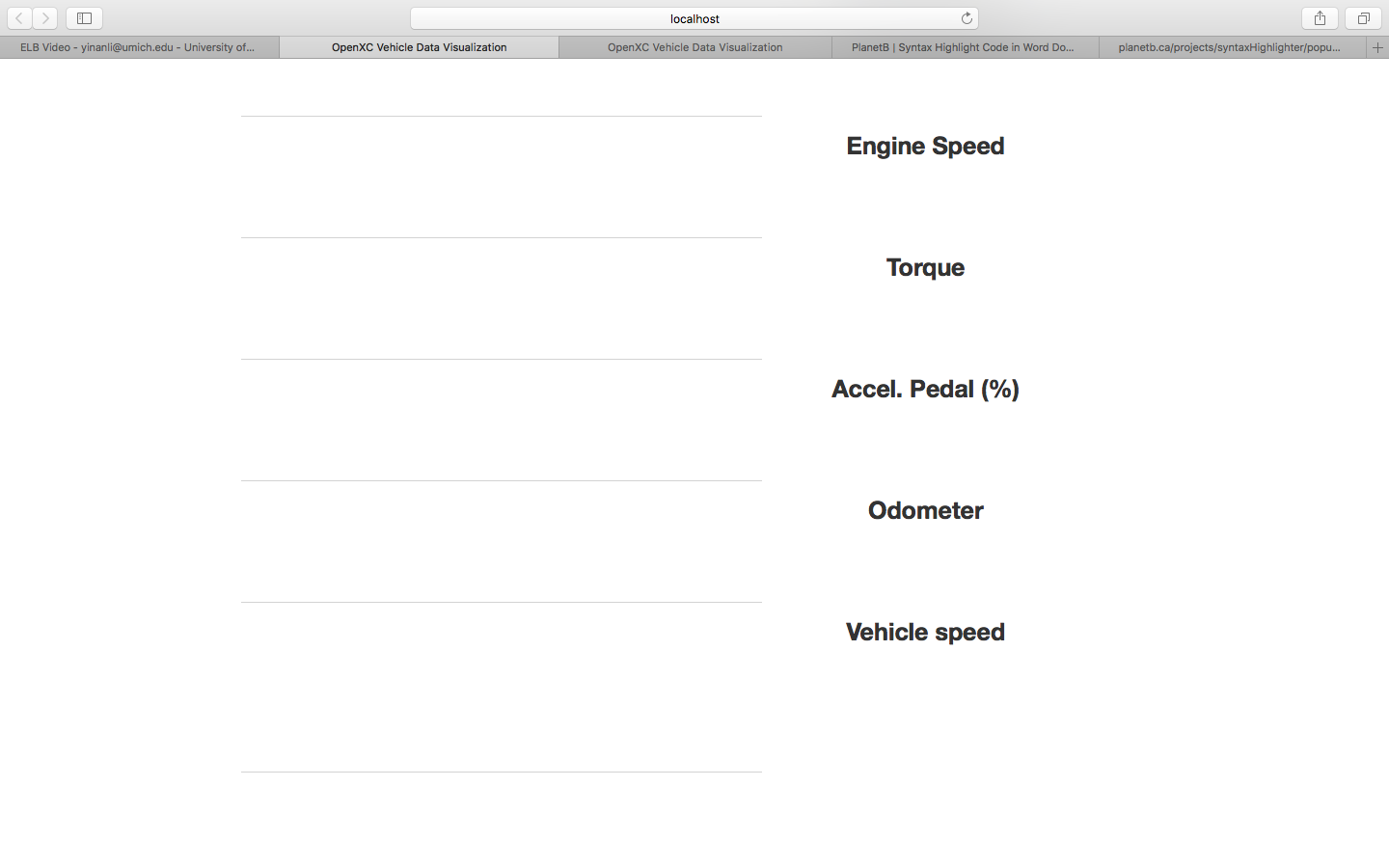
1. $('#sendURL').click(**function**(){
2. $('#step1').empty();
3. alert("send url to api");
4. $.ajax({
5. url: "http://141.215.223.21:4000/api/sendURL",
6. data:{
7. url:'http://openxcplatform.com.s3.amazonaws.com/traces/nyc/downtown-west.json'
8. },
9. dataType: 'JSONP',
10. jsonpCallback: 'callback'
11. }).then(**function**(data) {
12. $('#step1').append("Request status: " + data.requestStatus);
13. });
14. });

**Result**

This is the result shown on the caller’s user interface. Fuel information and the gear histogram are able to be shown successfully. However, all the time series graphs are missing. By inspecting the source codes, I found out all elements are successfully appended to caller’s user interface, but the data are not appended correctly. The average values of vehicle speed, engine speed, torque can be shown on HTML code.

****

****

****

**Further work to be done.**

1. *Exception handling*:

When the caller sends the URL of vehicle data file to RESTful API, we will have to consider the possibility that URL is not valid. We will need to write exception handling function to deal with errors like that.

1. *Flexibility:*

The caller now sends one fixed URL to RESTful API. We want the caller to be able to select different URL from local computer or from the cloud Server database filtered by certain conditions. In that case, database technique like JDBC, Database design might need to be applied.

1. *Modification on getting time series graphs:*

Some changes on drawTimeSeries() need to applied because we are not able to append the data correctly to draw the graph yet.