**FUEL FOOL: MID-TERM REPORT**

**CS 419: WINTER 2016**

**GROUP 18**

**Jennifer Erland**

**Charles Hardes**

**Ty Hatton**

**Jason Murray**

**ABSTRACT**

This report will outline the Fuel Fool web application; to include detailing up to date progress and remaining works to be completed. Particular attention is paid to the background data ingestion of the application, describing what datasets are used to run the final product. The sections of PROGRESS and WHAT’S LEFT have been broken into the two parts of the application: home page and main(map) page.

1. **INTRODUCTION**

The Fuel Fool web application is designed to give the user a simple, graphical way to view gasoline prices across the United States by displaying the average gasoline price for each zip code as a choropleth map. The user is able to select any of 4 gasoline types/grades: regular, midgrade, premium, and diesel. Upon selection, a new choropleth map of the United States will be displayed. This web application will consist of two web pages; a homepage to gather user inputs, and a map page.

**2. DATA**

The data for Fuel Fool can be broken into two parts; the fuel prices, and the zip codes. While the fuel prices are expected to be updated daily, the zip codes should remain constant. Fuel prices are to be delivered via a third party vendor to a Fuel Fool administrator. These fuel data are derived from over 130,000 retail stations across the U.S. for all zip codes containing at least one retail fuel vendor.

**2.1 Zip Codes**

Zip code data is maintained in a topoJSON file. The TopoJSON file is a JSON formatted string that contains information pertaining to each zip code (zip code, locale, state), but also the geographic location. From the author of the ToposJSON extension:

**TopoJSON** is an extension of GeoJSON that encodes topology. Rather than representing geometries discretely, geometries in TopoJSON files are stitched together from shared line segments called *arcs*. TopoJSON eliminates redundancy, offering much more compact representations of geometry than with GeoJSON; typical TopoJSON files are 80% smaller than their GeoJSON equivalents. [1]

**2.2 Fuel Prices**

The fuel prices will be received in the form of a .csv file. The file will contain zip code, regular fuel price, mid-grade fuel price, premium fuel price, and diesel fuel price. Given in the form:

zip,reg,mid,prem,diesel

It is important that the form is known, and consistent, as it will be relied upon when the price data is combined with the topoJSON file, as described below. All fuel prices will be in dollars ($) per gallon. .

**2.3 Combined (Zip Codes - Fuel Prices)**

Once the fuel price is received, it will need to be merged with the zip code data. The topoJSON file will be uploaded to Mapbox to create a dataset, which will form the basis of the final map’s data layer. To accomplish this, the zip code topoJSON file will need to contain all data, including the fuel prices. This is accomplished by merging the fuel prices (csv) with the topoJSON file. These two files are joined on the zip code property, and new properties (in the topoJSON file) are created for: regular, midgrade, premium, and diesel.

**3. USER INTERACTION**

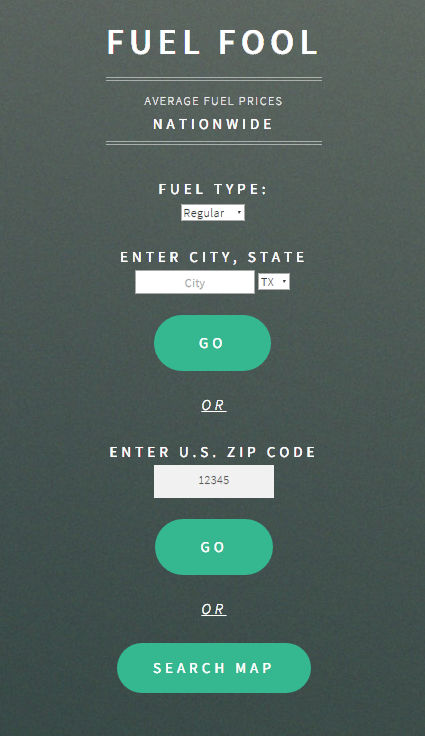
The user will have the ability to interact with the web application four different ways. These interactions will be the basis for allowing the user to both select a fuel type and a geographic region.

**3.1 Fuel type selection**

Selection of preferred fuel type is constrained within a drop-down menu. The default fuel will be regular gasoline and data for this type of fuel will be presented on the following map page. If the user desires to change fuel types, she will use the drop-down and all resulting data will be of this type.

**3.2 Geographic location**

The user can choose how they select the particular zip code defined area three different ways. The first method the user is presented with in a top to bottom manner is by a city/state combination, where the user inputs a known city and state to query. The next method is similar as the user enters a zip code. Both of these methods will allow the user to quickly find and review the fuel prices for the area they are interested. The third method is by directly interacting with the map. To do this the user simply selects the search map button. Each of these methods results in the web application returning the map page with various levels of geographic detail and information about the selected fuel type. The detail will be more absolute if the user inputs a zip code or city/state combination.

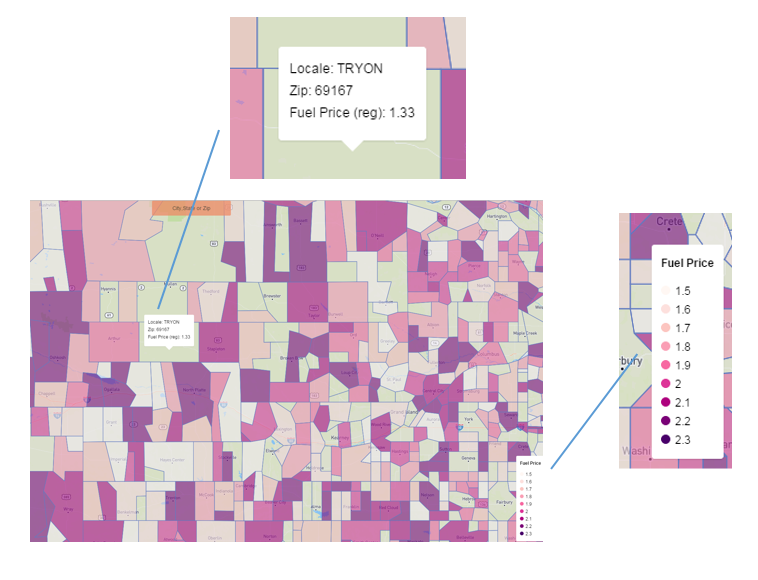


**4. PROGRESS**

The home page has some rudimentary styling as the effort was to minimize user input and focus the user to make selections based on the user's current level of information. Levels of information are if the user has a specific zip code, or slightly more general a city and state, or simply knows they want to search for fuel information by looking and targeting an area on the map page.

Aside from how the homepage is presented, the underlying function of this page has yet to be fully implemented. At the moment there is a disconnect between the homepage and the map page. Each of the four data query inputs will need to perform different tasks to access the requested data. The simplest being connecting the search map button to the map page. The zip code and the the city/state combination will need to have further work to utilize these inputs.

In the current state of the project, there is a base map of the United States and a data overlay of zip codes with fuel prices. The color scheme of the choropleth, and the price intervals are currently hard-coded for testing purposes, but sufficient to verify that the functionality is working properly. While the map is currently functional, it is currently not stylish; it is lacking a “finished look.”



**5. WHAT’S LEFT**

While the outline of the project is complete, the finer details of the application will still need to be completed.

**5.1 Home Page**

Currently the only GET parameter being passed to the main page is the fuel grade, eventually a location (given in latitude, longitude) should also be sent if selected. If the user specifies a location (either by zip code or city,state) this should be converted to a latitude and longitude and passed to the main page. This location would then be used to center the map that’s displayed, and also the initial zoom level.

As this web application is anticipated being used in both mobile and non-mobile web browsers, more layout adjustments need to be made. Currently the display of the home page is too large to easily view.

*5.1.1 LatLongLookup.js*

This should be an external javascript file (rather than on the home page) so that the main page can also utilize its functionality.

**5.2 Main Page**

The main functionality of the choropleth map is in place, but as mentioned previously, it lacks a completed look.

*5.2.1 Styling*

Currently there is no container for the map, it is simply sitting in an html div container with no borders, nothing to distinguish it from a plain background. This will need some sort of styling, and also optimization for a mobile browser.

*5.2.2 Search box*

Currently upon loading, the map will load centered on the geographic center of the continental United States. Once the latitude and longitude parameters are passed from the home page the map will center on that location, but what if the user want’s to jump to another location? Currently the only way to do that is click and drag/pan to that location. A search box will be added to the map to allow the user the same options as the home page: entering a zip code, or city,state. This should reference a function in the aforementioned LatLongLookup.js file to return the latitude and longitude of that user-defined location, and “jump to” that location on the map.

**5.3 Miscellaneous**

A few items that should be completed, but do not necessarily fall into the above categories include an external method to determine the price interval of fuel prices and the colors to be used.

*5.3.1 JavaScript Utilities*

Given the fuel prices from a csv file, return the price intervals to be used by the map. This could be implemented multiple ways, for example: find the min and max, then calculate 10 even intervals in between. This would not work well if the prices are not evenly distributed. This utility could also maintain the previously mentioned LatLongLookup function, and also grow to include any new functionality that may arise.

**6. PROBLEMS ENCOUNTERED**

Other than the expected growing pains that come with any project utilizing unfamiliar frameworks, APIs, utilities, etc, the only problem was figuring out how to display such a vast amount of data, while still being responsive on a mobile device. There are 42,000 + zip codes in the United States, that is quite a bit of data to display. The first iteration of the choropleth map involved merging the csv fuel prices with the topoJSON in real-time, on loading the page. This would create a single, continuous layer of data overlaid over a basemap. So every time the page was zoomed or dragged, everything would need to be redrawn. This was doable in a desktop browser, but not feasible in a mobile browser. The answer to this problem was to simply NOT “draw” all the data at once.

**6.1 Problem Solved**

The solution here was to not draw all the data at once. This is accomplished by utilizing Mapbox to create a dataset (in particular, vector tiles) from the zip code topoJSON file. If the zip code topoJSON file is going to be used to create a dataset, it logically would follow that all necessary data should be embedded in this layer; this lead to the merging of the fuel price csv and zip code topoJSON file behind the scenes, rather than in the user’s browser. Once Mapbox has processed the topoJSON file and created a dataset, the Mapbox GL API is used to bring in this data as a source, then the choropleth is built by querying this data layer based on fuel grade and a price interval. Each price interval is then used to create a layer, which will be overlayed on the base map of the United States, as seen in the code below:

map.on('style.load', function () {

map.addSource('vector-data', {

type: 'vector',

url: 'mapbox://murrayja.70d5063c'

});//end addSource()

for(var p = 0; p < breaks.length; p++)

{

var filters;

if (p < breaks.length - 1)

{

filters = [ "all",

[ ">=", grade, breaks[p] ],

[ "<", grade, breaks[p + 1] ]

];

}//end if

else

{

filters = [ "all",

[ ">=", grade, breaks[p] ]

];

}//end else

/\*

Add layer

-source: the source of the layer data (from addSource() call above)

-source-layer: the source LAYER from the vector tile. this is the name of the layer within map ID: murrayja.70d5063c (this is found in Mapbox Studio)

\*/

map.addLayer({

id: 'data-layer-'+p,

type: 'fill',

source: 'vector-data',

'source-layer': 'zips\_prices\_numbers\_topoJSON', //this is the name of data layer in mapbox studio

layout: {

visibility: 'visible'

},

paint: {

'fill-color': colors[p],

'fill-opacity': 0.6

},

filter: filters

});//end addLayer()

**REFERENCES**

[1] TopoJSON https://github.com/mbostock/topojson