San Francisco Food Trucks

Requirements & Design & Implementation & Host & More

Preview

This web application is designed, implemented and hosted all by myself within 5 days. It best illustrates my full-stack skillsets, including both front-end and back-end tracks. Even I am most familiar with back-end track, I still tried very hard to render a simply and easy use front page.

This web service not only satisfies the requirements, but also provides more additional features, which will greatly improve the user experiences.

1. Project Overview

Provide San Francisco Food Trucks web service. It utilizes a minimal amount of source files to support all the functionalities, including a front page (index.html), a style sheet (home.css), a javascript API code file (map.js) and a folder containing all the supportive images.

1.1 Objective

Create a service that tells the user what types of food trucks might be found near a specific location on a map. The data is available on DataSF: Food Trucks

1.2 User Cases

A user wants to check the food trucks around a specific location by specifying an address in any form (i.e. address, establishments, or zip code). The application retrieves all the food trucks location nearby, renders them in the map.

Displaying nearby food trucks is vague, the user should be able to specify the search distance like within 1 miles or a further distance.

Besides those, the user wants to know more about the nearby food trucks, like the name, the detailed location and food items provided.

After located a potential food truck to visit, the user probably want to get the direction information to that place by many travel modes, i.e. driving, transit, biking, or walking.

1.3 Risks

Due to the free host service, I can only gain limited resources. The first consideration is the scalability of serving a large amount of users.

1.4 Out of Scope

The data provided is only aimed within San Francisco. Therefore, I am lacking data to support the services outside of San Francisco.

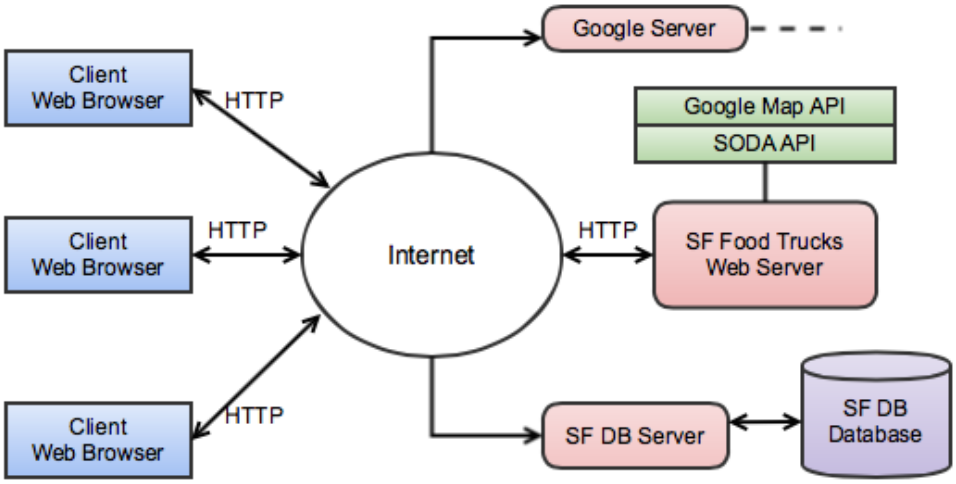
In addition, the data sets are not managed on an accessible database to me. Instead, I got the data from the dynamic link. Somehow, database is preferable like writing access to the file system (this app doesn't provide write access though); better performance in dealing with replication, concurrent access and data integrity etc.

2. Technical Design

According to the web service requirements, I planed to use HTML5, CSS, and JavaScript to realize the application. Because the data sets are provided thru the link, (https://data.sfgov.org/Economy-and-Community/Mobile-Food-Facility-Permit/rqzj-sfat?) i.e. I don't need to create or maintain any backend database. At beginning, I have difficulty to grab those aimed information out of that dynamic website using regular expressions. However, finally, I figured this out by reading thru the developers' manual and it explains the way to use this datasets. The datasets can be exported as another link and then SODA API generates the correct URL to reference the corresponding JSON file.

- For the server side, Google Map APIs and SODA APIs provide all the major resources.
- On the other hand, the HTML, CSS and JavaScript files, I created, are run on the client side.

According to the above description, the following diagram shows a high level system view of this web service:



Many client web browsers are able to send HTTP requests passing thru the Internet, to my SF Food Trucks web server/s. On the server side, Google Map APIs and SODA APIs provides majority supports to my service. The SODA API (AKA Socrata Open Data API) helps the client to access a wealth of open data resources from SF Data website. The Google Map APIs supports all the map related displaying and calculating.

3. Implementation

Here I focus on the JavaScript code part utilizing Google Map APIs and some SODA APIs.

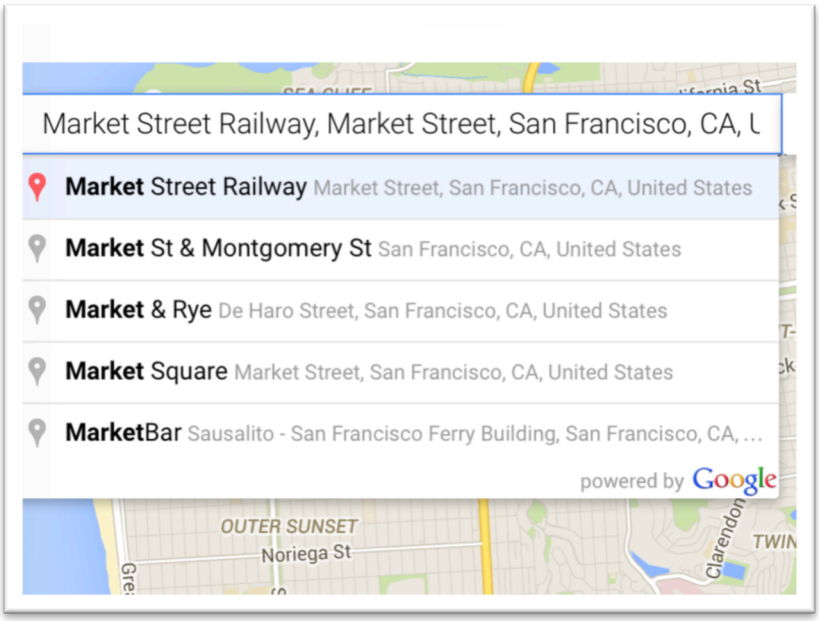
3.1 Load Google Map

In index.html, I declare the application as HTML5 using the <!DOCTYPE html> declaration; import Maps API JavaScript using a script tag; then create a div element named “map-canvas” to hold the Map. The following code snapshot shows the steps of map object creation and rendering. (37.7577, -122.4376) is the latitude and longitude of San Francisco.

```
$(window).load(function() {
    var mapOptions = {
        center: new google.maps.LatLng(37.7577, -122.4376),
        zoom: 13
    };
    map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);
```

3.2 Place Autocomplete

The Place Autocomplete is a web service that returns place predictions in response to an HTTP request. The request specifies a textual search string and optional geographic bounds. The service can be used to provide autocomplete functionality for text-based geographic searches, by returning places such as business, addresses and points of interest as a user types, shown as the following picture,



When the map loaded, I created an autocomplete object bound to the map object and when the autocomplete text box changed, it automatically provides a geometry object back based on the input string. The code snippet is shown below,

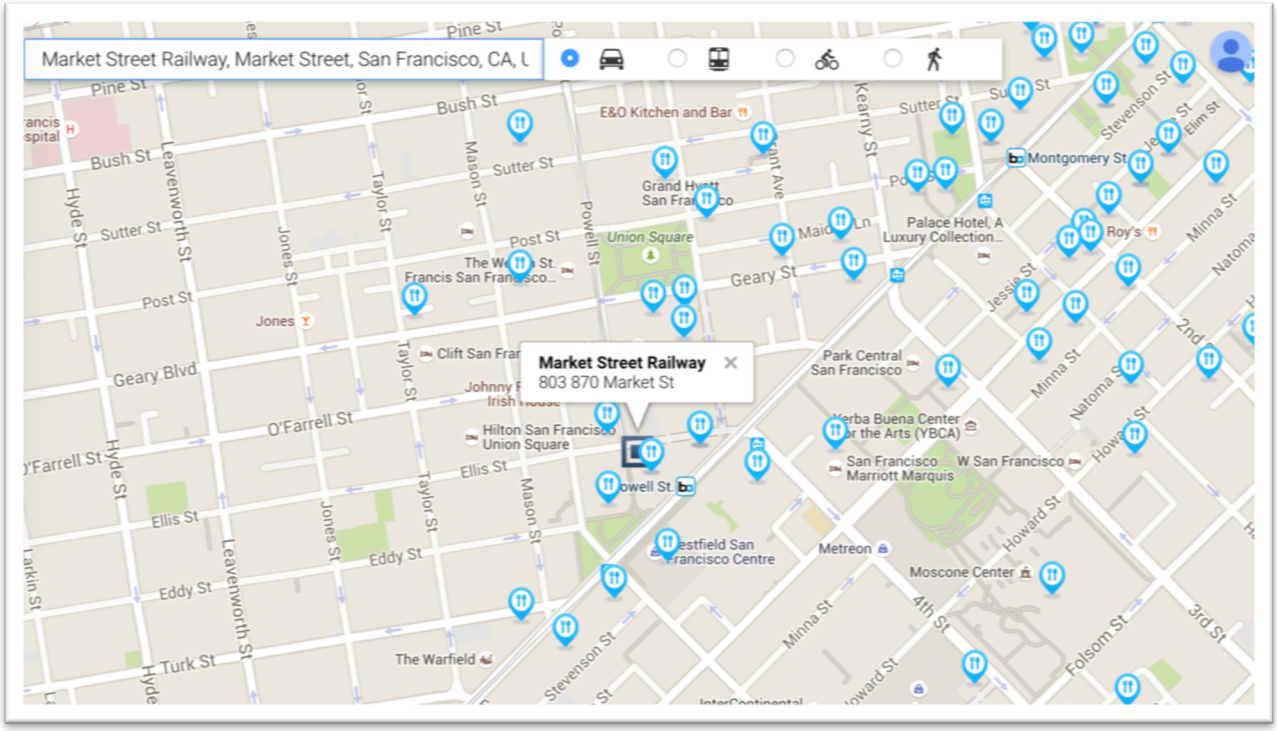
```
place = autocomplete.getPlace();
if (!place.geometry) {
    window.alert("Autocomplete's returned place contains no geometry");
    return;
}

// If the place has a geometry, then present it on a map.
if (place.geometry.viewport) {
    map.fitBounds(place.geometry.viewport);
} else {
    map.setCenter(place.geometry.location);
    map.setZoom(16); // Why 16? Because it looks good.
}
marker.setIcon( /** @type {google.maps.Icon} */ ({
    url: place.icon,
    size: new google.maps.Size(71, 71),
    origin: new google.maps.Point(0, 0),
    anchor: new google.maps.Point(17, 34),
    scaledSize: new google.maps.Size(35, 35)
}));
marker.setPosition(place.geometry.location);
marker.setVisible(true);
```

3.3 Load Food Trucks Locations

When autocomplete got updated, the JavaScript starts loading nearby food trucks locations by a jQuery call combined with SODA API to get each JSON object back. I just need to extract the information I needed out. The information includes location latitude, longitude for marker generation, the detailed address, the name of applicant as well as the food items for information window display.

```
// Construct the catalog query string
url = 'https://data.sfgov.org/resource/rqzj-sfat.json';

// Retrieve our data and plot it
$.getJSON(url, function(data, textstatus) {
    // console.log(data);
    var i = 0;
    $.each(data, function(i, entry) {
```

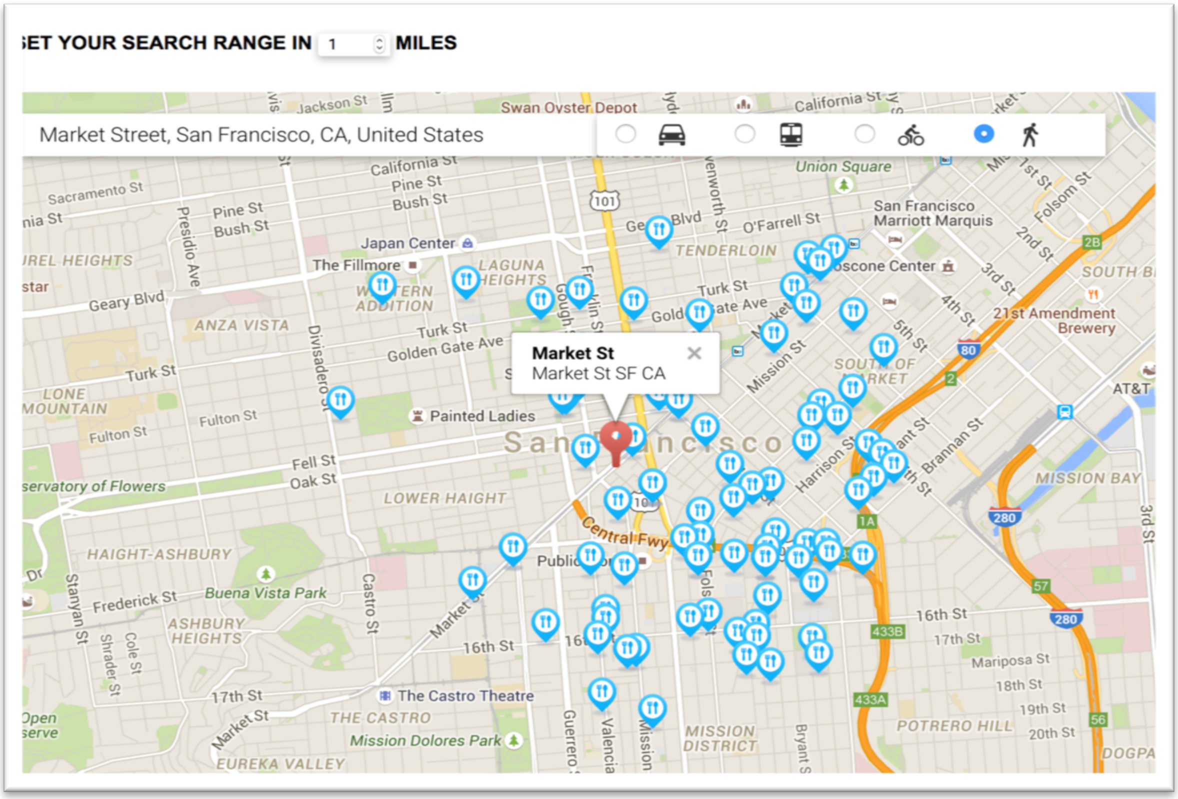
The picture below gives a example of above functionalities, rendering nearby food trucks markers,

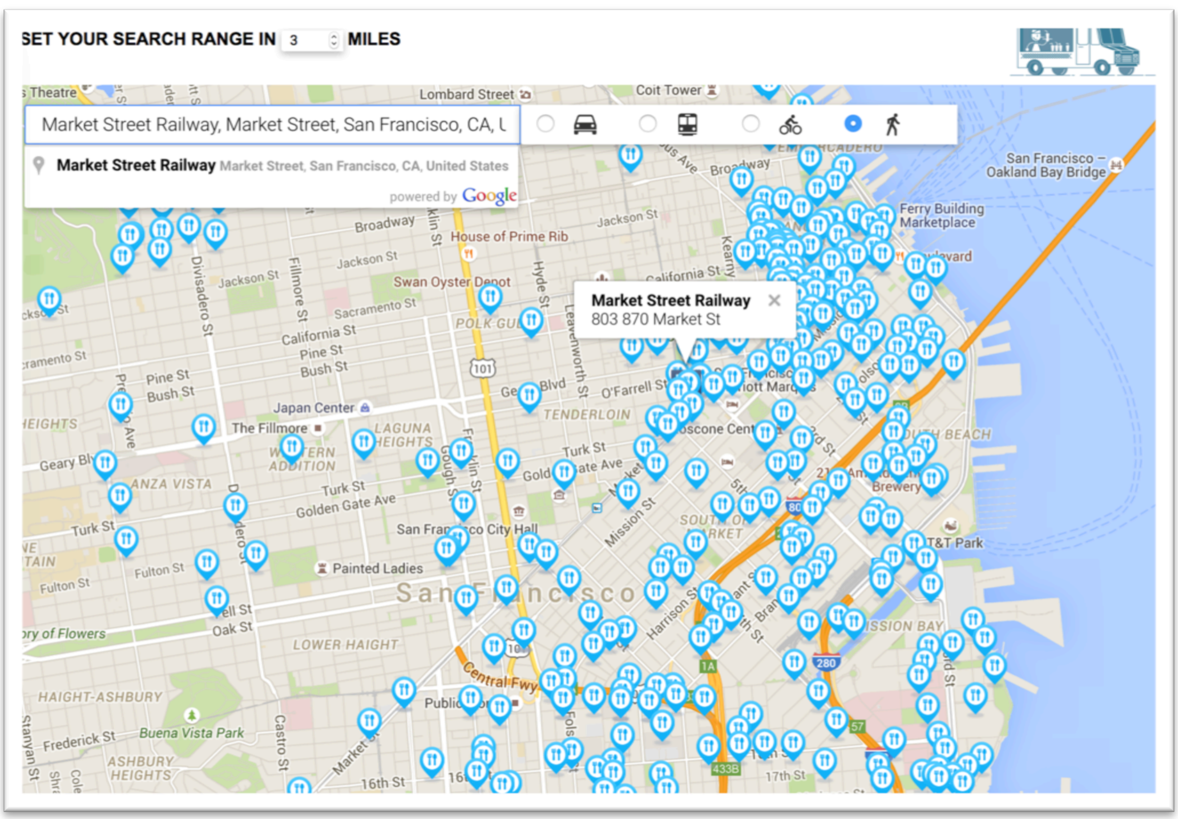


3.4 Searching Range

The requirement is to search the nearby food trucks around a place. The “nearby” is too vague to be implemented. Therefore, I provide an input box to let users decide the searching radius in miles. Then based on this range to render the food trucks markers. There is a function google.maps.geometry.spherical.computeDistanceBetween() does the distance calculation job for us. The effect picture shows below. By default, the search range is 1 mile.

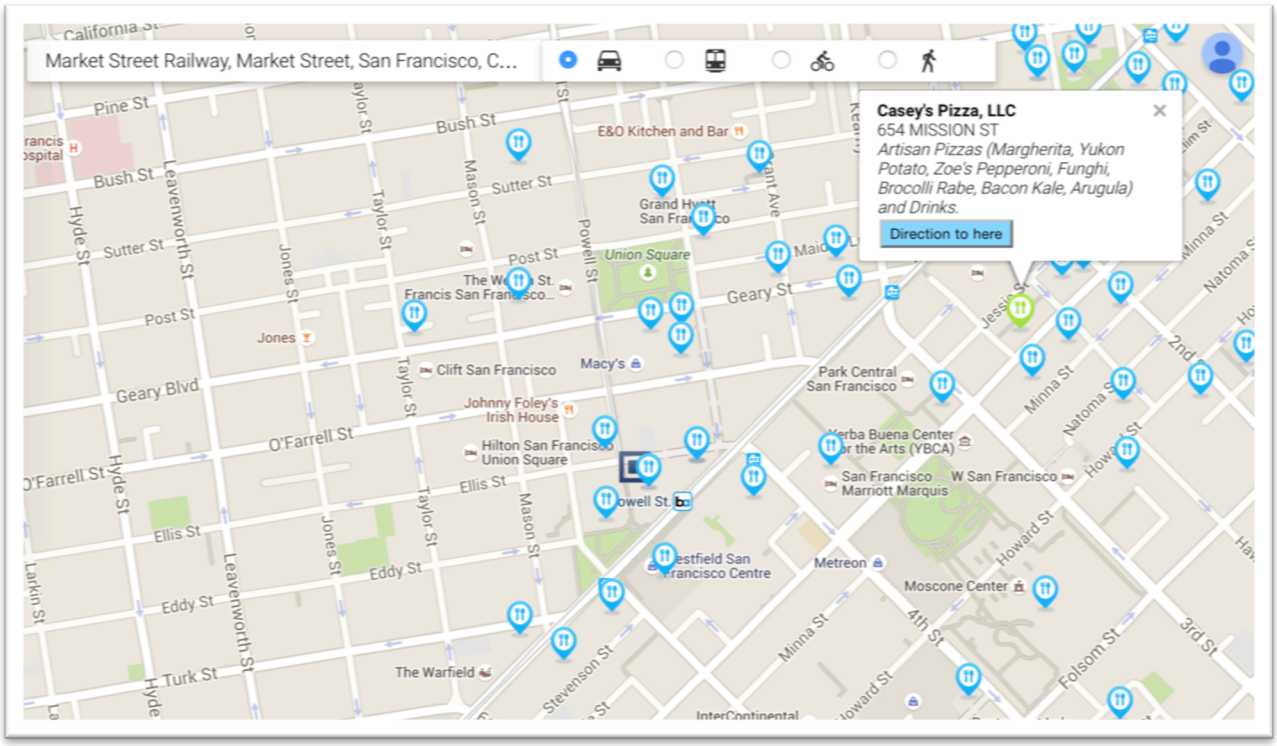
I zoom out the map, in order to give you better view about the range updates, from 1 mile to 3 miles.


SET YOUR SEARCH RANGE IN 1 MILES
Market Street, San Francisco, CA, United States
Market St
Market St SF CA


SET YOUR SEARCH RANGE IN 3 MILES
Market Street Railway, Market Street, San Francisco, CA, U
Market Street Railway Market Street, San Francisco, CA, United States
powered by Google
Market Street Railway
803 870 Market St

3.5 Information Window

Click on each food truck marker, it pops up an info window displaying more detailed information about this food truck, including the applicant name, the exact address, and food items provided. You may notice there is a “Direction to here” button embedded, which is described in the next section. A nice feature is that, when you click on the marker, the color and size of that marker changed. Since I used a global markers array to hold all markers, every time, the location changed, the markers array should be cleared out. Besides, in order to switch the previous selected marker color back to blue, I used another global variable to store the previous selected index in the markers array.



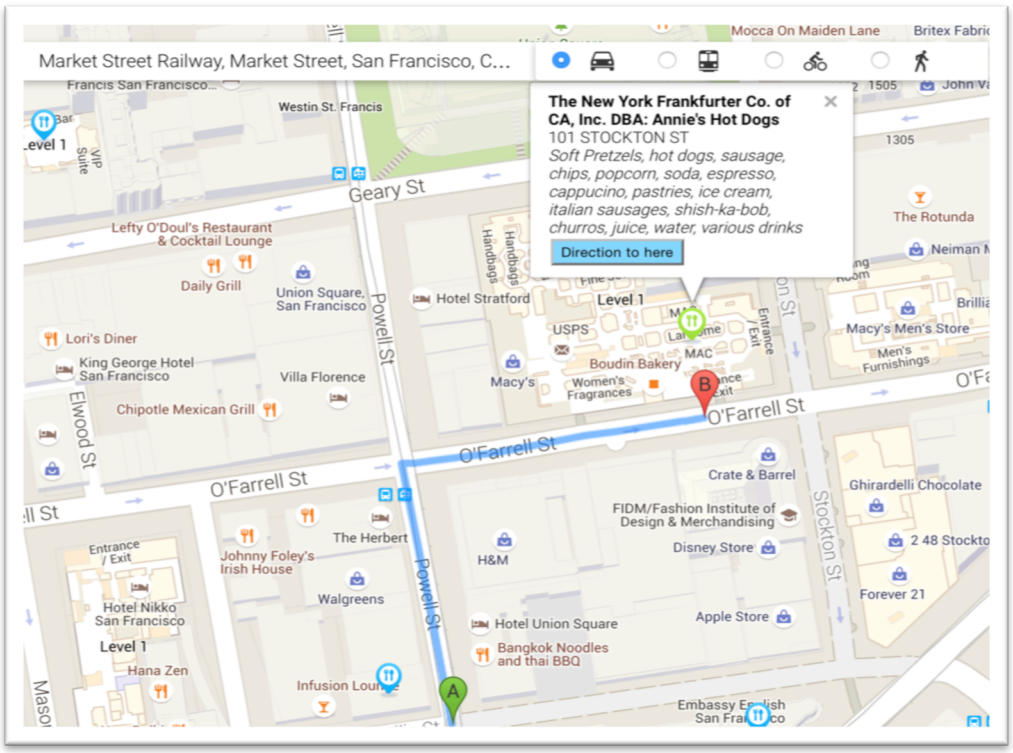
3.6 Directions

As shown above, each food truck information window has a “Direction to here” button. Whenever, the user clicks on the button. A route calculated will be displayed between the location user specified to the food truck. This feature is implemented based on Google direction services. Set up the directions display object when the map loaded; then calculate, render the route from the start to the end location object.

```
directionsDisplay = new google.maps.DirectionsRenderer();
directionsDisplay.setMap(map);
```

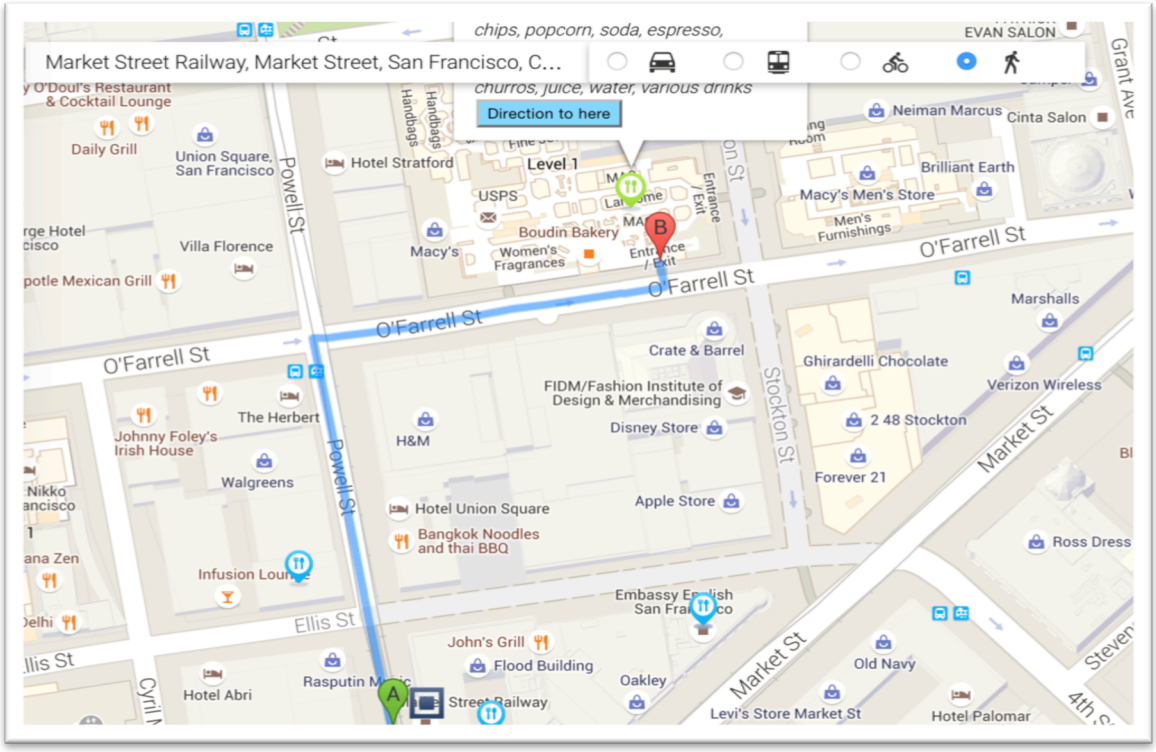
```
function calcRoute() {
    var start = place.geometry.location;
    var end = markersArray[selectedMarkerId].position;
    var request = {
        origin: start,
        destination: end,
        travelMode: travelWay //google.maps.TravelMode.DRIVING
    };
    directionsService.route(request, function(response, status) {
        if (status == google.maps.DirectionsStatus.OK) {
            directionsDisplay.setDirections(response);
        }
    });
}
```

The example routing picture is shown below,



3.7 Travel Modes

The above picture shows the driving direction. I wonder the users usually prefer biking or walking to grab some nearby hotdog. Therefore, it's reasonable to add travel modes options for the users. The following example image shows the walking route.



Fundamentally, the travel mode options are implemented as radio buttons. Each time when the user changes the travel mode, it will update a global variable called travelWay. Here is the code snippet,

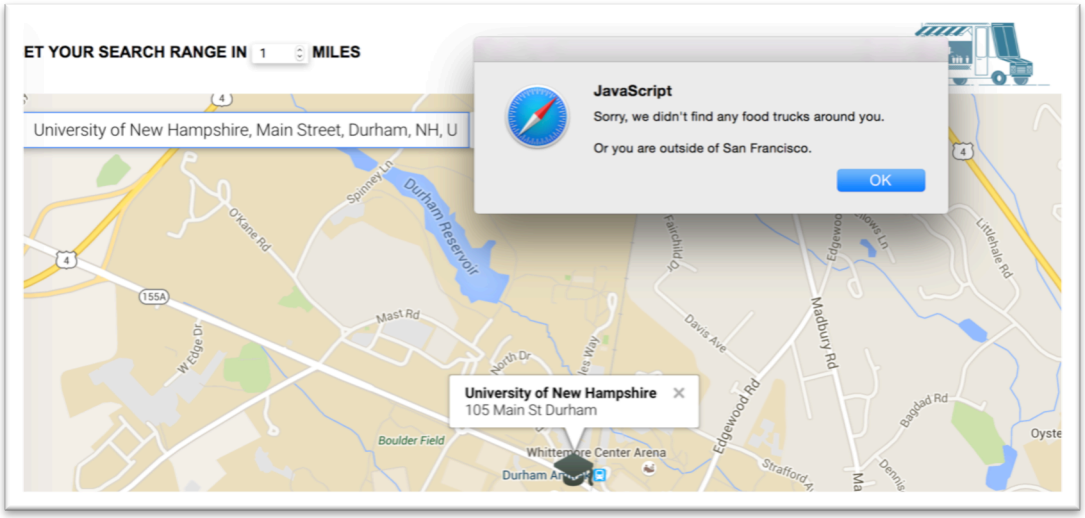
```
setupClickListener('Car');
setupClickListener('Transit');
setupClickListener('Bicycle');
setupClickListener('Walk');

// Set a listener on each radio button for different travel modes.
function setupClickListener(id) {
    var radioButton = document.getElementById(id);
    google.maps.event.addDomListener(radioButton, 'click', function() {
        //autocomplete.setTypes(types);
        var selectedMode = document.getElementById(id).value;
        travelWay = google.maps.TravelMode[selectedMode];
    });
}
```

3.8 More

Since we only provide food trucks information within San Francisco, how about the requested places are outside of San Francisco? There should have some notification pop up. It is not that straightforward to confine the borders of San Francisco. Then compare the place with the San Francisco borders to see whether it's within or not. An easy but naïve solution I found, is using a Boolean variable to keep track whether we found any food trucks around the place or not. If didn't find anything, the place is probably outside of SF.

```
// We didn't find any food trucks around you since you may be out of SF.
if (nothingFound == true)
    alert("Sorry, we didn't find any food trucks around you.\
    \nOr you are outside of San Francisco.");
```

The alert looks like this,



One more little feature, the food truck icon on the top-right corner is a link back to the web service homepage, works as refresh. And the image will change when the mouse hovers,

When the mouse hovers,

5. Host

Tried Amazon EC2 first, however the servers were down for those two days. Then explored Google App Engine, which is too complicate to hold my simple web applications. Eventually,

I found www.biz.nf is a popular web server provider. After uploading all the source files to the file manager, it generates an URL to visit,

http://sffoodtrucks.co.nf

6. Summary

I am quite familiar with data structure and algorithm. So, when I am coding with Google Map APIs/middleware level, I feel very comfortable and code very efficiently. During this web service development, I worked pretty hard to learn the front-end techniques like CSS and HTML by reading tons of tutorials.