INFO3067 Week 6 Class 2

Review

- Joins in Linq
- Web API
- getJSON

Today's Overview

Today's work is based on an article found on a popular asp.net website**: Four Guys from Rolla**. The article in question can be found with this URL:

http://www.4guysfromrolla.com/articles/081810-1.aspx

We're going to **make substantial changes** to the code in the article to get the google code to interact in our n-tier framework, but the overall functionality will work the same. The premise we're going to use for this page is that we will provide a mechanism for our users to be able to look up and display a map. **The map will provide the locations of the eStore's 3 closet branches** from any address on the globe.

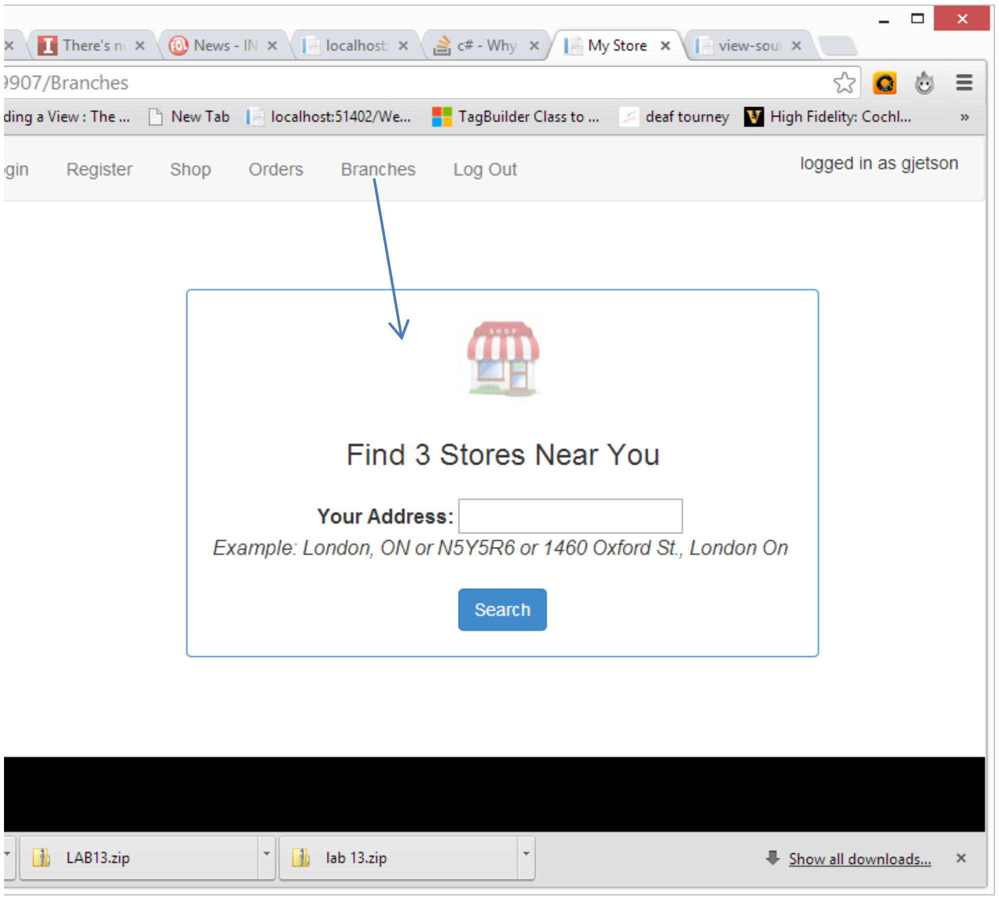
I have provided the SQL in a zip file on FOL that will create and populate a **Branches** table (see if you can figure out where I got the data for this table). You will need to run this script **and introduce the new table to the EF** before proceeding.

I have also provided the images used on this page:

- **marker1.png, marker2.png, marker 3.png** (needs to be placed in **images** folder)

To start with, place a **Branches** link on you menu to execute a new controller **BranchesController** and a new View to handle the default **Index** method. The new view should prompt the user enter the address from which they want to find the 3 closest branches. In the view's markup include a link to the google map service:

```
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

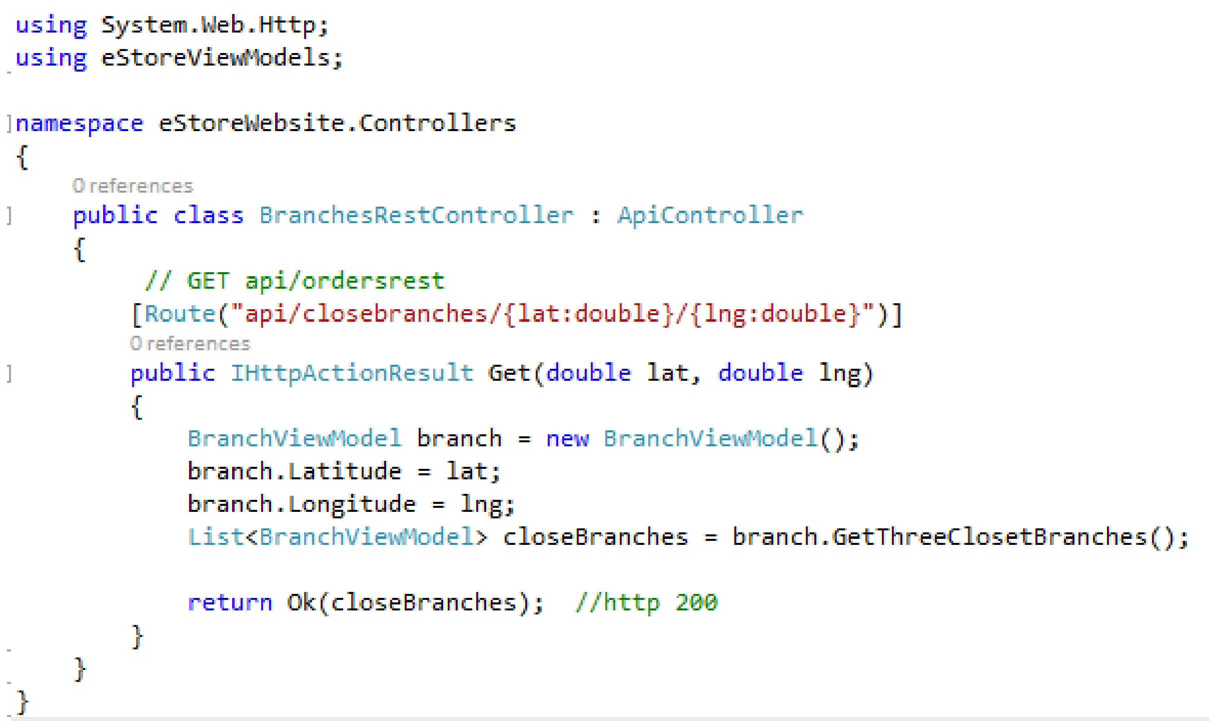


In the View markup, code the text box with an id attribute of **address** and the button with an id attribute with a value of **findthem**,. Next, code a JQuery event handler for the findthem button that has this code:

```
// event handler for finding branch button click
$('#findthem').click(function () {
    var address = $("#address").val(); // address textbox
    geocoder = new google.maps.Geocoder();                               // A service for converting between an a
    geocoder.geocode({ 'address': address }, function (results, status) {
        if (status == google.maps.GeocoderStatus.OK) {                          // only if google gives us the OK
            var lat = results[0].geometry.location.lat();
            var lng = results[0].geometry.location.lng();
            $.getJSON("api/closebranches/" + lat + "/" + lng +"/", null, function (locations, textStatus, jqXHR) {
                init_map(lat, lng, locations);
            }).error(function (jqXHR, textStatus, errorThrown)
            {
                console.log(textStatus + " - " + errorThrown);
                alert("there was a problem connecting to the host, try again later")
            });
        }
    });
}); //findthem
```

This routine will pass the entered address to **google's geocoder service** as the first parameter. An inline method that processes the returning data is the second parameter of this method call and will execute when google returns data. This data includes the **latitude** and **longitude** of the address. The method code then directly calls a Web API service using the getJSON function we looked at last class. Notice that we also pass the latitude and longitude as doubles.

Server side the service method should contain something like:



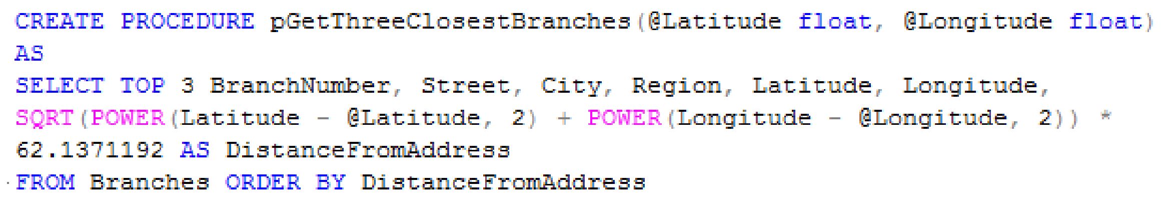
```
using System.Web.Http;
using eStoreViewModels;

namespace eStoreWebsite.Controllers
{
    0 references
    public class BranchesRestController : ApiController
    {
         // GET api/ordersrest
        [Route("api/closebranches/{lat:double}/{lng:double}")]
        0 references
        public IHttpActionResult Get(double lat, double lng)
        {
            BranchViewModel branch = new BranchViewModel();
            branch.Latitude = lat;
            branch.Longitude = lng;
            List<BranchViewModel> closeBranches = branch.GetThreeClosetBranches();

            return Ok(closeBranches);  //http 200
        }
    }
}
```

From here we see we have a new view model class called **BranchViewModel** which in turn will call a corresponding **BranchModel** class, because we're a little tight on time I have provided the code for these 2 classes on FOL.

These classes will generate some errors until you create a stored procedure and configure a function called **GetThreeClosestBranches** and a complex type called **ClosestBranchDetails** in the Entity Framework. The stored procedure calculates the distance between two points using this formula

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

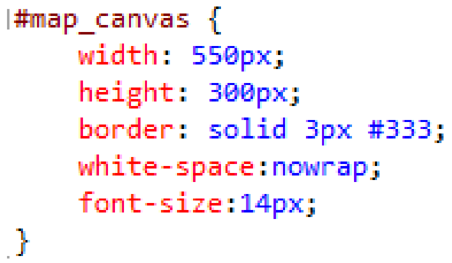
So in sql we would write this as:



```
CREATE PROCEDURE pGetThreeClosestBranches(@Latitude float, @Longitude float)
AS
SELECT TOP 3 BranchNumber, Street, City, Region, Latitude, Longitude,
SQRT(POWER(Latitude - @Latitude, 2) + POWER(Longitude - @Longitude, 2)) *
62.1371192 AS DistanceFromAddress
FROM Branches ORDER BY DistanceFromAddress
```

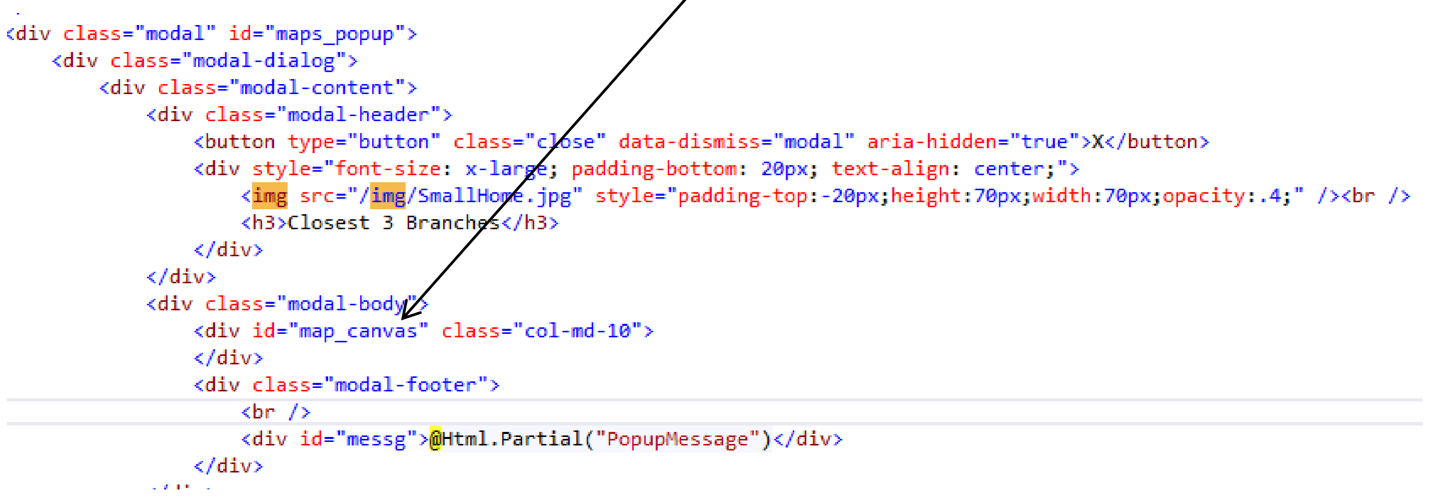
The original code asked for a location within 15 miles, I've taken that requirement out. For a full explanation of what he's doing with this SQL see the article (basically uses the Pythagorean Theorem for object location).

The result of these two classes is that the controller method receives a List of BranchViewModel instances which in turn get converted to JSON format and returned to the original script: in the variable called **locations** (see above). The JQuery routine then calls one last JQuery routine called **init_map**. Again, for time sakes I am providing code for this routine you will have to adjust to meet your styling needs. Take note of a couple of lines in this code as they are the keys to getting this working. The first call to **google.maps.Map** sets up the actual map and then 3 subsequent calls to **google.maps,Marker** places the actual markers on the map. The init_map routine will look for an element called **map_canvas** to place the google map. You will need to add a couple of more items to get the map to render. First add some styling in a style sheet to set up the size characteristics for the map



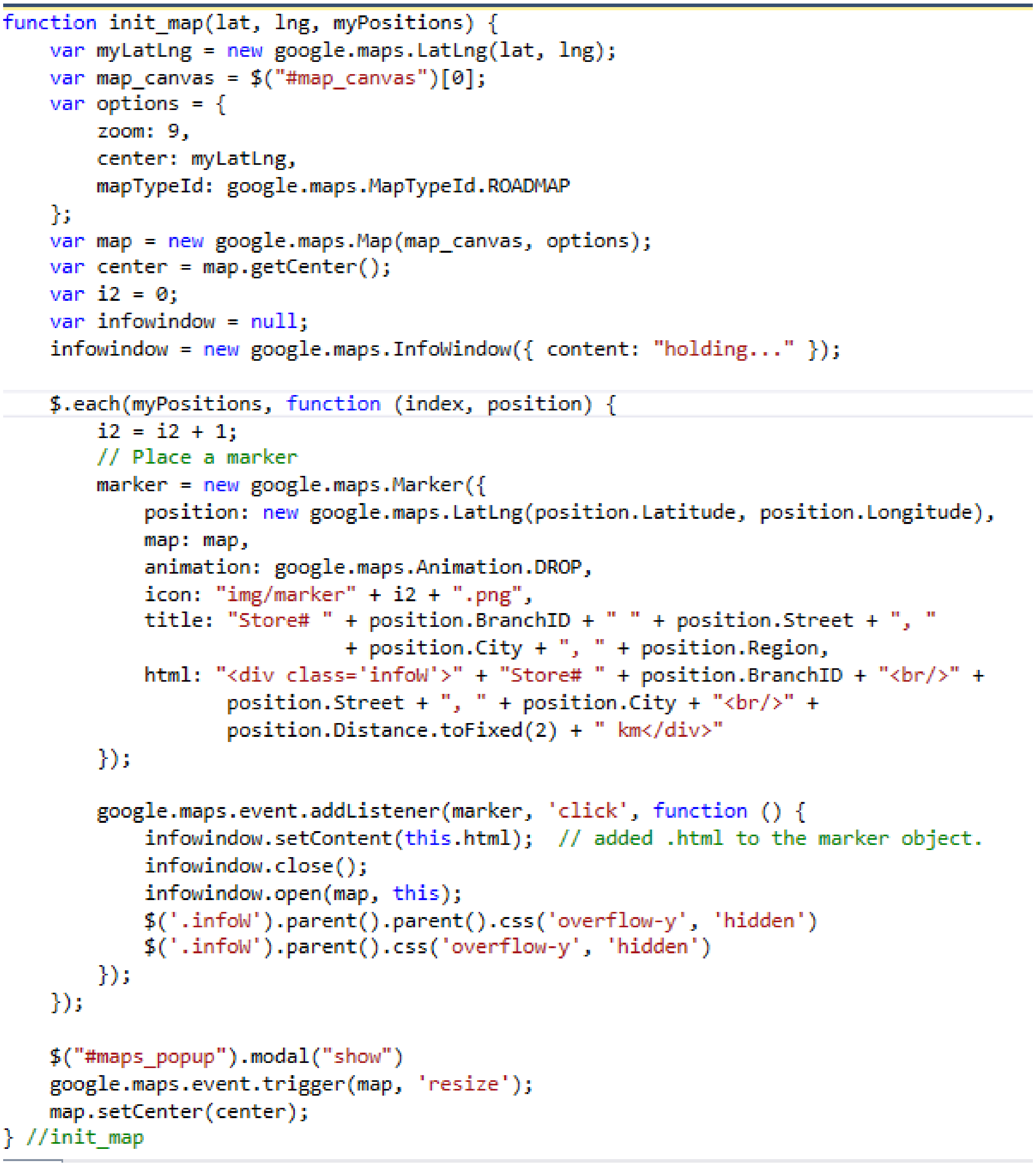
```
#map_canvas {
    width: 550px;
    height: 300px;
    border: solid 3px #333;
    white-space:nowrap;
    font-size:14px;
}
```

And lastly you need to provide a div called **map_canvas** that the init_map routine will place all of the output in:



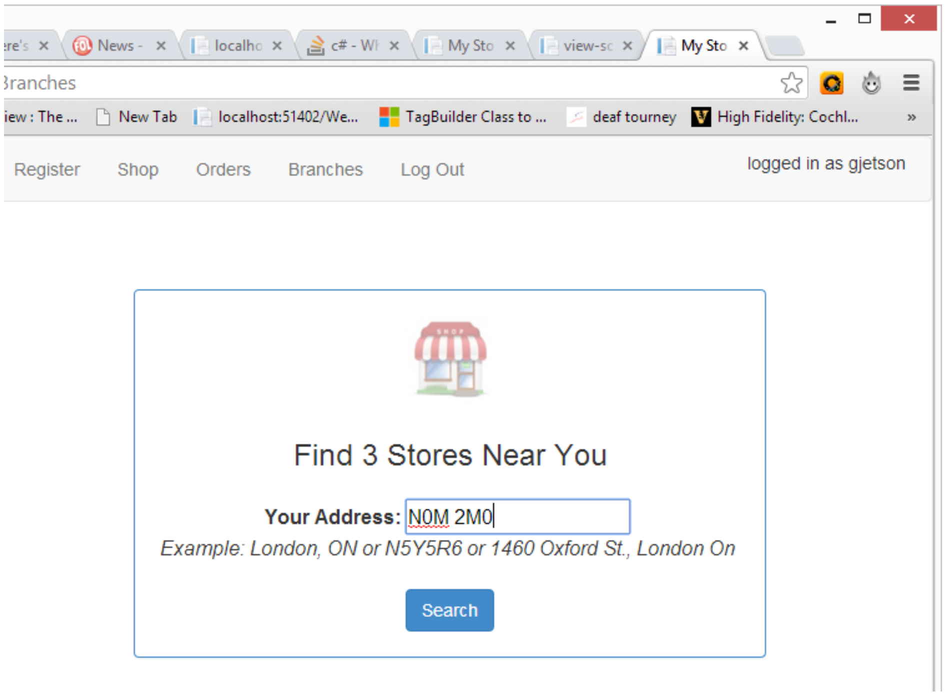
```
<div class="modal" id="maps_popup">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-hidden="true">X</button>
                <div style="font-size: x-large; padding-bottom: 20px; text-align: center;">
                    <img src="/img/SmallHome.jpg" style="padding-top:-20px;height:70px;width:70px;opacity:.4;" /><br />
                    <h3>Closest 3 Branches</h3>
                </div>
            </div>
            <div class="modal-body">
                <div id="map_canvas" class="col-md-10">
                </div>
                <div class="modal-footer">
                    <br />
                    <div id="messg">@Html.Partial("PopupMessage")</div>
                </div>
```

The findthem routine in turn calls another routine called **init_map**, the focus of this routine is to render the map complete with marker icons for the 3 closest locations. This routine is configurable by changing the **options** variable. Also in this routine the markers are configured complete with a “click” event handler written with some JQuery that fixes a scroll bar issue in Chrome.



```
function init_map(lat, lng, myPositions) {
    var myLatLng = new google.maps.LatLng(lat, lng);
    var map_canvas = $("#map_canvas")[0];
    var options = {
        zoom: 9,
        center: myLatLng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var map = new google.maps.Map(map_canvas, options);
    var center = map.getCenter();
    var i2 = 0;
    var infowindow = null;
    infowindow = new google.maps.InfoWindow({ content: "holding..." });

    $.each(myPositions, function (index, position) {
        i2 = i2 + 1;
        // Place a marker
        marker = new google.maps.Marker({
            position: new google.maps.LatLng(position.Latitude, position.Longitude),
            map: map,
            animation: google.maps.Animation.DROP,
            icon: "img/marker" + i2 + ".png",
            title: "Store# " + position.BranchID + " " + position.Street + ", "
                                 + position.City + ", " + position.Region,
            html: "<div class='infoW'>" + "Store# " + position.BranchID + "<br/>" +
                   position.Street + ", " + position.City + "<br/>" +
                   position.Distance.toFixed(2) + " km</div>"
        });

        google.maps.event.addListener(marker, 'click', function () {
            infowindow.setContent(this.html);  // added .html to the marker object.
            infowindow.close();
            infowindow.open(map, this);
            $('.infoW').parent().parent().css('overflow-y', 'hidden')
            $('.infoW').parent().css('overflow-y', 'hidden')
        });
    });

    $("#maps_popup").modal("show")
    google.maps.event.trigger(map, 'resize');
    map.setCenter(center);
} //init_map
```

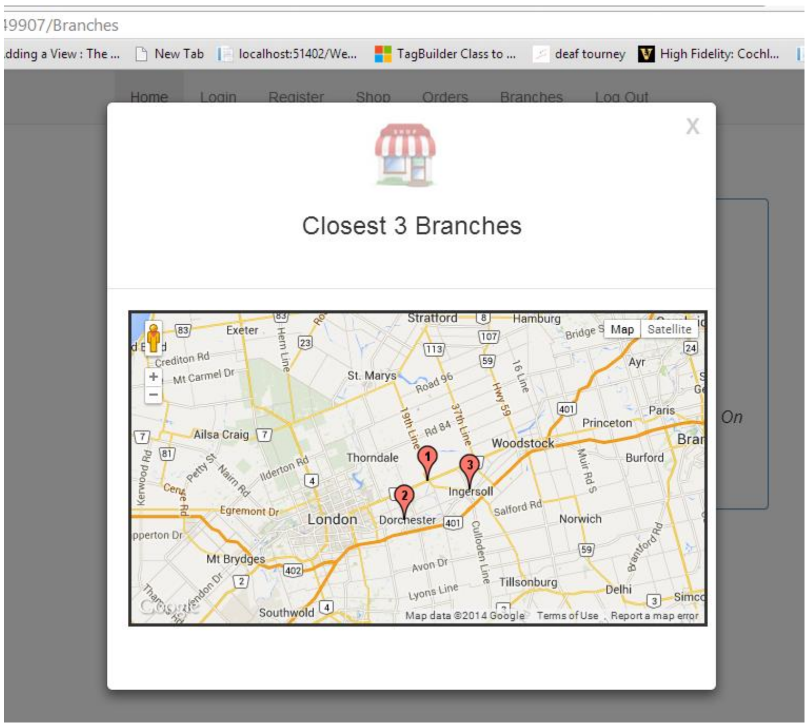
Notice how everything is in a for loop, and how the individual marker contents are loaded using the data passed back from the Web API method, If everything is in place try out your home address and you should see the following:



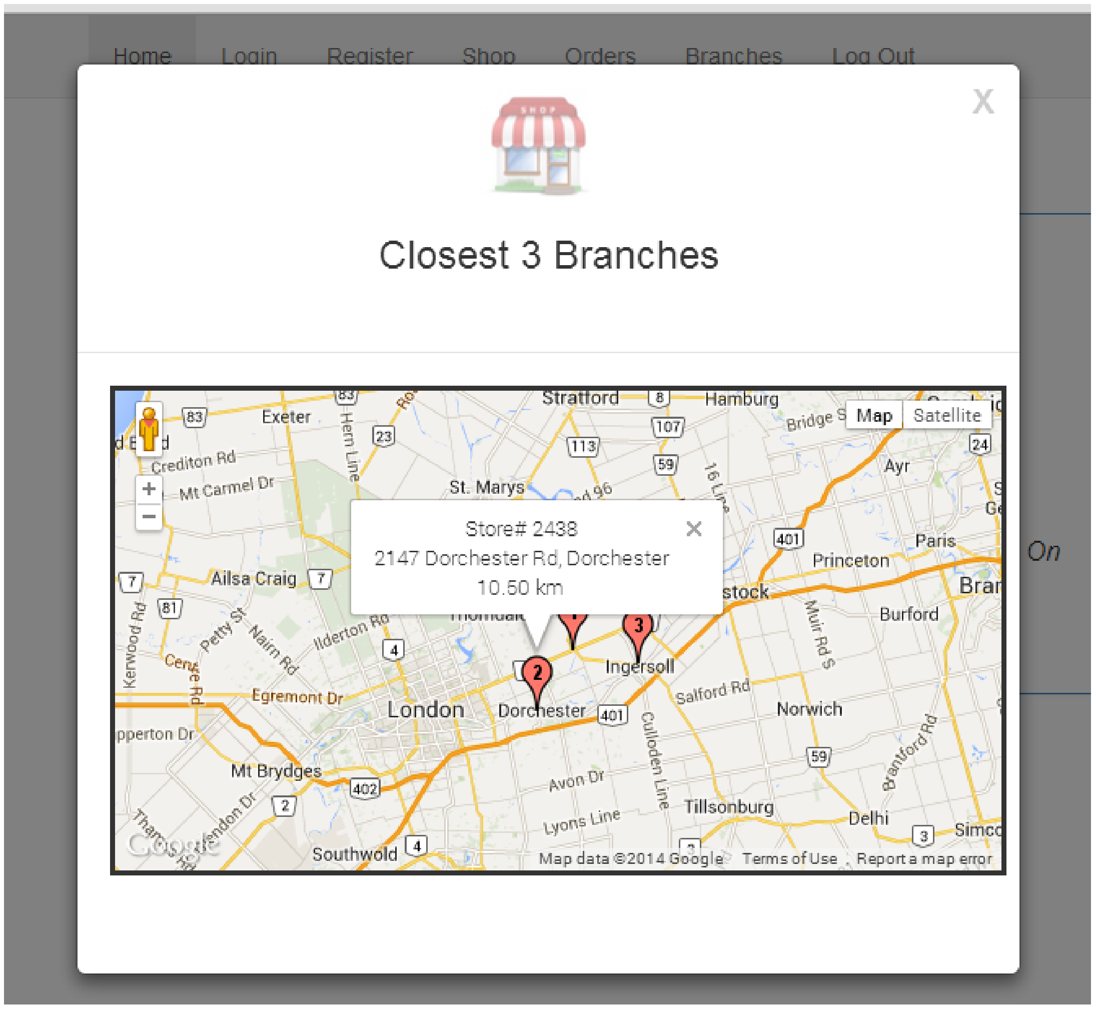
Find 3 Stores Near You

Your Address: N0M 2M0

Example: London, ON or N5Y5R6 or 1460 Oxford St., London On



And lastly, clicking any of the markers, should present the additional bubble popup containing the individual branch's information:



Closest 3 Branches

Store# 2438
2147 Dorchester Rd, Dorchester
10.50 km

LAB 10

- Extract and place all of the helper code and files in their correct locations
- Code the necessary pieces from the code in this pdf
- Submit a screen shot of the map showing the 3 closest locations to **your home postal code** and click on marker 1 to show the bubble popup contents