INFO3067 Week 6 Class 2

Rev 1.2

Today's Overview

Today's work is based on an article found on a popular asp.net website**: Four Guys from Rolla**. The article is really old now, but the idea is still good a good one, it can be found with this URL:

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

Most of the code is not relevant to our setup so don't look at the code in the article too closely (except how he uses the Pythagorean Theorem to calculated distances), just get a feel for what the author was intending.

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 store's 3 closet stores** (assuming we're a bricks and mortar org.) from any address in Ontario.

3 Closest Stores

1. I have provided some data for the locations in the form of a comma separated file (csv) in a file called **storeRaw.csv** on FOL. Download this file and place it in the wwwroot folder of your exercises project.

2. Update the DataController so the constructor and a new private variable look like this (note you'll need a using for Microsoft.AspNet.Hosting;)

```
IHostingEnvironment _env;
public DataController(AppDbContext context, IHostingEnvironment env)
{
    _db = context;
    _env = env;
}
```

3. Add a new method in the DataController called **Csv** with the following code:

```
public IActionResult Csv()
{
    StoreModel model = new StoreModel(_db);
    bool storesLoaded = model.LoadFromFile(_env.WebRootPath);

    if (storesLoaded)
        ViewBag.CsvLoadedMsg = "Csv Loaded Successfully";
    else
```

```
            ViewBag.CsvLoadedMsg = "Csv NOT Loaded";

        return View("Index");
    }
```

4. Add a new file called **StoreModel.cs** in the Models folder and place the following code in it:

```
using Microsoft.Data.Entity;
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;

namespace ASPNetExercises.Models
{
    public class StoreModel
    {
        private AppDbContext _db;

        public StoreModel(AppDbContext context)
        {
            _db = context;
        }

        public bool LoadFromFile(string path)
        {
            bool addWorked = false;
            try
            {
                // clear out the old rows
                _db.Stores.RemoveRange(_db.Stores);
                _db.SaveChanges();

                var csv = new List<string[]>();
                var csvFile = path + "\\storeRaw.csv";
                var lines = System.IO.File.ReadAllLines(csvFile);

                foreach (string line in lines)
                    csv.Add(line.Split(',')); // populate store with csv

                foreach (string[] x in csv)
                {
                    Store aStore = new Store();
                    aStore.Longitude = Convert.ToDouble(x[0]);
                    aStore.Latitude = Convert.ToDouble(x[1]);
                    aStore.Street = x[2];
                    aStore.City = x[3];
                    aStore.Region = x[4];
                    _db.Stores.Add(aStore);
                    _db.SaveChanges();
                }
                addWorked = true;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            return addWorked;
        }
```

```
        public List<Store> GetThreeClosetStores(float? lat, float? lng)
        {
            List<Store> storeDetails = null;
            try
            {
                var latParam = new SqlParameter("@lat", lat);
                var lngParam = new SqlParameter("@lng", lng);
                var query = _db.Stores.FromSql("dbo.pGetThreeClosestStores @lat = {0}, @lng =
{1}", lat, lng);
                storeDetails = query.ToList();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            return storeDetails;
        }
    }
}
```

5. Update the **Views/Data/Index.cshtml** code and replace the contents with the following:

```
<div style="top:50px;position:relative">
    <div class="col-xs-12">
        <span class="col-xs-2"><span id="busy"></span></span>
        <span class="col-xs-2">
            <button id="jsondatartns" class="btn btn-default btn-block">Menu Json</button>
        </span>
        <span class="col-xs-1"> </span>
        <span class="col-xs-7">@ViewBag.JsonLoadedMsg</span>
    </div>
    <div class="col-xs-12">
        <span class="col-xs-2"> </span>
        <span class="col-xs-2">
            <button id="csvdatartns" class="btn btn-default btn-block">Store CSV</button>
        </span>
        <span class="col-xs-1"> </span>
        <span class="col-xs-7">@ViewBag.CsvLoadedMsg</span>
    </div>
</div>
```

6. Update the exercises.js file and add the following code to the main jQuery routine:

```
$("#jsondatartns").on("click", function (e) {
    busySignal("/Data/Json");
});

$("#csvdatartns").on("click", function (e) {
    busySignal("/Data/Csv");
});
```

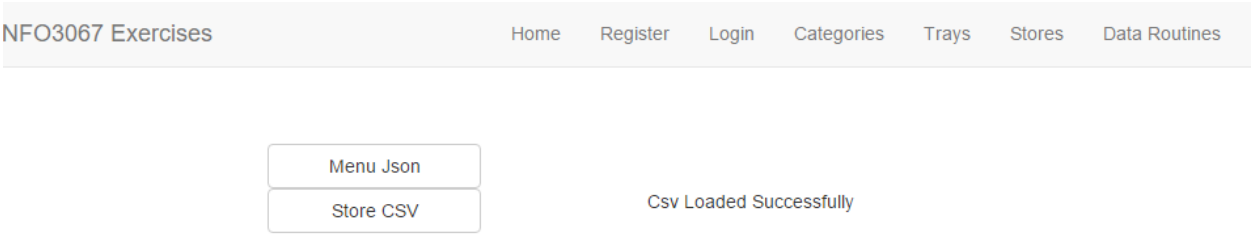
7. Update the exercises.js file and add the following function:

```
function busySignal(url)
{
    var busyImg = $('<img/>', {
```

```
        src: "/img/wait.gif"
    });

    $("#busy").empty();
    $("#busy").append(busyImg);
    window.location.href = url;
}
```

8. Download a file called **wait.gif** from FOL and place it in your img folder

9. Run the application and choose the Data Routines menu item, then **click the CSV** button:



10. Open a query window to the database you're working with (Azure/LocalDb) Create a stored procedure called **pGetThreeClosestStores** :

```
CREATE PROCEDURE pGetThreeClosestStores (@lat float, @lng float)
AS
SELECT TOP 3 Id, Street, City, Region, Latitude,
             Longitude, SQRT(POWER(Latitude - @lat, 2) + POWER(Longitude - @lng, 2)) *
             62.1371192 AS Distance  -- miles calculation
FROM Stores
ORDER BY Distance
```

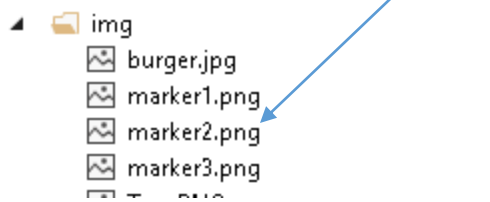
This stored procedure uses a derivative of the **Pythagorean Theorem** known as the **Distance Formula** which states: Given the two points (x1, y1) and (x2, y2), the distance between these points is given by the formula:

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

11. Modify the _Layout.cshtml file and add an entry for **Stores** with this code:

```
<li><a href="/Store">Stores</a></li>
```

12. Download the 3 marker images from FOL and place them in your **wwwroot\img** folder



13. Add a new controller called **StoreController.cs** and replace the generated code with the following:

```
using Microsoft.AspNet.Mvc;
using ASPNetExercises.Models;
using ASPNetExercises.Utils;
using Microsoft.AspNet.Http;

namespace eStore2016.Controllers
{
    public class StoreController : Controller
    {
        AppDbContext _db;
        public StoreController(AppDbContext context)
        {
            _db = context;
        }
        public ActionResult Index()
        {
            if (HttpContext.Session.GetString(SessionVars.Message) != null)
            {
                ViewBag.Message = HttpContext.Session.GetString(SessionVars.Message);
            }
            return View();
        }

        [Route("[action]/{lat:double}/{lng:double}")]
        public IActionResult GetStores(float lat, float lng)
        {
            StoreModel model = new StoreModel(_db);
            return Ok(model.GetThreeClosetStores(lat, lng));
        }
    }
}
```

14. Add a new view in a Views\Store folder called **Index.cshtml** with the following:

```
<div id="stores"></div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.1.0/react.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.1.0/react-dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react-bootstrap/0.29.4/react-
bootstrap.min.js"></script>
<script src="https://maps.googleapis.com/maps/api/js?v=3.exp&libraries=places"></script>
<script src="~/js/util.js"></script>
<script type="text/babel" src="~/js/store.jsx"></script>
```

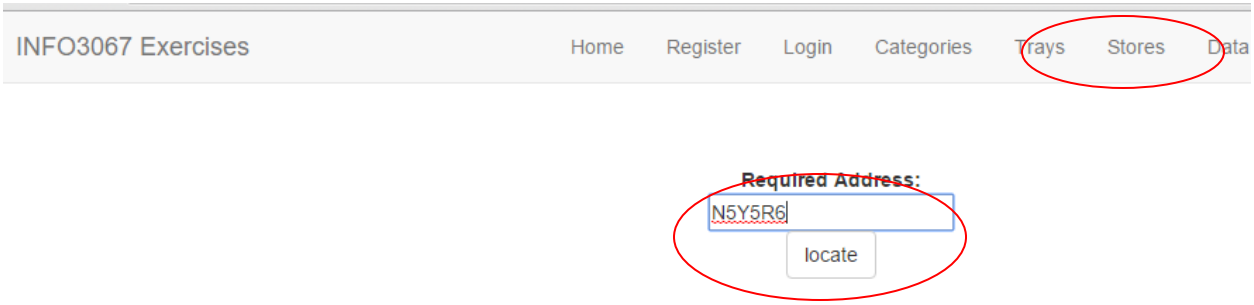
15. Create a new .jsx file called **store.jsx** in the wwwroot\js folder with the following contents:

```
//
// Bootstrap Component variables
//
var Modal = ReactBootstrap.Modal;
var Button = ReactBootstrap.Button;
```

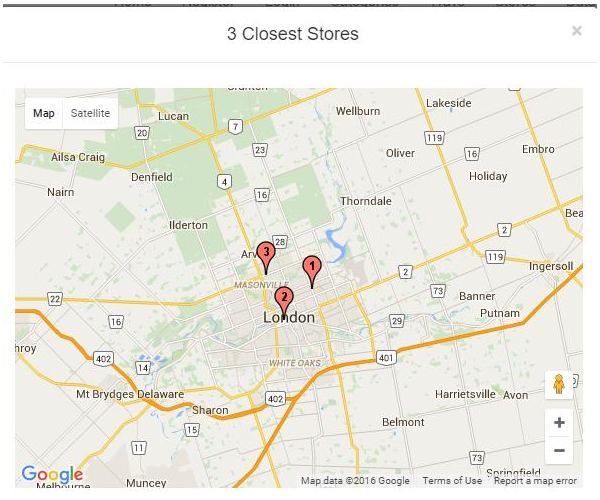
```
var mapStyle = {
    height: "100%",
    width: "100%",
    marginTop: "10px",
    minHeight: "400px"
}

//
//   StoreLocator Component
//
var StoreLocator = React.createClass({
    getInitialState() {
        return { showModal: false, branchdetails: [] };
    },
    close() {
        this.setState({ showModal: false });
    },
    open() {
        var _this = this;
        var lat, lng, map, i, marker;
        var address = this.refs.address.value.trim();
        var geocoder = new google.maps.Geocoder();   // A service for converting between an
address to LatLng

        geocoder.geocode({ "address": address }, function (results, status) {
            if (status == google.maps.GeocoderStatus.OK) { // only if google gives us the OK
                lat = results[0].geometry.location.lat();
                lng = results[0].geometry.location.lng();
                httpGet("/GetStores/" + lat + "/" + lng, function (data) {
                    var myLatLng = new google.maps.LatLng(lat, lng);
                    _this.setState({ showModal: true });
                    var map_canvas = _this.refs.map;
                    var options = {
                        zoom: 10,
                        center: myLatLng,
                        mapTypeId: google.maps.MapTypeId.ROADMAP
                    };
                    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..." });

                    for (i = 0; i < data.length; i++) {
                        i2 = i + 1;
                        marker = new google.maps.Marker({
                            position: new google.maps.LatLng(data[i].Latitude,
data[i].Longitude),
                            map: map,
                            animation: google.maps.Animation.DROP,
                            icon: "../img/marker" + i2 + ".png",
                            title: "Store# " + data[i].Id + " " + data[i].Street + ", "
                                            + data[i].City + ", " + data[i].Region,
                            html: "<div>" + "Store# " + data[i].Id + "<br/>" +
                                   data[i].Street + ", " + data[i].City + "<br/>" +
                                   data[i].Distance.toFixed(2) + " mi</div>"
                        });
                        google.maps.event.addListener(marker, 'click', function () {
                            infowindow.setContent(this.html);  // added .html to the marker
object.
                            infowindow.close();
                            infowindow.open(map, this);
```

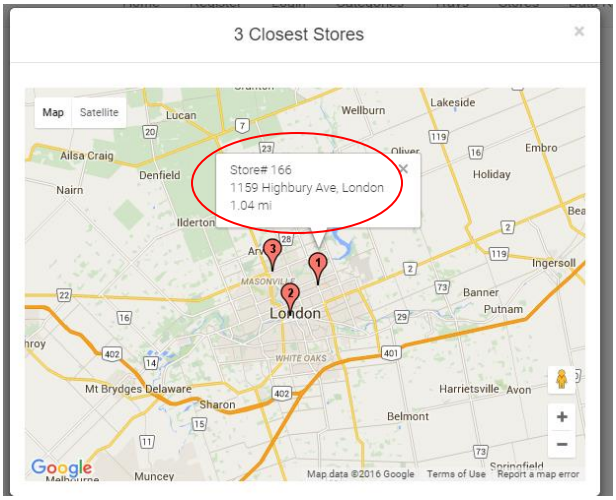
```
                        });
                    }
                    map.setCenter(center);
                    google.maps.event.trigger(map, "resize");

                }.bind(_this));
            }
        });

    },
    render: function () {
        return (
            <div style={{top:"50px", position:"relative"}}>
                <div className="col-sm-4 col-xs-1"> </div>
                <div className="std-box col-sm-4 col-xs-10 text-center">
                    <div>
                        <b>Required Address:</b><br />
                        <input type="text" ref="address" /><br />
                        <input type="button" onClick={this.open} value="locate" className="btn
btn-default"  />
                    </div>
                </div>
                <Modal show={this.state.showModal} onHide={this.close}>
                    <Modal.Header closeButton>
                        <Modal.Title>
                            <div className="text-center">3 Closest Stores</div>
                        </Modal.Title>
                    </Modal.Header>
                    <Modal.Body>
                            <div ref="map" id="map" style={mapStyle}></div>
                    </Modal.Body>
                </Modal>
            </div>
        )
    }
});

ReactDOM.render(
        <StoreLocator />,
        document.getElementById("stores") // html tag
)
```

16. Run the application, select the Stores option and test with an address or postal code like the college's postal code (or your own if you like)



17. Click on one of the markers (assuming it worked, if it didn't check the chrome browser's developer tools for an error)





18. If the info window (bubble popup) does not appear, try adding this style to the end of your style sheet :

```
#map_canvas img{max-width:none}
```

Notes on steps 1-18

1. The raw data in comma separated value format, make sure to place the file in the wwwroot folder as this is where the provided model code will be looking for it. Data was gleaned from a McDonald's restaurant source for Ontario establishments

2. Update to the DataController. The constructor should accept an additional parameter that is of type IHostingEnvironment. We need this to get at the application root (which is where the csv file is). We store the reference in a local variable called _env. You'll also need an additional using to get a reference to this variable.

3. A new method for the controller called Csv that will pass the path from the _env.WebRootPath property to the model layer, call on the model to add the data to the db, and finally pass a message back to the View.

4. The entire StoreModel. There are 2 methods in there. The **LoadFromFile** method that converts the csv file to the database, called the from the controller method in step 3.

   The other method **GetThreeClosestStores** method will receive a lat and lng parameter from client and execute a stored procedure that we create in step 9. Notice how the parameters are added and the results returned back to the controller

5. Updated View markup for Views\Data\Index.cshtml. The new markup utilizes both bootstrap classes and the exercises.js jQuery routines. Each button click will fire the jQuery routine and a message will be displayed when the corresponding message is returned.

6. The click event handlers for the 2 buttons, each will determine the controller url for the corresponding buttons. Both routines employ a common routine.

7. The common routine for calling the server's controller method

8. Loading app uses an animated gif, download the file to the correct location

9. Sample execution screen shot of the csv values being loaded

10. The stored procedure to calculate the distance between two points using the Pythagorean Theorum as discussed in the Rolla article. The procedure will receive the lat and lng values from the model and return a list Store instances

11. A new entry in the default layout's menu

12. Screen shot of the 3 images loaded from FOL into the wwwroot\img folder of the application

13. The entire StoreController code. The Index method simply returns the default view so the user can enter an address to search from. The GetStores method receives the lat and lng values from the client's jsx code. Then sends the values to the model and returns the list in json format back to the client's jsx code

14. The entire Views\Store\Index.cshtml view. The markup includes a store.jsx file which uses ReactJS to render the view. We also added a link to google geocoder service which the react code will employ.

15. The react code. This code will get the entered address from the view using an intrinsic react attribute called **ref** and passes it to **google's Geocoder service** as the first parameter (**this.refs.address.value.trim**). **An inline method** that processes the returning data is the second parameter of this method call and will execute when google returns data (if the status was Ok). This data includes the **latitude** and **longitude** of the entered address. The method code then directly calls the server side controller code and passes the latitude and longitude as parameters to the controller method.

    The result of this is that the controller method sends a List of Store instances in JSON format back to the script: in the variable called **data.** The react code then calls **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 routine will look for an element called **map_canvas** to place the google map. Note, this routine is configurable by changing the **options** variable.

    Also, notice how everything is in a for loop, and how the individual marker contents are loaded using the data passed back from the server. Lastly, clicking any of the markers, should present the additional bubble popup containing the individual branch's information:

16. Execution of the View with the college's postal code entered as the source for the data

17. The resulting modal showing 3 markers, with one of the markers clicked to show the store details.

18. Fix (provided by Stackoverflow) if info window does not appear

Lab10 (2%)

Submit a single screen shot of the modal showing a map with the 3 closest locations to **your home postal code** make sure you click on marker 1 to show the bubble popup contents in the screen shot.

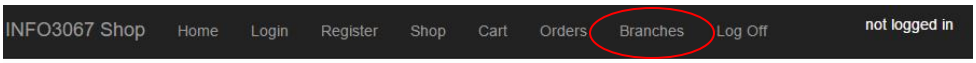
Case Study Notes

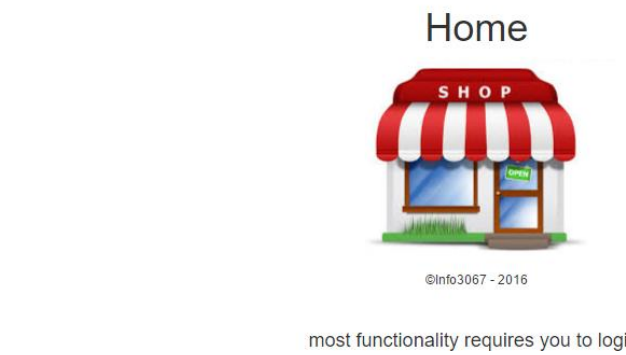
1. There are a couple of files on FOL to help you do this kind of processing in the case study. **BranchCS.txt** is the code for an entity called Branches (equivalent to the Store entity in the exercises project). Note there is no namespace in this code, you'll have to add it. The other file BranchesSQL.txt is an SQL script that will populate the Branches table (because there is no data controller in the case study). The Branches SQL is country wide not just for Ontario – fyi.

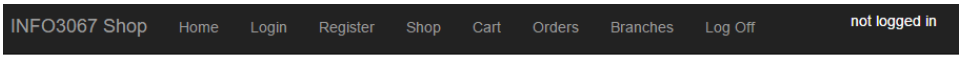
2. You'll need a BranchController (similar to the StoreController), a BranchModel (similar to the StoreModel) a View, and branch.jsx file all similar to the exercises

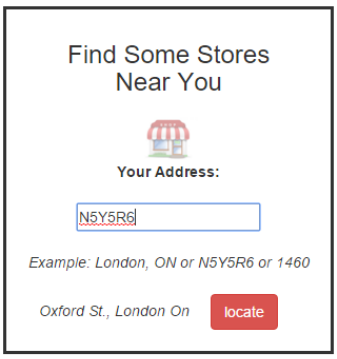
,.

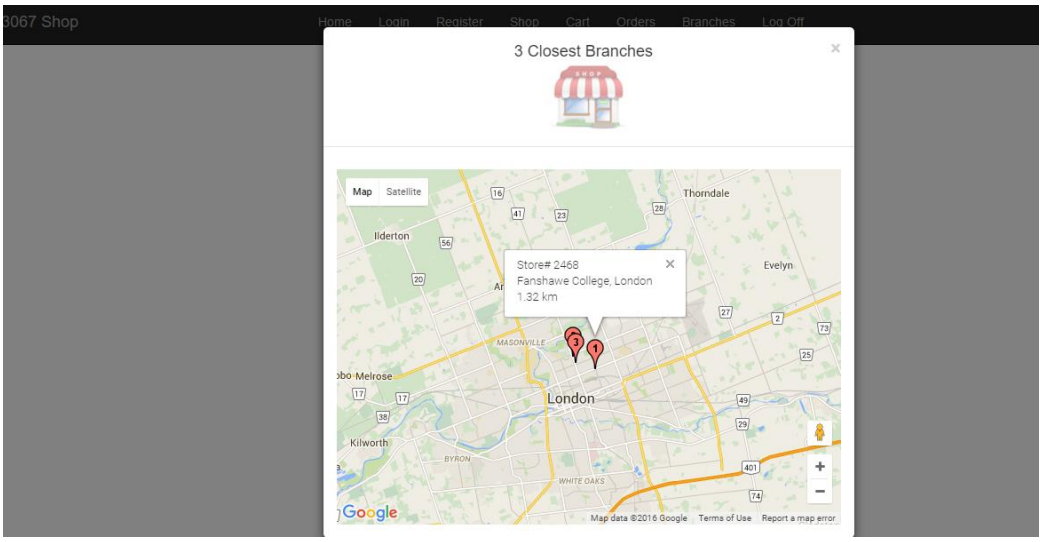
3. Polish up the styling for the view and modal so that it reflects your application:











Notice how the bubble help is **in Kilometers** not Miles, you'll need to change the given stored procedure to kilometers from miles as this is a requirement for the case study