Introduction

OpenCelliD – BigData Presentation

Student: Mobin Idrees
Supervisor: Dr. Frederic Stahl

OpenCelliD is BigData collection of Cell Towers and WiFi Aps.

This database is built by the community and is free available for download.

The row count is 39,374,666, *Last updated on : 2018-02-14*

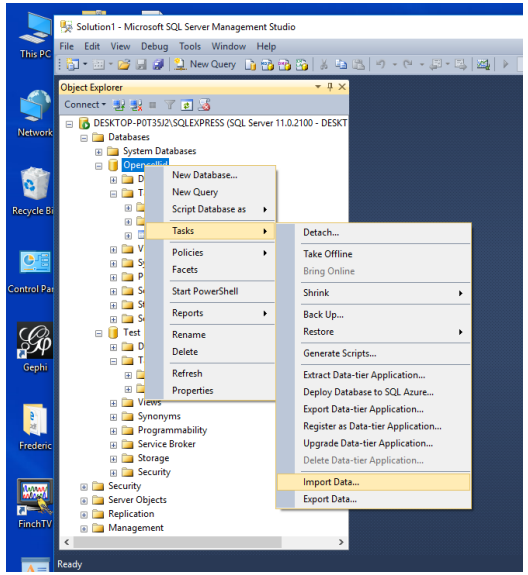
Database link in CSV format
https://opencellid.org/downloads

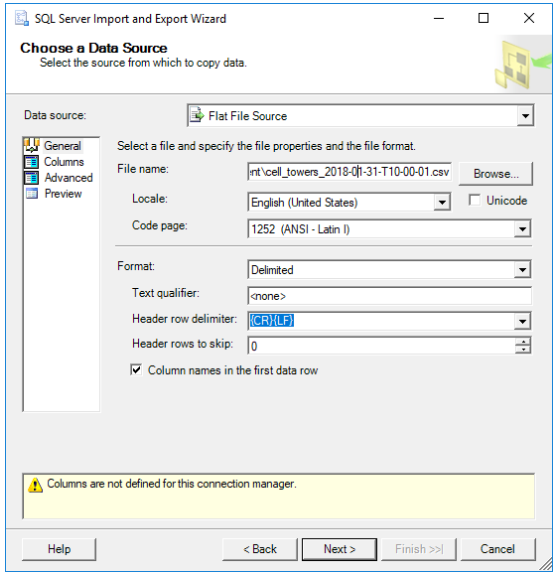
Database

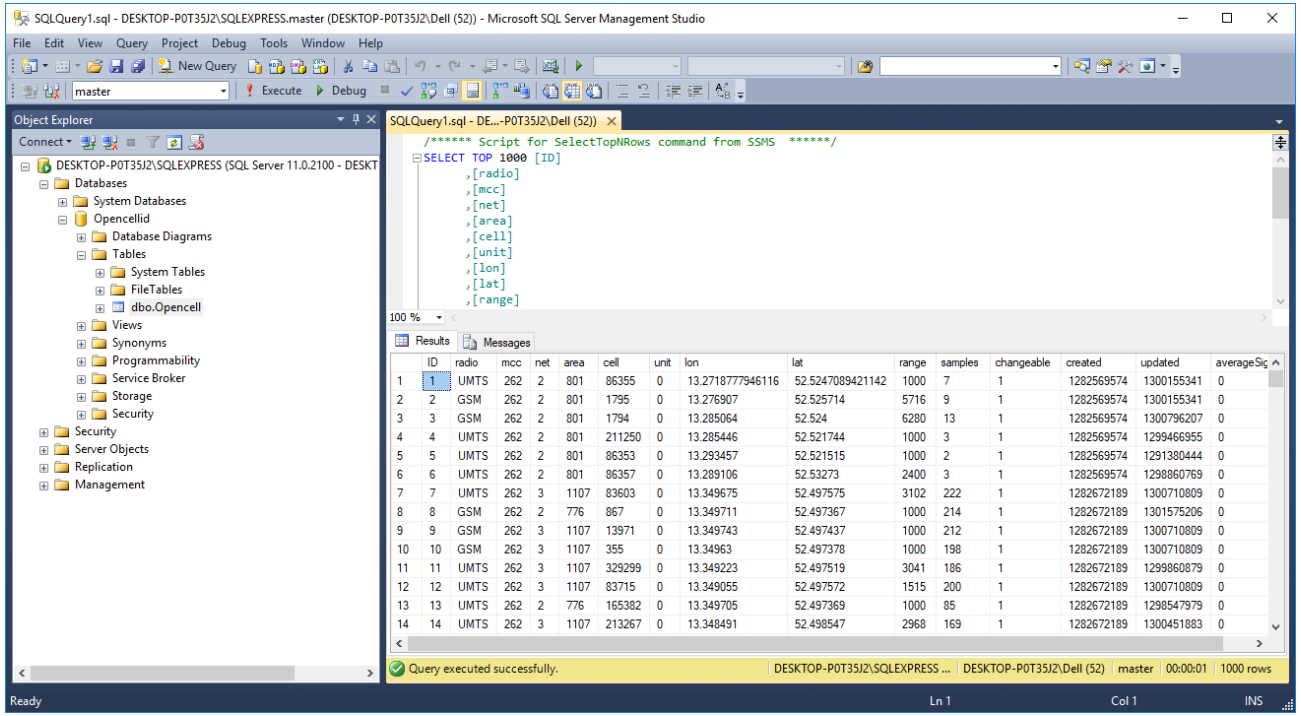
Data Base Structure

Field	Description (based on official documentation where available)
ID	This field is auto Incremented.
Radio	UMTS/ GSM /LTE
Area	Location Area Code
Cell	Cell ID represents the ID of Cell for which you want to display the data.
Range	Estimated Range of the cell in meters
Samples	Total number of readings taken from that cell tower
Latitude	GPS latitude of the measurement
Longitude	GPS longitude of the measurement
Mcc	Mobile Country Code
Mnc	Mobile Network Code
Created	Date the Cell tower was added
Updated	Date the Measurement is Updated

Import CSV in SQL Server







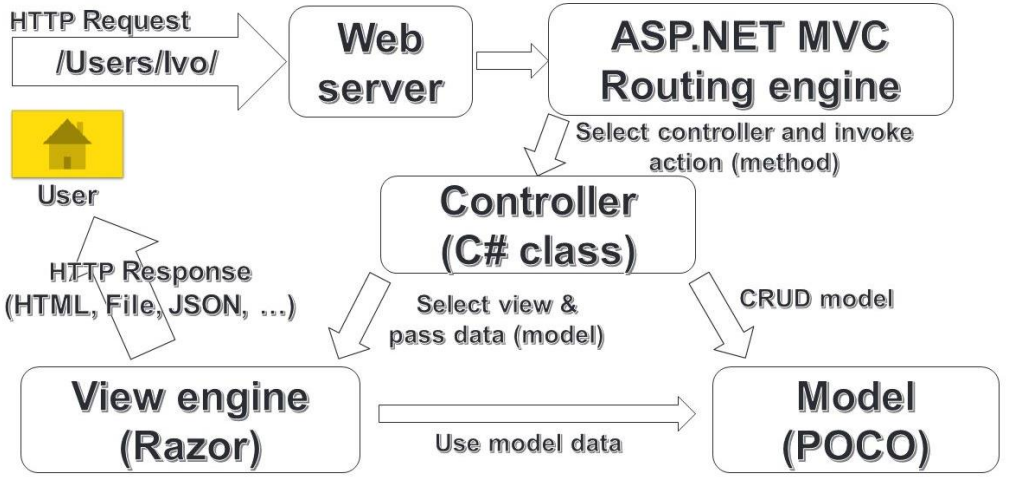
Import **Configuration** **Query**

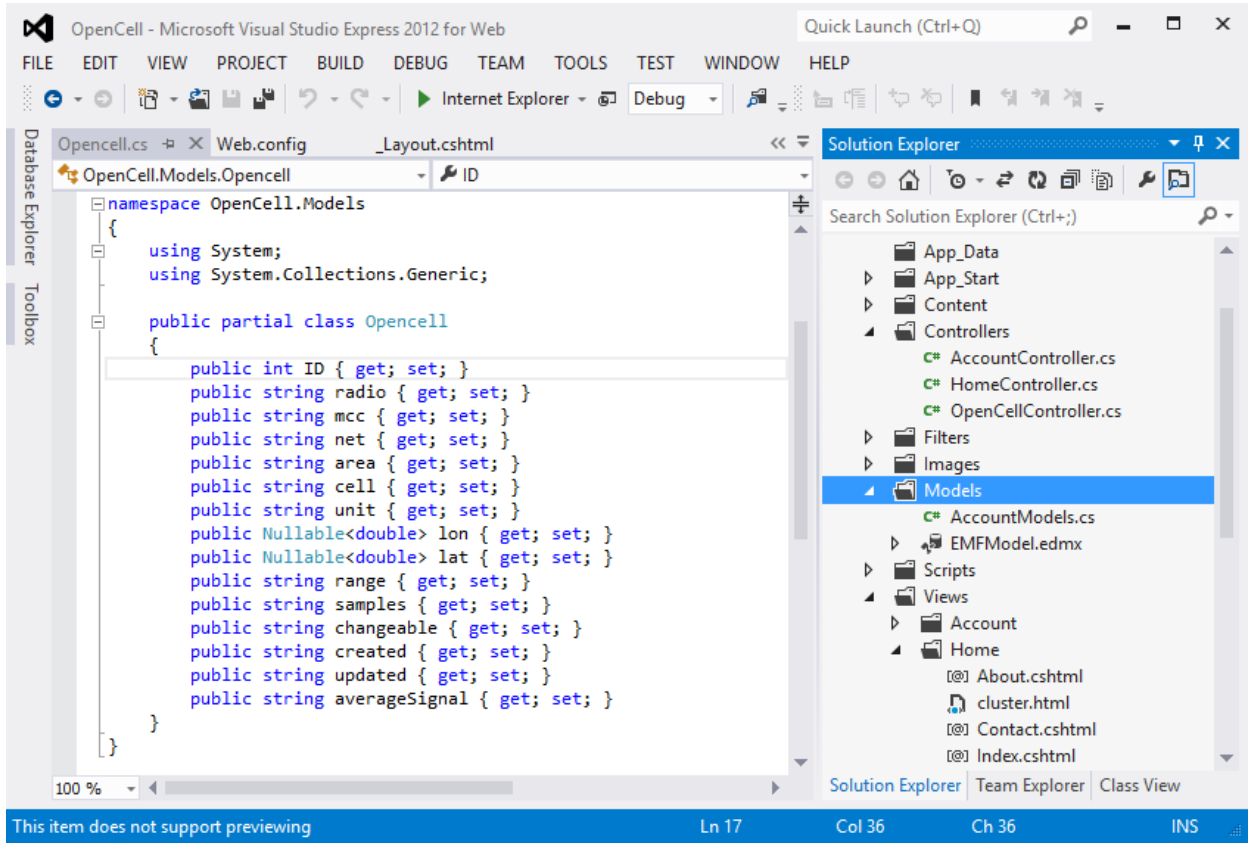
Application

Source Code available at : https://github.com/mimm1/OpenCell

- ✓ The Application provides a simple interface for requesting geo-registered map images from one or more distributed geospatial databases.
- ✓ The interface is developed in Visual Studio and Model-View-Controller is used to create interfaces. The map is integrated in the application using Asynchronous JavaScript and XML (AJAX) and the JavaScript Object Notation (JSON) format has been used to serialize and transmit the structured data.
- ✓ A utility library has been created to interact with Bing Maps to display a map with location markers. The feature is added to determine the geographic location of the users from their IP addresses.
- ✓ The geospatial features of SQL Server are implemented to perform a location-based radius search. The overlay objects are tied to the latitude and longitude coordinates to display markers and relevant information on the map.

Model View Controller (MVC)





OpenCell Class

Controllers

Model

View

- ✓ The OpenCell view in the Razor view engine and the strongly-typed view is selected for the model class. The scaffold template is used for creating list, details, edit, and delete views based on the master layout.
- ✓ The related information is stored in the database and passed to the map and appears on the info window on the Mouse Click event on the Markers. The geographic locations of OpenCell towers are stored in the SQL table with latitude and longitude values received from the maps.
- ✓ Partial views are used to create, edit, or view the geographic locations of the Cell towers. The mapping functionality is utilized in several places within the application; therefore a shared Map functionality has been utilized within a single partial template to reuse in multiple controllers and views. The callback event handler function is used; this adds a pin to the map for location identification. Draggable pushpins are used on the map for location identification of Cell towers.

Bing MAP V8 API

Shared -> _Layout.cshtml

```
<script type='text/javascript' src='https://www.bing.com/api/maps/mapcontrol?branch=experimental&key=KeyHere&
callback=onMapLoaded' async defer></script>
```

Callback function

Bing Maps V8 API

Required for clustering

View -> _Index.cshtml

Loop to generate dynamic pins from the database

Call Clustering Function

```
 function onMapLoaded() {

                var map = new Microsoft.Maps.Map(document.getElementById('theMap'), {
                    /* No need to set credentials if already passed in URL */
                    center: new Microsoft.Maps.Location(39.393486, -98.100769),
                    zoom: 3 });
                 Microsoft.Maps.loadModule('Microsoft.Maps.Clustering', function () {
                    // Creating sample Pushpin data within map view
                    var pins = [];

                     @foreach (var item in Model) {
        <text>
                    var pin = new Microsoft.Maps.Pushpin(new Microsoft.Maps.Location(@item.lat, @item.lon));
                    pin.metedata = item;
                    pins.push(pin);
        </text>
{
                    //var pushpins = Microsoft.Maps.TestDataGenerator.getPushpins(1000, map.getBounds());
                    var clusterLayer = new Microsoft.Maps.ClusterLayer(pins, { gridSize: 100 });
                    map.layers.insert(clusterLayer);
;({

{
```

- ✓ The application is created in ASP.NET C# using Visual Studio and SQL Server. The Model-View-Controller Version 4 is used to enable desktop and mobile views.
- ✓ The Razor is used as a view engine because its syntax is compact and reduces typing.
- ✓ The controllers are created employing a scaffolding template "read, write, actions, and views" using Razor view and the OpenCell. The controller is responsible for controlling how a user interacts with an MVC application.
- ✓ Entity Framework (EF) approach applied as the database was already created using OpenCell CSV file, when the database is already created, the EF automatically generates a data model that consists of classes and properties

OpenCell.js

MAP Function Scripts

```
OpenCell.LoadMap = function (latitude, longitude, onMapLoaded) {

    OpenCell._map = new Microsoft.Maps.Map(document.getElementById('theMap'), {});
    // Create draggable Pin in the center
    var center = OpenCell._map.getCenter();
    var Location = Microsoft.Maps.Location;
    var Pushpin = Microsoft.Maps.Pushpin;

    if (latitude != null && longitude != null) {
        this._map.setView({
            center: new Microsoft.Maps.Location(latitude, longitude),
            zoom: 15
;({
        OpenCell._pins = new Pushpin(new Location(latitude, longitude), { color:
'#00f', draggable: true });
{
    else {
        OpenCell._pins = new Pushpin(new Location(center.latitude,
center.longitude), { color: '#00f', draggable: true });

{
    OpenCell._map.entities.push(OpenCell._pins);
    // Binding the events
    OpenCell.EnableCallback();
{
```

```
OpenCell.LoadPins = function (latitude, longitude) {

    OpenCell._map = new Microsoft.Maps.Map(document.getElementById('theMap'), {});
    // Create draggable Pin in the center
    var center = OpenCell._map.getCenter();
    var Location = Microsoft.Maps.Location;
    var Pushpin = Microsoft.Maps.Pushpin;

    if (latitude != null && longitude != null) {
        this._map.setView({
            center: new Microsoft.Maps.Location(latitude, longitude),
            zoom: 15
;({
OpenCell._pins = new Pushpin(new Location(latitude, longitude), { color: '#00f',
draggable: false });
{
    else {
OpenCell._pins = new Pushpin(new Location(center.latitude, center.longitude), {
color: '#00f', draggable: false });

{
    OpenCell._map.entities.push(OpenCell._pins);
    // Binding the events
    //OpenCell.EnableCallback();
{
```

Events Handlers

```
OpenCell.EnableCallback = function () {

    var Events = Microsoft.Maps.Events;
    Events.addHandler(OpenCell._pins, 'drag', function (e) { onMouseUp(e); });
{
```

```
function onMouseUp(e) {
    var loc = e.location;
    $("#lat").val(loc.latitude.toString());
    $("#lon").val(loc.longitude.toString());
{
```

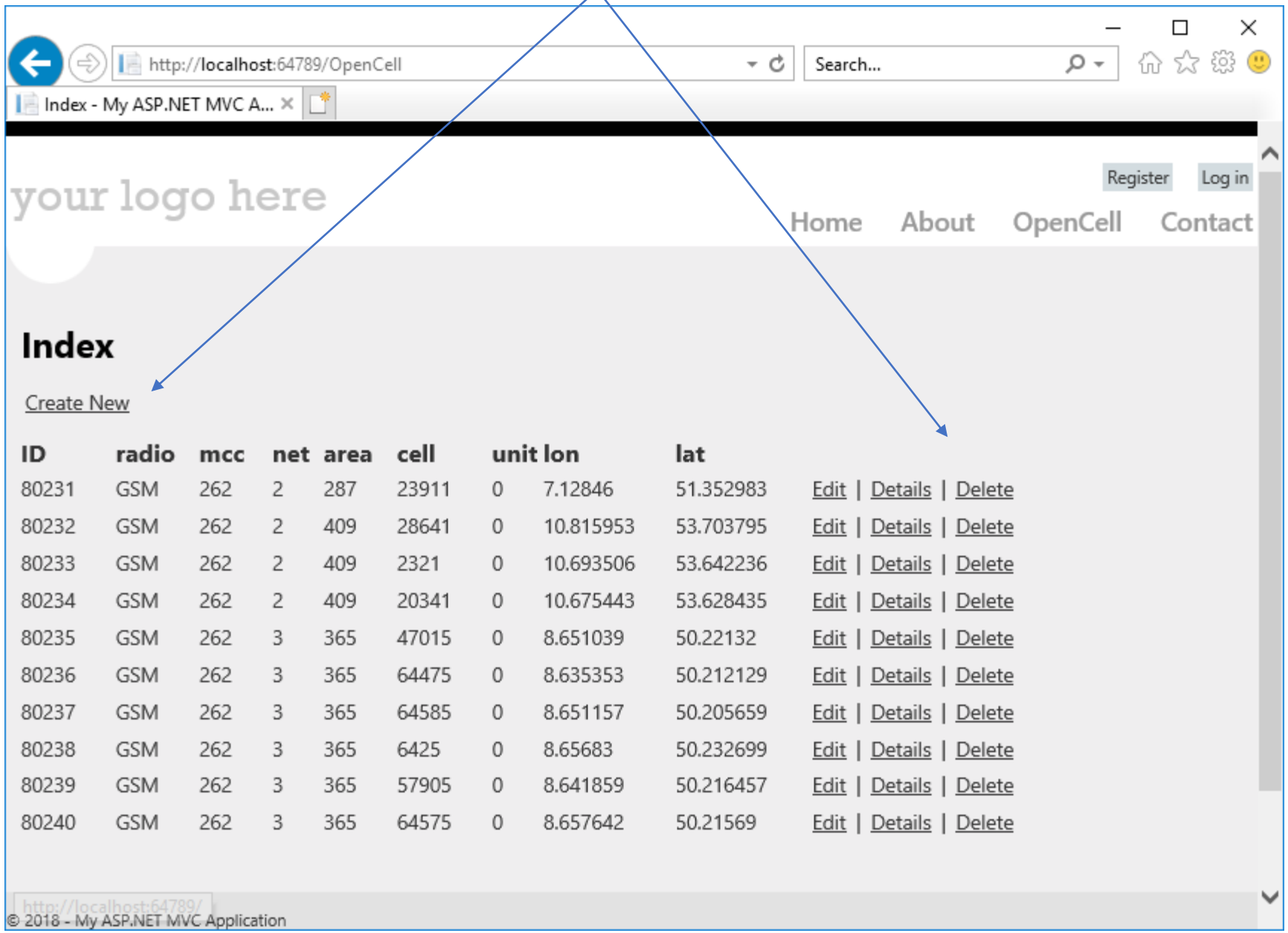
Runtime

CRUD Functionality

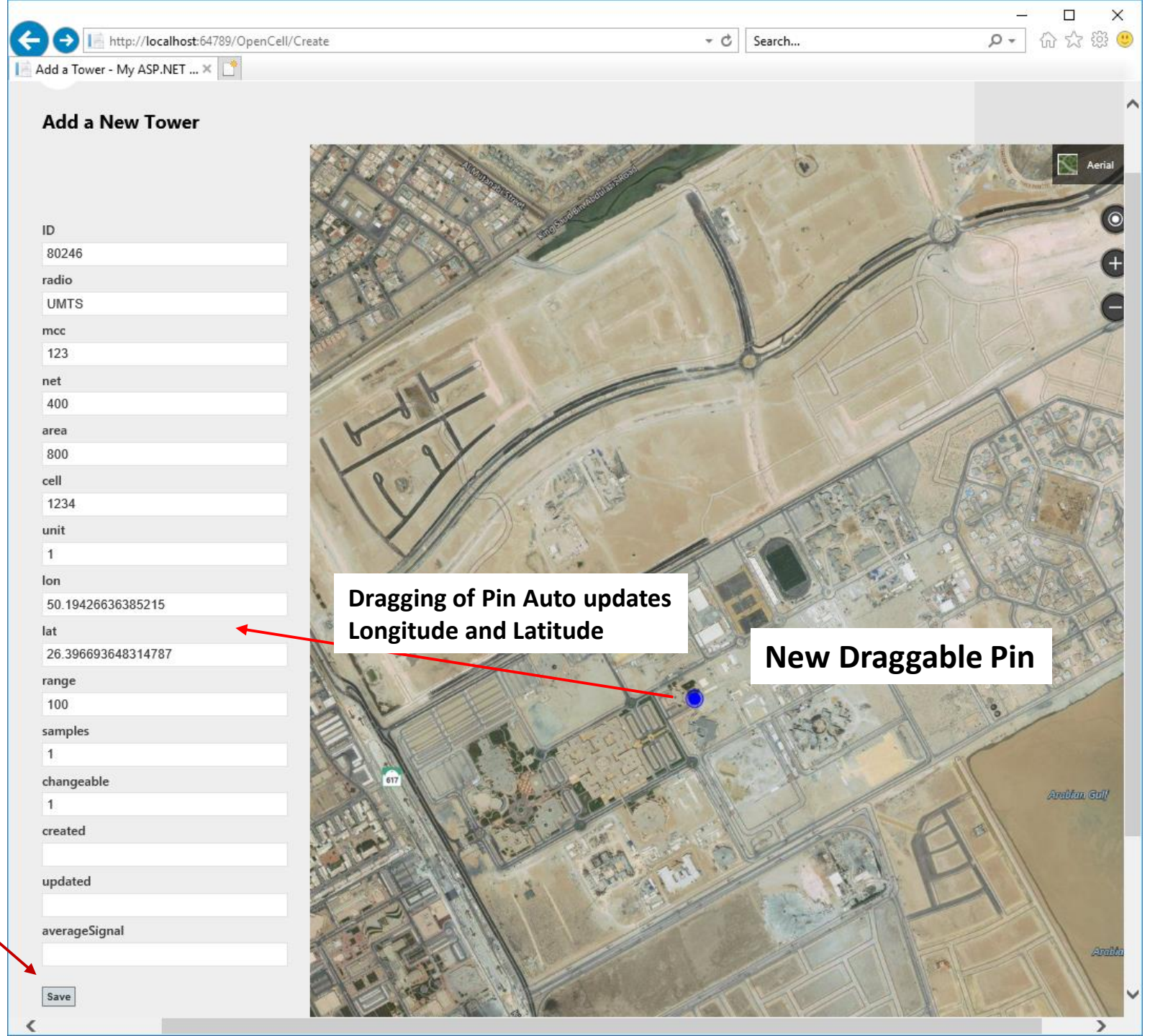
List top 10 rows

OpenCell.Controllers

```
  public ActionResult Index()
}
  return View(db.Opencells.Where(x=>x.ID<=10).ToList());
{
```

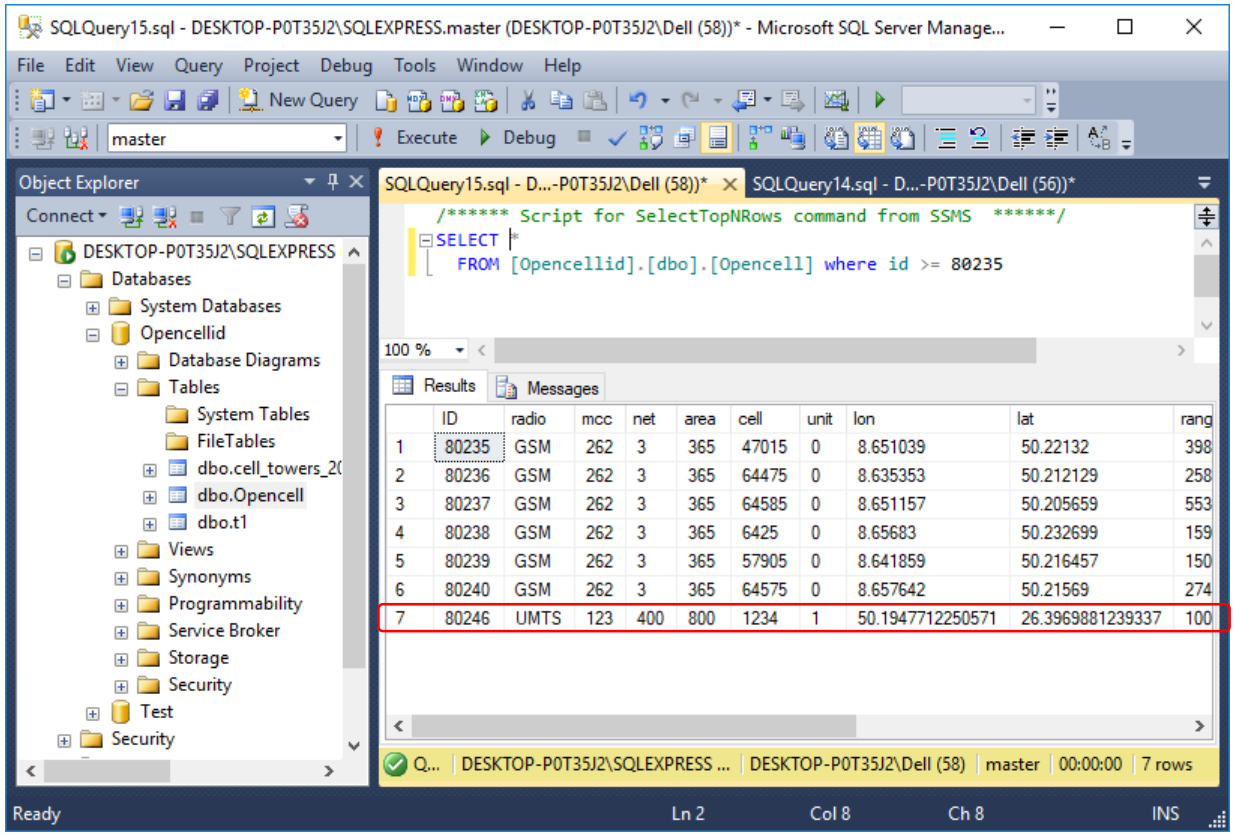


Create New

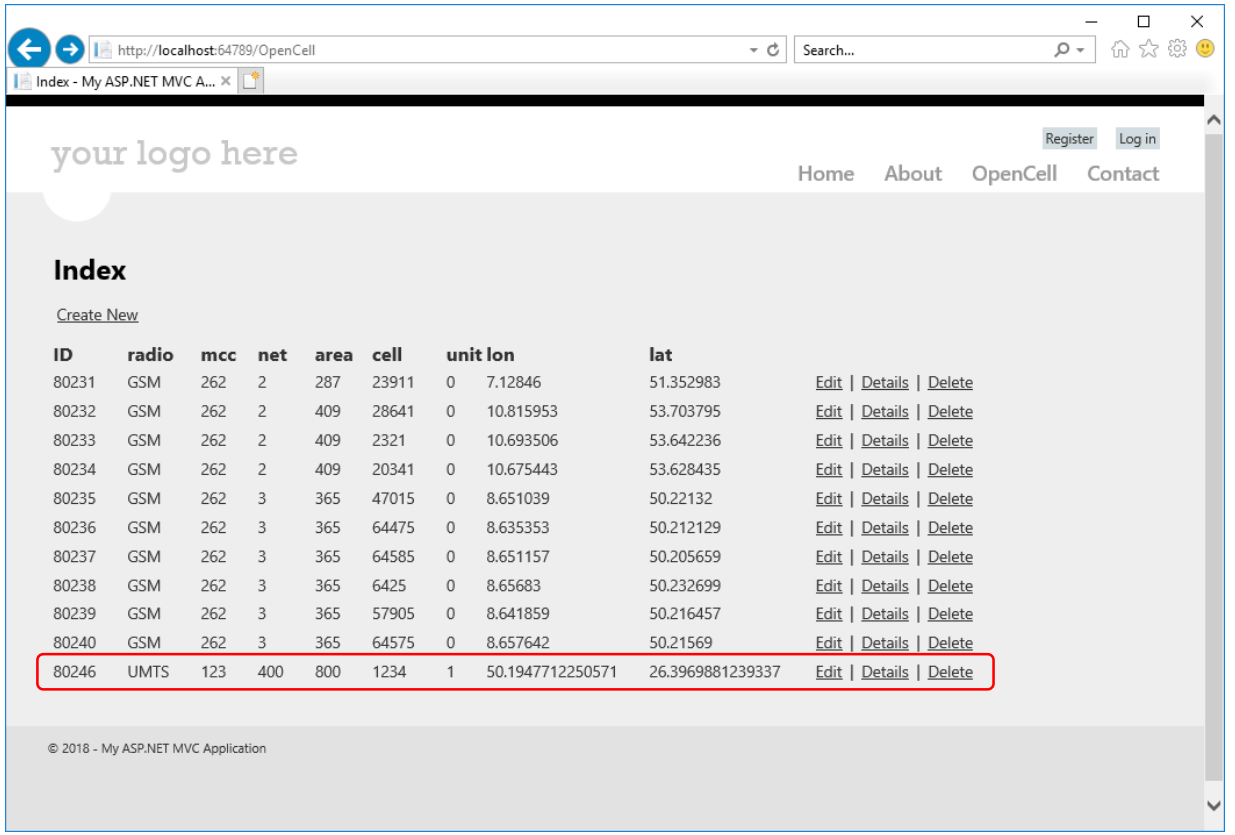


Save New Tower ID “80246” In Database

Locate New ID



New ID “80246”
In the Database

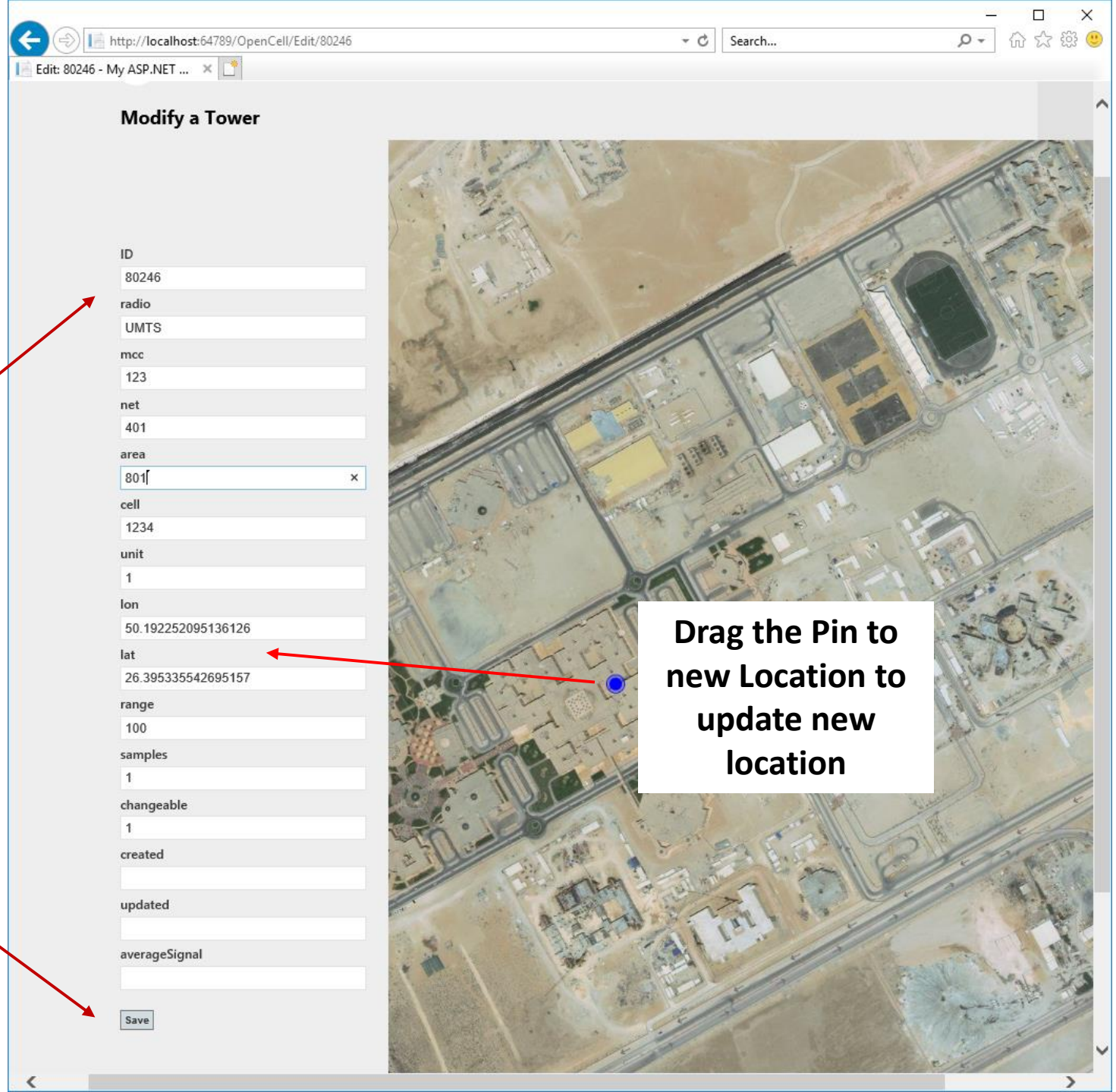


New ID “80246”
In the View

Modify the Location

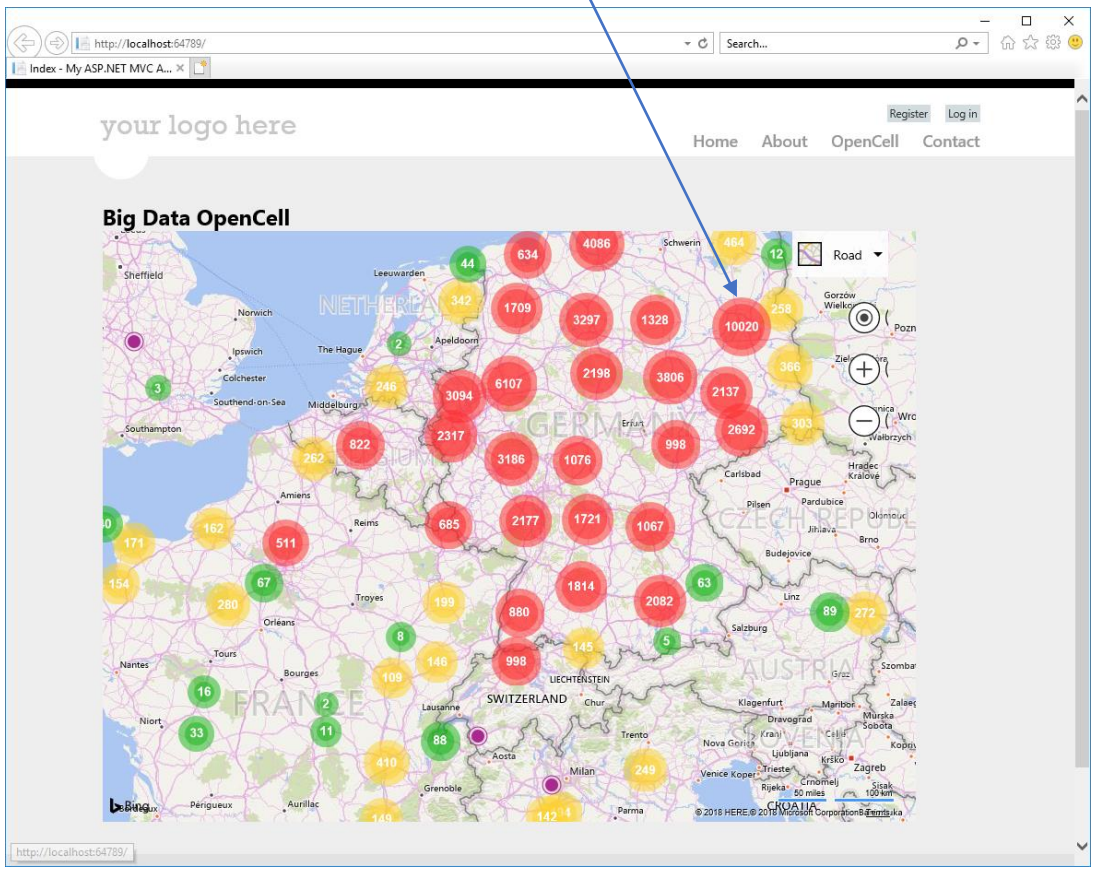
Load Tower ID “80246” From the Database

Update Tower ID “80246” In the Database

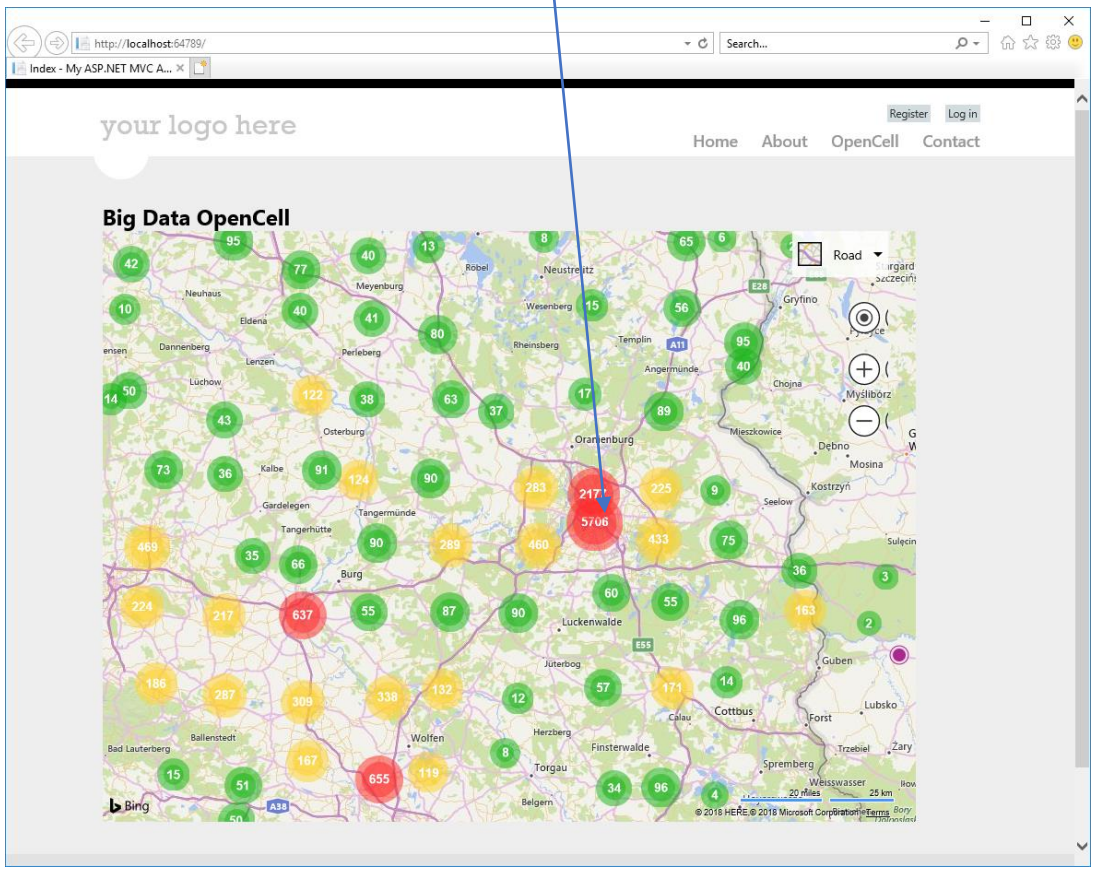


Germany : Largest cluster size “10020”

‘Oranienburg’ region Largest cluster size “5706”

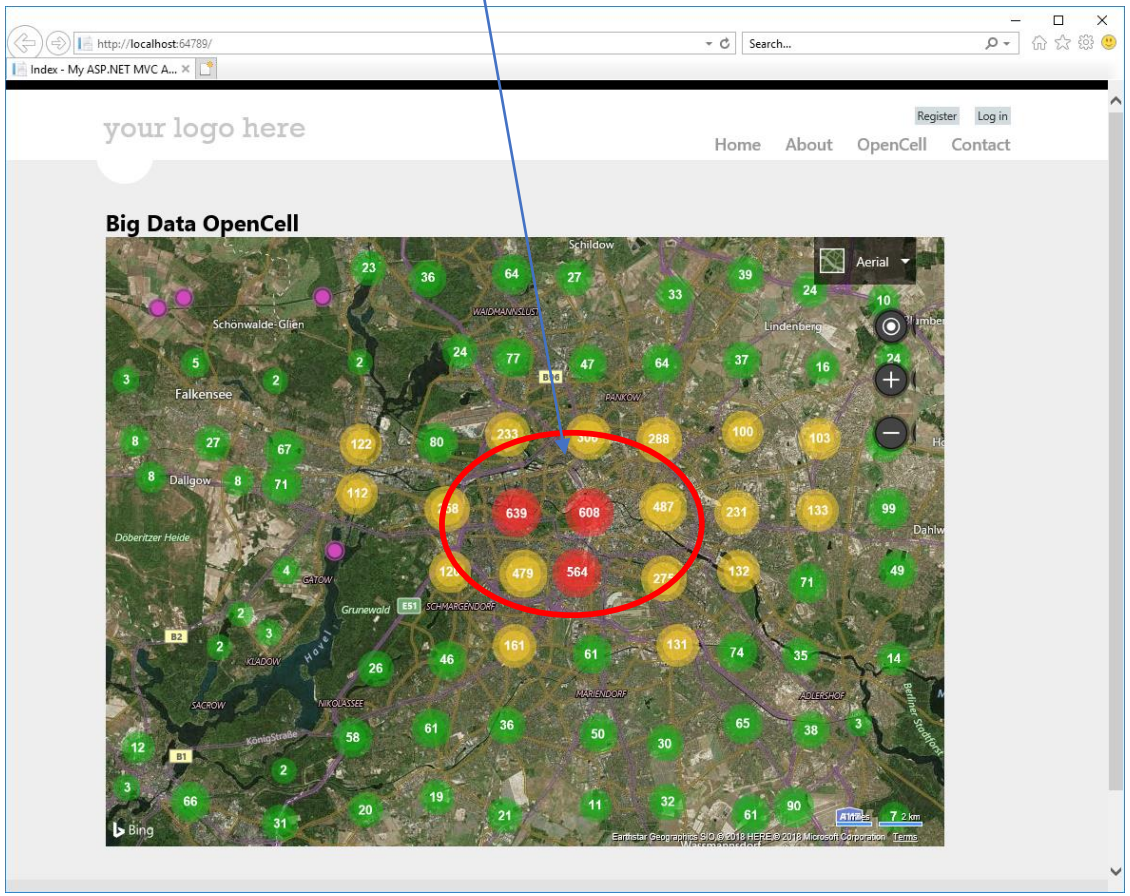


Germany Zoom Level : 4



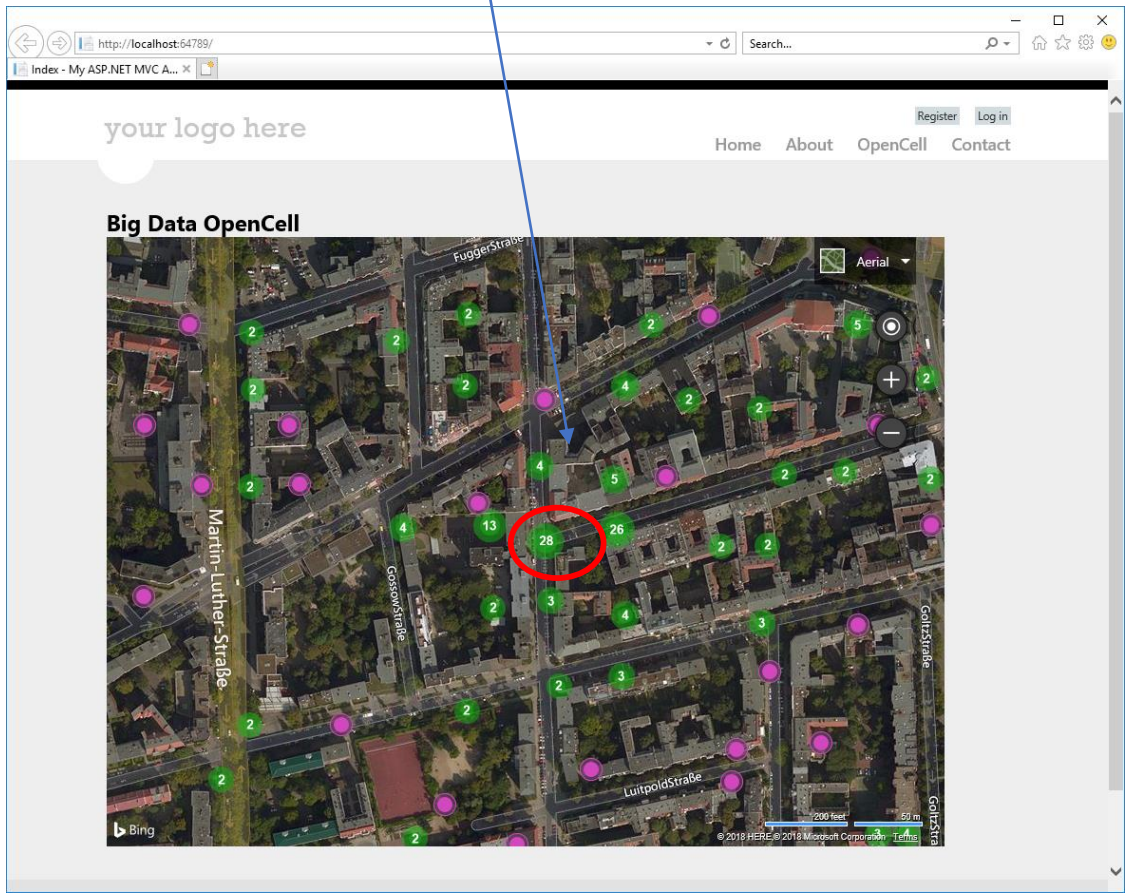
Germany Zoom Level : 6

Dense Cluster in Wilmersdorf Area



City View

Dense Cluster in Eisenacher StraBe “28” Measurements



Street View

Thanks