Getting Started with Open Data with ASP.NET MVC

An eBook by: Brian McKeiver, BizStream

Twitter: @mcbeev
Email: bmckeiver@bizstream.com
Blog: http://www.mcbeev.com
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Author—Brian McKeiver</td>
<td>3</td>
</tr>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Intended Audience</td>
<td>4</td>
</tr>
<tr>
<td>Setup / Required Tools</td>
<td>4</td>
</tr>
<tr>
<td>What is Open Data?</td>
<td>4</td>
</tr>
<tr>
<td>Who is Socrata?</td>
<td>4</td>
</tr>
<tr>
<td>What is Socrata Open Data API™ (SODA)?</td>
<td>5</td>
</tr>
<tr>
<td>SODA Developers Site</td>
<td>5</td>
</tr>
<tr>
<td>How to Choose Your Applications Open Data</td>
<td>5</td>
</tr>
<tr>
<td>Accessing Dataset via SODA API</td>
<td>6</td>
</tr>
<tr>
<td>Determining API Endpoint</td>
<td>6</td>
</tr>
<tr>
<td>Socrata Setup</td>
<td>6</td>
</tr>
<tr>
<td>Create Your Account</td>
<td>6</td>
</tr>
<tr>
<td>Register Application and App Token</td>
<td>8</td>
</tr>
<tr>
<td>The Sample Application</td>
<td>8</td>
</tr>
<tr>
<td>Create your Solution</td>
<td>9</td>
</tr>
<tr>
<td>Install SODA.NET</td>
<td>10</td>
</tr>
<tr>
<td>Getting to know SodaClient</td>
<td>10</td>
</tr>
<tr>
<td>Socrata Query Language (SoQL)</td>
<td>10</td>
</tr>
<tr>
<td>Retrieving Datasets from API</td>
<td>10</td>
</tr>
<tr>
<td>The Helper</td>
<td>11</td>
</tr>
<tr>
<td>The Model</td>
<td>13</td>
</tr>
<tr>
<td>The Result</td>
<td>13</td>
</tr>
<tr>
<td>Working with Datasets</td>
<td>13</td>
</tr>
<tr>
<td>Paging</td>
<td>13</td>
</tr>
<tr>
<td>Filtering</td>
<td>14</td>
</tr>
<tr>
<td>Sorting</td>
<td>16</td>
</tr>
<tr>
<td>The Controller</td>
<td>17</td>
</tr>
<tr>
<td>The View</td>
<td>18</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>20</td>
</tr>
<tr>
<td>API Limits</td>
<td>20</td>
</tr>
<tr>
<td>What's Next</td>
<td>21</td>
</tr>
<tr>
<td>Summary</td>
<td>21</td>
</tr>
<tr>
<td>About Socrata</td>
<td>21</td>
</tr>
</tbody>
</table>
About the Author – Brian McKeiver

As a senior solution architect and co-owner of BizStream, a web and custom software development company in Allendale, Michigan, I have been programming professionally for 14 years. I lead a team of software engineers who focus on Microsoft development platforms and technologies.

My passion for software development has not changed since day one of my first job. I enjoy helping customers across many different industries solve real-world business problems with technology. I have helped many small, medium, and large companies get past the pain of repetitive data entry, automate manual processes, integrate disparate systems, and make it so their websites and applications are lightning fast, easy to update, and a breeze to maintain.

Specialties include: ASP.NET 2.0, 3.5, 4.0, 4.5, Web Forms, MVC, C#, LINQ, Entity Framework, JavaScript, AJAX, jQuery, Kentico CMS, and SharePoint.

When not working, which is not often, I enjoy hanging out with my wife, chasing around my three children, and vigorously rooting for the Michigan State Spartans and Detroit Lions.

Introduction

This series of walkthroughs and tutorials is designed to illustrate creating a web application that works with government-based open data that is most easily accessed via tools powered by Socrata, a company specializing in open data. The core of this exercise is to walk through what it takes to setup a sample application, to connect to a remote dataset, and finally to consume that dataset inside of a web application. Tools such as Visual Studio, ASP.NET MVC, and C# will be used to create the application.

At the completion of the eBook you will be able to programmatically access government-based datasets easily inside of a Microsoft technology platform-based application. Doing so will open up a wealth of open data resources from governments, non-profits, and NGOs around the world.

When I was first asked to write this eBook, I was truly honored and humbled to do so. I will do my best to give you all of the knowledge needed to understand the concepts of creating a Microsoft ASP.NET application using the Socrata Open Data API.

The full source code of this eBook can be found at: https://github.com/mcbeev/SocrataSodaNetSample

A live demo of the sample application can be viewed at: http://sodanetsample.azurewebsites.net
Intended Audience

The intended audience of this eBook is experienced ASP.NET developers who are new to using an open data dataset and API. A developer interested in this tutorial should have the following skills:

- Familiar with an object oriented programming (OOP) language
- Familiar with ASP.NET web forms or MVC development methodologies
- Familiar with web development concepts (HTML, CSS, JavaScript)
- Familiar with JSON representation of data
- Familiar with REST API design pattern and related HTTP verbs
- Familiar with Microsoft Visual Studio IDE

Setup / Required Tools

Completing this exercise requires the following toolsets installed on a local workstation with internet connectivity:

- Microsoft Visual Studio 2013 Express or higher edition
- ASP.NET MVC 5
- ASP.NET Framework 4.5 / C#
- NuGet Package Manager

If your setup does not contain any of the above requirements please visit http://www.visualstudio.com/ and http://www.asp.net/mvc to get started.

What is Open Data?

Wikipedia defines open data as the idea that certain data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents, or other mechanisms of control. The goals of the open data movement are similar to those of other “open” movements, such as open source, open hardware, open content, and open access.

The philosophy behind open data has been long established, but the term “open data” itself is recent, gaining popularity with the rise of the internet and World Wide Web and, especially, with the launch of open data government initiatives such as Data.gov and Data.gov.uk.

To me the term “open data” is new, but the concept is something that I have worked with for more than 10 years. I have worked with local area government agencies here in the state of Michigan since 2002, helping them track data in a niche sector.

My company designed, implemented, and currently runs an application that provides juvenile facility management for these court-based agencies. The application is used in a dozen counties in Michigan and elsewhere in the United States. It is a central database for the different cases, the activity that happens on a daily basis inside of a facility, and assists in reporting different sets of data to the state. In essence the application has provided data from the county level to the state so that the state can analyze how well the facilities run and the results of the juvenile programs. We would be unable to do that without the data that the application collects.

There are many other documented examples of how open data is changing the way we think about trends, analytics, and measuring results. It is just one Google search away.

Who is Socrata?

Socrata is a privately-held cloud software company headquartered in Seattle’s historic Pioneer Square, with offices in Washington, D.C. and London. The company was founded in 2007 with a goal of creating a cloud platform that enables public sector organizations to easily manage and share data. This platform makes the data accessible to anyone with a web connection, while also giving them the power to visualize and analyze the data with ease.

Socrata continues to build on its original concept, offering products that enable researchers, public policy experts, entrepreneurs, and busy citizens alike to better understand and make use of public sector services. It also continues to expand, developing and releasing new services that address key challenges faced by many organizations, such as performance measurement and financial accountability.

For more information you can visit their website at http://www.socrata.com.
What is Socrata Open Data API™ (SODA)?

SODA provides an open, standards-based, RESTful application programming interface to access government datasets. This interface can be called from many different client libraries on many different platforms including Google Android, Apple iOS, Java, PHP, and Microsoft .NET.

Because of the RESTful API design each open data API endpoint can be manipulated with HTTP methods of GET, POST, PUT, and DELETE. It is assumed for this eBook that you have knowledge and experience with using RESTful verbs.

An endpoint in a SODA API is simply a unique URL that represents an object or collection of objects. Every Socrata dataset, and even every data record, has its own endpoint. The endpoint is what you’ll point your HTTP client at to interact with data resources.

A SODA endpoint works with basic datatypes of String, Numeric, Boolean, Timestamp, and Location. Each endpoint also has the ability to filter, sort, group, and perform full text searching on its data. The endpoint itself is normally structured in the following format:


SODA Developers Site

The SODA Developers site is the primary resource for developers looking to learn about Socrata and SODA. More background and documentation is found at the SODA Developer Site [http://dev.socrata.com/](http://dev.socrata.com/).

Included in the developer site is also a good Getting Started guide from Socrata that new developers should start with, [http://dev.socrata.com/consumers/getting-started.html](http://dev.socrata.com/consumers/getting-started.html).

The developer site provides links to the different client libraries that are available to work with, documentation on how to query a SODA dataset, and where to go for help and support.

How to Choose Your Applications Open Data

There are many different locations that host open data. Socrata hosts more than 100 different data catalogs for governments, non-profits, and NGOs around the world, so finding an open data catalog to work with is easy:

- Check to see if your local government or state already has an open data site. Check your city or state website, or even Google “open data” and your government’s name. You’ll find something pretty quickly.
• Review the online listing of Socrata-powered open data sites. There's probably one near you.
• Check to see if there's a community group in your area with its own catalog hosted on http://communities.socrata.com.
• Check out Data.gov, a site for U.S. government data.

Once you're on your chosen open data site, scroll down to the data catalog and use the search box and browse filters to find datasets that interest you.

If your data site has custom API Foundry-managed APIs, you can use the “API” filter on the left-hand side to show only those custom APIs. But if your dataset doesn't have the red API icon, don't fret—every dataset is accessible via the SODA API.

For this eBook I chose to use the City of Chicago's data portal. Chicago's data portal, https://data.cityofchicago.org/, is powered by Socrata, which makes it easy to use, and I really enjoy deep dish pizza. What can I say?

There are many types of datasets in an open data portal. The most common ones are listed on the left hand side of any data portal. In my experience the Dataset type is the most commonly used to integrate applications with, followed closely by the Map type.

For the remaining purposes of this eBook I chose the business licenses dataset. This dataset lists the business licenses issued by the Department of Business Affairs and Consumer Protection in the City of Chicago.

This dataset gave me the idea for creating an application to help me find a list of CrossFit® gyms in Chicago. Yes, I know most developers are not physical fitness fans. But, most of the developers I work with love CrossFit.

Accessing Dataset via SODA API

The Socrata Open Data API (SODA) provides programmatic access to this dataset, including the ability to filter, query, and aggregate data. Each open data portal allows you to also create your own saved filter or query against a dataset, as long as you register an account and login.

Determining API Endpoint

I created a simple filter, using the data portal at the City of Chicago's site, of the business licenses dataset to look for any legal name or DBA of CrossFit or gym.

To do that, I navigated to the endpoint I wanted to work with in a browser. From there, I simply clicked the Filter button towards the top left of the web page. Doing so opened up the filter sidebar showing the existing filters that were already added to the business license dataset.
I then clicked the Add a New Filter Condition button to add a condition to filter on Legal Name that contains the value of “CrossFit”. Now that the filter is correctly setup to show me all active businesses in the city of Chicago that had the word CrossFit in them, I can save that view of the data to get a unique 4x4 resource and name the view accordingly.

With the newly created resource endpoint, I can easily find the API endpoint that represents it by clicking on the Export button and looking at the API Endpoint field that the data portal makes available for me.

Socrata Setup

One of the first things you will need to do is to create your account on the Socrata Open Data platform.

Create Your Account

To create your account, navigate to the login page at https://opendata.socrata.com/login. This page will allow you to register an account so that you can create a Socrata application and get the required app token.

From the login page click the Sign up now link towards the bottom of the form to create a Socrata account. Note you can also choose to use an existing social network account.
Fill in the required form fields and click Create My Account. Once your account is created you should land on the Profile page.

**Register Application and App Token**

From the Profile page you can register an Application by clicking the Manage link on the title bar of the Applications widget.

Once the Manage link is clicked, you are taken to the Your Applications page. From this page you can either create a new application or edit an existing one.

To create your first application click on Create New Application.

For scope purposes, let's focus on what it takes to read information from an open data API and present it on the web page. Inserting and updating data will not be discussed in the contents of this book, but know that it is possible to add and modify data via the API.
Create your Solution

Now that you have chosen your dataset and know a bit about accessing an endpoint, let’s work on creating a sample application to put it all together. To do this, open up Visual Studio 2013. Create a new project by following these steps.

1. Create a new C# Web project named SODANETSample.

3. Click OK to create the project.

After the project is created you should have a new fresh project that uses ASP.NET MVC 5. The Model-View-Controller architecture and convention is already setup for you. The project also contains just about everything you would need to create a modern web application, including references and assets to jQuery, Bootstrap, and Modernizr.

2. In the New ASP.NET Project dialog box select the MVC template.

At this point you can continue using the default project organization structure and theme for the application. Or you can modify the structure and theme to match your own coding standards and best practices.

I like to move the fonts, scripts, and other assets type folders in my MVC projects to the Content directory to clean it up a little. I also make an Helpers folder to hold utility classes and other miscellaneous class files. In the official demo of the sample project, I also choose to use a custom Bootstrap theme. You can easily find one at http://bootswatch.com.
Install SODA.NET

Kegan Maher from the City of Santa Monica created a client library for the Socrata Open Data API at https://github.com/CityofSantaMonica/SODA.NET. The rest of this eBook is centered on using this client library for ASP.NET because it does an excellent job of consuming an open data endpoint. Kudos to Kegan and the City of Santa Monica for releasing the full source.

The easiest way to acquire the library into your project is to use the NuGet package manager. Using NuGet you can easily install the dll and reference it for use in your project.

To install the library, open up Visual Studio to your solution. From the Tools menu, select NuGet Package Manager, then Package Manager Console.

In the Package Manager Console window, make sure the Package source is nuget.org and the default project is <Your Solution Name>, and then enter the following command:

```
Install-Package CSM.SodaDotNet
```

The package manager should download and install SODA.NET, plus it's dependencies to the excellent Newtonsoft.Json library.

If you do not have access to NuGet, you can download the project files and/or full source from GitHub and manually reference the code to your project as well.

Getting to Know SodaClient

Once you have the library installed you should now be able to reference the main SODA.NET classes. The most common namespace you will be using is SODA.

```csharp
using SODA;
var client = new SodaClient();
```

These classes contain the most important object you will be working with, the SodaClient. The SodaClient object does the work of making the requests out to the Socrata Open Data API and interacting with the data portals, or endpoints, you choose to use. It handles making the HTTP transactions and moving the payloads to and from your application.

Before we dig into some code samples, let me introduce to you the way you will interact with the data that is available at an open data API endpoint.

Socrata Query Language (SoQL)

The Socrata Query Language (SoQL) is a simple, SQL-like query language specifically designed for ease in working with data on the web. The language is both powerful and easy to learn, and everything works via GET parameters. If you are familiar with SQL syntax you should have no issue working with SoQL. Just remember it cannot, of course, do everything that SQL can do.

The detailed documentation on SoQL is found at: http://dev.socrata.com/docs/queries.html

The advantage of using the SODA.NET client library and SodaClient object is that much of the raw SoQL is abstracted out for you as the developer. You can write fluent C# to select the data, order it, and group it.

Retrieving Datasets from API

To retrieve data from a SODA API endpoint you will want to instantiate a SodaClient object while passing in a valid resource URL and identifier (4x4 endpoint). Let’s walk through how that is done. The full code to do this is found inside of SODAHelper.cs in the sample project that accompanies this e-book.
The Helper

The code snippet below shows the main concept of creating a client and returning data.

```csharp
using PagedList;
using SODA;
using System.Linq;
using WebApplication.Models;

namespace WebApplication.Helpers
{
    public static class SODAHelper
    {
        #region Constants
        /// <summary>
        /// App Token from Socrata registered app - get yours at https://opendata.socrata.com/login
        /// </summary>
        private const string _AppToken = "362kGXXXXXXXXXXXXXjkozax";

        /// <summary>
        /// HostName of "http://data.cityofchicago.org/resource/xj6s-q5eb.json";
        /// </summary>
        private const string _APIEndPointHost = "data.cityofchicago.org";

        /// <summary>
        /// Socrata 4x4 Identifier
        /// </summary>
        private const string _APIEndPoint4x4 = "xj6s-q5eb";
        #endregion

        #region Methods
        /// <summary>
        /// Gets sorted list of all Business Locations in the dataset by filter
        /// </summary>
        /// <param name="SearchQuery">Query to filter on</param>
        /// <param name="PageNumber">Current page number</param>
        /// <param name="PageSize">Number of items per page</param>
        /// <param name="OrderBy">Column name to sequence the list by</param>
        /// <param name="OrderByAscDesc">Sort direction</param>
        /// <returns>PagedList<object> PagedList</returns>
        public static PagedList<BusinessLocation> GetBusinessLocations(string SearchQuery, int PageNumber, int PageSize, string OrderBy, bool OrderByAscDesc)
        {
            //Create client to talk to OpenDat API Endpoint
            var client = new SodaClient(_APIEndPointHost, _AppToken);

            //get a reference to the resource itself the result (a Resource object) is a generic type
            //the type parameter represents the underlying rows of the resource
            var dataset = client.GetResource <PagedList<BusinessLocation>>(_APIEndPoint4x4);

            //Build the select list of columns for the SoQL call
            string[] columns = new[] { "legal_name", "doing_business_as_name", "date_issued", "city", "state", "zip_code", "latitude", "longitude" };

            CONTINUED ON FOLLOWING PAGE...
        }
    }
}
```
There are a few interesting notes in the code snippet. First, after you create a SodaClient, you must make a reference to the resource itself (your API endpoint). This reference is what sets up the REST call for you. This is done in the client. GetResource line. Once you have a reference, you can choose to work with it directly, which means all columns and all rows by calling dataset.GetRows(), or as the code snippet shows you can choose to reduce the amount of columns by passing in a SoQL query.

Here is fluent C# style syntax to construct a SoQL query that would look like the following:

```
$select=legal_name AS legalname, doing_business_as_name AS dba, date_issued AS issuedon, city, state, zip_code, latitude, longitude&$order=:id ASC
```

Again the power of the SODA.NET library is that query isn’t manually crafted. The .Select, .As(), and .Order() methods are dynamically building it for us.
The Model

In the section above you probably noticed that the dataset is being returned as a PagedList collection of the BusinessLocation type. That is the next thing to take note.

Below is the full code for the BusinessLocation type. Notice how the names of the values in the .Select() method from above line up with the names of the properties of the type, as well as, the values of the aliases in the .As() method.

Again the SODA.NET library handles serializing and deserializing the JSON that the API endpoint returns. You do not have to spend the time casting and building an enumeration of objects. It is mostly automatic, much like Entity Framework handles building POCO objects from SQL database tables.

```csharp
using System;

namespace WebApplication.Models
{
    public class BusinessLocation
    {
        public string ID { get; set; }
        public string LegalName { get; set; }
        public string DBA { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Zip { get; set; }
        public DateTime IssuedOn { get; set; }
        public double? Latitude { get; set; }
        public double? Longitude { get; set; }
    }
}
```

The Result

After the GetBusinessLocations method is done executing you end up with a result that looks like this:

Working with Datasets

Paging

Because open data datasets usually contain large amounts of rows, showing everything on one web page may be problematic. In the sample application for this eBook I implemented a simple PageList type built for ASP.NET MVC.

To install the PagedList.MVC NuGet library, open up Visual Studio to your solution. From the Tools menu, select NuGet Package Manager and then Package Manager Console.

In the Package Manager Console window, make sure the Package source is nuget.org and the Default project is <Your Solution Name>, then enter the following command:

```
Install-Package PagedList.MVC
```
Once you have installed the package you simply have to add a using statement to your C# class that returns a PagedList collection, add an @model declaration line to your view that you want to use the pager on, and link up the CSS file that makes the pager look nice.

```csharp
@model PagedList.IPagedList<WebApplication.Models.BusinessLocation>
@using PagedList.Mvc;
<link href="/Content/Styles/PagedList.css" rel="stylesheet" type="text/css" />  
```

When properly implemented, you should see the following:

Filtering

To filter a dataset you have two options. The first option is a FullTextSearch method the SodaClient object offers. This method takes your query and translates it into the SoQL syntax of:

`http://data.city.org/resource/xxxx-yyyyy.json?$q=SearchForTermGoesHere`

The full text search will perform a search through all columns in a dataset for the value specified

The second way to filter a dataset is to use the .Where() clause in your fluent C# SoQL call. For full information on the options available, check out the queries documentation at `http://dev.socrata.com/docs/queries.html`. 
The sample application for this eBook has the FullTextSearch example built into it.

There are two parts to making the filter work. First the View must have a simple HTML form as seen below:

```html
<div class="row">
  <div class="col-md-12">
    @using (Html.BeginForm("Index", "Home", FormMethod.Get))
    {
      
      Filter Name: @Html.TextBox("SearchQuery", ViewBag.CurrentFilter as string)
      <input type="submit" value="Search" />
    }
  </div>
</div>
```

Secondly, you must call the .FullTextSearch method in your GetBusinessLocations helper also seen below:

```csharp
//using SoQL and a fluent query building syntax
var soql = new SoqlQuery().Select(columns)
if(!string.IsNullOrEmpty(SearchQuery))
{
  soql = new SoqlQuery().FullTextSearch(SearchQuery);
}
var results = dataset.Query<BusinessLocation>(soql);
```

Author's note: At time of publishing this eBook, the City of Chicago's FullTextSearch method issues a "500 Internal Server Error" for any query that contains a $q= parameter. But other open data portals do correctly filter datasets, such as this one: http://data.cityofnewyork.us/resource/spgx-ssye.json?$q=Brian
Sorting

In the screenshot below, is the dataset ordered by the Legal Name column instead of the natural sort order. This is handled on the backend of the code through the .Order() clause of the SoQL query.

```csharp
//using SoQL and a fluent query building syntax
var soql = new SoqlQuery().Select(columns)
                .As(aliases)
                .Order(SoqlOrderDirection.ASC, new [] {
                    <OrderByColumnName>" });
```

![Screenshot of Chicago CrossFit Gyms](soda/networksample/azurewebsites.net/S/O/SOOrder_localhost.png)
The Controller

The controller in the sample application is a pretty simple one. The class handles setting up the configuration for paging, sorting, filtering, and calling the helper class of GetBusinessLocations. The only noteworthy part of this code is the fact that the SortOrder parameter handles which column the dataset is sorted by, as well as which sort direction is going to be used.

Since the main focus of this eBook is about using the SODA.NET API this code is a good example of how to simply call out to the SODA API and return a PagedList collection, the rest is pretty boilerplate code.

```csharp
using System.Web.Mvc;
using WebApplication.Helpers;
namespace WebApplication.Controllers
{
    public class HomeController : Controller
    {
        public ViewResult Index(string SortOrder, string CurrentFilter, string SearchQuery, int? CurrentPage)
        {
            //Handle Paging
            int pageSize = 20;
            int pageNumber = (CurrentPage ?? 1);

            //Handle Sorting by Column
            if (!string.IsNullOrEmpty(SortOrder))
            {
                SortOrder = "";
            }
            ViewBag.CurrentSort = SortOrder;
            ViewBag.NameSortParm = (SortOrder == "legal_name") ? "legal_name_desc" : "legal_name";
            ViewBag.DBASortParm = (SortOrder == "doing_business_as_name") ? "doing_business_as_name_desc" :
            "doing_business_as_name";
            ViewBag.ZipSortParm = (SortOrder == "zip_code") ? "zip_code_desc" : "zip_code";
            ViewBag.IssuedSortParm = (SortOrder == "date_issued") ? "date_issued_desc" : "date_issued";

            bool sortAsc = (!SortOrder.Contains("_desc")) ? true : false;

            //Handle Filtering
            if (!string.IsNullOrEmpty(SearchQuery))
            {
                CurrentPage = 1;
            }
            else
            {
                SearchQuery = CurrentFilter;
            }
            ViewBag.CurrentFilter = SearchQuery;

            if (!string.IsNullOrEmpty(SearchQuery))
            {
                SearchQuery = SearchQuery.ToUpper().Trim();
            }
        }
    }
}
```

CONTINUED ON FOLLOWING PAGE...
The main line, of course, is the `SODAHelper.GetBusinessLocations()` line of code. This is where the controller calls the SodaClient.

### The View

Once the data is returned to the View, there is little work left to display the table on the web page. Basically, the model's data is bound to a row in a HTML table via standard ASP.NET MVC practices. Note that the correct declarations were applied to make the PagedList collection work correctly at the top of the page then render the PagedListPager html element below the main table thanks to the PagedList.MVC library.

```csharp
@model PagedList.IPagedList<WebApplication.Models.BusinessLocation>
@using PagedList.Mvc;
<br />@{ ViewBag.Title = “Home Page”; }

<div class="row">

    <div class="col-md-12">
        <h2>Chicago CrossFit Gyms</h2>
        <hr class="compact" />
        <table class="table table-striped table-hover ">
            <thead>
                <tr>
                    <th>@Html.ActionLink("Legal Name", "Index", new { SortOrder = ViewBag.NameSortParm, CurrentFilter = ViewBag.CurrentFilter })</th>
                    ...
                </tr>
            </thead>
            <tbody>
                ...
            </tbody>
        </table>
    </div>
</div>
```

CONTINUED ON FOLLOWING PAGE...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>@Html.DisplayFor(modelItem =&gt; item.LegalName)</td>
<td>@Html.DisplayFor(modelItem =&gt; item.DBA)</td>
<td>@Html.DisplayFor(modelItem =&gt; item.City)</td>
<td>@Html.DisplayFor(modelItem =&gt; item.State)</td>
<td>@Html.DisplayFor(modelItem =&gt; item.Zip)</td>
<td>@String.Format(“{0:MM/dd/yyyy}”, item.IssuedOn)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@Html.DisplayFor(modelItem =&gt; item.Latitude != null) {</td>
<td></td>
<td></td>
<td>@String.Format(“{0:MM/dd/yyyy}”, item.IssuedOn)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;a href=&quot;https://google.com/maps/place/@item.Latitude,@item.Longitude&quot; target=&quot;_blank&quot;&gt; &lt;span class=&quot;glyphicon glyphicon-globe&quot;&gt;&lt;/span&gt; &lt;/a&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page @(Model.PageCount < Model.PageNumber ? 0 : Model.PageNumber) of @Model.PageCount

After correctly adding all of the parts of the MVC structure you should now have a working page that displays the data from the API endpoint.

I frequently used this tool to test out my queries to my API endpoint. A sample call is below:

Troubleshooting

As always, the Visual Studio debugger is your friend when it comes to issues with your ASP.NET code, but sometimes it is nice to have other go-to resources. As mentioned, there is documentation on the Socrata Developers site. The SODA.NET GitHub repository has a list of issues that you can review. Contacting Socrata is also a possibility.

There is one more tool worth noting that I really found helpful when first learning SODA and SoQL, and that is [http://hurl.it](http://hurl.it). Hurl.it is like a live version of the popular utility curl. The tool lets you execute HTTP requests and has a simple interface to add in parameters.

Clicking on Launch Request gave me the results (and errors) that my ASP.NET code was finding:

API Limits

Another note worth mentioning is Socrata and the Socrata Open Data portal does have a set of API limits. You can make a certain number of requests without an application token, but they come from a shared pool and you’re eventually going to be cut off. When you include an App Token that limit is raised.

*The API Limit is 1000 per hour when using an App Token*
What’s Next

This eBook has shown you how to use the Socrata Open Data API with ASP.NET MVC from a data consumption standpoint. The API does also allow for creating new datasets, adding items into datasets, and updating items inside of datasets. For more on how to accomplish these advanced tasks, please see the Socrata Developers site at: http://dev.socrata.com/publishers/getting-started.html

Summary

You learned a lot in this eBook about programming ASP.NET MVC and C# with consuming the Socrata Open Data API using a great open source library SODA.NET. Specifically covered:

- What is SODA
- How to access a SODA dataset
- How to setup a Socrata developer account
- How to install SODA.NET
- How to consume an API endpoint via SODA.NET in ASP.NET MVC
- Viewing, sorting, paging, and filtering a dataset

A big thank you goes out to the excellent team at Socrata for providing a strong platform to work on, and an equally big thanks goes out to the City of Santa Monica for providing the excellent SODA.NET C# client library.

About Socrata

Socrata is the world leader in cloud solutions for open data and data-driven governments. Its innovative customers include the cities of New York, Chicago, San Francisco, Los Angeles, Melbourne and Eindhoven; the states of New York, Illinois and Texas; US Health and Human Services; Centers for Medicare & Medicaid Services; the UN and the World Bank. Socrata's solutions—including the recently launched Open Data Network™ which unleashes the full potential of government data to help drive connected communities around the world—assist government leaders in improving transparency, modernizing citizen access to information and bringing data into every decision, all with unprecedented speed and cost savings. Delivered as turnkey services, Socrata's technologies unlock data trapped in enterprise silos, mobilize and transform it into useful information that everyone can easily access, visualize, share and reuse. To learn more about Socrata, visit www.socrata.com.