|  |
| --- |
| PGD Cloud – Enterprise Frameworks |
| Awesome Enterprise App |
| Project Report |
|  |
| **Jack Lysaght – x11104449 – Git: Spuriosity**  **Jer Hogan –**  **Michal –**  **Tony -** |
|  |

|  |
| --- |
|  |

Contents

[Background research 2](#_Toc331201165)

[Project Plan 4](#_Toc331201166)

[1. Overview 4](#_Toc331201167)

[2. Functionality 4](#_Toc331201168)

[3. UI Prototypes 5](#_Toc331201169)

[Requirements analysis 6](#_Toc331201170)

[Use Cases 7](#_Toc331201171)

[Architectural approach 11](#_Toc331201172)

[Data Model Examples 13](#_Toc331201173)

[Security for the Models 14](#_Toc331201174)

[Testing Approach 14](#_Toc331201175)

[Use of Client-Side Processing/AJAX 19](#_Toc331201176)

[Use of Web Services 20](#_Toc331201177)

[Use of an ORM tool 20](#_Toc331201178)

[References 21](#_Toc331201179)

[Appendix: New York Filming Locations XML Data 21](#_Toc331201180)

# 

# Background research

As far back as 1996 (Streeter, Kraut, Lucas & Caby 1996) Open Data was recognized as important for business. Open data allows sharing of digital data with customers, suppliers and others. Investment in this area gains a business competitive advantage. In 1996 the Internet was in its early days as an information provider and (Streeter et al. 1996) found it (along with Bitnet and Tymnet) as bitty with different user communities. While they were calling for something like the Internet as it is in 2012 they also prefigured the use of open data from companies and public institutions which is provided by only a few entities even now. They compared the US at the time (patchy coverage) vs. France at the time (widespread use of Minitel network). They analysed three statements, one of which was “Open networks improve business productivity, efficiency, and quality of service.”. They found that firms using open networks were more effective and richer and had better relationships with their customers. They found that increased infrastructure would help small companies by reducing money and knowledge disadvantages. The increased infrastructure from 1996-2012 has proved that in Spades but there is scope for greater improvement if extensive open data was provided by all public companies.

In (Auer, Bizer, Kobilarov, Lehmann, Cyganiak & Ives 2007) the DBpedia program was outlined, which read information from Wikipedia and make this available on the Internet. This also linked other datasets to Wikipedia, and made it all available via sophisticated queries. (Auer et al. 2007) pointed out that most people in the industry see that bringing all the world’s structured information and making it available to search queries is a very worthwhile pursuit. This would extend greatly the current scope for mash-ups. This was very bitty at the time and they expected the Semantic Web (the next generation of the Web) idea to provide a framework for providing and returning all the world’s data via structured queries. In providing a method for storing and allowing retrieval of Wikipedia they faced many problems:

* Wikipedia was huge (1.95 million English articles).
* Wikipedia supported over 250 languages.
* Wikipedia is constantly being revised and edited by multiple authors
* Wikipedia has inconsistent taxonomies (Section Headings etc.).

DBpedia datasets can be used by third-party programs or used online by some DBpedia UIs.

The internet was searched in order to find open data which could be used as input to provide a workable project using Enterprise Frameworks. The Fingal Open data site[[1]](#footnote-1)is one which is close to home. This has over 170 data sets in CSV and XML. These include;

* A list of Art Centres
* Weather Statistics
* Cinemas
* Libraries (including Mobile)
* Tourist Information Locations
* Playing pitches
* Airports
* Train Stations

The New York city open data site[[2]](#footnote-2) is another source of data sets which is extensive. It claims over 1000 data sites. Included are:

* Film Shooting Locations
* Restaurant Locations
* Park Locations
* Public Transport (Subway, Buses, Rail)
* Museums and Galleries
* Green Market Locations
* Landmarks
* Map of Monuments
* Library Branches

Our team considered these open data sources and we thought about a mash-up between two datasets such as Fingal Cinemas and their nearby Train Stations displayed on a Map. The Cinemas would be on a drop-down menu and a map would display the cinema and nearest Train stations. We also considered a Restaurant Rating system. In this, Restaurant owners would register and provide information about their set menus and allow diners to view these along with location information. In the end we went for a mash-up between New York Film Shooting Locations and displaying the nearest restaurants within a user-defined radius. This needed a database of shooting locations and restaurants, an XML reader of shooting locations, a view allowing film and location to be chosen and restaurant information to be entered and displayed, business logic to discover the nearby restaurants given a latitude and longitude and an output API allowing further use of our data.

# Project Plan

## Overview

The AwesomeEnterpriseApp project is an application for tourists in New York City. It is specifically aimed at tourists interested in film history. It is designed to help the tourist to find a restaurant near the filming location of their chosen film. The user will be able to select a film location and will be presented with a list of restaurants that fall within a user-specified radius of the location. These restaurants will be presented with location, contact details and business hours.

The restaurants that are presented to the user will be entities that are registered with the application. The restaurant owner is required to make an account with which they can create or edit a restaurant entity. This creates business potential for both the restaurant and the application. The restaurants are part of a select list associated with the film locations, and for the application they generate advertising revenue.

The application is designed with New York City film locations in mind, but will be expandable to other cities upon presentation of an API meeting the criteria of the New York Open City API.

## Functionality

The AwesomeEnterpriseApp project shall perform the following functionality:

* For the restaurant owner:
  + Present the user with a joint login/registration form (see fig.1)
  + This will create a user account within the application
  + Name, email and password will be entered into the database
  + The password will be encrypted
  + Upon registration the user is presented with a restaurant details form (see fig.2)
  + This collects details such as name, cuisine type, contact details and address
  + This information is sent to the application where the map coordinates of the address are determined using the Google Maps API
  + These details along with the extracted map coordinates are saved to the database
  + Upon login the user is presented with the same form populated with the details of their registered restaurant
* For the end-user:
  + Present the user with a form with a drop-down menu populated with a list of film titles (see fig.3)
  + These titles are aggregated from the New York Open City API.
  + Upon selection of a film title, the second drop-down menu is populated with a list of locations in New York where filming has taken place for the selected choice, coming from the same API
  + The user also selects a range (radius) within which to search
  + This information is sent to the application
  + The application pulls the restaurant coordinates from the database and an algorithm is used to determine a list of entries whose coordinates lie within the selected radius of the selected film location coordinates
  + The details of these restaurants are then pulled from the database and returned to the user form presenting a list of details of restaurants within the selected parameters

## UI Prototypes

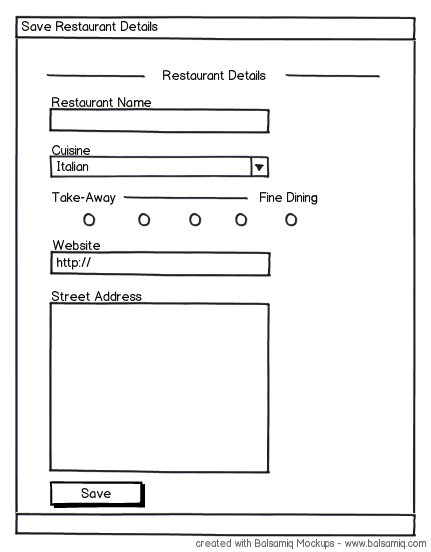
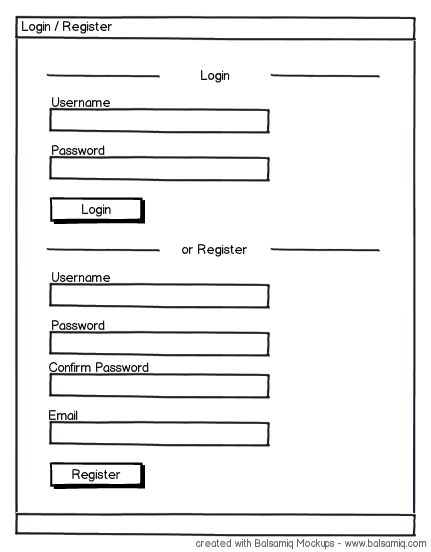


Fig.1 Fig.2

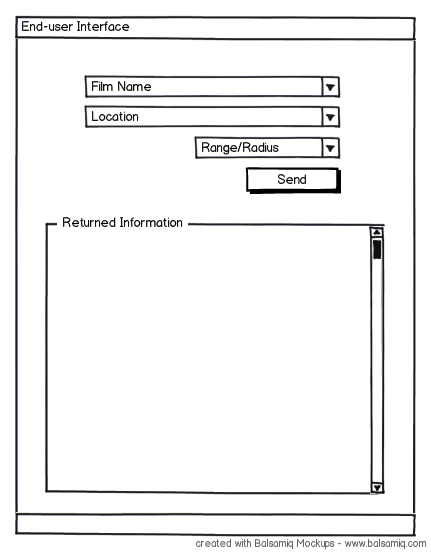


Fig. 3

# Requirements analysis

**The application needs to be accessible via a wide variety of devices from desktops to mobile phones and tablets.**

This is achieved by creating the application with a web UI that can be accessed from multiple platforms

**The application has to be easily scalable (as in responsive to increasing or decreasing traffic, ready for cloud hosting) and extensible to cities other than New York.**

New York is meant to be testing ground for the application’s attractiveness for tourists and catering vendors. It should be easily localized to other big cities in the US and Europe.

**Application will initially use New York City Open API for film shot locations but it will not be dependent on it.**

Application should at some stage be prepared for offline functioning.

**Tourists will not require logging in to the application.**

Tourists are the main consumers of the service provided by the application and they should have an easy access to it.

**Restaurant or café owners/managers in New York willing to use the application for custom acquisition will need to register and fill out a short form describing their outlet.**

It should an easy, 2-step process where after signing in business user is redirected to view/review/create/delete their profile. It will also be free.

**The application should be able to keep track of where their user is if a device has geo-location feature enabled.**

It should make use of Web maps such as Google, Yahoo or Bing.

**The service that a tourist user receives presents a list of restaurants or cafes located nearby the chosen filming spot.**

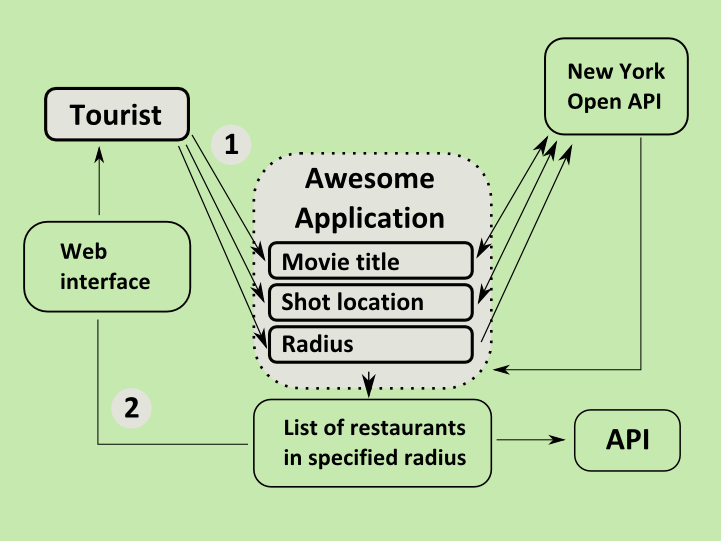
Restaurants or cafes shown in the listing will be the ones that owners or managers registered with the application.

# Use Cases

**1. Tourist visiting the Web version of the application for the first time (see Fig.1):**

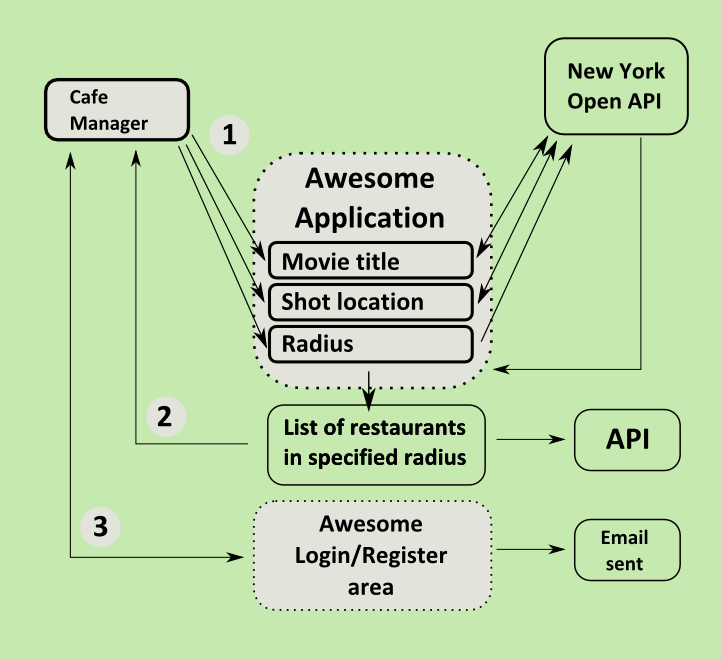
* + application is accessed via laptop or desktop interface
  + 'Movie title' drop-down list is used for movie selection
  + user (tourist) chooses a desired shot location from the second drop-down list
  + user makes a final choice of radius he/she would like to find a restaurant in from the shot location (from 0 to 100 meters)
  + user request is submitted and processed – the choice of restaurants/cafes is displayed beside/underneath the drop-down lists
  + users chooses the cafe/restaurant and reviews its profile

***Fig.1***



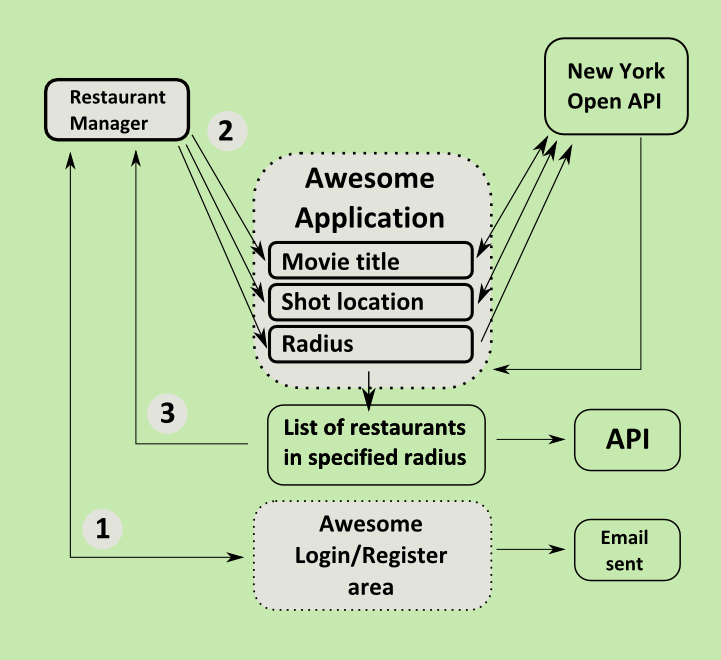
1. **Restaurant manager accessing the application for the first time (see Fig.2):**
   * clicks the 'Movie title' drop-down list (1)
   * clicks the 'Shot location' drop-down (2)
   * clicks radius or range (3)
   * after a few thoughtless attempts he realizes what the application is about
   * realizes that his/her restaurant is not there and browses the page in search of sign-up information
   * manager goes to 'Register/Sign-up' section
   * fills out a form and submits it
   * he/she is immediately redirected to a profile page
   * he/she fills out the profile and submits it
   * confirmation email is sent to the given email address
   * goes back to search area

***Fig. 2***



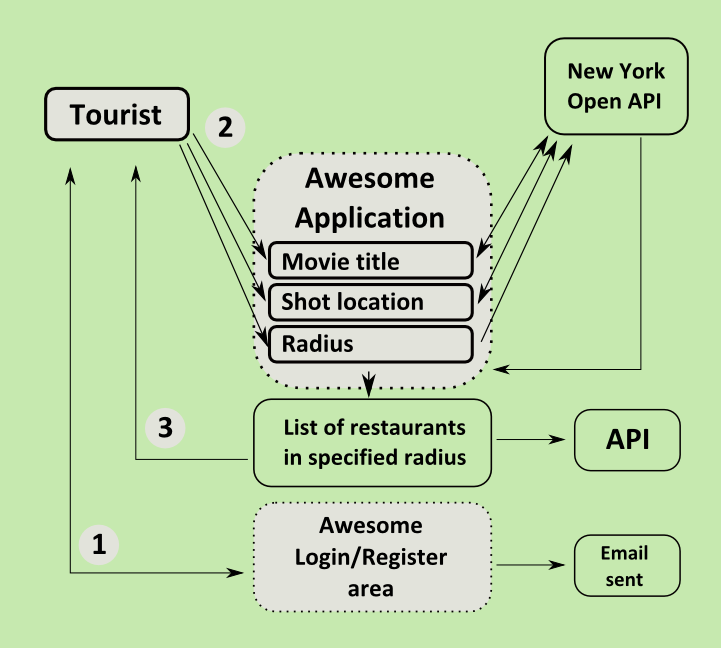
1. **Restaurant manager accessing the site/application the n-th time to update the restaurant's profile (see Fig.3):**
   * goes straight to login section
   * updates opening hours, cuisine type and the menu
   * saves changes
   * goes to the search page and selects movie title,location and radius to bring up a selection of restaurants
   * reviews the profile of his/her restaurant for submitted updates

***Fig.3***



1. **Tourist playing with application (see Fig.4):**
   * goes to register/sign-up section
   * fills out the details
   * registration form is validated (although no validation against the list of already existing restaurants will be implemented unless many such random 'stub/dummy' registrations occur) against SQL and JavaScript injection, also the address and email input have to be impeccable
   * form is submitted and notification sent to application administrator

***Fig.4***



# Architectural approach

The application follows a 5-tier structure. On the bottom-most level is the database. This provides the persistence of data, namely Film Locations and their dependent objects, and Restaurants and their dependent objects. The Film Locations objects contain lists of specific Locations, each of which contains a Point object. The Point object is a pair of coordinates. This model is referenced by both the Location object and the Restaurant object.

These models are used to represent existing constructs, as per OOP standards. The Restaurant object, for example contains details that one would expect representative of a restaurant, cuisine type, quality, etc., in addition to physical aspects such as the Address, held in a separate model.

Each model is mapped to a table in the database using EntityFramework, an ORM tool designed for use with C#. The EntityFramework provides Domain Model persistence. It looks after the relational mapping of inter-object relationships and the way that they are stored in the application. Due to the scale of the application and the code-first model with basic relationships, impedance mismatch was not an issue in developing the lower-tier structure.

Object Models are accessed through the second tier, the Data Access Layer (DAL). This layer provides a level of abstraction from the data models, providing a one-stop-shop for access methods including ‘CRUD’ functionality. A design decision was taken here to include the object creation functionality within the DAL. This means that constituent parts of objects, i.e. names, coordinates etc., are passed into the DAL methods, constructed into objects and then persisted. The reasoning behind this decision was to retain a clean space for logic in the Business Logic Layer.

As mentioned, the Point model is separate from both main models but also referenced by both. This is to make data processing more fluent in the Business Logic Layer (BLL). Here ViewModels are received from the top-most two tiers and broken out into their constituent parts. These parts, depending on the required action, are then passed to the DAL for management and persistence or passed into the Location Calculator (through the Restaurant Finder) for processing. This tool uses the Point object of the incoming Location to search through the existing persisted Restaurants matching the Restaurant’s coordinates against those of the Location to find Restaurants within a given radius. This functionality uses the algorithm in the Location Calculator.

Above the BBL lie the Presentation and View/Client-Side layers. These have their own ViewModel constructs that convey subsets of information between the UI and the BLL. Manipulation of these objects is performed in the BLL, calling persisted objects through the DAL when required.

This layered application architecture allows a clearly exposed workflow. Maintenance is made easier by defining subsets of functionality in set levels of the architecture. There is a clear separation of concerns enabling replacement of sets of functionality without excessive side-effects. The application makes extensive use of OOP principles in order to keep datasets in strict cohesion with relative elements while retaining a loosely coupled relationship with other datasets. The general communication style between layers and objects is the call-and-return style. This enables flexibility within the application.

### Data Model Examples

The data model consists of two main classes Restaurant and FilmLocations both shown below in simplified C#:

public class Restaurant

{

public int Id;

public String name;

public int cuisine;

public int fanciness;

public String websiteUrl;

public Address address;

public Point point;

}

public class FilmLocations

{

public int Id;

public string filmTitle;

public virtual ICollection<Location> locations;

}

Each Restaurant has a name, and a cuisine (an integer with for example 1 = Chinese, 2 = French, 3 = Italian etc.). It has a “fanciness” which is a rating integer from 1 to 5. There is also a website URL, a postal address and a point location, both of which latter types are shown below in simplified C#.

public class Address

{

public int Id;

public String houseNumber;

public String streetAddress1;

public String streetAddress2;

public String zipCode;

public String city;

}

public class Point

{

public int Id;

public double x;

public double y;

}

The point is a geo-location consisting of longitude and latitude. This is used to compute the restaurants within a chosen distance of the selected Movie location. The Location class is shown below in simplified C#. It contains location text (such as “5th Avenue, Manhattan”) and a geo-location point. Each FilmLocations contains a collection of Location as one movie may have more than one shooting location.

public class Location

{

public int Id;

public string locnText;

public Point point;

}

### Security for the Models

Security is handled internally within the application by the default MVC2 libraries.

# Testing Approach

The initial testing approach of the XML Reader was to run it as a stand-alone prototype and print out the return variable and verify it was a list of movies and shooting locations. The following code took care of the printing out of the variable.

foreach (var loc in locations)

{

Console.WriteLine("Film Index " + loc.index);

Console.WriteLine("Film Title " + loc.filmTitle);

foreach (var sLoc in loc.locn)

{

Console.WriteLine("Locn Index " + sLoc.index);

Console.WriteLine("Locn Display Text " + sLoc.locnText);

Console.WriteLine("Locn lat " + sLoc.latCoord);

Console.WriteLine("Locn lng " + sLoc.lngCoord);

Console.WriteLine("Locn Radius " + sLoc.radius);

}

// Console.ReadLine();

}

}

Console.ReadLine();

The main XML Reader class could be slotted in into this prototype for debugging if it stopped working in the main application. The following is an excerpt of the start of the XML output to screen.

Film Index 0

Film Title \*batteries not included

Locn Index 0

Locn Display Text E. 5th St.<br>East Village<br>Manhattan

Locn lat 40.7224452961828

Locn lng -73.9786505699157

Locn Radius 0

Film Index 1

Film Title 12 Angry Men

Locn Index 0

Locn Display Text New York County Courthouse<br>40 Foley Square<br>Lower Manhattan

Locn lat 40.7137

Locn lng -74.0079

Locn Radius 0

Film Index 2

Film Title 13 Going on 30

Locn Index 0

Locn Display Text W. 47th St. and Seventh Ave.<br>Times Square<br>Manhattan

Locn lat 40.7592204876521

Locn lng -73.9846211671829

Locn Radius 0

The following is an excerpt from the end of the XML data.

Film Index 176

Film Title You're a Big Boy Now

Locn Index 0

Locn Display Text Steeplechase Park<br>Coney Island<br>Brooklyn

Locn lat 40.574725662771

Locn lng -73.9803296327591

Locn Radius 0

Locn Index 1

Locn Display Text New York Public Library<br>Fifth Ave. and 41st St.<br>Manhattan

Locn lat 40.7528651331099

Locn lng -73.9815366268158

Locn Radius 0

Film Index 177

Film Title You've Got Mail

Locn Index 0

Locn Display Text Verdi Square<br>W. 73rd St. & Broadway<br>Manhattan

Locn lat 40.7792578185718

Locn lng -73.9815366268158

Locn Radius 0

Locn Index 1

Locn Display Text 91st Street Community Garden<br>Riverside Park<br>Manhattan

Locn lat 40.7928238690811

Locn lng -73.9774167537689

Locn Radius 0

This data provided information for further unit testing of the XML Reader and code which called it. addEntry which takes care of adding a film and one location, has such a unit test. The unit test is as follows:

public void addEntryTest()

{

LocnXMLReader\_Accessor target = new LocnXMLReader\_Accessor();

string filmName = "Die Hard";

double latCoord = 30.5;

double lngCoord = -70.25;

string locnDisplayText = "Manhattan";

target.addEntry(filmName, latCoord, lngCoord, locnDisplayText);

Assert.AreEqual(target.filmLocations.Count, 1);

Assert.AreEqual(target.filmLocations[0].filmTitle, "Die Hard");

Assert.AreEqual(target.filmLocations[0].locations.Count, 1);

Assert.AreEqual(target.filmLocations[0].locations[0].point.x, 30.5);

Assert.AreEqual(target.filmLocations[0].locations[0].point.y, -70.25);

Assert.AreEqual(target.filmLocations[0].locations[0].locnText, "Manhattan");

filmName = "Die Hard";

latCoord = 25.5;

lngCoord = -55.25;

locnDisplayText = "Brooklyn";

target.addEntry(filmName, latCoord, lngCoord, locnDisplayText);

Assert.AreEqual(target.filmLocations.Count, 1);

Assert.AreEqual(target.filmLocations[0].filmTitle, "Die Hard");

Assert.AreEqual(target.filmLocations[0].locations.Count, 2);

Assert.AreEqual(target.filmLocations[0].locations[0].point.x, 30.5);

Assert.AreEqual(target.filmLocations[0].locations[0].point.y, -70.25);

Assert.AreEqual(target.filmLocations[0].locations[0].locnText, "Manhattan");

Assert.AreEqual(target.filmLocations[0].locations[1].point.x, 25.5);

Assert.AreEqual(target.filmLocations[0].locations[1].point.y, -55.25);

Assert.AreEqual(target.filmLocations[0].locations[1].locnText, "Brooklyn");

filmName = "Die Hard 2";

latCoord = 22.75;

lngCoord = -66.66666666667;

locnDisplayText = "Queens";

target.addEntry(filmName, latCoord, lngCoord, locnDisplayText);

Assert.AreEqual(target.filmLocations.Count, 2);

Assert.AreEqual(target.filmLocations[1].filmTitle, "Die Hard 2");

Assert.AreEqual(target.filmLocations[1].locations.Count, 1);

Assert.AreEqual(target.filmLocations[1].locations[0].point.x, 22.75);

Assert.AreEqual(target.filmLocations[1].locations[0].point.y, -66.66666666667);

Assert.AreEqual(target.filmLocations[1].locations[0].locnText, "Queens");

}

This unit test is especially necessary as the function was changed to interface directly with the Data Access Layer and thereby indirectly to the database. The View does not access the XML Reader but accesses the database via the Business Logic and Data Access Layers. This unit test will guard against further changes to the program breaking addEntry.

The LocationFinder has also a straight forward unit test based on the XML data read by the stand-alone prototype. The unit test is as follows:

public void getLocationsForFilmTest()

{

LocationFinder target = new LocationFinder();

String filmName;

LocationListUI expected, actual;

filmName = "12 Angry Men";

expected = new LocationListUI();

expected.filmName = filmName;

expected.locations = new List<String>();

expected.locations.Add ("New York County Courthouse<br>40 Foley Square<br>Lower Manhattan");

actual = target.getLocationsForFilm(filmName);

Assert.AreEqual(expected, actual);

filmName = "15 Minutes";

expected = new LocationListUI();

expected.filmName = filmName;

expected.locations = new List<String>();

expected.locations.Add("E. 60-66th St.and Madison Ave.<br>Upper East Side<br>Manhattan");

actual = target.getLocationsForFilm(filmName);

Assert.AreEqual(expected, actual);

filmName = "25th Hour";

expected = new LocationListUI();

expected.filmName = filmName;

expected.locations = new List<String>();

expected.locations.Add("World Trade Center<br>Lower Manhattan");

expected.locations.Add("Carl Schurz Park<br>Upper East Side<br>Manhattan");

actual = target.getLocationsForFilm(filmName);

Assert.AreEqual(expected, actual);

}

The main View is the HomeController. The unit test for this also makes use of the XML data read by the stand-alone XML prototype. The code for this follows:

public void Index()

{

// Arrange

HomeController controller = new HomeController();

// Act

ViewResult result = controller.Index() as ViewResult;

// Assert

ViewDataDictionary viewData = result.ViewData;

Assert.AreEqual("Welcome to Awesome!", viewData["Message"]);

List<SelectListItem> flicks = (List <SelectListItem> )viewData["movieList"];

Assert.AreEqual(178, flicks.Count);

Assert.AreEqual("\*batteries not included", flicks[0].Text);

Assert.AreEqual("12 Angry Men", flicks[1].Text);

Assert.AreEqual("15 Minutes", flicks[3].Text);

Assert.AreEqual("Die Hard: With a Vengeance", flicks[37].Text);

Assert.AreEqual("You've Got Mail", flicks[177].Text);

List<SelectListItem> locs = (List<SelectListItem>)viewData["locationList"];

Assert.AreEqual(1, locs.Count);

if (locs.Count > 0)

Assert.AreEqual("E. 5th St.<br>East Village<br>Manhattan", locs[0].Text);

}

Unit tests are very useful in finding bugs (via the Test Debug option) and also ensuring that if the code breaks due to future changes it usually flags the break.

# Use of Client-Side Processing/AJAX

Awesome Enterprise Application is no different to a majority of currently created Web applications using JavaScript and AJAX technique to process data on the client side.

AJAX finds a lot of uses in A.E. Application – as it is created with rich end user interactions in mind. Data output either on the User Interface on a mobile device (main target group of users) or any other human-computer type of interface is best updated in runtime, easing end user of refreshing the page to see thir search results.

As the application uses ASP.NET MVC 2 framwork the default client side processing engine is the same as in classic Web Forms – it uses .aspx pages to present data for interaction with the user.

As MVC framework comes with jQuery library built into the list of default project featuresit is also handling creating XMLHttpRequest object and processing the request to the server.

AJAX calls are route to the a specified controller and its action (method) passing data that is an argument for that method.

Features powered by AJAX in Awesome Enterprise Application:

* Populating 2nd dropdown list on the homepage (list of movie shot locations) – this call is a response to end user action of choosing the movie title.
* Home page form submission – collection of data fromn the 3 dropdown lists is passed to another controller (RestaurantFinderController) that then communicates with a RestaurantFinder class in application's Business Logic layer.
* (Not implemented) Restaurant profile form and business user registration form will also use AJAX (no data will be output, just an ellegant message appended to the form on a successful save to a database).

# Use of Web Services

The Film Locations stored in the application are drawn directly from the API at

[**https://nycopendata.socrata.com/Business-and-Economic/Filming-Locations-Scenes-from-the-City-/qb3k-n8mm**](https://nycopendata.socrata.com/Business-and-Economic/Filming-Locations-Scenes-from-the-City-/qb3k-n8mm)

This is part of the New York City Open Data Initiative, designed to give public access to many elements of the data collection of New York, from train times to restaurant menus.

This application takes advantage of a particular API that lists all the locations in New York where famous films have been made. This API is ingested in XML format. It is parsed with a custom built XML Parser. The resulting information is passed into the DAL where it is aggregated into persistable FilmLocations objects.

# Use of an ORM tool

The Application uses EntityFramework as its ORM tool. This is a complex framework designed specifically for C#. It handles both database construction/maintenance and data persistence.

In this application it has been used with the Domain Model approach. Models were built that would relate on OOP principles to the objects that they define. EntityFramework was then used to give these models general table mappings. The object relationships and dependencies were not mapped in the code.

The aim of this approach is persistence ignorance. Using the OOP structure in combination with EntityFramework’s DbContext class, it was possible for EntityFramework to define the relationships within the database without manual interference from the developers.

# References

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007) ‘DBPedia: A Nucleus for a Web of Open Data’ In: *The Semantic Web: Lecture Notes in Computer Science.* Berlin:Springer, 722-735.

Kadluczka, P. (2011) ‘Effective Xml Part 1: Choose the right API’ in *Microsoft XML Team Blog,* Available from: <http://blogs.msdn.com/b/xmlteam/archive/2011/09/14/effective-xml-part-1-choose-the-right-api.aspx> [Accessed 14th July 2012]

Streeter, L.A., Kraut, R.E., Lucas, H.C. & Caby, L. (1996) ‘How Open Data networks Influence Business Performance and Market Structure’. *Communications of the ACM* 39 (7): 62-73.

# Appendix: New York Filming Locations XML Data

The XML Data is plainly downloaded from an MS Office Excel spreadsheet file. This can be seen from its structure. It is divided into rows and cells unlike a straight forward XML file, which might contain the following excerpt.

<Film>Die Hard

<Location>

<Place>

<Name>Manhattan</Name>

<Latitude> 40.7792578185718</Latitude>

<Longitude>-73.9815366268158</Longitude>

</Place>

</Film>

In the actual XML file the first row contains nothing relevant. The second row contains the titles for each column. Further rows contain the data such as film title and location name. The type of data is given by the cell position in the row. Each cell’s type is given by that cell’s title in the second row. For instance if cell 1 in row 2 is “Film” then all rows (after the second) contain the film title in cell 1. Similarly “Location Display Text” indicates which cell contains the location names. To indicate more than one shooting location for a film the film name is repeated in the next entry. For instance:

{“Die Hard”, “Manhattan”, “30.0”, “-75.0”},

{“Die Hard”, “Queens”, “35.0”, “-74.0”},

The user interface sees this as one film (“Die Hard”) with two locations (“Manhattan”, and “Queens”). The input XML data is structured as follows:

<Workbook>

<DocumentProperties>

...

</DocumentProperties>

<OfficeDocumentSettings>

...

</OfficeDocumentSettings>

<ExcelWorkbook>

...

</ExcelWorkbook>

<Styles>

...

</Styles>

<Worksheet ss:Name="FullMapList">

<Names>

...

</Names>

<Table>

<Column>

...

</Column>

<Row>

...

<Row ss:Index="2" ss:Height="18.75">

<Cell ss:StyleID="s32"><Data ss:Type="String">Scenes from the City: Locations</Data></Cell>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s97"/>

<Cell ss:StyleID="s57"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s90"/>

<Cell ss:StyleID="s47"/>

<Cell ss:StyleID="s48"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s32"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s56"/>

</Row>

<Row ss:Height="38.25">

<Cell ss:StyleID="s24"><Data ss:Type="String">Film</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Year</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s96"><Data ss:Type="String">URL Encoded name</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Image File Name</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Agency Credit</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Artist Credit</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Director/Filmmaker Name</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Director/Filmmaker IMDB Link</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s91"><Data ss:Type="String">Location Display Text</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s49"><Data ss:Type="String">LATITUDE</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s49"><Data ss:Type="String">LONGITUDE</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Borough</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Neighborhood</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Scene Type</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Media</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">IMDB LINK</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Client or book location indicator</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Notes</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Book Image</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Book Page</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Display?</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">IMAGE OF LOCATION</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

</Row>

...

<Row ss:Height="25.5" ss:StyleID="s37">

<Cell ss:StyleID="s26"><Data ss:Type="String">Hannah and Her Sisters</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><Data ss:Type="Number">1986</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s95"><Data ss:Type="String">Hannah%20and%20Her%20Sisters</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s46"><Data ss:Type="String">Directed by</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><Data ss:Type="String">Woody Allen</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">http://www.imdb.com/name/nm0000095/</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s93"><Data ss:Type="String">W. 95th St. and West End Ave.&lt;br&gt;Upper West Side&lt;br&gt;Manhattan</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s50"><Data ss:Type="Number">40.794600000000003</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s50"><Data ss:Type="Number">-73.973699999999994</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">Manhattan</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">Upper West Side</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">N/A</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">Film</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">http://www.imdb.com/title/tt0091167/</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">95th Street and West End Avenue</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s43"><Data ss:Type="String">N</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s43"><Data ss:Type="String">N</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

</Row>

...

1. http://data.fingal.ie/ [↑](#footnote-ref-1)
2. https://nycopendata.socrata.com/ [↑](#footnote-ref-2)