National College of Ireland

2011/2012

Rate My Amenity



Project for:

Enterprise Frameworks Module

3rd Semester

PGDip Cloud Computing

Project Members:

Anna Coffey, x10210504

Ken Ryan, x10208828

Gerard Tamakloe, x10209778

Brian Wade, x10210989**Table of Contents**

1. Project Summary 3

2. Project Plan 3

3. Software development methodology employed 4

4. Requirements analysis 5

5. Use Cases 6

6. Application Framework 7

7. Architecture/Design Approach 8

8. Models (Class Models / Data Models) 13

9. Data 15

10. Development Phase 17

11. Implementation of particular OOP constructs 18

12. How cross-cutting concerns have been handled 19

13. Security of the application 20

14. Configuration of the application 21

15. Scalability of the application 22

16. Testing Approach 23

17. Appendices 25

# Project Summary

The ‘Rate my Amenities’ enterprise application is intended to provide users with a place to search for and view amenities in their local area, rate the amenity and view other users ratings and comments.

Users will be able to search for amenities using a range of criteria including location and amenity type. Selecting a specific amenity displays the amenities details, all rating details against this amenity, the location of the amenity on a map and disabled parking details in the area. From this section users can opt to add a rating to the chosen amenity. Users who are not logged in will be prompted to do so.

Logged in users with administration privelages will also have the ability to view a list of all amenties and either edit, delete or add to this list. Admin users will also be able to view all ratings, link to a ratings api and view average ratings per amenity.

# Project Plan

|  |  |
| --- | --- |
| Anna Coffey, x10210504 | * Coding of ViewModels, DAL, BLL, and associated Views and Controllers. |
| Ken Ryan, x10208828 | * Coding of Ratings API & Unit Tests * Coding support & research. * User Authentication & Security * Technical Documentation |
| Gerard Tamakloe, x10209778 | * Coding of Bing Maps, Unit Tests & Ratings API, CSS & Ajax functionality. |
| Brian Wade, x10210989 | * Technical Documentation * Wireframes * User Authentication & Security * Coding support & research |

# Software development methodology employed

This application was developed using Microsoft Visual Studio 2010, ASP.NET, C#, JavaScript and Ajax.

The database used in the development of this application was SQL Server Compact Edition. This is a free, embedded, file based database which does not require any installation or configuration.

We also used the Entity Framework version 4 to query and update the database. This is a flexible object relational mapping data API (ORM) which queries and updates the data stored in the database in an object oriented manner.

The application was developed using a code first methodology which generates a new database from the model.

The basic shape of the model is set by using conventions which are rules which automatically configure a conceptual model based on class definitions. These conventions are defined in the System.Data.Entity.ModelConfiguration.Conventions found in the /DAL/RateMyAmenityContext.cs

# Requirements analysis

* **Functional Requirements**

1. Call a variety of data sources using an APIfrom Fingal Open Data <http://data.fingal.ie/> and save it to a database.
2. Use search filters to display appropriate data to the user.
3. Use a Bing Maps API to display locations of core data.
4. Provide a Twitter information feed related to news in Fingal.
5. Provide an Administrator Login to be stored in the Membership Database
6. Provide a User Login to be stored in the Membership Database
7. Allow logged in users to insert comments.
8. Allow logged in users with administration privileges to view additional information.
9. Provide an API on the ratings data.

* **Non Functional Requirements**
  1. Provide a basic user interface.
  2. AJAX implementation to display amenities data summary.

# Use Cases

Actors

1. Non Registered User

2. Registered User

3. Administrator

4, System

Non Registered User

1. User browses index page.

2. User searches for amenity

Registered User

1. User browses index page.

2. User searches for amenity

3. User adds ratings

4. User adds amenity.

Administrator

1. User browses index page.

2. User searches for amenity

3. User adds ratings

4. User adds amenity.

5. User edits and deletes ratings

6. User edits and deletes amenities

System

1. System interacts with database

2. System interacts with Bing Maps by means of an API.

3. System interacts with external data source.

# Application Framework

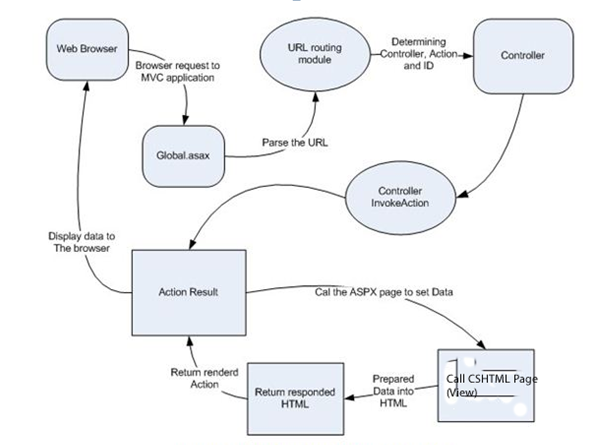


Figure 6.1. Representaiton ASP.Net MVC Flow

# Architecture/Design Approach

An architectural pattern expresses a fundamental structural organization for software systems. It provides a set of predefines subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

Design Patterns provide a scheme for refining the components of a software system and the relationships between them.

The architectural patterns used in this application were the use of layers and MVC.

This application sought to implement a 5 layer architectural pattern as illustrated in figure 7.1 below.

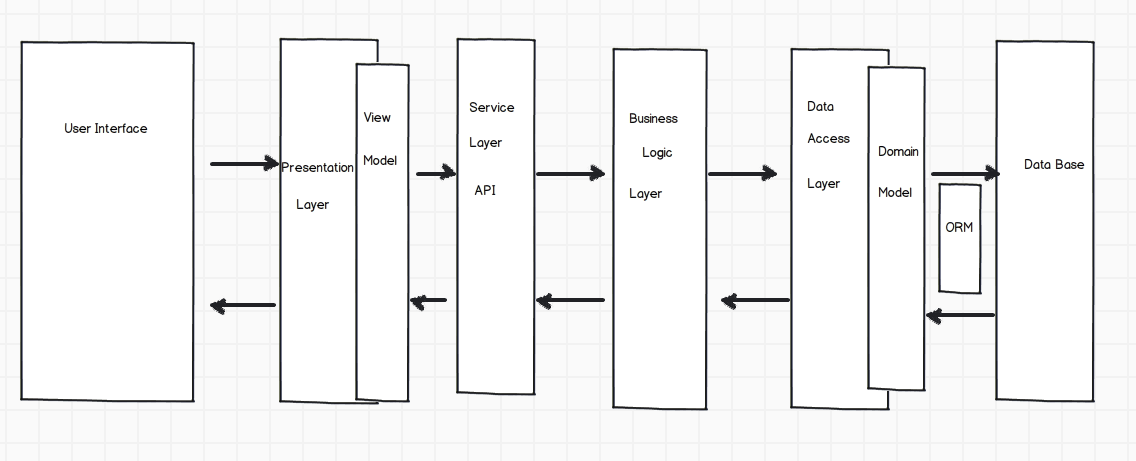


Figure 7.1. 5-Tier Architectural Model

**The Layers**

1. User Interface

This is the layer which the user sees and interacts with. The primary files in this layer are outlined below.

* Views/Account
  + Register & Login. These are standard views available in the MVC4 framework.
* Views/Amenity
  + Index.cshtml – this is the home page of the application (as specified in /AppStart/RouteConfig.cs). This page displays the follows partials.
    - \_Home (/Shared/\_Home) which displays the search box and amenity details.
    - \_MapCheck (Shared/\_MapCheck) – which displays the map.
    - \_Twidget – which displays the Twitter feed.
  + List.cshtml – this page displays the full list of amenities (to an authenticated admin user) along with edit, delete and add links.
  + Details.cshtml – This page displays the details of a chosen amenity, along with the ratings for this amenity. It also displays the partials associated with disabled parking in the area (/views/amenity/\_parkingdetails) and the map (views/amenity/\_map).
* Views/Home
  + About.cshtml – a general about page
  + Apiview.cshtml – displays the api data in html format.
* Views/Rating
  + Index.cshtml – this page is only available to authenticated users with administration privilegas. It displays the following links and details
    - Links to the XML API data and HTML API data
    - A list of all ratings in the database
    - A summary of ratings grouped by AmenityID (/views/rating/\_ratingdetails)

1. Presentation Layer

The Presentation Layer is responsible for handling the interaction between the user and the application. It includes the ViewModel which provides a way to manage and shape data from one or more data models that will be rendered by the view. It provides a single object to render and lessens the need for UI logic code in the view.

The following viewmodels were used in this project:

* RatingDetails.cs
* RateAPI.cs
* ParkingDetails.cs

1. Service Layer – API

Asp.net includes a Web API framework that makes it easy to build HTTP services that are accessible by a broad range of clients. The Rating API included in RateMyAmenities exposes the Ratings data converting it to XML/JSon, depending on the client.

The API includes the following files.

* /Controller/RateAPIController.cs
* /ViewModels/RateAPI.cs
* /Home/APIView.cshtml
* /Scripts/apiam.js

1. Business Logic Layer

The Business Logic Layer includes the core logic and functional algorithms of the application. It uses Transaction Script which organizes each procedure to handle a single request from the Presentation Layer

The Transactions Scripts are based on the use cases.

Within the scope of this project the BLL layer contains the following file.

* BLLGetCSV.cs - reads the csv files from Fingal Co.Co, sends the data to a CSV parser (/dataimport/csvparsers.cs & IdataParser.cs), and passes the result to the DAL to save into the DB.

1. Data Access Layer

The Data Access Layer (DAL) is responsible for reading Data from and persisting data to the data store. The DAL makes use of the Domain Model which allows it to become a separate layer.

RateMyAmenity contains the following DAL files.

* AmenityDal.cs – contains two methods as follows:
  + GetAvgRating – to retrieve the ratings for a particular amenity and group into an average
  + GetParkingDetails – Retrieves disabled parking details, within a given range, for a particular amenity.
* GetCSV.cs – stores the imported CSV data (from Fingal Co.Co) to the amenities and parking databases.
* RateMyAmenityContext.cs – the file that coordinates Entity Framework functionality for the Amenities, Ratings and Parking data models.

1. Database

Data is stored in the Database which is created by the use of the Domain Model. The database tables are as per figure 7.2 below.

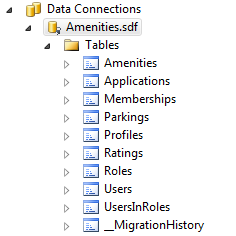


Figure 7.2 – Amenities.sdf Tables.

**MVC Design**

We also employed the MVC design pattern. We chose MVC for a number of reasons

* It’s action centric, focusing on the users actions when using the web page
* It exhibits a high degree of Separation of Concerns
* User requests are resolved by calling a method in the controller
* MVC employs a RESTful framework
* MVC is a modular architecture which facilitates a high degree of flecibility

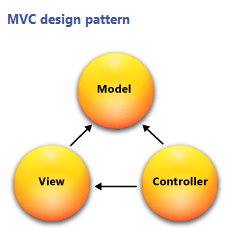


Figure 7.3 – MVC Model

**Strategy and Adapter Patterns**

The strategy pattern is an OOP Pattern which allows the combining of a group of operations into a small class hierarchy. This pattern was used in conjunction with the Adapter design.

The Adapter Designis a structural design pattern which allows the application to use classes whose interfaces don’t completely match its requirements.An example of this in RateMyAmenities is the importation of the external CSV file where only a subset of the data was imported.

The abstraction layer between the Data Access Layer (DAL) and the Business Logic Layers of the application are created using the Repository and Unit of Work patterns.

# Models (Class Models / Data Models)

A number of Data Models and classes were created based on the data in our datasets. We then created relationships between the models.

**Models**

* Account Model
* Amenity Model
* Parking Model
* Rating Model

**Classes**

See figure 8.1 below.

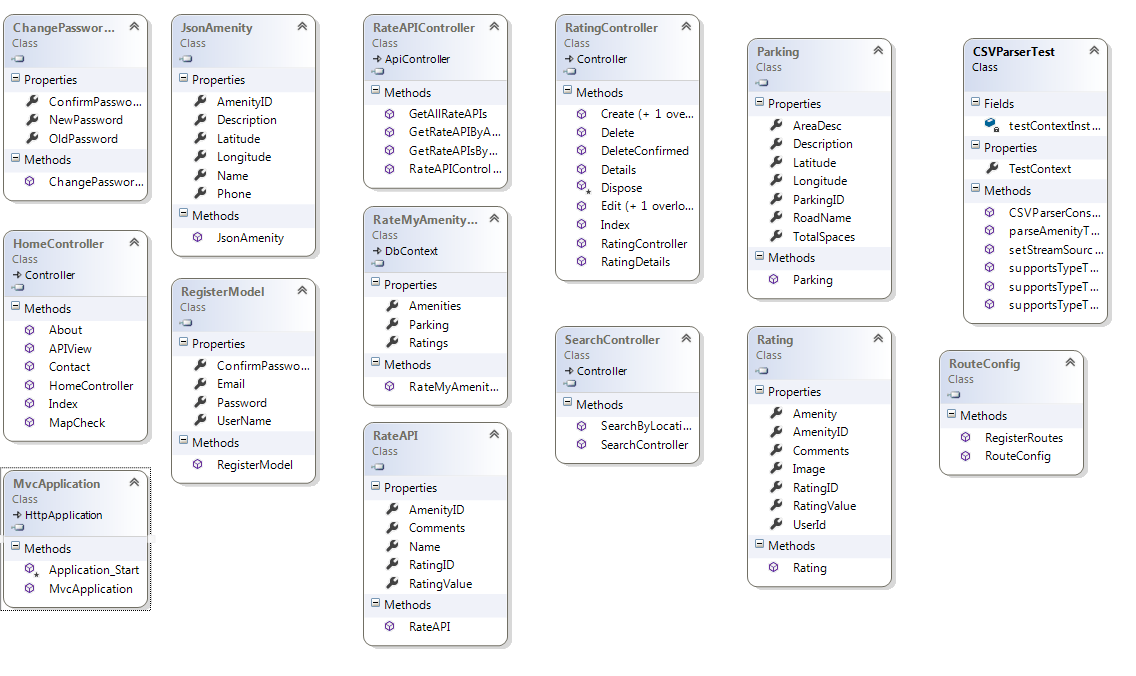
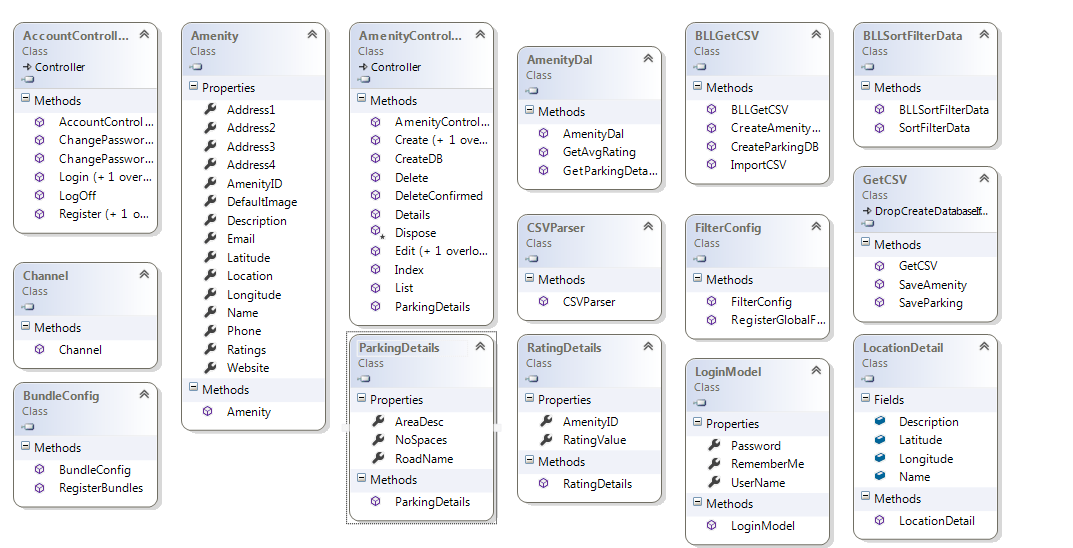


Figure 8.1 – Classes

# Data

Data for the amenities used in this project are obtained from the public datasets provided by Fingal County Council ([http://data.fingal.ie/).](http://data.fingal.ie/).%20%20)  As the data is not subject to regular changes the data will be imported once at the start of the project. For this project the datasets have been limited to a small range of amenities only and have not included services i.e. Garda Stations. The amenities used are and their corresponding URL is as follows.

* + Art Centres - http://data.fingal.ie/datasets/csv/Arts\_Centres.csv
  + Cinemas - http://data.fingal.ie/datasets/csv/Cinemas.csv
  + Heritage Venues - http://data.fingal.ie/datasets/csv/Heritage\_Venues.csv
  + Parks - http://data.fingal.ie/datasets/csv/Parks.csv
  + Play Areas - http://data.fingal.ie/datasets/csv/Play\_Areas.csv
  + Parking - http://data.fingal.ie/datasets/csv/Parking.csv

**File Format and Data Set**

For most of the amenities the data is available in CSV, XML and KML formats. However for this project the CSV file type was used.

While a number of the amenities have an extensive range of data available only a subset of the data was used for this project. The fields used were common across all of the datasets and these were Name, Address1, Address2, Address3, Address4, Phone, Email, Website, LAT and LONG.

## Data Import

As outlined in section 2 the data for the amenities is imported from public datasets provided by Fingal Co. Co. To import the data the following code was implemented.

* Each of the dataset URL’s and a definition of its ‘type’ were added to a multidimensional array called amenities in the model/DBInitialiser.cs.
* A For loop is used to navigate through each of the datasets and complete the following.
  + Create a new instance of the CSVParser
  + Fetch the data from the URL using a HTTPWebRequest
  + Stream the data through the parser
  + Pass in the amenity ‘type’.
  + Add and save the data to the database.

# Development Phase

## To setup construct for import CSV data file

DataImport Code:

IDataParser.cs : located in /DataImport, created to insure that any class created for that interface includes three methods.

CSVParser.cs: located in /DataImport , creates a new instance of IDataParser.

Unit Tests:

Units Tests located in /CSVPaserTest.cs

Unit test were carried out to test the parser to:

* + test for csv files
  + test for non csv files
  + test for null file (csv parser file insert value == null)
  + set up test array for expected amenities
  + test for object within CSV data fields

## Database Migrations

To create the initial migration we ran the Package Manager Console

* Tools => Library Package Manager =>Package Control Console
* Enable – migrations ‘creates initial migration class’ InitialCreate.cs
* Add migration
  + Name new image column
* This create a new migration class called newimagecolumn.cs’

# Implementation of particular OOP constructs

We ensured the use of loose coupling by using an MVC Framework. This ensured that the components of the application were independent of each other.

# How cross-cutting concerns have been handled

Cross cutting refers to the functionality found in the application which does not fit neatly into one place and breaks the separation of concerns pattern.

We employed the use of filters to inject extra logic into the application which deal with any cross cutting concerns.

Our particular use of filters included the use of authorizations which controlled member access to features which we accessible by the administrator.

The authorization was applied to the action method of the controllers to which limited accessibility was required.

# Security of the application

The application uses ASP.Net’s Membership Framework to manage user accounts.

ASP.Net manages security in a number of ways.

1. By authenticating the user.
2. By authorizing the user
3. Setting up User Accounts
4. Setting up Roles.

Some key terms associated with security include.

* Authentication is the process of ascertaining the users identity.
* Authorization is the process of determining whether the user has the authority to access a specific resource.
* A user account is a store for persisting information about a particular user.
* A role is a label assigned to a user and provides a means for defining authorization rules and page level functionality.
* The application defines Administrator and User Roles with the Administrator having full access to the applications functions.

In this version of the application we chose not to implement the following security features due to time constraints

* SQL Injection
* Cross Site Request Forgery(CSRF)
* Mass Assignment
* Cross Site Scripting (XSS)
* Data Validatation

# Configuration of the application

The Web Configuration file is where the Entity Framework is connected to the database by matching the main attribute with the context class.

<connectionStrings>

<add name="RateMyAmenityContext" connectionString="Data Source=|DataDirectory|Amenities.sdf" providerName="System.Data.SqlServerCe.4.0"/>

</connectionStrings>

In addition, by editing the RouteConfig file we mapped the applications route to the Index page

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Amenity", action = "Index", id = UrlParameter.Optional }

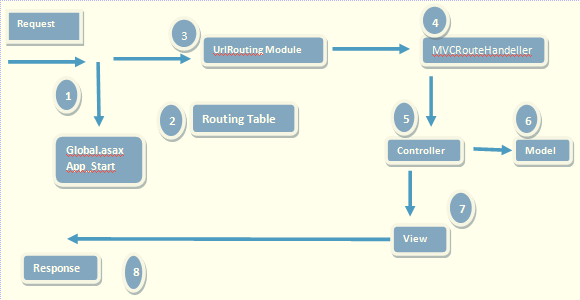


Figure 14.1 – Routing Diagram

# Scalability of the application

An application is considered scalable when it can accommodate more loads and a larger data set

The application by its nature is scalable for the following reasons

It utilizes Language Interface Query (LINQ) which provides it with a common querying language.

It allows bi-directional mapping by using code first.

The development of the application allows us to change data sources and expand the amount of data with which it interfaces.

The Application was made scalable by the use of interfaces and classes in order to provide a layer of abstraction from an import source which can be modified without altering the code

# Testing Approach

We approached testing by developing tests as we progressed with the development of the application.

ASP.Net provides tools for the development of Unit Tests.

**Definition: Unit Testing**

Unit Testing is a method by which units of source code are tested to see if they are fit for use. Unit Tests can be written quickly and are fully automated.

**CSVParser.Test**

// test to check format = csv

[TestMethod()]

public void supportsTypeTest1()

{

IDataParser target = new CSVParser();

string format = "csv";

bool expected = true;

bool actual;

actual = target.supportsType(format);

Assert.AreEqual(expected, actual);

}

// test to check format for null (i.e. no file)

[TestMethod()]

public void supportsTypeTest2()

{

IDataParser target = new CSVParser();

string format = null;

bool expected = false;

bool actual;

actual = target.supportsType(format);

Assert.AreEqual(expected, actual);

}

// test to check format = something other than csv

[TestMethod()]

public void supportsTypeTest3()

{

IDataParser target = new CSVParser();

string format = "csv";

bool expected = false;

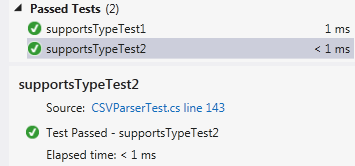
bool actual;

actual = target.supportsType(format);

Assert.AreEqual(expected, actual);

}

**Result**



**Search Controller Test**

[TestMethod()]

[UrlToTest("http://localhost:50265/")]

public void SearchByLocationTest()

{

SearchController target = new SearchController(); // TODO: Initialize to an appropriate value

float longitude = 0F; // TODO: Initialize to an appropriate value

float latitude = 0F; // TODO: Initialize to an appropriate value

ActionResult expected = target.SearchByLocation(longitude, latitude); // TODO: Initialize to an appropriate value

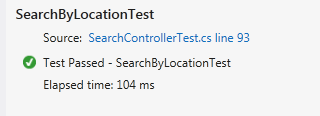
ActionResult actual;

actual = target.SearchByLocation(longitude, latitude);

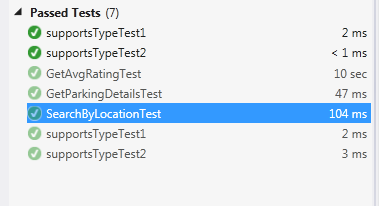
Assert.AreEqual(longitude, latitude);

}

**Results**



**Other Tests**



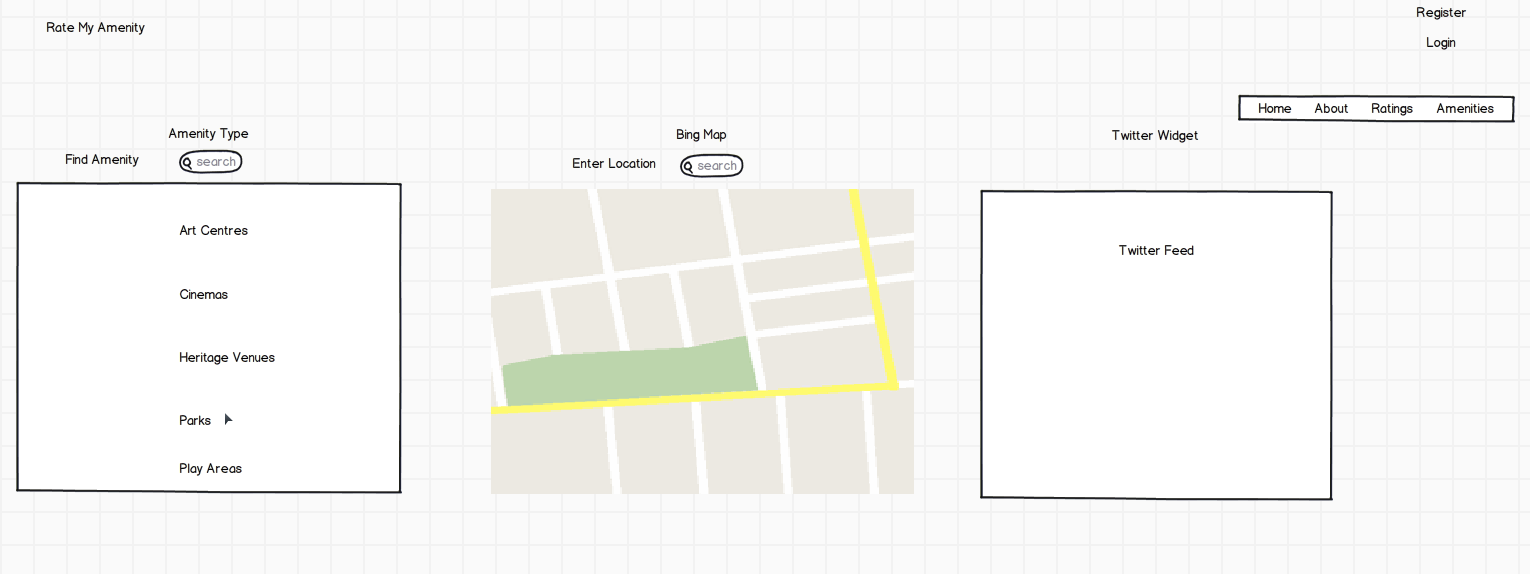
**Other relevant features of the application, e.g.,**

* Client-side processing
  + JavaScript was used for the Bing Mapping Tool which pulls in a list of amenities based on the search parameters which are displayed as pins for each amenity with mouse rollover info boxes displaying the type and name of the amenity.
* Ajax
  + Ajax was employed to create a real time clock on the home page which automatically updates every second.
  + We also implemented Ajax on the Home Page drop down list which hides the amenities within the amenity category.
  + A twitter feed is also provided on the Home page
  + We also implemented a getJSON function to send an ajax request. The response will be an array of JSON objects. The second parameter to getJSON is a callback function that is invoked when the request is completed which dispalys an XML file of the amenities
* ORM tool
  + While we did not use any specific ORM tools, we employed a Data Migration

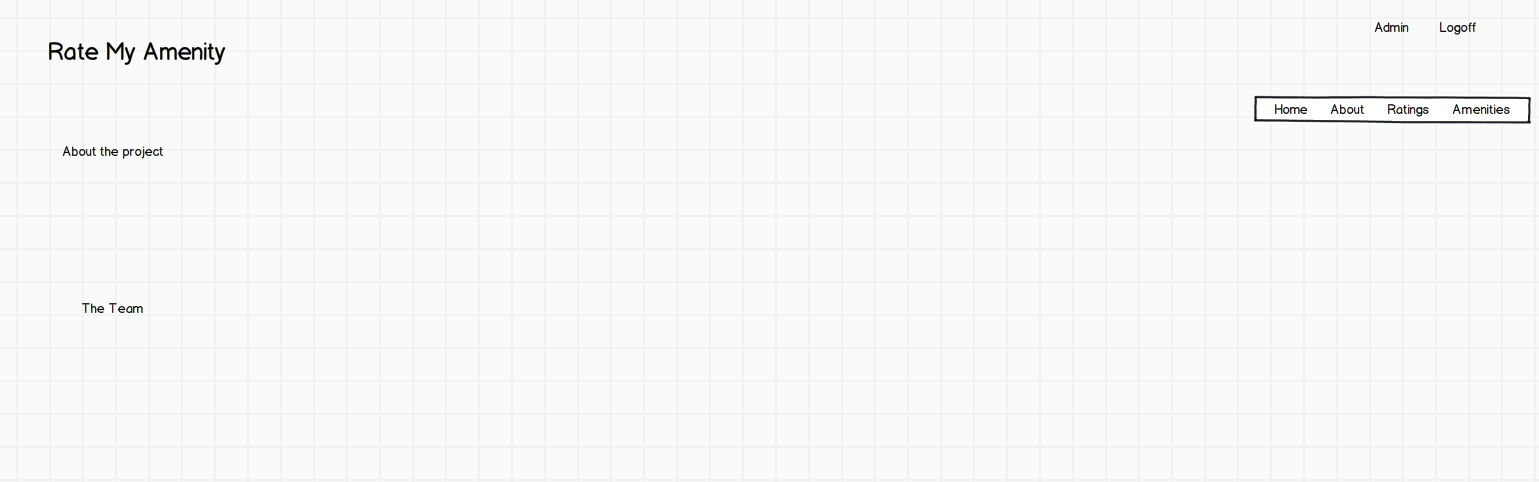
# Appendices

WireFrames

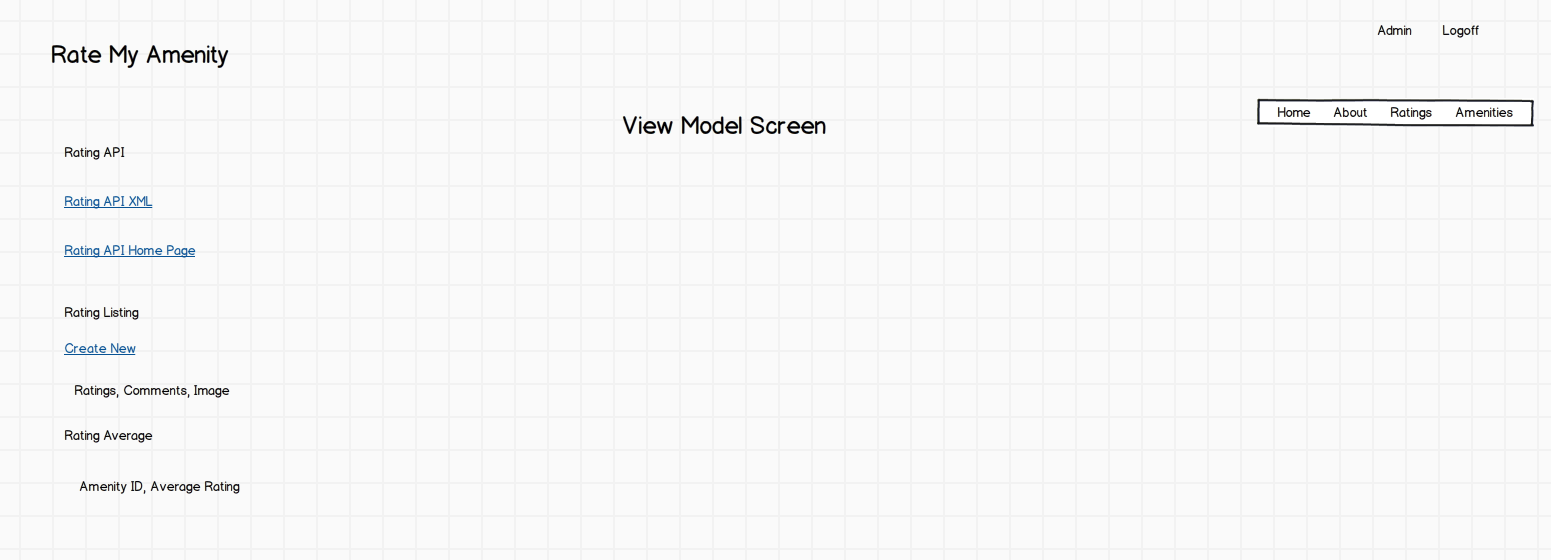
**Home Page**



**About Page**



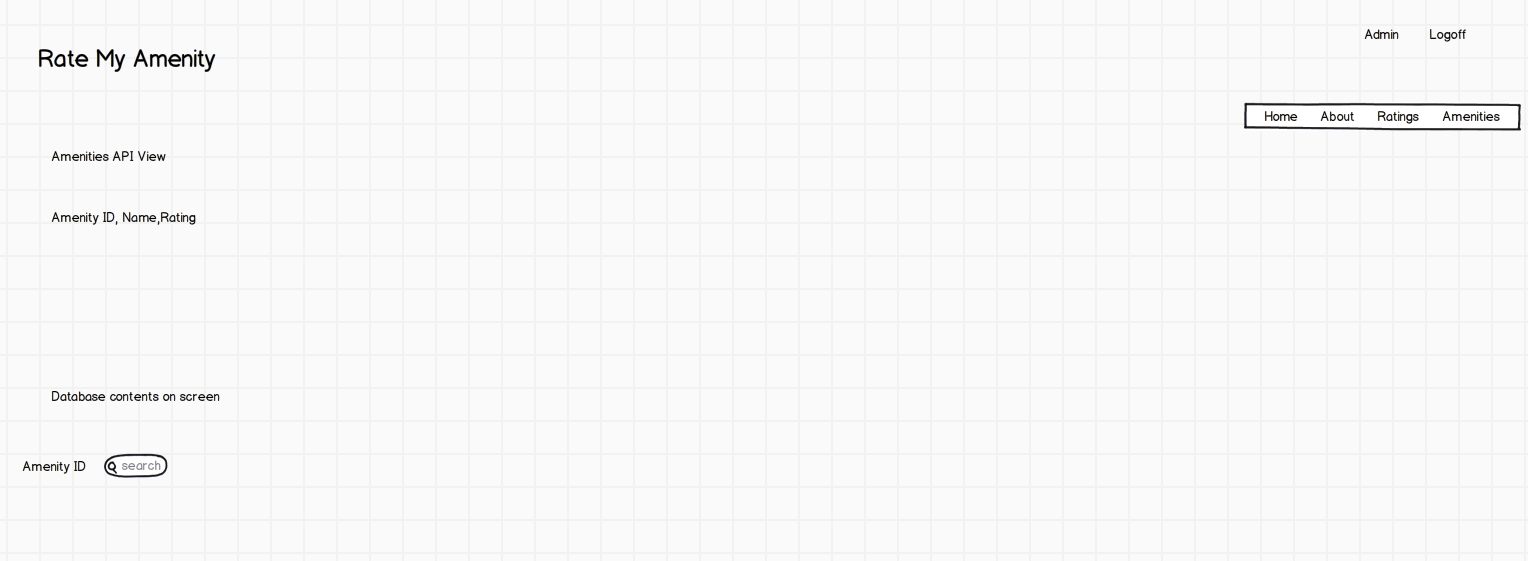
**Ratings**



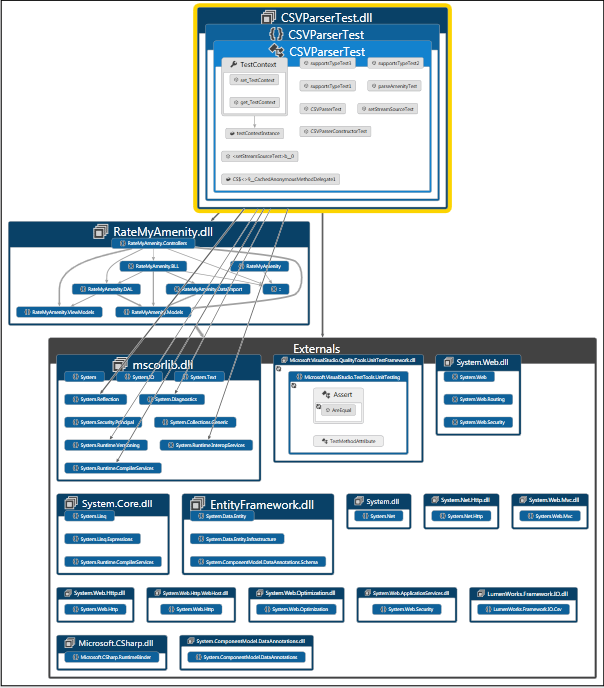
**Amenities**



**Amenity API**



Assembly Dependencies Diagram



Application Profile Report

