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. Background research and investigations 2

2. Project Plan 2

3. Software development methodology employed 3

4. Requirements analysis 3

5. Use cases 3

6. Architecture/Design approach 4

7. Models (Class Models / Data Models etc) 5

8. Data 6

Data Import 6

9. Development Phase 7

To setup construct for import CSV data file 7

Database Migrations 7

10. Implementation of particular OOP constructs 8

11. Design patterns and architectural patterns implemented in the application 8

12. How cross-cutting concerns have been handled 8

13. Security of the application 8

14. Configuration of the application 9

15. Scalability of the application 9

16. Testing Approach (in terms of both functional and non-functional requirements) 9

# Background research and investigations

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, type of amenity and rating. Alternatively users can search within a selected radius. This may be from a selected location or from a user’s current location, if GPS is available on the users device.

**Scope**

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 are able to search for amenities using a range of criteria including location, type of amenity and rating. Alternatively users can search within a selected radius. This may be from a selected location or from a user’s current location, if GPS is available on the users device.

# Project Plan

Due care should be taken to accurately record details of which team member was assigned responsibility for each activity

**HOW DO WE DIVIDE UP THIS?**

# Software development methodology employed

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

It was done using a code first development 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 RateMyAmenityContext.cs

# Requirements analysis

* **Functional Requirements**

1. Call data using an APIfrom Fingal Open Data <http://data.fingal.ie/> and display it.
2. Using search filters display appropriate data to the user.
3. Using a Google Maps API, display the search requirements.
4. Provide an Administrator Login to be stored in the Membership Database
5. Provide a User Login to be stored in the Membership Database
6. Provide a Twitter feed from Fingal Council.

7. Allow logged in users to insert comments.

* **Non Functional Requirements**
  1. Provide a basic user interface.

**5. 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 Google Maps by means of an API

(need to insert diagrams)

# 6. Application Framework

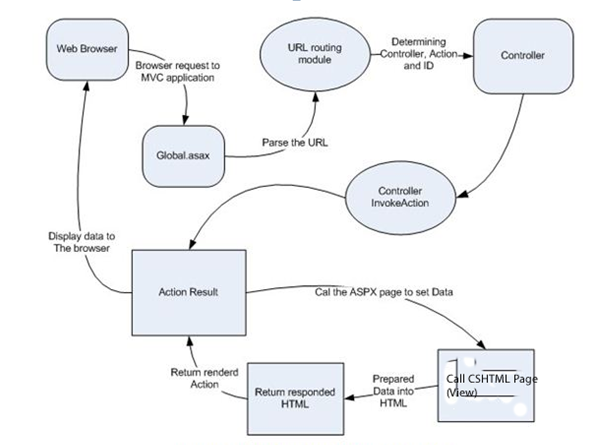


Figure 1. Representaiton ASP.Net MVC Flow

# 7. 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.

Layers

Client

Presentation Layer

The Presentation Layer is responsible for handling the interaction between the user and the application

Business Logic Layer

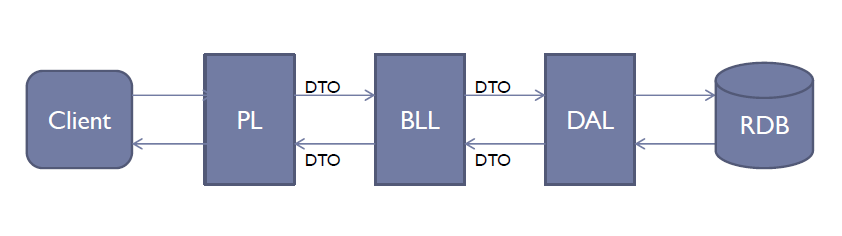
The Business Logic Layer includes the core logic and functional algorithms of the application. It uses Transaction Script which organizes each procedure to handle a single request from the Presentation Layer

The Transactions Scripts are based on the use cases.

Data Access Layer

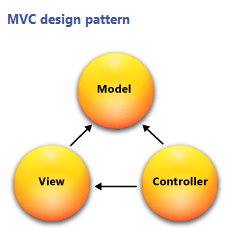
The Data Access Layer (DAL) is responsible for reading Data from and persisting data to the data store. The DAL makes use of the Domain Model which allows it to become a separate layer.

Database



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



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.

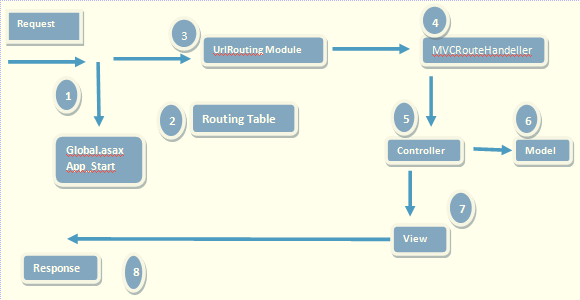


Figure 2 **Strategy pattern** is a OOP Pattern which allows the combining of a group of operations into a small class hierarchy. This pattern was used in conjunction with the **Adapter design.**

**The Adapter Design** is a structural design pattern which allows the application to use classes whose interfaces don’t completely match its requirements.

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

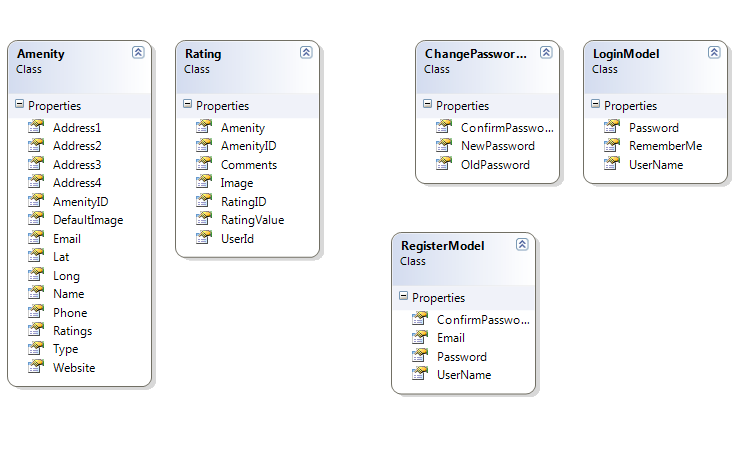
# 8. Models (Class Models / Data Models etc)

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

Models

* Account Model
* Amenity Model
* DataBaseDB
* DBInitialiser
* Ratings

**Classes**



# 9. 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 checked for updates periodically and imported to an internal database. 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

**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.

# 10. 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’

# 11. 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.

# 12. Design patterns and architectural patterns implemented in the application

**Strategy pattern** is a 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 Design** is a structural design pattern which allows the application to use classes whose interfaces don’t completely match its requirements.

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

# 13. 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 if 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.

# 14. Security of the application We need to include or exclude validation

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.
5. Authentication is the process of ascertaining the users identity.
6. Authorization is the process of determining whether the user has the authority to access a specific resource.
7. A user account is a store for persisting information about a particular user.
8. A role is a label assigned to a user and provides a means for defining authorization rules and page level functionality.
9. 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)

Future iterations will have them included

# 15. Configuration of the application

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

<connectionStrings>

<!-- <add name="RateMyAmenityContext" providerName="System.Data.SqlClient" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=aspnet-RateMyAmenity-20120627212345;Integrated Security=SSPI" /> -->

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

<!-- <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 }

# 16. 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

# 17. Testing Approach (in terms of both functional and non-functional requirements)

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.

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

* use of client-side processing,

Need to put in what Jscript we used

* use of Ajax,

Ajax was employed to create a clock on the home page which updated automatically.

* use of web services,
* use of a workflow engine etc.
* use of an ORM tool

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.

* use of dependency injection / IoC containers

References: (complete as Harvard)

Figure 1 - ASP.Net MVC application execution - Krishna G. January 2012, Articles | ASP.NET MVC with C#, C# Corner. Available from:

<http://www.c-sharpcorner.com/UploadFile/krishnasarala/Asp-Net-mvc-basics-part-3/>

Accessed: 16 Jul. 12.