|  |  |  |
| --- | --- | --- |
| Team Canada | 2012 | |
| Group project for Enterprise Frameworks , pgCloud | |  |

Contents

[1. Project Team 1](#_Toc331155891)

[2. Introduction 2](#_Toc331155892)

[3. Background Research and Investigations 2](#_Toc331155893)

[4. Project Plan 3](#_Toc331155894)

[5. Datasets Used in Project 4](#_Toc331155895)

[6. Software Development Methodology Employed 5](#_Toc331155896)

[7. Requirements Analysis 6](#_Toc331155897)

[8. Use Cases 7](#_Toc331155898)

[9. Models 11](#_Toc331155899)

[10. Populating the database with the data 15](#_Toc331155900)

[11. Architecture/Design Approach 16](#_Toc331155901)

[12. Design Patterns and Architectural Patterns Implemented 19](#_Toc331155902)

[13. Security of Application 20](#_Toc331155903)

[14. Configuration of Application 21](#_Toc331155904)

[15. Testing Approach 22](#_Toc331155905)

[16. Conclusion 24](#_Toc331155906)

[17. References 26](#_Toc331155907)

## Project Team

Fintan Costello – X11106441

Darragh Breathnach – X11106417

Alan McCormack – X11102616

Eoin Ó Loideáin – X11109513

## Introduction

The aim application is to allow users to log into a website, and build custom reports based on data pertaining to campaign contributions and votes received during the 2006 and 2010 local elections in Toronto, Canada.

The two main functions of our web application are as follows:

1. Users will be able to filter reports by a number of criteria, and their reports will be outputted to the screen
2. Sample reports will be provided for users

The application was initially to be hosted on Azure; however, this project requirement was subsequently omitted.

## Background Research and Investigations

To start the project the acquisition of a comprehensive dataset was required. We were looking for a good quality open dataset that required a minimum amount of ‘cleaning’. With the growing trend towards open data, one obvious possibility was datasets published by government organisations. Exploration began with the Dublin open datasets published at www.dublinked.ie , however initial investigation revealed that much of the datasets here are very unstructured and the data itself is ‘noisy’ and would require substantial effort to clean up for use as the basis of our web application.

Further investigations led us to a Canadian government website; www.toronto.ca where we acquired several large datasets, containing relatively well formatted data that could be parsed and manipulated with a minimum need for cleaning. Several datasets were explored and the contributions and the election results were finally selected as there was a convenient overlap in the two datasets that would allow us to join the two datasets together.

W3schools and the .net website were utilised to get a basic understanding of C# and MVC projects. Sample projects were created here touching on a number of relevant are as to the application. The class tutorial for ingesting the CSV was completed and then modified for use in the team Canada context.

## Project Plan

The plan was to create a web application that would allow easy access to the cleaned up datasets to allow for customised reports, and to also provide a basic data analysis tool.

As there was a tight deadline and a steep learning curve, the project was immediately split into tasks and assigned to team members as seen on Github.

Github shows the contributions of each team member over the past few months. Regular updating took place with contributions from all team members in each area of code, research and report writing.

## Datasets Used in Project

The datasets used were acquired from www.toronto.ca/open and consisted of four files in Excel format that detailed campaign contributions and results for:

* Mayoral Race 2006, 2010
* Council Elections, 2006, 2010

The files ranged in size from 1MB to 2.3MB and required some cleaning up before they were suitable for use in the project. Each line of data needed to be associated with a ward number and then the four files amalgamated down to two files, one for contributions and one for election results. This work was carried out in MS Excel using vlookups and also required some cleaning and formatting of unstructured data.

Toronto is divided into 44 electoral wards. Each ward is has 42 subdivisions, numbered 1-34 and then 93-99.

*Mayoral Election 2006 / 2010*

* Each candidate runs in all 44 city wards and an individual vote count for each ward and each subdivision are given. Totals for each subdivision and an overall total are given
* Campaign contributions are not listed with a ward number as the candidate runs in all wards
* Campaign contributors are listed by postcode, amount donated, contribution type (i.e. cash) and candidate donated to

*Council Elections 2006 / 2010*

* Each candidate runs in 1 ward and an individual vote count for that ward and each subdivision are given. Totals for each subdivision and an overall total for that ward are given
* Campaign contributors are listed by postcode, ward number, amount donated, contribution type (i.e. cash) and candidate donated to

## Software Development Methodology Employed

Test-driven development is an agile development technique used by Team Canada. It relies on a simple, iterative process that ensures that all the code in the finished project has been tested and it instils confidence that all the code in the completed application should behave as it was intended it should behave.



*Fig.1 Test-driven development cycle (source: Wikipedia)*

The TDD process is illustrated in Fig.1 above and adheres to the following simple procedure:

1. Write a failing test to prove that a functionality or code is missing from the completed code.
2. Write code or correct it to allow the test to pass
3. When it passes, the code can be cleaned up or refactored to increase readability and to avoid duplication.
4. Start process again and write a test for the next feature requirement.

## Requirements Analysis

**Functional Requirements**

The following functions will be required of the completed application. There are two main categories, starting with the Administrative functions. This will entail reading in of the data, which will be an irregular event.

* Parse data from CSV files
* Populate a database / datasource with data

User functions will include:

* Filter results/contributions by year or election
* User report selection
* Generate reports
* Analyse data

**Non-Functional Requirements**

The application’s non functional requirements will include:

* Web application is to be programmed in C#, and built on ASP.NET MVC Architecture
* The web application must be fully tested
* Web application is to be programmed in C#, and built on ASP.NET MVC Architecture
* Have enough capacity for our datasets, with room for expansion
* To be able to process our data efficiently
* To be responsive
* Easy to use
* Deployed to Azure

## Use Cases

By defining use cases for our MVC application we translated our functional requirements into a concise guide for generating tests for initiating our test-driven development process. They act to identify and to clarify our requirements. By creating specific use cases for our requirements, it makes them easier to read and to track. The purpose is not to design a UI; it is to literally describe how our application should work.

For use case #1 – “Generate a Report”, the typical report building sequence will be:

1. Select a specific election (i.e. 2010 city council election)
2. Filter by one of the following:
   1. Candidate name
   2. Contributor name
   3. Post Code or Ward Name
   4. Amount of Donation
   5. Contribution type (cash, goods/services, etc.)
3. Create report and display on screen

When filtering by contributor, the following conditions will apply:

* If the contributor is an individual, no postal address will be returned.
* If the contributor is an organisation, a postal address will be returned.

Use case #1 is summed up in table 1 below.

|  |  |
| --- | --- |
| **Use Case #1** | |
| **Name:** | Generate a Report |
| **Actors:** | User and Web Application |
| **Summary:** | The user can build a report based on the election and contributions dataset. The user can Filter by one of the following:   * 1. Candidate name   2. Contributor name   3. Post Code or Ward Name   4. Amount of Donation   5. Contribution type (personal, services, etc.)   Reports can be displayed on screen |
| **Pre-conditions:** | Data parsed and loaded into in the system & user selects a report |
| **Flow of events:** | 1. Presentation layer serves form for user to make report selection 2. Report is built 3. Report is displayed on screen |
| **Error conditions:** | No data in DB to create a report. Or there are empty/null fields due to unclean data that was missed by our testing. |
| **Post-conditions:** | Potential to export report to PDF / CSV |

*Table 1. – Use case #1 - Generate a Report*

Does this need to be removed now?

For use case #2 – Data analysis will be performed using the “Least Squares Regression method”, the total number of votes and contributions per candidate will be analyzed to determine approximately how much in contributions it takes to achieve a given number of votes. The analysis can also be reversed to determine the number of votes to expect for a given level of contributions. We will allow the user to analyze and compare data from the two different year’s results to ascertain what the difference was and to help make predictions on the next election results.

|  |  |
| --- | --- |
| Total No. Votes |  |

Total Contributions ($)  
  
*Chart 1. – Analysis of Total Votes vs. Total Contributions, for Toronto Mayoral Elections 2006*

Chart 1., is a scatter plot depicting how many votes each candidate achieved along with the corresponding total value of contributions received by that candidate. By using the least squares regression analysis, a ‘best fit’ line can be plotted through our given data points. It allows us to predict what number of votes can be expected if a certain level of contributions is achieved and vice versa.

The Least Squares Regression is based on the following model:

|  |  |  |  |
| --- | --- | --- | --- |
| slope = m | = | |  | | --- | | n(Description: http://people.hofstra.edu/stefan_waner/calctopic1/SYMB/SIG.GIFxy) - (Description: http://people.hofstra.edu/stefan_waner/calctopic1/SYMB/SIG.GIFx)(Description: http://people.hofstra.edu/stefan_waner/calctopic1/SYMB/SIG.GIFy) Description: http://people.hofstra.edu/stefan_waner/calctopic1/SYMB/FR.GIF n(Description: http://people.hofstra.edu/stefan_waner/calctopic1/SYMB/SIG.GIFx2) - (Description: http://people.hofstra.edu/stefan_waner/calctopic1/SYMB/SIG.GIFx)2 | |
|  |  |  |

And;

|  |  |  |  |
| --- | --- | --- | --- |
| intercept = b | = | |  | | --- | | Description: http://people.hofstra.edu/stefan_waner/calctopic1/SYMB/SIG.GIFy - m(Description: http://people.hofstra.edu/stefan_waner/calctopic1/SYMB/SIG.GIFx) Description: http://people.hofstra.edu/stefan_waner/calctopic1/SYMB/FR.GIF n | |

Therefore for any given x-coordinate a corresponding ‘best fit’ y-coordinate can be calculated using the simple formula:

y = mx + b

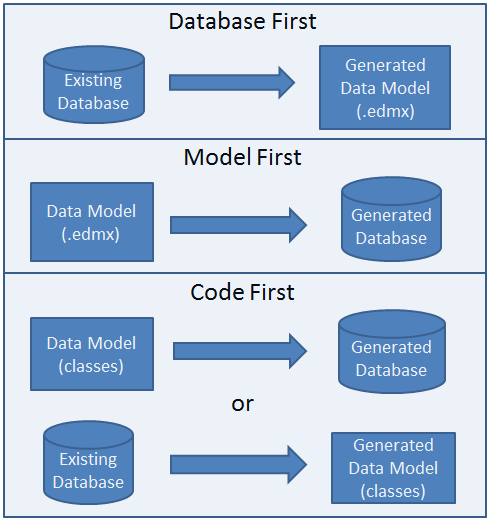
|  |  |
| --- | --- |
| **Use Case #2** | |
| **Name:** | Data Analysis |
| **Actors:** | User and Web Application |
| **Summary:** | The user can select from two years election results and run an analysis to show the relationship between Contributions and Votes. Like use case# 1, reports will be displayed on screen. |
| **Pre-conditions:** | Data parsed and loaded into in the system & user selects a report |
| **Flow of events:** | 1. Presentation layer serves form for user to make report selection 2. User chooses which dataset is to be analyzed and submits 3. The report is displayed on screen |
| **Error conditions:** | No data in DB to create a report. Or there are empty/null fields due to unclean data that was missed by our testing. |
| **Post-conditions:** | Potential to export report to PDF / CSV |

*Table 2. – Use case #2 – Analyze Election Data*

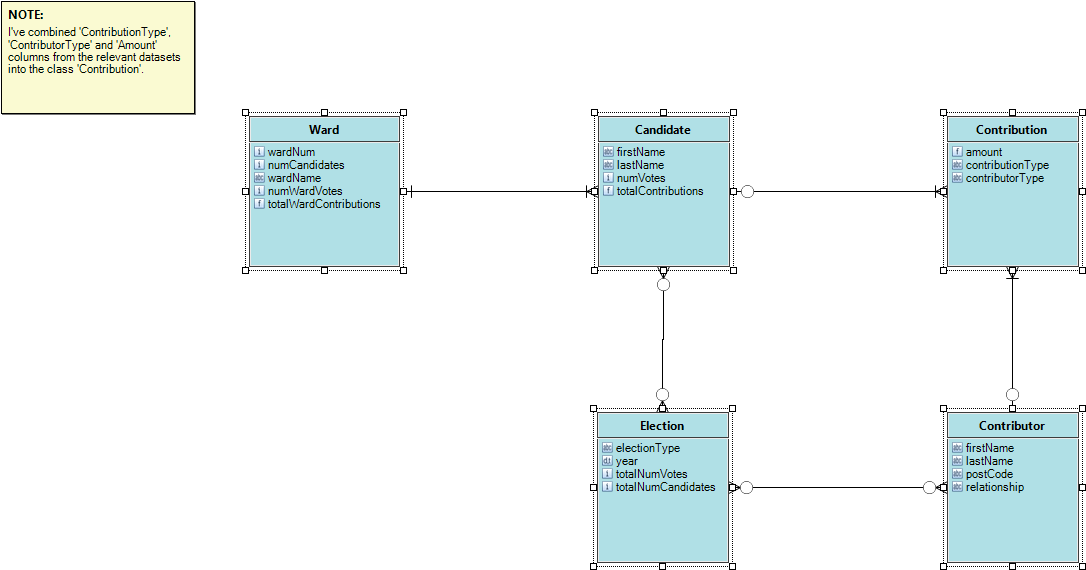
## Models

**Domain/data Model**

There are three possible approaches to for dealing with data in the entity framework.

*****Fig.2 – Approaches to building Data Model*

The “model first” approach was initially chosen for this project, with the data model being generated, using a modelling tool called Rise. By plotting the different relationships between the various attributes of our dataset, the Rise tool generated a data model for the application, which could be imported into Visual Studio.

*Fig3. Initial Data Model, generated using Rise tool*

While Rise was found to be a useful tool for initial exploration of possible data model structures this approach was later discarded in favour a code-first approach, where models are used to create the database and tables. The data models created consists of:

**Contribution Model (Contributions.cs)**

* ELECTIONTYPE
* ELECTIONYEAR
* CONTRIBUTORLASTNAME
* CONTRIBUTORFIRSTNAME
* CONTRIBUTORADDRESS
* CONTRIBUTORPOSTCODE
* WARDNUM
* AMOUNT
* CONTRIBUTIONTYPEDESC
* CONTRIBUTORTYPEDESC
* CANDIDATEFIRSTNAME
* CANDIDATELASTNAME

**Results Model (Results.cs)**

* ELECTIONTYPE
* ELECTIONYEAR
* WARDNUM
* NUMVOTES
* CANDIDATELASTNAME
* CANDIDATEFIRSTNAME

**View Model (ReportsUI.cs)**

A view model was created to handle generated reports. The view model, which is strongly typed, provides you with a convenient way of representing complex view data. It allows you to to pass data from a controller to a view.

## Populating the database with the data

There was a requirement for the data contained in the CSV files to be read into the database and stored in the correct tables. This data is currently clean, although prolific. Some data normalisation will take place to integrate postal code areas and wards. This is necessary to eliminate further problems down the line.

A third-party parser was required to read the CSV files into the database. The LumenWorks CSV parser reference was added to the project for this purpose.

Code snippet:

public class LoadCSV

{

public AllDataParser all\_data = new DataParse();

// results

public List<Results> loadResults()

{

List<Results> lst = new List<Results>();

StreamReader reader = new StreamReader("C:\\EF Project\\EF-Group-Project\\datasets\\final\_csv\\all\_results.csv", true);

all\_data.setStreamSource(reader);

lst = all\_data.parseResults();

return lst;

}

As there was a requirement to load multiple CSVs into the project, the parser was amended to allow for this scenario. Inserting the data and saving changes to the database was handled by a class in the DAL layer.

/\* Start CSV parsing \*/

public void insertcsv()

{

List<Results> results = null;

List<Contributions> contributions = null;

LoadCSV import = new LoadCSV();

results = import.loadResults();

foreach (Results r in results)

{

db.ElectionResults.Add(r);

}

contributions = import.loadContributions();

foreach (Contributions r in contributions)

{

db.ElectionContributions.Add(r);

}

db.SaveChanges();

}

}

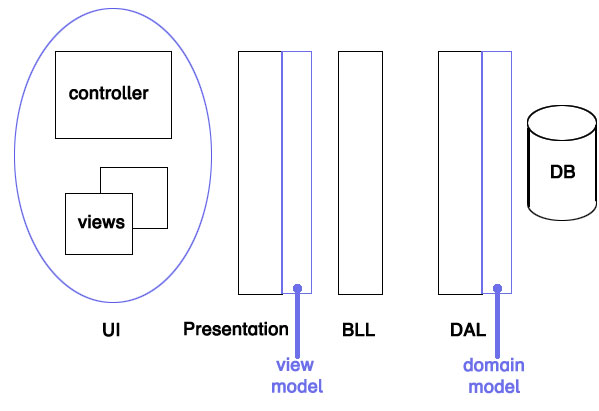
To prevent the CSVs being parsed and inserted every time the project was run, a check was added to determine if the tables were empty first. If there was no data in the tables, then insertcsv was run; otherwise no parsing took place.

## Architecture/Design Approach

The architectural approach taken from a design perspective was a Layered Architectural style (Fig.4). By grouping related functionality within the application allows for strong separation of concerns. This approach lends the completed application many benefits including, greater reusability, for instance our parser can be used to parse several different files with minimum modification. A standalone parser need not be created for each separate file. Another major benefit is the greater flexibility, testability and easier maintenance of the finished application. For example the business logic calculations carried out within the Business Logic Layer, can be tested in isolation from the DAL and from the UI.

*****Fig4. Layered Architecture System*

Our goal of a Layered Architecture was achieved initially by application of the basic Model-View-Controller (MVC) design pattern. With the MVC pattern the UI is represented by the view, the model represents our data, with the controller handling requests and manipulating the model. As development progressed the MVC pattern was replaced with the Model-View ViewModel (MVVM) pattern. In this particular web application there are in fact two types of models; view models and data models. MVVM originated with Microsoft and is actually based on MVC, with the controller having been replaced by the model view. The view model converts data into view information and passes commands from the view into the model (Fig.5).



Best practice dictates that the view model is kept simple with the heavier data manipulation carried out in the domain model and passed to the business logic layer (BLL) for performing calculations before being passed to the view model.

Calculations for the two use cases are carried out in the BLL. For the reports in use case #1 these entail:

* summing up the contributions & calculate the largest total contribution
* summing up the contributions & calculate the smallest total contribution
* summing up the contributions & calculate the average total contribution
* sort candidates returned, alphabetically
* sort candidates returned, from largest to smallest
* sum up the contributions & sort
* sum up the contributions & sort by ward number
* calculate average per ward

Similar calculations will be performed for use case #2. The typical flow of events for use case #2 can be summarised as:

1. The view model will need to pass the query to the DAL,
2. The DAL will query the database to return all the votes received by all the candidates, and all the contributions received by all the candidates,
3. The DAL will pass the response to the BLL for processing, where it will sum up all the votes per candidate and associate this list of values with the sum of all the contributions per candidate.
4. A calculation will them be carried out as per our least squares model above and the result will be returned to the user via the presentation layer.

## Design Patterns and Architectural Patterns Implemented

**Domain Model**

No design pattern was adhered to in a rigid fashion for developing this web application.

From an architectural pattern perspective a domain model pattern was followed.

*“An object model of the domain that incorporates both behaviour and data”* (Fowler 2003)

It consists of business entities, the relationships with each other, and how data flows between as detailed in section 9.

The main benefit of using the domain model is that in the case of where business logic is subject to a lot of change, the domain model decreases the “cost” of those changes. Objects are ignorant of any behaviour relating to persistence of information to the database. Business Logic can also be reused in different applications.

Using this pattern requires more upfront coding costs, time and effort, but the intention is that these initial costs will pay off in the long term as an application increases in complexity.

## Security of Application

Our application contains the basic account registration and login features of an MVC4 application. We haven’t expanded on this functionality at this stage. Due to the nature of the data contained, and the fact that it is open data, greater security isn’t required.

## Configuration of Application

Adhering to the “convention over configuration” design paradigm and building the web application using .Net naming and coding conventions there was little or no configuration required apart from manual configuration to facilitate Repository tests pattern.

The enormous benefit of not having to worry about configuration issues comes at a relatively small cost of some limits to flexibility.

## Testing Approach

At an early stage of development, a test project was created along with the main team Canada project to ensure our focus would remain on the testing element.

As coding progressed, tests were created, modified and performed on the code.

Using sample tests created in class, we’ve tested the integrity of our CSV parsing and using a sample repository patter, we’ve enhanced the application testing.

ASP.NET MVC provides a substantial set of in-built testing tools that provide a framework for writing unit tests. Two broad sets of tests were written, one for the CSV parser that ingested our raw data and one for the business logic layer. Testing was carried out on all associated classes and methods.

For the parser tests the following cases were tested:

1. Test for file format type, i.e. CSV
2. Test for file not being a CSV type
3. Test the actual parsing of the “Contribution” file, (edited file for testing)
4. Test the parsing of the “Results” file, (edited file for testing)

**Repository Pattern**

For the purpose of testing our database access, we’ve implemented the repository pattern in relation to our “contributions” entity.

To create the tests, a mock repository was created, IContributionRepository, which is a class that is used to save the Contributions in a database. We also added an interface for the Contribution repository. We created another class called the EntityContributionRepositoryManager, which will implement the IContributionRepository interface for the repository object.

We then created a mock repository in the test project and began implementing unit testing. We created a second “HomeController” file, (in keeping with the tutorial we had) and renamed it as so. We created an associated test HomeController, “HomeController2Test”, and in here created our tests.

We tested adding data to the repository and checking to see if it was valid. We passed some dummy data into the repository and checked to see if was created.

We checked if the correct amount of records were added. We also created a test to fail by checking to see if a third record was added, when we hadn’t added it.

All tests performed as expected.

## Conclusion

Lessons learnt & shit

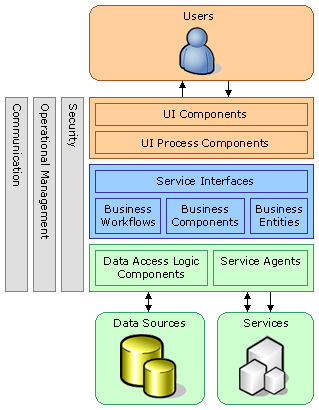
**Migration to Azure**

In order to deploy the application to Azure, the following steps are required:

* Open the shortcut menu on Teamcanda project.
* Choose “Add windows azure deployment project.
* Open the shortcut menu again on main web project
* Choose “Publish to Windows Azure”
* Extra steps are required to create the database.

Future development of this project would involve some of the following:

* Migration to Azure – to take advantage of the scalability afforded by Azure
* Security – following on from a migration to Azure and having a live web application security will need to be looked at
* Cross-cutting concerns – see diagram below security, communication etc



## References

* Testing approach:   
  <http://artofunittesting.com/storage/chapters/SampleChapter1.htm>
* Use Case 2:  
   <http://people.hofstra.edu/stefan_waner/calctopic1/regression.html>)
* MVC 4 Tutorials  
  [**http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application**](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application)

<http://msdn.microsoft.com/en-us/library/ee658117.aspx>