**ASP.Net Web Dev. 2 Coursework**

THSurveys: Survey System

**System Design**

**Table of Contents**

[Application Setup 3](#_Toc341553703)

[Database setup 3](#_Toc341553704)

[Projects and Components 3](#_Toc341553705)

[Packages 5](#_Toc341553706)

[Application Structure and Logic 6](#_Toc341553707)

[Overview 6](#_Toc341553708)

[Menu 6](#_Toc341553709)

[Styling 7](#_Toc341553710)

[Screen / Page Layouts 7](#_Toc341553711)

[Validation Rules (Business Logic) 7](#_Toc341553712)

[Use of the MVC Framework 9](#_Toc341553713)

[Overview 9](#_Toc341553714)

[Components 9](#_Toc341553715)

[Application Component Details 10](#_Toc341553716)

[IoC 10](#_Toc341553717)

[Controllers and Actions 10](#_Toc341553718)

[Routing 10](#_Toc341553719)

[View Models 11](#_Toc341553720)

[Mapping Classes 11](#_Toc341553721)

[Model and Binding 11](#_Toc341553722)

[Filters 12](#_Toc341553723)

[Validation 12](#_Toc341553724)

[Cross cutting Concerns 14](#_Toc341553725)

[Security: 14](#_Toc341553726)

[Exception Handling 15](#_Toc341553727)

[Code Modules and Patterns 15](#_Toc341553728)

[Unit Testing 17](#_Toc341553729)

[Future Development 19](#_Toc341553730)

[Templates for Survey Questions and Responses 19](#_Toc341553731)

[Acknowledgements 20](#_Toc341553732)

[Use of the MVC Charting tool 20](#_Toc341553733)

# Application Setup

## Database setup

The system uses Entity Framework (EF), Code First to create the Domain model. When the model is accessed for the first time, through the THSurveys DbContext, EF creates the underlying database and populates it with sample data and template data.

An instance of LocalDB is used as the database. This is a recent addition and functions like a genuine SQL database, but doesn’t need an instance of SQL Server (Express) to manage it, as it runs as a process. It can be a bit unstable as it’s a very new product.

## Projects and Components

The architecture of the application is designed around the “Onion model”, which is a layered design but with the business model at the core and other layers communicating directly with the core. The other layers in the application are the UI layer, which is the MVC project, and the infrastructure project which has the concrete implementations of the Data context and access to them.

The model does not allow communication or reference between componenets at the same layer, but only towards the core.

This can cause a problem for the location of the IoC container, which needs to reference both Core and Infrastructure layers. This is the single exception and is set up as a project that is allowed access to both layers. Having this as a separate layer, removes the need for any reference across layers in the UI layer.

### Core Project

This project contains the business model, interfaces, factories and Services used by the application.

|  |  |
| --- | --- |
| Folder | Contents |
| Model Folder | The business model is contained within the Model folder. It has a class for each object in the model, some of which are abstract to facilitate the use of abstract factories to create concrete instances of them. Such classes will also have a concrete version within this folder. |
| Interfaces Folder | This folder contains the interfaces for the various repositories and the UnitOfWork for the data context |
| Factories Folder | This folder contains the factories used to create concrete instances of various objects, used within the UI layer. |
| Services Folder | This contains the Survey Analysis service, which will calculate the various statistics based upon a supplied survey. It is used in the Analyse pages |

The Unit Of Work exists to ensure that, in conjunction with Ninject, the same instance of the data context is used by all the repositories, avoiding exceptions related to the lifetime of the data context.

### Infrastructure Project

|  |  |
| --- | --- |
| Data Folder | This folder contains the Entity Framework data context, which uses Code First based upon the classes in the Core/Model. |
| Migrations Folder | This should contain the migrations for the Code First database generation. However, problems experienced early in the project meant that this feature doesn’t currently work properly. The database schema has settled for this application so making the Migrations feature of Entity Framework function correctly is left for future development. |
| Repositories Folder | This folder contains the concrete repositories used by the application UI controllers. |

### IoC Project

This project exists to allow the Ninject Dependency Resolver bind the interfaces and abstract classes of the Core project with the concrete implementations contained in the Infrastructure project.

This is required as these binding cannot be achieved in the UI project, where the Dependency Resolver is created, since to reference the Infrastructure project from the UI project causes a circular reference.

### Models and Documents Project

This project contains the class and Use case diagrams for the application. It also contains the documentation, of which this document is one.

### THSurveys Project

This is the UI project, which is the MVC application project. It was created from the standard MVC4 Internet Application template and contains the standard folders.

Additional folders added to the project, contain the components which allow the application to functions.

|  |  |
| --- | --- |
| Filters Folder | This folder contains the Action Filters used by the application. Most of them map various classes to other viewModel classes and vice versa. There is an additional AjaxActionOnly filter. |
| Helpers | Within this folder is the only Html helper method. |
| Infrastructure | This is subdivided into the following: |
| Factories | This contains the factory for creating the MVCChart used to display survey results. |
| Interfaces | Contains interfaces for the mapping classes used by the Take Survey view. It allows them to be injected into the controller and facilitates better unit testing of the controller action. |
| ModelBinders | This contains the only custom model binder used in the application. It handles the binding for the Take Survey viewmodel and the **Home** controller **Take** action method. |
| Mappings Folder | This contains the mapping classes used for converting the returning View model classes to the corresponding business model classes. |
| Models Folder | This is the standard models folder for an MVC project but it has split into separate folders to contain view models that relate to the correspondingly names controller. |

All models within these folders, are considered to be view models and each view has an associated view model. The restructuring of the folders happened after the coding had started, so some of the namespaces of the view models do not necessarily correspond to the folder they are contained in.

It is a future development refactoring task to correct this situation, but it doesn’t hinder the running of the application itself.

### ThSurveys.Tests Project

This project as the name implies, contains the unit tests for a sample of test for as many of the components used within the application.

The tests included are by no means exhaustive, but hopefully are indicative of possible ways to unit test the individual components.

The tests for each of the components are held in the subfolders of the project.

|  |  |
| --- | --- |
| Folder | Tests Contained |
| Controllers | Test on the Home controller. |
| Filters | Test on the AjanActionOnly filter |
| HtmlHelpers | Test of the RadioButtonListHelper |
| Mappings | Test on the MapTakeSurveyViewModel mappers of the THSurveys project |
| Model | Contains the mock data for the unit tests. |
| ModelBinders | Test on the TakeSurveyModelBinder |
| Routes | Test on the Custom Routing tables for incoming and outgoing Url’s. It also contains the original RouteTable, as this was developed outside the application towards the end of development and then plugged in when it passed all tests. Akin to Test Driven Development. |

## Packages

The following packages are used within the project solution to support the application

|  |  |
| --- | --- |
| **Package** | **Function** |
| Ninject | Dependency Injection |
| AutoMapper | Mapping between business classes and ViewModel classes |
| Glimpse | Installed but not used much to verify routes being taken. |
| MVCChart | Charting tool supplied as part of .Net framework, since .Net 3.5. |
| MSTest | Unit Testing framework, part of VS2012 |
| Moq | Mocking framework for unit testing, object creation. |
| Fakes | MS tool, fake objects and allow Static or Extension method to be ‘mocked’. |

# Application Structure and Logic

## Overview

The application is structured to support the requirements, with as simple a presentation layer as possible.

The design will take account of the URL to represent the features identified in the requirements

The application will be split into the main sections

**Participate in a Survey**

1. View a list of available surveys and select one
2. Take a survey:

**Creation of Surveys**

1. Create a Survey
2. Add questions to a survey
3. View a list of surveys and their status
4. Publish a Survey (submit for Approval)

**Publish Survey**

1. View list of surveys pending approval
2. Approve a Survey

**Review Survey Results**

1. List of published surveys and how many respondents have taken the survey
2. View the summary of responses to a survey
3. View the results of surveys summarised by the personal information

The application is secured so that Creation of surveys and Review Survey Results can only be access by registered users of the application. The surveys will be restricted to those belonging to the registered user.

A system Administrator will be solely responsible for the Publish Survey functionality.

The Participation in a Survey is unrestricted and available to anyone.

NB:

There will be no method implemented to ensure that a respondent doesn’t take a particular survey more than once. Such would be considered under Future Development.

## Menu

An application menu is provided as a series of links which gives access to the parts of the application directly accessible by users.

### Menu options

Menu Item Description

Home Show the Top 5 Surveys

Participation

Top 5 Surveys Shows the Top 5 Surveys

Choose by Category Allows surveys for each category to be listed

Surveys

List My Surveys List surveys for the user who is logged in.

Create Allows a registered user to create a new survey

Administration Allows the system administrator(s) to approve surveys for release

About General information: disclaimer and acknowledgements.

## Styling

Styling of the application making use of CSS is not a primary concern of this project in terms of the coursework requirements. However, a simple standard layout is applied across all pages of the application.

The styles are achieved using CSS. The styles for the menu are based on freely available css templates, specifically <http://cssmenumaker.com/>.

## Screen / Page Layouts

The following pages are included within the application

Home page

The home page of the application, it contains a list of the top 10 surveys.

For Each page show

Layout image

Describe the purpose

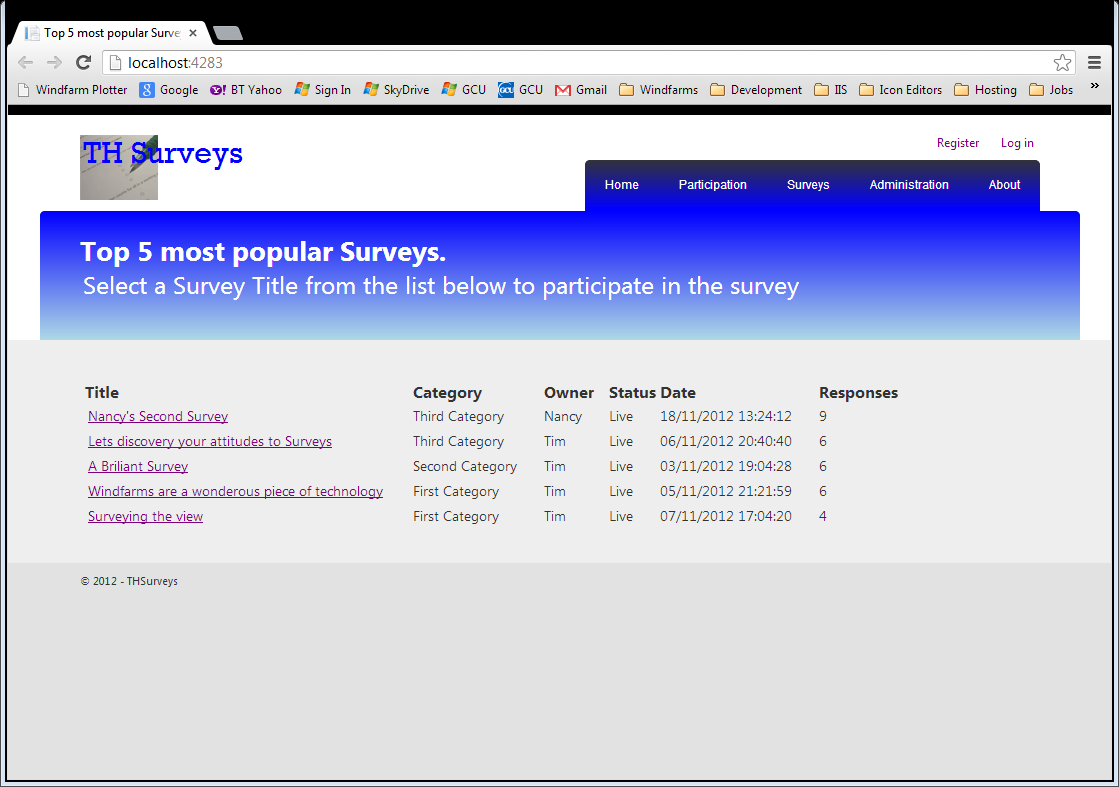
Describe the links

Describe the functionality.

Describe the validation where appropriate.

### Home Page

This lists the top 5 surveys in terms of the number of respondents.

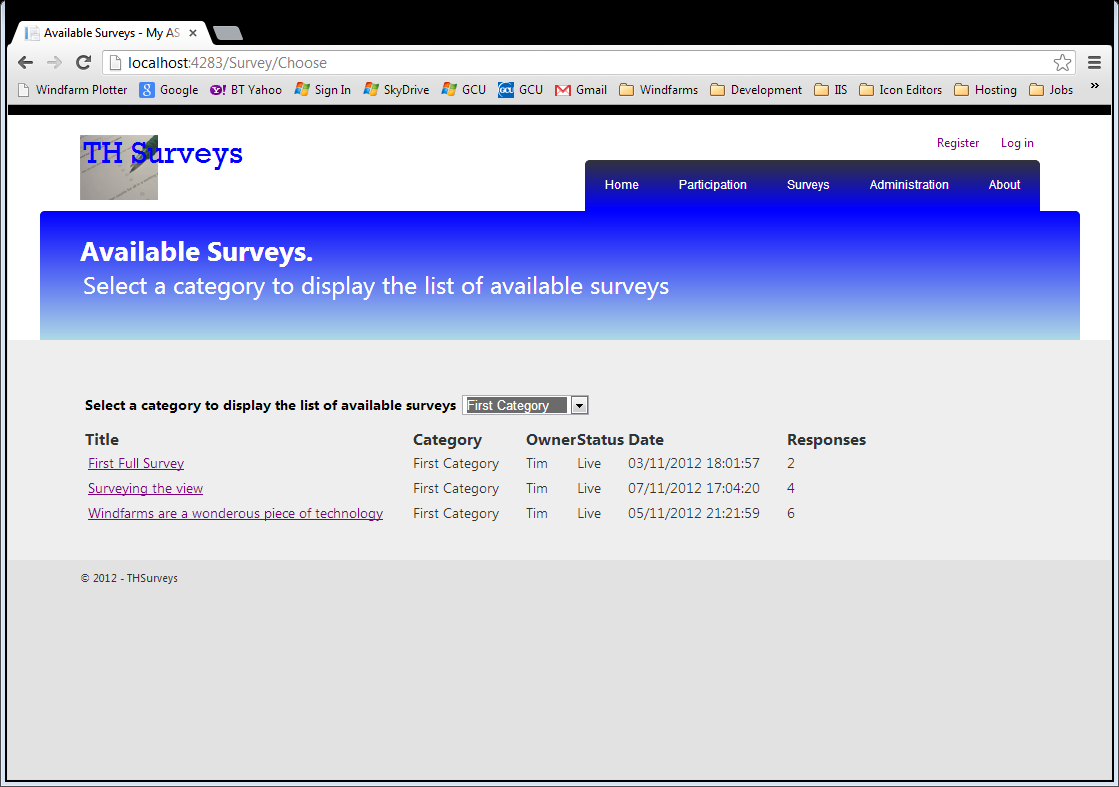


This is the default page, but is also accessible from the **Top 5 Surveys** option on the Participate menu.

The links for the title of each survey take the user to the Take Survey page.

### Choose a Survey by Category

This shows a list of surveys that can be taken by a respondent. It is access from the **Choose** option from the **Participate** menu.



###### Functionality

Selecting a category from the drop down box will redisplay the surveys for that category. There is an option to select all categories.

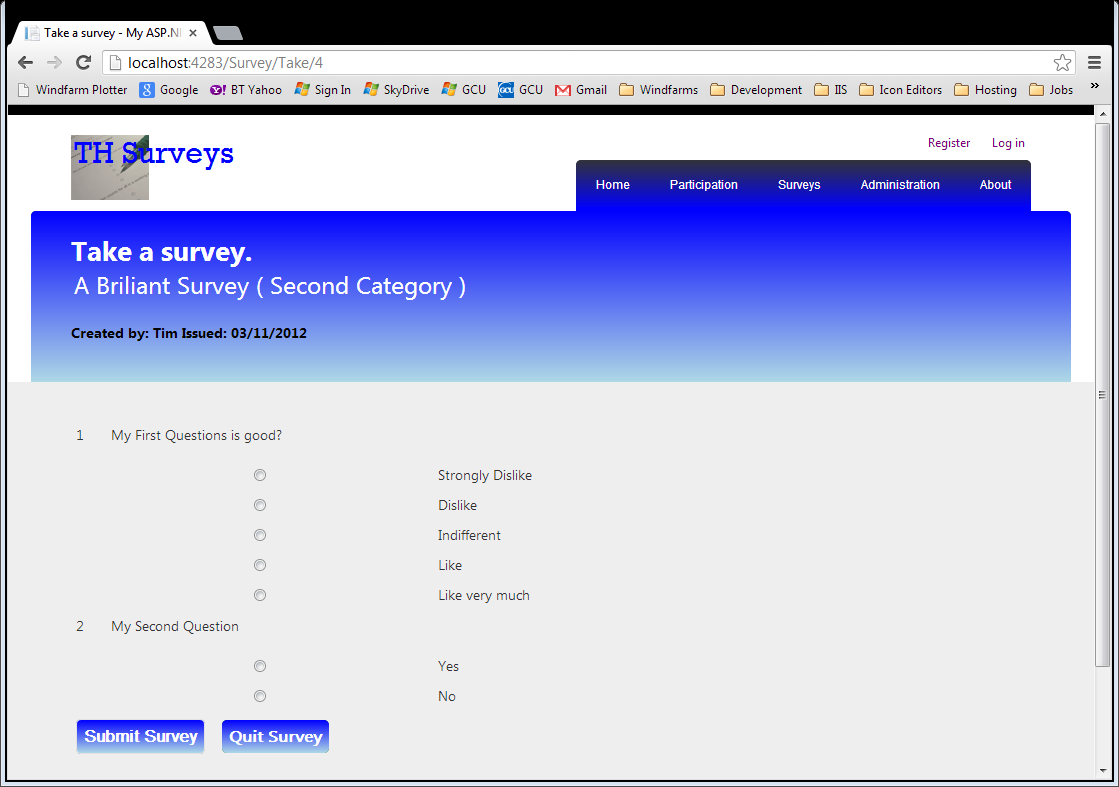
It makes use of the \_SurveysList partial view to display the survey information. The call to repopulate the list when the category is changed is performed

###### Components used

|  |  |
| --- | --- |
| Home Controller, Choose | Initialised the viewmodel and returns the view. |
| Home Controller | SurveyList which returns partial view, and responds to an ajax call so that the only the partial view is update, improving performance. |
| HomeChooseviewModel | The view model for the view |
|  |  |

### Take a Survey

This is the page used to participate in a survey. The desired response to each question is indicated by select the appropriate radio button.



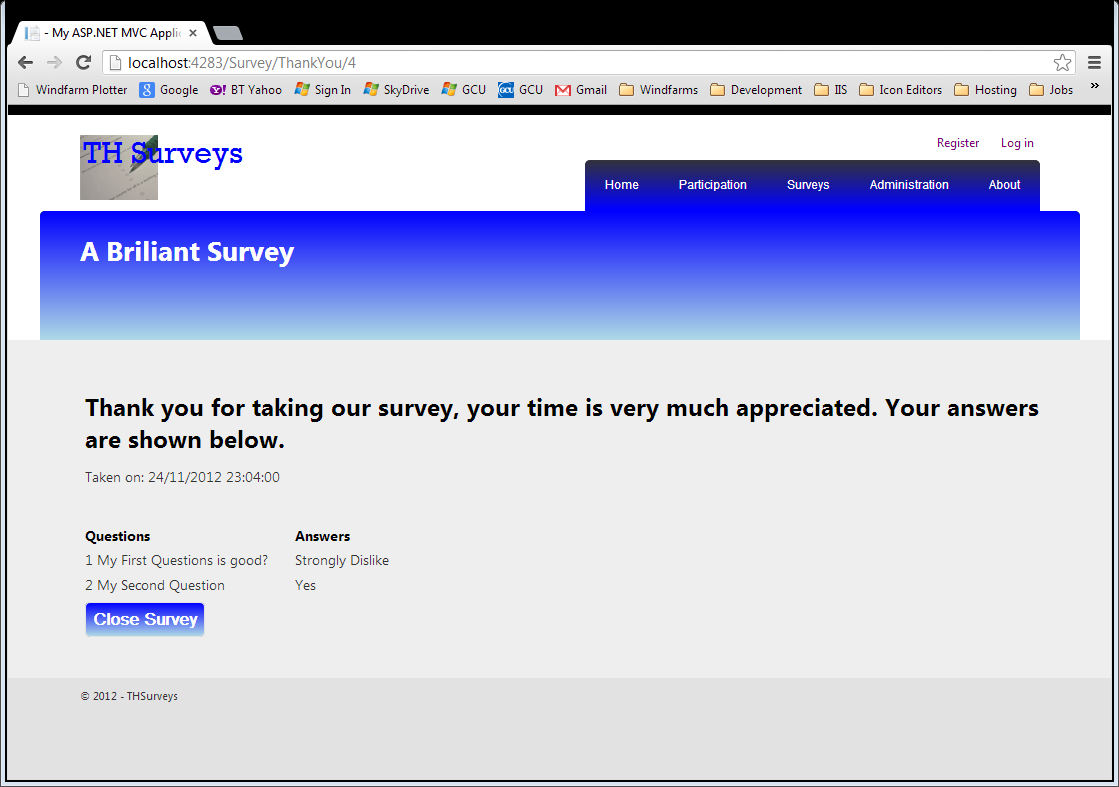
Validation is shown by messages at the top of the page for any question that remain unanswered when the Submit Survey button is clicked.

Components used:

Future Dev: add paging to this survey so that many questions can be included OR to display a single question or group of related questions and progress to the next part of the survey, showing a progress indicator across the top of the page showing how far through the survey the respondent is.

Take Survey Completed page

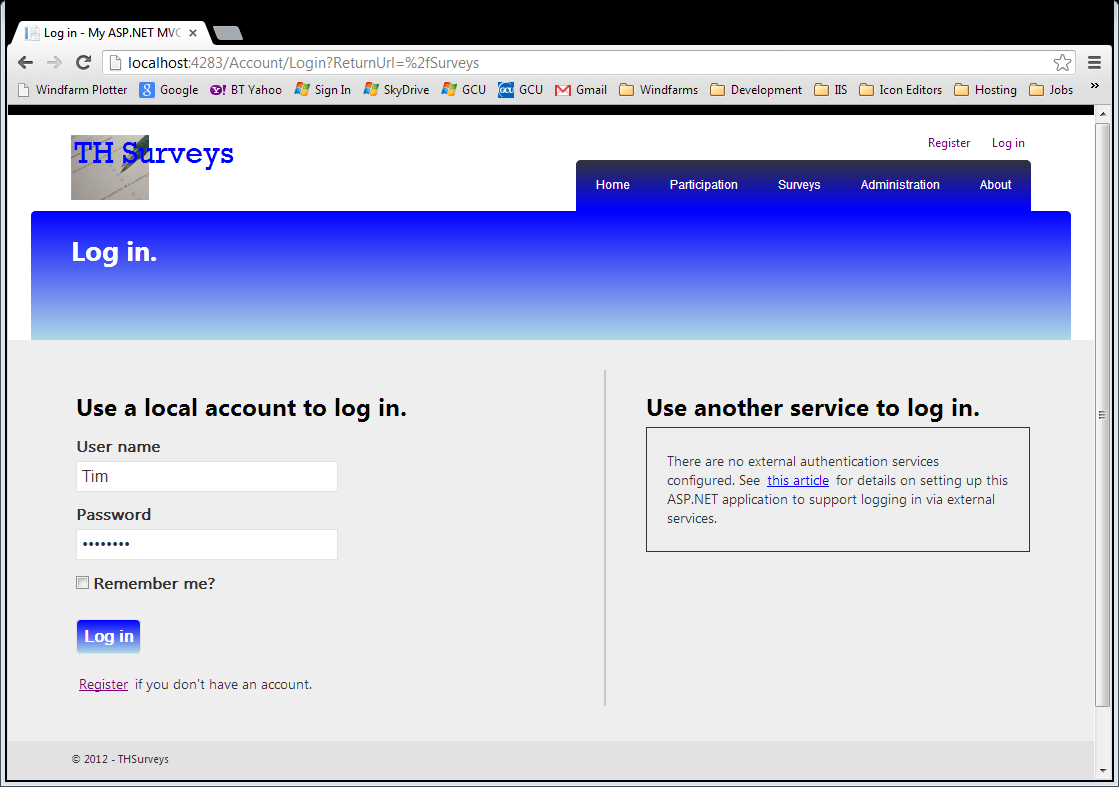
This is the response page shown when the survey is successfully completed by the respondent.



Close takes you back to the Home page.

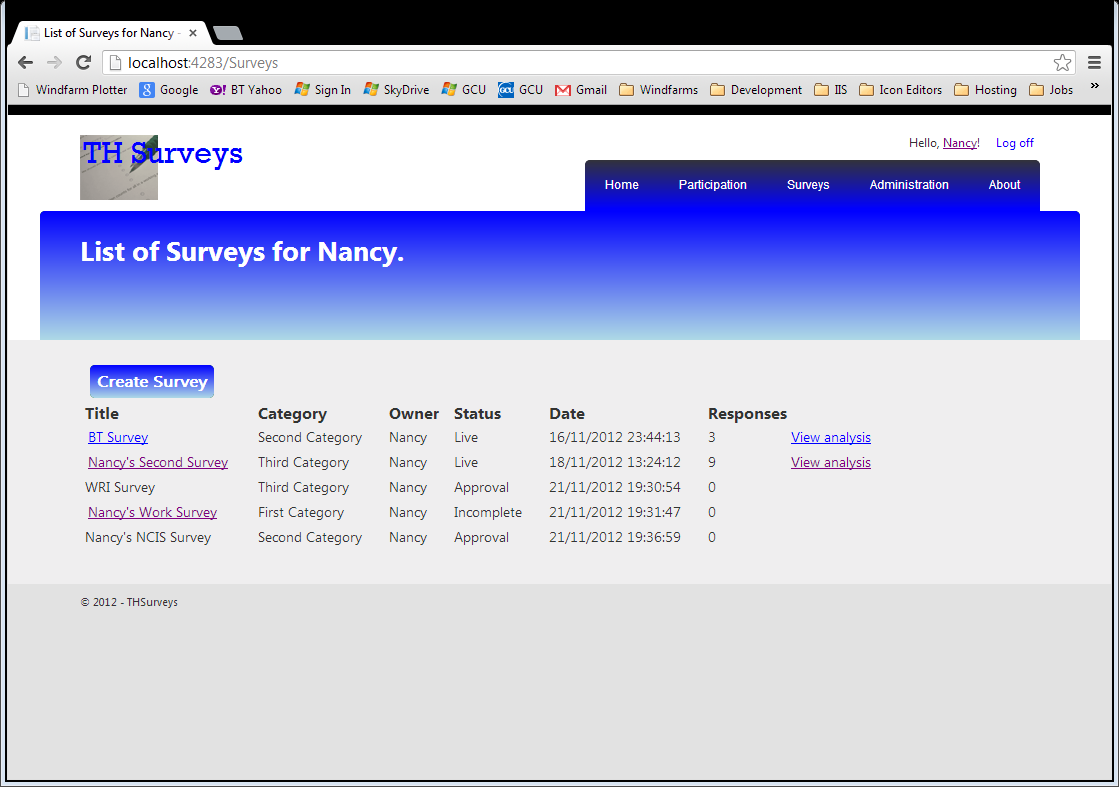
Login form

This is the standard MVC provided form, displayed automatically when the controller and / or action methods are secured. It has been modified slightly to take the styling of the application.



My Surveys Page

This page shows all surveys for the logged on user. It indicates the status of the survey and provides links to view the results or add question to incomplete surveys. It also provides a route to creating a survey, separate from the menu option.



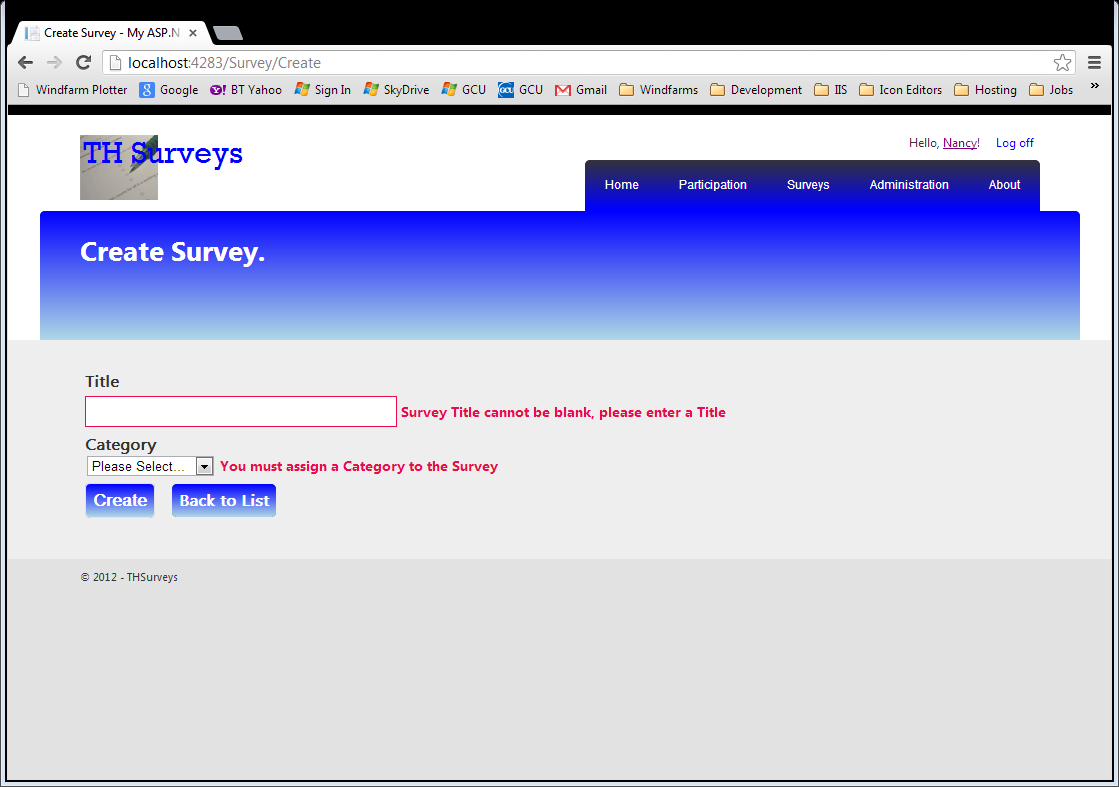
This is the main page for each registered user.

Progress from here is Create Survey, or Add Questions by clicking the Title link for a survey with a status of **incomplete**.

If a survey displays responses, with a status of **Live**, the clicking the Title Link or the ViewAnalysis link moves the the Analysis page for the survey.

Create Survey page

This is the page where surveys are created, but the logged on user.



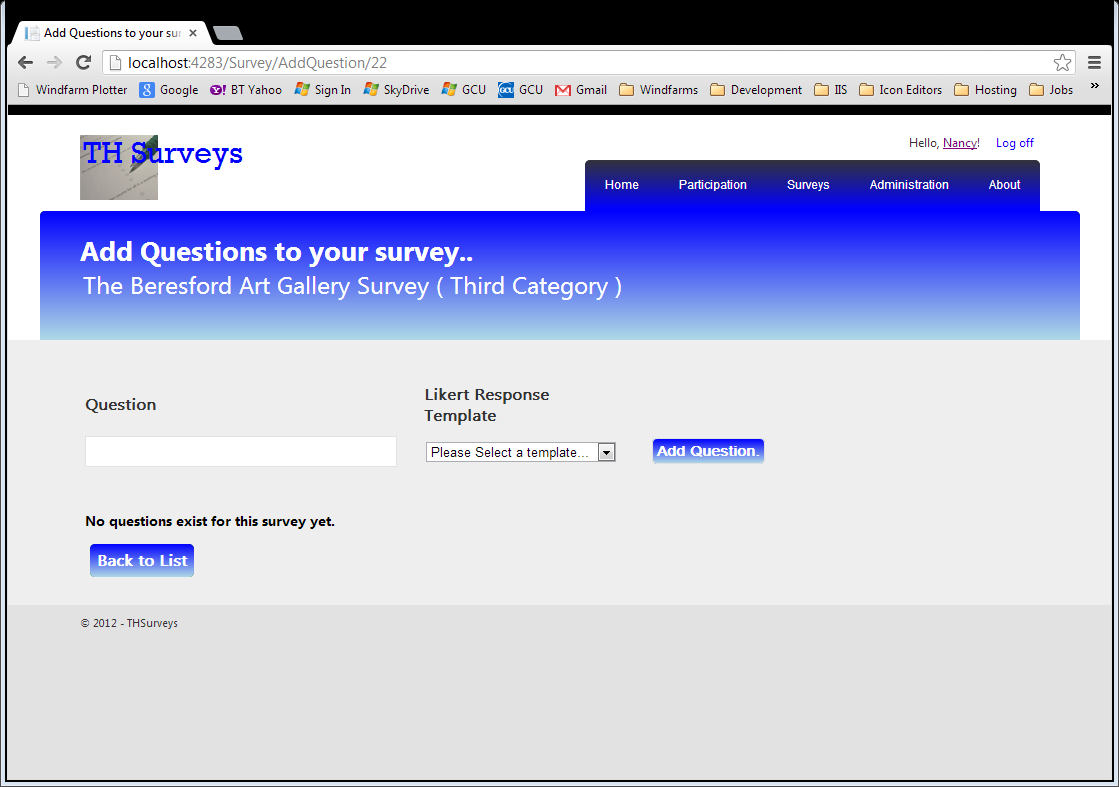
This shows the validation, complete the title and select a category from the drop down list. Click Create to proceed to Add Questions to the survey.

Clicking **Back to list** returns to My Surveys.

Validation is happing on both the client and the server.

Add Questions to Survey

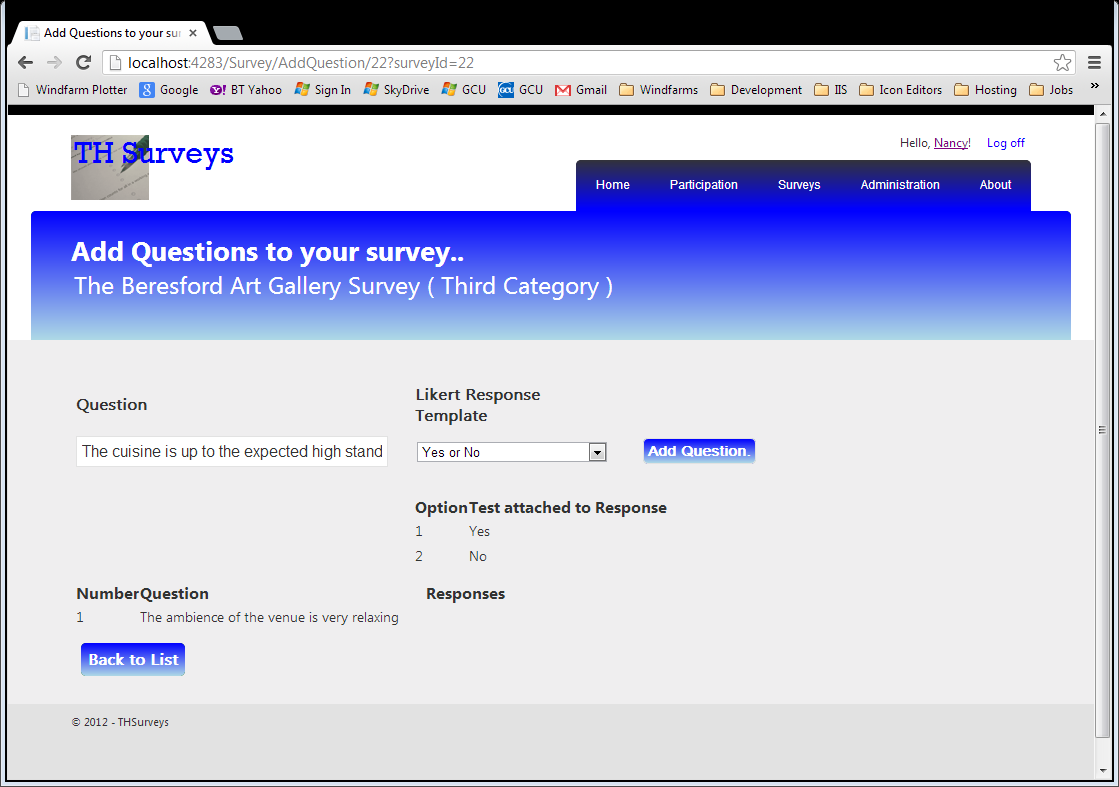
This is the main page for setting up the survey. It allows questions to be added, shows the questions as they are set up. Allows the possible responses to be selected from a template and allows the survey to be submitted for approval by the Administrators before going Live.



This shows the initialised state, when questions exist they are shown where the “No questions…” message is shown. A button to submit the survey for approval also appears, when there are questions but nothing in the input fields. This stops the survey being submitted for approval if a question is currently being added.

Add question, adds the question to the survey, and returns to this page. The question shows along with any other questions at the bottom of the page.

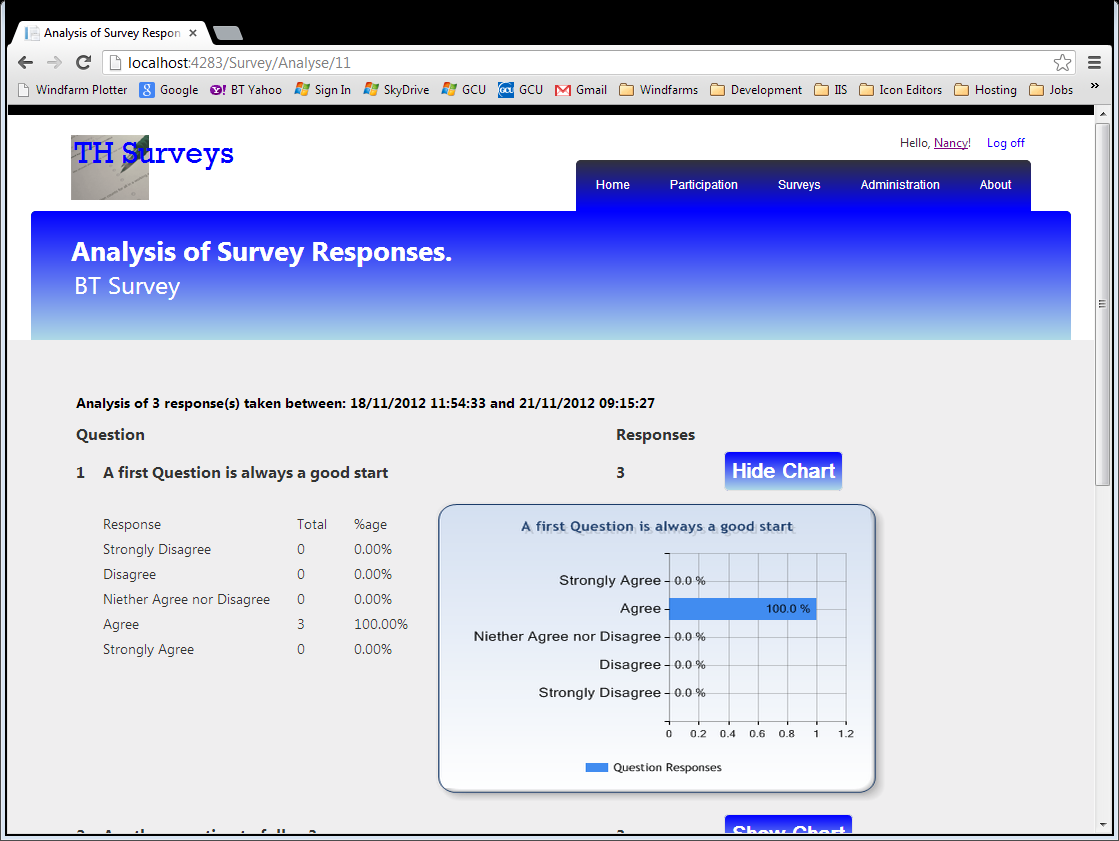
When selecting a response template from the drop down, and ajax call to the QuestionController GetLikertResponses action method retrieves the possible answers and displays them below the dropdown. As shown below:



The submit for approval is an ajax driven controlled by the use of jquery within the page. Controlling the display of the Submit button is handled with jQuery in the page. The submit process displays a dialog to confirm. Confirmation calls the SurveyController Approve action method to change the status of the survey. Upon return a result dialog box is displayed after which the user is returned to the list of surveys.

View Results page

This is the page where the analysis of the responses to a survey can be viewed. The analysis is simple for each question in the survey. The view below shows a partial results page and includes one of the question analysis shown as a chart.



Clicking the **Show chart** button, launches launches some jQuery, which adds an <img> to the view and sets src attribute to the AnalysisController GetSurveyChart method. This method creates a chart using the MVC Chart tool, and returns it as a file in a FileResult. The <img> displays the chart as it would any other image.

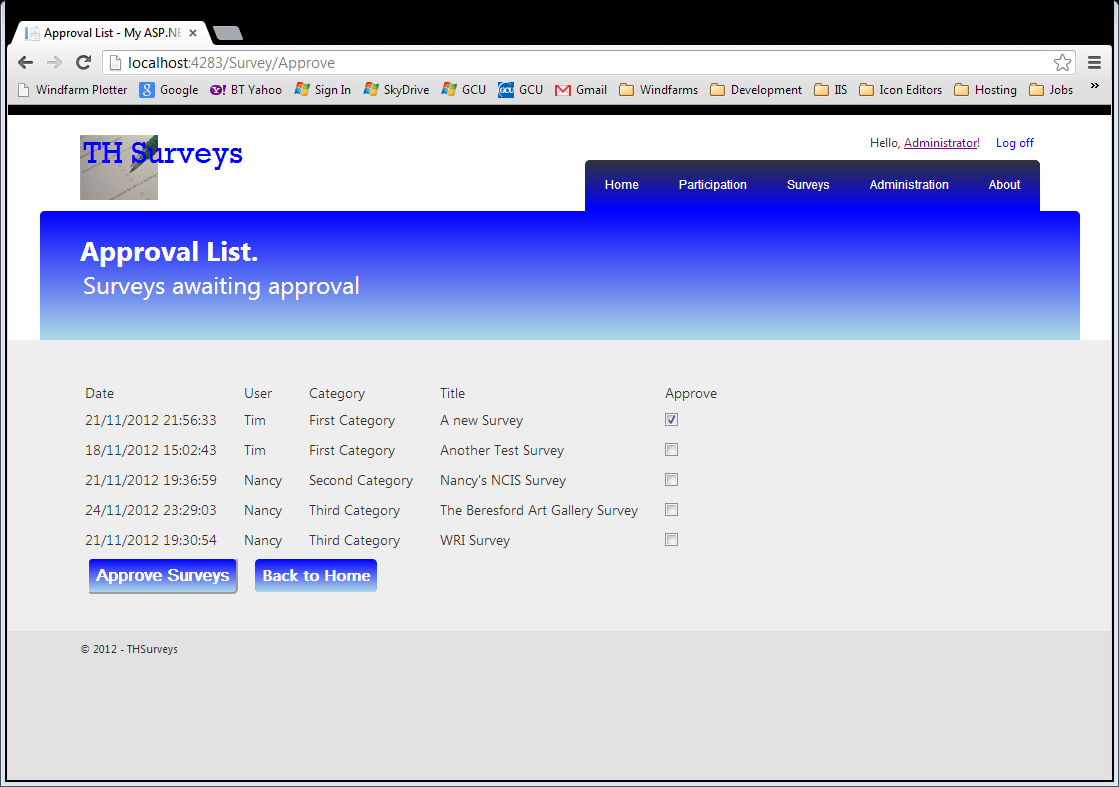
Clicking the **Hide Chart** button removes the chart image from the view.

Clicking **Back to Surveys** returns to the list of surveys for the user.

Components used:

Approve surveys Page

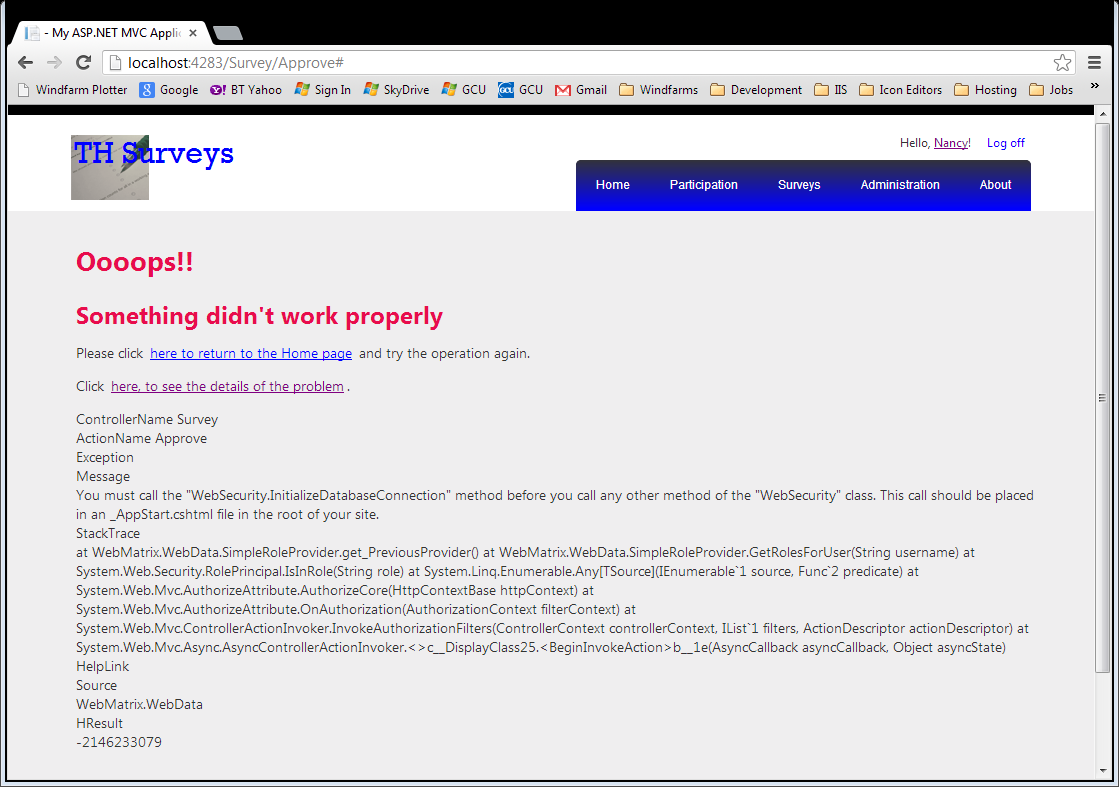
This page lists the surveys awaiting approval. The Administrator selects a survey to approve by checking the checkbox next to the survey.



This form uses a form action of POST to call the SurveyController Approve method which processes the input and returns to this page.

The ViewModel has been carefully constructed to allow the default model binder to be used.

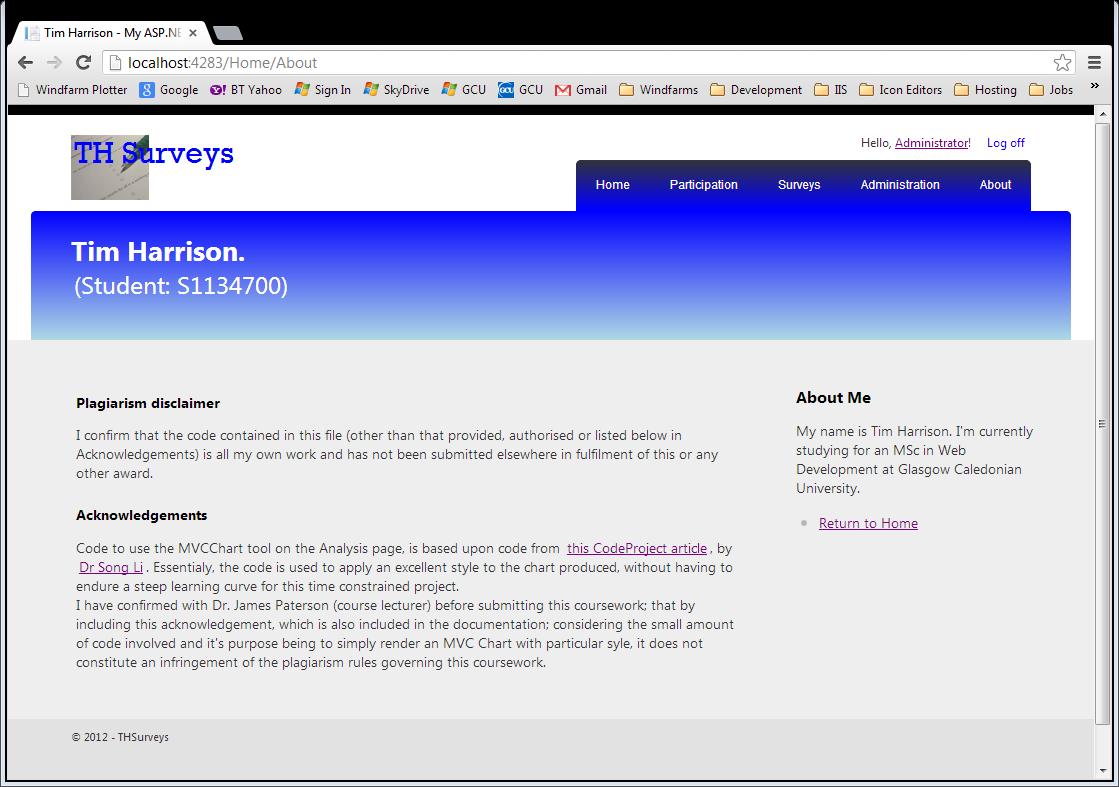
An Error page



This demonstrates the security problem mentioned in the Exception handler section. Here, the Administration page was selected from the menu, but with the app running for some time, the membership and authentication seems to have stopped. It causes the “Invalid Operation Exception” with the reason stated in the sample. Clicking the link to “return to the home page” will force a logoff (using jQuery to check the form to determine if the user is still logged on, as the link to the membership seems not to be working) and return the user to the home page. This seems to clear the problem with the membership database and allows the app to continue .

About Page

This page shows the acknowledgements and disclaimer required by the coursework.



## Validation Rules (Business Logic)

1. Sequence Number of a question should be unique within each survey. Its intention is to allow the order of questions to be altered.
   1. Applies to Question.SequenceNumber
2. Sequence number of a response on the Likert Scale should be unique. This is so that the Likert Scale can be preserved.
   1. Applies to AvailableResponse.LikerScaleNumber and similarly on the LikertResponse template.

# Use of the MVC Framework

## Overview

Various parts of the system will be coded to utilise the ASP.Net framework, but in a way that extends and customises the behaviour where it benefits the system.

The main project makes use of the standard Internet template application provided by VS2012. Benefits of this are the automatic inclusion of the Simple Membership provider which can be built into the systems database much easier than before.

The areas of the MVC framework considered are described in the following sections. Not all areas are appropriate to diverge from the default behaviour, but the considered reasons are discussed for each of them

## Components

Some components in the application design have been selected to demonstrate differing ways of using the MVC framework to achieve the desired results.

Where appropriate, use is made of jQuery to keep the view as responsive as possible. This is in the form of the validation, provided as part of the framework, in addition to call the various methods of the controllers asynchronously so that the UI remains responsive.

This is used primarily on the Choose Surveys by Category, the Add questions to a survey to show the responses for a template and the existing questions of the survey as well as the Submit for Approval process. It is also used within the Survey analysis view to retrieve the chart.

# Application Component Details

## IoC

Ninject is the package used to provide Dependency injection for the application. In MVC4 Ninject instantiates a custom version of the DepencencyResolver. This is automatically done by locating the code in the **App\_Start** folder. It creates a **Bootstrapper**, which is then responsible for launching the application.

Some components, to map them successfully, use components in both the Core and Infrastructure projects. The Ninject DependencyResolver is located within the UI layer, specifically the THSurveys MVC project. This must not, indeed cannot, contain a reference to the Infrastructure project as it would cause a circular reference. Therefore such component bindings are located within a separate project which references both Core and Infrastructure. This approach removes the need for the UI project to reference the Infrastructure.

However, additional component bindings required completely within the UI layer are added to the egisterRoutes within the UI project.

Thus the Ninject Dependency Resolver contains all bindings necessary for the application.

## Controllers and Actions

The controllers will be split to reflect the responsibility of each, but also taking account of the security aspect.

Home: unsecured, Listing surveys and allowing participation

Survey: Secured, User maintenance of their surveys, submission for approval, approval of surveys.

Question: Secured, responsible for adding questions to a survey

Analysis: Secured, responsible for providing analysis of the responses to a survey.

Account: mixed, responsible for account registration, logon and logoff.

These will be coded to reflect the structure of the application so that they can be mapped to the appropriate route.

They will make use of attributes, dependency injection for the reference to the repositories for the business layer.

## Routing

To keep the Url’s for the system as simple, short and as guessable as possible, the application supports url’s of the structure Survey/doSomething/anOptionalId. This means a user has the option of typing any of the following:

/Survey/Top5

/Survey/Choose

/Survey/Take/id of survey

/Surveys

/Survey/List

/Survey/Create

/Survey/Approve

/Survey/AddQuestion/id of survey

/Survey/Analyse/id of survey

A set of custom routes for the application achieves this.

Other controller action methods are secured so that typing them directly in the browser address bar will result in a PageNotFound error. The actual additional routes that work for the application do not necessarily follow the above pattern, and should therefore not be guessable.

## View Models

The decision to use ViewModels for each view with the application was taken so that no business model class should be referenced within them. This promotes good separation of concerns between the business model in the Core project, and the UI itself.

This also allows the views to reflect the required functionality and not be tied to the structure of the data they operate with. This has the effect of abstracting the business model from the UI, and therefore reduces the visibility of the data structure to the user.

## Mapping Classes

Various classes and ViewModels are mapped between each other within the application. A series of mapping classes are provided to do this mapping.

To keep a consistent approach throughout the application, and attempt to keep the code in the controller action methods as light as possible, following the Single Responsibility principle, mapping tasks are abstracted out of the controller action methods.

For controllers that respond to HttpGet requests, the mapping is generally performed using the AutoMapper package, with the code being placed within an ActionFilterAttribute. These Action filters are placed within the UI project under the Filters folder.

The HttpPost action methods do not use AutoMapper, instead use specific mapping classes, as they often need to use repositories to retrieve data to complete the mapping process. It is not a good idea to perform this type of processing within an Action filter as it can cause problems with the data context, such as the context being disposed by the time the filter requires to access it. These mapping classes are held within the UI Mappings folder.

## Model and Binding

The model binding within the application has been performed using the both the default model binder and a custom model binder. This is to demonstrate both techniques but also it is not always appropriate or necessary to use custom model binding as the default model binder can cope with many situations.

Specific examples are for the Take Survey view, to bind the TakeSurvey view to the TakeSurveyViewModel on the HttpPost, which is done with a custom model binder.

The Approve Survey, which shows a list of surveys available for approval and the allows the Administrator to check those to be approved uses a complex object structured so that the default model binder can perform the binding.

## Filters

Custom Action filters are used for mapping the business models to the view models for specific views. where possible, to simplilfy the coding in the controllers. Overriding the OnActionExecuted(). This allows standard and common mappings to be completed by adding an attribute to the relevant controller.

Use of Built-In Filters:

The built in filters used within the application are

**ValidateAntiForgeryToken**: Ensures the anti-forgery token is checked each time.

**OutputCache**: This can be used to improve the performance of the application but should only be used for static data or data that is not particularly volatile, but has not been implemented yet within this application.

**ChildActionOnly**: This marks an action method for execution as a child method and is used to restrict the use of action methods, and prohibit users from accessing them from the browser address bar.

Custom Filter

A custom action filter, that overrides the OnActionExecuting method, has been written to provide similar security as the ChildActionOnly attribute, but for methods that are the target for Ajax only calls from the client layer. This is called the AjaxActionOnlyAttribute. It validates the requrest to be an Ajax request, and returns a Http 404 request if the request is not an Ajax request.

Error handling Filters

The standard HandleError attribute is used to provide the error handling for the application. This is in conjunction with the configuration within the web.config file to set the default 404, 401 and 402 errors.

## Validation

To keep the application responsive, the client side validation as well as the server side validation is used. The client side validation makes use of the data annotation attributes. There is not a huge amount of validation required in this application.

Use is made of the data annotation attributes to apply validation rules to the view models as well as the displayable names for each item on the view.

Validation using these attributes also support the validation at the server side, especially should the browser have javascript disabled, which is how the client side validation works.

Additionally, the Business models will have the validation encoded in the same way, as these are the business rules. They may differ from the specific requirements of the UI layer.

Client validation is switched on in the web.config file, Add the following to the appsettings sections

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

# Cross cutting Concerns

## Security:

### Protection and Robustness against attack

All secured action methods should do the necessary checks to ensure the url is not entered from the address bar in an attempt to circumvent the security.

Defaults: ensure that default values cannot be set outside the binding, via the Url. Eg Demo code (IsApproved). This would apply to the status of the Survey (XSRF)?

* Wk 4 – Over eager binding – [Bind(Exclude=”fieldname”)] – sloide #10.

The types of common attack fall into the following

XSS Cross Site Scripting

The MVC Razor engine automatically guards against this type of attack by ensuring that all html content is encoded by default. Use of the @Html.Raw helper will disable this encoding, if required. This application makes no use of the @Html.Raw helper to protect against XSS type attacks.

XSRF Cross Site Request Forgery

MVC 4 framework provides an @Html.AntiForgeryToken helper and a ValidateAntiForgeryTokenAttribute, designed to protect against this type of attack. It works by generating a random token, which is placed within the form using the helper. The attribute, which marks a POST action method, validates this token and passes the request to the method if the token is valid. This protects because sites are not able to get at the token placed within the form.

This only works on the POST request, so to help protect against this type of attack , HttpGet requests should never update anything. This application follows this principle and also implements the AntiForgeryToken on the form posts.

Cookie Stealing

This is where external javascript can read cookies to gain information and access sensitive information. This can be disabled within the Web.Config file, by setting the

### Membership, authentication and Authorisation

Registrations of Https

* Registration should be over Https, as should all registered activities as the user authentication token is in cluded in the Request. How exactly is this done?

Custom Authentication Attributes

* The application will NOT code custom authentication attributes. They are difficult and require vast amounts of unit testing. Security coding is a huge task in its own right and many developers have spent eons developing security related code. The MVC framework comes with built-in security including authentication and membership routines. Given the short timescales for developing this application, it would not make sense to attempt to code custom security.
* The default membership and roles included within the framework will be sufficient for the application.
* As a rule, you should never consider custom security engines, save to enhance the default with some specific requirements. DO NOT RE-INVENT THE WHEEL.

## Exception Handling

The standard error handler mechanism is used. Activated from within the web.config file, but adding

<customErrors mode="On" defaultRedirect="~/Error.cshtml">

<error redirect="~/ErrorNotAuthorised" statusCode="401" />

<error redirect="~/PageNotFound" statusCode="404" />

</customErrors>

to the system.web section.

The specific pages are stored in the view/shared folder. Each error page is designed to allow the error information to be viewed and also a link to return to the Home page of the application.

Depending on whether a user is signed on or not, the route to the home page might be by logging off the application.

This approach is adopted as some errors occurred in debugging, where the Razor engine is checking the User.Identity contained within the request, but the connection to the web security has broken and the invalidOperation exception occurs. This is clear by logging off the application.

## Code Modules and Patterns

### Unit of Work Pattern

Entity Framework (EF), when performing updates to the data context can experience problems when multiple objects are updated within the same **SaveChanges** operation. When information is retrieved using EF, it sets up a change tracker so that it can determine what changes are made and how to apply them to the database. If the source of the updates comes from different read operations, multiple instances of the change tracker are created. The update will then fail because EF cannot determine the changes correctly across all change tracker instances. To avoid this, each read operation must use the same instance of the DB context so that a single change tracker is used.

The Unit Of Work pattern is adopted, in conjunction with Ninject to control the lifetime of the Entity Framework Data Context. All repositories access the DB context through this Unit of Work, which is injected into all repositories. The instance of the DB context is maintained by the IoC container, which ensures the same instance is used for the lifetime of the HttpRequest.

### Inversion of Control (DI)

Use of the Ninject Dependency Injection package is made, specifically the MVC4 version, is used to implement Dependency Injection for the application. Ninject follows the MVC4 implementation by providing the container as a custom Dependency Resolver.

It is used to inject concrete instances of class in the Infrastructure project, allowing the UI to reference the interfaces of abstract classes defined in the Core Project. This maintains the separation of concern between the three main projects.

### Factories

The use of Factories and Abstract factories is made to remove the creation of dependent classes from the main code and avoid using the **new** keyword.

Most were set up as static classes originally which work perfectly, but this did not help the unit testing aspect, so were changed to abstract factories to make it possible to mock the creation of the classes within the unit tests.

Not all factories were changed to abstract ones. Future development would be to refactor this so that all factories used are abstract ones.

Code Layers

* Code is to be modularised, so that MVC project contains only UI related components.
* Code for Domain models and Data Access will not, however, be created in separate projects, but will be included in the Business Layer project.

# Unit Testing

A major emphasis of this project is to demonstrate the testability of the various components and customisations made within the MVC framework.

The testing in this project however, is not exhaustive, rather an indication of some techniques that can be employed to define unit tests.

The MSTest framework, delivered with VS is used to run the tests. Where appropriate, the Moq framework is used to mock objects and limit the unit testing to the logic within the class being tests.

Unit tests will therefore be included for each of the following, where appropriate:

### Custom Routes

The test on the custom routing table was the one area where it was extensive. With the url structure and the resulting routing table determined late on in the project, it was necessary to design the route table completely separately from the application. Once tested, it would then be plugged into the application without breaking it, or at least minimising the amount of debugging required to integrate the table. This is akin to a Test Driven Development of the Routing table.

### Controllers

As a sample of controller action method tests, the Take survey process was chosen as it involves the most customisation and demonstrates how the individual components can be unit tested knowing how they integrate within the application. Other than that, there are too many controllers and action methods to adequately provide units for them all.

### Custom Model Binders

The take survey process has the only custom model binder, **TakeSurveyModelBinder**, within the application so the unit tests are centred round this module.

### Action Filters

The only non-mapping action filter, namely the AjaxActionOnly filter was chosen as the module to test. This was a good example in that it shows the limit of the basic tools available for unit testing.

This module basically checks that the incoming HttpRequest is an ajax originated request. It does this by checking the IsAjaxRequest() method on the HttpRequest. As it turns out, MS have implemented this method as a Static extension method. Moq can not deal with either of these class types. This left the choice of finding a technology that does. This points to MS’s own Fakes and Shims testing tools. The way to mock or intercept a static or extension method is to employ a shim, which essentially rewrites the code for the object at runtime. It appears simple to use on the face of it, but causes major problems with MVC and causes a severe error “Operation may destabilize the runtime”. This is well documented on the internet and basically demonstrates that the tools are, for the moment, incompatible.

Time constraints of the project mean that this test is left not functioning but not replaced. The code, however, is still in the project but the libraries have been removed as they stop any other tests from running.

### Html Helpers

The only Html Helper was also created as part ot eh Take Survey process. The “RadioButtonList” helper method was created to allow multiple groups of radio buttons to be displayed on the view at once. This represents the options for each question. The existing RadioButton helpers leaves all radio buttons with the same name and therefore only one would be selectable from the entire set of questions. This helper groups each of the options for each question allowing the respondent to answer each question.

### Mappings

There are various mapping classes to convert the view model classes to the business model classes. There are two associated with the Take Survey process

### Unit Test Libraries

Moq: this will be used to mock the various objects required for each unit test

MSTest or Nunit this library will be used to run the unit tests.

Glimpse This will be used to provide server side debugging of Routes.

NInject or Unity provide the necessary dependency injection to the various modules.

# Future Development

## Templates for Survey Questions and Responses

The ability to add template for surveys, question and Likert responses would be added to the system. This would facilitate quicker and simpler techniques for setting up surveys.

The current implementation will include the ability to store templates within the Domain Model, and populate these with test data. The maintenance of these templates would be reserved for future development.

This should be reflected in the comments in the About panel, if there is time.

# Acknowledgements

## Use of the MVC Charting tool

To avoid a steep learning curve and to understand how it works, various sources were investigate on the internet. A specific CodeProject article [Using ASP Net Charting with Image Map in MVC](http://www.codeproject.com/Articles/297677/Using-ASP-Net-Charting-with-Image-Map-in-MVC) proved to be the main source, by [Dr Song Li](http://www.codeproject.com/script/Membership/View.aspx?mid=5002996). The code used in the project is similarly structured but not identical, except for the code that sets the Chart settings, essentially the style of the chart.

Having checked beforehand, with Dr. James Paterson, that this is not plagiarising any code, as it doesn’t constitute a major part of the work, I have chosen to use the style from Dr. Song Li’s article.