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

GCUSurveys: Survey System

**System Design**

**Table of Contents**

[Project Setup 3](#_Toc338339742)

[Database setup 3](#_Toc338339743)

[Application Structure and Logic 5](#_Toc338339744)

[Overview 5](#_Toc338339745)

[Screen / Page Layouts 5](#_Toc338339746)

[Validation Rules (Business Logic0) 6](#_Toc338339747)

[Use of the MVC Framework 7](#_Toc338339748)

[Routing 7](#_Toc338339749)

[Controllers and Actions 7](#_Toc338339750)

[Model and Binding 7](#_Toc338339751)

[Validation 8](#_Toc338339752)

[Client / Server Functionality 8](#_Toc338339753)

[Filters 8](#_Toc338339754)

[Security: Protection and Robustness against attack 9](#_Toc338339755)

[Security: Membership, authentication and Authorisation 9](#_Toc338339756)

[Exception Handling 9](#_Toc338339757)

[Code Modules 10](#_Toc338339758)

[Unit Testing 10](#_Toc338339759)

[Future Development 11](#_Toc338339760)

[Templates for Survey Questions and Responses 11](#_Toc338339761)

# Project 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 GCUSurveys DbContext, EF creates the underlying database and populates it with sample data and template data.

The following code is included in the ***Application\_Start*** method of ***Global.asax*** to ensure that the underlying database is dropped and created if the model changes.

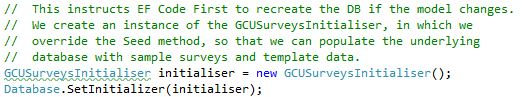


Figure - ER entry in Application\_Start method

As part of the Domain model, a custom database initialiser containing the following code was created which overrides the Seek method which is where the sample data is loaded into the newly created database, by EF.

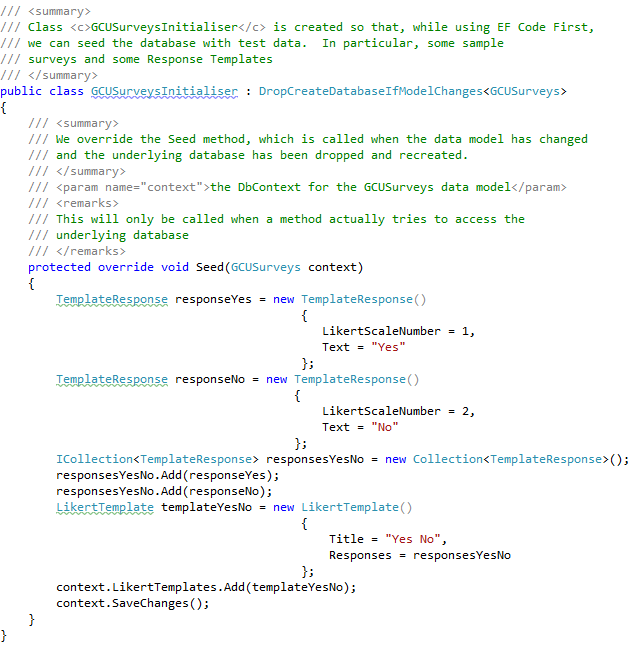


Figure - DB Initialiser

The sample code above shows a LikertScale template response being added to the database.

# Application Structure and Logic

## Overview

The application I sstructured 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 or remove questions to or from 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.

## 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.

## Validation Rules (Business Logic0)

1. Number of questions per survey should be restrict to 10, so that surveys are not too long ie. Between 1 and 10.
2. 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
3. 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

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 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

## Routing

To enable the RESTful approach to the Url’s for the application, a set of custom routes should be defined.

Add **Glimpse** (NuGet) server side debugging for the routes

Add **Moq** (NuGet also) mocking library, for unit testing the routes.

Unit Testing: module 2, lab, gives example of unit testing routes

## Controllers and Actions

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.

## Model and Binding

Domain folder:

* Contains domain model and Repositories
* This will use EF code first and reference an SQL Express 2008 database in the background.
* This puts a requirement on the application.
* Investigate how to point EF Code First to SQLCompact version 4.

Models Folder:

* Used with all views, held in models folder and represent viewModels to support the views of the system.

Make use of AutoMapper for mapping the domain and view models and vice-versa.

* This can be done as appropriate, using Action Filters for the gets and if possible, for the Posts too.
* The posts may be better to code the mapping within the controller.

Custom Binding

* Keep any custom binders in the Infrastructure folder.
* The Status and UserId (Owner) should be injected into the model binding, as this would avoid the need to code the setting of these fields within the controller action methods themselves.

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.

## Validation

Client Side Validation

* To keep the application as responsive as possible, despite the fact that it is an MVC app and not a single page application, we will make use of the client side validation.
* Where necessary, the setting of the likes of the status field should be done using jQuery libraries to set the values within the client, again to keep the performance of the system at an optimum.
* Individual cases where the Data for the status change must be set, but should NOT be editable by the user, can use either of the two following methods:
  + Use jQuery to set the StatusDate field when the user changes the status of the syrvey via the dropdown box[[1]](#footnote-1).
  + Use a custom binder to inject these into the model, as this avoids having to remember to code this in the controllers.
* Validation will be set up using validation attributes and decorate the view models, this will allow the server side to augment the validation should the browser be configured with javascript disabled.
* 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 / Server Functionality

Validation

* Validation will be shared between both the client and the server sides of the application. This optimises the performance of the application from the clients perspective, and also ensures the safety of the application by provided the security of server side validation.
* The use of server side validation would also protect the application should the user use a suitably encoded the url from the address bar. The server side validation will still apply.

## Filters

Model Mapping

* Filters will be used for mapping the models to the viewmodels. Where possible, to simplilfy the coding in the controllers. (Action Filters: OnActionExecuted() and onActionExecuting() methods)
* This allows standard and common mappings to be completed by adding an attribute to the relevant controller.

Built-In Filters

* The built in filters could be made use of. They are:
  + **RequireHttps**: Forces the request to be redirected through https. This may be significant for the secured methods.
  + **ValidateAntiForgeryToken**: Ensures the anti forgery token is checked each time. Investigate whether this needs encryption certificates to be placed on the client machine? Investigate how if it is a requirement.
  + **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.
  + **ChildActionOnly**: This marks an action method for execution as a child method?

Error handling Filters

* Develop a custom filter for error handling: to log the fault to a text file and then direct to an Error page.

## 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.

## Security: 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 requirments. DO NOT RE-INVENT THE WHEEL.

## Exception Handling

Error Handler

* Develop a custom Error Handler that can be placed as a global attribute.
* It should include logging the error information to a text file.

## Code Modules

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.
* The main emphasis of the project is MVC and coding the presentation layer of an application. It is therefore considered more work than necessary for the project to code a strict layered application model for the Business layer and below.

Dependency Injection

* References to the Business project from the Presentation project will be injected into the controllers of the MVC application.
* The MVC application, being MVC3, will use Ninject MVC as the DI/IoC library. This version of Ninject creates its own custom Dependency Resolver and is extremely simple to implement.
* The Business Layer will expose an interface following the Façade pattern (GoF).
* This will aid unit testing of the controllers as it will allow the Business layer to be mocked.
* It will also facilitate unit testing of the Business Layer. The data access will be mocked in such tests.

## Unit Testing

A major emphasis of this project is to demonstrate the testability of the various components and customisations made within the MVC framework. Unit tests will therefore be included for each of the following, where appropriate:

### Custom Routes

The various custom routes will be unit tested

Examples can be found in Wk 2 notes.

### Controllers

All controllers will be unit tested, using mocked repositories of the Business Layer Façade.

Exmples to be found in Wk 3 notes and labs

### Domain Model

The business Layer will be unit tested, for each method exposed by the Façade, to ensure the presentation layer receives the data expected.

The Data access will be mocked within these unit tests.

### Custom Model Binders

These will also be unit tested, with examples found in Wk 4 Demo code

### Action Filters

These will be unit tested.

### 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.

1. This may not be user editable either, but coded behind the scenes for the relevant page with the application. [↑](#footnote-ref-1)