|  |
| --- |
| **SEDCO**  **CVM TEAM**  **C# Windows Form Documentation**  **Survey Question Configurator**  **Mahdi Suleiman**  **Date: 25th May 2022** |

Table of Contents

[Students' Property Right Declaration and Anti-Plagiarism Statement i](#_Toc61328405)

[Abstract v](#_Toc61328408)

[Chapter 1: User 1](#_Toc61328409)

[Chapter 2: Admin & Installation 5](#_Toc61328410)

[Chapter 3: Developer 10](#_Toc61328411)

[References 35](#_Toc61328417)

# Abstract

Survey Question Configurator is a C# Windows Form Application for configuring 3 types of questions

1. Smiley Question, where you have to set:
   1. Question Order
   2. Question Text
   3. Number of smiley faces
2. Slider Question, where you have to set:
   1. Question Order
   2. Question Text
   3. Start Value
   4. Start Value Caption
   5. End Value
   6. End Value Caption
3. Stars Question:
   1. Question Order
   2. Question Text
   3. Number of smiley stars.

You can Add, Edit, View and Delete any question.

All changes are reflected to a Microsoft SQL server database.

**Keywords:**

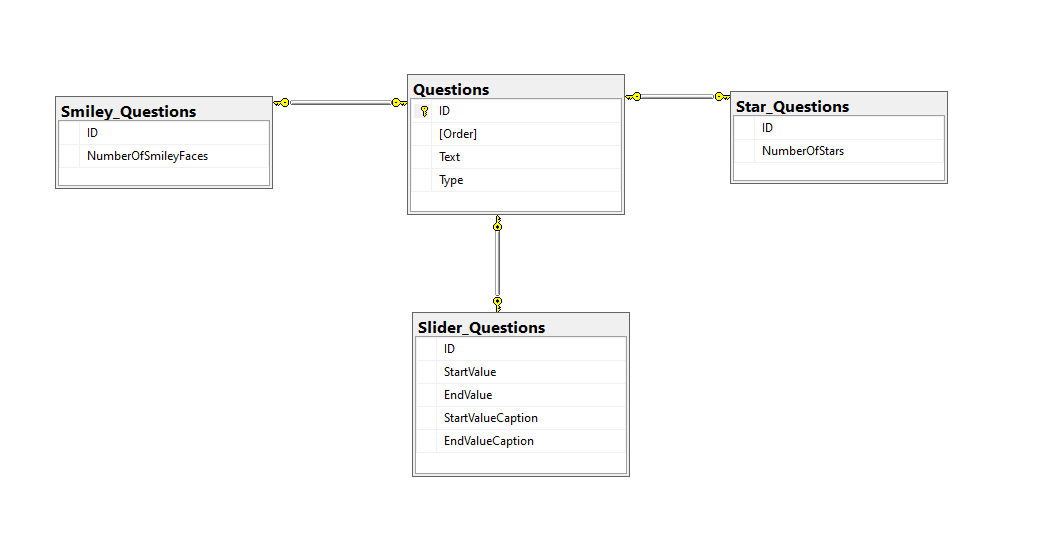
Microsoft, C#, SQL, Windows Form

# Chapter 1: Developer

**3.1 Design Overview**

**3.1.1 Database Design**

The figure below shows database entity-relation diagram.



\*Notes on tables:

1. Questions table:
   1. Types
2. ID [INT].
3. Order [INT].
4. Text [nvarchar(4000)].
5. Type [INT].
   1. Constraints
      * ID -> auto increment primary key.
      * Order -> Unique key.
      * Text -> Less than 4000 characters.
      * Type -> must be 0 or 1 or 2.
        + 0 -> Smiley type question.
        + 1-> Slider type question.
        + 2-> Star type question.
6. Smiley\_Questions
   1. Types
      * ID [INT].
      * NumberOfSmileyFaces [INT].
   2. Constraints
      * ID-> Foreign key + Unique key + NOT NULL.
        + On update cascade
        + On delete cascade
      * NumberOfSmileyFaces -> Between 2 and 5 **inclusive**.
7. Slider\_Questions
   1. Types
      * ID [INT].
      * StartValue [INT].
      * EndValue [INT].
      * StartValueCaption [nvarchar(100)].
      * EndValueCaption [nvarchar(100)].
   2. Constraints
      * ID-> Foreign key + Unique key + NOT NULL.
        + On update cascade
        + On delete cascade
      * StartValue -> Between 1 and 99 **inclusive**.
      * EndValue -> Between 2 and 100 **inclusive**.
      * StartValueCaption-> Less than 100 characters.
      * EndValueCaption-> Less than 100 characters.
8. Star\_Questions
   1. Types
      * ID [INT].
      * NumberOfStars [INT].
   2. Constraints
      * ID-> Foreign key + Unique key + NOT NULL.
        + On update cascade
        + On delete cascade
      * NumberOfStars-> Between 1 and 10 **inclusive**.

**3.1.2 Stored Procedures**

There are 6 stored procedures.

3 for inserting questions

1. INSERT\_SMILEY\_QUESTION
2. INSERT\_SLIDER\_QUESTION
3. INSERT\_STAR\_QUESTION

And 3 for updating questions

1. UPDATE\_SMILEY\_QUESTION
2. UPDATE\_SLIDER\_QUESTION
3. UPDATE\_STAR\_QUESTION

**INSERT\_SMILEY\_QUESTION:**

**Paramters:**

@ORDER INT,

@TEXT NVARCHAR(4000),

@TYPE INT,

@NumberOfSmileyFaces INT

**Description:**

Takes Parameters and insert into 2 tables:

First, insert Order, Text and type into the [Questions] table Then it carries the ID from the first operation and insert it along with NumberOfSmileyFaces into the [Smiley\_Questions] table.

**Returns:**

A table with the either success or error as ErrorCode column, and ERROR\_MESSAGE() as ErrorMessage column.

SUCCESS = 1

ERROR = -1

**INSERT\_SLIDER\_QUESTION:**

**Paramters:**

@Order INT,

@Text NVARCHAR(4000),

@Type INT,

@StartValue INT,

@EndValue INT,

@StartValueCaption nvarchar(100),

@EndValueCaption nvarchar(100)

**Description:**

Takes Parameters and insert into 2 tables:

First, insert Order, Text and type into the [Questions] table Then it carries the ID from the first operation and insert it along with Start & End values and their captions into the [Slider\_Questions] table.

**Returns:**

A table with the either success or error as ErrorCode column, and ERROR\_MESSAGE() as ErrorMessage column.

SUCCESS = 1

ERROR = -1

**INSERT\_STAR\_QUESTION:**

**Paramters:**

@ORDER INT,

@TEXT NVARCHAR(4000),

@TYPE INT,

@NumberOfStars INT

**Description:**

Takes Parameters and insert into 2 tables:

First, insert Order, Text and type into the [Questions] table Then it carries the ID from the first operation and insert it along with NumberOfStars into the [Star\_Questions] table.

**Returns:**

A table with the either success or error as ErrorCode column, and ERROR\_MESSAGE() as ErrorMessage column.

SUCCESS = 1

ERROR = -1

**UPDATE\_SMILEY\_QUESTION:**

**Paramters:**

@ID INT,

@Order INT,

@Text NVARCHAR(4000),

@Type INT,

@NumberOfSmileyFaces INT

**Description:**

Takes Parameters and updates 2 tables:

First, insert Order and Text from [Questions] table based on the provided ID, then updates NumberOfSmileyFaces from [Smiley\_Questions] table.

**Returns:**

A table with the either success or error as ErrorCode column, and ERROR\_MESSAGE() as ErrorMessage column.

SUCCESS = 1

ERROR = -1

**UPDATE\_SLIDER\_QUESTION:**

**Paramters:**

@ID INT,

@Order INT,

@Text NVARCHAR(4000),

@Type INT,

@StartValue INT,

@EndValue INT,

@StartValueCaption nvarchar(100),

@EndValueCaption nvarchar(100)

**Description:**

Takes Parameters and updates 2 tables:

First, insert Order and Text from [Questions] table based on the provided ID, then updates Start and End values and their captions from [Slider\_Questions] table.

**Returns:**

A table with the either success or error as ErrorCode column, and ERROR\_MESSAGE() as ErrorMessage column.

SUCCESS = 1

ERROR = -1

**UPDATE\_STAR\_QUESTION:**

**Paramters:**

@ORDER INT,

@TEXT NVARCHAR(4000),

@TYPE INT,

@NumberOfSmileyFaces INT

**Description:**

Takes Parameters and updates 2 tables:

First, insert Order and Text from [Questions] table based on the provided ID, then updates NumberOfStars from [Star\_Questions] table.

**Returns:**

A table with the either success or error as ErrorCode column, and ERROR\_MESSAGE() as ErrorMessage column.

SUCCESS = 1

ERROR = -1

**3.1.3 Functions**

There are 2 Scalar-valued functions.

1. CheckIfOrderExist
2. CheckIfUpdatingSameQuestion

**CheckIfOrderExist**

**Paramters:**

@ORDER INT

**Description:**

Checks the count of ID’s that match the input order

**Returns:**

Return SUCCESS = 1 if the input order does **NOT** exist

Or SQL\_VALIDATION = 2 if the input order already **exists**.

**CheckIfUpdatingSameQuestion**

**Paramters:**

@ORDER INT

@ID INT

**Description:**

Here we have 2 ID’s

One is already provided

And the other is returned from a select statement where we search for the ID of the provided order

And saved in a declared variable named [@ReturnedID]

A comparison happens between the two ID’s

**Returns:**

SUCCESS = 1 if the 2 ID’s are identical

ERROR = -1 if the 2 ID’s are different

**3.1.2 Code Design & Architecture**

The code is based upon:

1. Three-layer architecture
2. Repository design pattern

**Three-Layer Architecture:**

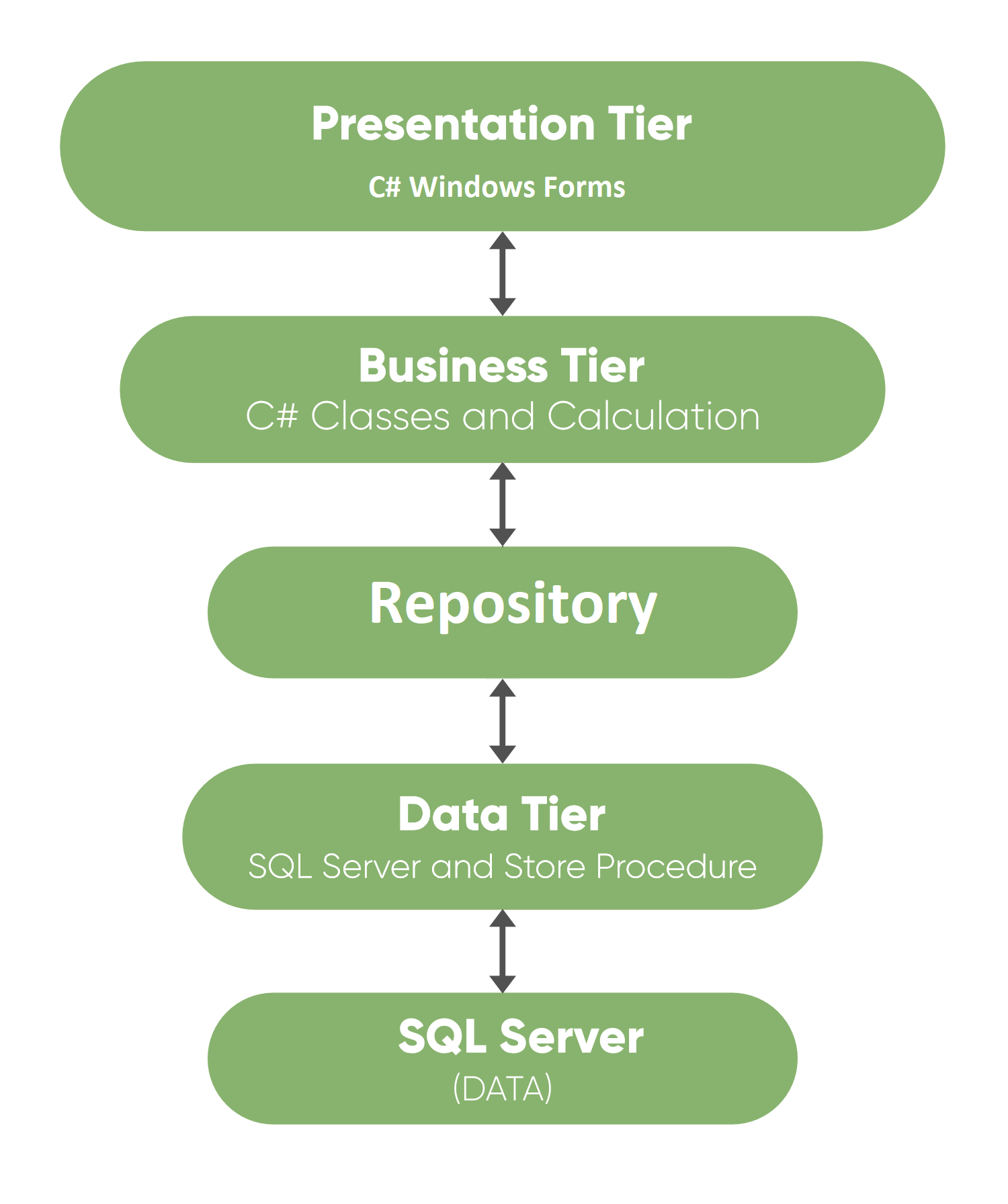
The code is decoupled into 3 layers:

1. Presentation Layer
2. Business Logic Layer
3. Data Access Layer

**Repository Design Pattern:**

The Repository Design Pattern in C# Mediates between the domain and the data mapping layers using a collection-like interface for accessing the domain objects.

In other words, we can say that a Repository Design Pattern acts as a middle layer between the rest of the application and the data access logic. That means a repository pattern isolates all the data access code from the rest of the application. The advantage of doing so is that, if you need to do any change then you need to do in one place. Another benefit is that testing your controllers becomes easy because the testing framework need not run against the actual database access code.



**3.2 Naming Convention**

**t**VariableName -> Temporary variable

**m**VariableName -> Private variable

**p**VariableName -> Parameter variable

**c**VariableName -> Constant variable

**3.3 Presentation Layer (Windows Forms)**

There are 4 main forms:

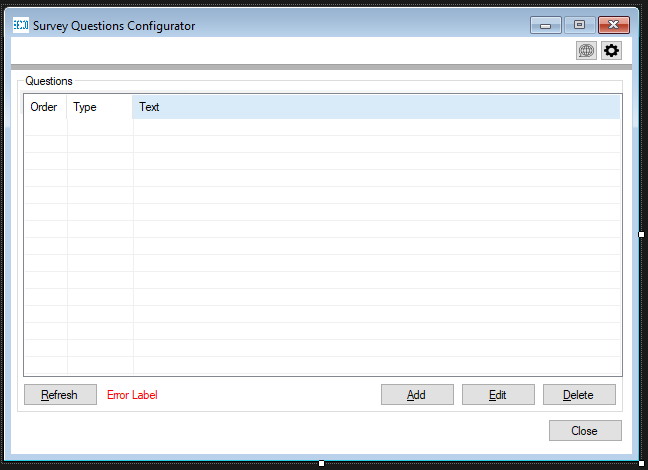
1. SurveyQuestionConfiguratorForm
2. AddQuestionForm
3. LanguageSettingsForm
4. ConnectionSettingsForm

Every form has 2 languages a user can change from

1. English (default)
2. Arabic

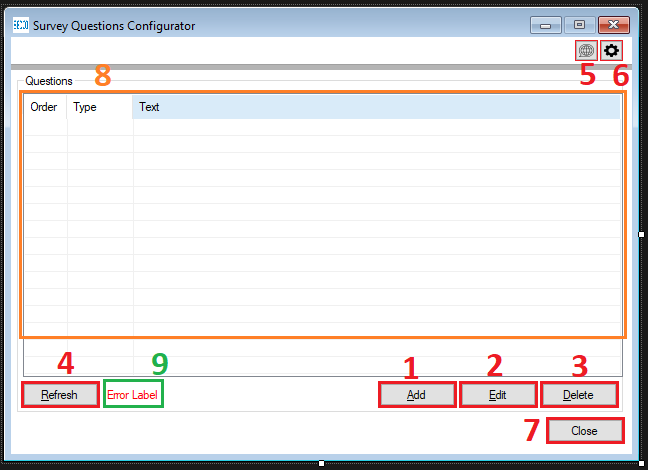
Every Languages has its own resource files to translate messages for both languages named like the following: FormName+”Strings”.resx for English resource files OR FormName+”Strings”.ar-JO.resx for Arabic resource files.

**SurveyQuestionConfiguratorForm**

****

The form has total 9 main controls

1. Add a question.
2. Edit a question.
3. Deleted a question.
4. Refresh the questions list.
5. Change language settings.
6. Change connection settings.
7. Close the application.
8. ListView
9. Label

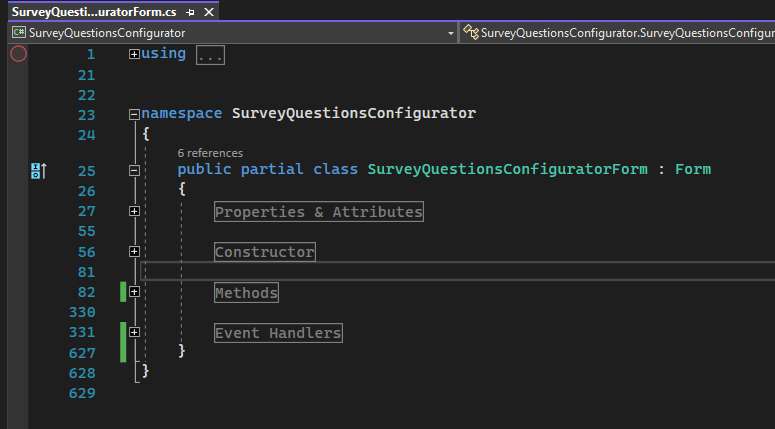


**Control’s Properties:**

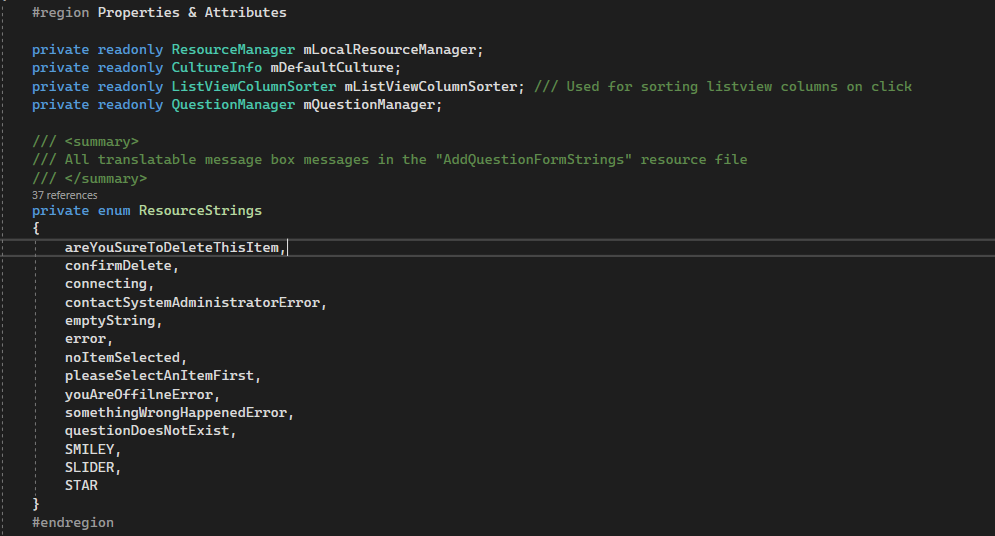
* Form
  1. MinimumSize = MaximumSize = Size
  2. Localizable = True
  3. AutoSize = False
* ListView
  1. FullRowSelect = True
  2. GridLines = True
  3. View = Details
  4. Activation = OnClick
  5. MultiSelect = False

**Code:**

The code is divided into regions based on their functionality.

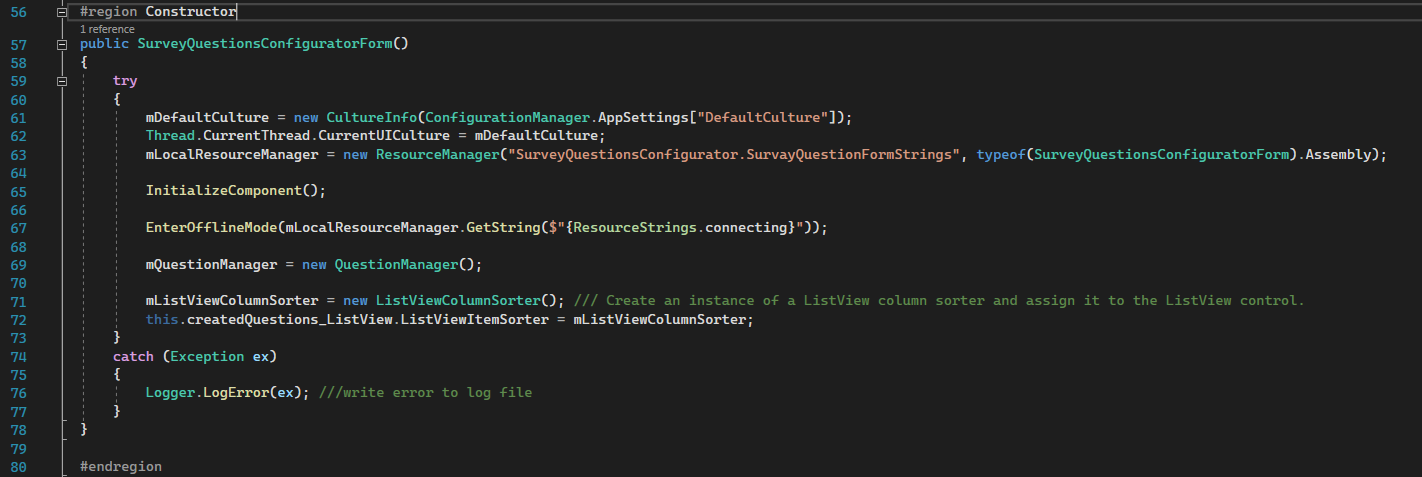


**Properties & Attributes**



* mLocalResourceManager-> used to access resource files.
* mDefaultCulture-> used to save current CultureInfo (language) of the application [fetched from app.config]
* mListViewColumnSorter-> used to sort ListView’s columns [copy pasted from MS docs]
* mQuestionManager-> object from QuestionManager [Business Logic Layer]
* enum ResourceStrings-> contains all the keys in "AddQuestionFormStrings" resource file which are used for displaying translatable messages

**Constructor**



Line 61 & 62: gets saved default culture in config file and assign it to current UI culture thread before initializing component

Line 63: Initializes a new instance of the ResourceManager class that looks up resources contained in files with the specified root name in the given assembly.

Line 67: makes the application enter an offline state and display “Connecting…” before connecting and getting data from database.

Lines 70 & 71: handling listview column click sorting (taken from MS docs)

Line 76: write any error/exception in the log file [located inside a folder named logs]

**Methods**

