

Survey Question Configurator

Code Documentation

Table of Contents

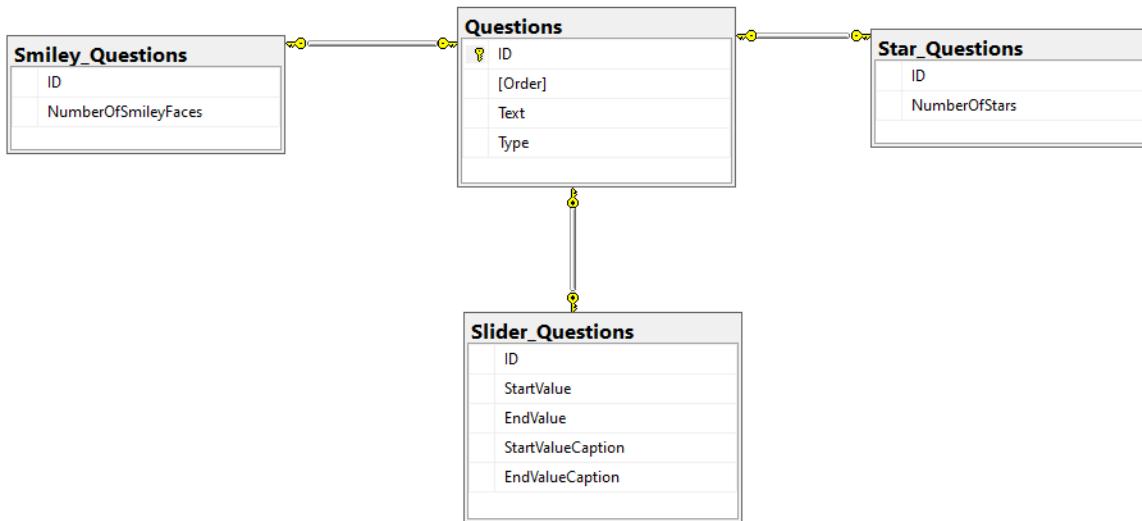
Chapter 1: Developer.....**Error! Bookmark not defined.**

Chapter 1: Developer

1.1 Design Overview

1.1.1 Database Design

The figure below shows database entity-relation diagram.



*Notes on tables:

A. Questions table:

1. Types
 1. ID [INT].
 2. Order [INT].
 3. Text [nvarchar(4000)].
 4. Type [INT].
2. 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.

B. 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**.

C. 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.

D. 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**.

1.1.2 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

1.1.3 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

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 it 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.

Presentation Tier

C# Windows Forms

Business Tier

C# Classes and Calculation

Repository

Data Tier

SQL Server and Store Procedure

SQL Server

(DATA)

1.3 Naming Convention

tVariableName -> Temporary variable

mVariableName -> Private variable

pVariableName -> Parameter variable

cVariableName -> Constant variable

1.4 Presentation Layer (Windows Forms)

There are 4 main forms:

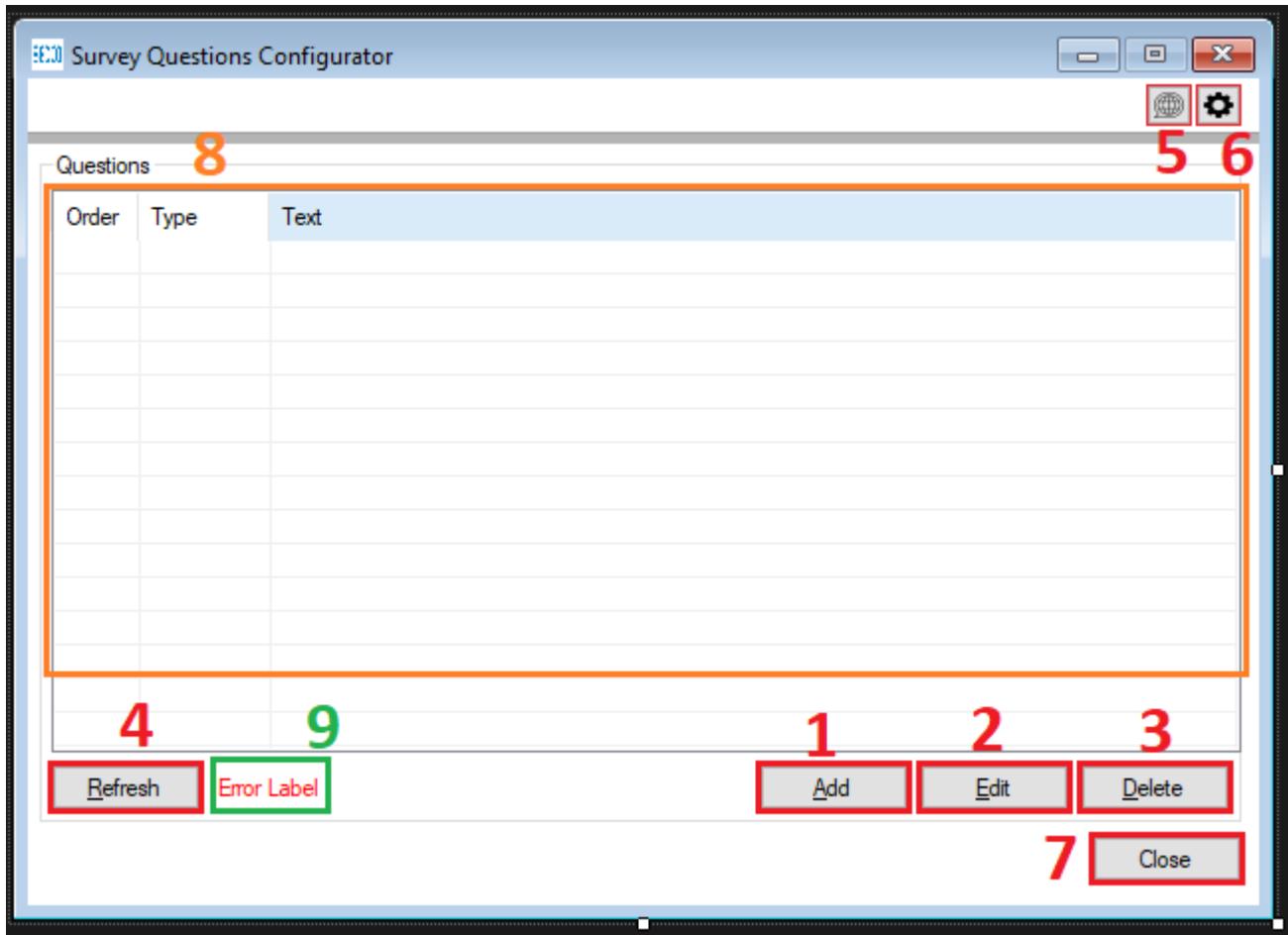
1. SurveyQuestionConfiguratorForm
2. AddQuestionForm
3. LanguageSettingsForm
4. ConnectionSettingsForm

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.

1.4.1 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.

A screenshot of a code editor showing the `SurveyQuestionsConfiguratorForm.cs` file. The code is organized into several regions, each represented by a box with a plus sign (+) and a label: `Properties & Attributes`, `Constructor`, `Methods`, and `Event Handlers`. These boxes are positioned around the corresponding sections of the code. The code itself is a C# class definition:

```
1  using ...
21
22
23  namespace SurveyQuestionsConfigurator
24  {
25      public partial class SurveyQuestionsConfiguratorForm : Form
26      {
27          // Properties & Attributes
28
29          // Constructor
30
31          // Methods
32
33          // Event Handlers
34      }
35
36  }
```

The code editor interface includes tabs for `C# SurveyQuestionsConfigurator` and `SurveyQuestionsConfigurator.SurveyQuestionsConfigur`, and a status bar at the bottom.

Properties & Attributes

```
#region Properties & Attributes

private readonly ResourceManager mLocalResourceManager;
private readonly CultureInfo mDefaultCulture;
private readonly ListViewColumnSorter mListviewColumnSorter; // Used for sorting listview columns on click
private readonly QuestionManager mQuestionManager;

/// <summary>
/// All translatable message box messages in the "AddQuestionFormStrings" resource file
/// </summary>
37 references
private enum ResourceStrings
{
    areYouSureToDeleteThisItem,
    confirmDelete,
    connecting,
    contactSystemAdministratorError,
    emptyString,
    error,
    noItemSelected,
    pleaseSelectAnItemFirst,
    youAreOfflineError,
    somethingWrongHappenedError,
    questionDoesNotExist,
    SMILEY,
    SLIDER,
    STAR
}
#endregion
```

- 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

```
56  #region Constructor
57  1reference
58  public SurveyQuestionsConfiguratorForm()
59  {
60      try
61      {
62          mDefaultCulture = new CultureInfo(ConfigurationManager.AppSettings["DefaultCulture"]);
63          Thread.CurrentThread.CurrentCulture = mDefaultCulture;
64          mLocalResourceManager = new ResourceManager("SurveyQuestionsConfigurator.SurveyQuestionFormStrings", typeof(SurveyQuestionsConfiguratorForm).Assembly);
65
66          InitializeComponent();
67
68          EnterOfflineMode(mLocalResourceManager.GetString($"{ResourceStrings.connecting}"));
69
70          mQuestionManager = new QuestionManager();
71
72          mListviewColumnSorter = new ListViewColumnSorter(); // Create an instance of a ListView column sorter and assign it to the ListView control.
73          this.createdQuestions_ListView.ListViewItemSorter = mListviewColumnSorter;
74      }
75      catch (Exception ex)
76      {
77          Logger.LogError(ex); //write error to log file
78      }
79  }
80  #endregion
```

Step 1: gets saved default culture in config file and assign it to current UI culture thread before initializing component

Step 2: initializes a new instance of the ResourceManager class that looks up resources contained in files with the specified root name in the given assembly.

Step 3: makes the application enter an offline state and display “Connecting...” before connecting and getting data from database.

Step 4: handling listview column click sorting (taken from MS docs)

Step 5: write any error/exception in the log file [located inside a folder named logs under debug folder]

Methods

ClearListView

```
82  #region Methods
83
84  /// <summary> Clear list view by deleting all of it's rows
85  /// <summary> Clear list view by deleting all of it's rows
86  /// <summary> Clear list view by deleting all of it's rows
87  public void ClearListView()
88  {
89      /// Remove each row
90      try
91      {
92          /// Perform thread safe call to prevent Cross-thread operation exception
93          if (createdQuestions_ListView.InvokeRequired)
94          {
95              createdQuestions_ListView.Invoke(new Action(() =>
96              {
97                  foreach (ListViewItem item in createdQuestions_ListView.Items)
98                  {
99                      item.Remove();
100                 }
101             }));
102         }
103     else
104     {
105         foreach (ListViewItem item in createdQuestions_ListView.Items)
106         {
107             item.Remove();
108         }
109     }
110 }
111 catch (Exception ex)
112 {
113     Logger.LogError(ex); //write error to log file
114 }
115 }
```

Parameters:

None.

Description:

This method is used to clear every row in the ListView-> foreach is used instead of .clear() methods because it causes the columns to be deleted too.

Step 1: Check if this control requires invoke, which essentially means check if this control is being modified by another thread (background thread) other than the one it was created on (main or UI thread).

Step 2: Else, you are on the same thread that created it.

Returns:

void.

BuildListView

```
117  /// <summary>
118  /// This method gets called from QuestionManager(BLL)
119  /// Refill list view with data when needed (on Add, Edit, Refresh, ...etc)
120  /// </summary>
121  /// <param name="pErrorCode">
122  /// <param name="pQuestionList">
123  /// Essential for allowing threads to call this function
124  /// </param>
125  1 reference
126  public void BuildListView(ErrorCode pErrorCode, List<Question> pQuestionList)
127  {
128      /// Connect to question table and fill the list view
129      try
130      {
131          switch (pErrorCode)
132          {
133              case ErrorCode.SUCCESS:
134              case ErrorCode.EMPTY:
135              {
136                  /// If connectin to DB is SUCCESS -> Enable buttons and list view
137                  EnterOnlineMode();
138
139                  /// Perform thread safe call to prevent Cross-thread operation exception
140                  if (this.createdQuestions_ListView.InvokeRequired)
141                  {
142                      this.createdQuestions_ListView.Invoke(new Action(() =>
143                          {
144                              FillListView(pQuestionList);
145                          }));
146                  }
147                  else
148                  {
149                      FillListView(pQuestionList);
150                  }
151              break;
152          }
153
154          ///If connectin to DB is NOT SUCCESS -> Disable buttons and list view
155          default:
156          {
157              string tOfflineMessage = mLocalResourceManager.GetString("${ResourceStrings.youAreOfflineError}");
158              EnterOfflineMode(tOfflineMessage);
159              break;
160          }
        }
```

Parameters:

pErrorCode-> ErrorCode. holds of the state of DB operation was successful, error, etc...

pQuestionList-> List<Question>. holds values return from DB to be viewed in the list

Description:

This method is called from the business logic layer (QuestionManager.cs) and holds all the values that came back from the database (error code and actual values).

Step 1: make the form go into online mode which is making all controls active to the user to interact with.

Step 2: fill the list with actual data.

Step 3: get error message to be shown to the user in the “errorLabel” from the resource manager depending on the current selected language (taken from “DefaultCultrue” section in the app.config file)

Step 4: make the form go into offline mode which is making only the refresh button active and interactive with the user.

Returns:

void.

FillListView

```
168
169  /// <summary> Fill list view with passed data
170  <reference>
171  private void FillListView(List<Question> pQuestionsList)
172  {
173      try
174      {
175          ClearListView();
176
177          ListViewItem tListViewItem; // Used for creating listview items (rows).
178
179          /// Perform thread safe call to prevent Cross-thread operation exception
180          if (this.createdQuestions_ListView.InvokeRequired)
181          {
182              this.createdQuestions_ListView.Invoke(new Action(() =>
183              {
184
185                  foreach (Question q in pQuestionsList)
186                  {
187                      string tQuestionType = mLocalResourceManager.GetString($"{{QuestionType}}{q.Type}");
188
189                      /// Add id as a tag to Order column -> use it while it's hidden
190                      tListViewItem = new ListViewItem($"{q.Order}")
191                      {
192                          Tag = q.ID //== SubItems[0].Tag = q.ID;
193                      };
194                      tListViewItem.SubItems.Add($"{tQuestionType}");
195
196                      tListViewItem.SubItems[1].Tag = q.Type; // Save question type in Type's column tag -> Decouple what is shown in UI from actual data
197                      tListViewItem.SubItems.Add($"{q.Text}");
198
199                      this.createdQuestions_ListView.Items.Add(tListViewItem); // add whole row
200                  }
201              }));
202          }
203      }
204  }
```

Parameters:

pQuestionList-> List<Question>. list of questions to be rendered to users.

Description:

This method fills the list view with the data passed to it.

Step 1: gets the question type string to be shown in the “Type” column based on the current select language.

Step 2: create new ListViewItem with the first element to be show is the order of the question.

Step 3: Save the ID of the question in its “Tag” property so it is not shown to the user but it’s still used in the DB operations.

Step 4: save the type of the question in the “Tag” property because it is different than what is shown in the list view so it can’t be taken directly from it and do DB operations with it.

Step 5: add the row that was just created to the list view.

Returns:

Void.

EnterOfflineMode

```
230
231     /// <summary>
232     /// Set the form into offline mode where user can't do any action that can interact with DB except changing the connection settings
233     /// </summary>
234     /// <param name="pOfflineMeesage">
235     /// Passed message to be displayed to error label
236     /// </param>
237     private void EnterOfflineMode(string pOfflineMeesage)
238     {
239         try
240         {
241             /// Perform thread safe call to prevent Cross-thread operation exception
242             if (this.InvokeRequired)
243             {
244                 this.Invoke(new Action(() =>
245                 {
246                     errorLabel.Text = pOfflineMeesage;
247
248                     addQuestionButton.Enabled = false;
249                     editQuestionButton.Enabled = false;
250                     deleteQuestionButton.Enabled = false;
251
252                     ClearListView();
253                     createdQuestions_ListView.Enabled = false;
254                 }));
255             }
256             else...
257
258             /// Move error label 5 pixels to fit all message in a neat way
259             if (pOfflineMeesage == mLocalResourceManager.GetString("${ResourceStrings.youAreOffilneError}"))
260             {
261                 if (errorLabel.InvokeRequired)
262                 {
263                     errorLabel.Invoke(new Action(() =>
264                     {
265                         errorLabel.Top = 309;
266                     }));
267                 }
268                 else
269                 {
270                     errorLabel.Top = 309;
271                 }
272             }
273         }
274     }
```

Parameters:

pOfflineMeesage-> string. message to be displayed to the user in the error label

Description:

Step 1: checks if the form needs to be invoked and changed multiple controls at once.

Step 2: move the label 5 pixels to the top so it can fit the long offline error message with a nice UI.

Returns:

void.

EnterOnlineMode

```
292     ///<summary>
293     /// Reset Form to online mode where the user can interact with it normally
294     /// </summary>
295     private void EnterOnlineMode()
296     {
297         try
298         {
299             /// Perform thread safe call to prevent Cross-thread operation exception
300             if (this.InvokeRequired)
301             {
302                 this.Invoke(new Action(() =>
303                 {
304                     errorLabel.Text = "";
305
306                     createdQuestions_ListView.Enabled = true;
307                     addQuestionButton.Enabled = true;
308                     editQuestionButton.Enabled = true;
309                     deleteQuestionButton.Enabled = true;
310                 }));
311             }
312             else
313             {
314                 errorLabel.Text = "";
315
316                 createdQuestions_ListView.Enabled = true;
317                 addQuestionButton.Enabled = true;
318                 editQuestionButton.Enabled = true;
319                 deleteQuestionButton.Enabled = true;
320             }
321
322         }
323         catch (Exception ex)
324         {
325             Logger.LogError(ex);
326             throw; // Throw error to show it on a higher level function
327         }
328     } //End Function.
329
330 #endregion
```

Parameters:

None.

Description:

Returns all controls enabled back.

Returns:

None.

Event Handlers:

SurveyQuestionsConfiguratorForm_Load:

```
332     #region Event Handlers
333     private void SurveyQuestionsConfiguratorForm_Load(object sender, EventArgs e)
334     {
335         try
336         {
337             QuestionManager.refreshDataEvent += BuildListView; // Add BuildListView() reference to event
338             mQuestionManager.WatchForChanges(); // Subscribe to data changes event
339         }
340         catch (Exception ex)
341         {
342             ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}",
343             Logger.LogError(ex);
344         }
345     }///End event
346
```

Description:

This event handler adds BuildListView reference to the event that calls this function when DB “Questions” table changes and subscribes to this event by calling WatchForChanges() function.

createdQuestions_ListView_ColumnClick

```
347     /// <summary>
348     /// Sort list for each column
349     /// </summary>
350     private void createdQuestions_ListView_ColumnClick(object sender, ColumnEventArgs e)
351     {
352         try
353         {
354             // Determine if clicked column is already the column that is being sorted.
355             if (e.Column == mListviewColumnSorter.SortColumn)
356             {
357                 // Reverse the current sort direction for this column.
358                 if (mListviewColumnSorter.Order == System.Windows.Forms.SortOrder.Ascending)
359                 {
360                     mListviewColumnSorter.Order = System.Windows.Forms.SortOrder.Descending;
361                 }
362                 else
363                 {
364                     mListviewColumnSorter.Order = System.Windows.Forms.SortOrder.Ascending;
365                 }
366             }
367             else
368             {
369                 // Set the column number that is to be sorted; default to ascending.
370                 mListviewColumnSorter.SortColumn = e.Column;
371                 mListviewColumnSorter.Order = System.Windows.Forms.SortOrder.Ascending;
372             }
373
374             // Perform the sort with these new sort options.
375             this.createdQuestions_ListView.Sort();
376         }
377         catch (Exception ex)
378         {
379             ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error);
380             Logger.LogError(ex);
381         }
382     }///End event
```

Description:

Sorts ListView's columns based on each column values.

This code was taken from [MS docs](#).

addQuestionButton_Click

```
384     /// <summary>
385     /// Handle add question button click
386     /// </summary>
387     private void addQuestionButton_Click(object sender, EventArgs e)
388     {
389         try
390         {
391             AddQuestionForm addQuestionForm = new AddQuestionForm();
392             DialogResult tResult = addQuestionForm.ShowDialog(this);
393             if (tResult == DialogResult.OK)
394             {
395                 mQuestionManager.InstantlyRefreshList();
396             }
397         }
398         catch (Exception ex)
399         {
400             ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}");
401             Logger.LogError(ex);
402         }
403     }///End event
```

Description:

This event handler handles opening up new form for adding a question to the question list.

Step 1: if the add operation was successful the list will be refresh instantly to show the new added question to the user.

deleteQuestionButton_Click

```
405     /// <summary>
406     /// Handle delete question click
407     /// </summary>
408     private void deleteQuestionButton_Click(object sender, EventArgs e)
409     {
410         try
411         {
412             ///If at least one question is selected
413             if (createdQuestions_ListView.SelectedItems.Count > 0)
414             {
415                 /// Save selected item before it is unchecked by dialog box (prevent error)
416                 var selectedItem = createdQuestions_ListView.SelectedItems[0];
417
418             ///Display confirmation dialog first
419             var confirmResult = ShowMessage.Box(${ResourceStrings.areYouSureToDeleteThisItem}, ${ResourceStrings.confirmDelete}, MessageBoxButtons.YesNo);
420             if (confirmResult == DialogResult.Yes)
421             {
422                 int tQuestionId; /// question to be deleted
423                 ErrorCode result;
424
425                 /// Get ID from hidden ID column
426                 tQuestionId = Convert.ToInt32(selectedItem.Tag);
427                 result = mQuestionManager.DeleteQuestionByID(tQuestionId);
428
429                 /// unselect item -> avoid errors
430                 createdQuestions_ListView.SelectedItems.Clear();
431                 switch (result)
432                 {
433                     case ErrorCode.SUCCESS:
434                         mQuestionManager.InstantlyRefreshList();
435                         break;
436
437                     case ErrorCode.VALIDATION:
438                         ShowMessage.Box(${ResourceStrings.questionDoesNotExist}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error);
439                         break;
440
441                     default:
442                         ShowMessage.Box(${ResourceStrings.contactSystemAdministratorError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error);
443                         break;
444                 }
445             }
446         }
447     }
```

Description:

This event handler checks at first if there is any row selected to be deleted or not and save it to a variable.

Then displays a confirmation message box dialog to confirm the deletion process from the user.

If the user selected “Yes”-> try to delete question from the DB and do some action based on the returned result, either refresh the list or display a corresponding message to the user.

refreshDataButton_Click

```
459     /// <summary>
460     /// Handle refresh button click
461     /// </summary>
462     private void refreshDataButton_Click(object sender, EventArgs e)
463     {
464         try
465         {
466             mQuestionManager.InstantlyRefreshList();
467         }
468         catch (Exception ex)
469         {
470             ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, Message
471             Logger.LogError(ex);
472         }
473     } //End event
```

Description:

This event handler handles the refresh button click event which is calling the method that invokes BuildListView method that is subscribed to it.

editQuestionButton_Click

```
475     /// <summary>
476     /// Handle edit question button click
477     /// </summary>
478     private void editQuestionButton_Click(object sender, EventArgs e)
479     {
480         try
481         {
482             AddQuestionForm tAddQuestionForm = null; // Accept Qustion object
483             DialogResult tFormDialogResult = DialogResult.None; // Used to instantly refresh list if a question was added or edited successfully
484             if (createdQuestions_ListView.SelectedItems.Count > 0) // If at least one question is selected
485             {
486                 int tQuestionId = Convert.ToInt32(createdQuestions_ListView.SelectedItems[0].Tag);
487                 Question tQuestion = null;
488                 if (createdQuestions_ListView.SelectedItems[0].SubItems[1].Tag.ToString() == QuestionType.SMILEY.ToString())
489                 {
490                     tQuestion = new Question(tQuestionId, QuestionType.SMILEY);
491
492                     tAddQuestionForm = new AddQuestionForm(tQuestion);
493                 }
494                 else if (createdQuestions_ListView.SelectedItems[0].SubItems[1].Tag.ToString() == QuestionType.SLIDER.ToString())
495                 {
496                     tQuestion = new Question(tQuestionId, QuestionType.SLIDER);
497
498                     tAddQuestionForm = new AddQuestionForm(tQuestion);
499                 }
500                 else if (createdQuestions_ListView.SelectedItems[0].SubItems[1].Tag.ToString() == QuestionType.STAR.ToString())
501                 {
502                     tQuestion = new Question(tQuestionId, QuestionType.STAR);
503
504                     tAddQuestionForm = new AddQuestionForm(tQuestion);
505                 }
506
507                 // prevent exception if form was closed due to some error
508                 if (!tAddQuestionForm.IsDisposed)
509                 {
510                     tFormDialogResult = tAddQuestionForm.ShowDialog();
511                 }
512
513                 if (tFormDialogResult == DialogResult.OK)
514                 {
515                     mQuestionManager.InstantlyRefreshList();
516                 }
517             }
518         }
519     }
```

Description:

This event handler handle **2 events**:

1. Edit button click
2. And ListView's row double click

This method checks if an item was selected before editing it, then checks the type of the selected question and based on it creates a new question with the ID and Type of question. And pass it to the form's constructor that is use to also adding a question.

But in this case the form knows it is editing a question not adding one.

connectionSettingsButton_Click

```
530     /// <summary>
531     /// Show change connection string settings form
532     /// </summary>
533     /// <param name="sender"></param>
534     /// <param name="e"></param>
535     private void connectionSettingsButton_Click(object sender, EventArgs e)
536     {
537         try
538         {
539             ConnectionSettingsForm connectionSettingsForm = new ConnectionSettingsForm();
540             DialogResult tResult = connectionSettingsForm.ShowDialog();
541             if (tResult == DialogResult.OK)
542             {
543                 mQuestionManager.InstantlyRefreshList();
544             }
545         }
546         catch (Exception ex)
547         {
548             ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}
549             Logger.LogError(ex);
550         }
551     }
552 }
```

Description:

This event handler handle showing the form which handles changing the connection string settings.

closeApplicationButton_Click

```
553     ///<summary> Handle close button click
554     1 reference
555     private void closeApplicationButton_Click(object sender, EventArgs e)
556     {
557         try
558         {
559             Application.Exit();
560         }
561         catch (Exception ex)
562         {
563             ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.
564             Logger.LogError(ex);
565
566         }
567     }///End event
```

Description:

This event handler handle closing the application.

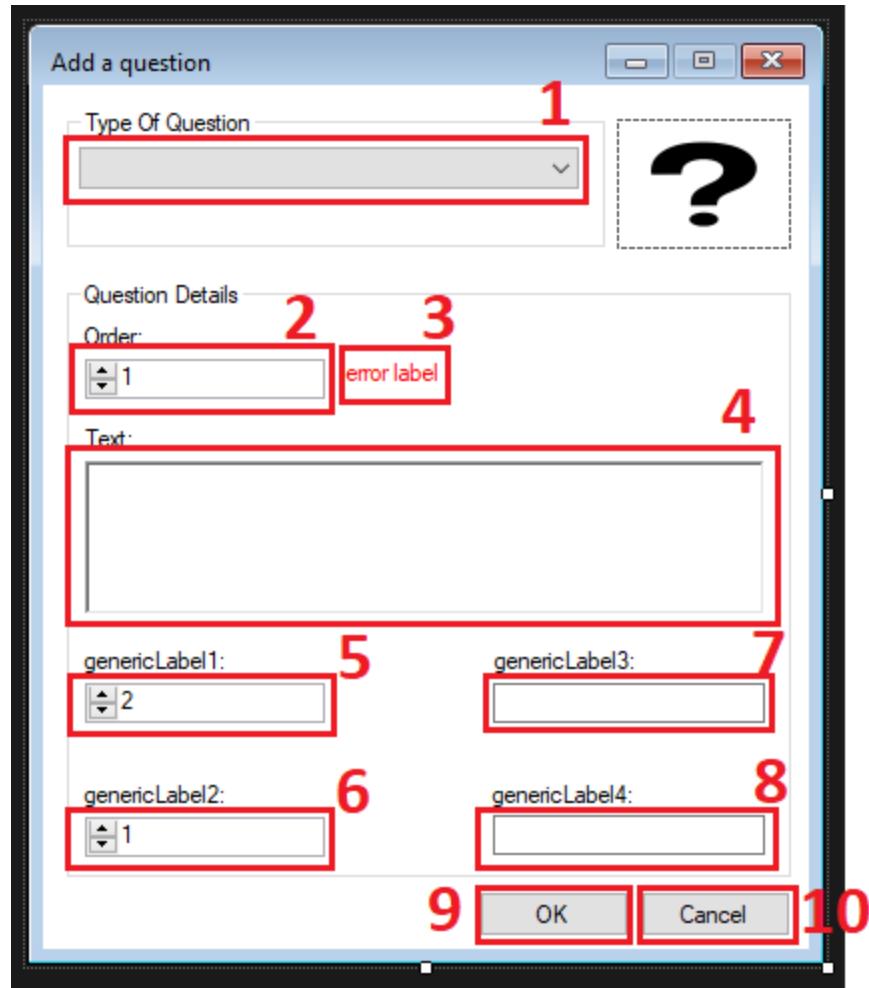
languageSettingsButton_Click

```
569     /// <summary>
570     /// Show language settings form
571     /// </summary>
572     private void languageSettingsButton_Click(object sender, EventArgs e)
573     {
574         try
575         {
576             LangaugeSettingsForm langaugeSettingsForm = new LangaugeSettingsForm();
577             langaugeSettingsForm.ShowDialog();
578         }
579         catch (Exception ex)
580         {
581             ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", Logger.LogError(ex));
582         }
583     }
584
585 #endregion
586
587 }
```

Description:

This event handler handle showing change language form.

1.4.2 AddQuestionForm



The form has total 10 main controls

1. Combo box.
2. questionOrderNumericUpDown.
3. Error label.
4. questionTextRichTextBox.
5. genericNumericUpDown1.
6. genericNumericUpDown2.
7. genericTextBox1.
8. genericTextBox2.
9. Ok/save button
10. Cancel button

Controls Properties:

- Combobox:
 - DropDownStyle-> DropDownList
- Order numeric up down
 - Maximum-> 9999999
- Rich box
 - MaxLength-> 3999
- genericNumericUpDown1
 - Minimun->1
 - Maximum->100
- genericNumericUpDown2
 - Minimun->1
 - Maximum->100
- genericTextBox1
 - MaxLength->99
- genericTextBox2
 - MaxLength->99

Code:

The code is divided into regions based on their functionality.

```
1  +using ...
15
16  -namespace SurveyQuestionsConfigurator
17  {
18      -public partial class AddQuestionForm : Form
19      {
20          +Properties & Attributes
21
22          +Form Constructors
23
24          +Event Handlers
25
26          +Editing Form Initializers
27
28          +ComboBox Selected Index Change Methods
29
30          +Add Question Methods
31
32          +Edit Question Methods
33
34          +Validation Methods
35
36      }
37
38  }
```

The code editor displays a C# class named `AddQuestionForm` within the `SurveyQuestionsConfigurator` namespace. The class is defined as a partial class. The code is annotated with several callout boxes highlighting different sections of the code:

- `Properties & Attributes`: A callout box pointing to the declaration of the class.
- `Form Constructors`: A callout box pointing to the constructor definition.
- `Event Handlers`: A callout box pointing to the event handler declarations.
- `Editing Form Initializers`: A callout box pointing to the initialization logic.
- `ComboBox Selected Index Change Methods`: A callout box pointing to the method handling changes in a ComboBox's selected index.
- `Add Question Methods`: A callout box pointing to methods related to adding questions.
- `Edit Question Methods`: A callout box pointing to methods related to editing questions.
- `Validation Methods`: A callout box pointing to methods responsible for validation.

Properties & Attributes

```
11 references
18  public partial class AddQuestionForm : Form
19  {
20      #region Properties & Attributes
21
22      private readonly QuestionManager mGeneralQuestionManager;
23      private readonly ResourceManager mLocalResourceManager;
24      private readonly CultureInfo mDefaultCulture;
25
26      private Question mGenericQuestion { get; set; } /// create global Question ID property
27
28      private FormStateType cStateForm { get; set; } /// Decide whether "OK" Form button is used to either ADD or EDIT a question
29
30      private QuestionType cSelectedQuestionType { get; set; }
31
32      public enum FormStateType
33      {
34          ADD,
35          EDIT
36      }
37
38      /// <summary> All translatable message box messages in the "AddQuestionFormString ...
39      private enum ResourceStrings
40      {
41          numberOfsmileyFaces,
42          numberOfStars,
43          questionOrderAlreadyInUseError,
44          sliderEndValue,
45          sliderStartValue,
46          somethingWrongHappenedError,
47          startValueCaption,
48          endValueCaption,
49          addAQuestion,
50          contactSystemAdministratorError,
51          editAQuestion,
52          error,
53          questionWasNotFoundOrDeletedError,
54          genericQuestionError,
55          sliderQuestionError
56      }
57
58      #endregion
```

Step 1: instance of QuestionManager (Business logic layer).

Step 2: Generic question to be used in multiple methods/event handles.

Step 3: save the setting of which the form is adding or editing.

Step 4: save the type of the current question.

Form Constructors

AddQuestionForm

```
58  #region Form Constructors
59  /// <summary> Form constructor for "Adding A Question"
60  reference
61  public AddQuestionForm()
62  {
63      try
64      {
65          mGeneralQuestionManager = new QuestionManager();
66          mDefaultCulture = new CultureInfo(ConfigurationManager.AppSettings["DefaultCulture"]);
67          mLocalResourceManager = new ResourceManager("SurveyQuestionsConfigurator.AddQuestionFormStrings", typeof(AddQuestionForm).Assembly);
68      }
69      Thread.CurrentThread.CurrentCulture = mDefaultCulture;
70
71      InitializeComponent();
72      this.Text = mLocalResourceManager.GetString($"{ResourceStrings.addAQuestion}");
73      cStateForm = FormStateType.ADD;
74      errorLabel.Text = "";
75
76      catch (Exception ex)
77      {
78          ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, m
79          Logger.LogError(ex);
80      }
81  }
```

Description:

Initialize the form for adding a question

AddQuestionForm(Question pQuestion):

```
84  /// <summary> Form constructor for "Editing A Question"
85  // 3 references
86  public AddQuestionForm(Question pQuestion)
87  {
88      try
89      {
90          mDefaultCulture = new CultureInfo(ConfigurationManager.AppSettings["DefaultCulture"]);
91          Thread.CurrentThread.CurrentUICulture = mDefaultCulture;
92          mLocalResourceManager = new ResourceManager("SurveyQuestionsConfigurator.AddQuestionFormStrings", typeof(SurveyQuestionsConfiguratorForm).Assembly);
93
94          InitializeComponent();
95          mGeneralQuestionManager = new QuestionManager();
96          this.Text = mLocalResourceManager.GetString("${ResourceStrings.editAQuestion}");
97          cStateForm = FormStateType.EDIT;
98          errorLabel.Text = "";
99          mGenericQuestion = pQuestion;
100         questionTypeComboBox.Enabled = false;
101
102         if (pQuestion.Type == QuestionType.SMILEY)
103         {
104             cSelectedQuestionType = QuestionType.SMILEY; // 0
105             InitializeEditingSmileyQuestion(pQuestion);
106         }
107         else if (pQuestion.Type == QuestionType.SLIDER)
108         {
109             cSelectedQuestionType = QuestionType.SLIDER; // 1
110             InitializeEditingSliderQuestion(pQuestion);
111         }
112         else if (pQuestion.Type == QuestionType.STAR)
113         {
114             cSelectedQuestionType = QuestionType.STAR; // 2
115             InitializeEditingStarQuestion(pQuestion);
116         }
117     }
118
119     catch (Exception ex)
120     {
121         ShowMessage.Box("${ResourceStrings.somethingWrongHappenedError}", "${ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResourceManager);
122         Logger.LogError(ex);
123     }
124 }
125 #endregion
```

Description:

Initialize the form for editing a selected question

Step 1: disable changing current edited question type.

Step 2: Load the settings of the passed question based on its ID and Type

Event Handlers

AddQuestionForm_Load

```
127     #region Event Handlers
128     /// <summary>
129     /// Handle form's load and load correct question type based what question type is passed to the constructor
130     /// </summary>
131     private void AddQuestionForm_Load(object sender, EventArgs e)
132     {
133         try
134         {
135             questionTypeComboBox.SelectedIndex = (int)cSelectedQuestionType;
136         }
137         catch (Exception ex)
138         {
139             ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK,
140             MessageBoxIcon.Error);
141             Logger.LogError(ex);
142         }
143     } // End event
144 }
```

Description:

Sets combo box selected index

cancelButton_Click

```
144  
145     /// <summary> Handle close click  
146     1 reference  
147     private void cancelButton_Click(object sender, EventArgs e)  
148     {  
149         try  
150         {  
151             this.Close();  
152         }  
153         catch (Exception ex)  
154         {  
155             ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStri  
156             Logger.LogError(ex);  
157         }  
158     }  
159 } // End event
```

Description:

Close the application

okButton_Click

```
161     ///<summary>
162     /// Handle saving settings based on what the form is doing (Adding or Editing)
163     /// </summary>
164     private void okButton_Click(object sender, EventArgs e)
165     {
166         try
167         {
168             /// Close form if operation was successful
169             ErrorCode OperationSuccess = ErrorCode.ERROR;
170             if (cStateForm == FormStateType.ADD)
171             {
172                 try
173                 {
174                     if (questionTypeComboBox.SelectedIndex == 0)
175                     {
176                         OperationSuccess = InsertSmileyQuestion();
177                     }
178                     else if (questionTypeComboBox.SelectedIndex == 1)
179                     {
180                         OperationSuccess = InsertSliderQuestion();
181                     }
182                     else if (questionTypeComboBox.SelectedIndex == 2)
183                     {
184                         OperationSuccess = InsertStarQuestion();
185                     }
186                 }
187                 catch (Exception ex)
188                 {
189                     Logger.LogError(ex);
190                     throw;
191                 }
192             }
193             else if (cStateForm == FormStateType.EDIT)
194             {
195                 try
196                 {
197                     if (questionTypeComboBox.SelectedIndex == 0)
198                     {
199                         OperationSuccess = UpdateSmileyQuestion();
200                     }
201                     else if (questionTypeComboBox.SelectedIndex == 1)
202                     {
203                         OperationSuccess = UpdateSliderQuestion();
204                     }
205                     else if (questionTypeComboBox.SelectedIndex == 2)
206                     {
207                         OperationSuccess = UpdateStarQuestion();
208                     }
209                 }
210                 catch (Exception ex)
211                 {
212                     Logger.LogError(ex);
213                     throw;
214                 }
215             }
216         }
217
218         if (OperationSuccess == ErrorCode.SUCCESS)
219         {
220             this.Close();
221             this.DialogResult = DialogResult.OK;
222         }
223     }
224     catch (Exception ex)
225     {
226         ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK);
227         Logger.LogError(ex);
228     }
229 }
```

Description:

If cStateForm = Add, it attempts a DB insert operation

If cStateForm = Edit, it attempts DB update operation

And saved the operation result on in a variable.

If it was successful-> close the form and return OK as dialog result to the parent form

Editng Form Intializers

LoadSmileyQuestion, LoadSliderQuestion & LoadStarQuestion

```
233     #region Editng Form Intializers
234
235     /// <summary> Initialize editing smiley question on combobox index change event
236     private void LoadSmileyQuestion(Question pQuestion)
237     {
238         /// Get selected question from DB and fill input fields with its data
239         try
240         {
241             mGenericQuestion = pQuestion;
242             SmileyQuestion tSmileyQuestion = new SmileyQuestion(pQuestion.ID);
243
244             ErrorCode tResult = mGeneralQuestionManager.GetSmileyQuestionByID(ref tSmileyQuestion);
245             switch (tResult)
246             {
247                 case ErrorCode.SUCCESS:
248                     questionOrderNumericUpDown.Value = Convert.ToDecimal(tSmileyQuestion.Order);
249                     questionTextRichTextBox.Text = tSmileyQuestion.Text.ToString();
250                     genericNumericUpDown1.Value = Convert.ToDecimal(tSmileyQuestion.NumberOfSmileyFaces);
251                     break;
252
253                 case ErrorCode.VALIDATION:
254                     ShowMessage.Box(${ResourceStrings.questionWasNotFoundOrDeletedError}, ${ResourceStrings.error}, MessageBoxButtons.OK);
255                     this.Close();
256                     break;
257
258                 default:
259                     ShowMessage.Box(${ResourceStrings.contactSystemAdministratorError}, ${ResourceStrings.error}, MessageBoxButtons.OK);
260                     this.Close();
261                     break;
262             }
263         }
264         catch (Exception ex)
265         {
266             Logger.LogError(ex); //write error to log file
267             throw;
268         }
269     } // End function
270
271 }
```

```

273     /// <summary> Initialize editing slider question on combobox index change event
274     private void LoadSliderQuestion(Question pQuestion)
275     {
276         ///| Get selected question from DB and fill input fields with its data
277         try
278         {
279             mGenericQuestion = pQuestion;
280             SliderQuestion tSliderQuestion = new SliderQuestion(pQuestion.ID);
281
282             ErrorCode result = mGeneralQuestionManager.GetSliderQuestionByID(ref tSliderQuestion);
283             switch (result)
284             {
285                 case ErrorCode.SUCCESS:
286                     questionOrderNumericUpDown.Value = Convert.ToDecimal(tSliderQuestion.Order);
287                     questionTextRichTextBox.Text = tSliderQuestion.Text.ToString();
288                     genericNumericUpDown1.Value = Convert.ToDecimal(tSliderQuestion.StartValue);
289                     genericNumericUpDown2.Value = Convert.ToDecimal(tSliderQuestion.EndValue);
290                     genericTextBox1.Text = tSliderQuestion.StartValueCaption.ToString();
291                     genericTextBox2.Text = tSliderQuestion.EndValueCaption.ToString();
292                     break;
293                 case ErrorCode.VALIDATION:
294                     ShowMessage.Box(${ResourceStrings.contactSystemAdministratorError}, ${ResourceStrings.error}, MessageBoxButtons.OKCancel);
295                     this.Close();
296                     break;
297                 default:
298                     ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OKCancel);
299                     this.Close();
300                     break;
301             }
302         }
303         catch (Exception ex)
304         {
305             Logger.LogError(ex); //write error to log file
306             throw;
307         }
308     } // End function
309
310
311     /// <summary> Initialize editing star question on combobox change
312     private void LoadStarQuestion(Question pQuestion)
313     {
314         /// Select wanted question from DB and fill input fields with its data
315         try
316         {
317             mGenericQuestion = pQuestion;
318             StarQuestion tStarQuestion = new StarQuestion(pQuestion.ID);
319
320             ErrorCode result = mGeneralQuestionManager.GetStarQuestionByID(ref tStarQuestion);
321             switch (result)
322             {
323                 case ErrorCode.SUCCESS:
324                     questionOrderNumericUpDown.Value = Convert.ToDecimal(tStarQuestion.Order);
325                     questionTextRichTextBox.Text = tStarQuestion.Text.ToString();
326                     genericNumericUpDown1.Value = Convert.ToDecimal(tStarQuestion.NumberOfStars);
327                     break;
328                 case ErrorCode.VALIDATION:
329                     ShowMessage.Box(${ResourceStrings.contactSystemAdministratorError}, ${ResourceStrings.error}, MessageBoxButtons.OKCancel);
330                     this.Close();
331                     break;
332                 default:
333                     ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OKCancel);
334                     this.Close();
335                     break;
336             }
337         }
338         catch (Exception ex)
339         {
340             Logger.LogError(ex); //write error to log file
341             throw;
342         }
343     } // End function
344
345
346

```

Description:

All of three methods load question details to be edited and fill the appropriate controls with data.

ComboBox Selected Index Change Methods

questionTypeComboBox_SelectedIndexChanged

```
349  #region ComboBox Selected Index Change Methods
350
351  /// <summary> Handle combo box selected index changed
352  1 reference
353  private void questionTypeComboBox_SelectedIndexChanged(object sender, EventArgs e)
354  {
355      try
356      {
357          if (questionTypeComboBox.SelectedIndex == 0)
358          {
359              InitializeEditingSmileyQuestion();
360          }
361          else if (questionTypeComboBox.SelectedIndex == 1)
362          {
363              InitializeEditingSliderQuestion();
364          }
365          else if (questionTypeComboBox.SelectedIndex == 2)
366          {
367              InitializeEditingStarQuestion();
368          }
369      }
370      catch (Exception ex)
371      {
372          ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK);
373          Logger.LogError(ex);
374      }
375  }/// End event
```

Description:

Change the form visible controls and label's text based on what question the user want to add.

Add Question Methods

InsertSmileyQuestion, InsertSliderQuestion & InsertStarQuestion

```
469     #region Add Question Methods
470     ///<summary>
471     /// Handle adding a question
472     /// Create a question object and pass to (Business Logic Layer)
473     ///</summary>
474     [reference]
475     private ErrorCode InsertSmileyQuestion()
476     {
477         try
478         {
479             int tQuestionOrder, tNumberOfSmileyFaces;
480             string tQuestionText;
481             ErrorCode tResult;
482
483             tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
484             tQuestionText = questionTextRichTextBox.Text;
485             tNumberOfSmileyFaces = Convert.ToInt32(genericNumericUpDown1.Value);
486
487             /// Pass -1 as ID because ID is not needed here & there is no constructor that do not accept ID
488             SmileyQuestion tSmileyQuestion = new SmileyQuestion(-1, tQuestionOrder, tQuestionText, QuestionType.SMILEY, tNumberOfSmileyFaces);
489
490             /// Try to insert a new question into "Questions" and "Smiley_Questions" tables in DB
491             if (CheckSmileyQuestionInputFields(tSmileyQuestion) == ErrorCode.SUCCESS) //if Question text is not null or empty
492             {
493                 tResult = mGeneralQuestionManager.InsertSmileyQuestion(tSmileyQuestion);
494
495                 switch (tResult)
496                 {
497                     case ErrorCode.SUCCESS:
498                         errorLabel.Text = "";
499                         return ErrorCode.SUCCESS;
500                     case ErrorCode.VALIDATION:
501                         errorLabel.Text = mLocalResourceManager.GetString("${ResourceStrings.questionOrderAlreadyInUseError}");
502                         break;
503                     default:
504                         errorLabel.Text = "";
505                         ShowMessage.Box("${ResourceStrings.somethingWrongHappenedError}", "${ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error);
506                         break;
507                 }
508             }
509             else
510             {
511                 ShowMessage.Box("${ResourceStrings.genericQuestionError}", "${ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error);
512             }
513         }
514     }
```

```

524     /// <summary>
525     /// Handle adding a question
526     /// Create a question object and pass to (Business Logic Layer)
527     /// </summary>
528     private ErrorCode InsertSliderQuestion()
529     {
530         try
531         {
532             int tQuestionOrder, tQuestionStartValue, tQuestionEndValue;
533             string tQuestionText, tQuestionStartValueCaption, tQuestionEndValueCaption;
534             ErrorCode tResult;
535             tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
536             tQuestionText = (string)questionTextRichTextBox.Text;
537             tQuestionStartValueCaption = (string)genericTextBox1.Text;
538             tQuestionEndValueCaption = (string)genericTextBox2.Text;
539             tQuestionStartValue = Convert.ToInt32(genericNumericUpDown1.Value);
540             tQuestionEndValue = Convert.ToInt32(genericNumericUpDown2.Value);

541             /// Pass -1 as ID because ID is not needed here & there is no constructor that do not accept ID
542             SliderQuestion tSliderQuestion = new SliderQuestion(-1, tQuestionOrder, tQuestionText, QuestionType.SLIDER, tQuestionStartVa
543
544             if (CheckSliderQuestionInputFields(tSliderQuestion) == ErrorCode.SUCCESS)
545             {
546                 /// Try to insert a new question into "Questions" and "Slider_Questions" tables in DB
547                 tResult = mGeneralQuestionManager.InsertSliderQuestion(tSliderQuestion);
548
549                 switch (tResult)
550                 {
551                     case ErrorCode.SUCCESS:
552                         errorLabel.Text = "";
553                         return ErrorCode.SUCCESS;
554                     case ErrorCode.VALIDATION:
555                         errorLabel.Text = mLocalResourceManager.GetString("${ResourceStrings.questionOrderAlreadyInUseError}");
556                         break;
557                     default:
558                         errorLabel.Text = "";
559                         ShowMessage.Box("${ResourceStrings.somethingWrongHappenedError}", "${ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error);
560                         break;
561                 }
562             }
563             else
564             {
565                 ShowMessage.Box("${ResourceStrings.sliderQuestionError}", "${ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error);
566             }
567         }
568     }

```

```

578     /// <summary> Handle adding a question Create a question object and pass to (Bus ...
579     private ErrorCode InsertStarQuestion()
580     {
581         try
582         {
583             int tQuestionOrder, tNumberOfStars;
584             string tQuestionText;
585             ErrorCode tResult;
586
587             tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
588             tQuestionText = questionTextRichTextBox.Text;
589             tNumberOfStars = Convert.ToInt32(genericNumericUpDown1.Value);

590             /// Pass -1 as ID because ID is not needed here & there is no constructor that do not accept ID
591             StarQuestion tStarQuestion = new StarQuestion(-1, tQuestionOrder, tQuestionText, QuestionType.STAR, tNumberOfStars);

592             /// Try to insert a new question into "Questions" and "Star_Questions" tables in DB
593             if (CheckStarQuestionInputFields(tStarQuestion) == ErrorCode.SUCCESS) // If question input fields are not null or empty
594             {
595                 tResult = mGeneralQuestionManager.InsertStarQuestion(tStarQuestion);
596
597                 switch (tResult)
598                 {
599                     case ErrorCode.SUCCESS:
600                         errorLabel.Text = "";
601                         return ErrorCode.SUCCESS;
602                     case ErrorCode.VALIDATION:
603                         errorLabel.Text = mLocalResourceManager.GetString("${ResourceStrings.questionOrderAlreadyInUseError}");
604                         break;
605                     default:
606                         errorLabel.Text = "";
607                         ShowMessage.Box("${ResourceStrings.somethingWrongHappenedError}", "${ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error);
608                         break;
609                 }
610             }
611             else
612             {
613                 ShowMessage.Box("${ResourceStrings.genericQuestionError}", "${ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error);
614             }
615         }
616     }
617     return ErrorCode.ERROR;
618 }
619 }
620 }

```

Description:

These methods get data from controls.

And create a corresponding question object with ID = -1.

Then check if the values are valid and pass it to BLL and try to insert it in the database.

Returns:

ErrorCode based on the operation.

Edit Question Methods

UpdateSmileyQuestion, UpdateSliderQuestion & UpdateStarQuestion:

```
632     #region Edit Question Methods
633     /// <summary>
634     /// Handle editing a question
635     /// Create a question object and pass to (Business Logic Layer)
636     /// </summary>
637     [reference]
638     private ErrorCode UpdateSmileyQuestion()
639     {
640         try
641         {
642             int tQuestionId, tQuestionOrder, tNumberOfSmilyFaces;
643             string tQuestionText;
644             ErrorCode tResult;
645
646             tQuestionId = mGenericQuestion.ID;
647             tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
648             tQuestionText = questionTextRichTextBox.Text;
649             tNumberOfSmilyFaces = Convert.ToInt32(genericNumericUpDown1.Value);
650
651             SmileyQuestion tSmileyQuestion = new SmileyQuestion(tQuestionId, tQuestionOrder, tQuestionText, QuestionType.SMILEY, tNumberOfSmilyFaces);
652
653             if (CheckSmileyQuestionInputFields(tSmileyQuestion) == ErrorCode.SUCCESS)
654             {
655                 /// Try to Update a new question into "Questions" and "Smiley_Questions" tables in DB
656                 tResult = mGeneralQuestionManager.UpdateSmileyQuestion(tSmileyQuestion);
657
658                 switch (tResult)
659                 {
660                     case ErrorCode.SUCCESS:
661                         errorLabel.Text = "";
662                         return ErrorCode.SUCCESS;
663                     case ErrorCode.VALIDATION:
664                         errorLabel.Text = mLocalResourceManager.GetString("${ResourceStrings.questionOrderAlreadyInUseError}");
665                         break;
666                     default:
667                         errorLabel.Text = "";
668                         ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error);
669                         break;
670                 }
671             }
672             else
673             {
674                 ShowMessage.Box(${ResourceStrings.genericQuestionError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResourceManager.GetString("${ResourceStrings.errorTitle}"));
675             }
676         }
677         return ErrorCode.ERROR;
678     }
```

```
685     /// <summary>
686     /// Handle editing a question
687     /// Create a question object and pass to (Business Logic Layer)
688     /// </summary>
689     [reference]
690     private ErrorCode UpdateSliderQuestion()
691     {
692         try
693         {
694             int tQuestionId, tQuestionOrder, tQuestionStartValue, tQuestionEndValue;
695             string tQuestionText, tQuestionStartValueCaption, tQuestionEndValueCaption;
696             ErrorCode tResult;
697
698             tQuestionId = mGenericQuestion.ID;
699             tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
700             tQuestionText = (string)questionTextRichTextBox.Text;
701             tQuestionStartValue = Convert.ToInt32(genericNumericUpDown1.Value);
702             tQuestionEndValue = Convert.ToInt32(genericNumericUpDown2.Value);
703             tQuestionStartValueCaption = (string)genericTextBox1.Text;
704             tQuestionEndValueCaption = (string)genericTextBox2.Text;
705
706             SliderQuestion tSliderQuestion = new SliderQuestion(tQuestionId, tQuestionOrder, tQuestionText, QuestionType.SLIDER, tQuestionStartValue, tQuestionEndValue);
707
708             if (CheckSliderQuestionInputFields(tSliderQuestion) == ErrorCode.SUCCESS)
709             {
710                 /// Try to Update a new question into "Questions" and "Slider_Questions" tables in DB
711                 tResult = mGeneralQuestionManager.UpdateSliderQuestion(tSliderQuestion);
712
713                 switch (tResult)
714                 {
715                     case ErrorCode.SUCCESS:
716                         errorLabel.Text = "";
717                         return ErrorCode.SUCCESS;
718                     case ErrorCode.VALIDATION:
719                         errorLabel.Text = mLocalResourceManager.GetString("${ResourceStrings.questionOrderAlreadyInUseError}");
720                         break;
721                     default:
722                         errorLabel.Text = "";
723                         ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error);
724                         break;
725                 }
726             }
727             else
728             {
729                 ShowMessage.Box(${ResourceStrings.sliderQuestionError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResourceManager.GetString("${ResourceStrings.errorTitle}"));
730             }
731         }
732         return ErrorCode.ERROR;
733     }
```

```

740 //////////////////////////////////////////////////////////////////
741 // Handle editing a question Create a question object and pass to (Bu ...
742 //////////////////////////////////////////////////////////////////
743
744 private ErrorCode UpdateStarQuestion()
745 {
746     try
747     {
748         int tQuestionId, tQuestionOrder, tNumberOfStars;
749         string tQuestionText;
750         ErrorCode result;
751
752         tQuestionId = mGenericQuestion.ID;
753         tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
754         tQuestionText = questionTextRichTextBox.Text;
755         tNumberOfStars = Convert.ToInt32(genericNumericUpDown1.Value);
756
757         StarQuestion tStarQuestion = new StarQuestion(tQuestionId, tQuestionOrder, tQuestionText, QuestionType.STAR, tNumberOfStars);
758
759         if (CheckStarQuestionInputFields(tStarQuestion) == ErrorCode.SUCCESS)
760         {
761             // Try to Update a new question into "Questions" and "Star_Questions" tables in DB
762             result = mGeneralQuestionManager.UpdateStarQuestion(tStarQuestion);
763
764             switch (result)
765             {
766                 case ErrorCode.SUCCESS:
767                     errorLabel.Text = "";
768                     return ErrorCode.SUCCESS;
769                 case ErrorCode.VALIDATION:
770                     errorLabel.Text = mLocalResourceManager.GetString("${ResourceStrings.questionOrderAlreadyInUseError}");
771                     break;
772                 default:
773                     errorLabel.Text = "";
774                     ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error);
775                     break;
776             }
777         }
778         else
779         {
780             ShowMessage.Box(${ResourceStrings.genericQuestionError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResou
781         }
782     }
783     catch (Exception ex)
784     {

```

Description:

These methods get data from controls.

And create a corresponding question object with the appropriate ID.

Then check if the values are valid and pass it to BLL and try to update it in the database.

Returns:

ErrorCode based on the operation.

Validation Methods

```
793     #region Validation Methods
794
795     /// <summary> Check smiley question input fields
796     2 references
797     private ErrorCode CheckSmileyQuestionInputFields(SmileyQuestion pSmileyQuestion)
798     {
799
800         try
801         {
802             return mGeneralQuestionManager.CheckSmileyQuestionValues(pSmileyQuestion);
803         }
804         catch (Exception ex)
805         {
806             ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK,
807             Logger.LogError(ex);
808             return ErrorCode.ERROR;
809         }
810     }/// Function end
811
812     /// <summary> Check slider question input fields
813     2 references
814     private ErrorCode CheckSliderQuestionInputFields(SliderQuestion pSliderQuestion)
815     {
816
817         try
818         {
819             return mGeneralQuestionManager.CheckSliderQuestionValues(pSliderQuestion);
820         }
821         catch (Exception ex)
822         {
823             ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK,
824             Logger.LogError(ex);
825             return ErrorCode.ERROR;
826         }
827     }/// Function end
828
829     /// <summary> Check star question input fields
830     2 references
831     private ErrorCode CheckStarQuestionInputFields(StarQuestion pStarQuestion)
832     {
833
834         try
835         {
836             return mGeneralQuestionManager.CheckStarQuestionValues(pStarQuestion);
837         }
838         catch (Exception ex)
839         {
840             ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK,
841             Logger.LogError(ex);
842             return ErrorCode.ERROR;
843         }
844     }/// Function end
845     #endregion
```

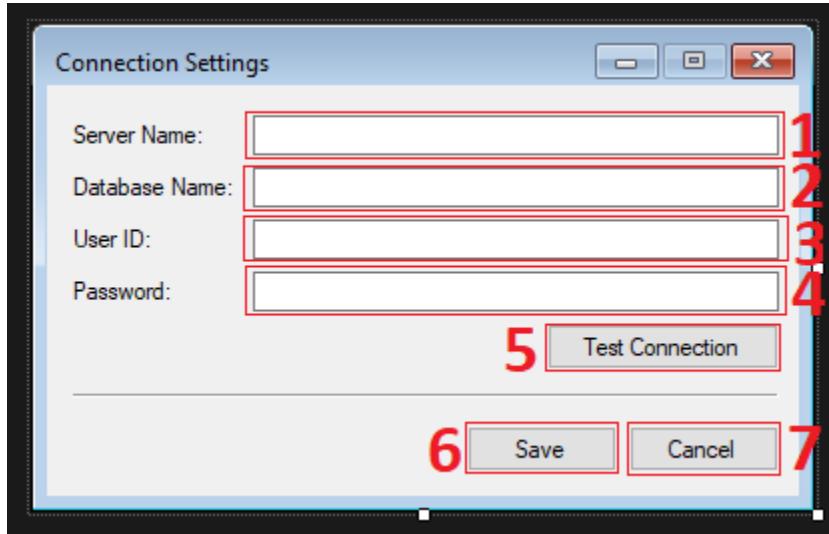
Description:

Check the fields and input of each question type.

Returns:

ErrorCode based if the data are valid or not.

1.4.3 ConnectionSettingsForm



The form has total 7 main controls

1. Server name text box.
2. Database name text box.
3. User ID text box.
4. Password text box.
5. Test connection button.
6. Save settings button.
7. Cancel button.

Control's Properties:

All text boxes have MaxLength = 128

Code:

The code is divided into regions based on their functionality.

A screenshot of a code editor displaying a C# file. The code defines a partial class `ConnectionSettingsForm` within the `SurveyQuestionsConfigurator` namespace. The class has several regions highlighted by dashed boxes:

- `Properties & Attributes`
- `Constructor`
- `Event Handlers`
- `Validation methods`
- `General Methods`

```
1  +using ...
18
19  -namespace SurveyQuestionsConfigurator
20  {
21      -public partial class ConnectionSettingsForm : Form
22      {
23          +    Properties & Attributes
24
25          +    Constructor
26
27          +    Event Handlers
28
29          +    Validation methods
30
31          +    General Methods
32
33      }
34
35  }
```

Properties & Attributes

```
23  | #region Properties & Attributes
24  |
25  |     private readonly ResourceManager mLocalResourceManager;
26  |     private readonly CultureInfo mDefaultCulture;
27  |     private SqlConnectionStringBuilder mBuilder; // passed to ConnectionSettingsManager (Business Logic Layer)
28  |     private readonly ConnectionSettingsManager mConnectionSettingsManager;
29  |
30  |     /// <summary>
31  |     /// All translatable message box messages in the "ConnectionSettingsFormStrings" resource file
32  |     /// </summary>
33  |     private enum ResourceStrings
34  |     {
35  |         somethingWrongHappenedError,
36  |         couldNotSaveSettingsError,
37  |         testConnectionSucceeded,
38  |         testConnectionFailedError,
39  |         emptyInputFieldsError,
40  |         error,
41  |         success
42  |     }
43  | #endregion
```

Constructor

ConnectionSettingsForm

```
45  #region Constructor
46  1reference
47  public ConnectionSettingsForm()
48  {
49      try
50      {
51          mBuilder = new SqlConnectionStringBuilder();
52          mConnectionSettingsManager = new ConnectionSettingsManager();
53          mLocalResourceManager = new ResourceManager("SurveyQuestionsConfigurator.ConnectionSettingsFormStrings", typeof(ConnectionSettingsForm).Assembly);
54          mDefaultCulture = new CultureInfo(ConfigurationManager.AppSettings["DefaultCulture"]);
55
56          Thread.CurrentThread.CurrentUICulture = mDefaultCulture;
57
58          InitializeComponent();
59      }
60      catch (Exception ex)
61      {
62          ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalRes
63          Logger.LogError(ex);
64      }
65  }
66 #endregion
```

Step 1: Gets the default UI culture from config file and sets it to the current thread's CurrentUICulture property

Event Handlers

ConnectionSettingsForm_Load:

```
67
68     #region Event Handlers
69
70     /// <summary>
71     /// Get current connection string details from config file, save it to the generic mBuilder and fill the form with it
72     /// </summary>
73     private void ConnectionSettingsForm_Load(object sender, EventArgs e)
74     {
75         try
76         {
77             mBuilder = mConnectionSettingsManager.GetConnectionString();
78
79             LoadCurrentConnectionStringSettings();
80         }
81         catch (Exception ex)
82         {
83             ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK,
84             Logger.LogError(ex);
85         }
86     }
87
```

Description:

Get current connection string details from config file, save it to the generic mBuilder and fill the form with it

cancelButton_Click:



```
    /// <summary>
    /// Close the form
    /// </summary>
    private void cancelButton_Click(object sender, EventArgs e)
    {
        try
        {
            this.Close();
        }
        catch (Exception ex)
        {
            ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK,
Logger.LogError(ex);
        }
    }
}
```

Description:

Closes the form

testConnectionButton_Click:

```
104     /// <summary>
105     /// Tests the connection of the inputs
106     /// </summary>
107     private void testConnectionButton_Click(object sender, EventArgs e)
108     {
109         try
110         {
111             mBuilder.Clear();
112             FillConnectionStringBuilderFields();
113
114             if (CheckConnectionStringInputFields(mBuilder) == ErrorCode.SUCCESS) // if the connection string fields are valid
115             {
116                 if (mConnectionStringManager.CheckConnectivity(mBuilder) == ErrorCode.SUCCESS) // check actual connectivity
117                 {
118                     ShowMessage.Box(${ResourceStrings.testConnectionSucceeded}, ${ResourceStrings.success}, MessageBoxButtons.OK, MessageBoxIcon.Information, mLocalResources);
119                 }
120                 else
121                 {
122                     ShowMessage.Box(${ResourceStrings.testConnectionFailedError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResources);
123                 }
124             }
125             else
126             {
127                 ShowMessage.Box(${ResourceStrings.emptyInputFieldsError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResources);
128             }
129         }
130         catch (Exception ex)
131         {
132             ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResources);
133             Logger.LogError(ex);
134         }
135     }
136 }
```

Description:

Fill the mBuilder with corresponding inputs and text them

saveButton_Click

```
137
138     /// <summary>
139     /// Saves the current inputs to the config file under "connectionStrings" section
140     /// </summary>
141     private void saveButton_Click(object sender, EventArgs e)
142     {
143         try
144         {
145             FillConnectionStringBuilderFields();
146
147             ErrorCode isSaved = mConnectionSettingsManager.SaveConnectionString(mBuilder);
148             if (isSaved == ErrorCode.SUCCESS)
149             {
150                 this.Close();
151                 this.DialogResult = DialogResult.OK;
152             }
153             else
154             {
155                 ShowMessage.Box(${ResourceStrings.couldNotSaveSettingsError}, ${ResourceStrings.error}, MessageBoxButtons.OK,
156             }
157         }
158         catch (Exception ex)
159         {
160             ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error);
161             Logger.LogError(ex);
162         }
163     }
164 }
```

Description:

Fill the mBuilder save its fields to the config file.

Validation methods

CheckConnectionStringInputFields:

```
167  #region Validation methods
168
169  /// <summary> Check connection string fields
170  /// reference
171  private ErrorCode CheckConnectionStringInputFields(SqlConnectionStringBuilder pBuilder)
172  {
173      try
174      {
175          return mConnectionSettingsManager.CheckConnectionStringInputFields(pBuilder);
176      }
177      catch (Exception ex)
178      {
179          ShowMessage.Box("${ResourceStrings.somethingWrongHappenedError}", "${ResourceStrings.error}", MessageBoxButtons.OK,
180          Logger.LogError(ex);
181          return ErrorCode.ERROR;
182      }
183  }
184
185  #endregion
```

Description:

Check the fields in of the mBuilder in the Business Logic Layer

Returns:

ErrorCode

General Methods

FillConnectionStringBuilderFields:

```
192     #region General Methods
193
194     public void FillConnectionStringBuilderFields()
195     {
196         try
197         {
198             mBuilder.DataSource = dataSourceTextBox.Text.ToString();
199             mBuilder.InitialCatalog = initialCatalogTextBox.Text.ToString();
200             mBuilder.UserID = userIDTextBox.Text.ToString();
201             mBuilder.Password = passwordTextBox.Text.ToString();
202         }
203         catch (Exception ex)
204         {
205             Logger.LogError(ex);
206             throw;
207         }
208     }
```

Description:

Fill mBuilder with corresponding text boxes values.

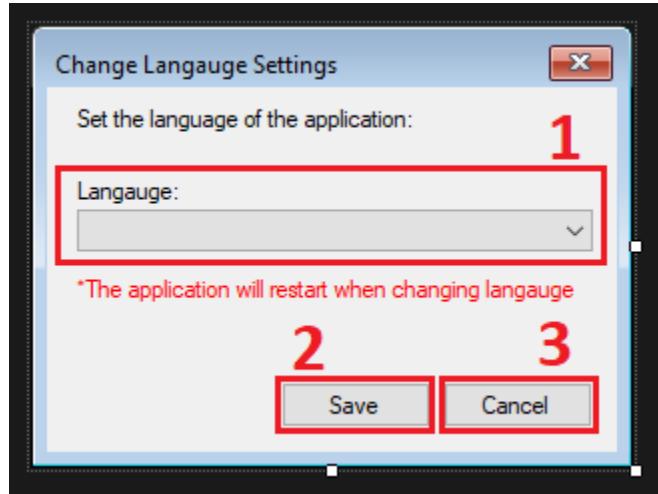
FillConnectionStringBuilderFields:

```
1 reference
210     public void LoadCurrentConnectionStringSettings()
211     {
212         try
213         {
214             dataSourceTextBox.Text = mBuilder.DataSource;
215             initialCatalogTextBox.Text = mBuilder.InitialCatalog;
216             userIDTextBox.Text = mBuilder.UserID;
217             passwordTextBox.Text = mBuilder.Password;
218         }
219         catch (Exception ex)
220         {
221             Logger.LogError(ex);
222             throw;
223         }
224     }
225 }
```

Description:

Fill input text boxes with corresponding data from mBuilder.

1.4.4 LangaugeSettingsForm



The form has total 3 main controls

1. Language combo box.
2. Save button.
3. Cancel button.

Control's Properties:

- languageComboBox
 - DropDownList

Code:

The code is divided into regions based on their functionality.

A screenshot of a code editor displaying a C# class definition. The code is as follows:

```
20  public partial class LangaugeSettingsForm : Form
21  {
22
23      [+] Properties & Attributes
24
25      [+] Constructor
26
27      [+] Event Handlers
28
29      [+] Generic Methods
30  }
```

The code is annotated with several regions:

- Properties & Attributes
- Constructor
- Event Handlers
- Generic Methods

A vertical dashed line on the left side of the code editor separates the numbered lines from the annotated regions. The regions are enclosed in boxes with plus signs (+) indicating they are expandable.

Properties & Attributes

```
22
23     #region Properties & Attributes
24
25     private readonly ResourceManager mLocalResourceManager;
26     private readonly CultureInfo mDefaultCulture;
27
28     private ComboBoxLanguages mLoadedComboBoxLanguage; // used to check if selected language is changed
29     private readonly string mConfigEnglishLangauge;
30     private readonly string mConfigArabicLangauge;
31
32     private readonly Configuration tConfigFile;
33     private readonly KeyValueConfigurationCollection tSettings;
34
35     private readonly string mDefaultCultureString;
36
37     /// <summary>
38     /// All translatable message box messages in the "LanguageSettingsFormStrings" resource file
39     /// </summary>
40     16 references
41     private enum ResourceStrings
42     {
43         somethingWrongHappenedError,
44         error,
45         success,
46         errorSavingSettings
47     }
48     +    /// <summary> Languages to be set to or get from combo box
49     11 references
50     private enum ComboBoxLanguages
51     {
52         English = 0,
53         Arabic = 1,
54         NoLanguage = -1
55     }
56
```

Step 1: saves loaded language settings.

Step 2: access config file.

Step 3: xml attribute to be saved.

Constructor

LangaugeSettingsForm:

```
60  #region Constructor
61  public LangaugeSettingsForm()
62  {
63      try
64      {
65          mLocalResourceManager = new ResourceManager("SurveyQuestionsConfigurator.LanguageSettingsFormStrings", typeof(LangaugeSettingsForm).Assembly);
66          mDefaultCulture = new CultureInfo(ConfigurationManager.AppSettings["DefaultCulture"]);
67
68          mLoadedComboBoxLanguage = ComboBoxLanguages.NoLanguage;
69          mConfigEnglishLangauge = "en-US";
70          mConfigArabicLangauge = "ar-JO";
71
72          tConfigFile = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
73          tSettings = tConfigFile.AppSettings.Settings;
74          mDefaultCultureString = "DefaultCulture";
75
76          Thread.CurrentThread.CurrentCulture = mDefaultCulture;
77
78          InitializeComponent();
79      }
80      catch (Exception ex)
81      {
82          ShowMessage.Box(${ResourceStrings.somethingWrongHappenedError}, ${ResourceStrings.error}, MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResourceManager);
83          Logger.LogError(ex);
84      }
85  }
86
87  #endregion
```

Setup the form's controls and strings to be used later.

Event Handlers

cancelButton_Click

```
91  /// <summary> Close this form
92  /// <reference>
93  ///
94  private void cancelButton_Click(object sender, EventArgs e)
95  {
96      try
97      {
98          this.Close();
99      }
100     catch (Exception ex)
101     {
102         ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK,
103         Logger.LogError(ex);
104     }
105 }
```

Description:

This event handler closes the form

LangaugeSettingsForm_Load

```
107  /// <summary> Load current langauge settings to form
108  // 1 reference
109  private void LangaugeSettingsForm_Load(object sender, EventArgs e)
110  {
111      try
112      {
113          LoadLangaugeSettings();
114      }
115      catch (Exception ex)
116      {
117          ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}",
118          Logger.LogError(ex);
119      }
120  }
```

Description:

This event handler load saved language settings to the form (language combo box)

saveButton_Click:

```
123  /// <summary>
124  /// Handle saving new language settings
125  /// Restart application if new language is saved other than current application's language
126  /// </summary>
127  private void saveButton_Click(object sender, EventArgs e)
128  {
129      try
130      {
131          ComboBoxLanguages tCurrentSelectedIndex = (ComboBoxLanguages)languageComboBox.SelectedIndex;
132
133          /// close the form if the selected language didn't change
134          if (tCurrentSelectedIndex == mLoadedComboBoxLanguage)
135          {
136              this.Close();
137          }
138          else
139          {
140              /// Saving english language
141              if (tCurrentSelectedIndex == ComboBoxLanguages.English)
142              {
143                  ErrorCode tResult = SaveLanguageSettings(mConfigEnglishLanguage);
144                  if (tResult == ErrorCode.SUCCESS)
145                  {
146                      Application.Restart();
147                  }
148                  else
149                  {
150                      ShowMessage.Box(${ResourceStrings.errorSavingSettings}, ${ResourceStrings.error}, MessageBoxButtons.OK,
151                      MessageBoxIcon.Error);
152                  }
153              /// Saving arabic language
154              else if (tCurrentSelectedIndex == ComboBoxLanguages.Arabic)
155              {
156                  ErrorCode tResult = SaveLanguageSettings(mConfigArabicLanguage);
157                  if (tResult == ErrorCode.SUCCESS)
158                  {
159                      Application.Restart();
160                  }
161                  else
162                  {
163                      ShowMessage.Box(${ResourceStrings.errorSavingSettings}, ${ResourceStrings.error}, MessageBoxButtons.OK,
164                      MessageBoxIcon.Error);
165                  }
166              }
167          }
168      }
169  }
```

Description:

This event handler handles saving new language settings.

If settings didn't change the form will close.

If settings changed, new settings will be saved and the application will restart to reflect the new language.

LoadLangaugeSettings

```
179  /// <summary>
180  /// Load combo box with the current language of the application
181  /// </summary>
182  private void LoadLangaugeSettings()
183  {
184      try
185      {
186          string tReturnedLangaugeValue = tSettings[mDefaultCultureString].Value;
187
188          if (tReturnedLangaugeValue == mConfigEnglishLangauge)
189          {
190              languageComboBox.SelectedIndex = (int)ComboBoxLanguages.English;
191              mLoadedComboBoxLanguage = (ComboBoxLanguages)languageComboBox.SelectedIndex;
192          }
193          else if (tReturnedLangaugeValue == mConfigArabicLangauge)
194          {
195              languageComboBox.SelectedIndex = (int)ComboBoxLanguages.Arabic;
196              mLoadedComboBoxLanguage = (ComboBoxLanguages)languageComboBox.SelectedIndex;
197          }
198          else
199          {
200              languageComboBox.SelectedIndex = (int)ComboBoxLanguages.NoLanguage;
201          }
202      }
203      catch (Exception ex)
204      {
205          ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK,
206          Logger.LogError(ex);
207      }
208  }
```

Description:

This method load the combo box with saved language settings

SaveLangaugeSettings

```
210
211     /// <summary>
212     /// Saves new settings to config file
213     /// </summary>
214     /// <returns>
215     /// ErrorCode
216     /// </returns>
217     private ErrorCode SaveLangaugeSettings(string pLangaugeToSave)
218     {
219         try
220         {
221             tSettings[mDefaultCultureString].Value = pLangaugeToSave;
222             tConfigFile.Save(ConfigurationSaveMode.Modified);
223             ConfigurationManager.RefreshSection(tConfigFile.AppSettings.SectionInformation.Name);
224
225             return ErrorCode.SUCCESS;
226         }
227         catch (Exception ex)
228         {
229             Logger.LogError(ex); // write error to log file
230             return ErrorCode.ERROR;
231         }
232     }
233 #endregion
```

Description:

Saves new settings to config file

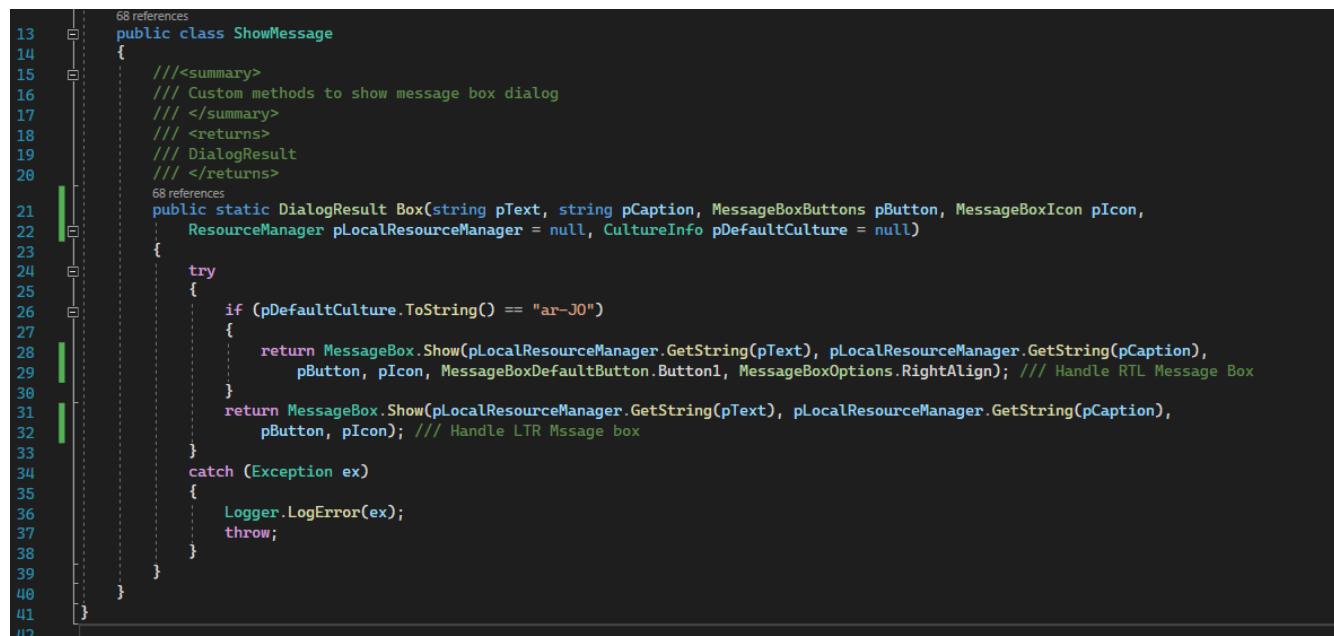
Returns:

ErrorCode

1.4.5 ShowMessage class

This class contains a method that is responsible for showing MessageBoxes

Box method:



The screenshot shows a code editor with the following code:

```
13  public class ShowMessage
14  {
15      ///<summary>
16      /// Custom methods to show message box dialog
17      /// </summary>
18      /// <returns>
19      /// DialogResult
20      /// </returns>
21      public static DialogResult Box(string pText, string pCaption, MessageBoxButtons pButton, MessageBoxIcon pIcon,
22                                     ResourceManager pLocalResourceManager = null, CultureInfo pDefaultCulture = null)
23      {
24          try
25          {
26              if (pDefaultCulture.ToString() == "ar-JO")
27              {
28                  return MessageBox.Show(pLocalResourceManager.GetString(pText), pLocalResourceManager.GetString(pCaption),
29                                         pButton, pIcon, MessageBoxButtons.DefaultButton.Button1, MessageBoxIcon.RightAlign); // Handle RTL Message Box
30              }
31              return MessageBox.Show(pLocalResourceManager.GetString(pText), pLocalResourceManager.GetString(pCaption),
32                                         pButton, pIcon); // Handle LTR Message box
33          }
34          catch (Exception ex)
35          {
36              Logger.LogError(ex);
37              throw;
38          }
39      }
40  }
```

Parameters:

pText-> string

pCaption-> string

pButton-> MessageBoxButtons

pIcon-> MessageBoxIcon

pLocalResourceManager = null-> ResourceManager

pDefaultCulture = null-> CultureInfo

Description:

This function receives normal message box parameters with ResourceManager and CultrueInfo to handle the language and the translation of messages that appear on the message box.

Step 1: shows a messages box that handles RTL languages

Step 2: shows a normal LTR message box

Returns:

DialgoResult

1.5 Business Logic Layer

1.5.1 QuestionManager

```
13
14  namespace SurveyQuestionsConfigurator.QuestionLogic
15  {
16      public class QuestionManager
17      {
18          [Properties & Attributes]
19
20          [Constructor]
21
22          [Delegates And Events]
23
24          [Add Question Functions]
25
26          [Edit Question Functions]
27
28          [Get Question Functions]
29
30          [Delete Question Functions]
31
32          [Validation Functions]
33
34      }
35
36 }
```

Properties & Attributes

```
17
18     {
19         #region Properties & Attributes
20
21         private SmileQuestionRepository mSmileyQuestionRepository;
22         private SliderQuestionRepository mSliderQuestionRepository;
23         private StarQuestionRepository mStarQuestionRepository;
24         private GenericRepository mRepository;
25         private static List<Question> mChachedQuestions; /// use to check if databsae data changed
26         private static bool mKeepWatchingFlag; /// flag used in the thread that keep watching if data has changed
27
28     #endregion
29 }
```

Step 1: create instances of every question repository.

Step 2: use to check if database data changed

Step 3: flag used in the thread that keep watching if data has changed

Constructor

QuestionManager

```
28
29  #region Constructor
30  3 references
31  public QuestionManager()
32  {
33      try
34      {
35          mSmileyQuestionRepository = new SmileyQuestionRepository();
36          mSliderQuestionRepository = new SliderQuestionRepository();
37          mStarQuestionRepository = new StarQuestionRepository();
38          mRepository = new GenericRepository();
39          mChachedQuestions = new List<Question>();
40          mKeepWatchingFlag = true; // keep checking for changes
41      }
42      catch (Exception ex)
43      {
44          Logger.LogError(ex);
45      }
46  #endregion
47
```

Step 1: set flag to true-> keep checking for changes.

Delegates And Events

```
48
49     #region Delegates And Events
50
51     public delegate void DataChanged(ErrorCode pErrorCode, List<Question> pQuestionList);
52     public static event DataChanged refreshDataEvent;
53
```

Step 1: create delegate that holds reference of functions that wants to subscribe for it.

Step 2: create event for that delegate.

WatchForChanges:

```
54  /// <summary>
55  /// Subscribe to a thread that continuously watch for DB changes every 30 seconds
56  /// </summary>
57  [reference]
58  public void WatchForChanges()
59  {
60      try
61      {
62          int AutoRefreshTimer = GetAutoRefreshTimerFromConfigFile();
63
64          List<Question> tList = new List<Question>();
65          ErrorCode tResult = ErrorCode.ERROR;
66
67          Thread tAutoRefreshThread = new Thread(new ThreadStart(delegate
68          {
69              while (mKeepWatchingFlag)
70              {
71                  tList.Clear();
72                  tResult = mRepository.GetAll(ref tList);
73
74                  if (mChachedQuestions.SequenceEqual(tList) == false) // if there is a difference
75                  {
76                      if (tResult != ErrorCode.ERROR)
77                      {
78                          ResetChachedQuestionsList(tList);
79                      }
80                      refreshDataEvent?.Invoke(tResult, tList);
81                  }
82
83                  Thread.Sleep(AutoRefreshTimer);
84              }
85          }));
86
87          tAutoRefreshThread.IsBackground = true;
88          tAutoRefreshThread.Start();
89      }
90      catch (Exception ex)
91      {
92          Logger.LogError(ex);
93      }
94  }
```

Description:

Create a thread that continuously watch for changes in the database and compare them to the chanced list and invoke referenced function then sleeps again.

Sleep time is fetched from config file.

InstantlyRefreshList

```
96         /// <summary>
97         /// Refresh list at anytime without any delay
98         /// </summary>
99         public void InstantlyRefreshList()
100        {
101            try
102            {
103                List<Question> tList = new List<Question>();
104                ErrorCode tResult = ErrorCode.ERROR;
105
106                tList.Clear();
107                tResult = mRepository.GetAll(ref tList);
108
109                refreshDataEvent.Invoke(tResult, tList);
110            }
111            catch (Exception ex)
112            {
113                Logger.LogError(ex);
114                throw;
115            }
116        }
117    }
```

Description:

This function invokes event without any condition.

StopWatching

```
118     /// <summary>
119     /// Breaks the infinite loop of "tAutoRefreshThread"
120     /// </summary>
121     public void StopWatching()
122     {
123         try
124         {
125             mKeepWatchingFlag = false;
126         }
127         catch (Exception ex)
128         {
129             Logger.LogError(ex);
130             throw;
131         }
132     }
133 }
```

Description:

Breaks the infinite loop of WatchForChanges() method's thread

GetAutoRefreshTimerFromConfigFile

```
134  /// <summary> Get auto refresh timer from config file
135  1 reference
136  private int GetAutoRefreshTimerFromConfigFile()
137  {
138      try
139      {
140          string tSectionName = "AutoRefreshTimer";
141          var tConfigFile = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
142          var tSettings = tConfigFile.AppSettings.Settings;
143          string tValue = tSettings[tSectionName].Value;
144          return Int32.Parse(tValue);
145      }
146      catch (Exception ex)
147      {
148          Logger.LogError(ex);
149          int tDefaultTimer = 30000;
150      }
151  }
```

Description:

Gets and return auto refresh timer from config file

ResetChachedQuestionsList

```
158     /// <summary>
159     /// reset cached list with new passed list
160     /// </summary>
161     private static void ResetChachedQuestionsList(List<Question> pQuestionList)
162     {
163         try
164         {
165             lock (mChachedQuestions)
166             {
167                 mChachedQuestions.Clear();
168                 mChachedQuestions.AddRange(pQuestionList);
169             }
170         }
171         catch (Exception ex)
172         {
173             Logger.LogError(ex);
174         }
175     }
176 }
```

Description:

Reset cached list with new passed list and lock the resetting process so one single thread at a time can access it.

InsertSmileyQuestion, InsertSliderQuestion & InsertStarQuestion

```
180  |  /// <summary> Check values and pass it to repository layer
188  |  public ErrorCode InsertSmileyQuestion(SmileyQuestion pSmileyQuestion)
189  |
190  |  {
191  |      try
192  |      {
193  |          if (CheckSmileyQuestionValues(pSmileyQuestion) == ErrorCode.SUCCESS)
194  |          {
195  |              return mSmileyQuestionRepository.Add(pSmileyQuestion);
196  |          }
197  |          else
198  |          {
199  |              return Generic.ErrorCode.ERROR;
200  |          }
201  |      }
202  |      catch (Exception ex)
203  |      {
204  |          Logger.LogError(ex);
205  |      }
206  |  }
```

```
208  |  /// <summary> Check values and pass it to repository layer
216  |  public ErrorCode InsertSliderQuestion(SliderQuestion pSliderQuestion)
217  |
218  |  {
219  |      try
220  |      {
221  |          if (CheckSliderQuestionValues(pSliderQuestion) == ErrorCode.SUCCESS)
222  |          {
223  |              return mSliderQuestionRepository.Add(pSliderQuestion);
224  |          }
225  |          else
226  |          {
227  |              return Generic.ErrorCode.ERROR;
228  |          }
229  |      }
230  |      catch (Exception ex)
231  |      {
232  |          Logger.LogError(ex);
233  |      }
234  |  }
```

```
236  /// <summary> Check values and pass it to repository layer
237  1 reference
244  public ErrorCode InsertStarQuestion(StarQuestion pStarQuestion)
245  {
246      try
247      {
248          if (CheckStarQuestionValues(pStarQuestion) == ErrorCode.SUCCESS)
249          {
250              return mStarQuestionRepository.Add(pStarQuestion);
251          }
252          else
253          {
254              return Generic.ErrorCode.ERROR;
255          }
256      }
257      catch (Exception ex)
258      {
259          Logger.LogError(ex);
260          return Generic.ErrorCode.ERROR;
261      }
262  }
263
```

Description:

All of these methods validate their corresponding passed question and pass them to their corresponding repository layer.

UpdateSmileyQuestion, UpdateSliderQuestion & UpdateStarQuestion

```
268  /// <summary> Check values and pass it to repository layer
269  1 reference
270  public ErrorCode UpdateSmileyQuestion(SmileQuestion pSmileyQuestion)
271  {
272      try
273      {
274          if (CheckSmileyQuestionValues(pSmileyQuestion) == ErrorCode.SUCCESS)
275          {
276              return mSmileyQuestionRepository.Update(pSmileyQuestion);
277          }
278          else
279          {
280              return Generic.ErrorCode.ERROR;
281          }
282      }
283      catch (Exception ex)
284      {
285          Logger.LogError(ex);
286          return Generic.ErrorCode.ERROR;
287      }
288  }
```

```
296  /// <summary> Check values and pass it to repository layer
297  1 reference
298  public ErrorCode UpdateSliderQuestion(SliderQuestion pSliderQuestion)
299  {
300      try
301      {
302          if (CheckSliderQuestionValues(pSliderQuestion) == ErrorCode.SUCCESS)
303          {
304              return mSliderQuestionRepository.Update(pSliderQuestion);
305          }
306          else
307          {
308              return Generic.ErrorCode.ERROR;
309          }
310      }
311      catch (Exception ex)
312      {
313          Logger.LogError(ex);
314          return Generic.ErrorCode.ERROR;
315      }
316  }
```

```
323
324     /// <summary> Check values and pass it to repository layer
325     1 reference
326     public ErrorCode UpdateStarQuestion(StarQuestion pStarQuestion)
327     {
328         try
329         {
330             if (CheckStarQuestionValues(pStarQuestion) == ErrorCode.SUCCESS)
331             {
332                 return mStarQuestionRepository.Update(pStarQuestion);
333             }
334             else
335             {
336                 return Generic.ErrorCode.ERROR;
337             }
338         }
339         catch (Exception ex)
340         {
341             Logger.LogError(ex);
342             return Generic.ErrorCode.ERROR;
343         }
344     }
345 }
```

Description:

All of these methods validate their corresponding passed question and pass them to their corresponding repository layer.

Get Question Functions

GetSmileyQuestionByID, GetSliderQuestionByID, GetStarQuestionByID & GetAllQuestions:

```
356  /// <summary> Get a question from DB and check it values  
364  1 reference  
365  public ErrorCode GetSmileyQuestionByID(ref SmileyQuestion pSmileyQuestion)  
366  {  
367      try  
368      {  
369          if (pSmileyQuestion.ID > 0)  
370          {  
371              return mSmileyQuestionRepository.Get(ref pSmileyQuestion);  
372          }  
373          else  
374          {  
375              return Generic.ErrorCode.ERROR;  
376          }  
377      }  
378      catch (Exception ex)  
379      {  
380          Logger.LogError(ex);  
381          return Generic.ErrorCode.ERROR;  
382      }  
383 }
```

```
384  /// <summary> Get a question from DB and check it values  
392  1 reference  
393  public ErrorCode GetSliderQuestionByID(ref SliderQuestion pSliderQuestion)  
394  {  
395      try  
396      {  
397          if (pSliderQuestion.ID > 0)  
398          {  
399              return mSliderQuestionRepository.Get(ref pSliderQuestion);  
400          }  
401          else  
402          {  
403              return Generic.ErrorCode.ERROR;  
404          }  
405      }  
406      catch (Exception ex)  
407      {  
408          Logger.LogError(ex);  
409          return Generic.ErrorCode.ERROR;  
410      }  
411 }
```

```
411
412     /// <summary> Get a question from DB and check it values
413     1 reference
420     public ErrorCode GetStarQuestionByID(ref StarQuestion pStarQuestion)
421     {
422         try
423         {
424             if (pStarQuestion.ID > 0)
425             {
426                 return mStarQuestionRepository.Get(ref pStarQuestion);
427             }
428             else
429             {
430                 return Generic.ErrorCode.ERROR;
431             }
432         }
433         catch (Exception ex)
434         {
435             Logger.LogError(ex);
436             return Generic.ErrorCode.ERROR;
437         }
438     }
439
```

```
440
441     /// <summary> Get all questions from DB and check it values
442     0 references
448     public ErrorCode GetAllQuestions(ref List<Question> questionsList)
449     {
450         try
451         {
452             var returnValue = mRepository.GetAll(ref questionsList);
453             ResetChachedQuestionsList(questionsList);
454             return returnValue;
455         }
456         catch (Exception ex)
457         {
458             Logger.LogError(ex);
459             return Generic.ErrorCode.ERROR;
460         }
461     }
462
```

Description:

All of these methods validate their corresponding passed question and pass them to their corresponding repository layer.

Delete Question Functions

```
467     /// <summary> Pass ID to the question of which to be deleted
468     1 reference
469     public ErrorCode DeleteQuestionByID(int pQuestionId)
470     {
471         try
472         {
473             if (pQuestionId > 0)
474             {
475                 return mRepository.Delete(pQuestionId);
476             }
477             else
478             {
479                 return Generic.ErrorCode.ERROR;
480             }
481         }
482         catch (Exception ex)
483         {
484             Logger.LogError(ex);
485             return Generic.ErrorCode.ERROR;
486         }
487     }
488 }
```

Description:

This method validate question and pass them to its corresponding repository layer.

Validation Functions

These functions validate their passed question and return their corresponding ErrorCode.

1.5.2 ConnectionSettingsManager

Code:

The code is divided into regions based on their functionality.

```
14 3 references
15 public class ConnectionSettingsManager
16 {
17     [Properties & Attributes]
18     [Constructor]
19     [Methods]
20     [Validation Methods]
21 }
22 }
23 }
24 }
```

Properties & Attributes

```
15
16      #region Properties & Attributes
17
18      DbConnect mDbConnect; // Data Access Layer
19
20      #endregion
21
```

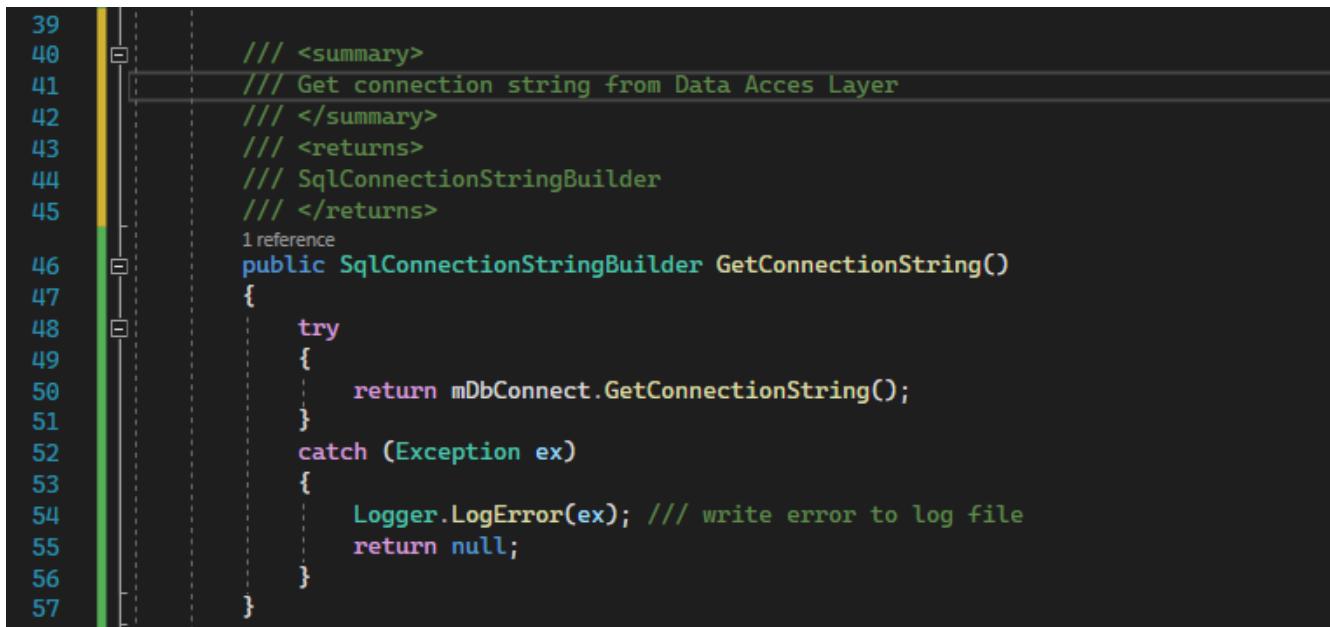
Constructor

```
21
22  #region Constructor
23
24  public ConnectionSettingsManager()
25  {
26      try
27      {
28          mDbConnect = new DbConnect();
29      }
30      catch (Exception ex)
31      {
32          Logger.LogError(ex); // write error to log file
33      }
34 }
```

Initialize properties

Methods

GetConnectionString



The screenshot shows a code editor with a dark theme. A vertical yellow bar highlights the first few lines of code. The code itself is a C# method named `GetConnectionString`. It includes XML documentation comments at the top, a try-catch block to handle exceptions, and a call to `Logger.LogError` if an exception occurs. The code is color-coded with syntax highlighting.

```
39
40     /// <summary>
41     /// Get connection string from Data Acces Layer
42     /// </summary>
43     /// <returns>
44     /// SqlConnectionStringBuilder
45     /// </returns>
46     public SqlConnectionStringBuilder GetConnectionString()
47     {
48         try
49         {
50             return mDbConnect.GetConnectionString();
51         }
52         catch (Exception ex)
53         {
54             Logger.LogError(ex); // write error to log file
55             return null;
56         }
57     }
```

Description:

Get connection string from Data Access Layer

Returns:

`SqlConnectionStringBuilder`

CheckConnectivity

```
59     /// <summary>
60     /// Check connectivity of the passed SqlConnectionStringBuilder connection string and return the corresponding ErrorCode
61     /// </summary>
62     /// <returns>
63     /// ErrorCode
64     /// </returns>
65     public ErrorCode CheckConnectivity(SqlConnectionStringBuilder pBuilder)
66     {
67         try
68         {
69             return mDbConnect.CheckConnectivity(pBuilder);
70         }
71         catch (Exception ex)
72         {
73             Logger.LogError(ex); // write error to log file
74             return ErrorCode.ERROR;
75         }
76     }
```

Description:

Check connectivity of the passed SqlConnectionStringBuilder connection string and return the corresponding ErrorCode

Returns:

ErrorCode

SaveConnectionString

```
77
78     ///<summary>
79     /// Save passed SqlConnectionStringBuilder connection string
80     ///</summary>
81     ///<returns>
82     /// ErrorCode
83     ///</returns>
84     public ErrorCode SaveConnectionString(SqlConnectionStringBuilder pBuilder)
85     {
86         try
87         {
88             return mDbConnect.SaveConnectionString(pBuilder);
89         }
90         catch (Exception ex)
91         {
92             Logger.LogError(ex); // write error to log file
93             return ErrorCode.ERROR;
94         }
95     }
96
97 #endregion
```

Description:

Save passed SqlConnectionStringBuilder connection string

Returns:

ErrorCode

Validation Methods

CheckConnectionStringInputFields

```
98
99
100
101    #region Validation Methods
102
103        /// <summary>
104        /// Check connection string validation
105        /// </summary>
106        /// <returns>
107        /// ErrorCode
108        /// </returns>
109        1 reference
110        public ErrorCode CheckConnectionStringInputFields(SqlConnectionStringBuilder pBuilder)
111    {
112        try
113        {
114            if (!String.IsNullOrEmpty(pBuilder.DataSource.ToString()) && pBuilder.DataSource.Length <= 128)
115                if (!String.IsNullOrEmpty(pBuilder.InitialCatalog.ToString()) && pBuilder.InitialCatalog.Length <= 128)
116                    if (!String.IsNullOrEmpty(pBuilder.UserID.ToString()) && pBuilder.UserID.Length <= 128)
117                        if (!String.IsNullOrEmpty(pBuilder.Password.ToString()) && pBuilder.Password.Length <= 128)
118                            return ErrorCode.SUCCESS;
119                    return ErrorCode.ERROR;
120                }
121                catch (Exception ex)
122                {
123                    Logger.LogError(ex); // write error to log file
124                    return ErrorCode.ERROR;
125                }
126        #endregion
```

Description:

Check connection string validation

Returns:

ErrorCode

1.6 Repositories

1.6.1 GenericRepository

This repository is generic and it's inherited by other more specific repositories.

It has 2 functions:

1. Delete
2. GetAll

Code:

```
15  public class GenericRepository
16  {
17
18      protected DbConnect mDbConnect = new DbConnect(); // Instance of Data Access Layer
19
20      ///<summary>
21      /// Delete a question by it's passed ID
22      ///</summary>
23      public ErrorCode Delete(int pID)
24      {
25          try
26          {
27              return mDbConnect.DeleteQuestionByID(pID);
28          }
29          catch (Exception ex)
30          {
31              Logger.LogError(ex);
32              return ErrorCode.ERROR;
33          }
34      }
35
36      ///<summary>
37      /// Get all question from Questions table and save them into passed by reference list
38      ///</summary>
39      public ErrorCode GetAll(ref List<Question> pQuestionsList)
40      {
41          try
42          {
43              return mDbConnect.GetAllQuestions(ref pQuestionsList);
44          }
45          catch (Exception ex)
46          {
47              Logger.LogError(ex);
48              return ErrorCode.ERROR;
49          }
50      }
51  }
```

Delete:

```
19          ///<summary>
20          /// Delete a question by it's passed ID
21          ///</summary>
22          1 reference
23          public ErrorCode Delete(int pID)
24          {
25              try
26              {
27                  return mDbConnect.DeleteQuestionByID(pID);
28              }
29              catch (Exception ex)
30              {
31                  Logger.LogError(ex);
32                  return ErrorCode.ERROR;
33              }
34          }
```

Parameter:

Int pID -> Question ID to be deleted

Description:

Delete a question by its passed ID

Returns:

ErrorCode

GetAll:

```
36     ///<summary>
37     /// Get all question from Questions table and save them into passed by reference list
38     ///</summary>
39     public ErrorCode GetAll(ref List<Question> pQuestionsList)
40     {
41         try
42         {
43             return mDbConnect.GetAllQuestions(ref pQuestionsList);
44         }
45         catch (Exception ex)
46         {
47             Logger.LogError(ex);
48             return ErrorCode.ERROR;
49         }
50     }
```

Parameter:

`ref List<Question> pQuestionsList`

Description:

Get all question from Questions table and save them into passed by reference list

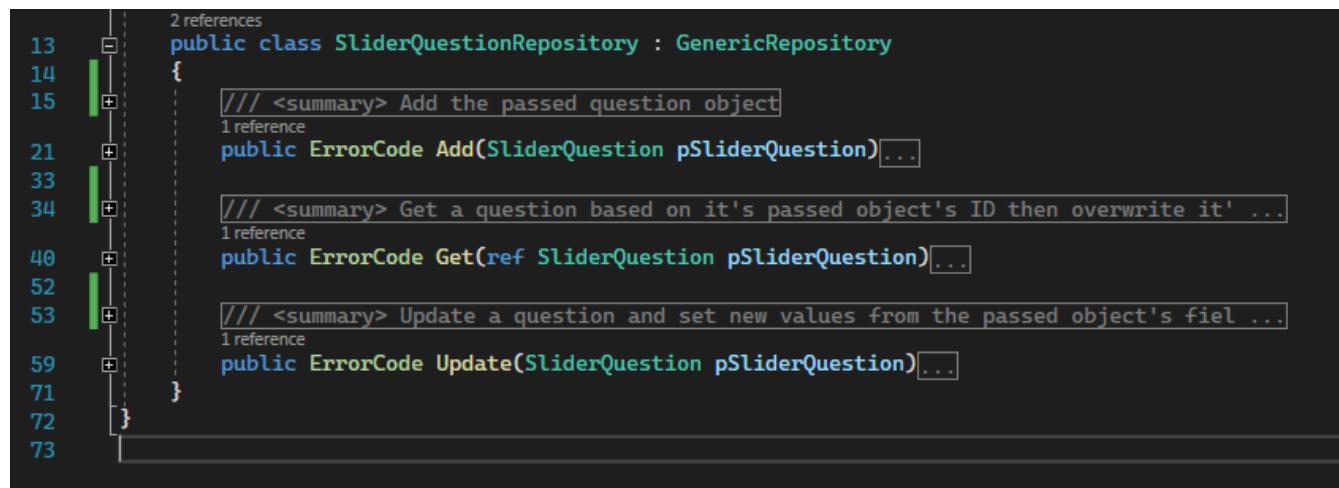
Returns:

`ErrorCode`

1.6.2 SliderQuestionRepository

This class inherits from GenericRepository Class

Code:



The screenshot shows a code editor window with the following code:

```
13  2 references
14  public class SliderQuestionRepository : GenericRepository
15  {
16      /// <summary> Add the passed question object
17      /// <reference>
18      public ErrorCode Add(SliderQuestion pSliderQuestion)...
19
20      /// <summary> Get a question based on it's passed object's ID then overwrite it' ...
21      /// <reference>
22      public ErrorCode Get(ref SliderQuestion pSliderQuestion)...
23
24      /// <summary> Update a question and set new values from the passed object's fiel ...
25      /// <reference>
26      public ErrorCode Update(SliderQuestion pSliderQuestion)...
27
28  }
```

The code defines a class named SliderQuestionRepository that inherits from GenericRepository. It contains three methods: Add, Get, and Update. Each method has a summary comment and a reference to another part of the code.

Add:

```
13  2 references
14  public class SliderQuestionRepository : GenericRepository
15  {
16      /// <summary>
17      /// Add the passed question object to the database through Data Access Layer
18      /// </summary>
19      /// <returns>
20      /// ErrorCode
21      /// </returns>
22      1 reference
23      public ErrorCode Add(SliderQuestion pSliderQuestion)
24      {
25          try
26          {
27              return mDbConnect.InsertSliderQuestion(pSliderQuestion);
28          }
29          catch (Exception ex)
30          {
31              Logger.LogError(ex);
32              return ErrorCode.ERROR;
33          }
34      }
35  
```

Parameters:

pSliderQuestion-> SliderQuestion

Description:

Add the passed question object to the database through Data Access Layer

Returns:

ErrorCode

Get:

```
33
34     /// <summary>
35     /// Get a question based on it's passed object's ID then overwrite it's fields
36     /// </summary>
37     /// <returns>
38     /// ErrorCode
39     /// </returns>
40     public ErrorCode Get(ref SliderQuestion pSliderQuestion)
41     {
42         try
43         {
44             return mDbConnect.GetSliderQuestionByID(ref pSliderQuestion);
45         }
46         catch (Exception ex)
47         {
48             Logger.LogError(ex);
49             return ErrorCode.ERROR;
50         }
51     }
```

Parameters:

pSliderQuestion-> SliderQuestion

Description:

Get a question based on its passed object's ID then overwrite its fields

Returns:

ErrorCode

Update:

```
52
53     /// <summary>
54     /// Update a question and set new values from the passed object's fields
55     /// </summary>
56     /// <returns>
57     /// ErrorCode
58     /// </returns>
59     public ErrorCode Update(SliderQuestion pSliderQuestion)
60     {
61         try
62         {
63             return mDbConnect.UpdateSliderQuestion(pSliderQuestion);
64         }
65         catch (Exception ex)
66         {
67             Logger.LogError(ex);
68             return ErrorCode.ERROR;
69         }
70     }
```

Parameters:

pSliderQuestion-> SliderQuestion.

Description:

Update a question and set new values from the passed object's fields

Returns:

ErrorCode

1.6.3 SliderQuestionRepository

This class does the exact functionality that SliderQuestionRepository class in section 3.5.2 does but to its corresponding slider questions

1.6.4 StarQuestionRepository

This class does the exact functionality that SliderQuestionRepository class in section 3.5.2 does but to its corresponding star questions

1.7 Data Access Layer (DbConnect class)

This class is used to access the database through SQL quires, SQL stored procedures, SQL functions and ADO.NET classes and methods.

Code:

The code is divided into regions based on their functionality.

The screenshot shows a code editor with the following code snippet:

```
1  [+using ...]
14  [-namespace SurveyQuestionsConfigurator.DataAccess
15  {
16      [-public class DbConnect
17      {
18          [+Properties & Attributes
19
20          [+Constructor
21
22          [+Common Methods
23
24          [+INSERT Methods
25
26          [+UPDATE Methods
27
28          [+DELETE Methods
29
30          [+GET Methods
31
32      }
33
34  }
```

A vertical dashed line on the left side of the code separates the code from a list of regions on the right. The regions are:

- Properties & Attributes
- Constructor
- Common Methods
- INSERT Methods
- UPDATE Methods
- DELETE Methods
- GET Methods

Properties & Attributes

```
18  | #region Properties & Attributes  
19  |  
20  |     private ConnectionStringSettings mSqlConnectionSettings; /// get connection string information from App.config, "connectionStrings" section  
21  |  
22  | #endregion  
23  |
```

mSqlConnectionSettings-> get connection string information from App.config, "connectionStrings" section.

Constructor

```
24      #region Constructor
25      2 references
26      public DbConnect()
27      {
28          try
29          {
30              mSqlConnectionSettings = ConfigurationManager.ConnectionStrings[0];
31          }
32          catch (Exception ex)
33          {
34              Logger.LogError(ex); // write error to log file
35          }
36      }
37      #endregion
```

Get the required connection string section from app.config

Common Methods

CheckIfOrderExist

```
41     /// <summary>
42     /// Execute and SQL function and Return SUCCESS if order is not already in use
43     /// </summary>
44     /// <param name="pSqlConnection"></param>
45     /// <param name="pOrder"></param>
46     /// <returns>
47     /// ErrorCode.SUCCESS
48     /// ErrorCode.VALIDATION
49     /// </returns>
50     private ErrorCode CheckIfOrderExist(SqlConnection pSqlConnection, int pOrder)
51     {
52         try
53         {
54             using (SqlCommand cmd = pSqlConnection.CreateCommand())
55             {
56                 cmd.CommandText = "dbo.CheckIfOrderExist";
57                 cmd.CommandType = CommandType.StoredProcedure;
58                 SqlParameter[] parameters = new SqlParameter[]
59                 {
60                     new SqlParameter($"{QuestionColumn.ReturnValue}", SqlDbType.Int),
61                     new SqlParameter($"{QuestionColumn.Order}", pOrder),
62                 };
63                 parameters[0].Direction = ParameterDirection.ReturnValue;
64                 cmd.Parameters.AddRange(parameters);
65
66                 cmd.ExecuteNonQuery();
67                 return (ErrorCode)parameters[0].Value;
68             }
69         }
70         catch (Exception ex)
71         {
72             Logger.LogError(ex); // write error to log file
73             return ErrorCode.ERROR;
74         }
75     }
```

Parameters:

pSqlConnection-> SqlConnection

pOrder-> int

Description:

This function uses an already open SqlConnection and SqlTransaction(TransactionScope) to execute an SQL function. And return ErrorCode.SUCCESS if question order is not taken, ErrorCode.VALIDATION if question order is already in use.

Step 1-> sets an output parameter that saved the returned value form the SQL function.

Returns:

ErrorCode.SUCCESS

ErrorCode.VALIDATION

CheckIfUpdatingSameQuestion:

```
75     /// <summary>
76     /// Return the ID of question based on it's order
77     /// </summary>
78     /// <param name="pSqlConnection"></param>
79     /// <param name="pOrder"></param>
80     /// <returns>
81     /// ErrorCode.SUCCESS
82     /// ErrorCode.ERROR
83     /// </returns>
84     3 references
85     private ErrorCode CheckIfUpdatingSameQuestion(SqlConnection pSqlConnection, int pOrder, int pQuestionID)
86     {
87         try
88         {
89             using (SqlCommand cmd = pSqlConnection.CreateCommand())
90             {
91                 cmd.CommandText = "dbo.CheckIfUpdatingSameQuestion";
92                 cmd.CommandType = CommandType.StoredProcedure;
93                 SqlParameter[] parameters = new SqlParameter[]
94                 {
95                     new SqlParameter("${QuestionColumn.ReturnValue}", SqlDbType.Int),
96                     new SqlParameter("${QuestionColumn.Order}", pOrder),
97                     new SqlParameter("${QuestionColumn.ID}", pQuestionID),
98                 };
99                 parameters[0].Direction = ParameterDirection.ReturnValue; // returned value from function
100                cmd.Parameters.AddRange(parameters);
101
102                cmd.ExecuteNonQuery();
103
104                return (ErrorCode)parameters[0].Value;
105            }
106            catch (Exception ex)
107            {
108                Logger.LogError(ex); // write error to log file
109                return ErrorCode.ERROR;
110            }
111        }
112    }
```

Paramaeters:

pSqlConnection -> SqlConnection

pOrder -> int

pQuestionID -> int

Description:

This function uses an already open SqlConnection and SqlTransaction(TransactionScope) to execute an SQL function.

Checks if the passed question order and ID belongs to the same question or not.

Returns:

ErrorCode.SUCCESS

ErrorCode.VALIDATION

CheckConnectivity:

```
112
113     /// <summary>
114     /// Test connecting string
115     /// </summary>
116     /// <returns>
117     /// ErrorCode.SUCCESS
118     /// ErrorCode.ERROR
119     /// </returns>
120     public ErrorCode CheckConnectivity(SqlConnectionStringBuilder pBuilder)
121     {
122         try
123         {
124             using (SqlConnection tSqlConnection = new SqlConnection())
125             {
126                 tSqlConnection.ConnectionString = pBuilder.ConnectionString;
127                 tSqlConnection.Open();
128                 if (tSqlConnection.State == ConnectionState.Open)
129                 {
130                     return ErrorCode.SUCCESS;
131                 }
132                 return ErrorCode.ERROR;
133             }
134         }
135         catch (Exception ex)
136         {
137             Logger.LogError(ex); // write error to log file
138             return ErrorCode.ERROR;
139         }
140     }
```

Parameters:

pBuilder-> SqlConnectionStringBuilder

Description:

Check if the passed SqlConnectionStringBuilder.ConnectionString has a valid connection or not.

Returns:

ErrorCode.SUCCESS

ErrorCode.VALIDATION

SaveConnectionString:

```
141
142     /// <summary>
143     /// Save new connection string to config file
144     /// </summary>
145     /// <returns>
146     /// ErrorCode.SUCCESS
147     /// ErrorCode.ERROR
148     /// </returns>
149     1 reference
150     public ErrorCode SaveConnectionString(SqlConnectionStringBuilder pBuilder)
151     {
152         try
153         {
154             string tProviderName = "System.Data.SqlClient";
155             string tSectionName = "connectionStrings";
156
157             Configuration tConfig = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
158             tConfig.ConnectionStrings.ConnectionStrings[0].ConnectionString = pBuilder.ConnectionString;
159             tConfig.ConnectionStrings.ConnectionStrings[0].ProviderName = tProviderName;
160             tConfig.Save(ConfigurationSaveMode.Minimal);
161
162             ConfigurationManager.RefreshSection(tSectionName);
163             UpdateLocalConnectionSettings();
164             return ErrorCode.SUCCESS;
165         }
166         catch (Exception ex)
167         {
168             Logger.LogError(ex); // write error to log file
169             return ErrorCode.ERROR;
170         }
171     }
```

Parameters:

pBuilder-> SqlConnectionStringBuilder

Description:

Saves the passed connection string to the app.config file.

Step 1-> Update local global connecting string with the new saved connection string.

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

GetConnectionString:

```
171  /// <summary>
172  /// Get used connection string from config file
173  /// </summary>
174  /// <returns>
175  /// ErrorCode.SUCCESS
176  /// ErrorCode.ERROR
177  /// </returns>
178  1 reference
179  public SqlConnectionStringBuilder GetConnectionString()
180  {
181      try
182      {
183          SqlConnectionStringBuilder tbuilder = new SqlConnectionStringBuilder();
184          tbuilder.ConnectionString = mSqlConnectionSettings.ConnectionString;
185          return tbuilder;
186      }
187      catch (Exception ex)
188      {
189          Logger.LogError(ex); // write error to log file
190          return null;
191      }
192  }
```

Description:

Get the connection string from app.config file that is saved in mSqlConnectionSettings.

Returns:

SqlConnectionStringBuilder

null

UpdateGlobalConnectionSettings:

```
193
194     /// <summary>
195     /// Sets mSqlConnectionSettings to newest data
196     /// </summary>
197     private ErrorCode UpdateGlobalConnectionSettings()
198     {
199         try
200         {
201             mSqlConnectionSettings = ConfigurationManager.ConnectionStrings[0];
202             return ErrorCode.SUCCESS;
203         }
204         catch (Exception ex)
205         {
206             Logger.LogError(ex); // write error to log file
207             return ErrorCode.ERROR;
208         }
209     }
210 }
```

Description:

Update global connecting settings after it was saved in the app.config file

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

InsertSmileyQuestion

```
219     /// <summary>
220     /// 1) Open a transaction
221     /// 2) Check if desired order is already in use before inserting (SQL Function)
222     /// 3) Perform insert stored procedure -> get the result as ErrorCode
223     /// 4) return return from function with the corresponding ErrorCode
224     /// </summary>
225     /// <returns>
226     /// ErrorCode.SUCCESS
227     /// ErrorCode.ERROR
228     /// ErrorCode.VALIDATION
229     /// </returns>
230     1 reference
231     public ErrorCode InsertSmileyQuestion(SmileyQuestion pSmileyQuestion)
232     {
233         /// Try to insert a new smiley question into "Smiley_Questions" table
234         try
235         {
236             /// ErrorCode to be returned
237             ErrorCode tOrderStatusResult = ErrorCode.ERROR;
238
239             using (TransactionScope transactionScope = new TransactionScope())
240             {
241                 using (SqlConnection sqlConnection = new SqlConnection())
242                 {
243                     sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
244                     sqlConnection.Open();
245
246                     /// Check if order is already in use
247                     tOrderStatusResult = CheckIfOrderExist(sqlConnection, pSmileyQuestion.Order);
248
249                     /// return if order already exist
250                     if (tOrderStatusResult == ErrorCode.VALIDATION)
251                         return ErrorCode.VALIDATION;
252
253                     /// if order is not in use -> insert a question with the same order
254                     using (SqlCommand cmd = sqlConnection.CreateCommand())
255                     {
256                         cmd.CommandText = "dbo.INSERT_SMILEY_QUESTION";
257                         cmd.CommandType = CommandType.StoredProcedure;
258
259                         SqlParameter[] parameters = new SqlParameter[]
260                         {
261                             new SqlParameter($"{QuestionColumn.Order}", pSmileyQuestion.Order),
262                             new SqlParameter($"{QuestionColumn.Text}", pSmileyQuestion.Text),
263                             new SqlParameter($"{QuestionColumn.Type}", pSmileyQuestion.Type),
264                             new SqlParameter($"{QuestionColumn.NumberOfSmileyFaces}", pSmileyQuestion.NumberOfSmileyFaces)
265                         };
266                         cmd.Parameters.AddRange(parameters);
267                         tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
268
269                         if (tOrderStatusResult == ErrorCode.SUCCESS)
270                         {
271                             /// If everything is okay -> COMMIT Transaction
272                             transactionScope.Complete();
273                             return ErrorCode.SUCCESS;
274                         }
275                     }
276
277                 }
278             }
279         }
280     }
```

Parameters:

pSmileyQuestion-> SmileyQuestion

Description:

This method opens a transaction then an sql connection, then check if the order of the question to be inserted is taken or not.

Than it executes a stored procedure that insert a smiley question into 2 tables: “Questions” and “Smiley_Questions”

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

ErrorCode.VALIDATION

InsertSliderQuestion:

```
286     /// <summary>
287     /// 1) Open a transaction
288     /// 2) Check if desired order is already in use before inserting (SQL Function)
289     /// 3) Perform insert stored procedure -> get the result as ErrorCode
290     /// 4) return return from function with the corresponding ErrorCode
291     /// </summary>
292     /// <returns>
293     /// ErrorCode.SUCCESS
294     /// ErrorCode.ERROR
295     /// ErrorCode.VALIDATION
296     /// </returns>
297     public ErrorCode InsertSliderQuestion(SliderQuestion pSliderQuestion)
298     {
299
300         try
301         {
302             /// ErrorCode to be returned
303             ErrorCode tOrderStatusResult = ErrorCode.ERROR;
304
305             using (TransactionScope transactionScope = new TransactionScope())
306             {
307                 using (SqlConnection sqlConnection = new SqlConnection())
308                 {
309                     sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
310                     sqlConnection.Open();
311
312                     /// Check if order is already in use
313                     tOrderStatusResult = CheckIfOrderExist(sqlConnection, pSliderQuestion.Order);
314
315                     /// return if order already exist
316                     if (tOrderStatusResult == ErrorCode.VALIDATION)
317                         return ErrorCode.VALIDATION;
318
319                     /// if order is not in use -> insert a question with the same order
320                     using (SqlCommand cmd = sqlConnection.CreateCommand())
321                     {
322                         cmd.CommandText = @"dbo.INSERT_SLIDER_QUESTION";
323                         cmd.CommandType = CommandType.StoredProcedure;
324
325                         SqlParameter[] parameters = new SqlParameter[]
326                         {
327                             new SqlParameter($"{QuestionColumn.Text}", pSliderQuestion.Text),
328                             new SqlParameter($"{QuestionColumn.Order}", pSliderQuestion.Order),
329                             new SqlParameter($"{QuestionColumn.Type}", pSliderQuestion.Type),
330                             new SqlParameter($"{QuestionColumn.StartValue}", pSliderQuestion.StartValue),
331                             new SqlParameter($"{QuestionColumn.EndValue}", pSliderQuestion.EndValue),
332                             new SqlParameter($"{QuestionColumn.StartValueCaption}", pSliderQuestion.StartValueCaption),
333                             new SqlParameter($"{QuestionColumn.EndValueCaption}", pSliderQuestion.EndValueCaption)
334                         };
335                         cmd.Parameters.AddRange(parameters);
336
337                         tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
338                     }
339
340                     if (tOrderStatusResult == ErrorCode.SUCCESS)
341                     {
342                         /// If everything is okay -> COMMIT Transaction
343                         transactionScope.Complete();
344                     }
345
346                     return ErrorCode.ERROR;
347                 }
348             }
349         }
350     }
```

Parameters:

pSliderQuestion-> SliderQuestion

Description:

This method opens a transaction then an sql connection, then check if the order of the question to be inserted is taken or not.

Than it executes a stored procedure that insert a smiley question into 2 tables: “Questions” and “Slider_Questions”

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

ErrorCode.VALIDATION

InsertSliderQuestion

```
357     /// <summary>
358     /// 1) Open a transaction
359     /// 2) Check if desired order is already in use before inserting (SQL Function)
360     /// 3) Perform insert stored procedure -> get the result as ErrorCode
361     /// 4) return return from function with the corresponding ErrorCode
362     /// </summary>
363     /// <returns>
364     /// ErrorCode.SUCCESS
365     /// ErrorCode.ERROR
366     /// ErrorCode.VALIDATION
367     /// </returns>
368     1 reference
369     public ErrorCode InsertStarQuestion(StarQuestion pStarQuestion)
370     {
371         try
372         {
373             /// ErrorCode to be returned
374             ErrorCode tOrderStatusResult = ErrorCode.ERROR;
375
376             using (TransactionScope transactionScope = new TransactionScope())
377             {
378                 using (SqlConnection sqlConnection = new SqlConnection())
379                 {
380                     sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
381                     sqlConnection.Open();
382
383                     /// Check if order is already in use
384                     tOrderStatusResult = CheckIfOrderExist(sqlConnection, pStarQuestion.Order);
385
386                     /// return if order already exist
387                     if (tOrderStatusResult == ErrorCode.VALIDATION)
388                         return ErrorCode.VALIDATION;
389
390                     /// if order is not in use -> insert a question with the same order
391                     using (SqlCommand cmd = sqlConnection.CreateCommand())
392                     {
393                         cmd.CommandText = "dbo.INSERT_STAR_QUESTION";
394                         cmd.CommandType = CommandType.StoredProcedure;
395
396                         SqlParameter[] parameters = new SqlParameter[]
397                         {
398                             new SqlParameter($"{QuestionColumn.Order}", pStarQuestion.Order),
399                             new SqlParameter($"{QuestionColumn.Text}", pStarQuestion.Text),
400                             new SqlParameter($"{QuestionColumn.Type}", pStarQuestion.Type),
401                             new SqlParameter($"{QuestionColumn.NumberOfStars}", pStarQuestion.NumberOfStars)
402                         };
403                         cmd.Parameters.AddRange(parameters);
404                         tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
405                     }
406
407                     if (tOrderStatusResult == ErrorCode.SUCCESS)
408                     {
409                         /// If everything is okay -> COMMIT Transaction
410                         transactionScope.Complete();
411                         return ErrorCode.SUCCESS;
412                     }
413
414                     return ErrorCode.ERROR;
415                 }
416             }
417         }
418         catch (Exception ex)
```

Parameters:

pStarQuestion-> StarQuestion

Description:

This method opens a transaction then an sql connection, then check if the order of the question to be inserted is taken or not.

Than it executes a stored procedure that insert a smiley question into 2 tables: “Questions” and “Star_Questions”

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

ErrorCode.VALIDATION

UPDATE Methods

UpdateSmileyQuestion

```
426
427     #region UPDATE Methods
428
429     /// <summary>
430     /// 1) Open a transaction
431     /// 2) Check if desired order is already in use before inserting (SQL Function)
432     /// 3) Get ID of the desired order you want to insert
433     /// 4) if order already in use and the ID of that order is not the same ID the question that I am already updating
434     /// 5) -> Perform update stored procedure -> get the result as ErrorCode
435     /// 6) return from function with the corresponding ErrorCode
436     /// </summary>
437     /// <returns>
438     /// ErrorCode.SUCCESS
439     /// ErrorCode.ERROR
440     /// ErrorCode.VALIDATION
441     /// </returns>
442     public ErrorCode UpdateSmileyQuestion(SmileyQuestion pSmileyQuestion)
443     {
444         /// Try to Update a new pQuestion into "Smiley_Questions" table
445         try
446         {
447             ErrorCode tOrderStatusResult = ErrorCode.ERROR;
448             ErrorCode tIsEditingSameQuestion = ErrorCode.ERROR;
449
450             using (TransactionScope transactionScope = new TransactionScope())
451             {
452                 using (SqlConnection sqlConnection = new SqlConnection())
453                 {
454                     sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
455                     sqlConnection.Open();
456
457                     /// Check if order is already in use
458                     tOrderStatusResult = CheckIfOrderExist(sqlConnection, pSmileyQuestion.Order);
459
460                     /// get question ID form it's unique order
461                     tIsEditingSameQuestion = CheckIfUpdatingSameQuestion(sqlConnection, pSmileyQuestion.Order, pSmileyQuestion.ID);
462
463                     //return if order already exist && the order is taken by another questionID . . .
464                     if (tOrderStatusResult == ErrorCode.VALIDATION && tIsEditingSameQuestion == ErrorCode.ERROR)
465                         return ErrorCode.VALIDATION;
466
467
468                     using (SqlCommand cmd = sqlConnection.CreateCommand())
469                     {
470                         cmd.CommandText = "dbo.UPDATE_SMILEY_QUESTION";
471                         cmd.CommandType = CommandType.StoredProcedure;
472
473                         SqlParameter[] parameters = new SqlParameter[]
474                         {
475                             new SqlParameter("${QuestionColumn.ID}", pSmileyQuestion.ID),
476                             new SqlParameter("${QuestionColumn.Order}", pSmileyQuestion.Order),
477                             new SqlParameter("${QuestionColumn.Text}", pSmileyQuestion.Text),
478                             new SqlParameter("${QuestionColumn.Type}", pSmileyQuestion.Type),
479                             new SqlParameter("${QuestionColumn.NumberOfSmileyFaces}", pSmileyQuestion.NumberOfSmileyFaces)
480                         };
481                         cmd.Parameters.AddRange(parameters);
482                         tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
483
484                     if (tOrderStatusResult == ErrorCode.SUCCESS)
485                     {
486                         /// If everything is okay -> COMMIT Transaction
487                         transactionScope.Complete();
488                         return ErrorCode.SUCCESS;
489                     }
490                     return ErrorCode.ERROR;
491
492                 }
493             }
494         catch (Exception ex)
```

Parameters:

pSmileyQuestion-> SmileyQuestion

Description:

This method opens a transaction then an sql connection, then check if the order of the question to be inserted is taken or not., then check if it's updating the same question or not.

Then updates to tables: “Questions” and “Smiley_Questions” using stored procedure.

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

ErrorCode.VALIDATION

UpdateSliderQuestion:

```
502     /// <summary>
503     /// 1) Open a transaction
504     /// 2) Check if desired order is already in use before inserting (SQL Function)
505     /// 3) Get ID of the desired order you want to insert
506     /// 4) if order already in use and the ID of that order is not the same ID the question that I am already updating
507     /// 5) -> Perform update stored procedure -> get the result as ErrorCode
508     /// 6) return from function with the corresponding ErrorCode
509     /// </summary>
510     /// <returns>
511     /// ErrorCode.SUCCESS
512     /// ErrorCode.ERROR
513     /// ErrorCode.VALIDATION
514     /// </returns>
515     public ErrorCode UpdateSliderQuestion(SliderQuestion pSliderQuestion)
516     {
517         try
518         {
519             ErrorCode tOrderStatusResult = ErrorCode.ERROR;
520             ErrorCode tIsEditingSameQuestion = ErrorCode.ERROR;
521
522             using (TransactionScope transactionScope = new TransactionScope())
523             {
524                 using (SqlConnection sqlConnection = new SqlConnection())
525                 {
526                     sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
527                     sqlConnection.Open();
528
529                     /// Check if order is already in use
530                     tOrderStatusResult = CheckIfOrderExist(sqlConnection, pSliderQuestion.Order);
531
532                     /// get question ID form it's unique order
533                     tIsEditingSameQuestion = CheckIfUpdatingSameQuestion(sqlConnection, pSliderQuestion.Order, pSliderQuestion.ID);
534
535                     /// return if order already exist && the order is taken by another questionID ....
536                     if (tOrderStatusResult == ErrorCode.VALIDATION && tIsEditingSameQuestion == ErrorCode.ERROR)
537                         return ErrorCode.VALIDATION;
538
539
540                     using (SqlCommand cmd = sqlConnection.CreateCommand())
541                     {
542                         cmd.CommandText = "dbo.UPDATE_SLIDER_QUESTION";
543                         cmd.CommandType = CommandType.StoredProcedure;
544
545                         SqlParameter[] parameters = new SqlParameter[] {
546                             new SqlParameter($"{QuestionColumn.ID}", pSliderQuestion.ID),
547                             new SqlParameter($"{QuestionColumn.Order}", pSliderQuestion.Order),
548                             new SqlParameter($"{QuestionColumn.Text}", pSliderQuestion.Text),
549                             new SqlParameter($"{QuestionColumn.Type}", pSliderQuestion.Type),
550                             new SqlParameter($"{QuestionColumn.StartValue}", pSliderQuestion.StartValue),
551                             new SqlParameter($"{QuestionColumn.EndValue}", pSliderQuestion.EndValue),
552                             new SqlParameter($"{QuestionColumn.StartValueCaption}", pSliderQuestion.StartValueCaption),
553                             new SqlParameter($"{QuestionColumn.EndValueCaption}", pSliderQuestion.EndValueCaption),
554                         };
555                         cmd.Parameters.AddRange(parameters);
556                         tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
557                     }
558
559                     if (tOrderStatusResult == ErrorCode.SUCCESS)
560                     {
561                         /// If everything is okay -> COMMIT Transaction
562                         transactionScope.Complete();
563                         return ErrorCode.SUCCESS;
564                     }
565                     return ErrorCode.ERROR;
566                 }
567             }
568         }
569         catch (Exception ex)
```

Parameters:

pSliderQuestion-> SliderQuestion

Description:

This method opens a transaction then an sql connection, then check if the order of the question to be inserted is taken or not., then check if it's updating the same question or not.

Then updates to tables: “Questions” and “Slider_Questions” using stored procedure.

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

ErrorCode.VALIDATION

UpdateStarQuestion:

```
577     /// 1) Open a transaction
578     /// 2) Check if desired order is already in use before inserting (SQL Function)
579     /// 3) Get ID of the desired order you want to insert
580     /// 4) if order already in use and the ID of that order is not the same ID the question that I am already updating
581     /// 5) -> Perform update stored procedure -> get the result as ErrorCode
582     /// 6) return from function with the corresponding ErrorCode
583     /// </summary>
584     /// <returns>
585     /// ErrorCode.SUCCESS
586     /// ErrorCode.ERROR
587     /// ErrorCode.VALIDATION
588     /// </returns>
589     [reference]
590     public ErrorCode UpdateStarQuestion(StarQuestion pStarQuestion)
591     {
592         try
593         {
594             ErrorCode tOrderStatusResult = ErrorCode.ERROR;
595             ErrorCode tIsUpdatingSameQuestion = ErrorCode.ERROR;
596
597             using (TransactionScope transactionScope = new TransactionScope())
598             {
599                 using (SqlConnection sqlConnection = new SqlConnection())
600                 {
601                     sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
602                     sqlConnection.Open();
603
604                     /// Check if order is already in use
605                     tOrderStatusResult = CheckIfOrderExist(sqlConnection, pStarQuestion.Order);
606
607                     /// get question ID form it's unique order
608                     tIsUpdatingSameQuestion = CheckIfUpdatingSameQuestion(sqlConnection, pStarQuestion.Order, pStarQuestion.ID);
609
610                     /// return if order already exist && the order is taken by another questionID . . .
611                     if (tOrderStatusResult == ErrorCode.VALIDATION && tIsUpdatingSameQuestion == ErrorCode.ERROR)
612                         return ErrorCode.VALIDATION;
613
614                     using (SqlCommand cmd = sqlConnection.CreateCommand())
615                     {
616                         cmd.CommandText = "dbo.UPDATE_STAR_QUESTION";
617                         cmd.CommandType = CommandType.StoredProcedure;
618
619                         SqlParameter[] parameters = new SqlParameter[]
620                         {
621                             new SqlParameter("${QuestionColumn.ID}", pStarQuestion.ID),
622                             new SqlParameter("${QuestionColumn.Order}", pStarQuestion.Order),
623                             new SqlParameter("${QuestionColumn.Text}", pStarQuestion.Text),
624                             new SqlParameter("${QuestionColumn.Type}", pStarQuestion.Type),
625                             new SqlParameter("${QuestionColumn.NumberOfStars}", pStarQuestion.NumberOfStars)
626                         };
627                         cmd.Parameters.AddRange(parameters);
628                         tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
629
630                     if (tOrderStatusResult == ErrorCode.SUCCESS)
631                     {
632                         /// If everything is okay -> COMMIT Transaction
633                         transactionScope.Complete();
634                         return ErrorCode.SUCCESS;
635                     }
636                     return ErrorCode.ERROR;
637                 }
638             }
639         }
640         catch (Exception ex)
641         {
```

Parameters:

pStarQuestion-> StarQuestion

Description:

This method opens a transaction then an sql connection, then check if the order of the question to be inserted is taken or not., then check if it's updating the same question or not.

Then updates to tables: “Questions” and “Star_Questions” using stored procedure.

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

ErrorCode.VALIDATION

DELETE Methods

DeleteQuestionByID

```
649     /// <summary>
650     /// 1) Open an SQL connection
651     /// 2) Perform a delete query based on passed ID
652     /// 3) return error code based on affected rows number
653     /// </summary>
654     /// <returns>
655     /// ErrorCode.SUCCESS
656     /// ErrorCode.ERROR
657     /// ErrorCode.VALIDATION
658     /// </returns>
659     1 reference
660     public ErrorCode DeleteQuestionByID(int pQuestionId)
661     {
662         try
663         {
664             using (SqlConnection sqlConnection = new SqlConnection(mSqlConnectionSettings.ConnectionString))
665             {
666                 sqlConnection.Open();
667                 using (SqlCommand cmd = sqlConnection.CreateCommand())
668                 {
669                     cmd.CommandText = $""
670                     | delete from Questions where ID = @{QuestionColumn.ID}";
671
672                     SqlParameter[] parameters = new SqlParameter[] {
673                         new SqlParameter($"{QuestionColumn.ID}", pQuestionId),
674                     };
675                     cmd.Parameters.AddRange(parameters);
676
677                     return cmd.ExecuteNonQuery() > 0 ? Generic.ErrorCode.SUCCESS : Generic.ErrorCode.VALIDATION;
678                 }
679             }
680         }
681         catch (Exception ex)
682         {
683             Logger.LogError(ex); // write error to log file
684             return Generic.ErrorCode.ERROR;
685         }
686     }
687     } /// Function end
```

Parameters:

pQuestionId-> int

Description:

This function opens an sql connection and try to execute a query which deletes a question based on the passed ID.

Return value based on the number of rows affected by the operation

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

ErrorCode.VALIDATION

GET Methods

GetAllQuestions

```
692     #region GET Methods
693
694     /// <summary>
695     /// 1) Open an SQL connection
696     /// 2) Get all questions from DB
697     /// 3) Create corresponding question objects and fill the passed question list with them
698     /// </summary>
699     /// <returns>
700     /// ErrorCode.SUCCESS
701     /// ErrorCode.EMPTY
702     /// ErrorCode.ERROR
703     /// </returns>
704     public ErrorCode GetAllQuestions(ref List<Question> pQuestionsList)
705     {
706         try
707         {
708             using (SqlConnection sqlConnection = new SqlConnection())
709             {
710                 UpdateGlobalConnectionSettings();
711                 sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
712                 sqlConnection.Open();
713
714                 using (SqlCommand cmd = sqlConnection.CreateCommand())
715                 {
716                     cmd.CommandText = $@"
717                         SELECT [ID], [Order], [Text], [Type] FROM Questions ORDER BY [ORDER] ASC;";
718
719                     DataTable dataTable = new DataTable();
720                     SqlDataAdapter adapter = new SqlDataAdapter(cmd);
721                     adapter.Fill(dataTable);
722
723                     int tID, tOrder;
724                     string tText;
725                     QuestionType tType;
726                     foreach (DataRow row in dataTable.Rows)
727                     {
728                         tID = (int)row["{QuestionColumn.ID}"];
729                         tOrder = (int)row["{QuestionColumn.Order}"];
730                         tText = (string)row["{QuestionColumn.Text}"];
731                         tType = (QuestionType)row["{QuestionColumn.Type}"];
732
733                         Question q = new Question(tID, tOrder, tText, tType);
734                         pQuestionsList.Add(q);
735                     }
736
737                     if (pQuestionsList.Count == 0)
738                     {
739                         return ErrorCode.EMPTY;
740                     }
741                     else if (pQuestionsList.Count > 0)
742                     {
743                         return ErrorCode.SUCCESS;
744                     }
745
746                     return ErrorCode.ERROR;
747                 }
748             }
749         }
750         catch (Exception ex)
```

Parameters:

ref pQuestionsList-> List<Question>.

Description:

This method selects all columns from “Questions” table and fill passed pQuestionsList with the returned data.

Returns:

ErrorCode.SUCCESS

ErrorCode.EMPTY

ErrorCode.ERROR

GetSmileyQuestionByID

```
757     /// <summary>
758     /// 1) Open an SQL connection
759     /// 2) Get corresponding question details from DB
760     /// 3) Fill the passed question object properties
761     /// </summary>
762     /// <returns>
763     /// ErrorCode.SUCCESS
764     /// ErrorCode.ERROR
765     /// </returns>
766     1 reference
767     public ErrorCode GetSmileyQuestionByID(ref SmileyQuestion pSmileyQuestion)
768     {
769         try
770         {
771             using (SqlConnection sqlConnection = new SqlConnection())
772             {
773                 sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
774                 sqlConnection.Open();
775
776                 using (SqlCommand cmd = sqlConnection.CreateCommand())
777                 {
778                     cmd.CommandText = $@"
779                         select Q.[ID], Q.[Order], Q.[Text], Q.[Type], SmQ.NumberOfSmileyFaces
780                         from Questions AS Q
781                         inner join Smiley_Questions AS SmQ
782                         on Q.ID = SmQ.ID
783                         where Q.ID = @{QuestionColumn.ID}
784                         ORDER BY [ORDER] ASC";
785
786                     SqlParameter[] parameters = new SqlParameter[]
787                     {
788                         new SqlParameter($"{QuestionColumn.ID}", pSmileyQuestion.ID),
789                     };
790                     cmd.Parameters.AddRange(parameters);
791
792                     DataTable dataTable = new DataTable();
793                     SqlDataAdapter adapter = new SqlDataAdapter(cmd);
794                     adapter.Fill(dataTable);
795
796                     int tID, tOrder, tNumberOfSmileyFaces;
797                     string tText;
798                     QuestionType tType;
799                     foreach (DataRow row in dataTable.Rows)
800                     {
801                         tID = (int)row[$"{QuestionColumn.ID}"];
802                         tOrder = (int)row[$"{QuestionColumn.Order}"];
803                         tText = (string)row[$"{QuestionColumn.Text}"];
804                         tType = (QuestionType)row[$"{QuestionColumn.Type}"];
805                         tNumberOfSmileyFaces = (int)row[$"{QuestionColumn.NumberOfSmileyFaces}"];
806                         pSmileyQuestion = new SmileyQuestion(tID, tOrder, tText, tType, tNumberOfSmileyFaces);
807                     }
808
809                     return pSmileyQuestion.NumberOfSmileyFaces > 0 ? Generic.ErrorCode.SUCCESS : Generic.ErrorCode.VALIDATION; // RETURN INT32
810
811                 }
812             }
813             catch (Exception ex)
```

Parameters:

`ref pSmileyQuestion-> SmileyQuestion.`

Description:

This method selects ID, Order, Text and Type columns from “Questions” table and NumberOfSmileyFaces column from “Smiley_questions” table.

And fill the data with passed object.

Returns VALIDATION if the question was not found (deleted for example)

Returns:

ErrorCode.SUCCESS

ErrorCode.VALIDATION

ErrorCode.ERROR

GetSliderQuestionByID:

```
818     /// <summary>
819     /// 1) Open an SQL connection
820     /// 2) Get corresponding question details from DB
821     /// 3) Fill the passed question object properties
822     /// </summary>
823     /// <returns>
824     /// ErrorCode.SUCCESS
825     /// ErrorCode.ERROR
826     /// </returns>
827     [reference]
828     public ErrorCode GetSliderQuestionByID(ref SliderQuestion pSliderQuestion)
829     {
830         try
831         {
832             using (SqlConnection sqlConnection = new SqlConnection())
833             {
834                 sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
835                 sqlConnection.Open();
836
837                 using (SqlCommand cmd = sqlConnection.CreateCommand())
838                 {
839                     cmd.CommandText = $@"
840                         select Q.[ID], Q.[Order], Q.[Text], Q.[Type], SQ.StartValue, SQ.EndValue, SQ.StartValueCaption, SQ.EndValueCaption
841                         from Questions AS Q
842                         inner join Slider_Questions AS SQ
843                         on Q.ID = SQ.ID
844                         where Q.ID = @[{QuestionColumn.ID}]
845                         ORDER BY [ORDER] ASC";
846
847                     SqlParameter[] parameters = new SqlParameter[] {
848                         new SqlParameter($"@{QuestionColumn.ID}", pSliderQuestion.ID),
849                     };
850                     cmd.Parameters.AddRange(parameters);
851
852                     DataTable dataTable = new DataTable();
853                     SqlDataAdapter adapter = new SqlDataAdapter(cmd);
854                     adapter.Fill(dataTable);
855
856
857                     int tID, tOrder, tStartValue, tEndValue;
858                     string tText, tStartValueCaption, tEndValueCaption;
859                     QuestionType tType;
860                     foreach (DataRow row in dataTable.Rows)
861                     {
862                         tID = (int)row[$"{QuestionColumn.ID}"];
863                         tOrder = (int)row[$"{QuestionColumn.Order}"];
864                         tText = (string)row[$"{QuestionColumn.Text}"];
865                         tType = (QuestionType)row[$"{QuestionColumn.Type}"];
866                         tStartValue = (int)row[$"{QuestionColumn.StartValue}"];
867                         tEndValue = (int)row[$"{QuestionColumn.EndValue}"];
868                         tStartValueCaption = (string)row[$"{QuestionColumn.StartValueCaption}"];
869                         tEndValueCaption = (string)row[$"{QuestionColumn.EndValueCaption}"];
870
871                         pSliderQuestion = new SliderQuestion(tID, tOrder, tText, tType,
872                             tStartValue, tEndValue, tStartValueCaption, tEndValueCaption);
873                     }
874
875                     return pSliderQuestion.StartValue > 0 ? Generic.ErrorCode.SUCCESS : Generic.ErrorCode.VALIDATION; // RETURN INT32
876                 }
877             }
878         catch (Exception ex)
879         {

```

Parameters:

ref pSliderQuestion-> SliderQuestion.

Description:

This method selects ID, Order, Text and Type columns from “Questions” table and StartValue, EndValue, StartValueCaption and EndValueCaption columns from “Slider_questions” table.

And fill the data with passed object.

Returns VALIDATION if the question was not found (deleted for example)

Returns:

ErrorCode.SUCCESS

ErrorCode.VALIDATION

ErrorCode.ERROR

GetStarQuestionByID

```
886     /// <summary>
887     /// 1) Open an SQL connection
888     /// 2) Get corresponding question details from DB
889     /// 3) Fill the passed question object properties
890     /// </summary>
891     /// <returns>
892     /// ErrorCode.SUCCESS
893     /// ErrorCode.VALIDATION
894     /// ErrorCode.ERROR
895     /// </returns>
896     1 reference
897     public ErrorCode GetStarQuestionByID(ref StarQuestion pStarQuestion)
898     {
899         try
900         {
901             using (SqlConnection sqlConnection = new SqlConnection())
902             {
903                 sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
904                 sqlConnection.Open();
905
906                 using (SqlCommand cmd = sqlConnection.CreateCommand())
907                 {
908                     cmd.CommandText = $@"
909                         select Q.[ID], Q.[Order], Q.[Text], Q.[Type], StQ.NumberOfStars
910                         from Questions AS Q
911                         inner join Star_Questions AS StQ
912                         on Q.ID = StQ.ID
913                         where Q.ID = @{QuestionColumn.ID}
914                         ORDER BY [ORDER] ASC";
915
916                     SqlParameter[] parameters = new SqlParameter[]
917                     {
918                         new SqlParameter($"{QuestionColumn.ID}", pStarQuestion.ID),
919                     };
920                     cmd.Parameters.AddRange(parameters);
921
922                     DataTable dataTable = new DataTable();
923                     SqlDataAdapter adapter = new SqlDataAdapter(cmd);
924                     adapter.Fill(dataTable);
925
926                     int tID, tOrder, tNumberOfStars;
927                     string tText;
928                     QuestionType tType;
929
930                     foreach (DataRow row in dataTable.Rows)
931                     {
932                         tID = (int)row["{QuestionColumn.ID}"];
933                         tOrder = (int)row["{QuestionColumn.Order}"];
934                         tText = (string)row["{QuestionColumn.Text}"];
935                         tType = (QuestionType)row["{QuestionColumn.Type}"];
936                         tNumberOfStars = (int)row["{QuestionColumn.NumberOfStars}"];
937
938                         pStarQuestion = new StarQuestion(tID, tOrder, tText, tType, tNumberOfStars);
939                     }
940
941                 }
942             }
943         }
944         catch (Exception ex)
```

Parameters:

`ref pStarQuestion-> StarQuestion.`

Description:

This method selects ID, Order, Text and Type columns from “Questions” table and NumberOfStars column from “Star_questions” table.

And fill the data with passed object.

Returns VALIDATION if the question was not found (deleted for example)

Returns:

ErrorCode.SUCCESS

ErrorCode.VALIDATION

ErrorCode.ERROR

1.8 Entities

1.8.1 Generic

Code:

```
7  namespace SurveyQuestionsConfigurator.Entities
8  {
9      public class Generic
10     {
11         public enum ErrorCode
12         {
13             ERROR = -1,
14             SUCCESS = 1,
15             VALIDATION = 2,
16             EMPTY = 3
17         }
18
19         public enum QuestionType
20         {
21             SMILEY = 0,
22             SLIDER = 1,
23             STAR = 2
24         }
25         public enum QuestionColumn
26         {
27             ID,
28             Order,
29             Text,
30             Type,
31             NumberOfSmileyFaces,
32             NumberOfStars,
33             StartValue,
34             EndValue,
35             StartValueCaption,
36             EndValueCaption,
37             ReturnValue
38         }
39     }
40 }
```

Generic data types that are used across all the application.

1.8.2 Question

Code:

```
9  Enamespace SurveyQuestionsConfigurator.Entities
10 {
11     Epublic class Question
12     {
13         E     Epublic int ID { get; set; }
14         E     Epublic int Order { get; set; }
15         E     Epublic string Text { get; set; }
16         E     Epublic QuestionType Type { get; set; }
17
18     E     Epublic Question(int pID)
19     E     {
20         E         Etry
21         E         {
22             E             ID = pID;
23         E         }
24         E         Ecatch (Exception ex)
25         E         {
26             E             Logger.LogError(ex);
27         E         }
28     E     }
29
30     E     Epublic Question(int pID, QuestionType pType)
31     E     {
32         E         Etry
33         E         {
34             E             ID = pID;
35             E             Type = pType;
36         E         }
37         E         Ecatch (Exception ex)
38         E         {
39             E             Logger.LogError(ex);
40         E         }
41     E     }
42
43     E     Epublic Question(int pID, int pOrder, string pText, QuestionType pType)
44     E     {
45         E         Etry
46         E         {
47             E             ID = pID;
48             E             Order = pOrder;
49             E             Text = pText;
50             E             Type = pType;
51         E         }
52         E         Ecatch (Exception ex)
53         E         {
54             E             Logger.LogError(ex);
55         E         }
56     E     }
57
58     E     Epublic Question(Question pQuestion) :
59     E     {
60         E         this(pQuestion.ID, pQuestion.Order, pQuestion.Text, pQuestion.Type)
61     E     }
62 }
```

```

63     0 references
64     public override bool Equals(object pObject)
65     {
66         try
67         {
68             if (
69                 pObject == null ||
70                 this.GetType() != pObject.GetType()
71             )
72             {
73                 return false;
74             }
75
76             Question q = (Question)pObject;
77             if (
78                 q.ID == this.ID &&
79                 q.Order == this.Order &&
80                 q.Text == this.Text
81             )
82             {
83                 return true;
84             }
85
86             return false;
87         }
88         catch (Exception ex)
89         {
90             Logger.LogError(ex);
91             return false;
92         }
93     }
94
95     /// <summary>
96     /// Must be overridden along with Equals
97     /// </summary>
98     0 references
99     public override int GetHashCode()
100    {
101        return base.GetHashCode();
102    }

```

Description:

This generic question type class is inherited by other classes (other question types) and it overrides the “Equals” methods to compare 2 objects of the same type.

It accepts multiple constructors, each one was needed in a specific occasion.

1.8.3 SmileyQuestion

Inherits “Question” class and modify its corresponding constructors.

1.8.4 SliderQuestion

Inherits “Question” class and modify its corresponding constructors.

1.8.5 StarQuestion

Inherits “Question” class and modify its corresponding constructors.

1.9 Common Helpers

```
8
9  namespace SurveyQuestionsConfigurator.CommonHelpers
10 {
11     public class Logger
12     {
13         private static string mExeFolder = System.IO.Path.GetDirectoryName(Assembly.GetEntryAssembly().Location); //Gets the directory name of the current exe
14         private static object mBalanceLock = new object(); // used to lock writing on file
15         private static object mBalanceLock2 = null; // used to lock writing on file for the backup function
16
17         public static void LogError(Exception pEx)...
18
19         /// Log error in case main "LogError" faced an exception
20         public static void BackUpLogger(Exception pEx)...
21
22     }
23
24 }
```

1.9.1 LogError:

```
17  public static void LogError(Exception pEx)
18  {
19      try
20      {
21          // lock file writing to prevent exception System.IO.IOException :
22          // The process cannot access the file 'LogFile.txt' because it is being used by another process
23          lock (mBalanceLock)
24          {
25              using (StreamWriter sw = new StreamWriter($"{mExeFolder}/logs/LogFile.txt", append: true))
26              {
27                  sw.WriteLine($"----- ({DateTime.Now}) -----");
28
29 -Exception Type: {pEx.GetType()}
30 -Exception Call Site: {pEx.TargetSite}
31 -Exception Short Message: {pEx.Message}
32 -Exception Long Message: {pEx}
33 -Exception Stack Trace: {pEx.StackTrace}
34 ");
35
36          }
37      }
38      catch (Exception ex2)
39      {
40          BackUpLogger(ex2);
41      }
42  }
43 }
```

Parameters:

pEx-> Exception

Description:

This function logs any caught error to a log file and write on it using StreamWriter class.

If it fails-> a backup logger will catch and log the error instead.

1.9.2 BackUpLogger:

```
45  | // Log error in case main ".LogError" faced an exception
46  | 1 reference
47  | public static void BackUpLogger(Exception pEx)
48  | {
49  |     try
50  |     {
51  |         mBalanceLock2 = new object();
52  |         lock (mBalanceLock2)
53  |         {
54  |             using (StreamWriter sw = new StreamWriter($"{mExeFolder}/logs/LogFile.txt", append: true))
55  |             {
56  |                 sw.WriteLine($"----- ({DateTime.Now}) -----");
57  |                 -Exception Type: {pEx.GetType()}
58  |                 -Exception Call Site: {pEx.TargetSite}
59  |                 -Exception Short Message: {pEx.Message}
60  |                 -Exception Long Message: {pEx}
61  |                 -Exception Stack Trace: {pEx.StackTrace}
62  |             ");
63  |         }
64  |     }
65  | }
66  | catch
67  | {
68  | }
69  |
70  | }
```

Parameters:

pEx-> Exception

Description:

This function is called in the catch block of the “.LogError” function if it faced an exception.