Survey Question Configurator

Code Documentation

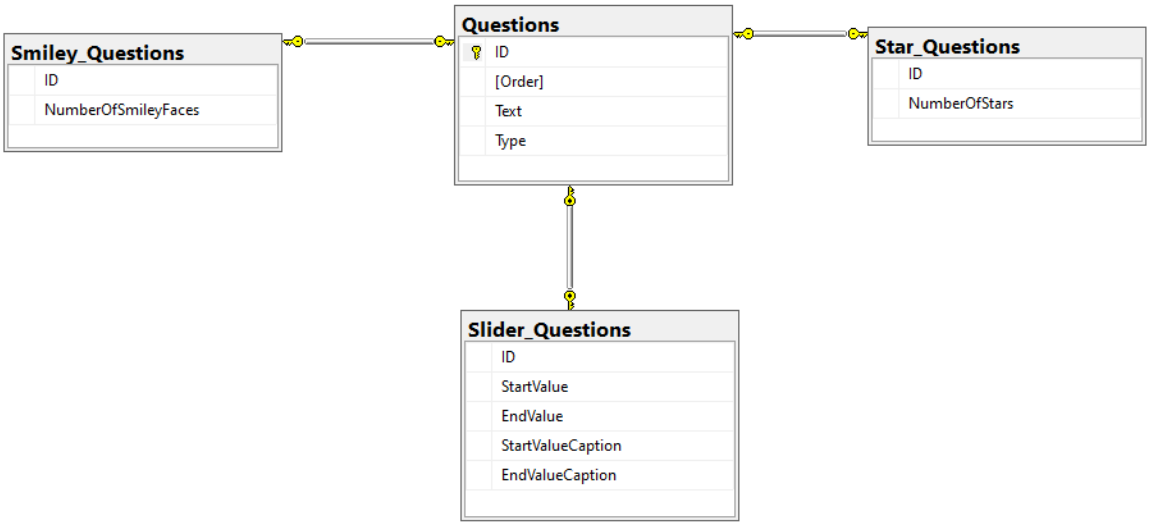
Table of Contents

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:

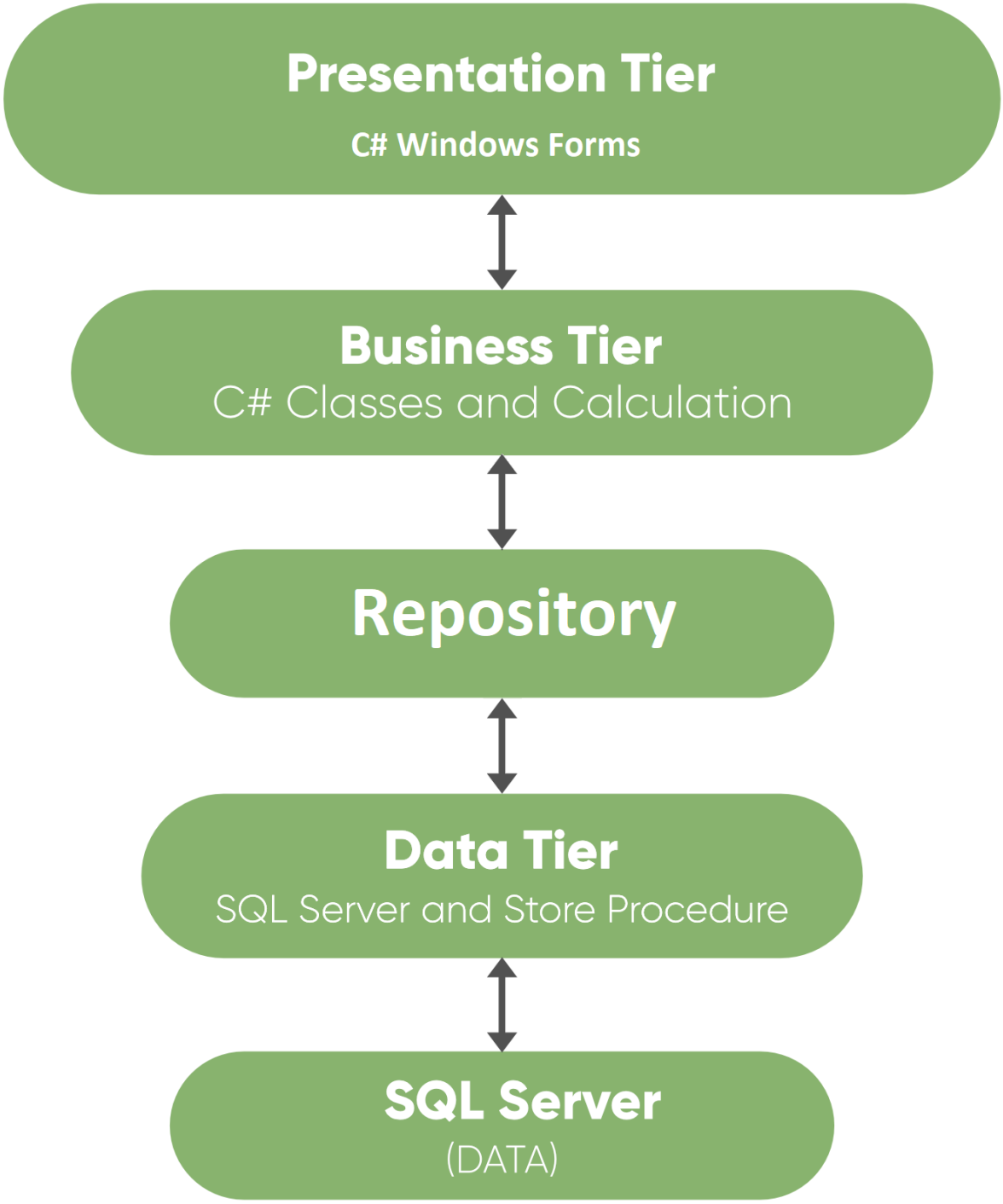
The code is decoupled into 3 layers:

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 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:

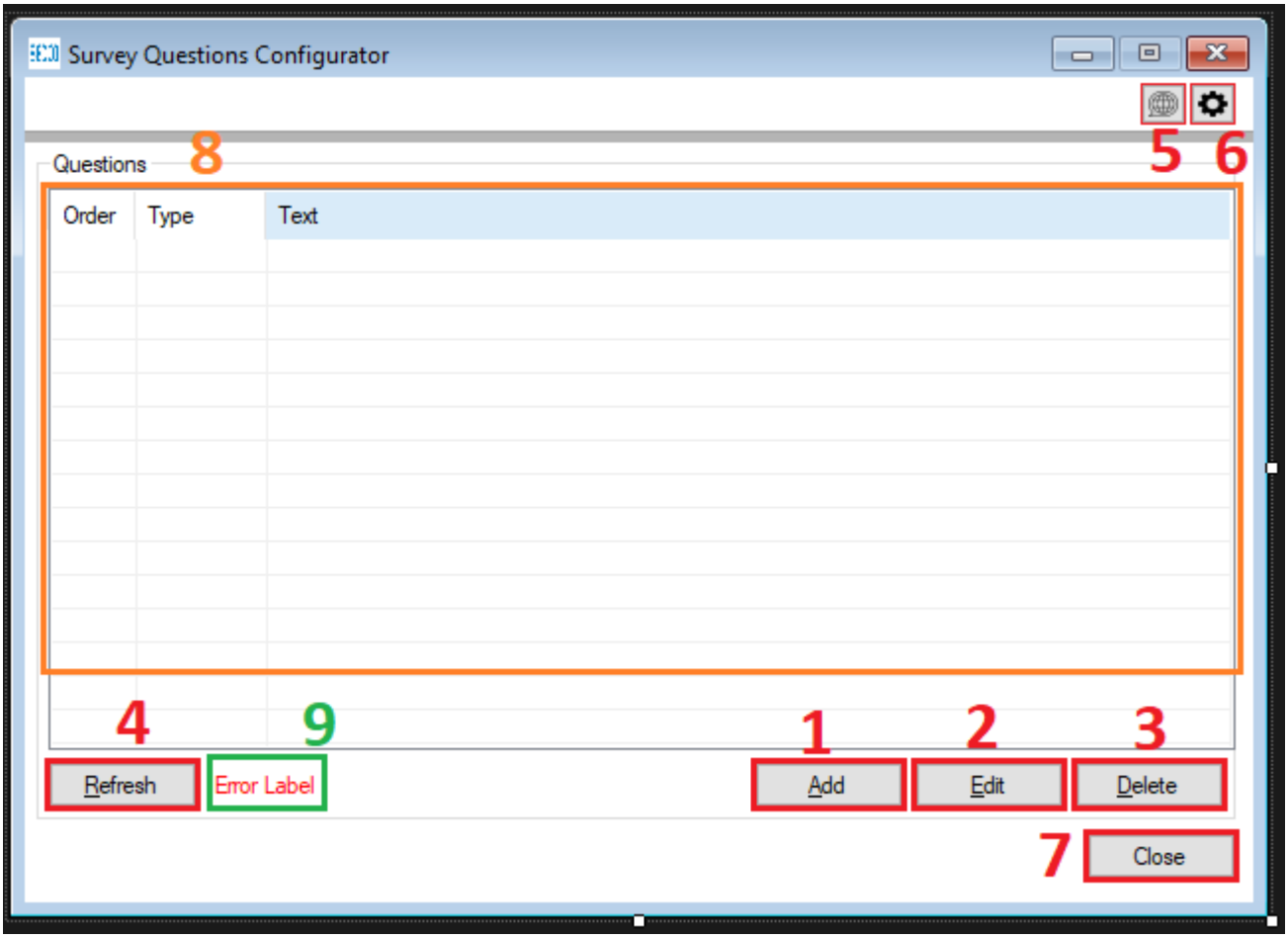
1. SurveyQuestionConfiguratorForm
2. AddQuestionForm
3. LanguageSettingsForm
4. ConnectionSettingsForm

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

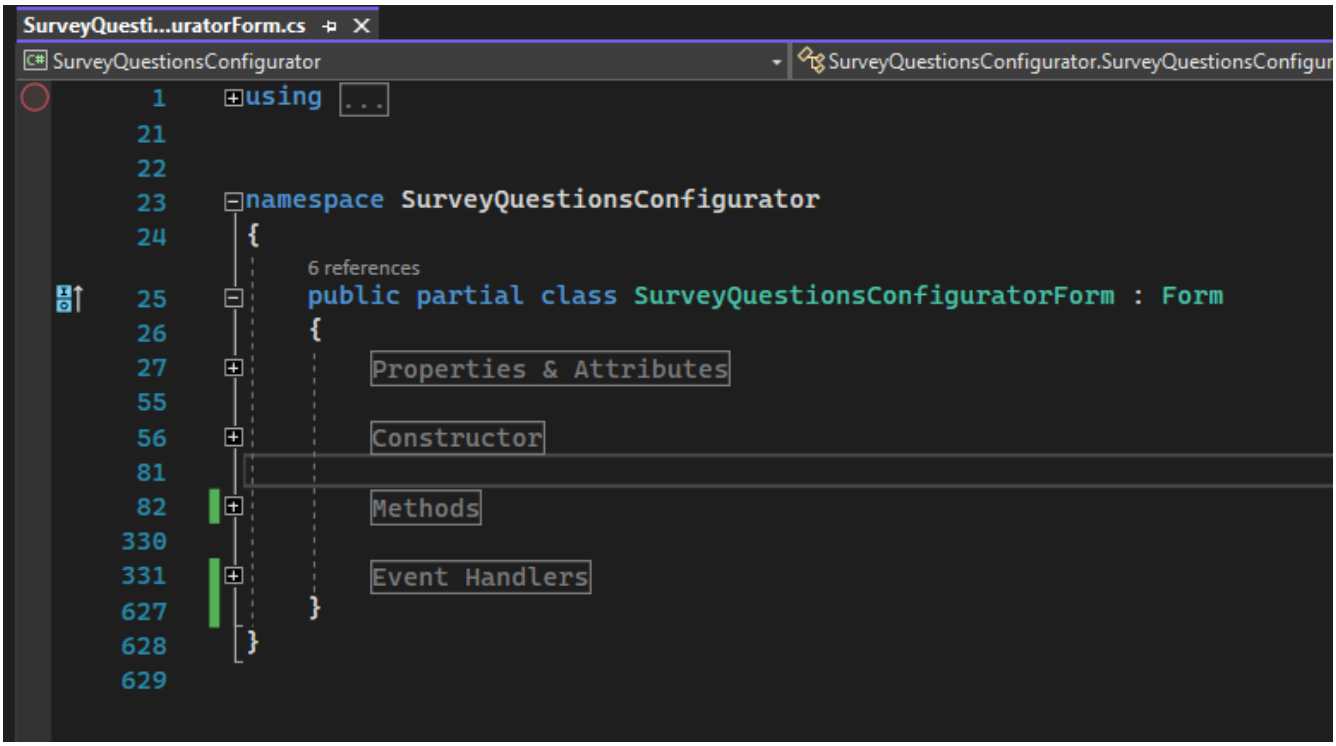
1. Add a question.
2. Edit a question.
3. Deleted a question.
4. Refresh the questions list.
5. Change language settings.
6. Change connection settings.
7. Close the application.
8. ListView
9. Label

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.

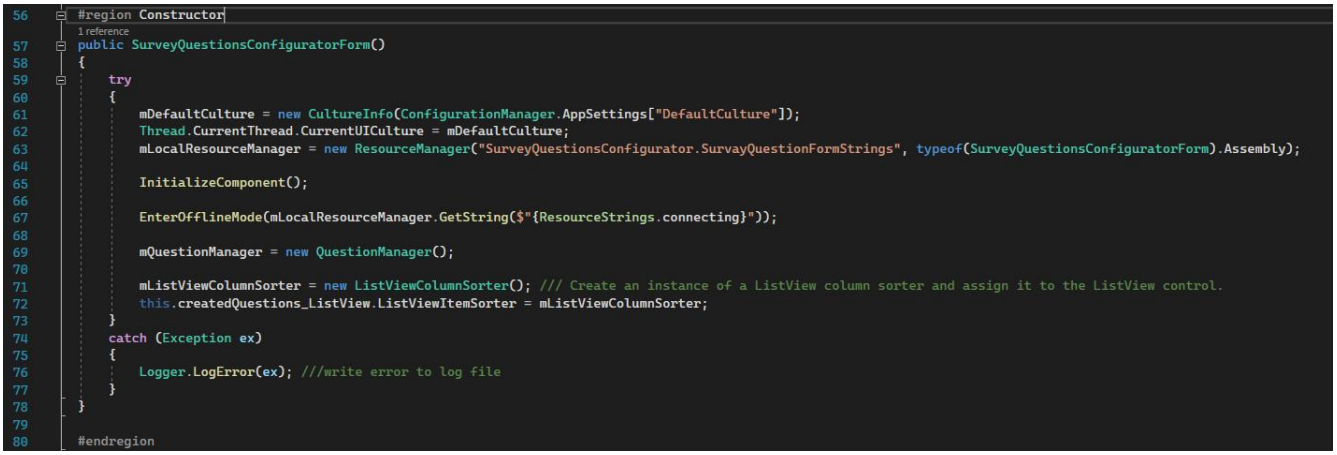


Properties & Attributes

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

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

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

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

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

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

Methods

ClearListView

```
82  #region Methods
83
84  /// <summary> Clear list view by deleting all of it's rows
    3 references
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.

Line 93-> 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).

Line 103-> 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>
        1 reference
125     public void BuildListView(ErrorCode pErrorCode, List<Question> pQuestionList)
126     {
127         /// Connect to quesion table and fill the list view
128         try
129         {
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.youAreOffilneError}");
158                         EnterOfflineMode(tOfflineMessage);
159                         break;
160                     }
```

Parameters:

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

pQuestionList-> 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).

Line 137-> make the form go into online mode which is making all controls active to the user to interact with.

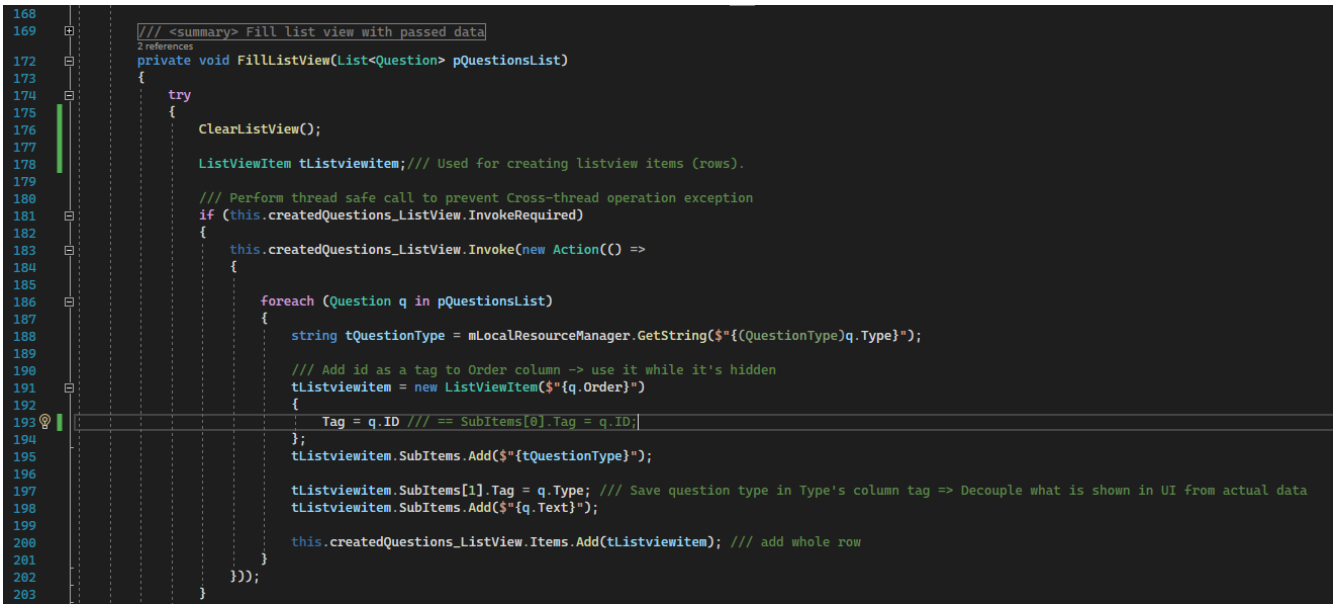
Line 144&149-> fill the list with actual data.

Line 157-> 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)

Line 58-> 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
            2 references
172         private void FillListView(List<Question> pQuestionsList)
173         {
174             try
175             {
176                 ClearListView();
177
178                 ListViewItem tListviewitem;/// Used for creating listview items (rows).
179
180                 /// Perform thread safe call to prevent Cross-thread operation exception
181                 if (this.createdQuestions_ListView.InvokeRequired)
182                 {
183                     this.createdQuestions_ListView.Invoke(new Action(() =>
184                     {
185
186                         foreach (Question q in pQuestionsList)
187                         {
188                             string tQuestionType = mLocalResourceManager.GetString($"{(QuestionType)q.Type}");
189
190                             /// Add id as a tag to Order column -> use it while it's hidden
191                             tListviewitem = new ListViewItem($"{q.Order}")
192                             {
193                                 Tag = q.ID /// == SubItems[0].Tag = q.ID;
194                             };
195                             tListviewitem.SubItems.Add($"{tQuestionType}");
196
197                             tListviewitem.SubItems[1].Tag = q.Type; /// Save question type in Type's column tag => Decouple what is shown in UI from actual data
198                             tListviewitem.SubItems.Add($"{q.Text}");
199
200                             this.createdQuestions_ListView.Items.Add(tListviewitem); /// add whole row
201                         }
202                     }));
203                 }
```

Parameters:

pQuestionList-> list of questions to be rendered to users.

Description:

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

Line 188-> gets the question type string to be shown in the “Type” column based on the current select language.

Line 191-> create new ListViewItem with the first element to be show is the order of the question.

Line 193-> 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.

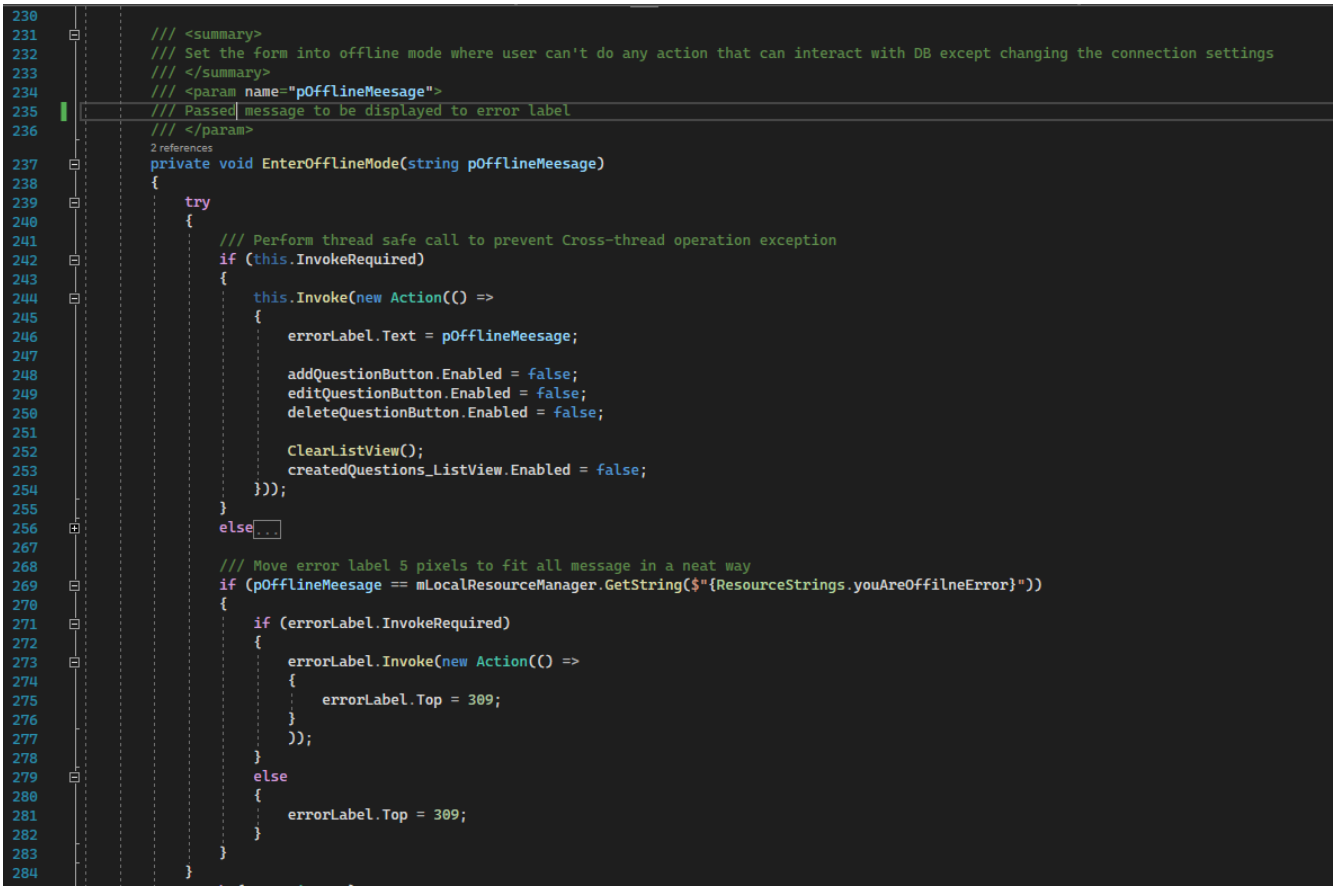
Line 197-> save the type of the question in the “Tag” property because it is different than what is shown in the list view so it can’t be taken directly from it and do DB operations with it.

Line 200-> 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>
            2 references
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...
267
268                 /// Move error label 5 pixels to fit all message in a neat way
269                 if (pOfflineMeesage == mLocalResourceManager.GetString($"{ResourceStrings.youAreOffilneError}"))
270                 {
271                     if (errorLabel.InvokeRequired)
272                     {
273                         errorLabel.Invoke(new Action(() =>
274                         {
275                             errorLabel.Top = 309;
276                         }
277                         ));
278                     }
279                     else
280                     {
281                         errorLabel.Top = 309;
282                     }
283                 }
284             }
```

Parameters:

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

Description:

Line 244-> checks if the form needs to be invoked and changed multiple controls at once.

Line 269-> 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>
            1 reference
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:

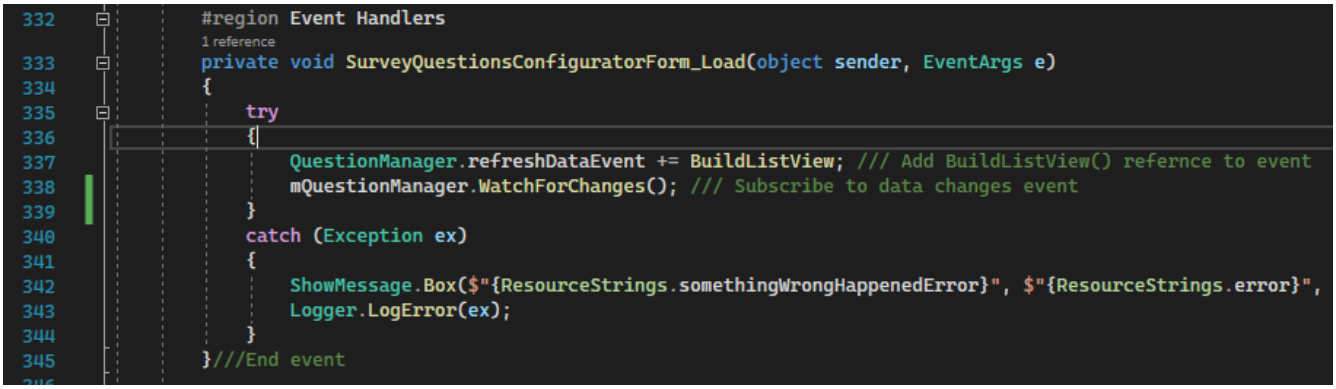
Returns all controls enabled back.

Returns:

None.

Event Handlers:

SurveyQuestionsConfiguratorForm_Load:



```
#region Event Handlers
1 reference
private void SurveyQuestionsConfiguratorForm_Load(object sender, EventArgs e)
{
    try
    {
        QuestionManager.refreshDataEvent += BuildListView; /// Add BuildListView() refernce to event
        mQuestionManager.WatchForChanges(); /// Subscribe to data changes event
    }
    catch (Exception ex)
    {
        ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}",
        Logger.LogError(ex);
    }
}///End event
```

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

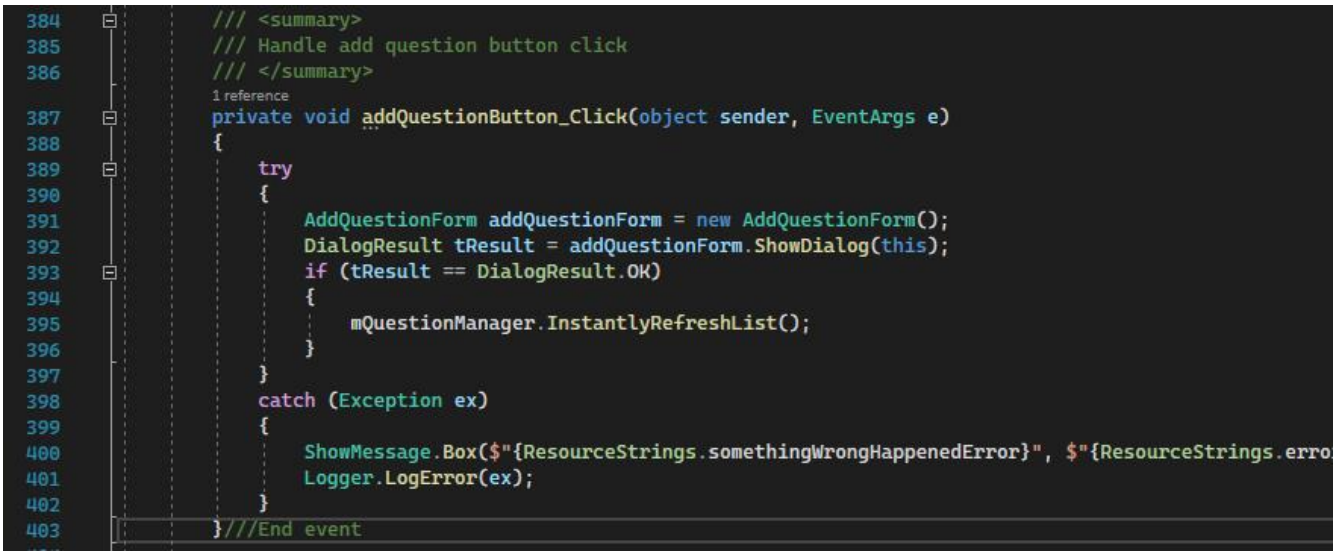
```
347         /// <summary>
348         /// Sort list for each column
349         /// </summary>
            1 reference
350         private void createdQuestions_ListView_ColumnClick(object sender, ColumnClickEventArgs e)
351         {
352             try
353             {
354                 /// 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.Er
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



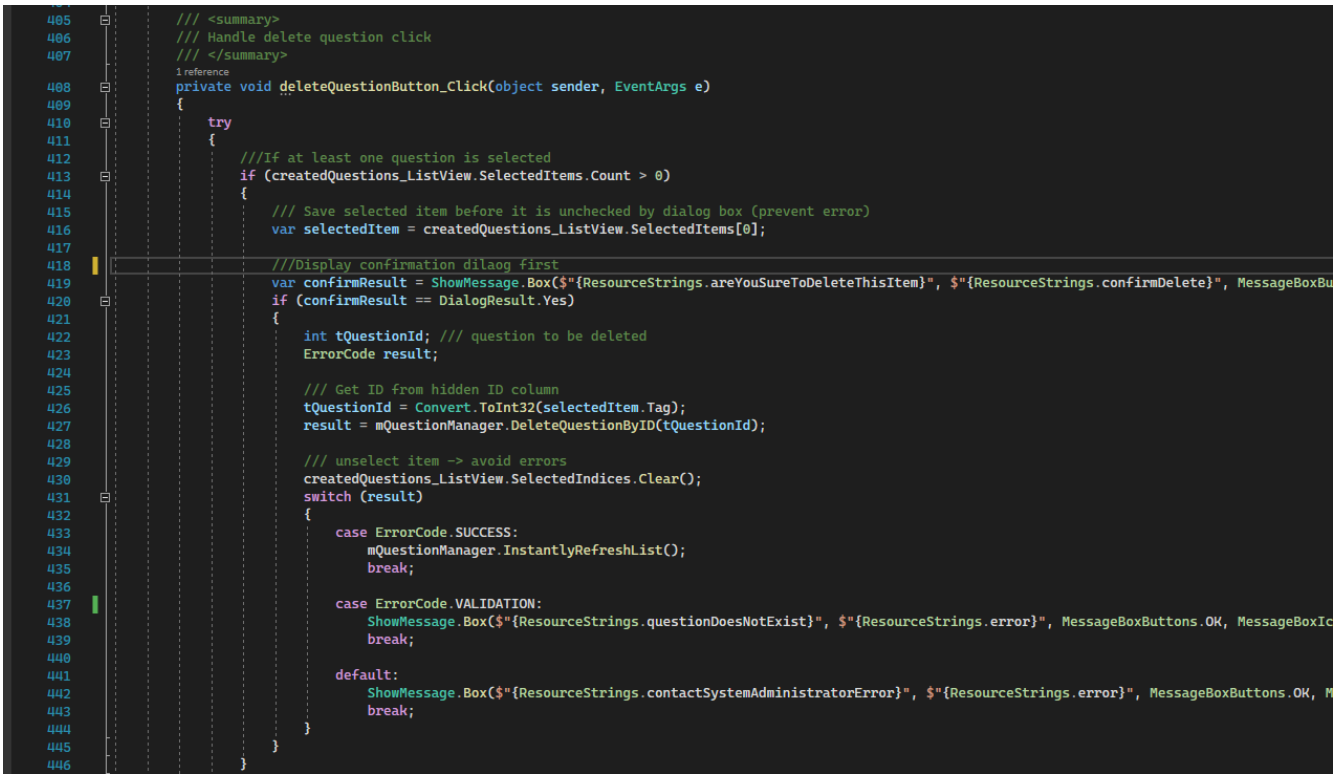
```
384     /// <summary>
385     /// Handle add question button click
386     /// </summary>
        1 reference
387     private void addQuestionButton_Click(object sender, EventArgs e)
388     {
389         try
390         {
391             AddQuestionForm addQuestionForm = new AddQuestionForm();
392             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.erro
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.

Line 395-> 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>
        1 reference
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 dilaog first
419                 var confirmResult = ShowMessage.Box($"{ResourceStrings.areYouSureToDeleteThisItem}", $"{ResourceStrings.confirmDelete}", MessageBoxBu
420                 if (confirmResult == DialogResult.Yes)
421                 {
422                     int tQuestionId; /// question to be deleted
423                     ErrorCode result;
424
425                     /// Get ID from hidden ID column
426                     tQuestionId = Convert.ToInt32(selectedItem.Tag);
427                     result = mQuestionManager.DeleteQuestionByID(tQuestionId);
428
429                     /// unselect item -> avoid errors
430                     createdQuestions_ListView.SelectedIndices.Clear();
431                     switch (result)
432                     {
433                         case ErrorCode.SUCCESS:
434                             mQuestionManager.InstantlyRefreshList();
435                             break;
436
437                         case ErrorCode.VALIDATION:
438                             ShowMessage.Box($"{ResourceStrings.questionDoesNotExist}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIc
439                             break;
440
441                         default:
442                             ShowMessage.Box($"{ResourceStrings.contactSystemAdministratorError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, M
443                             break;
444                     }
445                 }
446             }
```

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.

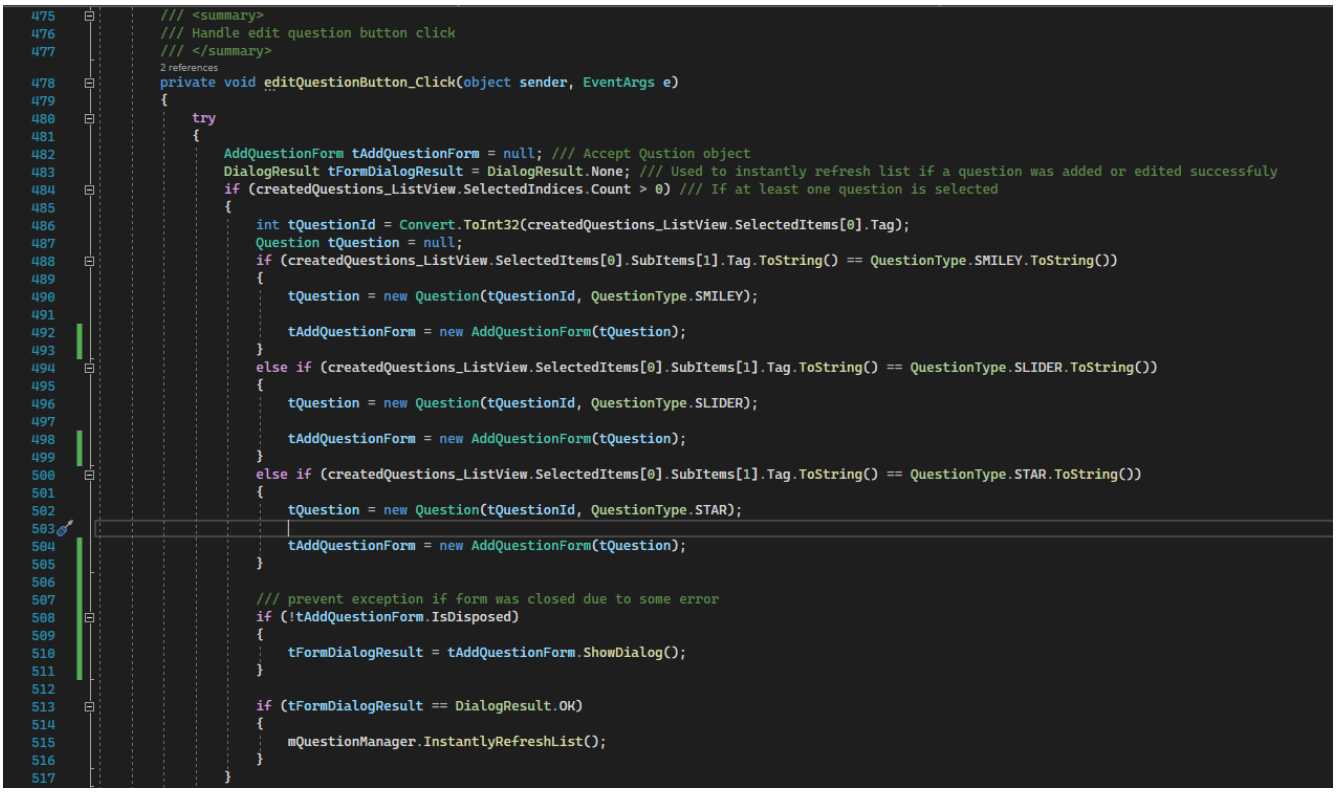
refreshDataButton_Click

```
459         /// <summary>
460         /// Handle reresh button click
461         /// </summary>
            1 reference
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>
            2 references
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 successfuly
484                 if (createdQuestions_ListView.SelectedIndices.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                 }
```

Description:

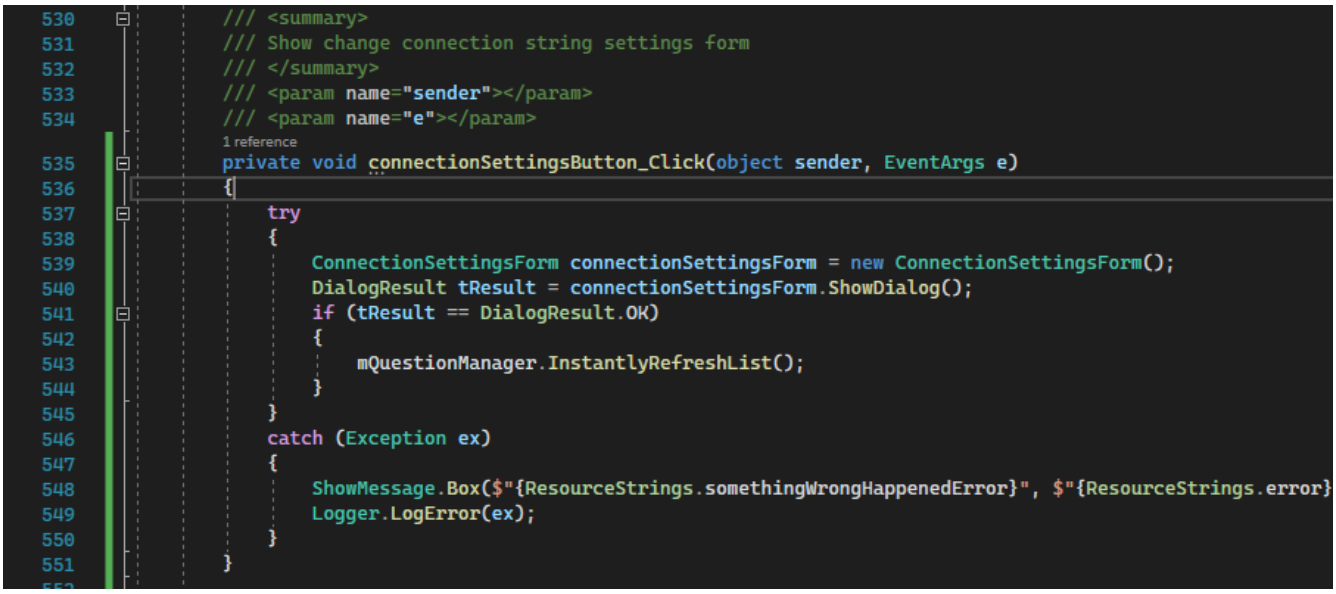
This event handler handle **2 events**:

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

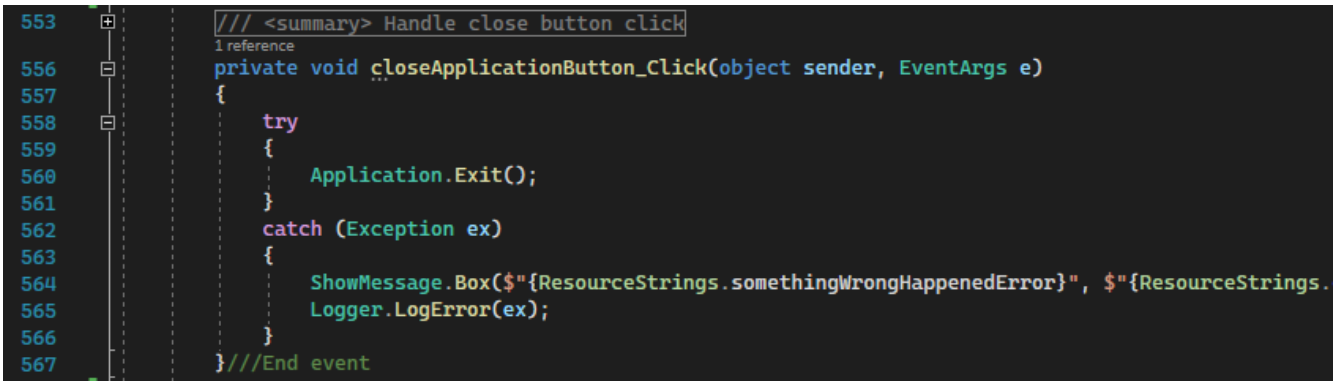


```
530         /// <summary>
531         /// Show change connection string settings form
532         /// </summary>
533         /// <param name="sender"></param>
534         /// <param name="e"></param>
            1 reference
535         private void connectionSettingsButton_Click(object sender, EventArgs 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         }
```

Description:

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

closeApplicationButton_Click

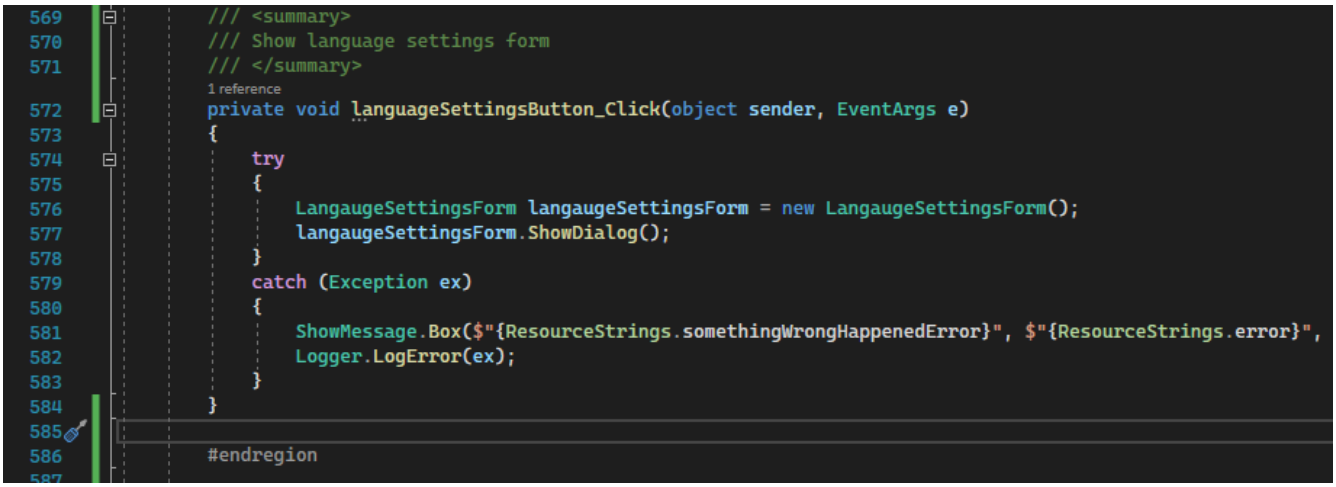


```
553     /// <summary> Handle close button click
        1 reference
556     private void closeApplicationButton_Click(object sender, EventArgs e)
557     {
558         try
559         {
560             Application.Exit();
561         }
562         catch (Exception ex)
563         {
564          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

Description:

This event handler handle closing the application.

languageSettingsButton_Click

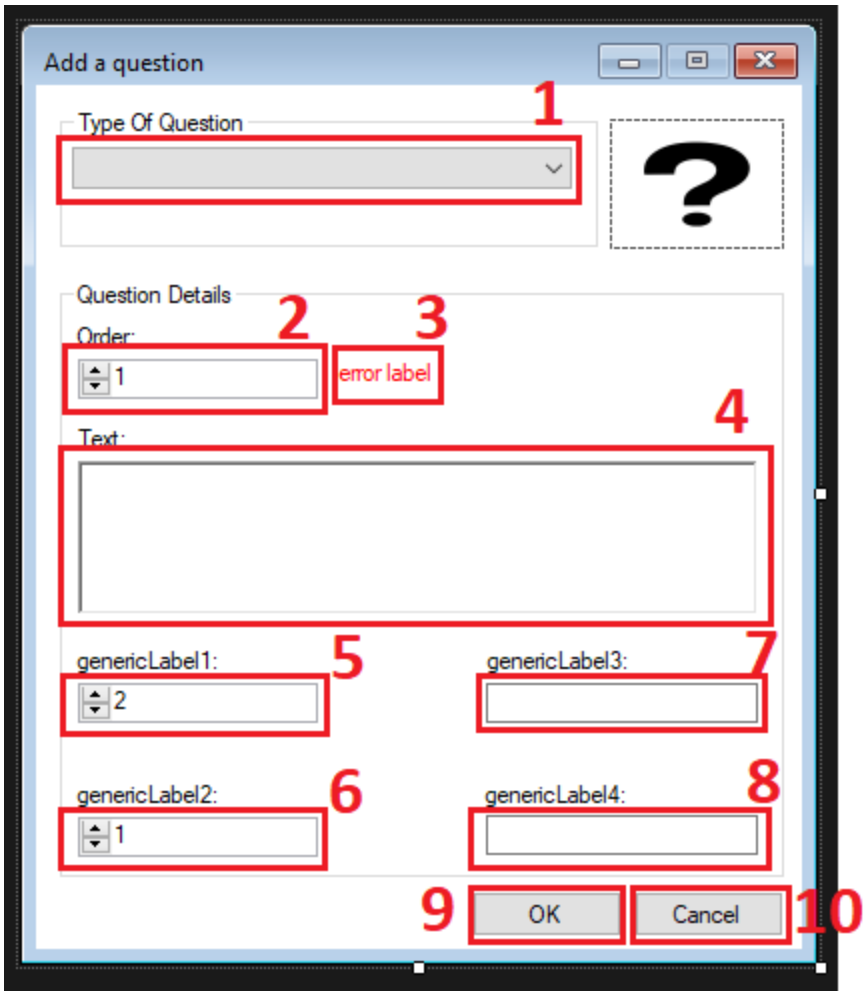


```
569         /// <summary>
570         /// Show language settings form
571         /// </summary>
            1 reference
572         private void languageSettingsButton_Click(object sender, EventArgs e)
573         {
574             try
575             {
576                 LangaugeSettingsForm langaugeSettingsForm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

Description:

This event handler handle showing change language form.

1.4.2 AddQuestionForm



The form has total 10 main controls

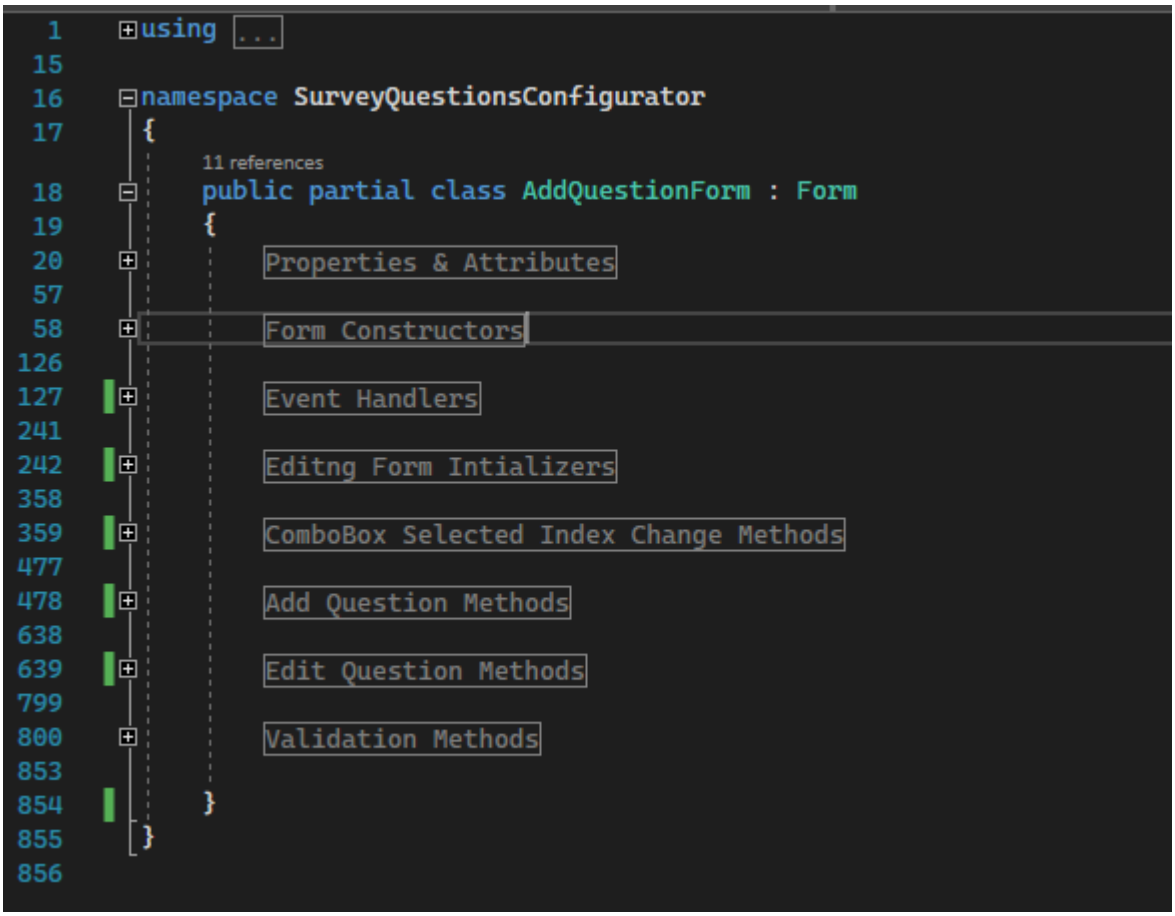
1. Combo box.
2. questionOrderNumericUpDown.
3. Error label.
4. questionTextRichTextBox.
5. genericNumericUpDown1.
6. genericNumericUpDown2.
7. genericTextBox1.
8. genericTextBox2.
9. Ok/save button
10. Cancel button

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.



Properties & Attributes

```
11 references
public partial class AddQuestionForm : Form
{
    #region Properties & Attributes

    private readonly QuestionManager mGeneralQuestionManager;
    private readonly ResourceManager mLocalResourceManager;
    private readonly CultureInfo mDefaultCulture;
    7 references
    private Question mGenericQuestion { get; set; } /// create global Question ID property
    4 references
    private FormStateType cStateForm { get; set; } /// Decide whether "OK" Form button is used to either ADD or EDIT a question
    4 references
    private QuestionType cSelectedQuestionType { get; set; }
    5 references
    public enum FormStateType
    {
        ADD,
        EDIT
    }

    /// <summary> All translatable message box messages in the "AddQuestionFormStrin ...
    90 references
    private enum ResourceStrings
    {
        numberOfSmileyFaces,
        numberOfStars,
        questionOrderAlreadyInUseError,
        sliderEndValue,
        sliderStartValue,
        somethingWrongHappenedError,
        startValueCaption,
        endValueCaption,
        addAQuestion,
        contactSystemAdministratorError,
        editAQuestion,
        error,
        questionWasNotFoundOrDeletedError,
        genericQuestionError,
        sliderQuestionError
    }

    #endregion
```

Line 22-> instance of QuestionManager (Business logic layer).

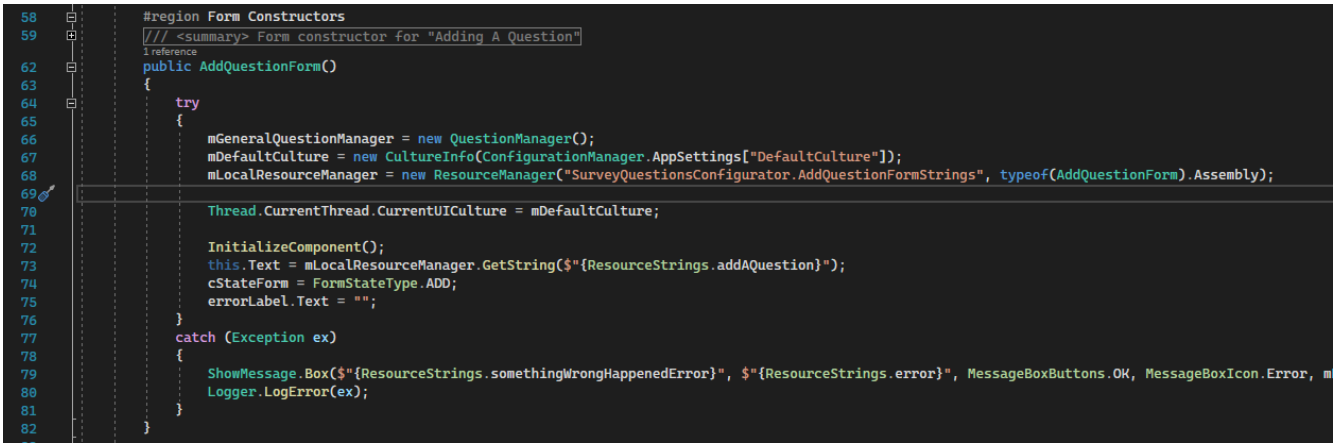
Line 25-> Generic question to be used in multiple methods/event handles.

Line 26-> save the setting of which the form is adding or editing.

Line 27-> save the type of the current question.

Form Constructors

AddQuestionForm



```
58      #region Form Constructors
59      /// <summary> Form constructor for "Adding A Question"
        1 reference
62      public AddQuestionForm()
63      {
64          try
65          {
66              mGeneralQuestionManager = new QuestionManager();
67              mDefaultCulture = new CultureInfo(ConfigurationManager.AppSettings["DefaultCulture"]);
68              mLocalResourceManager = new ResourceManager("SurveyQuestionsConfigurator.AddQuestionFormStrings", typeof(AddQuestionForm).Assembly);
69
70              Thread.CurrentThread.CurrentUICulture = mDefaultCulture;
71
72              InitializeComponent();
73              this.Text = mLocalResourceManager.GetString($"{ResourceStrings.addAQuestion}");
74              cStateForm = FormStateType.ADD;
75              errorLabel.Text = "";
76          }
77          catch (Exception ex)
78          {
79              ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, m
80              Logger.LogError(ex);
81          }
82      }
```

Description:

Initialize the form for adding a question

AddQuestionForm(Question pQuestion):

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

Description:

Initialize the form for editing a selected question

Line 101-> disable changing current edited question type.

Lines 106, 111 & 116 -> Load the settings of the passed question based on its ID and Type

Event Handlers

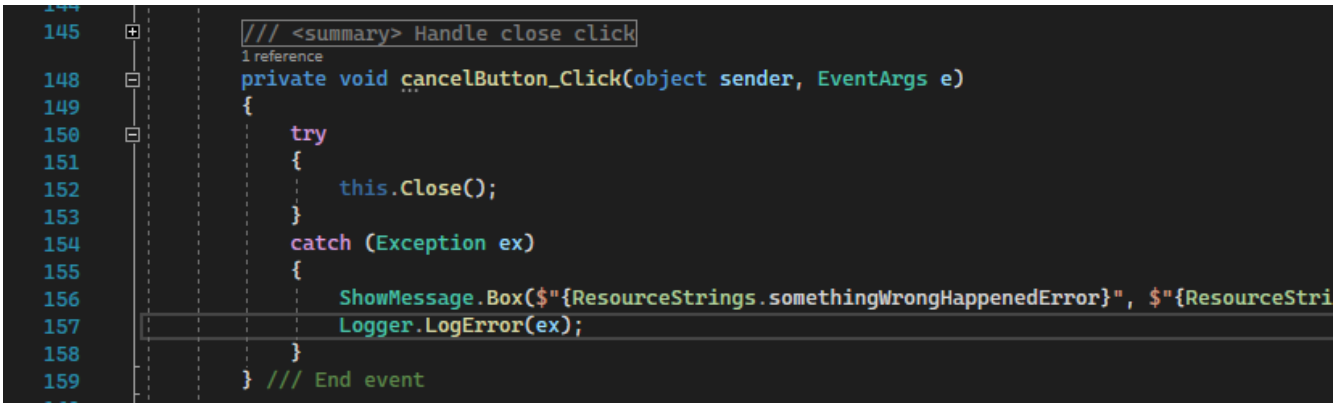
AddQuestionForm_Load

```
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>
            1 reference
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
141                 Logger.LogError(ex);
142             }
143         } /// End event
144
```

Description:

Sets combo box selected index

cancelButton_Click



Description:

Close the application

okButton_Click

```
161         ///<summary>
162         /// Handle saving settings based on what the form is doing (Adding or Editing)
163         /// </summary>
            1 reference
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                     }
211                     catch (Exception ex)
212                     {
213                         Logger.LogError(ex);
214                         throw;
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}", MessageBoxBut
227                 Logger.LogError(ex);
228             }
229         }
230
```

Description:

If cStateForm = Add, it attempts a DB insert operation

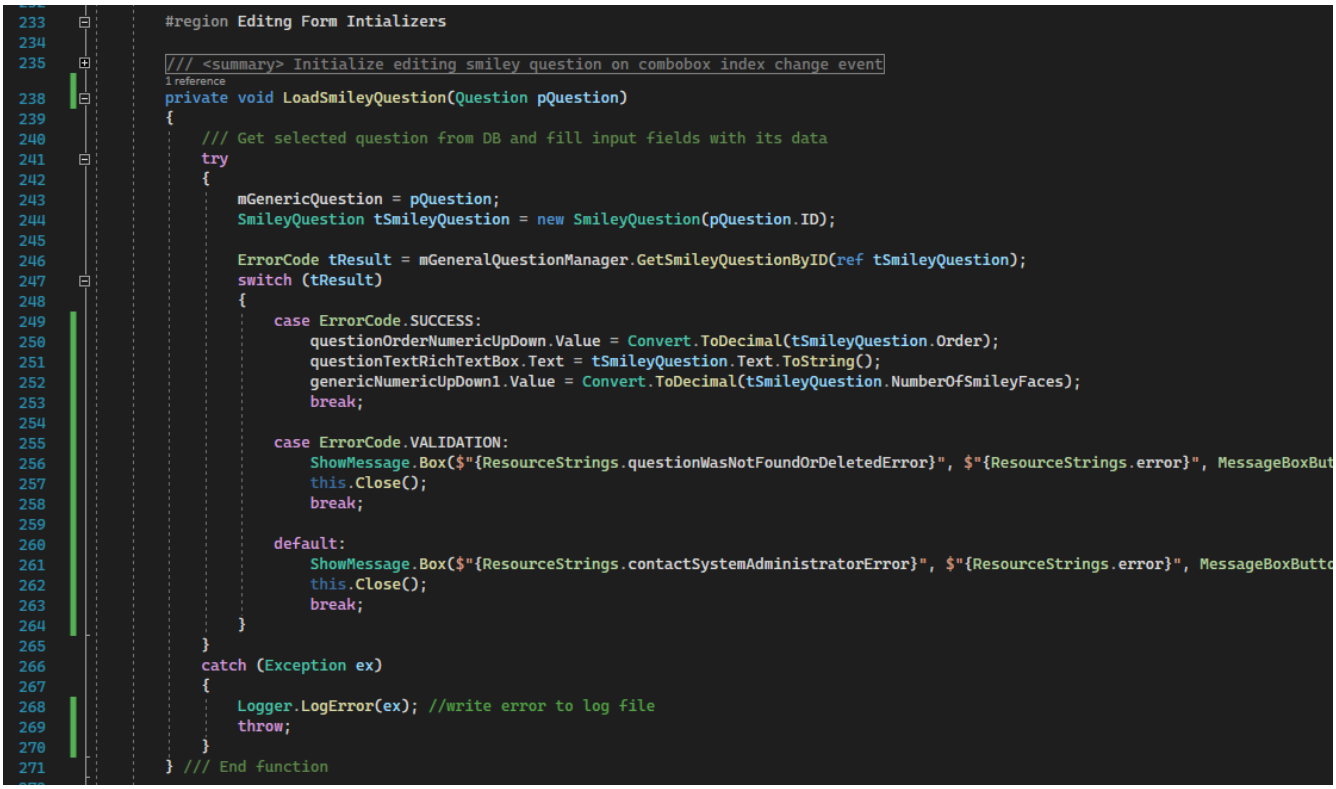
If cStateForm = Edit, it attempts DB update operation

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



```
#region Editng Form Intializers

/// <summary> Initialize editing smiley question on combobox index change event
1 reference
private void LoadSmileyQuestion(Question pQuestion)
{
    /// Get selected question from DB and fill input fields with its data
    try
    {
        mGenericQuestion = pQuestion;
        SmileyQuestion tSmileyQuestion = new SmileyQuestion(pQuestion.ID);

        ErrorCode tResult = mGeneralQuestionManager.GetSmileyQuestionByID(ref tSmileyQuestion);
        switch (tResult)
        {
            case ErrorCode.SUCCESS:
                questionOrderNumericUpDown.Value = Convert.ToDecimal(tSmileyQuestion.Order);
                questionTextRichTextBox.Text = tSmileyQuestion.Text.ToString();
                genericNumericUpDown1.Value = Convert.ToDecimal(tSmileyQuestion.NumberOfSmileyFaces);
                break;

            case ErrorCode.VALIDATION:
                ShowMessage.Box($"{ResourceStrings.questionWasNotFoundOrDeletedError}", $"{ResourceStrings.error}", MessageBoxBut
                this.Close();
                break;

            default:
                ShowMessage.Box($"{ResourceStrings.contactSystemAdministratorError}", $"{ResourceStrings.error}", MessageBoxButto
                this.Close();
                break;
        }
    }
    catch (Exception ex)
    {
        Logger.LogError(ex); //write error to log file
        throw;
    }
} /// End function
```

```
273     /// <summary> Initialize editing slider question on combobox index change event
        1 reference
276     private void LoadSliderQuestion(Question pQuestion)
277     {
278         /// Get selected question from DB and fill input fields  with its data
279         try
280         {
281             mGenericQuestion = pQuestion;
282             SliderQuestion tSliderQuestion = new SliderQuestion(pQuestion.ID);
283
284             ErrorCode result = mGeneralQuestionManager.GetSliderQuestionByID(ref tSliderQuestion);
285             switch (result)
286             {
287                 case ErrorCode.SUCCESS:
288                     questionOrderNumericUpDown.Value = Convert.ToDecimal(tSliderQuestion.Order);
289                     questionTextRichTextBox.Text = tSliderQuestion.Text.ToString();
290                     genericNumericUpDown1.Value = Convert.ToDecimal(tSliderQuestion.StartValue);
291                     genericNumericUpDown2.Value = Convert.ToDecimal(tSliderQuestion.EndValue);
292                     genericTextBox1.Text = tSliderQuestion.StartValueCaption.ToString();
293                     genericTextBox2.Text = tSliderQuestion.EndValueCaption.ToString();
294                     break;
295                 case ErrorCode.VALIDATION:
296                     ShowMessage.Box($"{ResourceStrings.contactSystemAdministratorError}", $"{ResourceStrings.error}", MessageBoxButto
297                     this.Close();
298                     break;
299                 default:
300                     ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.O
301                     this.Close();
302                     break;
303             }
304         }
305         catch (Exception ex)
306         {
307             Logger.LogError(ex); //write error to log file
308             throw;
309         }
310     } /// End function
```

```
312     /// <summary> Initialize editing star question on combobox change
        1 reference
315     private void LoadStarQuestion(Question pQuestion)
316     {
317         /// Select wanted question from DB and fill input fields with its data
318         try
319         {
320             mGenericQuestion = pQuestion;
321             StarQuestion tStarQuestion = new StarQuestion(pQuestion.ID);
322
323             ErrorCode result = mGeneralQuestionManager.GetStarQuestionByID(ref tStarQuestion);
324             switch (result)
325             {
326                 case ErrorCode.SUCCESS:
327                     questionOrderNumericUpDown.Value = Convert.ToDecimal(tStarQuestion.Order);
328                     questionTextRichTextBox.Text = tStarQuestion.Text.ToString();
329                     genericNumericUpDown1.Value = Convert.ToDecimal(tStarQuestion.NumberOfStars);
330                     break;
331                 case ErrorCode.VALIDATION:
332                     ShowMessage.Box($"{ResourceStrings.contactSystemAdministratorError}", $"{ResourceStrings.error}", MessageBoxBu
333                     this.Close();
334                     break;
335                 default:
336                     ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButton
337                     this.Close();
338                     break;
339             }
340         }
341         catch (Exception ex)
342         {
343             Logger.LogError(ex); //write error to log file
344             throw;
345         }
346     }/// End function
```

Desctiption:

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

ComboBox Selected Index Change Methods

questionTypeComboBox_SelectedIndexChanged

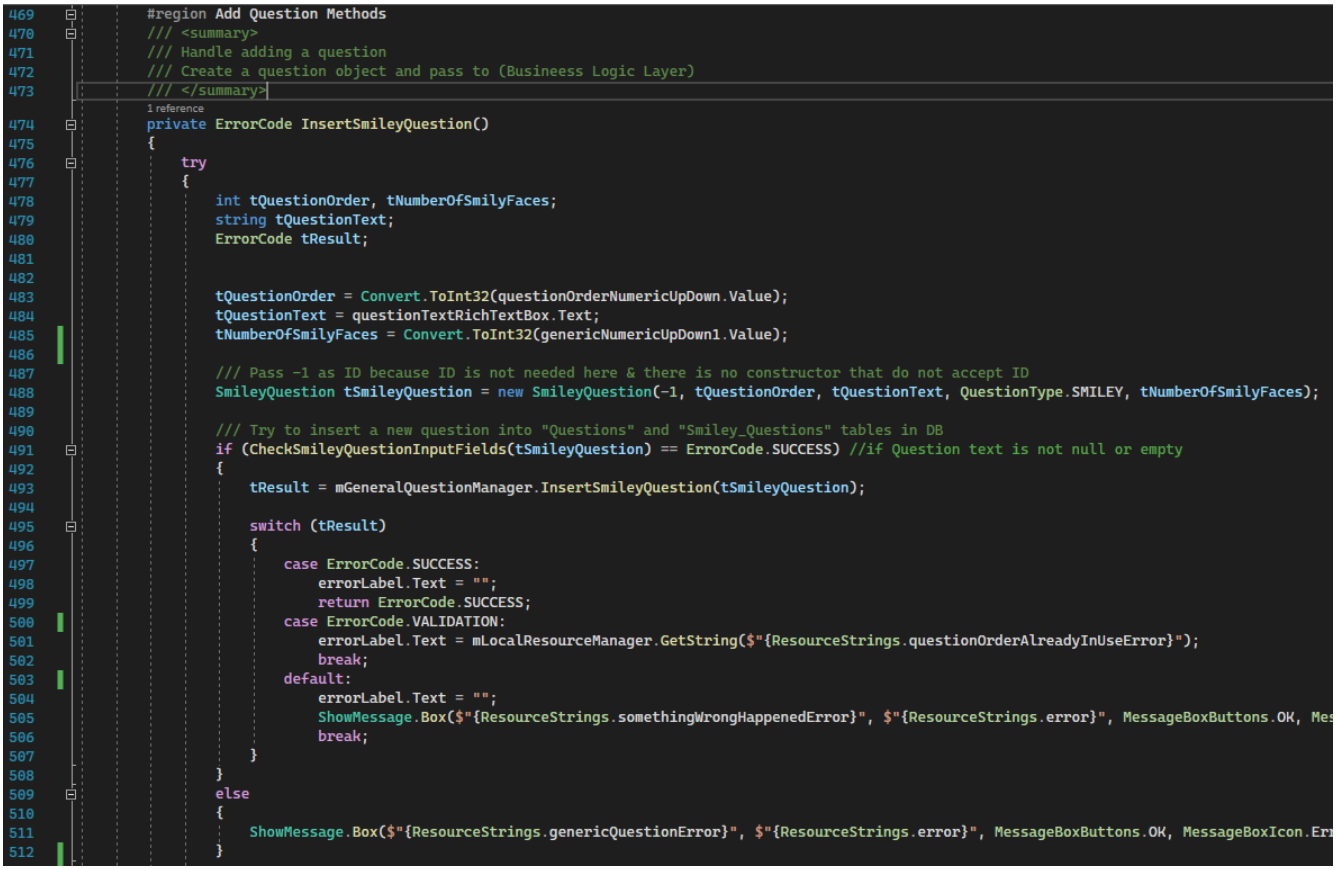
```
#region ComboBox Selected Index Change Methods

/// <summary> Handle combo box selected index changed
1 reference
private void questionTypeComboBox_SelectedIndexChanged(object sender, EventArgs e)
{
    try
    {
        if (questionTypeComboBox.SelectedIndex == 0)
        {
            InitializeEditingSmileyQuestion();
        }
        else if (questionTypeComboBox.SelectedIndex == 1)
        {
            InitializeEditingSliderQuestion();
        }
        else if (questionTypeComboBox.SelectedIndex == 2)
        {
            InitializeEditingStarQuestion();
        }
    }
    catch (Exception ex)
    {
        ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBox
        Logger.LogError(ex);
    }
}/// 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



```
#region Add Question Methods
/// <summary>
/// Handle adding a question
/// Create a question object and pass to (Busineess Logic Layer)
/// </summary>
1 reference
private ErrorCode InsertSmileyQuestion()
{
    try
    {
        int tQuestionOrder, tNumberOfSmilyFaces;
        string tQuestionText;
        ErrorCode tResult;


        tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
        tQuestionText = questionTextRichTextBox.Text;
        tNumberOfSmilyFaces = Convert.ToInt32(genericNumericUpDown1.Value);

        /// Pass -1 as ID because ID is not needed here & there is no constructor that do not accept ID
        SmileyQuestion tSmileyQuestion = new SmileyQuestion(-1, tQuestionOrder, tQuestionText, QuestionType.SMILEY, tNumberOfSmilyFaces);

        /// Try to insert a new question into "Questions" and "Smiley_Questions" tables in DB
        if (CheckSmileyQuestionInputFields(tSmileyQuestion) == ErrorCode.SUCCESS) //if Question text is not null or empty
        {
            tResult = mGeneralQuestionManager.InsertSmileyQuestion(tSmileyQuestion);

            switch (tResult)
            {
                case ErrorCode.SUCCESS:
                    errorLabel.Text = "";
                    return ErrorCode.SUCCESS;
                case ErrorCode.VALIDATION:
                    errorLabel.Text = mLocalResourceManager.GetString($"{ResourceStrings.questionOrderAlreadyInUseError}");
                    break;
                default:
                    errorLabel.Text = "";
                    ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, Mes
                    break;
            }
        }
        else
        {
            ShowMessage.Box($"{ResourceStrings.genericQuestionError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Err
        }
```

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

```
578     /// <summary> Handle adding a question Create a question object and pass to (Bus ...
        1 reference
582     private ErrorCode InsertStarQuestion()
583     {
584         try
585         {
586             int tQuestionOrder, tNumberOfStars;
587             string tQuestionText;
588             ErrorCode tResult;
589
590             tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
591             tQuestionText = questionTextRichTextBox.Text;
592             tNumberOfStars = Convert.ToInt32(genericNumericUpDown1.Value);
593
594
595             /// Pass -1 as ID because ID is not needed here & there is no constructor that do not accept ID
596             StarQuestion tStarQuestion = new StarQuestion(-1, tQuestionOrder, tQuestionText, QuestionType.STAR, tNumberOfStars);
597
598             /// Try to insert a new question into "Questions" and "Star_Questions" tables in DB
599             if (CheckStarQuestionInputFields(tStarQuestion) == ErrorCode.SUCCESS) /// If question input fields are not null or empty
600             {
601                 tResult = mGeneralQuestionManager.InsertStarQuestion(tStarQuestion);
602
603                 switch (tResult)
604                 {
605                     case ErrorCode.SUCCESS:
606                         errorLabel.Text = "";
607                         return ErrorCode.SUCCESS;
608                     case ErrorCode.VALIDATION:
609                         errorLabel.Text = mLocalResourceManager.GetString($"{ResourceStrings.questionOrderAlreadyInUseError}");
610                         break;
611                     default:
612                         errorLabel.Text = "";
613                         ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, Messa
614                         break;
615                 }
616             }
617             else
618             {
619                 ShowMessage.Box($"{ResourceStrings.genericQuestionError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error
620             }
621             return ErrorCode.ERROR;
```

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.

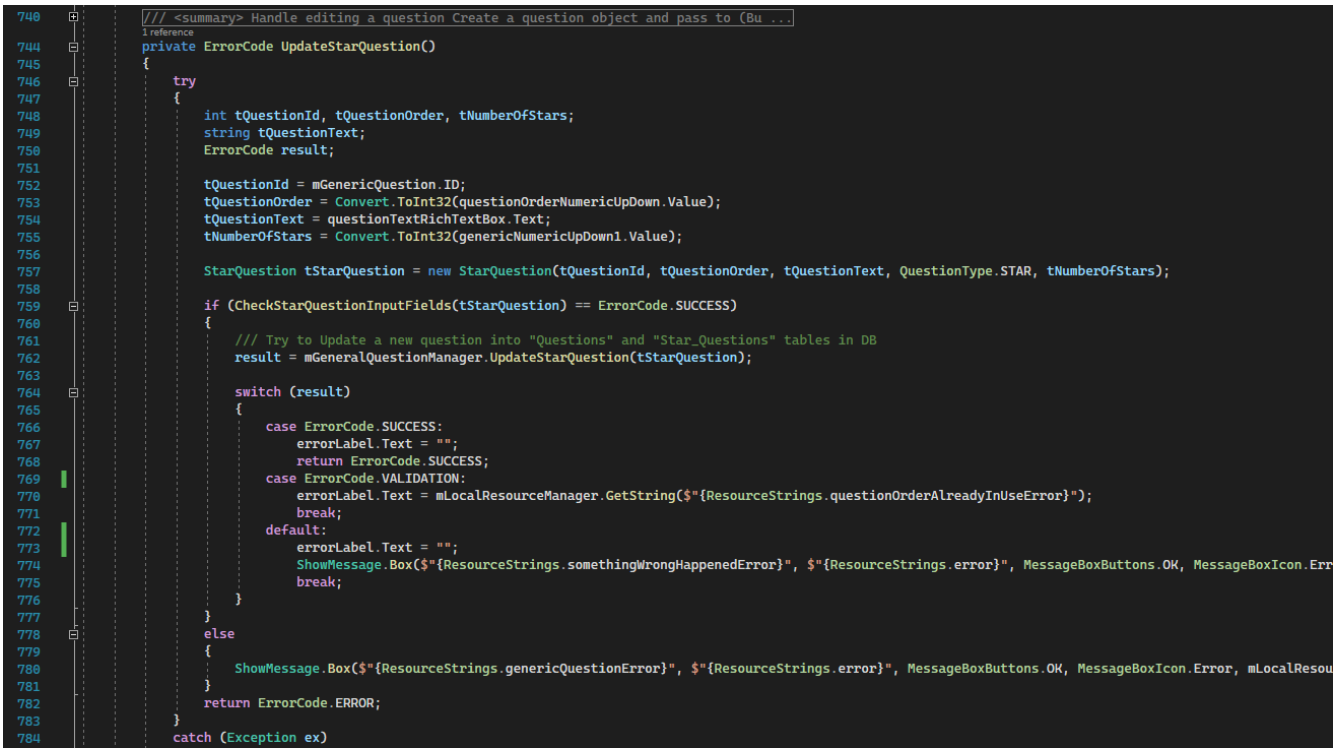
Returns:
ErrorCode based on the operation.

Edit Question Methods

UpdateSmileyQuestion, UpdateSliderQuestion & UpdateStarQuestion:

```
#region Edit Question Methods
/// <summary>
/// Handle editing a question
/// Create a question object and pass to (Busineess Logic Layer)
/// </summary>
1 reference
private ErrorCode UpdateSmileyQuestion()
{
    try
    {
        int tQuestionId, tQuestionOrder, tNumberOfSmilyFaces;
        string tQuestionText;
        ErrorCode tResult;

        tQuestionId = mGenericQuestion.ID;
        tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
        tQuestionText = questionTextRichTextBox.Text;
        tNumberOfSmilyFaces = Convert.ToInt32(genericNumericUpDown1.Value);

        SmileyQuestion tSmileyQuestion = new SmileyQuestion(tQuestionId, tQuestionOrder, tQuestionText, QuestionType.SMILEY, tNumberOfSmilyFaces);

        if (CheckSmileyQuestionInputFields(tSmileyQuestion) == ErrorCode.SUCCESS)
        {
            /// Try to Update a new question into "Questions" and "Smiley_Questions" tables in DB
            tResult = mGeneralQuestionManager.UpdateSmileyQuestion(tSmileyQuestion);

            switch (tResult)
            {
                case ErrorCode.SUCCESS:
                    errorLabel.Text = "";
                    return ErrorCode.SUCCESS;
                case ErrorCode.VALIDATION:
                    errorLabel.Text = mLocalResourceManager.GetString($"{ResourceStrings.questionOrderAlreadyInUseError}");
                    break;
                default:
                    errorLabel.Text = "";
                    ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon
                    break;
            }
        }
        else
        {
            ShowMessage.Box($"{ResourceStrings.genericQuestionError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalR
        }
        return ErrorCode.ERROR;
```

```
/// <summary>
/// Handle editing a question
/// Create a question object and pass to (Busineess Logic Layer)
/// </summary>
1 reference
private ErrorCode UpdateSliderQuestion()
{
    try
    {
        int tQuestionId, tQuestionOrder, tQuestionStartValue, tQuestionEndValue;
        string tQuestionText, tQuestionStartValueCaption, tQuestionEndValueCaption;
        ErrorCode tResult;

        tQuestionId = mGenericQuestion.ID;
        tQuestionOrder = Convert.ToInt32(questionOrderNumericUpDown.Value);
        tQuestionText = (string)questionTextRichTextBox.Text;
        tQuestionStartValue = Convert.ToInt32(genericNumericUpDown1.Value);
        tQuestionEndValue = Convert.ToInt32(genericNumericUpDown2.Value);
        tQuestionStartValueCaption = (string)genericTextBox1.Text;
        tQuestionEndValueCaption = (string)genericTextBox2.Text;

        SliderQuestion tSliderQuestion = new SliderQuestion(tQuestionId, tQuestionOrder, tQuestionText, QuestionType.SLIDER, tQuestionStartValue, tQue

        if (CheckSliderQuestionInputFields(tSliderQuestion) == ErrorCode.SUCCESS)
        {
            /// Try to Update a new question into "Questions" and "Slider_Questions" tables in DB
            tResult = mGeneralQuestionManager.UpdateSliderQuestion(tSliderQuestion);

            switch (tResult)
            {
                case ErrorCode.SUCCESS:
                    errorLabel.Text = "";
                    return ErrorCode.SUCCESS;
                case ErrorCode.VALIDATION:
                    errorLabel.Text = mLocalResourceManager.GetString($"{ResourceStrings.questionOrderAlreadyInUseError}");
                    break;
                default:
                    errorLabel.Text = "";
                    ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon
                    break;
            }
        }
        else
        {
            ShowMessage.Box($"{ResourceStrings.sliderQuestionError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalRe
```



```
740     /// <summary> Handle editing a question Create a question object and pass to (Bu ...
        1 reference
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}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Err
775                         break;
776                 }
777             }
778             else
779             {
780                 ShowMessage.Box($"{ResourceStrings.genericQuestionError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResou
781             }
782             return ErrorCode.ERROR;
783         }
784         catch (Exception ex)
```

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
        2 references
798     private ErrorCode CheckSmileyQuestionInputFields(SmileyQuestion pSmileyQuestion)
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
```

```
812     /// <summary> Check slider question input fields
        2 references
815     private ErrorCode CheckSliderQuestionInputFields(SliderQuestion pSliderQuestion)
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
```

```
829     /// <summary> Check star question input fields
        2 references
832     private ErrorCode CheckStarQuestionInputFields(StarQuestion pStarQuestion)
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
```

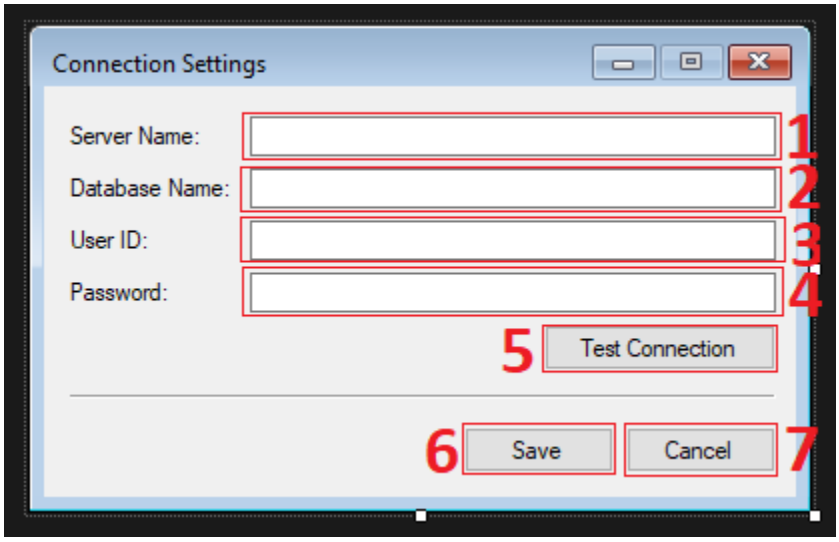
Description:

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

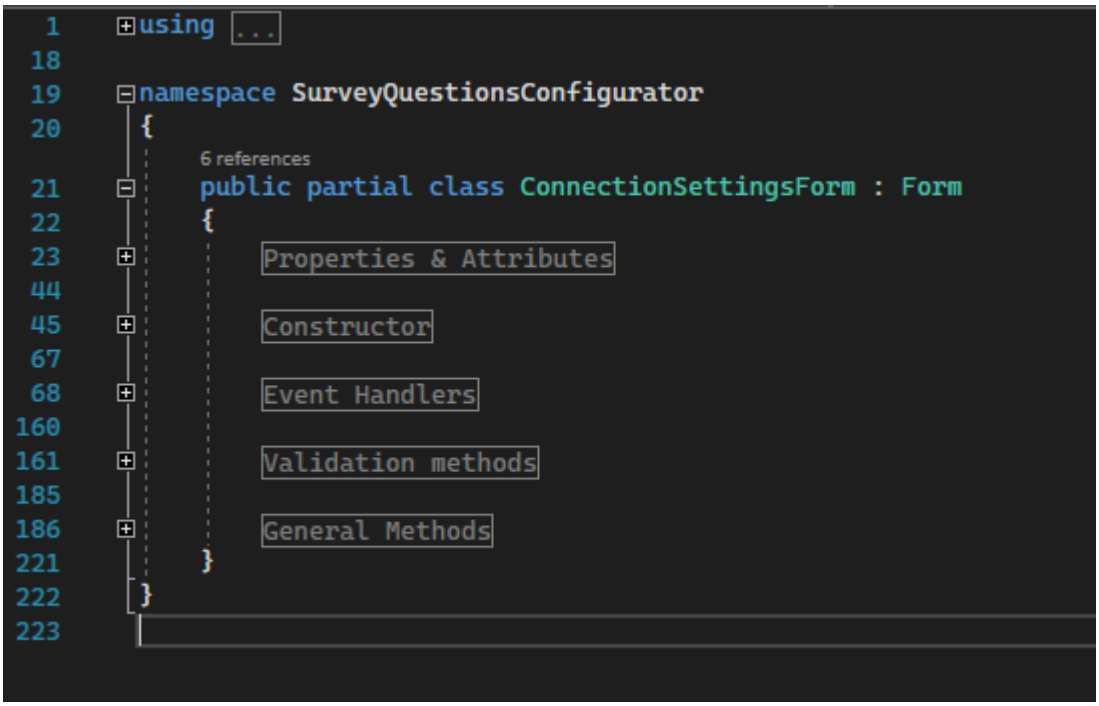
1. Server name text box.
2. Database name text box.
3. User ID text box.
4. Password text box.
5. Test connection button.
6. Save settings button.
7. Cancel button.

Control's Properties:

All text boxes have MaxLength = 128

Code:

The code is divided into regions based on their functionality.



Properties & Attributes

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

Constructor

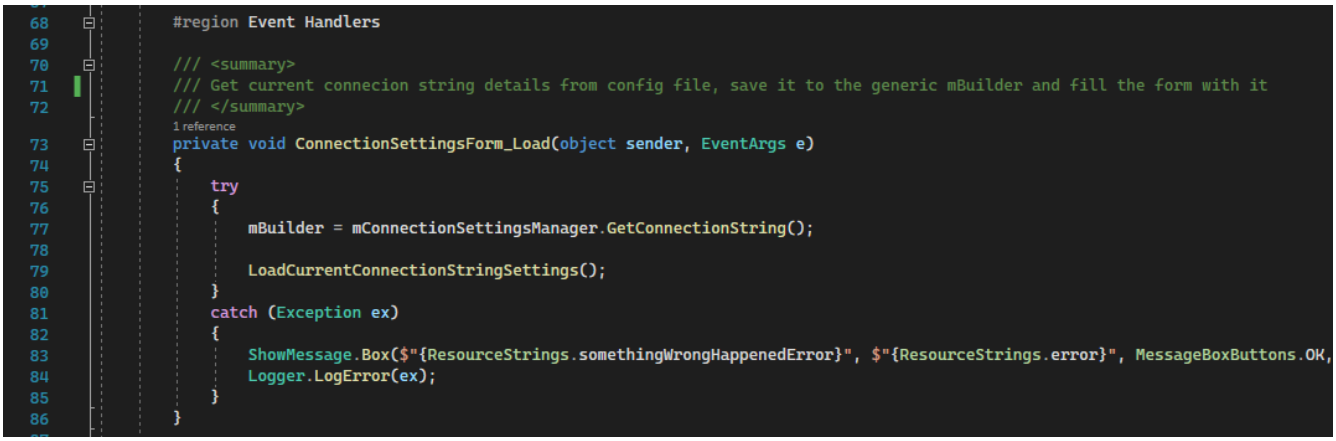
ConnectionSettingsForm

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

Lines 53 & 55-> Gets the default UI culture from config file and sets it to the current thread's CurrentUICulture property

Event Handlers

ConnectionSettingsForm_Load:



```
68      #region Event Handlers
69
70      /// <summary>
71      /// Get current connecion string details from config file, save it to the generic mBuilder and fill the form with it
72      /// </summary>
        1 reference
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      }
```

Description:

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

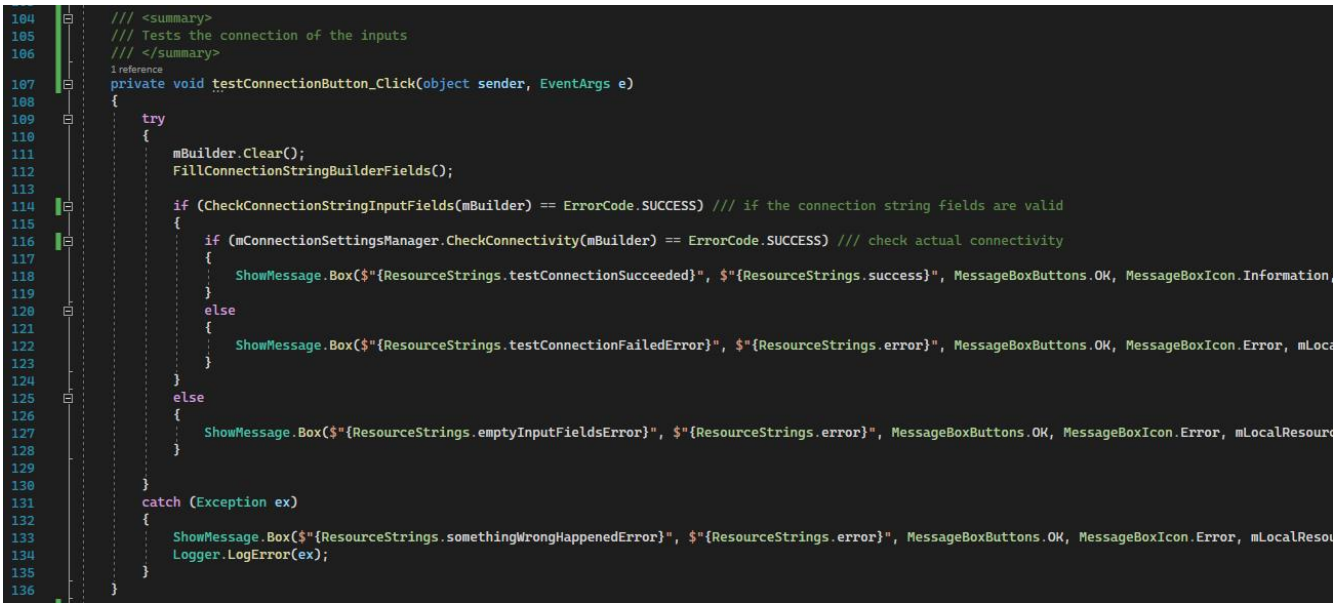
cancelButton_Click:

```
/// <summary>
/// Close the form
/// </summary>
1 reference
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>
        1 reference
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 (mConnectionSettingsManager.CheckConnectivity(mBuilder) == ErrorCode.SUCCESS) /// check actual connectivity
117                 {
118                     ShowMessage.Box($"{ResourceStrings.testConnectionSucceeded}", $"{ResourceStrings.success}", MessageBoxButtons.OK, MessageBoxIcon.Information,
119                 }
120                 else
121                 {
122                     ShowMessage.Box($"{ResourceStrings.testConnectionFailedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, mLoca
123                 }
124             }
125             else
126             {
127                 ShowMessage.Box($"{ResourceStrings.emptyInputFieldsError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalResourc
128             }
129
130         }
131         catch (Exception ex)
132         {
133             ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, mLocalReso
134             Logger.LogError(ex);
135         }
136     }
```

Description:

Fill the mBuilder with corresponding inputs and text them

saveButton_Click

```
138     /// <summary>
139     /// Saves the current inputs to the config file under "connectionStrings" section
140     /// </summary>
        1 reference
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, Me
161             Logger.LogError(ex);
162         }
163     }
```

Description:

Fill the mBuilder save its fields to the config file.

Validation methods

CheckConnectionStringInputFields:

```
#region Validation methods

/// <summary> Check connection string fields
1 reference
private ErrorCode CheckConnectionStringInputFields(SqlConnectionStringBuilder pBuilder)
{
    try
    {
        return mConnectionSettingsManager.CheckConnectionStringInputFields(pBuilder);
    }
    catch (Exception ex)
    {
        ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK,
        Logger.LogError(ex);
        return ErrorCode.ERROR;
    }
}

#endregion
```

Description:

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

Returns:

ErrorCode

General Methods

FillConnectionStringBuilderFields:

```
192     #region General Methods
193
            2 references
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.

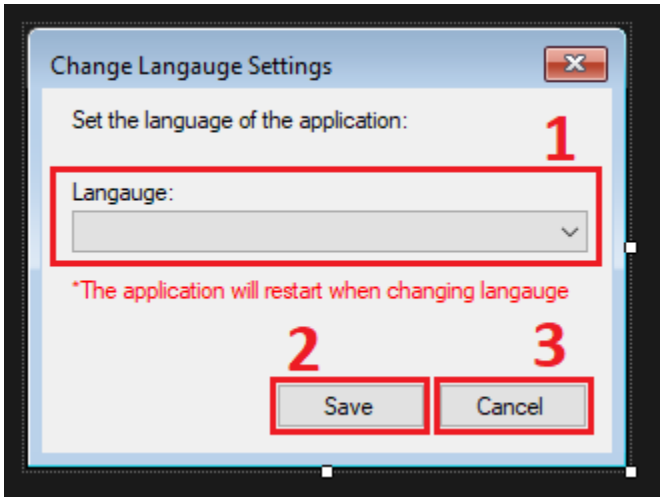
FillConnectionStringBuilderFields:

```
1 reference
public void LoadCurrentConnectionStringSettings()
{
    try
    {
        dataSourceTextBox.Text = mBuilder.DataSource;
        initialCatalogTextBox.Text = mBuilder.InitialCatalog;
        userIDTextBox.Text = mBuilder.UserID;
        passwordTextBox.Text = mBuilder.Password;
    }
    catch (Exception ex)
    {
        Logger.LogError(ex);
        throw;
    }
}
```

Description:

Fill input text boxes with corresponding data from mBuilder.

1.4.4 LangaugeSettingsForm



The form has total 3 main controls

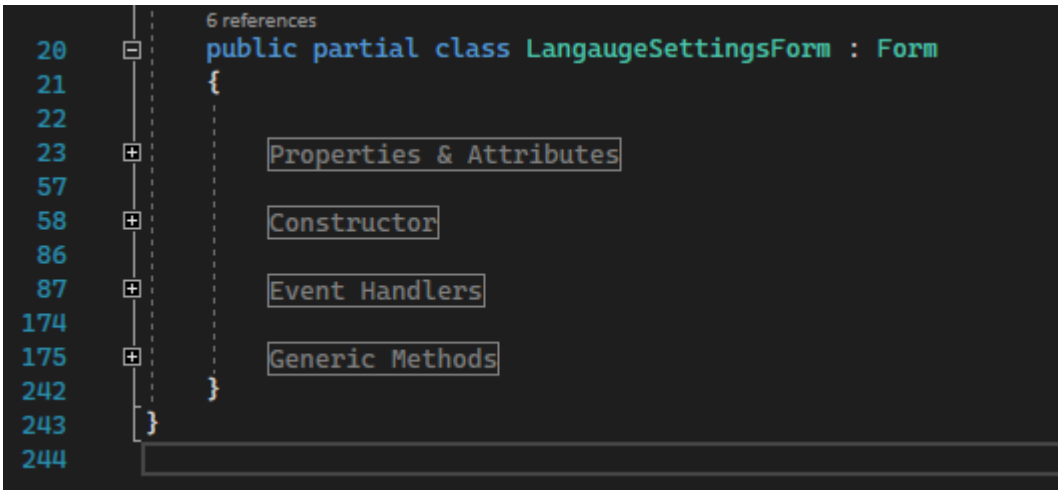
1. Language combo box.
2. Save button.
3. Cancel button.

Control's Properties:

- languageComboBox
    - DropDownStyle-> DropDownList

Code:

The code is divided into regions based on their functionality.



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 langauge 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>
        16 references
40      private enum ResourceStrings
41      {
42          somethingWrongHappenedError,
43          error,
44          success,
45          errorSavingSettings
46      }
47
48      /// <summary> Langauges to be set to or get from combo box
        11 references
51      private enum ComboBoxLanguages
52      {
53          English = 0,
54          Arabic = 1,
55          NoLanguage = -1
56      }
```

Line 28 -> saves loaded language settings.

Line 32-> access config file.

Line 33-> xml attribute to be saved.

Constructor

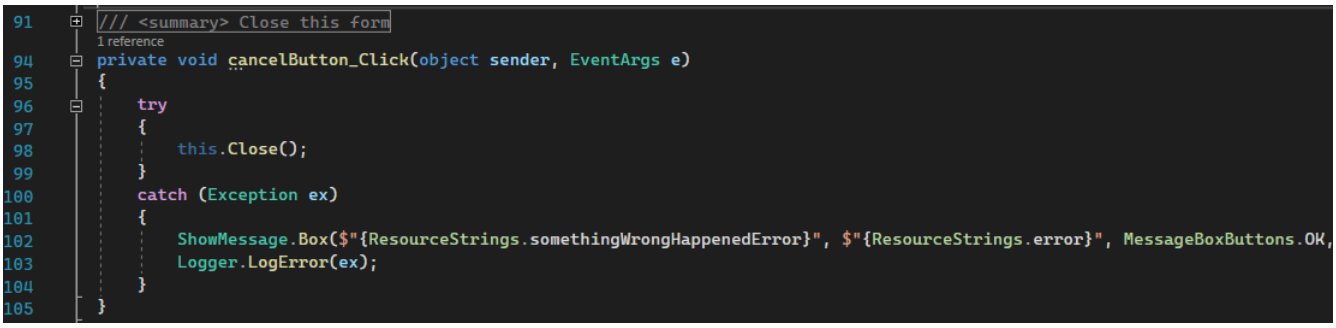
LangaugeSettingsForm:

```
#region Constructor
1 reference
public LangaugeSettingsForm()
{
    try
    {
        mLocalResourceManager = new ResourceManager("SurveyQuestionsConfigurator.LanguageSettingsFormStrings", typeof(LangaugeSettingsForm).Assembly);
        mDefaultCulture = new CultureInfo(ConfigurationManager.AppSettings["DefaultCulture"]);

        mLoadedComboBoxLanguage = ComboBoxLanguages.NoLanguage;
        mConfigEnglishLangauge = "en-US";
        mConfigArabicLangauge = "ar-JO";

        tConfigFile = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
        tSettings = tConfigFile.AppSettings.Settings;
        mDefaultCultureString = "DefaultCulture";

        Thread.CurrentThread.CurrentUICulture = mDefaultCulture;

        InitializeComponent();
    }
    catch (Exception ex)
    {
        ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK, MessageBoxIcon.Error, mLoca
        Logger.LogError(ex);
    }
}

#endregion
```

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

Event Handlers

cancelButton_Click



```
91      /// <summary> Close this form
        1 reference
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
        1 reference
110     private void LangaugeSettingsForm_Load(object sender, EventArgs e)
111     {
112         try
113         {
114             LoadLangaugeSettings();
115         }
116         catch (Exception ex)
117         {
118             ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}",
119             Logger.LogError(ex);
120         }
121     }
```

Description:

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

saveButton_Click:

```
123     /// <summary>
124     /// Handle saving new langauge settings
125     /// Restart application if new langauge is saved other than current application's language
126     /// </summary>
        1 reference
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 langauge
141                 if (tCurrentSelectedIndex == ComboBoxLanguages.English)
142                 {
143                     ErrorCode tResult = SaveLangaugeSettings(mConfigEnglishLangauge);
144                     if (tResult == ErrorCode.SUCCESS)
145                     {
146                         Application.Restart();
147                     }
148                     else
149                     {
150                         ShowMessage.Box($"{ResourceStrings.errorSavingSettings}", $"{ResourceStrings.error}", MessageBoxButtons.OK,
151                     }
152                 }
153                 /// Saving arabic langauge
154                 else if (tCurrentSelectedIndex == ComboBoxLanguages.Arabic)
155                 {
156                     ErrorCode tResult = SaveLangaugeSettings(mConfigArabicLangauge);
157                     if (tResult == ErrorCode.SUCCESS)
158                     {
159                         Application.Restart();
160                     }
161                     else
162                     {
163                         ShowMessage.Box($"{ResourceStrings.errorSavingSettings}", $"{ResourceStrings.error}", MessageBoxButtons.OK,
164                     }
```

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.

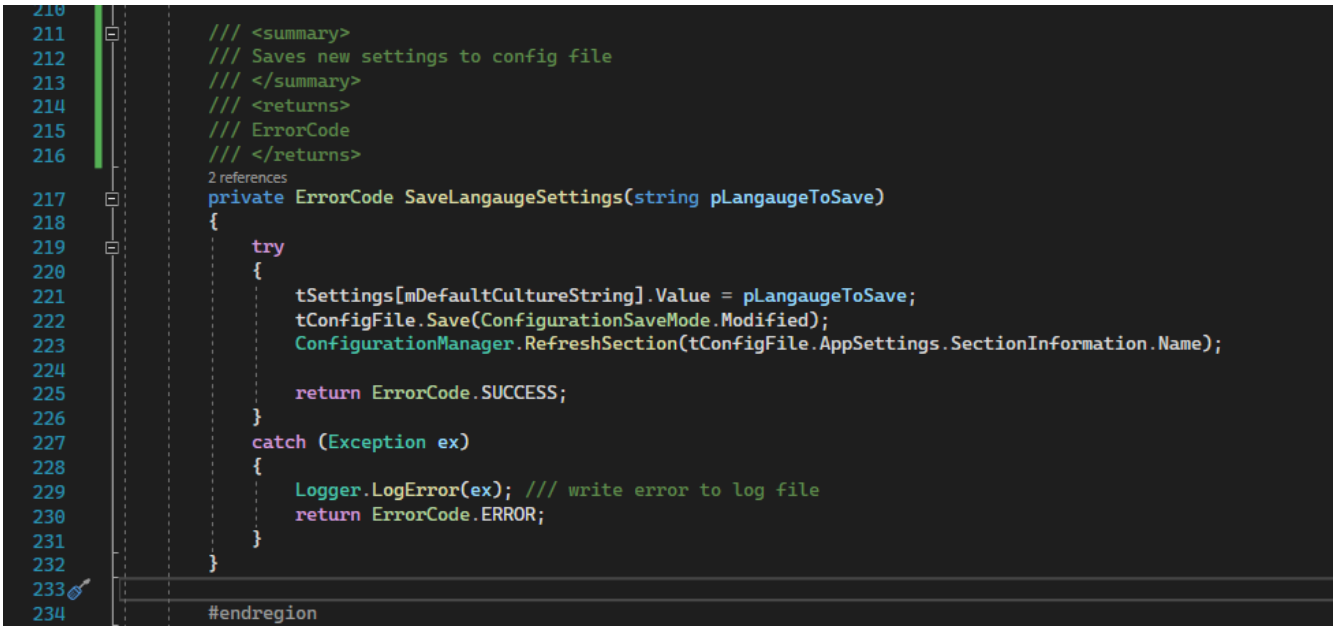
LoadLangaugeSettings

```
179         /// <summary>
180         /// Load combo box with the current language of the application
181         /// </summary>
            1 reference
182         private void LoadLangaugeSettings()
183         {
184             try
185             {
186                 string tReturnedLangaugeValue = tSettings[mDefaultCultureString].Value;
187
188
189                 if (tReturnedLangaugeValue == mConfigEnglishLangauge)
190                 {
191                     languageComboBox.SelectedIndex = (int)ComboBoxLanguages.English;
192                     mLoadedComboBoxLanguage = (ComboBoxLanguages)languageComboBox.SelectedIndex;
193                 }
194                 else if (tReturnedLangaugeValue == mConfigArabicLangauge)
195                 {
196                     languageComboBox.SelectedIndex = (int)ComboBoxLanguages.Arabic;
197                     mLoadedComboBoxLanguage = (ComboBoxLanguages)languageComboBox.SelectedIndex;
198                 }
199                 else
200                 {
201                     languageComboBox.SelectedIndex = (int)ComboBoxLanguages.NoLanguage;
202                 }
203             }
204             catch (Exception ex)
205             {
206                 ShowMessage.Box($"{ResourceStrings.somethingWrongHappenedError}", $"{ResourceStrings.error}", MessageBoxButtons.OK,
207                 Logger.LogError(ex);
208             }
209         }
```

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>
            2 references
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
234         #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:

```
68 references
public class ShowMessage
{
    ///<summary>
    /// Custom methods to show message box dialog
    /// </summary>
    /// <returns>
    /// DialogResult
    /// </returns>
    68 references
    public static DialogResult Box(string pText, string pCaption, MessageBoxButtons pButton, MessageBoxIcon pIcon,
        ResourceManager pLocalResourceManager = null, CultureInfo pDefaultCulture = null)
    {
        try
        {
            if (pDefaultCulture.ToString() == "ar-JO")
            {
                return MessageBox.Show(pLocalResourceManager.GetString(pText), pLocalResourceManager.GetString(pCaption),
                    pButton, pIcon, MessageBoxDefaultButton.Button1, MessageBoxOptions.RightAlign); /// Handle RTL Message Box
            }
            return MessageBox.Show(pLocalResourceManager.GetString(pText), pLocalResourceManager.GetString(pCaption),
                pButton, pIcon); /// Handle LTR Mssage box
            }
            catch (Exception ex)
            {
                Logger.LogError(ex);
                throw;
            }
        }
    }
}
```

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.

Line 28-> shows a messages box that handles RTL languages

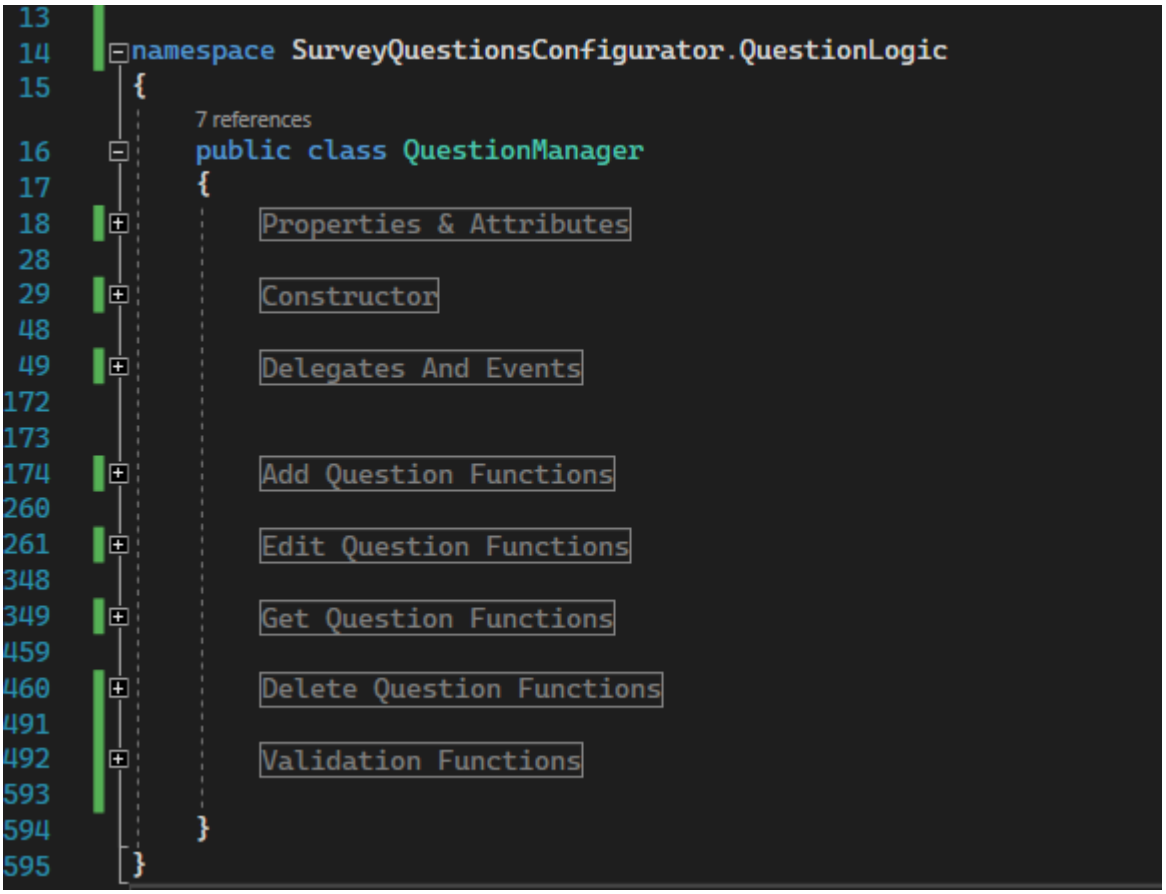
Line 31-> shows a normal LTR message box

Returns:

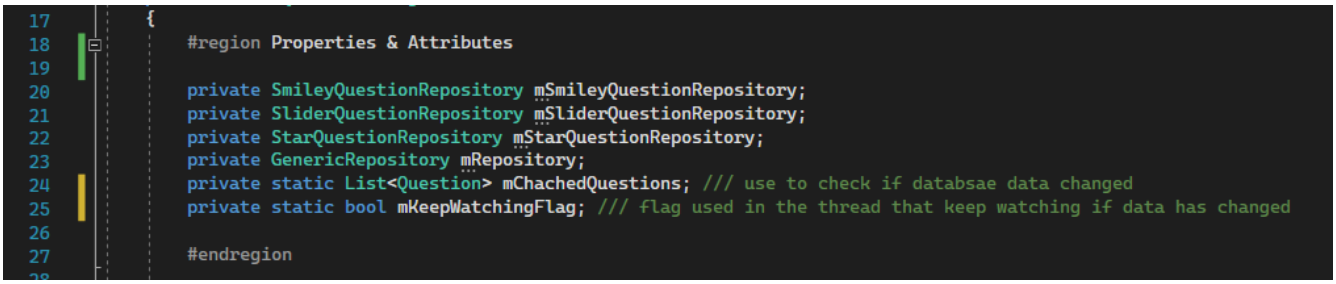
DialgoResult

1.5 Business Logic Layer

1.5.1 QuestionManager



Properties & Attributes



Lines 20-23: instances of every question repository.

Line 24: use to check if databsae data changed

Line 25: flag used in the thread that keep watching if data has changed

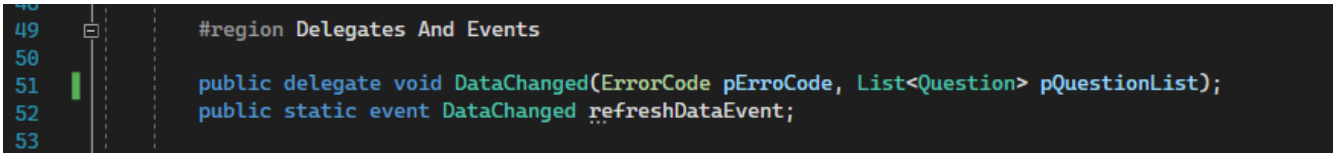
Constructor

QuestionManager

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

Line 39: set flag to true-> keep checking for changes.

Delegates And Events



Line 51: create delegate that holds reference of functions that wants to subscribe for it.

Line 52: create event for that delegate.

WatchForChanges:

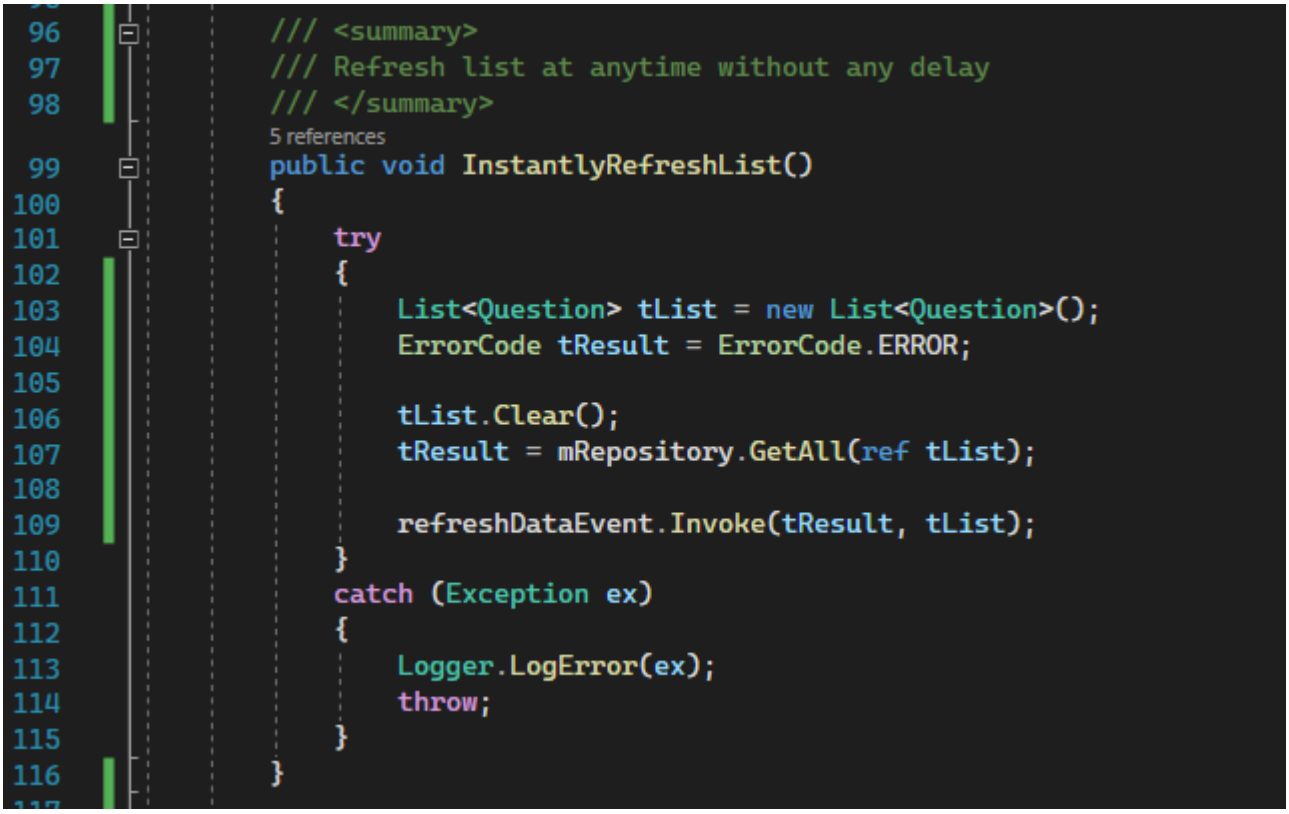
```
54          /// <summary>
55          /// Subscribe to a thread that continuously watch for DB changes every 30 seconds
56          /// </summary>
            1 reference
57          public void WatchForChanges()
58          {
59              try
60              {
61                  int AutoRefreshTimer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

Description:

This function invokes event without any condition.

StopWatching

```
118         /// <summary>
119         /// Breaks the infinite loop of "tAutoRefreshThread"
120         /// </summary>
            0 references
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

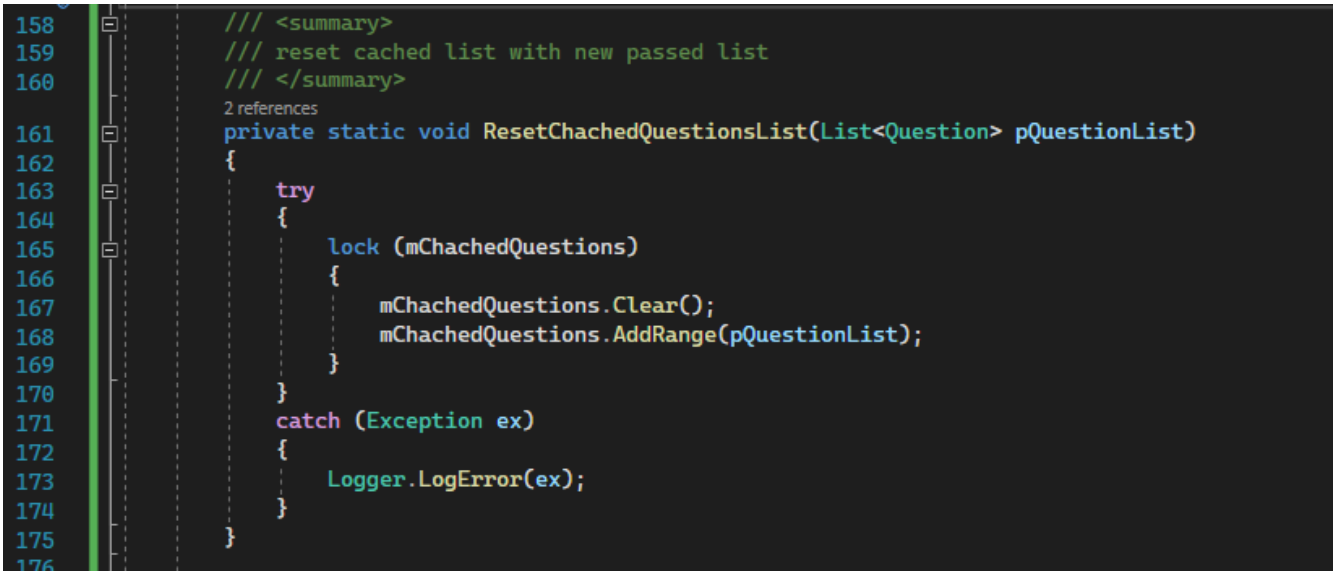
GetAutoRefreshTimerFromConfigFile

```
134     /// <summary> Get auto refresh timer from config file
        1 reference
140     private int GetAutoRefreshTimerFromConfigFile()
141     {
142         try
143         {
144             string tSectionName = "AutoRefreshTimer";
145             var tConfigFile = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
146             var tSettings = tConfigFile.AppSettings.Settings;
147             string tValue = tSettings[tSectionName].Value;
148             return Int32.Parse(tValue);
149         }
150         catch (Exception ex)
151         {
152             Logger.LogError(ex);
153             int tDefaultTimer = 30000;
154             return tDefaultTimer;
155         }
156     }
157
```

Description:

Gets and return auto refresh timer from config file

ResetChachedQuestionsList



```
158         /// <summary>
159         /// reset cached list with new passed list
160         /// </summary>
            2 references
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         }
```

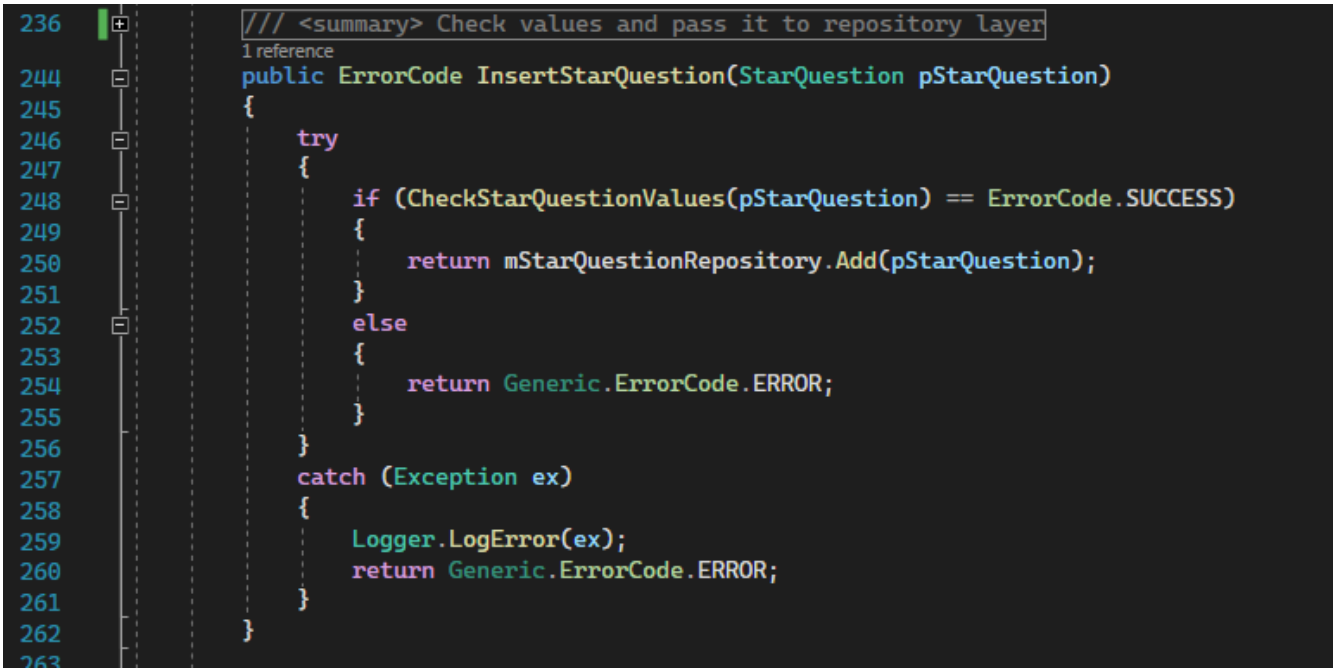
Description:

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
        1 reference
188     public ErrorCode InsertSmileyQuestion(SmileyQuestion pSmileyQuestion)
189     {
190         try
191         {
192             if (CheckSmileyQuestionValues(pSmileyQuestion) == ErrorCode.SUCCESS)
193             {
194                 return mSmileyQuestionRepository.Add(pSmileyQuestion);
195             }
196             else
197             {
198                 return Generic.ErrorCode.ERROR;
199             }
200         }
201         catch (Exception ex)
202         {
203             Logger.LogError(ex);
204             return Generic.ErrorCode.ERROR;
205         }
206     }
```

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



```
236     /// <summary> Check values and pass it to repository layer
        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     }
```

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
        1 reference
276     public ErrorCode UpdateSmileyQuestion(SmileyQuestion pSmileyQuestion)
277     {
278         try
279         {
280             if (CheckSmileyQuestionValues(pSmileyQuestion) == ErrorCode.SUCCESS)
281             {
282                 return mSmileyQuestionRepository.Update(pSmileyQuestion);
283             }
284             else
285             {
286                 return Generic.ErrorCode.ERROR;
287             }
288         }
289         catch (Exception ex)
290         {
291             Logger.LogError(ex);
292             return Generic.ErrorCode.ERROR;
293         }
294     }
295
```

```
296     /// <summary> Check values and pass it to repository layer
        1 reference
304     public ErrorCode UpdateSliderQuestion(SliderQuestion pSliderQuestion)
305     {
306         try
307         {
308             if (CheckSliderQuestionValues(pSliderQuestion) == ErrorCode.SUCCESS)
309             {
310                 return mSliderQuestionRepository.Update(pSliderQuestion);
311             }
312             else
313             {
314                 return Generic.ErrorCode.ERROR;
315             }
316         }
317         catch (Exception ex)
318         {
319             Logger.LogError(ex);
320             return Generic.ErrorCode.ERROR;
321         }
322     }
```

```
324     /// <summary> Check values and pass it to repository layer
        1 reference
332     public ErrorCode UpdateStarQuestion(StarQuestion pStarQuestion)
333     {
334         try
335         {
336             if (CheckStarQuestionValues(pStarQuestion) == ErrorCode.SUCCESS)
337             {
338                 return mStarQuestionRepository.Update(pStarQuestion);
339             }
340             else
341             {
342                 return Generic.ErrorCode.ERROR;
343             }
344         }
345         catch (Exception ex)
346         {
347             Logger.LogError(ex);
348             return Generic.ErrorCode.ERROR;
349         }
350     }
```

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
        1 reference
364     public ErrorCode GetSmileyQuestionByID(ref SmileyQuestion pSmileyQuestion)
365     {
366         try
367         {
368             if (pSmileyQuestion.ID > 0)
369             {
370                 return mSmileyQuestionRepository.Get(ref pSmileyQuestion);
371             }
372             else
373             {
374                 return Generic.ErrorCode.ERROR;
375             }
376         }
377         catch (Exception ex)
378         {
379             Logger.LogError(ex);
380             return Generic.ErrorCode.ERROR;
381         }
382     }
```

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

```
412     /// <summary> Get a question from DB and check it values
        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     }
```

```
440     /// <summary> Get all questions from DB and check it values
        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
        1 reference
475     public ErrorCode DeleteQuestionByID(int pQuestionId)
476     {
477         try
478         {
479             if (pQuestionId > 0)
480             {
481                 return mRepository.Delete(pQuestionId);
482             }
483             else
484             {
485                 return Generic.ErrorCode.ERROR;
486             }
487         }
488         catch (Exception ex)
489         {
490             Logger.LogError(ex);
491             return Generic.ErrorCode.ERROR;
492         }
493     }
```

Description:

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

Validation Functions

```
499     /// <summary> Check coommon question values and validate them
        3 references
506     private ErrorCode CheckCommonQuestionInputFields(Question pQuestion)
507     {
508         try
509         {
510             if (!String.IsNullOrWhiteSpace(pQuestion.Text)
511                 && pQuestion.Text.Length < 4000) //if Question text is not null or empty
512                 return ErrorCode.SUCCESS;
513
514             return ErrorCode.ERROR;
515         }
516         catch (Exception ex)
517         {
518             Logger.LogError(ex);
519             return ErrorCode.ERROR;
520         }
521     }
522
523     /// <summary> Check smiley question values and validate them
        3 references
530     public ErrorCode CheckSmileyQuestionValues(SmileyQuestion pSmileyQuestion)
531     {
532         try
533         {
534             if (CheckCommonQuestionInputFields(pSmileyQuestion) == ErrorCode.SUCCESS)
535                 if (pSmileyQuestion.NumberOfSmileyFaces >= 2 && pSmileyQuestion.NumberOfSmileyFaces <= 5)
536                     return ErrorCode.SUCCESS;
537
538             return ErrorCode.ERROR;
539         }
540         catch (Exception ex)
541         {
542             Logger.LogError(ex);
543             return ErrorCode.ERROR;
544         }
545     } /// Function end
```

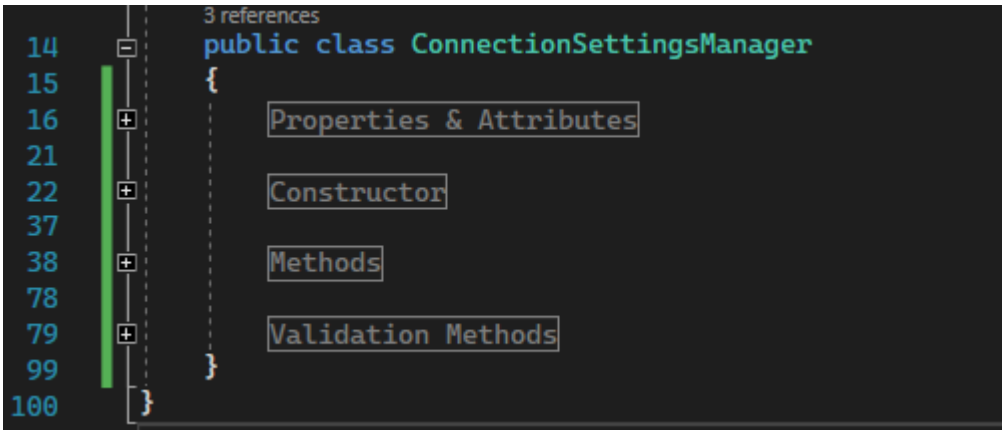
```
547     /// <summary> Check slider question values and validate them
        3 references
554     public ErrorCode CheckSliderQuestionValues(SliderQuestion pSliderQuestion)
555     {
556         try
557         {
558             if (CheckCommonQuestionInputFields(pSliderQuestion) == ErrorCode.SUCCESS)
559                 if (!String.IsNullOrWhiteSpace(pSliderQuestion.StartValueCaption) && pSliderQuestion.StartValueCaption.Length < 100)
560                     if (!String.IsNullOrWhiteSpace(pSliderQuestion.EndValueCaption) && pSliderQuestion.EndValueCaption.Length < 100)
561                         if (pSliderQuestion.StartValue < pSliderQuestion.EndValue)
562                             return ErrorCode.SUCCESS;
563
564             return ErrorCode.ERROR;
565         }
566         catch (Exception ex)
567         {
568             Logger.LogError(ex);
569             return ErrorCode.ERROR;
570         }
571     } /// Function end
572
573     /// <summary> Check star question values and validate them
        3 references
580     public ErrorCode CheckStarQuestionValues(StarQuestion pStarQuestion)
581     {
582         try
583         {
584             if (CheckCommonQuestionInputFields(pStarQuestion) == ErrorCode.SUCCESS)
585                 if (pStarQuestion.NumberOfStars >= 1 && pStarQuestion.NumberOfStars <= 10)
586                     return ErrorCode.SUCCESS;
587
588             return ErrorCode.ERROR;
589         }
590         catch (Exception ex)
591         {
592             Logger.LogError(ex);
593             return ErrorCode.ERROR;
594         }
595     } /// Function end
596
```

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

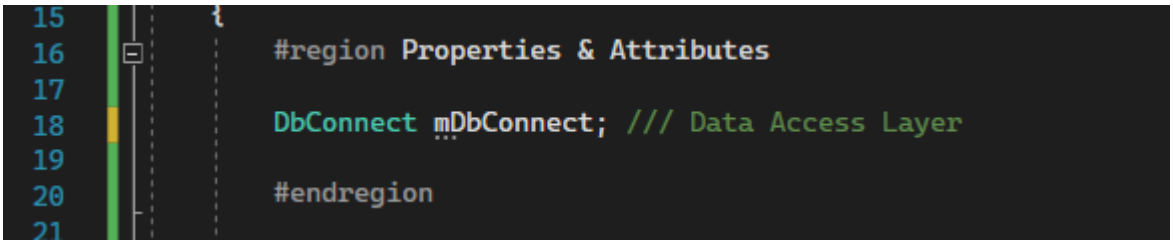
1.5.2 ConnectionSettingsManager

Code:

The code is divided into regions based on their functionality.



Properties & Attributes



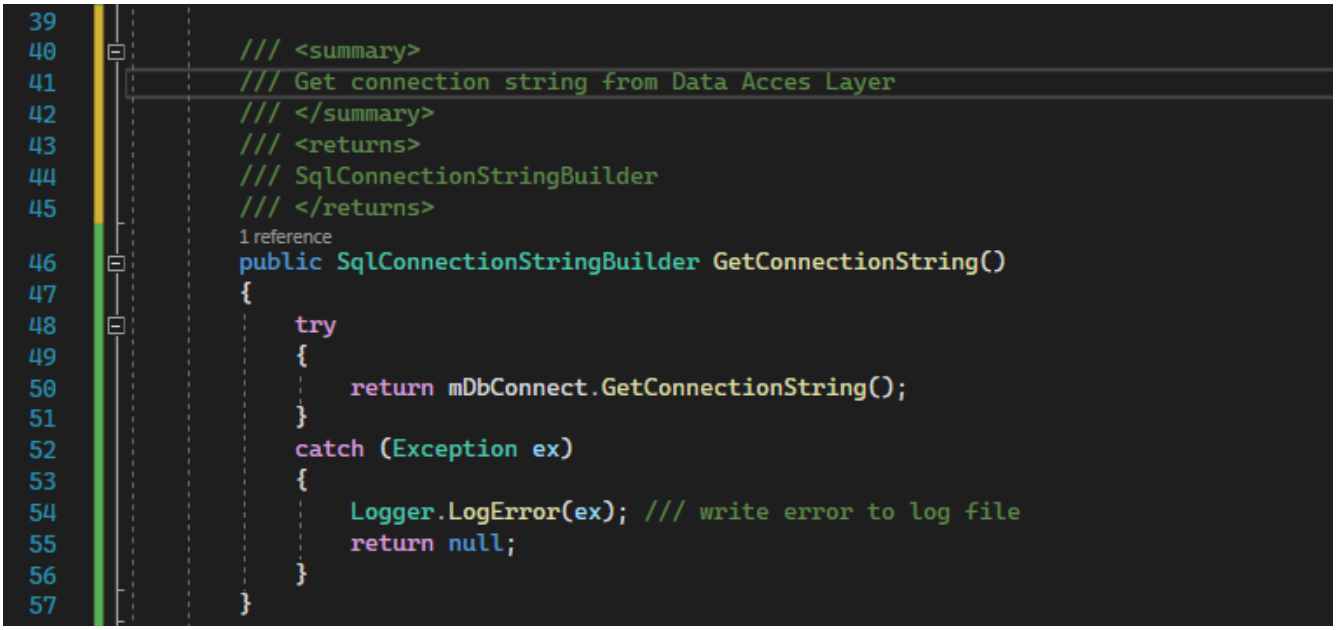
Constructor

```
21
22      #region Constructor
23
        1 reference
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



```
39
40          /// <summary>
41          /// Get connection string from Data Acces Layer
42          /// </summary>
43          /// <returns>
44          /// SqlConnectionStringBuilder
45          /// </returns>
            1 reference
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>
            1 reference
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

```
78          /// <summary>
79          /// Save passed SqlConnectionStringBuilder connection string
80          /// </summary>
81          /// <returns>
82          /// ErrorCode
83          /// </returns>
            1 reference
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:

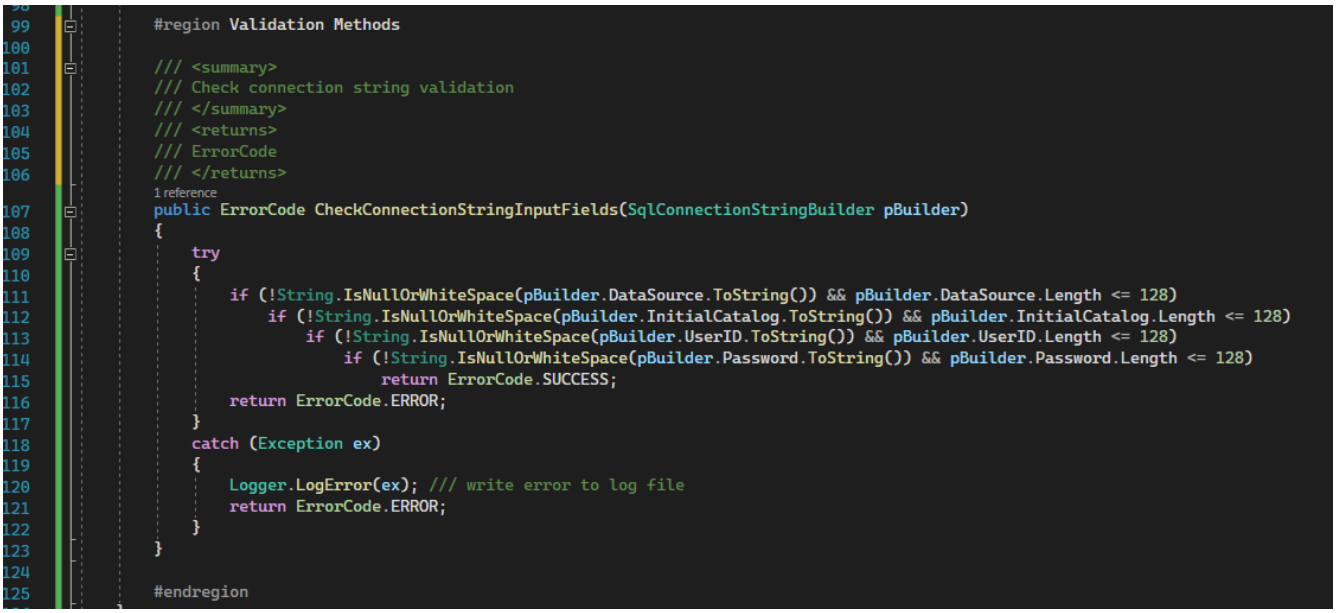
Save passed SqlConnectionStringBuilder connection string

Returns:

ErrorCode

Validation Methods

CheckConnectionStringInputFields



```
#region Validation Methods

/// <summary>
/// Check connection string validation
/// </summary>
/// <returns>
/// ErrorCode
/// </returns>
1 reference
public ErrorCode CheckConnectionStringInputFields(SqlConnectionStringBuilder pBuilder)
{
    try
    {
        if (!String.IsNullOrWhiteSpace(pBuilder.DataSource.ToString()) && pBuilder.DataSource.Length <= 128)
            if (!String.IsNullOrWhiteSpace(pBuilder.InitialCatalog.ToString()) && pBuilder.InitialCatalog.Length <= 128)
                if (!String.IsNullOrWhiteSpace(pBuilder.UserID.ToString()) && pBuilder.UserID.Length <= 128)
                    if (!String.IsNullOrWhiteSpace(pBuilder.Password.ToString()) && pBuilder.Password.Length <= 128)
                        return ErrorCode.SUCCESS;
        return ErrorCode.ERROR;
    }
    catch (Exception ex)
    {
        Logger.LogError(ex); /// write error to log file
        return ErrorCode.ERROR;
    }
}

#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>
            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          }
35
36          ///<summary>
37          /// Get all question from Questions table and save them into passed by reference list
38          ///</summary>
            3 references
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      }
52  }
```

Delete:

```
20      ///<summary>
21      /// Delete a question by it's passed ID
22      ///</summary>
        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

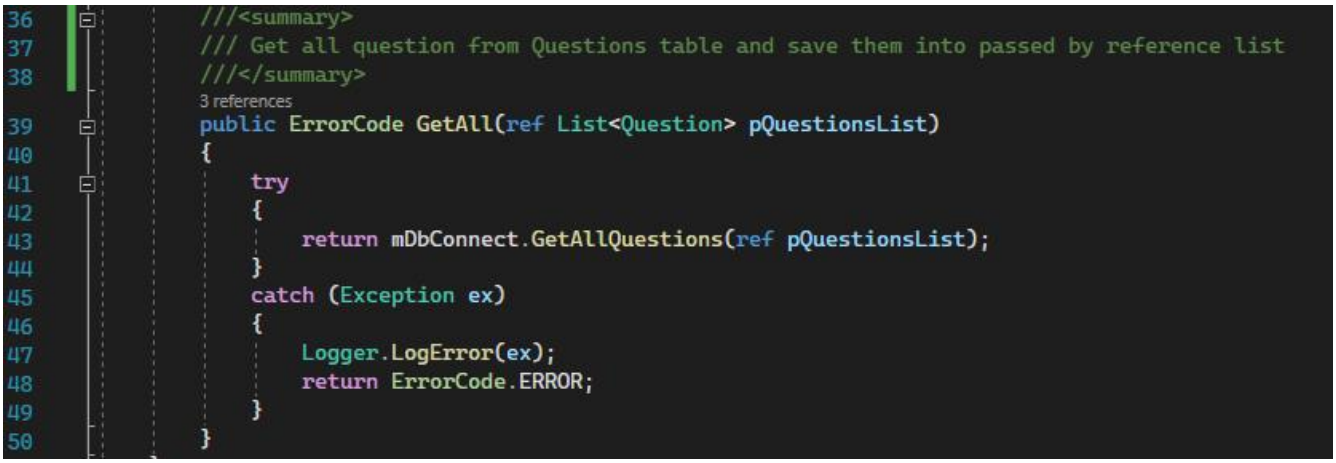
Description:

Delete a question by its passed ID

Returns:

ErrorCode

GetAll:



```
///<summary>
/// Get all question from Questions table and save them into passed by reference list
///</summary>
3 references
public ErrorCode GetAll(ref List<Question> pQuestionsList)
{
    try
    {
        return mDbConnect.GetAllQuestions(ref pQuestionsList);
    }
    catch (Exception ex)
    {
        Logger.LogError(ex);
        return ErrorCode.ERROR;
    }
}
```

Parameter:

```
ref List<Question> pQuestionsList
```

Description:

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

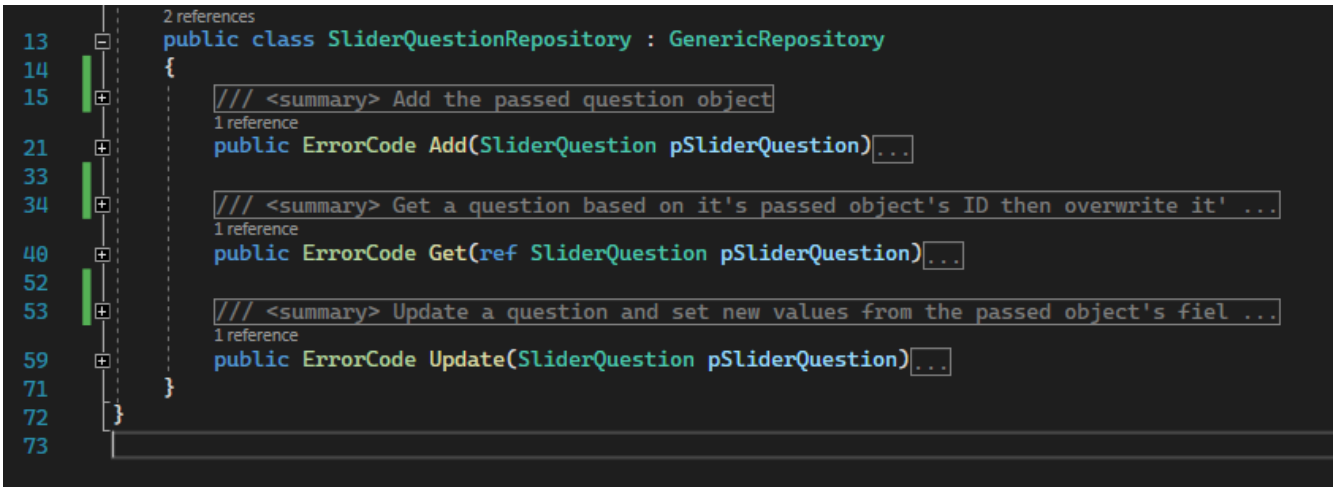
Returns:

ErrorCode

1.6.2 SliderQuestionRepository

This class inherits from GenericRepository Class

Code:



Add:

```
2 references
public class SliderQuestionRepository : GenericRepository
{
    /// <summary>
    /// Add the passed question object to the database through Data Access Layer
    /// </summary>
    /// <returns>
    /// ErrorCode
    /// </returns>
    1 reference
    public ErrorCode Add(SliderQuestion pSliderQuestion)
    {
        try
        {
            return mDbConnect.InsertSliderQuestion(pSliderQuestion);
        }
        catch (Exception ex)
        {
            Logger.LogError(ex);
            return ErrorCode.ERROR;
        }
    }
```

Parameters:

pSliderQuestion

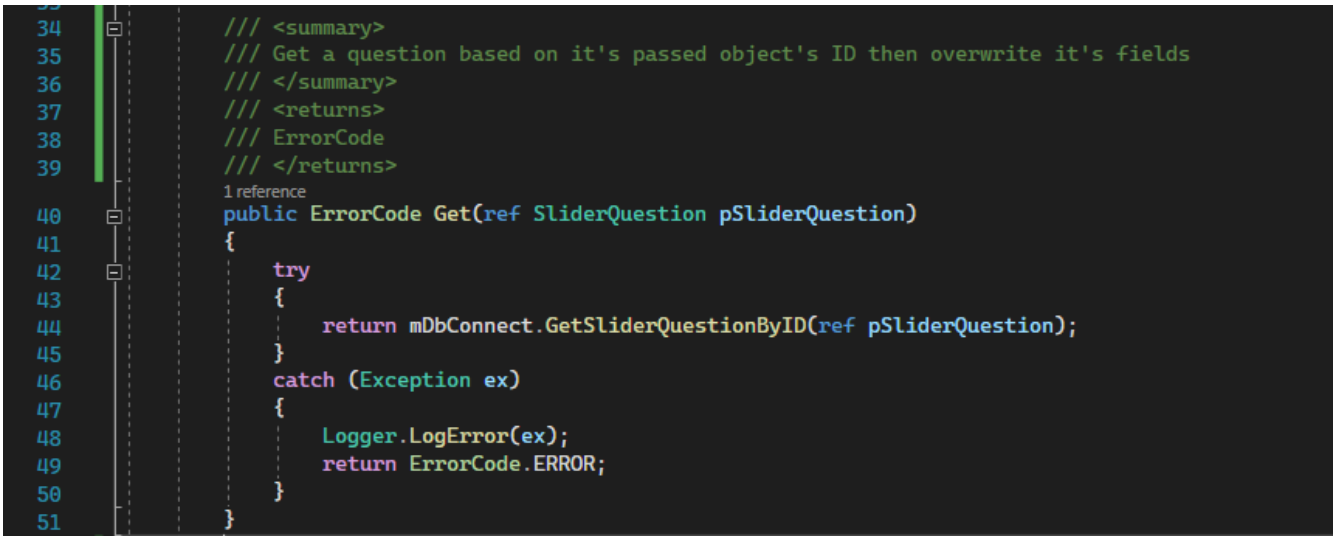
Description:

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

Returns:

ErrorCode

Get:



```
/// <summary>
/// Get a question based on it's passed object's ID then overwrite it's fields
/// </summary>
/// <returns>
/// ErrorCode
/// </returns>
1 reference
public ErrorCode Get(ref SliderQuestion pSliderQuestion)
{
    try
    {
        return mDbConnect.GetSliderQuestionByID(ref pSliderQuestion);
    }
    catch (Exception ex)
    {
        Logger.LogError(ex);
        return ErrorCode.ERROR;
    }
}
```

Parameters:

pSliderQuestion

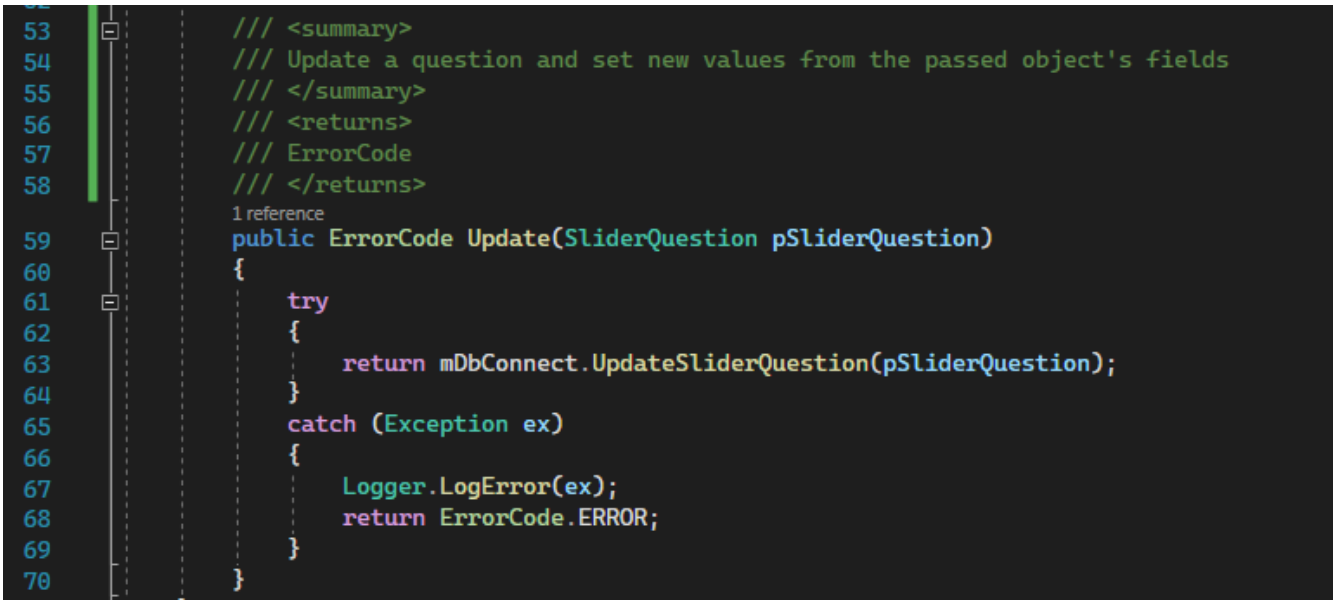
Description:

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

Returns:

ErrorCode

Update:



```
53      /// <summary>
54      /// Update a question and set new values from the passed object's fields
55      /// </summary>
56      /// <returns>
57      /// ErrorCode
58      /// </returns>
        1 reference
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

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

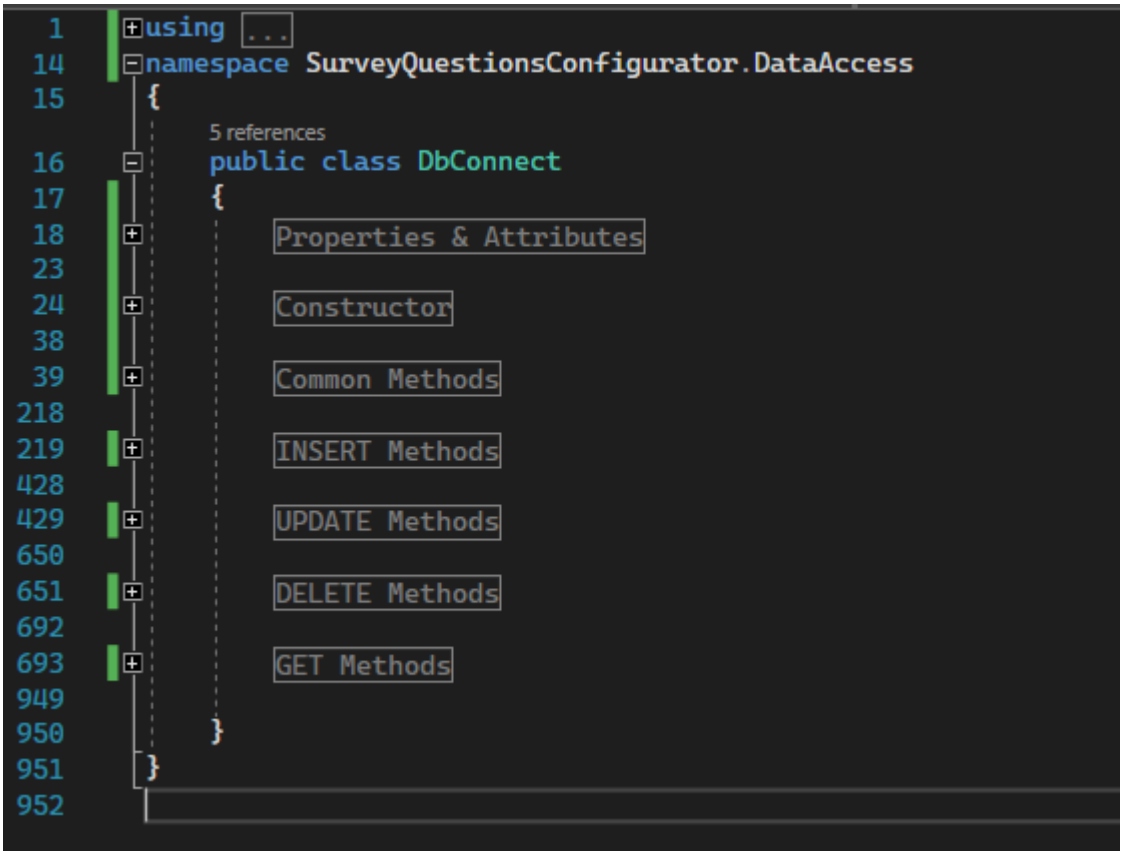
This class does the exact functionality that SliderQuestionRepository class in section 3.5.2 does but to its corresponding star questions

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.



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.

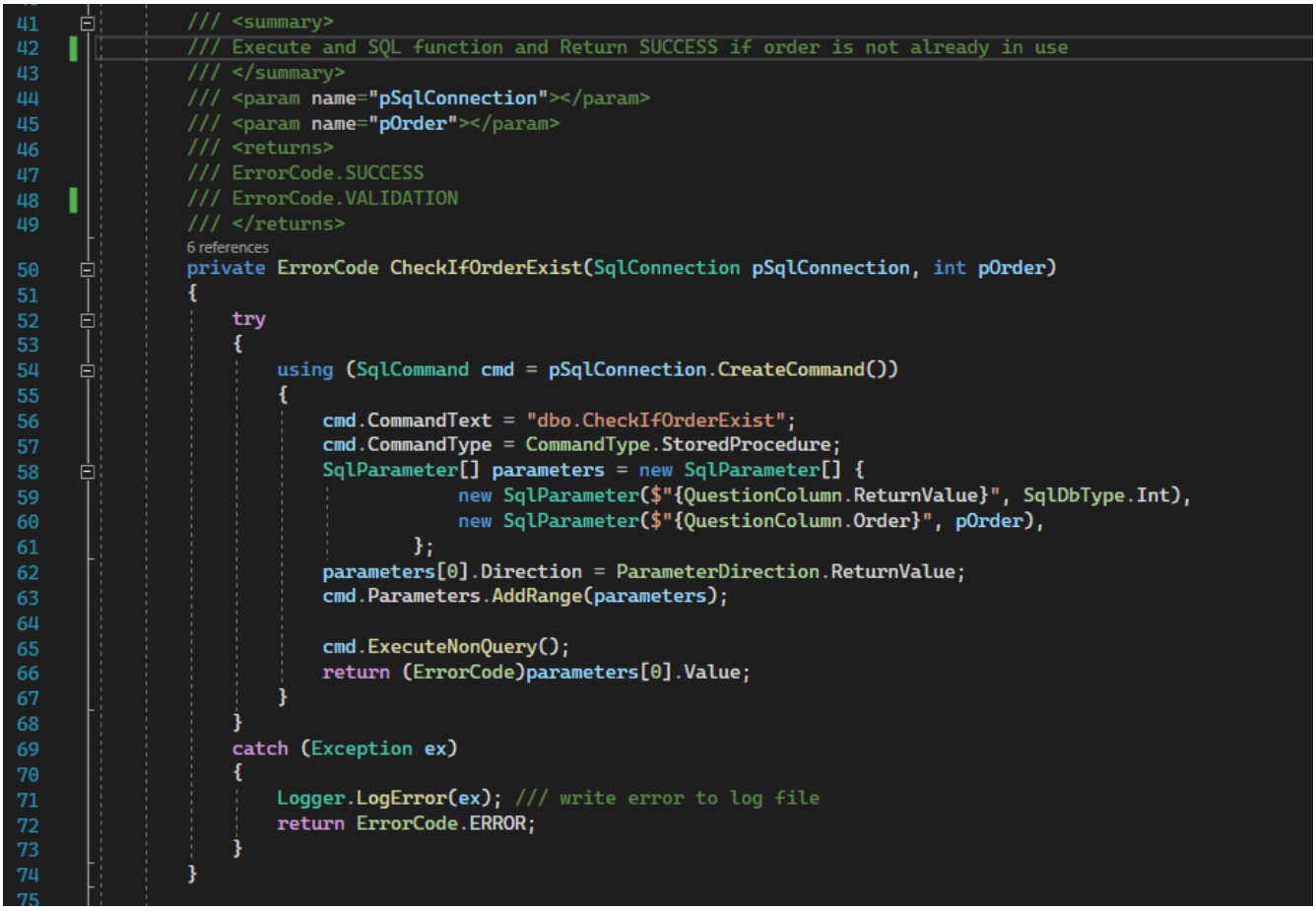
Constructor

```
24      #region Constructor
        2 references
25      public DbConnect()
26      {
27          try
28          {
29              mSqlConnectionSettings = ConfigurationManager.ConnectionStrings[0];
30          }
31          catch (Exception ex)
32          {
33              Logger.LogError(ex); /// write error to log file
34          }
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>
        6 references
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                              new SqlParameter($"{QuestionColumn.ReturnValue}", SqlDbType.Int),
60                              new SqlParameter($"{QuestionColumn.Order}", pOrder),
61                          };
62                  parameters[0].Direction = ParameterDirection.ReturnValue;
63                  cmd.Parameters.AddRange(parameters);
64
65                  cmd.ExecuteNonQuery();
66                  return (ErrorCode)parameters[0].Value;
67              }
68          }
69          catch (Exception ex)
70          {
71              Logger.LogError(ex); /// write error to log file
72              return ErrorCode.ERROR;
73          }
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.

Line 59-> sets an output parameter that saved the returned value form the SQL function.

Returns:

ErrorCode.SUCCESS

ErrorCode.VALIDATION

CheckIfUpdatingSameQuestion:

```
75
76          /// <summary>
77          /// Return the ID of question based on it's order
78          /// </summary>
79          /// <param name="pSqlConnection"></param>
80          /// <param name="pOrder"></param>
81          /// <returns>
82          /// ErrorCode.SUCCESS
83          /// ErrorCode.ERROR
84          /// </returns>
            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                                  new SqlParameter($"{QuestionColumn.ReturnValue}", SqlDbType.Int),
95                                  new SqlParameter($"{QuestionColumn.Order}", pOrder),
96                                  new SqlParameter($"{QuestionColumn.ID}", pQuestionID),
97                              };
98                      parameters[0].Direction = ParameterDirection.ReturnValue; /// retured value from function
99                      cmd.Parameters.AddRange(parameters);
100
101                     cmd.ExecuteNonQuery();
102
103                     return (ErrorCode)parameters[0].Value;
104                 }
105             }
106             catch (Exception ex)
107             {
108                 Logger.LogError(ex); /// write error to log file
109                 return ErrorCode.ERROR;
110             }
111         }
```

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:

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

Parameters:

pBuilder -> SqlConnectionStringBuilder

Description:

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

Line 162-> Update local global connecting string with the new saved connection string.

Returns:

ErrorCode.SUCCESS

ErrorCode.ERROR

GetConnectionString:

```
171
172         /// <summary>
173         /// Get used connection string from config file
174         /// </summary>
175         /// <returns>
176         /// ErrorCode.SUCCESS
177         /// ErrorCode.ERROR
178         /// </returns>
            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:

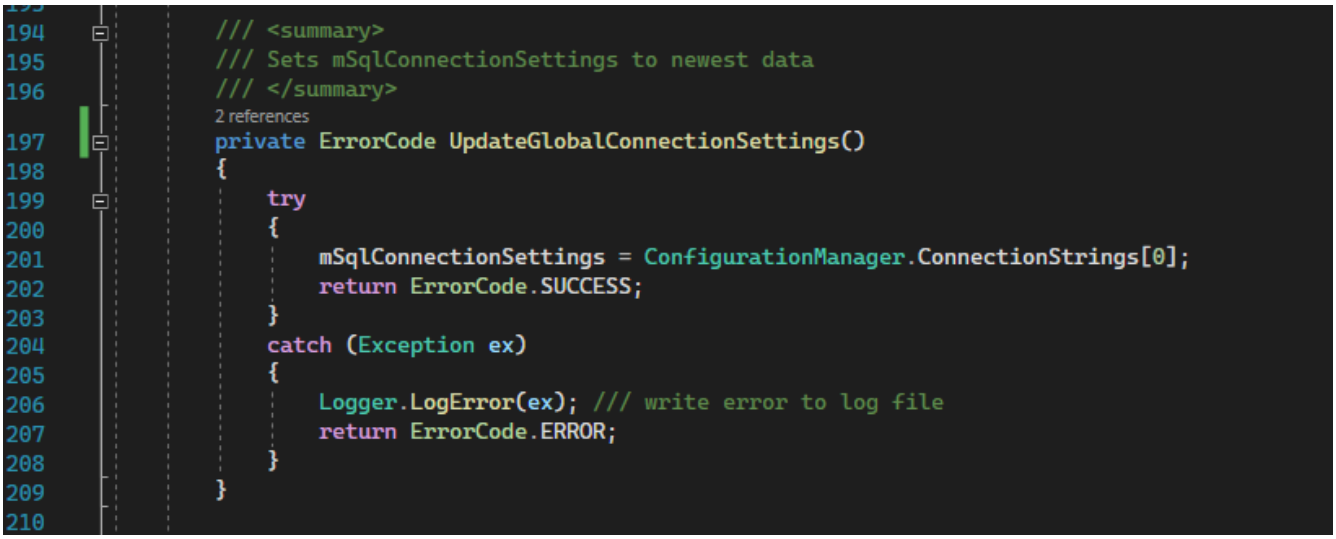
Get the connection string from app.config file that is saved in mSqlConnectionSettings.

Returns:

SqlConnectionStringBuilder

null

UpdateGlobalConnectionSettings:



```
194     /// <summary>
195     /// Sets mSqlConnectionSettings to newest data
196     /// </summary>
        2 references
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

```
/// <summary>
/// 1) Open a transaction
/// 2) Check if desired order is already in use before inserting (SQL Function)
/// 3) Perform insert stored procedure -> get the result as ErrorCode
/// 4) return return from function with the corresponding ErrorCode
/// </summary>
/// <returns>
/// ErrorCode.SUCCESS
/// ErrorCode.ERROR
/// ErrorCode.VALIDATION
/// </returns>
1 reference
public ErrorCode InsertSmileyQuestion(SmileyQuestion pSmileyQuestion)
{
    /// Try to insert a new smiley question into "Smiley_Questions" table
    try
    {
        /// ErrorCode to be returned
        ErrorCode tOrderStatusResult = ErrorCode.ERROR;

        using (TransactionScope transactionScope = new TransactionScope())
        {
            using (SqlConnection sqlConnection = new SqlConnection())
            {
                sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
                sqlConnection.Open();

                /// Check if order is already in use
                tOrderStatusResult = CheckIfOrderExist(sqlConnection, pSmileyQuestion.Order);

                /// return if order already exist
                if (tOrderStatusResult == ErrorCode.VALIDATION)
                    return ErrorCode.VALIDATION;

                /// if order is not in use -> insert a question with the same order
                using (SqlCommand cmd = sqlConnection.CreateCommand())
                {
                    cmd.CommandText = "dbo.INSERT_SMILEY_QUESTION";
                    cmd.CommandType = CommandType.StoredProcedure;

                    SqlParameter[] parameters = new SqlParameter[] {
                        new SqlParameter($"{QuestionColumn.Order}", pSmileyQuestion.Order),
                        new SqlParameter($"{QuestionColumn.Text}", pSmileyQuestion.Text),
                        new SqlParameter($"{QuestionColumn.Type}", pSmileyQuestion.Type),
                        new SqlParameter($"{QuestionColumn.NumberOfSmileyFaces}", pSmileyQuestion.NumberOfSmileyFaces)
                    };
                    cmd.Parameters.AddRange(parameters);
                    tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
                }

                if (tOrderStatusResult == ErrorCode.SUCCESS)
                {
                    /// If everything is okay -> COMMIT Transaction
                    transactionScope.Complete();
                    return ErrorCode.SUCCESS;
                }

                return ErrorCode.ERROR;
            }
        }
    }
```

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>
        1 reference
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 = $@"dbo.INSERT_SLIDER_QUESTION";
323                         cmd.CommandType = CommandType.StoredProcedure;
324
325                         SqlParameter[] parameters = new SqlParameter[] {
326                             new SqlParameter($"{QuestionColumn.Text}", pSliderQuestion.Text),
327                             new SqlParameter($"{QuestionColumn.Order}", pSliderQuestion.Order),
328                             new SqlParameter($"{QuestionColumn.Type}", pSliderQuestion.Type),
329                             new SqlParameter($"{QuestionColumn.StartValue}", pSliderQuestion.StartValue),
330                             new SqlParameter($"{QuestionColumn.EndValue}", pSliderQuestion.EndValue),
331                             new SqlParameter($"{QuestionColumn.StartValueCaption}", pSliderQuestion.StartValueCaption),
332                             new SqlParameter($"{QuestionColumn.EndValueCaption}", pSliderQuestion.EndValueCaption)
333                         };
334                         cmd.Parameters.AddRange(parameters);
335
336                         tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
337                     }
338
339                     if (tOrderStatusResult == ErrorCode.SUCCESS)
340                     {
341                         /// If everything is okay -> COMMIT Transaction
342                         transactionScope.Complete();
343                         return ErrorCode.SUCCESS;
344                     }
345
346                     return ErrorCode.ERROR;
347                 }
```

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>
            1 reference
368         public ErrorCode InsertStarQuestion(StarQuestion pStarQuestion)
369         {
370             try
371             {
372                 /// ErrorCode to be returned
373                 ErrorCode tOrderStatusResult = ErrorCode.ERROR;
374
375                 using (TransactionScope transactionScope = new TransactionScope())
376                 {
377                     using (SqlConnection sqlConnection = new SqlConnection())
378                     {
379                         sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
380                         sqlConnection.Open();
381
382                         /// Check if order is already in use
383                         tOrderStatusResult = CheckIfOrderExist(sqlConnection, pStarQuestion.Order);
384
385                         /// return if order already exist
386                         if (tOrderStatusResult == ErrorCode.VALIDATION)
387                             return ErrorCode.VALIDATION;
388
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                                 new SqlParameter($"{QuestionColumn.Order}", pStarQuestion.Order),
398                                 new SqlParameter($"{QuestionColumn.Text}", pStarQuestion.Text),
399                                 new SqlParameter($"{QuestionColumn.Type}", pStarQuestion.Type),
400                                 new SqlParameter($"{QuestionColumn.NumberOfStars}", pStarQuestion.NumberOfStars)
401
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

```
#region UPDATE Methods

/// <summary>
/// 1) Open a transaction
/// 2) Check if desired order is already in use before inserting (SQL Function)
/// 3) Get ID of the desired order you want to insert
/// 4) if order already in use and the ID of that order is not the same ID the question that I am already updating
/// 5) -> Perform update stored procedure -> get the result as ErrorCode
/// 6) return from function with the corresponding ErrorCode
/// </summary>
/// <returns>
/// ErrorCode.SUCCESS
/// ErrorCode.ERROR
/// ErrorCode.VALIDATION
/// </returns>
1 reference
public ErrorCode UpdateSmileyQuestion(SmileyQuestion pSmileyQuestion)
{
    /// Try to Update a new pQuestion into "Smiley_Questions" table
    try
    {
        ErrorCode tOrderStatusResult = ErrorCode.ERROR;
        ErrorCode tIsEditingSameQuestion = ErrorCode.ERROR;

        using (TransactionScope transactionScope = new TransactionScope())
        {
            using (SqlConnection sqlConnection = new SqlConnection())
            {
                sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
                sqlConnection.Open();

                /// Check if order is already in use
                tOrderStatusResult = CheckIfOrderExist(sqlConnection, pSmileyQuestion.Order);

                /// get question ID form it's unique order
                tIsEditingSameQuestion = CheckIfUpdatingSameQuestion(sqlConnection, pSmileyQuestion.Order, pSmileyQuestion.ID);

                /// return if order already exist && the order is taken by another questionID .. ...
                if (tOrderStatusResult == ErrorCode.VALIDATION && tIsEditingSameQuestion == ErrorCode.ERROR)
                    return ErrorCode.VALIDATION;


                using (SqlCommand cmd = sqlConnection.CreateCommand())
                {
                    cmd.CommandText = "dbo.UPDATE_SMILEY_QUESTION";
                    cmd.CommandType = CommandType.StoredProcedure;

                    SqlParameter[] parameters = new SqlParameter[] {
                        new SqlParameter($"{QuestionColumn.ID}", pSmileyQuestion.ID),
                        new SqlParameter($"{QuestionColumn.Order}", pSmileyQuestion.Order),
                        new SqlParameter($"{QuestionColumn.Text}", pSmileyQuestion.Text),
                        new SqlParameter($"{QuestionColumn.Type}", pSmileyQuestion.Type),
                        new SqlParameter($"{QuestionColumn.NumberOfSmileyFaces}", pSmileyQuestion.NumberOfSmileyFaces)
                    };
                    cmd.Parameters.AddRange(parameters);
                    tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
                }

                if (tOrderStatusResult == ErrorCode.SUCCESS)
                {
                    /// If everything is okay -> COMMIT Transaction
                    transactionScope.Complete();
                    return ErrorCode.SUCCESS;
                }
                return ErrorCode.ERROR;
            }
        }
    }
    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:

```
/// <summary>
/// 1) Open a transaction
/// 2) Check if desired order is already in use before inserting (SQL Function)
/// 3) Get ID of the desired order you want to insert
/// 4) if order already in use and the ID of that order is not the same ID the question that I am already updating
/// 5) -> Perform update stored procedure -> get the result as ErrorCode
/// 6) return from function with the corresponding ErrorCode
/// </summary>
/// <returns>
/// ErrorCode.SUCCESS
/// ErrorCode.ERROR
/// ErrorCode.VALIDATION
/// </returns>
1 reference
public ErrorCode UpdateSliderQuestion(SliderQuestion pSliderQuestion)
{
    try
    {
        ErrorCode tOrderStatusResult = ErrorCode.ERROR;
        ErrorCode tIsEditingSameQuestion = ErrorCode.ERROR;

        using (TransactionScope transactionScope = new TransactionScope())
        {
            using (SqlConnection sqlConnection = new SqlConnection())
            {
                sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
                sqlConnection.Open();

                /// Check if order is already in use
                tOrderStatusResult = CheckIfOrderExist(sqlConnection, pSliderQuestion.Order);

                /// get question ID form it's unique order
                tIsEditingSameQuestion = CheckIfUpdatingSameQuestion(sqlConnection, pSliderQuestion.Order, pSliderQuestion.ID);

                /// return if order already exist && the order is taken by another questionID .. ...
                if (tOrderStatusResult == ErrorCode.VALIDATION && tIsEditingSameQuestion == ErrorCode.ERROR)
                    return ErrorCode.VALIDATION;


                using (SqlCommand cmd = sqlConnection.CreateCommand())
                {
                    cmd.CommandText = "dbo.UPDATE_SLIDER_QUESTION";
                    cmd.CommandType = CommandType.StoredProcedure;

                    SqlParameter[] parameters = new SqlParameter[] {
                        new SqlParameter($"{QuestionColumn.ID}", pSliderQuestion.ID),
                        new SqlParameter($"{QuestionColumn.Order}", pSliderQuestion.Order),
                        new SqlParameter($"{QuestionColumn.Text}", pSliderQuestion.Text),
                        new SqlParameter($"{QuestionColumn.Type}", pSliderQuestion.Type),
                        new SqlParameter($"{QuestionColumn.StartValue}", pSliderQuestion.StartValue),
                        new SqlParameter($"{QuestionColumn.EndValue}", pSliderQuestion.EndValue),
                        new SqlParameter($"{QuestionColumn.StartValueCaption}", pSliderQuestion.StartValueCaption),
                        new SqlParameter($"{QuestionColumn.EndValueCaption}", pSliderQuestion.EndValueCaption),
                    };
                    cmd.Parameters.AddRange(parameters);
                    tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
                }

                if (tOrderStatusResult == ErrorCode.SUCCESS)
                {
                    /// If everything is okay -> COMMIT Transaction
                    transactionScope.Complete();
                    return ErrorCode.SUCCESS;
                }
                return ErrorCode.ERROR;
            }
        }
    }
    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>
            1 reference
589         public ErrorCode UpdateStarQuestion(StarQuestion pStarQuestion)
590         {
591             try
592             {
593                 ErrorCode tOrderStatusResult = ErrorCode.ERROR;
594                 ErrorCode tIsUpdatingSameQuestion = ErrorCode.ERROR;
595
596                 using (TransactionScope transactionScope = new TransactionScope())
597                 {
598                     using (SqlConnection sqlConnection = new SqlConnection())
599                     {
600                         sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
601                         sqlConnection.Open();
602
603                         /// Check if order is already in use
604                         tOrderStatusResult = CheckIfOrderExist(sqlConnection, pStarQuestion.Order);
605
606                         /// get question ID form it's unique order
607                         tIsUpdatingSameQuestion = CheckIfUpdatingSameQuestion(sqlConnection, pStarQuestion.Order, pStarQuestion.ID);
608
609                         /// 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                                 new SqlParameter($"{QuestionColumn.ID}", pStarQuestion.ID),
621                                 new SqlParameter($"{QuestionColumn.Order}", pStarQuestion.Order),
622                                 new SqlParameter($"{QuestionColumn.Text}", pStarQuestion.Text),
623                                 new SqlParameter($"{QuestionColumn.Type}", pStarQuestion.Type),
624                                 new SqlParameter($"{QuestionColumn.NumberOfStars}", pStarQuestion.NumberOfStars)
625                             };
626                             cmd.Parameters.AddRange(parameters);
627                             tOrderStatusResult = (ErrorCode)cmd.ExecuteScalar();
628                         }
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     #region DELETE Methods
650
651     /// <summary>
652     /// 1) Open an SQL connection
653     /// 2) Perform a delete query based on passed ID
654     /// 3) return error code based on affected rows number
655     /// </summary>
656     /// <returns>
657     /// ErrorCode.SUCCESS
658     /// ErrorCode.ERROR
659     /// ErrorCode.VALIDATION
660     /// </returns>
        1 reference
661     public ErrorCode DeleteQuestionByID(int pQuestionId)
662     {
663         try
664         {
665             using (SqlConnection sqlConnection = new SqlConnection(mSqlConnectionSettings.ConnectionString))
666             {
667                 sqlConnection.Open();
668                 using (SqlCommand cmd = sqlConnection.CreateCommand())
669                 {
670                     cmd.CommandText = $@"
671                         delete from Questions where ID = @{QuestionColumn.ID}";
672
673                     SqlParameter[] parameters = new SqlParameter[] {
674                         new SqlParameter($"{QuestionColumn.ID}", pQuestionId),
675                     };
676                     cmd.Parameters.AddRange(parameters);
677
678                     return cmd.ExecuteNonQuery() > 0 ? Generic.ErrorCode.SUCCESS : Generic.ErrorCode.VALIDATION;
679                 }
680             }
681         }
682         catch (Exception ex)
683         {
684             Logger.LogError(ex); /// write error to log file
685             return Generic.ErrorCode.ERROR;
686         }
687
688     } /// 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>
            1 reference
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
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>
        1 reference
766     public ErrorCode GetSmileyQuestionByID(ref SmileyQuestion pSmileyQuestion)
767     {
768         try
769         {
770             using (SqlConnection sqlConnection = new SqlConnection())
771             {
772                 sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
773                 sqlConnection.Open();
774
775                 using (SqlCommand cmd = sqlConnection.CreateCommand())
776                 {
777                     cmd.CommandText = $@"
778                         select Q.[ID], Q.[Order], Q.[Text], Q.[Type], SmQ.NumberOfSmileyFaces
779                         from Questions AS Q
780                         inner join Smiley_Questions AS SmQ
781                         on Q.ID = SmQ.ID
782                         where Q.ID = @{QuestionColumn.ID}
783                         ORDER BY [ORDER] ASC";
784                     SqlParameter[] parameters = new SqlParameter[] {
785                         new SqlParameter($"{QuestionColumn.ID}", pSmileyQuestion.ID),
786                     };
787                     cmd.Parameters.AddRange(parameters);
788
789                     DataTable dataTable = new DataTable();
790                     SqlDataAdapter adapter = new SqlDataAdapter(cmd);
791                     adapter.Fill(dataTable);
792
792
793                     int tID, tOrder, tNumberOfSmileyFaces;
794                     string tText;
795                     QuestionType tType;
796                     foreach (DataRow row in dataTable.Rows)
797                     {
798                         tID = (int)row[$"{QuestionColumn.ID}"];
799                         tOrder = (int)row[$"{QuestionColumn.Order}"];
800                         tText = (string)row[$"{QuestionColumn.Text}"];
801                         tType = (QuestionType)row[$"{QuestionColumn.Type}"];
802                         tNumberOfSmileyFaces = (int)row[$"{QuestionColumn.NumberOfSmileyFaces}"];
803                         pSmileyQuestion = new SmileyQuestion(tID, tOrder, tText, tType, tNumberOfSmileyFaces);
804                     }
805
806                     return pSmileyQuestion.NumberOfSmileyFaces > 0 ? Generic.ErrorCode.SUCCESS : Generic.ErrorCode.VALIDATION; /// RETURN INT32
807                 }
808             }
809         }
810         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>
            1 reference
827         public ErrorCode GetSliderQuestionByID(ref SliderQuestion pSliderQuestion)
828         {
829             try
830             {
831                 using (SqlConnection sqlConnection = new SqlConnection())
832                 {
833                     sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
834                     sqlConnection.Open();
835 
836                     using (SqlCommand cmd = sqlConnection.CreateCommand())
837                     {
838                         cmd.CommandText = $@"
839                             select Q.[ID], Q.[Order], Q.[Text], Q.[Type], SQ.StartValue, SQ.EndValue, SQ.StartValueCaption, SQ.EndValueCaption
840                             from Questions AS Q
841                             inner join Slider_Questions AS SQ
842                             on Q.ID = SQ.ID
843                             where Q.ID = @{QuestionColumn.ID}
844                             ORDER BY [ORDER] ASC";
845 
846                         SqlParameter[] parameters = new SqlParameter[] {
847                             new SqlParameter($"{QuestionColumn.ID}", pSliderQuestion.ID),
848                         };
849                         cmd.Parameters.AddRange(parameters);
850 
851                         DataTable dataTable = new DataTable();
852                         SqlDataAdapter adapter = new SqlDataAdapter(cmd);
853                         adapter.Fill(dataTable);
854 
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             }
879             catch (Exception ex)
```

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)

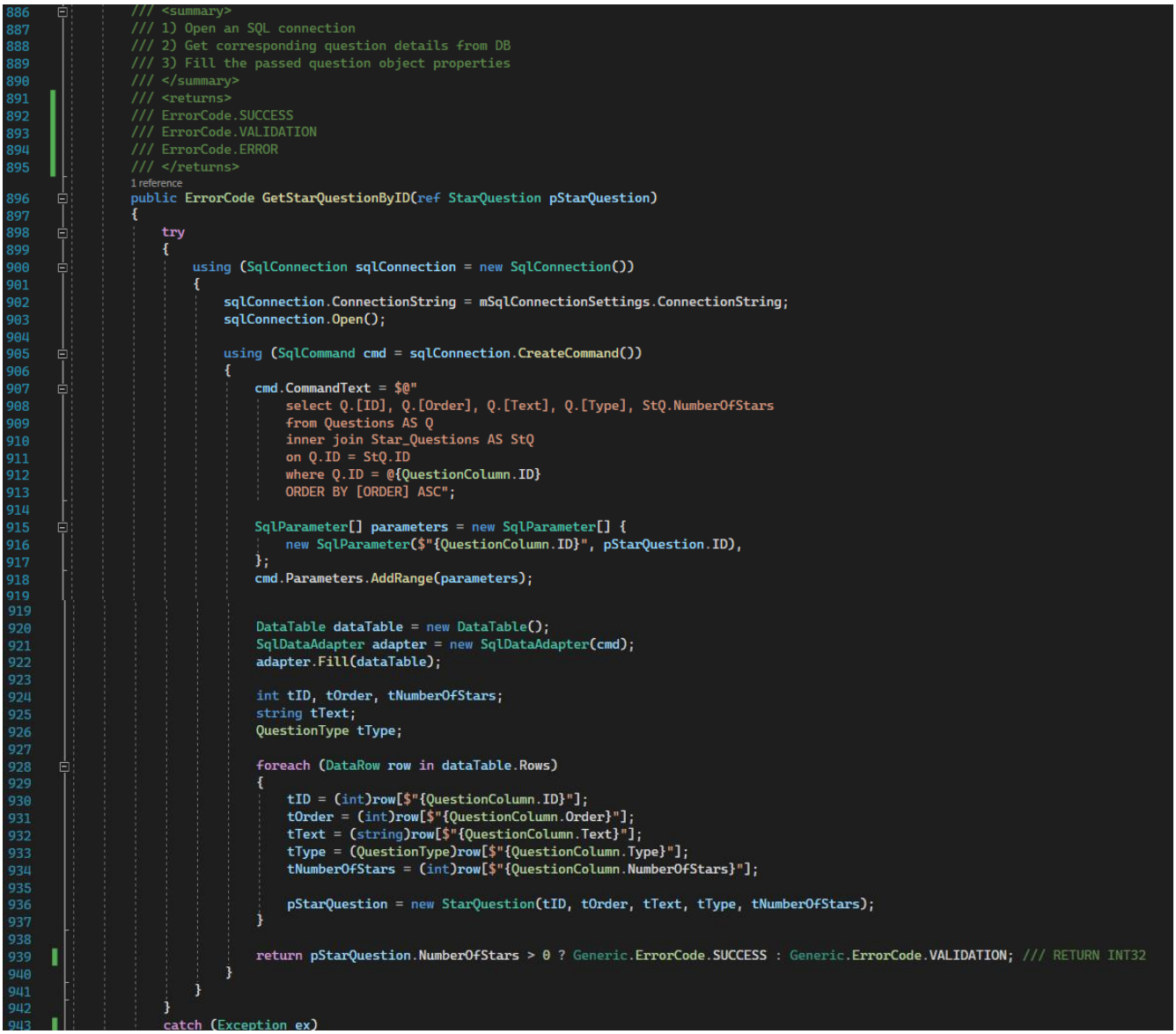
Returns:

ErrorCode.SUCCESS

ErrorCode.VALIDATION

ErrorCode.ERROR

GetStarQuestionByID



```
/// <summary>
/// 1) Open an SQL connection
/// 2) Get corresponding question details from DB
/// 3) Fill the passed question object properties
/// </summary>
/// <returns>
/// ErrorCode.SUCCESS
/// ErrorCode.VALIDATION
/// ErrorCode.ERROR
/// </returns>
1 reference
public ErrorCode GetStarQuestionByID(ref StarQuestion pStarQuestion)
{
    try
    {
        using (SqlConnection sqlConnection = new SqlConnection())
        {
            sqlConnection.ConnectionString = mSqlConnectionSettings.ConnectionString;
            sqlConnection.Open();

            using (SqlCommand cmd = sqlConnection.CreateCommand())
            {
                cmd.CommandText = $@"
                    select Q.[ID], Q.[Order], Q.[Text], Q.[Type], StQ.NumberOfStars
                    from Questions AS Q
                    inner join Star_Questions AS StQ
                    on Q.ID = StQ.ID
                    where Q.ID = @{QuestionColumn.ID}
                    ORDER BY [ORDER] ASC";

                SqlParameter[] parameters = new SqlParameter[] {
                    new SqlParameter($"{QuestionColumn.ID}", pStarQuestion.ID),
                };
                cmd.Parameters.AddRange(parameters);


                DataTable dataTable = new DataTable();
                SqlDataAdapter adapter = new SqlDataAdapter(cmd);
                adapter.Fill(dataTable);

                int tID, tOrder, tNumberOfStars;
                string tText;
                QuestionType tType;

                foreach (DataRow row in dataTable.Rows)
                {
                    tID = (int)row[$"{QuestionColumn.ID}"];
                    tOrder = (int)row[$"{QuestionColumn.Order}"];
                    tText = (string)row[$"{QuestionColumn.Text}"];
                    tType = (QuestionType)row[$"{QuestionColumn.Type}"];
                    tNumberOfStars = (int)row[$"{QuestionColumn.NumberOfStars}"];

                    pStarQuestion = new StarQuestion(tID, tOrder, tText, tType, tNumberOfStars);
                }

                return pStarQuestion.NumberOfStars > 0 ? Generic.ErrorCode.SUCCESS : Generic.ErrorCode.VALIDATION; /// RETURN INT32
            }
        }
    }
    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:

```
namespace SurveyQuestionsConfigurator.Entities
{
    55 references
    public class Generic
    {
        99+ references
        public enum ErrorCode
        {
            ERROR = -1,
            SUCCESS = 1,
            VALIDATION = 2,
            EMPTY = 3
        }

        35 references
        public enum QuestionType
        {
            SMILEY = 0,
            SLIDER = 1,
            STAR = 2
        }
        68 references
        public enum QuestionColumn
        {
            ID,
            Order,
            Text,
            Type,
            NumberOfSmileyFaces,
            NumberOfStars,
            StartValue,
            EndValue,
            StartValueCaption,
            EndValueCaption,
            ReturnValue
        }
    }
}
```

Generic data types that are used across all the application.

1.8.2 Question

Code:

```
namespace SurveyQuestionsConfigurator.Entities
{
    44 references
    public class Question
    {
        29 references
        public int ID { get; set; }
        27 references
        public int Order { get; set; }
        20 references
        public string Text { get; set; }
        19 references
        public QuestionType Type { get; set; }

        3 references
        public Question(int pID)
        {
            try
            {
                ID = pID;
            }
            catch (Exception ex)
            {
                Logger.LogError(ex);
            }
        }

        3 references
        public Question(int pID, QuestionType pType)
        {
            try
            {
                ID = pID;
                Type = pType;
            }
            catch (Exception ex)
            {
                Logger.LogError(ex);
            }
        }

        5 references
        public Question(int pID, int pOrder, string pText, QuestionType pType)
        {
            try
            {
                ID = pID;
                Order = pOrder;
                Text = pText;
                Type = pType;
            }
            catch (Exception ex)
            {
                Logger.LogError(ex);
            }
        }

        0 references
        public Question(Question pQuestion) :
            this(pQuestion.ID, pQuestion.Order, pQuestion.Text, pQuestion.Type)
        { }
```

```
0 references
public override bool Equals(object pObject)
{
    try
    {
        if (
            pObject == null ||
            this.GetType() != pObject.GetType()
            )
        {
            return false;
        }

        Question q = (Question)pObject;
        if (
            q.ID == this.ID &&
            q.Order == this.Order &&
            q.Text == this.Text
            )
        {
            return true;
        }

        return false;
    }
    catch (Exception ex)
    {
        Logger.LogError(ex);
        return false;
    }

}

/// <summary>
/// Must be overridden along with Equals
/// </summary>
0 references
public override int GetHashCode()
{
    return base.GetHashCode();
}
```

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

```
namespace SurveyQuestionsConfigurator.CommonHelpers
{
    99+ references
    public class Logger
    {
        private static string mExeFolder = System.IO.Path.GetDirectoryName(Assembly.GetEntryAssembly().Location); ///Gets the directory name of the current exe
        private static object mBalanceLock = new object(); /// used to lock writing on file
        private static object mBalanceLock2 = null; /// used to lock writing on file for the backup function

        99+ references
        public static void LogError(Exception pEx)...

        /// Log error in case main "LogError" faced an exception
        1 reference
        public static void BackUpLogger(Exception pEx)...
    }
}
```

1.9.1 LogError:

```
        99+ references
        public static void LogError(Exception pEx)
        {
            try
            {
                /// lock file writing to prevent exception System.IO.IOException :
                /// The process cannot access the file 'LogFile.txt' because it is being used by another process
                lock (mBalanceLock)
                {
                    using (StreamWriter sw = new StreamWriter($"{mExeFolder}/logs/LogFile.txt", append: true))
                    {
                        sw.WriteLine($@"
-------------- ({DateTime.Now}) --------------

-Exception Type: {pEx.GetType()}
-Exception Call Site: {pEx.TargetSite}
-Exception Short Message: {pEx.Message}
-Exception Long Message: {pEx}
-Exception Stack Trace: {pEx.StackTrace}
");
                    }
                }
            }
            catch (Exception ex2)
            {
                BackUpLogger(ex2);
            }
        }
```

Parameters:

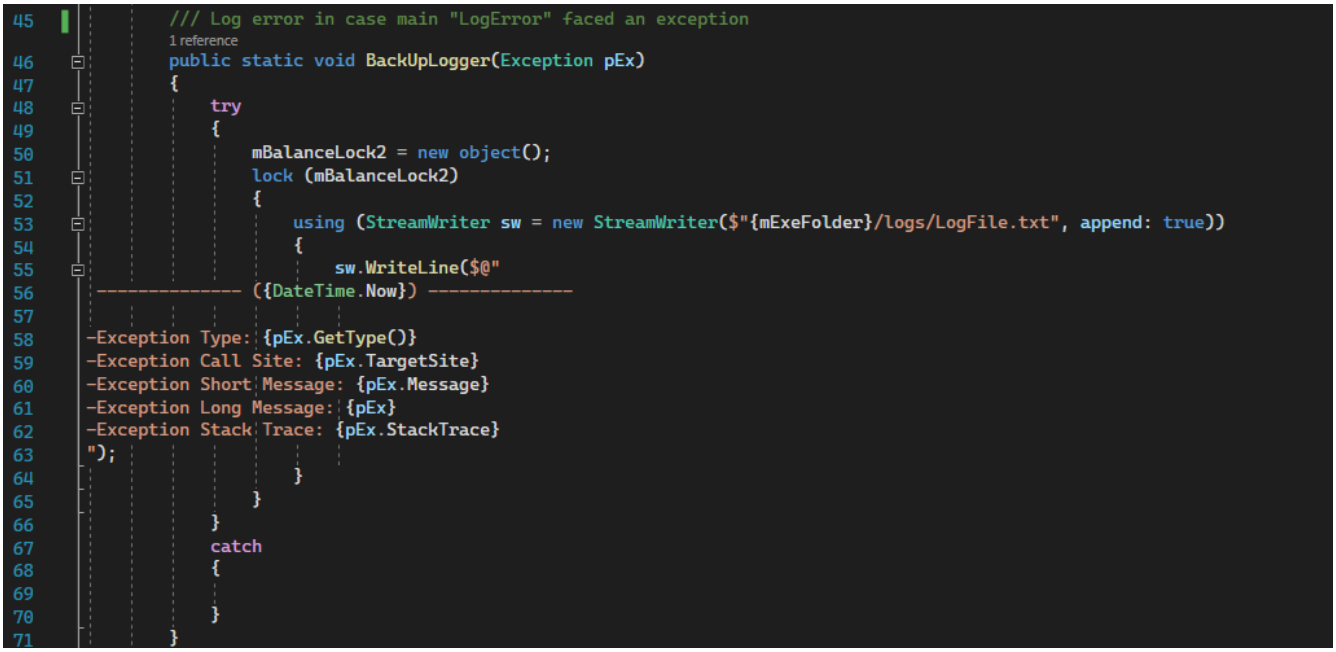
pEx-> Exception

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
            1 reference
46          public static void BackUpLogger(Exception pEx)
47          {
48              try
49              {
50                  mBalanceLock2 = new object();
51                  lock (mBalanceLock2)
52                  {
53                      using (StreamWriter sw = new StreamWriter($"{mExeFolder}/logs/LogFile.txt", append: true))
54                      {
55                          sw.WriteLine($@"
56  ---------- ({DateTime.Now}) ----------
57
58  -Exception Type: {pEx.GetType()}
59  -Exception Call Site: {pEx.TargetSite}
60  -Exception Short Message: {pEx.Message}
61  -Exception Long Message: {pEx}
62  -Exception Stack Trace: {pEx.StackTrace}
63  ");
64                      }
65                  }
66              }
67              catch
68              {
69
70              }
71          }
```

Parameters:

pEx-> Exception

Description:

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