**Question Creator Form Documentation**

This document is a documentation of the Question Creator windows form application, it will be sectioned off into three sections, **End user documentation**, **Administrator documentation** and **Developer documentation**.

**End user documentation**:

The Questions application is an application that helps manage lists of questions by editing, adding and deleting them, so they can be used in an actual survey or to be used in other applications.

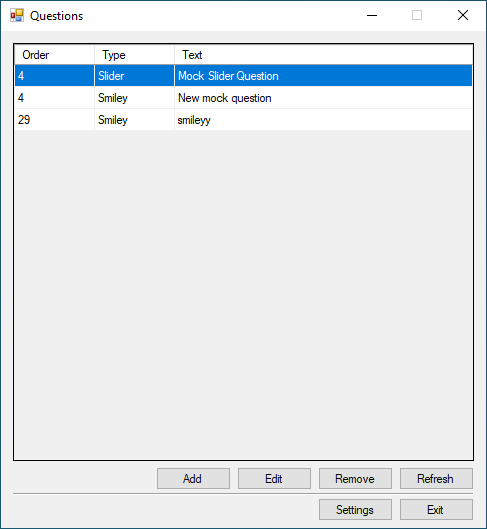
This documentation will walk you through how to do each and every single of the above abilities

**Changing the language:**

First off we will go through how to change the language, and the user is provided with two options, Arabic and English.

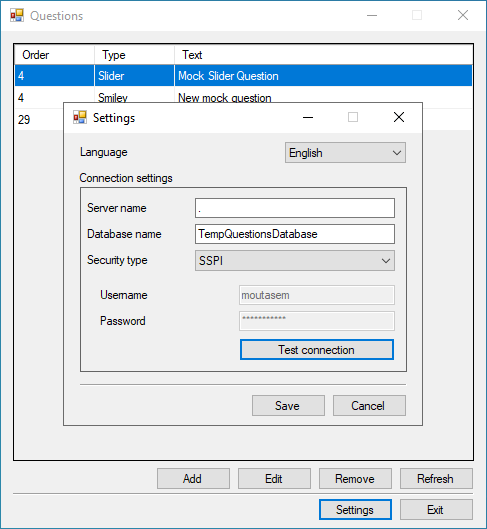
**Step 1:**

Load up the application.



**Step 2:**

Press the settings button.

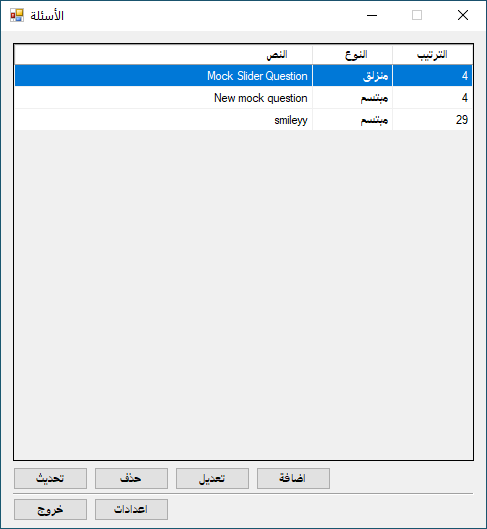
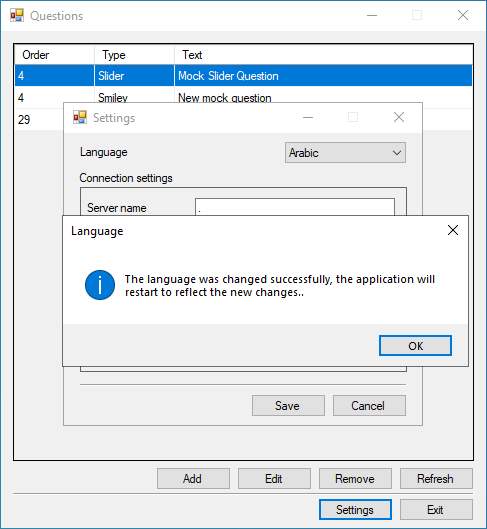


**Step 3:**

A new window will pop and there will be a drop down list that contains Arabic and English, the currently selected language will show in the box.

**Step 4:**

Chose the wanted language from the drop down then a pop-up message will show letting you know that the language was changed and a restart Is needed, then the application will restart and the selected language will be the main language.



**Note:** There will be other options to change in the settings window but please be careful on what you change if you don’t know what they mean, only an administrator or a person who knows what they mean should change them.

**Managing the questions:**

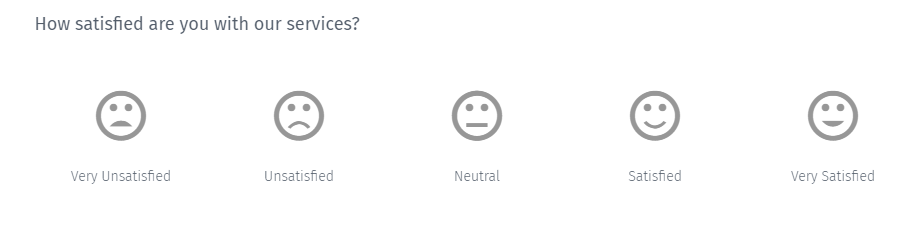
In the main application page there will be a list of questions, those can be managed by either deleting/editing/adding/sorting them.

**Adding a question:**

There are three types of questions that the user can choose to create from, I’ll first explain each one then show how to add a new question.

**Question types:**

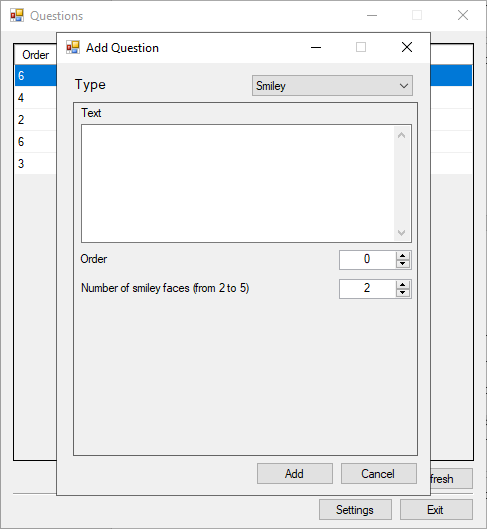
**1.Smiley question**: Smiley questions can be used to measure the satisfaction of a specific thing like opinion, feedback and experience.



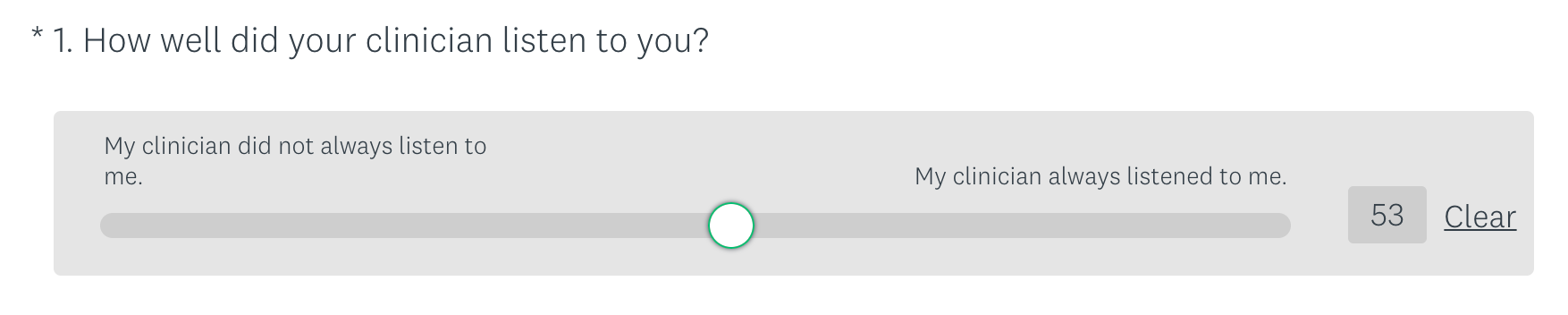
**A) Order:** the order of the question, the order can be any value from 0 to 100.

**B) Text:** the text of the question, it can’t be more than 255 characters long.

**C) Number of smiley faces:** the number of the smiley faces this question has, it can be any value from 2 to 5.



**2.Slider question:** The slider question lets the respondents to the question rate something based on a numerical scale.



**A) Order:** the order of the question, the order can be any value from 0 to 100.

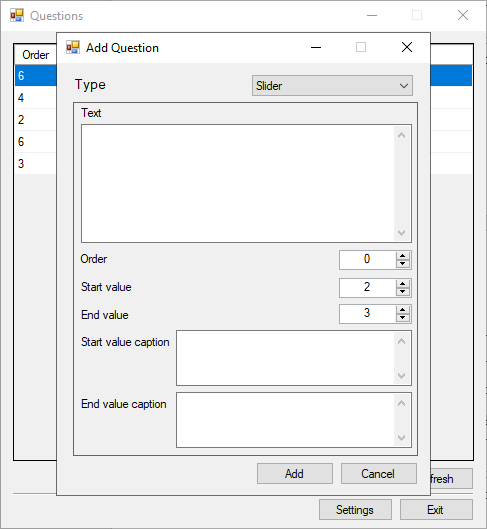
**B) Text:** the text of the question, it can’t be more than 255 characters long.

**C) Start value**: the start value of the slider question, it can be any value from 1 to 100 and it can’t be higher than the end value.

**D) End value:** the end value of the slider question, it can be any value from 1 to 100 and it can’t be lower than the start value.

**E) Start value caption:** the caption of the start value, this explains what the start value represents, it can’t be more than 255 characters long.

**F) End value caption**: the caption of the end value, this explains what the end value represents, it can’t be more than 255 characters long.



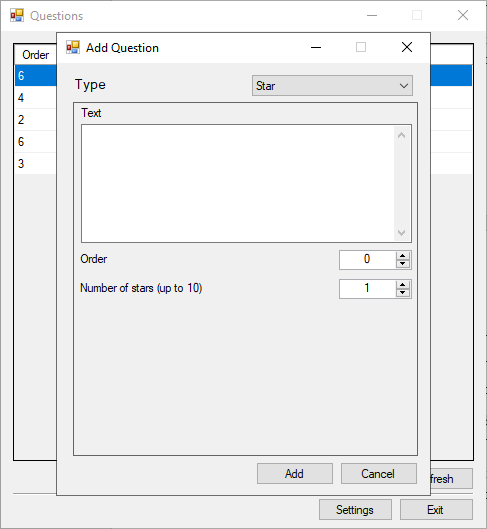
**3.Star question:** The star question can be used to respond to behavioral based questions.



**A)** **Order:** the order of the question, the order can be any value from 0 to 100.

**B)** **Text:** the text of the question, it can’t be more than 255 characters long.

**C)** **Number of stars:** the number of stars this question has, it can be any value from 1 to 10.



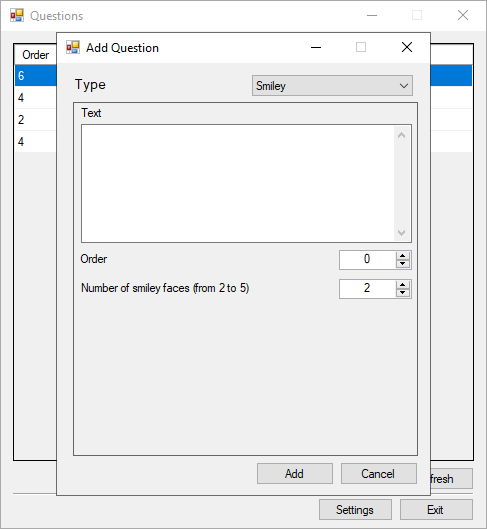
**Steps to adding new questions:**

**Step 1:**

Press the Add button.

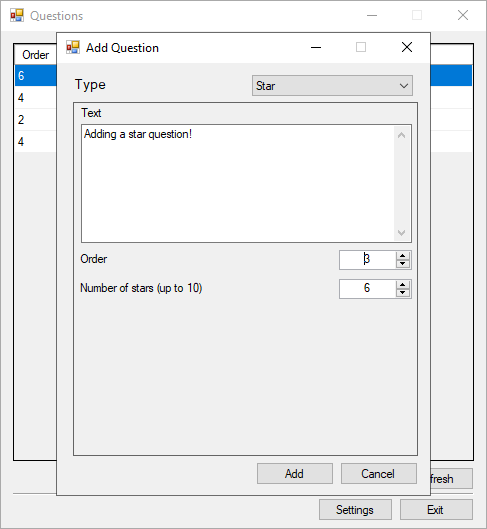
**Step 2:**

A new window will popup and will let you chose and input the different question data.



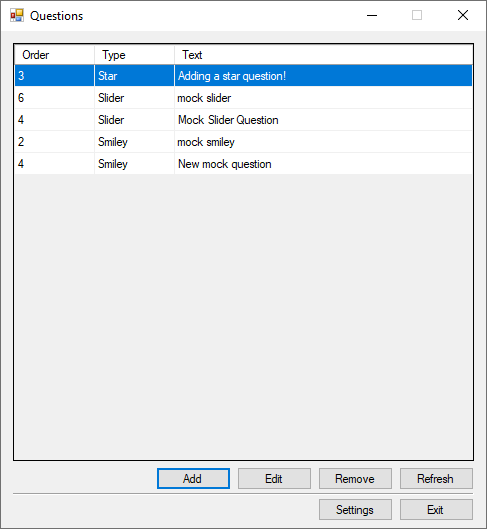
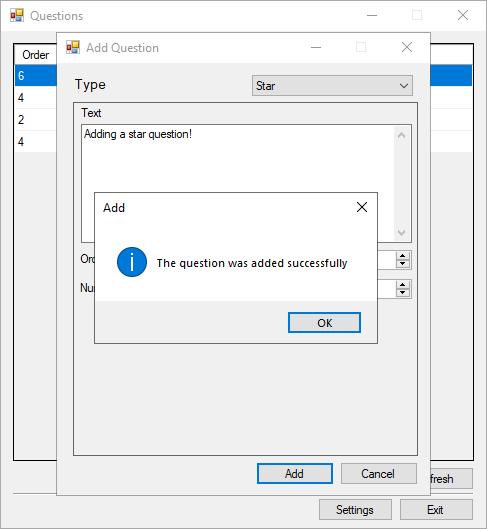
**Step 3:**

After inputting/choosing the different data for the question, press the Add button.



**Step 4:**

A popup message will appear letting you know if the question was added successfully or not, if it was the question form window will close and return to the main window and show the newly created question.

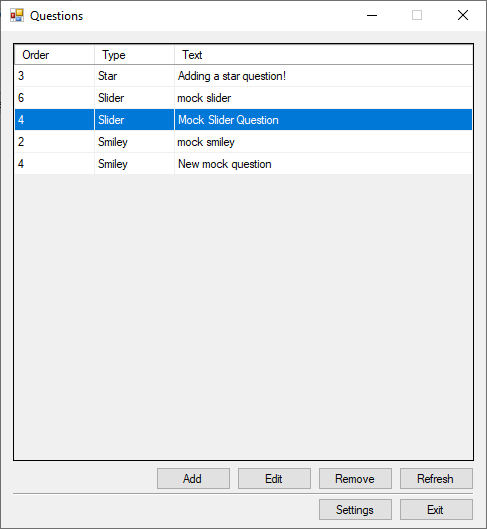


**Note:** pressing the Cancel button at any moment will close the new question window and nothing will occur.

**Editing a question:**

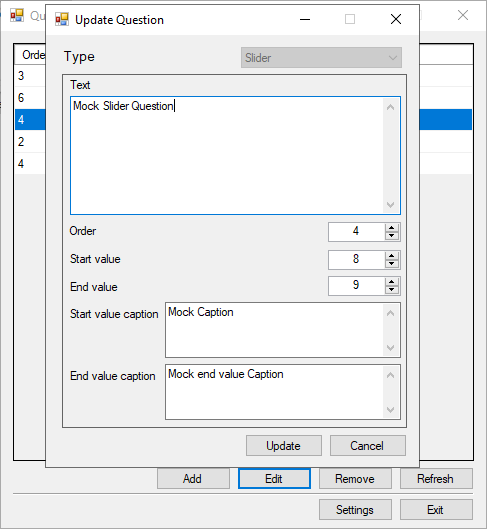
**Step 1:**

Select the question you want to edit.



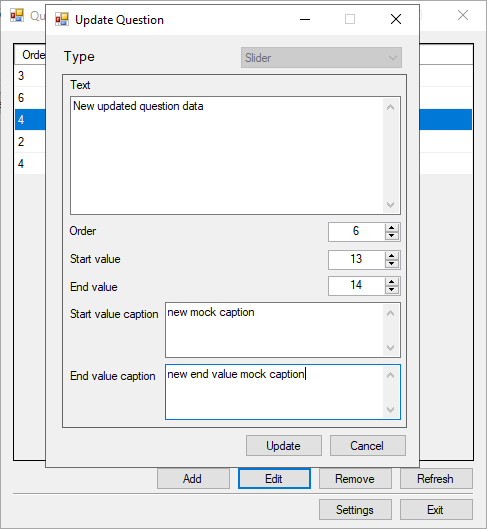
**Step 2:**

Press the Edit button.



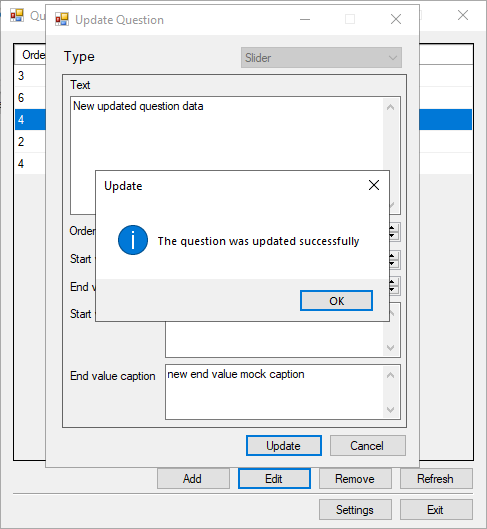
**Step 3:**

A new window will appear with the questions data, then you can change what ever new data you want in that question.



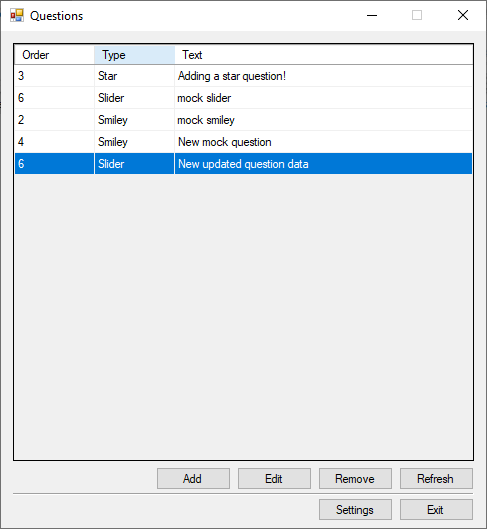
**Step 4:**

After finishing changing the data, press the Update button to update the questions data.



**Step 5:**

A popup message will appear letting you know if the question was updated successfully or not, if it was the question form window will close and return to the main window and show the new updated question.

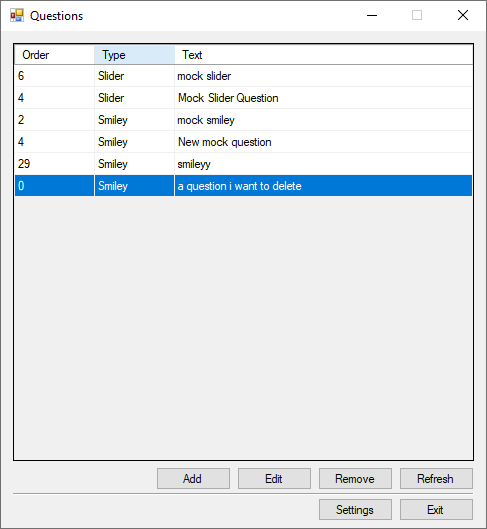


**Note:** pressing the Cancel button at any moment will close the new question window and nothing will occur.

**Deleting questions:**

**Step 1:**

Select the question that you want to be deleted.

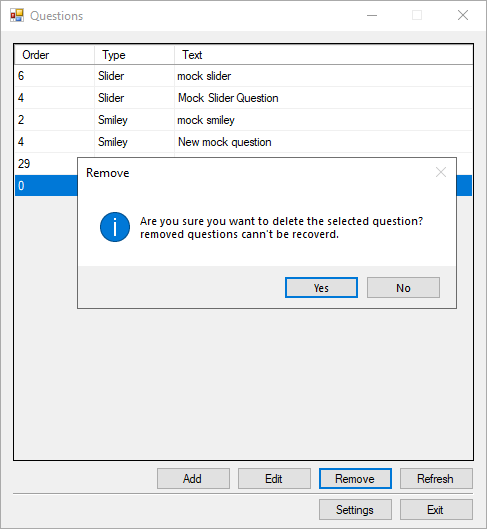


**Step 2:**

Press the remove button.

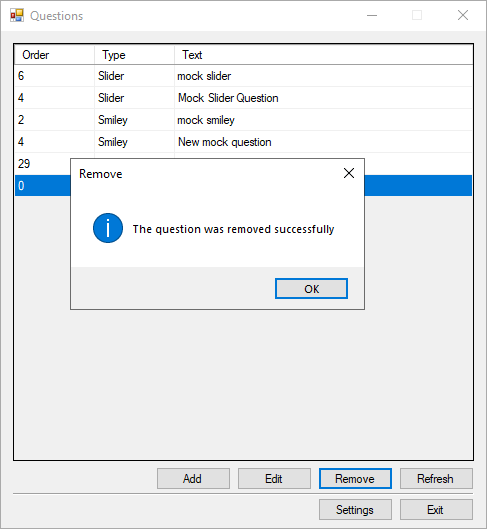
**Step 3:**

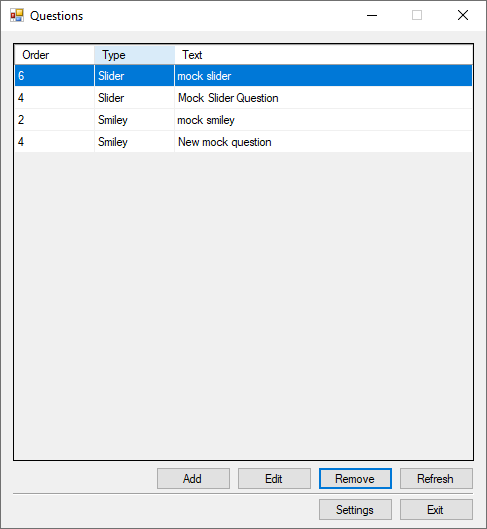
Respond to the confirmation of deletion.



**Step 4:**

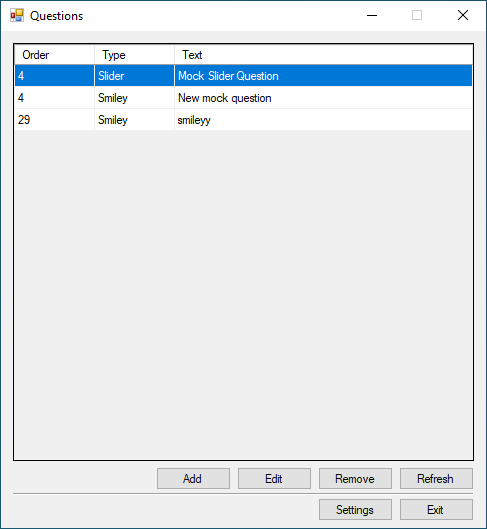
A popup message will appear letting you know if the selected question was deleted or not.

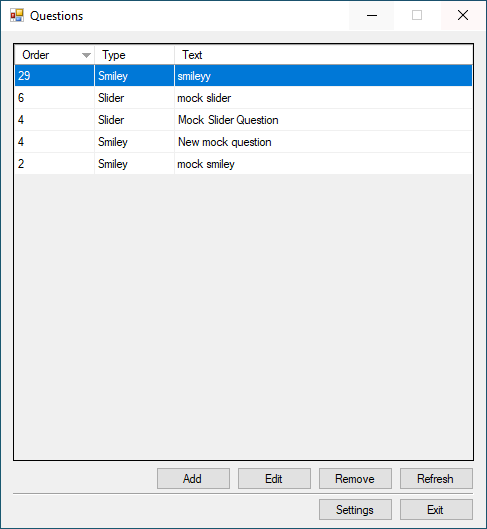
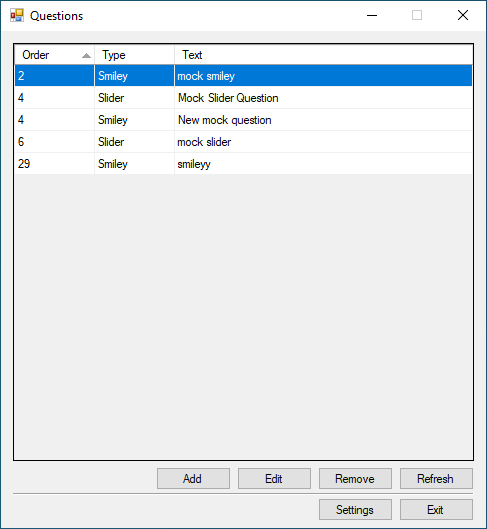




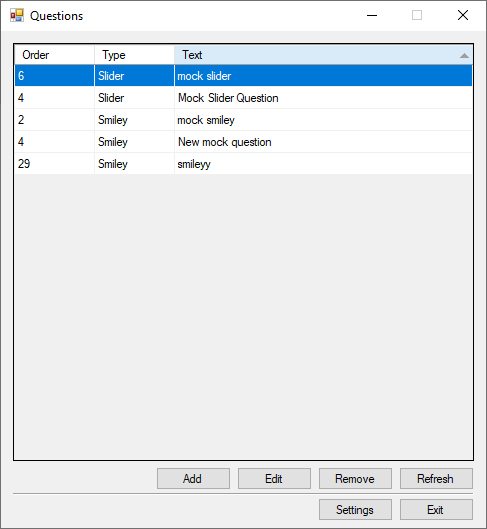
**Sorting questions:**

To sort the questions, you simple just need to click on the header you want the questions to be sorted with, tap one time on the header to sort them in an ascending order, two times to sort them in descending order.



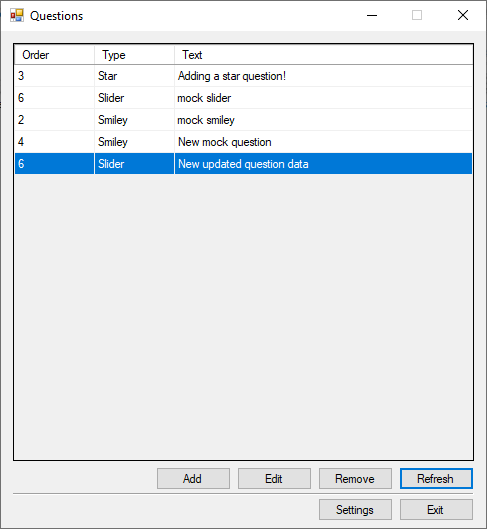


Note: when you press on a different header to sort the data by, the data will only be sorted by that header, and ignoring any previously chosen headers to be sorted with.



**Refreshing the questions data:**

Refreshing the data is very simple, you just simply press the Refresh button to get the updated questions.



**Note:** the questions data is automatically refreshed every 10 seconds, so you don’t have to bother with updating it every once in a while.

**Administrator documentation**:

The application administrator should be the one changing the connection settings.

This documentation will explain the connection settings form and what they should have in them.

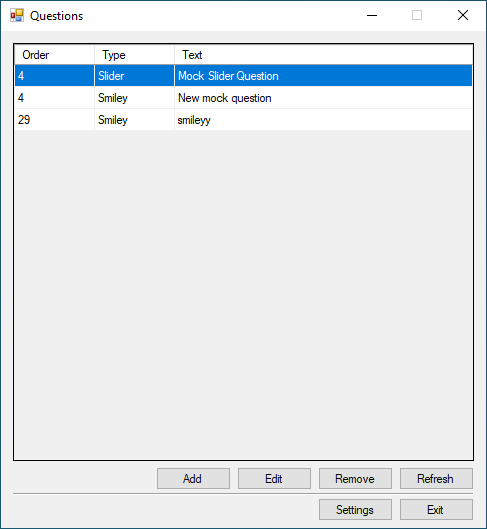
**The connection settings fields:**

* **Server name:** The name of the server that is currently hosting the database.
* **Database name:** The name of the database that the application should connect to.
* **Security type:** The connection security type, this has two options:
  + **SSPI (Windows user authentication):** This just means that the connection will be built upon the current signed in user in the current windows machine, having this option selected will disable the username/password input fields since they are not needed.
  + **SQL Server authentication:** Selecting this type will enable the administrator to input a username and a password, these should be registered in the database server and should have authentication for using the selected database.

**Changing the connection settings:**

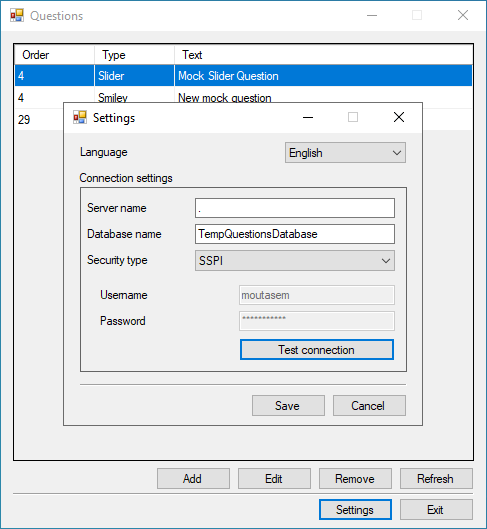
**Step 1:**

Load up the application.



**Step 2:**

Press the settings button.

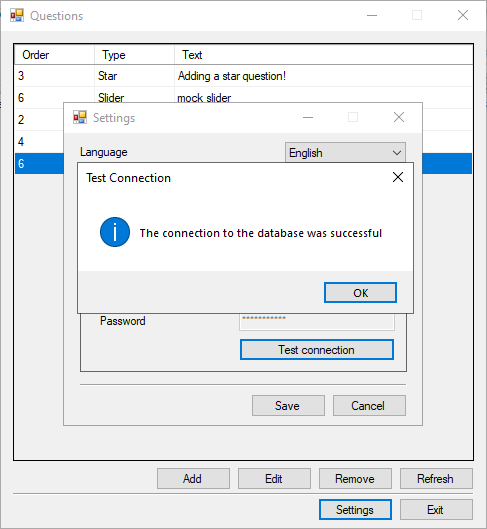


**Step 3:**

A new window will appear that contains the application settings from current language to the connection settings.

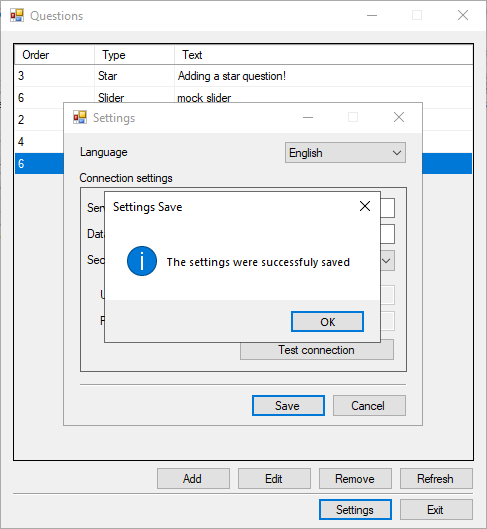
**Step 4:**

Change the connection settings as you like, then always try testing the connection settings before saving them, you can test the connection settings by simply pressing the test connection button.

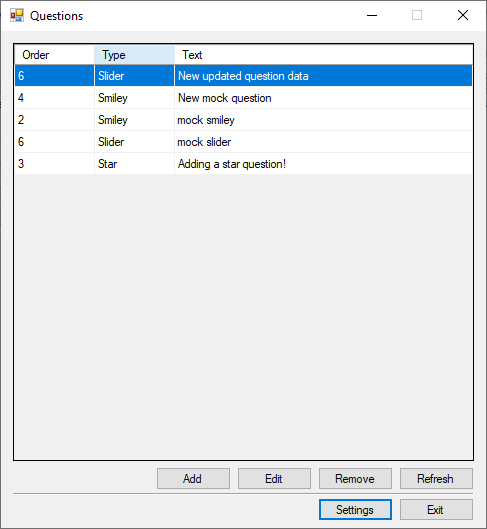


**Step 5:**

After changing the settings and testing the connection successfully, press the Save button to save the new connection settings in the application, now from now on the application will launch with those new settings.



**Note:** after saving the new connection settings the window will close and return to the main window, then fresh data will comeback using the new connection settings that were saved in the settings window.



**Developer documentation**:

The questions application aims to make creating/editing/deleting questions and importing them to other applications or using them anywhere done easily.

This documentation will walk through what design decisions were made and go through each layer and explain what each layer does.

The main architecture of the application is a three layered application; any other layers are support layers that help the application out:

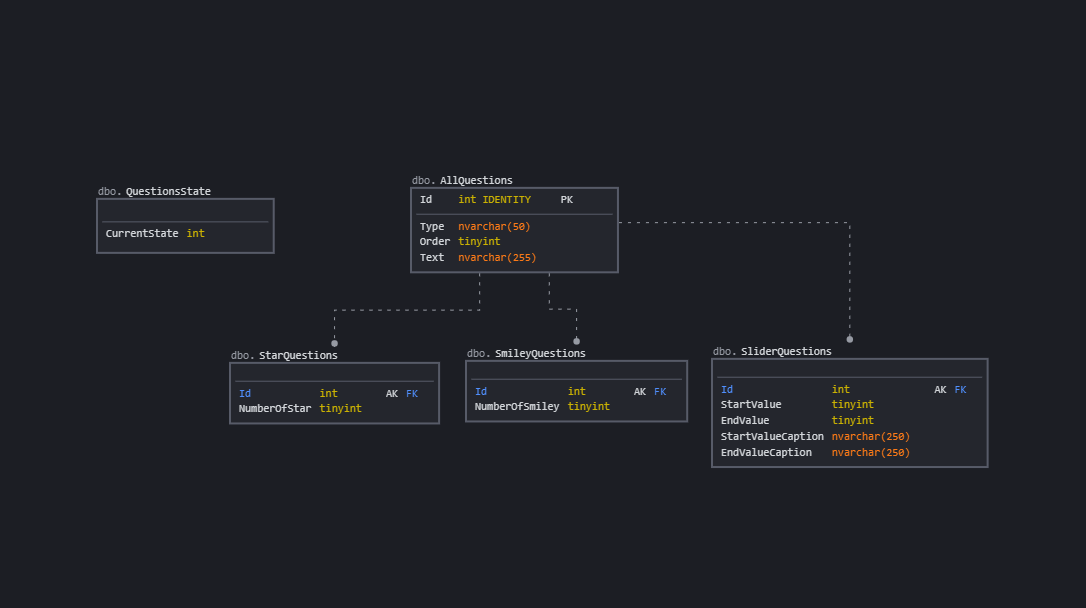
* **The UI Layer:** this is built with windows forms using C#, consisting of one main form and two sub forms and a utility class:
  + **LandingForm:** The main driver behind the application and shows initially on page load and lets the user:
    - Remove questions
    - Refresh the data
    - Sort questions
    - Open the settings form
    - Open the add/edit form
  + **QuestionForm:** The subform that allows the users to:
    - create new questions.
    - edit existing questions.
  + **SettingsForm:** The subform that allows users/admins to update different settings in the application such as:
    - Language.
    - Connection settings.

And lets the user test new connection settings.

* + **MessagesUtility:** This class is a utility class that helps out the application in different ways such as:
    - Getting resources from the resources files.
    - Showing popup messages with correct text corresponding to the result code.
    - Getting the correct message to show in popup messages.
* **The Business Layer:** this is built using C#, which consists of a QuestionsController, and the responsibilities of this layer include:
  + **In memory data:** it contains a List<Questions> that can be accessed from the UI, this List is constructed from data from the database.
  + **Adding questions.**
  + **Editing questions.**
  + **Removing questions.**
  + **Refreshing data:** This layer runs a background thread for updating the List<Questions> every ten seconds.
  + **Notifying Data changed:** This layer also has an eventHandler which fires every time the List<Questions> gets updated.
* **The Data Layer:** This is built using C# and SQL, and is mainly responsible for connecting to the database and querying it
  + **DatabaseController:** This is the main driver class for connecting to the database and querying it.
  + **ConnectionString:** This is a model class for the connection string the database needs, this also has some helpful utility functions that help in getting the correct connection string and saving the connection string to the .config file.
* **The Model Layer:** This is built using C#, this contains the different question classes and some utility classes and enums and can be found in QuestionEntities
  + **Question:** This is the parent class of the questions subclasses/subtypes and it includes the shared props of the questions, it also has helper functions that help with assigning data and getting the properties of the class as well as validating the data.
  + **SmileyQuestion:** This is a subclass of Question and it includes the extra data the Smiley question type needs, this also overrides the helper function of class Question and implements them.
  + **SliderQuestion:** This is a subclass of Question and it includes the extra data the Slider question type needs, this also overrides the helper function of class Question and implements them.
  + **StarQuestion:** This is a subclass of Question and it includes the extra data the Star question type needs, this also overrides the helper function of class Question and implements them.
  + **QuestionsFactory**: This is a Factory class that helps instantiate instances of Questions in runtime.
  + **QuestionUtilites:** This class contains different enums for response codes and for database error codes, this also implements a function that maps the database error code to the main response codes enum.
* **The Logging Layer:** This is a utility layer that handles the logging of the exceptions that are thrown anywhere in the application to a log file, it implements a single static class and that class has a static function that logs the errors to a .txt file with the current date and the exception information.

The three layered architecture was used because it separates the different components of the application and creates a high level of abstraction, so whenever something needs to change in any of the layers, only that layer will be changed and no other layers will be affected.

**Database Diagram:**



The database consists of Five tables:

* AllQuestions: This is the main table for the data, it has an Id primary key, Type, Order, Text, so basically the main Question attributes.
* SmileyQuestions: This is a sub type of Questions, it only has an Id and the extra data this sub type has, the Id is a FK (Foreign Key) for the allQuestions table, since the main data of any SmileyQuestions existing currently in this table are in the AllQuestions table.
* SliderQuestions: This is also another sub type of Questions, and the same behavior that the SmileyQuestions has also applies here.
* StarQuestions: This is also another sub type of Questions, and the same behavior that the SmileyQuestions has also applies here.
* QuestionsState: This table only consists of an Int and is used to determine the current state of the data in the other tables, so It’s basically a table that the developer can access and see if the current application data is up to data with the data in the database or not.

The database also consists of different Procedures that do different kind of operations such as:

* **[dbo].Add\_StarQuestion:** Adds a new question of type Star, **input parameters:**
  + **@Text** nvarchar(250): the text of the new question.
  + **@Order** INT: the order of the new question.
  + **@NumberOfStar** INT: the number of the stars the new question has.

**output parameters:**

* + **@Id** INT: the id of newly created question.

CREATE OR ALTER PROCEDURE [dbo].Add\_StarQuestions

(@Text nvarchar(250), @Order INT, @NumberOfStar INT, @Id INT = NULL OUTPUT)

AS

BEGIN

BEGIN TRY

BEGIN TRAN

SET XACT\_ABORT ON;

-- Insert the new question in the main table

INSERT INTO AllQuestions VALUES('Star', @Order, @Text);

-- Gets the newly created question Id

SET @Id = SCOPE\_IDENTITY();

-- Inserts the new question into the StarQuestions table

INSERT INTO StarQuestions VALUES(@Id, @NumberOfStar);

-- Call the update\_CurrentState procedure to notify for changes

EXEC [dbo].Update\_CurrentState;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN

THROW;

END CATCH

END

GO

* **[dbo].Add\_SliderQuestions:** Adds a new question of type Slider, **input parameters:**
  + **@Text** nvarchar(250): the text of the new question.
  + **@Order** INT: the order of the new question.
  + **@StartValue** INT: the start value of the question.
  + **@EndValue** INT: the end value of the question.
  + **@StartValueCaption** nvarchar(250): the start value caption of the question.
  + **@EndValueCaption** nvarchar(250): the end value caption of the question.

**output parameters:**

* + **@Id** INT: the id of newly created question.

CREATE OR ALTER PROCEDURE [dbo].Add\_SliderQuestions

( @Text nvarchar(250), @Order INT, @StartValue INT, @EndValue INT, @StartValueCaption nvarchar(250), @EndValueCaption nvarchar(250), @Id INT = NULL OUTPUT)

AS

BEGIN

BEGIN TRY

BEGIN TRAN

SET XACT\_ABORT ON;

-- Insert the new question in the main table

INSERT INTO AllQuestions VALUES('Slider', @Order, @Text);

-- Gets the newly created question Id

SET @Id = SCOPE\_IDENTITY();

-- Inserts the new question into the SliderQuestions table

INSERT INTO SliderQuestions VALUES(@Id, @StartValue, @EndValue, @StartValueCaption, @EndValueCaption);

-- Call the update\_CurrentState procedure to notify for changes

EXEC [dbo].Update\_CurrentState;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

* **[dbo].Add\_SmileyQuestions**: Adds a new question of type Smiley, **input parameters:**
  + **@Text** nvarchar(250): the text of the new question
  + **@Order** INT: the order of the new question
  + **@NumberOfSmiley** INT: the number of the stars the new question has.

**output parameters:**

* + **@Id** INT: the id of newly created question.

CREATE OR ALTER PROCEDURE [dbo].Add\_SmileyQuestions

( @Text nvarchar(255), @Order INT, @NumberOfSmiley INT, @Id INT = NULL OUTPUT)

AS

BEGIN

BEGIN TRY

BEGIN TRAN

SET XACT\_ABORT ON;

-- Insert the new question in the main table

INSERT INTO AllQuestions VALUES('Smiley', @Order, @Text);

-- Gets the newly created question Id

SET @Id = SCOPE\_IDENTITY();

-- Inserts the new question into the SliderQuestions table

INSERT INTO SmileyQuestions VALUES(@Id, @NumberOfSmiley);

-- Call the update\_CurrentState procedure to notify for changes

EXEC [dbo].Update\_CurrentState;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

* **[dbo].Update\_StarQuestions:** Updates an existing question of type Star with new data,

**input parameters:**

* + **@Text** nvarchar(250): the text of the question
  + **@Order** INT: the order of the question
  + **@NumberOfSmiley** INT: the number of the stars the question has.
  + **@Id** INT: the id of the question to update.

CREATE OR ALTER PROCEDURE [dbo].Update\_StarQuestions

( @Text nvarchar(255), @Order INT, @NumberOfStar INT, @Id INT )

AS

BEGIN

BEGIN TRY

BEGIN TRAN

SET XACT\_ABORT ON;

-- Set the totlaRows variable to track number of rows changed

DECLARE @totalRows INT;

SET @totalRows = 0;

-- Update the question in the main table with new data

UPDATE AllQuestions SET Text = @Text, [Order] = @Order WHERE Id = @Id AND Type = 'Star';

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- Updates the question in the StarQuestions with new data

UPDATE StarQuestions SET NumberOfStar = @NumberOfStar WHERE Id = @Id;

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- If affected rows = 2, update the currentState with new values sine the

-- question got updated successfuly

if (@totalRows = 2)

EXEC [dbo].Update\_CurrentState;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

* **[dbo].Update\_SliderQuestions:** Updates an existing question of type Star with new data,

**input parameters:**

* + **@Text** nvarchar(250): the text of the question.
  + **@Order** INT: the order of the question.
  + **@StartValue** INT: the start value of the question.
  + **@EndValue** INT: the end value of the question.
  + **@StartValueCaption** nvarchar(250): the start value caption of the question.
  + **@EndValueCaption** nvarchar(250): the end value caption of the question.
  + **@Id** INT: the id of the question to update.

CREATE OR ALTER PROCEDURE [dbo].Update\_SliderQuestions

( @Text nvarchar(250), @Order INT, @StartValue INT, @EndValue INT, @StartValueCaption nvarchar(250), @EndValueCaption nvarchar(250), @Id INT )

AS

BEGIN

BEGIN TRY

BEGIN TRAN

SET XACT\_ABORT ON;

-- Set the totlaRows variable to track number of rows changed

DECLARE @totalRows INT;

SET @totalRows = 0;

-- Update the question in the main table with new data

UPDATE AllQuestions SET Text = @Text, [Order] = @Order WHERE Id = @Id AND Type = 'Slider';

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- Updates the question in the SliderQuestions with new data

UPDATE SliderQuestions SET StartValue = @StartValue, EndValue = @EndValue, StartValueCaption = @StartValueCaption, EndValueCaption = @EndValueCaption WHERE Id = @Id;

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- If affected rows = 2, update the currentState with new values sine the

-- question got updated successfuly

if (@totalRows = 2)

EXEC [dbo].Update\_CurrentState;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

* **[dbo].Update\_SmileyQuestions:** Updates an existing question of type Star with new data,

**input parameters:**

* + **@Text** nvarchar(250): the text of the question
  + **@Order** INT: the order of the question
  + **@NumberOfSmiley** INT: the number of the stars the question has.
  + **@Id** INT: the id of the question to update.

CREATE OR ALTER PROCEDURE [dbo].Update\_SmileyQuestions

( @Text nvarchar(255), @Order INT, @NumberOfSmiley INT, @Id INT )

AS

BEGIN

BEGIN TRY

BEGIN TRAN

SET XACT\_ABORT ON;

-- Set the totlaRows variable to track number of rows changed

DECLARE @totalRows INT;

SET @totalRows = 0;

-- Update the question in the main table with new data

UPDATE AllQuestions SET Text = @Text, [Order] = @Order WHERE Id = @Id AND Type = 'Smiley';

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- Updates the question in the SmileyQuestions with new data

UPDATE SmileyQuestions SET NumberOfSmiley = @NumberOfSmiley WHERE Id = @Id;

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- If affected rows = 2, update the currentState with new values sine the

-- question got updated successfuly

if (@totalRows = 2)

EXEC [dbo].Update\_CurrentState;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

* **[dbo].Delete\_StarQuestions:** Deletes an existing question of type Star with new data,

**input parameters:**

* + **@Id** INT: the id of the question to be deleted.

CREATE OR ALTER PROCEDURE [dbo].Delete\_StarQuestions

( @Id INT )

AS

BEGIN

BEGIN TRY

BEGIN TRAN

SET XACT\_ABORT ON;

-- Set the totlaRows variable to track number of rows changed

DECLARE @totalRows INT;

SET @totalRows = 0;

-- Deletes a question from the StarQuestions table

DELETE FROM StarQuestions WHERE Id = @Id;

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- Deletes a question from the main questions table

DELETE FROM AllQuestions WHERE Id = @Id AND Type = 'Star';

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- If affected rows = 2, update the currentState with new values sine the

-- question got updated successfuly

if (@totalRows = 2)

EXEC [dbo].Update\_CurrentState;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

* **[dbo].Delete\_SliderQuestion:** Deletes an existing question of type Slider with new data,

**input parameters:**

* + **@Id** INT: the id of the question to be deleted.

CREATE OR ALTER PROCEDURE [dbo].Delete\_SliderQuestions

( @Id INT )

AS

BEGIN

BEGIN TRY

BEGIN TRAN

SET XACT\_ABORT ON;

-- Set the totlaRows variable to track number of rows changed

DECLARE @totalRows INT;

SET @totalRows = 0;

-- Deletes a question from the SliderQuestions table

DELETE FROM SliderQuestions WHERE Id = @Id;

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- Deletes a question from the main questions table

DELETE FROM AllQuestions WHERE Id = @Id AND Type = 'Slider';

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- If affected rows = 2, update the currentState with new values sine the

-- question got updated successfuly

if (@totalRows = 2)

EXEC [dbo].Update\_CurrentState;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

* **[dbo].Delete\_SmileyQuestions:** Deletes an existing question of type Smiley with new data,

**input parameters:**

* + **@Id** INT: the id of the question to be deleted.

CREATE OR ALTER PROCEDURE [dbo].Delete\_SmileyQuestions

( @Id INT )

AS

BEGIN

BEGIN TRY

BEGIN TRAN

SET XACT\_ABORT ON;

-- Set the totlaRows variable to track number of rows changed

DECLARE @totalRows INT;

SET @totalRows = 0;

-- Deletes a question from the SmileyQuestions table

DELETE FROM SmileyQuestions WHERE Id = @Id;

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- Deletes a question from the main questions table

DELETE FROM AllQuestions WHERE Id = @Id AND Type = 'Smiley';

-- Add the number of rows affected

SET @totalRows = @totalRows + @@ROWCOUNT

-- If affected rows = 2, update the currentState with new values sine the

-- question got updated successfuly

if (@totalRows = 2)

EXEC [dbo].Update\_CurrentState;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

* **[dbo].Get\_SliderQuestion:** Gets question data based on the Id of the question,

**input parameters:**

* + **@Id** INT: the id of the question to get.

CREATE OR ALTER PROCEDURE [dbo].Get\_SliderQuestions

(@Id INT)

AS

BEGIN

SELECT AllQuestions.Type, AllQuestions.[Order], AllQuestions.Text, SliderQuestions.StartValue, SliderQuestions.EndValue, SliderQuestions.StartValueCaption, SliderQuestions.EndValueCaption

FROM AllQuestions

INNER JOIN SliderQuestions ON AllQuestions.Id = SliderQuestions.Id

WHERE AllQuestions.Id = @Id;

END

GO

* **[dbo].Get\_SmileyQuestion:** Gets question data based on the Id of the question,

**input parameters:**

* + **@Id** INT: the id of the question to get.

CREATE OR ALTER PROCEDURE [dbo].Get\_SmileyQuestions

(@Id INT)

AS

BEGIN

SELECT AllQuestions.Type, AllQuestions.[Order], AllQuestions.Text, SmileyQuestions.NumberOfSmiley

FROM AllQuestions

INNER JOIN SmileyQuestions ON AllQuestions.Id = SmileyQuestions.Id

WHERE AllQuestions.Id = @Id;

END

GO

* **[dbo].Get\_StarQuestion:** Gets question data based on the Id of the question,

**input parameters:**

* + **@Id** INT: the id of the question to get.

CREATE OR ALTER PROCEDURE [dbo].Get\_StarQuestions

(@Id INT)

AS

BEGIN

SELECT AllQuestions.Type, AllQuestions.[Order], AllQuestions.Text, StarQuestions.NumberOfStar

FROM AllQuestions

INNER JOIN StarQuestions ON AllQuestions.Id = StarQuestions.Id

WHERE AllQuestions.Id = @Id;

END

GO

* **[dbo].Update\_CurrentState:** Updates the currentState of the database data in the QuestionsState table.

**Procedure code:**

CREATE OR ALTER PROCEDURE [dbo].Update\_CurrentState

AS

BEGIN

BEGIN TRY

BEGIN TRAN

UPDATE QuestionsState SET [CurrentState] = [CurrentState] + 1;

DECLARE @QuestionsStateCount INT;

SET @QuestionsStateCount = (SELECT MAX([CurrentState]) FROM QuestionsState);

IF (@QuestionsStateCount = 2147483647)

UPDATE QuestionsState SET [CurrentState] = 0;

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN

THROW;

END CATCH

END

GO

The project also includes a runnable SQL script that you should run before running the application, which creates the database, and creates all tables and the procedures, it is named SQLScript.sql.