Survey Question Configurator

Technical Report

1. Introduction

Survey question configurator is a desktop application developed using the .NET framework with SQL Server database to manage survey questions by adding, editing, and deleting them from the database, all application requirements are available in the question configurator requirements document.

In this report, we will discuss the major architecture and design decisions.

2. Architecture Overview

The design uses 4-layer architecture in addition to essential question classes.

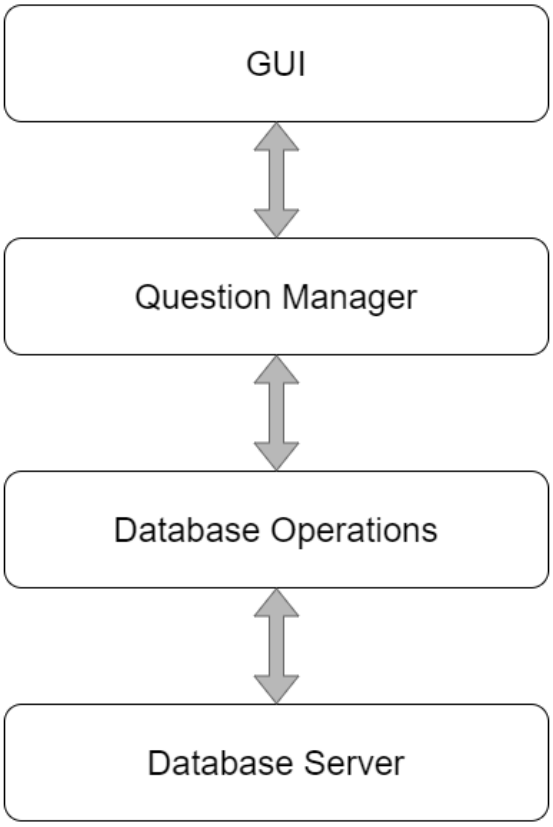
a) **Question classes:** these classes are used to create different question types and hold their properties.

b) **Database Server:** SQL Server database management system to store data in a relational database.

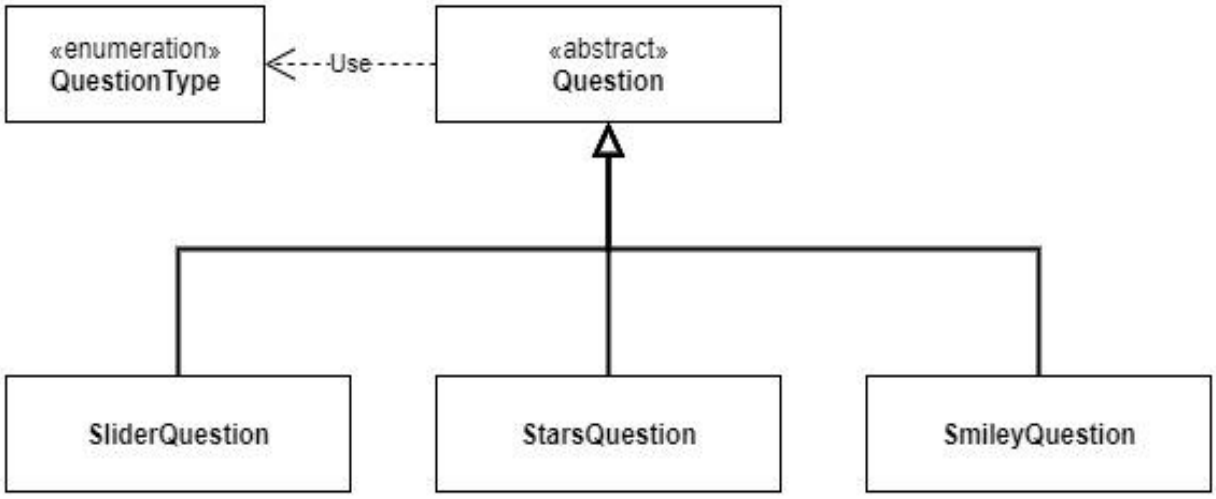
c) **Database Operations:** data access layer that mange's the connection and interaction with the database server using ADO .NET framework.

d) **Question Manager:** the business logic layer also serves as a separation layer between the GUI and Data access layer.

e) **GUI:** windows forms to view questions and provide users with different interaction and customization capabilities.



3. Question Classes



a) **QuestionType (Enum)** : an enumeration type to represent all supported question types in the system.

b) **Question (Abstract class)** : an abstract class to encapsulate common properties of all question types such as text, order and type.

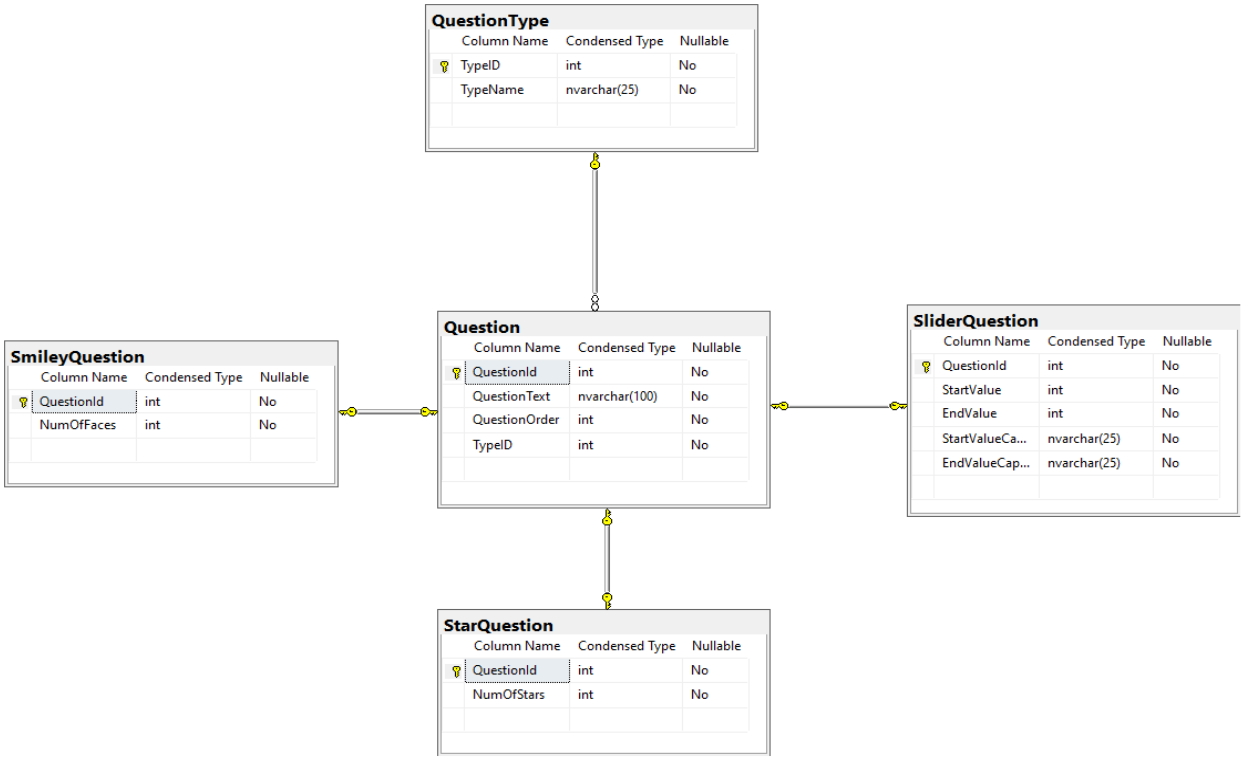
c) **SliderQuestion (Class)** : a class that inherits Question class properties and provide additional SliderQuestion properties which are start value, end value, start value caption and end value caption.

d) **StarsQuestion (Class)** : a class that inherits Question class properties and provide additional StarsQuestion property which is number of stars.

e) **SmileyQuestion (Class)** : a class that inherits Question class properties and provide additional SmileyQuestion property which is number of smiley faces.

4. Database Server

Using SQL Server, We have written a script to create the database that contains tables presented in ER diagram below, We have used generalization to represent common question properties in one table (Question) with other question types referencing it using foreign key.



.

a) **QuestionType** : store question types.

b) **Question** : store common questions properties.

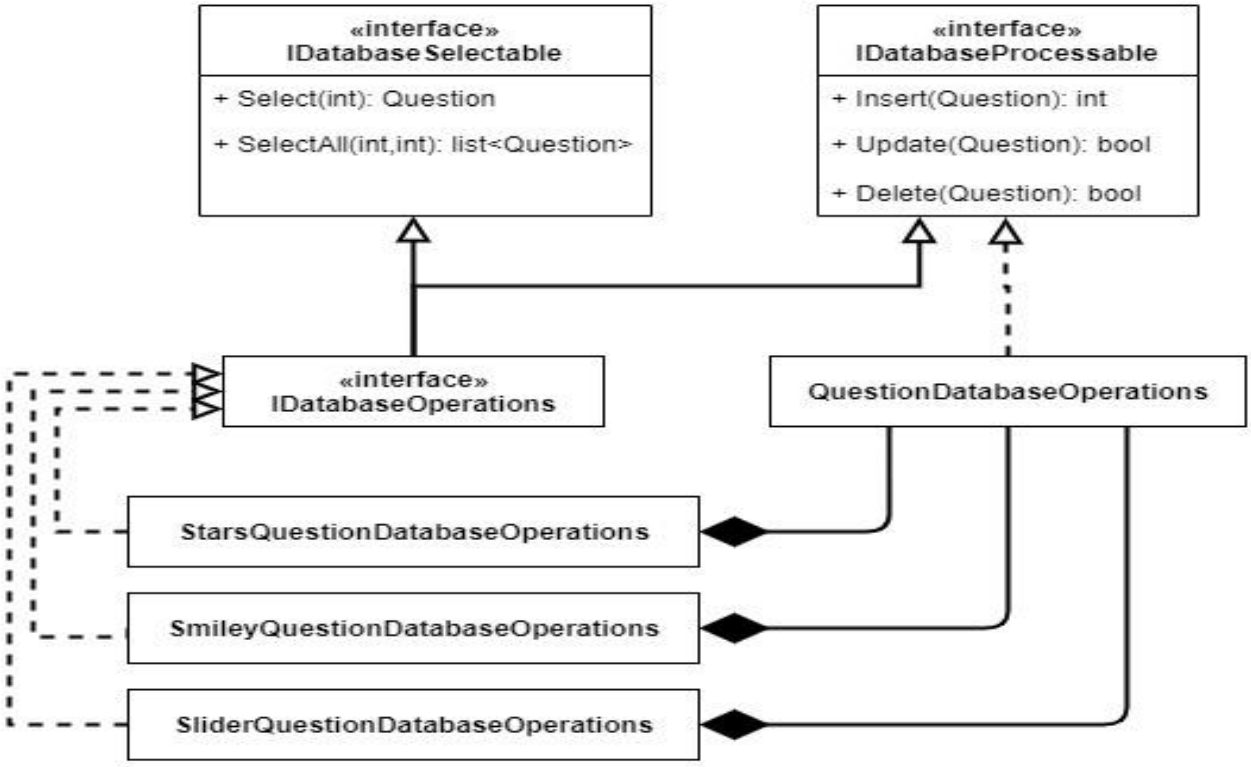
c) **SliderQuestion** : store slider questions properties with a reference to general question.

d) **StarsQuestion** : store stars questions properties with a reference to general question.

e) **SmileyQuestion** : store smiley questions properties with a reference to general question.

5. Database Operations

Data access layer that mange's the connection and interaction with the database server using ADO .NET framework.



a) **IDatabaseSelectable (Interface)** : an interface that provides Select and SelectAll behaviors to the implementing class.

b) **IDatabaseProcessable (Interface)** : an interface to provide Insert and Update and Delete behaviors to the implementing class.

c) **IDatabaseOperations (Interface)** : an interface that inherits both IDatabaseSelectable and IDatabaseProcessable to provide both of its behaviors to the implementing class.

d) **QuestionDatabaseOperations (Class)** : a class that implements IDatabaseProcessable interface only, because database contains Question table that we need to insert, update and delete it's content, but it's not logical to select it's content individually since Question class is abstract and no question can be created without a type.

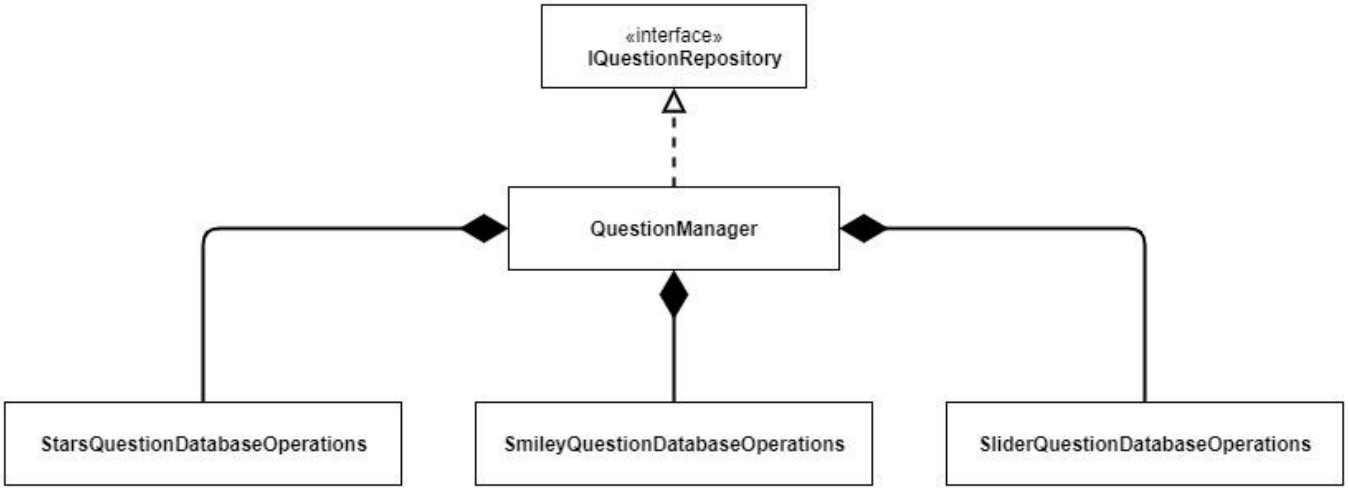
e) **StarsQuestionDatabaseOperations (Class)** : a class that implements IDatabaseOperations interface to insert , update, delete and select Stars questions from database, It has an instance of QuestionDatabaseOperations class as composition to interact with Question table.

f) **SmileyQuestionDatabaseOperations (Class)** : a class that implements IDatabaseOperations interface to insert , update, delete and select Smiley questions from database, It has an instance of QuestionDatabaseOperations class as composition to interact with Question table.

g) **SliderQuestionDatabaseOperations (Class)** : a class that implements IDatabaseOperations interface to insert , update, delete and select Slider questions from database, It has an instance of QuestionDatabaseOperations class as composition to interact with Question table.

6. Question Manager

Business logic layer that separates DatabaseOperations layer (Data access layer) logic from GUI (Presentation layer), to reduce connecting and retrieving data from the database, this layer also provides a local list that holds a copy of the data to interact with, when connecting to the database is not necessary, for example, if we want to insert question to the database we also insert a copy to our local list, so when we need to retrieve this question we retrieve the copy in our local list instead of the database.



a) **IQuestionRepository (Interface)** : an interface that provides Repository behavior to the implementing class.

b) **QuestionManager (Class)** : a class that implements IQuestionRepository interface to insert, update, delete, select, maintain a local list of questions and get the latest copy of questions from the database, it has an instance of StarsQuestionDatabaseOperations, SmileyQuestionDatabaseOperations, and SliderQuestionDatabaseOperations classes as a composition to interact with the database.

7. GUI

GUI is the presentation layer which the end-user interacts with, its developed using .NET forms and designed to match standard widows and office application, currently, the application interface supports two language which are English and Arabic .

a) **Main form** : the initial form that opens when the user runs the application, we have tried to make it as simple as possible for the user to understand and interact with.

The question list is implemented using DataGridView with 3 columns which are question text, question order, and question type, it supports selection using mouse or keyboard, sorting the list according to a specific column by clicking on its header, open and edit form by double-clicking on specific question row.

There are three action buttons for the user to use Add, Edit, and delete, Add button will open an empty question form, the edit button will open a form with selected question data populated in it, and the delete button will show a confirmation box to delete the selected question, also there are two customization controls Refresh button and Language ComboBox, refresh button will retrieve the latest copy of data from the database and language combo box will change the interface according to the selected language.

b) **Properties Form** : used when the user is adding a new question or editing an existing question, it is usually opened as a form dialog invoked by the main form when the user clicks on add or edit buttons.

When the user is adding a new question, the forms open with default values using a constructor that only takes QuestionManager reference, it consists of two parts, the first one is the common question properties such as question text and order and type, it's always visible, the other part changes according to question type selection, initially only one question type properties GroupBox is shown and the others are hidden, so when the user changes question type selection, the selected type group box will be shown and the others are hidden.

When the user is editing a question, a constructor with two parameters is called, the first parameter is QuestionManager and the second one is Question reference, so when the form is loaded the given question properties are loaded to the form and the user will edit them, but in this case, the question type ComboBox is disabled to prevent a user from changing question type.

The form has two buttons, Save and Cancel, Save will add the new question in case the form was invoked using the add button, and it will change given question properties in case of the edit form, the cancel button will discard all changes and close the opened properties form.

8. Error Logger

Error logger is a static class that is used by all layers to log errors and messages into the log file using a specific format, the class has Log method which is overloaded to use either string parameter or Exception object parameter, it formats the given object and logs it into error.log file .

9. Naming Conventions

The following naming conventions were used in the project:

a) **Class** : using pascal case (Example: ClassName)

b) **Method**: using pascal case (Example: MethodName)

c) **Property** : using pascal case (Example: PropertyName)

d) **Public attribute :** using pascal case (Example: PublicAttribute)

e) **Private attribute** : using pascal case with prefix "m" (Example: mPrivateAttribute)

f) **Constants** : using full caps separated with underscore "_" (Example: CONSTANT_VALUE)

g) **Local variable** : using pascal case with prefix "t" (Example: tLocalVariable)

h) **Method parameters** : using camel case (Example: methodParameter)