Survey configurator

Developer manual

Contents

[1.0 About Survey configurator project 3](#_Toc168182287)

[1.1 Overview 3](#_Toc168182288)

[1.2 Architecture 4](#_Toc168182289)

[2.0 Code details 5](#_Toc168182290)

[2.1 Question class 5](#_Toc168182291)

[2.2 User interface 7](#_Toc168182292)

[2.2.1 MainScreen class 7](#_Toc168182293)

[2.2.2 AddEdit class 8](#_Toc168182294)

[2.2.3 ConnectionSettings class 8](#_Toc168182295)

[2.2.4 Custom controls folder 9](#_Toc168182296)

[2.2.5 App.config file 9](#_Toc168182297)

[2.3 Business logic 9](#_Toc168182298)

[2.4 Database communication 9](#_Toc168182299)

[2.5 SharedResources classes 9](#_Toc168182300)

[2.5.1 Question class and its derived classes 9](#_Toc168182301)

[2.5.2 SharedData 9](#_Toc168182302)

[2.5.3 UtilityMethods 10](#_Toc168182303)

[2.5.4 ConnectionString 10](#_Toc168182304)

[2.5.5 OperationResult 10](#_Toc168182305)

[2.5.6 Resources files 11](#_Toc168182306)

[3.0 Making Changes to the code 12](#_Toc168182307)

[3.1 Adding a new Language 12](#_Toc168182308)

[3.2 Adding a new question type 14](#_Toc168182309)

# About Survey configurator project

## 1.1 Overview

-Survey configurator is a desktop application built using C# .NET and Windows forms technologies, and for the database it’s connected to a Microsoft SQL Server database, this application enables the users to create questions to be used in surveys, where each question has a number, text and a type, and each type has further more specific details, the user is able to edit or delete any of the existing questions, also the user can change font size or language of the application or open the manual for more help.

-this application also supports multi-instances or multiple users using it at the same time, such that if a user performs any CRUD operation on the questions in the data base it will be almost immediately reflected to the other users.

## 1.2 Architecture

-This application was developed using the 3-tier architecture methodology, such that the application is divided into three layers:

-Database communication layer: this layer is solely responsible for communicating with the database and almost all of its functions returns a result indicating whether the operation was successful or not, and on the success of the operation the data will be moved to Business logic layer.

-Business logic layer: this layer acts as an interface between the database layer and the User Interface layer at receives the data from the data base applies business logic to it and moves it to the User Interface layer and vice versa, all kinds of logic and validation happens in this layer.

-User Interface layer: this layer is the communication interface between the user and the application.

-additionally this project contains files and classes that contains data and methods shared between the layers.

-this approach was chosen to enhance code structure and for the matter of separation of interests, also it will be easier to change the User interface or the database for such project as the change will only happen to the said layer the other layers won’t require any change most of the time.

-this project follows a specific naming convention that you will notice when reading the code, especially in the variables, class members, and constants and here’re the details for this convention:

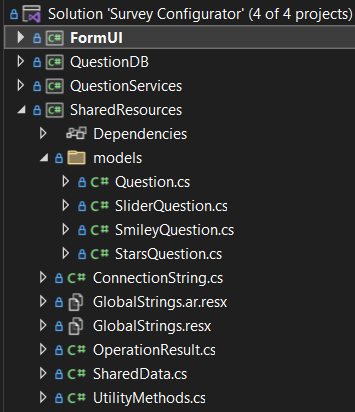
* m: Class member.
* t: Temporary.
* p: Parameter.
* c: Constant.
* e: Enumeration.

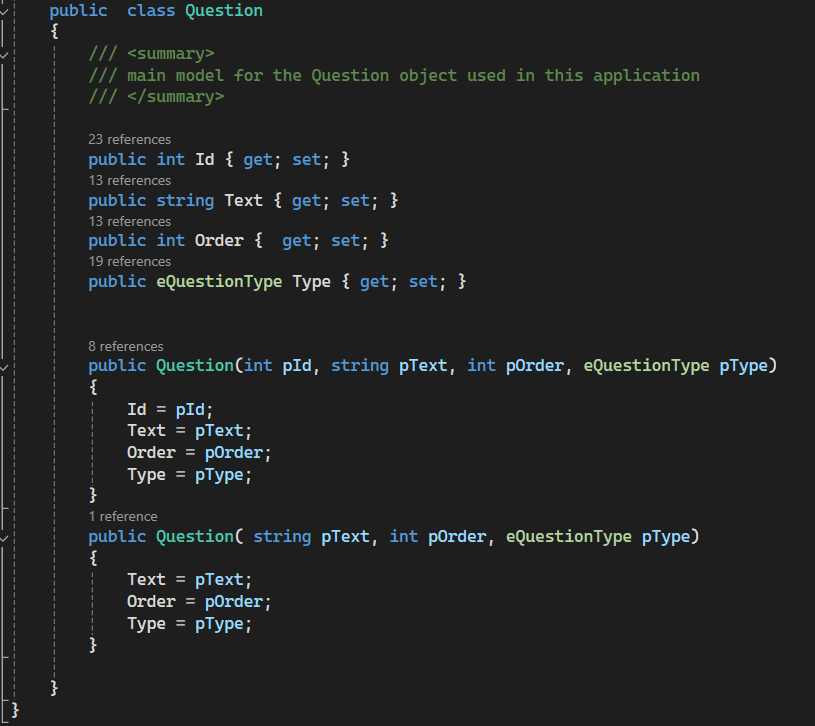
-if you edit the code please be sure to follow these conventions.

# Code details

## 2.1 Question class

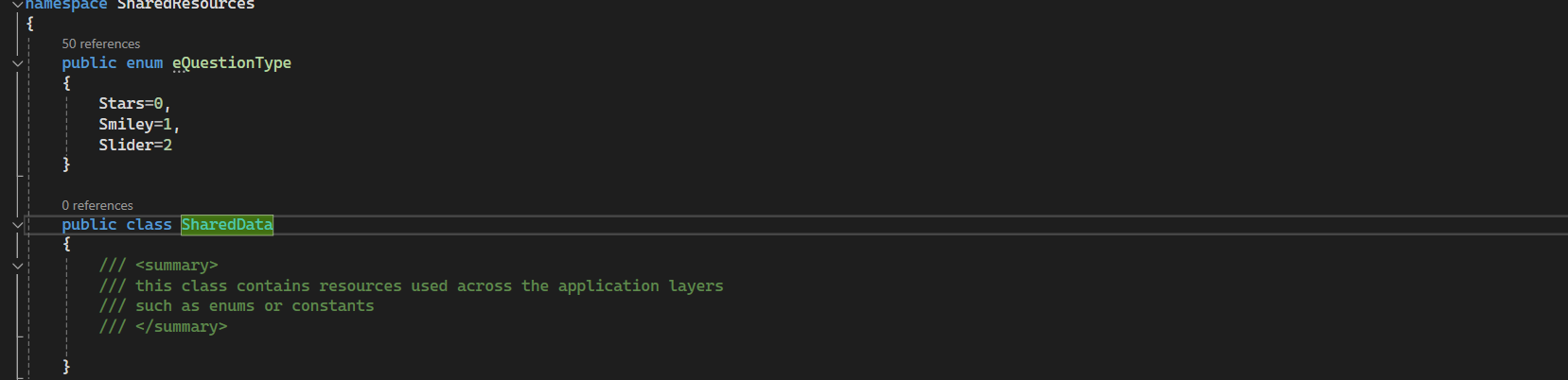
- the main building block of this project is the “Question” class, found in the “SharedResources” project inside the “models” folder:



-this class is the base class for all other types of questions it contains the general information of a questions as shown: 

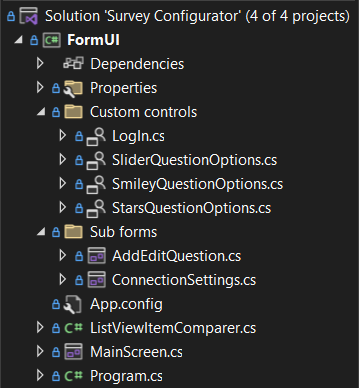
-The reason of having two constructors one with the Id property and the other without is because of the mechanism of generating Ids for the questions, the generation of Ids is left for the database as to make the Ids consistent if the app was connected to multiple User interfaces and used by multiple users at once.

-The question type is of type “eQuestionType” an enumeration type used across the project to avoid any mistakes while typing the type name of any question in addition to making the values similar to how the questions type are stored in the data base, this will be discussed in later sections, the question type enum can be found in the SharedResources namespace in the SharedData class:



## 2.2 User interface

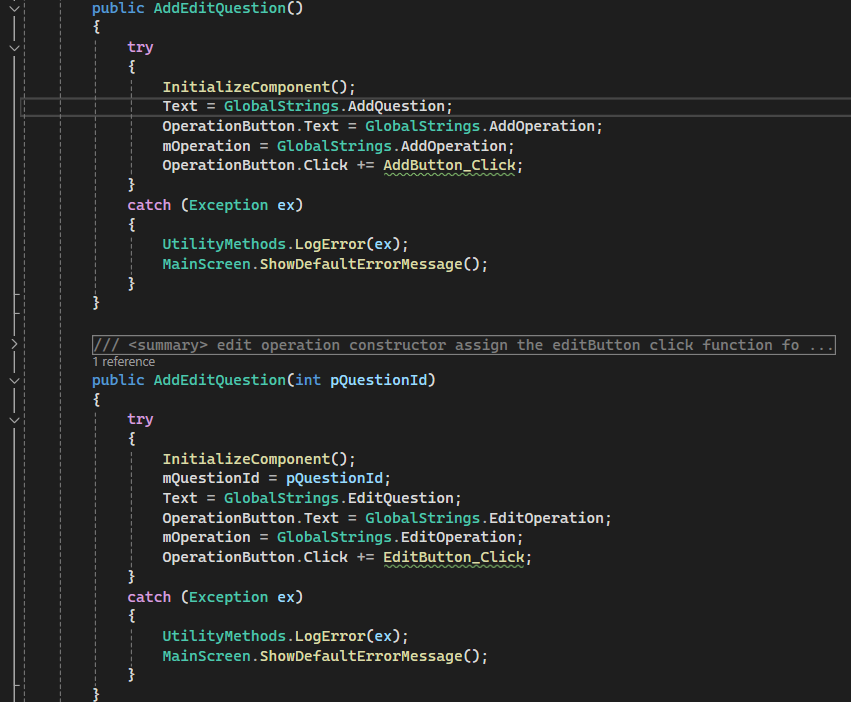
-for the User interface Microsoft Windows forms was used, this layer contains multiple components:



### 2.2.1 MainScreen class

-The “MainScreen” class, as the name suggests this class generates the form that appears to the user first thing when the application is opened, it contains most of the functionality, everything is commented so you won’t find a difficulty understanding the purpose of most of the code parts.

### 2.2.2 AddEdit class

-The “AddEditQuestion” class, this class shows the Add/Edit form, this class has two constructors one for the add operation, the other is for the edit operation where it receives the Id of the question to fetch its information:  


-Also each constructor assigns a click event handler differently based on which constructor was used.

-When the user changes the type of the question using the QuestionTypeComboBox control a new control will be generated based on the chosen type to display more fields depending on the chosen type more details on this in the following sections.

### 2.2.3 ConnectionSettings class

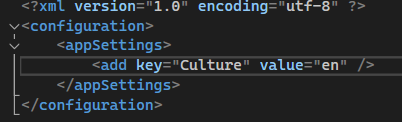
-This class is responsible for showing a window for the user to set the options of connection to the database, it also tests the connection before starting the application.

### 2.2.4 Custom controls folder

-This folder contains User-control (custom controls) for the various forms in this project more on this will be discussed in the following sections.

### 2.2.5 App.config file

-this file contains various settings for application, currently it only contains the settings for the language of the application, any kind of settings can be saved here:



## 2.3 Business logic

-this layer contains only one class responsible for handling the logic and validation processes, the “QuestionOperations” class, this class contains multiple methods to coordinate the communication between the UI layer and the database layer, and as you would notice most of its methods return an object of type “OperationResult” this type and its purpose will be discussed in the following sections.

## 2.4 Database communication

-this is the layer that performs the CRUD operations on the data in the database in addition to any kind of data required from the database like the checksum of a table, which is used for the purpose of checking whether the database has changed, you’ll notice that most of the functions of this class as well return an object of type “OperationResult”.

## 2.5 SharedResources classes

-As mentioned before these classes contains data or methods shared between the 3 layers:

### 2.5.1 Question class and its derived classes

- the mentioned before question class and its derived classes, when adding a new question type a new class for it must be added here, this will be discussed in the following sections.

### 2.5.2 SharedData

-this class contains the enumeration type for the Questions types.

### 2.5.3 UtilityMethods

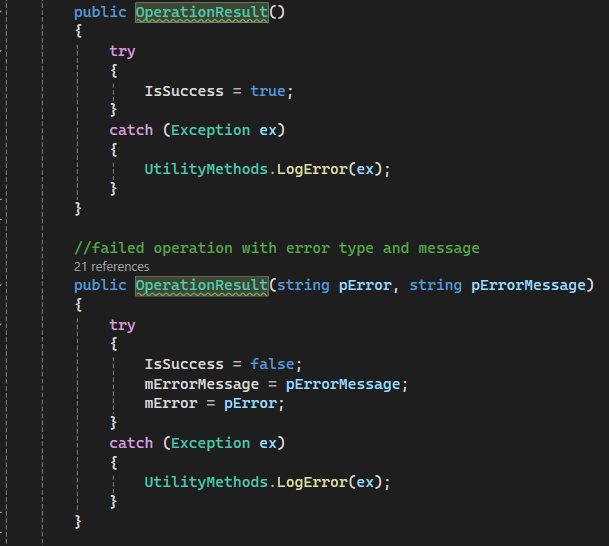
-this class contains methods used across all layers such as the LogError method inside it, which is used to log any exception in the errorlog.txt file inside the errorlogs directory in the projects directory.

### 2.5.4 ConnectionString

-This class is used to create connection string objects and create a connection strings out of them which is used to connect to the database.

### 2.5.5 OperationResult

-this class is used to implement the Operation result-based approach; such that specific operations will be executed based on the result of another one for example: adding a new question will add that question to the User interface only if the it was added successfully to the database, otherwise there will be inconsistencies in the data, so if the operation succeeds a new object will be created with the default constructor and will be returned, otherwise there’s a constructor for failing operations indicating the error type and message:



-So if you’re adding any function that communicates with any source outside of the code itself such as a database or an API where the communication or the requested operation would fail for any reason be sure use this class as a return type for your function, and in case you want to return some data do so using ref or out keywords.

### 2.5.6 Resources files

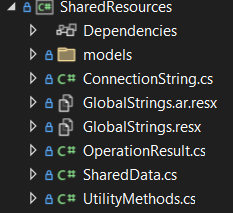
-Resources files which have an extension of .resx such as “GlobalStrings.resx” contains resources such as strings or images that are used across the project, and also used for the purpose of localization; which means enabling the project to support multiple languages.

# Making Changes to the code

## 3.1 Adding a new Language

-To add a new language you have to follow a set of steps:

1. Add a resource file in the “SharedResource” project and name it   
   “GlobalStrings.[language abbreviation here].resx”, where the language abbreviation should be something like “fr” for French or “en” for English:



1. Inside the newly created resource file you will see something like this:



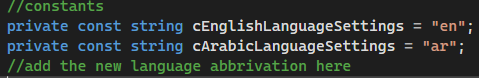
Here the “Name” column refers to the name of variable you will use to access the value stored in the “Value” column, for example in the English resource file you can see the following:

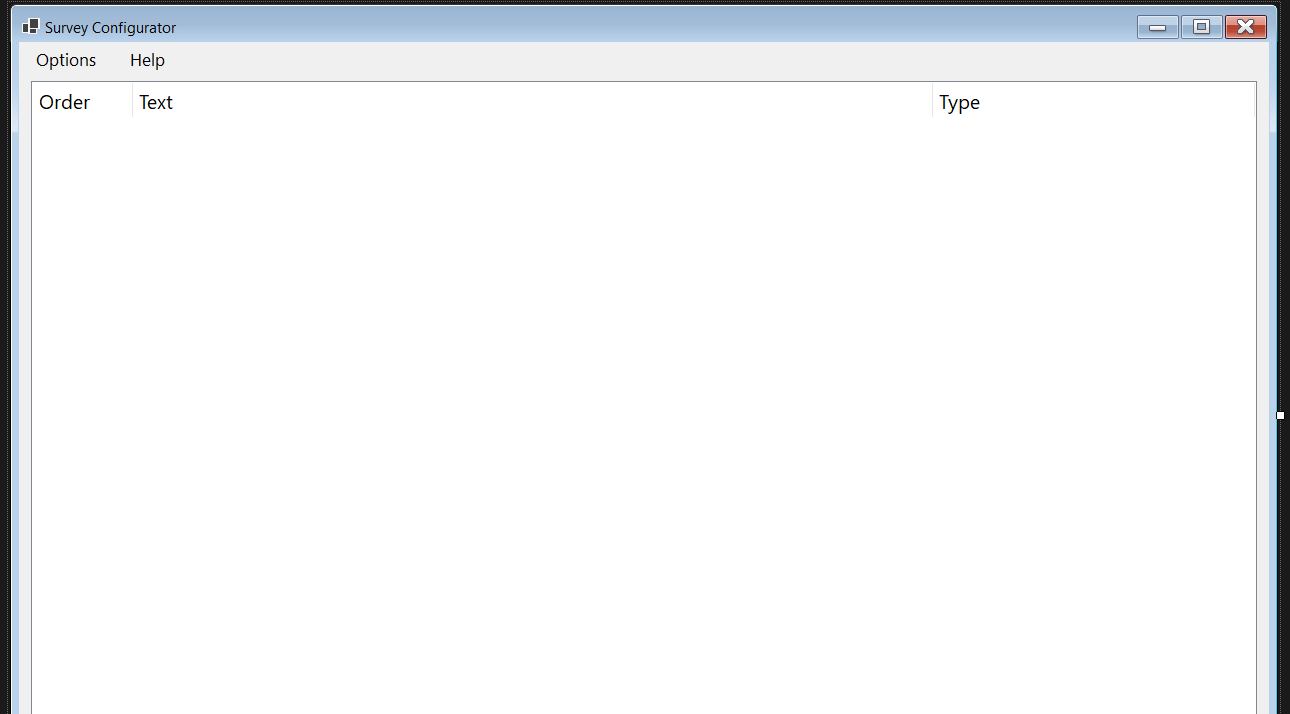
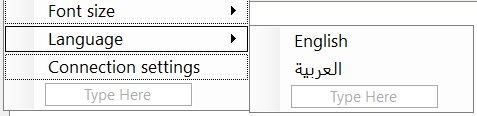


As you can see the “AddOperation” variable contains the “Add” value, for another language you  
 have to use the same variable name “AddOperation” for example with a different value:



Here you can see the same variable name was used with another value, do the same for the rest of the variables, and the correct value will be used according to the used language in the   
 application.

1. Now go to the MainScreen.cs class file in the FormUI project and add the same language abbreviation you used as a constant string:  
   

Lastly press double click on the MainScreen.cs file to see the GUI form   
  
  
  
in the above screen click on the options menu button a menu will appear on the screen, click on the “Language” option and another menu will show up:  
  
now in the second menu that contains the languages choices and in the “Type Here” field indicated in the image above type the name of the language press Enter, then double click the language name choice that got created, you’ll be moved to the MainScreen.cs class file and a new event handler will be created for you and connected to the language choice you created:  
  
your language name will be here instead of “English”, past this code in the method:

try

{

string tAppLanguage = ConfigurationManager.AppSettings[**cLanguageSettingsKey**];

if (tAppLanguage != **cEnglishLanguageSettings**)

{

ChangeAppLanguage(cEnglishLanguageSettings);

Application.Restart();

}

}

catch (Exception ex)

{

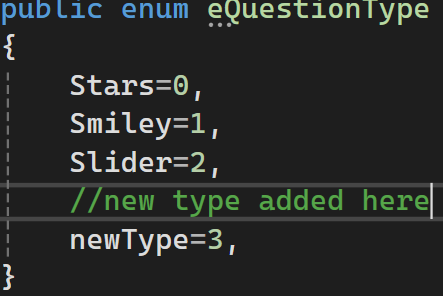
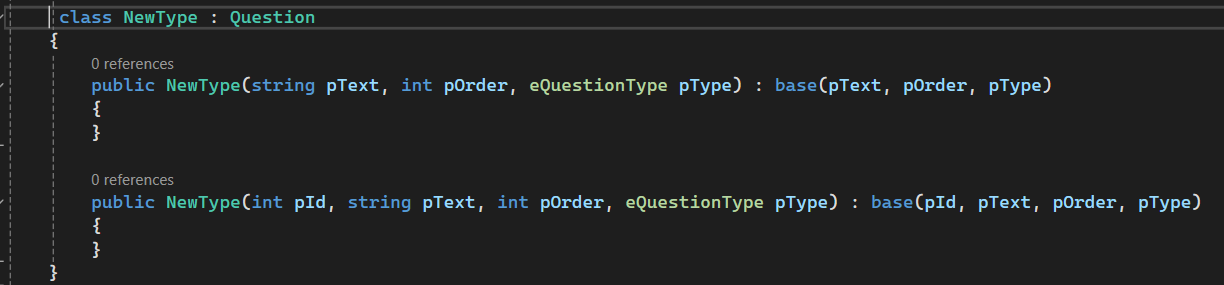
UtilityMethods.LogError(ex);

}  
  
change the underlined and bold “**cEnglishLanguageSettings**” to the language abbreviation constant you created, after that you should be good to go.

## 3.2 Adding a new question type

-adding a new question type requires making changes in multiple places, including running an SQL script

So make sure to follow the steps accordingly:

1. First go the “SharedData” class in the “SharedResources” project and add the new type name to the enum type “QuestionType”, make sure to add it after the other types:  
   
2. Go the “Models” folder inside the “SharedResources” project and add a class for your new type and make sure it implements the “Question” class and that it has two constructors similar to the other derived types:  
   
3. Now go open the “SQL Server Management Studio” on your machine and connect to the server that contains the database that is used for this project, its name should be “Questions\_DB”:  
     
   and then create a new query and enter the following:  
     
   GO

IF OBJECT\_ID('dbo.QuestionType', 'U') IS NOT NULL

BEGIN

IF (NOT EXISTS (SELECT 1 FROM QuestionType))

Begin

INSERT INTO QuestionType ([Id], [Type])

Values

(0, 'Stars'),

(1, 'Smiley'),

(2, 'Slider')

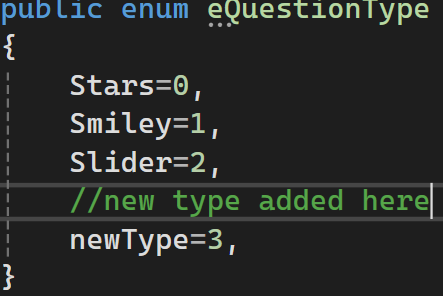
END

END

INSERT INTO QuestionType ([Id], [Type])

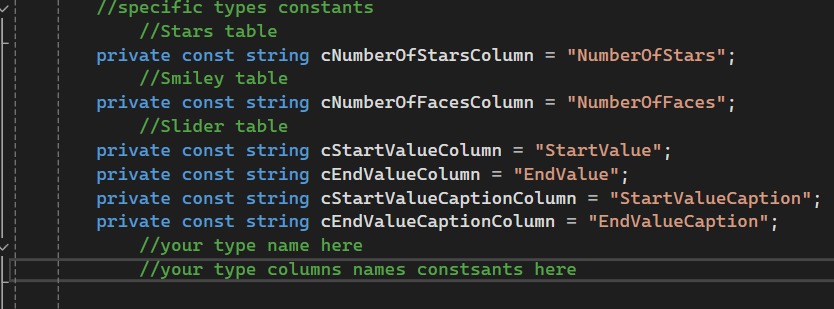
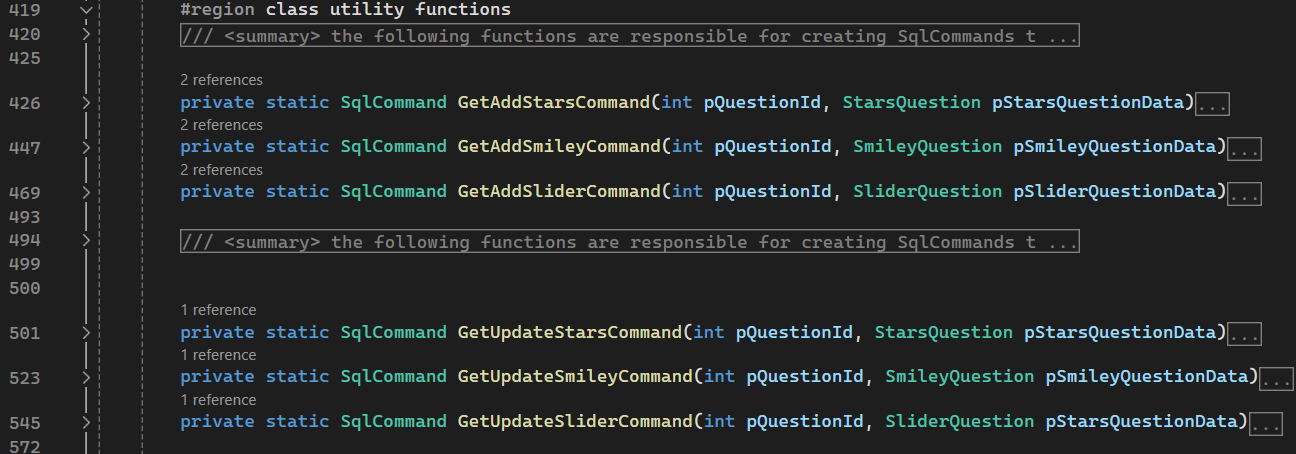
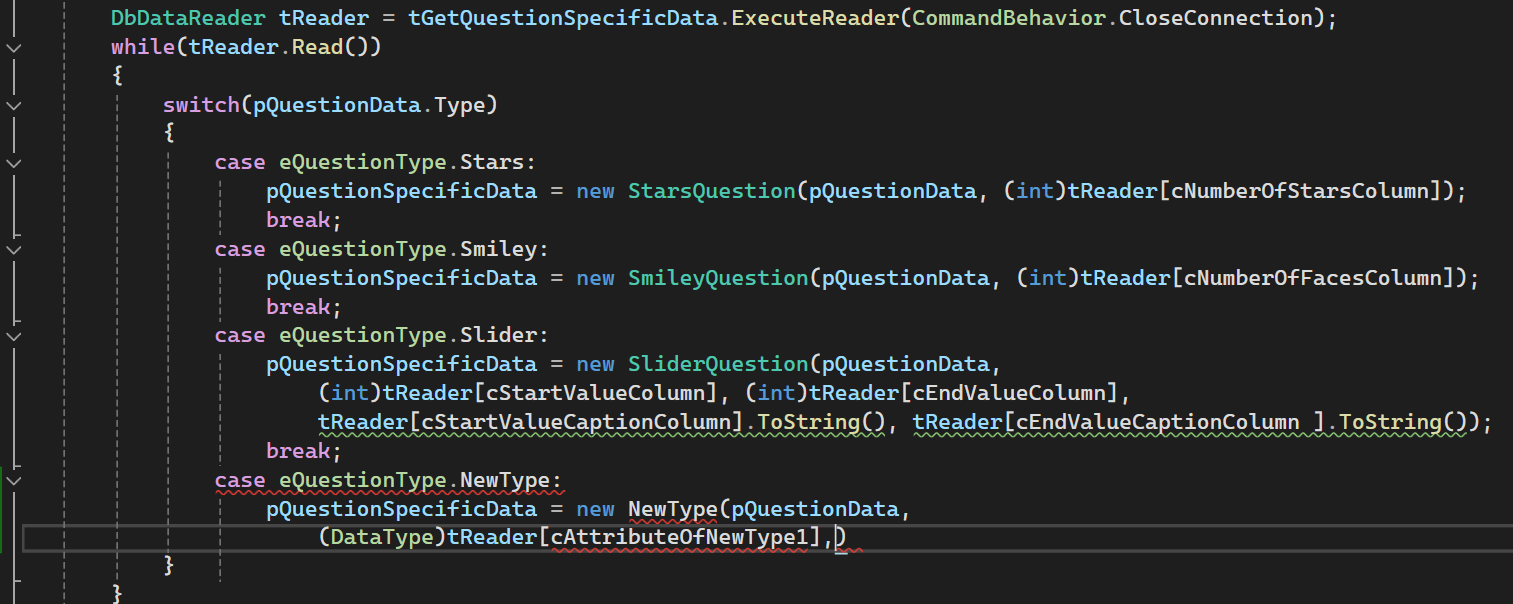
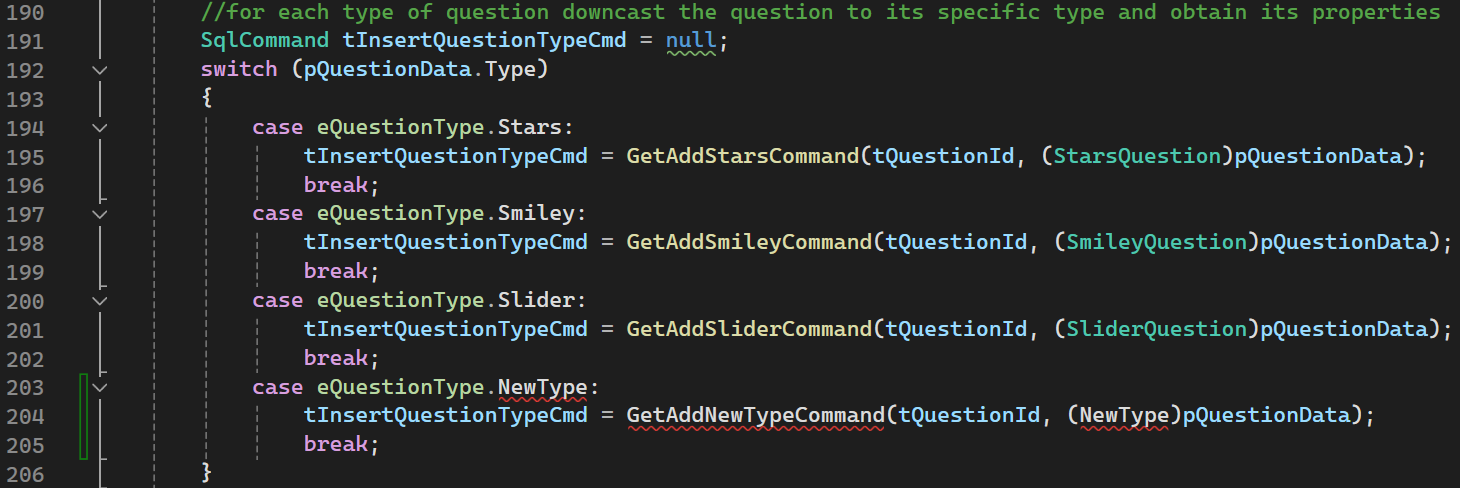
values

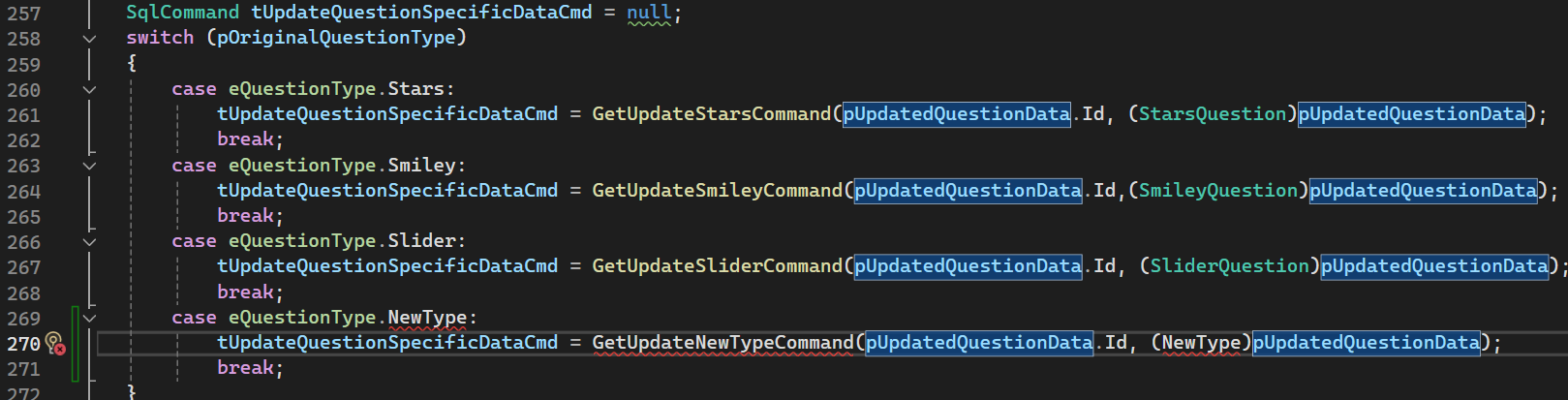
( 3, 'newTypeNameHere');

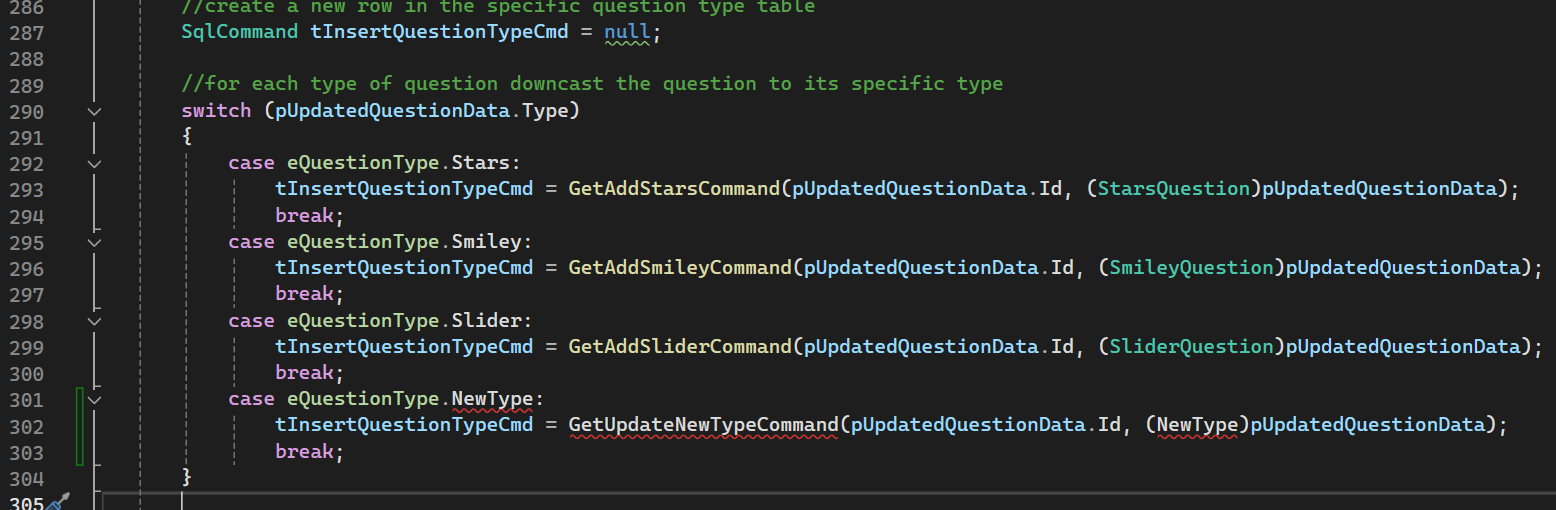
bear in mind that the value 3 in the last line depends on the value that the question type have in the enum: 

the value in the last line of the previous script should match the value you enter here,

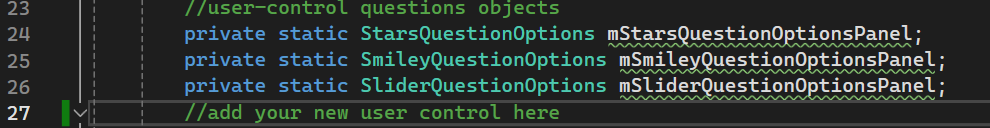
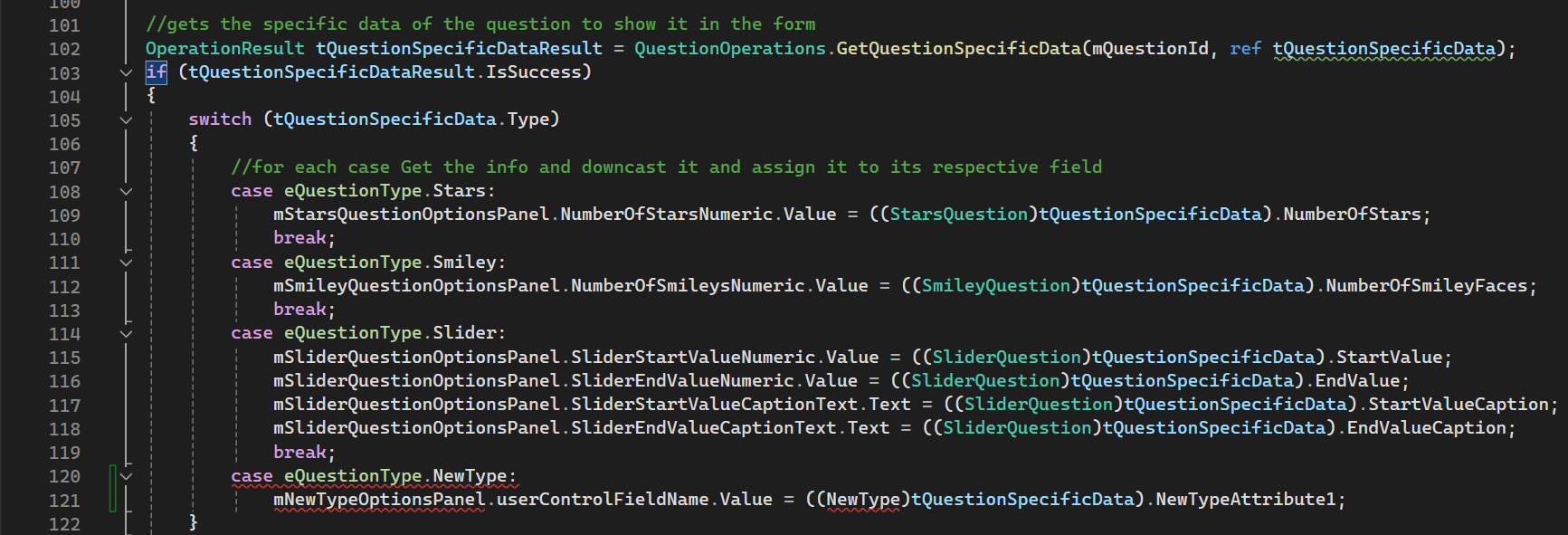
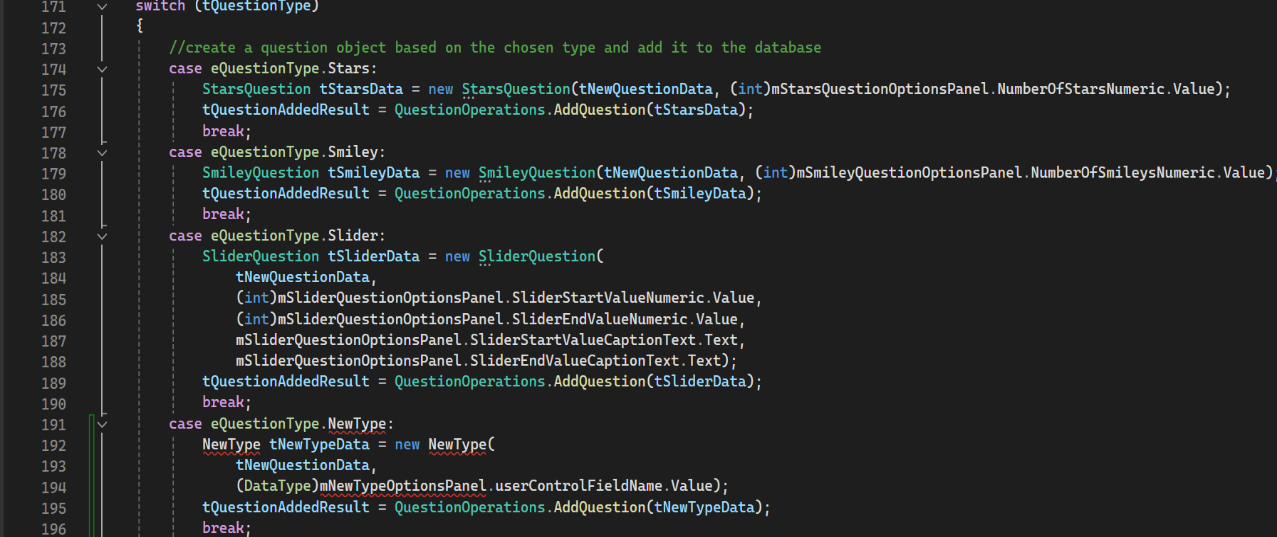
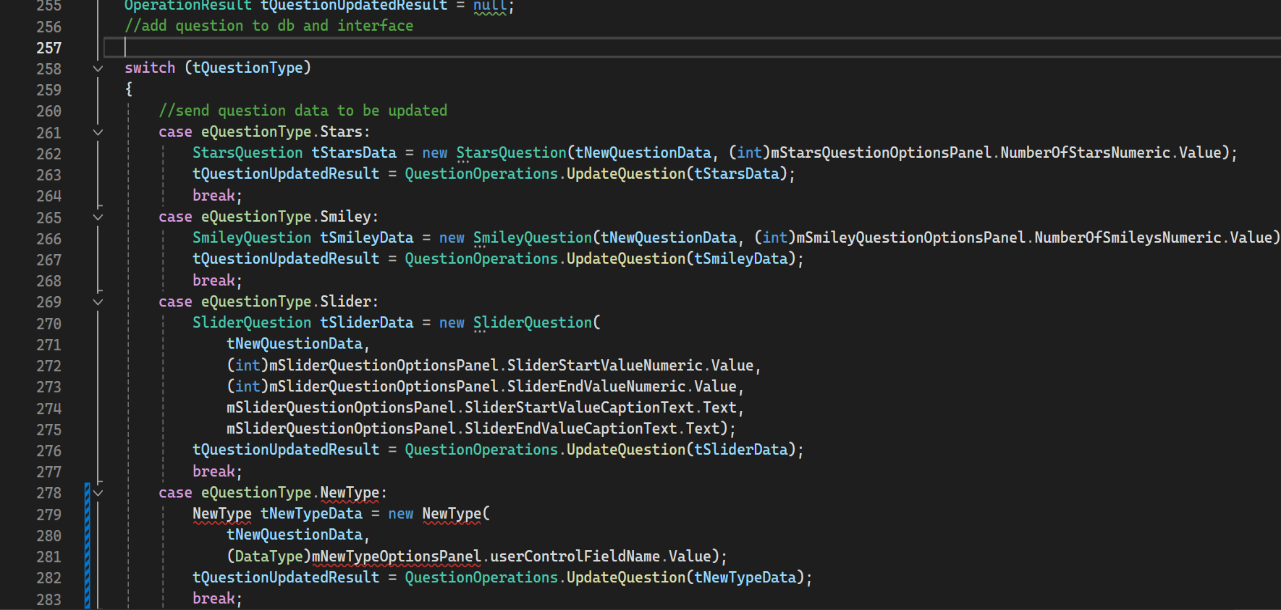
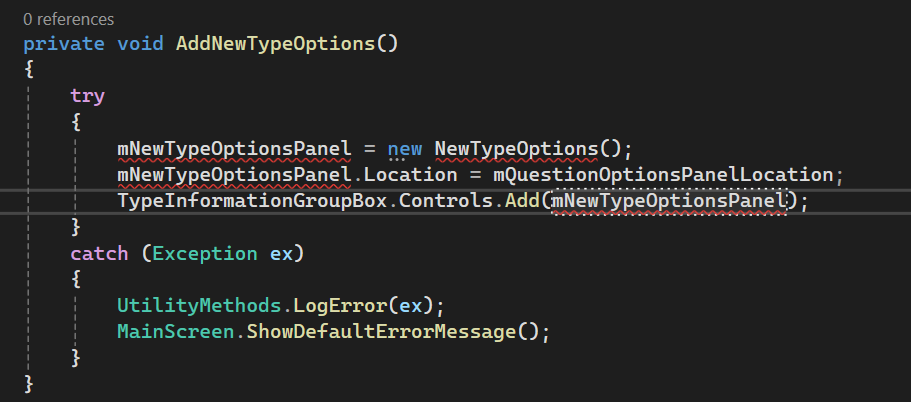
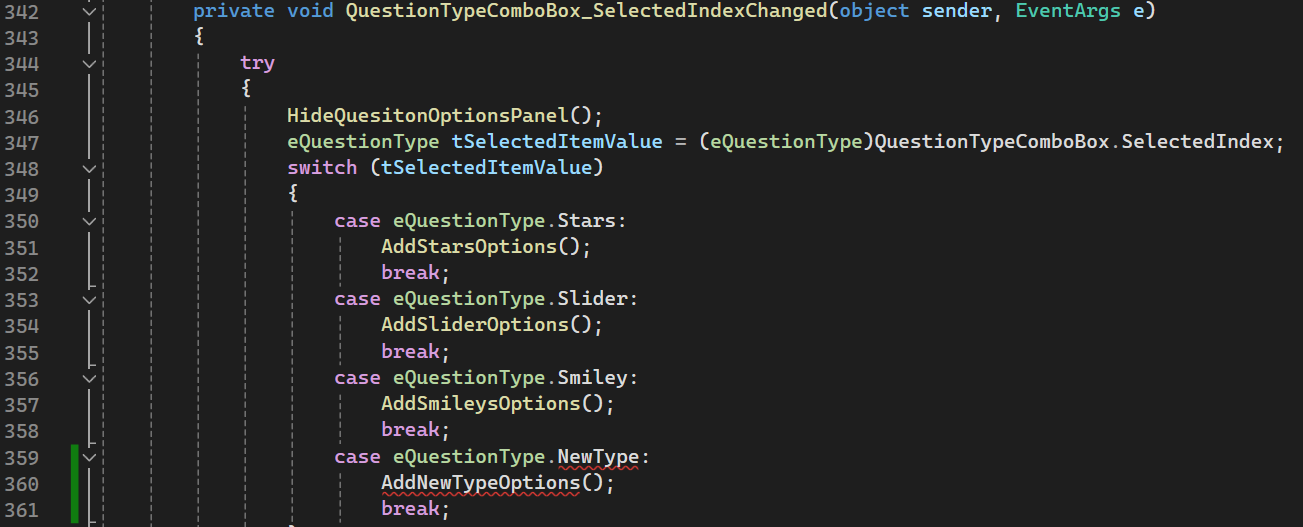
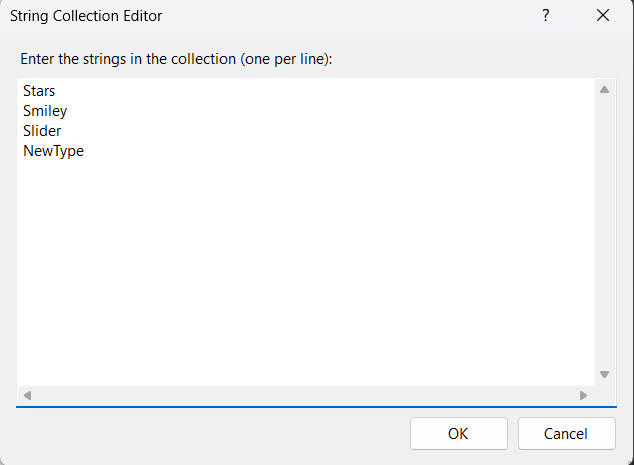
now run the scripts.

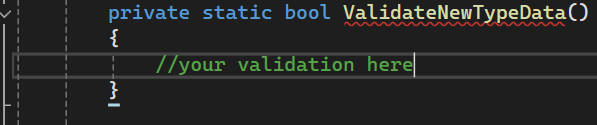
1. Next you’ll have to create a table for the new type you created:  
     
   CREATE TABLE ‘NewTypeNameHere’ (  
    [Id] int FOREIGN KEY REFERENCES Question(Id) NOT NULL,  
    [here add the other columns that your new type will have],  
    );
2. Now back to the project, go to the “Database.cs” class in the “QuestionDB” project and do the following:
   1. Add constant strings representing the names of the columns of the table you just created for your new type:  
      
   2. Now go to the bottom of the file until line 420, here you have to add two functions for your new type:  
        
      -first and add function matching the naming convention of the other add functions, is should look like this:  
        
      where you would replace the “NewType” word for the actual name of the type you created, and the “nAttributeOfNewType1” with the name of your new column for the new type, add as many as the number of columns for the new type, and don’t forget to SQLParameter for each new column you add.  
        
        
        
        
        
        
        
        
        
      -next you have to add an update function for the new type, it’s similar to the add function except for some changes:  
      
   3. Next go to the “GetQuestionSpecificDataFromDB” function in line 109 and in the switch in this function add a case for your type:  
        
      where the “DataType” would be the data type of that value, the “tReader” object reads rows from the database, so you can access the columns value by using it with brackets and put the column name in these brackets as the image above shows, add as many as the columns you have.
   4. Next to the “AddQuestionToDB” function line 160 and similarly in the switch add a case for your type:  
        
      this goes similarly as before replace the “NewType” with the name of your type.
   5. Lastly go to the “UpdateQuestionOnDB” function line 244 and here it’s similar to what you did in the previous step except that you have to do it in two switches:

First here:  
  
Then here:



6.0 Lastly you’ll have to do some changes to the User Interface layer as follows:

* 1. First you need to create a new User-control (custom control), go the “Custom controls” folder in the “FormUI” project and create a new User-control there,  
     it should look like this:  
       
     be mindful that the size of any field should be 386 width, 27 height, and you need to adjust the space between the label and the field as needed in order to be consistent with other fields in the “AddEdit” form.
  2. After you create the user control you need to create a reference for it in the “AddEditQuestion.cs” class file in the “Sub forms” folder, go to that class file and add a class member of the newly created user control type:  
     
  3. Next go to the “AddEdit\_Load” function and then to the switch inside of that function and add a case for your type as follows:  
     
  4. Next go to the “AddButton\_Click” function and then to the switch inside that function and add a case for your type:  
     
  5. Next go to the “EditButton\_Click” function and then to the switch inside that function and add a case for your type:  
     
  6. Next you have to create a function to add the newly created User-control to the “AddEdit” form:  
     
  7. Now you have to edit two places related to the “QuestionTypeComboBox” control:
     1. First, the “QuestionTypeComboBox\_SelectedIndexChanged” function: 
     2. Second, you have to add the type name manually to the “items” of the “QuestionTypeComboBox” control, double click on the “AddEditQuestion.cs”, when the GUI appears right click on the comboBox control on the screen:  
           
        then click on “Properties”, a panel should open now, click on the small icon next to the “items” property:   
          
        a new window will appear containing the names of other types, just add the name of your type at the end of the list:  
        
  8. Lastly, this step depends on whether you have any kind of validation on the new type fields:

-first create a function that contains your validation for the fields of the new type:  
upon violation of the validation it should show the user a message of the error they did with their input.  
  
-next you have to add this function as a case in the switch in the “ValidateInput) function in the same class.

