System Maintenance

System overview

**Brief overall system description:**

My system is a revision/teaching tool for GCSE maths. There are 3 different users, the administrator, teacher and student, each having different privileges which effect how they can use the system.

Admin

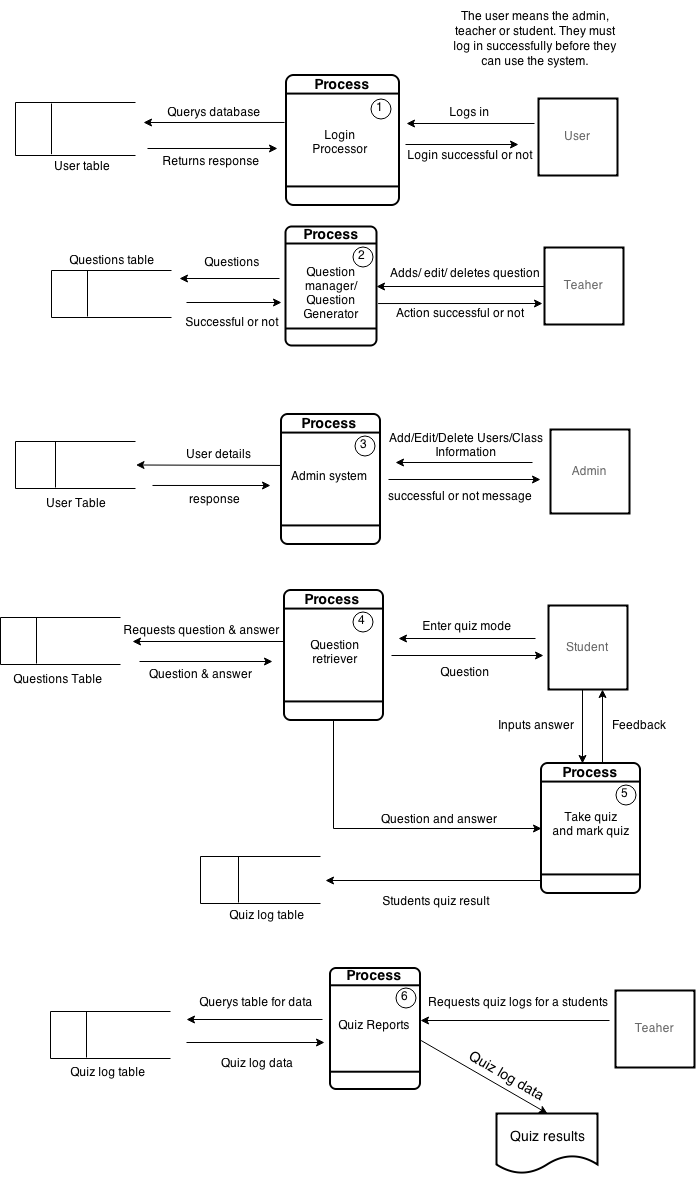
* The role of the administrator would be to manage the user accounts in the system. The administrator create teacher, student and admin accounts, and is able to create new classes in the system. The admin can then assign a teachers and students to a class.

Teacher

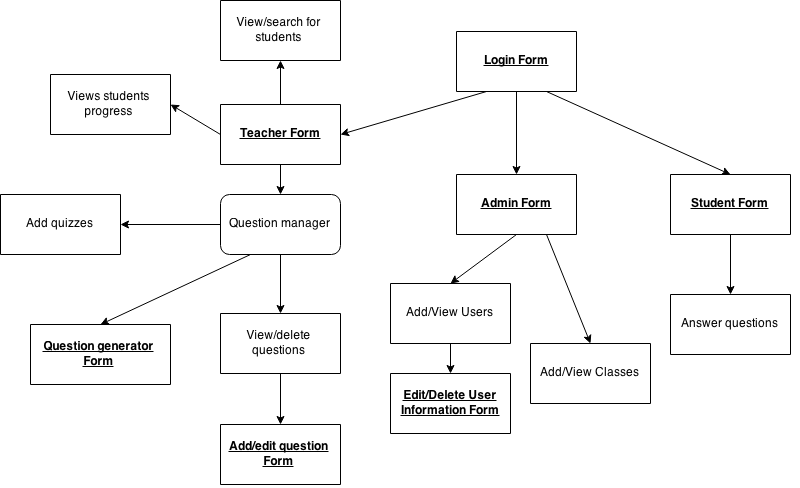
* The main role of the teacher is to manage the learning of their students. One way they can do this is by adding questions for students to answer, either manually using a form or automatically using a question generator feature. To generate questions automatically the teacher would fill in a criteria on a form which generates questions based on the criteria.
* The teacher could also manage questions by being able to view, edit update or delete any of the existing questions in the database. In addition, the teacher could select a set of existing questions and add them to a quiz. Students can then use the system to complete all the quizzes set by their teacher. The teacher could then use a feature to search for the students assigned to their class and view their performance in all the quizzes they have completed.

Student

* The main role of the student is to learn. This is done by them completing any outstanding quizzes set by their teacher. Once a quiz is completed the result is returned to their teacher. If they have no outstanding quizzes to complete they can practise a random set of questions in system, but the result doesn’t get sent to the teacher.

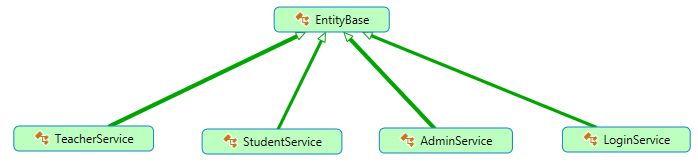
Level 1 Data Flow Diagram:

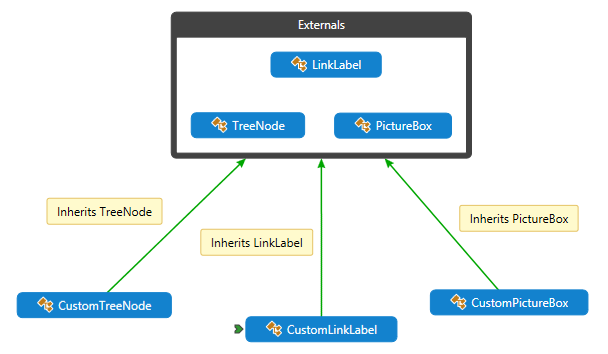
Modular system structure



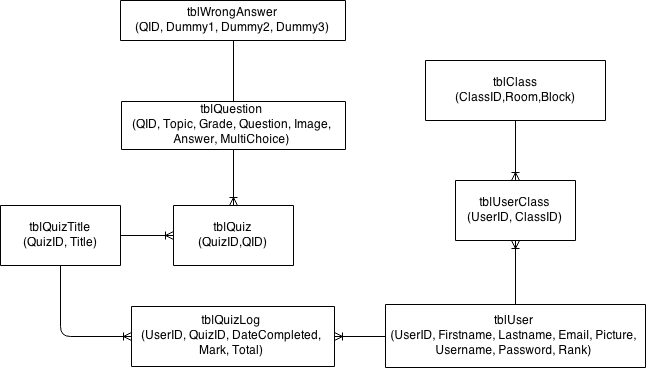
Class overview

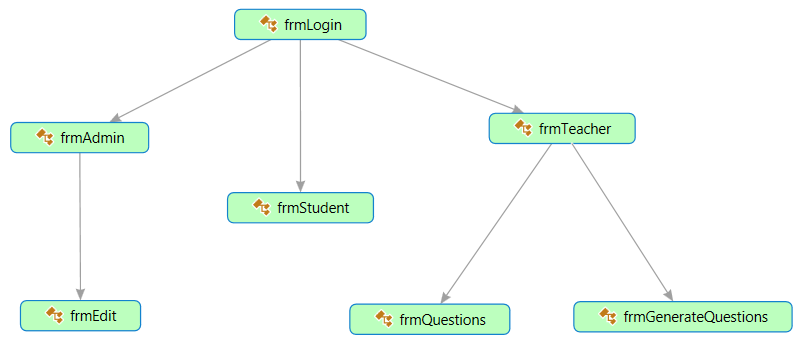
Inheritance

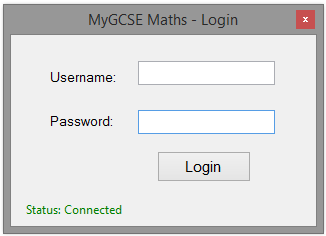
This is an overview of the inheritance in my system for my service classes.

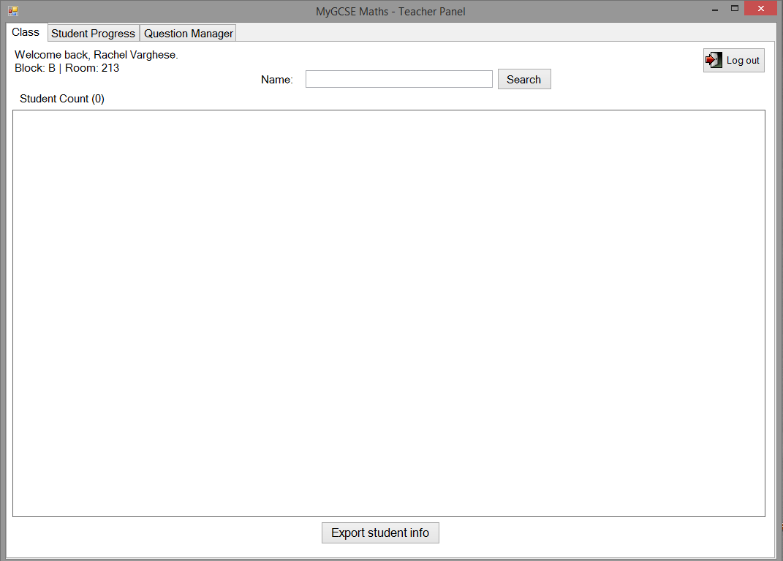
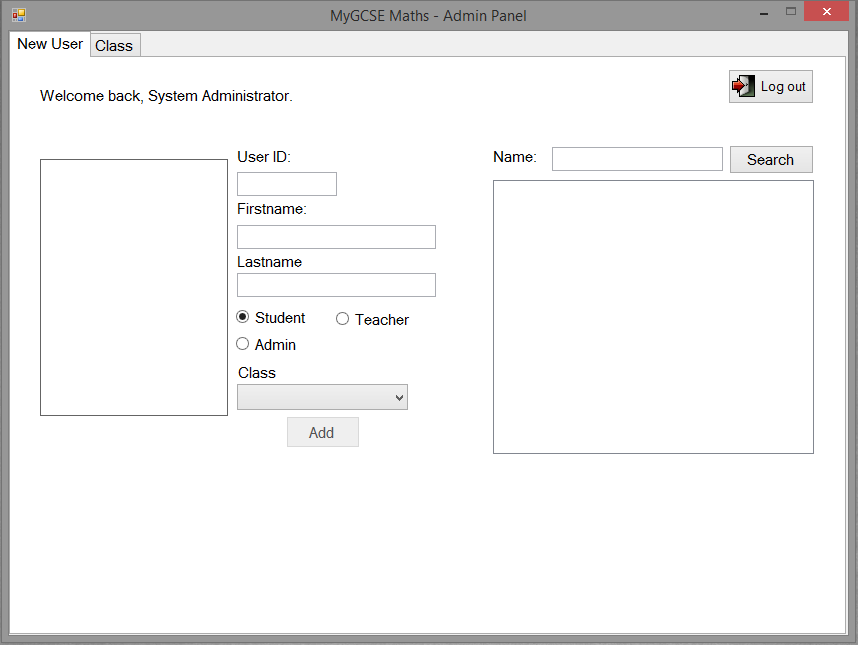
Custom controls and Inheritance – The diagram below illustrates the inheritance of the custom controls I’ve defined. They all inherit former windows UI controls classes and extend their functionality.

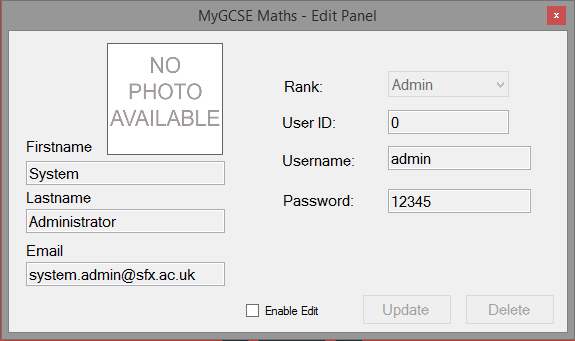
Detailed E-R diagram

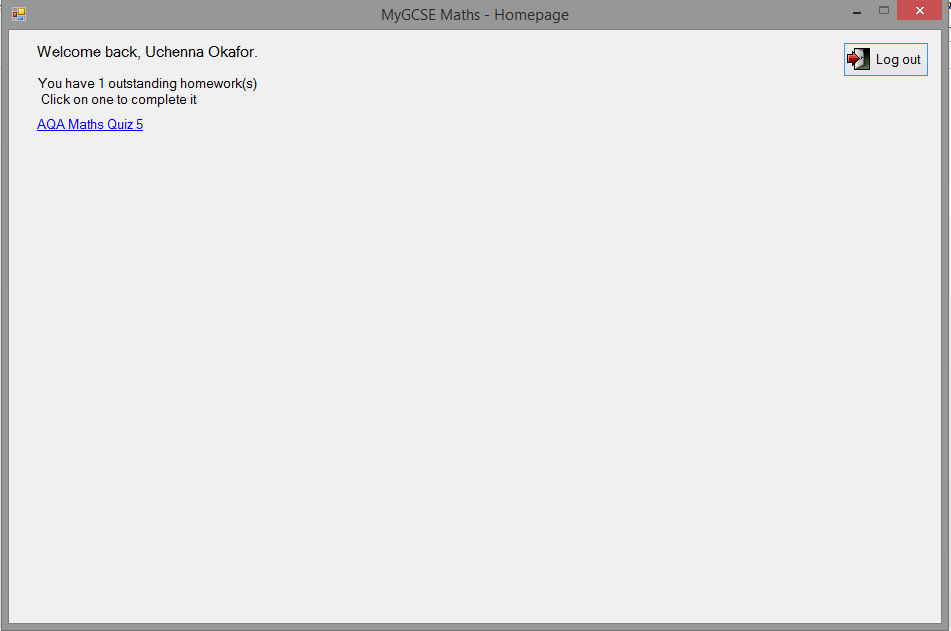
This is a detailed diagram of the entity relationship of the tables in my database.

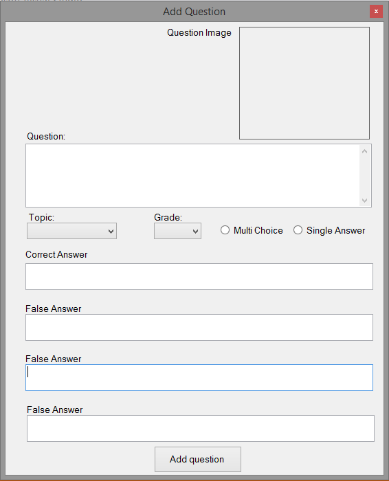
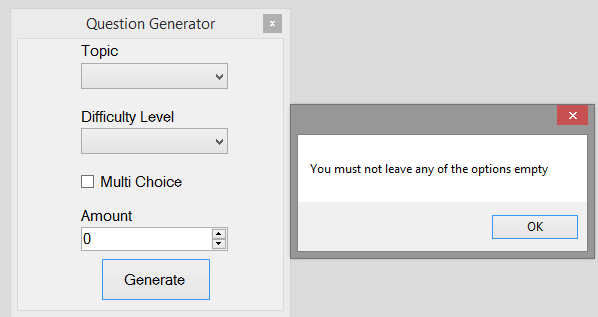
Form overview – This is an overview of how all my forms link together – This image does not show all the different tab pages for each form.

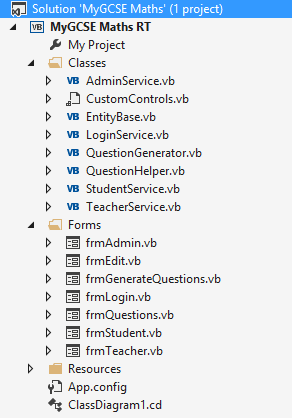
Form navigation:









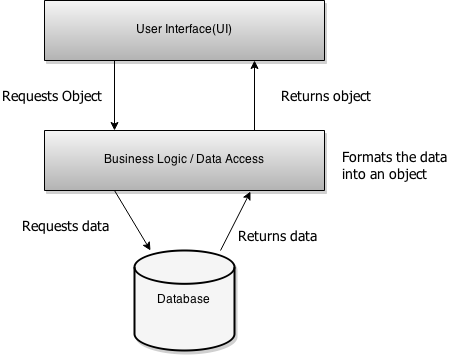
Overall solution files overview

This is an overview of all the files that contain all the code I have written for my project. The classes are grouped into the class folder, and forms are grouped to a forms folder. Each class essentially is paired up with a form. For example, the login form is paired with the LoginService class where the methods in LoginService is only for logging in users into the system and nothing else. Each class aids the UI layer/form to perform specific tasks.

The concept of my classes is to mimic an API where you make a call to the function and it will return the data you need. The caller can then do whatever they wish with the data.

This therefore makes my code very manageable and easy to read & understand.

Class Explanation

The concept behind my class setup is that there are methods in the business logic which when called assists in completing tasks in the UI layer.

I have adopted this approach because it makes my code structured and well organized. The class setup I have described is the 3-Tier Application model. The UI layer will make calls to the methods in the business logic, then the business logic will query the database directly and return the response object back to the UI to be processed or manipulated. This means that the UI is not involved in the backend work.

**The image above illustrates the described 3 Tier model that I have implemented.**

Description of each class

Note:

Throughout my code I have made use of dynamic arrays using the generic classes available in VB.NET. When I use the term “list”, most likely I’m referring to the use of dynamic array which uses generics. An example is List(Of T), the T representing the generic object that can be used.

|  |  |
| --- | --- |
| **Name** | **Description** |
| *CustomControls*   |  |  | | --- | --- | | **Name** | **Description** | | *CustomPictureBox: Inherits PicutreBox* | *A custom picture box control I created* | | *CustomLinkLabel:*  *Inherits LinkLabel* | *A custom link label control I created* | | *CustomTreeNode:*  *Inheirts TreeNode* | *A custom tree node control I created* | | *Contains 3 classes of custom windows UI controls I’ve created by inheriting former window UI controls.* |

**CustomPictureBox-** This in the CustomPictureBox Class/Object

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *\_UserID* | *String* | *Stores the UserID of the picture box object* |
| *private* | *\_Username* | *String* | *Stores the Username of the picture box object* |
| *private* | *\_Password* | *String* | *Stores the Password of the picture box object* |
| *public* | *UserID* | *Integer : Property* | *Getter and Setter for the UserID property* |
| *public* | *Username* | *String :*  *Property* | *Getter and Setter for the Username property* |
| *public* | *Password* | *String :*  *Property* | *Getter and Setter for the Password property* |

**CustomLinkLabel** – This is the custom Link Label Class/Object

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *\_QuizID* | *Integer* | *Stores the* QuizID of *the link label object* |
| *public* | *QuizID* | *Integer* | *Getter and Setter for the QuizID property of the link label object* |

**CustomTreeNode –**This is the CustomTreeNode Class/Object

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *\_QuestionID* | *Integer* | *Stores the* QuestionID of *the link label object* |
| *public* | *QuestionID* | *Integer* | *Getter and Setter for the QuestionID property of the tree node object* |

**EntityBase –** *This is the base class where all the shared attributes and methods are defined so they can be used in all other classes throughout my solution.*

|  |
| --- |
| **Procedures/Functions** |
| **Access modifier** | **Name** | **Description** | **Parameters** | **Output/Returned Values** |
| *public* | *OpenConnection* | *Opens an SQL connection* | *()* | *()* |
| *public* | *CloseConnection* | *Closes the current SQL connection* | *()* | *()* |
| *public* | *PromptLogout* | *Prompts the user if they want to logout* | *()* | *()* |
| *public* | *Logout* | *Disposes of all forms and logs the user out* | *()* | *()* |

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *public* | *SQLCon* | *SqlConnection* | *The SQL connector which contains the connection string* |
| *public* | *SQLCmd* | *SqlCommand* | *Used to perform SQL commands queries* |
| *public* | *MyUser* | *User* | *The current user logged into the system* |

**MyExtension –** *This Module contains functions which extend the functions available for the image object and byte array data type.*

|  |
| --- |
| **Procedures/Functions** |
| **Access modifier** | **Name** | **Description** | **Parameters** | **Output/Returned Values** |
| *public* | *ToBytes* | *Converts the image object to an array of bytes* | *img* | *Converts and returns the image as an array of bytes.* |
| *public* | *ToStream* | *Converts “value” which is an array of bytes, to a memory stream.* | *value* | *Returns a memory stream containing the data byte data* |

**LoginService –** *This class is used to handle login requests to the system*

|  |
| --- |
| **Procedures/Functions** |
| **Access modifier** | **Name** | **Description** | **Parameters** | **Output/Returned Values** |
| *public* | *LoginCorrect* | *Logs in a user* | *Username, Password* | *Returns boolean resultant value from the credentials used.* |
| *private* | *GetUserID* | *Gets the class Id of the User* | *UserID* | *Returns the users classID* |
| *public* | *HasConnection* | *Checks if an SQL connection can be made* | *()* | *Returns true or false* |

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *public* | *IsConnected* | *Boolean* | *A flag to show if the initial SQL connection was established successfully* |

**QuestionHelper –** *The question helper class is used to handle any question related requests like adding deleting, editing, and fetching questions.*

|  |
| --- |
| **Procedures/Functions** |
| **Access modifier** | **Name** | **Description** | **Parameters** | **Output/Returned Values** |
| *public* | *AddQuestion* | *Adds the question object to the SQL db* | *Question* | *Returns true or false if adding the question was successful or not* |
| *private* | *AddFalseAnswers* | *Adds false answers for multi choice question* | *False*  *Answers()* | *Returns true or false if adding the question was successful or not* |
| *public* | *DeleteQuestion* | *Deletes a question from the SQL db* | *QuestionID* | *Returns true if successful or false if it’s not* |
| *public* | *UpdateQuestion* | *Updates a question on the SQL db* | *QuestionID, Question* | *Returns true if successful or false if it’s not* |
| *private* | *UpdateMultiChoice* | *Updates the MultiChoice Questions* | *Question,*  *QuestionID* | *Returns true if successful or false if it’s not* |
| *private* | *GetNextIncrement* | *Gets the Next question Id from the SQL table* | *()* | *Returns an Integer of next Question ID* |
| *Public* | *GetQuestion* | *Get a question from the SQL db* | *QuestionID* | *Returns the question requested* |
| *private* | *GetFalseAnswers* | *Gets the false answers of a question* | *QuestionID* | *Returns an array of the multi choice answers* |

**StudentService –** *The student class is used to handle all requests made by the student the system*

|  |
| --- |
| **Procedures/Functions** |
| **Access modifier** | **Name** | **Description** | **Parameters** | **Output/Returned Values** |
| *public* | *PromptLogout* | *An overridden method that prompts the user if they want to logout* | *()* | *()* |
| *private* | *GetRandom*  *QuestionIDs* | *Gets the amount of question Ids request for. However it gets random questions Ids.* | *amount* | *Returns a list of the question IDs found.* |
| *public* | *GetUncompleted*  *Quizzes* | *Gets a list of all the uncompleted*  *quizzes for a specific student.* | *userId* | *Return a list of all the quizzes the student hasn’t completed.* |
| *public* | *GetQuestionIDs* | *Gets all the question Ids in a quiz* | *quizId* | *Returns a list of all the question Ids* |
| *private* | *AnswerIsCorrect* | *Marks a question answered by the student* | *actualAnswer,*  *userAnswer* | *Returns true if answer is correct or false if it’s not.* |
| *private* | *SaveQuizResult* | *Saves the result the student got in a quiz* | *quizId, userId, mark, total*  *questions* | *Returns a Boolean to indicate if saving quiz result was successful or not* |

**TeacherService –** *This class is used to assist the Teacher Form to perform backend actions.*

|  |
| --- |
| **Procedures**  **/Functions** |
| **Access Modifier** | **Name** | **Description** | **Parameters** | **Output/Returned Values** |
| *Public* | *GetClassInfo* | *Gets the block name, room number of a class* | *classID,* | *Returns a string of the block name, room number in a specified format.* |
| *Public* | *GetStudents* | *Finds all the students in the teacher’s class which match the name criteria. This is used for searching for students in a class by their name.* | *classID, name,* | *Returns a list of all the students found in the form of the “User” object. The List returned makes use of generics.* |
| *Private* | *CreateQuizID* | *Creates a new QuizID for a new quiz* | *()* | *Returns an Integer of the QuizID created* |
| *Public* | *AddQuiz* | *Adds a list/array of question Ids to a new quiz* | *questionIdList, quizTitle,* | *Return the boolean resultant of the method* |
| *Private* | *AddQuizTitle* | *Adds the quiz title for a quiz* | *quizId, title,* | *Return the boolean resultant of the method* |
| *Public* | *GetStudent*  *QuizResults* | *Gets all the completed quizzes for a student from a time frame.* | *userId, dateFrom, dateTo,* | *Returns a list of all the completed quizzes for the specific student requested.* |
| *Private* | *Calculate*  *Percentage* | *Calculates the percentage of two values* | *total, mark,* | *Returns the percentage* |
| *Public* | *FetchQuiz* | *Gets all the quizzes in the database and adds it to a tree node object.* | *()* |  |
| *Public* | *PopulateTreeView* | *Used to search for questions in the database based upon the grade and or topic* | *grade, topic,* |  |
| *Private* | *AddQuiz*  *ToTree* | *Adds all the question for each quiz to a tree view control* | *quizList,* |  |
| *Public* | *Export*  *Questions* | *Gets a list of all the questions and answers for each question Id in the questionIdList* | *quesionIdList,* | *Returns a string which contains all the question and answer for each question requested.* |
| *Public* | *GetStudentsQuizResults* | *Gets a list of the UserIDs of students who have completed more than 1 quiz.* | *classId,* | *Returns a list of the StudentQuizInfo object that contains students quiz information* |
| *Private* | *GetStudent*  *QuizInfo* | *Gets a detailed list of all the quizzes the student has completed.* | *studentId,* | *Returns all the quizzes a student has completed in an object* |

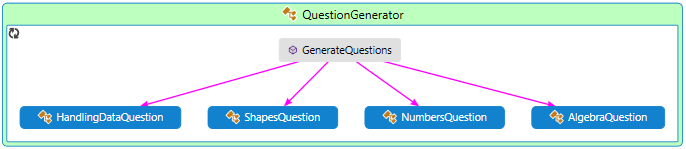
**QuickSort –** *A quick sort classed used to sort custom objects. This is used when the teacher wants to export a summary that shows the total average mark for all the quizzes each student in the teachers’ class has completed.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **Name** | **Description** | **Parameters** | **Output/Returned Values** |
| *Public* | *Sort* | *Performs validations, and if the validation criteria has been met, it then sorts the data by calling the SortList method.* | *arrayList,* | *Sorted array* |
| *Private* | *SortList* | *The method required to make the recursive calls for the quick sort algorithm.* | *lowIndex, hiIndex, arrayList,* | *Sorted array* |
| *Private* | *GetPivot*  *Position* | *This method sorts the item in the list of items, and it also returns the index of the pivot.* | *lowIndex, hiIndex, arrayList,* | *Integer* |
| *Private* | *Swap* | *Used to swap elements in the list of objects to sort.* | *indexFrom, indexTo,*  *ByRef arrayList* |  |

**AdminService -***This class is used to handle SQL related requests that the admin can perform*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Public* | *GetUsers* | *Gets a list of all the users in the system, who’s name match the name criteria* | *name,* | *Returns a list of all the found users. This returns a list of the user object* |
| *Public* | *AddAccount* | *Adds a new user to the SQL tables.* | *userId, firstname, lastname, userRank, img, classId,* | *Returns true if it was successful or false if it wasn’t* |
| *Private* | *AddUserClass* | *Assigns a user to a class in the SQL database* | *userId, classId,* | *Returns a Boolean if action was successful* |
| *Private* | *ParseStr* | *Used to parse the firstname and lastame of a user. It will remove all the non-alphabetic characters found in the name* | *txt,* | *Returns the new string with all the non- alphabetic characters removed.* |
| *Private* | *GenerateEmail* | *Generates an email for the user depending on parameter values. This mimics the SFX college system.* | *userId, firstname, lastname, userRank,* | *Returns the generated email* |
| *Private* | *Generate*  *Username* | *Generates a username for the user based on the values of the parameters. This also mimics the colleges system.* | *userId, firstname, lastname, userRank,* | *Returns the generated username* |
| *Private* | *Generate*  *Password* | *Generates a random password* | *()* | *Returns the generated password* |
| *Public* | *GetAllClasses* | *Gets a list of all the classes in the system* | *()* | *Returns a list in the form of the ClassRoom object* |
| *Public* | *CreateClass* | *Creates a new class in the SQL db* | *block, room,* | *Returns a Boolean if action was successful or not* |
| *Public* | *AddOccupied*  *Classes* | *Adds the info of all the classes that is assigned to at least 1 teacher, to a DataGridView* | *()* |  |
| *Public* | *AddEmpty*  *Classes* | *Adds the info of all the classes that is not assigned to at least 1 teacher, to a DataGridView* | *()* |  |
| *Public* | *DeleteAccount* | *Deletes a user from the SQL tables* | *userId,* | *Boolean to represent success state* |
| *Public* | *UpdateAccount* | *Update a user’s information from the database* | *userId,*  *firstname, lastname, email, password, img, classId, userRank,* | *Returns true if it was successful or false if it wasn’t* |
| *Public* | *GetUser*  *Information* | *Gets detailed information of specific user.* | *userId, userRank,* | *Returns the user information.* |

**QuestionGenerator *–*** *This class is used to generate different types of maths questions. The image below illustrates the hierarchy structure of this class.*

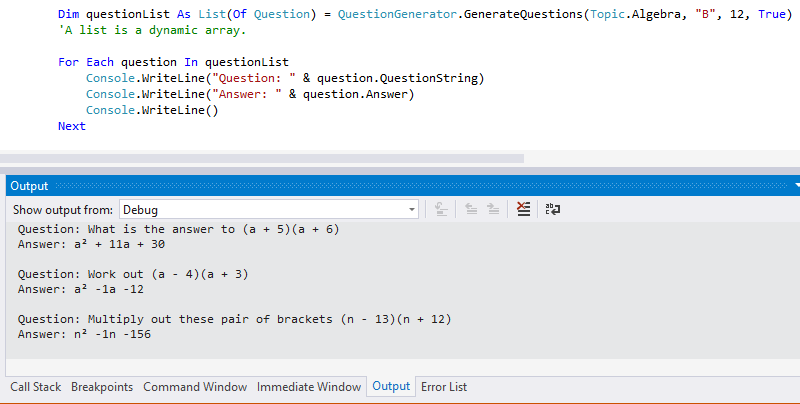
****

Inside the QuestionGenerator class, there is one public method visible outside of the class scope, which is GenerateQuestions(). Essentially the GenerateQuestions() function will call any of the nested classes inside the QuestionGenerator class *(e.g. Handling Data Question, Shapes Questions, Numbers Questions and Algebra Question class)* in order to generate questions. Each nested class has a function called GenerateQuestion, which when called will generate question of its type. For example, the AlgebraQuestion class will only generate and return algebra questions. Each generated question is accumulated into an array which is then returned to the caller from the GenerateQuestions() method in the QuestionGenerator class.

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *WordStarters* | *String Array* | *An array of all the words a sentence for a worded question can start with* |
| *private* | *Variabes* | *String array* | *An array of mathematical variables. E.g. “x” “y”* |
| *private* | *Powers* | *String Array* | *An array of Unicode characters representing powers* |
| *private* | *Operators* | *String Array* | *An array of arithmetic operators. E.g. + and -* |
| *private* | *R* | *Random* | *The random object, used for generating pseudo-random numbers* |

|  |
| --- |
| ***Methods and procedures*** |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Public* | *Generate*  *Questions* | *Generates questions, depending on the topic selected, grade difficulty, and amount requested.* | *qTopic, grade, amount*  *Requested, multiChoice,* | *Returns a list/array of all the questions generated.* |
| *Private* | *GetDifficulty* | *Returns the difficulty level of the param value as an enum.* | *grade,* | *Returns the difficulty level of the param value as an enum.* |
| *Private* | *GetRandom*  *Power* | *Gets a random power sign from the power array* | *()* | *Returns a random power from the powers array* |
| *Private* | *GetRandom*  *Word* | *Gets a random word from the word array* | *()* | *Returns a random word from the words array* |
| *Private* | *GetRandom*  *Operator* | *Gets a random operator sign from the operator array* | *()* | *Returns a random operator from the operators array* |
| *Private* | *GetRandom*  *Variable* | *Gets a random variable from the variable array* | *()* | *Returns a random variable from the variable array* |
| *Private* | *GetRandom*  *Number* | *Gets a random number from the number array* | *difficultyLvl,* | *Returns a random number from the numbers array* |

I made the GenerateQuestion a public static/shared method. This is because of the simplicity when it’s called to generate questions. It’s easy to generate questions and manipulate them.

For example, generating questions can be done in one line of code.

These are all the algebra questions that have just been generated at runtime.

It looks very natural when generating questions. From here, the caller (me) can choose to do anything with the generated questions. In my system, I’d just add them to the SQL table. In the future I could use this concept to generate questions for students to answer on the fly. This makes the class easy and elegant to use. Every method inside the class is well encapsulated thus hiding all the complexity from the caller.

**ShapesQuestion *–*** *This class is for generating square, triangle and rectangle type question.*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *thisQuestion* | *Question* | *This is the current question that is being generated. It’s of Type Question, which is a custom object.* |
| *private* | *Shapes* | *Shape Array* | *An array of an enum which contains all the three shapes that can be generated. Which is Shape.Triangle, Shape.Square and Shape.Rectangle.* |
| *private* | *QTypes* | *QuestionType Array* | *An array that contains all the QuestionTypes that can be generated. There is only two. Which Is QuestionType.Area and QuestionType.Perimeter.* |
| *private* | *Measurements* | *String Array* | *An array containing all the possible unit measurements that can be used. Which are “cm” and “m”* |

|  |
| --- |
| **Procedures/**  **Functions** |
| **Access Modifier** | **Name** | **Description** | **Parameters** | **Output/Returned Values** |
| *Public* | *Generate*  *Question* | *Generates a question* | *()* | *Returns the generated question to the caller.* |
| *Private* | *Draw*  *Triangle* | *Draws the image of the triangle question being generated.* | *type, unit,* | *Returns the image that was generated* |
| *Private* | *Draw*  *Square* | *Draws the image of the square question being generated.* | *type, unit,* | *Returns the image that was generated* |
| *Private* | *Draw*  *Rectangle* | *Draws the image of the rectangle question being generated.* | *type, unit,* | *Returns the image that was generated* |
| *Private* | *SolveTriangle*  *Question* | *Solves the answer to the triangle question generated.* | *type, hypo, opposite, adjacent, unit,* | *Saves the question generated to the thisQuestion var* |
| *Private* | *Solve*  *Square*  *Question* | *Solves the answer to the square question generated.* | *type, side, unit,* | *Saves the question generated to the thisQuestion var* |
| *Private* | *Solve*  *Rectangle*  *Question* | *Solves the answer to the rectangle question generated.* | *type, base, height, unit,* | *Saves the question generated to the thisQuestion variable* |
| *Private* | *Dispose*  *Objects* | *Disposes the objects used to draw the images* | *ByRef b, ByRef f,* |  |

**AlgebraQuestion *-*** *This class is for generating algebra type questions.*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *thisQuestion* | *Question* | *This is the current question that is being generated. It’s of Type Question, which is a custom object.* |

|  |
| --- |
| **Procedures/**  **Functions** |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Public* | *Generate*  *Question* | *Generates the algebra question* | *()* | *Returns the generated question to the caller.* |
| *Private* | *Solve*  *Question* | *Solves the question that has been generated.* | *variable, firstBracket, secondBracket,* | *Returns the answer to the question that had been generated.* |
| *Private* | *Generate*  *FalseAnswers* | *Generates false answers to the question. This is for multiple choice questions.* | *var, firstBracket, secondBracket,* | *Returns the false answers created.* |

**NumbersQuestion *–*** *This class is for generating numbers type questions.*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *thisQuestion* | *Question* | *This is the current question that is being generated. It’s of Type Question, which is a custom object.* |

|  |
| --- |
| **Procedures/**  **Functions** |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Public* | *Generate*  *Question* | *Generates a numbers type question* | *difficultyLevel,* | *Returns the generated question to the caller.* |
| *Private* | *Solve*  *Question* | *Solves the question that has been generated.* | *number, power,* | *Returns the answer to the question that had been generated.* |
| *Private* | *Generate*  *FalseAnswers* | *Generates false answers to the question. This is for multiple choice questions.* | *correctAnswer,* | *Returns the false answers created.* |

**HandlingDataQuestion *–*** *This class is for generating handling data type questions.*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *thisQuestion* | *Question* | *This is the current question that is being generated. It’s of Type Question, which is a custom object.* |

|  |
| --- |
| **Procedures/**  **Functions** |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Public* | *Generate*  *Question* | *Generates a handling data type question based on the difficulty level chosen* | *difficultyLevel,* | *Returns the generated question to the caller.* |
| *Private* | *Solve*  *Question* | *Solves the question that has been generated.* | *Percentage, number* | *Returns the answer to the question that had been generated.* |
| *Private* | *Generate*  *FalseAnswers* | *Generates false answers to the question. This is for multiple choice questions.* | *answer,* | *Returns the false answers created.* |

*Description of each form*

**frmLogin -** *This is the login form, this is where the user logs in*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *Login* | *LoginService* | *Used to access the methods in the LoginService class.* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Private* | *CreateControls* | *Creates the controls that will be placed on the form.* | *()* |  |
| *Private* | *New* | *This is the constructor for this form. This is where the controls are created.* | *()* |  |
| *Private* | *txtPassword*  *\_KeyDown* | *An event handler for when the enter button is pressed whilst txtPassword has focus. It’s used to send a login request to the system.* | *sender, e,* |  |
| *Private* | *LoginForm*  *\_Shown* | *An event handler for when the form loads. It will connect to the SQL server.* | *sender, e,* |  |
| *Private* | *CheckFor*  *Connection* | *Checks if the system can connect to the SQL server.* | *()* | *Returns if it has successfully connected or not as boolean* |
| *Private* | *UpdateLabel* | *Updates the text of the label depending on the connection status.* | *text, foreColor,* |  |
| *Private* | *btnLogin\_Click* | *Event handler for when the login button is clicked.* | *sender, e,* |  |
| *Private* | *ShowForm* | *Redirects the user that has successfully logged in to the correct form depending to their rank* | *userRank,* |  |

**frmAdmin –** *This is the admin form where the admin can manage user accounts.*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *Admin* | *AdminService* | *Used to access the methods in the AdminService class.* |

|  |
| --- |
| ***Procedures/Functions*** |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Private* | *CreateTab1*  *Controls* | *Creates all the controls required for the first tab page* | *()* |  |
| *Private* | *CreateTab2*  *Controls* | *Creates all the controls required for the second tab page* | *()* |  |
| *Private* | *New* | *This is the constructor for this form. This is where the controls are created.* | *()* |  |
| *Private* | *frmAdmin\_*  *Load* | *Gets the welcome message when the form opens* |  |  |
| *Private* | *frmAdmin\_*  *Closing* | *Exists the whole application when the event is raised.* | *sender, e,* |  |
| *Private* | *GetRank* | *Gets the rank of the user that is about to be added to the system in the form of an enum. The rank returned is dependent on the radio button checked.* | *()* | *Returns the rank of the user as the enum Rank. If the radio button teacher was selected. It would return Rank.Teacher* |
| *Private* | *GetClassID* | *Gets the ClassID of selected item in the combo box control.* | *()* | *Returns the ClassID* |
| *Private* | *btnAdd\_Click* | *Prompts the user if they’re sure if they want to add the user before adding a new user* | *sender, e,* |  |
| *Private* | *ClearControls* | *Resets the content of the controls that were used to add a user.* | *()* |  |
| *Private* | *btnSearch*  *\_Click* | *Searches for all the students in system* | *sender, e,* |  |
| *Private* | *lstUsers\_*  *SelectedIndex*  *Changed* | *This event is for when a user has been selected from the list box.* | *sender, e,* |  |
| *Private* | *GetUserInfo* | *Opens frmEdit form which contains detailed information on the selected user from the list box.* | *userId, userRank,* |  |
| *Private* | *picUser\_Click* | *Opens an OpeFileDialog allowing the admin to import an image for the users picture* | *sender, e,* |  |
| *Private* | *txtSearch*  *\_KeyDown* | *Searches for all the users in system* | *sender, e,* |  |
| *Private* | *rdo\_Checked*  *Changed* | *Refresh the items in the class combo box depending on the radio button that is currently checked. If the student or teacher combo box is selected. The class combo box will show.* | *sender, e,* |  |
| *Private* | *LblsVisible* | *Changes the labels visibility.* | *value,* |  |
| *Private* | *RefreshClass*  *List* | *Gets a list of all classes in the system* | *()* |  |
| *Private* | *ActivateAdd*  *Btn* | *Activates the add button if all required fields are filled in.* | *()* |  |
| *Private* | *btnLogout*  *\_Click* | *Logs the user out* | *sender, e,* |  |
| *Private* | *btnAddClass*  *\_Click* | *Adds a class to the system* | *sender, e,* |  |
| *Private* | *btnSearch*  *Teacher\_Click* | *Searches for all the classes in the system* | *sender, e,* |  |

**frmEdit-***This is the form where a user’s account is updates or deleted.*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *Admin* | *AdminService* | *Used to access the methods in the AdminService class.* |

|  |
| --- |
| ***Procedures/***  ***Functions*** |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Private* | *Create*  *Controls* | *Creates the controls for the form* | *()* |  |
| *Private* | *New* | *This is the constructor for this form. This is where the controls are created.* | *()* |  |
| *Private* | chkBoxEdit\_  Checked  Changed | *Changes the state of the controls* |  |  |
| *Private* | *ChangeState* | *Changes the state of the controls.* | *value,* |  |
| *Private* | *btnDelete*  *\_Click* | *Deletes the current user from the SQL tables.* | *sender, e,* |  |
| *Private* | *btnUpdate*  *\_Click* | *Updates the users information* | *sender, e,* |  |
| *Private* | *cboRank*  *\_SelectedIndex*  *Changed* | *Occurs when the rank combo box is selected.* | *sender, e,* |  |
| *Private* | *ChangeVisibility* | *Changes the visibility of the class combo box and label, so it only shows for certain user types.* | *Value,* |  |
| *Private* | *picUser\_Click* | *Used to import an image for the user using an OpenFileDialog* | *sender, e,* |  |
| *Private* | *IsValidEmail* | *This function will validate the email passed. It will see if the email passed can be matched with the email regex pattern* | *email* | *Returns true if the email passed is a valid email and false if it isn’t* |

**frmTeacher *–*** *This form is the form for the teacher. They do teacher related things using this form.*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *Teacher* | *TeacherService* | *Used to access the methods in the TeacherService class.* |
| *private* | *FirstTime* | *Boolean* | *To indicate if it’s the first time the Form\_Resize event has been raised* |
| *private* | *PreviousState* | *FormWindowState* | *Previous state of the form. E.g. Minimized, Maximized.* |

|  |
| --- |
| ***Procedures/***  ***Functions*** |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Private* | *New* | *This is the constructor for this form. This is where the controls are created.* | *()* |  |
| *Private* | *CreateTab1*  *Controls* | *Create all the controls in tab control 1* | *()* |  |
| *Private* | *CreateTab2*  *Controls* | *Create all the controls in tab control 2* | *()* |  |
| *Private* | *CreateTab3*  *Controls* | *Create all the controls in tab control 3* | *()* |  |
| *Private* | *frmTeacher*  *\_Load* | *Gets the welcome back message and the class information.* | *sender, e,* |  |
| *Private* | *Form\_Closing* | *Closes all parent forms when form closing event is fired.* | *sender, e,* |  |
| *Private* | *btnLogout\_Click* | *Logs out the teacher from the system* | *sender, e,* |  |
| *Private* | *frmTeacher*  *\_Resize* | *Resizes all the controls to proportion when the teacher resizes the controls.* | *sender, e,* |  |
| *Private* | *ReLocateCtrls* | *Relocates all the controls on the form.* | *xLocation,* |  |
| *Private* | *InitializeChart* | *Labels and adds titles to the chart.* | *()* |  |
| *Private* | *SaveToTxt* | *Used to save the content of a string to the user’s computer.* | *content,* |  |
| *Private* | *txtSearchField\_KeyDown* | *Used to search for all the students in the teachers class.* | *sender, e,* |  |
| *Private* | *ClearControls* | *Clears all the controls in the Panel of students.* | *()* |  |
| *Private* | *btnSearch\_Click* | *Used to search for all the students in the teachers class.* | *sender, e,* |  |
| *Private* | *CreateDynamicStudents* | *An algorithm that will create dynamic picture boxes representing students in the system.* | *student* | *The student variable is an array of students.* |
| *Private* | *CreateDynamicLabels* | *Creates the label of the student the picture box above represents.* | *name, xLocation, yLocation,* |  |
| *Private* | *GetEstimateX* | *Gets the estimated x value the label should be incremented by* | *lbl,* | *X location as integer* |
| *Private* | *MeasureLabel* | *Measures a label* | *lbl,* | *Returns label size* |
| *Private* | *GetMidPoint* | *Gets the midpoint of a picturebox* | *p,* | *Location of midpoint* |
| *Private* | *StudentPicture\_Clicked* | *When a dynamic picture box is clicked. It will go to the next tab and show their progress* | *sender, e,* |  |
| *Private* | *FirstDayOf*  *Month* | *Gets the date for the first day of the current month* | *()* | *DateTime* |
| *Private* | *btnExportInfo\_Click* | *Exports the MyGCSE account information for each student in the viewing panel.* | *sender, e,* |  |
| *Private* | *GetStudentsInfo* | *Gets the information of all the dynamic students in the viewing panel.* | *()* | *Returns a string of the information* |
| *Private* | *btnExportResult\_Click* | *Exports the overall summary of all the quiz results in the teacher’s class.* | *sender, e,* |  |
| *Public* | *DateTime*  *\_ValueChanged* | *Updates the quiz result date range depending on the value picked on date time picker.* | *sender, e,* |  |
| *Private* | *btnAdd*  *Question\_Click* | *Opens a form that is for adding questions.* | *sender, e,* |  |
| *Private* | *btnSearch*  *Question\_Click* | *Searches for all the questions in the system* | *sender, e,* |  |
| *Private* | *HandleQuestion* | *Handles the request to search for questions in the system* | *topic, grade,* |  |
| *Private* | *tvQuestions\_*  *AfterCheck* | *Raised when a parent tree node is checked.* | *sender, e,* |  |
| *Private* | *btnGenerate*  *Question\_Click* | *Opens a form for generating questions* | *sender, e,* |  |
| *Private* | *GetChecked*  *Count* | *Used for data validation* | *()* | *Integer* |
| *Private* | *btnAddQuiz*  *\_Click* | *Adds a new quiz to the system* | *sender, e,* |  |
| *Private* | *DeleteQstn* | *Handles the request for deleting questions.* | *()* |  |
| *Private* | *btnDelete*  *Question\_Click* | *Deletes a question* | *sender, e,* |  |
| *Private* | *btnExport*  *Answers\_Click* | *Exports the answers of a series of question to a .txt form* | *sender, e,* |  |
| *Private* | *tvQuestions*  *\_AfterSelect* | *Event raised for when a child tree node is selected.* | *sender, e,* |  |

**frmQuestions *–*** *This is the form where questions can be added or edited.*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *public* | *CurrentAction* | *ActionType* | *Since this form can be used for either adding a question, or editing a question. This variable indicates what type of action it is currently being used for.* |
| *public* | *CurrentQuestionID* | *Integer* | *This is the question ID of the question that is being viewed. When updating a question it knows the question Id in order to update.* |

|  |
| --- |
| ***Procedures/***  ***Functions*** |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Private* | *New* | *This is the constructor for this form. This is where the controls are created.* | *()* |  |
| *Private* | *CreateControls* | *Creates the controls for the form* | *()* |  |
| *Private* | *frmQuestions*  *\_Shown* | *Brings the form to the front when called.* | *sender, e,* |  |
| *Private* | *rdoSingleAnswer*  *\_Checked*  *Changed* | *If single answer radio button is checked, the layout of the form will change. All the multiple choice textboxes will disappear. Calls ChangeVisibility.* | *sender, e,* |  |
| *Private* | *rdoMultiChoice\_*  *Checked*  *Changed* | *If multi choice radio button is checked, the layout of the form will change. All the multiple choice textboxes will show. Calls ChangeVisibility.* | *sender, e,* |  |
| *Public* | *ChangeVisibility* | *Changes visibility of the text boxes depending on the radio button that was checked.* | *Value,* |  |
| *Private* | *picQuestion*  *\_Click* | *Opens an OpenFileDialog that allows the user to import an image for the question.* | *sender, e,* |  |
| *Private* | *btn\_Click* | *This is a multi-purpose button, that will do different things depending on the value of CurrentAction variable* | *sender, e,* |  |
| *Private* | *InputEmpty* | *Validates user input to make sure the appropriate text boxes are not empty. The value it returns depends on the radio button selected.* | *()* | *Boolean* |
| *Private* | *FormatText* | *Replaces plain mathematical representations to the Unicode equivalent. E.g. ^2 is changed to* ² | *()* |  |
| *Private* | *GetCurrent*  *Question* | *Gets the current question on the form by extracting data from the text boxes and other controls, then adds it to the Question object* | *()* | *Returns the question as an object* |
| *Private* | *AddQuestion* | *Handles the request for adding a question* | *()* |  |
| *Private* | *EditQuestion* | *Handles the request for editing a question* | *()* |  |
| *Public* | *UsageMode* | *Used to set the usage mode for the multi-purpose button. Either add or edit.* | *action,* |  |

**frmGenerateQuestions –** *This is the form used to generate questions.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output/Returned Values*** |
| *Private* | *New* | *This is the constructor for this form. This is where the controls are created.* | *()* |  |
| *Private* | *CreateControls* | *Creates all the controls required for the form* | *()* |  |
| *Private* | *btnGenerate*  *\_Click* | *The button is used to submit the generate request to the question generator class.* | *sender, e,* |  |

**frmStudent *–*** *The form the student logs in to take quizzes set by their teacher or attempt random questions in the system*

|  |
| --- |
| **Variables** |
| **Access modifier** | **Name** | **Type** | **Description** |
| *private* | *Student* | *StudentService* | *Used to access the methods in the StudentService class.* |
| *private* | *QuestionIdQueue* | *Queue* | *A queue of all the questions Id’s the student hasn’t yet answered* |
| *private* | *TotalQuestions* | *Integer* | *The initial amount of questions in the list of questions the student is answering.* |
| *private* | *CurrentQuestion* | *Question* | *The current question being answered* |
| *private* | *CurrentQuizID* | *Integer* | *The current quiz ID of the quiz the student is participating in.* |
| *private* | *CurrentMark* | *Integer* | *The students current quiz score* |

|  |
| --- |
| ***Procedures/***  ***Functions*** |
| ***Access Modifier*** | ***Name*** | ***Description*** | ***Parameters*** | ***Output***  ***/Returned Values*** |
| *Private* | *CreateControls* | *Creates all the controls for the form* | *()* |  |
| *Private* | *New* | *This is the constructor for this form. This is where the controls are created.* | *()* |  |
| *Private* | *btnLogout*  *\_Click* | *Logs the user out of the system* | *sender, e,* |  |
| *Private* | *frmStudent*  *\_Closing* | *Closes all parent forms when the user closes the form* | *sender, e,* |  |
| *Private* | *frmStudent*  *\_Load* | *Shows welcome message and calls the GetUnCompleted*  *Quizzes Sub* | *sender, e,* |  |
| *Private* | *GetUncompletedQuizzez* | *Gets a list of all the quizzes the student hasn’t completed and processes it.* | *()* |  |
| *Private* | *UpdateLabel* | *Updates a label to show the number of uncompleted quizzes* | *count,* |  |
| *Private* | *ClearLinkLbls* | *Clears all the dynamically created link labels in the current form.* | *()* |  |
| *Private* | *CreateQuizLbls* | *An algorithm used to place each link label of the uncompleted quiz in a pre- defined pattern.* | *quizList,* |  |
| *Public* | *PromptStart*  *Quiz* | *Asks the student if they want to start the quiz* | *sender, e,* |  |
| *Public* | *PromptRandomQuestions* | *Prompts the user if they want to start a randomized set of questions* | *sender, e,* |  |
| *Private* | *ShowControls* | *Changes the visibility state of the controls on the form* | *value,* |  |
| *Private* | *btnNext\_Click* | *Used to submit the answer of the current question, and get the next question.* | *sender, e,* |  |
| *Private* | *GetNext*  *Question*  *Handler* | *Gets the next question in the Queue and displays it according to the user* | *()* |  |
| *Private* | *Randomize*  *Answers* | *Randomizes an array of answers.* | *qstnArray* |  |
| *Private* | *GetUserAnswer* | *Gets the users answer depending if the current question is multiple choice or not* | *()* | *Returns the user answer* |
| *Private* | *ClearUser*  *Answers* | *Clears all the input boxes for answering questions* | *()* |  |
| *Private* | *ChangeControlsVisibility* | *Changes the visibility state of the controls on the form* | *value,* |  |
| *Private* | *ShowResult* | *Shows a dialog box to show students their result* | *mark, total,* |  |
| *Public* | *Update*  *Information* | *Updates the current progress. E.g. shows how much questions completed out of the total* | *()* |  |
| *Private* | *txtAnswer\_TextChanged* | *Formats the text to the Unicode equivalent* | *sender, e,* |  |

Detailed algorithm design

|  |  |  |
| --- | --- | --- |
| **Title** | ParseStr | **Location**: AdminService class |
| **Description** | A function that removes every non-alphabetic character in any given string using a regex pattern. This is used for the GenerateUsername and GenerateEmail functions. This is for keeping data integrity and for having data in the correct format. | |
| **Pseudo Code** | Regex.Pattern = “[^a-zA-Z\- ]”  String final = Replace(Regex.Pattern, “ ”)  Return final | |
| **Real code** | Private Function ParseStr(txt As String) As String  ' Replaces every occurance of a non alphabetic text, excluding - and returns the new value  Return Regex.Replace(txt, "[^a-zA-Z\- ]", String.Empty)  End Function | |

|  |  |  |
| --- | --- | --- |
| **Title** | GenerateUsername | **Location**: AdminService class |
| **Description** | Generates the “MyGCSE Maths” account username based upon the rank, Firstname, Lastname and UserID that is passed down by the parameters Generally the algorithm works by extracting substrings. | |
| **Pseudo Code** | If Rank.Student  Username = Lastname & Firstname.FirstLetter  If Lastname.Length Is > 7 Then Lastname = Lastname.Last7Characters  Else  Username = Firstname.FirstLetter & Lastname  Finish If  Return Username.ToLowerCase | |
| **Real code** | Private Function GenerateUsername(userId As Integer, firstname As String, lastname As String, userRank As Rank) As String  Dim username As String = String.Empty  'username is what we are going to return  'If they're a student, we generate their username differently..  If userRank = Rank.Student Then  'If their lastname is greater than 7, then get the first 7 letters of their firstname, if it's not it leaves it  If lastname.Length > 7 Then lastname = lastname.Substring(0, 7)  username = lastname & firstname.Substring(0, 1) & userId.ToString  'their username their lastname, first character of their surname, and their userid  Else  'If they're a admin or teacher, we generate their username differently..  username = firstname.Substring(0, 1) & lastname  End If  Return username.ToLower()  End Function | |
| **Title** | GeneratePassword | **Location**: AdminService class |
| **Description** | Generates a random password by running a For Loop 7 times and picking a random substring from the word variable and appending the substring it picks up to the password variable. At the end of the loop it returns the generated password. | |
| **Pseudo Code** | String words = “abcdefghiklmnopqrstuwvxyz123456789”  var sb  For Loop 4 Times  Int rnd = rand.Next(0, words.length)  sb.Append(words.Substring(rnd, 2))  End Loop  Return sb.ToString() | |
| **Real code** | Private Function GeneratePassword() As String  Dim words As String = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"  Dim r As New Random  Dim sb As New StringBuilder  For i = 1 To 4  Dim rdnNum As Integer = r.Next(0, words.Length - 1)  sb.Append(words.Substring(rdnNum, 2))  ' Picks a random substring from the words variable and appends it to a string builder  Next  Return sb.ToString()  End Function | |

|  |  |  |
| --- | --- | --- |
| **Title** | IsValidEmail | **Location**: frmEdit |
| **Description** | Validates the email address by parsing it using the Regular expression class in VB.NET. Returns true if the email is valid or false if it is not valid. | |
| **Pseudo Code** | var email = “Emai here”  var emailPattern = “[\w\d-+\_.]+@[\w.-]+\.[\w]{2,5}”  Boolean isMatch = Regex.Match(email, emailPattern)  Return isMatch | |
| **Real code** | Private Function IsValidEmail(email As String) As Boolean  Const emailPattern As String = "[\w\d-+\_.]+@[\w.-]+\.[\w]{2,5}"  Dim isMatch As Boolean = Regex.Match(email, emailPattern).Value = email  Return isMatch  'The email in the parameters is validated using the Regex expression.  'If the Regex engine mathes the email the the function returns true, and if it doesn't  'it returns false  End Function | |

|  |  |  |
| --- | --- | --- |
| **Title** | CreateQuizLbls | **Location**: frmStudent |
| **Description** | This algorithm will be used for dynamically creating the link labels objects for quizzes that have not yet been completed by a student.  First, the algorithm calls a function that returns an array of data containing all the quizzes the student hasn’t yet completed from the SQL database. The data returned is of type Quiz, which is a custom object.  The algorithm will then create a new CustomLinkLabel object for each item/quiz in the array using the data from the quizList array. After that the location and size of the link label is then defined. The Y Axis location of the label will be incremented upon each iteration, therefore the location of next label will be different to the previous one. Once 10 labels have been created, the algorithm will increment the X Location and reset the Y Location which places the label back to the top of the form.  At the end of each iteration an event handler for the click event will be added, which will allow students to start a quiz by clicking on the label. | |
| **Pseudo Code** | int xLocation = 25int yLocation = 86int amountCreated = 0List(Of Quiz) quizList = GetUnCompleteQuizzes(studentID)’A list is just a dynamic arrayFor indx = Loop From 0 To quizList.count - 1Declare lblQuiz = New CustomLinkLabellblQuiz .QuizID = quizList (indx).QuizIDlblQuiz .Text = quizList(indx).TitlelblQuiz .Location = (xLocation, yLocation)lblQuiz.AutoSize = TrueamountCreated += 1yLocation += 33If amountCreated == 10 ThenxLocation += 350yLocation = 86amountCreated = 0Finish IfAddEvent lblQuiz.Click To StartQuiz()Controls.Add(lblQuiz) **End** Loop | |
| **Real code** | Dim quizList As List(Of Quiz) = Student.GetUncompletedQuizzes(MyUser.UserID)  'Gets a list/array of all the quizzes the student hasn't yet completed. The quizList is passed down to sub through the parameters.  CreateQuizLbls(quizList)  Private Sub CreateQuizLbls(quizList As List(Of Quiz))  ClearLinkLbls() 'Clears pre-existing CustomLinkLabels on the form  'Initial location  Dim xLocation As Integer = 25  Dim yLocation As Integer = 86  Dim amountCreated As Integer = 0  For i = 0 To quizList.Count - 1  Dim linkLbl As New CustomLinkLabel 'Creates a new instance on each iteration  linkLbl.QuizID = quizList(i).QuizID  linkLbl.Text = quizList(i).Title  linkLbl.Location = New Point(xLocation, yLocation)  linkLbl.AutoSize = True  'Defines the properties of the CustomLinkLabel  amountCreated += 1  yLocation += 33  'Checks the amount of labels created. If it's equal to 10  'It increments the xLocation to move to the right of the screen.  'And resets the yLocation and amount created.  If amountCreated = 10 Then  xLocation += 350  yLocation = 86  amountCreated = 0  End If  'Adds an event handler for the clicked event  AddHandler linkLbl.Click, AddressOf PromptStartQuiz  Me.Controls.Add(linkLbl)  Next  End Sub | |

|  |  |  |
| --- | --- | --- |
| **Title** | RandomizeAnswers | **Location**: frmStudent |
| **Description** | This algorithm takes an array and randomizes the order of each item. The parameters passed is ByRef so the changes made will reflect on the original array. This will be used to randomize multiple choice answers to a question before displaying it to the user. | |
| **Pseudo Code** | Begin Sub (ByRef ArrayA() As String)For i = 0 to ArrayA.Length - 1String temp = ArrayA(i)Int index = RandomInt(Array.Length – 1)ArrayA(i) = ArrayA(index)ArrayA(i) = tempFinish For Loop Finish Sub | |
| **Real code** | Private Sub RandomizeAnswers(ByRef qstnArray() As String)  'Takes in a string array.  'Randomizes each item in the array.  'Takes the current item and swaps it with a random item  For i = 0 To qstnArray.Count - 1  Dim tmp As String = qstnArray(i)  Dim indx As Integer = New Random().Next(qstnArray.Count - i) + i  qstnArray(i) = qstnArray(indx)  qstnArray(indx) = tmp  Next  End Sub | |

|  |  |  |
| --- | --- | --- |
| **Title** | CreateDynamicStudents | **Location**: frmTeacher |
| **Description** | This algorithm will be used to display picture boxes and labels in a defined pattern to the teacher. This algorithm will be used when the teacher wants to search for students in their class. The teacher enters the name they’re looking for into a text box and a function returns an array of the User object containing all the students found which satisfies the teachers search criteria.  The algorithm will loop through each item in the array and create a new CustomPictureBox control, setting the user Id, username and password of the picture box based upon the current item in the studentsList array. The X Location of each picture box is changed upon each iteration and once is 5 picture boxes have been created, the X location is reset and the Y location is incremented.  An event handler is then added so that when the teacher clicks the picture box, they can perform actions like viewing students’ performance.  The image below is an illustration of the end result of the use of this algorithm. Each picture box and label represents the student found that are assigned to the teachers class.  C:\Users\Toshiba\Desktop\Screenshot_2.png | |
| **Pseudo Code** | int xLocation = 20  int yLocation = 20  int amountCreated = 0  int classId = Teacher.ClassID  List(Of User) studentsList = GetStudents(classId, “name”)  For i = Loop From 0 To studentList.count - 1  Declare picbox = New CustomPictureBox  picbox .UserID = studentsList(i).QuizID  picbox .Name = studentsist(i).Firstname & studentsList(i).Lastname  picbox .Location = (xLocation, yLocation)  picbox .Size = (167, 186)  picbox.Image = studentsList(i).Image  CreateDynamicLabels(picbox.Name, xLocation, yLocation)  amountCreated += 1  xLocation += 220  If amountCreated == 5 Then  xLocation = 20  yLocation += 220  amountCreated = 0  Finish If  AddHandler picBox.Click To ShowQuizResults()  AddControl(picBox)  End Loop | |
| **Real code** | Dim studentList As List(Of User) = Teacher.GetStudents(MyUser.ClassID, txtSearchField.Text)  CreateDynamicStudents(studentList)  Private Sub CreateDynamicStudents(students As List(Of User))  ClearControls() 'Clears all the pre-exiting controls in the panel first.  Invoke(Sub() btnSearch.Enabled = False)  'Initial locations  Dim xLocation As Integer = 20  Dim yLocation As Integer = 20  Dim amountCreated As Integer = 0  For i = 0 To students.Count - 1  Dim picBox As New CustomPictureBox  'Instanciate the CustomPictureBox object.  'Defines all its properties.  picBox.Name = students(i).Firstname & " " & students(i).Lastname  picBox.UserID = students(i).UserID  picBox.Username = students(i).Username  picBox.Password = students(i).Password  picBox.Size = New Size(167, 186)  picBox.Location = New Point(xLocation, yLocation)  picBox.BorderStyle = BorderStyle.FixedSingle  picBox.SizeMode = PictureBoxSizeMode.Zoom  picBox.Cursor = Cursors.Hand  picBox.Image = Image.FromStream(students(i).Image.ToStream())  CreateDynamicLabels(picBox.Name, xLocation, yLocation)  'Creates the label of the student the picturebox represents underneath the picturebox  amountCreated += 1  xLocation += 200  'Checks the amount created. If it's 5. The it changes it increments the yLocation.  'Which moves it down, and resets the xLocation to the left of the Parent Container.  If amountCreated = 5 Then  xLocation = 20  yLocation += 220  amountCreated = 0  End If  AddHandler picBox.Click, AddressOf StudentPicture\_Clicked  Invoke(Sub() Me.studentsView.Controls.Add(picBox))  'Adds the object to the Panel1.Controls.  'We also add a click event handler.  'We're invoking the methods because we're running this on a seperate thread'  'And to make a safetly call we need to Invoke the call using a delegate sub.  'Or we'd get a CrossThreadCallExcpetion thrown.  Next  Invoke(Sub() Me.studentsView.Focus())  Invoke(Sub() btnSearch.Enabled = True)  End Sub | |

|  |  |  |
| --- | --- | --- |
| **Title** | CreateDynamicLabels | **Location**: frmTeacher |
| **Description** | This algorithm creates an instance of the label object, then sets the text of the label to the value of the name variable passed down the parameters. The method calls a function that will calculate the X Location of where the label should be positioned. This algorithm is for creating the label for the name of the student the picture box above represents. | |
| **Pseudo Code** | Var lblStudentName = New Label(.Text = name, .Size = Auto)  xLocation += GetEstimatex(lblStudentName)  yLocation += 190  lblStudentName.Location = (xLocation, yLocation)  Me.Controls.Add(lblStudentName) | |
| **Real code** | Private Sub CreateDynamicLabels(name As String, xLocation As Integer, yLocation As Integer)  'Creates the label of the student  Dim lblStudentName As New Label With {.Text = name, .AutoSize = True}  xLocation += GetEstimateX(lblStudentName)  yLocation += 190  lblStudentName.Location = New Point(xLocation, yLocation)  Invoke(Sub() Me.Panel1.Controls.Add(lblStudentName))  End Sub | |

|  |  |  |
| --- | --- | --- |
| **Title** | GetEstimateX & MeasureLabel | **Location**: frmTeacher |
| **Description** | I’ve used this algorithm to estimate how much the X Location of the label containing the name of the student should be incremented by in order to find a perfect midpoint location relative to the length of the label. To find the midpoint, the algorithm calculates the coordinates of the bottom left corner of the picture box (see image on the right) by incrementing the Y Location by the width of the picture box (190). The label needs to be moved to the right in order for it to be in the middle relative to the name of the student. This is done by measuring the width of the label then estimating how much it should be incremented by in order to fit a perfect midpoint. The perfect midpoint would be “2” in the picture above.  The function then returns the estimated X Location. | |
| **Pseudo Code** | Int amount  Select Case measurelabel(lbl).Width  Case <= 30 : Amount = 60  Case <= 40 : amount = 55  Case <= 50 : amount = 50  Case <= 65 : amount = 40  Case <= 70 : amount = 35  Case <= 80 : amount = 30  Case <= 80 : amount = 25  Case >= 101 : amount 5  Finish Case  Return amount | |
| **Real code** | Private Function GetEstimateX(lbl As Label) As Integer  'This provides an estimate of the xLocation of the label.  'In order to place the label in the right place.  Dim amount As Integer  'Measures the labels width  Select Case MeasureLabel(lbl).Width  Case Is <= 30 : amount = 60  Case Is <= 40 : amount = 55  Case Is <= 50 : amount = 50  Case Is <= 60 : amount = 45  Case Is <= 65 : amount = 40  Case Is <= 70 : amount = 35  Case Is <= 80 : amount = 30  Case Is <= 100 : amount = 25  Case Is >= 101 : amount = 5  End Select  'amount is just a guess of how much to the right the xLocation should be moved to  Return amount  End Function  Private Function MeasureLabel(lbl As Label) As SizeF  'Measures the label and returns the size  Return Me.CreateGraphics.MeasureString(lbl.Text, lbl.Font)  End Function | |

|  |  |  |
| --- | --- | --- |
| **Title** | GetMidPoint | **Location**: frmTeacher |
| **Description** | This algorithm is used to find the midpoint location of the bottom side of the picture box object created at runtime using the CreateDynamicStudents algorithm. Using this algorithm, the algorithm is able to calculate the location of “2” (see image on the right) by finding the location of “1” and “3” then using a mathematical formula to calculate the midpoint.  This algorithm worked but didn’t find the midpoint relative to the name of the label. At last I didn’t use this algorithm because the GetEstimateX algorithm was a more useful solution. | |
| **Pseudo Code** | Int x1 = p.xLocation  Int y1 = p.yLocation + p.Height  Int x2 = p.xLocation + p.Width  Int y2 = p.yLocation + p.Height  Var midpoint = ((x2 + x1) / 2), ((y2 + y1)/2)  Return midpoint | |
| **Real code** | Private Function GetMidPoint(p As PictureBox) As Point  'The size of the picturebox == p.Size = New Size(167, 186) (Width, Height)  'So to get the co-ordinate of the bottom left, we keep the xlocation the same  'to find the yLocation, we move down by the size of the Height of the picutrebox  'That gives us the x & y co-ordinates for the bottom left of the picturebox  'We use the same concept to find the x & y co-ordinates for the bottom right co-ordinate  'Gets the midpoint location of the bottom end of a picturebox.  Dim x1 As Integer = (p.Location.X)  Dim y1 As Integer = (p.Location.Y + p.Size.Height)  Dim x2 As Integer = (p.Location.X + p.Size.Width)  Dim y2 As Integer = (p.Location.Y + p.Size.Height)  Dim bottomLeft As New Point(x1, x2)  Dim bottomRight As New Point(y1, y2)  'Calculates the midpoint then returns it  Dim midPoint As New Point(CInt((x2 + x1) / 2), CInt((y2 + y1) / 2))  Return midPoint  End Function | |

|  |  |  |
| --- | --- | --- |
| **Title** | QuickSort | **Location**: TeacherSerivce->QuickSort |
| **Description** | This quick sort algorithms sorts an array of the custom object StudentQuizInfo. The algorithm sorts based upon the value of the ‘average’ variable.  Note: ‘arrayOfItem’ is an array of type StudentQuizInfo.  This quick sort class has a method called Sort() which is public and thus can be accessed outside of the scope of the class. The sort method is then called when a list of items need to be sorted.  The algorithm first finds the initial pivot so it knows how to divide the items into sub lists which can then be sorted recursively.  The GetPivotPositon is where the moving of items takes place. The algorithm selects a pivot, and sorts each item on the left and right of the pivot.  After it has finished moving items, it then returns the index of the new pivot back to the SortList method. The SortList method then makes a recursive call that sorts each item in the sub list of the new pivot index returned.  The laws of computer science state that a recursion needs a conditional statement in order to exit the recursive loop. Therefore the recursive loop only stops when all the items in the array have been sorted. | |
| **Pseudo Code** | Class QuickSort  Begin Public Sub Sort(arrayOfItems)  if Length(arrayOfItems) <= 1  return arrayOfItems  else  var sortedList = SortList(0, Length(arrayOfItems), ArrayOfItems)  Return sortedList  Finish if  Finish Sub  Begin Function SortList(int lo, int hi, arrayOfItems) Returns ->SortedList  If hi > lo then  Int divider = GetPivotPosition(lo, hi, arrayOfItems)  SortList(lo, divider - 1, array)  SortList(divider + 1, hi, array)  Finish If  Return arrayOfItems  Finish Function  Function GetPivotPosition(int lo, int hi, arrayOfItems)  Int x = lo  Int y = hi - 1  Int pivot = arrayOfItems(hi).Avergae  Start Do Loop  While arrayOfItems(i).Average <= Pivot And i < hi  i = i + 1  Finish While  While arrayOfItems(i).Average <= Pivot And j > lo  j = j + 1  Finish While  If array(i).Average > pivot then  Swap(I, j, arrayOfItems)  End If  Finish Do Loop  Return i  Finish Function  Begin Sub Swap(I, j , ByRef arrayOfItems)  Var selecteditem = arrayOfItems(i)  ArrayOfItems(i) = arrayOfItems(j)  ArrayOfItems(j) = selectedStudent  Finish Sub  Finish Class | |
| **Real code** | Private Class Quicksort  Public Shared Function Sort(arrayList As List(Of StudentQuizInfo)) As List(Of StudentQuizInfo)  'If list is empty of has 1 item then its already sorted  If arrayList.Count <= 1 Then  Return arrayList  Else  'Makes a call to the recursive sort algorithm  Dim sortedList As List(Of StudentQuizInfo) = SortList(0, arrayList.Count - 1, arrayList)  Return sortedList  End If  End Function  Private Shared Function SortList(lowIndex As Integer, hiIndex As Integer, arrayList As List(Of StudentQuizInfo)) As List(Of StudentQuizInfo)  If hiIndex > lowIndex Then  Dim initialPivot As Integer = GetPivotPosition(lowIndex, hiIndex, arrayList)  SortList(lowIndex, initialPivot - 1, arrayList)  SortList(initialPivot + 1, hiIndex, arrayList)  End If  Return arrayList 'Returns the sorted list at the end of the recusive loop  End Function  Private Shared Function GetPivotPosition(lowIndex As Integer, hiIndex As Integer, arrayList As List(Of StudentQuizInfo)) As Integer  Dim x As Integer = lowIndex  'Uses the first index as the starting point  Dim y As Integer = hiIndex - 1  'The pivot is going to be used to compare each item in the list  Dim pivot As Integer = arrayList(hiIndex).Average  Do  'Compares each item in the left sub-list that is greater than the pivot  While arrayList(x).Average <= pivot AndAlso x < hiIndex  x += 1  End While  'Compares each item in the right sub-list that is less than the pivot  While arrayList(y).Average >= pivot AndAlso y > lowIndex  y -= 1  End While  If x < y Then Swap(x, y, arrayList) ' Swaps X and Y  Loop While x < y 'Keeps looping until Y is greater than X  ' Swaps the data from the right to the new position in the list  If arrayList(x).Average > pivot Then Swap(x, hiIndex, arrayList) 'Exchanges the data  Return x ' Returns the position of the pivot  End Function  Private Shared Sub Swap(indexFrom As Integer, indexTo As Integer, ByRef arrayList As List(Of StudentQuizInfo))  Dim selectedStudent As StudentQuizInfo = arrayList(indexFrom) 'selected student to swap  arrayList(indexFrom) = arrayList(indexTo)  arrayList(indexTo) = selectedStudent  'Swaps the current with the specified index  End Sub  End Class | |

|  |  |  |
| --- | --- | --- |
| **Title** | QuestionGenerator - GenerateQuestions | **Location**: QuestionGenerator |
| **Description** | Inside the QuestionGenerator class there is one public function called “GenerateQuestions” which is the only public method visible outside the scope of the class. When called, the method will call the corresponding nested classes to generate a specific type of questions, then return the generated question and add it to an array of generated questions which is returned to the caller. | |
| **Pseudo Code** | Begin Function (qTopic, grade, amountRequested, multiChoice)  Var difficultyLevel = GetDifficulty(grade)  Var questionList = New List<question>  For loop = 1 to amountRequested  Var question = new Question  Select Case qTopic  Topic.Algebra  Question = new AlgebraQuestion().GenerateQuestion()  Topic.Numbers  Question = new NumbersQuestion().GenerateQuestion(difficultyLevel)  Topic.Shape  Question = new ShapesQuestion().GenerateQuestion()  Topic.HanldingData  Question = new HandlingDataQuestion().GenerateQuestion(difficultyLevel)  Finish Case  question.Grade = grade  question.isMultiChoice= multiChoice  questionList.Add(question)  Finish for loop  Return questionList  Finish function | |
| **Real code** | Public Shared Function GenerateQuestions(qTopic As Topic, grade As Char, amountRequested As Integer, multiChoice As Boolean) As List(Of Question)    Dim difficultyLevel As Difficulty = GetDifficulty(grade)  Dim questionList As New List(Of Question)  For i = 1 To amountRequested  Dim question As New Question  Select Case qTopic  Case Topic.Algebra  question = New AlgebraQuestion().GenerateQuestion()  Case Topic.Numbers  question = New NumbersQuestion().GenerateQuestion(difficultyLevel)  Case Topic.Shapes  question = New ShapesQuestion().GenerateQuestion()  Case Topic.HandlingData  question = New HandlingDataQuestion().GenerateQuestion(difficultyLevel)  End Select  question.Grade = grade  question.IsMultiChoice = multiChoice  questionList.Add(question)  Next  Return questionList  End Function | |

|  |  |  |
| --- | --- | --- |
| **Title** | ShapesQuestion- GenerateShapesQuestion | QuestionGenerator->ShapesQuestion |
| **Description** | This algorithm is used to generate random shape questions.  Three structures are defined inside the scope of the ShapeQuestion class. One of the structure is TPoint which represents the location where text will be displayed on a drawn image. The variable format indicates whether the text will be printed sideways, landscape or portrait.  The shapes structure is shape and question type.  This algorithm can generate three types of question for square, rectangle and triangle shapes.  I will only show the triangle algorithm as it’s the most complex (see the next algorithm).  First the algorithm must choose what type of shape question to generate. This happens by the algorithm selecting a random shape from a shape array, generating a question according to the shape picked then formulating a string representation of the question.  Assuming the chosen shape was a triangle the algorithm on the next page describes the procedures that follows. | |
| **Pseudo Code** | Private var thisQuestion = new Question()  Structure TPoint  Var txt  Int xLocation, yLocation  Var StringFormatFlags format  Begin Sub GenerateQuestion()  var RandomShape = GetRandomShape  var QuesType = GetRandomQuestionType  Select Case RandomShape  Shape.Triangle  thisQuestion.Image = DrawTriangleQuestion  Shape.Square  thisQuestion.Image = DrawSquareQuestion  Shape.Rectangle  thisQuestion.Image = DrawRectQuestion  Finish case  var words[] = “Work out the” , “Calculate the” “What is the”  var randomWord = words(R.Next(0, words.Length – 1)  thisQuestion.QuestionString = randomWord & “ “ QuesType “ of this “ & RandomShape.ToString()  Return thisQuestion Finish Sub | |
| **Real code** | Private Structure TPoint  Dim Text As String  Dim xLocation As Integer  Dim yLocation As Integer  Dim Format As StringFormatFlags  End Structure  Dim Shapes() As Shape = {Shape.Rectangle, Shape.Square, Shape.Triangle}  Private QTypes() As QuestionType = {QuestionType.Area, QuestionType.Perimeter}  Private Measurements() As String = {"cm", "m"}  Private thisQuestion As New Question With {.Topic = "Shapes"}  Public Function GenerateQuestion() As Question  ReDim thisQuestion.FalseAnswers(2)  Dim rndShape As Shape = Shapes(R.Next(0, Shapes.Length)) 'Gets a random shape from an array of shapes.  Dim questionType As QuestionType = QTypes(R.Next(0, QTypes.Length)) 'Gets random question type. Perimeter or Area  Dim m As String = Measurements(R.Next(0, Measurements.Length)) ' cm or m  Select Case rndShape  ' It will also solve the question, and the false answers for the questions too  Case Shape.Rectangle  thisQuestion.Image = Image.FromStream(DrawRectangle(questionType, m)).ToBytes()  Case Shape.Square  thisQuestion.Image = Image.FromStream(DrawSquare(questionType, m)).ToBytes()  Case Shape.Triangle  thisQuestion.Image = Image.FromStream(DrawTriangle(questionType, m)).ToBytes()  End Select  Dim a() As String = {"Work out the", "What is the", "Calculate the"}  Dim b As String = a(R.Next(0, a.Length)) ' Picks random word from the a() array  thisQuestion.QuestionString = String.Format("{0} {1} of this {2}", b, questionType.ToString.ToLower, rndShape.ToString)  'E.g {Work out the} {area} of this {Triangle}  Return thisQuestion  End Function | |

|  |  |  |
| --- | --- | --- |
| **Title** | ShapesQuestion – DrawTriangleQuestion | **Location**: QuestionGenerator  ->ShapesQuestion |
| **Description** | This algorithm generates a triangle question. Three variables are declared, the hypotenuse, adjacent and opposite. These are the lengths of each side of the triangle.  A loop is used to generate random numbers for each of the side of the triangle. Mathematical laws state that the hypotenuse must be greater than the adjacent, which in turn is greater than the opposite. The loop will only exit if these conditions are fulfilled.  Three labels of TPoint are declared. The Text property holds the value of the triangle side along with the unit. The location of where the label should be drawn is then predefined.  An instance of the Bitmap object is created of the size (260, 260), this is called Canvas. This is where the algorithm will draw the triangle image on.  An array of 3 coordinates is then defined. Each coordinate defines the position of each of the 3 corners of the triangle.  The algorithm then uses the graphics classes in vb.net and the array of coordinates to draw the triangle onto the canvas. After the triangle is drawn, the labels lblHypotenouse, lblAdjacent and lblOpposite are then drawn onto the sides of the triangle on the canvas.  The location of the triangle and labels always stays the same. However it’s the values of the labels that always change. The value could be “4cm” or “91m^2”  After the algorithm has finished generating the image, it calls a method that calculates the answer to the question string. The next page shows the algorithm that calculates the answers. | |
| **Pseudo Code** | var hypot, adjac, oppos  char unit = Random{“m” or “c”}  Do  Hypot = RandomNumFrom(5 To 20)  Adjac = RandomNumFrom(3 To 15)  Oppos = RandomNumFrom(2 To 10)  Do until hypot > adjac & oppos < adjac  var lblHypotenouse = {.Text = hypot & unit, .xLocation = 140, .yLocation = 100}  var lblAdjacent = {.Text = hypot & unit, .xLocation = 140, .yLocation = 100}  var lblOpposite = {.Text = hypot & unit, .xLocation = 140, .yLocation = 100}  var lblCollection = {lblHypotenouse, lblAdjacent, lblOpposite}  var canvas = size(260, 260)  var points(2,2) = {(58, 29), (58, 212), (214, 212)}  Graphics.draw(points)  for each label in lblCollection  Draw each label onto triangle  End For Each  Canvas.SaveImg()  SolveTriangleQuestion(questionType, hypot, adjac, opposite)  Return Canvas.Image() | |
| **Real**  **code** | Private Function DrawTriangle(type As QuestionType, unit As String) As MemoryStream  Dim hypot As Integer = 0  Dim adjac As Integer = 0  Dim oppos As Integer = 0  ' These are the numeric values that will be plotted on each side of the triangle  Do  hypot = R.Next(5, 20)  adjac = R.Next(3, 15)  oppos = R.Next(2, 10)  Loop Until hypot > adjac And oppos < adjac  ' This is to make sure the Adjacent and Opposite sides are never greater than  ' the hypotenouse. Therefore generating a more realistic question  Dim lblHypotenouse As New TPoint With {.Text = hypot & unit, .xLocation = 140, .yLocation = 100}  Dim lblAdjacent As New TPoint With {.Text = adjac & unit, .xLocation = 20, .yLocation = 115}  Dim lblOpposite As New TPoint With {.Text = oppos & unit, .xLocation = 115, .yLocation = 216}  Dim coordinateCollection As New Collection From {lblHypotenouse, lblAdjacent, lblOpposite}  'add to collection so I can take advantage of the for each loop  Dim canvas As New Bitmap(260, 260)  Dim myFont As New Font("Arial", 10)  Dim points(2) As Point ' Each point of the triangle I am yet to draw  points(0) = New Point(58, 29)  points(1) = New Point(58, 212)  points(2) = New Point(214, 212)  Using g As Graphics = Graphics.FromImage(canvas) : g.Clear(Color.White)  g.DrawPolygon(Pens.Red, points) ' Draws Triangle  For Each point As TPoint In coordinateCollection  g.DrawString(point.Text, myFont, Brushes.Green, New Point(point.xLocation, point.yLocation))  'Draws labels on the triangle  Next  g.Save()  End Using  SolveTriangleQuestion(type, hypot, oppos, adjac, unit)  'We then save the canvas image to a memory stream  'and then return the content of the memory stream  Dim memoryStrm As New MemoryStream()  canvas.Save(memoryStrm, ImageFormat.Jpeg)  DisposeObjects(canvas, myFont)  Return memoryStrm  End Function | |

|  |  |  |
| --- | --- | --- |
| **Title** | ShapesQuestion - SolveTriangleQuestion | **Location**: QuestionGenerator  ->ShapesQuestion |
| **Description** | This function is called by the DrawTriangleQuestion method and it takes in 5 arguments which it uses to calculate the answer to the generate question string.  It takes the questionType, all the sides of the triangle, and the unit of the triangle (e.g. “cm” or “m”).  It calculates the answers to the question, along with false answers for multiple questions support.  It assigns the false answers to an array then returns the correct answer. | |
| **Pseudo Code** | If questiontype == area  thisQuestion.answer = ((opposite \* adjacent) / 2) & unit & ”²”.  thisQuestion.FalseAnwers(0) = hypo + opposite + adjacent) & unit  thisQuestion.FalsweAnswers(1) = (opposite \* adjacent) & unit “²”  Else if questionType == perimeter  thisQuestion.answer = ((opposite + adjacent + opposite ) / 2) & unit  thisQuestion.FalsweAnswers(0) = ((opposite \* adjacent) / 2) & unit & ”²”.  thisQuestion.FalseAnwers(1) = (hypo + opposite) & unit  finish if  thisQuestion.FalseAnswers(2) = RandomNum(opposite, hypo) & unit | |
| **Real code** | Private Sub SolveTriangleQuestion(type As QuestionType, hypo As Integer, opposite As Integer, adjacent As Integer, unit As String)  If type = QuestionType.Area Then  thisQuestion.Answer = ((opposite \* adjacent) / 2) & unit & "²" ' Calculates the area  thisQuestion.FalseAnswers(0) = (hypo + opposite + adjacent) & unit ' Confuses student by calculating perimeter  thisQuestion.FalseAnswers(1) = (opposite \* adjacent) & unit & "²"  ElseIf type = QuestionType.Perimeter Then  thisQuestion.Answer = (hypo + opposite + adjacent) & unit ' Calculates the perimeter  thisQuestion.FalseAnswers(0) = ((opposite \* adjacent) / 2) & unit & "²" ' Confuses student by calculating area  thisQuestion.FalseAnswers(1) = hypo + opposite & unit  End If  thisQuestion.FalseAnswers(2) = R.Next(opposite, hypo) & unit  End Sub | |

|  |  |  |
| --- | --- | --- |
| **Title** | ToBytes | **Location**: EntityBase |
| **Description** | This method is an extension function to the Image object/data type. This function allows me to call ToStream on any variable that is an instance of the Image object/data type.  I will use this when converting an Image to an array of bytes which can then be stored in the SQL database.  This function is useful because it will return a default image if the image its converting is empty, therefore is a more efficient than using a TryCatch in preventing errors  The function can be called like imgData.ToBytes()  The use of this function takes full advantage of the features in VB.NET as I am adding custom methods to extend the functionality of primitive data types. | |
| **Pseudo Code** | <Extension()>  Begin Function ToBytes (Image img) this->Function Returns Bytes[]  var MemStream = New Memory Stream  If img == Is Empty Then  DefaultIMG. SaveRaw to MemStream  Else  Img.SaveRaw To MemStream  Finish if  Return MemStream.GetBytes()  Finish Function | |
| **Real code** | <Extension()>  Public Function ToBytes(img As Image) As Byte()  'Converts the Image data/object to an array of bytes.  Dim ms As New MemoryStream  If img Is Nothing Then  My.Resources.DefaultImg.Save(ms, My.Resources.DefaultImg.RawFormat)  Else  img.Save(ms, img.RawFormat)  End If  Return ms.GetBuffer()  End Function | |

|  |  |  |
| --- | --- | --- |
| **Title** | ToStream | **Location**: EntityBase |
| **Description** | This method is an extension function to the bytes array datatype. This function allows me to call ToStream on any variable that is an instance of the byte array data type.  I will use this when converting a byte array which contains raw image data into a memory stream/image. I can then present the image to the user visually.  I could use the function like  Dim img[ ] As Byte = GetUserImg  PicBox1.Image = Image.FromStream(img.ToStream())  This is a more natural way of calling the function than  PicBox1.Image = Image.FromSteam(ToStream(img)) | |
| **Pseudo Code** | <Extension()>  Begin Function ToSteam(bytes[] data) this->Function Returns MemoryStream  If data == Is Empty Then  Return MemoryStream(DefaultImg)  Else  Return MemoryStream Insert New (data)  Finish if  Finish Function | |
| **Real code** | 'This is a custom Extension, this extends the byte datatype and picture box. So I can call .ToStream()  'on any variable of the datatype Byte() same thing with picture box, I can call it like PicBox1.Image.ToBytes()  'It's just my preference, and makes calling this function look neater than doing ToBytes(PicBox1.Image)  <Extension()>  Public Function ToStream(value As Byte()) As MemoryStream  'Converts the byte array to a memory stream  If value Is Nothing Then  Return New MemoryStream(My.Resources.DefaultImg.ToBytes())  Else  Return New MemoryStream(value)  End If  End Function | |

Data definition language (DDL)

This is the DDL script to create all of the tables required for my new system.

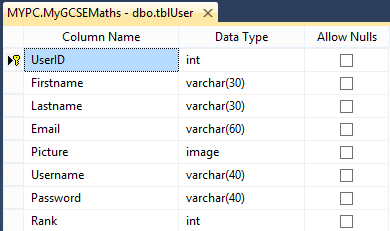
Identity (1, 1) means auto increment by 1.

|  |  |
| --- | --- |
| Create Table tblUser  (  UserID INT NOT NULL PRIMARY KEY,  Firstname Varchar (30) NOT NULL,  Lastname Varchar (30) NOT NULL,  Email Varchar(60) NOT NULL,  Picture Image NULL  Username Varchar(40) NOT NULL,  Password Varchar(40) Not NULL  ) | Create Table tblClass  (  ClassID INT IDENTITY(1,1) NOT NULL,  Block Char(1) NULL,  Room Varchar (5) NULL  ) |
| Create Table tblUserClass (  UserID INT NOT NULL,  ClassID INT NOT NULL  ) | Create Table tblQuestion (  QID INT Identity(1,1) PRIMARY KEY,  Topic Varchar(20) NOT NULL,  Grade Char(1) NOT NULL,  Question Varchar(100) NOT NULL,  Image Image/Byte NULL,  Answer Varchar(50) NOT NULL,  MultiChoice Boolean NOT NULL  ) |
| Create Table tblWrongAnswers (  QID INT Primary Key,  Dummy1 Varchar(50) NOT NULL,  Dummy2 Varchar(50) NOT NULL,  Dummy3 Varchar(50) NOT NULL  ) | Create Table tblQuiz (  QuizID Int NOT NULL,  QID Int NOT NULL,  Composite Key(QuizID, QID)  ) |
| Create Table tblQuizLog (  UserID INT NOT NULL,  QuizID INT NOT NULL,  Mark INT NOT NULL,  Total INT NOT NULL,  DateCompleted DateTime NOT NULL  Composite Key(UserID, QuizID)  ) | Create Table tblQuizTitle (  QuizID Int Primary Key NOT NULL,  Title Varchar(50) NOT NULL  ) |

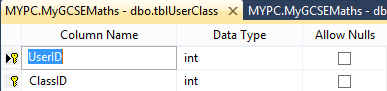
SQL Tables Design view

|  |  |
| --- | --- |
| **Description:** | Design view of all my SQL tables in MSSSQL. |

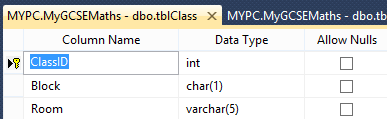
tblUser



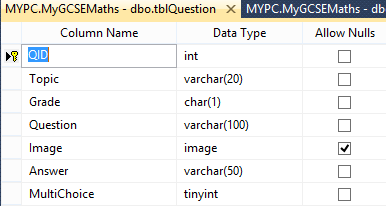
tblUserClass



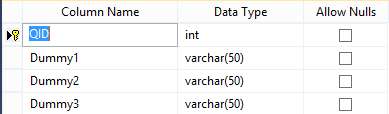
tblClass



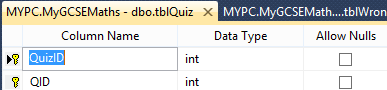
tblQuestion



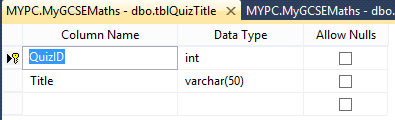
tblWrongAnswers



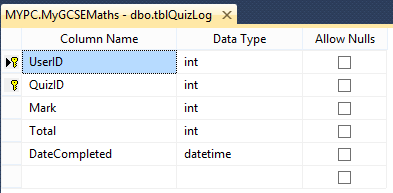
tblQuiz



tblQuizTitle



tblQuizLog



List of descriptive system settings

These are the system requirements needed in order for the system to work.

Minimum system Requirements

* *Windows XP operating system*
* *Microsoft .NET Framework 4.5 or above*
* *40MB of free HDD space*
* *Monitor size greater than 11.6” inches*
* *Coloured Monitor*
* *Need to be connected to the St Francis Xavier local network.*