Physics Quiz

By Tom Pearson

Contents

Analysis

Problem Definition

Client: St Mary`s Catholic High School and Sixth Form Physics Department

Stakeholders:

- Ian Delaney
- Michael Fieldhouse

Contact:

Ian Delaney, Manchester Road, Astley M29 7EE

Background Information

The St Mary`s A Level Physics department is one of the smallest departments in the sixth form, composing of only 2 teachers, Mr Fieldhouse and Mr Delaney. They teach around 20 students each year the AQA Physics specification. Part of this specification is the multiple-choice question section that accounts for around 20% of the student`s grades.

Problem Definition

Currently, both teachers have a word document containing multiple choice questions which are sorted by subject area. When a multiple-choice question is needed, each relevant document is searched and the desired questions are copied and pasted into a word document. This document is then printed and handed out to students. The whole system is very outdated and time consuming.

Interview with Mr Delaney – The Major Stakeholder

Q - 1: How do you currently set the students multiple choice questions?

A **- 1**: Currently the multiple-choice questions are selected from a bank of questions stored on a Microsoft word document and are then set using a printed handout, students write their answers and mark them by hand also. It would be a lot better is a computer could do this for me however as it is very time consuming.

Q - 2: Are there any benefits to the current system?

A **- 2**: There are not many benefits to the current system. I suppose it is good that students are completing them on paper in the same format that they will appear in the real exam is a good thing, however I do not think moving to a computer/app-based system would detract from that really

Q - 3: What features would you like from a new system?

A - 3: It should present the student with a question and a range of correct and incorrect answers. I would also like an automated marking system, that can provide feedback on how students have performed overall and if there are any specific strengths of weaknesses. It would be great if it could present the questions they struggle with more often.

Q - 4: Do you currently analyse the students correct and incorrect answers to provide further insight?

A - 4: Yes, I do analyse the results for further insight, although this can be quite time consuming to do. It would be great if the program could send me their scores via email or some other form of communication.

Q - 5: How is data entered onto the system and how does the system present the analysis?

A - 5: It is done by manually typing in the score that each student got on each question into a spreadsheet, which then generates a table of information using excel formulas. If there is a way that this could be cut out then that would be great!

Q - 6: Is there a database containing the multiple-choice questions in one place that is easily searchable? If not, would this provide a benefit to how you set work?

A - 6: A searchable database would be very helpful

Q - 7: Would it aid students' revision if they could revise from a database of multiple-choice questions?

A - 7: Yes, revising from a database of multiple-choice questions would be a major benefit for the students, largely due to the fact that similar multiple-choice questions come up year or year and some familiarity with these types of questions would be a massive help!

Analysis of Existing System

The current system is composed of three main parts. The data bank of questions in the word documents for the teachers to select from, the print outs they give the students with marking and the results Excel sheets.

1. Questions are selected from the database
   1.1 The teacher selects the desired multiple-choice question topic from the file - See Image 1
   1.2 Upon opening the word document, the file is scanned and the desired questions are extracted using the snipping tool. (See Figure 2)

Figure 1
Word Documents Containing the Multiple-Choice Questions

1. Particles	16/07/2020 13:29	Microsoft Word D...	526 KB
2. Waves	16/07/2020 13:29	Microsoft Word D...	360 KB
3. Mechanics	16/07/2020 13:29	Microsoft Word D...	1,664 KB
4. Materials	16/07/2020 13:29	Microsoft Word D...	628 KB
5. Electricity	16/07/2020 13:29	Microsoft Word D...	895 KB

Figure 2
Example Question from the Word Document

Multiple Choice – Particles

Oxford International January 2019

What is the specific charge of a $^{9}_{4}Be^{2+}$ ion?

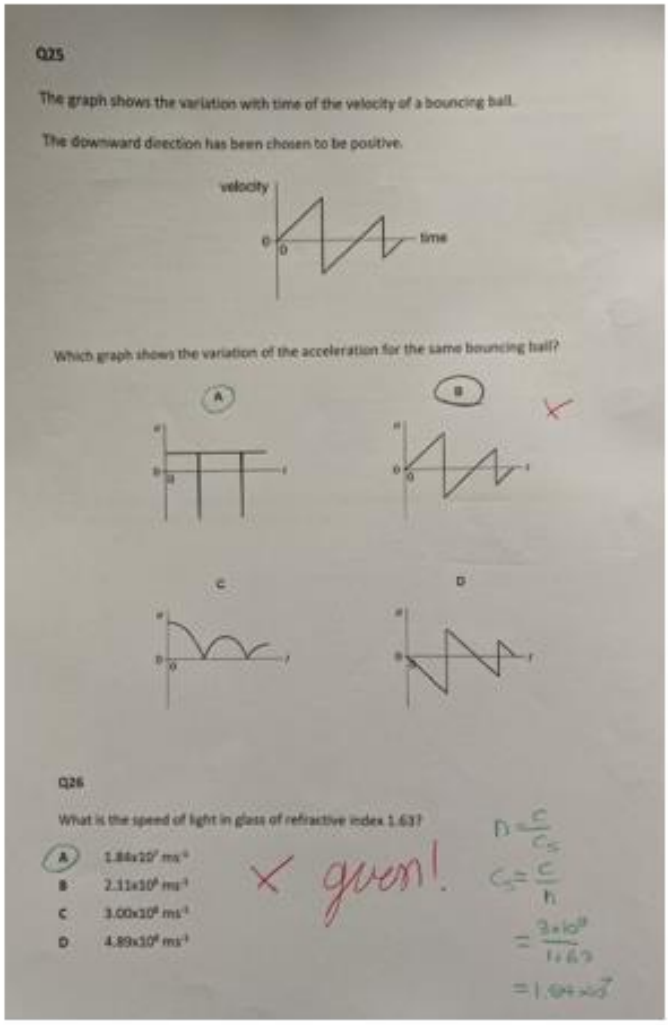
A 2.1×10^{7} C kg^{-1}
B 4.3×10^{7} C kg^{-1}
C 9.6×10^{7} C kg^{-1}
D 2.2×10^{-1} C kg^{-1}

ANS = A

2.The teacher adds the selected questions to a word document
- 2.1. The teacher removes the correct answer
- 2.2. The document is printed and handed over to the students. (See Figure 3)

Figure 3

Questions given to the student containing the students answer, the teacher`s marking and green pen annotation.



Q25

The graph shows the variation with time of the velocity of a bouncing ball.

The downward direction has been chosen to be positive.

velocity

time

Which graph shows the variation of the acceleration for the same bouncing ball?

A

B

C

D

Q26

What is the speed of light in glass of refractive index 1.63?

A 1.84×10^8 ms^{-1}

B 2.11×10^8 ms^{-1}

C 3.00×10^8 ms^{-1}

D 4.89×10^8 ms^{-1}

✗ given!

$n = \frac{c}{c_s}$

$c_s = \frac{c}{n}$

$= \frac{3 \times 10^8}{1.63}$

$= 1.84 \times 10^8$

3. The student answers the sheet and records their answers in the given areas. (See Figure 3)

4. Marking and uploading

4.1 The teacher compares the received answer to the ones stored on the system (See Figure 3)

4.2 If correct, the teacher gives them a mark. If not, the correct answer is selected in green pen so the student knows where they have gone wrong. (See Figure 3)

4.3 The students' scores are uploaded into the Excel document manually. which then calculates the areas the student is struggling in and where extra work is needed. (See Figure 4)

Figure 4

The Spreadsheet where all students' scores are entered.

					Assessment 2 Electricity																
Question	1a	1b	1c	2	3a	3b	4	5	6a	6b	7a	7b	7c	7d	7e	8	9	10
Student One	2	3	2	1	2	2	1	1	2	3	0	3	2	2	0	0	1	1

Analysis of the Spreadsheet

The spreadsheet is a relatively new addition to the process, being created by Mr Delaney less than a year ago. It takes the inputs received and calculates the areas where more work is needed. This way he can plan his lessons around what the students need the most. It also provides insight into areas such as recall or calculations to see which students may have revised (higher recall scores) or which students may just be naturally talented (higher calculations). owever, due to the highly specific nature of this spreadsheet, a new one must be made for each test he presents to the students. For example, the spreadsheet in figure 4 is from an Electricity Assessment and the spreadsheet in Figure 5 is from a November mock. Both have a similar output, but the results table and which questions correspond to which areas are very different.

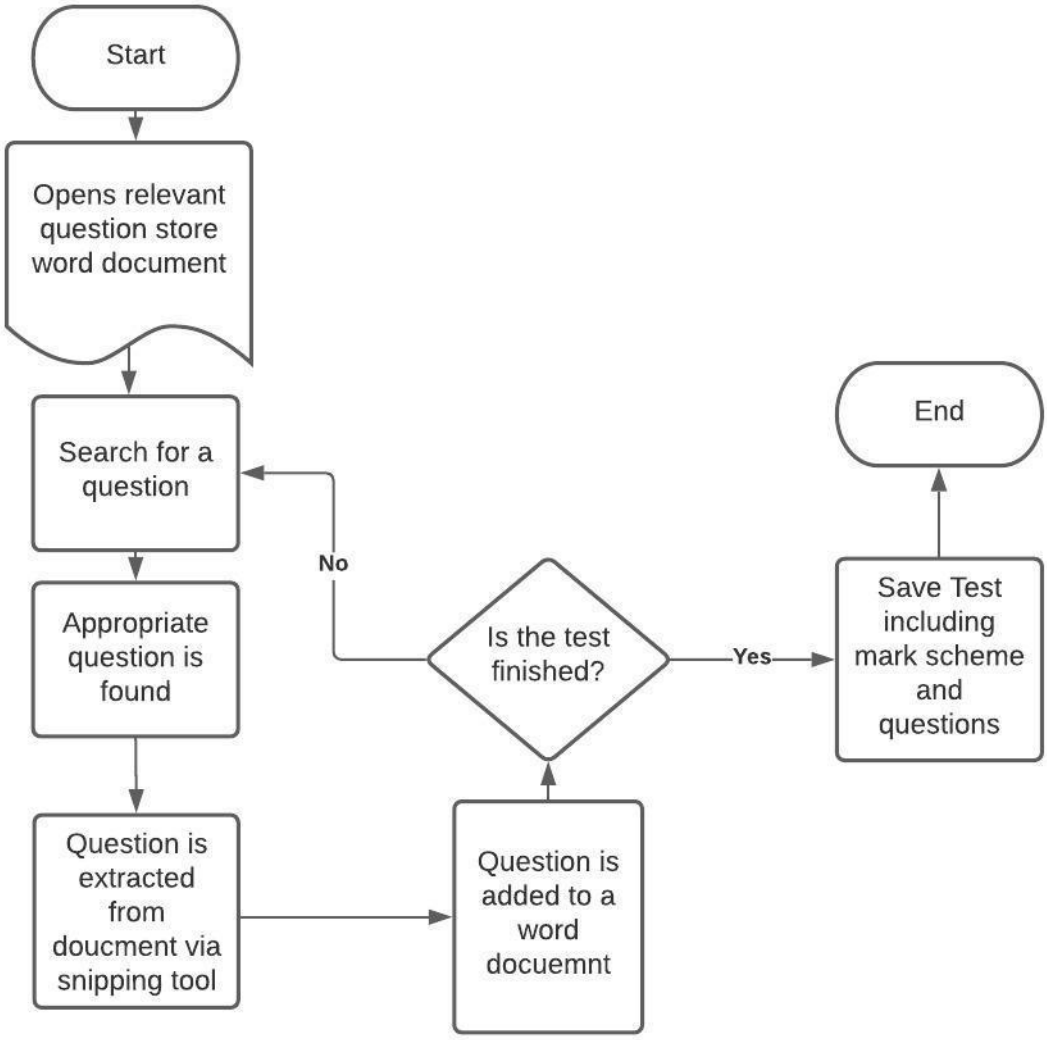
Figure 5
The Spreadsheet where all students' scores are entered.

	November Mock																			
Question	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**2.1**	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**3.1**	**3.2**	**3.3**	**3.4**	**4.1**	**4.2**	**4.3**	**4.4**
Student One	0	0	1	2	2	1	0	1	0	2	0	1	1	2	3	4	3	1	1	3
Student Two	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Student Three	0	0	1	0	0	1	1	0	0	0	0	0	1	1	3	3	0	1	0	0
Student Four	1	1	1	3	1	1	1	1	0	0	1	0	2	2	1	4	3	0	0	1
Student Five	1	1	1	3	0	1	1	1	0	0	0	0	2	2	1	3	3	0	1	0
Student Six																				
Student Seven	1	0	1	3	4	1	1	0	0	2	1	2	1	2	2	3	3	1	1	1
Student Eight	0	0	1	0	0	1	1	1	0	0	1	0	1	2	0	3	3	1	0	3
Student Nine	1	1	1	1	1	0	1	1	1	0	0	0	0	2	3	5	3	1	0	0
Student Ten	0	1	1	2	1	1	1	1	0	0	1	0	1	2	1	5	0	1	0	0
Student Eleven	0	1	0	1	2	1	0	1	1	1	0	1	1	2	1	2	1	1	0	0
Student Twelve	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	4	0	0	0	0
Student Thirteen	0	0	1	0	0	1	1	1	1	0	1	1	0	0	0	3	3	1	0	3
Student Fourteen	0	0	0	1	4	1	1	1	1	0	0	0	2	2	3	3	3	1	1	3
Mean	0.30769	0.46154	0.69231	1.23077	1.23077	0.84615	0.76923	0.69231	0.30769	0.38462	0.38462	0.38462	0.92308	1.61538	1.38462	3.23077	1.92308	0.69231	0.30769	1.07692
Percentage	31	46	69	31	31	85	77	69	31	19	38	19	46	81	46	54	64	69	31	36
Available	1	1	1	4	4	1	1	1	1	2	1	2	2	2	3	6	3	1	1	3

Pupil	Score	Grade	SMT
Student 1	47	A	
Student 2	4	U	
Student 3	14	E	
Student 4	28	D	
Student 5	38	B	
Student 6			
Student 7	48	A	
Student 8	28	D	
Student 9	31	C	
Student 10	24	D	
Student 11	32	C	
Student 12	11	U	
Student 13	33	C	
Student 14	45	A	

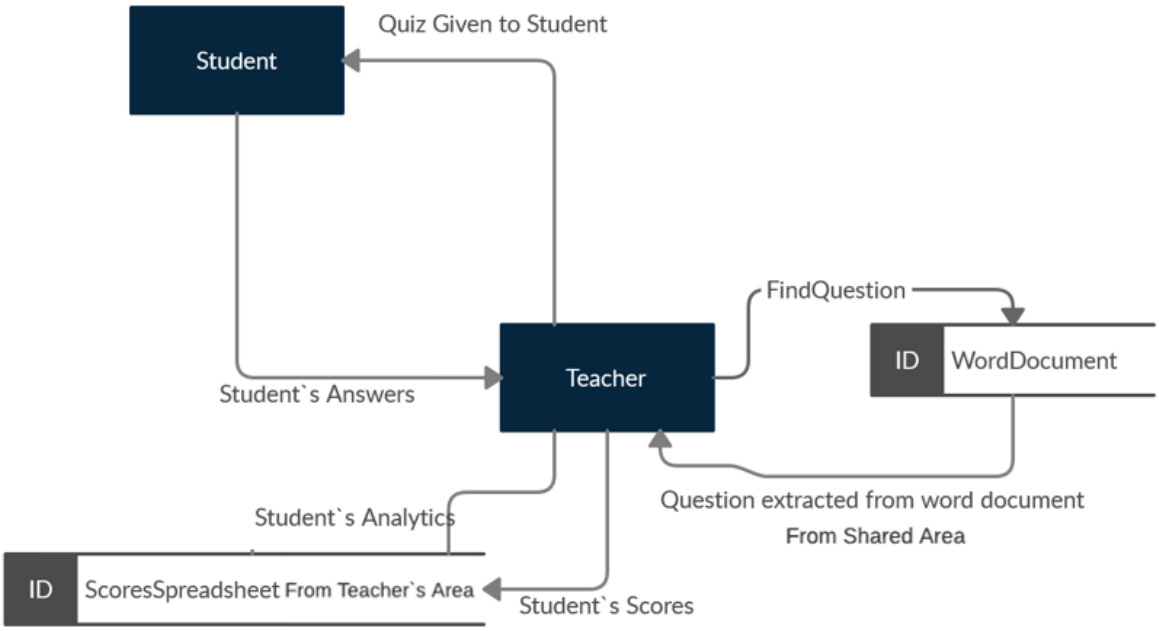
Pupil	Recall	Required Pracs	Calculations	Application	Electricity	Materials	Waves
Student 1	83	63	90	58	73	63	67
Student 2	0	11	0	8	5	8	0
Student 3	50	9	25	25	13	38	0
Student 4	83	40	40	42	25	67	33
Student 5	100	57	55	25	55	58	33
Student 6	0	0	0	0	0	0	0
Student 7	83	77	70	58	68	71	67
Student 8	83	26	50	33	50	29	17
Student 9	67	34	50	50	35	63	33
Student 10	67	37	20	50	15	58	67
Student 11	67	51	35	42	50	42	50
Student 12	83	26	50	33	50	29	17
Student 13	33	46	45	33	68	17	33
Student 14	100	57	70	67	65	63	67

Flow Chart of Current System



This flowchart outlines the aspects of the system the need improving. The most inefficient part is adding the scores to a spreadsheet as this is very time consuming.

Data Flow Diagram

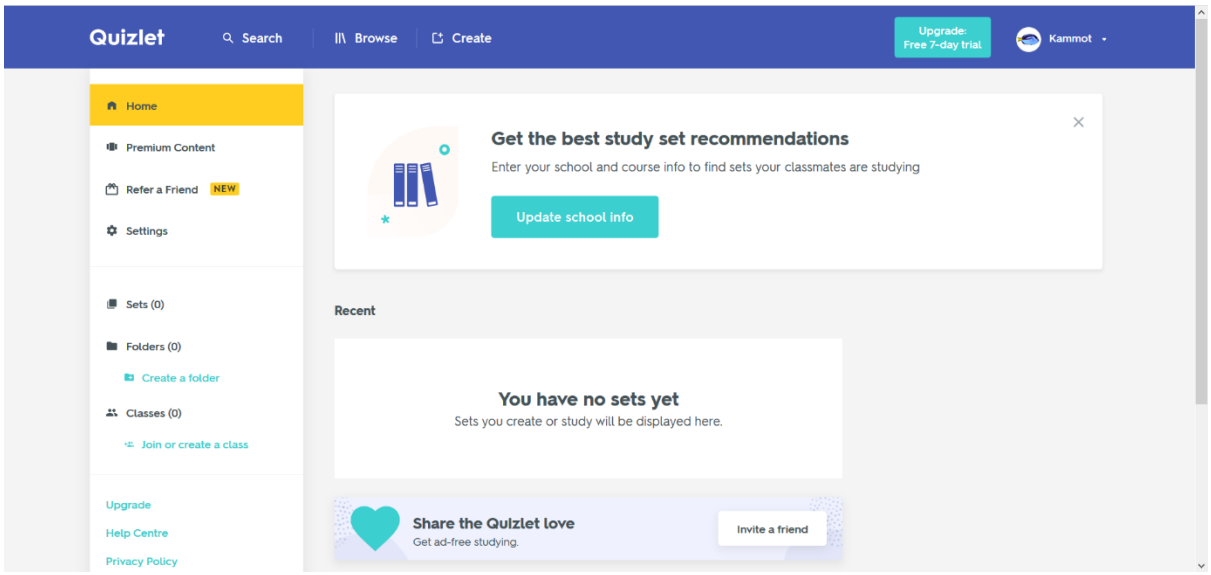


Current Inputs, Processes and Outputs

Inputs	Processes	Outputs
None	Finding Quiz Question	Question Correct Answers False Answers
List of questions, List of Answers, List of Incorrect Answers	Adding it to a word document	Multiple Choice Question Quiz (Question, List of Possible Answers)
Multiple Choice Question Quiz (Question, List of Possible Answers)	Student Completing Quiz	Student`s Answers, Student`s Name, Class
Student`s Answers, Student`s Name, Class	Marking Quiz	Student`s Scores
Student`s Scores, Student`s Name	Spreadsheet	Needed areas of improvement, Student`s Name

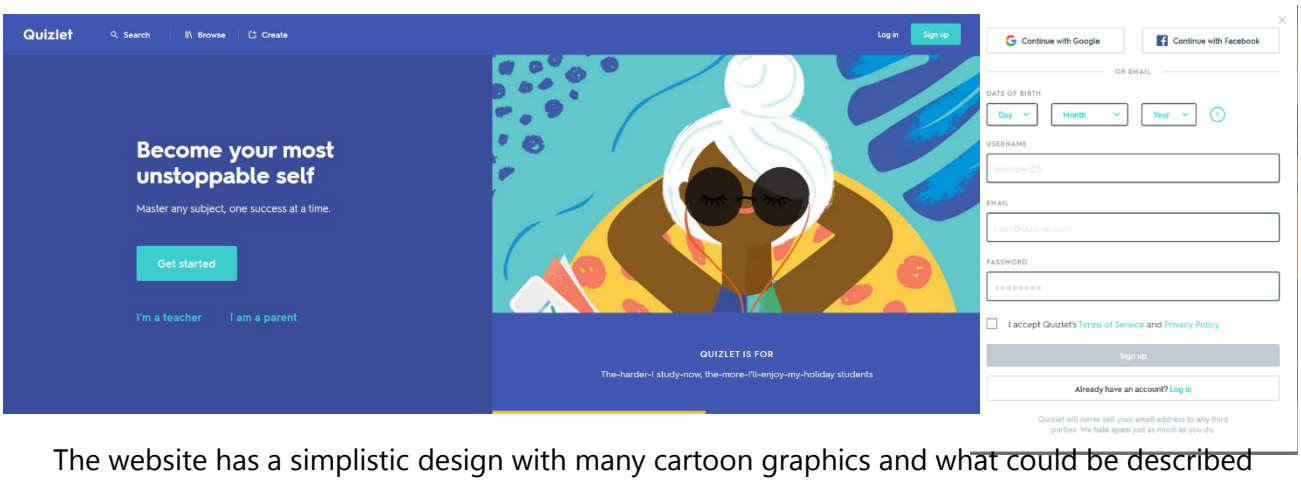
Analysis of Existing Product - Quizlet

Introduction



Quizlet is an existing quiz application, primarily used by students, that allows the user to revise content through a variety of ways, including but not limited to: “Flashcards”, “Learn”, “Write”, “Spell” and “Test”. It is available on their website https://www.quizlet.com/ and their apps, both on iOS and Android. I will be reviewing the program on the website platform. As it has the most functionality in comparison to its mobile counterparts. Even though the site was initially launched back in 2007 they are still updating it and its functionality, with some new features so new that they are still in beta (Live is in beta as of 20/07/2020). This continued support makes it the ideal app to review as it shows that even though the developers have already created a working platform, they are continuing to better it and improve upon their creation.

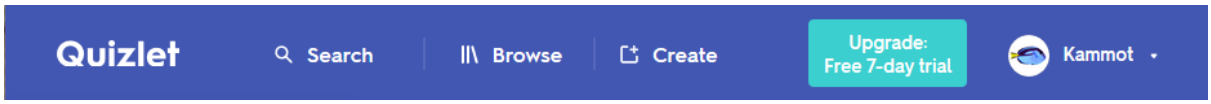
First Use



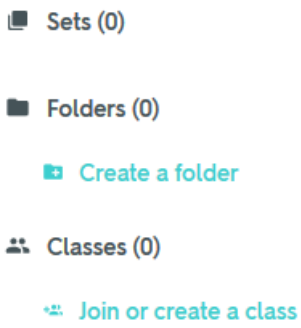
The website has a simplistic design with many cartoon graphics and what could be described as a vector graphic design. If the user isn`t signed in, the website will ask them if they would like to sign up or login. These actions, while recommended are not necessary to use the sites resources. The site also has Google and Facebook plugins, and their accounts can be used to sign up. This is a very useful feature as while it won`t be possible to include in my program, it

is certainly a welcome alternative to creating yet another username and password to be remembered.

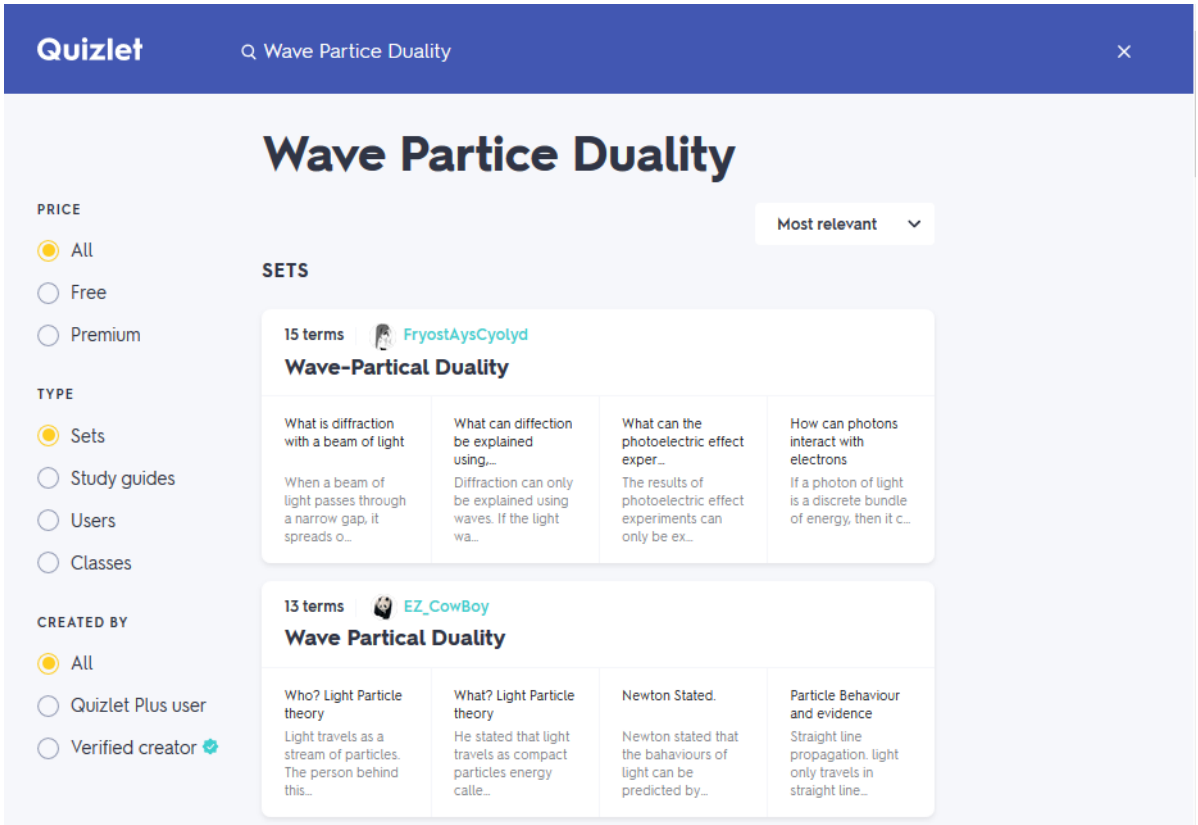
Navigation



At all points on the website the navigation bar is displayed at the top of the screen. This placement is convenient and effective as if at any point the users feels lost, they can quickly get back on track. Sets of quizzes are stored in the user`s private sets but can also be sorted effectively into folders so that the user can order their own sets. That way they can study a specific area of interest even if it is split up into multiple sets.



Finding/Creating a Set

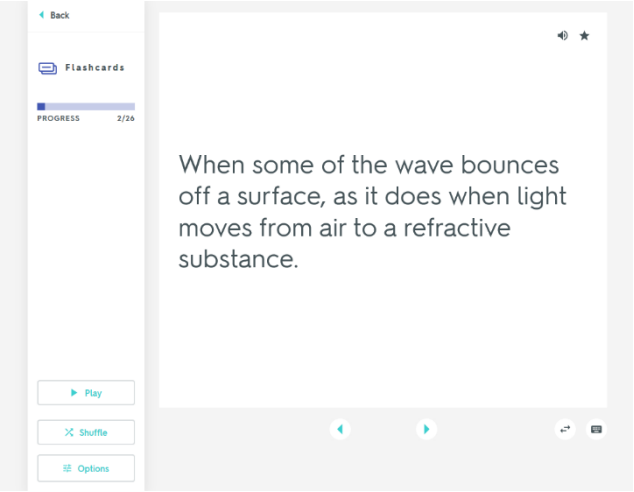


One of the navigation options is the search. This search option will allow the user to search the whole site for any public set that meets their search criteria. It is very useful as that way relevant sets that have already been created can be found and copied, with minimal effort. Allowing the user to spend less time creating a set and more stime studying it.

Study Options

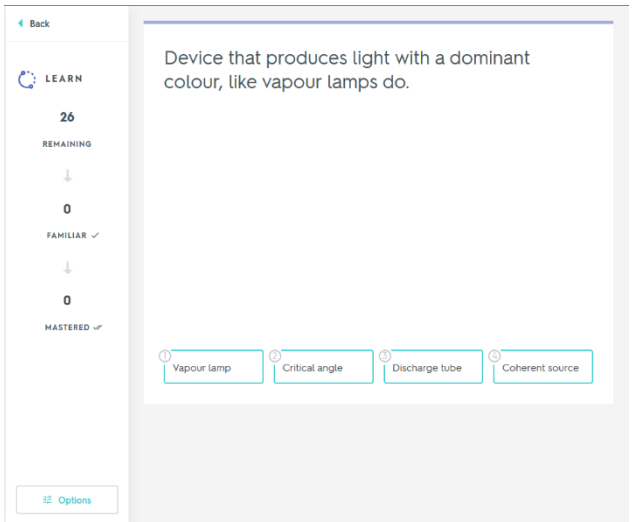
All study options (accept flashcards) filter the questions presented to present to questions answered incorrectly more often than the ones that are answered correctly. However, each one has a different way of presenting each set. They are as follows:

Flashcards



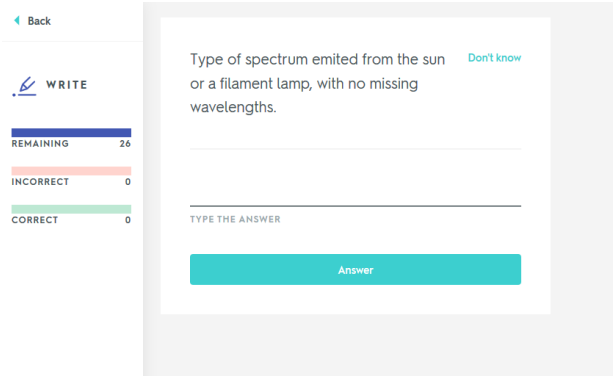
This option converts the sets into flash cards. With the user`s Term as one side and the Explanation on the other side. It makes it easy to study unknown terms without simply guessing at examples.

Learn



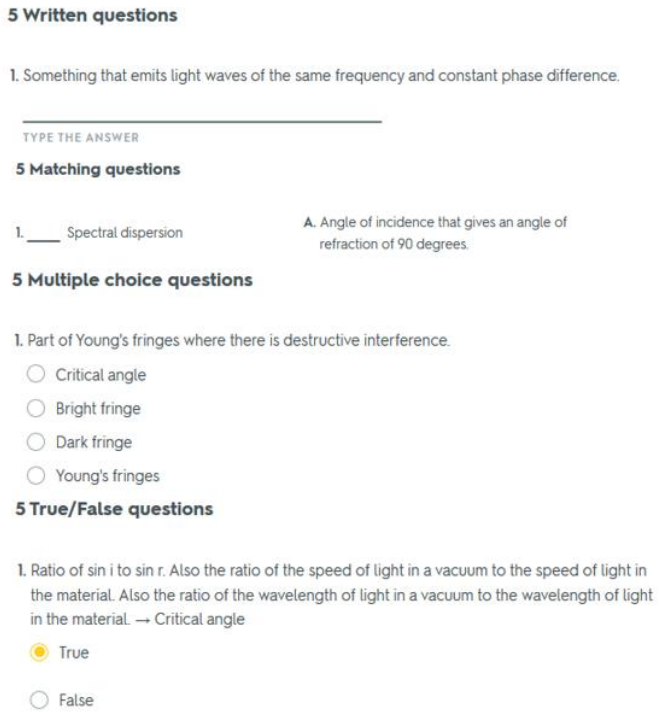
Learn presents the user with their term and a variety of explanations, consisting of the real explanation and three others from other terms in the set. This gives the user a number of options to chose from and works for the majority of the time to provide convincing alternative options, however, that is not always the case and the correct option can sometimes be clear. This problem can be solved by providing alternative options to choose from when programming a new set into the application. While it wouldn`t be reasonable for an application like Quizlet to implement, I will include it in my program to ensure that the user has the best chance of understanding the question properly.

Write



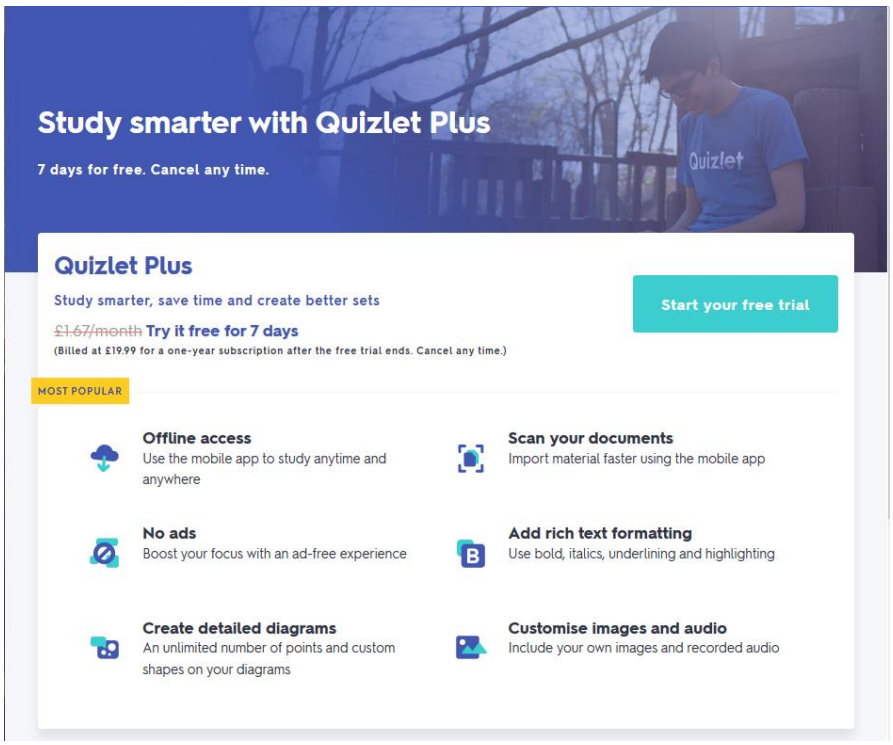
The write option is a traditional approach to answering a question where the user writes out their response to the term. This is a great option when the answer is only a limited number of words and the user also has to learn how to spell the term as well. But when the term is a sentence of a few lines long, it isn`t reasonable to type out the answer spelt correctly word for word, which is required in order to answer the prompt correctly. Making it effective in some situations, but not all. A "Write" option will not be included in the program this option as it is frustrating when the answer input is correct, but not word for word.

Test



In test, the data is presented in four different ways with 5 questions within each. The first is true or false. It presents a term and gives an explanation leaving it up to the user to decide if this option is true of false. Some may find it effective, however it doesn`t always present the opportunity to improve their knowledge as if it is false, the user isn`t informed of the correct answer, giving them to possibility to test their knowledge. Next, is the written answer which poses the same problems and benefits of the write section confined into a smaller section. The last two are similar. There is a match up section where there is a set of definitions and terms and it is up to the user to match them up correctly. Lastly, there is a multiple-choice section. Here a definition is given and the user must identify the correct term for it. Both these questions mix up the traditional means of answering questions and can break the normal monotony of revising. Yet, I find that the other means of revision included are more effective than these options.

Quizlet Plus



Some of Quizlet features are locked behind a subscription service called "Quizlet Plus". This service currently includes the ability to study offline, no ads, ability to upload images and audio for your sets, ability to scan documents for quick upload and extra font options. This is entirely optional and the site can be used perfectly without ever purchasing this service, and unlike some other websites, the prompts never feel intrusive or forceful.

Conclusion of analysis

There are some notable features about Quizlet and definitely a lot to learn from the site. Its notable features include:
-Simple yet effective user interface
-Ability to search though sets based on search criteria
-Ability to present incorrect answers more often than correct answers
-Sorting of saved sets

Along with these positives there are some negatives that need to be improved upon:
-More advanced false answers are required, as otherwise it is easier to spot the real answer over the fake ones
-There is no "Report" of which areas are needed for focus and improvement

Objectives for proposed system

Proposed System

An A Level physics quiz application that will adapt the questions presented to the user based upon which questions they answer correctly, therefore they are presented with the questions they answer incorrectly more often.

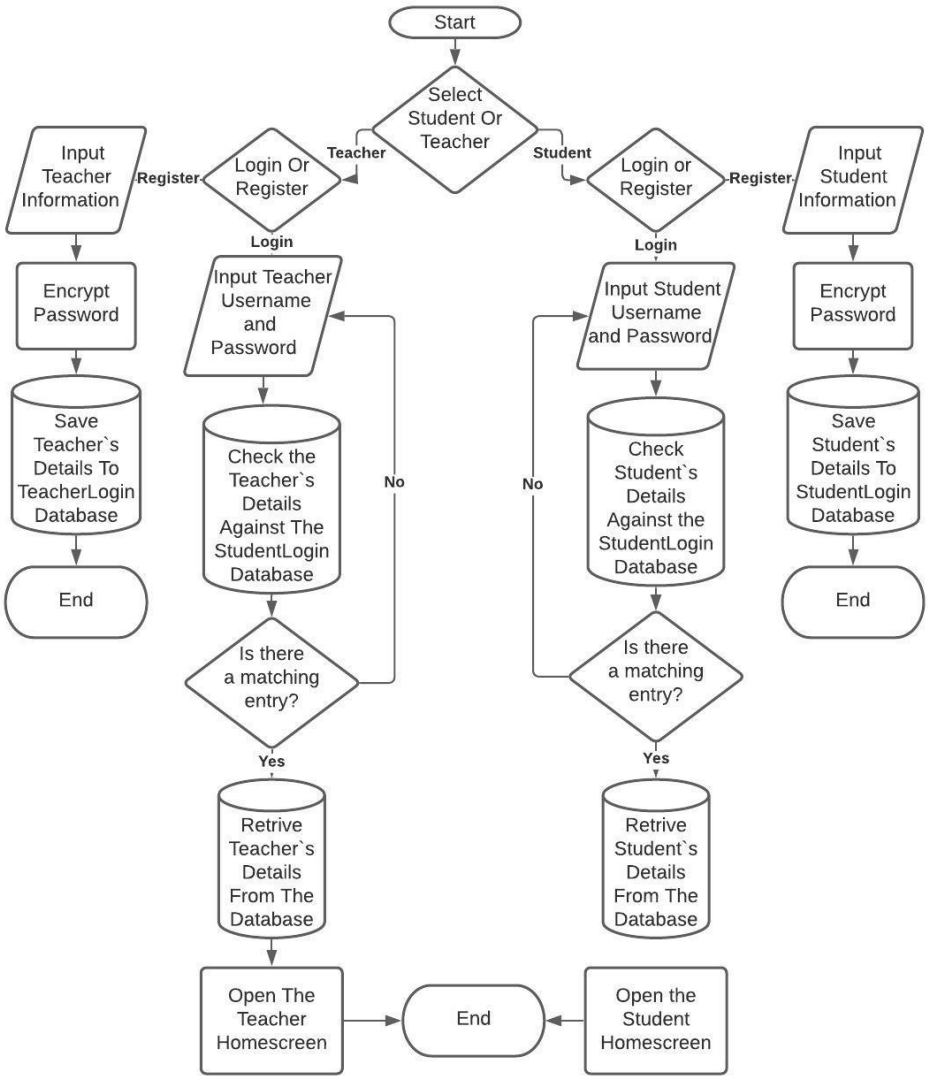
<u>Objectives for proposed system</u>

1. The system must be able to present the user with a question and register an answer. *(Based on Answer 3 from the interview with Mr Delaney)*
   1.1. The system correctly identifies if the answer is correct based on the stored correct answer in the database.
   1.2. The system must provide feedback to the user on which questions they answered incorrectly.
2. The system must be able to store all the relevant details about each question entered. *(Based on Answer 3 from the interview with Mr Delaney)*
3. Data should be saved using a database via SQL implementation.
   3.1. Stored procedures will be used to query the database which will be written in SQL.
4. Each student will have a unique identifier, which will relate to their progress using the program. They will also be allowed to create a username which will be used as the primary key in the "StudentLogin" table.
5. Each teacher will be identified by a unique integer value, and a username which will act as the primary key in the "TeacherLogin" table. They will also be given a "ClassId" integer, so that the students can register to be in their class.
6. The student`s information and teacher`s information will be stored using object orientated programming via the use of classes.
7. Each question must be identified with a question, answer, area, topic. Each topic, and area will be a foreign key, relating to independent tables, making that database relational.
8. The program will use machine learning to present the user with questions they get wrong at a higher frequency than the ones that they get correct.
   (Based on Answer 3 from the interview with Mr Delaney)
9. There will be a difficulty rating for each question that will allow the program to scale based on the user's skill. There will be a pre-defined difficulty and a difficulty based on user`s scores (machine learning). This can be interchanged at the user`s request.
10. The user will be able to view all the questions and answers in a separate menu
    (Based on Answer 6 from the interview with Mr Delaney)
    10.1. The question database must be searchable
    10.2. The user interface will allow the user to filter the questions by physics topic via the use of multiple check boxes, drop combo boxes and keyword search.
11. Users will be able to create quizzes manually by adding questions independently
12. There will be an option to generate a randomized quiz based on the criteria set by the user. The criteria will include the specified topics, area (recall or calculation), difficulty (either programmer defined or machine defined). It will be randomized by a random number generator.
    (Based on Answer 1 from the interview with Mr Delaney)
13. The password of teachers and students will be encrypted using hash set encryption.

14. There will be navigation options at the top of the screen to move between forms or pages.
15. A report of the quiz can be generated upon request displaying user`s scores and weak areas. This will be generated using the database.
    (Based on Question 5 from the interview with Mr Delaney)
    15.1. The report can be sent to the teacher via email.
    15.2. The report will break down each question`s main focus and give a percentage on which types of questions are answered correctly more often.

Design

Login and Register Screen for students



New Database Queries for Login and Register *(Objective 3.1)*

Retrieving Password Salt for Student
SELECT Salt FROM StudentLogin WHERE Username = ?

Attempting Student Login
SELECT * FROM StudentLogin WHERE Username = ? AND Password = ?

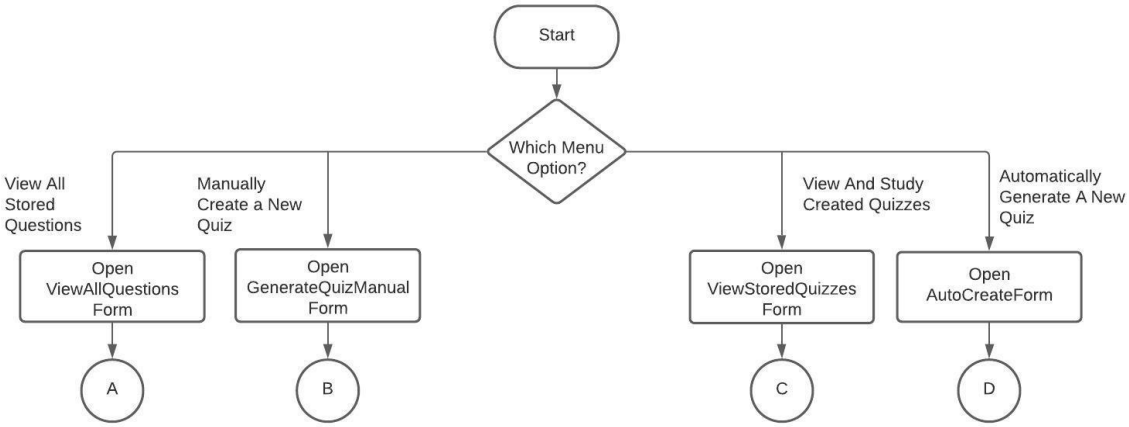
Saving Teacher`s Login Credentials
INSERT INTO TeacherLogin(Title, SecondName, Username, Password, ClassId, Email ,Salt) VALUES(?,?,?,?,?,?,?)

Attempting Teacher Login
SELECT * FROM TeacherLogjn WHERE Username = ? AND Password = ?

Retrieving Password Salt for Teacher
SELECT Salt FROM TeacherLogin WHERE Username = ?

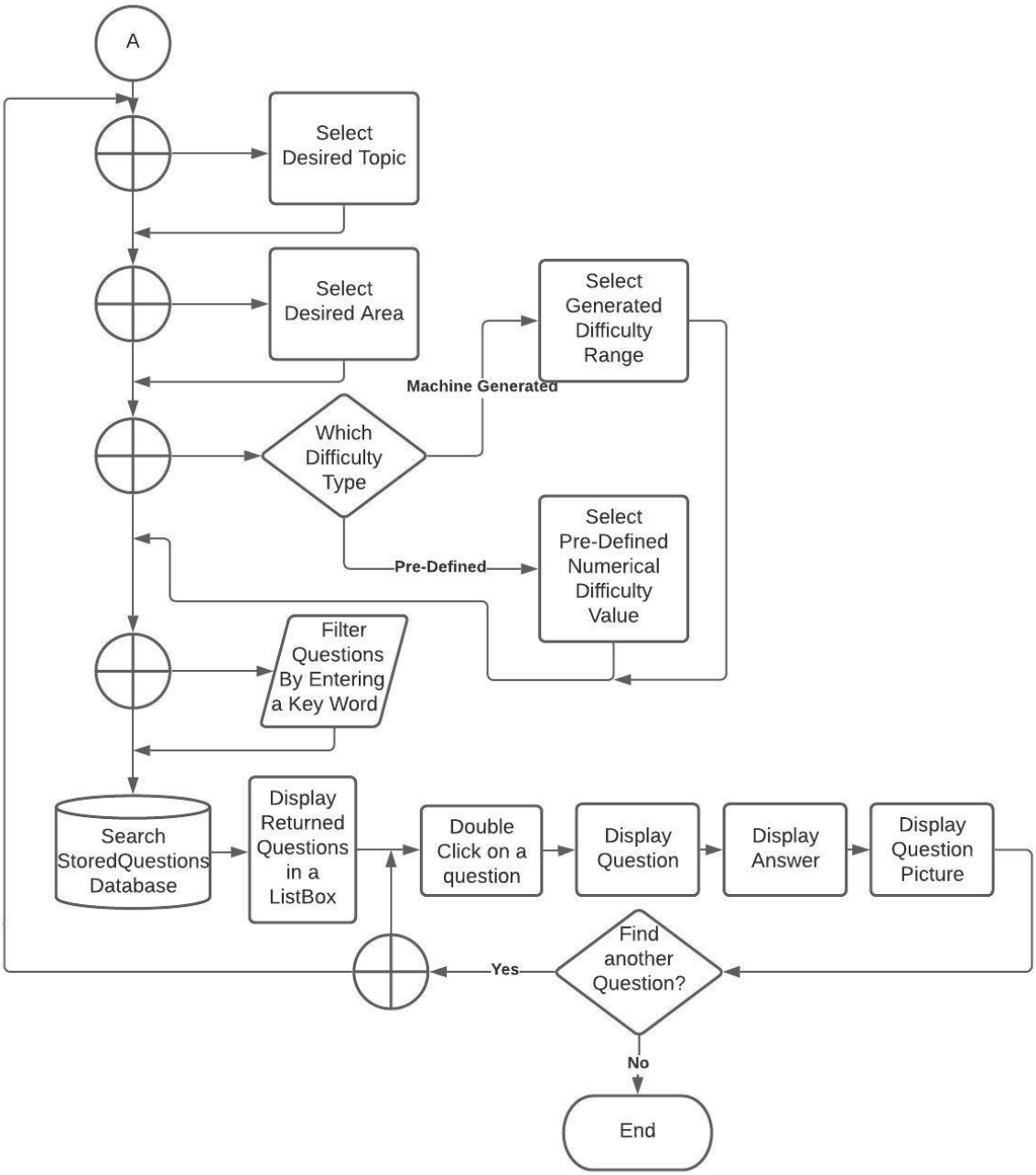
Input	Process
Username Password First Name Surname Class ID	Generate Password Salt Combine Password and Salt Encrypt Password
Storage	Output
Student`s Login Credentials saved to StudentLogin	Student username for login StudentID

Saving Student`s Login Credentials
INSERT INTO StudentLogin(Firstname, SecondName, Username, Password,ClassId, Salt) VALUES(?,?,?,?,?,?)



Student Main Menu

Input	Process
Chose form buttons	Open New Form
Storage	Output
Null	New Form Open



<u>View All Questions</u>

(Objective 10)

Input	Process
• Selected Topics • Selected Areas • Selected Difficulty Type - Generated Difficulty Range - Predefined Difficulty Rating • Filter Key Word	Create Database Query
Storage	Output

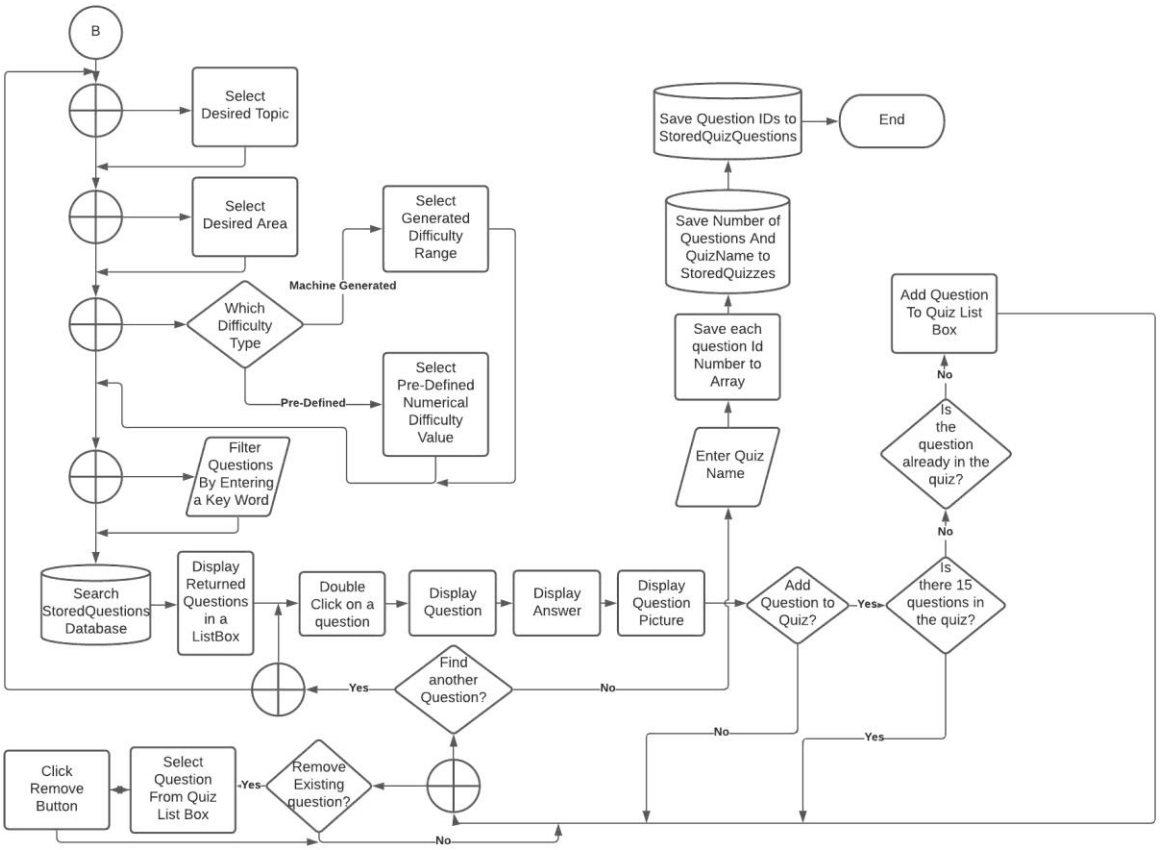
Return Stored Questions that match the criteria set by the user from the database table StoredQuestions	Display Question List Display Questions Display Answers Display Question Picture

View All Questions New Database Quires *(Objective 3.1)*

Retrieving All Stored Questions
SELECT * FROM StoredQuestions

Retrieving Questions Based on Criteria From Stored Questions
SELECT * FROM StoredQuestions WHERE (TopicID = ?) AND (Question LIKE ?) AND (Area = ?) AND (Difficulty = ?)

Generate Quiz Manually Form



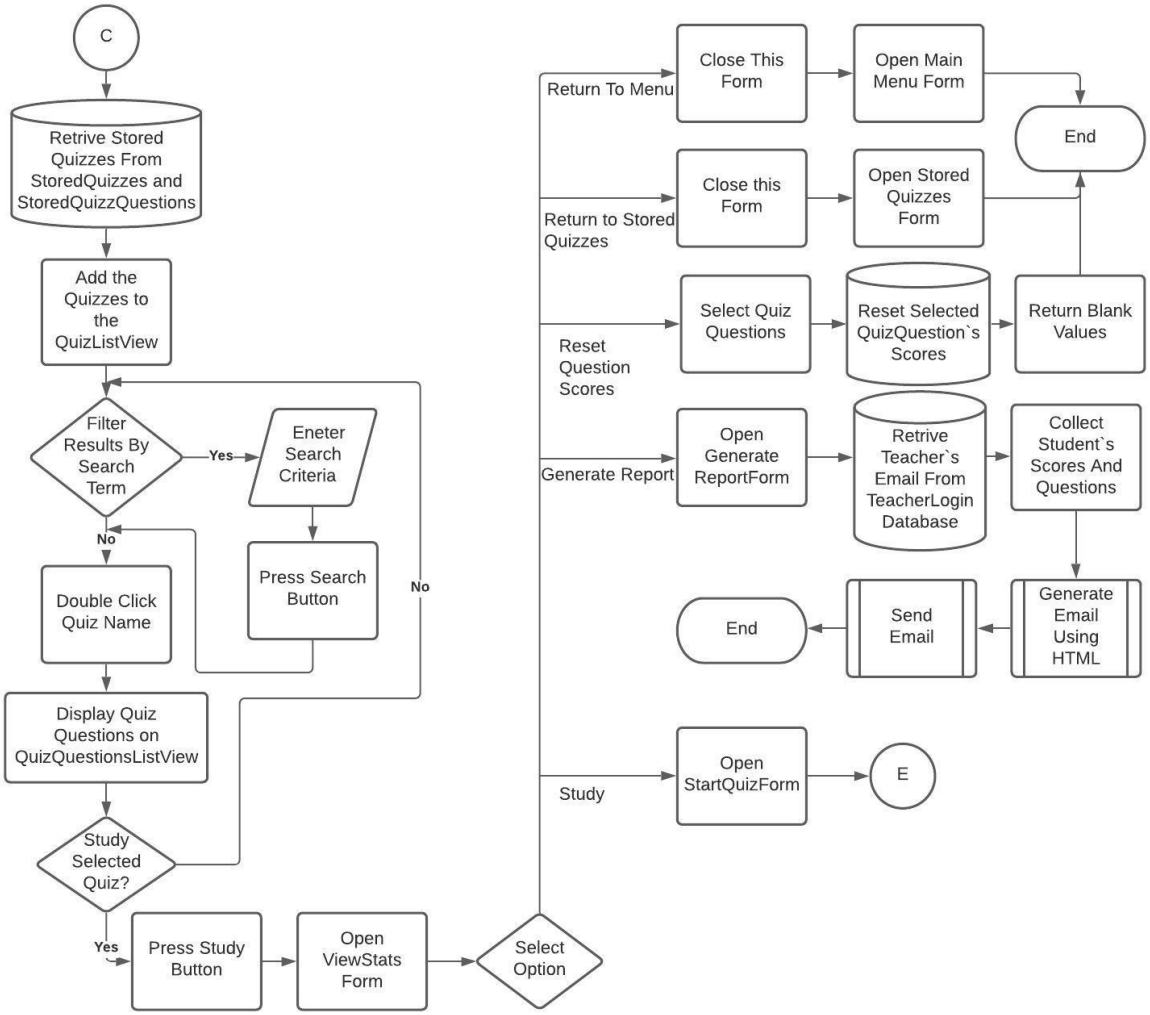
Input	Process
Selected Topics Selected Areas Selected Difficulty Type	Create Database Query Add Selected Question to Question List View Calculate number of remaining questions

Generated Difficulty Range Predefined Difficulty Rating Filter Key Word Quiz Name	Generate query to save quiz
Storage	Output
Return Stored Questions that match the criteria set by the user from the database table StoredQuestions Save quiz to StoredQuizQuestions and StoredQuizzes	Display Question List Display Questions Display Answers Display Question Picture List of Selected Questions Remaining Number of Questions

Generate Quiz Manually New Database Queries *(Objective 3.1)*

Creating A New Quiz Manually
INSERT INTO StoredQuizzes(Name, Length) VALUES (?,?) INSERT INTO StoredQuizQuestions(QuizID, QuestionID) VALUES (IDENT_CURRENT('StoredQuizzes'), ?)

View Stored Quizzes



Input	Process
Quiz Name Select Quiz	Create query to retrieve stored quizzes Create Query to Reset Question Scores Generate HTML code for scores Send Email
Storage	**Output**
Retrieve Teacher`s Email from TeacherLogin Database Retrieve Stored Quizzes from The StoredQuizzes Database Retrieve Stored Questions from StoredQuestions Database and StoredQuizQuestions Database	Questions Questions Areas Questions Difficulty Questions Score Number of Times Questions Answered Correct Number of Times Questions Answered

View Stored Quizzes New Database Queries *(Objective 3.1)*

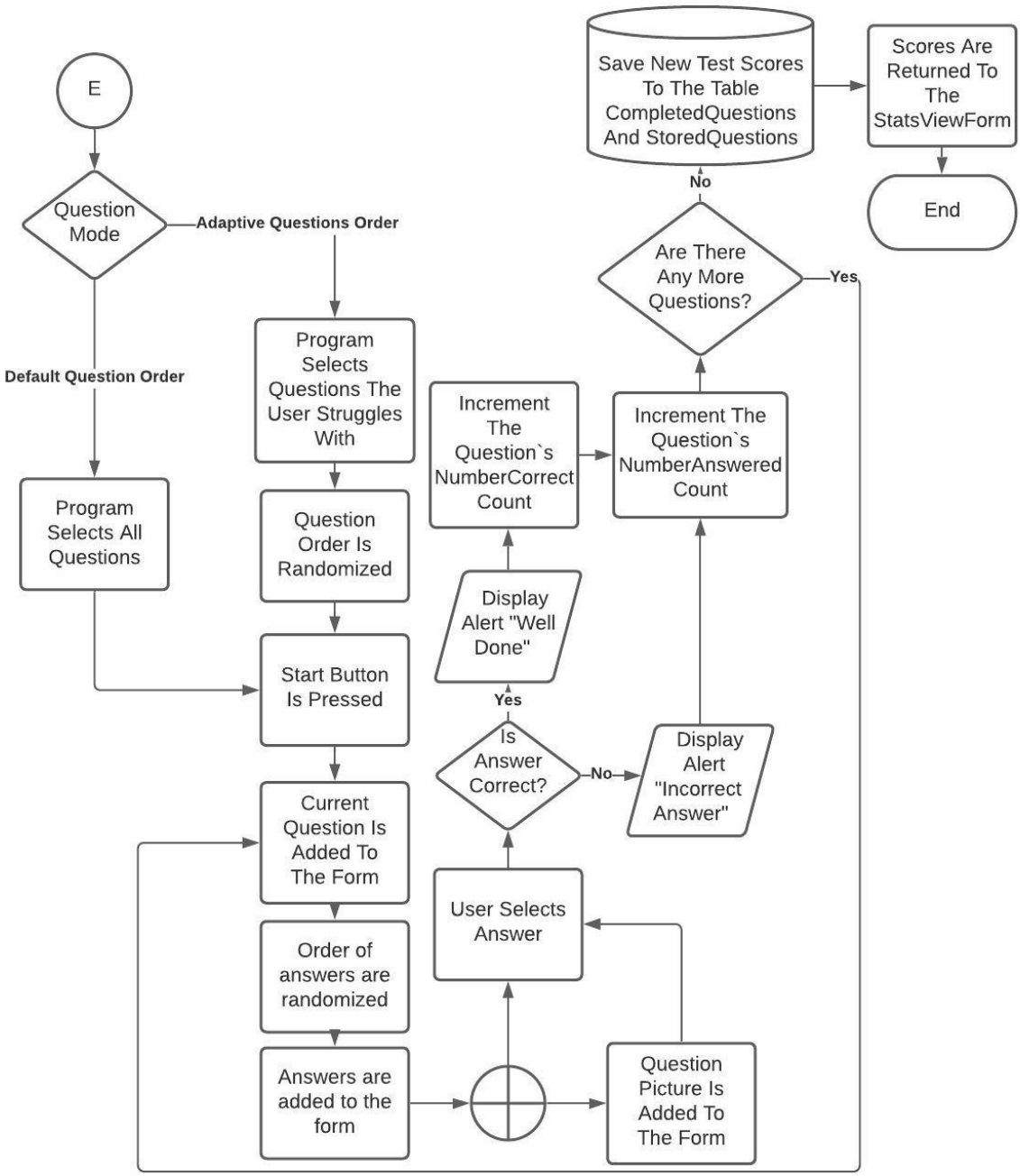
Reset Stored Questions Scores
UPDATE CompletedQuestion Set XCompleted = 0, XCorrect = 0, CalcaulatedDifficulty = 0

WHERE QuestionID = ? AND StudentID = ?

Create Completed Questions
INSERT INTO CompletedQuestion(StudentID, QuestionID) VALUES (?,?)

Get Completed Question
SELECT * FROM CompletedQuestion WHERE QuestionID = ? AND StudentID = ?

Study Question



(Objective 1)

Input	Process
Quiz Type (Adaptive or Standard) Answer	Calculate Question Difficulty Randomize Question Order Generate Question Form Check If Answer Is Correct
Storage	Output
Save Test Scores into CompletedQuestions Database	Question Question Answered Question Picture

	Question Results

Study Question New Database Queries *(Objective 3.1)*

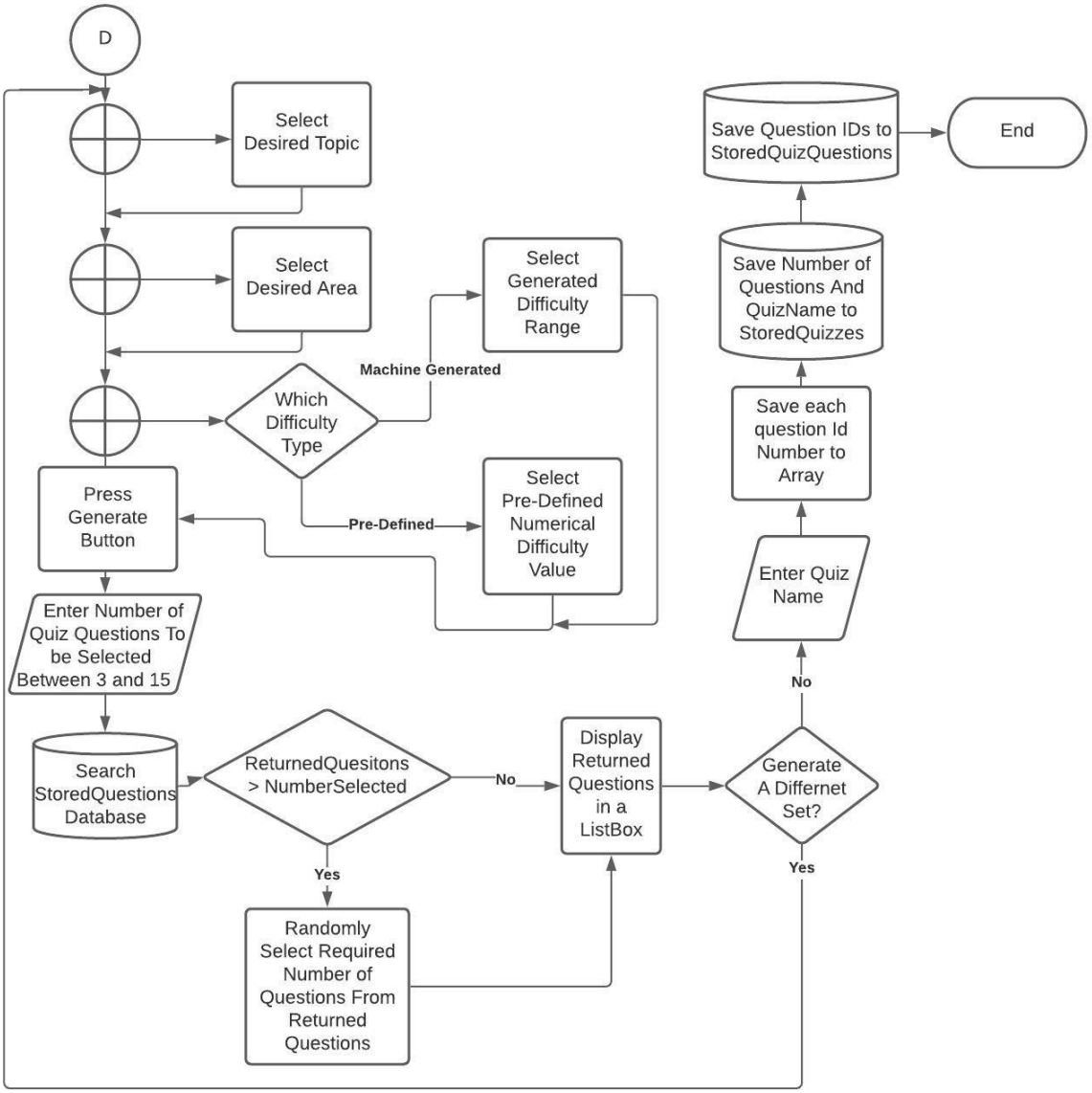
Update Completed Question
UPDATE CompletedQuestion SET XCompleted = ?, XCorrect = ?, CalculatedDifficulty = ? WHERE QuestionID = ? AND StudentID = ?

Update Stored Questions
UPDATE StoredQuestions SET XAnswered = ?, XAnsweredCorrect = ?, CalculatedDifficulty = ? WHERE QuestionID = ? AND StudentID = ?

Automatically Generate Quiz

(Objective 12)

Input	Process
Quiz Area/s Quiz Topic/s Difficulty Type Difficulty Range Number of Questions Quiz Name	Generate Query to Return Questions Based on Criteria Select Correct Number of Questions
Storage	Output
Retrieve Quiz Questions from StoredQuestions Database Save Quiz to StoredQuizQuestions and StoredQuizzes	Selected Quiz Questions



Object Orientated Programming Design and Data Dictionary

CompletedQuestion					
Field	Data Type	Validation Check	Validation Description	Valid Data	Erroneous Data
StudentID	Int	Lookup	Must be associated to a valid StudentID	1	A
QuestionID	Int	Lookup	Must be associated to a valid QuestionID	1	A
XAnsered	Int	Type Check	Must be a numerical value	94	G

XAnswerredCorrectly	Int	Type Check	Must be a numerical value	94	G
CalculatedDifficulty	Int	Range Check	Must be between 0 - 100	5	98562

Stores the scores of the student's questions based on their student ID and QuestionID. It also stores the difficulty that has been calculated. As it is not linked to a specific quiz, it means that progress from questions is carried across questions *(Objective 9)*.

SearchCriteria

Field	Data Type	Validation Check	Validation Description	Valid Data	Erroneous Data
Search	String	None	Search term can take any form, it can also be empty if the user wants	Pendulum	None
Difficulty1 int,	Int	Range Check	The difficulty must be a numerical value between 1 and 4	1	87
Difficulty2 int,	Int	Range Check	The difficulty must be a numerical value between 1 and 4	1	87
Difficulty3 int,	Int	Range Check	The difficulty must be a numerical value between 1 and 4	1	87
Difficulty4 int,	Int	Range Check	The difficulty must be a numerical value between 1 and 4	1	87
Area int,	Int	Range Check	The area must be a numerical value between 1 and 2	1	94
Area1 int,	Int	Range Check	The area must be a numerical value between 1 and 2	1	894

Topic1 int,	Int	Range Check	The Topic must be a numerical value between 1 and 4	1	894
Topic2 int,	Int	Range Check	The Topic must be a numerical value between 1 and 4	1	894
Topic3 int,	Int	Range Check	The Topic must be a numerical value between 1 and 4	1	894
Topic4 int,	Int	Range Check	The Topic must be a numerical value between 1 and 4	1	894
Topic5 int	Int	Range Check	The Topic must be a numerical value between 1 and 4	1	894

Holds the data that will be used to query the database when searching for a specific or multiple questions *(Objective 10.1)*. The search string will filter the database`s questions based on the string entered. The difficulty 1-4 holds the possible difficulties that could be selected. Area and area1 hold the areas (Recall and Calculation) that can be selected. The Topics 1-5 hold the range of topics that can be selected *(Objective 10.2)*.

StoredQuestions

Field	Data Type	Validation Check	Validation Description	Valid Data	Erroneous Data
QuestionId	Int	Type Check	Program designates the ascending value in the SQL table	7	A
CorrectAns	string	Presence Check	Must be a correct answer	A	“”
IncorrectAns1	string	Presence Check	Must be an incorrect answer	B	“”
IncorrectAns2	string	Presence Check	Must be an incorrect answer	C	“”
IncorrectAns3	string	Presence Check	Must be an incorrect answer	D	“”
PictureURL	string	Format Check	Must be in the correct file path format	C:\Users\Tom Pearson\Desktop\PhysicsQuiz\Code\PhysicsQuiz1.0\QuestionPictures	Desktop

TopicId	Int	Range Check, Type Check, Presence Check	Must be a numerical value between 1 and 4 and all questions must have a topic	1	894
Area	Int	Range Check, Type Check, Presence Check	Must be a numerical value between 1 and 2 and all questions must have an area	1	89
DifficultyRating	Int	Range Check	Must be between 1 and 4	1	94
Question	string	Presence Check	Must be a question answer	What is the specific heat capacity of water?	“”
XAnswered	Int	Type Check	Must be a numerical value	94	G
XAnsweredCorrectly	Int	Type Check	Must be a numerical value	94	G
CalculatedDifficulty	Int	Range Check	Must be between 0 - 100	5	98562

Holds the stored questions for the program. The question ID is used as the primary key in the table *(Objective 7).* It uniquely identifies each question. It is also used in other tables such as completed question to relate their scores to the individual questions. The CorrectAns stores the correct answer (Objective 1.1), it can be anything such as a letter or a sentence, therefore it is a string. The incorrect answers must also be input so that the correct answer isn`t obvious as it is the answer relating to the question. The PictureURL stores the path to the file in the program *(Objective 2).* It is allowed to be null as some questions don`t need a picture. TopicID holds the number of the topic that the question relates to in a similar way to area which holds the area *(Objective 7)*. Difficulty rating is a predefined difficulty that can be used to filter questions. Question holds the text from the main question body *(Objective 9)*. XAnswered holds the number of times the question has been answered and XAnsweredCorrectly. Both of these values are then used to calculate the calculated difficulty. This allows the calculated difficulty to scale based on how often it is answered correctly, making it the most accurate difficulty rating when it has been answered a large number of times *(Objective 9)*. However, if it has only been answered a limited number of times it may incorrectly represent the question`s difficulty.

Stored Quiz Questions					
Field	Data Type	Validation Check	Validation Description	Valid Data	Erroneous Data
QuizID	int	Lookup	Must be associated with only one valid QuizID	87	AAA

QuestionID	int	Lookup	Must be associated with only one valid QuestionID	54	AAA

Stored Questions holds the questions that each quiz contains. When a quiz is created each question that it is related to is saved in this table. The QuizID holds the ID of the quiz that the entry is refering to and then the QuestionID relates to the StoredQuestion which is saved to the Quiz. That way when Quizzes of different lengths are created, there will be no wasted space. *(Objective 11)*

StoredQuizzes

Field	Data Type	Validation Check	Validation Description	Valid Data	Erroneous Data
QuizID	int	Lookup	Must be associated with only one valid QuizID	87	AAA
Name	string	Presence Check	Name must be added to quiz	Chapter 1 Questions	“”
Length	Int	Presence Check	Must be given a number of questions to store	7	0

StoredQuizzes holds the name of the quiz and the number of questions that the quiz contains. The QuizID identifies the individual quiz records as the primary key. Name is the name that has been assigned to it by the user and Length contains how many questions are included in the quiz.

Login: **Parent to StudentLogin and TeacherLogin**

Field	Data Type	Validation Check	Validation Description	Valid Data	Erroneous Data
SecondName	string	Presence, Presence	Must enter a surname	Thompson	“”
ClassID	int	Lookup, Presence	Must be associated with a valid ClassID	9	AA
Salt	string	Range, Presence	Must be a random string of 20 characters	DhstCpDcKRR/mPcjvkO8	Asd
Username	string	Presence, Unique	Must be associated with no other student	TomP38	“”
Password	string	Presence, Range, Format	Must contain an uppercase letter,	BigFish211	Password

			lowercase letter, number and be between 8 to 15 characters		

This class is the parent to StudentLogin and TeacherLogin and should never be initialized so therefore is abstract. It holds variables that are present in both classes so therefore **Inheritance** is used. Surname variables are present in both classes although used in different use cases. For students it is needed for their full name where as for teachers it is placed before their title. ClassID is an int containing the ID of their class. The ClassId is an int that contains either the class the student is part of or the class the teacher is the owner off. Salt and Password both store information about the user`s login information. The salt string contains a randomly generated string which is added on the end of the password before it is encrypted using hash set encryption *(Objective 13)*. This makes the password even more secure. As the password is also encrypted, it means that even if an unauthorised user gains access to the database they won`t be able to decipher what the password is. Username is this table`s primary key and is mainly used when the user logins in as that way they don`t have to remember a login number, but instead a personalised string.

Student Login

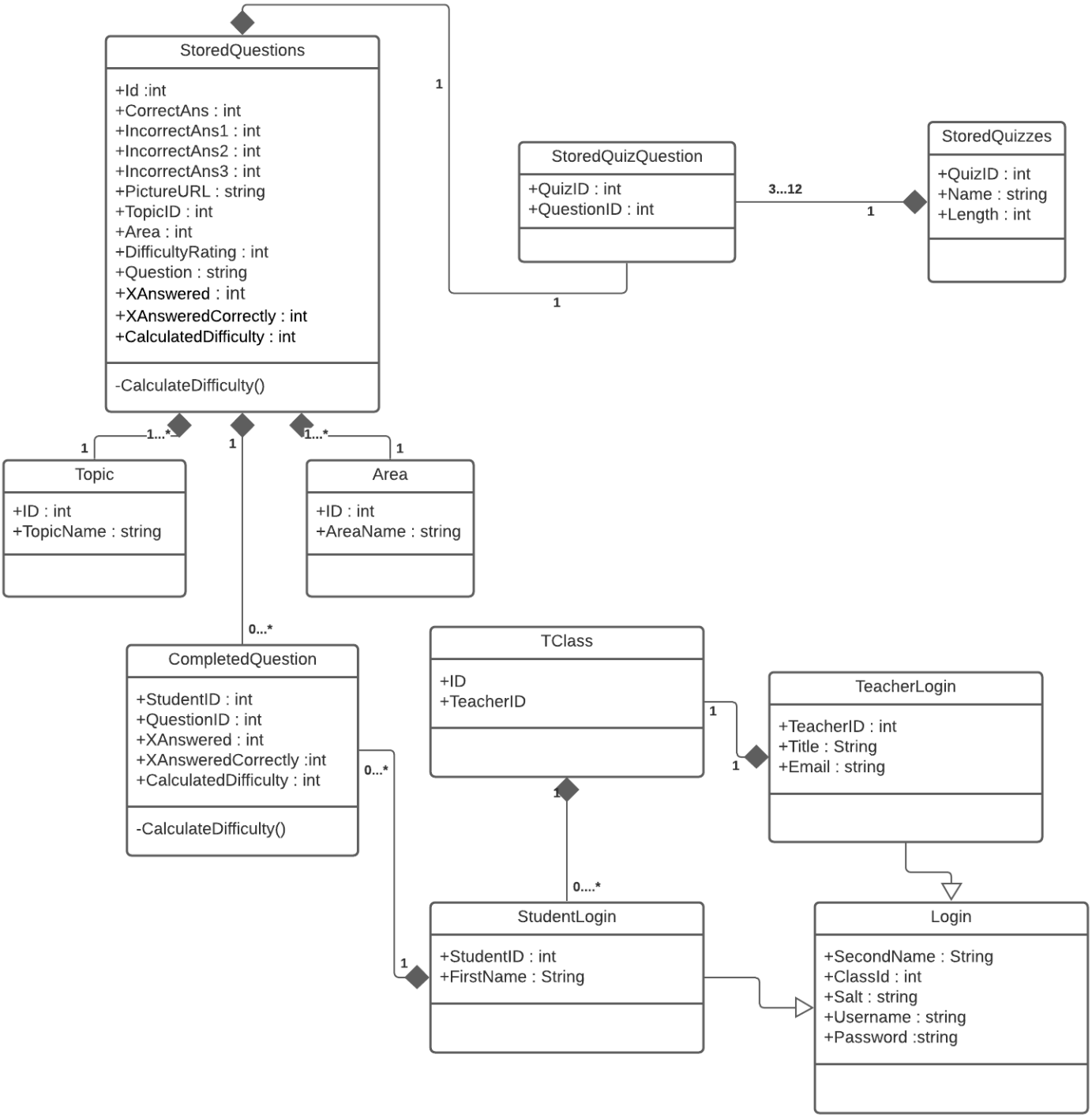
Field	Data Type	Validation Check	Validation Description	Valid Data	Erroneous Data
StudnetId	int	Unique	Must be associated with no other student	7	AA
FirstName	string	Presence, Presence	Must enter a name	Tom	“”
FullName	string	None`	This variable is assigned its value by the program by combining the first name and surname of the user	Tom Thompson	“”

Holds the login information for the student *(Objective 6)*. The StudentID is a foreign key for multiple tables such as completed question. It allows us to identify each student easily and efficiently *(Objective 4)*. The Firstname and Surname variables stores the student`s names and the FullName string combines the two for ease of use. ClassID is an int containing the ID of their class. This makes it so that they are easily identifiable by their teacher and therefore can send emails to their teacher`s email with their progress. Salt and Password both store information about the user`s login information. The salt string contains a randomly generated string which is added on the end of the password before it is encrypted using hash set encryption *(Objective 13)*. This makes the password even more secure. As the password is also encrypted, it means that even if an unauthorised user gains access to the database they won`t be able to decipher what the password is. Username is this table`s primary key and is mainly used when the user logins in as that way they don`t have to remember a login number, but instead a personalised string.

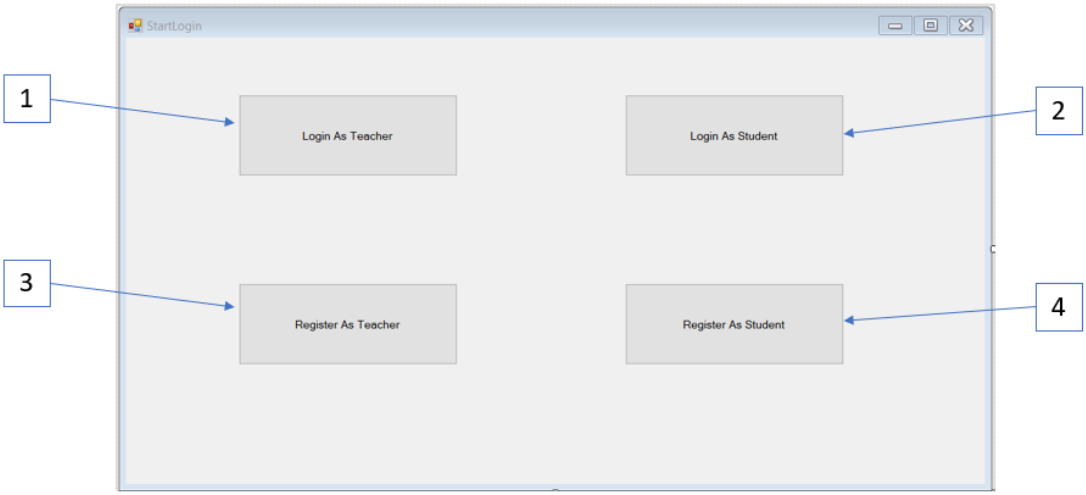
TClass					
Field	Data Type	Validation Check	Validation Description	Valid Data	Erroneous Data
ID	int	Unique	Must be an new ID value	15	“”
TeacherID	int	Lookup	Must be associated to a teacher	1	“”
A composite primary key containing the unique class ID created by an incrementing counter and the Teacher`s ID. *(Objective 5)*					

TeacherLogin					
Field	Data Type	Validation Check	Validation Description	Valid Data	Erroneous Data
TeacherId	int	Unique	Must be a new TeacherID value	1	A
Title	string	Lookup	Must be either Mr, Mrs, Miss, Ms, Dr	Mr	Angel
Email	string	Format	Must be an email	test@example.com	test
TeacherLogin inherits the variables from Login and this class holds the login information for the teacher (*Objective 6)*. The TeacherID is a foreign key for multiple tables such as completed question. It allows us to identify each teacher easily and efficiently *(Objective 5)*. The teacher`s email is taken so that their student`s results can be emailed to them *(Objective 15).*					

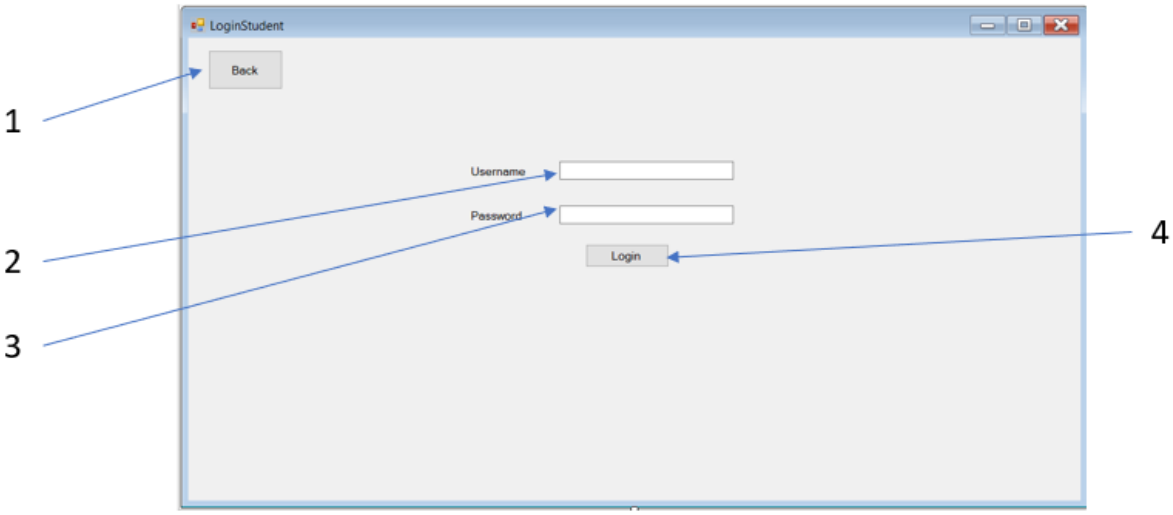
Class Diagram



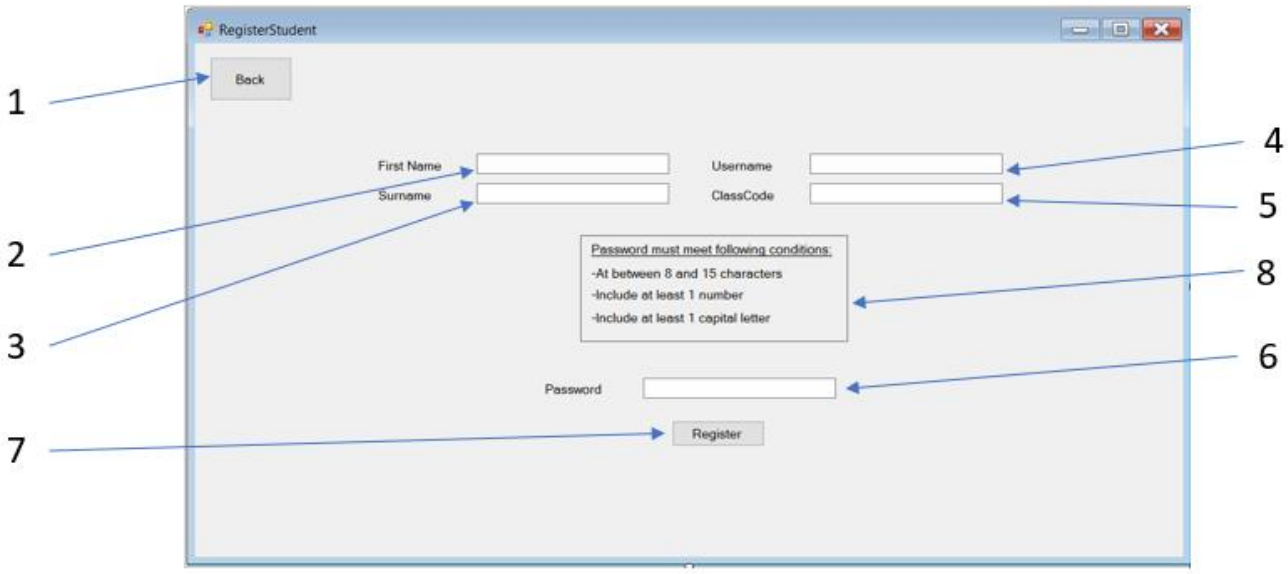
Login Forms Design



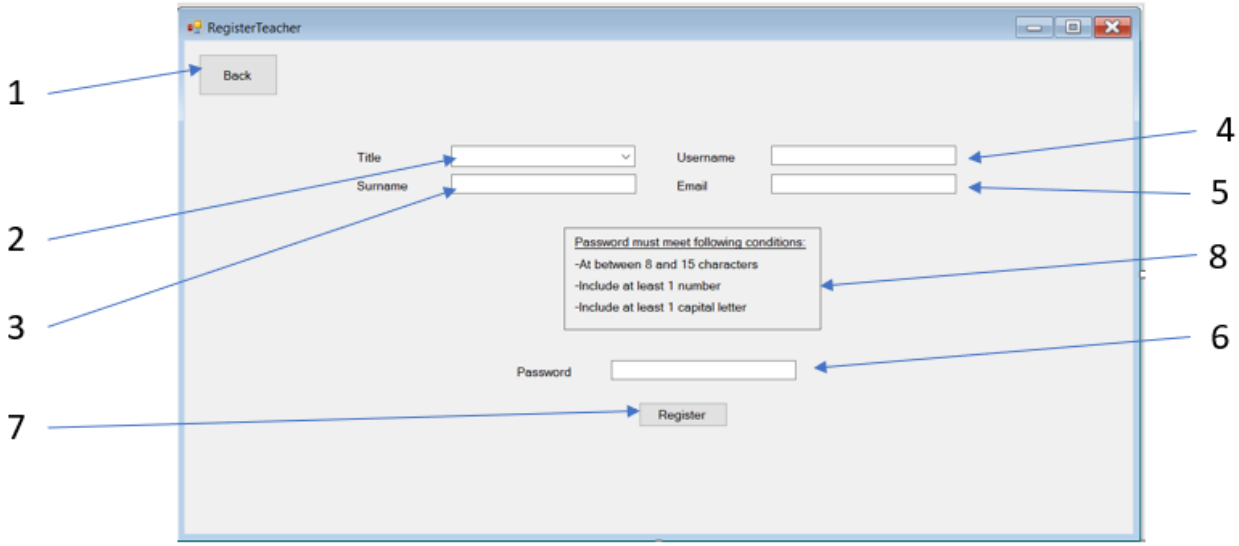
StartLogin Form	
1	Button – Launches TeacherLogin Form
2	Button – Launches StudentLogin Form
3	Button – Launches RegisterTeacher Form
4	Button – Launches RegisterStudent From



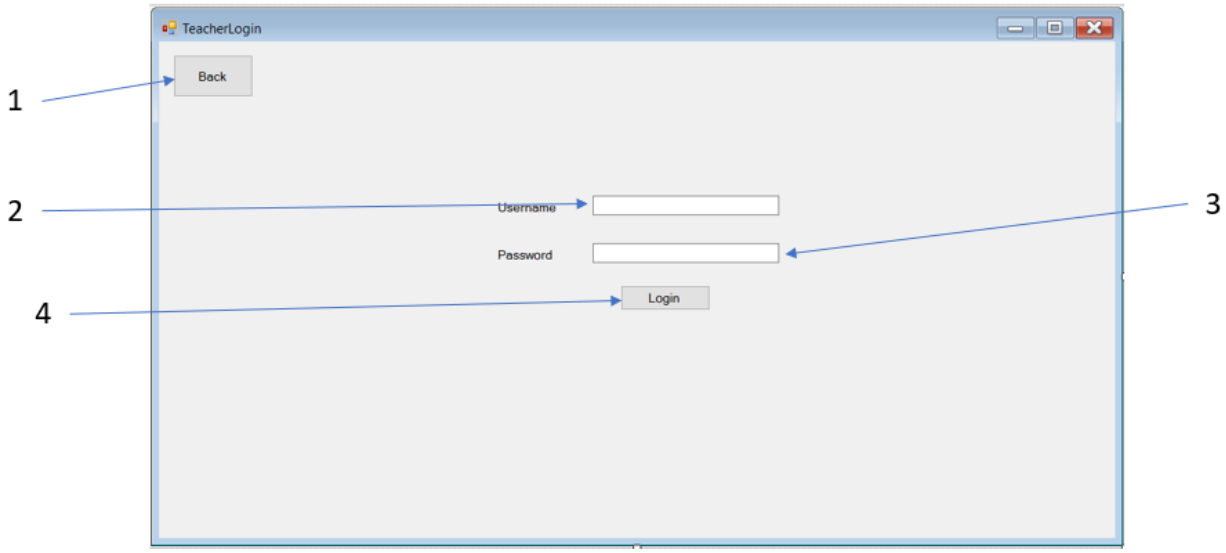
LoginStudent Form	
1	Button – Returns to the previous form *(Objective 14)*
2	TextBox – User inputs username
3	TextBox – User inputs Password
4	Button – Login Button Submits the Username and Password



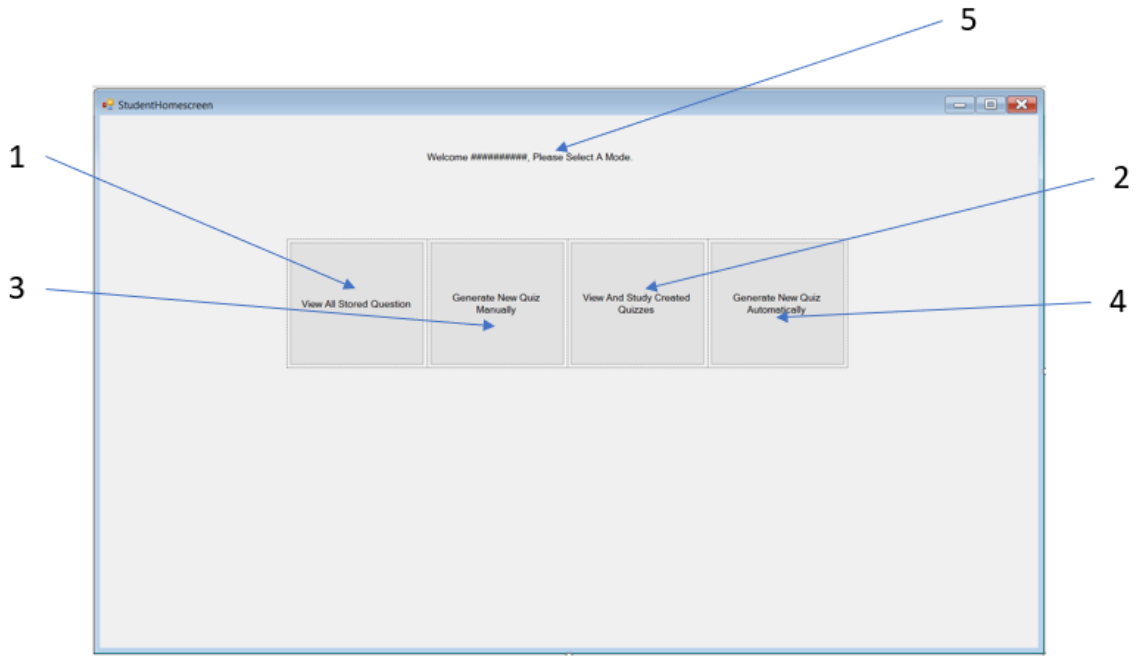
RegisterStudent Form	
1	Button – Returns to the previous form *(Objective 14)*
2	TextBox – First Name input from user to register account
3	TextBox –Surname input from user to register account
4	TextBox – User inputs unique username
5	TextBox – User inputs class code
6	TextBox – User inputs password that must correspond to the password criteria set out above
7	Button – Triggers the create account code
8	Label – Contains the criteria needed to create a password



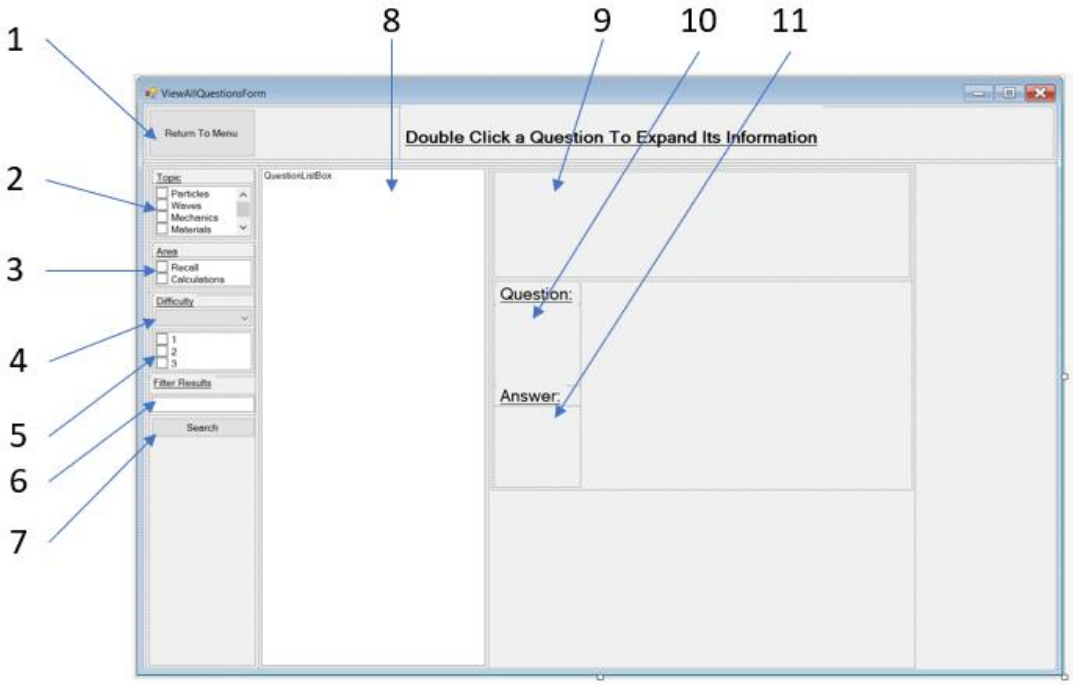
RegisterTeacher Form	
1	Button – Returns to the previous form *(Objective 14)*
2	TextBox – First Name input from user to register account
3	TextBox –Surname input from user to register account
4	TextBox – User inputs unique username
5	TextBox – User inputs email
6	TextBox – User inputs password that must correspond to the password criteria set out above
7	Button – Triggers the create account code
8	Label – Contains the criteria needed to create a password



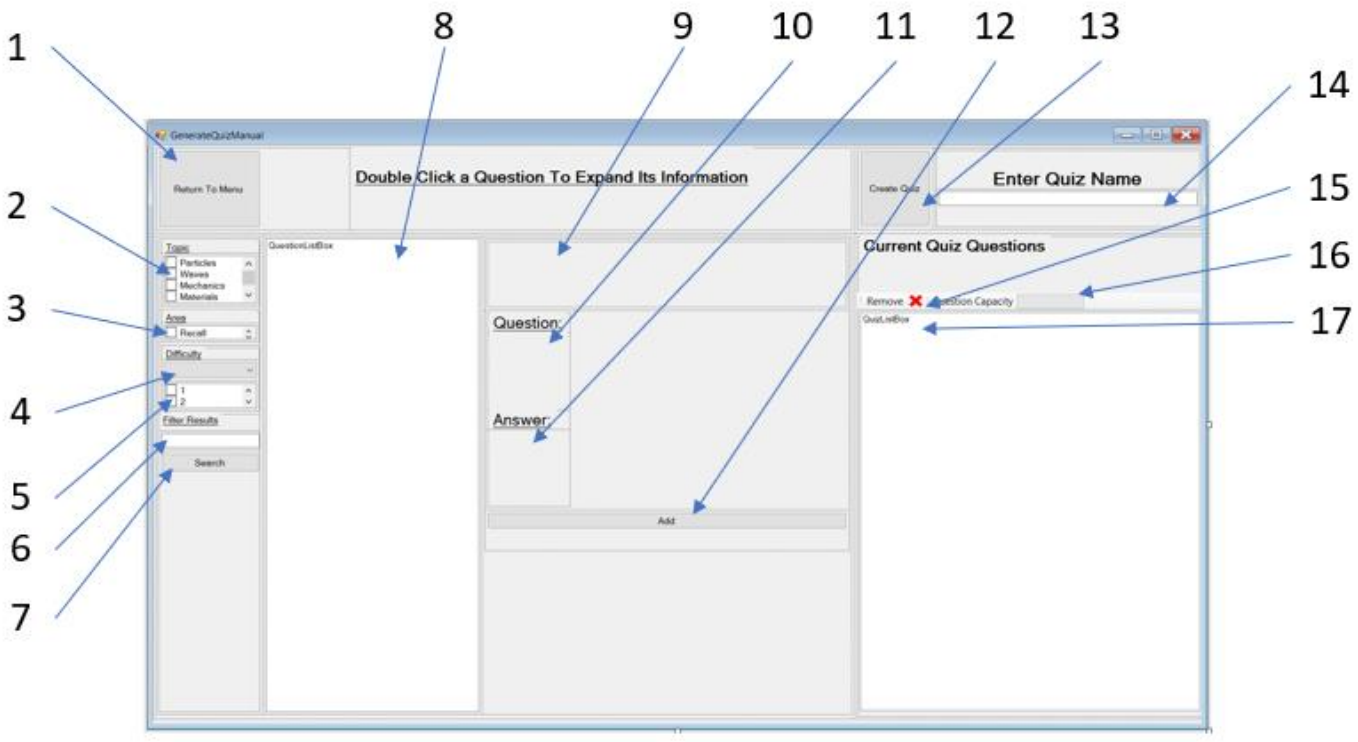
TeacherLogin Form	
1	Button – Returns to the previous form *(Objective 14)*
2	TextBox – User inputs username
3	TextBox – User inputs Password
4	Button – Login Button Submits the Username and Password



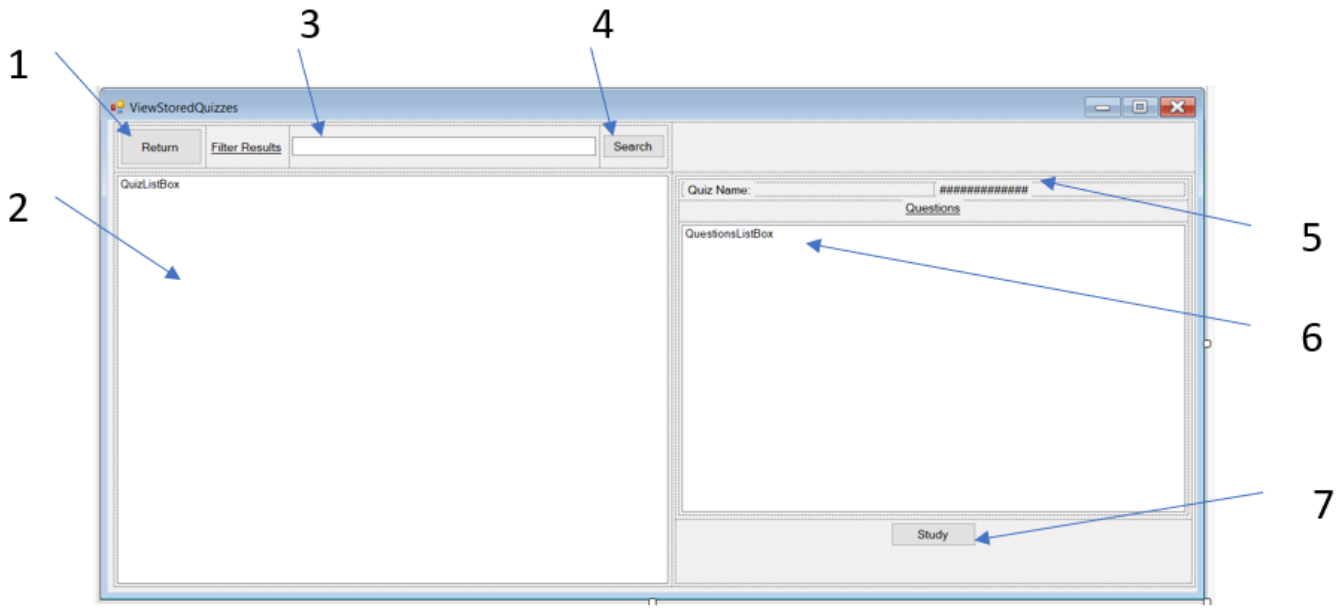
Student Home Screen Form	
1	Button – Opens the view all questions form
2	Button – Opens the Generate New Quiz form
3	Button – Opens the View and Study Created Quizzes form
4	Button – Opens the generate a new quiz automatically form.
5	Label – Presents the user with a welcome message displaying their full name



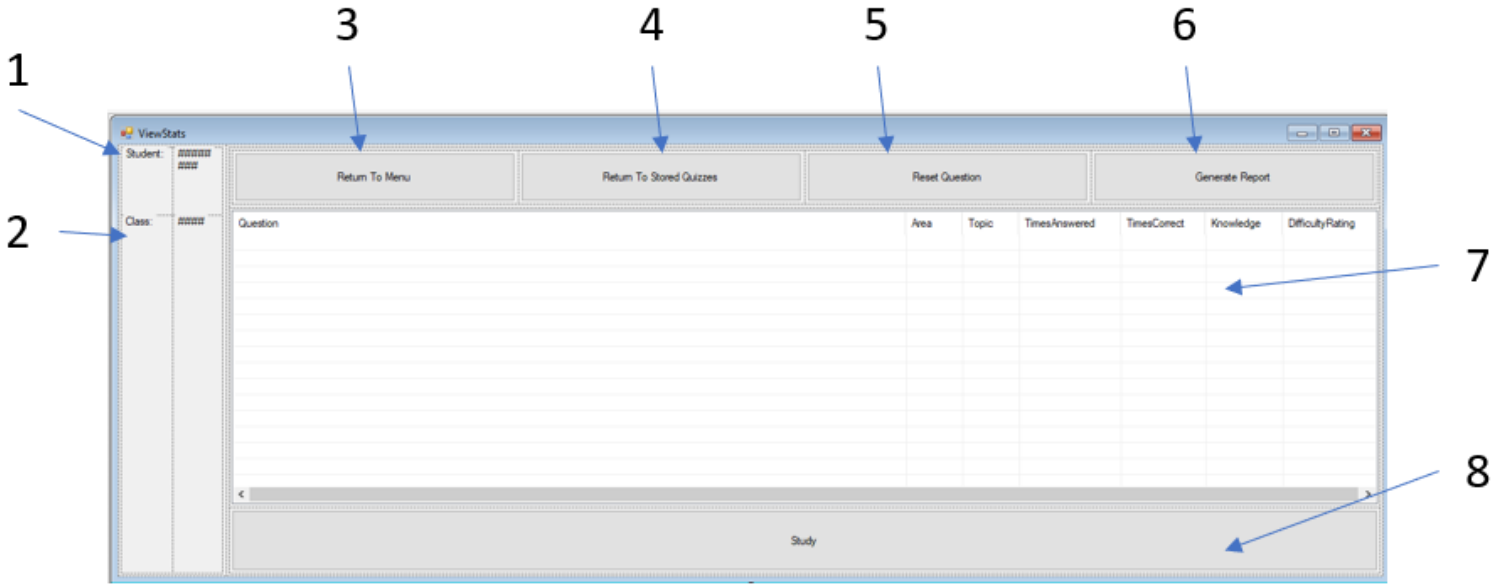
View All Questions Form *(Objective 10)*	
1	Button – Returns to the home screen *(Objective 14)*
2	Checked List Box – User selects criteria for searched topics
3	Checked List Box – User selects criteria for searched Area
4	Combo Box – User can change the difficulty type between Pre-defined and Machine Generated Difficulty Setting
5	Checked List Box – User selects criteria their desired difficulty for the questions
6	Text Box – User can enter keywords for search criteria
7	Button – Tiggers the search
8	List Box – Displays the questions that match the search criteria defined by the user
9	Picture Box – Displays the question`s associated picture
10	Label – Displays the question`s associated Question
11	Label - Displays the question`s associated Answer



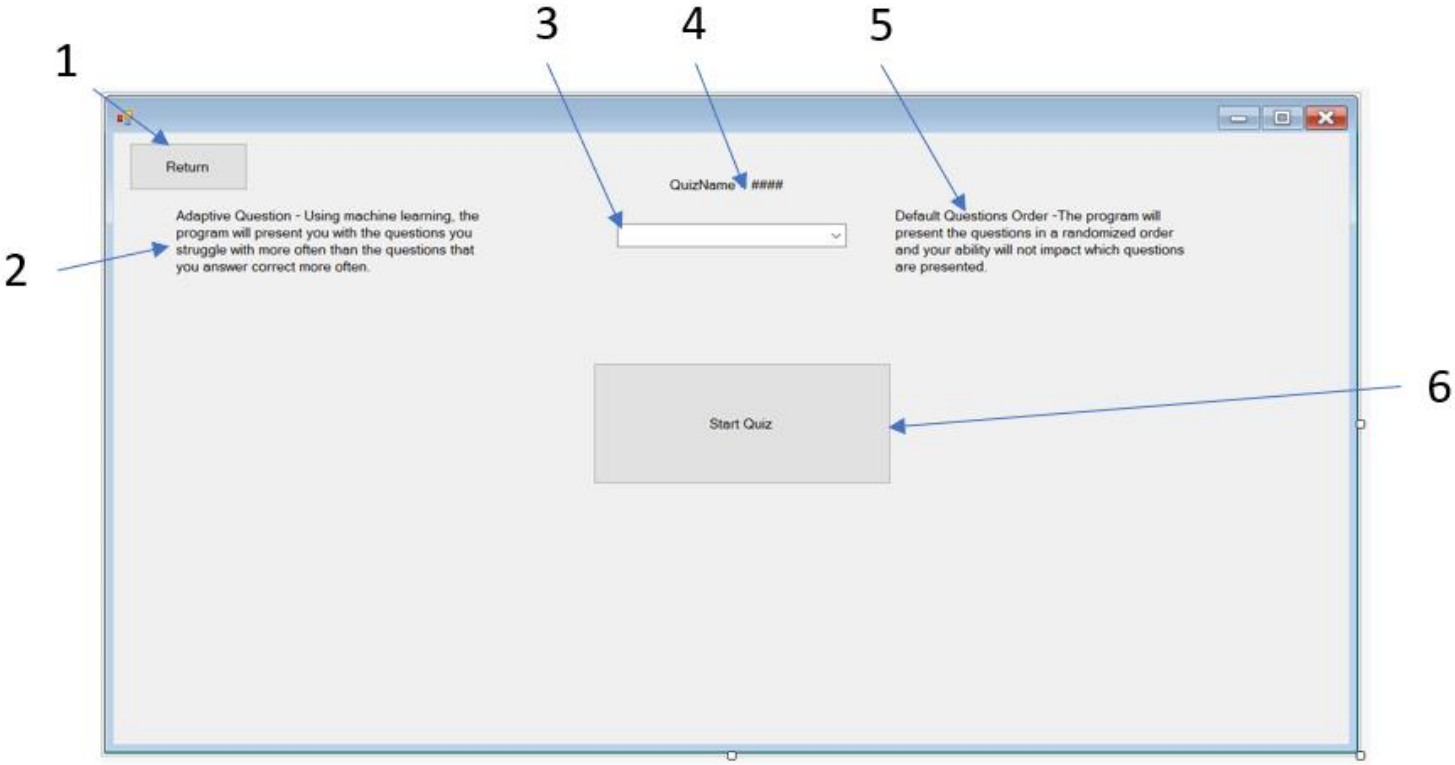
Create Quiz Manually Form *(Objective 11)*	
1	Button – Returns to the home screen *(Objective 14)*
2	Checked List Box – User selects criteria for searched topics
3	Checked List Box – User selects criteria for searched Area
4	Combo Box – User can change the difficulty type between Pre-defined and Machine Generated Difficulty Setting
5	Checked List Box – User selects criteria their desired difficulty for the questions
6	Text Box – User can enter keywords for search criteria
7	Button – Tiggers the search
8	List Box – Displays the questions that match the search criteria defined by the user
9	Picture Box – Displays the question`s associated picture
10	Label – Displays the question`s associated Question
11	Label - Displays the question`s associated Answer
12	Button – Adds the question to the quiz
13	Button – Creates the quiz
14	Text Box – User enters the quiz name
15	Button – Removes selected question from the list box
16	Progress Bar – Displays the quiz capacity, becomes full when no more questions can be added
17	List Box – Display the questions that have been added to the quiz



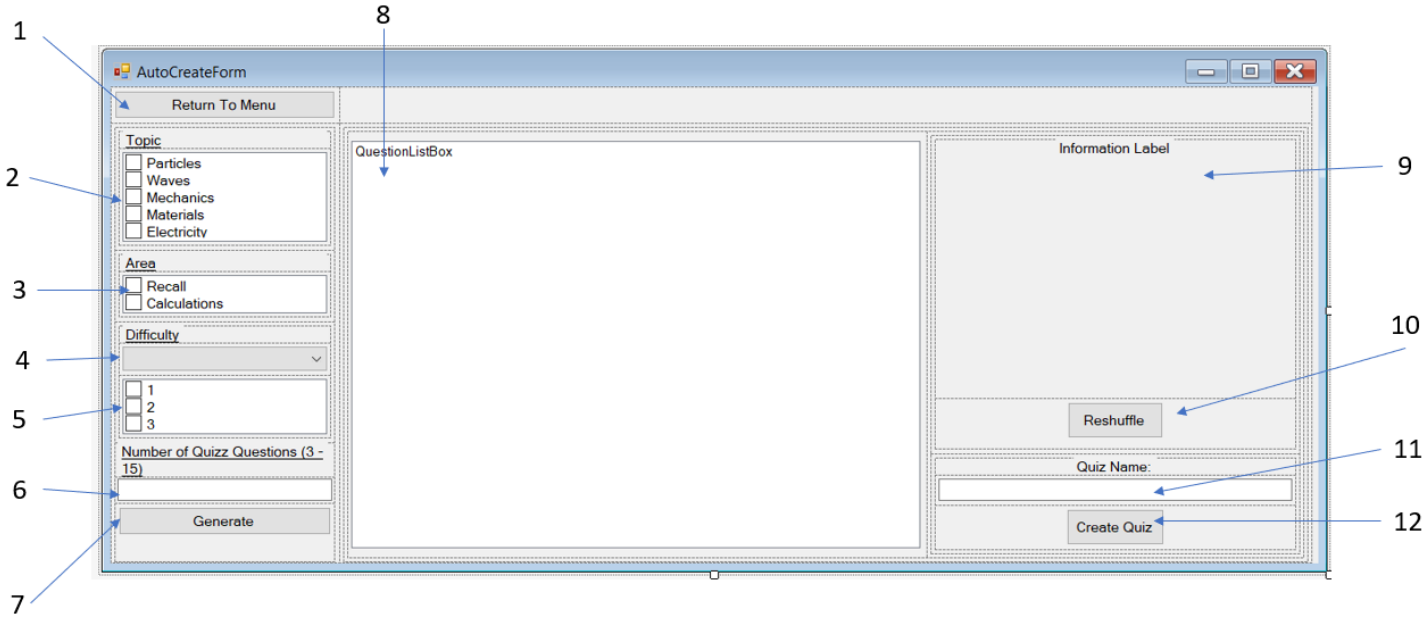
View All Quizzes Form	
1	Button - Returns to the home screen *(Objective 14)*
2	List Box – Displays the stored quizzes
3	Text Box – User enters search term to filter quizzes
4	Button – Triggers the search
5	Label – Displays quiz name
6	List Box – Displays quiz questions
7	Button – Opens the study form



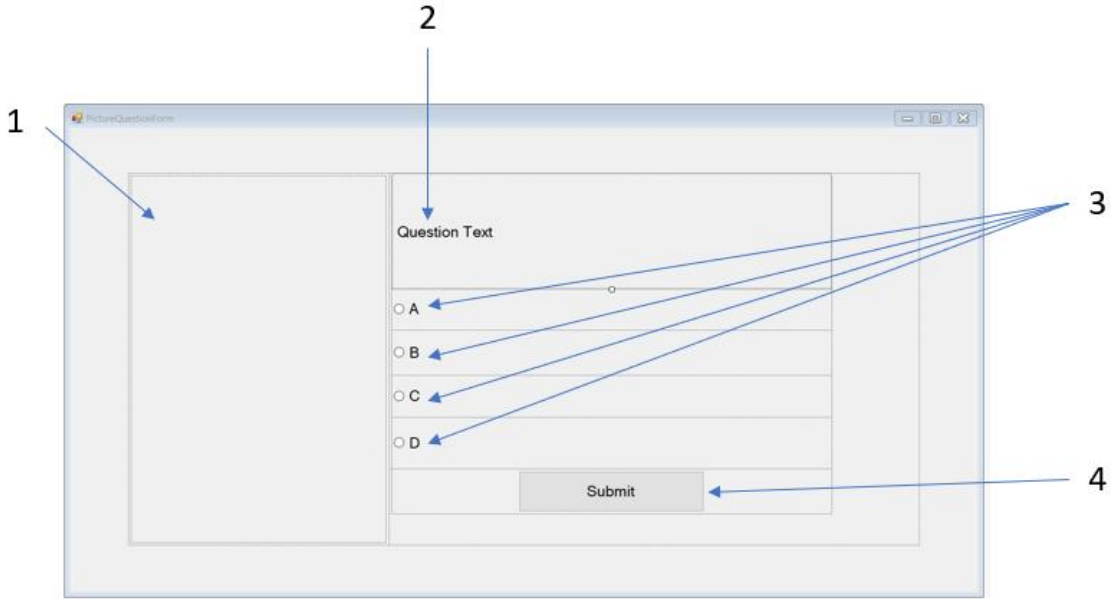
View Stats Form	
1	Label - Displays the student`s name
2	Label – Displays the class Id
3	Button – Returns to main menu *(Objective 14)*
4	Button – Returns to the stored quizzes page
5	Button – Resets the question scores
6	Button – Generates a quiz report and emails it to teacher *(Objective 15)*
7	List View – Displays stats about the student`s past tests such as questions answered correct, knowledge rating and the question`s areas
8	Button – Starts study quiz form



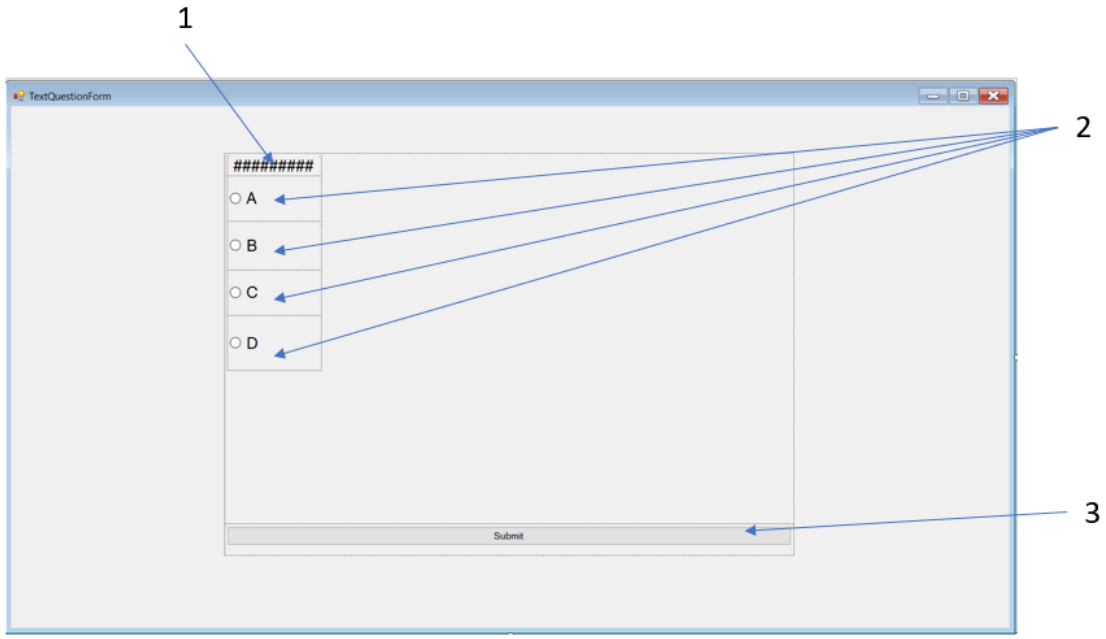
Start Quiz Form	
1	Button – Returns to View Stats Form *(Objective 14)*
2	Label – Displays what the adaptive question option is *(Objective 8)*
3	Button – Combo box displaying the adaptive question option or the default question order
4	Label – Displays the quiz name
5	Label – Displays what the default question order is
6	Button – Starts the quiz



Auto Create Quiz Form	
1	Button – Returns to the home screen *(Objective 14)*
2	Checked List Box – User selects criteria for searched topics
3	Checked List Box – User selects criteria for searched Area
4	Combo Box – User can change the difficulty type between Pre-defined and Machine Generated Difficulty Setting
5	Checked List Box – User selects criteria their desired difficulty for the questions
6	Text Box – User enters how many questions they want in their quiz
7	Button – Tiggers the generation
8	List Box – Displays the questions that match the search criteria defined by the user
9	Label – Displays the number of questions returned by the search and how many have been selected from the returned results
10	Button – Reshuffles the selected questions
11	Text Box – User enters quiz name
12	List Box – Display the questions that have been added to the quiz



Picture Quiz Form *(Objective 1)*	
1	Picture Box – Displays question`s associated picture
2	Label – Displays the stored question`s question
3	Radar Buttons – Displays the possible answers
4	Button – Submits answer



Picture Quiz Form *(Objective 1)*	
1	Label – Displays the stored question`s question
2	Radar Buttons – Displays the possible answers
3	Button – Submits answer

SQL Tables Design

(Objective 3)

	Primary Key
	Foreign Key
	Attribute

CompletedQuestion				
StudentId	QuestionId	XCompleted	XCorrect	CalculatedDifficulty

AreaList	
Id	Name

StoredQuestions						
QuestionId	CorrectAns	IncorrectAns1	IncorrectAns2	IncorrectAns3	PictureUrl	TopicId

Area	DifficultyRating	Question	xAnswered	xAnsweredCorrectly	CalculatedDifficulty

StoredQuizQuestions	
QuizId	QuestionId

StoredQuizzes		
QuizId	Name	Length

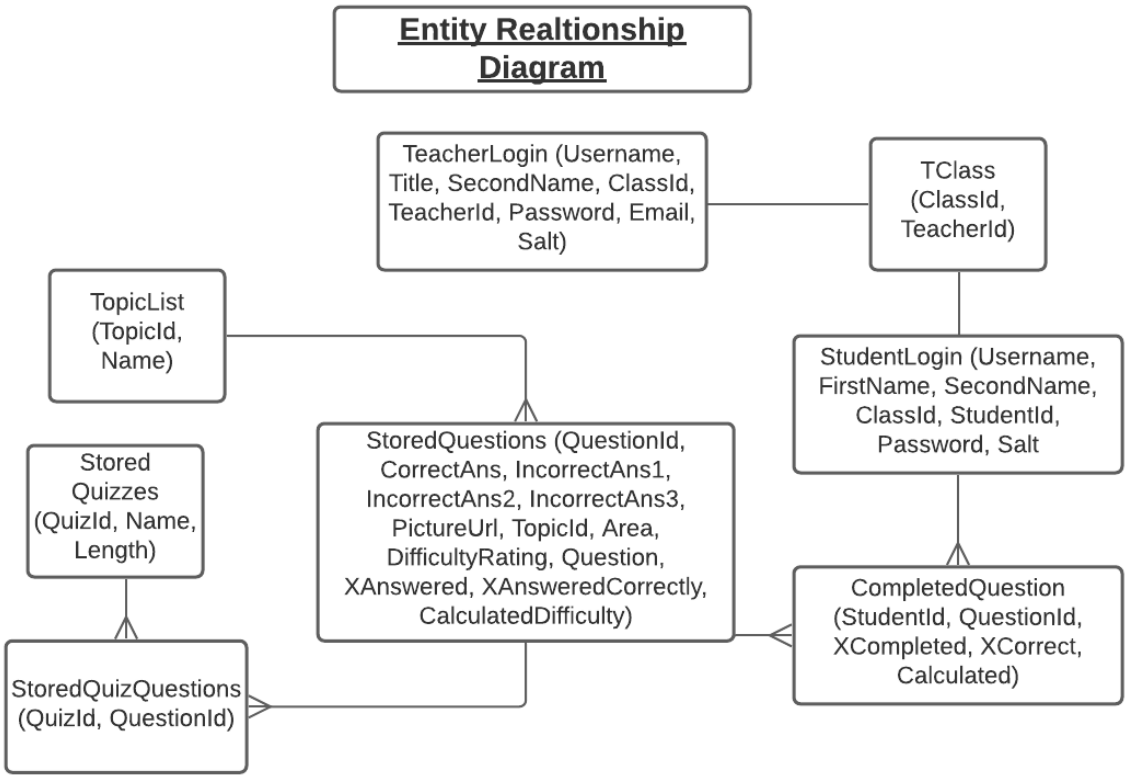
StudentLogin						
Username	FirstName	SecondName	ClassID	StudentId	Password	Salt

TClass	
Id	TeacherId

TeacherLogin							
Username	Title	SecondName	ClassId	TeacherId	Password	Email	Salt

TopicList	
Id	Name

Entity Relationship Diagram



Objective 15

Algorithms Design

Register Student

IF Details <> “” THEN
IF StudentLogin.Contains(Username = UserInputUserName) <> TRUE THEN
IF TClass.Contains(Id = UserInputClassCode) = TRUE THEN
IF UserInputPassword = ValidPassword THEN
Salt ← RandomString(15)
EncryptedPassword ← Cryptography.SHA256(Password + Salt)
Student ← AttemptLogin(Username, EncrypedPassword)
StudentLogin.Add(Details)
ELSE
PRINT “Invalid Password”
END IF
ELSE
PRINT “Invalid Class Code”
END IF
ELSE

```
                PRINT "Username is already taken"
        END IF
ELSE
        PRINT "Please enter your details."
END IF
```

The user must first input details so a presence check is conducted. Then the username is checked to see if it is taken in the table (Lookup Check). Next the class code is checked (Lookup Check). Finally, the password is checked (Validation Check). If the data is incorrect or already used then an error message is returned, otherwise, their password is encrypted and their details are added to the table StudentLogin.

Attempting Student Login

```
Salt ← GetSalt(Username)
EncryptedPassword ← Cryptography.SHA256(Password + Salt)
Student ← AttemptLogin(Username, EncrypedPassword)
IF Student <> NULL
        RETURN Student
ELSE
        PRINT "Incorrect Login Details"
END IF
```

When logging in the user enters their username and password. As all passwords are encrypted, we must first encrypt their password. This is done by adding the salt (a string of 15 characters that has been randomly generated) to the end of their password then encrypting it using the SHA256 cryptography standard. Then the username and encrypted password is compared against the SQL table StudentLogin. If it is found, it returns the student`s details, otherwise it prints an error message.

Register Teacher

```
IF Details <> "" THEN
        IF TeacherLogin.Contains(Username = UserInputUserName) <> TRUE THEN
                IF UserInputPassword = ValidPassword THEN
                        Salt ← RandomString(15)
                        EncryptedPassword ← Cryptography.SHA256(Password + Salt)
                        Student ← AttemptLogin(Username, EncrypedPassword)
                        TeacherLogin.Add(Details)
                        PRINT ClassId
                ELSE
                        PRINT "Invalid Password"
                END IF
        ELSE
                PRINT "Username is already taken"
        END IF
ELSE
        PRINT "Please enter your details."
END IF
```

The user must first input details so a presence check is conducted. Then the username is checked to see if it is taken in the table (Lookup Check). Finally, the password is checked (Validation

Check). If the data is incorrect or already used then an error message is returned, otherwise, their password is encrypted and their details are added to the table TeacherLogin. Their generated class code is then returned to them so that they can tell their students which class they are in.

Question Form

```
Answers[] =←Answers.Randomise();
AnswerRadioButton1 = Answers [0]
AnswerRadioButton2 = Answers [1]
AnswerRadioButton3 = Answers[2]
AnswerRadioButton4 = Answers [3]
```

The user must first input details so a presence check is conducted. Then the username is checked to see if it is taken in the table (Lookup Check). Finally, the password is checked (Validation Check). If the data is incorrect or already used then an error message is returned, otherwise, their password is encrypted and their details are added to the table TeacherLogin. Their generated class code is then returned to them so that they can tell their students which class they are in.

Retrieve Quiz Questions

```
Topics[] = USERINPUT
Area[] = USERINPUT
Difficulty[] = USERINPUT
SearchTerm = USERINPUT
FOREACH StoredQuestion Question in StoredQuestions
        IF Question.Topic = Topic AND Question.Area = Area AND Question.Difficulty =
        Difficulty AND Question.Question = SearchTerm THEN
                Questions.Add(Question)
        END IF
END LOOP
RETURN Questions
```

The StoredQuestions will be stored in the database under the table stored questions. The user can filter which questions they will see by selecting various criteria. If they don`t filter it in any category then it will select all questions.

Check Answer and Calculate DIfficulty

```
IF SelectedRadioButton.Text = Question.CorrectAns THEN
        Question. XCorrect ← Question. XCorrect + 1
        CompletedQuestion. XCorrect ← CompletedQuestion. XCorrect + 1
END IF
Question. XAnswered ← Question. XAnswered + 1
CompletedQuestion. XAnswered ← CompletedQuestion. XAnswered + 1
Question. CalculatedDifficulty ← (Question. XCorrect / Question. XAnswered) * 100
CompletedQuestion. CalculatedDifficulty ← (CompletedQuestion. XCorrect /
CompletedQuestion. XAnswered) * 100
```

Every time the user answers a question their score is recorded. If they get the question correct then the tally increases for correct answer and the tally for times answered also increases. If incorrect, the only tally that increases is the answered tally. The calculated difficulty value is then worked out and is a percentage of times answered correctly. It is calculated for completed question which is the student`s personal score and tracks their progress and is also calculated for StoredQuestions which is the overall difficulty rating of the question. This value is used for

creating other quizzes and filtering the questions by the ones that are answered incorrectly more often.

Adding a Question to a Quiz

IF Questions.Contains(NewQuestion) THEN
PRINT "Error, Question Already in Quiz."
ELSE IF Questions.Count > 15
PRINT "Error, Max Question Count Reached."
ELSE
Questions.Add(NewQuestion)
END IF

Each time a question is added to the quiz it needs to be checked against the existing questions in the quiz to ensure that there are no duplicate questions in the quiz. It is then checked to be under 15 entries as exceeding this would cause a range exception. If all the previous criteria have been met then the question will be added to the quiz.

Adaptive Questions Order

IF SelectedMode = "Adaptive Questions Order" THEN
FOREACH Completed Question CQ in CompletedQuestions
IF CQ.CalculatedDifficulty > 80 AND CQ.XCompleted > 5
StoredQuizQuestions.Remove(WHERE CQ.QuestionId)
ENDIF
ENDFOR
ENDIF

When starting the quiz, the user has the option to choose which question order they would like it in. If they want it to be in an adaptive order where the questions, they answer are the ones that they get wrong the most. In order for the question to be removed, it must fit the following criteria: be answered correctly more that 80% of the time and be answered more than 5 times.

Results Email in HTML Design

Question Breakdown

Question	Area	Topic	Score	Times Answered	Times Correct

Area Results

Recall	Calculations

Topic Results

Particles	Waves	Mechanics	Materials	Electricity

(Objective 15)

Data Dictionary for Non-OOP Variables

Field	Data Type	Length	Description	Validation Check	Validation Description	Valid Data	Erroneous Data
ValidLogin	Boolean	N/A	Ensures the user has input valid login credentials before proceeding	N/A	N/A	True or False	Not true or false
FormClosing	Boolean	N/A	Check to see if the user wants to close form	N/A	N/A	True or False	Not True or False
Message	String	N/A	Displays a message to the user	N/A	N/A	Any character	“”
SelectedMode	int	1 Character	Stores which difficulty mode the user wants to use – 1 = Pre-defined Difficulty, 2 = Machine generated difficulty	List	Only 1 or 2	1 or 2	3

Code – Login Screens

Student Login

```
using PhysicsQuiz1._0.Classes;
using PhysicsQuiz1._0.StudentForms;
using System;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.LoginScreen
{
    public partial class LoginStudent : Form
    {
        bool validlogin = false;
        public StartLogin Fom;
        public LoginStudent(StartLogin Frm)
        {
            InitializeComponent();
            Fom = Frm; //The login form is saved to the global variable
        }

        private void FirstNameInsertBox_TextChanged(object sender, EventArgs e)
        {

        }

        private void label3_Click(object sender, EventArgs e)
        {

        }

        private void RegisterButton_Click(object sender, EventArgs e)
        {
            DataAccess db = new DataAccess(); //The class data access is initilized

            StudentLogin user = db.AttemptStudentLogin(UsernameInsertBox.Text,
PasswordInsertBox.Text); //The method attempy student login is called,

//this takes in the parameters of the username and the password

//that the user has just input


            if (user == null)
            {
                //If the class returned an invalid login then it defaults to a blank user
and therefore the login
                //is not authorized and the form display an incorrect login message
                MessageBox.Show("Incorrect Login Credentials", "Error",
MessageBoxButtons.OK);
            }
            else
            {
                //If a valid user login was input then the form wipes the current form clean
and opens the student home screen
                //passing in the parameters of the returned login information
                validlogin = true;
                UsernameInsertBox.Text = "";
                PasswordInsertBox.Text = "";

                StudentHomescreen StHome = new StudentHomescreen(user);
                StHome.Show();
```

```
                this.Close();
            }

        }

        private void BackButton_Click(object sender, EventArgs e)
        {
            //Displays the previous form and closes this one
            Fom.Show();
            this.Close();
        }

        private void PasswordInsertBox_KeyDown(object sender, KeyEventArgs e)
        {
            //If the student presses enter in the password input box this code will trigger
the login
            if (e.KeyCode == Keys.Enter)
            {
                RegisterButton_Click(sender, e);
            }
        }

        private void LoginStudent_Load(object sender, EventArgs e)
        {

        }

        protected override void OnFormClosing(FormClosingEventArgs e)
        {
            //When the form is closed the previous form is displayed and this one is closed.
            if (validlogin != true)
            {
                Fom.Show();
            }
            base.OnFormClosing(e);
        }
    }
}
```

Register Student

```
using PhysicsQuiz1._0.Classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.LoginScreen
{
    public partial class RegisterStudent : Form
    {
        StartLogin Fom;
        public RegisterStudent(StartLogin Frm)
        {
            InitializeComponent();
            Fom = Frm; //The login form is saved to the global variable
```

```
        }

        private void FirstNameInsertBox_TextChanged(object sender, EventArgs e)
        {

        }

        private void RegisterButton_Click(object sender, EventArgs e)
        {
            var db = new DataAccess(); //The data access class is created


            if (checkdetails() == true) //The function check ddetails is ran. If it returns
true then the
                                        //following code is executed.
            {
                if (db.CheckStudentUsername(UsernameInsertBox.Text) == false) //The username
is checked against
                                                                              //the database
to check it isn`t taken
                {
                    if (db.CheckClassCode(int.Parse(ClassCodeInsertBox.Text)) == true) //The
class code is checked

//against the database to check
                                                                                      //if
it is valid
                    {
                        db.CreateStudent(FirstNameInsertBox.Text, SurnameInsertBox.Text,
UsernameInsertBox.Text,
                            PasswordInsertBox.Text, int.Parse(ClassCodeInsertBox.Text));
                        //The method create student is ran. It adds the student`s details to
the database

                        FirstNameInsertBox.Text = "";
                        SurnameInsertBox.Text = "";
                        UsernameInsertBox.Text = "";
                        PasswordInsertBox.Text = "";
                        ClassCodeInsertBox.Text = "";
                        //All assets are reset for the next student to register
                    }
                    else
                    {
                        //Message will display if the user inputs the incorrect class code
                        MessageBox.Show("Incorrect Class Code, Please Check and Try Again",
"Error", MessageBoxButtons.OK);
                    }
                }
                else
                {
                    //Message will display if the username is already taken
                    MessageBox.Show("Username is taken, please enter a different username",
"Error", MessageBoxButtons.OK);
                }
            }
        }

        private void BackButton_Click(object sender, EventArgs e)
        {
            //Closes this form and displays the previous form
            Fom.Show();
            this.Hide();
```

```
        }

        private bool checkdetails()
        {
            //Checks if all fields have had data input.
            if (UsernameInsertBox.Text == "")
            {
                MessageBox.Show( "Please Enter a username", "Error", MessageBoxButtons.OK);
                return false;
            }
            else if (ClassCodeInsertBox.Text == "" ||
ClassCodeInsertBox.Text.Any(char.IsLetter) == true) //Checks if the string

//contains any letters

//which a class code cannot
            {
                MessageBox.Show( "Please Enter a Class Code", "Error",
MessageBoxButtons.OK);
                return false;
            }
            else if (SurnameInsertBox.Text == "" || SurnameInsertBox.Text.Any(char.IsDigit)
== true) //Checks if the surname has

//any numbers in it which isn`t

//allowed
            {
                MessageBox.Show( "Please Enter a Surname", "Error", MessageBoxButtons.OK);
                return false;
            }
            else if (FirstNameInsertBox.Text == "" ||
FirstNameInsertBox.Text.Any(char.IsDigit) == true) //Checks if the firstname

//has any numbers in it which

//isn`t allowed
            {
                MessageBox.Show( "Please Enter a First name", "Error",
MessageBoxButtons.OK);
                return false;
            }
            else if (PasswordInsertBox.Text == "" ||
!PasswordInsertBox.Text.Any(char.IsUpper) ||
                !PasswordInsertBox.Text.Any(char.IsDigit) || PasswordInsertBox.Text.Length <
8 || PasswordInsertBox.Text.Length > 15)
            {
                //All passwords must meet a criteria of having an uppercase letter, a
lowercase letter, a digit, and a
                //length between 8 and 15 characters
                MessageBox.Show( "Please Enter a Valid Password", "Error",
MessageBoxButtons.OK);
                return false;
            }
            else
            {
                //If none of the conditions previously are met which make the details
invalid, then the function retuns true.
                //Otherwise the value returned is false
                return true;
            }
        }
```

```
        private void RegisterStudent_Load(object sender, EventArgs e)
        {

        }

        protected override void OnFormClosing(FormClosingEventArgs e)
        {
            //Closes this form and displays the previous form
            Fom.Show();
            base.OnFormClosing(e);
        }
    }
}
```

Register Teacher

```
using PhysicsQuiz1._0.Classes;
using System;
using System.Linq;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.LoginScreen
{
    public partial class RegisterTeacher : Form
    {
        StartLogin Fom;
        public RegisterTeacher(StartLogin Frm)
        {
            InitializeComponent();
            Fom = Frm; //The login form is saved to the global variable
        }

        private void RegisterButton_Click(object sender, EventArgs e)
        {
            DataAccess db = new DataAccess(); //The data access class is created

            if (checkdetails() == true) //The function check ddetails is ran. If it returns
true then the following code is
                                        //executed.
            {
                if (db.CheckTeacherUsername(UsernameTextBox.Text) == false) //The username
is checked against the database to
                                                                            //check it isn`t
taken
                {
                    db.CreateNewTeacher(TitleSelectComboBox.Text, SurnameTextBox.Text,
UsernameTextBox.Text,
                        PasswordTextBox.Text, EmailTextBox.Text);
                    //The method create teacher is ran. It adds the teacher`s details to the
database
                    TitleSelectComboBox.SelectedItem = null;
                    SurnameTextBox.Text = "";
                    UsernameTextBox.Text = "";
                    PasswordTextBox.Text = "";
                    EmailTextBox.Text = "";
                    //All assets are reset for the next teacher to register
                }
                else
                {
                    //Message will display if the username is already taken
```

```
                    MessageBox.Show("Username is taken, please enter a different username",
"Error", MessageBoxButtons.OK);
                }
            }
        }

        private void BackButton_Click(object sender, EventArgs e)
        {
            //Closes this form and displays the previous form
            Fom.Show();
            this.Hide();
        }

        private bool checkdetails()
        {
            //Checks if all fields have had data input.
            if (UsernameTextBox.Text == "")
            {
                MessageBox.Show("Please Enter a username", "Error", MessageBoxButtons.OK);
                return false;
            }
            else if (EmailTextBox.Text == "" || IsValidEmail(EmailTextBox.Text) == false)
//Checks to see if the email

//entered is valid by running the

//IsVaildEmail function
            {
                MessageBox.Show("Please Enter a Valid Email", "Error",
MessageBoxButtons.OK);
                return false;
            }
            else if (SurnameTextBox.Text == "" || SurnameTextBox.Text.Any(char.IsDigit) ==
true) //Checks if the surname has

//any numbers in it which

//isn`t allowed
            {
                MessageBox.Show("Please Enter a Surname", "Error", MessageBoxButtons.OK);
                return false;
            }
            else if (TitleSelectComboBox.SelectedItem == null) //Checks for data input
            {
                MessageBox.Show("Please Select a Title", "Error", MessageBoxButtons.OK);
                return false;
            }
            else if (PasswordTextBox.Text == "" || !PasswordTextBox.Text.Any(char.IsUpper)
||
                !PasswordTextBox.Text.Any(char.IsDigit) || PasswordTextBox.Text.Length < 8
|| PasswordTextBox.Text.Length > 15)
            {
                //All passwords must meet a criteria of having an uppercase letter, a
lowercase letter, a digit, and a length
                //between 8 and 15 characters
                MessageBox.Show("Please Enter a Valid Password", "Error",
MessageBoxButtons.OK);
                return false;
            }
            else
            {
```

```
                //If none of the conditions previously are met which make the details
invalid, then the function retuns true.
                //Otherwise the value returned is false
                return true;
            }
        }

        bool IsValidEmail(string email)
        {
            try
            {
                //Trys to convert the text input to an email address
                var e = new System.Net.Mail.MailAddress(email);
                return e.Address == email;
            }
            catch
            {
                //If it is unable to convert it to an email it will throw an error which
will be caught here.
                //A false will then be returned as it is an invalid email
                return false;
            }
        }

        private void RegisterTeacher_Load(object sender, EventArgs e)
        {

        }

        protected override void OnFormClosing(FormClosingEventArgs e)
        {
            //Closes this form and displays the previous form
            Fom.Show();
            base.OnFormClosing(e);
        }
    }
}
```

Start Login

```
using PhysicsQuiz1._0.Classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.LoginScreen
{
    public partial class StartLogin : Form
    {
        public StartLogin()
        {
            //The first form displayed to the user. It contains the different login and
register screens for them to use
            InitializeComponent();
        }
```

```
        private void RegisterAsTeacherButton_Click(object sender, EventArgs e)
        {
            //Launches the register teacher form
            Form SecForm = new RegisterTeacher(this);
            SecForm.Show();
            this.Hide();
        }

        private void LoginAsTeacherButton_Click(object sender, EventArgs e)
        {
            //Launchest the login teacher form
            Form SecForm = new TeacherLoginForm(this);
            SecForm.Show();
            this.Hide();
        }

        private void StudentLoginButton_Click(object sender, EventArgs e)
        {
            //Launches the login student form
            Form SecForm = new LoginStudent(this);
            SecForm.Show();
            this.Hide();
        }

        private void RegisterStudentButton_Click(object sender, EventArgs e)
        {
            //Launches the register student form
            Form SecForm = new RegisterStudent(this);
            SecForm.Show();
            this.Hide();
        }

        private void StartLogin_Load(object sender, EventArgs e)
        {

        }

        private void StartLogin_FormClosed(object sender, FormClosedEventArgs e)
        {

        }

        private void StartLogin_FormClosing(object sender, FormClosingEventArgs e)
        {

        }
    }
}
```

Teacher Login Form

```
using PhysicsQuiz1._0.Classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.LoginScreen
{
    public partial class TeacherLoginForm : Form
    {
        bool validlogin = false;
        StartLogin Fom;
        public TeacherLoginForm(StartLogin Frm)
        {
            InitializeComponent();
            Fom = Frm;  //The login form is saved to the global variable
        }

        private void LoginButton_Click(object sender, EventArgs e)
        {
            DataAccess db = new DataAccess();  //The class data access is initilized
            var Teach = new TeacherLogin(); //User is created upon the teacher class
            Teach = db.AttemptTeacherLogin(UsernameInsertBox.Text, PasswordInsertBox.Text);
//The teach is assigned

//the value returned from the

//method attemptstudent login
            if (Teach == null)
            {
                //If the teacher couldn`t be found based upon the details that have been
input then the method will return a
                //blank teacherlogin and this message will be displayed
                MessageBox.Show("Incorrect Login Credentials", "Error",
MessageBoxButtons.OK);
            }
            else
            {
                //Otherwise it will be a valid login and the teacher`s details will be
passed on to the next form and the
                //other parts of this login form will be reset
                validlogin = true;
                UsernameInsertBox.Text = "";
                PasswordInsertBox.Text = "";
            }
        }

        private void BackButton_Click(object sender, EventArgs e)
        {
            //Displays the previous form and closes this one

            Fom.Show();
            this.Hide();
        }

        private void TeacherLoginForm_Load(object sender, EventArgs e)
        {

        }

        protected override void OnFormClosing(FormClosingEventArgs e)
        {
            //When the form is closed the previous form is displayed and this one is closed.
```

```
            if (validlogin != true)
            {
                Fom.Show();
            }
            base.OnFormClosing(e);
        }
    }
}
```

Code - Quiz Forms

Auto Create Form

```
using PhysicsQuiz1._0.Classes;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.StudentForms
{
    public partial class AutoCreateForm : Form
    {
        int selectedmode = 0;
        List<StoredQuestions> questions = new List<StoredQuestions>(); //Holds all of the
relevant stored questions based upon criteria
        List<StoredQuestions> AllQuestions = new List<StoredQuestions>(); //Holds the all of
the stored questions

        public event EventHandler ClosedPage; //Triggers an event from when the form is
closed
        bool formclosing = false; //A boolean used to hold if the form is closing or not.

        public AutoCreateForm()
        {
            InitializeComponent();
            DifficultyCheckBox.Hide();
        }

        private void DifficultyTypeComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            //Changes the contents of the difficulty combo box based upon which option the
user selects
            DifficultyCheckBox.Show();
            if (DifficultyTypeComboBox.SelectedItem.ToString() == "Pre-defined Difficulty
Setting")
            {
                selectedmode = 1;
                DifficultyCheckBox.Items.Clear();
                DifficultyCheckBox.Items.Add("1");
                DifficultyCheckBox.Items.Add("2");
                DifficultyCheckBox.Items.Add("3");
                DifficultyCheckBox.Height = 49;
```

```
            }
            else if (DifficultyTypeComboBox.SelectedItem.ToString() == "Machine Generated
Difficulty Setting")
            {
                selectedmode = 2;
                DifficultyCheckBox.Items.Clear();
                DifficultyCheckBox.Items.Add("Advanced");
                DifficultyCheckBox.Items.Add("Hard");
                DifficultyCheckBox.Items.Add("Average");
                DifficultyCheckBox.Items.Add("Easy");
                DifficultyCheckBox.Height = 64;
            }
            else
            {
                selectedmode = 0;
                DifficultyCheckBox.Hide();
            }

        }

        private void SearchButton_Click(object sender, EventArgs e)
        {
            int numberselected;
            QuestionClass qc = new QuestionClass();
            SearchCriteria sc = new SearchCriteria();

            try
            {
                //The user specifies number of questions that they want the quiz to contain
                if (int.Parse(NumberOfQuestionsTextBox.Text) <= 15 &&
int.Parse(NumberOfQuestionsTextBox.Text) >= 3)
                {
                    numberselected = int.Parse(NumberOfQuestionsTextBox.Text);
                }
                else
                {
                    MessageBox.Show("Outside bounds, please enter a value between 3 and 15",
"Error", MessageBoxButtons.OK);
                    return;
                }
            }
            catch (Exception)
            {
                //If the user doesn`t input a correct number of questions then the exception
is thrown
                MessageBox.Show("Invaid Number Entered, please enter a value between 3 and
15", "Error", MessageBoxButtons.OK);
                return;
            }

            if ((TopicCheckedListBox.CheckedItems.Count == 0) ||
(TopicCheckedListBox.CheckedItems.Count == 5))
            {
                //If no question topics or all of them are selected then all of the topics
are selected
                sc.Topic1 = 1;
                sc.Topic2 = 2;
                sc.Topic3 = 3;
                sc.Topic4 = 4;
                sc.Topic5 = 5;
            }
            else
```

```
            {
                //Otherwise each question must be checked individually to see if it has been
selected
                if (TopicCheckedListBox.CheckedItems.Contains("Particles"))
                {
                    sc.Topic1 = 1;
                }

                if (TopicCheckedListBox.CheckedItems.Contains("Waves"))
                {
                    sc.Topic2 = 2;
                }

                if (TopicCheckedListBox.CheckedItems.Contains("Mechanics"))
                {
                    sc.Topic3 = 3;
                }


                if (TopicCheckedListBox.CheckedItems.Contains("Materials"))
                {
                    sc.Topic4 = 4;
                }


                if (TopicCheckedListBox.CheckedItems.Contains("Electricity"))
                {
                    sc.Topic5 = 5;
                }
            }

            if (AreaCheckedListBox.CheckedItems.Count == 0 ||
AreaCheckedListBox.CheckedItems.Count == 2)
            {
                //If no question areas or all of them are selected then all of the areas are
selected
                sc.Area = 1;
                sc.Area1 = 2;
            }
            else
            {
                //Otherwise each question must be checked individually to see if it has been
selected
                if (AreaCheckedListBox.CheckedItems.Contains("Recall"))
                {
                    sc.Area = 1;
                    sc.Area1 = 1;
                }
                else
                {
                    sc.Area = 2;
                    sc.Area1 = 2;
                }
            }

            //The selected difficulty must be chosen and it will then assign values to
search criteria based upon it
            if (selectedmode == 2)
            {
                sc = GeneratedDifficulty(sc);
                AllQuestions = qc.GetQuestionsForSearch(sc, true);
            }
```

```
            else
            {
                sc = PredefDifficultySearch(sc);
                AllQuestions = qc.GetQuestionsForSearch(sc, false);
            }

            //The stored questions are reshuffled by this line of code
            List<StoredQuestions> ShuffledQuizQuestions = AllQuestions.OrderBy(x =>
Guid.NewGuid()).ToList();
            questions = new List<StoredQuestions>();

            //The first to specified number by the user number of questions is saved to the
list to be returned containing the quiz questions
            int count = 0;
            while(count < int.Parse(NumberOfQuestionsTextBox.Text) &&
ShuffledQuizQuestions.Count() > count)
            {
                questions.Add(ShuffledQuizQuestions.ElementAt(count));
                count++;
            }

            //Display members are clarified
            QuestionListBox.DataSource = questions;
            QuestionListBox.DisplayMember = "Question";

            //Displays how many question have been selected from the specified count in case
there weren`t enough questions based upon their criteria
            InformationLabel.Text = "Selected " + questions.Count + " questions from
criteria returning " + AllQuestions.Count() + " Questions";
        }

        public SearchCriteria GeneratedDifficulty(SearchCriteria sc)
        {
            //If the user selects Generated difficulty this function is called in order to
save the correct data to the Search Criteria
            if (DifficultyCheckBox.CheckedItems.Count == 4 ||
DifficultyCheckBox.CheckedItems.Count == 0)
            {
                //If the user doesn`t select a difficulty then all are selected
                sc.Difficulty = 1;
                sc.Difficulty1 = 2;
                sc.Difficulty2 = 3;
                sc.Difficulty3 = 4;
            }
            else
            {
                //Otherwise the selected difficulties are added to the search criteria
                if (DifficultyCheckBox.CheckedIndices.Contains(0))
                {
                    sc.Difficulty = 1;
                }
                if (DifficultyCheckBox.CheckedItems.Contains(1))
                {
                    sc.Difficulty1 = 2;
                }
                if (DifficultyCheckBox.CheckedItems.Contains(2))
                {
                    sc.Difficulty2 = 3;
                }
                if (DifficultyCheckBox.CheckedItems.Contains(3))
                {
                    sc.Difficulty3 = 4;
```

```
                }
            }

            return sc;
        }

        private SearchCriteria PredefDifficultySearch(SearchCriteria sc)
        {
            //If the user selects predefined difficulty this function is called in order to
save the correct data to the Search Criteria
            if ((DifficultyCheckBox.CheckedItems.Count == 3) ||
(DifficultyCheckBox.CheckedItems.Count == 0))
            {
                sc.Difficulty = 1;
                sc.Difficulty1 = 2;
                sc.Difficulty2 = 3;
            }
            else
            {
                if (DifficultyCheckBox.CheckedItems.Contains("1"))
                {
                    sc.Difficulty = 1;
                }

                if (DifficultyCheckBox.CheckedItems.Contains("2"))
                {
                    sc.Difficulty1 = 2;
                }

                if (DifficultyCheckBox.CheckedItems.Contains("3"))
                {
                    sc.Difficulty2 = 3;
                }
            }

            return sc;
        }

        private void ReshuffleButton_Click(object sender, EventArgs e)
        {
            //Selects a different number of questions from the specified criteria
            List<StoredQuestions> ShuffledQuizQuestions = AllQuestions.OrderBy(x =>
Guid.NewGuid()).ToList();
            questions = new List<StoredQuestions>();

            int count = 0;
            while (count < int.Parse(NumberOfQuestionsTextBox.Text) &&
ShuffledQuizQuestions.Count() > count)
            {
                questions.Add(ShuffledQuizQuestions.ElementAt(count));
                count++;
            }


            QuestionListBox.DataSource = questions;
            QuestionListBox.DisplayMember = "Question";
        }

        private void CreateQuizButton_Click(object sender, EventArgs e)
        {
            //User names the quiz
            Name = QuizNameTextBox.Text;
```

```
            if (Name == "")
            {
                MessageBox.Show("Error Enter A Quiz Name", "Error", MessageBoxButtons.OK);
                return;
            }
            else if (questions.Count() < 3 || questions.Count > 15)
            {
                //If the program has returned an invalid number of questions this error
message will be displayed
                MessageBox.Show("Error, Invalid number of items, please select between 3 and
15 questions. Remove items or create multiple quizzes.", "Error", MessageBoxButtons.OK);
                return;
            }

            //Holds the ID`s of the question that have been selected
            int[] IdNum = new int[questions.Count()];

            foreach (StoredQuestions id in questions)
            {
                IdNum[questions.IndexOf(id)] = id.QuestionId;
            }

            //The questions that have been returned based upon the criteria are passed into
the CreateQuiz Method of questionclass
            QuestionClass qc = new QuestionClass();
            qc.CreateQuiz(IdNum, Name);

            //Displays a success message
            MessageBox.Show("Quiz Created!", "Success", MessageBoxButtons.OK);

        }

        private void ReturnButton_Click(object sender, EventArgs e)
        {
            //Closes the form by clicking return button
            formclosing = true;
            this.Close();
            ClosedPage?.Invoke(this, EventArgs.Empty);
        }

        protected override void OnFormClosed(FormClosedEventArgs e)
        {
            //Closes the form by clicking the x
            if (formclosing != true)
            {
                ReturnButton_Click(this, EventArgs.Empty);
            }
            base.OnFormClosed(e);
        }
    }
}
```

Picture Question Form

```
using PhysicsQuiz1._0.Classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.StudentForms
{
    public partial class PictureQuestionForm : Form
    {
        public event EventHandler<bool> Answered; //Event for when the question has been
answered is created

        StoredQuestions CurrentQuestion = new StoredQuestions(); //The current question is
stored here

        public PictureQuestionForm(StoredQuestions CQuestions)
        {
            InitializeComponent();

            CurrentQuestion = CQuestions; //The paramter of the current question is passed
into the form

            QuestionLabel.Text = CurrentQuestion.Question; //The currentquestion`s question
is displayed

            QuestionPictureBox.Image = Image.FromFile(CurrentQuestion.PictureUrl); ; //The
currentquestion`s picture is displayed

            List<string> Answers = new List<string>(); //A new list of answers is created

            //Each individual answer that is stored in currentquestion including the correct
answer and the 3 incorrect ones
            Answers.Add(CurrentQuestion.CorrectAns);
            Answers.Add(CurrentQuestion.IncorrectAns1);
            Answers.Add(CurrentQuestion.IncorrectAns2);
            Answers.Add(CurrentQuestion.IncorrectAns3);

            //Answers are shuffled
            Answers = Answers.OrderBy(x => Guid.NewGuid()).ToList();

            //The answers are assigned to the radio buttons
            AnswerRadioButton1.Text = Answers.ElementAt(0);
            AnswerRadioButton2.Text = Answers.ElementAt(1);
            AnswerRadioButton3.Text = Answers.ElementAt(2);
            AnswerRadioButton4.Text = Answers.ElementAt(3);

        }

        private void SubmitButton_Click(object sender, EventArgs e)
        {
            RadioButton checkedButton;

            //Checks to see if the user has selected an answer
            if (AnswerRadioButton1.Checked == true)
            {
                checkedButton = AnswerRadioButton1;
            }
            else if (AnswerRadioButton2.Checked == true)
            {
                checkedButton = AnswerRadioButton2;
            }
            else if (AnswerRadioButton3.Checked == true)
            {
                checkedButton = AnswerRadioButton3;
```

```
            }
            else if (AnswerRadioButton4.Checked == true)
            {
                checkedButton = AnswerRadioButton4;
            }
            else
            {
                return;
            }

            //Answer selected is compared against the correct answer
            //A message is displayed telling the user their result and their result is
returned
            if (checkedButton.Text == CurrentQuestion.CorrectAns)
            {
                MessageBox.Show("Correct", "Well Done", MessageBoxButtons.OK);
                Answered?.Invoke(this, true);
            }
            else
            {
                MessageBox.Show("Incorrect", "Try Again", MessageBoxButtons.OK);
                Answered?.Invoke(this, false);
            }

            this.Close(); //Form is closed
        }
    }
}
```

Student Home Screen

```
using PhysicsQuiz1._0.Classes;
using PhysicsQuiz1._0.GeneralForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.VisualStyles;

namespace PhysicsQuiz1._0.StudentForms
{
    public partial class StudentHomescreen : Form
    {
        private StudentLogin student;

        public StudentHomescreen(StudentLogin user)
        {
            InitializeComponent();
            student = user; //The students login info is made global in the form
            WelcomeLabel.Text = $"Welcome {user.FirstName}, Please Select A Mode."; //A
welcome message us displayed
        }

        private void WelcomeLabel_Click(object sender, EventArgs e)
        {
```

```
        }

        private void button1_Click(object sender, EventArgs e)
        {

        }

        private void tableLayoutPanel1_Paint(object sender, PaintEventArgs e)
        {

        }

        private void ViewQuestionsButton_Click(object sender, EventArgs e)
        {
            //Launches the viewquestionsform
            ViewAllQuestionsForm pq = new ViewAllQuestionsForm();
            this.Hide();
            pq.Show();
            pq.ClosedPage += (source, EventArgs) =>
            {
                this.Show();
            };

            }

        private void StudentHomescreen_Load(object sender, EventArgs e)
        {

        }

        protected override void OnFormClosing(FormClosingEventArgs e)
        {
            //Closes all the remaining open forms
            Application.OpenForms[0].Close();
            base.OnFormClosing(e);
        }

        private void button13_Click(object sender, EventArgs e)
        {
            //Launches the GenerateQuizManual form
            GenerateQuizManual gq = new GenerateQuizManual();
            this.Hide();
            gq.Show();
            gq.ClosedPage += (source, EventArgs) =>
            {
                this.Show();
            };
        }

        private void button3_Click(object sender, EventArgs e)
        {
            //Launches the storedquizzes form
            ViewStoredQuizzes SQ = new ViewStoredQuizzes();
            this.Hide();
            SQ.Show();
            SQ.ClosedPage += (source, EventArgs) =>
            {
                this.Show();
            };

            SQ.SelectedQuiz += ViewStatsCall;
        }
```

```
        private void ViewStatsCall(ViewStoredQuizzes VS,StoredQuizzes storedQuizzes,
List<StoredQuestions> storedQuestions, List<StoredQuizQuestions> storedQuizQuestions)
        {
            //Launches the viewstats form
            ViewStats vs = new ViewStats(student, storedQuizzes, storedQuestions,
storedQuizQuestions);
            vs.Show();

            vs.FormClosed += (source, EventArgs) =>
            {
                this.Show();
            };

            vs.OpenStoredQuizzes += (source, EventArgs) =>
            {
                //The user can specify to return to the storedquizzes, if they wish to do
that this event is called
                button3_Click(null, EventArgs.Empty);
            };
        }

        private void GenerateNewQuizAutoButton_Click(object sender, EventArgs e)
        {
            //Launches the auto create quiz form
            AutoCreateForm ACF = new AutoCreateForm();
            this.Hide();
            ACF.Show();
            ACF.ClosedPage += (source, EventArgs) =>
            {
                this.Show();
            };
        }
    }
}
```

Text Question Form

```
using PhysicsQuiz1._0.Classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.StudentForms
{
    public partial class TextQuestionForm : Form
    {
        public event EventHandler<bool> Answered; //Event for when the question has been
answered is created

        StoredQuestions CurrentQuestion = new StoredQuestions(); //The current question is
stored here

        public TextQuestionForm(StoredQuestions CQuestions)
```

```
        {
            InitializeComponent();

            CurrentQuestion = CQuestions; //The paramter of the current question is passed
into the form

            QuestionLabel.Text = CurrentQuestion.Question; //The currentquestion`s question
is displayed

            List<string> Answers = new List<string>(); //A new list of answers is created

            //Each individual answer that is stored in currentquestion including the correct
answer and the 3 incorrect ones
            Answers.Add(CurrentQuestion.CorrectAns);
            Answers.Add(CurrentQuestion.IncorrectAns1);
            Answers.Add(CurrentQuestion.IncorrectAns2);
            Answers.Add(CurrentQuestion.IncorrectAns3);

            //Answers are shuffled
            Answers = Answers.OrderBy(x => Guid.NewGuid()).ToList();

            //The answers are assigned to the radio buttons
            AnswerRadioButton1.Text = $"{Answers.ElementAt(0)}";
            AnswerRadioButton2.Text = $"{ Answers.ElementAt(1)}";
            AnswerRadioButton3.Text = $"{ Answers.ElementAt(2)}";
            AnswerRadioButton4.Text = $"{ Answers.ElementAt(3)}";
        }

        private void SubmitButton_Click(object sender, EventArgs e)
        {
            RadioButton checkedButton;

            //Checks to see if the user has selected an answer
            if (AnswerRadioButton1.Checked == true)
            {
                checkedButton = AnswerRadioButton1;
            }
            else if (AnswerRadioButton2.Checked == true)
            {
                checkedButton = AnswerRadioButton2;
            }
            else if (AnswerRadioButton3.Checked == true)
            {
                checkedButton = AnswerRadioButton3;
            }
            else if (AnswerRadioButton4.Checked == true)
            {
                checkedButton = AnswerRadioButton4;
            }
            else
            {
                return;
            }

            //Answer selected is compared against the correct answer
            //A message is displayed telling the user their result and their result is
returned
            if (checkedButton.Text == CurrentQuestion.CorrectAns)
            {
                MessageBox.Show("Correct", "Well Done", MessageBoxButtons.OK);
                Answered?.Invoke(this, true);
            }
```

```
            else
            {
                MessageBox.Show("Incorrect", "Try Again", MessageBoxButtons.OK);
                Answered?.Invoke(this, false);
            }

            this.Close(); //Form is closed
        }

        private void tableLayoutPanel1_Paint(object sender, PaintEventArgs e)
        {

        }
    }

}
```

Generate Quiz Manual

```
using PhysicsQuiz1._0.Classes;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.GeneralForms
{
    public partial class GenerateQuizManual : Form
    {
        int selectedmode = 0;

        List<StoredQuestions> questions = new List<StoredQuestions>(); //Holds all of the
relevant stored questions based upon criteria
        List<StoredQuestions> AllQuestions = new List<StoredQuestions>(); //Holds the all of
the stored questions
        ObservableCollection<StoredQuestions> NewQuiz = new
ObservableCollection<StoredQuestions>(); //Holds the selected quiz questions by the user

        public event EventHandler ClosedPage; //Triggers an event from when the form is
closed
        bool formclosing = false; //A boolean used to hold if the form is closing or not.

        public GenerateQuizManual()
        {
            InitializeComponent();
            QuestionClass qc = new QuestionClass();
            AllQuestions = qc.LoadAllQuestions(); //Loads the questions from the SQL
database
            QuestionListBox.DataSource = AllQuestions; //Adds the questions to the question
list box
            QuestionListBox.DisplayMember = "DisplayItem";
            QuestionHeaderLabel.Hide();
            AnswerHeaderLabel.Hide();
            DifficultyCheckBox.Hide();
            //Sets the question to hide the empty question details
```

```
        }

        private void tableLayoutPanel1_Paint(object sender, PaintEventArgs e)
        {

        }

        private void toolStrip1_ItemClicked(object sender, ToolStripItemClickedEventArgs e)
        {

        }

        private void QuestionListBox_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            //When the user selects a question the question information is displayed over
the previous, this code completes that action
            AnswerHeaderLabel.Show();
            QuestionHeaderLabel.Show();
            if (QuestionListBox.SelectedItem == null)
            {

            }
            else
            {
                //Assigning the selected question to a variable
                StoredQuestions SelectedQuestion =
(StoredQuestions)QuestionListBox.SelectedItem;
                if (SelectedQuestion.PictureUrl == "" || SelectedQuestion.PictureUrl ==
null)
                {
                    QuestionPictureBox.Hide();
                }
                else
                {
                    QuestionPictureBox.Show();
                    QuestionPictureBox.Image = Image.FromFile(SelectedQuestion.PictureUrl);
                }

                QuestionLabel.Text = SelectedQuestion.Question;

                AnswerLabel.Text = SelectedQuestion.CorrectAns;
            }
        }

        private void SearchButton_Click_1(object sender, EventArgs e)
        {
            //All relevant question search criteria must be saved to the class
SearchCriteria
            QuestionClass qc = new QuestionClass();
            SearchCriteria sc = new SearchCriteria();

            sc.Search = SearchBarTextBox.Text;
            //The user`s selected topics are added the the criteria
            //If they have selected no topics it will select them all
            if ((TopicCheckedListBox.CheckedItems.Count == 0) ||
(TopicCheckedListBox.CheckedItems.Count == 5))
            {
                sc.Topic1 = 1;
                sc.Topic2 = 2;
                sc.Topic3 = 3;
                sc.Topic4 = 4;
                sc.Topic5 = 5;
```

```
            }
            else
            {
                if (TopicCheckedListBox.CheckedItems.Contains("Particles"))
                {
                    sc.Topic1 = 1;
                }

                if (TopicCheckedListBox.CheckedItems.Contains("Waves"))
                {
                    sc.Topic2 = 2;
                }

                if (TopicCheckedListBox.CheckedItems.Contains("Mechanics"))
                {
                    sc.Topic3 = 3;
                }


                if (TopicCheckedListBox.CheckedItems.Contains("Materials"))
                {
                    sc.Topic4 = 4;
                }


                if (TopicCheckedListBox.CheckedItems.Contains("Electricity"))
                {
                    sc.Topic5 = 5;
                }
            }
            //If the user selects no areas all of them are selected otherwise it will follow
onto
            if (AreaCheckedListBox.CheckedItems.Count == 0 ||
AreaCheckedListBox.CheckedItems.Count == 2)
            {
                sc.Area = 1;
                sc.Area1 = 2;
            }
            else
            {
                if (AreaCheckedListBox.CheckedItems.Contains("Recall"))
                {
                    sc.Area = 1;
                    sc.Area1 = 1;
                }
                else
                {
                    sc.Area = 2;
                    sc.Area1 = 2;
                }
            }

            //The selected difficulty must be chosen and it will then assign values to
search criteria based upon it
            if (selectedmode == 2)
            {
                sc = GeneratedDifficulty(sc);
                questions = qc.GetQuestionsForSearch(sc, true);
            }
            else
            {
                sc = PredefDifficultySearch(sc);
```

```
                questions = qc.GetQuestionsForSearch(sc, false);
            }


            QuestionListBox.DataSource = questions;
            QuestionListBox.DisplayMember = "DisplayItem";
        }

        private SearchCriteria PredefDifficultySearch(SearchCriteria sc)
        {
            //If the user selects predefined difficulty this function is called in order to
save the correct data to the Search Criteria
            if ((DifficultyCheckBox.CheckedItems.Count == 3) ||
(DifficultyCheckBox.CheckedItems.Count == 0))
            {
                sc.Difficulty = 1;
                sc.Difficulty1 = 2;
                sc.Difficulty2 = 3;
            }
            else
            {
                if (DifficultyCheckBox.CheckedItems.Contains("1"))
                {
                    sc.Difficulty = 1;
                }

                if (DifficultyCheckBox.CheckedItems.Contains("2"))
                {
                    sc.Difficulty1 = 2;
                }

                if (DifficultyCheckBox.CheckedItems.Contains("3"))
                {
                    sc.Difficulty2 = 3;
                }
            }

            return sc;
        }

        private SearchCriteria GeneratedDifficulty(SearchCriteria sc)
        {
            //If the user selects Generated difficulty this function is called in order to
save the correct data to the Search Criteria
            if (DifficultyCheckBox.CheckedItems.Count == 4 ||
DifficultyCheckBox.CheckedItems.Count == 0)
            {
                //If the user doesn`t select a difficulty then all are selected
                sc.Difficulty = 1;
                sc.Difficulty1 = 2;
                sc.Difficulty2 = 3;
                sc.Difficulty3 = 4;
            }
            else
            {
                //Otherwise the selected difficulties are added to the search criteria
                if (DifficultyCheckBox.CheckedIndices.Contains(0))
                {
                    sc.Difficulty = 1;
                }
                if (DifficultyCheckBox.CheckedItems.Contains(1))
                {
```

```
                    sc.Difficulty1 = 2;
                }
                if (DifficultyCheckBox.CheckedItems.Contains(2))
                {
                    sc.Difficulty2 = 3;
                }
                if (DifficultyCheckBox.CheckedItems.Contains(3))
                {
                    sc.Difficulty3 = 4;
                }
            }

            return sc;
        }

        private void AddButton_Click(object sender, EventArgs e)
        {
            //When the add button is pressed the question must be coppied from the stored
questions, to the NewQuiz collection
            if (QuestionListBox.SelectedItem == null)
            {
                return;
            }
            else if(NewQuiz.Count() == 15)
            {
                //Only 15 questions MAX are allowed per quiz so if they try to add too many
the program will display this error
                MessageBox.Show("Error, Invalid number of items, please select between 3 and
15 questions. Remove items or create multiple quizzes.", "Error", MessageBoxButtons.OK);
                return;
            }
            StoredQuestions SelectedQuestion =
(StoredQuestions)QuestionListBox.SelectedItem;

            //If the quiz already contains the question then this message will display
            if (NewQuiz.Contains(SelectedQuestion))
            {
                string message = "This question has already been added to the quiz!";
                string caption = "Error Detected in Input";
                MessageBoxButtons buttons = MessageBoxButtons.OK;
                MessageBox.Show(message, caption, buttons);
                return;
            }

            //If none of the other criteria have occured it will add the question to the
quiz
            NewQuiz.Add(SelectedQuestion);

            //Resets the variables
            QuizListBox.DataSource = null;
            QuizListBox.DataSource = NewQuiz;
            QuizListBox.DisplayMember = "Question";

            QuestionCapacityProgressBar.Increment(1);
        }

        private void RemoveButton_Click(object sender, EventArgs e)
        {
            //If no question is selected then the question cannot be removed
            if (QuizListBox.SelectedItem == null)
            {
                return;
```

```
            }

            //The selected question is saved to a variable
            StoredQuestions SelectedQuestion = (StoredQuestions)QuizListBox.SelectedItem;

            //Removes the question from the quiz
            NewQuiz.Remove(SelectedQuestion);

            QuizListBox.DataSource = null;
            QuizListBox.DataSource = NewQuiz;
            QuizListBox.DisplayMember = "Question";

            QuestionCapacityProgressBar.Increment(-1);

        }

        private void QuestionCapacityProgressBar_Click(object sender, EventArgs e)
        {
            //When the progress bar is pressed this message is displayed
            string message = $"There are {QuestionCapacityProgressBar.Value} out of 15
questions entered";
            string caption = "Information";
            MessageBoxButtons buttons = MessageBoxButtons.OK;
            MessageBox.Show(message, caption, buttons);
        }

        private void CreateQuizButton_Click(object sender, EventArgs e)
        {
            //Once the user has decided that they want to create the quiz then a few
validity checks are ran
            Name = QuizNameTextBox.Text;
            if(Name == "")
            {
                MessageBox.Show("Error Enter A Quiz Name", "Error", MessageBoxButtons.OK);
                return;
            }
            else if(NewQuiz.Count() < 3 || NewQuiz.Count > 15)
            {
                MessageBox.Show("Error, Invalid number of items, please select between 3 and
15 questions. Remove items or create multiple quizzes.", "Error", MessageBoxButtons.OK);
                return;
            }

            //If all criteria is passed then the number of questions in the quiz is added to
the int[] array with their question ID
            int[] IdNum = new int[NewQuiz.Count()];

            foreach(StoredQuestions id in NewQuiz)
            {
                IdNum[NewQuiz.IndexOf(id)] = id.QuestionId;
            }

            QuestionClass qc = new QuestionClass();

            //Queries the SQL database to create the quiz
            qc.CreateQuiz(IdNum, Name);

            MessageBox.Show("Quiz Created!", "Success", MessageBoxButtons.OK);

        }

        private void GenerateQuizManual_Load(object sender, EventArgs e)
```

```
        {

        }

        private void ReturnToMenuButton_Click(object sender, EventArgs e)
        {
            //Returns to the menu
            formclosing = true;
            this.Close();
            ClosedPage?.Invoke(this, EventArgs.Empty);
        }

        protected override void OnFormClosed(FormClosedEventArgs e)
        {
            if (formclosing != true)
            {
                ReturnToMenuButton_Click(this, EventArgs.Empty);
            }
            base.OnFormClosed(e);
        }

        private void DifficultyTypeComboBox_SelectedIndexChanged_1(object sender, EventArgs
e)
        {
            //Changes the contents of the difficulty combo box based upon which option the
user selects
            DifficultyCheckBox.Show();
            if (DifficultyTypeComboBox.SelectedItem.ToString() == "Pre-defined Difficulty
Setting")
            {
                selectedmode = 1;
                DifficultyCheckBox.Items.Clear();
                DifficultyCheckBox.Items.Add("1");
                DifficultyCheckBox.Items.Add("2");
                DifficultyCheckBox.Items.Add("3");
                DifficultyCheckBox.Height = 49;
            }
            else if (DifficultyTypeComboBox.SelectedItem.ToString() == "Machine Generated
Difficulty Setting")
            {
                selectedmode = 2;
                DifficultyCheckBox.Items.Clear();
                DifficultyCheckBox.Items.Add("Advanced");
                DifficultyCheckBox.Items.Add("Hard");
                DifficultyCheckBox.Items.Add("Average");
                DifficultyCheckBox.Items.Add("Easy");
                DifficultyCheckBox.Height = 64;
            }
            else
            {
                selectedmode = 0;
                DifficultyCheckBox.Hide();
            }
        }
    }
}
```

Send Quiz Info

```
using PhysicsQuiz1._0.Classes;
using System.Collections.Generic;
```

```
using System.Net;
using System.Net.Mail;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.GeneralForms
{
    public partial class SendQuizInfo : Form
    {
        //The user can select to send a project report to their teacher based upon their
email that was declared then they created an account.

        StudentLogin Student = new StudentLogin();
        List<StoredQuestions> SQ = new List<StoredQuestions>();
        List<CompletedQuestion> CQ = new List<CompletedQuestion>();
        CreateHTMLTable createemail = new CreateHTMLTable();
        StoredQuizzes storedquizzes = new StoredQuizzes();
        DataAccess DA = new DataAccess();


        public SendQuizInfo(StudentLogin student, List<StoredQuestions> sq,
List<CompletedQuestion> cq, StoredQuizzes storedQuizzes)
        {
            //The student, storedquestions that they have answered, their completed question
and the quiz are passed into the sub and then made global by this sub.
            InitializeComponent();
            Student = student;
            SQ = sq;
            CQ = cq;
            storedquizzes = storedQuizzes;
        }

        private void SendQuizScoresButton_Click(object sender, System.EventArgs e)
        {
            //When the send quiz button is pressed, the teacher`s email is retreived from
the database by this function
            string teacheremail = DA.GetTeacherEmail(Student.ClassId);

            //This portion of code signs into the email account created to send emails based
on google mail
            SmtpClient clientDetails = new SmtpClient();
            clientDetails.Port = 587;
            clientDetails.Host = "smtp.gmail.com";
            clientDetails.EnableSsl = true;
            clientDetails.DeliveryMethod = SmtpDeliveryMethod.Network;
            clientDetails.UseDefaultCredentials = false;
            clientDetails.Credentials = new
NetworkCredential("physicsquizemailsend@gmail.com", "M!necraft1");

            //This portion creates the mail message
            MailMessage mailDetails = new MailMessage();
            mailDetails.From = new MailAddress("physicsquizemailsend@gmail.com");
            mailDetails.To.Add(teacheremail); //The recepient`s details are added here
            mailDetails.Subject = Student.FirstName + " " + Student.SecondName + "`s Scores
for Test Named:" + storedquizzes.Name;
            mailDetails.IsBodyHtml = true;

            //This class creates the email body
            mailDetails.Body = createemail.createtable(SQ, CQ);

            //The email is then sent by this line of code here
            clientDetails.Send(mailDetails);
```

```
            MessageBox.Show("Your mail has been sent.");
        }

        private void ReturnButton_Click(object sender, System.EventArgs e)
        {
            this.Close();
        }
    }
}
```

View All Questions Form

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using PhysicsQuiz1._0.Classes;

namespace PhysicsQuiz1._0.GeneralForms
{
    public partial class ViewAllQuestionsForm : Form
    {

        bool formclosing = false;
        int selectedmode = 0;

        List<StoredQuestions> questions = new List<StoredQuestions>();
        List<StoredQuestions> AllQuestions = new List<StoredQuestions>();

        public event EventHandler ClosedPage;

        public ViewAllQuestionsForm()
        {
            InitializeComponent();
            DifficultyCheckBox.Hide();
            QuestionClass qc = new QuestionClass();
            //All of the questions are loaded from the class and are used as a base to refer
to
            AllQuestions = qc.LoadAllQuestions();
            QuestionListBox.DataSource = AllQuestions;
            QuestionListBox.DisplayMember = "DisplayItem";
            QuestionHeaderLabel.Hide();
            AnswerHeaderLabel.Hide();
            //Sets the question to hide the empty question details
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {

        }

        private void label1_Click(object sender, EventArgs e)
        {

        }
```

```
        private void ViewAllQuestionsForm_Load(object sender, EventArgs e)
        {

        }

        private void tableLayoutPanel2_Paint(object sender, PaintEventArgs e)
        {

        }

        private void tableLayoutPanel7_Paint(object sender, PaintEventArgs e)
        {

        }

        private void SearchButton_Click(object sender, EventArgs e)
        {
            //All relevant question search criteria must be saved to the class
SearchCriteria
            QuestionClass qc = new QuestionClass();
            SearchCriteria sc = new SearchCriteria();

            sc.Search = SearchBarTextBox.Text;
            //The user`s selected topics are added the the criteria
            //If they have selected no topics it will select them all
            if ((TopicCheckedListBox.CheckedItems.Count == 0) ||
(TopicCheckedListBox.CheckedItems.Count == 5))
            {
                sc.Topic1 = 1;
                sc.Topic2 = 2;
                sc.Topic3 = 3;
                sc.Topic4 = 4;
                sc.Topic5 = 5;
            }
            else
            {
                if (TopicCheckedListBox.CheckedItems.Contains("Particles"))
                {
                    sc.Topic1 = 1;
                }

                if (TopicCheckedListBox.CheckedItems.Contains("Waves"))
                {
                    sc.Topic2 = 2;
                }

                if (TopicCheckedListBox.CheckedItems.Contains("Mechanics"))
                {
                    sc.Topic3 = 3;
                }


                if (TopicCheckedListBox.CheckedItems.Contains("Materials"))
                {
                    sc.Topic4 = 4;
                }


                if (TopicCheckedListBox.CheckedItems.Contains("Electricity"))
                {
                    sc.Topic5 = 5;
                }
```

```
            }
            //If the user selects no areas all of them are selected otherwise it will follow
onto
            if (AreaCheckedListBox.CheckedItems.Count == 0 ||
AreaCheckedListBox.CheckedItems.Count == 2)
            {
                sc.Area = 1;
                sc.Area1 = 2;
            }
            else
            {
                if (AreaCheckedListBox.CheckedItems.Contains("Recall"))
                {
                    sc.Area = 1;
                    sc.Area1 = 1;
                }
                else
                {
                    sc.Area = 2;
                    sc.Area1 = 2;
                }
            }
            //The selected difficulty must be choesen and it will then assign values to
search criteria based upon it
            if (selectedmode == 2)
            {
                sc = GeneratedDifficulty(sc);
                questions = qc.GetQuestionsForSearch(sc, true);
            }
            else
            {
                sc = PredefDifficultySearch(sc);
                questions = qc.GetQuestionsForSearch(sc, false);
            }

            QuestionListBox.DataSource = questions;
            QuestionListBox.DisplayMember = "DisplayItem";
        }

        private void ReturnToMenuButton_Click(object sender, EventArgs e)
        {
            formclosing = true;
            this.Close();
            ClosedPage?.Invoke(this, EventArgs.Empty);
        }

        protected override void OnFormClosed(FormClosedEventArgs e)
        {
            if (formclosing != true)
            {
                ReturnToMenuButton_Click(this, EventArgs.Empty);
            }
            base.OnFormClosed(e);
        }

        private void DifficultyTypeComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            //Changes the contents of the difficulty combo box based upon which option the
user selects
            DifficultyCheckBox.Show();
```

```
            if (DifficultyTypeComboBox.SelectedItem.ToString() == "Pre-defined Difficulty
Setting")
            {
                selectedmode = 1;
                DifficultyCheckBox.Items.Clear();
                DifficultyCheckBox.Items.Add("1");
                DifficultyCheckBox.Items.Add("2");
                DifficultyCheckBox.Items.Add("3");
                DifficultyCheckBox.Height = 49;
            }
            else if (DifficultyTypeComboBox.SelectedItem.ToString() == "Machine Generated
Difficulty Setting")
            {
                selectedmode = 2;
                DifficultyCheckBox.Items.Clear();
                DifficultyCheckBox.Items.Add("Advanced");
                DifficultyCheckBox.Items.Add("Hard");
                DifficultyCheckBox.Items.Add("Average");
                DifficultyCheckBox.Items.Add("Easy");
                DifficultyCheckBox.Height = 64;
            }
            else
            {
                selectedmode = 0;
                DifficultyCheckBox.Hide();
            }
        }

        private SearchCriteria PredefDifficultySearch(SearchCriteria sc)
        {
            //If the user selects predefined difficulty this function is called in order to
save the correct data to the Search Criteria
            if ((DifficultyCheckBox.CheckedItems.Count == 3) ||
(DifficultyCheckBox.CheckedItems.Count == 0))
            {
                //If the user doesn`t select a difficulty then all are selected
                sc.Difficulty = 1;
                sc.Difficulty1 = 2;
                sc.Difficulty2 = 3;
            }
            else
            {
                //Otherwise the selected difficulties are added to the search criteria
                if (DifficultyCheckBox.CheckedItems.Contains("1"))
                {
                    sc.Difficulty = 1;
                }

                if (DifficultyCheckBox.CheckedItems.Contains("2"))
                {
                    sc.Difficulty1 = 2;
                }

                if (DifficultyCheckBox.CheckedItems.Contains("3"))
                {
                    sc.Difficulty2 = 3;
                }
            }

            return sc;
        }
```

```
        private SearchCriteria GeneratedDifficulty(SearchCriteria sc)
        {
            //If the user selects Generated difficulty this function is called in order to
save the correct data to the Search Criteria
            if (DifficultyCheckBox.CheckedItems.Count == 4 ||
DifficultyCheckBox.CheckedItems.Count == 0)
            {
                //If the user doesn`t select a difficulty then all are selected
                sc.Difficulty = 1;
                sc.Difficulty1 = 2;
                sc.Difficulty2 = 3;
                sc.Difficulty3 = 4;
            }
            else
            {
                //Otherwise the selected difficulties are added to the search criteria
                if (DifficultyCheckBox.CheckedIndices.Contains(0))
                {
                    sc.Difficulty = 1;
                }
                if (DifficultyCheckBox.CheckedItems.Contains(1))
                {
                    sc.Difficulty1 = 2;
                }
                if (DifficultyCheckBox.CheckedItems.Contains(2))
                {
                    sc.Difficulty2 = 3;
                }
                if (DifficultyCheckBox.CheckedItems.Contains(3))
                {
                    sc.Difficulty3 = 4;
                }
            }

            return sc;
        }





        private void QuestionListBox_SelectedIndexChanged(object sender, EventArgs e)
        {

        }

        private void QuestionListBox_DoubleClick(object sender, EventArgs e)
        {
            //When the user selects a question the question information is displayed over
the previous, this code completes that action
            AnswerHeaderLabel.Show();
            QuestionHeaderLabel.Show();
            if (QuestionListBox.SelectedItem == null)
            {

            }
            else
            {
                StoredQuestions SelectedQuestion =
(StoredQuestions)QuestionListBox.SelectedItem;
                if (SelectedQuestion.PictureUrl == "" || SelectedQuestion.PictureUrl ==
null)
```

```
                {
                    QuestionPictureBox.Hide();
                }
                else
                {
                    //Assigning the selected question to a variable
                    QuestionPictureBox.Show();
                    QuestionPictureBox.Image = Image.FromFile(SelectedQuestion.PictureUrl);
                }

                QuestionLabel.Text = SelectedQuestion.Question;

                AnswerLabel.Text = SelectedQuestion.CorrectAns;
            }
        }

        private void SelectDifficultyComboBox_SelectedIndexChanged(object sender, EventArgs
e)
        {

        }

        private void tableLayoutPanel9_Paint(object sender, PaintEventArgs e)
        {

        }

    }
}
```

View Stats

```
using PhysicsQuiz1._0.Classes;
using PhysicsQuiz1._0.LoginScreen;
using PhysicsQuiz1._0.QuizForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.GeneralForms
{
    public partial class ViewStats : Form
    {
        //Views all the stats from the quiz that the student has answered
        private StudentLogin Student = new StudentLogin();

        List<StoredQuestions> ListOfStoredQuestions = new List<StoredQuestions>();

        StoredQuizzes SelectedQuiz = new StoredQuizzes();

        List<StoredQuizQuestions> QuizQuestionsId = new List<StoredQuizQuestions>(); //For
each answered quiz question the quiz questionID must be saved to this list

        List<CompletedQuestion> completedQuestion = new List<CompletedQuestion>();

        CompletedQuiz CQuiz = new CompletedQuiz();
```

```
        public event EventHandler OpenStoredQuizzes;

        public ViewStats(StudentLogin student, StoredQuizzes SQuiz, List<StoredQuestions>
storedQuestions, List<StoredQuizQuestions> storedQuizQuestions)
        {
            InitializeComponent();
            setup(student, SQuiz, storedQuestions, storedQuizQuestions, null);
        }

        private void setup(StudentLogin student, StoredQuizzes SQuiz, List<StoredQuestions>
storedQuestions, List<StoredQuizQuestions> storedQuizQuestions, List<CompletedQuestion>
cquestion)
        {
            //In a seperate sub as the page may need to be refreshed when the student has
answered the quiz again

            if (cquestion == null)
            {
                StudentInputNameLabel.Text = ($"{student.FirstName} {student.SecondName}");

                InputClassIdLabel.Text = student.ClassId.ToString();

                QuestionClass qc = new QuestionClass();
                //If the CompletedQuiz has not be loaded from the data base it will load it
here
                if (CQuiz == null)
                {
                    CQuiz.Id = SQuiz.QuizId;
                    CQuiz.StudentId = student.StudentId;
                    CQuiz.Length = SQuiz.Length;
                    CQuiz = qc.CreateCompletedQuiz(CQuiz);
                }

                completedQuestion = qc.GetCompletedQuestion(CQuiz, storedQuizQuestions);
            }
            else
            {
                //If the questions have already been answered and the user wishes to return
to their stats then the page needs to be refreshed but we don`t need to
                //query the database again
                listView1.Items.Clear();
                listView1.Refresh();
                completedQuestion = cquestion;
            }

            foreach (StoredQuestions sq in storedQuestions)
            {
                //Assigns the relevant stats the the rows for each question on the table
                ListViewItem b = new ListViewItem(sq.Question);
                if (sq.Area == 1)
                {
                    b.SubItems.Add("Recall");
                }
                else
                {
                    b.SubItems.Add("Calculations");
                }

                if (sq.TopicId == 1)
                {
                    b.SubItems.Add("Particles");
```

```
                }
                else if (sq.TopicId == 2)
                {
                    b.SubItems.Add("Waves");
                }
                else if (sq.TopicId == 3)
                {
                    b.SubItems.Add("Mechanics");
                }
                else if (sq.TopicId == 4)
                {
                    b.SubItems.Add("Materials");
                }
                else if (sq.TopicId == 5)
                {
                    b.SubItems.Add("Electricity");
                }

                //For each question answered in completed questions, the loop checks to see
if it is equal to the current stored question ID.
                foreach (CompletedQuestion cq in completedQuestion)
                {
                    if (cq.QuestionId == sq.QuestionId)
                    {
                        //If the question is the same then the code adds the question`s
stats to the table containing their scores (times answered, times correct, difficulty score,
etc)
                        b.SubItems.Add(cq.XCompleted.ToString());
                        b.SubItems.Add(cq.XCorrect.ToString());
                        string score = DifficultyScore(cq.CalculatedDifficulty);
                        b.SubItems.Add(score);
                        b.SubItems.Add(cq.CalculatedDifficulty.ToString());
                        break; //braks the loop so that there is no more wasted loops
                    }
                }
                listView1.Items.Add(b);
            }


            Student = student;

            ListOfStoredQuestions = storedQuestions;

            SelectedQuiz = SQuiz;

            QuizQuestionsId = storedQuizQuestions;
            //saves varaibles to the program so that if the user starts a quiz then they
don`t need to be retrived and are ready to be called

        }


        private void listView1_SelectedIndexChanged(object sender, EventArgs e)
        {

        }

        private void ReturnButton_Click(object sender, EventArgs e)
        {
            this.Close();
        }
```

```
        private void ReturnedToStoredQuizzesButton_Click(object sender, EventArgs e)
        {
            this.Close();
            OpenStoredQuizzes?.Invoke(this, EventArgs.Empty);
            //Closes the form and opens the previous one
        }

        private void GenerateReportButton_Click(object sender, EventArgs e)
        {
            //When pressed this button the quiz generates a report which will be sent to the
teacher containing the student scores
            SendQuizInfo SQI = new SendQuizInfo(Student, ListOfStoredQuestions,
completedQuestion, SelectedQuiz); //This opens the form containing the contorols in which
the email is sent

            this.Hide();

            SQI.Show();

            SQI.FormClosed += (source, EventArgs) =>
            {
                this.Show();
            }; //The form closed event will be triggered when the user closes the send email
form
        }

        private void StudyButton_Click(object sender, EventArgs e)
        {
            StartQuizForm SQF = new StartQuizForm(Student, SelectedQuiz,
ListOfStoredQuestions, QuizQuestionsId, CQuiz, completedQuestion); //Opens the form
containing the starting quiz information

            this.Hide();

            SQF.Show();

            SQF.CompletedQuiz += NewViewStats; //Method retrives the stats from the start
quiz page based upon how well they did and then refreshes the page

            SQF.FormClosed += (source, EventArgs) =>
            {
                this.Show();
            }; //When the start quiz form is closed then this event is triggered

        }

        private void NewViewStats(StartQuizForm SQF, StudentLogin student, StoredQuizzes
SQuiz, List<StoredQuestions> storedQuestions, List<StoredQuizQuestions> storedQuizQuestions,
List<CompletedQuestion> cq)
        {
            setup(student, SQuiz, storedQuestions, storedQuizQuestions, cq); //Refresehs the
page to display any changes that may have been created when the student either deletes
question info or completes a question
        }

        private string DifficultyScore(int cq)
        {
            //This is where the numerical difficulty values are turned into worded
difficulty ratinging based upon their scores
            if(cq <= 20)
            {
                return "Poor";
```

```
            }
            else if (cq <= 40)
            {
                return "Worse";
            }
            else if (cq <= 60)
            {
                return "Good";
            }
            else
            {
                return "Great";
            }
        }

        private void ResetQuestionButton_Click(object sender, EventArgs e)
        {
            //This button will reset the question scores
            QuestionClass qc = new QuestionClass();

            //Resets the questions in the database
            qc.ResetScores(completedQuestion);

            //Creates new completed questions
            completedQuestion = new List<CompletedQuestion>();

            //Clears the table on the current page displaying the question stats
            listView1.Items.Clear();
            listView1.Refresh();

            //Runs the setup sub again
            setup(Student, SelectedQuiz, ListOfStoredQuestions, QuizQuestionsId, null);
        }

        private void tableLayoutPanel4_Paint(object sender, PaintEventArgs e)
        {

        }
    }
}
```

View Stored Quizzes

```
using PhysicsQuiz1._0.Classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PhysicsQuiz1._0.StudentForms
{
    public partial class ViewStoredQuizzes : Form
    {
        public List<StoredQuizzes> Quizzes = new List<StoredQuizzes>();
        public List<StoredQuestions> SelectedQuestions = new List<StoredQuestions>();
```

```
        public List<StoredQuizQuestions> SelectedQuestionsId = new
List<StoredQuizQuestions>();
        public StoredQuizzes ChosenQuiz = new StoredQuizzes();

        public event EventHandler ClosedPage;
        public event Action<ViewStoredQuizzes ,StoredQuizzes, List<StoredQuestions>,
List<StoredQuizQuestions>> SelectedQuiz;

        bool formclosing = false;

        public ViewStoredQuizzes()
        {
            InitializeComponent();
            QuestionClass qc = new QuestionClass();
            Quizzes = qc.LoadQuizzes("%"); //Loads the quizzes from the database that
contain any question text
            QuizListBox.DataSource = Quizzes; //Sets the source for the list box to be the
loaded quizzes
            QuizListBox.DisplayMember = "Name"; //The displayed item from the stored quizzes
to be the name of the quiz

            //Hides the quiz display info so that it only appears when the user clicks on an
item
            QuizNameLabel.Hide();
            InsertQuizNameLabel.Hide();
            QuestionsLabel.Hide();
            QuestionsListBox.Hide();
            ExpandButton.Hide();
        }

        private void ViewStoredQuizzes_Load(object sender, EventArgs e)
        {

        }

        private void QuizListBox_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            //Displays the quiz info
            QuizNameLabel.Show();
            InsertQuizNameLabel.Show();
            QuestionsLabel.Show();
            QuestionsListBox.Show();
            ExpandButton.Show();

            ChosenQuiz = (StoredQuizzes)QuizListBox.SelectedItem;  //The chosen quiz is
saved to be the selected item from the list view
            InsertQuizNameLabel.Text = ChosenQuiz.Name; //The quiz name is displayed
            QuestionClass qc = new QuestionClass();
            SelectedQuestionsId = qc.FindQuestionsId(ChosenQuiz); //Retrives the quiz`s
questions ID`s from the database
            SelectedQuestions = qc.GetStoredQuizQuestions(SelectedQuestionsId); //The ID`s
related questions are then returned from the database
            QuestionsListBox.DataSource = SelectedQuestions; //The data source for the list
box displaying the questions is set to the questions
            QuestionsListBox.DisplayMember = "Question"; //The displayed item from the
stored quizzes to be the name of the quiz
        }

        private void ReturnButton_Click(object sender, EventArgs e)
        {
            formclosing = true; //When the return button is pressed the form closing is set
to true and the code is triggered in order to close this form and open the previous form
```

```
            this.Close();
            ClosedPage?.Invoke(this, EventArgs.Empty); //Shows the previous form
        }

        protected override void OnFormClosed(FormClosedEventArgs e)
        {
            if (formclosing != true) //If the page hasn`t set the form to close already it
will summon the sub that manages it
            {
                ReturnButton_Click(this, EventArgs.Empty);
            }
            base.OnFormClosed(e); //It will then trigger the normal form closing event
        }

        private void SearchButton_Click(object sender, EventArgs e)
        {
            QuestionClass qc = new QuestionClass();
            List<StoredQuizzes> SearchedQuizzes = new List<StoredQuizzes>();
            SearchedQuizzes = qc.LoadQuizzes(SearchBarTextBox.Text + "%"); //Loads the
quizzes that start with the search criteria from the database
            QuizListBox.DataSource = SearchedQuizzes; //Sets the data source to be these
searched quizzes
            QuizListBox.DisplayMember = "Name"; //The quiz name is displayed
        }

        private void SearchBarTextBox_TextChanged(object sender, EventArgs e)
        {
            //If the search criteria is deleted then the quizzes displayed are returned to
default
            if (SearchBarTextBox.Text == "")
            {
                QuizListBox.DataSource = Quizzes;
                QuizListBox.DisplayMember = "Name";
            }
        }

        private void ExpandButton_Click(object sender, EventArgs e)
        {
            //Triggered when the user choses to expand a quizzes info
            //Form is closed
            formclosing = true;
            this.Close();
            SelectedQuiz?.Invoke(this, ChosenQuiz, SelectedQuestions, SelectedQuestionsId);
        }
    }
}
```

Start Quiz Form

```
using PhysicsQuiz1._0.Classes;
using PhysicsQuiz1._0.GeneralForms;
using PhysicsQuiz1._0.StudentForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace PhysicsQuiz1._0.QuizForms
{
    public partial class StartQuizForm : Form
    {
        public StudentLogin Student;
        public StoredQuizzes SQuiz;
        public List<StoredQuestions> storedQuestions;
        public List<StoredQuizQuestions> storedQuizQuestions;
        public CompletedQuiz completedQuiz;
        public List<CompletedQuestion> completedQuestion;

        public event Action<StartQuizForm ,StudentLogin, StoredQuizzes,
List<StoredQuestions>, List<StoredQuizQuestions>, List<CompletedQuestion>> CompletedQuiz;
//This event is used to return the values of the completed quiz to the stats form when this
form is closed

        public StartQuizForm(StudentLogin student, StoredQuizzes sQuiz,
List<StoredQuestions> sQuestions, List<StoredQuizQuestions> storedQQuestions, CompletedQuiz
cQuiz, List<CompletedQuestion> compQuestion)
        {
            InitializeComponent();
            //Parameters are assigned to the global variables
            Student = student;
            SQuiz = sQuiz;
            storedQuestions = sQuestions;
            storedQuizQuestions = storedQQuestions;
            completedQuiz = cQuiz;
            completedQuestion = compQuestion;
        }

        private void StartQuizButton_Click(object sender, EventArgs e)
        {
            //Hides the main form
            this.Hide();

            CalculateDifficulty cd = new CalculateDifficulty(); //Claculated difficulty is
initilised

            //Trys to decide which option the user wants.
            //Adaptive question order: Questions answered incorrectly are presented more
often than the correct ones
            //Standard: Questions are presented normally
            //If the user hasn`t selected an option a null exception is thrown
            try
            {
                if (SelectModeComboBox.SelectedItem.ToString() == "Adaptive Questions
Order")
                {
                    foreach (CompletedQuestion cq in completedQuestion)
                    {
                        if ((cq.CalculatedDifficulty > 80) && (cq.XCompleted > 5))
                        {
                            //This removes the question from the quiz so that the user
doesn`t answer this question that they have already answered correctly the majority of times
                            storedQuizQuestions.Remove(storedQuizQuestions.Find(x =>
x.QuestionId == cq.QuestionId));
                        }
                    }
                }
            }
            catch (System.NullReferenceException)
```

```
            {
                //The null excpetion is caught and this message is displayed
                MessageBox.Show("Please Select A Question Mode", "Error",
MessageBoxButtons.OK);
                return;
            }

            //The stored quiz question need to be shuffled so this line of code does that
            List<StoredQuizQuestions> ShuffledQuizQuestions = storedQuizQuestions.OrderBy(x
=> Guid.NewGuid()).ToList();

            foreach (StoredQuizQuestions QuizQuestion in ShuffledQuizQuestions)
            {
                StoredQuestions CurrentQuestion = (storedQuestions.Find(x => x.QuestionId ==
QuizQuestion.QuestionId)); //The current question is saved to the varaible called current
question

                CompletedQuestion CurrentCompletedQuestion = (completedQuestion.Find(x =>
x.QuestionId == QuizQuestion.QuestionId));

                if (CurrentQuestion.PictureUrl == "") //If there is no picture URL then it
doesn`t have a picture
                {
                    var page2 = new TextQuestionForm(CurrentQuestion); //The next form is
created

                    page2.Answered += (source, Correct) => //This event is used to return
the answer that the user selects and treats it accordingly based upon if the correct bool is
true or false
                    {
                        //Question is removed from both storedquestion and completedquestion
                        storedQuestions.Remove(CurrentQuestion);
                        completedQuestion.Remove(CurrentCompletedQuestion);

                        //If the user has ansewred correctly then the current quiz
question`s XAnsweredCorrect will increase by one
                        if (Correct == true)
                        {
                            CurrentQuestion.XAnsweredCorrectly++;
                            CurrentCompletedQuestion.XCorrect++;
                        }

                        //Regardless of if the answer was answered correctly the times
answered counter must also increment by one
                        CurrentQuestion.XAnswered++;
                        CurrentCompletedQuestion.XCompleted++;

                        //The difficulty rating must be recalculated
                        CurrentQuestion.CalculatedDifficulty =
cd.CalcDifficulty(CurrentQuestion.XAnswered, CurrentQuestion.XAnsweredCorrectly);
                        CurrentCompletedQuestion.CalculatedDifficulty =
cd.CalcDifficulty(CurrentCompletedQuestion.XCompleted, CurrentCompletedQuestion.XCorrect);

                        //The question is added back to the storedquestion and complted
question lists
                        storedQuestions.Add(CurrentQuestion);
                        completedQuestion.Add(CurrentCompletedQuestion);
                    };

                    //Displays the question answering form
                    page2.ShowDialog();
                }
```

```
                else
                {
                    var page2 = new PictureQuestionForm(CurrentQuestion);

                    page2.Answered += (source, Correct) =>
                    {
                        //Question is removed from both storedquestion and completedquestion
                        storedQuestions.Remove(CurrentQuestion);
                        completedQuestion.Remove(CurrentCompletedQuestion);

                        //If the user has ansewred correctly then the current quiz
question`s XAnsweredCorrect will increase by one
                        if (Correct == true)
                        {
                            CurrentQuestion.XAnsweredCorrectly++;
                            CurrentCompletedQuestion.XCorrect++;
                        }

                        //Regardless of if the answer was answered correctly the times
answered counter must also increment by one
                        CurrentQuestion.XAnswered++;
                        CurrentCompletedQuestion.XCompleted++;

                        //The difficulty rating must be recalculated
                        CurrentQuestion.CalculatedDifficulty =
cd.CalcDifficulty(CurrentQuestion.XAnswered, CurrentQuestion.XAnsweredCorrectly);
                        CurrentCompletedQuestion.CalculatedDifficulty =
cd.CalcDifficulty(CurrentCompletedQuestion.XCompleted, CurrentCompletedQuestion.XCorrect);

                        //The question is added back to the storedquestion and complted
question lists
                        storedQuestions.Add(CurrentQuestion);
                        completedQuestion.Add(CurrentCompletedQuestion);
                    };

                    //Displays the question answering form
                    page2.ShowDialog();
                }

            }
            //Question class is initilized
            QuestionClass qc = new QuestionClass();

            //The method update scored is called in order to refresh the scored that are in
the database
            qc.UpdateScores(storedQuestions, completedQuestion);

            this.Show();


        }

        private void ReturnButton_Click(object sender, EventArgs e)
        {
            //The scores are returned to the previous form in order to be refreshed in the
scores table
            CompletedQuiz?.Invoke(this, Student, SQuiz, storedQuestions,
storedQuizQuestions, completedQuestion);
            this.Close();
        }
    }
}
```

Code - Classes

Calculate Difficulty

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    public class CalculateDifficulty
    {
        public int CalcDifficulty(int XCompleted, int XCorrect)
        {
            double percent = ((double)XCorrect / XCompleted);
            int cqpercentage = (int)(Math.Round(percent, 2) * 100);
            return cqpercentage;
        }

        //Takes in two values of the number of times the question has been answered
        //and how many times it was answered correctly. It then works out the percentage of
the
        //time it was answered correctly based upon this information.
    }
}
```

Completed Question

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    public class CompletedQuestion
    {
        public int StudentId { get; set; }
        public int QuestionId { get; set; }
        public int XCompleted { get; set; }
        public int XCorrect { get; set; }
        public int CalculatedDifficulty { get; set; }
    }
    //Stores the scores of the student’s questions based on their student ID and QuestionID.
    //It also stores the difficulty that has been calculated. As it is not linked to a
specific quiz,
    //it means that progress from questions is carried across questions
}
```

Completed Quiz

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace PhysicsQuiz1._0.Classes
{
    public class CompletedQuiz
    {
        public int Id { get; set; }
        public int StudentId { get; set; }
        public int Length { get; set; }
        public int QuizId { get; set; }
    }


}
```

Create HTML Table

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    public class CreateHTMLTable
    {
        //Encapsulation
        //OOP
        //HTML Code
        //2D Array

        List<StoredQuestions> SQ = new List<StoredQuestions>();
        List<CompletedQuestion> CQ = new List<CompletedQuestion>();

        CalculateDifficulty CD = new CalculateDifficulty();

        //Declaration of variables where SQ will contain the stored questions that data will
be calculated about
        //CQ stores the questions that the students have answered
        //CD is the class of calculating difficulty based upon the data that has been input.


        int[,] Area = new int[2,2]; //Contains a running total containg how many times each
question topic has been answered and how many times it is correct.
        int[,] Topic = new int[5,2];

        public string createtable(List<StoredQuestions> sq, List<CompletedQuestion> CD)
        {
            SQ = sq;
            CQ = CD;

            string finalresult = "";
            finalresult = finalresult + createheader();
            finalresult = finalresult + CreateQuestionTable();
            finalresult = finalresult + "<br><br>";
            finalresult = finalresult + CreateAreaTable();
            finalresult = finalresult + "<br><br>";
            finalresult = finalresult + CreateTopicTable();
            finalresult = finalresult + "</body></ html >";
```

```
            return finalresult;

            //The main sub. It contains the while string finalresult which will return the
table that has been created in HTML
            //Each part of its deceleration is broken into subs and combined at the end.
        }

        private string createheader()
        {
            string header = "<html><head><style>table, th, td {  border: 1px solid
black;}</style></head><body><h2>Question Breakdown</h2>";
            return header;
            //Declares the table`s header
        }

        private string CreateQuestionTable()
        {
            string QuestionTable = "<table style=\"width: 100 % \"
><tr><th>Question</th><th>Area</th><th>Topic</th><th>Knowledge</th><th>Times
Answered</th><th>Times Correct</th></tr>";
            foreach (StoredQuestions sq in SQ)
            {
                string row = "";
                foreach (CompletedQuestion CD in CQ)
                {
                    if (CD.QuestionId == sq.QuestionId)
                    {
                        row = row + "<td>"+ sq.Question +"</td>";
                        if (sq.Area == 1)
                        {
                            row = row + "<td> Recall </td>";
                            Area[0, 0] = Area[0, 0] + CD.XCorrect;
                            Area[0, 1] = Area[0, 1] + CD.XCompleted;
                        }
                        else
                        {
                            row = row + "<td> Calculation </td>";
                            Area[1, 0] = Area[1, 0] + CD.XCorrect;
                            Area[1, 1] = Area[1, 1] + CD.XCompleted;
                        }
                        //Adds the current question type to the related array containg the
total times the category has been answered and how mamy times it has been enter correctly


                        if (sq.TopicId == 1)
                        {
                            row = row + "<td> Particles </td>";
                            Topic[0, 0] = Topic[0, 0] + CD.XCorrect;
                            Topic[0, 1] = Topic[0, 1] + CD.XCompleted;
                        }
                        else if (sq.TopicId == 2)
                        {
                            row = row + "<td> Waves </td>";
                            Topic[1, 0] = Topic[1, 0] + CD.XCorrect;
                            Topic[1, 1] = Topic[1, 1] + CD.XCompleted;
                        }
                        else if (sq.TopicId == 3)
                        {
                            row = row + "<td> Mechanics </td>";
                            Topic[2, 0] = Topic[2, 0] + CD.XCorrect;
                            Topic[2, 1] = Topic[2, 1] + CD.XCompleted;
                        }
```

```
                        else if (sq.TopicId == 4)
                        {
                            row = row + "<td> Materials </td>";
                            Topic[3, 0] = Topic[3, 0] + CD.XCorrect;
                            Topic[3, 1] = Topic[3, 1] + CD.XCompleted;
                        }
                        else if (sq.TopicId == 5)
                        {
                            row = row + "<td> Electricity </td>";
                            Topic[4, 0] = Topic[4, 0] + CD.XCorrect;
                            Topic[4, 1] = Topic[4, 1] + CD.XCompleted;
                        }

                        //Topic breakdown for the scores on question. It displays the
question topic and adds their score on the question to the topic array.

                        string score = DifficultyScore(CD.CalculatedDifficulty);
                        row = row + "<td>"+ score + "</td>";
                        row = row + "<td>" + CD.XCompleted.ToString() + "</td>";
                        row = row + "<td>" + CD.XCorrect.ToString() + "</td>";
                        row = row + "</tr>";
                        break;

                        //Displays the times the question is correct and how many times it
has been answered.
                    }
                }
                QuestionTable = QuestionTable + row;
            }
            QuestionTable = QuestionTable + "</table>";
            return QuestionTable;
            //Closes the table and returns it
        }

        private string DifficultyScore(int CD)
        {
            //Returns the worded difficulty rating for the percentage score that the user
has
            if (CD <= 20)
            {
                return "Poor";
            }
            else if (CD <= 40)
            {
                return "Worse";
            }
            else if (CD <= 60)
            {
                return "Good";
            }
            else
            {
                return "Great";
            }
        }

        private string CreateAreaTable()
        {
            //Creates the area table based upon the scores that have been gathered
previously when creating the question table in the Array Area[]
            string AreaTable;
```

```
            AreaTable = "<h2> Area Results </h2><table style=\"width:50
%\"><tr><th>Recall</th><th>Calculations</th></tr>";
            AreaTable = AreaTable + "<tr><td>"+ CD.CalcDifficulty(Area[0,1], Area[0,0])
+"%</td><td>" + CD.CalcDifficulty(Area[1, 1], Area[1, 0]) + "%</td></tr>";
            AreaTable = AreaTable + "</table>";
            return AreaTable;
        }

        private string CreateTopicTable()
        {
            //Creates the Topic table based upon the scores that have been gathered
previously when creating the question table in the Array Topic[]
            string TopicTable;
            TopicTable = "<h2>Topic Results</h2><table style=\"width: 100 %
\"><tr><th>Particles</th><th>Waves</th><th>Mechanics</th><th>Materials</th><th>Electricty</t
h></tr><tr>";

            if(Topic[0,0] != 0)
            {
                TopicTable = TopicTable + "<td>" + CD.CalcDifficulty(Topic[0, 1], Topic[0,
0]) + "%</td>";
            }
            else
            {
                TopicTable = TopicTable + "<td> </td>";
            }

            if (Topic[1, 0] != 0)
            {
                TopicTable = TopicTable + "<td>" + CD.CalcDifficulty(Topic[1, 1], Topic[1,
0]) + "%</td>";
            }
            else
            {
                TopicTable = TopicTable + "<td> </td>";
            }

            if (Topic[2, 0] != 0)
            {
                TopicTable = TopicTable + "<td>" + CD.CalcDifficulty(Topic[2, 1], Topic[2,
0]) + "%</td>";
            }
            else
            {
                TopicTable = TopicTable + "<td> </td>";
            }

            if (Topic[3, 0] != 0)
            {
                TopicTable = TopicTable + "<td>" + CD.CalcDifficulty(Topic[3, 1], Topic[3,
0]) + "%</td>";
            }
            else
            {
                TopicTable = TopicTable + "<td> </td>";
            }

            if (Topic[4, 0] != 0)
            {
                TopicTable = TopicTable + "<td>" + CD.CalcDifficulty(Topic[4, 1], Topic[4,
0]) + "%</td>";
            }
```

```
            else
            {
                TopicTable = TopicTable + "<td> </td>";
            }


            TopicTable = TopicTable + "</tr></table>";

            return TopicTable;

        }
    }
}
```

Data Access

```
using Dapper;
using System;
using System.Data;
using System.Text;

namespace PhysicsQuiz1._0.Classes
{
    public class DataAccess
    {
        public void CreateStudent(string firstname, string surname, string username, string
password, int classcode)
        {
            //Creates a student account on the database based on the input data
            //The salt is added to the password to make it more secure when encrypting and
is also stored in the database.
            string salt = GenerateSalt();

            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                var parameters = new { Firstname = firstname, SecondName = surname, Username
= username, Password = EncryptPassword(Salted(password, salt)), ClassId = classcode, Salt =
salt };

                //Calls the stored procedure create new student and inputs the values
created in parameters above
                connection.Execute("dbo.spStudentLogin_CreateNewStudent @Firstname,
@SecondName, @Username, @Password, @ClassId, @Salt", parameters);
            }
        }


        public StudentLogin AttemptStudentLogin(string Username, string password)
        {
            //Attempts to login a student based upon the username and password that they
have given.
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                try
                {
```

```
                    //Attempts to retrive the salt from the database based upon their
username. If it cannot find it, it will retun an ystem.Data.SqlClient.SqlException which
will be caught
                    //and then a null value will be returned instead. Showing that it
couldn`t be found.
                    string salt = GetSalt(Username, "dbo.spStudentLogin_GetSalt");

                    //connection.QuerySingle<string>("dbo.spStudentLogin_GetSalt @username",
new { username = Username });

                    if (salt == null)
                    {
                        return null;
                    }
                    else
                    {
                        //Encrypt password sub encrypts the password and salt input and
returns the encryped value
                        var parameters = new { username = Username, password =
EncryptPassword(Salted(password, salt)) };


                        //Calls the stored procedure AttemptLogin and inputs the values
created in parameters above
                        var user = connection.QuerySingle<StudentLogin>("exec
dbo.spStudentLogin_AttemptLogin @username, @password", parameters);
                        if (user == null)
                        {
                            return null;
                        }
                        else
                        {
                            return user;
                        }
                    }
                }
                catch (System.Data.SqlClient.SqlException)
                {
                    return null;
                }
                catch (System.InvalidOperationException)
                {
                    return null;
                }
        }
        }

        public bool CheckStudentUsername(string username)
        {
            //Used when creating a new student, it checks to see if the username is already
taken.
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                try
                {
                    string u = connection.QuerySingle<string>($"exec
dbo.spStudentLogin_IsUsernameTaken '{username}'");
                    return true;
                }
                catch
                {
```

```
                    return false;
                }
            }
        }

        public bool CheckTeacherUsername(string username)
        {
            //Used when creating a new teacher, it checks to see if the username is already
taken.
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                try
                {
                    int u = connection.QuerySingle<int>($"exec
dbo.spTeacherLogin_CheckUsername '{username}'");
                    return true;
                }
                catch
                {
                    return false;
                }
            }
        }

        public bool CheckClassCode(int classcode)
        {
            //Used when creating a new student, it checks to see if the class code input
exists.
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                try
                {
                    int u = connection.QuerySingle<int>($"exec
dbo.spTClass_CheckValidClassCode '{classcode}'");
                    return true;
                }
                catch (System.Data.SqlClient.SqlException)
                {
                    return false;
                }
                catch (System.InvalidOperationException)
                {
                    return false;
                }
            }
        }

        public string Salted(string password, string salt)
        {
            //Combines the passord and salt of the input parameters
            return (password + salt);
        }

        public string GetSalt(string Username, string SaltType)
        {
            //Retrives the salt
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                try
```

```
                {
                    var output = connection.QuerySingle<string>($"{SaltType} @username", new
{ username = Username });
                    return output;
                }
                catch (System.Data.SqlClient.SqlException)
                {
                    return null;
                }
                catch (System.InvalidOperationException)
                {
                    return null;
                }
            }
        }

        public string GenerateSalt()
        {
            //Generated a new salt to be added to the end of the password
            var rng = new System.Security.Cryptography.RNGCryptoServiceProvider();
            var buff = new byte[15];
            rng.GetBytes(buff);
            return Convert.ToBase64String(buff);
        }

        public string EncryptPassword(string saltedpassword)
        {
            //Encrypts the password based upon hashing encryption
            byte[] bytes = System.Text.Encoding.UTF8.GetBytes(saltedpassword);
            System.Security.Cryptography.SHA256Managed sha256hashstring = new
System.Security.Cryptography.SHA256Managed();
            byte[] hash = sha256hashstring.ComputeHash(bytes);

            StringBuilder hex = new StringBuilder(hash.Length * 2);

            foreach (byte b in hash)
            {
                hex.AppendFormat("{0:x2}", b);
            }
            return hex.ToString();
        }

        public void CreateNewTeacher(string title, string surname, string username, string
password, string email)
        {
            //Creates a new teacher based upon the input parameters
            string salt = GenerateSalt();
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                var parameters = new { Title = title, SecondName = surname, Username =
username, Password = EncryptPassword(password + salt), Email = email, Salt = salt };

                connection.Execute("dbo.TeacherLogin_CreateNewTeacher @Title, @SecondName,
@Username, @Password, @Email, @Salt", parameters);
            }
        }

        public TeacherLogin AttemptTeacherLogin(string Username, string pword)
        {
            //Attempts to login a teacher based upon the username and password that they
have given.
```

```
            //Attempts to retrive the salt from the database based upon their username. If
it cannot find it, it will retun an system.Data.SqlClient.SqlException which will be caught
            //and then a null value will be returned instead. Showing that it couldn`t be
found.
            string salt = GetSalt(Username, "dbo.spTeacherLogin_GetSalt");
            if (salt == null)
            {
                return null;
            }
            else
            {
                using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
                {
                    string encryptedPassword = EncryptPassword(pword + salt);
                    var parameters = new { Username, Password = encryptedPassword };
                    try
                    {
                        //Attempts to retrive the teachers information from the database
based upon their username and encrypted password. If it cannot find it, it will retun an
system.Data.SqlClient.SqlException which will be caught
                        //and then a null value will be returned instead. Showing that it
couldn`t be found.
                        var user = connection.QuerySingle<TeacherLogin>("exec
dbo.slTeacherLogin_AttemptLogin @Username, @Password", parameters);
                        return user;
                    }
                    catch (System.Data.SqlClient.SqlException)
                    {
                        return null;
                    }
                    catch (System.InvalidOperationException)
                    {
                        return null;
                    }
                }
            }
        }

        public string GetTeacherEmail(int ClassID)
        {
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                string email = connection.QuerySingle<string>("USE Physicsdb; execute
dbo.spTeacherLogin_GetTeacherEmail @classid;", new { classid = ClassID});
                return email;
            }
        }
    }
}
```

Login

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace PhysicsQuiz1._0.Classes
{
    public abstract class Login
    {
        public string SecondName { get; set; }
        public int ClassId { get; set; } //ClassID is an int containing the ID of their
class. This makes it so that they are easily identifiable by their teacher and therefore can
send emails to their teacher`s email with their progress.
        public string Salt { get; set; } //. The salt string contains a randomly generated
string which is added on the end of the password before it is encrypted using hash set
encryption (Objective 13). This makes the password even more secure.
                                            // As the password is also encrypted, it means that
even if an unauthorised user gains access to the database they won`t be able to decipher
what the password is.
        public string Username { get; set; } //Username is this table`s primary key and is
mainly used when the user logins in as that way they don`t have to remember a login number,
but instead a personalised string.
        public string Password { get; set; }
        public string Email { get; set; } //Their  email is taken so that the teacher can be
emailed the student`s progress.
    }


}
```

Question Class

```
using Dapper;
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;

namespace PhysicsQuiz1._0.Classes
{
    public class QuestionClass
    {
        //Contains all SQL for all question related forms
        public List<StoredQuestions> GetQuestionsForSearch(SearchCriteria sc, bool Type)
        {
            //Retreives the questions from the database that fit the search criteria which
is sent to the sub by the search critrtia parameter.
            List<StoredQuestions> questions = new List<StoredQuestions>();
            if (sc.Search == "")
            {
                sc.Search = "%";
            }
            else
            {
                sc.Search = "%" + sc.Search + "%";
            }
            //If the search criteria is empty or incomplete we must add the % sign to either
end. This way it searches for a string LIKE what has been specified instead of
            //exactly like it.
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                if (Type == false)
                {
```

```
                    //The type variable holds wether or not the user has searched for
generated difficulty (True) or predefined difficulty (False).
                    var parameters = new { Search = sc.Search, Topic1 = sc.Topic1, Topic2 =
sc.Topic2, Topic3 = sc.Topic3, Topic4 = sc.Topic4, Topic5 = sc.Topic5, Difficulty =
sc.Difficulty, Difficulty2 = sc.Difficulty1, Difficulty3 = sc.Difficulty2, Area = sc.Area,
Area2 = sc.Area1 };
                    questions = connection.Query<StoredQuestions>("exec
dbo.spStoredQuestions_SearchQuestionsStoredDiff @Search, @Topic1, @Topic2, @Topic3, @Topic4,
@Topic5, @Difficulty, @Difficulty2, @Difficulty3, @Area, @Area2", parameters).ToList();
                    return questions;
                }
                else
                {
                    //If the difficulty is generated then there are different stored
procedures for different rankings of difficulty this if statemement picks them
                    var parameters = new { Search = sc.Search, Topic1 = sc.Topic1, Topic2 =
sc.Topic2, Topic3 = sc.Topic3, Topic4 = sc.Topic4, Topic5 = sc.Topic5, Area = sc.Area, Area2
= sc.Area1 };
                    if(sc.Difficulty == 1)
                    {
                        var q = connection.Query<StoredQuestions>("exec
dbo.spStoredQuestions_SearchQuestionsGenDiff1 @Search, @Topic1, @Topic2, @Topic3, @Topic4,
@Topic5, @Area, @Area2", parameters).ToList();
                        questions.AddRange(q);
                    }
                    if (sc.Difficulty1 == 2)
                    {
                        var q = connection.Query<StoredQuestions>("exec
dbo.spStoredQuestions_SearchQuestionsGenDiff2 @Search, @Topic1, @Topic2, @Topic3, @Topic4,
@Topic5, @Area, @Area2", parameters).ToList();
                        questions.AddRange(q);
                    }
                    if (sc.Difficulty2 == 3)
                    {
                        var q = connection.Query<StoredQuestions>("exec
dbo.spStoredQuestions_SearchQuestionsGenDiff3 @Search, @Topic1, @Topic2, @Topic3, @Topic4,
@Topic5, @Area, @Area2", parameters).ToList();
                        questions.AddRange(q);
                    }
                    if (sc.Difficulty3 == 4)
                    {
                        var q = connection.Query<StoredQuestions>("exec
dbo.spStoredQuestions_SearchQuestionsGenDiff4 @Search, @Topic1, @Topic2, @Topic3, @Topic4,
@Topic5, @Area, @Area2", parameters).ToList();
                        questions.AddRange(q);
                    }

                    return questions;
                    //The questions are returned to the user
                }
            }
        }

        public List<StoredQuizQuestions> FindQuestionsId(StoredQuizzes SelectedQuiz)
        {
            //Retrevies the questions for the quiz beased upon the quiz that is sent.
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                var questions = connection.Query<StoredQuizQuestions>("exec
dbo.StoredQuizQuestions_GetQuestons @QuizId", new { QuizId = SelectedQuiz.QuizId
}).ToList();
```

```
                return questions;
            }
        }

        public List<StoredQuestions> LoadAllQuestions()
        {
            //loads all the questions in the stored question table
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                var questions = connection.Query<StoredQuestions>("exec
dbo.spStoredQuestions_LoadAllQuestions").ToList();
                return questions;
            }
        }

        public void CreateQuiz(int[] pastid, string Name)
        {
            //Transfers the question ID`s from pastid to the array id. If there are no more
to transfer, then the remaining spaces are saved to 0`s
            int[] Id = new int[15];

            for (int i = 0; i <= 14; i++)
            {
                try
                {
                    Id[i] = pastid[i];
                }
                catch (Exception)
                {
                    Id[i] = 0;
                };
            }


            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                //Parameters are passed to the stored procedure and from there are split
into normalized form as they are saved
                var parameters = new { question1 = Id[0], question2 = Id[1], question3 =
Id[2], question4 = Id[3], question5 = Id[4],
                    question6 = Id[5], question7 = Id[6], question8 = Id[7], question9 =
Id[8], question10 = Id[9], question11 = Id[10],
                    question12 = Id[11], question13 = Id[12], question14 = Id[13],
question15 = Id[14], name = Name, length = pastid.Length };

                connection.Execute("dbo.spStoredQuizzes_CreateQuiz @question1, @question2,
@question3, @question4, @question5, @question6, " +
                    "@question7, @question8, @question9, @question10, @question11,
@question12, @question13, @question14, @question15, @name, @length", parameters);
            }
        }

        public List<StoredQuizzes> LoadQuizzes(string Search)
        {
            //Loads the stored quizzes based upon their name from the search criteria
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                var questions = connection.Query<StoredQuizzes>("exec
dbo.StoredQuizzes_FindQuiz @search", new { search = Search}).ToList();
```

```
                return questions;
            }
        }

        public List<StoredQuestions> GetStoredQuizQuestions(List<StoredQuizQuestions>
SelectedQuiz)
        {
            //When viewing a stored quiz the QuizID is grabbed from get stored quiz
questions and each individual question ID is also grabbed. This sub then selects the stored
questions
            //From stored questions and then returns them for display
            List<StoredQuestions> sq = new List<StoredQuestions>();
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                foreach (StoredQuizQuestions a in SelectedQuiz)
                {
                    var questions = connection.QuerySingle<StoredQuestions>("exec
dbo.StoredQuestions_FindQuestions @question", new { question = a.QuestionId });
                    sq.Add(questions);
                }
                return sq;
            }
        }

        public CompletedQuiz CreateCompletedQuiz(CompletedQuiz quiz)
        {
            //If a student has never ran a quiz before the program must create the records
in the table of CompletedQuiz in order to store their progress.
            //It does this by creating a composite key consisting of StudentID and the quiz
ID
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                var Quiz = connection.QuerySingle<CompletedQuiz>("exec
dbo.CompletedQuiz_CreateQuiz @quizId, @studentId, @length", new { quizId = quiz.Id,
studentId = quiz.StudentId, length = quiz.Length });
                return Quiz;
            }
        }


        public List<CompletedQuestion> GetCompletedQuestion(CompletedQuiz cq,
List<StoredQuizQuestions> SQS)
        {
            //Gets the individual scores for each person and their questions answered
            List<CompletedQuestion> CQ = new List<CompletedQuestion>();
            CompletedQuestion question = new CompletedQuestion();
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                foreach(StoredQuizQuestions QuizQuestion in SQS)
                {
                    try
                    {
                        //The program will try to retervie the completed question that
contains their studentID and QuestionsID
                        question = connection.QuerySingle<CompletedQuestion>("exec
dbo.CompletedQuestion_GetQuestion @questionId, @studentId", new { questionId =
QuizQuestion.QuestionId, studentId = cq.StudentId });
                    }
                    catch(Exception)
```

```
                    {
                        //If the program cannot reterive the completed question it will
create a new one
                        question = connection.QuerySingle<CompletedQuestion>("
dbo.CompletedQuestion_Create @questionId, @studentId", new { questionId =
QuizQuestion.QuestionId, studentId = cq.StudentId });
                    }
                    CQ.Add(question);
                }
            }
            return CQ;
        }


        public void UpdateScores(List<StoredQuestions> sq, List<CompletedQuestion> cq)
        {
            //After a question has been answered the student`s completed question needs to
be updated so that the scores answered refelct the score that has been given.
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                foreach (StoredQuestions storedQuestions in sq)
                {
                    //Storedquestions are also updated for the global difficulty rating that
has been awarded after a question has been answered
                    connection.Execute("dbo.StoredQuestions_UpdateQuestion @questionid,
@xanswered, @xcorrect, @difficulty ", new { questionid = storedQuestions.QuestionId,
xanswered = storedQuestions.XAnswered, xcorrect = storedQuestions.XAnsweredCorrectly,
difficulty = storedQuestions.CalculatedDifficulty});
                }
                foreach (CompletedQuestion compquestion in cq)
                {
                    connection.Execute("dbo.CompletedQuestions_UpdateQuestion @questionid,
@xanswered, @xcorrect, @studentid, @difficulty ", new { questionid =
compquestion.QuestionId, xanswered = compquestion.XCompleted, xcorrect =
compquestion.XCorrect, studentid = compquestion.StudentId, difficulty =
compquestion.CalculatedDifficulty });
                }

            }
        }

        public void ResetScores(List<CompletedQuestion> cq)
        {
            //A user may wish to reset their progress for a question. If they do this, the
database is queried and every CompletedQuestion that contains their
            //StudentID and the StoredQuestionID is reset back to 0
            using (IDbConnection connection = new
System.Data.SqlClient.SqlConnection(Helper.CnnVal("Physicsdb")))
            {
                foreach(CompletedQuestion compquestion in cq)
                {
                    connection.Execute("dbo.CompltedQuestion_ResetQuestion @studentid,
@questionid", new { studentid = compquestion.StudentId, questionid = compquestion.QuestionId
});
                }
            }
        }
    }
}
```

Search Criteria

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    public class SearchCriteria
    {
        //When the user is searching for a question, the critiera that they use is stored
into this class
        public string Search { get; set; }
        public int Difficulty { get; set; }
        public int Difficulty1 { get; set; }
        public int Difficulty2 { get; set; }
        public int Difficulty3 { get; set; }
        public int Area { get; set; }
        public int Area1 { get; set; }
        public int Topic1 { get; set; }
        public int Topic2 { get; set; }
        public int Topic3 { get; set; }
        public int Topic4 { get; set; }
        public int Topic5 { get; set; }
    }
}
```

Stored Questions

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    public class StoredQuestions
    {
        //Holds the stored questions for the program.
        public int QuestionId { get; set; } // The question ID is used as the primary key in
the table
                                            // It uniquely identifies each question. It is
also used in other tables such as completed question
                                            // to relate their scores to the individual
questions.

        public string CorrectAns { get; set; } //The CorrectAns stores the correct answer
(Objective 1.1), it can be anything such as a letter or a sentence, therefore it is a
string.

        public string IncorrectAns1 { get; set; } //The incorrect answers must also be input
so that the correct answer isn`t obvious as it is the answer relating to the question.
        public string IncorrectAns2 { get; set; }
        public string IncorrectAns3 { get; set; }

        public string PictureUrl { get; set; } //The PictureURL stores the path to the file
in the program (Objective 2). It is allowed to be null as some questions don`t need a
picture.
```

```
        public int TopicId { get; set; } //TopicID holds the number of the topic that the
question relates to in a similar way to area which holds the area (Objective 7).
        public int Area { get; set; }

        public int DifficultyRating { get; set; }      //Difficulty rating is a predefined
difficulty that can be used to filter questions.
        public string Question { get; set; }           //Question holds the text from the
main question body (Objective 9).
        public int XAnswered { get; set; }             // XAnswered holds the number of
times the question has been answered and XAnsweredCorrectly. Both of these values are then
used to calculate the calculated difficulty.
        public int XAnsweredCorrectly { get; set; }    //This allows the calculated
difficulty to scale based on how often it is answered correctly, making
        public int CalculatedDifficulty { get; set; }  //it the most accurate difficulty
rating when it has been answered a large number of times(Objective 9).
                                                       //However, if it has only been
answered a limited number of times it may incorrectly represent the question`s difficulty.

        public string DisplayItem //Embedded function combines the question ID and the
question to form a display item that is used when displaying the question information.
        {
            get
            {
                return $"{QuestionId}. {Question}";
            }

        }
    }
}
```

Stored Quiz Questions

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    public class StoredQuizQuestions
    {
        //Stored Questions holds the questions that each quiz contains. When a quiz is
created each question that it is related to is saved in this table.
        //The QuizID holds the ID of the quiz that the entry is refering to and then the
QuestionID relates to the StoredQuestion which is saved to the Quiz.
        //That way when Quizzes of different lengths are created, there will be no wasted
        public int QuizId { get; set; }
        public int QuestionId { get; set; }
    }
}
```

Stored Quizzes

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    public class StoredQuizzes
    {
        public int QuizId { get; set; } //The QuizID identifies the individual quiz records
as the primary key.
        public string Name { get; set; } //Name is the name that has been assigned to it by
the user
        public int Length { get; set; } //Length contains how many questions are included in
the quiz
    }
}
```

Student Login

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    public class StudentLogin : Login
    {
        //Holds the login information for the student
        public int StudentId { get; set; } //foreign key for multiple tables such as
completed question. It allows us to identify each student easily and efficiently
        public string FirstName { get; set; }
        public string FullName //Encapsulation
        { //The Firstname and Surname variables stores the student`s names and the FullName
string combines the two for ease of use.
            get
            {
                return FirstName + " " + SecondName;
            }
        }
    }
}
```

TClass

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    class TClass
    {
        public int Id { get; set; } //A composite primary key containing the unique class ID
created by an incrementing counter and the Teacher`s ID.
        public int TeacherId { get; set; }
    }
}
```

Teacher Login

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysicsQuiz1._0.Classes
{
    public class TeacherLogin : Login
    {
        public int TeacherId { get; set; } //The TeacherID is a foreign key for multiple
tables such as completed question. It allows us to identify each teacher easily and
efficiently
        public string Title { get; set; }
        public string Email { get; set; } //Their  email is taken so that the teacher can be
emailed the student`s progress.

    }
}
```

SQL Queries

CompletedQuestion_Create

```
PROCEDURE [dbo].[CompletedQuestion_Create]
       @questionId int,
       @studentId int
as
begin
       INSERT INTO CompletedQuestion(StudentId, QuestionId) VALUES(@studentId, @questionId)

       SELECT *
       FROM CompletedQuestion
       WHERE StudentId = @studentId AND QuestionId = @questionId
end
```

CompletedQuestion_GetQuestion

```
PROCEDURE [dbo].[CompletedQuestion_GetQuestion]
       @questionId int,
       @studentId int
as
begin
       SELECT *
       FROM CompletedQuestion
       WHERE QuestionId = @questionId AND StudentId = @studentId
end
```

CompletedQuestions_UpdateQuestion

```
PROCEDURE [dbo].[CompletedQuestions_UpdateQuestion]
       @questionid int,
       @xanswered int,
       @xcorrect int,
```

```
        @studentid int,
        @difficulty int
as
begin
        UPDATE CompletedQuestion
        SET XCompleted = @xanswered, XCorrect=@xcorrect, CalculatedDifficulty = @difficulty
        WHERE QuestionId=@questionid AND StudentId = @studentid;
end
```

CompletedQuestion_ResetQuestion

```
PROCEDURE [dbo].[CompltedQuestion_ResetQuestion]
        @studentid int,
        @questionid int
as
begin
        UPDATE CompletedQuestion
        SET XCompleted = 0, XCorrect=0, CalculatedDifficulty = 0
        WHERE QuestionId=@questionid AND StudentId = @studentid;
end
```

TeacherLogin_AttemptLogin

```
PROCEDURE [dbo].[slTeacherLogin_AttemptLogin]
        @Username varchar(15),
        @Password varchar(256)
as
begin
        select *
        from dbo.TeacherLogin
        where Username = @Username and Password = @Password
end
```

StoredQuestions_LoadAllQuestions

```
procedure [dbo].[spStoredQuestions_LoadAllQuestions]
as
begin
        select *
        from dbo.StoredQuestions
end
```

StoredQuestions_SearchQuestionsGenDiff1

```
PROCEDURE [dbo].[spStoredQuestions_SearchQuestionsGenDiff1]
        @Search nvarchar(max),
        @Topic1 int,
        @Topic2 int,
        @Topic3 int,
        @Topic4 int,
        @Topic5 int,
        @Area int,
        @Area2 int
as
begin



        select *
        from dbo.StoredQuestions
```

```
        where (TopicId = @Topic1 OR TopicId = @Topic2 OR TopicId = @Topic3  OR TopicId =
@Topic4 OR TopicId = @Topic5)
        AND (Question LIKE @Search) AND ((Area = @Area) OR (Area = @Area2)) AND
CalculatedDifficulty <= 25
end
```

```
StoredQuestions_SearchQuestionsGenDiff2
PROCEDURE [dbo].[spStoredQuestions_SearchQuestionsGenDiff2]
        @Search nvarchar(max),
        @Topic1 int,
        @Topic2 int,
        @Topic3 int,
        @Topic4 int,
        @Topic5 int,
        @Area int,
        @Area2 int
as
begin



        select *
        from dbo.StoredQuestions
        where (TopicId = @Topic1 OR TopicId = @Topic2 OR TopicId = @Topic3  OR TopicId =
@Topic4 OR TopicId = @Topic5)
        AND (Question LIKE @Search) AND ((Area = @Area) OR (Area = @Area2)) AND
(CalculatedDifficulty <= 50 AND CalculatedDifficulty > 25)
end
```

```
StoredQuestions_SearchQuestionsGenDiff3
PROCEDURE [dbo].[spStoredQuestions_SearchQuestionsGenDiff3]
        @Search nvarchar(max),
        @Topic1 int,
        @Topic2 int,
        @Topic3 int,
        @Topic4 int,
        @Topic5 int,
        @Area int,
        @Area2 int
as
begin



        select *
        from dbo.StoredQuestions
        where (TopicId = @Topic1 OR TopicId = @Topic2 OR TopicId = @Topic3  OR TopicId =
@Topic4 OR TopicId = @Topic5)
        AND (Question LIKE @Search) AND ((Area = @Area) OR (Area = @Area2)) AND
(CalculatedDifficulty <= 75 AND CalculatedDifficulty > 50)
end
```

```
StoredQuestions_SearchQuestionsGenDiff4
PROCEDURE [dbo].[spStoredQuestions_SearchQuestionsGenDiff4]
        @Search nvarchar(max),
        @Topic1 int,
        @Topic2 int,
        @Topic3 int,
        @Topic4 int,
```

```
        @Topic5 int,
        @Area int,
        @Area2 int
as
begin



        select *
        from dbo.StoredQuestions
        where (TopicId = @Topic1 OR TopicId = @Topic2 OR TopicId = @Topic3  OR TopicId =
@Topic4 OR TopicId = @Topic5)
        AND (Question LIKE @Search) AND ((Area = @Area) OR (Area = @Area2)) AND
(CalculatedDifficulty <= 100 AND CalculatedDifficulty > 75)
end
```

StoredQuestions_SearchQuestionsStoredDiff

```
procedure [dbo].[spStoredQuestions_SearchQuestionsStoredDiff]
        @Search nvarchar(max),
        @Topic1 int,
        @Topic2 int,
        @Topic3 int,
        @Topic4 int,
        @Topic5 int,
        @Difficulty int,
        @Difficulty2 int,
        @Difficulty3 int,
        @Area int,
        @Area2 int
as
begin
        select *
        from dbo.StoredQuestions
        where (TopicId = @Topic1 OR TopicId = @Topic2 OR TopicId = @Topic3  OR TopicId =
@Topic4 OR TopicId = @Topic5)
                AND (Question LIKE @Search) AND ((Area = @Area) OR (Area = @Area2)) AND
((DifficultyRating = @Difficulty) OR (DifficultyRating = @Difficulty2) OR (DifficultyRating
= @Difficulty3))
end
```

StoredQuizzes_CreateQuiz

```
procedure [dbo].[spStoredQuizzes_CreateQuiz]
        @question1 int,
        @question2 int,
        @question3 int,
        @question4 int,
        @question5 int,
        @question6 int,
        @question7 int,
        @question8 int,
        @question9 int,
        @question10 int,
        @question11 int,
        @question12 int,
        @question13 int,
        @question14 int,
        @question15 int,
        @name VARCHAR(50),
        @length int
```

```
as
begin
		INSERT INTO StoredQuizzes(Name, Length)
				VALUES(@name, @length);

		IF @question1 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question1);
		IF @question2 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question2);
		IF @question3 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question3);
		IF @question4 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question4);
		IF @question5 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question5);
		IF @question6 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question6);
		IF @question7 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question7);
		IF @question8 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question8);
		IF @question9 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question9);
		IF @question10 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question10);
		IF @question11 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question11);
		IF @question12 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question12);
		IF @question13 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question13);
		IF @question14 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question14);
		IF @question15 != 0
			INSERT INTO StoredQuizQuestions(QuizId, QuestionId)
				VALUES (IDENT_CURRENT('StoredQuizzes'), @question15);
end
```

StudentLogin_AttemptLogin

```
procedure [dbo].[spStudentLogin_AttemptLogin]
	@username nvarchar(15),
	@password nvarchar(256)
as
begin
	SELECT *
	FROM StudentLogin
	WHERE Username = @username AND Password = @password
```

```
end
```

StudentLogin_CreateNewStudent

```
procedure [dbo].[spStudentLogin_CreateNewStudent]
	@Firstname varchar(15),
	@SecondName varchar(15),
	@Username varchar(15),
	@Password varchar(256),
	@ClassId int,
	@Salt varchar(25)
as
begin
	INSERT INTO StudentLogin(FirstName, SecondName, Username, Password, ClassId, Salt)
VALUES(@Firstname, @SecondName, @Username, @Password, @ClassId, @Salt);
end
```

StudentLogin_GetSalt

```
procedure [dbo].[spStudentLogin_GetSalt]
	@username nvarchar(15)
as
begin
	SELECT Salt
	FROM StudentLogin
	WHERE Username = @username
end
```

StudentLogin_IsUsernameTaken

```
procedure [dbo].[spStudentLogin_IsUsernameTaken]
	@username varchar(15)
as
begin
	select Username
	from dbo.StudentLogin
	where Username = @username
end
```

TClass_CheckValidClassCode

```
procedure [dbo].[spTClass_CheckValidClassCode]
	@id int
as
begin
	select 1
	from dbo.TClass
	where Id = @id
end
```

TeacherLogin_CheckUsername

```
procedure [dbo].[spTeacherLogin_CheckUsername]
	@username varchar(15)
as
begin
	select 1
	from dbo.TeacherLogin
	where Username = @username
end
```

TeacherLogin_GetSalt

```
procedure [dbo].[spTeacherLogin_GetSalt]
	@Username VARCHAR(15)
as
begin
	select Salt
	from dbo.TeacherLogin
	where Username = @Username
end
```

TeacherLogin_GetTeacherEmail

```
PROCEDURE [dbo].[spTeacherLogin_GetTeacherEmail]
	@classid int
as
begin
	SELECT Email
	FROM dbo.TeacherLogin
	WHERE ClassId = @classid
end
```

StoredQuestions_FindQuestions

```
PROCEDURE [dbo].[StoredQuestions_FindQuestions]
	@question int
as
begin
	SELECT *
	FROM StoredQuestions
	WHERE (QuestionId = @question)
end
```

StoredQuestions_UpdateQuestion

```
PROCEDURE [dbo].[StoredQuestions_UpdateQuestion]
	@questionid int,
	@xanswered int,
	@xcorrect int,
	@difficulty int
as
begin
	UPDATE StoredQuestions
	SET xAnswered = @xanswered, xAnsweredCorrect = @xcorrect, CalculatedDifficulty =
@difficulty
	WHERE QuestionId=@questionid;
end
```

StoredQuestions_GetQuestions

```
PROCEDURE [dbo].[StoredQuizQuestions_GetQuestons]
	@QuizId int
as
begin
	SELECT *
	FROM StoredQuizQuestions
	WHERE QuizId = @QuizId
end
```

StoredQuizzes_FindQuiz

```
PROCEDURE [dbo].[StoredQuizzes_FindQuiz]
	@search VARCHAR(MAX)
as
begin
	SELECT *
	FROM StoredQuizzes
	WHERE Name LIKE @search
End
```

TeacherLogin_CreateNewTeacher

```
PROCEDURE [dbo].[TeacherLogin_CreateNewTeacher]
	@Title nvarchar(3),
	@SecondName nvarchar(15),
	@Username nvarchar(15),
	@Password nvarchar(256),
	@Email nvarchar(50),
	@Salt nvarchar(25)
as
begin
	INSERT INTO TClass(TeacherId) VALUES(IDENT_CURRENT('TeacherLogin')+1);
	INSERT INTO TeacherLogin(Title, SecondName, Username, Password, ClassId, Email, Salt)
VALUES(@Title, @SecondName, @Username, @Password, IDENT_CURRENT('TClass'), @Email, @Salt);
end
```