Analysis

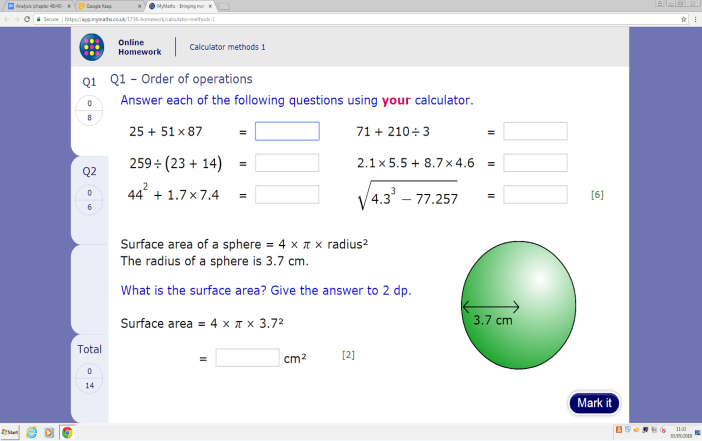
Idea

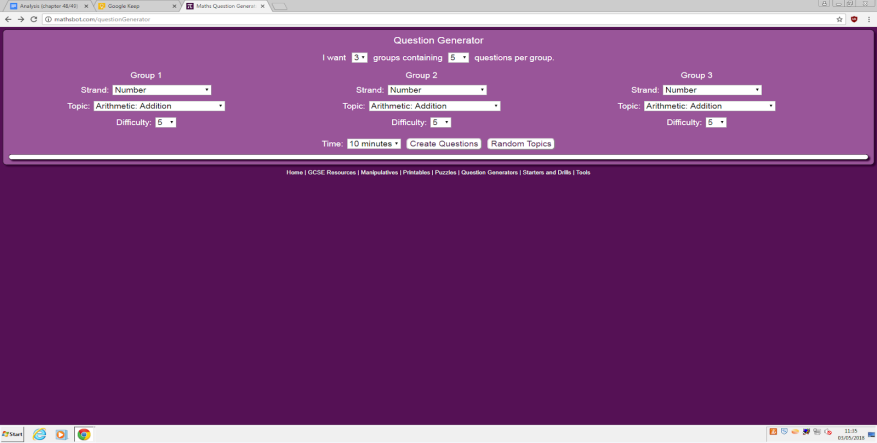
Although computers have made an entrance into essentially every school in the United Kingdom, methods of testing students on their knowledge of a subject is still typically done with the old method of paper and a pen. However, this has its drawbacks such as the teacher having to carry around a stack of papers with them whenever they wish to mark students’ papers, or if it’s a test that a student needs to complete at home, the student may lose the paper or forget to bring it to school on the deadline. There are many factors that make testing on pen and paper inconvenient, and so having a digital equivalent would make students’ and teachers’ lives a lot easier.

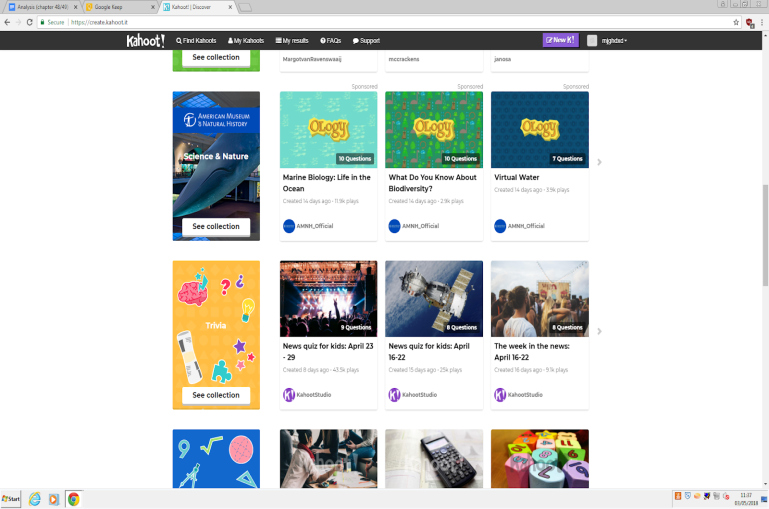
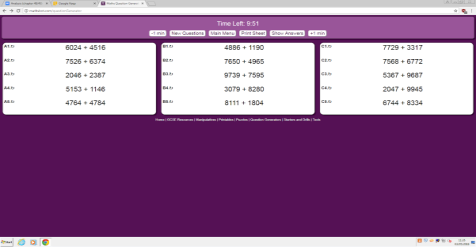
Research

What’s currently available?

There are multiple services available to provide different quiz experiences, however none fully fit the criteria and system that typical pen and paper testing usually ensues.

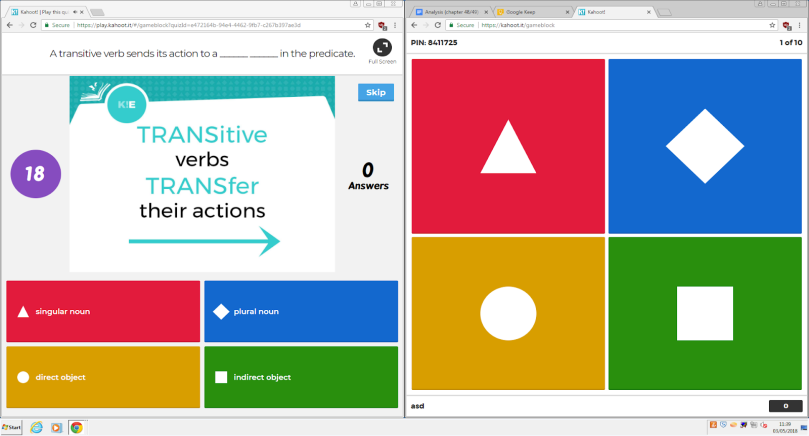
To start, possibly the most popular in the United Kingdom is MyMaths. This is used for maths lessons and consolidation of students’ knowledge, however it lacks the connection between teacher and student and the lack of ability to create custom quizzes means that a teacher may not be able to find exactly the correct quiz to give to their students. Perhaps most importantly, MyMaths only contains maths quizzes, so any other subject teachers cannot use their services, accentuated by the lack of custom quizzes. However, even with these limitations, it is the most sophisticated in terms of features for the teacher, such as being able to create classes to make homework for, as well as being able to see how many answers their students got right and how many attempts it took them. It also retains the pen and paper-like system of students answering a set of questions individually, which is uncommon in quiz websites or programs.

In a similar vein, MathsBot is a website in which a teacher can randomly select questions in relation to a topic that the teacher wishes to test their students on. Although this site lacks custom questions and the digital element that I wish to achieve, as once the questions are picked, the only option is to print it off to give to students physically, the element of randomly choosing questions in relation to a topic is one that will be useful to implement into my program, as it is a unique idea for a quiz program.



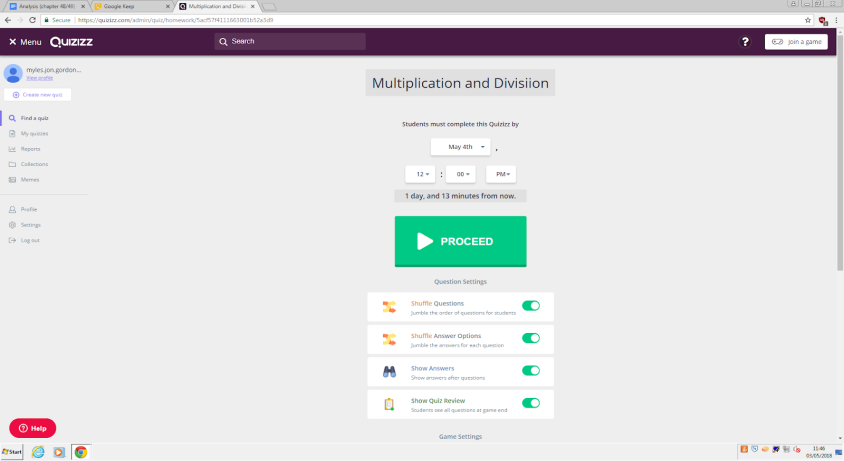
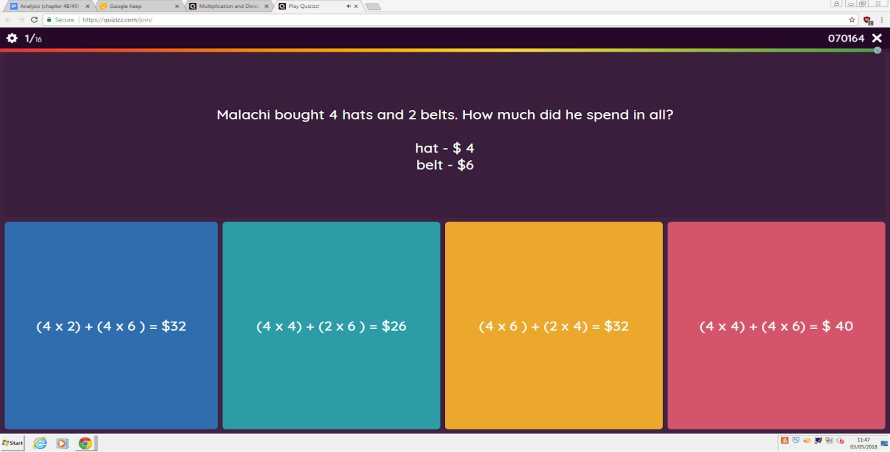
Another popular quiz website used in schools is Kahoot. On Kahoot, a teacher is able to choose or create a quiz and display it on a screen, and then the students must answer the questions in a given amount of time. This creates excitement while answering, and is a great way for students to consolidate material covered in previous lessons, however for a private test to consolidate knowledge or provide exam style questions, this format is unsuitable, as well as it being too informal.

Choosing a quiz on Kahoot

Kahoot also lacks text input answers, which are essential if a teacher wants to test a student’s exam technique.

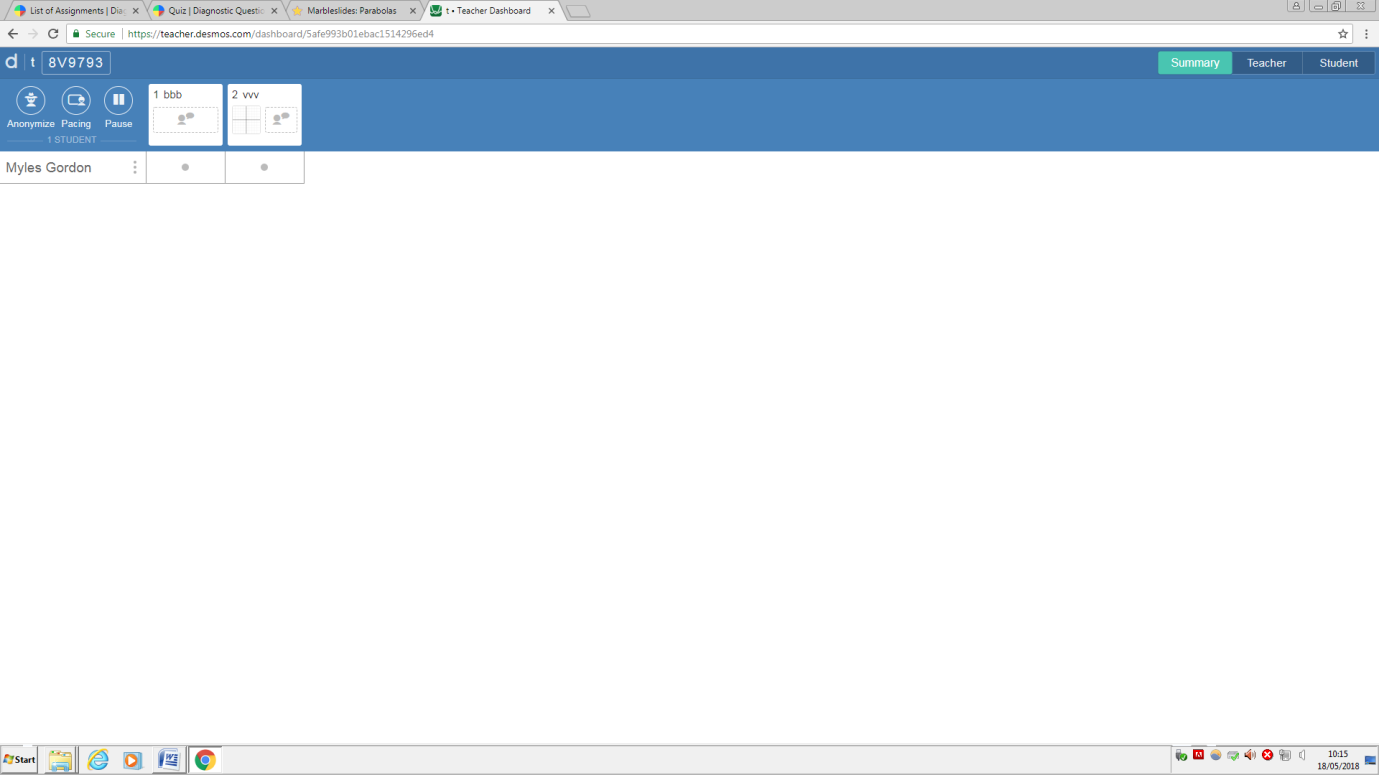
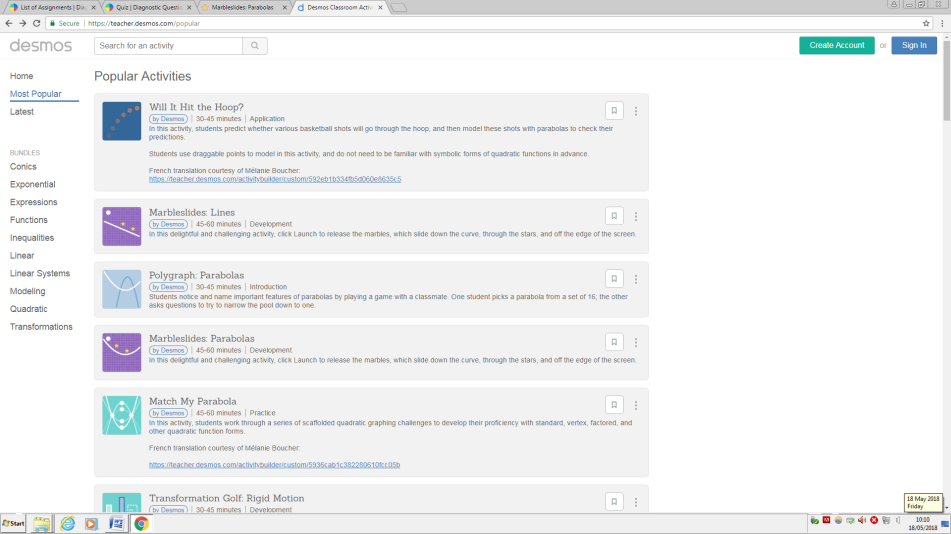
Doing a quiz on Kahoot

As well as this, Kahoot lacks the capability for a teacher to send a quiz to their students to do either at school or at home to do personally, and so the traditional style of testing that students will experience in exams will be lost.

Quizizz is similar to Kahoot, and does include homework tasks to a student to complete at home, which is definitely a step in the right direction. Once a student has finished a quiz, they are able to go over their incorrect answers and amend them, which is a good feature for consolidating knowledge and revision, however the teacher does not see this and so personal recommendations from the teacher cannot occur. Like Kahoot also, it lacks formality, especially with its inclusion of memes between questions and bright colour palette for the UI. It also lacks text input answers, which a teacher may prefer to give their students as it’s closer to exam style questioning.

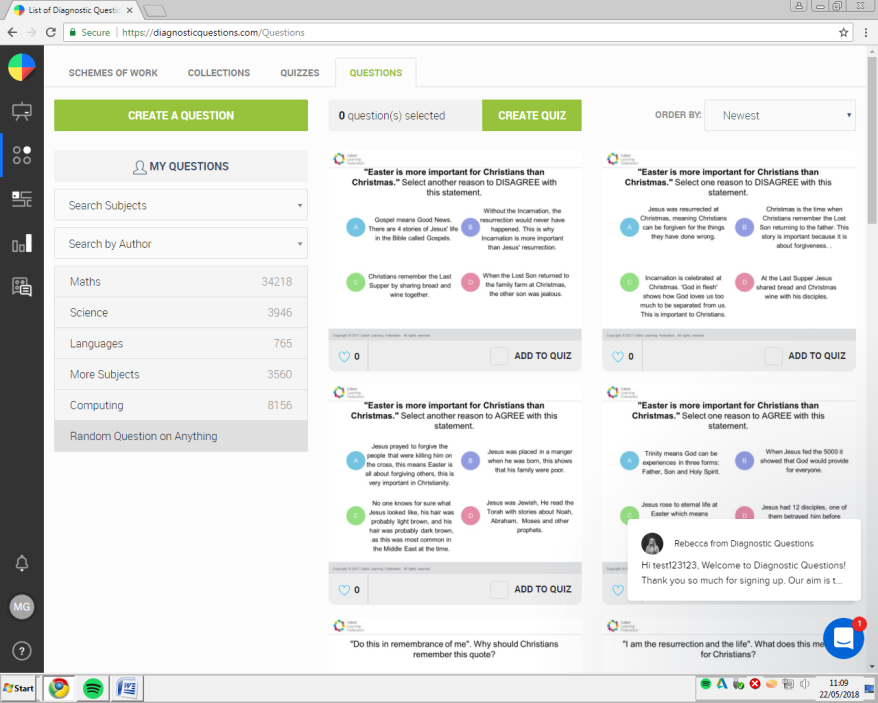
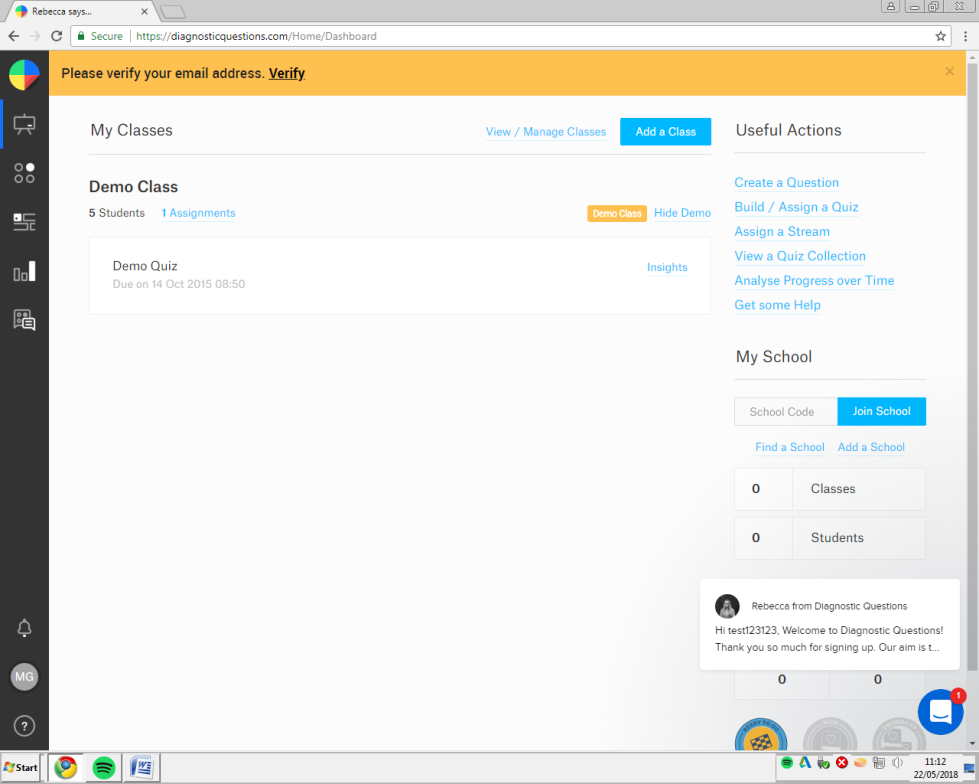
Doing a quiz on Quizizz

Setting a homework on Quizizz

Desmos is a website commonly used in maths lessons, with all of its activities being under mathematic categories. In this website, a teacher can create a class and invite students to particular games, where the teacher can track their progress in the activity. Custom quizzing is available, however instead of having a single class with all of the students to set quizzes to, the student has to join a quiz with a randomly generated code, as well as the teacher having no control over the name of the temporary class. This means that if a teacher wishes to set 3 quizzes to their class, the students will have to quit the quiz each time and then type in a randomly generated code, which is inconvenient for the student, and also the teacher has to keep giving out the quiz codes which is inconvenient for the teacher. However, this site does make good use of marking, with the students’ name being next to all of their answers, but it does lack instant marking in custom quizzes, and so if a teacher wishes to create a simple maths quiz where there’s only one acceptable answer, they have to waste a lot of time marking something that could be automatically marked.

Teacher's marking screen

Browsing the activities

Diagnostic Questions is a website that adopts the separation of teacher/student well, with a student being able to join a class that the teacher can send a quiz out on. A teacher can build a quiz and create questions, or search for a public quiz that they can send out to their classes.

Student's class screen

Browsing through questions

The main problem with this website, however, is that the teacher is unable to create a whole quiz with custom questions without creating each question and uploading it one by one, and then building a quiz with those previously created questions, which is extremely inefficient for a teacher who wishes to create a quiz from questions that they feel is appropriate to their class. Also, the only answering type is multiple choice, which is not always an appropriate way of answering questions, especially in an exam-like fashion for most subjects, which require the student to write a unique answer to a question (e.g. English exams)

User

The people who will benefit the most out of a new quiz program for schools will be the teachers creating the quizzes and looking at their students’ results and the students themselves, who will benefit hugely from a new way to consolidate their knowledge of a subject.

For the student, I asked Alexander Henderson, a student who takes Maths, Computing and Music:

1) Are you satisfied with how schools implement computers currently?

I'm satisfied, however from time to time I'd like to see a little bit more implementation.

2) How often do you use computers at home for homework/other consolidation of knowledge?

More or less every time I'm working at home, there's a computer open to use

3) How beneficial are quizzes to consolidation of knowledge for you personally?

They're fantastic as if you get something wrong you're more likely to remember it.

4) Would you say that a system of teachers sending out quizzes for you to do at any time like a normal homework would be beneficial?

Beneficial's an understatement, it would be very useful for exams, as teachers have their own ideas of what's going to be on, and the idea of being able to do it at any time would be helpful for exams.

5) What do you feel are the main drawbacks of using pen and paper for consolidation of knowledge?

It's slower to answer, you can't lose a digital copy of a quiz, pen and paper doesn't always give you the room to answer a question properly.

6) What would you like to see in a digital quizzing software as a student?

I'd like to see competition with other pupils which would make the quiz taking experience more fun

For the teacher, I asked Mr. Frost, who teaches Maths and Computing:

1) Do you think that students benefit from using websites and other software to consolidate knowledge?

It depends on a few things: the content, but also the user interface of the website is very important, like ease of data entry and feedback given

2) What do you feel are the main drawbacks of using pen and paper for consolidation of knowledge?

The time taken to mark them, there has to be feedback quickly otherwise students forget what they've done, and it's not as effective as having it given straightaway

3) What would you like to see in a quiz software geared towards consolidation of knowledge?

Design a quiz very quickly, minimise the time for questions being in my head and having it in a quiz, need to access student results in a way that's simple and as quick as possible, for example exporting to a spreadsheet

Objectives of the program

1. a) The user should be able to sign up with a username and password, and say whether they’re a teacher or a student

b) Once signed up, the user should be redirected their respective main menu (depending on whether they’re a teacher or student)

c) If the user tries to sign up with an existing username, they should be given an error message and stay on the login/signup page

d) The user should be able to log in with the details that they signed up with

e) If the user has entered incorrect credentials, then they should be told that what they’ve entered is incorrect

f) When someone has logged in, they should get the corresponding menu depending on whether they’re a teacher or student

1. a) If a teacher has logged in, they should be shown a main menu with the options to create a new quiz; open a draft quiz; view the quizzes that they have currently set to their classes as well as their quizzes set history; the quizzes they have yet to mark, view and amend the classes they’ve set up and to view all public quizzes.

b) If a student has logged in, then they should be shown a main menu with the options to view currently set quizzes; review their quiz history; request to join a class and to view all public quizzes.

1. a) A teacher should be able to create a quiz with as many questions as they’d like, custom timing, whether it’s an instant result quiz or not, different types of answering and custom questions.

b) Once the teacher has created a quiz, they should be able to send it to their students and/or be given the option to upload it to the public database of quizzes.

c) If a teacher hasn’t finished a quiz, they should be given the option to save as a draft

1. The teacher should be able to open their draft quizzes, and then edit them or delete them.
2. a) When taking a quiz, as soon as the quiz has started, the timer should start (displayed in the top right corner of the screen) and the user should be taken through the quiz one question at a time, with the question and corresponding answer type (such as multiple choice, text input box etc.). The current question number is also given in the top left corner alongside how many questions there are in total. When the user has finished a question, they can press “next” to go onto the next question. They are also given the option to go back to the previous question to amend their answer.

b) The user can quit at any point during the quiz with the “Quit” button, where they’re given a dialog asking whether they’re sure they want to quit or not.

c) When the user has pressed “Next” on the final question, they should be given a dialog asking if they are sure that they’re finished, and if they press “Yes”, then if it’s an instant quiz, their mark is displayed with their position on the leaderboard, and their mark is sent to the teacher. If it’s not, they’re taken back to the menu that they came back from and their answers are sent to the teacher to mark.

1. A student should be able to check if they have any homework that has been set by the teacher, and if they do, they should be able to take the quiz
2. a) A student or teacher should be able to browse and search the database of public quizzes

b) When a student clicks on a public quiz, if it’s an instant quiz then they should be shown a UI with the leaderboard and a “Play” button, where pressing “Play” should take them to the quiz.

c) When a teacher clicks on a public quiz, they should be given the options to “Play”, “Edit” or “Send to class”. If they press play, then they should play the quiz normally, if they press “Edit” a copy of the quiz can be modified by the teacher as if it were their own to put into their drafts or to send out to their students, and if they press “Send to class”, they are given a dialog as to which class they wish to send it out to.

1. a) A teacher should be able to view the quizzes with answers that they have yet to mark

b) The teacher should be able to click on the quiz that they wish to mark the answers to and be given a list of the questions and the students’ answers. Once they have gone through it, they should be able to give the student a mark, which is sent to the student.

1. a) A teacher should see their set quiz history

b) A student should see their quiz history with the amount of marks they received for it, and the ability to retake any quizzes

c) A teacher when looking at their quiz history should be able to export the student results as a CSV

1. a) A teacher should able to create classes, as well as amend any classes that they have

b) A student should also be able to request to join a class with the unique ID given to the teacher

Critical path of the project

In the program, there is a system of logging out and signing up, which dictates what homework you get, what class you’re in etc. As a result, a database is needed. As a result, I will be running a server program which will be run externally which handles all commands of trying to log in and sign up. The server program will then take this information, and check it against a SQLite database. In the SQLite database, there will be a “User” table for anything regarding users, which will have columns for the username, the password and the type of user (teacher or student). Due to the handling of passwords, SHA256 encryption will be used to store the passwords, as well as for when the password is being sent to the server in case an illicit packet sniffer is able to read the serialised data stream going to the server. If the response for the signing up or logging in query shows that the username and password were correct, then a session is created to store the username (to be displayed at various parts of the program) and the type of user (teacher or student) so that certain parts of the program can be adapted for whether the user is a student or teacher. If the response shows that it was false, then the user is shown a dialog saying that their details were incorrect, where they can then retype the username and/or password.

In the program, the teacher is able to create a quiz and save it as a draft or upload it to the public database and/or share it with a class. For this, I will need to create a custom file type and a “Quiz” table. In the custom file type, there will be a section for metadata including the name of the quiz that the user will see and the unique ID that is randomly generated and a section for the questions themselves. Each question will be one line in the file, with a comma separating the type of values that each question will require - the question itself, whether it’s an instant answer question (and the answer if it’s an instant answer question) and the type of answer. This unique ID is put into the “Quiz” table when the teacher saves it as either a draft or for the public database, with rows for the ID, username of the person who made the quiz and whether it is public or not, as well as if it’s a draft quiz

In order for the program to communicate with a database, I have decided to create a server side to the program to be hosted away from the client’s computer. This allows for server and client-side tasks to be done separately, and will allow for any server-side tasks such as accessing a database to be much easier. To send commands to the server, the client side will send the command via TCP (Transmission Control Protocol) which will then be received and interpreted by the server side, which will then send a response once it has completed the task. In order to allow multiple clients to send commands at once, this interpreting of commands and completing tasks will be done on different threads which will be created when there is a pending TCP request.

For any sort of UI, I have decided to use WPF (Windows Presentation Foundation, a GUI framework) with MVVM (Model-View-ViewModel, an architectural pattern), using the Stylet framework (<https://github.com/canton7/Stylet/>). This was the clear choice for me, as MVVM means that multiple pages are able to be created without any sort of issue, and it allows for extremely to read and maintain code, especially when properties set to notify the UI when there is a change are created. MVVM also means that conductors can be used, so the initial viewmodel (known usually as the ShellViewModel) can essentially choose what page is being shown and when. In order to incorporate this into my program, I will have the ShellViewModel class incorporate the event handling interface and conductor interface of Stylet, which will mean that I can raise an event on one page saying to change page to a certain page when something has happened, and then the ShellViewModel will activate the new ViewModel due to the conductor, and so showing a new page. This will be used, for example, on a successful login attempt, where the LogInViewModel will raise an event for changing page to the page number of the main menu (corresponding to whether they’re a teacher or student). The ShellViewModel will handle this event and activate the ViewModel of the teacher/student page, showing the main menu.

In order for the quiz to be shown one question at a time, there will be a counter corresponding to what line of the quiz file that the program is currently on, which will increase when the “Next” button is pressed, and will decrease when the “Back” button is pressed. Then, whenever a question is to be displayed, all of the information for the question will be read from that line that the counter corresponds to, due to the file format for the quizzes being one line per question (apart from the metadata).

In order to have the results of a quiz exported as a CSV (comma separated values), I will need to create a class to handle CSV creation, which will create a CSV file with the headers for student name, mark and percentage. The reason for creating a CSV file is that it is easily opened in Excel and other spreadsheet programs. It will also be easy to create a CSV file compared to other file types like .xls, as all CSV requires is a string of text, a comma and another string of text, with a new line for each new entry, which can be easily created using a loop to get the data for each of the lines, the StringBuilder() class to collect all of the data together to write to the file, and then the File.WriteAllText function. This CSV processing will be done server-end, as it will be easier to make SQL calls from the server-side. The finished CSV file will be transferred over TCP to be received by the client.

For some of the features of the program such as having quizzes in your to-do section, I will need to create separate tables for each user. These tables will be created upon user creation, and having these separate tables for each user for a specific purpose such as all the quizzes that the teacher has set them will make it so that loading all of the required information will be faster than having an all in one set quiz table with every user’s set quizzes, which would take a while to sort through and would be inefficient.

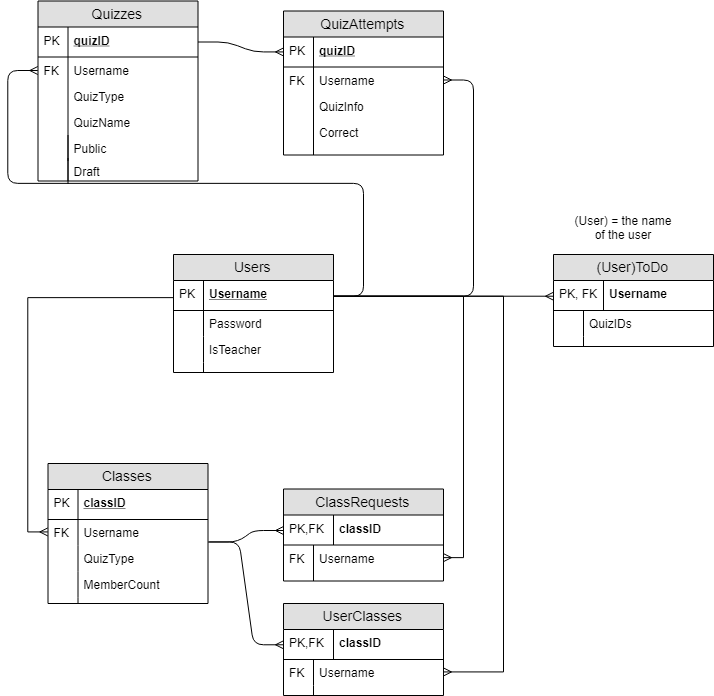
# Design

In the program, the MVVM architectural pattern will be used, meaning that each “page” of the program (such as a main menu, or the log-in page) will have 3 elements to it – a Model, a ViewModel and a View. The Model is essentially just a class of properties that both the View and ViewModel are able to read without interfering with the MVVM pattern (as the View and ViewModel shouldn’t be able to interact directly with each other). The ViewModel is where most of the logic takes place and handles all of the events and other functions in the program, and the View is the actual UI itself and also dictates what should happen when a button is clicked, for example.

However, the MVVM pattern cannot function without a graphical system, and for this I will be using WPF in conjunction with the framework Stylet. WPF is the follow up to Microsoft’s WinForms, with the main difference being the introduction of XAML (Extensible Application Markup Language, a markup language for designing in WPF), which forces a separate file for the View to be made in the application. For general ease of programming, I will be using Stylet, which is similar to the popular Caliburn Micro framework where it essentially makes using MVVM and WPF a lot easier. For example, Stylet handles all of the binding between ViewModel and the View, meaning that when a new object of a ViewModel is made in a conductor (which in my case will be the ShellViewModel), the respective View is also called.

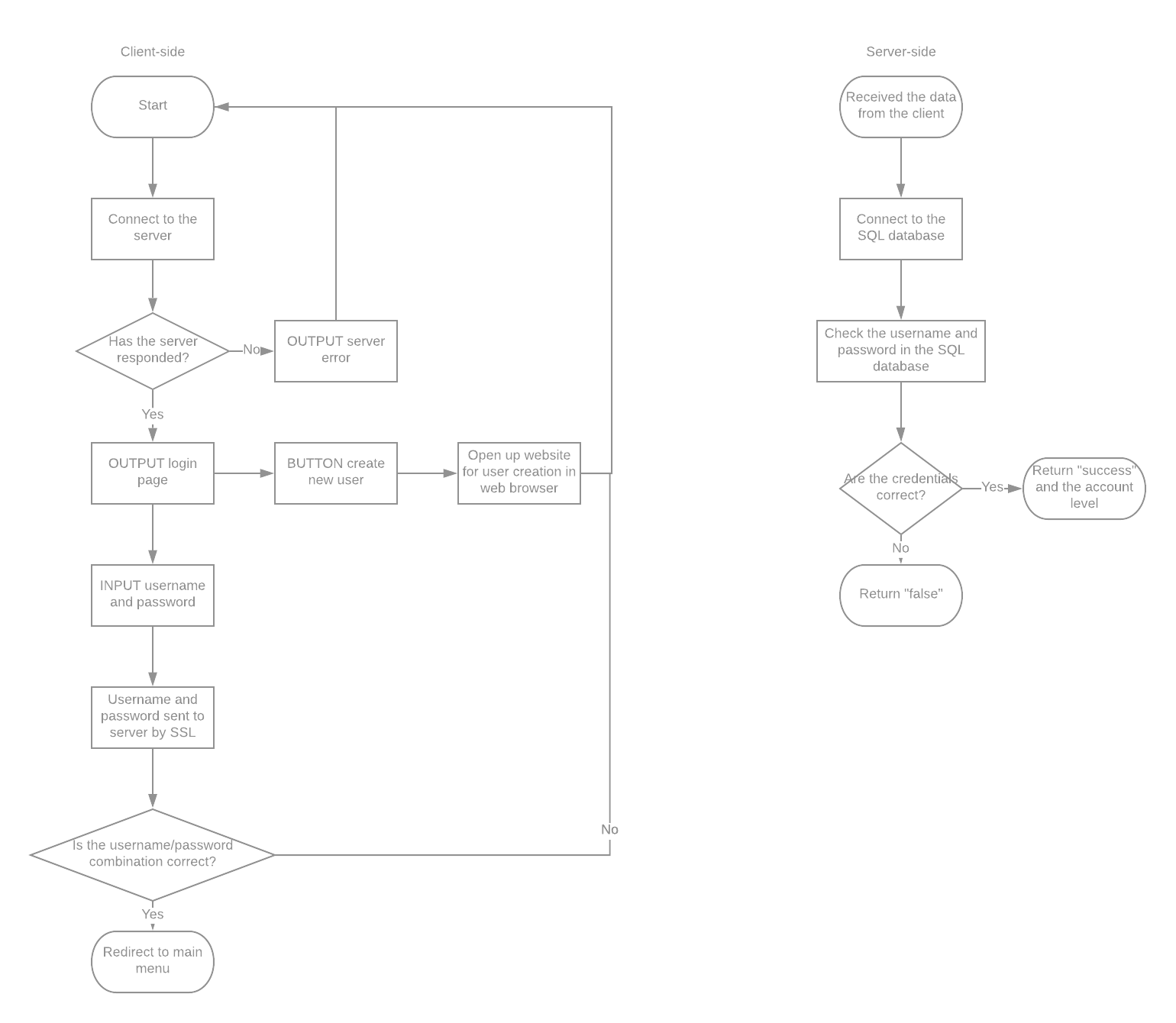
The quizzes in this program will all be stored as .zip files, with the name of them being the ID of the quiz. This is for the simple reason of there being a Microsoft library for opening and extracting the contents of .zip files, and it will be easier to transfer one file over TCP/IP than it will transmitting multiple files, as there is a “SendFile” function in the C# System.Net.Sockets library. Inside the .zip files, there’ll be a “questions.txt”, “answers.txt”, “metadata.txt” and “questiontypes.txt”. The questions.txt file will contain all of the questions, with each new line being a new question; the answers.txt file will contain all of the answers (if it’s an instant quiz) with each new line being an answer to the respective question; metadata.txt will contain the time allocated to the quiz and questiontypes.txt will dictate what type of question each question is, for example if it’s a multiple choice question or an integer input question.

The database for Tackle (the name of this project) will be using SQLite. This is because of the speed of using SQLite, and it also alleviates the inconsistencies and latency of using MySQL, which runs as a separate server on the server side. All of the tables and rows will be contained in one .db file to ensure that, when necessary, the relational database model is used, which is not possible when multiple database files are used. The database schema is shown below:



On the server side, whenever a command is sent to the server via TCP from the client, if the command requires something to do with the database, such as logging in, a new database connection will be established to the SQLite database using System.Data.SQLite. This library ensures the easy use of reading of data from an SQLite database using SQLiteDataReader, and also can execute SQL queries to the database.

For logging in, to ensure that the user’s username and password hash is identical to the details they supplied when they signed up, this SQL (structured query language) query will be run: “SELECT \* FROM Users WHERE username = '{username}' AND password = '{password}'” (where {} are the C# variables to be queried). The algorithm for this is shown below



For signing up, this simple SQL query is run: “INSERT INTO Users (Username,Password,UserType) VALUES ('{username}','{password}',{userType})”, with userType being either 0 for a student, and 1 for a teacher.

The way in which the server will check and execute commands has to be optimised for checking for and handling multiple requests at once, otherwise the server response time for clients will be extremely high, especially with more use. As a result, a new thread will be created that calls the function to handle the client’s request. The use of threads means that instead of the checking of whether a client is trying to connect and each individual request handling are separated and able to be executed at the same time, much like how asynchronous functions work by separating one specific task from the UI thread and other task threads.

Also in regards to the client-server communication, JSON (JavaScript Object Notation) will be used to transfer objects of classes which will be serialised at the client side and then deserialised on the server side. For example, when the quiz is submitted, there’ll be a QuizResults class:

1. **public** **class** QuizResults
2. {
3. **public** **int** quizID;
4. **public** **string** username;
5. **public** **string** quizType;
6. **public** **string**[] questions;
7. **public** **string**[] answers;
8. **public** **int** correctTotal;
9. **public** List<**bool**> correct;
10. }

When serialised to JSON on the client side, it will have this structure:

{

"quizID": 0,

"username": "teststudent",

"quizType": "Instant",

"questions": ["What is 2+3?"],

"answers": ["5"],

"correctTotal": 1,

"correct": [true]

}

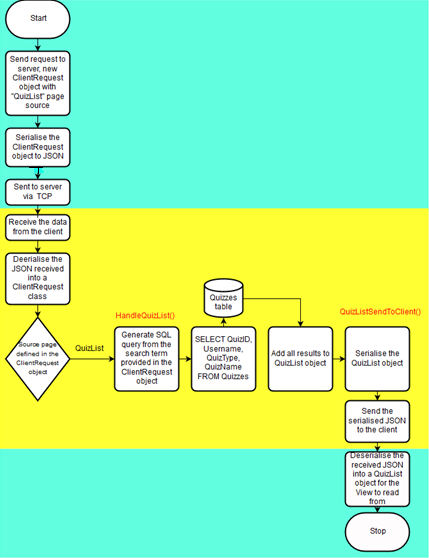
This will be using the “Newtonsoft.Json” library due to its ease of use for this task. For example, on the server side, deserialisation from JSON to a C# class object is all one in one line:

##### var clientRequest = JsonConvert.DeserializeObject<ServerRequest>(jsonReceived);

This use of serialisation ensures consistency on both client and server side by using the same classes, and it also means that when deserialised on the server side, it is a lot easier to just read a variable from a class object than it is having some sort of array and having the values stored in that way.

The reason for using JSON is that JSON is able to handle the use of arrays (with the parameters for sending information to the server being enclosed in a string array for example), whereas XML is not able to. Also, JSON has the advantage of being faster than XML (Extensible Markup Language) due to XML taking up more characters by being more verbose, and so JSON parsing libraries tend to be faster. Also, the storing of JSON due to it being less verbose in the database will take up less storage and will take less time to transfer from server to client.

A great example of where this JSON serialisation will be used is when retrieving the list of public quizzes, which is shown below in this diagram: (Turquoise = client side, yellow = server side)



As seen, the “QuizList” object, used to store all of the relevant information when searching for a quiz is serialised to JSON, and then deserialised on the client side to be used by the program. Instead of having to create a more complex algorithm for handling server responses in regards to areas such as quiz lists, this is a much more consistent and simple method of transferring objects from server to client and vice versa. This diagram also illustrates how the client-server connection will work fully – the ClientRequest class will consist of two members – the source of the request and a string array consisting of anything that the server side may need, such as the username, quiz ID, class ID etc. This ClientRequest object is then serialised and transferred over TCP/IP to the server side where it’s deserialised. The request is then handled based off the source of the request, and then the result is sent back to the client, in this case a serialised “QuizList” object, which contains the quiz ID, username, quiz type and quiz name to be displayed. This response is then handled appropriately by the client side.

However, this JSON serialisation will not always be necessary with server responses. For example, when the teacher creates a class to add students to, the only response that the client needs is whether it’s a success or not. In this case, a simple string saying “success” or “failed” will be returned rather than a full JSON serialisation, for example when the teacher has finished creating a quiz and they send it to the server to be saved as a draft – the only response that is needed here is whether the operation was a success or failure.

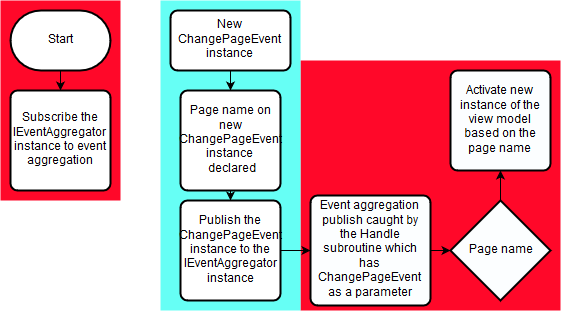
When the teacher is creating a quiz, lists will be used to store the information about each question such as the questions, the answers and the type of question. This use of lists is required as in C#, variable types with multiple elements are split explicitly into dynamic and static (array and list respectively). An array would be inappropriate to use here as the amount of questions is entirely up to the user, and so having a dynamic structure would be appropriate in order for the teacher to create as many questions as they would like and being able to delete any questions they’d like at any point. As a result, whenever lists are used, they’ll need to be initialised. For example, in CreateQuizModel.cs, I’ll need to initialise the lists in the constructor like so:

1. **public** **class** CreateQuizModel : PropertyChangedBase
2. {
3. **private** List<**string**> \_questions;
4. **public** List<**string**> Questions
5. {
6. **get** { **return** **this**.\_questions; }
7. **set** { SetAndNotify(**ref** **this**.\_questions, value); }
8. }
9. **public** CreateQuizModel()
10. {
11. **this**.Questions = **new** List<**string**>();
12. }
13. }

One other feature that is shown in the code above is the implementation of Stylet’s “PropertyChangedBase” interface (the implementation is seen on lines 6 and 7). This interface is essentially a wrapper for the C# interface “INotifyPropertyChanged” which updates the UI (user interface) whenever the contents of a variable change. The “PropertyChangedBase” implementation of this in Stylet makes it so that the only function that needs to be called for this to occur is “SetAndNotify” which sets the public variable to the private variable’s contents to update it, and then updates the UI thread to tell it that it needs to update its interface to show the new content. This also means that “Mode=TwoWay” when defining a XAML property no longer needs to be called, making the XAML less cluttered.

Another Stylet feature that I will be implementing is its implementation of the “ObservableCollection” C# type called “BindableCollection”. Throughout the program I intend to use the “ItemsSource” XAML property, which is unable to bind to typical lists and arrays, and so “ObservableCollection” is usually used, despite it acting remarkably similar to a List (as it is a dynamic data structure). For consistency purposes, I will be using Stylet’s implementation, however Stylet’s implementation does have the added bonus of it being thread-safe, with all of its features such as “Add” to add an element to the BindableCollection being dispatched to the UI thread, meaning that the method being called won't return until the action has been completed, which is good for efficiency.

The next Stylet feature that I will be including is the “IEventAggregator” interface. This interface in essence means that when an event is raised by one page or dialog, the values put into that event are then handled by another page. To put this into a more practical usage (and the usage that I’ll mainly be using it for), the ShellViewModel page controls which page is currently displayed on the screen by a WPF feature called “UserControl”. However, WPF doesn’t include an easy way to navigate between these UserControls. As a result, I will create a class called “ChangePageEvent” which will include the page that the user is trying to go to, which will be handled by the ShellViewModel. As a result, when the user wants to change pages, for example if they click on a button in the main menu, the ShellViewModel knows exactly what UserControl to activate. This algorithm is shown in the diagram below: (Red = ShellViewModel.cs, blue = any other view model)



### Page table

|  |  |  |  |
| --- | --- | --- | --- |
| File | Input | Output | Description |
| Program.cs | ClientRequest object, includes the page source and any needed parameters | Serialised JSON or a simple string relevant to the request | This is the server side of the program, and is a separate executable file altogether. It takes the user’s request and handles it usually by reading or inserting into the database. It then returns whatever is needed, whether it be serialised JSON, a string saying whether it was a success or in the case of logging in, whether the user is a teacher or not. |
| ShellViewModel.cs | - | - | Handles event aggregation and page navigation – is also the entry point to the program, pointed to by the bootstrapper |
| ShellView.xaml | - | - | Provides the UserControl property so that all of the views are able to be displayed |
| CreateQuizViewModel.cs | - | - | Handles all of the logic for creating a quiz such as the subroutines bound to the buttons |
| CreateQuizView.xaml | - | - | The view for the quiz creation process |
| CreateQuizModel.cs | - | - | Handles all of the properties to be accessed by the quiz creation process in the view model and view |
| JoinClassViewModel.cs | - | - | Handles all of the logic for joining a class such as submitting the request to the server |
| JoinClassView.xaml | - | - | The view for the class joining process – includes a simple text box and submit button |
| JoinClassModel.cs | - | - | Handles the properties accessed by the view model and view in the class joining process |
| LogInViewModel.cs | - | Username, user type | Handles the logic for logging in and signing up |
| LogInView.xaml | - | - | The view for the logging in/signing up process |
| LogInModel.cs | - | - | Handles the properties accessed by the view model and view in the logging in/signing up process |
| ManageClassesViewModel.cs | - | - | Handles the logic for the teacher managing their classes, such as sending server requests to accept a class join, deleting a class or creating a new class |
| ManageClassesView.xaml | - | - | The view for class management |
| ManageClassesModel.cs | - | - | Handles the properties accessed by the view model and view in the class management process |
| QuizListViewModel.cs | - | Quiz result CSV (if teacher) | Handles the logic for listing all the public quizzes and quiz history |
| QuizListView.xaml | - | - | The view for viewing all public quizzes and viewing quiz history |
| QuizListModel.cs | - | - | Handles the properties accessed by the view model and view while viewing public quizzes and quiz history |
| DraftQuizListViewModel.cs | Username | - | Handles the logic for listing all the draft quizzes that the teacher has |
| DraftQuizListView.xaml | - | - | The view for viewing the teacher’s draft quizzes |
| DraftQuizListModel.cs | - | - | Handles the properties accessed by the view model and view while viewing draft quiz lists |
| HomeworkListViewModel.cs | Username | - | Handles the logic for listing all the homework that the student has to do |
| HomeworkListView.xaml | - | - | The view for viewing the student’s to-do homework |
| HomeworkListModel.cs | - | - | Handles the properties accessed by the view model and view while viewing the list of homework to do |
| QuizScreenViewModel.cs | Quiz ID | - | Handles the logic for taking the quiz |
| QuizScreenView.xaml | - | - | The view for the quiz |
| QuizScreenModel.cs | - | - | Handles the properties accessed by the view model and view in the process of taking the quiz |
| StudentMainMenuViewModel.cs | - | ChangePageEvent object raised to the event aggregator | Handles the logic for the student’s main menu, mainly just button logic |
| StudentMainMenuView.xaml | - | - | The view for the student main menu |
| TeacherMainMenuViewModel.cs | - | - | Handles the logic for the teacher’s main menu, mainly just button logic |
| TeacherMainMenuView.xaml | - | - | The view for the teacher main menu |
| ViewQuizViewModel.cs | QuizAttempt object | - | Handles the logic for a teacher marking a quiz |
| ViewQuizView.xaml | - | - | The view for the quiz marking process |
| ViewQuizModel.cs | - | - | Handles the properties accessed by the view model and view in the quiz marking process |
| TeacherQuizHistoryViewModel.cs | - | QuizID (to take or edit a quiz) | Handles the logic for viewing the teacher’s set quiz history |
| TeacherQuizHistoryView.xaml | - | - | The view for viewing the teacher’s set quiz history |
| TeacherQuizHistoryModel.cs | - | - | Handles the properties accessed by the view model and the view when the teacher views their set quiz history |
| StudentQuizHistoryViewModel.cs | - | - | Handles the logic for viewing the student’s set quiz history |
| StudentQuizHistoryModel.cs | - | - | Handles the properties accessed by the view model and the view when the student views their set quiz history |
| StudentQuizHistoryView.xaml | - | - | The view for viewing the student’s set quiz history |
| MarkListView.xaml | - | - | The view for viewing the teacher’s marking list |
| MarkListViewModel.cs | - | QuizAttempt object | Handles the logic for the teacher viewing their marking list |
| MarkListModel.cs | - | - | Handles the properties accessed by the view model and view when the teacher is viewing their list of quizzes to mark |
|  |  |  |  |

Technical Solution

Server-side

Client-side

# Testing

### Test plan

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test no. | Objective | Test | Type | Test Data | Expected Result |
| 1 | 1a | Test whether a student can sign up with the expected response | Normal | Username: “student1”  Password:  “student123” | The new user should appear in the Users table in the database |
| 2 | 1a | Test whether a teacher can sign up with the expected response | Normal | Username: “teacher1”  Password:  “teacher123” | The new user should appear in the Users table in the database |
| 3 | 1a | Test the response of the program if a username and/or password hasn’t been entered | Erroneous | Username:  “”  Password:  “”  Username:  “”  Password:  “123”  Username:  “test”  Password:  “” | There should be a dialog box informing the user that they need to enter a username and/or password |
| 4 | 1b | Test if a student sign up redirects to the student main menu | Normal | - | The student should be redirected to their main menu, with the contents seen in objective 2b |
| 5 | 1b | Test if a teacher sign up redirects to the teacher main menu | Normal | - | The teacher should be redirected to their main menu, with the contents seen in objective 2a |
| 6 | 1c | Test if the program rejects a pre-existing username when signing up | Normal | Username:  “student1”  Password:  “test123” | There should be a dialog box informing the user that that user already exists |
| 7 | 1d, 1f | Test if a student can log in with a correct username and password and be redirected to the student main menu | Normal | Username:  “student1”  Password:  “student123” | The student should be redirected to the student main menu after logging in |
| 8 | 1d, 1f | Test if a teacher can log in with a correct username and password and be redirected to the teacher main menu | Normal | Username:  “teacher1”  Password:  “teacher123” | The teacher should be redirected to the teacher main menu after logging in |
| 9 | 1e | Test if the program rejects an incorrect username and/or password | Normal | Username:  “wrong”  Password:  “student123”  Username:  “student1”  Password:  “wrong” | There should be a dialog box informing that the username or password is incorrect |
| 10 | 2a | The teacher main menu should all relevant options (see objectives) | Normal | - | The teacher should be able to see and click on all of the options given to them seen in objective 2a |
| 11 | 2b | The student main menu should all relevant options (see objectives) | Normal | - | The student should be able to see and click on all of the options given to them seen in objective 2b |
| 12 | 3a | Test to see if the teacher can create a quiz with a standard amount of questions, sufficient time allocated, an instant or non-instant quiz and different questions and question types | Normal | 10 questions, 600 seconds, instant quiz, every type of question included | - |
| 13 | 3a | Test to see if the teacher can create a quiz with a large amount of questions | Extreme | 50 questions, 6000 seconds, non-instant quiz, every type of question | - |
| 14 | 3b | Test to see if the teacher can send a quiz to their class | Normal | A quiz with 5 questions | The SetQuizzes table should have the relevant class ID set by the teacher, and the quiz ID relevant to the quiz. The Quizzes table should have the quiz JSON and be marked as not public |
| 15 | 3b | Test to see if the teacher can make a quiz public | Normal | The quiz in test 12 | The Quizzes table should have the quiz JSON as submitted by the teacher and be marked as public |
| 16 | 3c | Test to see if the teacher can save a quiz as a draft | Normal | A quiz with 5 questions | The Drafts table should have a new entry with the quiz JSON and the teacher’s username |
| 17 | 4 | Test to see if the teacher can open their draft quizzes | Normal | The quiz set in test 16 | The draft screen should show the draft quiz set in test 16 |
| 18 | 4 | Test to see if the teacher can edit their draft quizzes | Normal | The quiz set in test 16 | The quiz editing page should appear with all of the quiz information that was saved as a draft |
| 19 | 4 | Test to see if the teacher can delete their draft quizzes | Normal | The quiz set in test 16 | The Drafts table should have the relevant record removed, and it should disappear from the drafts viewing page |
| 20 | 5a | Test to see if the user can normally take a quiz (as per objective 5a) | Normal | The quiz set in test 15 | The user should be able to input answers to relevant questions as per the question type, with the timer ticking down from the teacher’s set time limit |
| 21 | 5b | Test to see if the user can quit the quiz | Normal | - | The user should be able to quit the quiz and return to the relevant main menu |
| 22 | 5b | Test to see if the user can select “Quit” but change their mind | Normal | - | The user should be able to click on “Quit” and then decide against it to return to taking the quiz |
| 23 | 5c | Test to see if finishing the quiz works normally (as per objective 5c’s requirements) | Normal | - | The QuizAttempts table should have a new row containing the JSON containing the questions, the answers and whether the user got it correct or not (for instant quizzes). The user should be redirected to a leaderboard screen with the highest scores |
| 24 | 6 | Test to see if a student can check their homework and take it | Normal | The quiz set in test 14 | A student should be able to see what homework they have to do, and then take it as if it were a normal quiz. |
| 25 | 7a | Test to see if a user can browse public quizzes | Normal | - | All of the public quizzes should display |
| 26 | 7a | Test to see if a user can search for existing public quizzes | Normal | Quiz name set in test 15 | The quiz set in test 15 should appear |
| 27 | 7a | Test to see if a user can search for non-existing public quizzes | Erroneous | “asodija” | No quizzes should appear |
| 28 | 7b | Test to see if a student can take a quiz from the public quizzes page | Normal | Quiz set in test 15 | The quiz set in test 15 should appear in the quiz taking screen as per test 20 |
| 29 | 7c | Test to see if a teacher can take a quiz from the public quizzes page | Normal | Quiz set in test 15 | The quiz set in test 15 should appear in the quiz taking screen as per test 20 |
| 30 | 7c | Test to see if a teacher can edit a quiz from the public quizzes page | Normal | Quiz set in test 15 | The quiz set in test 15 should be put into the editing page |
| 31 | 7c | Test to see if a teacher can send a quiz to their class from the public quizzes page | Normal | Quiz set in test 15 | The quiz set in test 15 should be sent to a class |
| 32 | 8a | Test to see if the teacher can check their to-mark list | Normal | - | The test taken in test 24 should appear, as well as any other finished tests |
| 33 | 8b | Test to see if the teacher is able to mark the student’s attempt question by question | Normal | Quiz taken in test 23 | The answers given in test 24 should appear one by one with a checkbox underneath to say whether the answer is correct or not |
| 34 | 9a | Test to see if the teacher can see their set quiz history | Normal | - | Any quizzes set by the teacher should appear, such as the one set in test 31 |
| 35 | 9b | Test to see if the student is able to see their quiz history with the amount of marks they got for each quiz | Normal | Quiz marked in test 33 | Any tests taken should appear such as the one in test 24 and then the mark given for it, which may be automatic if it’s an instant quiz or manually marked or rectified by the teacher in test 33 |
| 36 | 9c | Test to see if the teacher can export student results of a quiz as a .csv | Normal | Quiz marked in test 33 | All of the student results of a quiz should be exported as a .csv file, with the quiz marked in test 33 being an example |
| 37 | 10a | Test to see if the teacher can create a class | Normal | - | A new class should appear in the Classes table with a unique number (set by SQLite auto incrementing), and should appear on the teacher’s class list |
| 38 | 10a | Test to see if the teacher can remove a member from their class | Normal | - | The student should be removed from the class via the UserClasses table |
| 39 | 10b | Test to see if a student can request to join a class given a correct ID | Normal | ID given to the teacher in test 37 | The student’s request should be put into the ClassRequest table in the database |
| 40 | 10b | Test to see if a student can request to join a class given an incorrect ID | Normal | ID - 23409823 | The student’s request should be declined by the server and then a message box should appear telling them that the ID was incorrect |

### Test evidence

1. Fully working with expected result – password appears scrambled due to hashing. I introduced a character limit of 40 characters to both the username and passwords as neither of which typically use more than that, and a set limit needs to introduced to prevent buffer overflow attacks, or simply one of them being too long for the buffer accidently resulting in the program most likely crashing.



Figure 1: signing up with the specified username and password

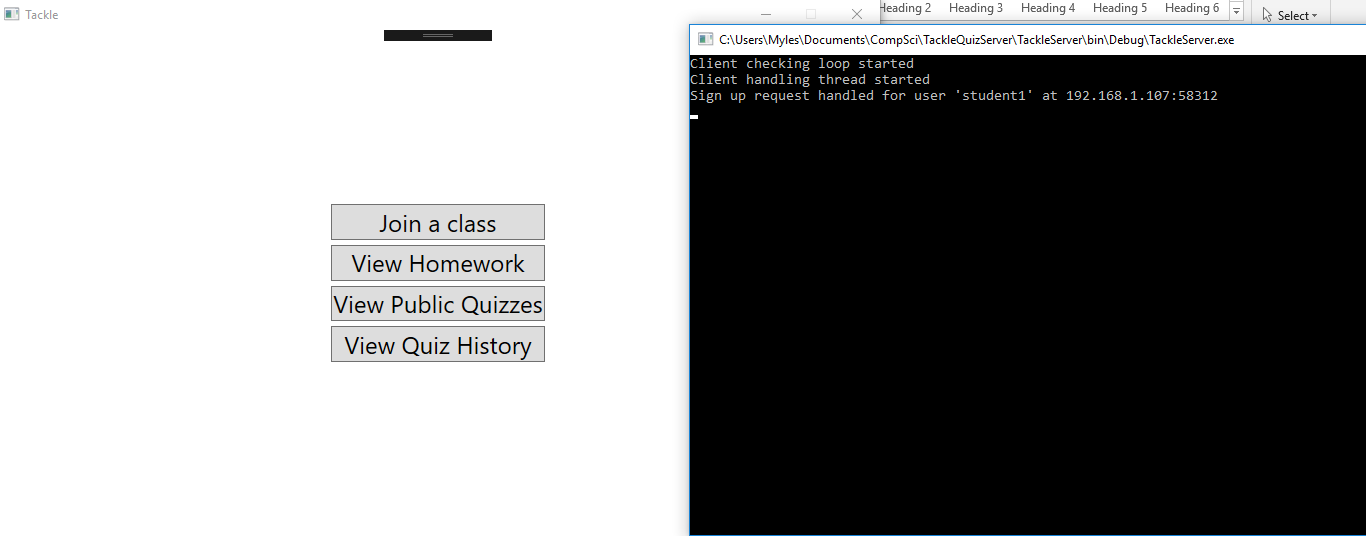


Figure 2: succesfully redirected to the StudentMainMenu

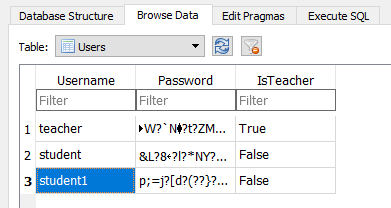


Figure 3: successfully added to the Users table

**2)** This test is fully working, with the same username and password character limits being applied here.

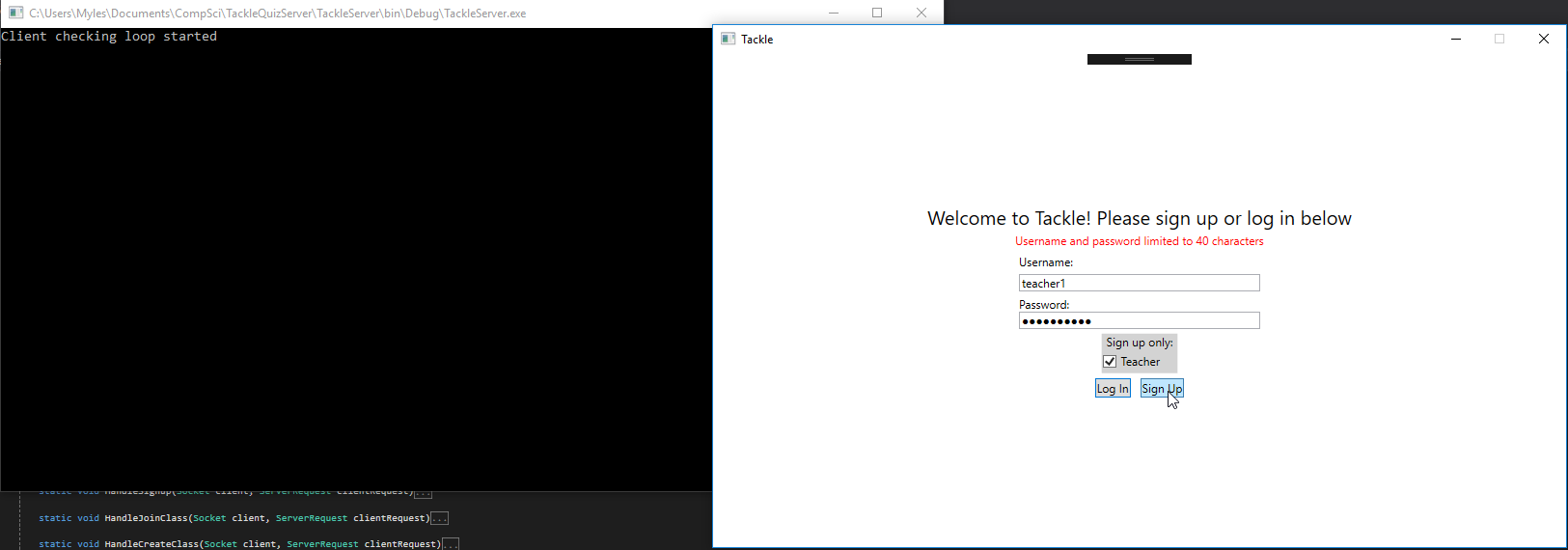


Figure 4: Signing up with the specified username and password

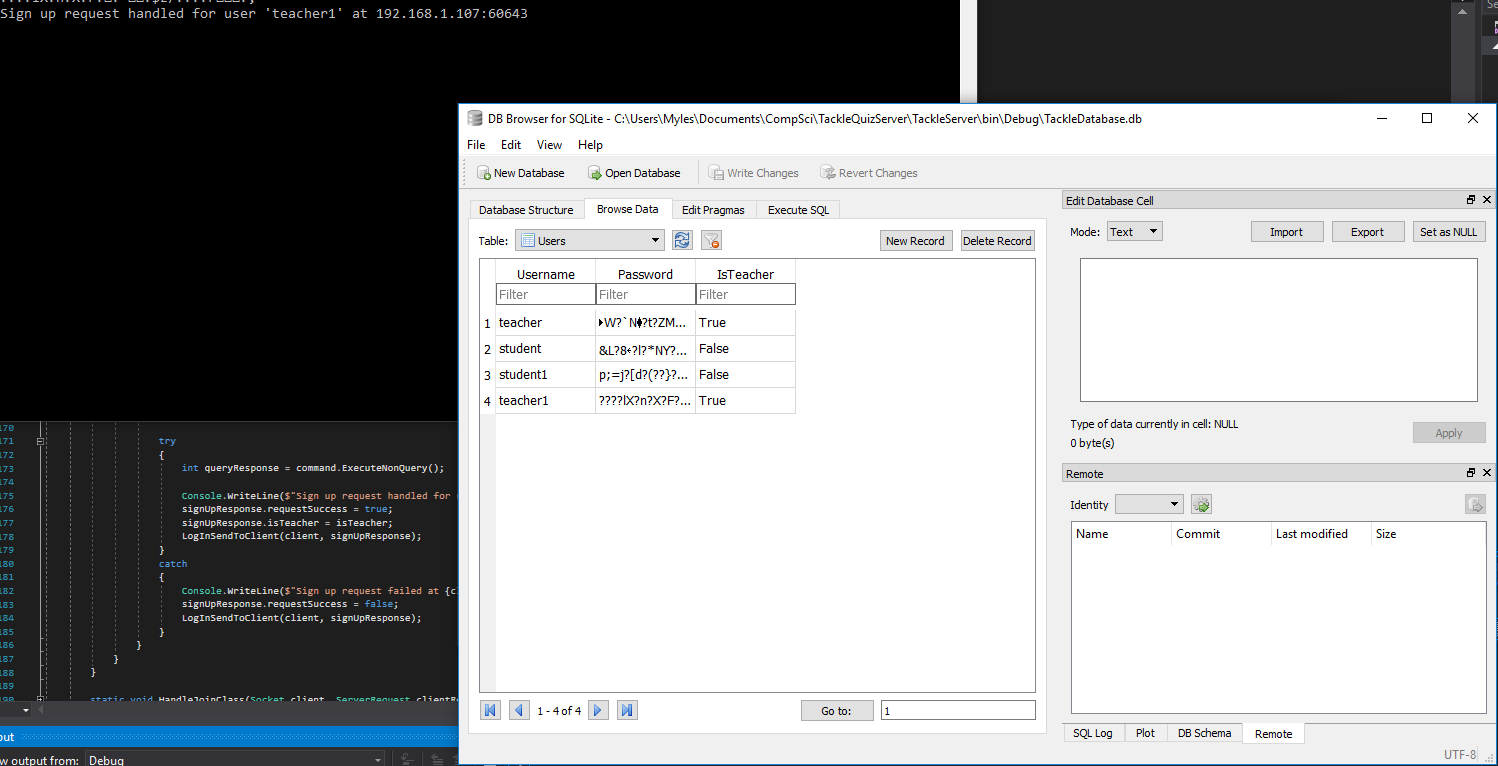
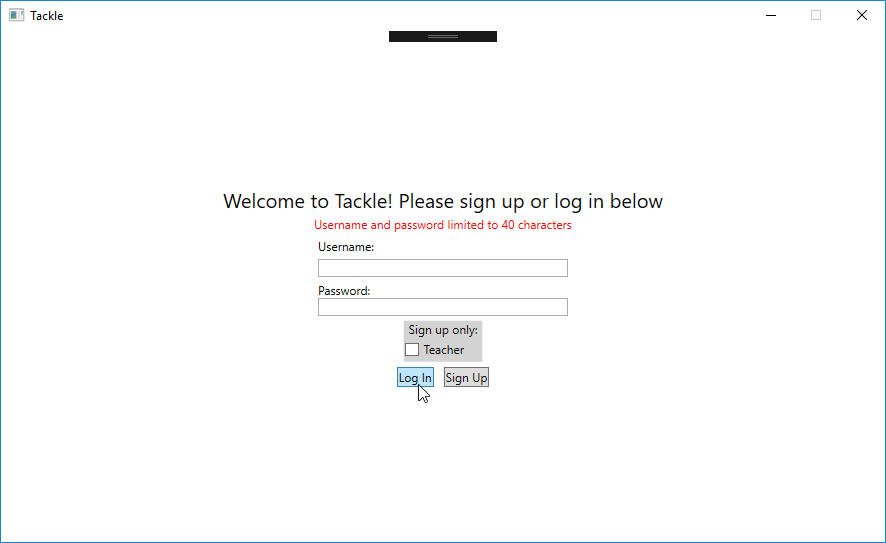


Figure 5: "teacher1" user added succesfully to the database

**3)** This test works as expected. The program checks if the username and/or password textbox has no input, and if it doesn’t have an input then a message box shows up telling the user that they need to input both the username and password.



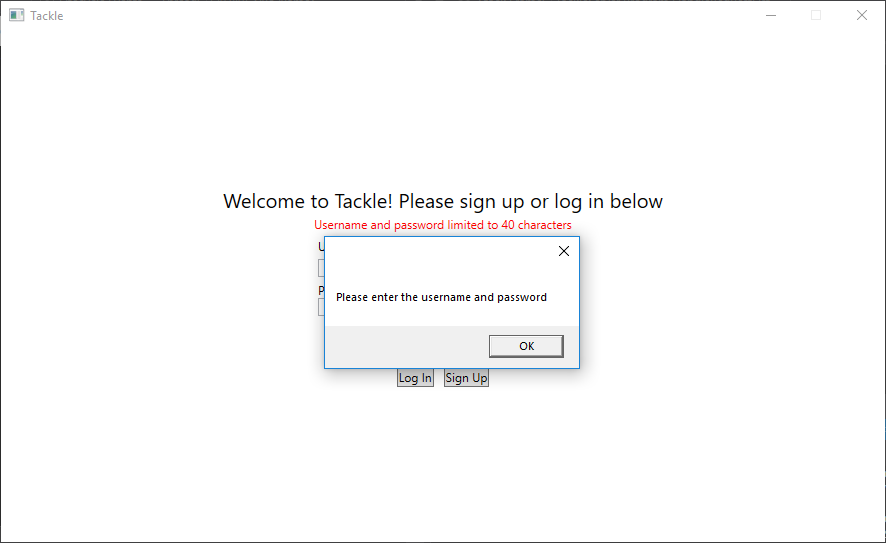
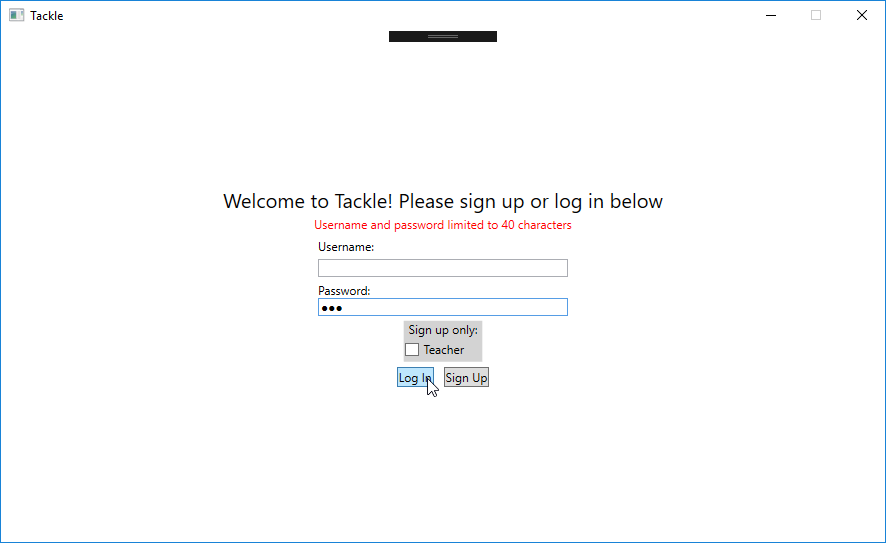


Figure 6: Log in attempt with no username or password



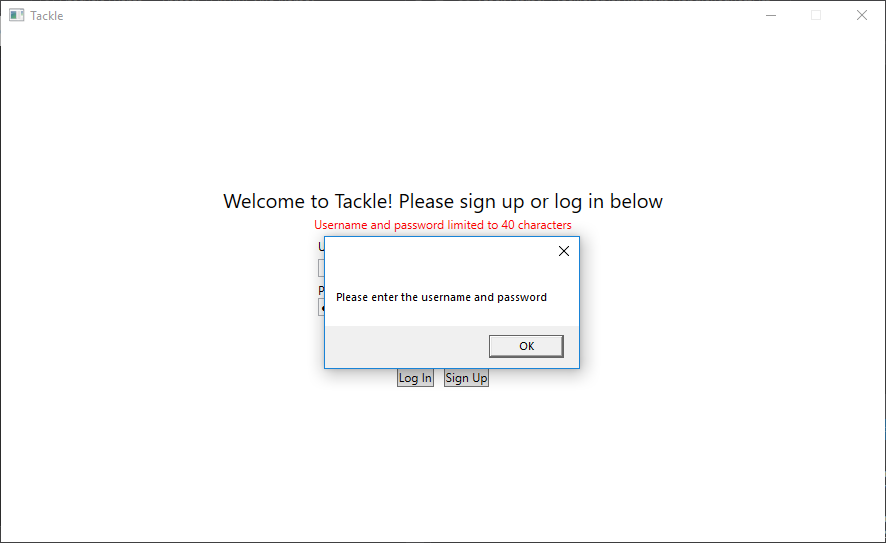


Figure 7: Log in attempt with no username



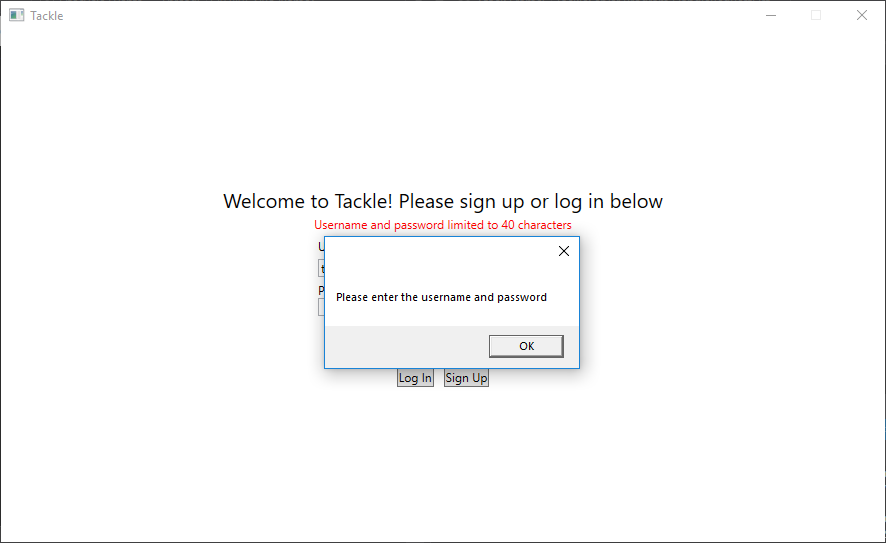


Figure 8: Log in attempt with no password

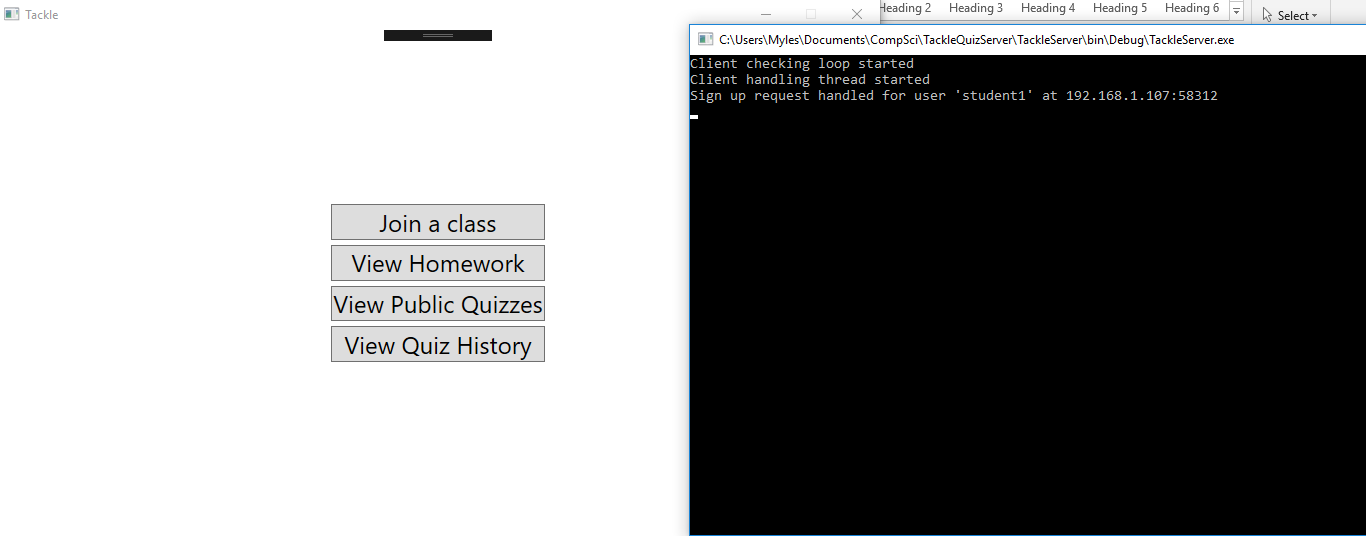
**4)**  This test works as intended

Figure 9: Server side showing the successful sign up and the resulting student main menu

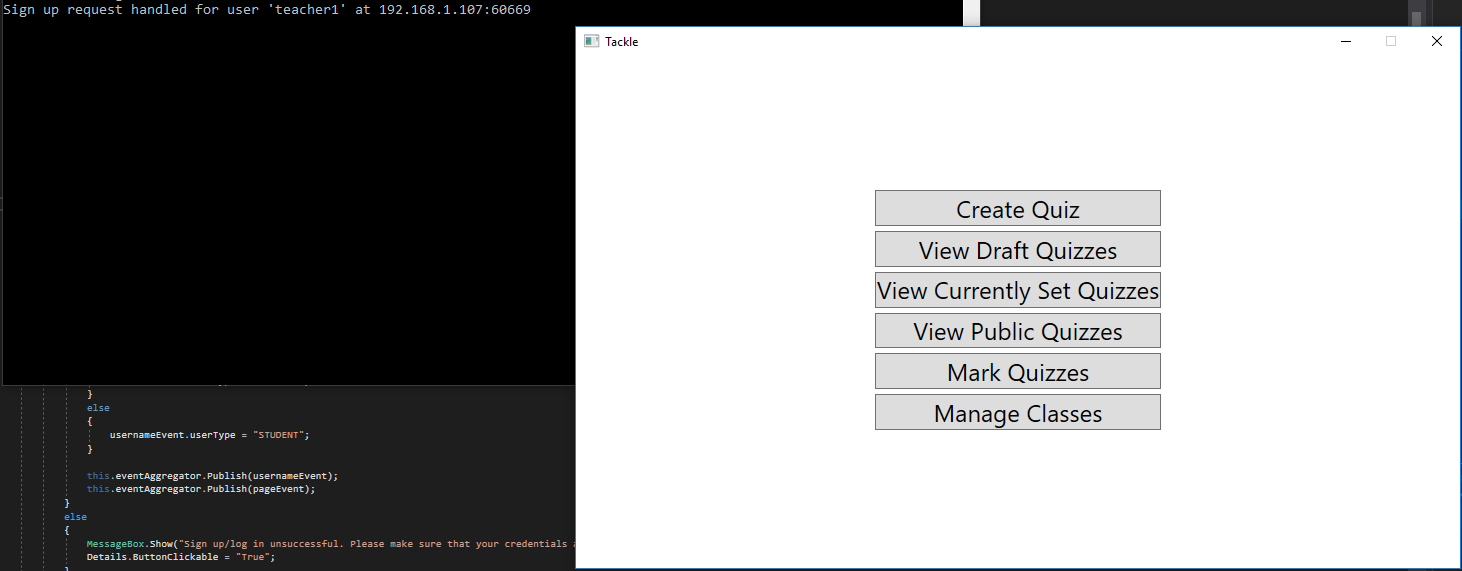
**5)** This test works as intended

Figure 10: Server side showing successful sign up, and the resulting teacher main menu

**6)** This test works exactly as expected. If there’s an attempt to sign up with an already existing username, then a message box will pop up saying that the sign up request was unsuccessful.

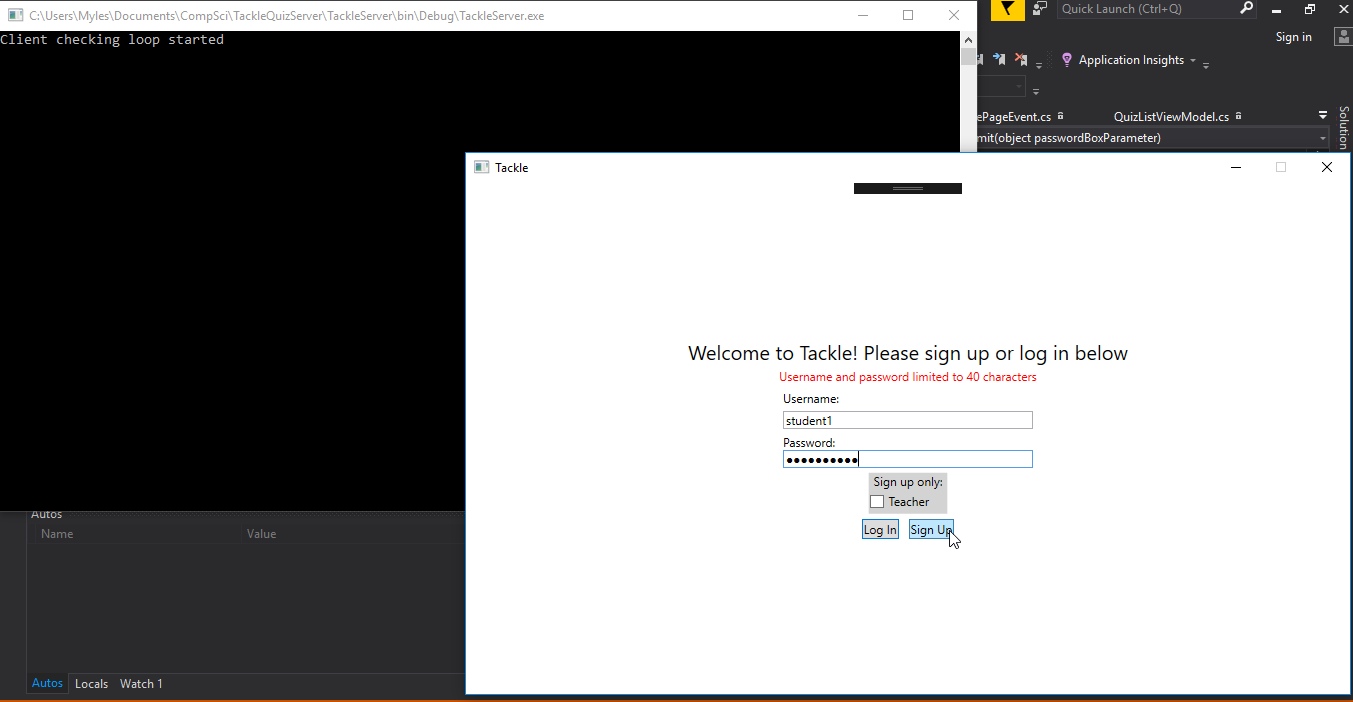


Figure 11: signing up with the pre existing username "student1"

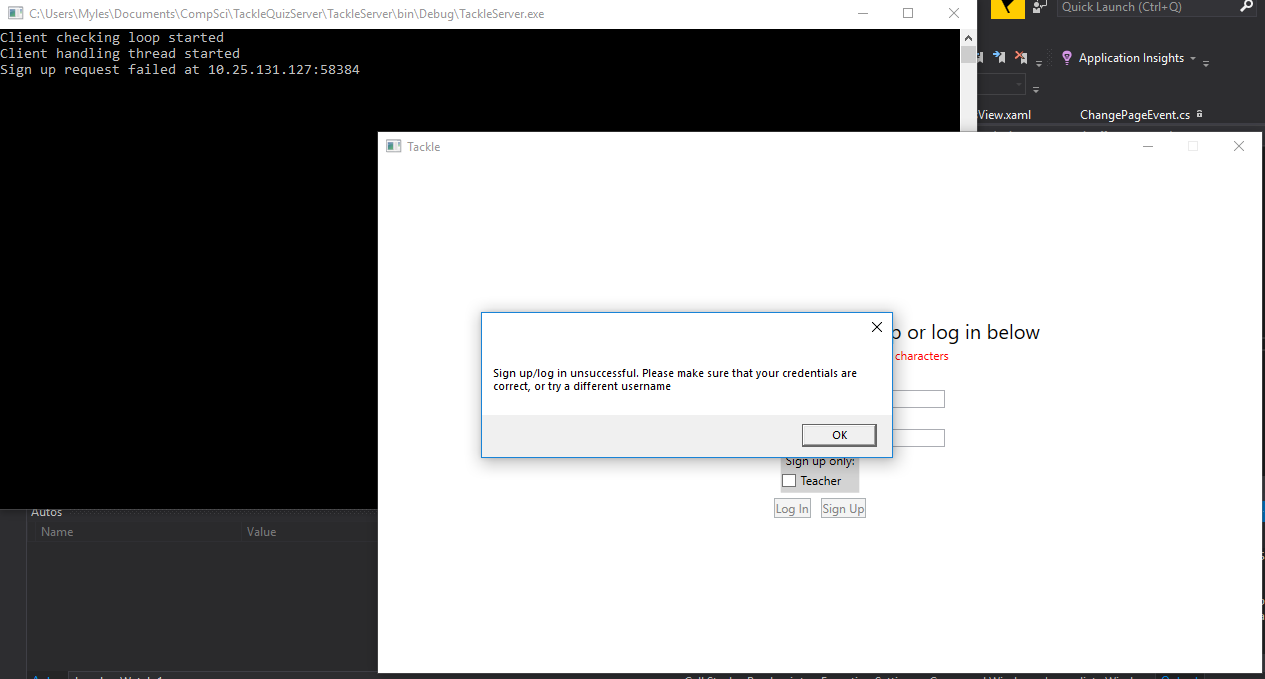


Figure 12: Message box showing up saying that the sign up request was unsuccessful, with the server side saying that the sign up request has failed

**7)** This test works as expected

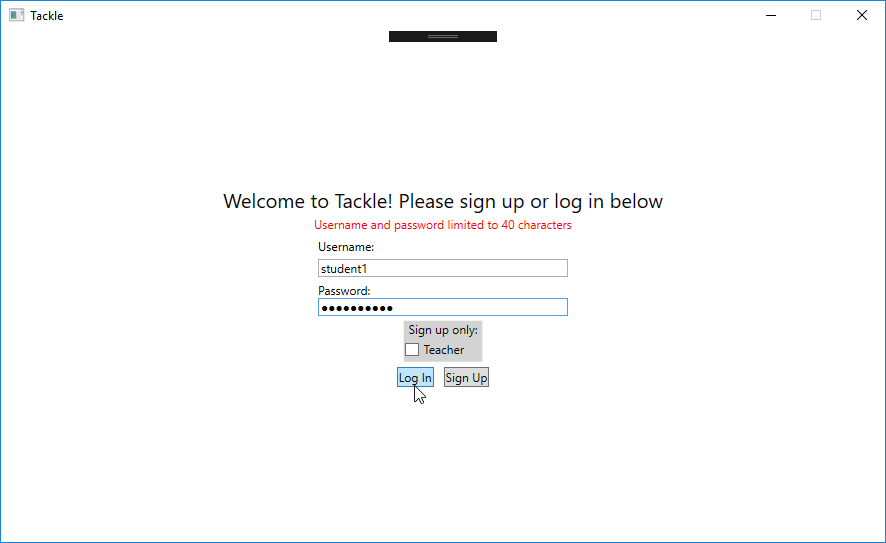


Figure 13: signing in with the correct username and password for a student

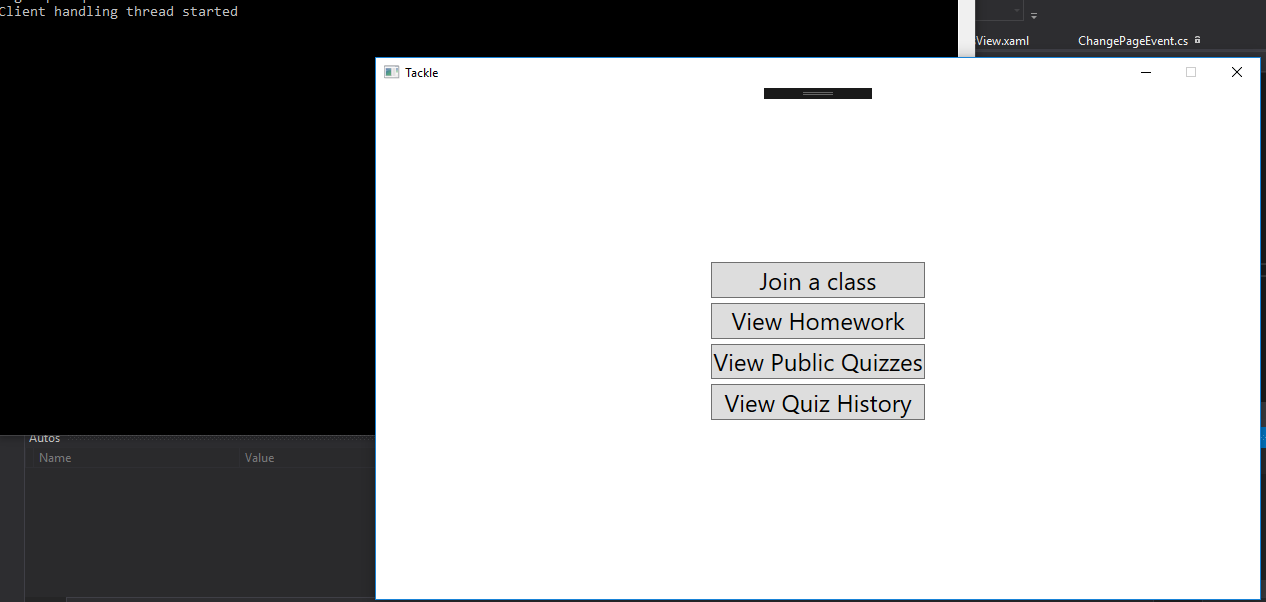


Figure 14: Redirecting to the student main menu

**8)** This test works as expected

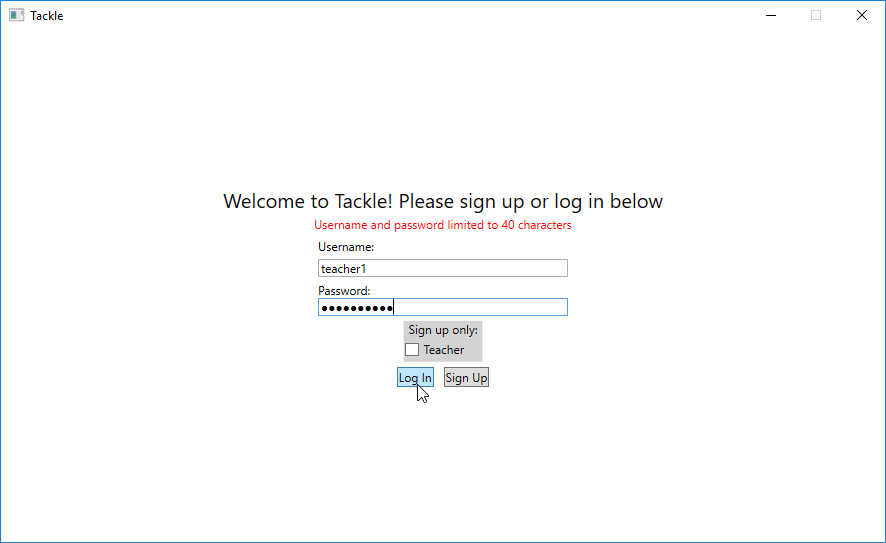


Figure 15: Signing in using the correct username and password for a teacher

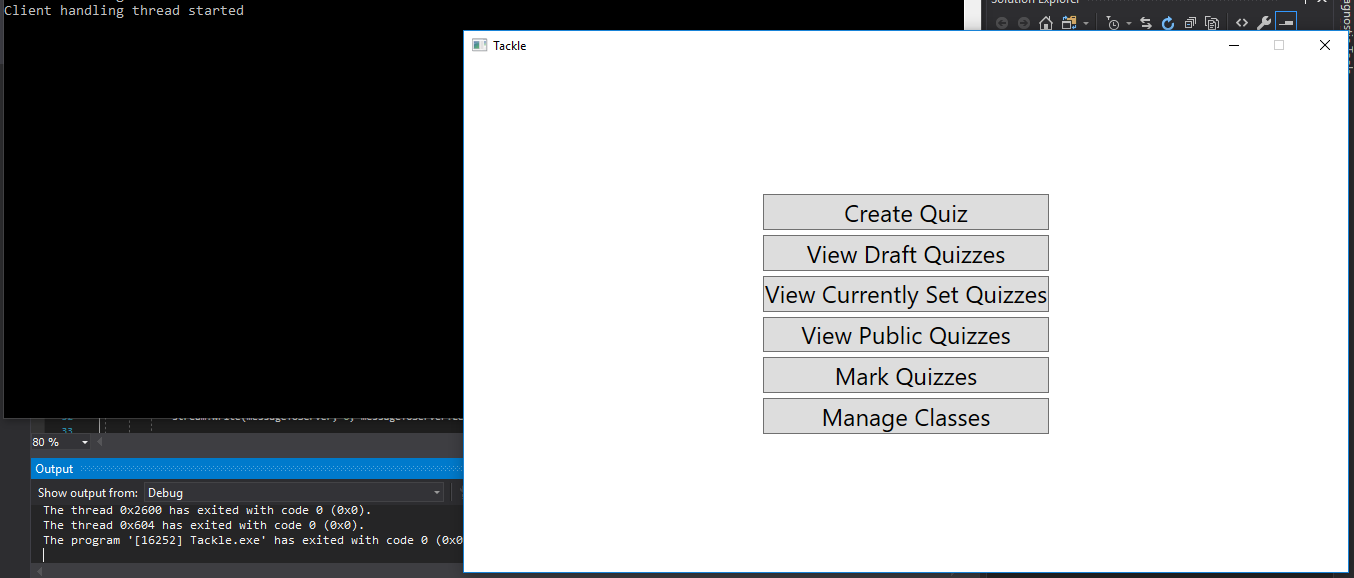
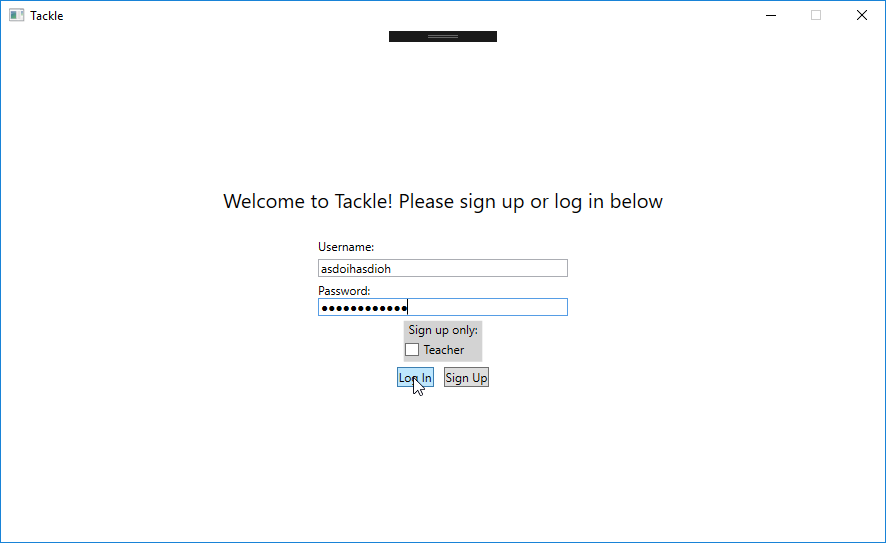
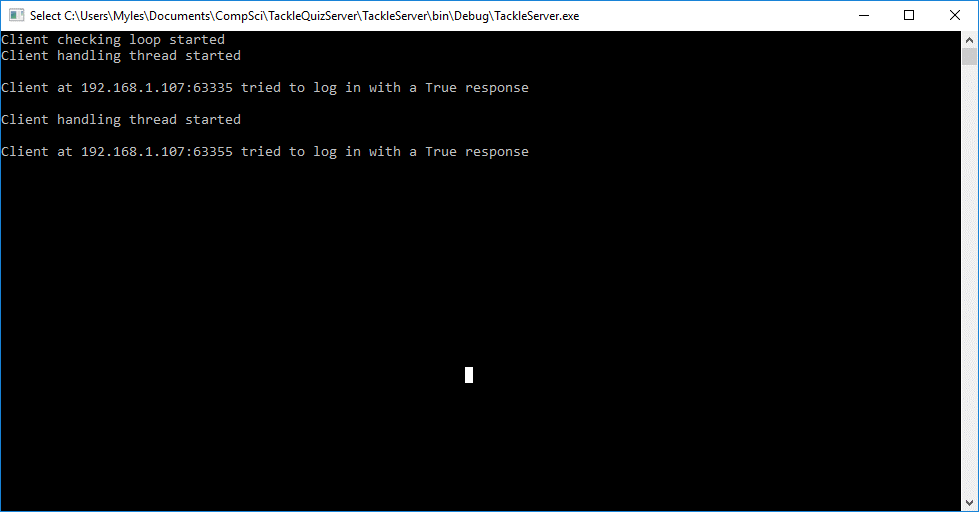


Figure 16: Successful redirect to the teacher main menu

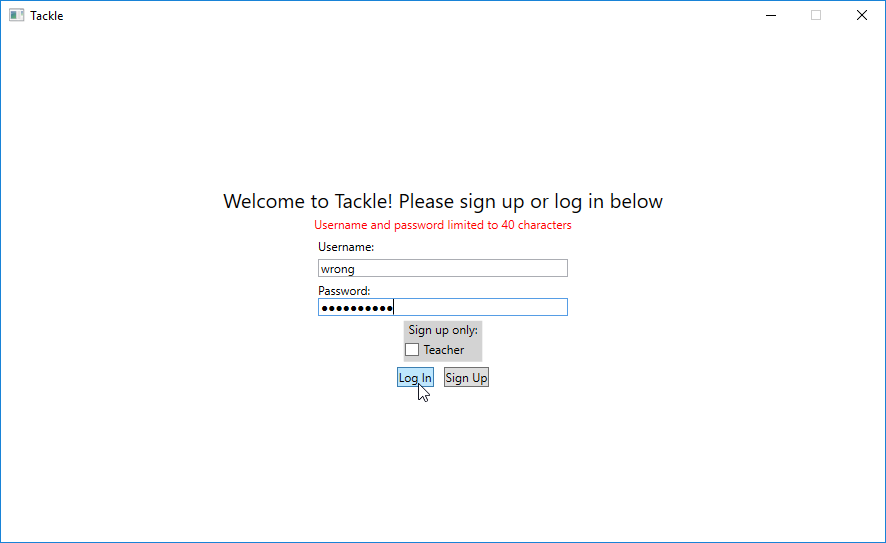
**9)** This test is fully working

24/09/18 – when an incorrect username or password was inputted, the client side crashed:





Fully working:



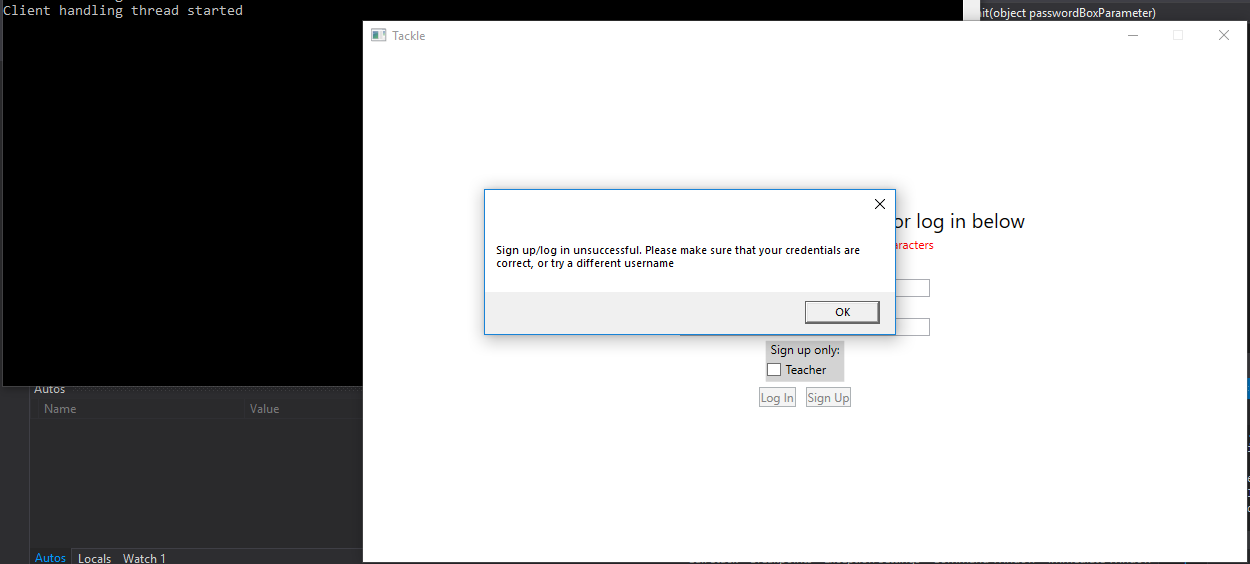
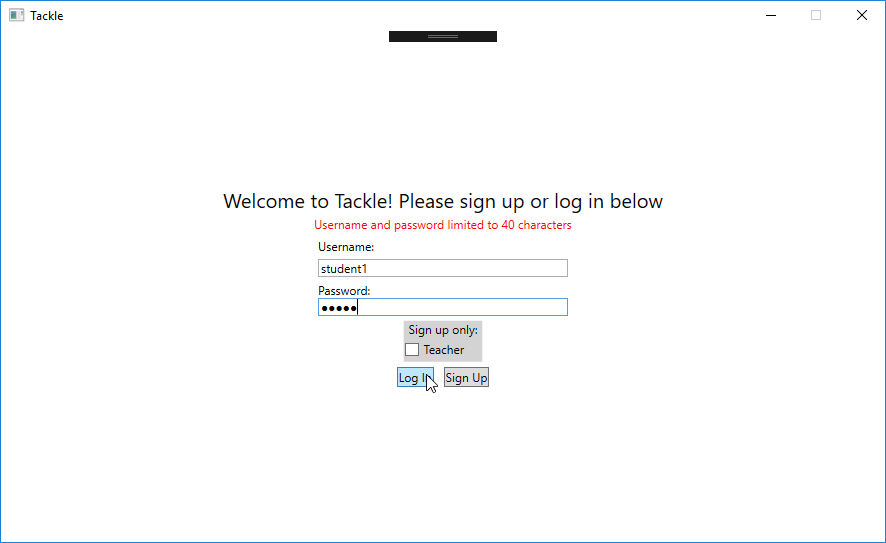


Figure 17: Message box showing that the sign in request was unsuccessful after a non existing username was put in



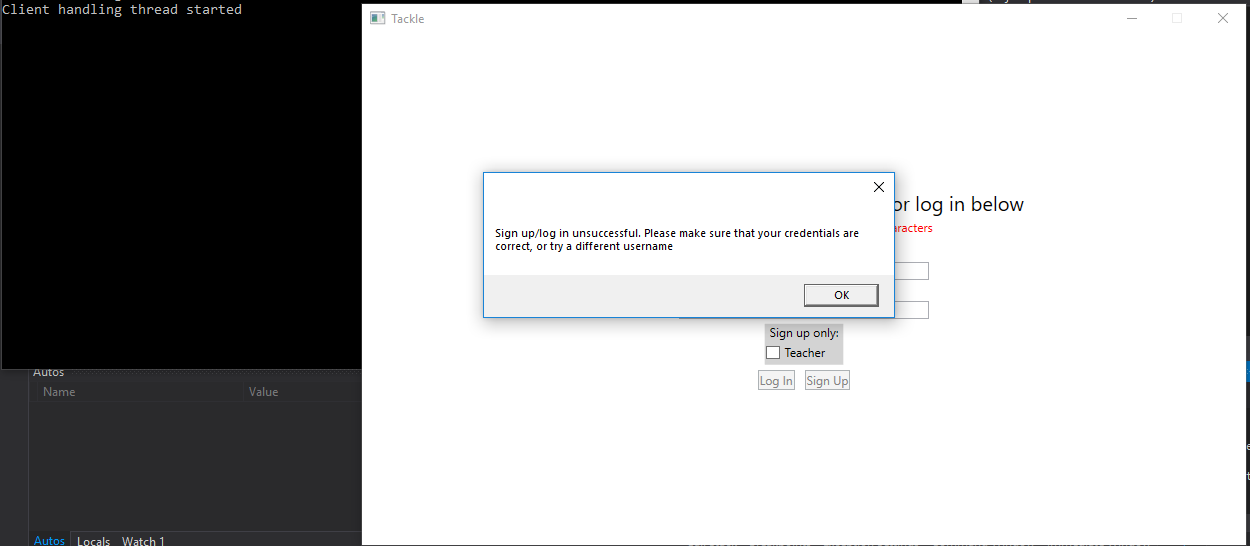


Figure 18: Message box showing that the sign in request was unsuccessful with an incorrect password

**10)**

This part of the program is fully functional, with each button leading directly to where it should go

During the process of making the program, I decided to incorporate certain parts of the program into each other. This resulted in the removing of one program feature from the main menu, which is the set quiz history. Instead of separating between quizzes that a teacher currently has set to their class and the quizzes that the teacher has sent to their classes before, this is all put into one section marked as just “View Set Quizzes”.

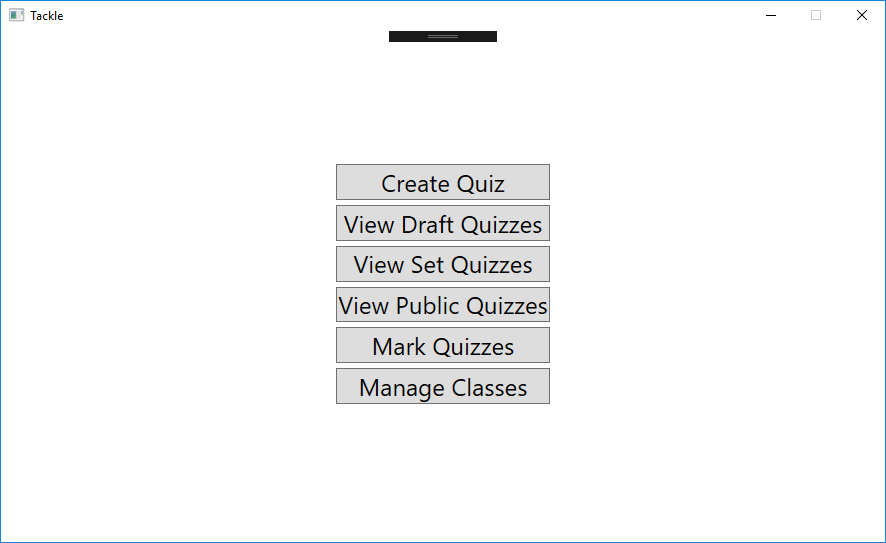


Figure 19: The new teacher main menu

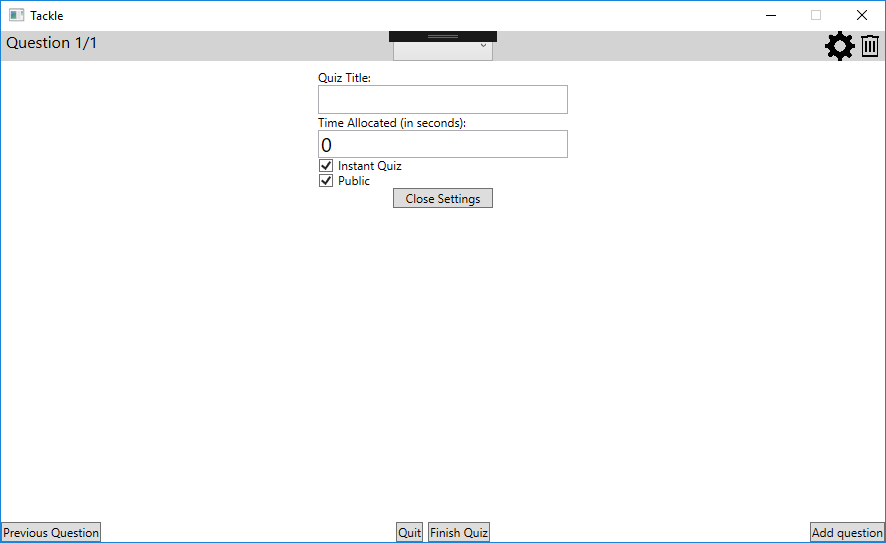


Figure 20: Create Quiz button works

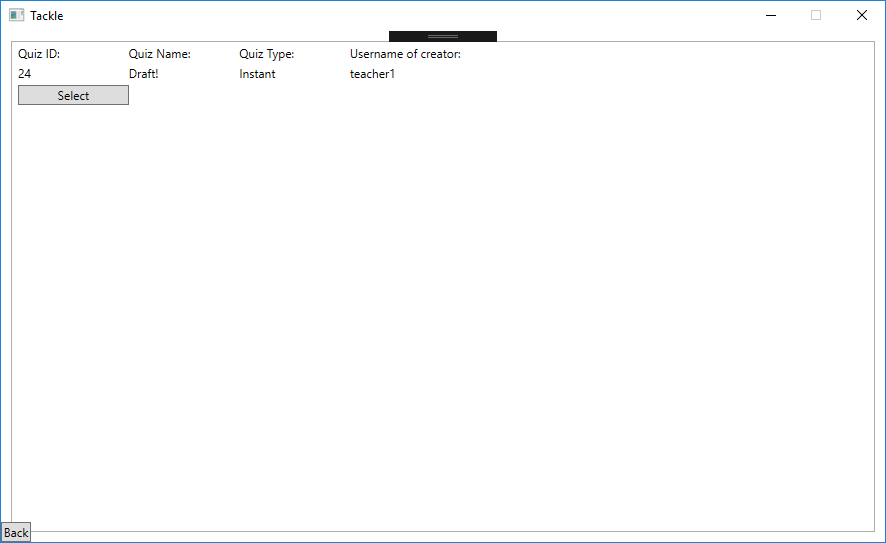


Figure 21: View draft button works

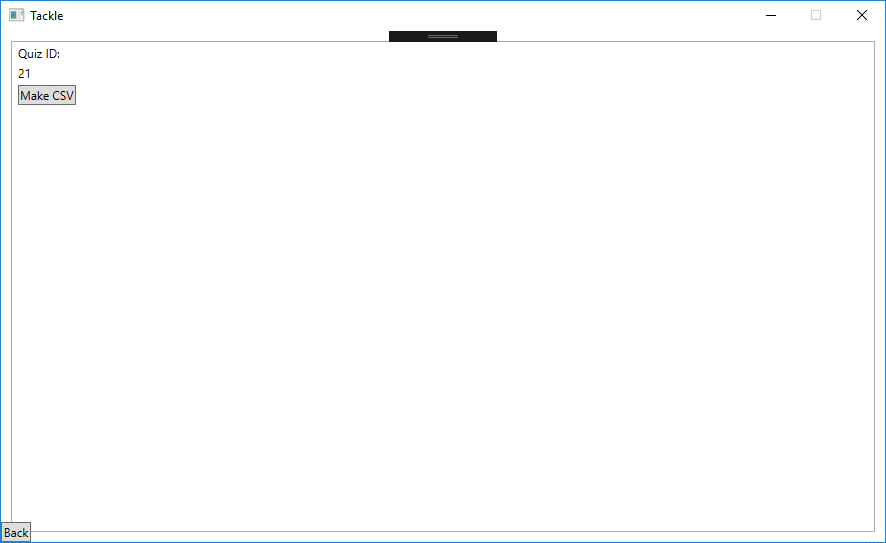


Figure 22: Viewing set quizzes works

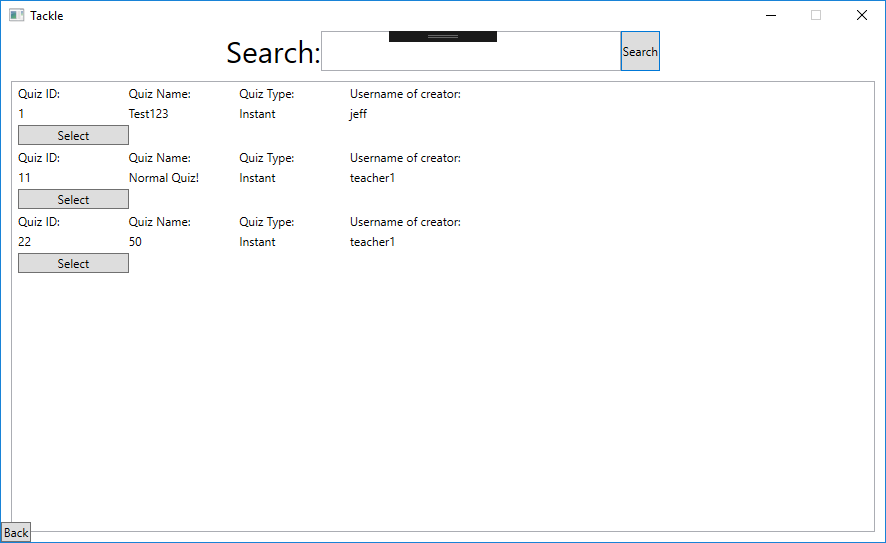


Figure 23: View public quiz button works

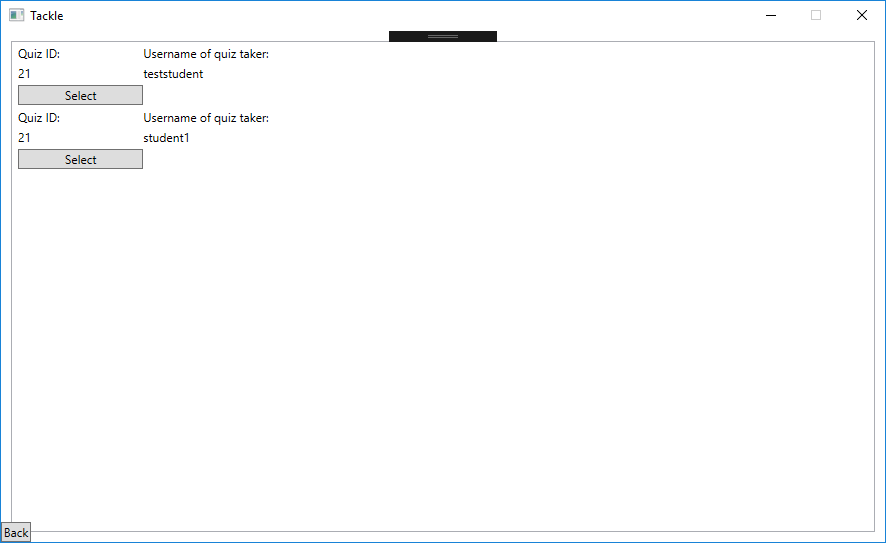


Figure 24: Mark quizzes button works

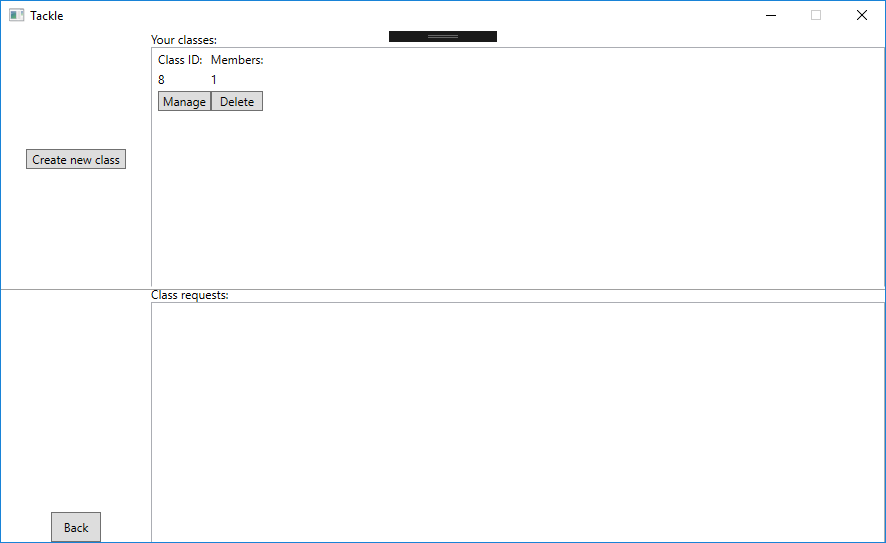


Figure 25: Manage classes button works

**11)**

This part of the program is also fully working, but also has changed to coincide with the changes made to the teacher main menu – here, once again, the “to-do” quizzes are mixed in with the already done quizzes. This change was made once again to streamline the experience of the program and prevent an unnecessarily bloated main menu.

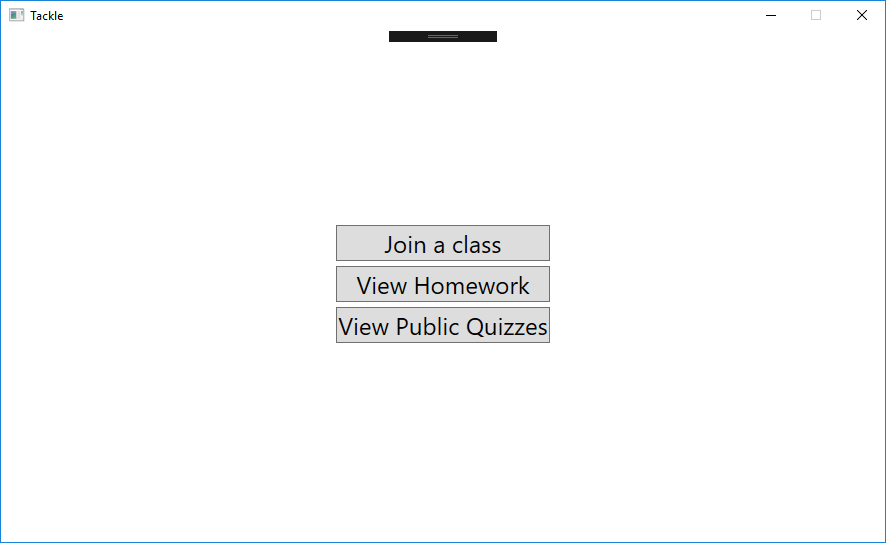


Figure 26: The new teacher main menu

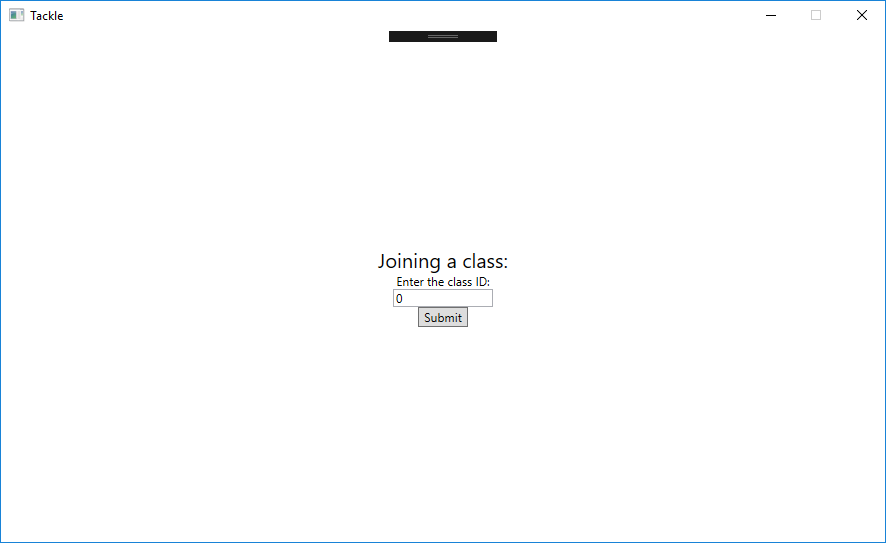


Figure 27: join a class button works

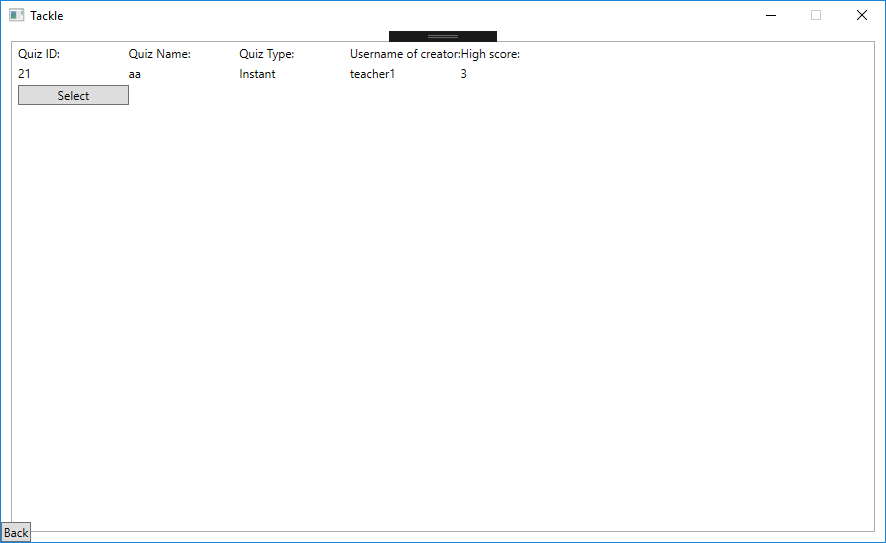


Figure 28: Viewing homework button works

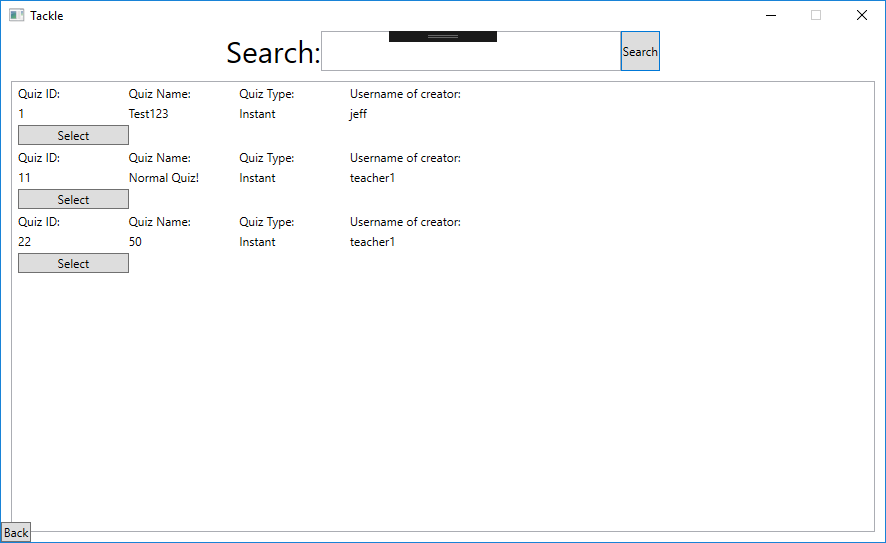
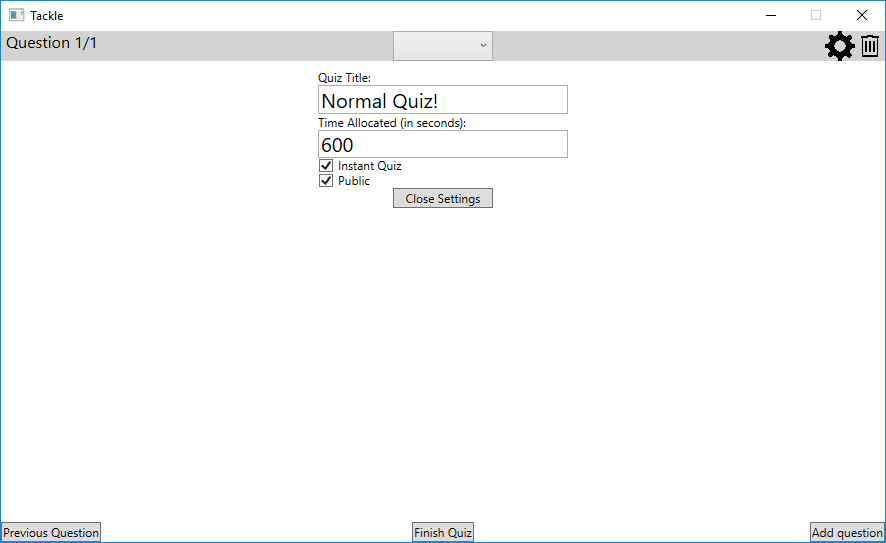
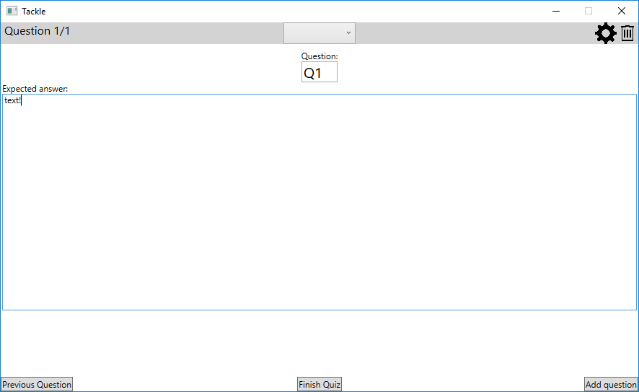
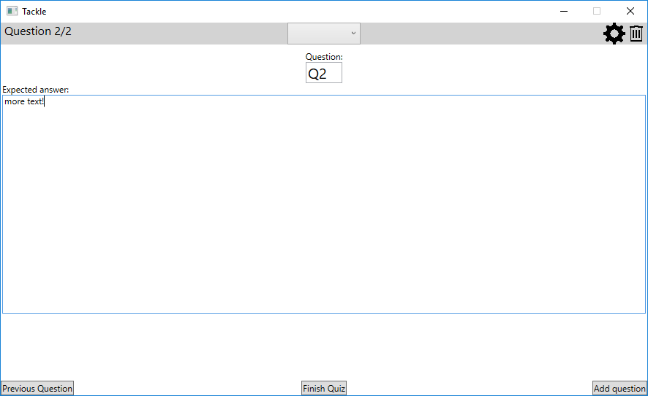


Figure 29: View public quizzes button works

**12)** Editing the quiz works exactly as expected. I decided to implement a settings screen to fulfil the role of choosing the type of quiz, time allocation and the quiz title which is displayed whenever the use decide to edit or create a quiz.



Figure 30: Settings screen displayed after choosing to edit/create a quiz



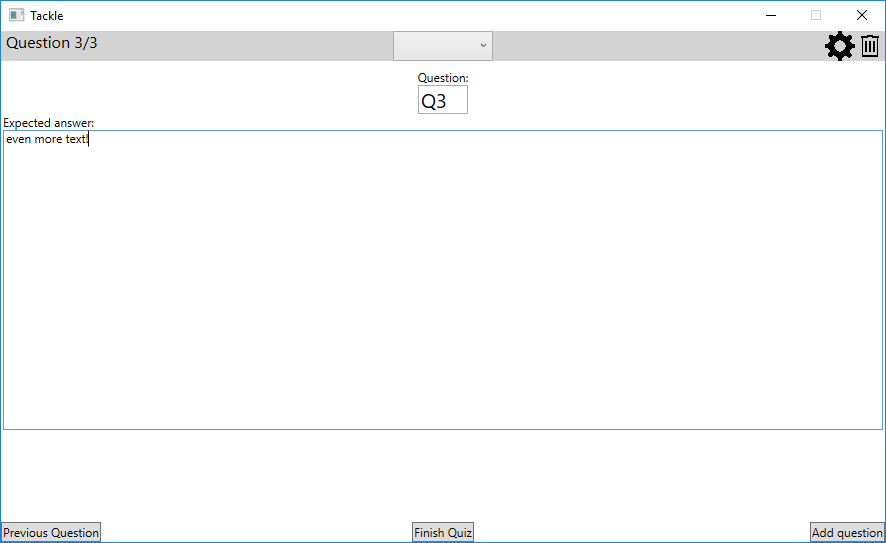
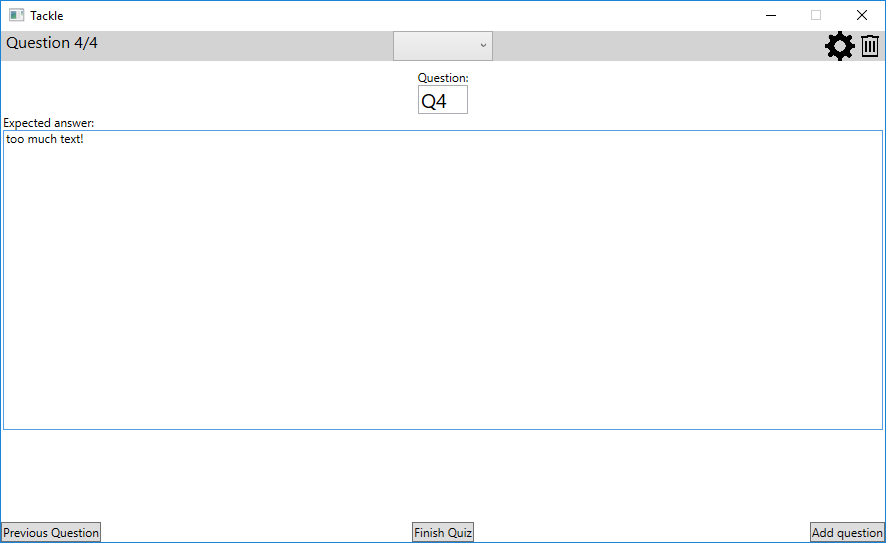
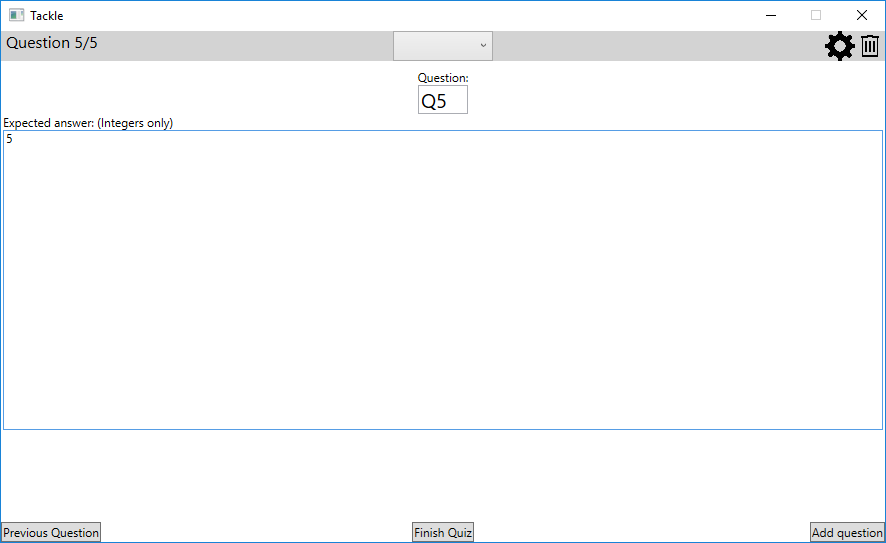
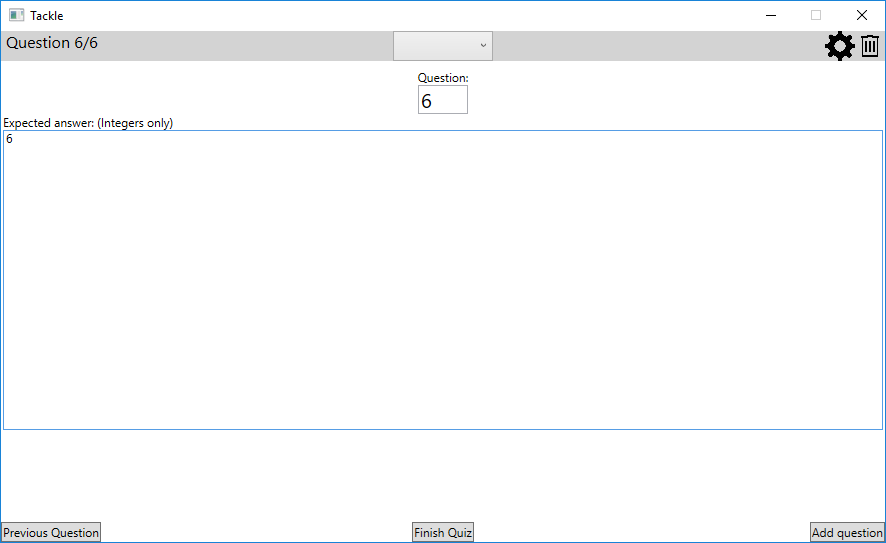
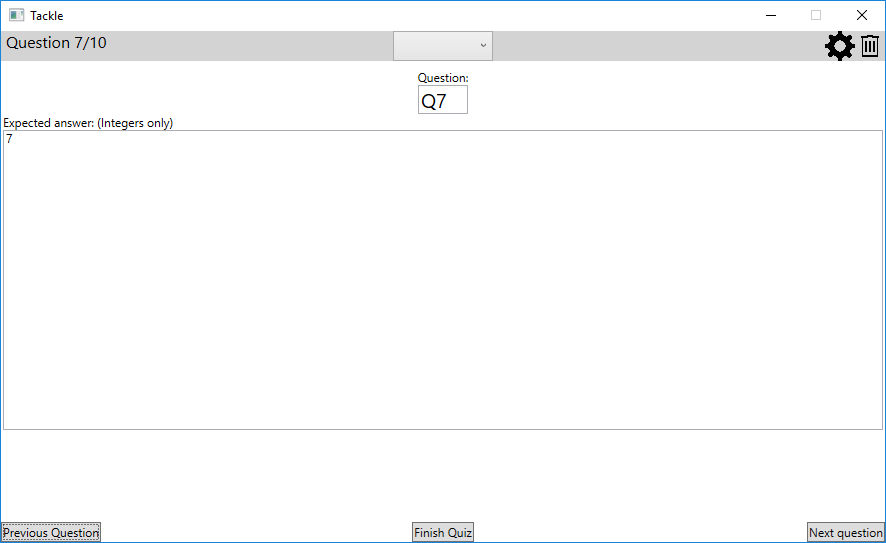
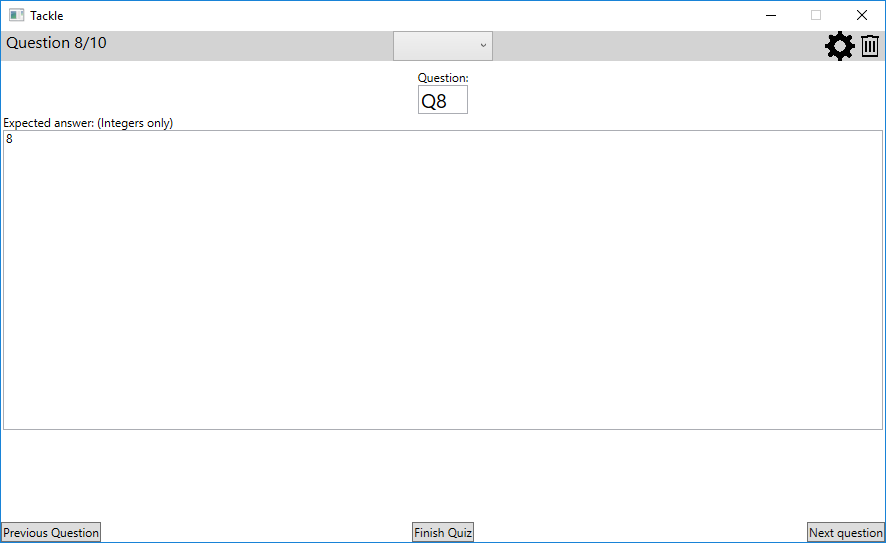
 

Figure 31: String input questions

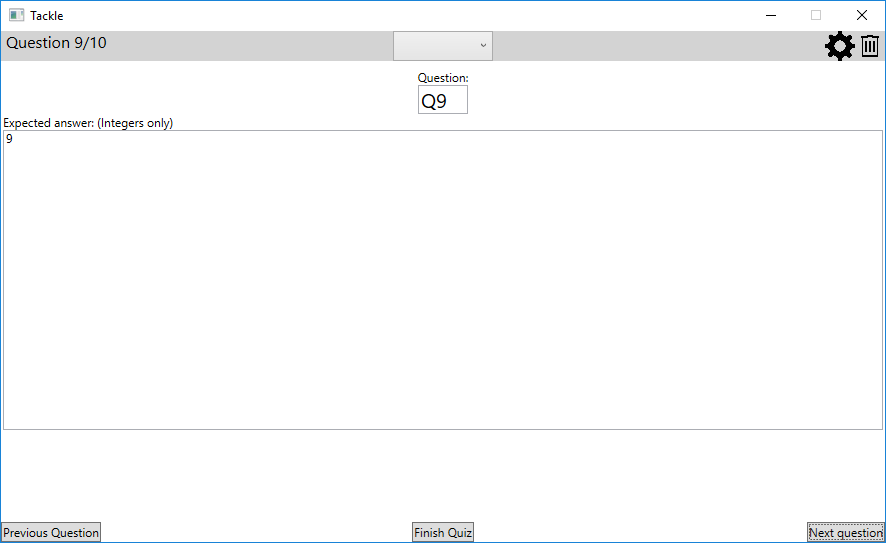


Figure 32: Integer input questions

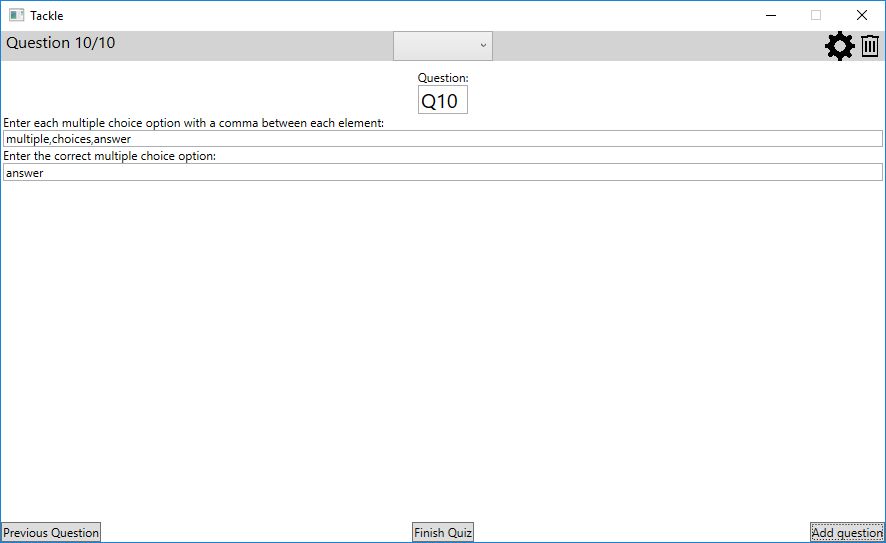
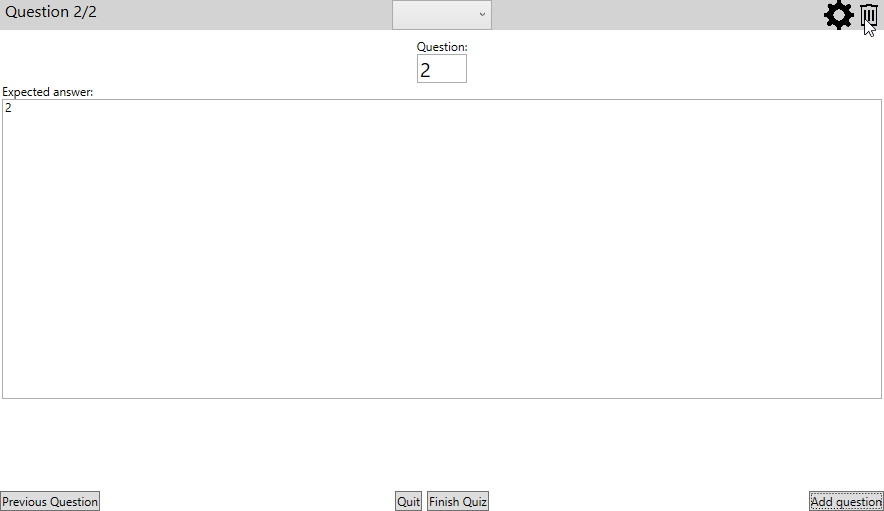
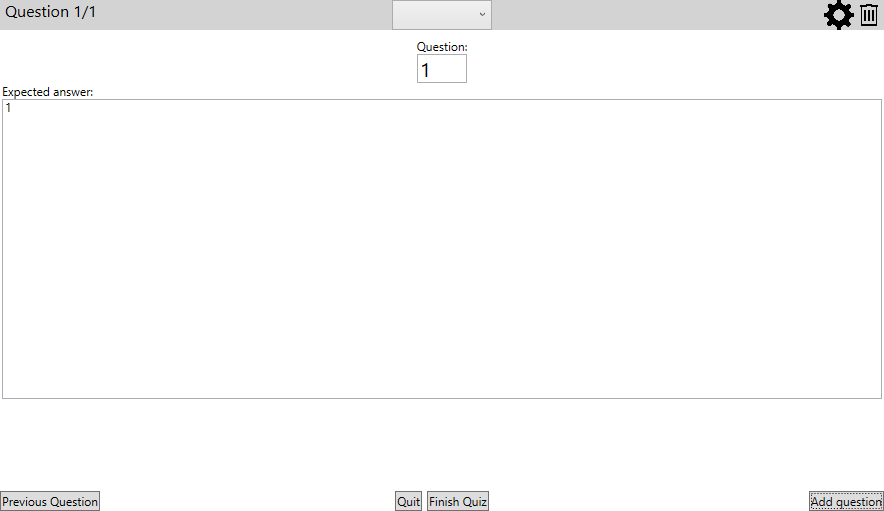


Figure 33: Multiple choice question

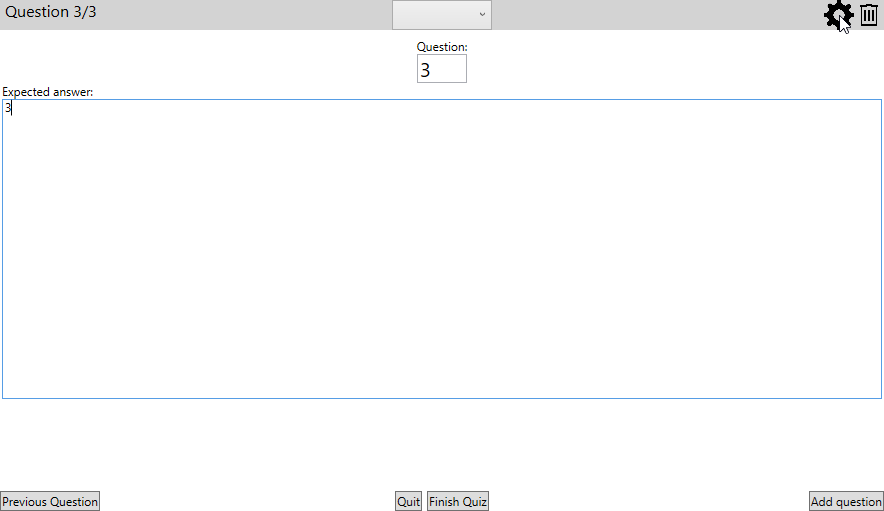
While making the quiz, the teacher is able to delete a question at any time by just clicking the delete button. For example:

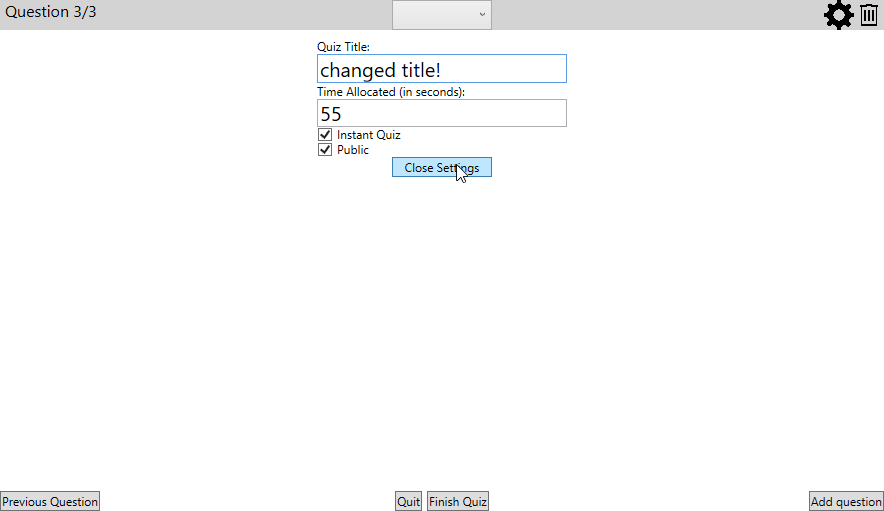




This was implemented as it’s entirely possible for a teacher to accidently add a question, and so they should be able to delete it. The only exception to this is if there is only one question, in which case the question won’t be deleted.

The teacher is also able to access the settings at any time while creating the quiz:





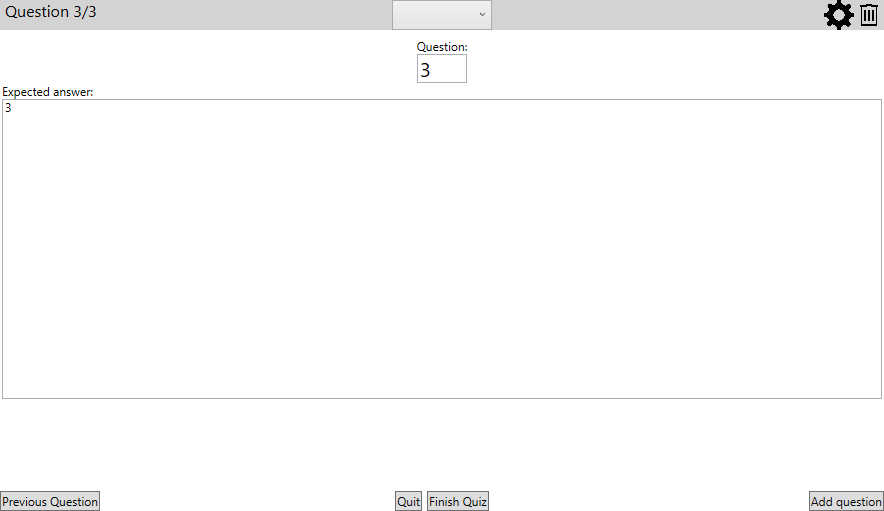
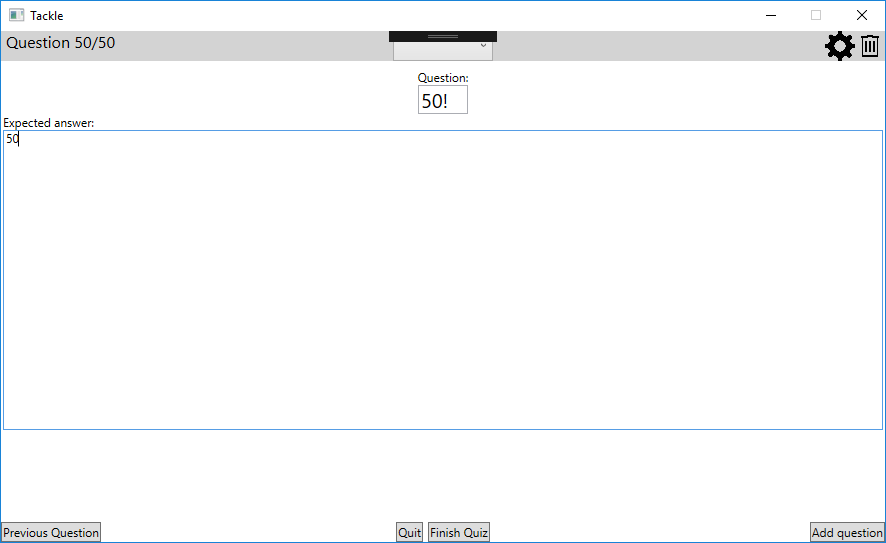
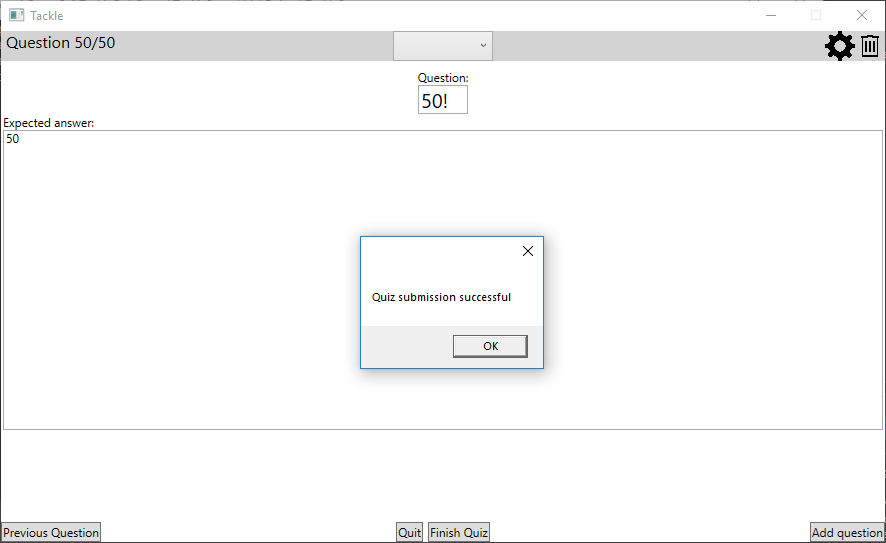


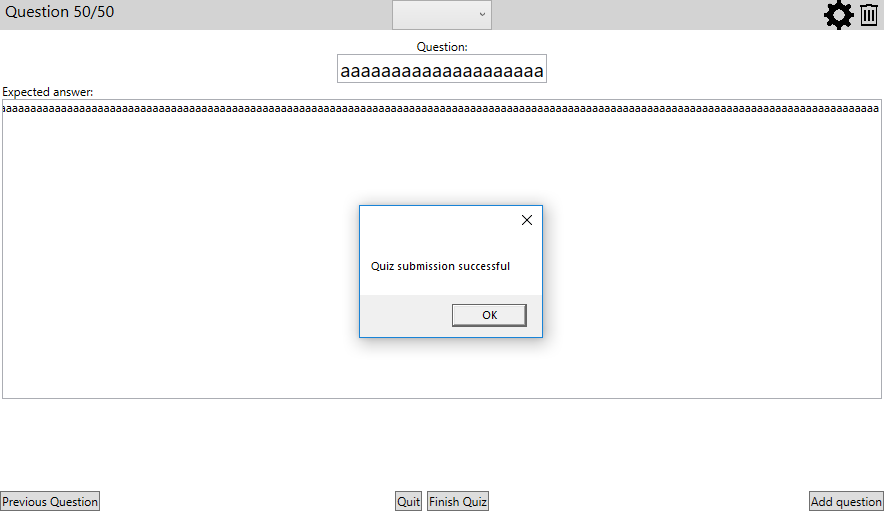
Figure 34: Opening and closing of the settings page

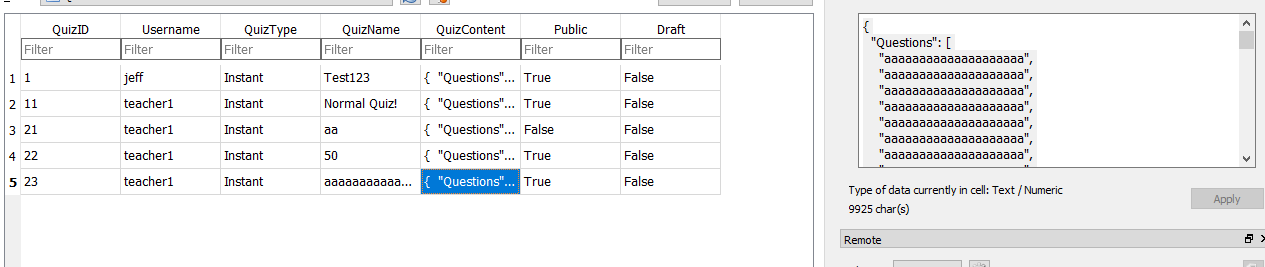
**13)** This test works as expected

As a result of this test, I decided to limit the amount of questions when creating a quiz to 50. This is because this amount is more than most teachers will need, and with an unlimited amount of questions there will be issues (as previously mentioned) with reaching the maximum size of the buffer, which may result in quizzes not being submitted properly to the database, and also could possibly result in a buffer overflow attack.





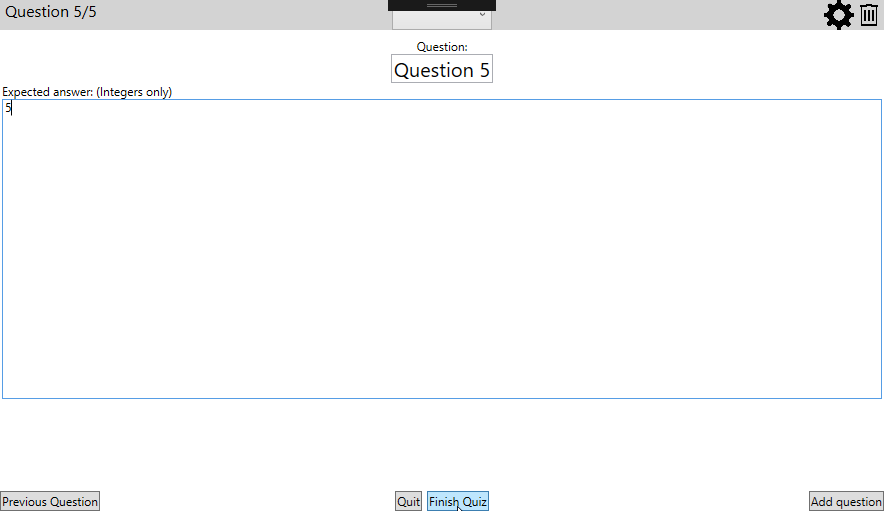
The quiz submits to the server perfectly fine. This is due to the extremely large byte buffer that the server side has of 50,000 bytes which may be overkill but ensures a complete lack of any buffer overflows. If the JSON string were to be too large for the server, then the server side would simply crash due to the JSON string being cut off unexpectedly, and so the serialisation would throw an error. This is also why I implemented a limit of 20 characters for the question and 150 for the answer, which works perfectly fine when submitted for every question as seen below:



**14)**

Between the design period and testing period, I decided to move from storing quizzes as a .zip file to using JSON for storing the quizzes in every scenario. This was done to make the program more consistent, as the quiz attempts themselves are stored as JSON strings, and so it would make little sense to store just the quiz information when taking and saving the quiz as .txt files stored in a .zip file. This quiz content JSON is stored in the Quizzes table under the column “QuizContent”. I also decided against creating a new table for each user to store the quizzes they have to do. This is because when a student joins a class, it would be much more efficient to assign the quizzes that they have to do to a class, and then just retrieve the quiz IDs that are relevant to the user. This is incorporated into the table “SetQuizzes”, which has the columns “QuizID” and “ClassID” to store which quiz the entry is set to, and the class that it’s being sent to. This stops unnecessarily creating tables, which would make database management much more difficult, and also is more space efficient due to the lack of repetition of entries (a key advantage of normalising a database)

For this test, I decided to only give the option of sending a class whenever the teacher chooses to not set the quiz as public. This is because the teacher is able to send the quiz to the class as a public quiz by going into the public quiz section, and so having this at the end of the quiz would be unnecessary. The option to send to a class is done at the end of a quiz, where the teacher is asked to type in the class ID of the class that they wish to send the quiz to.



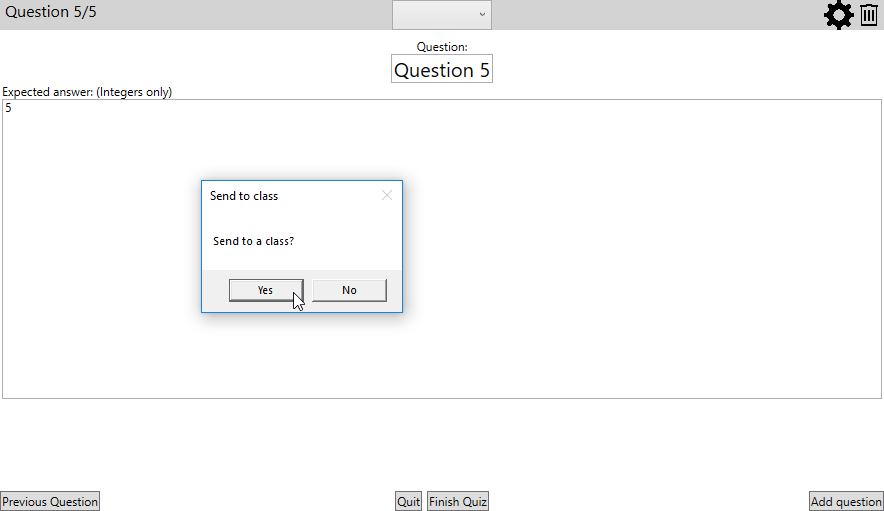


Figure 35: Option to send a quiz after completing a non-public quiz

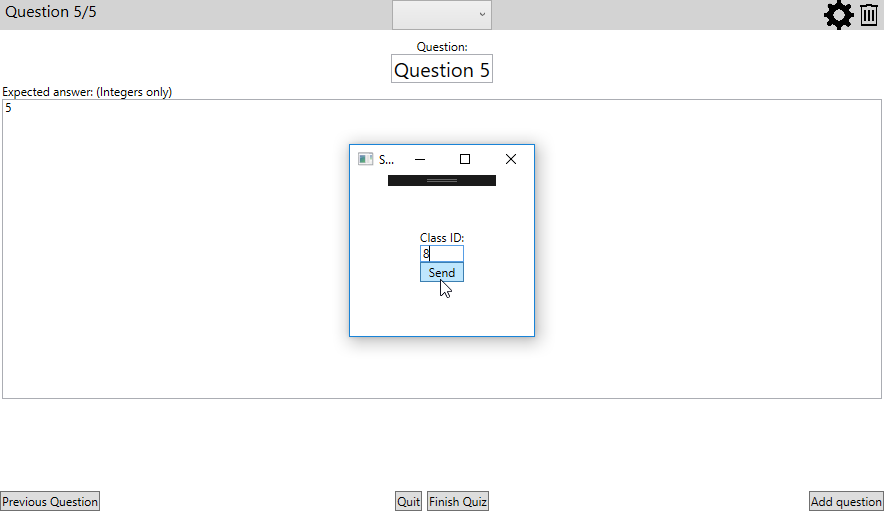


Figure 36: Typing in the class ID

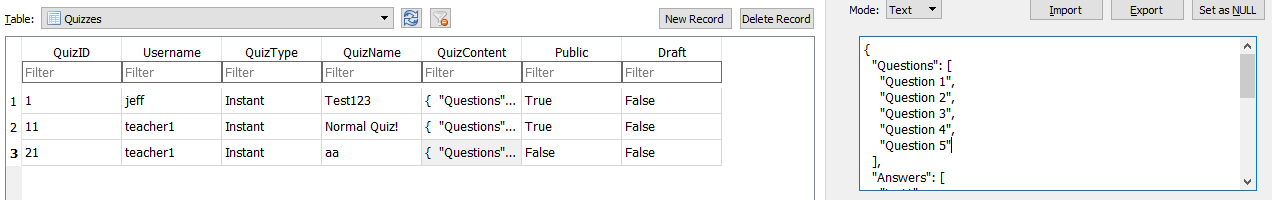


Figure 37: The record in the "Quizzes" table showing the quiz as non-public, and also not a draft

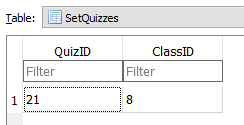


Figure 38: The quiz that was just made being shown in the SetQuizzes table

**15)** This test works as expected

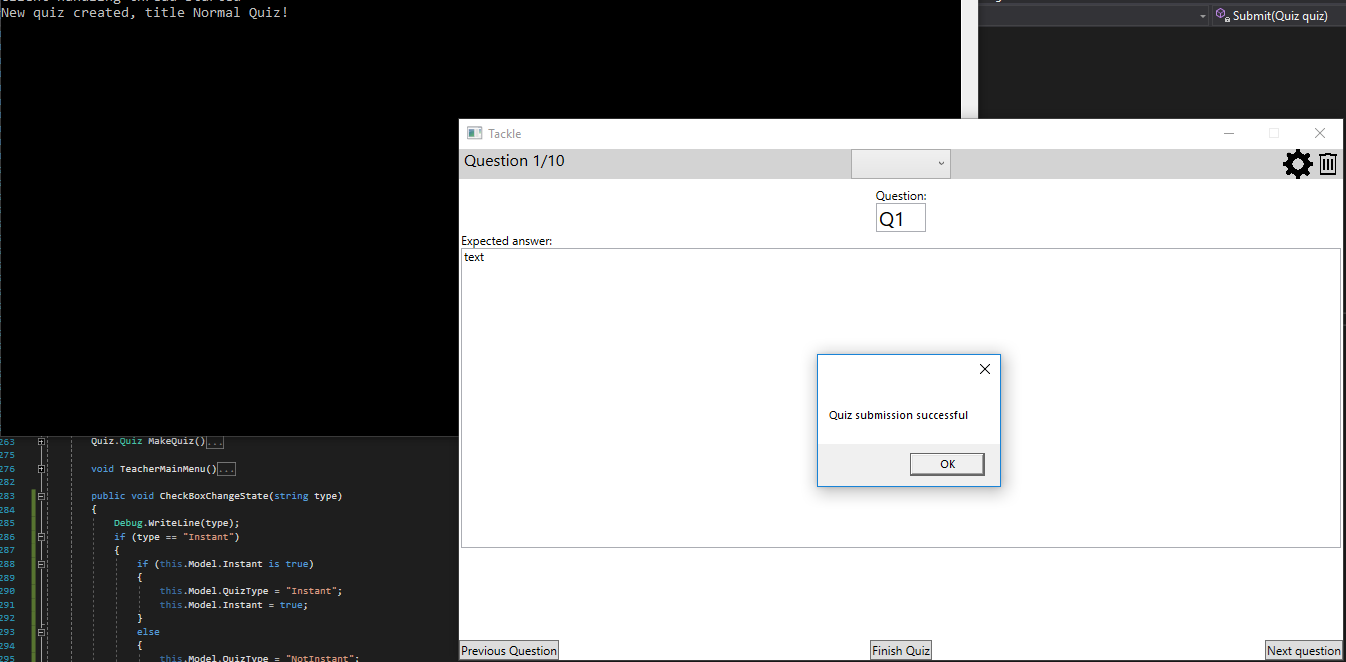


Figure 39: Teacher clicking "Finish Quiz" on a public quiz, with a message box saying that the quiz has been submitted successfully



Figure 40: The quiz submission shown in the Quizzes table, marked as public

**16)** This also works as expected

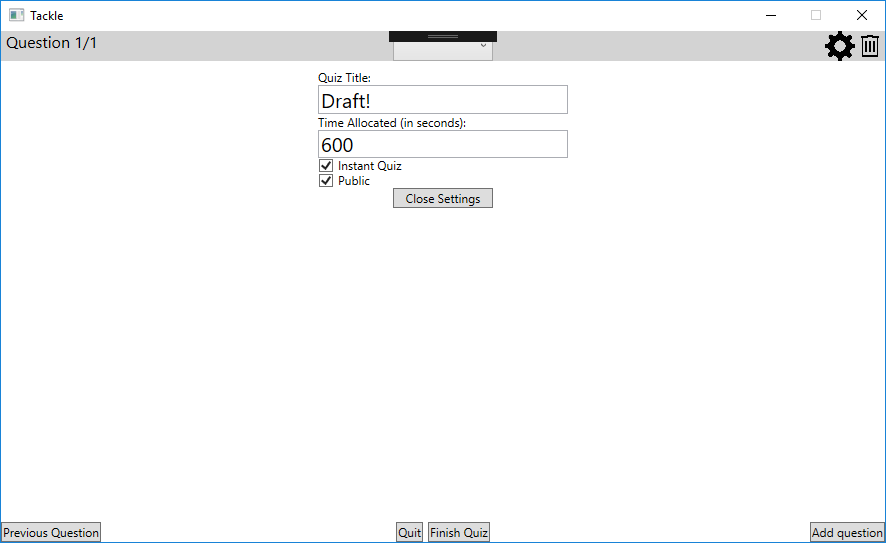


Figure 41: Creation of the to-be draft quiz

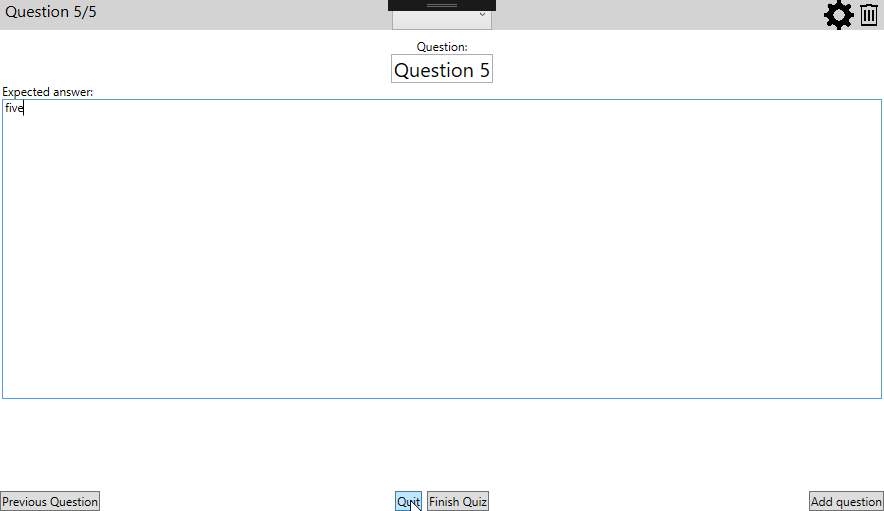


Figure 42: Teacher pressing "Quit" after typing in the questions

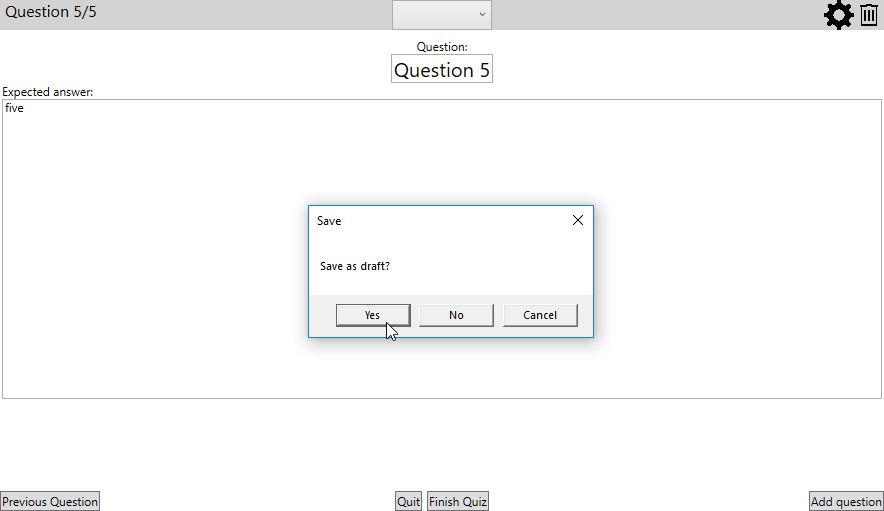


Figure 43: Dialog box asking if the teacher wants to save the quiz as a draft

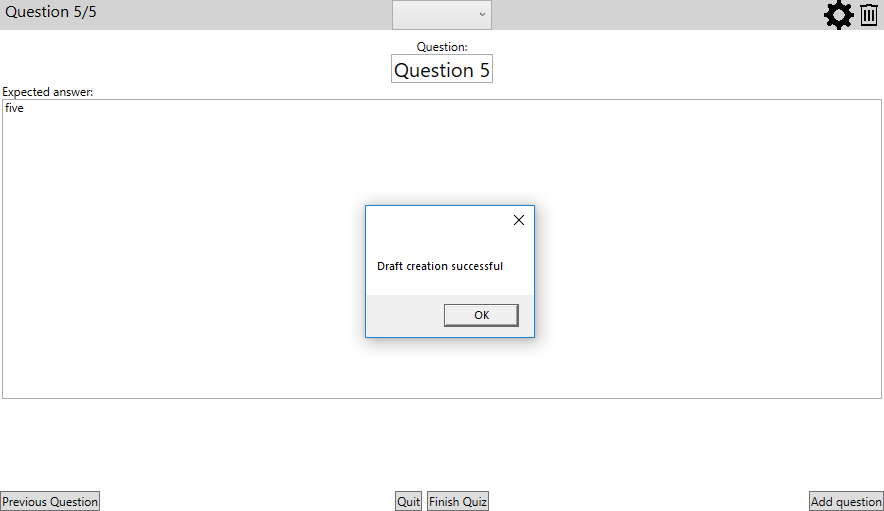


Figure 44: Message box confirming that the draft has been successfully submitted

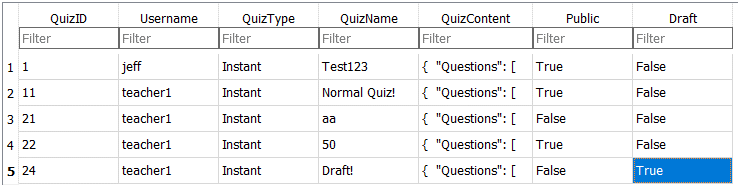
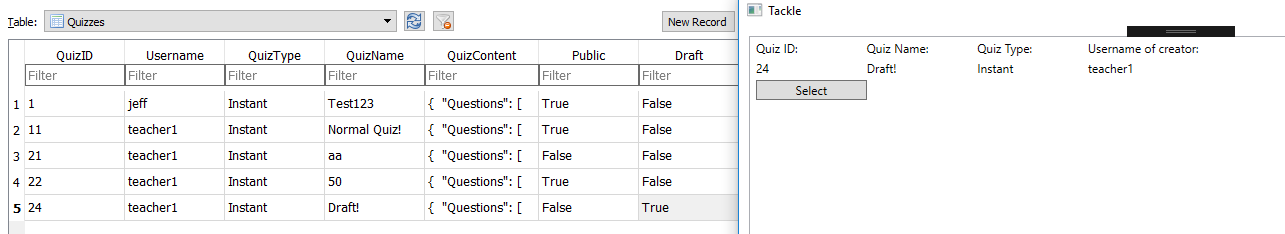


Figure 45: The draft quiz showing in the Quizzes table, with the Draft column being "True"

**17)** This test works as expected, with the draft quiz made in the previous test showing, and no others.



**18)** This test works perfectly, as when the teacher selects the draft quiz, they are given a dialog box and given the choice to edit the quiz. This dialog box was implemented for whenever a teacher selects a quiz in their drafts or in the public quiz interface, and gives them the options to do whatever they want to that quiz all from one dialog box. This dialog box is not displayed for students, as they are just expected to take the quiz.

Once the teacher has selected to edit the draft quiz, they are taken the quiz editing interface to edit the quiz as normal.

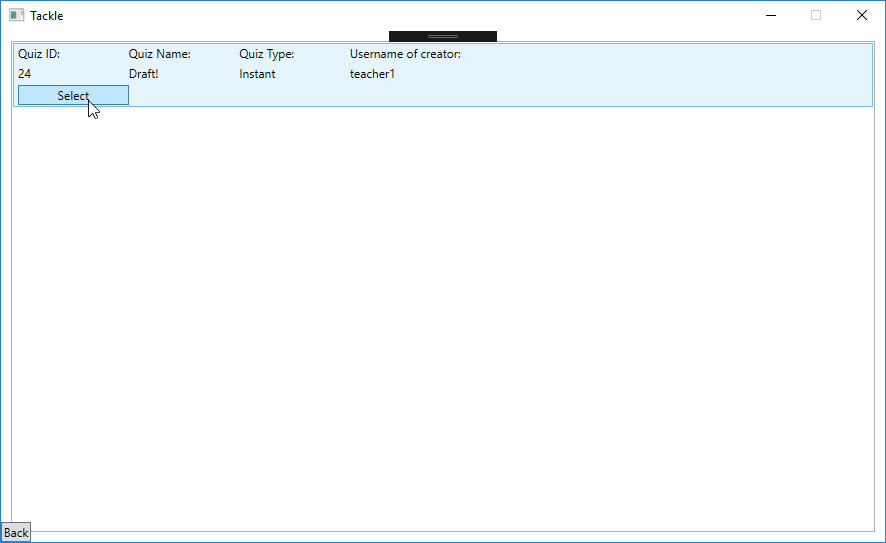


Figure 46: Selecting the draft quiz

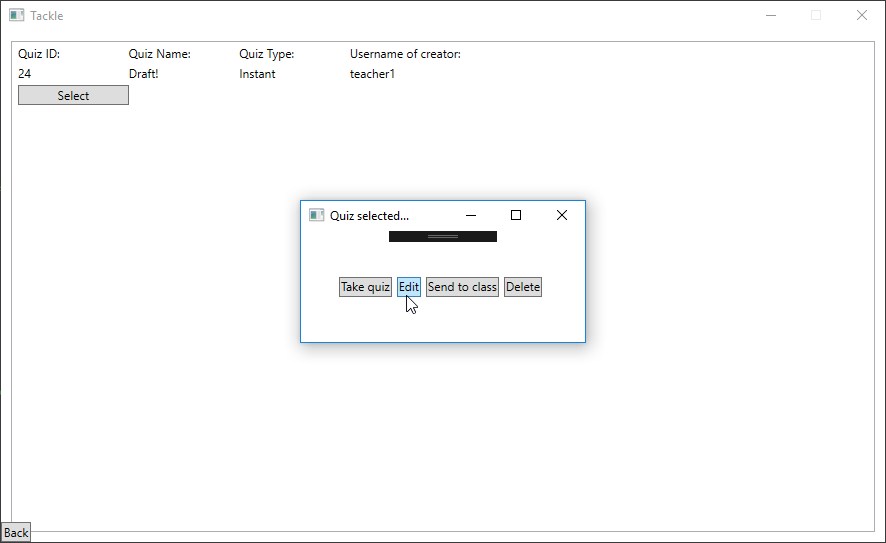


Figure 47: Selecting "Edit"

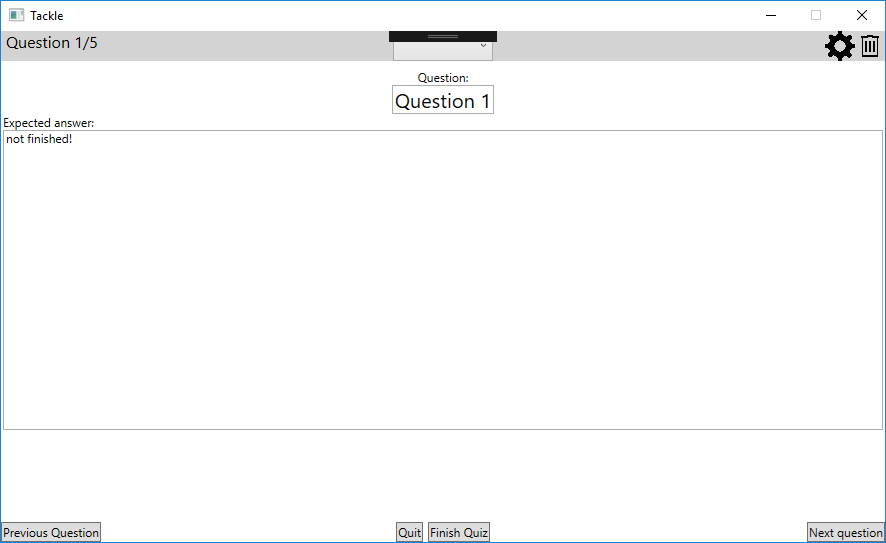


Figure 48: Editing the draft quiz

**19)** This test also works perfectly, and uses the same dialog box interface mentioned in test 18. This time, the teacher is able to click “Delete” to delete the draft quiz from their drafts.

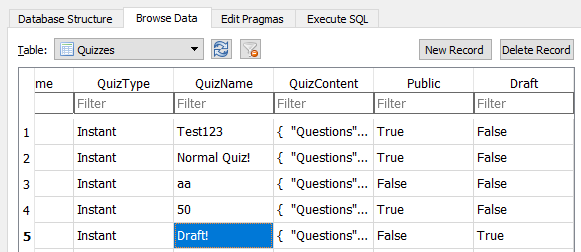


Figure 49: The draft quiz in the database

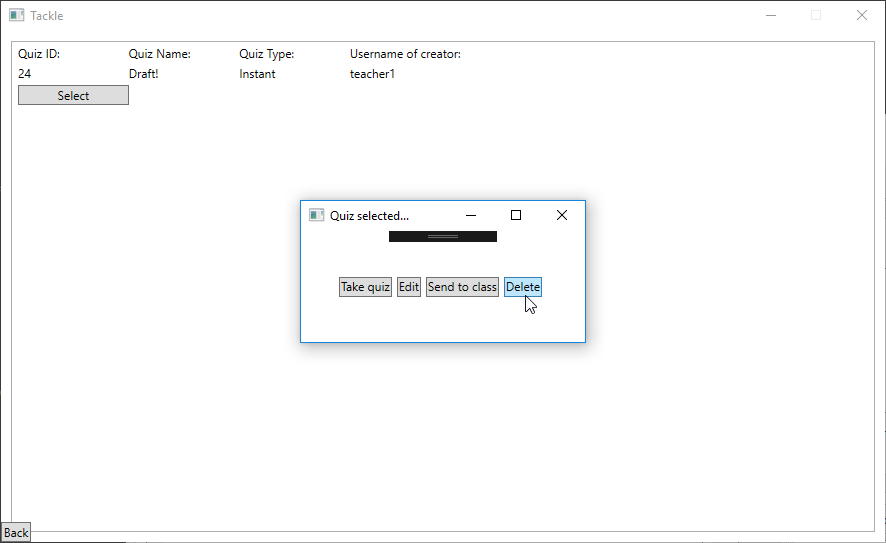


Figure 50: Pressing the delete button

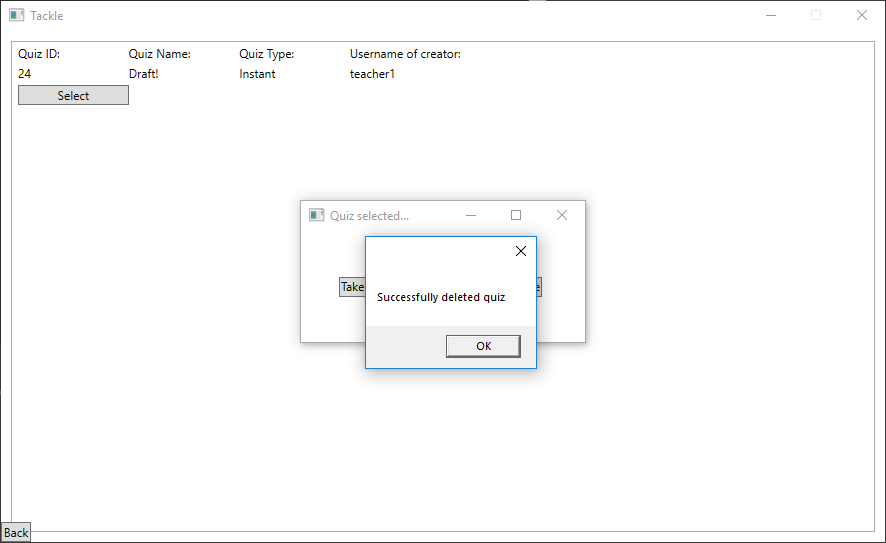


Figure 51: Message saying that the quiz has been deleted

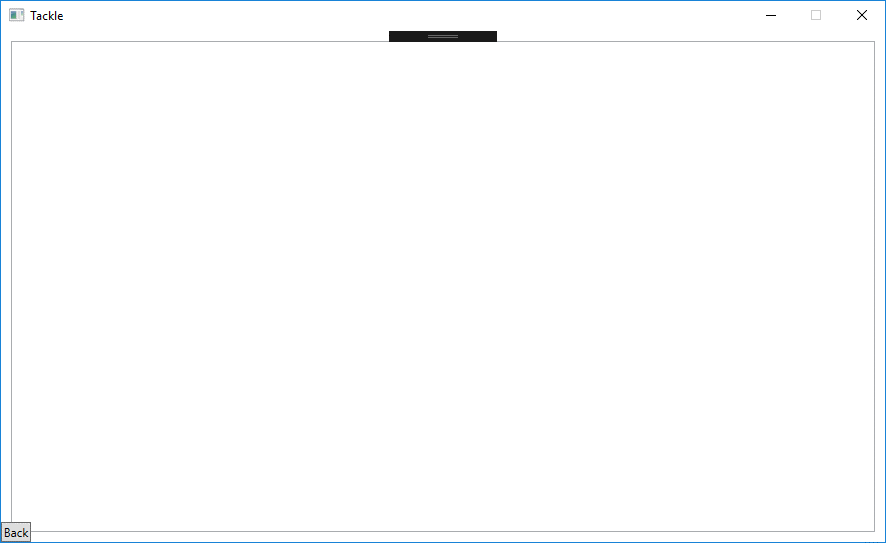


Figure 52: Draft has gone

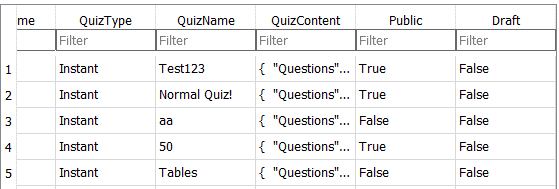


Figure 53: Draft quiz gone from database

**20)** Taking a quiz normally works perfectly fine for a normal quiz, as shown below.

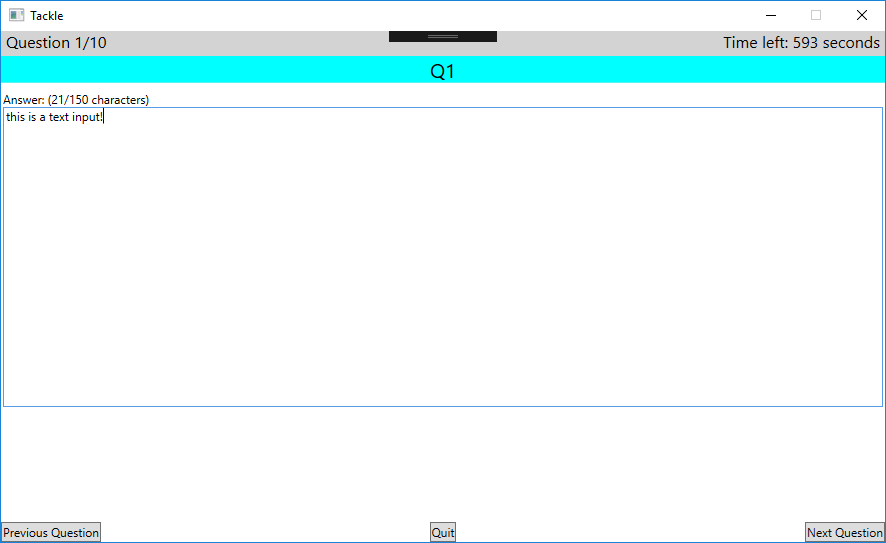


Figure 54: Text input question

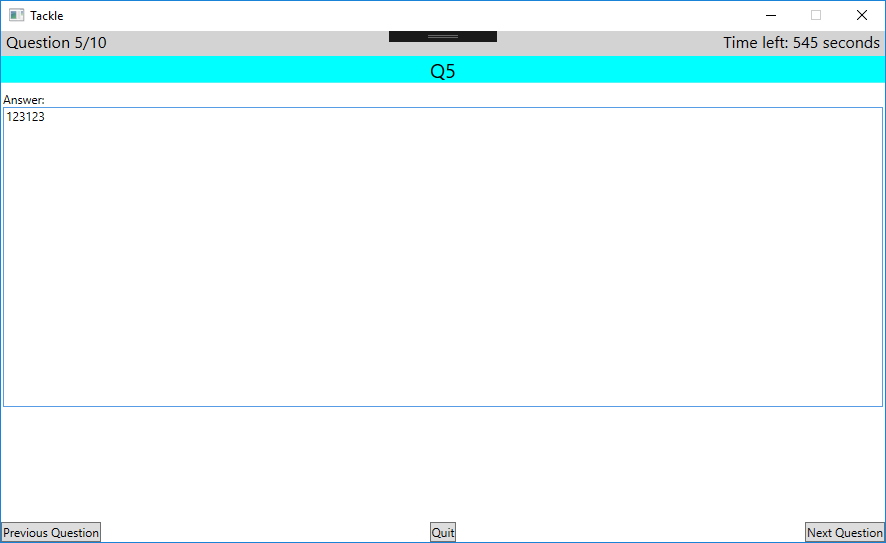


Figure 55: Integer input question - only numbers are allowed, all other inputs are blocked. As seen in the top right, the clock has also ticked down since question 1

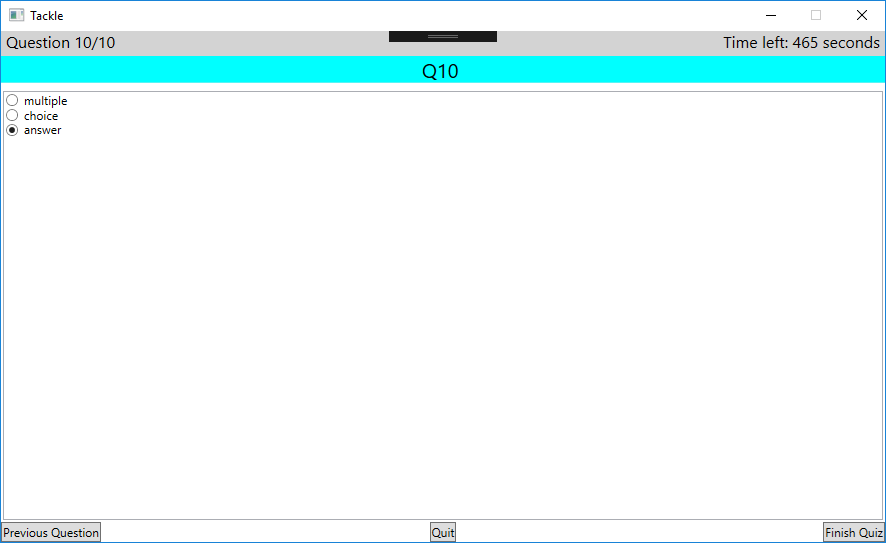
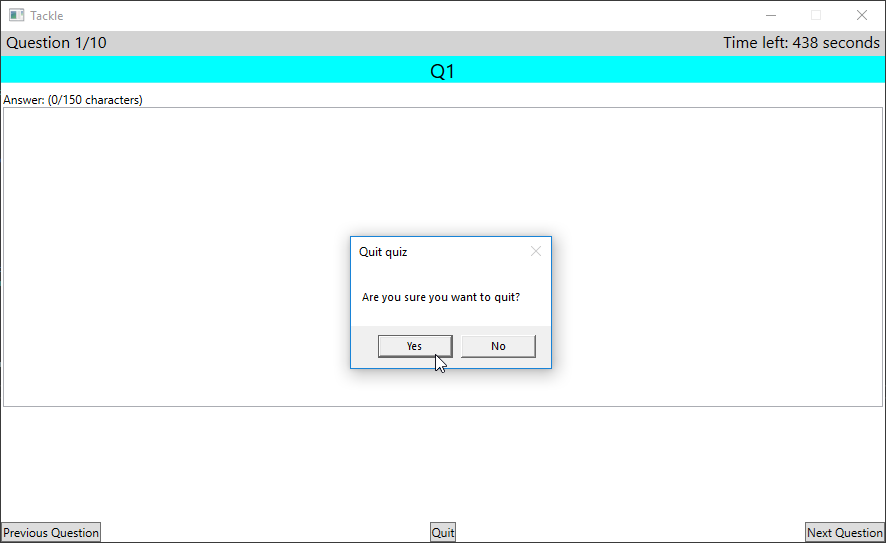


Figure 56: Multiple choice question, where only one option can be chosen at a time. This is also the last question, hence the "Finish Quiz" button instead of "Next Question"

**21)** This test works as expected - the user is able to quit the quiz and go back to their respective main menu

Student:



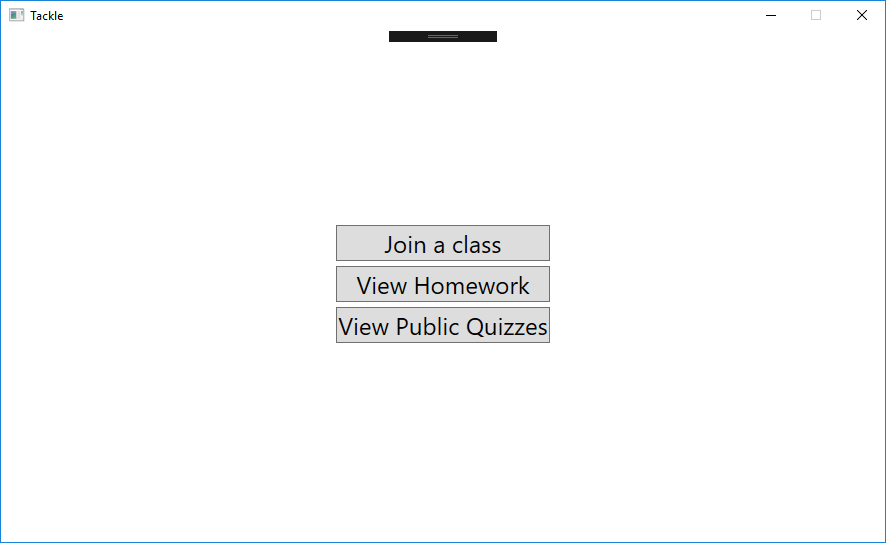
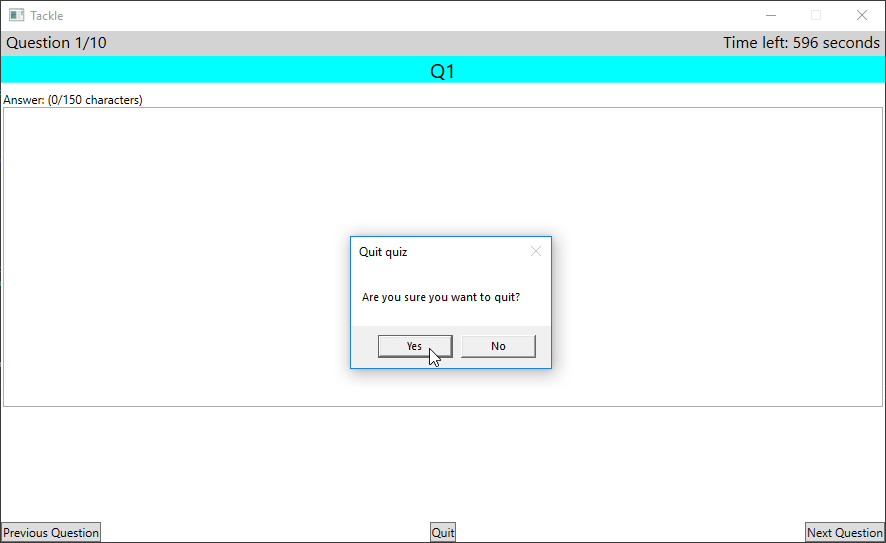


Figure 57: Returned back to the student main menu

Teacher:



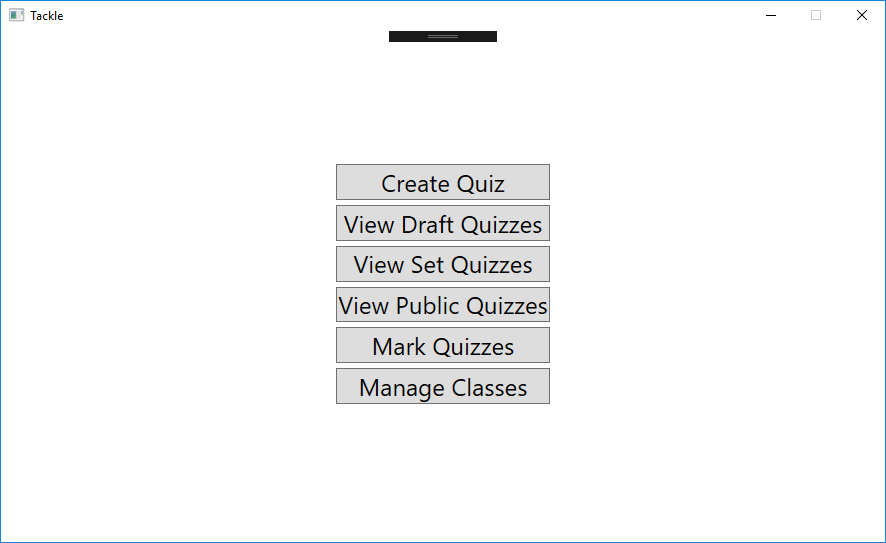


Figure 58: Return back to the teacher main menu

**22)** This test works as expected - the user is able to click on “Quit” and then press “No” to go back to taking the quiz

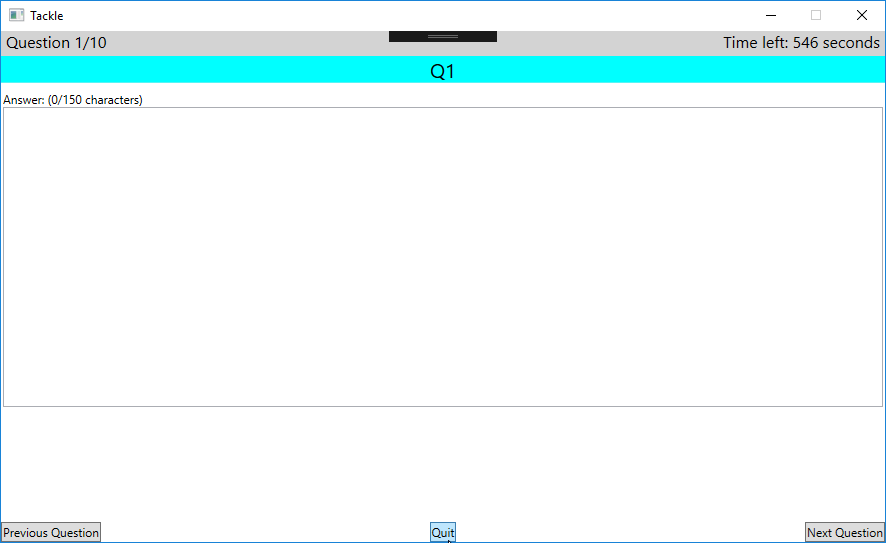


Figure 59: Clicking "Quit" button

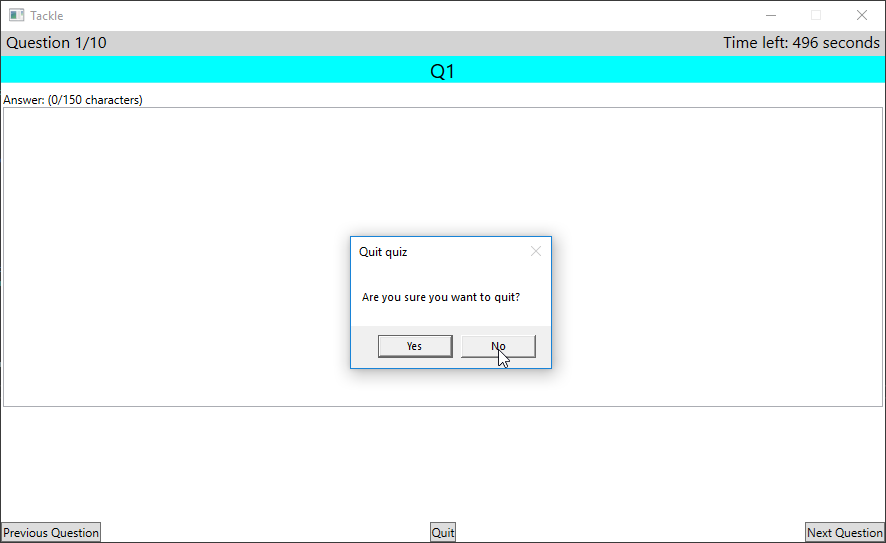


Figure 60: Clicking "No"

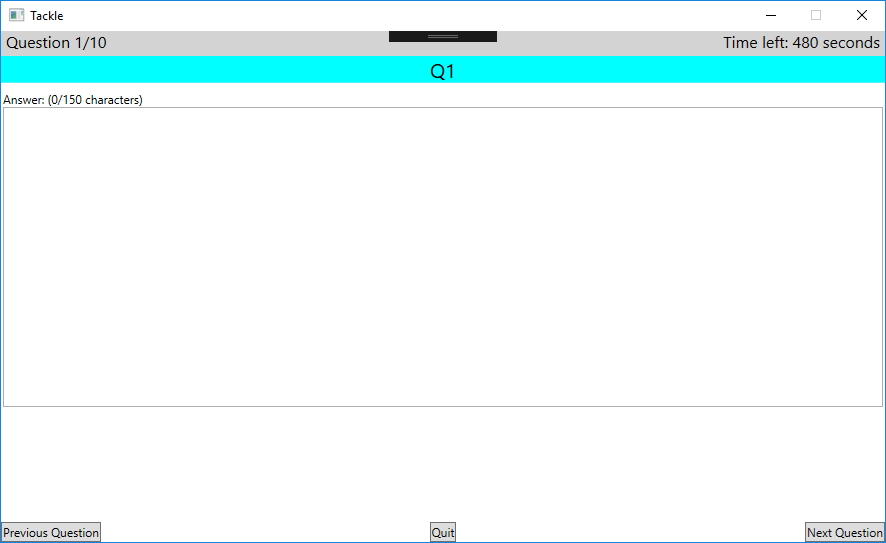


Figure 61: Back to the quiz

**23)** This test works as well. When the user clicks on “Finish Quiz”, they are redirected to a leaderboard which says their score and also displays all of the other scores for that quiz.

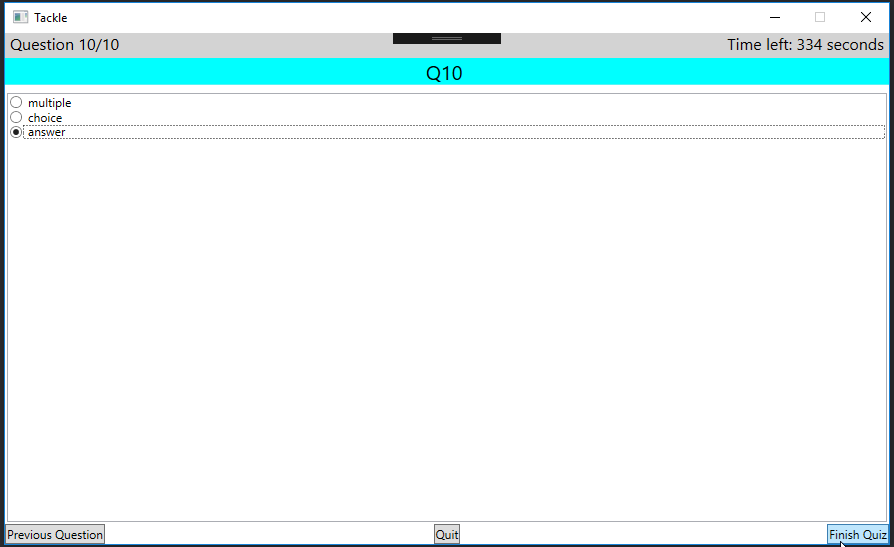


Figure 62: Clicking on "Finish Quiz"

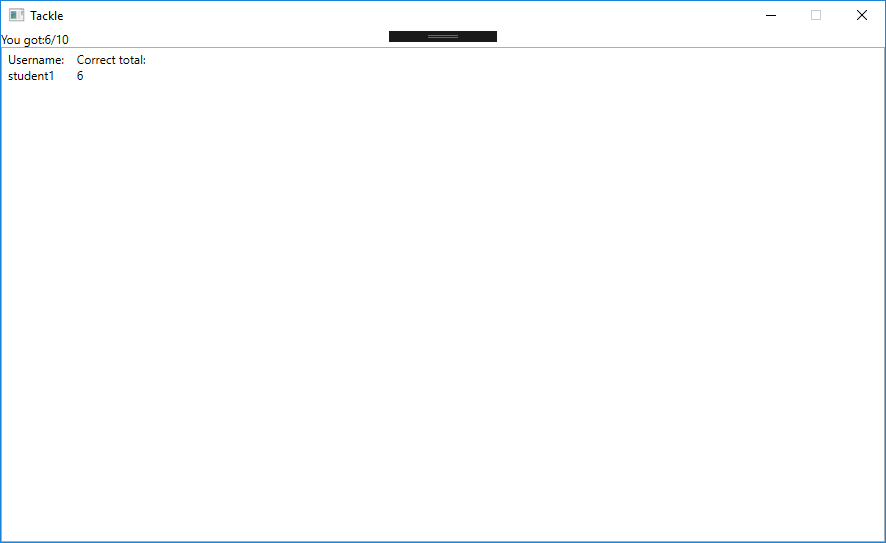


Figure 63: Showing the leaderboard



Figure 64: Database entry for the quiz attempt

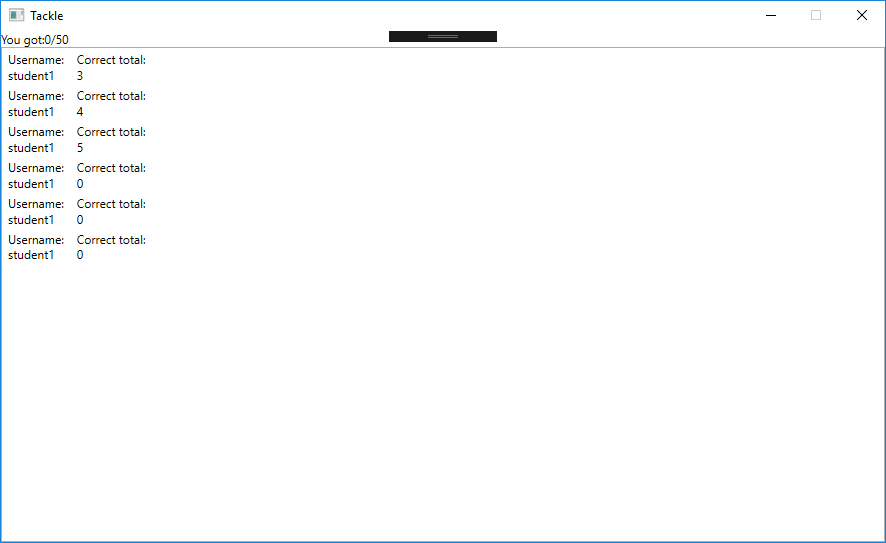


Figure 65: An example of a more full leaderboard

**24)** This test is fully functional. The student is able to check what homework they have set for them, and then take it like any other quiz.

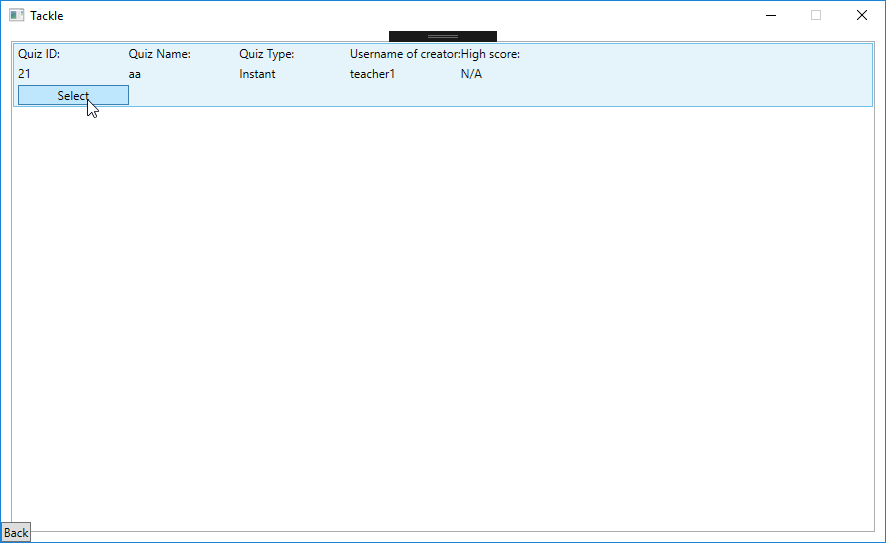


Figure 66: Selecting the quiz shown in test 14

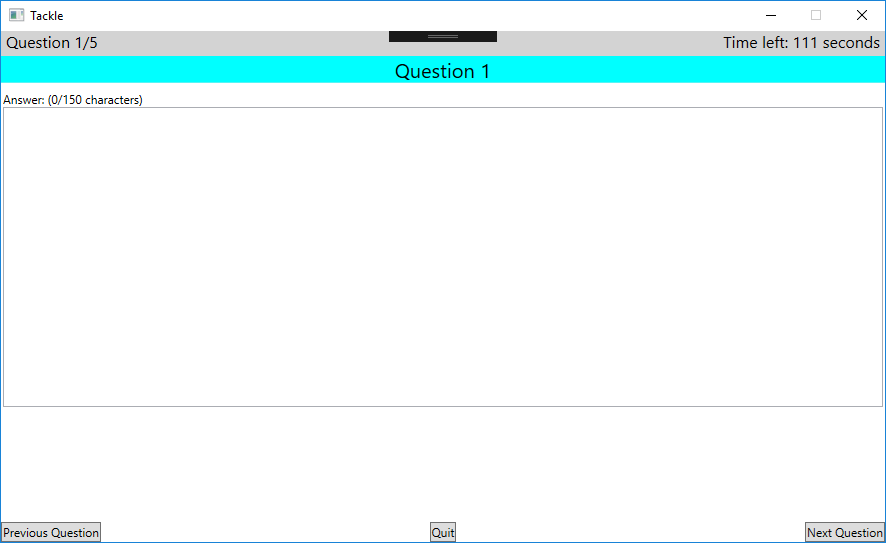
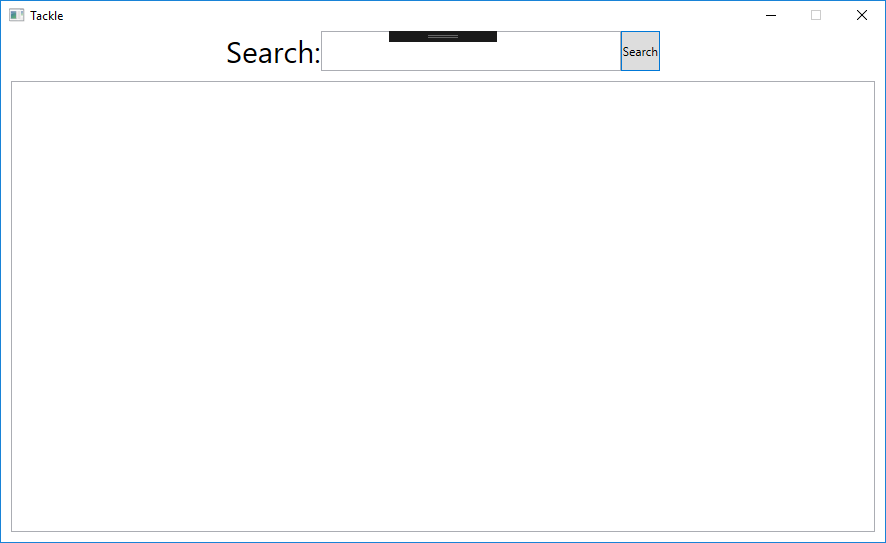


Figure 67: Taking the quiz normally

**25)** This test works as expected

20/02/19: Nothing was showing up, even when there wasn’t a search query



This was because the model for this page was being declared as a normal variable instead of a property, so this was a simple fix to turn

public QuizListModel Model

into

public QuizListModel Model {get;set;};

Fully working:

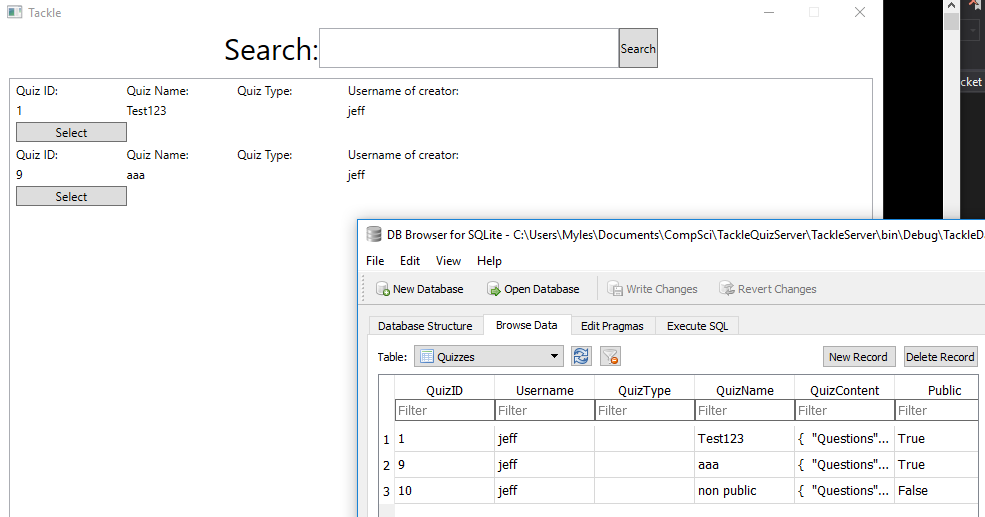


Figure 68: All of the quizzes marked as public in the database show as separate entries in the quiz list

**26)**

This test fully works, with the user being able to type in the name of any quiz on the database and it will show only the relevant results. I have also implemented it so that if a user searches for a specific quiz and then wants to see all of public quizzes without the search query, they’re able to simply remove the search query and then click “Search”

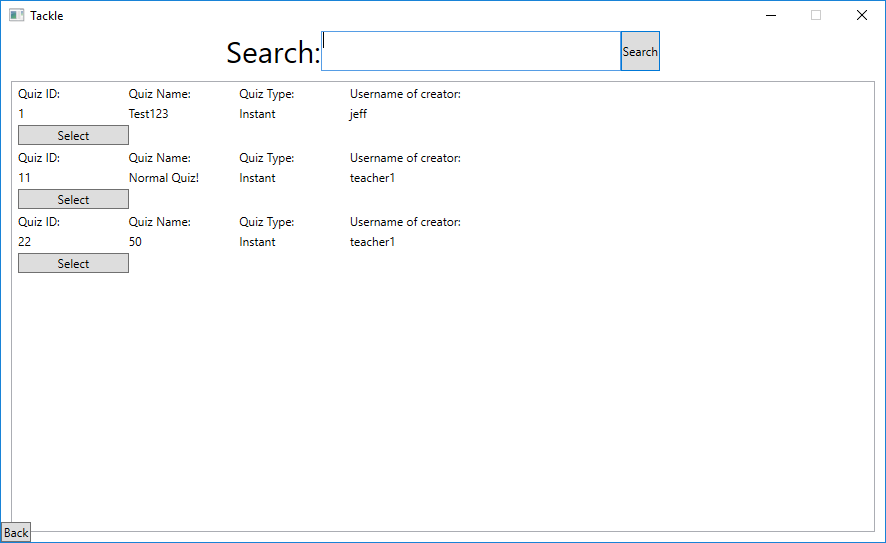


Figure 69: All of the public quizzes

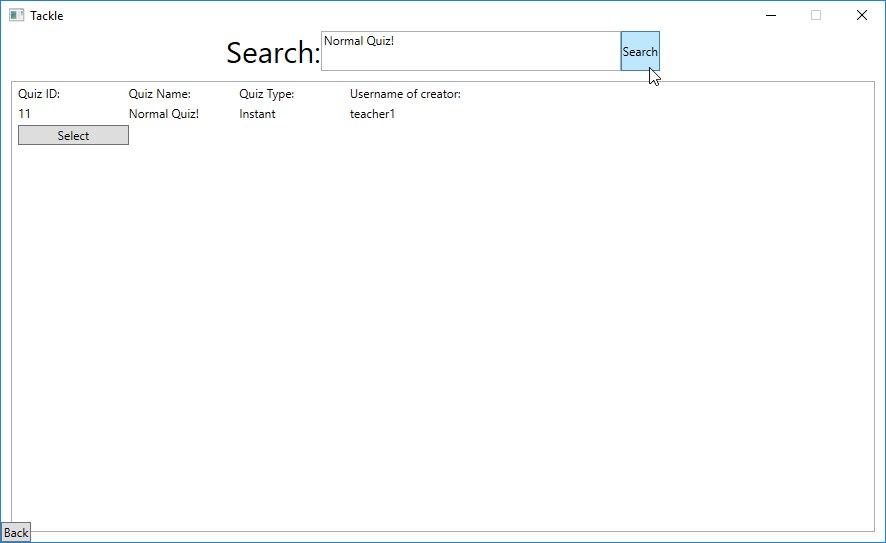


Figure 70: Search results being shown after entering the search query

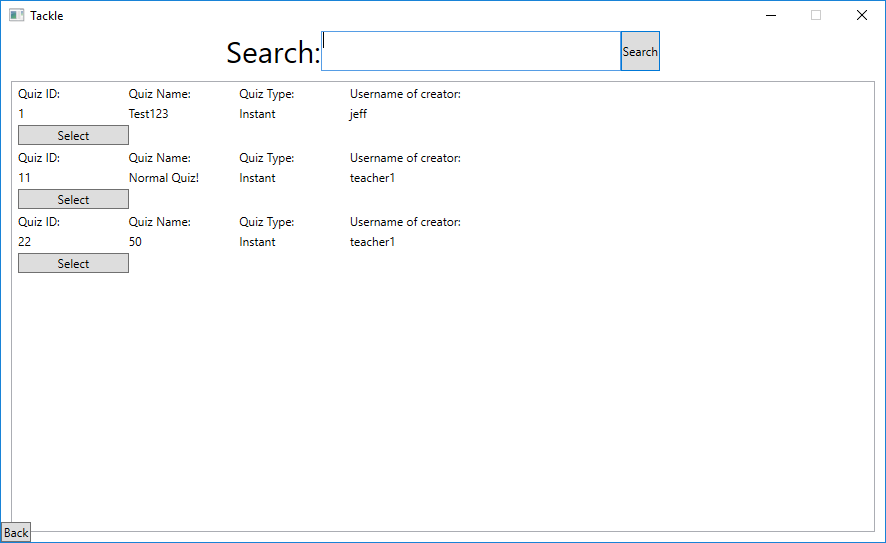


Figure 71: All of the public quizzes being shown after the removal of the search query

**27)** This test works as expected

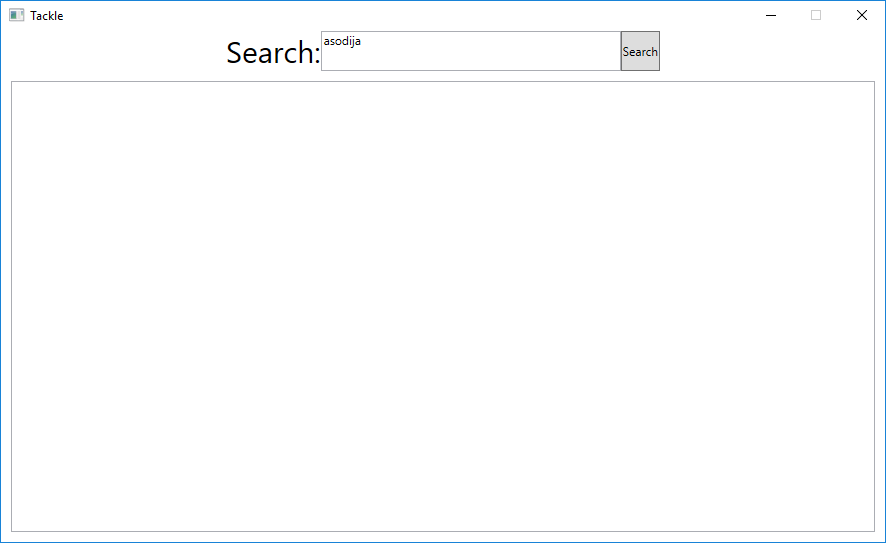


Figure 72: No quizzes showing after a search query for a quiz that doesn't exist

**28)** This test works as expected. With a student, just clicking “Select” on the desired quiz will take them to the quiz.

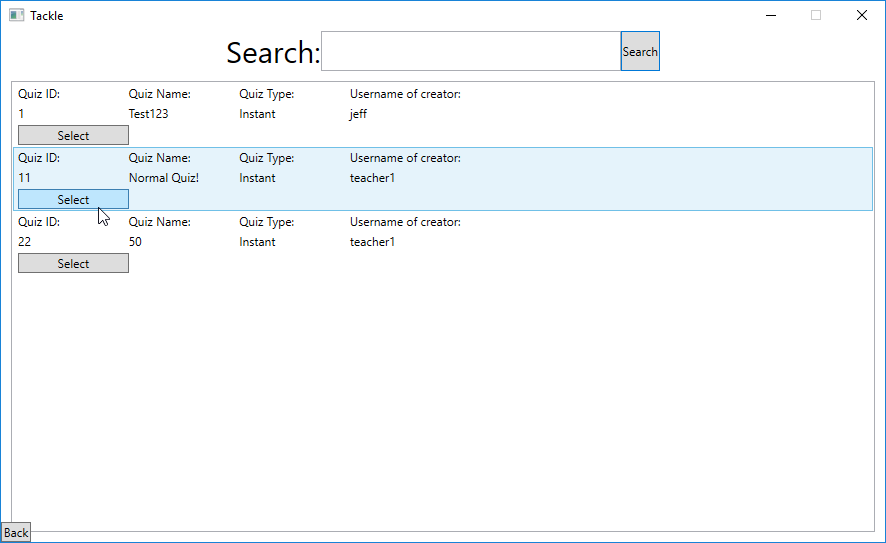


Figure 42: Clicking on "Select"

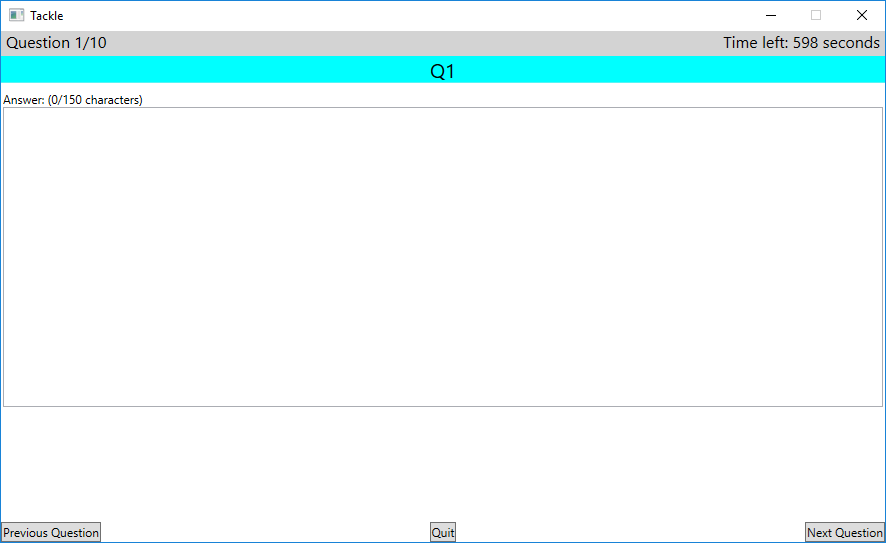


Figure 43: The quiz starts as normal

**29)** This test works as intended, with the teacher being able to select a quiz from the public quizzes page and then clicking “Select” which takes them to the selected quiz dialog box, where they can click “Take Quiz” and take the quiz normally

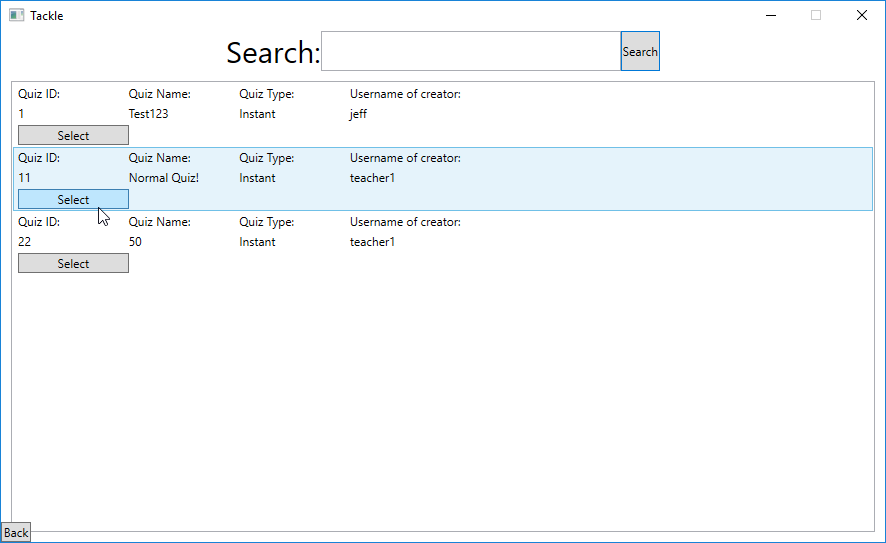


Figure 73: Selecting quiz

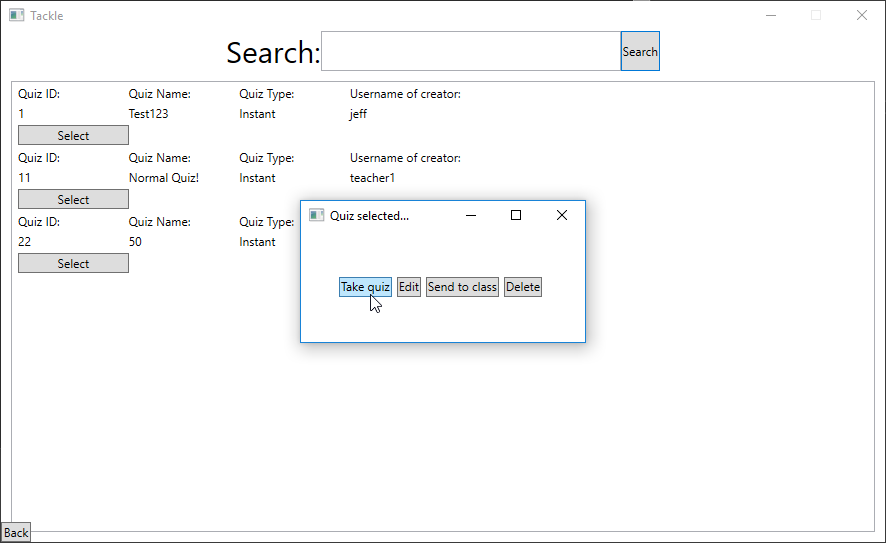


Figure 74: Clicking "Take Quiz"

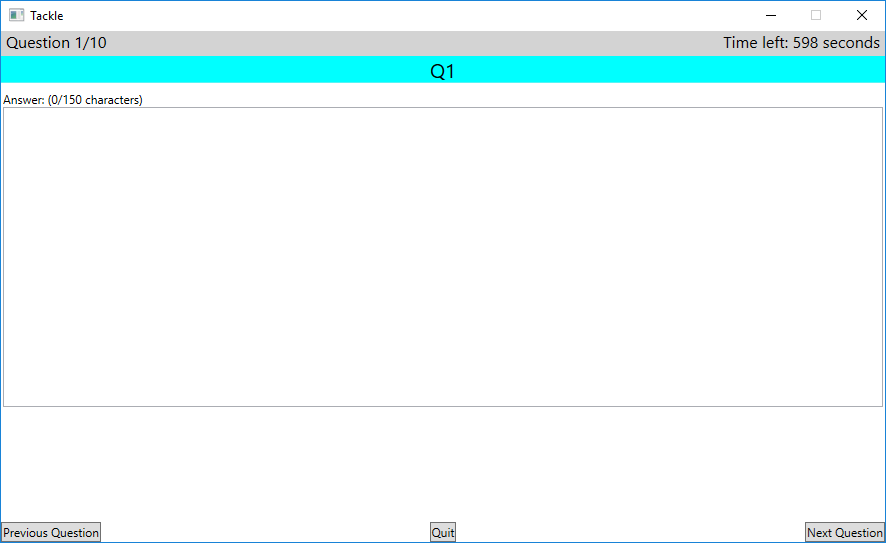


Figure 75: Taking the quiz normally

**30)** This test works by the aforementioned dialog box, instead this time clicking on “Edit”. This works as intended

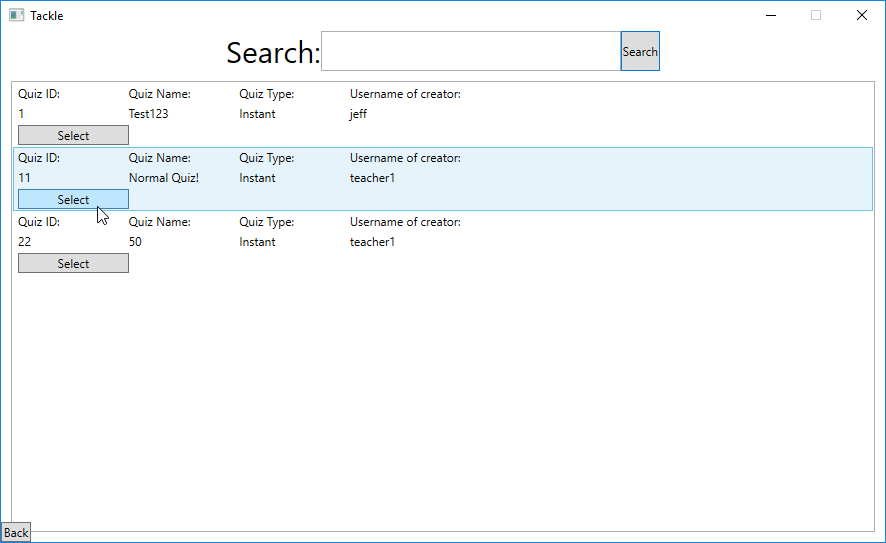


Figure 76: Selecting the quiz

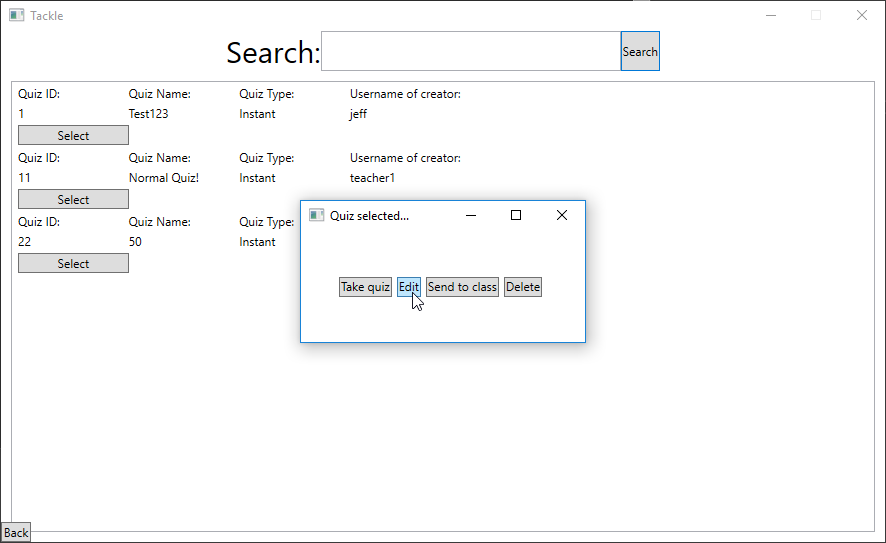


Figure 77: Clicking "Edit"

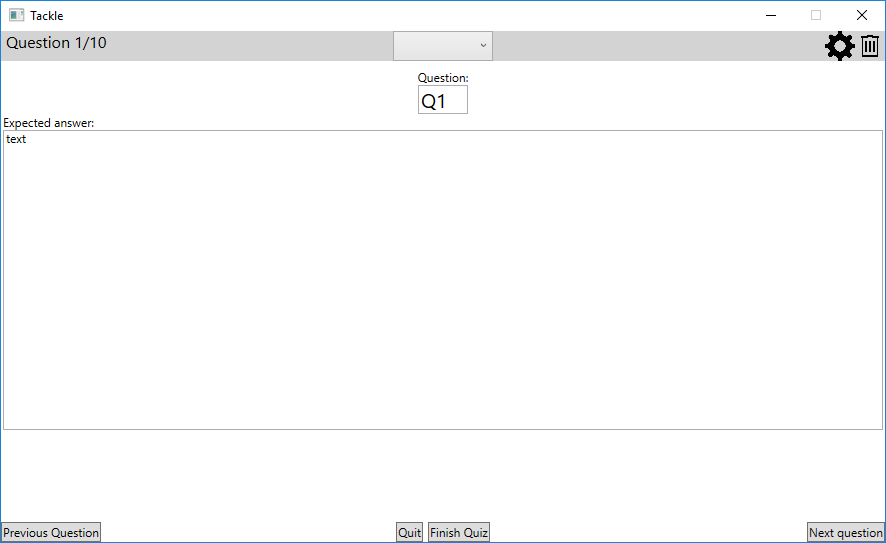


Figure 78: Editing the quiz as normal

**31)** This test works as intended also, with a dialog box being implemented asking the teacher which of their classes they wish to send the quiz to

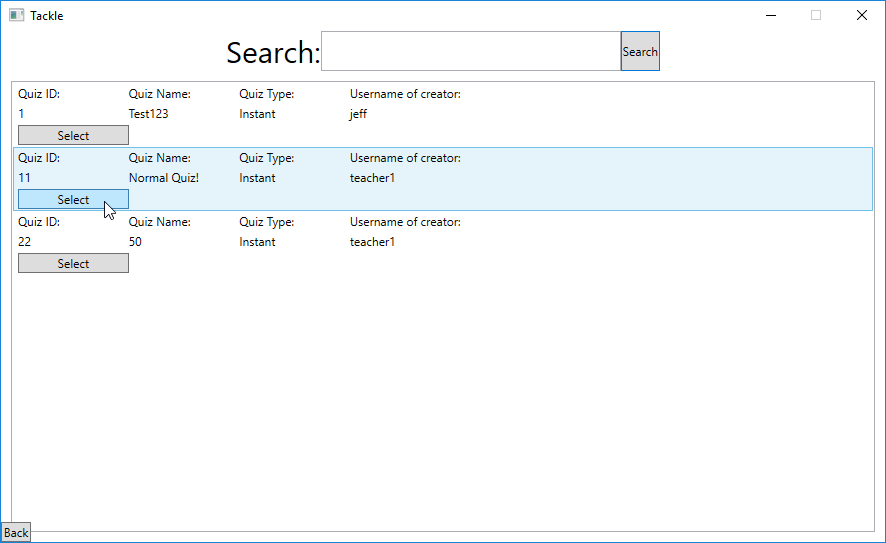


Figure 79: Selecting the quiz

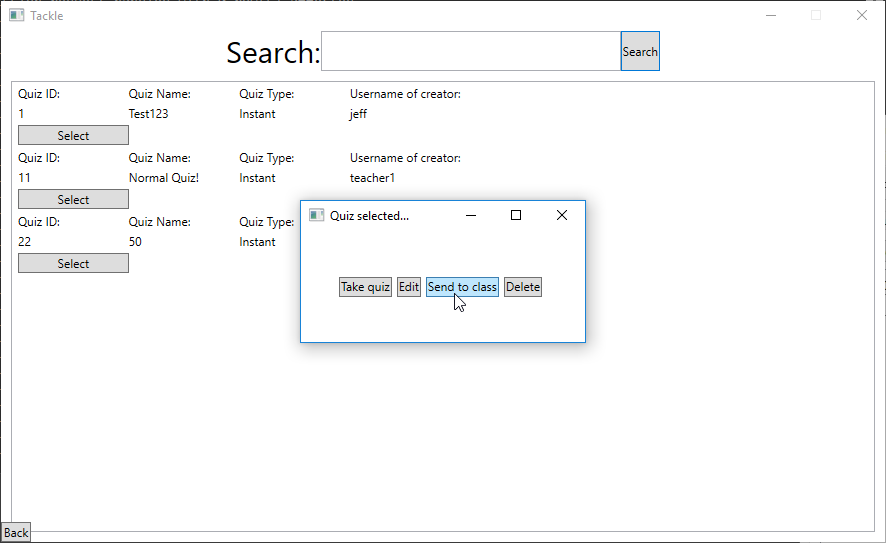


Figure 80: Selecting "Send to class"

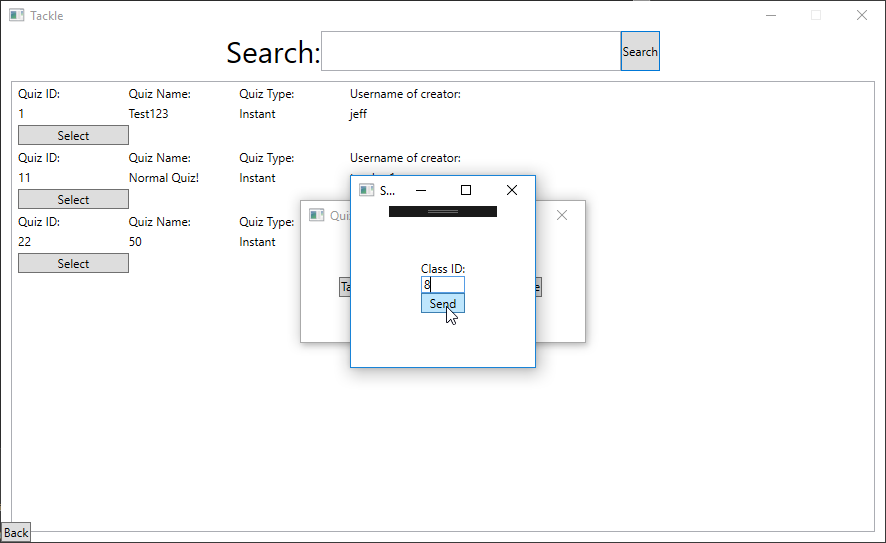


Figure 81: Dialog box asking for the class ID

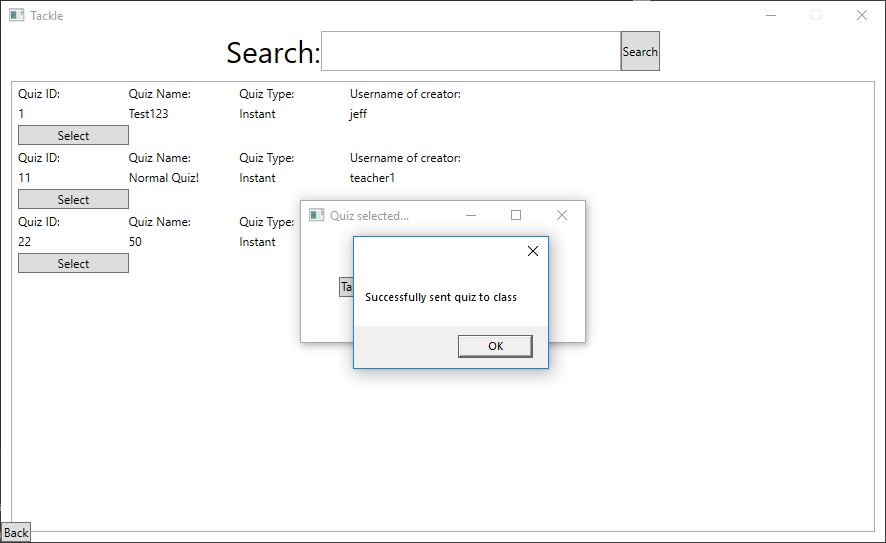


Figure 82: Message box showing that the quiz has been sent

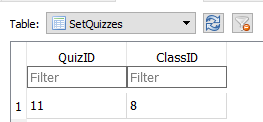


Figure 83: The set quiz showing in the database

**32)** This test works as intended, with each instance of a student taking the quiz being able to be marked by the teacher

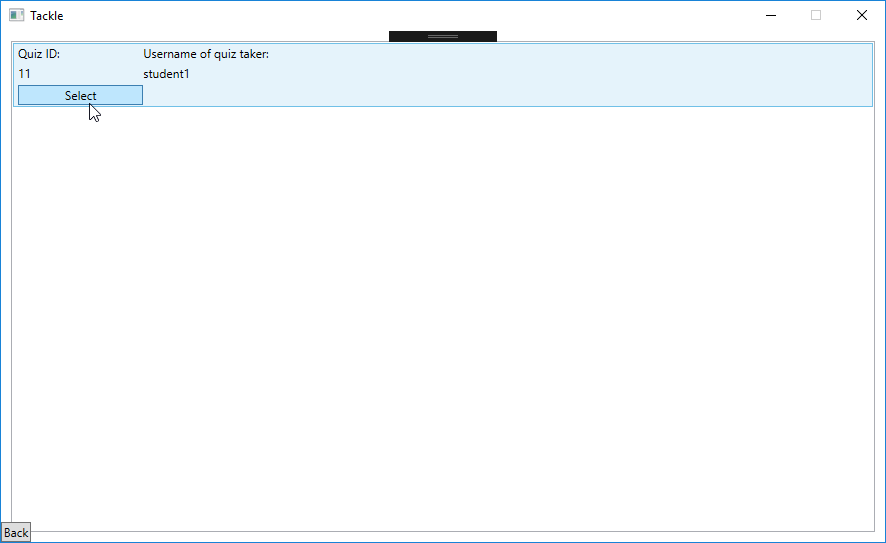


Figure 53: the quiz instance being shown on the teacher's marking list

**33)** This test also works as intended, with the teacher being able to select the quiz instance on their marking list, being able to mark the quiz question by question, finishing with their score then being updated for that quiz if necessary



Figure 54: The student's attempt at quiz ID 11 before manual marking, with 6 correct

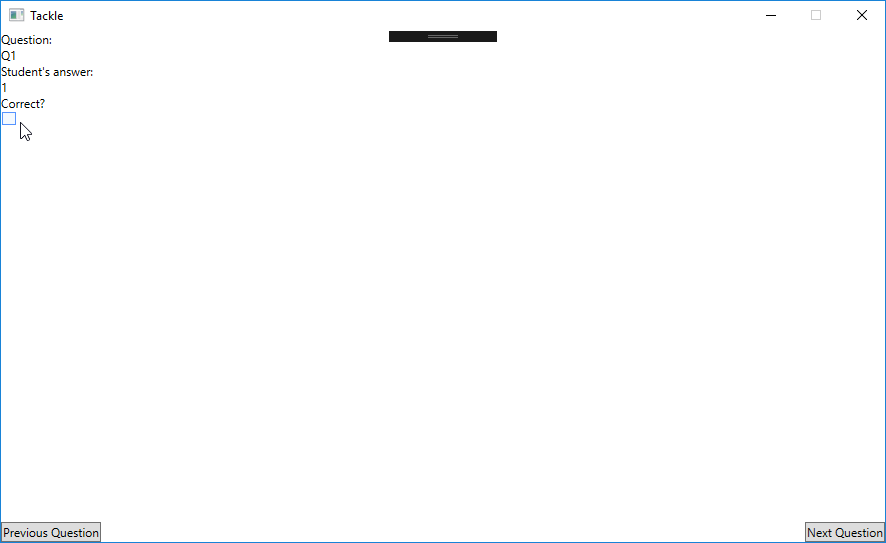
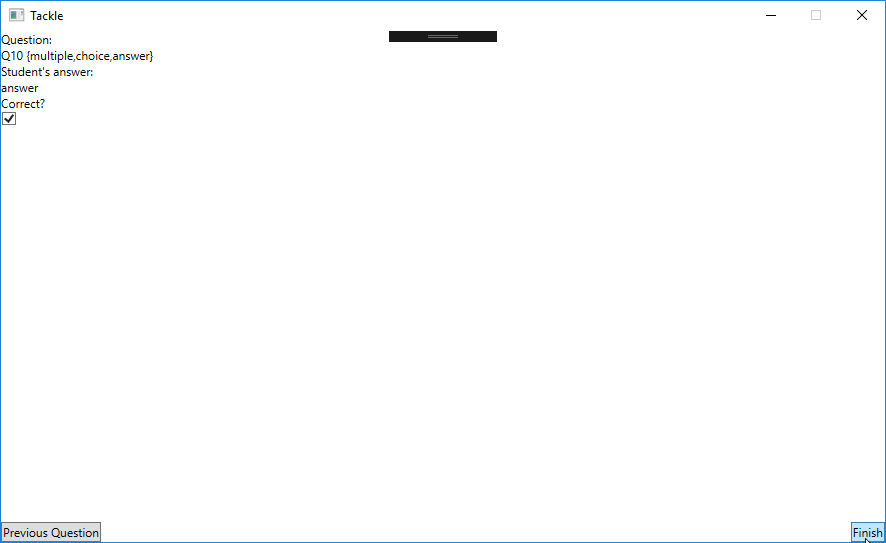


Figure 84: The teacher being able to go through each question manually checking if the answer's correct



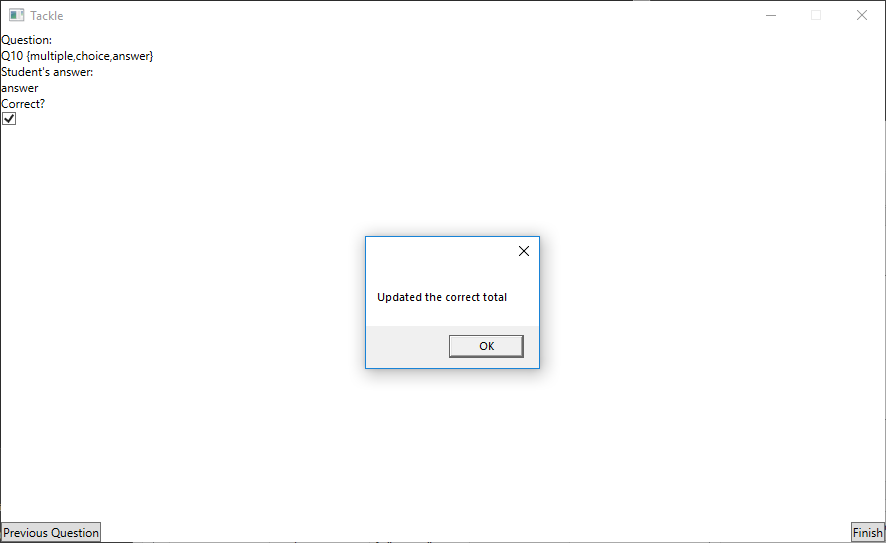


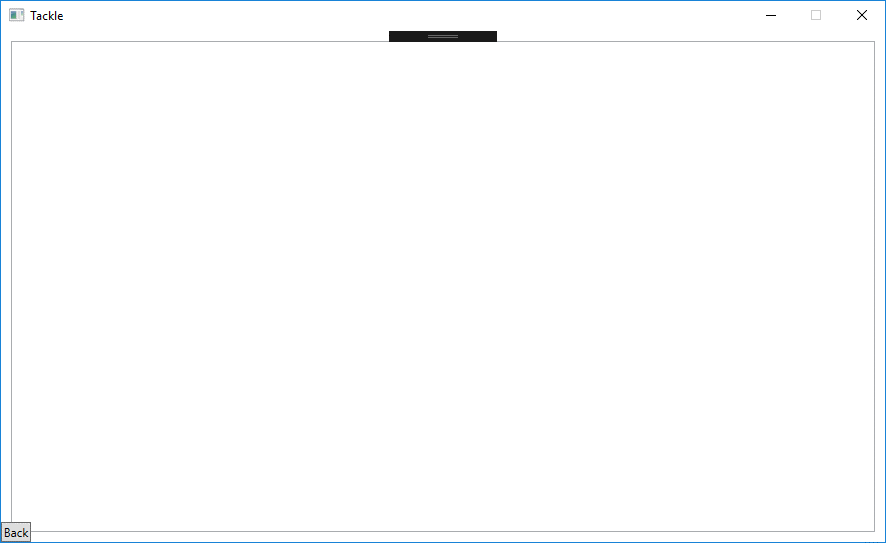
Figure 85: The teacher finishing their marking



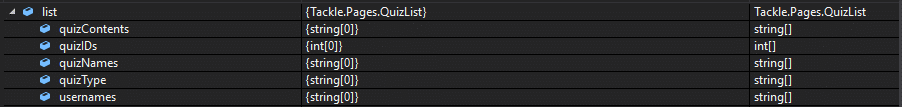
Figure 86: The total amount of questions marked as correct increased by the marking

**34)** This test works as intended

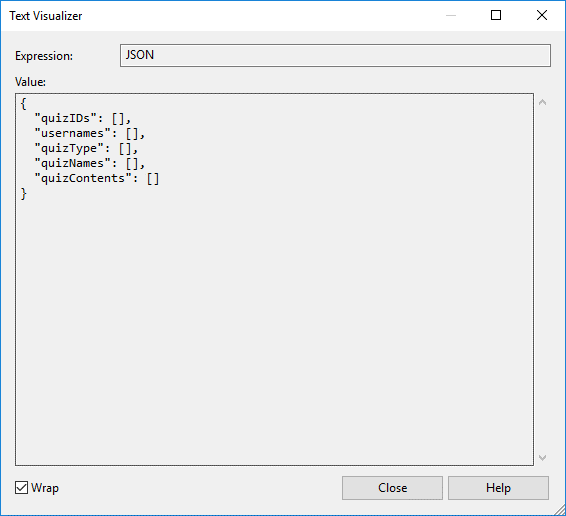
2/3/19: Nothing was showing up when attempting to retrieve the list of quizzes



Adding a breakpoint to where the model is filled shows an empty list

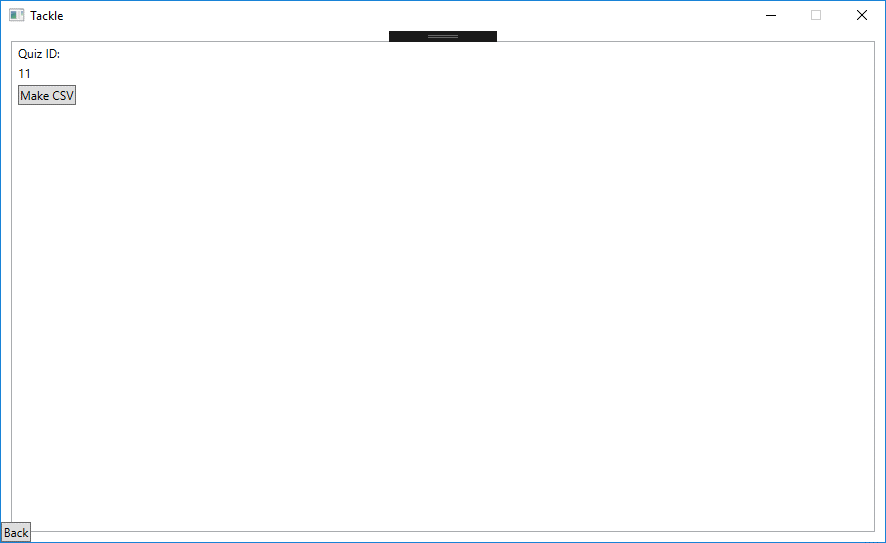


And also an empty JSON response from the server



This was due to the system of retrieving the relevant information from the server being too complicated, and so a new class called “SetQuizResponse” was created – this contained the quiz IDs that the teacher had which would be displayed and also all of the quiz attempts, which would be to put into the .CSV file export.

Fully working:



**35)** This test works as intended. For this test, I took the design decision while making the project to incorporate the student’s quiz history into the homework section, so that the student is able to see both the homework that has been recently set and quizzes that were set in the past. This was simply to declutter the main menu and user experience a bit more.

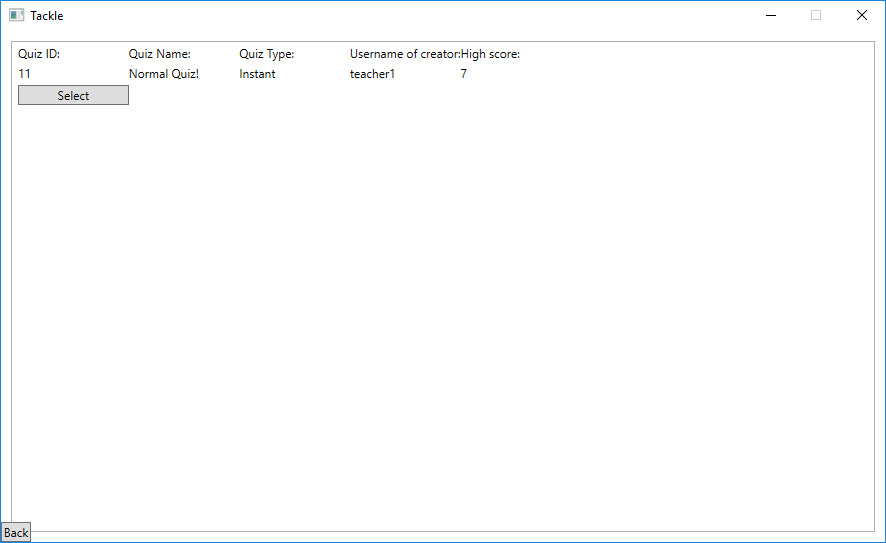


Figure 87: The homework that was set by the teacher is displayed, with the most updated high score that the respective user has on the quiz

**36)** This test also works as intended. On the teacher’s set quiz history page, they are able to click “Make CSV” under each quiz ID to export that quiz’ results as a CSV. The CSV columns are simply username and percentage, to show the student who took the quiz and a meaningful mark which may be manipulated in a spreadsheet by the teacher.

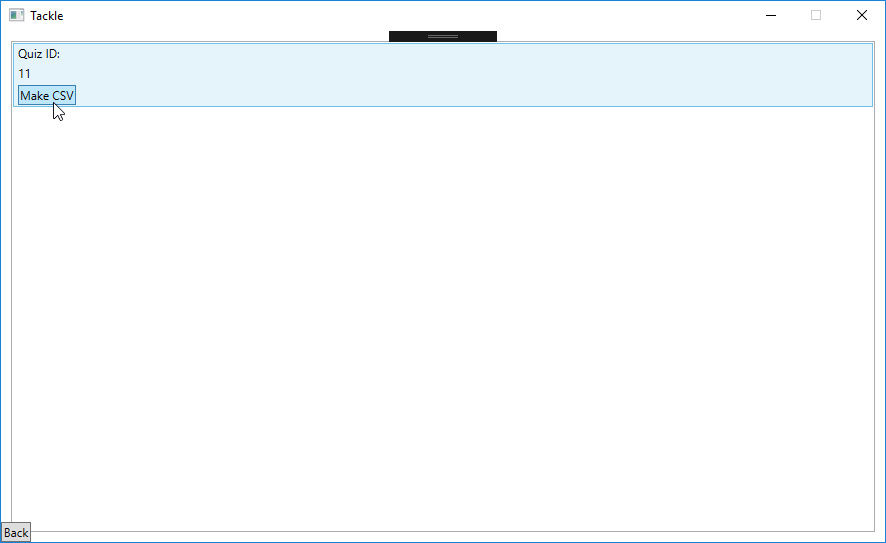


Figure 88: Clicking "Make CSV"

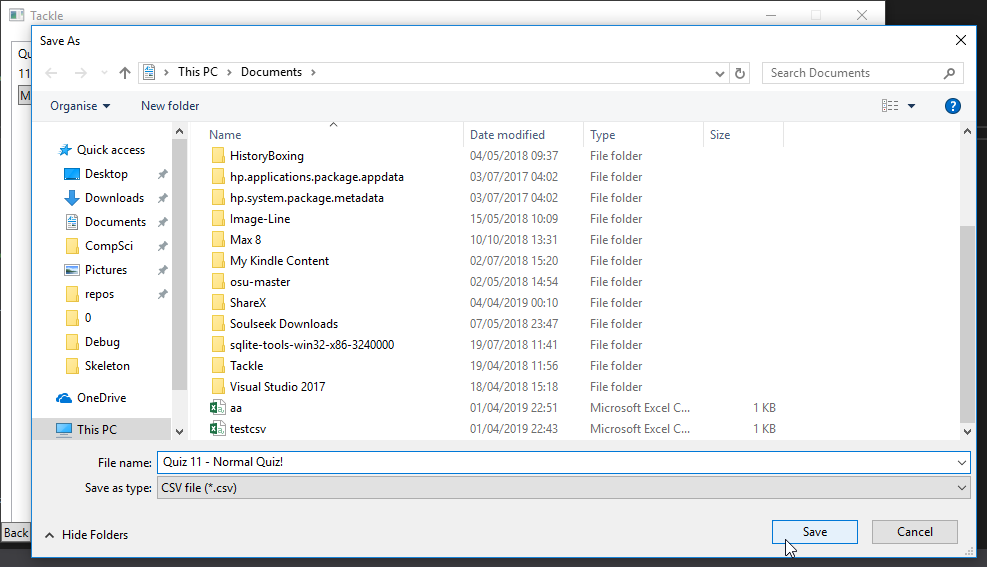


Figure 89: A save file dialog appears, allowing the user to save exactly where they want to

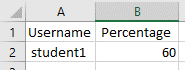
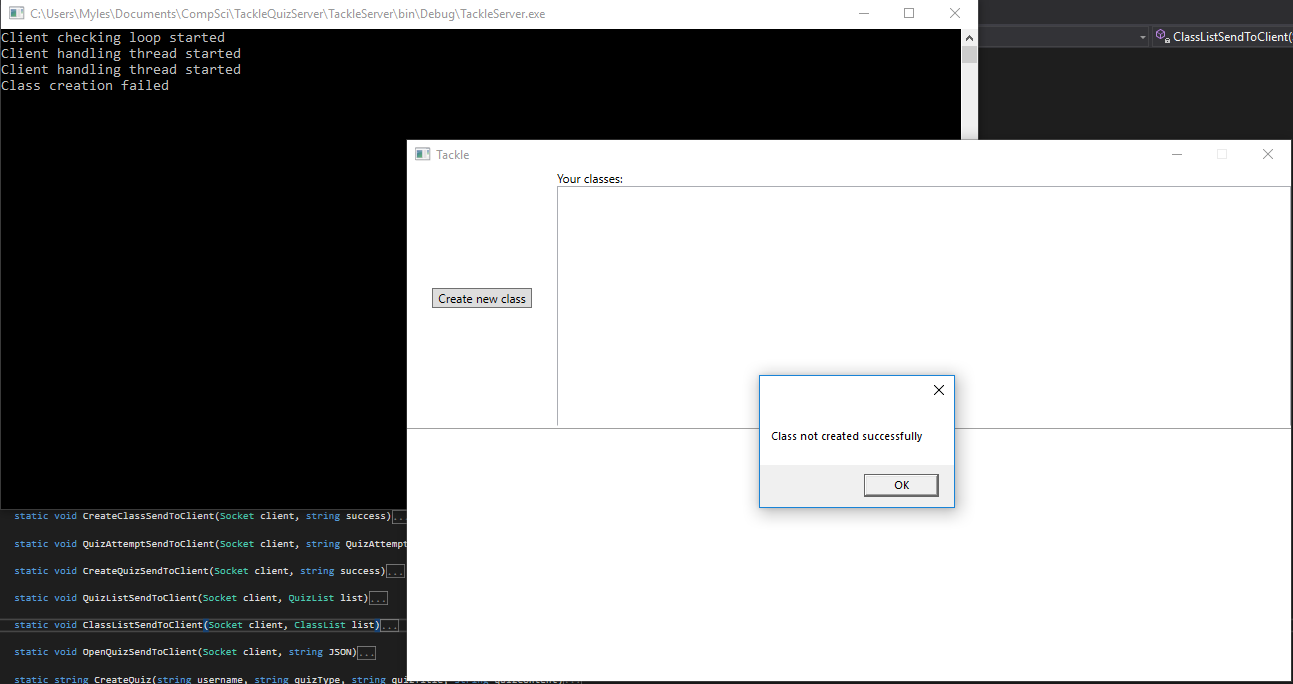
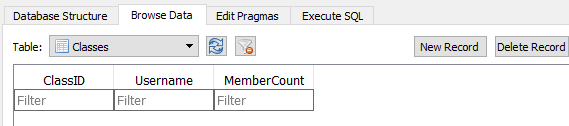


Figure 90: The results of the quiz are shown in a normal, manipulatable spreadsheet

**37)** This test works as expected

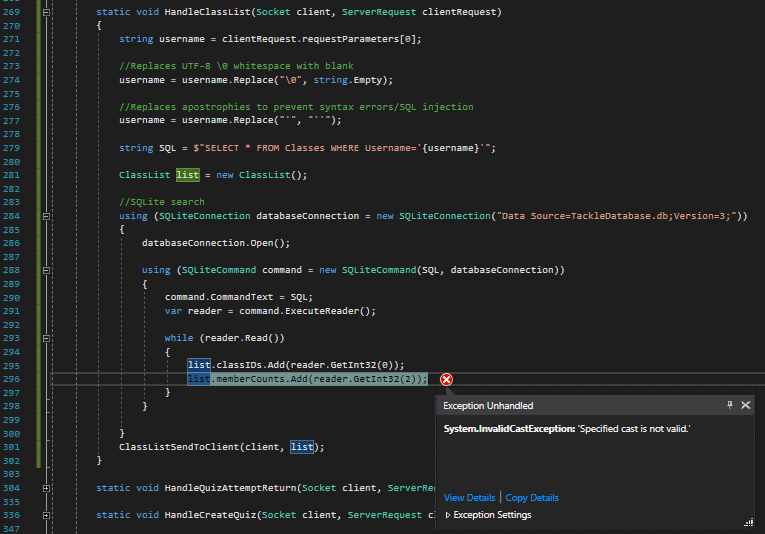
21/02/19: Class creation was simply not working, showing this same error message every time





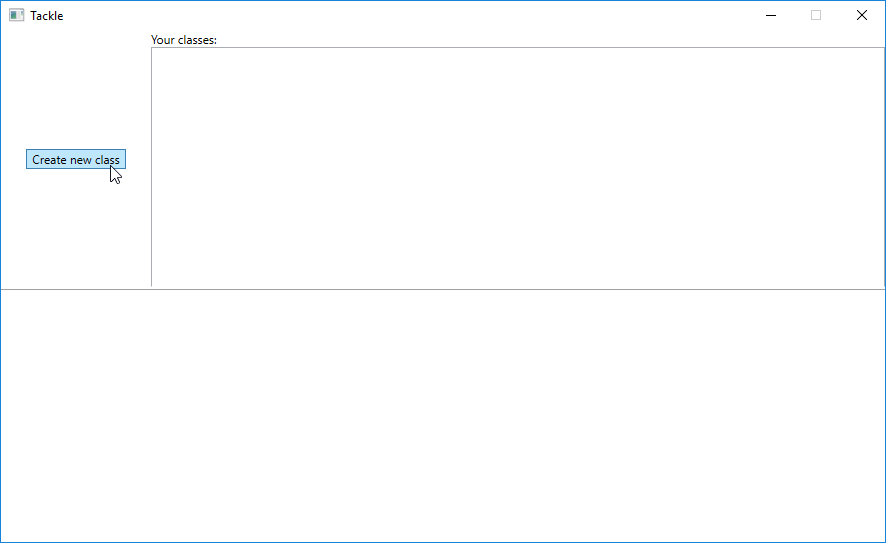
The solution to this was putting ‘ ‘ around the values in the SQL query. Although not putting apostrophes around values works for integers, for strings it is a requirement in SQL – an oversight I forgot about when implementing this initially.

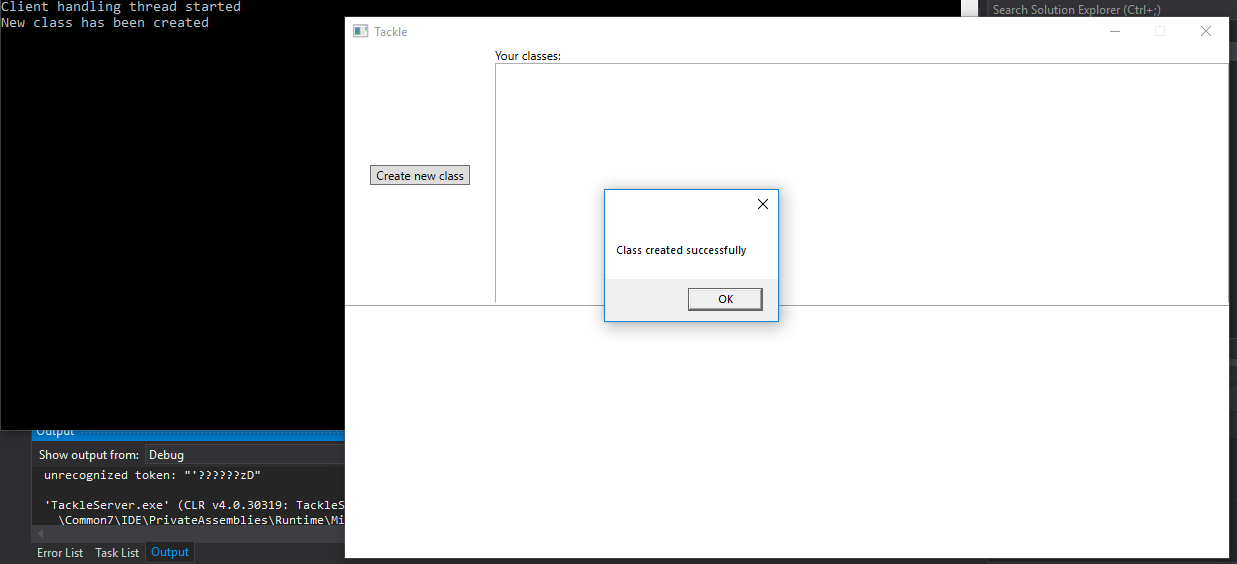
12/1/19: When the amount of members being in the class as a feature was being introduced, this error kept on occurring on the server side when trying to display the ClassList

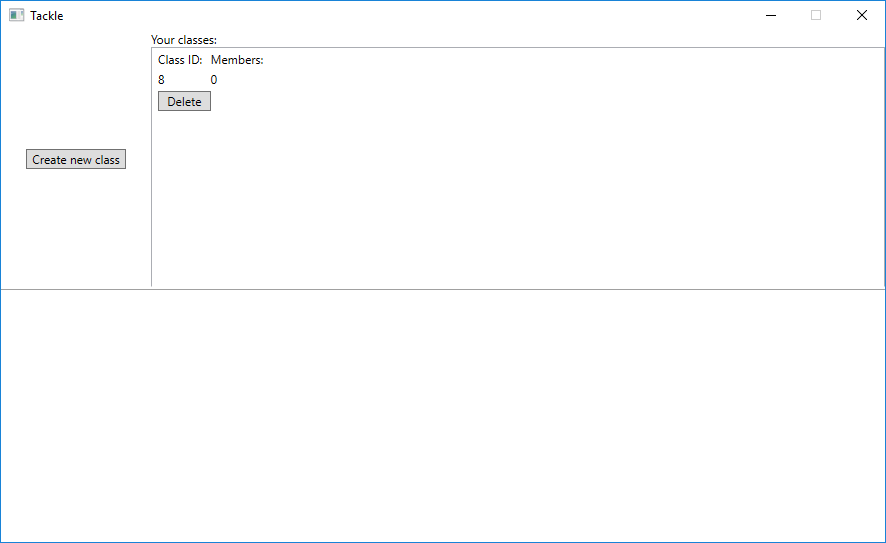


This was fixed by changing the SQL query so that when creating a new class, the MembersCount was set as 0 instead of not inserting anything resulting in that field being null. This is because “reader.GetInt32” does not account for null values in the database.

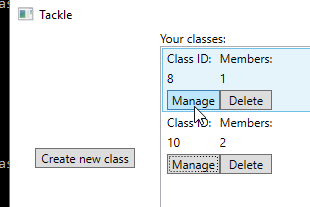
Fully working:

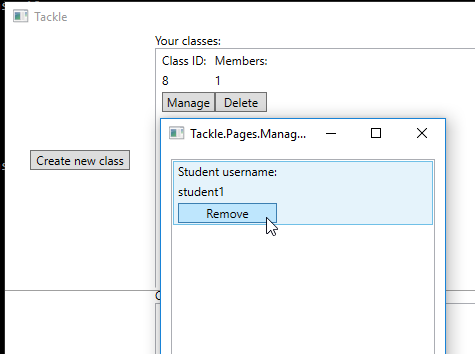






**38)** This test works as intended. In the “Manage Class” screen (accessed from the teacher’s main menu), they are able to select a class, click “Manage” and then remove a class member from there.





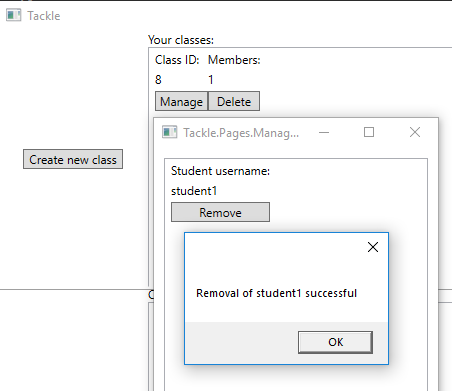


Figure 91: Clicking "Remove" on student1's entry shows a success message box

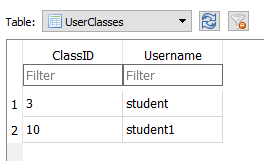
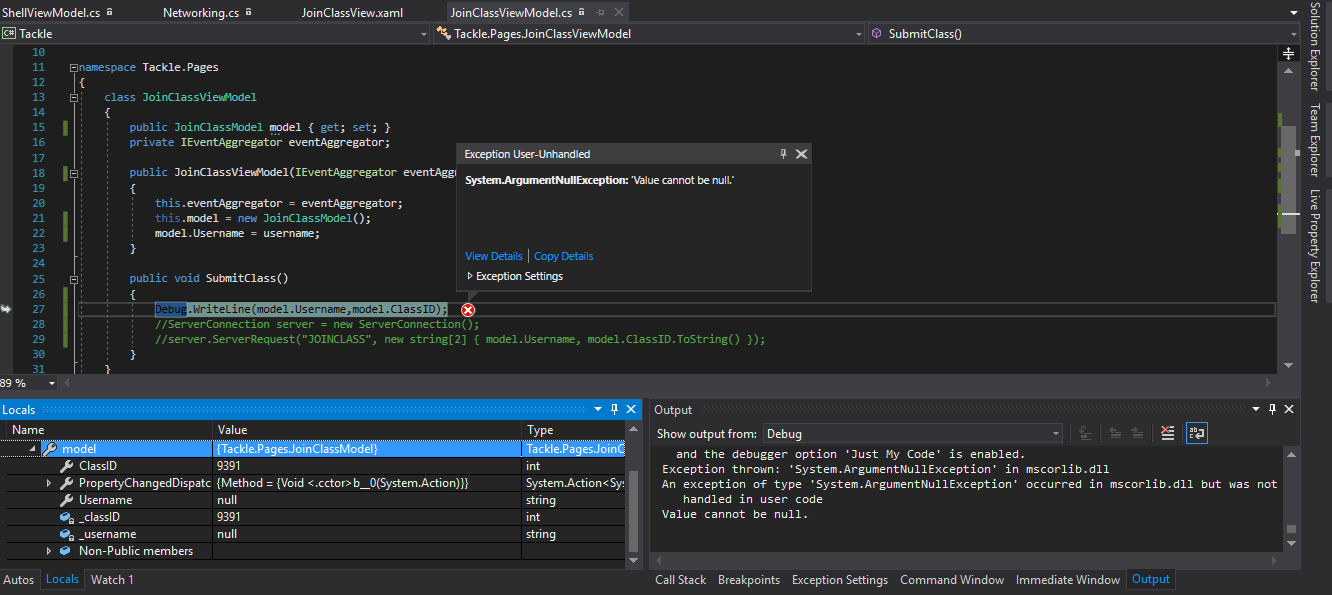


Figure 92: student1 is no longer in class 8

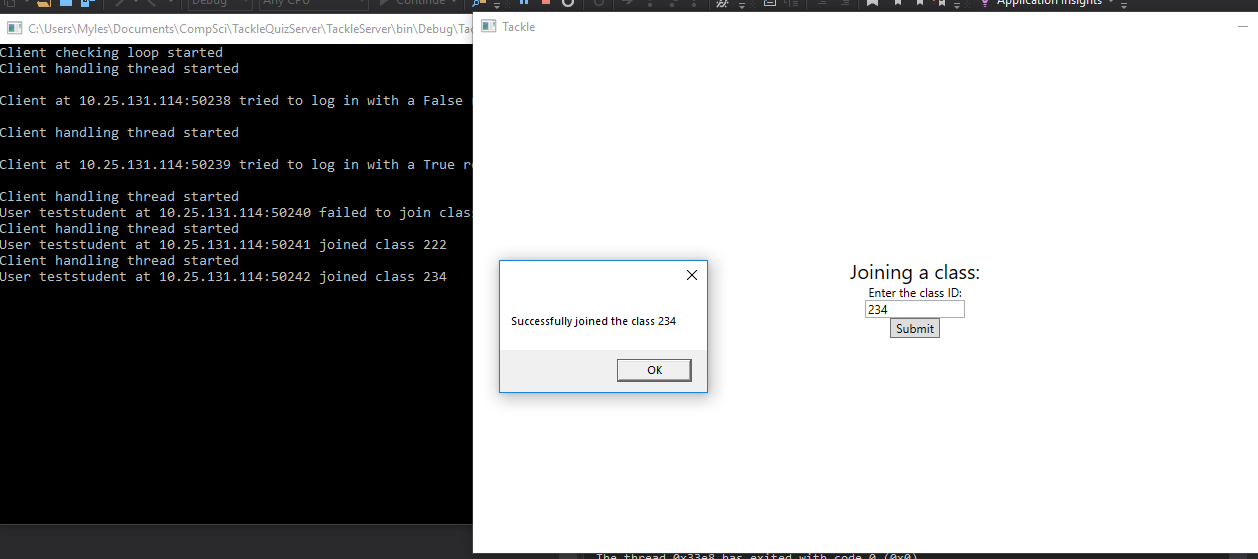
**39)** This test works as intended

31/10/18: When attempting to request to join a class, the client side crashed with the following error:



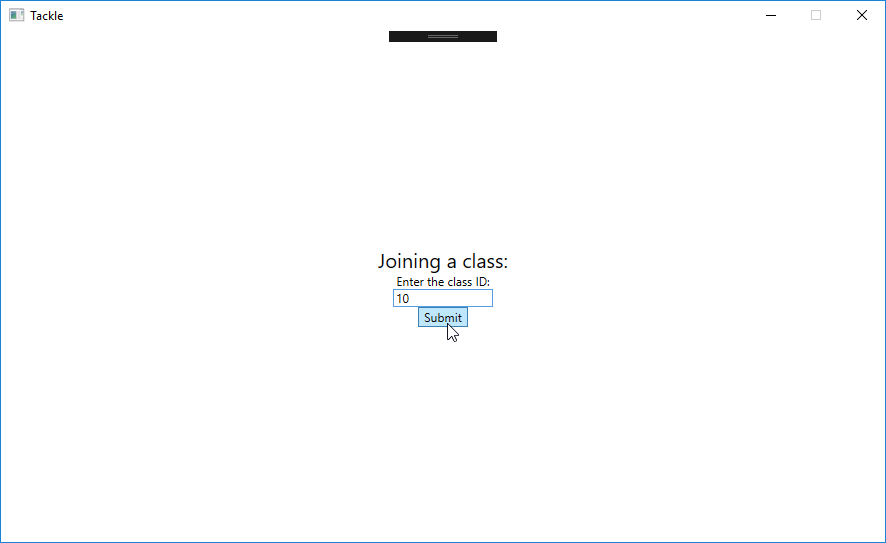
This was fixed by implementing the IHandle interface in ShellViewModel.cs for the UsernameEvent class. This means that the username could be properly passed to the ShellViewModel for later use, such as this. Without this implementation, the ShellViewModel was not capable of handling the UsernameEvent class, and so the value for username was always null.

8/11/18: The user is able to request to join classes that don’t exist



This was fixed by using ExecuteReader instead of ExecuteNonQuery. This is because ExecuteReader is the function intended to be used for SELECT queries – ExecuteNonQuery only returns the amount of rows affected by an UPDATE or INSERT statement, and will always return -1 when dealing with SELECT

Fully working:



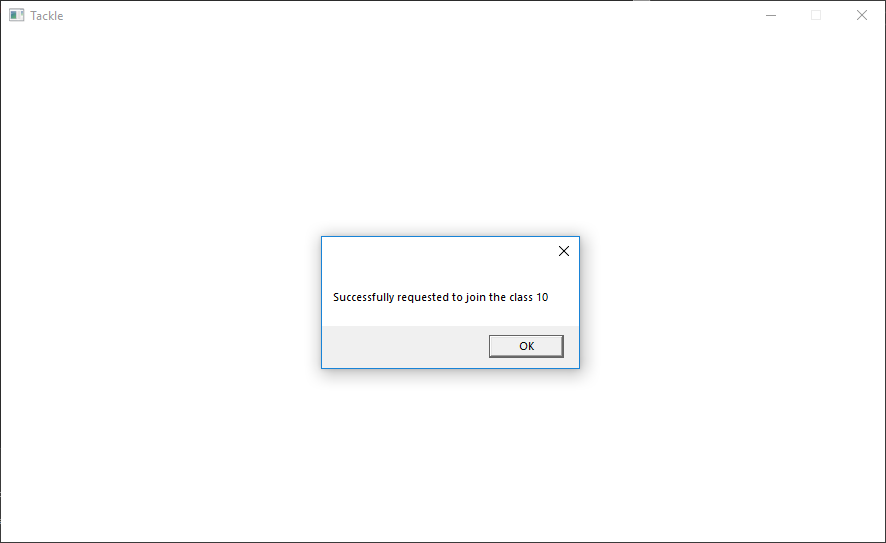


Figure 93: Requesting to join class 10

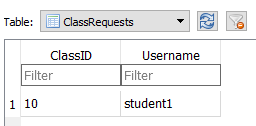
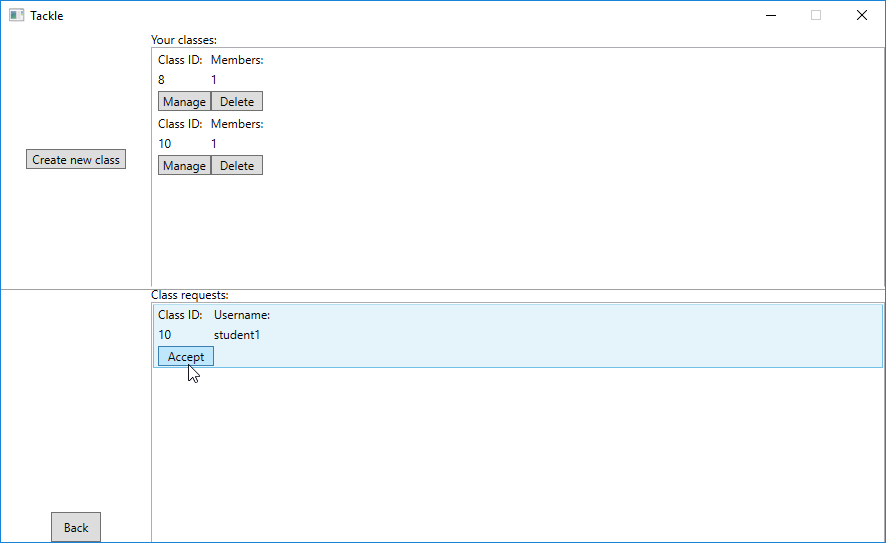


Figure 94: the class request showing in the database



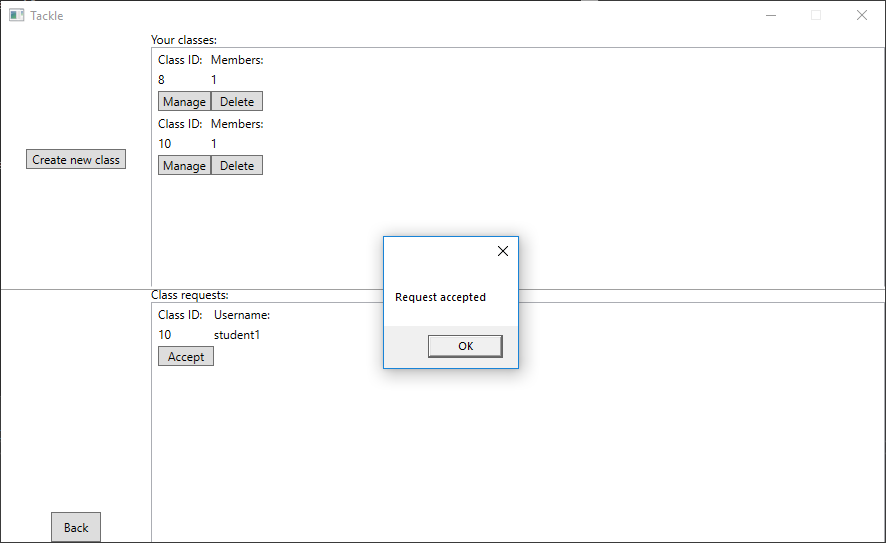


Figure 95: teacher accepting the request

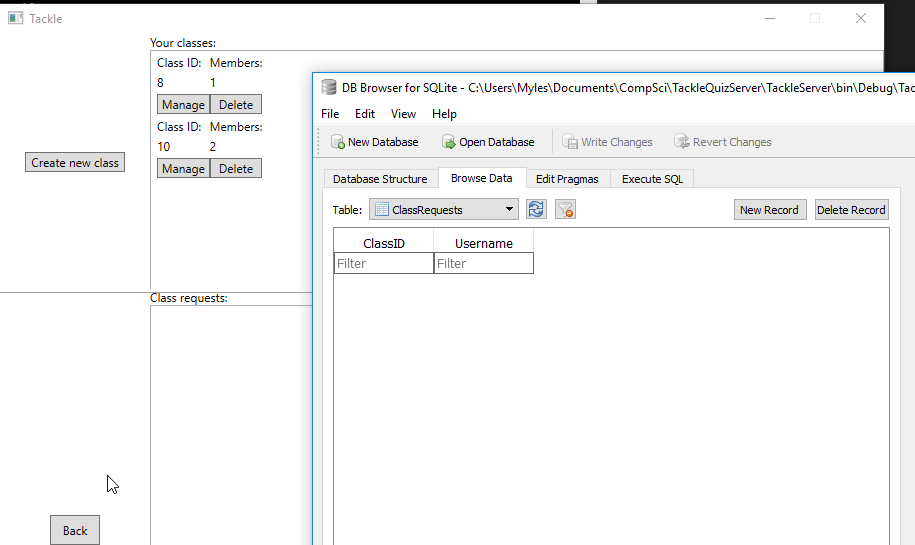


Figure 96: Request removed from ClassRequests

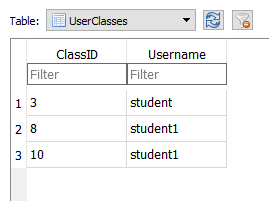


Figure 97: student1 is now in class 10

**40)** This test works as expected

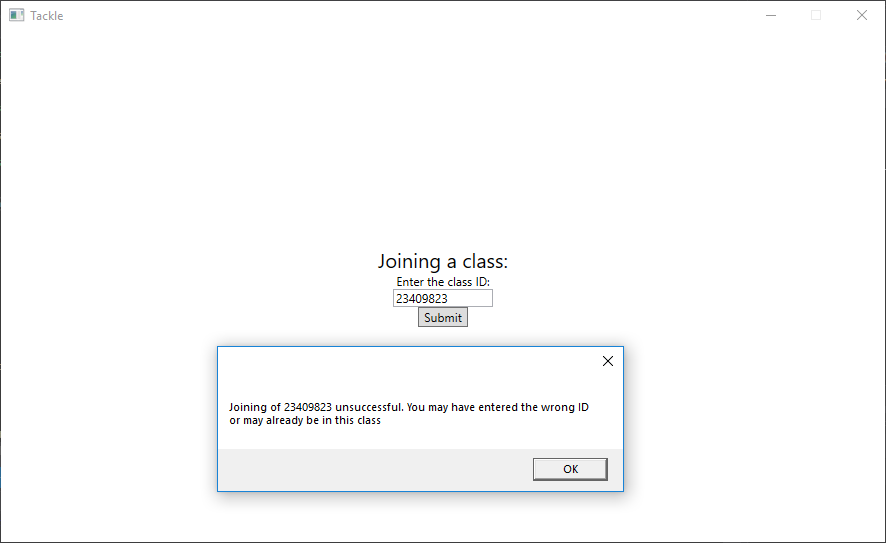


Figure 98: Entering a wrong ID brings up an error message saying that it was unsuccessful

# Evaluation

## User feedback

### Teacher – Mr. Frost

Mr. Frost had generally good feedback on the program, with him positively reviewing the user interface which he said was “professional” and resulted in being able to quickly be able to do things such as create a quiz. He also liked the fact that quiz creation wasn’t subject specific with the different types of question, which is something I aimed to achieve due to my research as part of the analysis. He also liked the feature that he requested, which was saving to a CSV file. He also positively reviewed the editing of quizzes, which he said was straight forward, with the archiving of set quizzes being good, as well as the fact that the program is able to search through all of the public quizzes.

However, during my presentation of the project, a bug was experienced that I had not foreshadowed in my testing. This was because Mr. Frost attempted to make a 1 question quiz, which the program could not handle properly and so saved the expected answer as a null value. As a result, when taking the quiz, the quiz opening algorithm could not handle the answer being null and crashed. If I were to do this project again, I would make these systems much more robust and include much more rigorous testing to include edge cases like these.

### Student – Alexander Henderson

Alexander also had good feedback on the program. As a student, he created a student account and explored the different options given to him such as requesting to join a class and taking public quizzes. Like Mr. Frost, he remarked how easy to use the user interface was, especially when viewing and playing public quizzes. He also liked the implementation of leaderboards, which I had implemented as a result of his desire to have competition in a quiz taking program which he felt would make it more fun. However, the leaderboards in the program are not ranked by score, which Alexander said could be an issue if he were trying to be competitive to beat a high score, as he would have to trawl through many people’s scores to find the one he wanted to try to beat.

## Meeting the objectives

|  |  |
| --- | --- |
| Objective number | How well it was met |
| 1. a) The user should be able to sign up with a username and password, and say whether they’re a teacher or a student | This objective was fully met. As seen in the testing, when signing up the user is able to enter a username and password of their choosing and then they’re able to say whether they’re a teacher or student with a check box. I also incorporated SHA256 hashing to ensure password security when signing up. |
| b) Once signed up, the user should be redirected their respective main menu (depending on whether they’re a teacher or student) | This objective was fully met, with the respective teacher or student main menu showing up when they have signed up. |
| c) If the user tries to sign up with an existing username, they should be given an error message and stay on the login/signup page | This objective was fully met, as any request to sign up with a username that already has an entry returns an error message prompting the user to try again. |
| d) The user should be able to log in with the details that they signed up with | This objective was fully met as per the requirements. |
| e) If the user has entered incorrect credentials, then they should be told that what they’ve entered is incorrect | This objective was fully met – if the user enters incorrect credentials, then a message box appears telling them that the sign in request was unsuccessful. |
| f) When someone has logged in, they should get the corresponding menu depending on whether they’re a teacher or student | This objective was fully met – when logging in, the server responds back when whether the credentials were correct and whether the user is a teacher or student. This is then handled and the respective main menu appears. |
| 1. a) If a teacher has logged in, they should be shown a main menu with the options to create a new quiz; open a draft quiz; view the quizzes that they have currently set to their classes as well as their quizzes set history; the quizzes they have yet to mark, view and amend the classes they’ve set up and to view all public quizzes. | This objective was fully met in principle – all of the options that are relevant to using all features of the program for the teacher are displayed, however as seen in the testing evidence I did change the exact entries of the menu to streamline the program a bit more and make it more simplistic to use. |
| b) If a student has logged in, then they should be shown a main menu with the options to view currently set quizzes; review their quiz history; request to join a class and to view all public quizzes. | This objective was also fully met in principle, with the change here being that currently set quizzes and quiz history was all incorporated into one feature rather than two separate ones. |
| 1. a) A teacher should be able to create a quiz with as many questions as they’d like, custom timing, whether it’s an instant result quiz or not, different types of answering and custom questions. | This objective was fully met. As seen by my testing evidence, I did limit the question amount to 50 in order to be safe with byte buffers, however for most quizzes that’s more than enough and so I believe that this objective has been fully met. |
| b) Once the teacher has created a quiz, they should be able to send it to their students and/or be given the option to upload it to the public database of quizzes. | This objective was fully met. When the teacher has finished the quiz, if they decide to make it not public, they are given the option to send it to a class, where they can type in the ID of the class they wish to send it to. If it’s a public quiz (as set in the settings screen), then it is sent to the database as a public quiz. |
| c) If a teacher hasn’t finished a quiz, they should be given the option to save as a draft | This objective was fully met – the Quizzes table’s Draft column stores a Boolean which works perfectly. The teacher is given this option to save as a draft after they are given the opportunity to sent it to a class in a non-public quiz. |
| 1. The teacher should be able to open their draft quizzes, and then edit them or delete them. | This objective was fully met, with the teacher being able to see all of their draft quizzes, with a select button underneath which makes a dialog box pop up with the options to take the quiz, edit the quiz, send it to a class or delete it. |
| 1. a) When taking a quiz, as soon as the quiz has started, the timer should start (displayed in the top right corner of the screen) and the user should be taken through the quiz one question at a time, with the question and corresponding answer type (such as multiple choice, text input box etc.). The current question number is also given in the top left corner alongside how many questions there are in total. When the user has finished a question, they can press “next” to go onto the next question. They are also given the option to go back to the previous question to amend their answer. | This objective was fully met. Some limitations on quiz taking were added as seen in the test evidence, such as the 150 character limit on answering a question. Each question’s relevant question type is displayed properly: for string inputs, the user is given a normal text box to enter in; for integers the same text box is displayed but only integers are allowed to be entered in and for multiple choice a check box system was implemented. Once the timer ticks down to 0 seconds, the quiz finishes regardless of whether the user had finished taking the quiz. |
| b) The user can quit at any point during the quiz with the “Quit” button, where they’re given a dialog asking whether they’re sure they want to quit or not. | This objective was fully met. When the user clicks “Quit”, a dialog box shows asking them whether they are sure they want to quit the quiz. If they click yes, then they are taken back to their respective main menu (based on user type), but if not then they go back to taking the quiz as normal. |
| c) When the user has pressed “Next” on the final question, they should be given a dialog asking if they are sure that they’re finished, and if they press “Yes”, then if it’s an instant quiz, their mark is displayed with their position on the leaderboard, and their mark is sent to the teacher. If it’s not, they’re taken back to the menu that they came back from and their answers are sent to the teacher to mark. | This objective was fully met in principle. Instead of “Next” showing on the final question like on any other question, instead the button says “Finish Quiz”, which makes the user more aware that they have just done the final question. I also wanted to show the leaderboard for non-instant quizzes, as even non-instant quizzes may be competitive in nature, and as a result I made the design choice to make both instant and non-instant quizzes go to the leaderboard. |
| 1. A student should be able to check if they have any homework that has been set by the teacher, and if they do, they should be able to take the quiz | This objective was fully met, with the aforementioned “SetQuizzes” amendment to the design of the program. Like any other quiz, the student is able to see the quizzes that the teacher has set them and is able to take them normally. |
| 1. a) A student or teacher should be able to browse and search the database of public quizzes | This objective was fully met, with each quiz having its own entry on the list of public quizzes. If a teacher selects a quiz, they get a dialog box with the options to edit, send to class, take the quiz or delete it, however due to it being a public quiz the program rejects deleting public quizzes. If the student selects it, they take it like any other quiz. The search function also works completely as intended, with a search only showing the relevant results. |
| b) When a student clicks on a public quiz, if it’s an instant quiz then they should be shown a UI with the leaderboard and a “Play” button, where pressing “Play” should take them to the quiz. | This objective was not met, with the leaderboard function only showing at the end of quizzes. However, the student is able to click on the quiz and take it like any other student. |
| c) When a teacher clicks on a public quiz, they should be given the options to “Play”, “Edit” or “Send to class”. If they press play, then they should play the quiz normally, if they press “Edit” a copy of the quiz can be modified by the teacher as if it were their own to put into their drafts or to send out to their students, and if they press “Send to class”, they are given a dialog as to which class they wish to send it out to. | This objective was met, with each function working exactly as intended. The only changes to this are that the wording of “Play” is changed to “Take quiz” to more explicitly show the function of the button, and due to the dialog box for quizzes being universal the “Delete” button also appears, however for public quizzes this only returns a message box saying that you cannot delete public quizzes. |
| 1. a) A teacher should be able to view the quizzes with answers that they have yet to mark | This objective was met, with the marking list showing each quiz attempt and a button to mark it. However, this page also includes the quizzes that the teacher has already marked, which is an advantage as the teacher can amend marks that they’ve already made if they made a mistake. |
| b) The teacher should be able to click on the quiz that they wish to mark the answers to and be given a list of the questions and the students’ answers. Once they have gone through it, they should be able to give the student a mark, which is sent to the student. | This objective was met, with a slight design change that the teacher is taken through the quiz attempt one question at a time, and given a checkbox on whether the answer given was correct or not. The total tally at the end is automatically added up. |
| 1. a) A teacher should see their set quiz history | This objective was met, but as already mentioned currently set quizzes and quiz history was merged all into one page. |
| b) A student should see their quiz history with the amount of marks they received for it, and the ability to retake any quizzes | This objective was met, once again with the feature that set quizzes and quiz history all being merged into one. It’s easily distinguishable whether the user has taken the quiz or not, as the marks given will show as “N/A” if the user has not taken the quiz yet. |
| c) A teacher when looking at their quiz history should be able to export the student results as a CSV | This objective proposed by Mr Frost was fully met, with the columns “Username” and “Percentage” giving a record of who took the quiz and what percentage they got. |
| 1. a) A teacher should able to create classes, as well as amend any classes that they have | This objective was fully met, with the teacher given a list of the classes that they have ownership of. If they click “Manage” on the class, then they are able to see the list of members of that class and remove any that they wish. They are also able to delete classes here by clicking the “Delete” button. |
| b) A student should also be able to request to join a class with the unique ID given to the teacher | This objective was fully met, with the user being able to request to join an existing class. The teacher is then able to see this class request and is able to accept, making that student a member of this class. |

## Ways to improve the solution as a whole

The main way to improve the solution as a whole would be to add universal robustness to every page. Although some pages feature this well, many pages are not able to handle faulty responses from the server, and as a result if the server side is not responding or is responding with the wrong thing, the client side just crashes. This would be done by incorporating more “try/catch” statements which are made to handle this exact kind of thing.

Another way to improve the solution as a whole would be to more carefully consider buffer sizes. Although the buffer sizes client side are sufficient for the testing, undoubtedly with a larger user base these buffers would easily be overrun, resulting in faulty responses, crashing of the program, or even possibly buffer overflow attacks.

Another way to improve the solution would be to incorporate the feature proposed in 7b where the student is given the leaderboard for instant quizzes before they decide to take it. This would most likely increase the competitive nature of the program as proposed by Alexander Henderson, making quiz taking more fun.