NEA COMPUTER SCIENCE PROJECT:  
DIGITAL QUIZ SYSTEM

Isaac Patrickson

Candidate No. 7507

Centre No. 39445

Whitburn Church of England Academy

Contents

**Analysis**

[**Project problem definition** 2](#_Toc100741840)

[**Background to the problem** 2](#_Toc100741841)

[**Interview with Helleni Tsentas – A Representative of the Primary Client** 3](#_Toc100741842)

[**Observation of Current Methods** 5](#_Toc100741843)

[**Flowchart of the researched quiz creating method** 7](#_Toc100741844)

[**Flow chart of the researched quiz taking method** 8](#_Toc100741845)

[**Prospective Users and Acceptable Limitations** 9](#_Toc100741846)

[**Evidence of current method** 10](#_Toc100741847)

[**Requirements** 16](#_Toc100741848)

[**Objectives** 16](#_Toc100741849)

[**Examples of online quiz services** 17](#_Toc100741850)

[**Modelling inspiration from example online quiz service** 19](#_Toc100741851)

[**Modelling of a typical online quiz service** 20](#_Toc100741852)

[**Tasks to be Computerized and the benefits of Computerizing** 21](#_Toc100741853)

[**Inputs, Outputs, Processing and Storage** 22](#_Toc100741854)

[**System Flowcharts** 23](#_Toc100741855)

[**User Interface** 30](#_Toc100741856)

[**Justification of Included Question Types** 31](#_Toc100741857)

[**Database Normalization and Design** 32](#_Toc100741858)

[**My system’s table relationship diagram** 33](#_Toc100741859)

[**Justification for using databases** 34](#_Toc100741860)

[**Sample of Possible SQL Queries** 35](#_Toc100741861)

[**Security** 36](#_Toc100741862)

[**Temporary Storage with OOP** 36](#_Toc100741863)

[**Algorithm Design** 37](#_Toc100741864)

[**Database Tables** 39](#_Toc100741865)

[**Database Connection** 42](#_Toc100741866)

[**Hashing Algorithm** 43](#_Toc100741867)

[**Annotated Code** 44](#_Toc100741868)

[**Completeness** 70](#_Toc100741869)

[**Test Plan** 71](#_Toc100741870)

[**Testing Evidence** 80](#_Toc100741871)

[**Testing Video** 100](#_Toc100741872)

[**How well does the project meet its requirements? - Improvements** 101](#_Toc100741873)

[**Independent Feedback – Helleni Tsentas (Client)** 102](#_Toc100741874)

[**Discuss the Independent Feedback** 103](#_Toc100741875)

**Analysis**

# **Project problem definition**

Client: Helleni Tsentas and other people who enjoy partaking in online quizzes for leisure.

The Problem: To research methods of how quizzes are developed and conducted online, and to design a purpose-built system which optimizes the process of creating, taking, and monitoring quizzes. The system should be user-friendly, and less time consuming when conducting a quiz.

# **Background to the problem**

During the Lockdown of 2020, quizzes became a popular pastime to those confined to their own homes. Using the video communications app “Zoom”, families and friends could virtually meet up and enjoy each other’s company. During these video calls, members of the call could host and take part in virtual pub quizzes as a form of entertainment.

These quizzes were displayed through Zoom’s screenshare function. This allowed the participants to see exactly what the host wanted them to see. Since there was no purpose-built software to handle quizzes, a quiz host would create quizzes on PowerPoints, Word documents and sometimes paper. This meant that the act of creating a quiz was not well optimised and was quite time consuming for the host.

The most popular method of participating in these quizzes relied on each participant writing their answers as a text message or on a piece of paper. The problem with this, is that once their answer was submitted, it could have easily been changed by the time the answers were read out. As well as this, the final submission of the answers via paper involved the participant sending a picture to the host. Submitted Answers were often scattered across multiple platforms which made it harder for the host to mark. In conclusion, this method is subject to cheating and can prove itself to be very time consuming when creating and marking.

# **Interview with Helleni Tsentas – A Representative of the Primary Client**

**What was your method for creating quizzes?**

* I decided how many rounds and the topics for each round.
* I would decide how many questions I wanted per round, this would depend on how long I wanted the quiz to be and how many rounds I had selected. For example, if I had lots of rounds I might choose fewer questions per round, so the quiz wouldn’t be too long.
* I would use the internet to research questions for each of my rounds.
* I would then use PowerPoint adding the questions to slides and using the formatting tools to create the look for each round.
* I added each question in the appropriate format. For example, there would be simple text-based questions, image questions where I would use a graphic with a transition to reveal an image slowly, music questions where I would insert an audio file, and for some I would insert video clips.
* After each round of questions, I would repeat the question slides but with the answers to each questions added. Normally I used a reveal function to bring up the answer to each question individually.

**What were the benefits of using this method?**

* Complete flexibility.
* By researching questions and using PowerPoint for the visuals, it allowed creative freedom and complete flexibility.
* Each quiz could be completely bespoke, different rounds, different questions, different format.

**What were the drawbacks of this method?**

* It was very time consuming.
* It took time to find appropriate questions and it took even longer to create the quiz in PowerPoint and do all the formatting.
* Had to mark the quizzes manually.

**Which features would be the most important to you in quiz making software?**

* An easy to use.
* Visually appealing user interface.
* Library of preloaded quiz questions that could be sorted by Topic so I could easily make multiple rounds.
* Option to add your own questions.
* Automatic marking.
* A leader board would be nice as it creates the element of competition and can be used to track progress.

**Which existing features do you find the most useful?**

* Leader board.
* Auto marking.
* Stores quiz question.

**How would you like the participants of the quizzes to answer the created questions?**

* Via an app on their phones or on a computer.

**Do you think that a separate, purpose-built digital system is ideal for quiz creating/taking?**

* Yes, a purpose built ‘Quiz Builder’ system would certainly make the process of creating quizzes much easier and quicker.
* It would allow you to create a quiz at short notice. It took me days to create a quiz with PowerPoint.

# **Observation of Current Methods**

As well as recalling from past personal experience, I observed the current methods used by Lecturers, teachers, and family members to get a better understanding of how participants interact with and create online quizzes and identify the problems they face when doing so.

Due to privacy issues, the screenshots below are examples rather than a real scenario.

**The most popular technique is as follows:**

1. Questions are accumulated over a brief period
   1. The Host is likely to have a notebook and/or digital note keeping app in which questions are recorded throughout the period.
   2. In most cases, family members or co-hosts of the quiz contribute to the process by submitting question through text or word-of-mouth.
   3. This means questions were often spread across multiple platforms which can increase confusion for the Quiz Host when the quiz creating process begins.
2. Questions are then read out in a video call
   1. The quiz host sets up a video call where participants can join
   2. Depending on the video calling software, calls can be limited. When using the video calling software “Zoom,” calls would be restricted to 40 minutes long unless the host pays the subscription fee
   3. This means that some participants might have to download/pay for software. If participants are not technically minded this could take effort
   4. The host or participants could drop out the call if there Wi-Fi connection is not stable.
3. Questions are answered and submitted through multiple methods
   1. After the host has read out the question, the participants answer by writing it down in a notebook or sending the answer to the host via text message
   2. Answers are usually submitted at the end and not after each question. Although this minimizes cheating, it would be very time consuming to check the answers of each participant after each question. This would slow the pace of the quiz and reduce enjoyment. When answers are submitted at the end, they are either sent to the host to mark them, or self-marked if the host reads the answers out.
   3. This method of marking all questions for all participants at the end can prove to be quite the workload and can leave the marking prone to mistakes due to the host trying to mark as quickly as possible. As for the other method, this produces the highest chance of cheating. Participants can change their answer to the correct one last minute to increase their chances of winning. This method purely relies on trusting each participant to be fair and a strict marker.
4. After questions have been marked, scores are tallied up and a winner is announced
   1. A number system (1st, 2nd, 3rd, 4th, etc.) is used to rank each participant
   2. If this were a quiz set in a lesson by a teacher during online school, ranking each student manually would consume valuable time
   3. Unless the participant records it themselves, there is usually no record of how many quizzes they have won or how many questions they have answered correctly
   4. This can cause the participant to lose interest in future quizzes as their efforts are not put towards anything
   5. This also means if this is used for educational purposes, there would be no true representation of the student’s ability in the area that quiz covered

# **Flowchart of the researched quiz creating method**

Diagram

Description automatically generated

This flowchart models a generic system of creating and hosting online quizzes for recreational purposes. Most of the processes in this diagram do not rely on technology and therefore human error can arise. This can offer a lot more freedom when it comes to question types, points, and announcements. However, it can often create a larger workload for the host when carrying out these tasks.

# **Flow chart of the researched quiz taking method**

Diagram

Description automatically generated

This flowchart models a generic system for joining and taking part in an online quiz, hosted via an online video calling software. Most of the processing in this diagram also don’t rely on technology, so the same issue of human error is present.

When it comes to designing my program, I will not be incorporating any video calling software. However, there are no restrictions for the user, and I don’t see why my program and video calling software cannot be used simultaneously.

# **Prospective Users and Acceptable Limitations**

This project is aimed at various groups and demographics, teachers, students, and quiz enjoyers. The primary user for this program will be people who enjoy taking part in online quizzes for leisure. Most online quiz takers, unless they are just discovering this pass-time, are computer literate which means their skills should not limit the complexity of the program. However, if a user is less technically minded this should not affect the program as ease-of-use and validation will be a priority.

Although students and teachers are not my primary clients, they could be affected if teachers decide to use the software to set homework or as a short warm-up task, so the question types should be like those seen in multiple-choice questions or short 1-mark exam questions. If the questions were longer, students could lose interest or struggle to recall substantial amounts of information.

**The limitations to the system are as follows:**

* Hardware and software constraints – Most online quizzes that occurred over lockdown took place on a family computer. Certain devices such as older laptops or phones were rarely used. However, my program will utilize databases created with MySQL. This means MySQL must be installed on any device which wishes to use this service. Helleni Tsentas is in possession of a modern functioning laptop which has the capacity to install MySQL.
* My Skills and knowledge – I have no experience in MySQL or how to connect MySQL to python. This is something I will have to teach myself so I can reach this project’s aims. However, I do have extensive knowledge in programming with python so once connections have been made, I will be able to manipulate them into whatever form I’d like. As for the user, they would not need any prior coding knowledge in python or MySQL to operate the program. Only the knowledge to install MySQL is required.
* Quiz constraints – To make these quizzes dissimilar to full exam papers, I am limiting each question to be either multiple-choice or open-ended. Open-ended answers are more flexible, however each word must match for the answer to be correct, so short answers are advised. These short, snappy questions complement the ‘Pub Quiz’ element of the software.
* Time constraints – I must complete the program by Easter of 2022

# **Evidence of current method**

Here are some screenshots and photographs of different stages of the current method

All screenshots are examples of PowerPoints or word documents that were displayed through the video communications software ‘Zoom.’

Short quizzes with fewer rounds tend to be simpler and quicker to make. They were usually made in a Microsoft word document

Table

Description automatically generatedLogo, company name

Description automatically generated

This is an example of a picture round. The quiz creator has used screenshots and a table in Microsoft word to display the images.

Each round is declared.

Longer quizzes are more complicated and are more time consuming to develop. They were usually made in PowerPoint.

The more questions a user chooses to have, the more work needs to be done when marking.

If the user wants to make the quiz more visually appealing; images, fonts, animations, and sound can be added. This can give the quiz more character and increase the level fun experienced by the quizzers. However, the drawbacks of this are that adding more content to the quiz, takes more time and effort.

**Graphical user interface, website

Description automatically generated**

Answers can also be revealed after each round. This process relies on the quizzer being trusted to mark their own work fairly.

**Graphical user interface, website

Description automatically generated**

Picture rounds were commonly used to add variety to the quizzes. They were generally easier than the other rounds and could often amuse quizzers with strange visuals.

**Graphical user interface, website

Description automatically generated**

Movie trivia rounds were usually accompanied by video clips. The clips were shown first, and then the question was read out. In theory, video clips in questions sounds like a good idea, but in practise, it was common for a quizzer to miss the clip entirely due to an audio/visual issue or unstable connection to the zoom call.

Clips were stored as files and were linked to the PowerPoint. This gave the quiz host a reliable way of storing and accessing clips. This saved time as it meant the quiz host would not need to search their file explorer for the previously downloaded clips

Text

Description automatically generatedGraphical user interface, website

Description automatically generated

After all rounds are complete and all answers are submitted and marked, the results are read out by the host. If the host has allowed the quizzers to mark their own answers, the quiz host would ask for their total score at the end. If the host must mark, the results would be read from an excel spreadsheet.

A picture containing calendar

Description automatically generated

# **Requirements**

The requirements, requested by the client, that my program needs to fulfil are as follows

* A user interface which is easy to operate and visually appealing
* An account which can save information between sessions. For example, user data, quiz points earned, position on the leaderboard, and quizzes taken
* A database that already contains quizzes for the user to take
* A method of creating quizzes
* A method to delete quizzes in case there is a mistake made in the creation process
* A way to access a leader board of all players who have taken a specific quiz

# **Objectives**

* Install the free MySQL package containing the MySQL workbench
* Learn the fundamentals of the SQL
* Create a MySQL database which the user can connect to using their MySQL user
* Establish a connection between the computer and a MySQL database
* Design the architecture behind the database (table interconnectivity, primary keys, and foreign keys
* Connect Python 3.9 to the MySQL database and manipulate the database via queries
* Create a system where the user can create a profile and login to an account
* Use a hashing algorithm to encrypt each user’s password incase the tables get hacked and data there is a data leak
* Provide an option to take a quiz where the user can select the quiz they want to take
  + The system must store all the relevant data about each take a user completes, i.e., who took the quiz, which quiz did they take, the score they achieved, the title of the quiz they took.
* Provide an option to create a quiz
  + The user can create three types of quizzes. Multiple-choice, open-ended, or mixed
  + The user can create a set number of questions within the quiz
  + Depending on the question type, the user can create the appropriate answers for each question
  + Any other relevant information about the quiz (quiz host, title, max score, and description) should be stored in the database
* Provide an option to delete a quiz from the database
  + Users can only delete quizzes they have made
* Provide an option to view a user’s score
* Provide an option to view other user’s scores via a leaderboard
* Every user input must be validation to prevent errors in the system and make my program more robust

# **Examples of online quiz services**

As well as the homemade method of creating a quiz for friends and family and displaying them via video streaming software, there are other ways to create and take part in quizzes online.

Online quizzes are a popular form of entertainment and a common type of eLearning. Educational quizzes are one of the most common eLearning patterns for many online courses. Some companies and schools use online quizzes to educate their employees or students. Popular websites for this purpose include Quizlet, Revision Quiz maker and Kahoot!

Many online quizzes are set up to test knowledge or identify a person's attributes. Some companies use online quizzes as an efficient way of assessing a potential hire's knowledge without that candidate needing to travel. Online dating services often use personality quizzes to find a match between similar members.

Most online quizzes are to be taken lightly. The results do not often reflect the true level of understanding in a subject. They are also rarely psychometrically valid. However, they may provide a reflection and function as a springboard for a person to explore ways in which they can improve on a subject.

An example of a popular online quiz service is the game-based learning platform “Kahoot!” It is commonly used in school and other educational institutions. Its learning games, “kahoots,” are user-generated multiple-choice quizzes that can be accessed via a web browser or the Kahoot app. Kahoot! Can be used to review a students’ knowledge, for a formative assessment, or as a break from traditional classroom activities. Kahoot is also a commonplace for trivia quizzes about movies, music, and popular culture.

It was designed for social learning, with learners gathered around a common screen such as an interactive whiteboard, projector, or a computer monitor. The site can also be used through screen-sharing tools such as Skype or Google Hangouts. The gameplay is simple; all players connect using a generated game PIN shown on the common screen and use a device to answer questions created by a teacher, business leader or other quiz creator. These questions can be changed to award points. The creator can choose whether the players can get 0 points, up to 1000 or 2000 points. The points the player gets are calculated on how long it takes them to answer. The sooner they answer, the greater the points if they answer correctly. The player can also get a streak, meaning they answered more than one question correctly in sequence. The longer their streak is, the more points they get when answering a question correctly.

Kahoot can be played through different web browsers and mobile devices through its web interface. It can also be played through the downloadable application version.

Groups of students taking part in a Kahoot displayed on a projector

A group of people sitting in a room looking at a screen

Description automatically generated with medium confidence A group of people looking at a projected image on a wall

Description automatically generated with low confidence

A Kahoot! Creator can use different question types. The first being a form of multiple-choice which requires a question and at least two options, one of which must be marked as the right answer. The second being an open-ended question, which requires a question and one answer. The player must type in the correct answer to get points. A puzzle question type is also available. Puzzle allows the player to align four options in order, which the creator sets as correct. For example: Align the countries by population from the least populated to the most populated.

At the end of the game, there is an animation of the three best players appearing on the winners’ podium. The players can then rate the Kahoot based on their experience.

# **Modelling inspiration from example online quiz service**

Taking inspiration from “Kahoot!” I will be incorporating some similar features in my project.

Features of my Program

* All the points a user earns during a quiz are accumulated at the end and their score for that take is shown.
* A podium will not be shown at the end of the quiz. Instead, a leader board can be viewed through a separate menu whenever the user chooses.
* Question order can be randomized or in order depending on the quiz takers choice.
* Only multiple-choice or open-ended question types will be available.
* The test will not be timed, and extra points will not be rewarded for completing the quiz under a limited time.
* Each question will award the user with one point.
* A players total points accumulated can be viewed through their profile

# **Modelling of a typical online quiz service**

Timeline

Description automatically generated

|  |  |
| --- | --- |
|  | Many-to-one |
|  | One-to-many |

If I choose to solve the problem using a database, the diagram above identifies the relationships between each table. A User can do many takes of many quizzes. Each quiz can contain many questions and each question can have many answers. Each time a user takes a quiz, they can submit many answers during that take.

# **Tasks to be Computerized and the benefits of Computerizing**

**Tasks to be Computerized**

* Creating a quiz
* Taking a quiz
* Quiz deletion
* Keeping a record of results on a particular quiz
* Keeping a record of the user’s score

**Qualitative Benefits**

* Creating the quiz follows the same formula each time, so no confusion surrounding the process is created
* A more user-friendly interface
* Each step will be instructed, so even people who are unfamiliar with computers can create a quiz
* When wanting to randomize question order, it is significantly easier to let a program do it for you than to do it manually

**Quantitative Benefits**

* It makes it possible to have infinite participants. Theoretically, if the system had enough storage, an infinite number of people could make an account
* Theoretically, without the limitations of storage, users could take an infinite number of quizzes and create an infinite number of quizzes

**Design**

# **Inputs, Outputs, Processing and Storage**

This table outlines what happens to the data put into my system at the most basic level.

|  |  |
| --- | --- |
| **Input** | **Process** |
| User   * First Name * Last Name * Email * Password   Quiz   * Title * Type * Number of questions * Description   Quiz Question   * Difficulty level * Type * Score * Content   Quiz Answer   * Correct * Content   Take   * Select Take ID   Take Answer   * Enter you answer | Creating a User or Logging In   * Check email has not been used before. If it hasn’t, create the account. If it has, inform the user * If the email hasn’t been used and all other fields of input are valid, create a salt for the user. Use this salt and their password to hash their password * Permanently store the user’s data in the “User” table in the Database and then log the user in * If the User Decides to log in, their email and password are required * If the email entered is in the database table, retrieve that users salt. Use the entered password and the salt to hash the password. If this results in the same value stored in the table, then the correct password has been entered. * Log the user in   Creating a Quiz or Taking a Quiz   * If the user wants to create a quiz, ask for a Title, the type of quiz (the user will select one of 3: Multiple-choice, Open-ended, or Mixed), the number of questions and a brief description of the quiz. * Then for the number of questions they wished to create (maximum 50), ask for the: * Difficulty level (1 easy, 3 hard) * Type would be set to either 1 (multiple-choice) or 2 (Open-ended) unless its mixed. If the quiz is mixed, the user will be able to choose each question type as they go along. * The score will be set to 1 as I will be using this to count the max score and count the number of questions in the quiz * The question contents * The questions’ correct answer. If the question is multiple-choice, the question will require wrong answers too. * When taking a quiz, inputs for answers will be required from the user and their input will be compared against the correct answer. If they are the same, then the participant has answered the question correctly |
| **Storage** | **Output** |
| * A Database schema called “quiz” * User table * Quiz table * Quiz Question table * Quiz Answer table * Take table * Take Answer table | * Python Shell Window |

# **System Flowcharts**

The following charts show how different sections of my project operate and interact with the database. Although the flowcharts are denser than that of the current system, most operations are automated, and the processes are less time consuming than the manual alternative.

**Login Menu**

Diagram

Description automatically generated

**Main Menu**

Diagram

Description automatically generated

**Take a Quiz Menu**

Diagram

Description automatically generated

**Create a quiz menu**

Diagram

Description automatically generated

**Modify quiz menu**

Diagram

Description automatically generated

**View profile**

Diagram

Description automatically generated

**Leader boards menu**

Diagram

Description automatically generated

# **User Interface**

The user interface for my program will be displayed in the python shell. The screenshots below show the menus I created and some small sections of the program working. I programmed the menus first so I could have a brief checklist of different functionalities. This checklist helped me prioritize different sections of the program and helped me choose my starting point.

Text

Description automatically generated

This is the opening menu the user is presented with once they run the program. The user has a choice of “1”, “2” or “0”.

“1” – Prompts the user to enter their login details as shown (the example email and password used are “s@” and “s”)

“0” – Closes the program

Text

Description automatically generated

“2” – Prompts the user to enter their details for their account creation. These details will be saved in the database.

The example data entered is “Isaac”, “Patrickson”, “isaac@patrickson.com”, “4a5secure9password1”

All the data meets the requirements for the database, so the account has been created. The user would then be taken to the Main menu after this process.

Text

Description automatically generated

This is the quiz selection screen where every quiz in the database is displayed on screen. For the screenshot, there are two quizzes in the database. If there were one hundred quizzes, they would all be displayed one after the other. A disadvantage of displaying quizzes like this would be that they would fly up the page and would require the user to scroll to find the ID of their desired quiz. This is impractical.

Each quiz has its important details displayed alongside it.

If this screen was done with Tkinter, there would be a scroll bar in a frame.

Text

Description automatically generated

The quiz with ID “25” has been selected. The user is then asked if they are sure they want to take this quiz. “Y” – Proceeds to the quiz

“N” – Takes the user back to the Take Quiz menu

The user is asked if they would like to randomise the question order. One that choice has been made, the quizzes title and description are displayed.

Each question of the quiz is displayed one at a time. The system only prints the next question if the current question has been answered.

In this case, option “3” has been selected which is choice 18

# **Justification of Included Question Types**

Below are the types of questions which appeared during my research of online quizzes completed over lockdown and my justification for including or not including them in the design of the new system. My design is dependent on which one’s I choose to develop into my project.

|  |  |  |
| --- | --- | --- |
| **Question type** | **To be included?** | **Justification** |
| Open-ended | Yes | They are one of the most common types of pub quiz question and can be applied to every topic |
| Multiple-choice | Yes | They’re a common format of online quiz question and can be answered quickly. |
| Audio file | No | It is possible to store audio files, but it is more common to store their file location or metadata. I will not be doing this because it would require the user to download image files in a specific folder each time, they would want to create a question containing an audio file. |
| Video file | No | It is possible to store a video file’s file location as a reference but like audio files, they would need to be inserted into a specific file location for that reference to be consistent. As well as this, vides could go on forever unless a max file size is incorporated. Having these necessary features could cause frustration among quiz creators as files could be inserted incorrectly and then could be too large due to high quality. Putting effort into this question type that barely came up when doing my research wouldn’t be sensible as there are a lot of variables that wouldn’t under the programs control. |
| Picture file | No | I will be planning to use a MySQL database to store each question. Although possible to store images in the database, it is not advisable, and it isn’t general practice. A general practice would be to store images in directories on the system and store the references to the images in the database. These references would have to stay constant for the stored image otherwise it could cause errors. |

# **Database Normalization and Design**

If my program uses a database, it will need to normalize it to avoid inconsistencies and the duplication of data. It will also save space by eliminating non-atomic data.

These are the tables of my database in third normal form

**Bold** – Primary key

user

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **id** | firstName | lastName | email | salt | passwordHash | profile |
|  |  |  |  |  |  |  |

quiz

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **id** | hostId | title | type | maxScore | content |
|  |  |  |  |  |  |

quiz\_question

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **id** | quizId | type | level | score | content |
|  |  |  |  |  |  |

quiz\_answer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | quizId | questionId | correct | content |
|  |  |  |  |  |

take

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | userId | quizId | score | content |
|  |  |  |  |  |

take\_answer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | takeId | questionId | answerId | content |
|  |  |  |  |  |

# **My system’s table relationship diagram**

This diagram illustrates the relationships between the normalized tables in my program. We can tell the tables have been fully normalized as there are no many-to-many relationships.

Unlike the previous table relationship diagram for a typical quiz service, I am not including date and time. I did not deem this feature necessary as I do not want my system to be used for deadline tasks such as homework. As stated previously, this program is for people who enjoy partaking in quizzes for leisure.

Diagram

Description automatically generated

# **Justification for using databases**

I am using a MySQL database so I can permanently store essential information across different sessions. This is important because each user is linked to every quiz they have made and every quiz they have taken.

Not storing this data permanently means that between sessions, the user’s data, and any actions they performed would be erased. I need this data to be saved between session, so users can login, view previously created quizzes, and view leader boards.

If I used a text file to store the data, there would be no structures for indexing or recognizing relationships between records. This would make handling the stored data impossible.

An advantage of a MySQL database would be that multiple users could connect to the database simultaneously. If the user has allowed remote access to the database, other computers from around the world can connect. This allows players to create quizzes for each other, build a community and compete for the top spot on the leader board.

# **Sample of Possible SQL Queries**

|  |
| --- |
| **Adding a new user to the database** |
| A picture containing diagram  Description automatically generated |

|  |
| --- |
| **Displaying the user’s full name** |
|  |

|  |
| --- |
| **Adding a user’s take to the database** |
|  |

|  |
| --- |
| **Updating the user’s take with their score once they have finished the take** |
|  |

|  |
| --- |
| **Displaying a user’s profile score** |
|  |

|  |
| --- |
| **Deleting a quiz from the database** |
|  |

|  |
| --- |
| **Displaying the questions in the quiz** |
|  |

|  |
| --- |
| **Displaying the answers for a multiple-choice question** |
|  |

|  |
| --- |
| **Displaying users in a leader board for a specific quiz** |
|  |

|  |
| --- |
| **Displaying all quizzes** |
|  |

# **Security**

I will be using a hashing algorithm to encrypt user passwords. This is to protect user data.

If the user’s password is the same password for other services, if the database is hacked, their password could be used on more important services such as their email or bank.

This hashing algorithm will create a random string of characters for each user. This will be the user’s “salt”. This salt and the user’s password will be combined and hashed using python’s hash library. The final hashed password will be stored in the user’s table alongside the user’s records.

To login to that user’s account, the user’s unique email and password must be entered. Once the email is entered, the corresponding salt is retrieved. The password entered is then hashed with the retrieved salt. If this hashed password matches the stored hashed password, the user is granted access.

To access the database, the user must enter their MySQL username and password. Only users who know the username and password for the MySQL account can proceed to the game. The user can create multiple MySQL users which have access to this database however I will be using the root user when testing my project.

To protect the integrity of the store data, all data entry will be controlled by strict validation rules. Since all entries will be free text input, I will be doing validation for every input. This will eliminate typographic errors which may cause the system to crash.

# **Temporary Storage with OOP**

**Object Oriented Programming**

Before data is stored in the database, it will be stored in object variables.

Each variable will represent the data which will be inserted into each column.

This data is set by the user through the “setter” functions.

That data is then added to the database.

But if the data of an object variable needs to be retrieved and put into another variable, the “getter” function can be used.

For example.

The variable “QuizId” is present in both the quiz and the quizzes’ questions.

To assign this “QuizId” to its questions, I can get the “QuizId” from the quiz class and store it in the question class.



This is a simpler method than retrieving the “QuizId” from the database each time.

Graphical user interface, application

Description automatically generated

# **Algorithm Design**

**Random Ordering of Multiple-Choice Answers**

|  |
| --- |
| **Explanation** |
| Every multiple-choice question needs to have a different order of answers each time a quiz is taken, i.e., the first option shown cannot always be the correct answer.  This algorithm randomizes the indexes of the array of four answers.  One correct answer, and one/three incorrect answers depending on the multiple-choice question |
| **Code** |
|  |

**User’s Quizzes Search**

|  |
| --- |
| **Explanation** |
| Users should be able to search for their own quizzes in case they want to delete them. Quizzes might need to be deleted due to errors in answers or spelling mistakes within the quiz details. A similar algorithm is used for searching for all quizzes. |
| **Code** |
|  |

**Validation of Text Free Entries**

|  |
| --- |
| **Explanation** |
| Entries need to be controlled by strict validation rules to minimize crashing. This validation will also filter the data before it goes into the database. If a record’s datatype is integer but a string is entered, an error will occur.  The algorithm below will check if the user’s Quiz Id input is available to access. It will also check if they would like to go back to the main menu.  If this wasn’t validated, the database might try and retrieve a quiz with a non-existing Quiz Id (which cannot be done.) |
| **Code** |
|  |

Technical Solution

# **Database Tables**

Creating the schema ‘quiz’ so the tables have somewhere to reside

**CREATE** **SCHEMA** `quiz` **DEFAULT** **CHARACTER** **SET** utf8mb4 **COLLATE** utf8mb4\_unicode\_ci**;**

Creating the User table

* All columns except ‘profile’ need to be able to store variable characters (both integers and strings)
* ‘salt’ and ‘passwordHash’ need to have a larger capacity for the encrypted passwords
* ‘profile’ is an integer as it stores the user’s total points accumulated

**CREATE** **TABLE** `quiz`**.**`**user**` **(**

`id` BIGINT **NOT** **NULL** AUTO\_INCREMENT**,**

`firstName` VARCHAR**(**50**)** **NULL** **DEFAULT** **NULL,**

`lastName` VARCHAR**(**50**)** **NULL** **DEFAULT** **NULL,**

`email` VARCHAR**(**50**)** **NULL,**

`salt` VARCHAR**(**512**)** **NOT** **NULL,**

`passwordHash` VARCHAR**(**512**)** **NOT** **NULL,**

`profile` INT**(**15**)** **NOT** **NULL** **DEFAULT** 0**,**

**PRIMARY** **KEY** **(**`id`**),**

**UNIQUE** **INDEX** `uq\_email` **(**`email` **ASC)** **);**

**CREATE** **TABLE** `quiz`**.**`quiz` **(**

`id` BIGINT **NOT** **NULL** AUTO\_INCREMENT**,**

`hostId` BIGINT **NOT** **NULL,**

`title` VARCHAR**(**75**)** **NOT** **NULL,**

`**type**` SMALLINT**(**6**)** **NOT** **NULL** **DEFAULT** 0**,**

‘id’ will be set as the primary key for each table

`maxScore` SMALLINT**(**6**)** **NOT** **NULL** **DEFAULT** 0**,**

`content` TEXT **NULL** **DEFAULT** **NULL,**

**PRIMARY** **KEY** **(**`id`**),**

‘hostId’ is being used as a foreign key. If someone tried to delete the user, they would have to delete all records linked to the host’s ID first

**INDEX** `idx\_quiz\_host` **(**`hostId` **ASC),**

**CONSTRAINT** `fk\_quiz\_host`

**FOREIGN** **KEY** **(**`hostId`**)**

**REFERENCES** `quiz`**.**`**user**` **(**`id`**)**

**ON** **DELETE** **NO** **ACTION**

**ON** **UPDATE** **NO** **ACTION);**

Setting ‘hostId’ as a foreign key

**CREATE** **TABLE** `quiz`**.**`quiz\_question` **(**

If ‘hostId’ were to be deleted or updated, deny the action

`id` BIGINT **NOT** **NULL** AUTO\_INCREMENT**,**

`quizId` BIGINT **NOT** **NULL,**

`**type**` VARCHAR**(**50**)** **NOT** **NULL,**

`**level**` SMALLINT**(**6**)** **NOT** **NULL** **DEFAULT** 0**,**

When a quiz is created, if the fields with ‘NOT NULL’ are left empty, the quiz will not be created.

`score` SMALLINT**(**6**)** **NOT** **NULL** **DEFAULT** 0**,**

`content` TEXT **NULL** **DEFAULT** **NULL,**

**PRIMARY** **KEY** **(**`id`**),**

**INDEX** `idx\_question\_quiz` **(**`quizId` **ASC),**

**CONSTRAINT** `fk\_question\_quiz`

**FOREIGN** **KEY** **(**`quizId`**)**

**REFERENCES** `quiz`**.**`quiz` **(**`id`**)**

**ON** **DELETE** **NO** **ACTION**

**ON** **UPDATE** **NO** **ACTION)**

ENGINE **=** InnoDB**;**

**CREATE** **TABLE** `quiz`**.**`quiz\_answer` **(**

‘id’ must be a big integer (an integer which can take up 8 bytes of storage). This is because, theoretically there could be many users, quizzes, questions, answers, etc.

`id` BIGINT **NOT** **NULL** AUTO\_INCREMENT**,**

`quizId` BIGINT **NOT** **NULL,**

`questionId` BIGINT **NOT** **NULL,**

`correct` TINYINT**(**1**)** **NOT** **NULL** **DEFAULT** 0**,**

`content` TEXT **NULL** **DEFAULT** **NULL,**

**PRIMARY** **KEY** **(**`id`**),**

‘correct’ only needs to be a tiny integer as the value only needs to be ‘1’ or ‘0’

**INDEX** `idx\_answer\_quiz` **(**`quizId` **ASC),**

**CONSTRAINT** `fk\_answer\_quiz`

**FOREIGN** **KEY** **(**`quizId`**)**

**REFERENCES** `quiz`**.**`quiz` **(**`id`**)**

I am using ‘InnoDB’ as my storage engine

The ‘InnoDB’ storage engine maintains its own buffer pool that caches table and index data in main memory as data is accessed. This cache speeds up processing.

**ON** **DELETE** **NO** **ACTION**

**ON** **UPDATE** **NO** **ACTION)**

ENGINE **=** InnoDB**;**

**ALTER** **TABLE** `quiz`**.**`quiz\_answer`

**ADD** **INDEX** `idx\_answer\_question` **(**`questionId` **ASC);**

**ALTER** **TABLE** `quiz`**.**`quiz\_answer`

**ADD** **CONSTRAINT** `fk\_answer\_question`

**FOREIGN** **KEY** **(**`questionId`**)**

**REFERENCES** `quiz`**.**`quiz\_question` **(**`id`**)**

**ON** **DELETE** **NO** **ACTION**

**ON** **UPDATE** **NO** **ACTION;**

**CREATE** **TABLE** `quiz`**.**`take` **(**

`id` BIGINT **NOT** **NULL** AUTO\_INCREMENT**,**

`userId` BIGINT **NOT** **NULL,**

`quizId` BIGINT **NOT** **NULL,**

`score` SMALLINT**(**6**)** **NOT** **NULL** **DEFAULT** 0**,**

`content` TEXT **NULL** **DEFAULT** **NULL,**

**PRIMARY** **KEY** **(**`id`**),**

**INDEX** `idx\_take\_user` **(**`userId` **ASC),**

**CONSTRAINT** `fk\_take\_user`

**FOREIGN** **KEY** **(**`userId`**)**

**REFERENCES** `quiz`**.**`**user**` **(**`id`**)**

I made alterations to the ‘quiz’ table and the ‘take’ table. I made a new index called ‘idx\_take\_quiz’ which is linked to the ‘quizId’ of the taken quiz.

Setting ‘quizId’ as the foreign key in the ‘take’ table.

If the quiz is to be deleted, every take with its quizId must be deleted first.

If the quiz where to change its ID, all takes of that quiz must change first.

If these delete or update actions are attempted without doing the instructions stated above, the quiz will not delete

**ON** **DELETE** **NO** **ACTION**

**ON** **UPDATE** **NO** **ACTION);**

**ALTER** **TABLE** `quiz`**.**`take`

**ADD** **INDEX** `idx\_take\_quiz` **(**`quizId` **ASC);**

**ALTER** **TABLE** `quiz`**.**`take`

**ADD** **CONSTRAINT** `fk\_take\_quiz`

**FOREIGN** **KEY** **(**`quizId`**)**

**REFERENCES** `quiz`**.**`quiz` **(**`id`**)**

**ON** **DELETE** **NO** **ACTION**

**ON** **UPDATE** **NO** **ACTION;**

**CREATE** **TABLE** `quiz`**.**`take\_answer` **(**

I made alterations to the take answer table

* Set foreign keys
* Set constraints

These constraints are vital to maintaining the integrity of the data within the database.

Without them, data could be updated or deleted, leaving other records possibly linked to nothing.

This unlinked data could clog up the system and cause errors if accessed

`id` BIGINT **NOT** **NULL** AUTO\_INCREMENT**,**

`takeId` BIGINT **NOT** **NULL,**

`questionId` BIGINT **NOT** **NULL,**

`answerId` BIGINT **NOT** **NULL,**

`content` TEXT **NULL** **DEFAULT** **NULL,**

**PRIMARY** **KEY** **(**`id`**),**

**INDEX** `idx\_answer\_take` **(**`takeId` **ASC),**

**CONSTRAINT** `fk\_answer\_take`

**FOREIGN** **KEY** **(**`takeId`**)**

**REFERENCES** `quiz`**.**`take` **(**`id`**)**

**ON** **DELETE** **NO** **ACTION**

**ON** **UPDATE** **NO** **ACTION)**

ENGINE **=** InnoDB**;**

**ALTER** **TABLE** `quiz`**.**`take\_answer`

**ADD** **INDEX** `idx\_tanswer\_question` **(**`questionId` **ASC);**

**ALTER** **TABLE** `quiz`**.**`take\_answer`

**ADD** **CONSTRAINT** `fk\_tanswer\_question`

**FOREIGN** **KEY** **(**`questionId`**)**

**REFERENCES** `quiz`**.**`quiz\_question` **(**`id`**)**

**ON** **DELETE** **NO** **ACTION**

**ON** **UPDATE** **NO** **ACTION;**

**ALTER** **TABLE** `quiz`**.**`take\_answer`

**ADD** **INDEX** `idx\_tanswer\_answer` **(**`answerId` **ASC);**

**ALTER** **TABLE** `quiz`**.**`take\_answer`

**ADD** **CONSTRAINT** `fk\_tanswer\_answer`

**FOREIGN** **KEY** **(**`answerId`**)**

**REFERENCES** `quiz`**.**`quiz\_answer` **(**`id`**)**

**ON** **DELETE** **NO** **ACTION**

**ON** **UPDATE** **NO** **ACTION;**

# **Database Connection**

To connect python to the MySQL database I needed to import the MySQL connector. This library was automatically installed to pip when I downloaded the MySQL package.

The library allows me to make a connection through my MySQL account which has access to the ‘quiz’ database. As well as making a connection, it allows me to make a cursor. This cursor is a tool which allows me to update, delete and apply other MySQL commands to my database through python.

**import** mysql**.**connector

This import retrieves the ‘errorcode’ function within ‘mysql.connector’ library. This function allows for exception handling when dealing with database errors. Errors can be formatted using the ‘.format(err)’ command. The error message will appear in the form of an error code followed by a brief explanation on why the error occurred.

**from** mysql**.**connector **import** errorcode

Something went wrong**:** 1045 **(**28000**):** Access denied **for** user 'fnd'**@**'localhost' **(**using password**:** YES**)**

**The algorithm for logging into a user’s MySQL account**

# This while loop will ask the user to enter their details of their MySQL user so they can connect to the database 'quiz'

**while** **True:**

**print()**

**global** cnxUser

**global** cnxPassword

cnxUser **=** **input(**"Enter your MySQL username: "**)**

cnxPassword **=** **input(**"Enter your MySQL password: "**)**

**try:**

cnx **=** mysql**.**connector**.**connect**(**user**=**cnxUser**,** password**=**cnxPassword**,**

database**=**'quiz'**)**

cursor **=** cnx**.**cursor**()**

**break**

# If the user's details are incorrect the error will be handled with

exception handling

**except** mysql**.**connector**.**Error **as** err**:**

**print()**

**print(**"Something went wrong: {}"**.format(**err**))**

Exception handling in case the entered username and password are incorrect

quiz\_connection**(**cnxUser**,** cnxPassword**)**

quiz\_answer\_connection**(**cnxUser**,** cnxPassword**)**

quiz\_question\_connection**(**cnxUser**,** cnxPassword**)**

sql\_executor\_connection**(**cnxUser**,** cnxPassword**)**

take\_connection**(**cnxUser**,** cnxPassword**)**

‘cnxUser’ and ‘cnxPassword’ are made global so they can be used in establishing the connections for all the other imported python files.

Every database change throughout the program is done through the MySQL user signed in at the start

take\_answer\_connection**(**cnxUser**,** cnxPassword**)**

user\_connection**(**cnxUser**,** cnxPassword**)**

# **Hashing Algorithm**

**The hashing algorithm I used to encrypt each user’s password**

I imported python’s inbuilt hash library, binascii (converts between binary and ASCII) and os is used to generate random bytes.

**import** hashlib**,** binascii**,** os

**def** salt**():**

salt **=** hashlib**.**sha256**(**os**.**urandom**(**60**)).**hexdigest**()**

**return** salt

I created a salt for each user using a random list of 60 bytes and sha-256. ‘Hexdigest’ returns the encoded data in hexadecimal format.

**def** hash\_password**(**password**,** salt**):**

salt **=** salt**.**encode**(**'ascii'**)**

passwordHash **=** hashlib**.**pbkdf2\_hmac**(**'sha512'**,** password**.**encode**(**'utf-8'**),**

salt**,** 100000**)**

The salt and the entered password are both used to create the hashed password

passwordHash **=** binascii**.**hexlify**(**passwordHash**)**

**return(**salt**+**passwordHash**).**decode**(**'ascii'**)**

**def** check\_password**(**salt**,** storedPasswordHash**,** enteredPassword**):**

passwordHash **=** hashlib**.**pbkdf2\_hmac**(**'sha512'**,** enteredPassword**.**encode**(**'utf-

8'**),**

salt**.**encode**(**'ascii'**),** 100000**)**

passwordHash **=** binascii**.**hexlify**(**passwordHash**).**decode**(**'ascii'**)**

**return** passwordHash **==** storedPasswordHash

The entered login password and the salt are used to create a hashed password. If this hashed password matches the one stored in the user’s account, this function returns True

# salt = salt()

# print(salt)

# storedPasswordHash = hash\_password(input('Set your password: '), salt)

# print(check\_password(storedPasswordHash, input('Enter your password: ')))

# Compares the entered password with the stored one

# **Annotated Code**

**Main.py**

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Application

Description automatically generated with low confidence

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

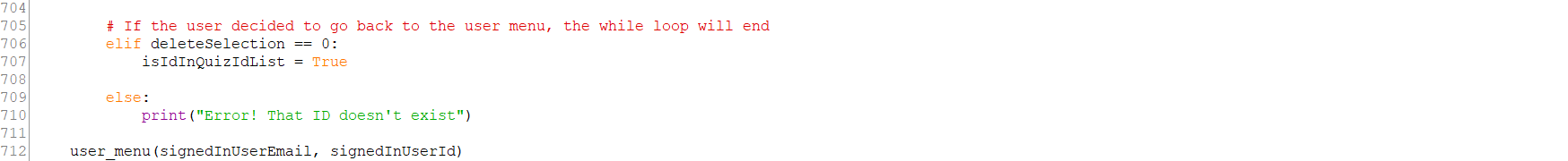
Description automatically generated

Text

Description automatically generated

Text

Description automatically generated



Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Hashing\_Algorithm.py**

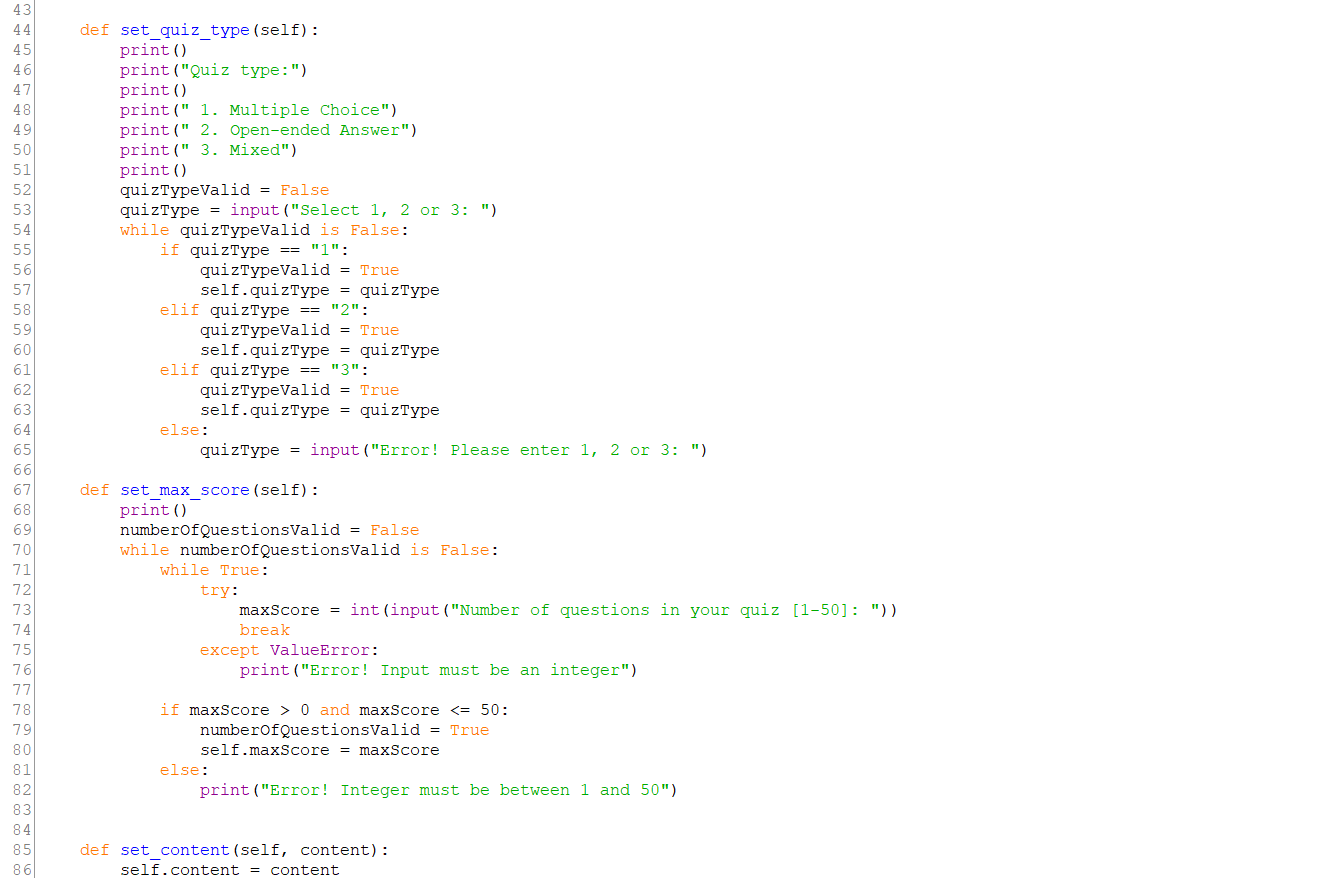
Text

Description automatically generated

**Quiz.py**

A picture containing text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Quiz\_Answer.py**

Text

Description automatically generated with medium confidence

Text

Description automatically generated

**Quiz\_Question.py**

A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**SQL\_Query\_Executor.py**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

A picture containing scatter chart

Description automatically generated

**Take.py**

A picture containing table

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Take\_Answer.py**

A picture containing table

Description automatically generated

Text

Description automatically generated

**User.py**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

# **Completeness**

Although I met all my objectives, an additional objective could have been set when considering the graphical user interface.

The client stated that a more attractive user interface would make the system easier to operate and more fun to play.

I would have liked to learn how to use python’s inbuilt GUI library ‘Tkinter’. By utilizing Tkinter, I could have developed an attractive graphical user interface which could encourage more quizzers to use the system.

Creating a GUI with Tkinter would have refined some features of my system.

* Instead of text moving up and off the screen (if the screen fills up) it would be displayed in a box with a scroll bar.
* ‘Go Back’ buttons would remove the need to type in ‘0’ to return to the previous screen
* Less validation would be needed as some inputs would be tied to buttons.
* The title of each screen would be at the top-middle of the screen instead of at the bottom.
* More attractive colours could have been used. The limitations of the shell are essentially stripped away.
* Error messages could have appeared in a message box, notifying the user instead of just telling them through text.
* Images, videos, and sounds could have been part of question contents.

Testing

# **Test Plan**

To ensure that all the system’s algorithms function correctly, and that all user inputs (correct and incorrect) are handled, I have planned to test the system.

To meet my objectives

* All inputs must be validated and not cause crashes
* All inputs that go into the database must be successfully added to the database

**Test Plan Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Test Data** | **Expected Results** |
| **Testing that a user can sign in with their MySQL user, to connect to the database** | | | |
| 01 | Check the user can enter the correct details to connect to the database. Correct Inputs. | root  Isaac250mysql! | * The user should be accepted into the database * User taken to the login menu. |
| 02 | Check the user can enter the correct details to connect to the database. Incorrect Inputs. | Groot  Pineapple300 | * The user is denied access to the database and the system. * The user is shown an error message and prompted to enter the details again. |
| 03 | Check the user can enter the correct details to connect to the database. Empty field is entered. | [None]  Isaac250mysql! | * The user is denied access to the database and the system. * The user is shown an error message and prompted to enter the details again. |
| **Testing that the user can enter a value corresponding to an action at the login menu** | | | |
| 04 | Check the user has entered either “1”, “2”, or “0” | “1” | * The user is taken through to the login process |
| 05 | Check the user has entered either “1”, “2”, or “0” | “2” | * The user is taken through to the sign-up process |
| 06 | Check the user has entered either “1”, “2”, or “0” | “0” | * The user exits the program, and a message box pops up * Message box asks “Your program is still running! Do you want to kill it?” * If “yes” is selected, the shell window closes * If “cancel” is selected, the shell window remains open but the program ends |

|  |  |  |  |
| --- | --- | --- | --- |
| 07 | Check the user has entered either “1”, “2”, or “0” | “F” | * The user is prompted to type either “1”, “2” or “0” again |
| 08 | Check the user has entered either “1”, “2”, or “0” | [None] | * The user is prompted to type either “1”, “2” or “0” again |
| **Testing that the user can login to the system** | | | |
| 09 | Check that the email and password are valid.  Correct inputs | [Make sure the user with email and password “test@” and “test” has been created on the system beforehand]  test@  testpassword | * The user is greeted by a welcome message * “Welcome, [first name + last name] * The user is then taken to the main menu |
| 10 | Check that the email and password are valid.  Incorrect data entered (no email) | [None] | * The user is told that there are no accounts with that email * The user is taken back to the login menu |
| 11 | Check that the email and password are valid.  Incorrect data entered (no password) | test@  [None] | * The user is told that that is the incorrect password for that email address * The user is taken back to the login menu |
| 12 | Check that the email and password are valid.  Incorrect data entered (email does not have “@” symbol) | test | * The user is told that the email must contain an “@” symbol * The user is prompted to enter the email again |
| **Testing that a new user can register on the system** | | | |
| 13 | Check the user can register an account.  Correct Inputs | Barry  White  Barry@White.com  Barrysecurepassword56! | * The user is told their account has been created * They are taken to the main menu |
| 14 | Check if all fields are entered | [None]  White  Barry@White.com  Barrysecurepassword56! | * If the username is blank, the user is told that only letters a-z are allowed * They are then prompted to enter the field in again * This is the same for the last name * The user is told that the email must contain an “@” symbol if the email is left blank * The user is prompted to enter the email again * The password must be at least 8 characters * The user is prompted to enter the password again |
| 15 | Check if fields meet length requirements | A first name, last name, or email longer than 50 characters | * The system provides an error message which tells the user to enter data which is less than 50 characters long * The user is prompted to input data again |
| 16 | Check if fields meet length requirements | [A password with length 7 or under]  Cake23 | * The system provides an error message which tells the user to enter a password which has a minimum of 8 characters * The user is prompted to input data again |
| **Testing that a user can take a quiz** | | | |
| 17 | Check that the user is provided with options of all the created quizzes | [The user has selected the take quiz option from the main menu]  1 | * The user is presented with every quiz in the quiz table |
| 18 | Check the user can select a quiz. Correct ID entered | [Assuming there is already a quiz created with the ID “4”]  4 | * The user is asked if they are sure they want to take the quiz |
| 19 | Check if the user wishes to go back to the main menu (If the quiz ID “0” is entered)  Correct Inputs | 0 | * The user is returned to the main menu |
| 20 | Check the user can select a quiz. Not an ID entered | [Quiz IDs can only be integers]  @sdasd’q2 | * The user is told that this isn’t an ID |
| 21 | Check the user can select a quiz. Incorrect ID entered | [Assuming there is no quiz with the ID 89]  89 | * The user is told that the ID entered is not in the list of available quizzes |
| 22 | Check the user is sure they want to take the quiz.  Correct inputs | [Not case sensitive]  Y | * The user is then asked if they want to randomize the question order |
| 23 | Check the user is sure they want to take the quiz. Correct inputs | [Not case sensitive]  N | * The user is taken back to the take quiz menu * All the available quizzes are displayed again |
| 24 | Check the user is sure they want to take the quiz. Incorrect inputs | [None] or not Y or not N  None | * The user is prompted again to enter either Y or N until Y or N is entered |
| 25 | Check if the user wants to randomize question order. Correct inputs | [Not case sensitive]  Y | * Question order is randomized * The quiz begins and the first question is displayed |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 26 | Check if the user wants to randomize question order. Correct inputs | | [Not case sensitive]  N | | | * The quiz begins and the first question is displayed |
| 27 | Check if the user wants to randomize question order. Incorrect inputs | | [None] or not Y or not N  None | | | * The user is prompted again to enter either Y or N until Y or N is entered |
| 28 | Check that each question in the selected quiz is displayed and the user has answered each one | | User inputs for each question | | | * Each question is displayed one at a time * The user must enter their answer before the next question is displayed |
| 29 | Check if the question is multiple-choice (MC), the options [either 2 or 4] are displayed in a random order | | User inputs for each question | | | * Each MC question has its choices displayed in a random order each take |
| 30 | Check if the user’s input for selecting a choice (for an MC question) is valid. Correct inputs | | [Answer is option 3]  3 | | | * The user is presented with a message that says “correct” * One point is added to their score * The next question is displayed |
| 31 | Check if the user’s input for selecting a choice (for an MC question) is valid. Incorrect inputs | | [Answer is option 1]  1 | | | * The user is presented with a message that says “incorrect” * The user’s score stays the same * The next question is displayed |
| 32 | Check if the user’s input for selecting a choice (for an MC question) is valid. Incorrect inputs | | The input is not 1,2,3 or 4  D56gsfe##  54 | | | * If the user has entered a non-integer, they are told to enter an integer and asked for the input again * If the user has entered an integer but not one of the integers preceding an option, they are asked to enter an integer preceding an option. They are asked for the input again |
| 33 | Check if the open-ended (OE) question is valid.  Correct inputs | | [Answer is “Blue Whale”]  [Not case sensitive]  Blue Whale | | | * The user is told that they are correct * The user’s score increases by one * The next question is displayed (or quiz ends if it was the last question) |
| 34 | Check if the open-ended (OE) question is valid.  Incorrect inputs | | [Answer is “Blue Whale”]  [Not case sensitive]  Not Blue Whale  [The incorrect answer could be anything]  Moby Dick | | | * The user is told that they are incorrect * The user’s score stays the same * The next question is displayed (or quiz ends if it was the last question) |
| 35 | Check when the quiz is finished.  Correct Inputs | | | Inputs for all the questions in the quiz | * The user is told the quiz is finished * Their score for the quiz is displayed * Their new, increased profile quiz points is also displayed * All data collected from the take has been recorded and stored in the take table and the take\_answer table * The user is then taken back to the Main Menu | |
| **Testing that a user can create a quiz** | | | | | | |
| 36 | Check that the user is provided an option to create a quiz.  Correct Inputs | | | [The user has selected the create quiz option from the main menu]  2 | * The user is put through the quiz making process * This starts with the user being asked to enter the quiz title | |
| 37 | Check the entered quiz title is valid.  Correct inputs | | | [a string of characters less than 75 characters]  General knowledge | * The title is saved to an object variable * The user is then asked what type their quiz is | |
| 38 | Check the entered quiz title is valid.  Incorrect inputs | | | [a string of characters more than 75 characters] | * Error message saying the title is too long * User is prompted to enter the title again | |
| 39 | Check the entered quiz type is valid.  Correct inputs | | | 1, 2 or 3 | * 1 – MC type quiz * 2 – OE type quiz * 3 – Mixed type quiz * Type is saved to an object variable * The user is then asked what the max score of their quiz will be | |
| 40 | Check the entered quiz type is valid.  Incorrect inputs | | | Not 1, not 2 or not 3  34 | * Error, please enter 1, 2 or 3 * User prompted to enter the quiz type again | |
| 41 | Check the entered Max Score is valid.  Correct Inputs | | | [A value between 1 to 50]  2 | * The max score is saved to an object variable * The user is then asked to create a question the number of times equal to the max score * Since each question is always worth one point, the max score is equivalent to the number of questions per quiz * The user is then asked to enter the quiz description | |
| 42 | Check the entered Max Score is valid.  Incorrect inputs | | | [A value outside 1 to 50 or a non-integer]  100  hello | * Error, integer must be between 1 and 50 * Error, input must be an integer * The user is prompted to enter the max score again | |
| 43 | Check the quiz description | | | [Quiz description can be anything]  A quite interesting quiz | * The quiz description is saved to an object variable * Every variable is now filled so the quiz can now be added to the database * The user is then prompted to create the questions | |
| **Testing that a user can create a question** | | | | | | |
| 44 | Check the entered question difficulty is valid. Correct inputs | | | 1, 2 or 3 | * This process repeats for a number which is equal to max score * 1 – Easy * 2 – Medium * 3 – Hard * The difficulty level is stored in an object variable * The user is then asked to enter the question content | |
| 45 | Check the entered question difficulty is valid. Incorrect inputs | | | Not 1, not 2 or not 3  Hard | * Error, input 1, 2 or 3 * The user is prompted to enter the difficulty level again | |
| 46 | Check the entered question content.  Correct inputs | | | [Question content can be anything] | * The question content is stored in an object variable * All question object variables are satisfied so the quiz can be added to the database * The user is then asked to create an answer for this question * MC – create multiple answers * OE – create one answer | |
| **Test that a user can create an answer** | | | | | | |
| 47 | | If OE check the answer content for one input  If MC check the answer input for all the choices. Either 2 or 4  Correct inputs | | [Answer content can be anything] | * If OE, the one answer inputted is set as the correct answer * The OE answer content is stored to an object variable * The OE answer is added to the database * The system asks for either 2 or 4 inputs for answer contents. * The first input will always be the correct answer * The rest are incorrect * The answer contents and their corresponding correct values are stored to an object variable * They are then added to the database * The quiz creation process is then over * The user is returned to the main menu | |
| **Test that a user can delete a quiz** | | | | | | |
| 48 | | Check that the user is provided with options of the created quizzes they have made.  Correct Inputs | | [The user has selected the delete quiz option from the main menu]  3 | * The user is presented with the quizzes they have made, retrieved from the quiz table | |
| 49 | | Check the user can select a quiz to delete. Correct ID entered | | [Assuming there is already a quiz created with the ID “50”]  50 | * The user is asked if they are sure they want to delete the quiz | |
| 50 | | Check if the user wishes to go back to the main menu (If the quiz ID “0” is entered) | | 0 | * The user is returned to the main menu | |
| 51 | | Check the user can select a quiz. Not an ID entered | | [Quiz IDs can only be integers]  @sdasd’q2 | * The user is told that this isn’t an ID | |
| 52 | | Check the user can select a quiz.  Incorrect ID entered | | [Assuming there is no quiz with the ID 64]  64 | * The user is told that the ID entered is not in the list of available quizzes | |
| 53 | | Check the user is sure they want to delete the quiz.  Correct inputs | | [Not case sensitive]  Y | * All the takes of the quiz and the quiz itself are deleted | |
| 54 | | Check the user is sure they want to delete the quiz.  Correct Inputs | | [Not case sensitive]  N | * The user is taken back to the delete quiz menu * All the available quizzes are displayed again | |
| 55 | | Check the user is sure they want to delete the quiz.  Incorrect inputs | | [None] or not Y or not N  YES! | * The user is prompted again to enter either Y or N until Y or N is entered | |
| **Test that a user can view their own score** | | | | | | |
| 56 | | Check that the user can view their own quiz points  Correct inputs | | [The user has selected the Profile option from the main menu]  4 | * The user is presented with their total quiz points earned * They are then given the option to return to the main menu | |
| 57 | | Check that the user wishes to go back to the main menu.  Correct inputs | | 0 | * The user has been returned to the main menu | |
| 58 | | Check that the user wishes to go back to the main menu.  Incorrect inputs | | Not 0  Go back please | * The user is prompted with the option to go back to the main menu again | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test that a user can view another user’s scores through a leader board** | | | |
| 59 | Check that the user can view another user’s score through a leader board  Correct inputs | [The user has selected the Leader boards option from the main menu]  5 | * The user is brought to the leader board menu * The user is presented with an option to view an overall leader board, view a leader board per quiz or go back to the main menu |
| 60 | Check that the user wants to view the overall leader board. Correct inputs | 1 | * A leader board with the top 50 players ranked from best to worst will be displayed * The user is then given the option to go back to the leader board menu (0. Go back) |
| 61 | Check that the user wants to view the leader boards per quiz. Correct inputs | 2 | * All quizzes are displayed * The user must select a quiz ID or select “0” to go back |
| 62 | Check that the user wants to go back to the main menu.  Correct inputs | 0 | * The user is returned to the main menu |
| 63 | Check that the user wants to make a valid decision at the leader board menu  Incorrect inputs | Not 1, not 2, or not 0  49 | * The user is prompted to enter 1, 2 or 0 again |
| 64 | Check if the user wants to go back to the leader board menu from the overall leader board. Correct inputs | 0 | * The user is returned to the leader board menu |
| 65 | Check if the user wants to go back to the leader board menu from the overall leader board. Incorrect inputs | Not 0  Sw3d3 | * The user is prompted to enter the go back input again |
| **Testing that a user can view leader boards per quiz** | | | |
| 66 | Check that the user wants to view the leader boards per quiz. Correct inputs | 2 | * All quizzes are displayed * The user must select a quiz ID or select “0” to go back |
| 67 | Check that the user wants to view the leader boards per quiz. Incorrect inputs | [Not 2]  567ytyh\* | * The user is prompted to enter their input again * Enter either 1,2 or 0 |
| 68 | Check the user can select a quiz from which they can view a leader board from. Correct ID entered | [Assuming there is already a quiz created with the ID “48”]  48 | * The user is shown the top 50 takes for the selected quiz * They are then asked if they would like to return to the leader board menu |
| 69 | Check the user can select a quiz. Not an ID entered | [Quiz IDs can only be integers]  @sdasd’q2 | * The user is told that this isn’t an ID |
| 70 | Check the user can select a quiz from which they can view a leader board from. Incorrect ID entered | [Assuming there is no quiz with the ID 89]  89 | * The user is told that the ID entered is not in the list of available quizzes |
| 71 | Check if the user wishes to go back to the main menu from the leader board menu (If the quiz ID “0” is entered)  Correct inputs | 0 | * The user is returned to the main menu |

# **Testing Evidence**

The following table is an overview of the basic input and output test conducted on the system. This is to make sure that the expected outcome of every action is the actual outcome when using the system.

|  |  |
| --- | --- |
| **Test ID** | **Actual Outcome** |
| **Testing that a user can sign in with their MySQL user, to connect to the database** | |
| 01 |  |
| 02 |  |
| 03 |  |
| **Testing that the user can enter a value corresponding to an action at the login menu** | |
| 04 |  |
| 05 |  |
| 06 |  |
| 07 |  |
| 08 |  |

|  |  |
| --- | --- |
| **Testing that the user can login to the system** | |
| 09 |  |
| 10 |  |
| 11 |  |
| 12 |  |

|  |  |
| --- | --- |
| **Testing that a new user can register on the system** | |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |

|  |  |
| --- | --- |
| **Testing that a user can take a quiz** | |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |

|  |  |
| --- | --- |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 |  |
| 31 |  |
| 32 |  |
| 33 |  |
| 34 |  |
| 35 |  |
| **Testing that a user can create a quiz** | |
| 36 |  |
| 37 |  |
| 38 |  |
| 39 |  |
| 40 |  |
| 41 |  |
| 42 |  |
| 43 |  |
| **Testing that a user can create a question** | |
| 44 |  |
| 45 |  |
| 46 |  |
| **Test that a user can create an answer** | |
| 47 |  |
| **Test that the user can delete a quiz** | |
| 48 |  |
| 49 |  |
| 50 |  |
| 51 |  |
| 52 |  |
| 53 |  |
| 54 |  |
| 55 |  |
| **Test that a user can view their own score** | |
| 56 |  |
| 57 |  |
| 58 |  |

|  |  |
| --- | --- |
| **Test that a user can view another user’s scores through a leader board** | |
| 59 |  |
| 60 |  |
| 61 |  |
| 62 |  |
| 63 |  |
| 64 |  |
| 65 |  |
| **Testing that a user can view leader boards per quiz** | |
| 66 |  |
| 67 |  |
| 68 |  |
| 69 |  |
| 70 |  |
| 71 |  |

# **Testing Video**

[Demonstrating the robustness of the solution](https://youtu.be/fmsjf4a4IUk)

In the video, I have demonstrated that every user input is fully validated before it gets processed. I did this by entering in normal data, erroneous data, and boundary data.

Evaluation

# **How well does the project meet its requirements? - Improvements**

I believe my program fulfils the requirements set by the client. The system allows users to create, take and track progress of quizzes by permanently storing data within database tables. In testing, this proved a reliable way to store data for retrieval later. Manipulation of this data through queries, made it possible to insert, update, select, and delete records. These queries form the basis of every function in my program and allow the user to interact with the database.

For the user to create an account, an email needs to be entered. This email is unique, and in conjunction with a password, provides a means of logging in once the account has been created. The validation requirements for the entered email are that it must contain the ‘@’ symbol, no spaces are allowed, and the email can be no longer than 50 characters. However, there is no validation to check whether the email exists or not – this can be dealt with as a future improvement. Another future improvement could consist of the user verifying their email address by clicking a link on a received email sent by the program.

Having an account stored in the database, is vital to the programs key functions. Each quiz created, needs to be linked with the user who created it. Similarly, each take the user does, needs to be traced back to the user so they can be displayed on leader boards. This is done through foreign keys and cross table parameterisation. In testing, this proved to be a viable way of linking users with their activity on the system. User feedback suggested that there should be an option to delete your user account and remove all records containing that user. If I were to improve the program in the future, I would consider including this feature.

Each quiz has a max score. This dictated the maximum points a user could earn from completing the quiz. However, it also controlled how many questions were in the quiz as each question was only worth one point. For example, a quiz could have fifty available points and therefore it would contain 50 questions. As a future improvement, I could introduce a new field into the quiz table called “numberOfQuestions” which would separate the points from the number of questions. This could also help me utilise the difficulty field in the question table, tying different question difficulties to different point rewards. For example, easy rewards ten points, medium thirty points, and hard fifty points.

The user interface is run through the python shell. In testing, it offered a simple way of receiving inputs and displaying outputs trough the text entry and display window. However, feedback from the client suggested that the user interface could be easier to navigate. This could be improved by implementing python’s Tkinter library to create an attractive, easy-to-use GUI. Navigation could be improved by using Tkinter buttons. These buttons could take the user to and from different menus un the program.

Another objective that I would have liked to give myself would be to create a function that modifies individual quiz records. This function would allow the user to change a quiz’s title and description as well as each individual question. This could save time, as user would not have to delete the quiz if they wished to make a change. For example, if a user created a quiz with fifty questions and they wanted to change one of them, they would not have to enter the content for all fifty questions again. This could also be applied to the user records; first name and last name.

# **Independent Feedback – Helleni Tsentas (Client)**

I love taking and making quizzes and produced quite a few quizzes during lockdown, so I was excited to try out this new quizzing programme.

The interface is very clear, simple and easy to use, which is a good thing, but I think I would like something more graphical.  I think a graphical interface would make it more engaging and exciting for users.  Not being a computer programmer myself, I'm used to using software and apps that are more visually orientated, so this programme seemed a little old fashioned looking even though it did everything it needed to.

I really like the fact that there was a database of prepared quizzes because in the past it's been very time consuming to create a new quiz, so to have some to hand was great.  I also liked the fact that I could create my own quizzes that would be saved so I could build my bank of quizzes and include different topic-based question.  Also being able to delete quizzes was very useful, because sometimes you can make a mistake with the questions or maybe the topic was no longer relevant, and you don't want these clogging up your database.

Having a function that automatically stored user’s results and even produced a leader board for each quiz was great.  Before this system marking quizzes and then collating the points was time consuming for me, so having instant access to that information was useful. I like the fact that people can use the leader board to check how they are doing, it helps create a little friendly competition and may spur people on to retake quizzes or do more quizzes to improve their ranking.

# **Discuss the Independent Feedback**

When considering different methods of displaying a user interface, I concluded that the python shell would suffice. My client stated that a more appealing interface could encourage more people to play the game. They also mentioned that the average player might not appreciate the work put into the program if it doesn’t look visually appealing. Acting on this, I designed a few windows which demonstrate what the graphical user interface would have looked like if I decided to improve the program in the future.

This user interface would be implemented using the python library “Tkinter.”

Graphical user interface, application

Description automatically generated

This would be the opening window which would be the first thing the user sees once they run the program.

Pressing the “Quit” button would close the program.

This would have been a part of the Login Menu process.

If the user were to press the “Login” button on the opening window, they would be taken to this screen, where they can enter their login details.

If the details typed in the boxes match that of a user in the “User” table, the user would proceed to the main menu. If the details are incorrect, an error message box would pop up and display an error message.

This would have been a part of the Login Menu process.

Graphical user interface

Description automatically generated

Graphical user interface, application

Description automatically generated

If the user were to press the “Sign Up” button on the opening window, they would be taken to this screen, where they can enter their account details.

If the “Create Account” button is then pressed, their details would then be saved to the “User” table in the database, unless there is an error in one or more fields entered i.e., their email does not contain an “@” symbol.

This would have been a part of the Login Menu process.

Graphical user interface, text, application

Description automatically generated

This is an example of when a user is taking part in a quiz.

They are taking part in the quiz titled “Music Quiz” and they are on the first question. This question is a multiple-choice question with four answer choices. The user has selected the fourth choice. The answer will only be submitted once the “submit answer” button has been pressed.

Once the “submit answer” button has been pressed, the screen will display the next question. The user will not be allowed to go back to a previous question.