GROUP ASSESSMENT - QUIZ DOCUMENTATION

Rafael Parian, Angelo Tankeko, Vidisha Sharma

Programming Principles (ICT105)

Anthony O’Shea

Stanley College

West Perth, WA

26 May, 2024

CONTENTS

[Introduction 3](#_Toc168007380)

[Application Overview 3](#_Toc168007381)

[Design Documentation 4](#_Toc168007382)

[Development Environment 8](#_Toc168007383)

[Source Code Structure 8](#_Toc168007384)

[As-Build Functionality 8](#_Toc168007385)

[Database and Data Management 10](#_Toc168007386)

[Testing and Quality Assurance 11](#_Toc168007387)

[Deployment and Configuration 14](#_Toc168007388)

[Security and Compliance 14](#_Toc168007389)

[Maintenance and Support 15](#_Toc168007390)

[Appendices 16](#_Toc168007391)

## Group Assessment – Quiz Documentation

## **Introduction**

The project in question was the creation of a multiple-question quiz using coding. This quiz application was created using the programming language Python in the code editor program, ‘Visual Studio Code.’ The aim of this quiz was to demonstrate our learnings from the programming class, and compile these learnings and subsequent concepts into a functional program. Through the documentation process, we were able to record the steps we took in creating the quiz, and relay the strengths, challenges, and workings of the code. As such, we aim to discuss the objectives, complete developmental process, and outcomes of our endeavour. Here, we will cover the environment for development, code structure, the creation and use of a database, testing procedures, features, and continuous outcomes (feedback-based maintenance) of our quiz.

### **Application Overview**

The application is a quiz, where the user is asked in steps to login or register, then to answer a series of ten multiple-choice questions, which are taken at random from a set database of questions. The questions each have four options as answers, and focus on general knowledge topics. The purpose of this quiz is to determine the level of knowledge a user has (currently, about general knowledge questions, ranging from sports, media, politics, and history) on a set subject. Any type of question can be entered into the question bank, allowing the quiz to have a multidisciplinary testing potential. The quiz takes asks the user ten questions, then displays their result. Following this, the user has the option to take the quiz again. If at any step the user answers with an input which is not a given option, the program redirects the user to the given options to try again (this includes while attempting to login with a previously registered username).

The quiz informs the user after each question, whether the answer they entered is correct or not, and of their final score in percentage, at the end of the quiz. It can be used to track the performance of a user and the growth or changes in their knowledge in comparison to their previous attempts. The target user base for this quiz is currently anyone who wishes to briefly test their general knowledge. Intended use cases include by organizations to get feedback from their employees to determine the level of understanding of the workers after, for example, training. Similarly, it can be used to gauge engagement or gather feedback from employees. In addition to this, a quiz-type application such as ours has applications for testing student knowledge in a fast and efficient manner.

These is a section of the code which allows for a file to be generated, which displays the most recent output information. In this file, the information present is amended as the users change. It displays the most recent username, and the time at which the user logged in. There are some expected improvements for this section described in the maintenance and support section of this report, which explores the possibility of using this file to display the results of users over time, and the overall scores which they may have. This was created using open(“quiz\_summary.txt”), where the name of the created file was “quiz summary,” and a function ‘f.write’. The information was defined and selected to be amended (‘a’) rather than overwritten (‘w’).

## **Design Documentation**

The modules of this application are as follows:

* Database: The application utilizes a SQLite database named "quizapp.db" to store user accounts (usernames and passwords) and quiz questions (questions, options, and answers).
* User Interface: The user interacts with the application through a text-based interface.
* Authentication: The application verifies user credentials (username and password) during the login process.
* Quiz Engine: This module randomly selects questions from the database, presents them to the user along with answer choices, validates user input, and calculates the final score.
* A file is created where the details of the previous user are stored. This includes the date and time at which the user registered their credentials, their username, as well as the scores they achieved during the quiz.

This quiz design has multiple specific features, such as for security in having a step for authentication (registering and logging in), preventing easy cheating by ensuring that the order of questions is shuffled between tests taken, guiding users by looping to the previous step if a non-prescribed value is entered, showing if each inputted answer is correct, as well as showing the overall score to users. Here, users must communicate their understanding by navigating the quiz using keyboard entries.

The process of these features flows in the following sequence:

1. User starts the application.
2. The application displays a menu with options to register, login, or exit.
3. Based on user selection:
   * Register: User enters a username and password. The application creates a new user account in the database.
   * Login: User enters a username and password. The application validates the credentials against the database. Upon successful login, the user can proceed to take the quiz.
   * Exit: The application terminates.
4. If the user chooses to take the quiz, the application:
   * Randomly selects questions from the database.
   * Presents each question and its corresponding answer choices to the user.
   * Validates the user's chosen answer.
   * Determines the final score on the number of correct answers.
   * Displays the final score to the user.
5. The application closes the connection to the database upon exit

This is all illustrated in the flow chart below.

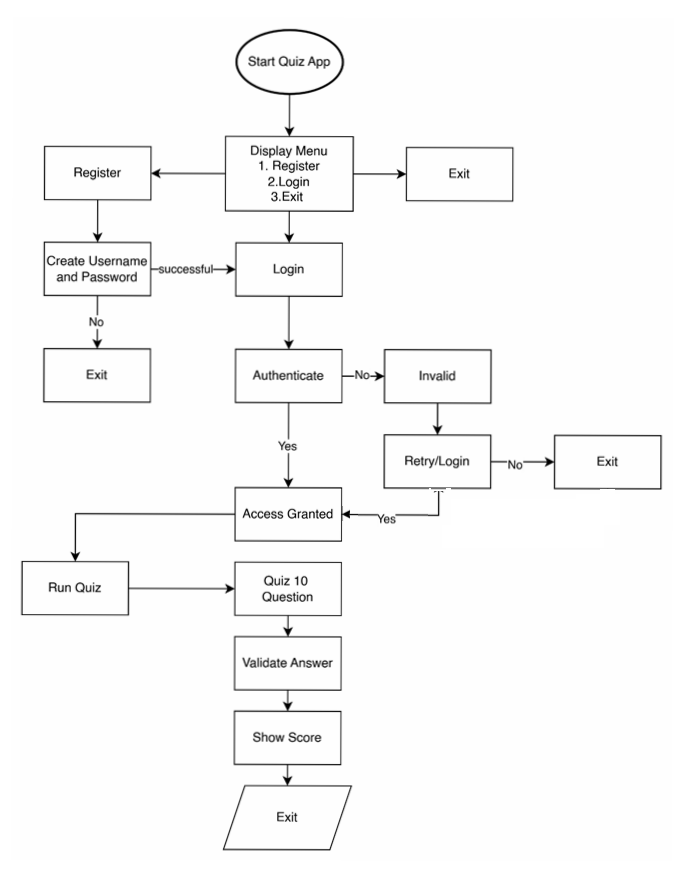


Figure 1 (System Flow Chart)

**Code Description**

The code used to achieve some of the modules listed above are as follows. Initially, we had to ensure that the program was first establishing a connection to the database, and that we were able to connect to it using SQLite. Following this, the changes that we make regarding the database, had to be locked in to allow them to be executed in the database. This was done using the ‘cursor’ variable. Next, we enabled a method for users to register themselves, using a username and a password. These details were made to be saved in the database. We did this registration using “def register\_user(username, password): in line 11 of the code.

As we came to create this registration method, we felt that it was necessary to ensure that no duplicate credentials existed, to avoid confusion while logging in. To achieve this, we used a count function, where the number greater than which the quiz needed to disallow username registration was 0 (if there were more than 0 of the username existing in the database, i.e. 1, the quiz printed that the “Username already exists. Please choose a different username.” Since that would not always be the case, the next lines of code were to register the new or original details (executing and committing that the username and password were inserted into the users table “users\_tbl”. As original credentials have no restrictions, those new values were to successfully register, and display to the user that the registration was successful.

After registering, a user is expected to select the next option, which was to log in to access the quiz. In order to log in, their credentials had to successfully be registered into the database. We ensured this was the case by using the function “def authenticate\_user(username, password):” to check the values against those stored in the users table.

Finally, the user could then begin the quiz. Of course, that means that we had to being to write the quiz itself. In order to be able to have questions which come in a different order each time, or randomized from the bank, we used “def fetch\_random\_question():”. This was ensured to be a question that is extracted once only, by random, from the table. As this occurs, the question is in a way labelled as having been asked already, to avoid being fetched as a duplicate in the same quiz session. That means the question ID is appended into the list of asked questions. If a question is not in the “asked questions,” only then it is available to be asked.

The quiz is run using “def run\_quiz(username):”. The original score is set to 0, and the number of questions that are to be asked is set as 10. The questions are numbered, and the answers are extracted from the database rows and columns which correspond with the extracted question as it is printed. As the questions begin, a loop is created. This loop is as follows: a question is selected and printed along with four answer options, the user has to input an appropriate option (between 1 and 4), and the corresponding answer is checked against the correct answer. This will lead to the input being validated or invalidated, depending on if their answer matches the set correct answer or not. Their score is updated and also logged, accordingly.

The log taken creates a file. That file contains information on the user, their score, and the date and time at which they engaged with the quiz. This can be used to compare the scores of a user against their past scores, and even to compare scores of various users to each other. Since the file is appended and not rewritten as the database grows, these values are comparable to one another.

As the user enters the quiz, the first interface they are shown is a while loop, where they are able to select from three options. These options are to register, to login, or to exit, using numerical input. In order to login, it is necessary to have registered once. Thus, if the user chooses to log in, but does not have valid credentials which are stored in the database, and can be checked by the program, the user is not able to access the quiz. They are instead redirected to register. The user has to input a unique set of credentials register, which will then allow them to login. The quiz then starts using “run\_quiz(username)” and their scores are calculated or tracked along the way. Finally, once the quiz is complete, the user is displayed with a message stating that their quiz is completed, along with their score as a percentage value. Should they choose to, the user can exit the quiz using the word “exit,” or by choosing the third option in the initial login/registration page. If the user enters non-approved values, they are redirected to enter another acceptable value to continue or exit. Finally, the connection with the database is closed.

### **Development Environment**

|  |  |
| --- | --- |
| **Programming Language** | Python 3.12.3 |
| **Internal Libraries** | Import SQLite |
| **Code Editor** | Visual Studio Code |
| **Version Control** | Github |
| **Collaboration Tool and Communication** | Microsoft Teams and Email |
| **Database** | SQLite |

*Development Environment Details*

In order to develop the quiz application, it was essential to utilize an Integrated Development Environment (IDE), version control system, as well as collaboration and communication tools. The developmental environment for this project was predominantly as follows: for the code editor, we used Virtual Studio Code (VS Code), along with the programming language Python. In specific, we used Python 3.12.3 to create the quiz. For the internal libraries, we worked to import SQLite3. In regard to version control, we made use of Github, to sync the changes from the local to the online repositories.

We were able to find our from *Python Software Foundation,* thatSQLITE “is a C library that provides a lightweight disk-based database”, and that it “doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.” (SQLITE3, 2024).

As we were creating this application, the majority of our communication (through which we were able to develop our concepts, ideas, requirements, and determine our steps), was in person after classes, and online using Microsoft Teams and Outlook. As such, we were able to collaborate and determine the segments each were responsible for, the time frame, and the particular elements we wanted in the quiz.

### **Source Code Structure**

To build the quiz application, we used five user-defined functions: ‘register\_user’, ‘authenticate\_user’, ‘fetch\_random\_questions’, ‘run\_quiz’, and ‘index’. Each of these functions served a distinct purpose within the overall functionality of the application, as outlined below:

|  |  |
| --- | --- |
| User Define Function | Description |
| Register\_User() | This function facilitates user registration for the quiz application. Should users lack existing login credentials, they may employ this feature to establish a username and password, thereby gaining access to the application's features. |
| Authenticate\_User() | This function verifies whether the username and password entered by a user are registered within the users table (users\_tbl) of the quizapp database. |
| Fetch\_random\_question() | This function ensures that questions will not be repeated. |
| Run\_quiz() | This function serves as the core component of the quiz application. It facilitates setting the number of questions to be asked, retrieving each question along with four possible options and the correct answer from the QuizApp database, displaying them, validating user responses for accuracy, and calculating the user's percentage score accordingly. |
| Index() | This function serves as the primary interface for the quiz application, presenting users with three distinct options. The first option allows users to register a new account. The second option is designated for returning users to log in to their existing accounts. The third and final option enables users to exit the quiz application. |

*Main Functions of the Application*

The quiz application initiates by presenting the user with three options: 1. Register, 2. Login, and 3. Exit. The user is required to select only from the numerical choices (1, 2, or 3). Should the user input any other number or text, an error message will be displayed stating, "Invalid Choice. Please enter a valid option." The application will continue to prompt the user until a valid selection is made.

### **As-Build Functionality**

The quiz application we developed is fundamentally aligned with a client who requires a testing mechanism’s initial requirements. Nevertheless, we have successfully integrated additional functionalities to significantly enhance both user experience and overall performance of the quiz. Outlined below are the comprehensive functionalities of our quiz applications that we have developed and implemented.

|  |  |
| --- | --- |
| **Functionality** | **Description** |
| Database | Our system utilizes SQLITE for database management and incorporates the creation of two essential tables. The first table, named 'quiz\_tbl,' is dedicated to housing all quiz-related data, including questions, four possible answer options per question, and the correct answers. The second table, termed 'user\_tbl,' is designed to store user information such as user IDs, usernames, and passwords. |
| User Login | Prior to accessing the quiz, users must authenticate their identity by logging in with a valid username and password. |
| Random questions | Our system employs an SQLITE database, allowing us to incorporate a substantial number of questions. By leveraging the `total\_questions` parameter, we can control and determine the number of questions presented in any given quiz session. Additionally, the random question generation function guarantees that each query fetched from the database is unique and not repeated within the same session. As a result, every time a user logs in to take a quiz, they will encounter a distinct set of questions. |
| User registration | In the event that a user does not possess valid login credentials, the application offers a registration module allowing them to create a username and password. |
| Validation and Error handling | The system mandates that users select a valid option from both the index page and the quiz itself. To ensure compliance, we have incorporated a validation feature: if an invalid option is chosen, an error message will prompt the user to re-enter a valid selection. This measure guarantees accurate inputs throughout the process. |
| Score Computation in Percentage | The system calculates the total number of correct answers provided by the user and subsequently determines the percentage score. |

*Main Program Modules*

To further demonstrate the function of the application, each step is documented below, from figures 2 to 6, in the form of screenshots directly from the application:

A black screen with white text

Description automatically generated

Figure 2 (Quiz application index page)

*A black screen with white text

Description automatically generated*

Figure 3(Quiz questions and choices)

**

Figure 4 (Quiz application registration)

**

Figure 5 (Quiz application login)

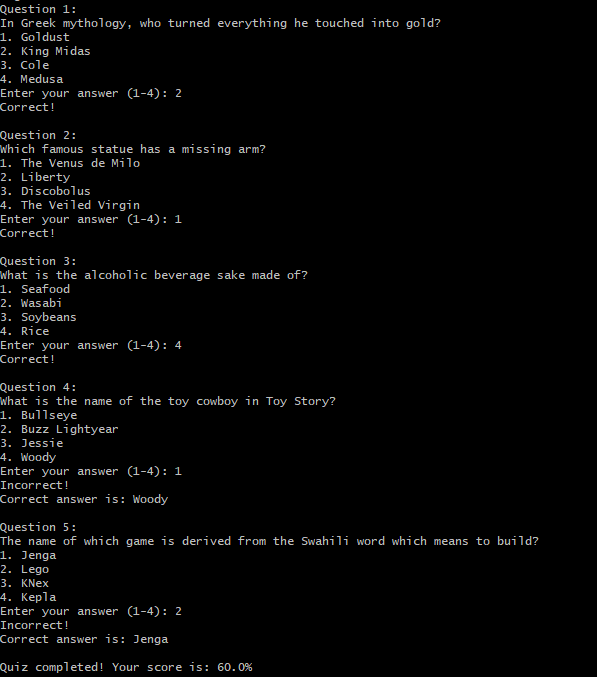
**

Figure 6 (Completed quiz with score)

### **Database and Data Management**

The database structure for the quiz application is designed to be highly efficient and straightforward. It involves the creation of a 'quizapp' database containing two primary tables: 'quiz\_tbl' and 'users\_tbl.'

In the quiz table, (“quiz\_tbl,”) there are fields designated for questions; ‘option1’, ‘option2’, ‘option3’, ‘option4’, and answers. Similarly, the in the users table, (“users\_tbl”) comprises fields for user\_id, user\_name, and password. The core principle behind this design is to store all essential elements such as questions with their respective options and answers alongside user credentials in a structured database format rather than within a list.

This approach ensures that modifications such as additions, deletions or updates to question sets do not necessitate alterations in the codebase itself; instead, these changes can be managed directly from the backend. Consequently, if users need to incorporate new queries into the system this can be seamlessly achieved via backend updates thus enhancing overall flexibility and maintainability of our quiz application.

*A diagram of a quiz app

Description automatically generated*

Figure 7 (Quiz application database structure)

### **Testing and Quality Assurance**

We undertook testing for each section of the code, as demonstrated in this section. Initially, we began with resting the authentication and validation at the beginning of the quiz.

1: User Registration

**Test Case 1:** Register with Valid Data

Test Case ID: TC\_REG\_01

Test Description: Verify user registration with valid data.

Preconditions: None.

The steps we undertook to test the registration, was to initially follow the instructions of the quiz, and to attempt to input alternate values. To guide a user to do the same, we documented the following steps, and provided sections in which the user may document their findings in comparison, as well:

Test Steps:

1. Navigate to the registration page.
2. Choose the "Register" option 1.
3. Enter a valid username and password.

After following these steps, a result that may be expected is as such: the user is registered successfully, and a confirmation message is displayed. User details are stored in the user\_tbl table.

Actual Result:

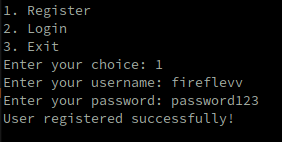


Figure 8 (Test Case 1 result - registration)

**Status:** I found that my username and password registered successfully. It was functional.

**Test Case 2:** Register with Existing Username

Test Case ID: TC\_REG\_02

Test Description: Verify registration fails when the username already exists.

Preconditions: User with the same username already exists in the user\_tbl table.

Test Steps:

1. Navigate to the registration page.
2. Enter an existing username and a valid password.

Expected Result: An error message is displayed indicating that the username already exists.

Actual Result:

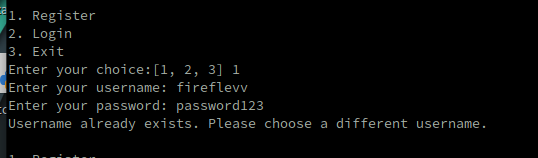


Figure 9 (Test case 2 - invalid credentials registration)

**Status:** Successful: the re-use of the same credentials was rejected, and the user was informed that it already existed.

2: User login

**Test Case 3:** Login with Valid Credentials

Test Case ID: TC\_LOGIN\_01

Test Description: Verify user login with valid credentials.

Preconditions: User is registered with valid username and password in the user\_tbl table.

Test Steps:

1. Navigate to the login page.
2. Enter a valid username and password.

Expected Result: User is logged in successfully and redirected to the quiz page.

Actual Result:

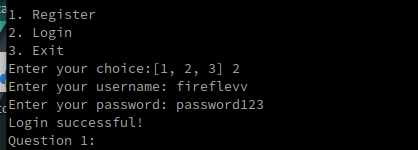


Figure 10 (Test case 3 - login)

**Status:** I was able to login with my registered username and password. I was taken into the quiz immediately.

**Test Case 4:** Login with Invalid Credentials

Test Case ID: TC\_LOGIN\_02

Test Description: Verify user login fails with invalid credentials.

Preconditions: User is registered with valid username and password in the user\_tbl table.

Test Steps:

1. Navigate to the login page.
2. Enter an invalid username or password.
3. Choose the "Login" option 2.

Expected Result: An error message is displayed indicating invalid credentials.

Actual Result:

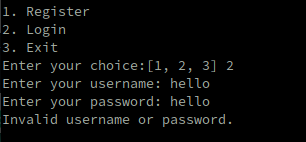


Figure 11 (Test case 4 - invalid credentials login)

**Status:** The quiz informed me that my username or password were invalid, and thus I wasn’t able to log in. It was functional.

3. Random Question Generation

**Test Case 5:** Unique Questions in a Quiz Session

Test Case ID: TC\_QUIZ\_01

Test Description: Verify that each quiz session presents unique questions.

Preconditions: A set of questions is available in the quiz\_tbl table.

Test Steps:

1. Log in with valid credentials.
2. Start a new quiz session.
3. Check each question.

Expected Result: Each question in the quiz is unique, with no repetition within the same session.

Actual result: Indeed each question was different.

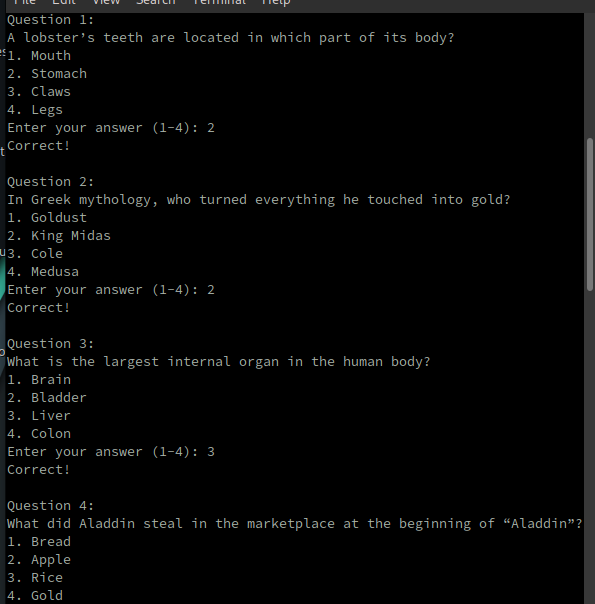


Figure 12 (Test case 5 - unique questions 1.1)



Figure 13 (Test case 5 - unique questions 1.2)

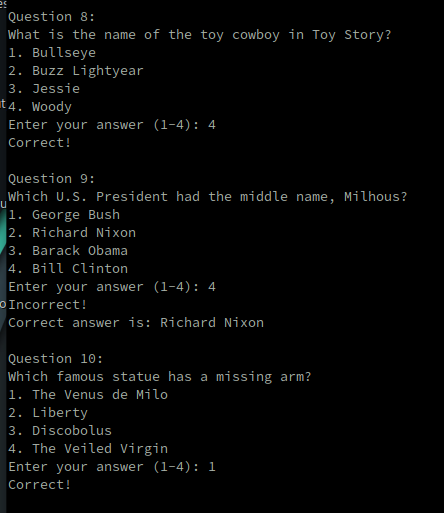


Figure 14 (Test case 5 - unique questions 1.3)

**Status:** functional.

4. Validation and Error Handling

**Test Case 6:** Invalid Option Selection in Quiz

Test Case ID: TC\_VALID\_01

Test Description: Verify the system handles invalid option selection in the quiz.

Preconditions: User is logged in and a quiz session is active.

Test Steps:

1. Attempt to submit an invalid option (e.g., select an option outside the given range), at any step.

Expected Result: An error message is displayed prompting the user to select a valid option.

Actual Result: The quiz informed me of my invalid input, then asked me to enter a number from the valid range (between 1 and 4).

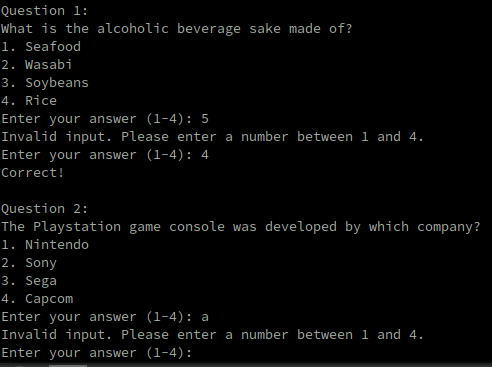


Figure 15 (Test case 6 - invalid input)

**Status:** Functional

5. Score Computation in Percentage

**Test Case 7:** Correct Score Calculation

Test Case ID: TC\_SCORE\_01

Test Description: Verify the system calculates the score correctly in percentage.

Preconditions: User has completed a quiz session.

Test Steps:

1. Answer a predefined set of questions with known correct and incorrect answers.
2. Submit the quiz.
3. View results.

Expected Result: The system calculates the score as the percentage of correct answers and displays it.

Actual Result: In the same trial as test case 5, where we determined each question was unique, the number of correct answers I got was 9/10. At the end of the quiz, I was shown this result;



Figure 16 (Test case 7 - scores)

which matched the number I got right converted into a percentage.

**Status:** Functional

Additional Comprehensive Test Cases

**Test Case 8:** Ensure Random Questions Across Sessions

Test Case ID: TC\_QUIZ\_02

Test Description: Verify that each new quiz session presents a different set of questions.

Preconditions: Multiple questions are available in the quiz\_tbl table.

Test Steps:

1. Log in and start a new quiz session.
2. Note the questions presented.
3. Exit and login again.
4. Start another new quiz session.
5. View each question.

Expected outcome: the questions asked differ from the first round, as there are an adequate number of questions.

Actual result:

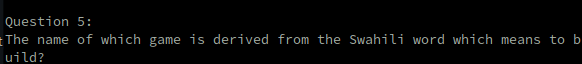


Figure 17-17.9 (Test case 8 - unique questions between rounds)



















These questions were completely separate from the previous trial, as demonstrated in the screenshots in test case 5.

**Status:** functional.

Each of the tests completed so far is functional. Some errors were found in previous versions, which we worked to fix. This includes an error where the database was storing the same login information for multiple users, without informing them that the credentials were already taken.

### **Deployment and Configuration**

The system requirements for all users are that they must have Python 3 or higher installed on their devices, in addition to the VS Code extension “SQLite”. This is usually included with Python installations, but can be installed separately as well. There must be an operating system on their device to ensure the above, and to run the programs. The device must have a keyboard or method for the user to input keystrokes or numbers.

To run or deploy the application, it is necessary for the user to save the application code files onto their environment, and ensure that their device meets the system requirements. Finally, the application can be run from the command line using the command “python quizapp.py” (depending on the name the user may have saved the file with).

1. Save the application code (provided) as a Python file (e.g., quiz\_app.py).
2. Ensure Python and SQLite are installed on the target system.
3. Run the application from the command line using the following command: python quiz\_app.py

3. Configuration

The application uses a SQLite database named "quizapp.db". This database is created automatically upon the first run of the application. No additional configuration is required.

### **Security and Compliance**

Authentication

The application uses a simple username and password-based authentication mechanism to restrict unauthorized access. It is recommended to store passwords securely using a hashing algorithm (not implemented in the provided code). Here, the user is given the option to create or register their credentials, or to login using existing credentials. It is not possible to enter duplicate sets of credentials, as they will be invalidated. Each user is unique.

Data Security

The application stores user credentials and quiz questions in a SQLite database. Sensitive information like passwords should be encrypted. It is necessary to keep this information private to protect the users and avoid leaking either their personal details, or the scores. Similarly, the database should be protected in order to keep the question bank private as well. If the application is deployed by a corporation or school, where the question bank may contain more specific or serious information unlike that which currently populates it, security should increase there.

Compliance

Currently, the application is not designed to comply with any specific security standards or regulations. It is intended for educational purposes and should not be used in production environments with sensitive data. In any future use, it must implement the security measures mentioned above, as well as maintaining a level of transparency for users, such that they may see their scores and answers. If updates or changes are made to the code using external sources, it is necessary to comply with legislation that prevents the theft of code.

### **Maintenance and Support**

Updating the Quiz

New quiz questions can be added to the database using a tool like SQLite DB Browser. The application will automatically pick up the new questions during subsequent runs. There is no limit to the number of questions addable, but some changed may need to be made reflecting the scale (such as ensuring the results are calculated accurately).

Bug Reporting

Any bugs or errors encountered while using the application can be reported through appropriate channels (depending on the deployment scenario). As the code is separated into an adequate number of lines, it is possible for users or checkers to take note of which lines or areas the issues may stem from. Using the comments, it is possible to explain, understand, and describe amendments to the code.

Enhancements

* Implement more secure password hashing.
* Improve the user interface with a more interactive or graphical design.
* Add features like tracking user progress, storing quiz history, and allowing creation/deletion of users by administrators.
* Improving the output report file to include more information on the user performances over time, rather than just the most recent user.
* Increasing the number of questions in the bank.
* Allowing the users to exit during the login process.

### **Appendices**

1. Database Schema

The application uses the following tables in the SQLite database:

* users\_tbl
  + Columns:
    - Username (text) - Unique username for each user.
    - Password (text) - User's password (stored in plain text in this example).
* quiz\_tbl
  + Columns:
    - id (integer) - Primary key for the question.
    - question\_text (text) - Text of the quiz question.
    - option1 (text) - Option 1 for the answer.
    - option2 (text) - Option 2 for the answer.
    - option3 (text) - Option 3 for the answer.
    - option4 (text) - Option 4 for the answer.
    - correct\_answer (text) - The correct answer text.

# **Bibliography**

*7. Input and output* (no date). <https://docs.python.org/3/tutorial/inputoutput.html>. (Accessed 01

May 2024).

Colavito, J. (2013) Harvard formatting and style guide, Uvocorp.com.

https://www.uvocorp.com/dl/Harvard%20Guide.pdf (Accessed: 16 May 2024).

Draw.io (n.d.). *Flowchart Maker & Online Diagram Software*. [Online] app.diagrams.net.

Available at: <https://app.diagrams.net/>.

freeCodeCamp.org (2020). *SQLite Databases with Python - Full Course*. [Online] YouTube.

Available at: https://www.youtube.com/watch?v=byHcYRpMgI4&t=2245s [Accessed 15 May 2024].

freeCodeCamp.org. (2023). *SQL Temp Table – How to Create a Temporary SQL Table*. [Online]

Available at: <https://www.freecodecamp.org/news/sql-temp-table-how-to-create-a-temporary-sql-table/>. [Accessed 20 May 2024].

Mertz, J. (2022) *Reading and writing files in Python (Guide)*.

[Online] <https://realpython.com/read-write-files-python/>. [Accessed 23 May 2024].

*Python (2019). The Python Tutorial — Python 3.8.0 documentation.* [Online] *Python.org.*

*A*vailable at: <https://docs.python.org/3/tutorial/>. [Accessed 05 May 2024]

*Python File write (unknown date)*. [Online] Available

at: <https://www.w3schools.com/python/python_file_write.asp>. [Accessed 06 May 2024]

*SQLite3 - DB-API 2.0 interface for SQLite databases*, 2024. *python.org.* [Online]

Available at: https://docs.python.org/3/library/sqlite3.html.[Accessed 05 May 2024].

Sweigart, A. (2019) 'Reading and Writing Files,' in *Automate the Boring Stuff with Python*. 2nd

edn. No Starch. https://automatetheboringstuff.com/2e/chapter9/. [Accessed 05 May 2024]