QUIZ APP

A PROJECT REPORT

Submitted by

REG. NUMBER	NAME
CH.EN.U4CYS22052	Terli Sai Krishna
CH.EN.U4CYS22029	Laleet Krishna R
CH.EN.U4CYS22024	K Jeyanth
CH.EN.U4CYS22007	B Gowthama raj

for the courses

19CSE201 ADVANCED PROGRAMMING

AND

20CYS204 DATABASE MANAGEMENT SYSTEM

AMRITA SCHOOL OF COMPUTING, CHENNAI

AMRITA VISHWA VIDYAPEETHAM

CHENNAI 601 103

NOVEMBER-2023

Certificate

This is to certify that Terli Sai Krishna (CH.EN.U4CYS22052), Laleet Krishna R (CH.EN.U4CYS22029),B. Gowthama Raj (CH.EN.U4CYS22007) and K. Jeyanth (CH.EN.U4CYS22024), have successfully completed the project title " Quiz APP" under the supervision and guidance in the fulfillment of requirements of Third Semester, Bachelor of Technology Computer Science & Engineering (Cybersecurity) of Amrita School of Computing, Chennai.

Dr. Chandralekha. M
Assistant Professor,
Dept of CSE (CYS),
Amrita School of Computing,
Amrita Vishwa Vidyapeetham,
Chennai.

Mr. S. Saravanan,
Assistant Professor,
Dept of CSE (CYS),
Amrita School of Computing,
Amrita Vishwa Vidyapeetham,
Chennai.

ABSTRACT {My_SQL}

The database component of the program manages the storage and retrieval of critical information required for the quiz application. It employs a MySQL database system to organize data into tables such as 'users', 'admins', 'topics', 'dynamic questions tables', and 'topic_high_scores'.

These tables are interconnected by unique identifiers and relationships. The 'users' table stores user authentication data, while the 'admins' table holds admin credentials. 'topics' encompasses information about available quiz topics, and dynamic question tables store topic-specific questions.

Additionally, 'topic_high_scores' keeps track of the highest scores attained in each topic. The database interactions enable the program to efficiently authenticate users, manage quizzes, and record user achievements.

ABSTRACT {PYTHON}

The Python component of the program constitutes the application logic and user interface. It is structured around classes and functions that facilitate user authentication, quiz generation, question presentation, and scoring. The program commences by initializing relevant classes, establishing connections to the MySQL database, and prompting users to log in or register.

Upon successful authentication, users are prompted to select a quiz topic. Questions are dynamically fetched from the database based on the chosen topic, randomized, and presented to the user. Users are prompted to select answers, with immediate feedback provided. Quiz has a time limit of 300 seconds.

The program maintains a running score and concludes the quiz, displaying the final score and any high score achievements. Admins have access to additional functionalities for managing questions, topics, and high scores. The program ensures clean resource management by closing the database connection upon completion.

CONTENTS

CAPSTOE PROJECT -QUIZ APP

INTRODUCTION:

Quiz applications are software programs or mobile apps designed for quizzes, assessments, and surveys.
They serve various purposes, including education, skill assessment, training, entertainment, and self-assessment.

USE CASE:

User Registration and Authentication:

Users can register with a unique username and password.
Upon subsequent visits, they can log in using their credentials.
User Takes a Quiz:

After logging in, the user selects a topic (Python, C++, SQL) to take a quiz.
The app presents a series of questions with multiple-choice answers.
The user selects an answer, and the app provides immediate feedback.
Scoring and High Scores:

The app keeps track of the user's score as they progress through the quiz.
At the end of the quiz, the app displays the user's score out of the total possible score.
If the user achieves a higher score than their previous attempts for that topic, it's recorded as a new high score.
Time Limit for Questions:

Each question has a time limit of 300 seconds (5 minutes).
If the user doesn't answer within the time limit, the app automatically moves on to the next question.
Admin Access:

Administrators can log in with their own credentials.
Admins have additional functionalities, including adding and removing questions/topics, and viewing high scores.
Admin Adds/Removes Questions:

Admins can add questions to the quiz database for Python, C++, and SQL topics.
They can also remove questions if needed.
Admin Manages Topics:

Admins can add new topics to the app, creating new quizzes for those subjects.
They can also remove topics, including all associated questions.
Admin Views High Scores:

Admins can view high scores for each topic, showing the highest scores achieved by users.
User Achieves New High Score:

When a user achieves a new high score for a particular topic, the app congratulates them and updates the high score record.
Maximum Login Attempts:

If a user fails to authenticate after 5 attempts, the app terminates and exits.
Database Interaction:

The app connects to a MySQL database to store user information, questions, and high scores.

SYSTEM REQUIREMENTS

1. Python Environment: Ensure Python 3.x is installed.
2. Python Libraries: Install the mysql.connector library using pip.
3. MySQL Database: Have a running MySQL server with proper connection parameters.
4. MySQL Database Schema: Create required tables in the database as defined in the code.
5. Internet Connection: Needed if the code accesses external resources.
6. Command Line or IDE: Run the code in a Python-compatible CLI or IDE.
7. Operating System: Compatible with Windows, macOS, and Linux.
8. MySQL Server Running: Ensure MySQL server is configured and accessible.
9. Database Initialization: Set up the database with schema and data.
10. User Input: Support user input for authentication and quiz participation.
11. Thread Support: Ensure Python environment supports multi-threading

PROJECT DESIGN

```
import mysql.connector
import random
import threading

class Question:
    def __init__(self, question_id, question_text, answer_choices,
correct_answer, additional_attribute):
        self.question_id = question_id
        self.question_text = question_text
        self.answer_choices = answer_choices
        self.correct_answer = correct_answer
        self.additional_attribute = additional_attribute


class PythonQuestion(Question):
    def __init__(self, question_id, question_text, answer_choices,
correct_answer, additional_python_attribute):
        super().__init__(question_id, question_text, answer_choices,
correct_answer)
        self.additional_python_attribute =
additional_python_attribute

class CppQuestion(Question):
    def __init__(self, question_id, question_text, answer_choices,
correct_answer, additional_cpp_attribute):
        super().__init__(question_id, question_text, answer_choices,
correct_answer)
        self.additional_cpp_attribute = additional_cpp_attribute
```

```
class SqlQuestion(Question):
    def __init__(self, question_id, question_text, answer_choices, correct_answer, additional_sql_attribute):
        super().__init__(question_id, question_text, answer_choices, correct_answer)
        self.additional_sql_attribute = additional_sql_attribute

class User:
    def __init__(self, id, username, password):
        self.id = id
        self.username = username
        self.password = password

class Authenticator:
    def __init__(self, database):
        self.database = database
        self.max_login_attempts = 5

    def authenticate(self):
        for _ in range(self.max_login_attempts):
            username = input("Enter your username: ")
            password = input("Enter your password: ")
            if self.check_credentials(username, password):
                return self.get_user(username)
            print("Authentication failed. Please try again.")
        print("Maximum login attempts reached. Exiting.")
        self.database.close()
        exit(1)

    def check_credentials(self, username, password):
```

```
        query = "SELECT username, password FROM users
WHERE username = %s AND password = %s"
        self.database.cursor.execute(query, (username, password))
        result = self.database.cursor.fetchone()
        return result is not None

    def get_user(self, username):
        query = "SELECT id, username, password FROM users
WHERE username = %s"
        self.database.cursor.execute(query, (username,))
        user_data = self.database.cursor.fetchone()
        if user_data:
            id, username, password = user_data
            return User(id, username, password)
        return None

    def register(self):
        username = input("Enter a new username: ")
        password = input("Enter a new password: ")
        if self.is_username_taken(username):
            print("Username already in use. Please choose another
one.")
            return self.register()
        else:
            self.create_user(username, password)
            print("Registration successful. You can now log in.")
            return self.get_user(username)

    def is_username_taken(self, username):
        query = "SELECT username FROM users WHERE
username = %s"
```

```
        self.database.cursor.execute(query, (username,))
        result = self.database.cursor.fetchone()
        return result is not None

    def create_user(self, username, password):
        query = "INSERT INTO users (username, password)
VALUES (%s, %s)"
        self.database.cursor.execute(query, (username, password))
        self.database.connection.commit()

class Admin:
    def __init__(self, id, username, password):
        self.id = id
        self.username = username
        self.password = password

class AdminInterface:
    def __init__(self, database):
        self.database = database
        self.admin = None

    def authenticate_admin(self):
        username = input("Enter your admin username: ")
        password = input("Enter your admin password: ")
        if self.check_admin_credentials(username, password):
            self.admin = self.get_admin(username)
        else:
            print("Authentication failed. Access denied.")
            return

    def check_admin_credentials(self, username, password):
```

```
        query = "SELECT username, password FROM admins
WHERE username = %s AND password = %s"
        self.database.cursor.execute(query, (username, password))
        result = self.database.cursor.fetchone()
        return result is not None

    def get_admin(self, username):
        query = "SELECT id, username, password FROM admins
WHERE username = %s"
        self.database.cursor.execute(query, (username,))
        admin_data = self.database.cursor.fetchone()
        if admin_data:
            id, username, password = admin_data
            return Admin(id, username, password)
        return None

    def add_question(self, topic_id, question_text, answer_choices,
correct_answer):
        topic = self.get_topic_name_by_id(topic_id)
        if not topic:
            print("Invalid topic ID.")
            return

        table_name = f"{topic.lower()}_questions"
        query = f"INSERT INTO {table_name} (question_text,
answer_choices, correct_answer) VALUES (%s, %s, %s)"
        self.database.cursor.execute(query, (question_text,
answer_choices, correct_answer))
        self.database.connection.commit()

    def get_topic_name_by_id(self, topic_id):
```

```
        query = "SELECT topic_name FROM topics WHERE
topic_id = %s"
        self.database.cursor.execute(query, (topic_id,))
        result = self.database.cursor.fetchone()
        if result:
            return result[0]
        return None

    def remove_question(self, topic_id):
        topic = self.get_topic_name_by_id(topic_id)
        if not topic:
            print("Invalid topic ID.")
            return

        table_name = f"{topic.lower()}_questions"
        self.list_questions(topic)
        question_id = input("Enter the question ID to remove: ")
        query = f"DELETE FROM {table_name} WHERE
question_id = %s"
        self.database.cursor.execute(query, (question_id,))
        self.database.connection.commit()


    def list_questions(self, topic):
        query = f"SELECT question_id, question_text FROM
{topic}_questions"
        self.database.cursor.execute(query)
        questions = self.database.cursor.fetchall()
        print(f"Questions for {topic}:\n")
        for question in questions:
            question_id, question_text = question
```

```
            print(f"ID: {question_id}, Text: {question_text}")
    def add_topic(self, topic_name):

        query = "INSERT INTO topics (topic_name) VALUES (%s)"
        self.database.cursor.execute(query, (topic_name,))
        self.database.connection.commit()

        query = f"CREATE TABLE IF NOT EXISTS
{topic_name}_questions (question_id INT AUTO_INCREMENT
PRIMARY KEY, question_text VARCHAR(255),
answer_choices VARCHAR(255), correct_answer INT)"
        self.database.cursor.execute(query)
        self.database.connection.commit()
    def get_topics(self):
        query = "SELECT topic_id, topic_name FROM topics"
        self.database.cursor.execute(query)
        topics = self.database.cursor.fetchall()
        return topics


    def remove_topic(self):
        self.list_topics()
        topic_id = input("Enter the topic ID to remove: ")

        topic_name = self.get_topic_name_by_id(topic_id)
        if not topic_name:
            print("Invalid topic ID.")
            return

        self.delete_questions_table(topic_name)
```

```
        query = "DELETE FROM topics WHERE topic_id = %s"
        self.database.cursor.execute(query, (topic_id,))
        self.database.connection.commit()
        print(f"Topic '{topic_name}' and its questions have been
removed.")

    def delete_questions_table(self, topic_name):
        table_name = f"{topic_name.lower()}_questions"
        query = f"DROP TABLE IF EXISTS {table_name}"
        self.database.cursor.execute(query)
        self.database.connection.commit()



    def list_topics(self):
        query = "SELECT topic_id, topic_name FROM topics"
        self.database.cursor.execute(query)
        topics = self.database.cursor.fetchall()
        return topics

def admin_menu(admin_interface):
    while True:
        print("\nAdmin Menu:")
        print("1. Add Question")
        print("2. Remove Question")
        print("3. Add Topic")
        print("4. Remove Topic")
        print("5. Log Out")
        choice = input("Enter your choice: ")

        if choice == "1":
            topics = admin_interface.list_topics()
```

```
            print("Available Topics:")
            for topic in topics:
                topic_id, topic_name = topic
                print(f"{topic_id}. {topic_name}")

            topic_id = input("Enter the topic ID: ")
            question_text = input("Enter the question text: ")
            answer_choices = input("Enter answer choices separated
by ';': ")
            correct_answer = input("Enter the correct answer (as a
number): ")
            admin_interface.add_question(topic_id, question_text,
answer_choices, correct_answer)
        elif choice == "2":
            topics = admin_interface.list_topics()
            print("Available Topics:")
            for topic in topics:
                topic_id, topic_name = topic
                print(f"{topic_id}. {topic_name}")

            topic_id = input("Enter the topic ID: ")
            admin_interface.remove_question(topic_id)
        elif choice == "3":
            topic_name = input("Enter the new topic name: ")
            admin_interface.add_topic(topic_name)
        elif choice == "4":
            topics = admin_interface.list_topics()
            print("Available Topics:")
            for topic in topics:
                topic_id, topic_name = topic
                print(f"{topic_id}. {topic_name}")
```

```
            admin_interface.remove_topic()
        elif choice == "5":
            print("Logging out as admin.")
            break
        else:
            print("Invalid choice. Please enter a valid option (1-5).")

        topics = admin_interface.list_topics()
        print("Available Topics:")
        for topic in topics:
            topic_id, topic_name = topic
            print(f"{topic_id}. {topic_name}")

class Database:
    def __init__(self, host, user, password, database):
        self.connection = mysql.connector.connect(
            host=host,
            user=user,
            password=password,
            database=database
        )
        self.cursor = self.connection.cursor()

    def close(self):
        self.connection.close()

    def get_questions_by_topic(self, topic_name):
        topic = f"{topic_name.lower()}_questions"

        if not self.check_if_topic_exists(topic_name):
```

```
            raise ValueError("Invalid topic")

        query = f"SELECT question_id, question_text,
answer_choices, correct_answer FROM {topic}"

        self.cursor.execute(query)
        rows = self.cursor.fetchall()
        questions = []

        question_class = globals().get(f"{topic_name}Question",
Question)

        for row in rows:
            question_id, question_text, answer_choices,
correct_answer = row
            answer_choices = answer_choices.split(';')
            questions.append(question_class(question_id,
question_text, answer_choices, correct_answer, f"Additional
{topic_name} attribute"))

        return questions


    def check_if_topic_exists(self, topic_name):
        query = "SELECT topic_id FROM topics WHERE
topic_name = %s"
        self.cursor.execute(query, (topic_name,))
        result = self.cursor.fetchone()
        return result is not None

def generate_quiz(database, topic, user_id):
```

```
    questions = database.get_questions_by_topic(topic)
    random.shuffle(questions)
    quiz = Quiz(questions, topic, user_id)
    return quiz

def start_quiz(quiz, database):

print("\n<==================================
==========>\n")
    print("\t Let's begin the Quiz!\n")
    print("\t You have 5 minutes\n")

print("\n<==================================
==========>\n\n")

    while not quiz.is_quiz_over():
        question = quiz.get_current_question()
        print(f"Question {quiz.current_question_index +
1}/{len(quiz.questions)}:\n\n")
        print(question.question_text)
        for i, answer_choice in enumerate(question.answer_choices,
1):
            print(f"{i}. {answer_choice}")

        user_answer = input("\nEnter the number of your answer:
").strip().lower()

        try:
            user_answer = int(user_answer)
            correct_answer = int(question.correct_answer)
            if user_answer == correct_answer:
```

```
            print("\nCorrect!\n")
            quiz.update_score(True)
        else:
            print(f"\nSorry, that's incorrect. The correct answer is:
{correct_answer}\n")
    except ValueError:
        print("\nInvalid input. Please enter a number as your
answer.")

    quiz.next_question()

  quiz.print_result(database)
  database.close()

def update_topic_high_score(database, topic, new_high_score,
user_id):
  query = "INSERT INTO topic_high_scores (topic, high_score,
user_id) VALUES (%s, %s, %s) ON DUPLICATE KEY
UPDATE high_score = %s, user_id = %s"



  database.cursor.execute(query, (topic, new_high_score, user_id,
new_high_score, user_id))
  database.connection.commit()

def get_topic_high_score(database, topic):
  query = "SELECT high_score FROM topic_high_scores
WHERE topic = %s"
  database.cursor.execute(query, (topic,))
  result = database.cursor.fetchone()
```

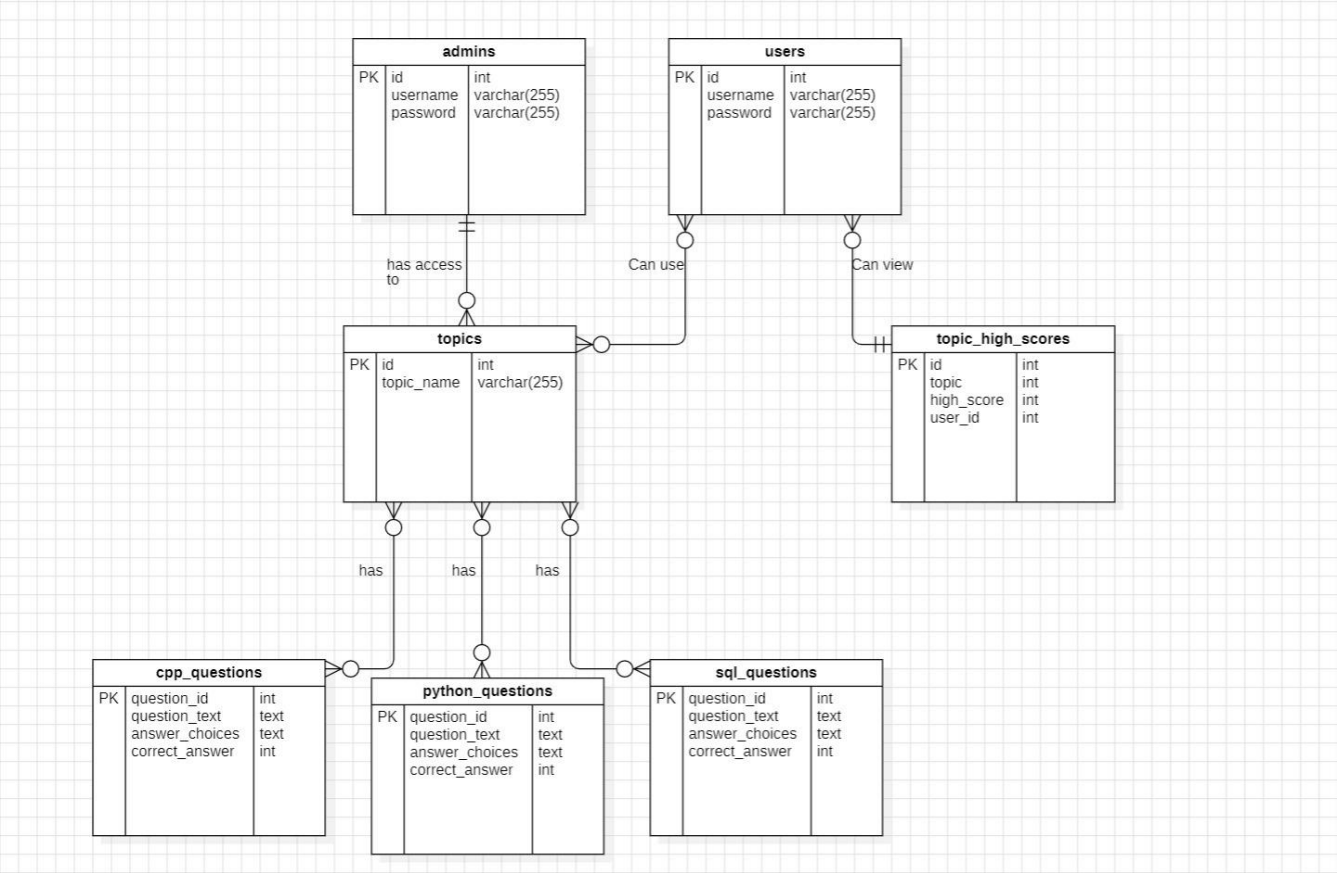
```
    if result:
        return result[0]
    return 0
class Quiz:
    def __init__(self, questions, topic, user_id):
        self.questions = questions
        self.current_question_index = 0
        self.score = 0
        self.timer_thread = None
        self.maximum_score = len(questions)
        self.topic = topic
        self.user_id = user_id

    def get_current_question(self):
        return self.questions[self.current_question_index]

    def is_quiz_over(self):
        return self.current_question_index >= len(self.questions)

    def next_question(self):
        if self.timer_thread:
            self.timer_thread.cancel()
        self.current_question_index += 1
        if not self.is_quiz_over():
            self.start_question_timer()

    def update_score(self, is_correct):
        if is_correct:
            self.score += 1

    def start_question_timer(self):
```

```
        self.timer_thread = threading.Timer(300, self.timer_expired)
        self.timer_thread.start()

    def timer_expired(self):
        print("\nTime is up!\n")
        self.current_question_index = len(self.questions)

    def print_result(self, database):
        print("\nQuiz Results:")
        print(f"Score: {self.score}/{self.maximum_score}")

        previous_high_score = get_topic_high_score(database,
self.topic)

        if self.score > previous_high_score:
            update_topic_high_score(database, self.topic, self.score,
self.user_id)
            print(f"Congratulations! You achieved a new high score for
this topic.")
        else:
            print(f"High score for this topic: {previous_high_score}")


if __name__ == "__main__":
    database = Database("localhost", "root", "sk@200545", "sk")
    authenticator = Authenticator(database)

    while True:

print("\n<==============================
==========>\n")
```

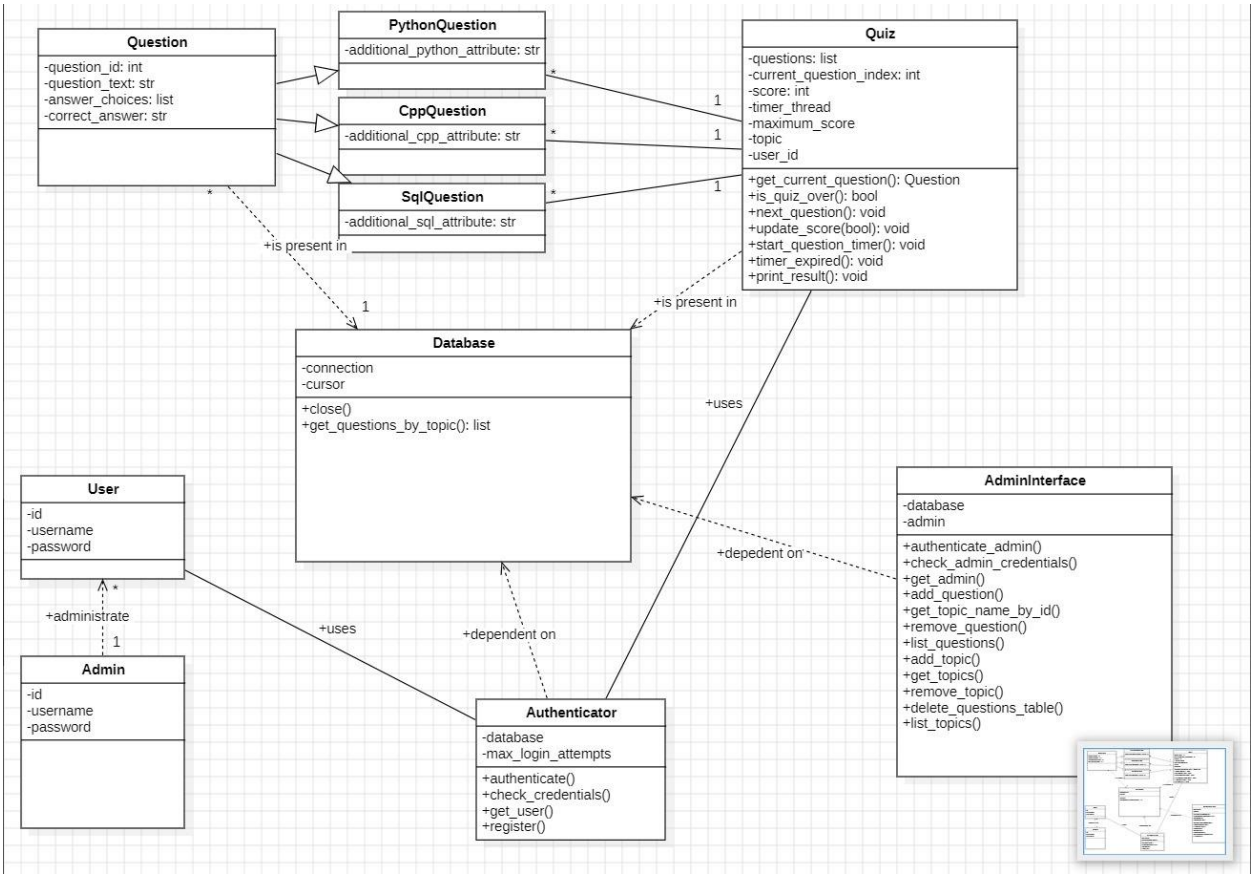
```
        print("\tWelcome to the Python Quiz App")

print("\n<==================================
==========>\n\n")
        print("1. Log in\n")
        print("2. Register\n")
        print("3. Admin Login\n")
        choice = input("Enter your choice (1/2/3): ")

        if choice == "1":
            user = authenticator.authenticate()
            if user.username == "admin":
                admin_interface = AdminInterface(database)
                admin_interface.authenticate_admin()
                if admin_interface.admin:
                    admin_menu(admin_interface)
            elif user:

print("\n<==================================
==========>\n")
                print("\n\tSelect a topic for the quiz:")

print("\n<==================================
==========>\n\n")

                user_interface = AdminInterface(database)

                topics = user_interface.get_topics()
                for topic_id, topic_name in topics:
                    print(f"{topic_id}. {topic_name}")
```

```
            topic = input("Enter the number of your chosen topic: ")
            quiz = generate_quiz(database, topic, user.id)
            start_quiz(quiz, database)
        break
    elif choice == "2":
        user = authenticator.register()
        if user:
            break
    elif choice == "3":
        admin_interface = AdminInterface(database)
        admin_interface.authenticate_admin()
        if admin_interface.admin:
            admin_menu(admin_interface)
    else:
        print("Invalid choice. Please enter 1 to log in, 2 to register,
or 3 for admin login.")
```

ER Diagram:

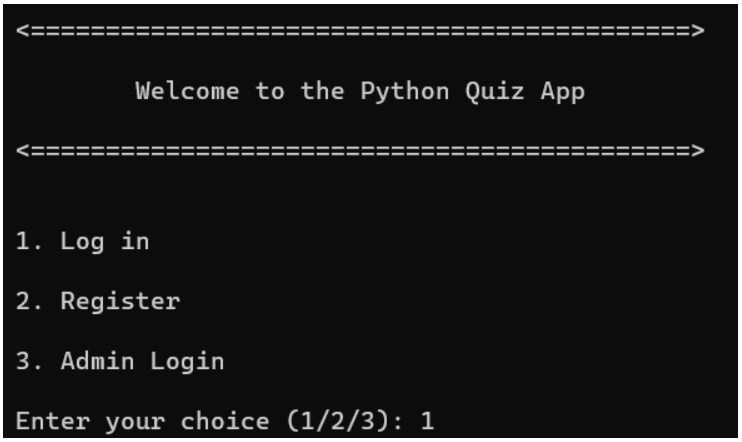

admins
PK id int
username varchar(255)
password varchar(255)
users
PK id int
username varchar(255)
password varchar(255)
has access to
Can use
Can view
topics
PK id int
topic_name varchar(255)
topic_high_scores
PK id int
topic int
high_score int
user_id int
has
has
has
cpp_questions
PK question_id int
question_text text
answer_choices text
correct_answer int
python_questions
PK question_id int
question_text text
answer_choices text
correct_answer int
sql_questions
PK question_id int
question_text text
answer_choices text
correct_answer int

UML Diagram:


Question
-question_id: int
-question_text: str
-answer_choices: list
-correct_answer: str
PythonQuestion
-additional_python_attribute: str
CppQuestion
-additional_cpp_attribute: str
SqlQuestion
-additional_sql_attribute: str
Quiz
-questions: list
-current_question_index: int
-score: int
-timer_thread
-maximum_score
-topic
-user_id
+get_current_question(): Question
+is_quiz_over(): bool
+next_question(): void
+update_score(bool): void
+start_question_timer(): void
+timer_expired(): void
+print_result(): void
+is present in
+is present in
Database
-connection
-cursor
+close()
+get_questions_by_topic(): list
+uses
User
-id
-username
-password
+administrate
Admin
-id
-username
-password
+uses
+dependent on
+depedent on
AdminInterface
-database
-admin
+authenticate_admin()
+check_admin_credentials()
+get_admin()
+add_question()
+get_topic_name_by_id()
+remove_question()
+list_questions()
+add_topic()
+get_topics()
+remove_topic()
+delete_questions_table()
+list_topics()
Authenticator
-database
-max_login_attempts
+authenticate()
+check_credentials()
+get_user()
+register()

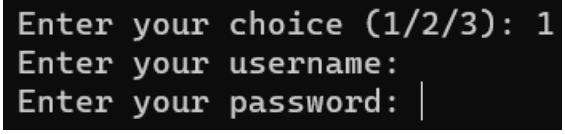
IMPLEMENTATION

1. The quiz starts with asking the user to login, register or login into the admin account.



2. If you are a new user then you can register.

3. If u select login u will be asked for your username and password.



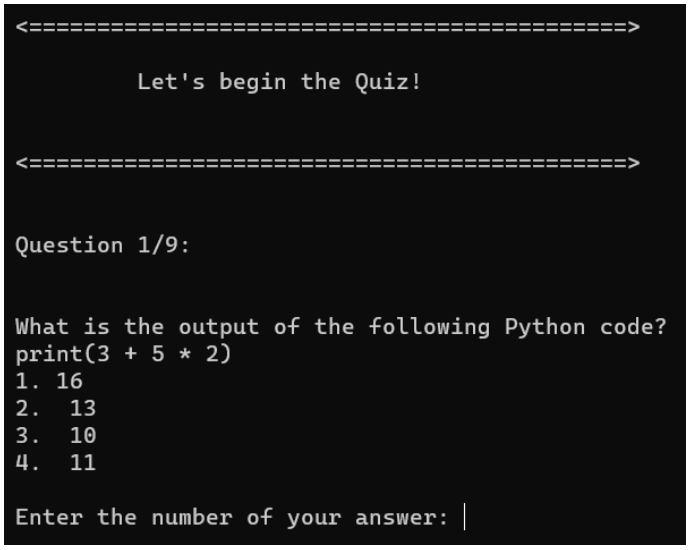
4. Is the entered username or password is wrong it askes to retry for 5 times till the values are correct and if the credentials are wrong the program stops.

```
Enter your username:
Enter your password:
Authentication failed. Please try again.
Enter your username: ascd
Enter your password: abcd
Authentication failed. Please try again.
Enter your username: qwer
Enter your password: rewq
Authentication failed. Please try again.
Enter your username: trew
Enter your password: iuyt
Authentication failed. Please try again.
Enter your username: sdfg
Enter your password: a1s2
Authentication failed. Please try again.
Maximum login attempts reached. Exiting.
```

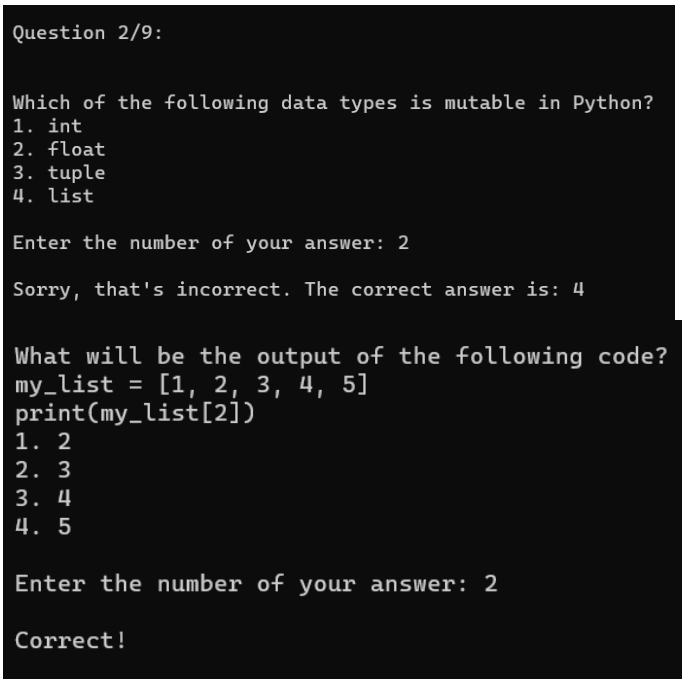
5. If the entered password is correct it will continue and ask the user to select the topic of the quiz he wants to attend.

```
Enter your username: sai krishna
Enter your password: tsk@2005

<=============================================>


        Select a topic for the quiz:

<=============================================>


1. Python

2. C++

3. SQL

Enter the number of your chosen topic (1/2/3):
```

6. After choosing a topic the quiz welcomes the user and the quiz starts.



7. After selecting an answer from the given options the app will check if the answer is correct or not. If the answer is correct the score increases, if the answer is wrong the app will say what answer is correct.



8. The quiz has a timer of 5 minutes and when the timer runs out the question that the user is attempting is the last question they can attempt.

```
What is the result of the following code?
print(my_string[7:12])
1. World
2. World!
3.  Wor
4. Worl

Enter the number of your answer:
Time is up!
3

Correct!


Quiz Results:
Score: 2/9
High score for this topic: 5
```

9. Each topic of a quiz has a high score recorded and it shows the high score at the end of the quiz.

```
High score for this topic: 5
```

10. There is an option where the faculty/adminstrator can login to add a question, remove a question, add a topic and remove a topic. You can login to that admin account in the welcome page and from there u can login and use the admin privileges.

```
<=============================================>

        Welcome to the Python Quiz App

<=============================================>


1. Log in

2. Register

3. Admin Login

Enter your choice (1/2/3): 3
Enter your admin username: sk
Enter your admin password: sk@200545

Admin Menu:
1. Add Question
2. Remove Question
3. Add Topic
4. Remove Topic
5. Log Out
Enter your choice:
```

11. You can add question using the admin privileges

```
Admin Menu:
1. Add Question
2. Remove Question
3. Add Topic
4. Remove Topic
5. Log Out
Enter your choice: 1
Available Topics:
1. python
2. cpp
3. sql
6. maths
Enter the topic ID: 6
Enter the question text: what is 2 + 2 = ?
Enter answer choices separated by ';': 2;3;4;5
Enter the correct answer (as a number): 3
```

12. You can remove the question for a topic.

```
Admin Menu:
1. Add Question
2. Remove Question
3. Add Topic
4. Remove Topic
5. Log Out
Enter your choice: 2
Available Topics:
1. python
2. cpp
3. sql
6. maths
Enter the topic ID: 6
Questions for maths:

ID: 2, Text: what is 2 + 2 = ?
Enter the question ID to remove: 2
```

13. You can add a new topic

```
Admin Menu:
1. Add Question
2. Remove Question
3. Add Topic
4. Remove Topic
5. Log Out
Enter your choice: 3
Enter the new topic name: science
Available Topics:
1. python
2. cpp
3. sql
6. maths
7. science
```

14. You can also remove the topics

```
Admin Menu:
1. Add Question
2. Remove Question
3. Add Topic
4. Remove Topic
5. Log Out
Enter your choice: 4
Available Topics:
1. python
2. cpp
3. sql
6. maths
7. science
Enter the topic ID to remove: 7
Topic 'science' and its questions have been removed.
Available Topics:
1. python
2. cpp
3. sql
6. maths
```

RESULT

Functionality: The quiz app successfully implements the following features:

User authentication and registration.
Ability to create, edit, and delete quizzes.
Ability to take quizzes with multiple choice questions.
Scoring system that calculates and displays the user's score.
Have a timer for the quiz being attempted.

Database Integration:

The app effectively uses MySQL to store and retrieve quiz data, user information, and quiz results.

User Interface:

The app provides a user-friendly interface for seamless navigation. The design is intuitive, ensuring users can easily understand and interact with the app.

Scoring and Results:

The scoring system correctly evaluates user responses and displays the score at the end of each quiz.
Security:

The app incorporates security measures such as password to authenticate its user.

CONCLUSION

1. **Enhanced User Experience:**
    - Implementing more interactive elements, such as progress bars or animations, can further engage users during the quiz-taking process.
2. **Performance Optimization:**
    - Depending on the scale of the app and user base, optimizing database queries and overall performance could be beneficial.
3. **Additional Features:**
    - Consider adding features like a leaderboard, or the ability to share quiz results on social media platforms.
4. **User Feedback and Testing:**
    - Gathering user feedback through testing and surveys can help identify areas for improvement and prioritize future updates.