

  
**Project Report on**

**Quizz**

**Submitted by:**

**Name: Karthikeyan Kakkuchira Puthenpurayil Sushimon**

**Group Members: Karthikeyan Kakkuchira Puthenpurayil Sushimon, Adhvaith Sibu and Razin Abdul Saleem**

**Grade: 12 A**

**Reg. no/Roll No: 015805/14(Karthikeyan) 011061/09(Adhvaith) 23000/18(Razin)**

**Subject Name: Computer Science**

**Subject Code: 083**

**Submitted to: Ms. Fouzia Salma**

**DATE:**



**AL NOOR INTERNATIONAL SCHOOL, BAHRAIN**

**CBSE (2022-2023)**

BONAFIDE CERTIFICATE

This is to certify that the project work done by Karthikeyan Kakkuchira Puthenpurayil Sushimon of class GRADE 12 A roll number \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ in the computer lab during the year 2022-2023 is carried out under my supervision and guidance.

Submitted for ALL INDIA SENIOR SCHOOL CERTIFICATE PRACTICAL examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_at

AL NOOR INTERNATIONAL SCHOOL, KINGDOM OF BAHRAIN.

Internal Examiner External Examiner

Date: \_\_\_\_\_\_\_\_\_\_

SIGNATURE OF THE HEAD TEACHER

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Subject** | **Page No.** |
| 1. | Acknowledgement | 5 |
| 2. | Preface | 6 |
| 3. | Introduction | 6 |
| 4. | Objective | 7 |
| 5. | Scope | 7 |
| 6. | Existing System | 8 |
| 7. | Proposed System | 8 |
| 8. | Input/output requirements | 9 |
| 9. | Hardware/Software requirements | 10 |
| 10. | Source Code | 11 |
| 9. | System Design | 32 |
| 10. | Security Control | 45 |
| 11. | Conclusion | 45 |
| 12. | Bibliography | 46 |

ACKNOWLEDGEMENT

It is with pleasure that I acknowledge my sincere gratitude to our teacher, Ms. Fouzia Salma who taught and undertook the responsibility of teaching the subject Computer Science. I have greatly benefited from her classes.

I am especially indebted to our Head Teacher Monika Sethi who has always been a source of encouragement and support and without whose inspiration this project would not have been a successful I would like to place on record heartfelt thanks to her.

I would also like to thank my teachers and parents who helped me a lot in finalizing this project within the limited time frame. Finally, I would like to express my sincere appreciation for all the students for my batch, their friendship & the fine times that we all shared together.

PREFACE

Our project was completed for practical experience in how to code a variety of programs that we can use in our daily life. We must keep pace with the rapid changes in technology around us and develop along with it. Over the past year and a half, our lifestyle has changed drastically due to the pandemic. Fear of contracting Covid-19 kept us from leaving our homes. As we remain indoors, schools have also become online. Online classes are causing students to become stressed about exams.

INTRODUCTION

As a result of the pandemic, our lifestyle has completely changed. Online classes and work from home were implemented to prevent contact with other people. We have been restricted to staying indoors. Classes are being taken online. Students are having difficulty learning and staying focused during class, which results in them copying on tests and exams.

This program assists students with studying for exams and tests. The student will not be able to memorise the option patterns to answer the questions as the options will be jumbled each time.

OBJECTIVE

The goal of this project is to alleviate this stress caused due the pandemic on the students. The program helps the students to prepare for tests and exams in a fun and enjoyable way.

The program allows the students to input the questions and the answers of those questions in the form of options. The program then asks the students these questions in the form of MCQs allowing them to revise for their upcoming tests or exam.

SCOPE

Our project could revolutionise how we learn, through the method of active recall. There will be competitions held within different systems in the internet. These competitions help in rediscovering competitive spirit and also encourage the students to learn. In the future, a message dialog box could be made so that players can communicate within themselves. Options to report, delete existing questions will also be made. Support to a waiting room will also be implemented.

EXISTING SYSTEM

Users have to read a plenty of books, resources, memorise theory in a subject and then further make questions based on this in a book along with proper explanations if required.

Furthermore, they have to try out practice papers to figure out the hardest questions in order to record them.

It is highly inconvenient to carry around a book with unclassified information. Not only that it is also very hard to brush through unclassified information a week before an exam.

PROPOSED SYSTEM

Our project can make things more classified and easier for a user. Instead of writing questions down our dedicated program can save questions directly as user input if requested to, use the questions through the interface. A large number of questions can be added unlike writing it onto a notebook with limited number of pages. It randomises the options in a question so that the user never can and never has to memorise the option but the answer. It also allows multiple players to play together using a server which makes it more enjoyable and interesting for the user. It also has a system controlling registered users.

INPUT REQUIREMENTS

From the Administrator: -

1. A username “Admin” with password must be created.
2. Login through the same.

From the user: -

1. A username with password must be created.
2. Login through the same.

OUTPUT REQUIREMENTS

For Administrator: -

Singleplayer Start Game, Multiplayer Start Game, Multiplayer Join Game, Add Questions, Delete Users.

For users: -

Singleplayer Start Game, Multiplayer Start Game, Multiplayer Join Game, Add Questions.

HARDWARE/SOFTWARE REQUIREMENTS

Hardware necessities:

* Computer system
* RAM: at least 2GB (4GB preferable)
* Storage: Minimum 1 GB

Software necessities:

* Python 3.7 Shell
* Windows 7 and above or Mac OS X 10.11
* The user has to type the password of the their mysql database and also run the following commands in the line client.

“CREATE DATABASE Quiz;”

“CREATE TABLE LoginData(ID INTEGER PRIMARY KEY AUTO\_INCREMENT, Username VARCHAR(50), Password VARCHAR(255));”

“ALTER TABLE LoginData AUTO\_INCREMENT=100;”

SOURCE CODE

Quiz.py

import mysql.connector #for connecting to the database

import socket #for  connecting to a server

import threading #for creating threads within the file

# such as the server etc

import time #for importing time.sleep

from tkinter import messagebox

#to create a messagebox when the tkinter window is prompted to close

import tkinter as tk #tkinter deals with the GUI of the program

def addQuestions(username):

    '''Used to add questions to questions.txt'''

    with open("questions.txt", "r+") as file:

        #opens questions.txt with read and write mode

        singleLine = file.readline()

        #the file contains all the questions as a

        #string of a dictionary in one line

        dict = eval(singleLine)

        #the string of dictionary is evaluated into a dictionary

        print()

        while True:

            question = input("Enter the Question \

to be added : ")

            #prompts the user to type in the question to be added

            if question in dict: #In the scenario that the given

                #question already exists

                print("Question already in directory.")

                print("Please try again.")

                continue

            print("Please enter the fourth \

option as the correct answer.")

            #the dict is defined in such a way that 3

            #the fourth option is the correct answer

            print()

            options = [input(f"Enter option {x} : ")\

                for x in range(1,5)]

            #text puntuation for getting

            # four input values as options

            dict[question] = options + [(username,)]

            #format of every question in the dictionary

            file.seek(0)

            file.write(str(dict))

            #overwrites the old singleline for the new one

            print()

            print("Question has been saved successfully.")

            print()

            break

def deleteUser(username):

    '''Function used to delete users which is only

    accessible to the admin'''

    if username != "Admin": #checks if username is "Admin"

        print("Please login as Admin to delete users.")

        return None

    connection = mysql.connector.connect(host='localhost',\

user='root', passwd='CsMath@007', database='Quiz',\

    auth\_plugin='mysql\_native\_password')

    #connects to the mysql database Quiz

    sqlCursor = connection.cursor()

    #creates a new cursor object

    sqlCursor.execute("""SELECT Username FROM LoginData;""")

    list = sqlCursor.fetchall()

    #fetches Username and Password as a tuple

    usernames = [tuple[0] for tuple in list]

    #text puntuation for producing all usernames into a list

    username = input("Enter the username to be deleted : ")

    while username not in usernames:#if username does not exist

        print()

        print("Username does not exist in the database.")

        print("Please try again.")

        print()

        username = input("Enter the username : ")

        #to enter the username again

    else:

        print()

        sqlCursor.execute(\

f"""DELETE FROM LoginData WHERE Username="{username}";""")

#deletes the record of the given username

        print(f"The login credentials of {username} \

have been successfully deleted.")

        connection.commit()

        #saves the changes

        connection.close()

def createAccount():

    '''Function used to create an account'''

    connection = mysql.connector.connect(host='localhost',\

user='root', passwd='CsMath@007', database='Quiz',\

auth\_plugin='mysql\_native\_password')

    #connects to the mysql database

    sqlCursor = connection.cursor()

    #creates a new cursor object

    commands = ("""CREATE DATABASE Quiz;""","""USE Quiz;"""\

,"""CREATE TABLE LoginData(ID INTEGER PRIMARY KEY, \

Username VARCHAR(50), Password VARCHAR(255));""",\

"""SELECT Username,Password FROM LoginData;""",\

"""ALTER TABLE LoginData AUTO\_INCREMENT=100;""")

    sqlCursor.execute(commands[3])

    list = sqlCursor.fetchall()

    #returns list of tuples containg username and password

    while True:

        username = input("Enter the username : ")

        for tuple in list:

            if tuple[0] == username:

                #if username already exists

                print("Username is already taken")

                print("Try another one.")

                break

        else:

            while True:

                print()

                password = input("Enter the password : ")

                if len(password) < 8:

                    #if password contains less than

                    # 8 characters

                    print(\

"Password should contain atleast 8 characters.")

                    print("Please try again.")

                else:

                    command = \

f"""INSERT into LoginData(Username,Password) \

VALUES("{username}","{password}");"""

                    sqlCursor.execute(command)

                    connection.commit()

                    #changes are saved

                    connection.close()

                    print()

                    print(\

"You have successfully created an account.")

                    return

def loginAccount():

    '''Function used to login to an account'''

    connection = mysql.connector.connect(\

host='localhost', user='root', \

passwd='CsMath@007', database='Quiz',\

auth\_plugin='mysql\_native\_password')

    #connects to the mysql database quiz

    sqlCursor = connection.cursor()

    #creates a new cursor object

    commands = (\

"""CREATE DATABASE Quiz;""","""USE Quiz;""",

"""CREATE TABLE LoginData(ID INTEGER PRIMARY KEY,\

 Username VARCHAR(50), Password VARCHAR(255));""",\

"""SELECT Username,Password FROM LoginData;""",\

"""ALTER TABLE LoginData AUTO\_INCREMENT=100;""")

    sqlCursor.execute(commands[3])

    list = sqlCursor.fetchall()

    #returns list of tuples containg username and password

    while True:

        print()

        username = input("Enter the username : ")

        for tuple in list:

            if tuple[0] == username:

                #Username is found

                print("Username found.")

                while True:

                    print()

                    password = input("Enter the password : ")

                    if password == tuple[1]:

                        #password is correct

                        print("You have successfully logged in.")

                        print()

                        print("Welcome back,", username)

                        return username

                    else:

                        print("Password is incorrect.")

                        print("Please try again.")

        else:

            print("Username does not exist in the database.")

            print("Please try again")

def startServer():

    '''target function in the main file used to start

    a server thread'''

    import LocalServer #imports the file

def menu():

    '''main interface of the program'''

    print('''

                                                            ||MENU||

1. Login to an existing Quizz profile.

2. Create a new Quizz profile.

    ''')

    response = input("Enter the option : ")

    print()

    if response.startswith("1"):

        username = loginAccount()

        print("\_"\*48)

        print('''

        What would you like to do?

1. Singleplayer Start Game

2. Multiplayer Start Game

3. Multiplayer Join Game

4. Add Questions

5. Delete Users [ADMIN ACCESS REQUIRED]

        ''')

        response = input("Enter the option : ")

        print()

        if response.startswith("1"):

            import Singleplayer #imports the file

        elif response.startswith("2"):

            server = threading.Thread(target=startServer)

            server.start() #starts a server thread

            return username

        elif response.startswith("3"):

            return username

        elif response.startswith("4"):

            addQuestions(username)

            return menu()

        elif response.startswith("5"):

            deleteUser(username)

            return menu()

    elif response.startswith("2"):

        createAccount()

        return menu()

    else:

        #invalid response is given

        print("Invalid Response.")

        print("Please try again.")

        return menu()

username = menu() #username from the menu function

#tkinter initialisation

window = tk.Tk() #tkinter window

window.geometry("550x600") #window breadth and height

window.title(f"Quizz--{username}") #title of the window

window.configure(bg='#001F3F') #background colour configuration

#static variables for client

PORT = 5050 #unsed port in most systems

CLIENT = socket.gethostbyname(socket.gethostname())

#retrieves the ip address of the system

FORMAT = "utf-8"

#format for transferring between binary and standard unicode

DISCONNECT\_MESSAGE = "!DISCONNECT"

ADDR = (CLIENT,PORT) #binds port and ip address

client = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

#creates a client socket object

client.connect(ADDR)

#client connects to the port

#initial variables for questions

n = 0

dict={}

score = 0

scoreSheet = []

status = "yet to recieve"

#flag variable when set to "yet to recieve", starts a while

# loop which stops only when the flag variable changes to

#"recieved"

t = 10 #time in between questions is 10 seconds

def send(msg):

    '''used in sending messages to the server'''

    message = msg.encode(FORMAT)

    client.send(message)

def clientInterface():

    '''function that deals with all the messages

    recieved from the server'''

    global n

    global question

    global correctAns

    global options

    global status

    global scoreSheet

    global score

    while True:

        message = client.recv(1024).decode(FORMAT)

        list1 = message.split("|")

        mode = list1[0]

        if mode:

            if mode == "Waiting for players":

                pass

            elif mode == "Multiplayer has started":

                pass

            elif mode == "Question":

                n += 1

                question = list1[1]

                options = eval(list1[2])

                correctAns = list1[3]

                status = "recieved"

                time.sleep(1)

                timeThread = threading.Thread(\

                    target=lambda : lapseTime(t))

                timeThread.start()

                statement = f"{n}. {question}"

                questionLabel.config(text=statement)

                ansOptionA.config(text=\

f"a) {returnOption(0)}",fg="#39CCCC",state="normal")

                ansOptionB.config(text=\

f"b) {returnOption(1)}",fg="#39CCCC",state="normal")

                ansOptionC.config(text=\

f"c) {returnOption(2)}",fg="#39CCCC",state="normal")

                ansOptionD.config(text=\

f"d) {returnOption(3)}",fg="#39CCCC",state="normal")

            elif mode == "Current Score":

                scoreSheet = list1[1]

                scoreSheet = eval(scoreSheet)

                scoreLabel.config(text=\

f"|{scoreSheet[0][0]} : {scoreSheet[0][1]}|\

 |{scoreSheet[1][0]} : {scoreSheet[1][1]}|")

            elif mode == "End Game":

                scoreLabel.config(text=str())

                ansOptionA.destroy()

                ansOptionB.destroy()

                ansOptionC.destroy()

                ansOptionD.destroy()

                timeLabel.destroy()

                questionLabel.config(text="Game has Ended.")

def handle\_queries(message):

    '''function used to handle certain initial queries

    from the server'''

    global username

    if message.startswith(("!USERNAME",)):

        send(username)

        recieveThread = threading.Thread(\

target=clientInterface)

        recieveThread.start()

        #starts a thread for recieving

        # messages from the server

        time.sleep(1)

        #this waits for one second so that the

        # gui is loaded up before the questions arrive

        #otherwise this will result in an error

        send("!Start Game")

        #notifies the server thats the client is ready

message = client.recv(512).decode(FORMAT)#decodes the message

handle\_queries(message)

def eventListener(event,ans=None):

    '''function handling an event in the tkinter user interface'''

    global n

    global question

    global correctAns

    global options

    global score

    for buttonOption in (0,1,2,3):

        changeColour(buttonOption,"#39CCCC")

    if event == "Check Answer":

        for buttonOption in (0,1,2,3):

                if dict[buttonOption] == "correct answer":

                    changeColour(buttonOption,"#2ECC40")

                else:

                    changeColour(buttonOption,"#FF851B")

        if ans == correctAns:

            score += 1

            send(f"!Update score:{score}")

        ansOptionA.config(state="disabled")

        ansOptionB.config(state="disabled")

        ansOptionC.config(state="disabled")

        ansOptionD.config(state="disabled")

def changeColour(buttonOption,colour):

    '''function used to change the colour of clicked buttons'''

    if buttonOption == 0:

        ansOptionA.config(disabledforeground=colour)

    elif buttonOption == 1:

        ansOptionB.config(disabledforeground=colour)

    elif buttonOption == 2:

        ansOptionC.config(disabledforeground=colour)

    elif buttonOption == 3:

        ansOptionD.config(disabledforeground=colour)

def returnOption(option):

    '''function used to return the required option

    as well as set the correct answer in a dict'''

    global n

    global items

    global options

    global dict

    global correctAns

    if options[option] == correctAns:

        dict[option] = "correct answer"

    else:

        dict[option] = "incorrect answer"

    return options[option]

def onClosing():

    '''target function that is prompted when

    a tkinter window is closed'''

    if messagebox.askokcancel(\

"Quit", "Do you want to quit?"):

        window.destroy()#tkinter window is destroyed

        send("!DISCONNECT")

        #client lets server know

        #that the client has disconnected

        exit()

def lapseTime(t):

    '''target function that is prompted after a

    question is recieved, to run for 10 secs

    changing the timeLabel each second'''

    while True:

        try:

            timeLabel.config(text=f"Next Question in {t}")

            #timeLabel configuration every one second

        except:

            # runtime or \_tcl.tkinter error occurs when the

            # timeLabel is destroyed i.e when game ends

            print("Game has ended.")

            exit()

        t -= 1

        if t == -1:

            break

        time.sleep(1)

window.protocol("WM\_DELETE\_WINDOW", onClosing)

while status == "yet to recieve":

    #while loop waits for flag to change

    pass

else:

    #tkinter widgets are defined

    scoreLabel = tk.Label(window,bg='#001F3F',fg="#FFFFFF",\

font=('Verdana',14)) #used to define score

    statement = f"{n}. {question}" #question statement

    questionLabel = tk.Label(window,text=statement,\

font=('Verdana',16),bg='#001F3F',fg="#39CCCC",wraplength=480)

    ansOptionA = tk.Button(window, text=f"a) ", \

command=lambda : eventListener("Check Answer",returnOption(0)),\

padx=20,font=('Verdana',14),bg='#001F3F',fg="#39CCCC",\

wraplength=480)

    ansOptionB = tk.Button(window, text=f"b) ", \

command=lambda : eventListener("Check Answer",returnOption(1)),\

padx=20,font=('Verdana',14),bg='#001F3F',fg="#39CCCC",\

wraplength=480)

    ansOptionC = tk.Button(window, text=f"c) ", \

command=lambda : eventListener("Check Answer",returnOption(2)),\

padx=20,font=('Verdana',14),bg='#001F3F',fg="#39CCCC",\

wraplength=480)

    ansOptionD = tk.Button(window, text=f"d) ", \

command=lambda : eventListener("Check Answer",returnOption(3)),\

padx=20,font=('Verdana',14),bg='#001F3F',fg="#39CCCC",\

wraplength=480)

    timeLabel = tk.Label(window,text=str(t),\

bg='#001F3F',fg="#EFD469",font=('Verdana',14))

    timeThread = threading.Thread(target=lambda : lapseTime(t))

    timeThread.start()

    #time thread starts for the first time

    #packing of all the widgets in order

    questionLabel.pack(pady=20)

    ansOptionA.pack(pady=10)

    ansOptionB.pack(pady=10)

    ansOptionC.pack(pady=10)

    ansOptionD.pack(pady=10)

    timeLabel.pack(pady=5)

    scoreLabel.pack(pady=10)

    window.mainloop()

    #mainloop starts the update loop for all tkinter objects

LocalServer.py

import socket

import threading

from random import shuffle

import time

PORT = 5050

#SERVER = socket.gethostbyname(socket.gethostname())

HOST = socket.gethostbyname(socket.gethostname())

ADDR = (HOST, PORT)

#When we bind our socket to a specific address,

# the address needs to be in a tuple. (HOST,PORT)

HEADER = 16

FORMAT = 'utf-8'

DISCONNECT\_MESSAGE = "!DISCONNECT"

#AF\_INET - over the internet

#socket.SOCK\_STREAM - streaming data throught the socket

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind(ADDR)

#To bind the socket to this address

#anything that connects to the socket connects to this address

class Player:

    '''class used in creating player objects for clients'''

    playerCount = 0

    playerList = []

    usernameList = []

    def \_\_init\_\_(self,username,client,address):

        '''constructor of the class'''

        self.username = username

        self.score = 0

        self.client = client

        self.address = address

        while self.username in Player.usernameList:

            self.username = self.username + "\_"

        Player.playerCount += 1

        Player.playerList.append(self)

        Player.usernameList.append(self.username)

    def deletePlayer(self):

        '''deletes the player object'''

        Player.playerCount -= 1

        Player.playerList.remove(self)

def useQuestions():

    '''function used in getting questions from questions.txt'''

    with open("questions.txt", "r") as file:

        singleLine = file.readline()

        dict = eval(singleLine)

    return dict

def broadcast(message):

    '''function used in broadcasting messages to all clients'''

    for player in Player.playerList:

        player.client.send(message.encode(FORMAT))

def main():

    '''main function that deals with client requests'''

    server.listen()

    #listens for new connections

    while True:

        clientObject, address = server.accept()

        #accepts new connections

        clientObject.send("!USERNAME".encode(FORMAT))

        #asks for username

        username = clientObject.recv(512).decode(FORMAT)

        player = Player(username,clientObject,address)

        #player object is created

        clientThread = threading.Thread(target=handle\_client,\

            args=(clientObject,player))

        #a client thread starts running handling

        # all messages with the client

        clientThread.start()

        print(f"\n[ACTIVE CONNECTIONS] \

{threading.activeCount()-3}")

def handle\_client(clientObject,player):

    '''handles all messages regarding one client'''

    global items

    global options

    global n

    userName = player.username

    print(f"[NEW CONNECTION] {userName} connected.")

    connected = True

    while connected:

        msg = clientObject.recv(512).decode(FORMAT)

        if msg: # if msg is not static

            if msg.startswith("!Start Game"):

                #client asks server to start game

                if Player.playerCount>1:

                    #if there is two players, game starts

                    gameThread = threading.Thread(target=game)

                    time.sleep(1)

                    gameThread.start()

                    statement = "Multiplayer has started|"

                else:

                    statement = "Waiting for players|"

                    #server tells the client to wait

            elif msg.startswith("!Update score"):

                #to update the score in the player object

                #  and to broadcast it

                player.score = int(msg.split(":")[1])

                tuple = ()

                for pl in Player.playerList:

                    tuple += ((pl.username,pl.score),)

                else:

                    broadcast(f"Current Score|{str(tuple)}|")

            if msg == DISCONNECT\_MESSAGE:

                #client has disconnected

                connected = False

                print(f"[DISCONNECTION] \

{userName} has disconnected.")

                player.deletePlayer()

                if len(Player.playerList) == 0:

                    #if all players have disconnected

                    print("[SERVER SHUT DOWN] \

All players have disconnected.")

                    exit()

            else:

                broadcast(statement)

    clientObject.close()

#initial question variables

n = 0

dict\_qs = useQuestions()

items = list(dict\_qs.items())

shuffle(items)

correctAns = items[n][1][3]

options = items[n][1][0:4]

shuffle(options)

def game():

    '''game thread that broadcasts all

    the questions'''

    global n

    global items

    while True:

        n+=1

        try:

            correctAns = items[n][1][3]

        except IndexError:

            print("[GAME OVER] Game has ended")

            broadcast("End Game")

            break

        options = items[n][1][0:4]

        shuffle(options)

        statement = f"Question|{str(items[n][0])}\

|{options}|{correctAns}|"

        broadcast(statement)

        time.sleep(11)

serverThread = threading.Thread(target=main)

serverThread.start()

#server thread starts running

print("[STARTING] server is starting...")

Singleplayer.py

from random import shuffle

from tkinter import messagebox

import tkinter as tk

#tkinter initial variables

window = tk.Tk()

window.geometry("550x600")

window.title("Quizz")

window.configure(bg='#001F3F')

def useQuestions():

    '''function used in getting questions from questions.txt'''

    with open("questions.txt", "r") as file:

        singleLine = file.readline()

        dict = eval(singleLine)

    return dict

def eventListener(event):

    '''function handling an event in the tkinter user interface'''

    global n

    global items

    global correctAns

    global options

    global score

    for buttonOption in (0,1,2,3):

        changeColour(buttonOption,"#39CCCC")

    if event == "Next Question":

        n += 1

        try:

            statement = f"{n+1}. {str(items[n][0])}"

        except IndexError:

            n = 0

            statement = f"{n+1}. {str(items[n][0])}"

        correctAns = items[n][1][3]

        options = items[n][1][0:4]

        shuffle(options)

        questionLabel.config(text=statement)

        ansOptionA.config(text=f"a) {returnOption(0)}")

        ansOptionB.config(text=f"b) {returnOption(1)}")

        ansOptionC.config(text=f"c) {returnOption(2)}")

        ansOptionD.config(text=f"d) {returnOption(3)}")

    elif event == "Previous Question":

        n -= 1

        statement = f"{n+1}. {str(items[n][0])}"

        correctAns = items[n][1][3]

        options = items[n][1][0:4]

        shuffle(options)

        questionLabel.config(text=statement)

        ansOptionA.config(text=f"a) {returnOption(0)}")

        ansOptionB.config(text=f"b) {returnOption(1)}")

        ansOptionC.config(text=f"c) {returnOption(2)}")

        ansOptionD.config(text=f"d) {returnOption(3)}")

    elif event == "Check Answer":

        for buttonOption in (0,1,2,3):

                if dict[buttonOption] == "correct answer":

                    changeColour(buttonOption,"#2ECC40")

                else:

                    changeColour(buttonOption,"#FF851B")

def changeColour(buttonOption,colour):

    '''function used to change the colour of clicked buttons'''

    if buttonOption == 0:

        ansOptionA.config(fg=colour)

    elif buttonOption == 1:

        ansOptionB.config(fg=colour)

    if buttonOption == 2:

        ansOptionC.config(fg=colour)

    if buttonOption == 3:

        ansOptionD.config(fg=colour)

def returnOption(option):

    '''function used to return the required option

    as well as set the correct answer in a dict'''

    global n

    global items

    global options

    global dict

    if options[option] == correctAns:

        dict[option] = "correct answer"

    else:

        dict[option] = "incorrect answer"

    return options[option]

def on\_closing():

    '''target function that is prompted when

    a tkinter window is closed'''

    if messagebox.askokcancel(\

"Quit", "Do you want to quit?"):

        window.destroy()

        exit()

window.protocol("WM\_DELETE\_WINDOW", on\_closing)

#initial variables

dict = {}

score = 0

n = 0

dict\_qs = useQuestions()

items = list(dict\_qs.items())

shuffle(items)

correctAns = items[n][1][3]

options = items[n][1][0:4]

shuffle(options)

#tkinter widgets

statement = f"{n+1}. {str(items[n][0])}"

questionLabel = tk.Label(window,text=statement,\

font=('Verdana',16),bg='#001F3F',fg="#39CCCC",wraplength=480)

nextQuestion = tk.Button(window, text="Next Question", \

command=lambda : eventListener("Next Question"),\

font=('Verdana',10),bg='#001F3F',fg="#39CCCC",padx=20)

prevQuestion = tk.Button(window, text="Previous Question", \

command=lambda : eventListener("Previous Question"),\

font=('Verdana',10),bg='#001F3F',fg="#39CCCC",padx=20)

ansOptionA = tk.Button(window, text=f"a) {returnOption(0)}", \

command=lambda : eventListener("Check Answer"),padx=20,\

font=('Verdana',14),bg='#001F3F',fg="#39CCCC",wraplength=480)

ansOptionB = tk.Button(window, text=f"b) {returnOption(1)}", \

command=lambda : eventListener("Check Answer"),padx=20,\

font=('Verdana',14),bg='#001F3F',fg="#39CCCC",wraplength=480)

ansOptionC = tk.Button(window, text=f"c) {returnOption(2)}", \

command=lambda : eventListener("Check Answer"),padx=20,\

font=('Verdana',14),bg='#001F3F',fg="#39CCCC",wraplength=480)

ansOptionD = tk.Button(window, text=f"d) {returnOption(3)}", \

command=lambda : eventListener("Check Answer"),padx=20,\

font=('Verdana',14),bg='#001F3F',fg="#39CCCC",wraplength=480)

#packing of widgets

questionLabel.pack(pady=20)

nextQuestion.pack(pady=10)

prevQuestion.pack(pady=10)

ansOptionA.pack(pady=10)

ansOptionB.pack(pady=10)

ansOptionC.pack(pady=10)

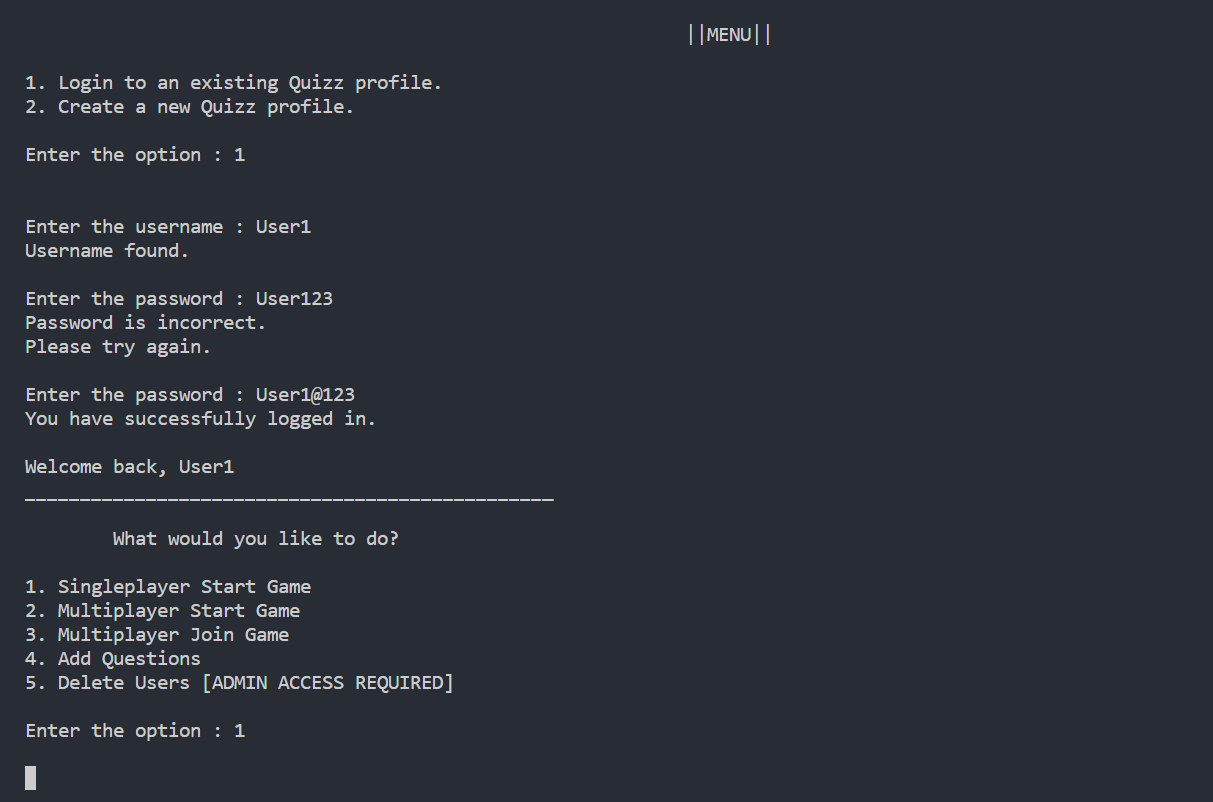
ansOptionD.pack(pady=10)

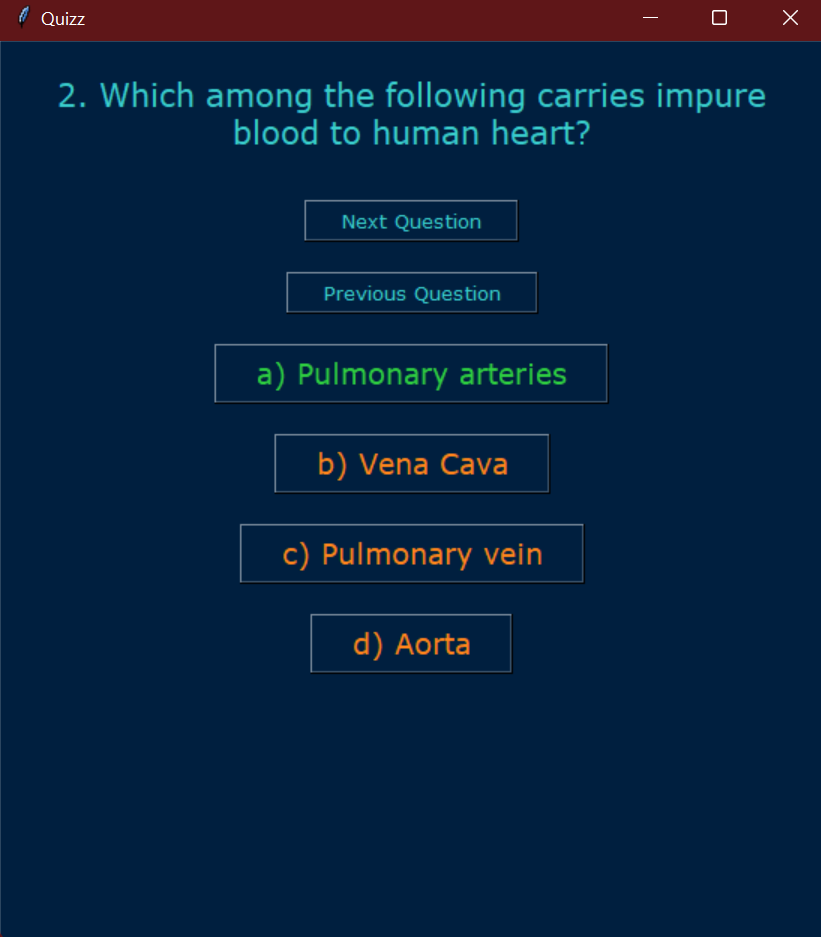
window.mainloop()

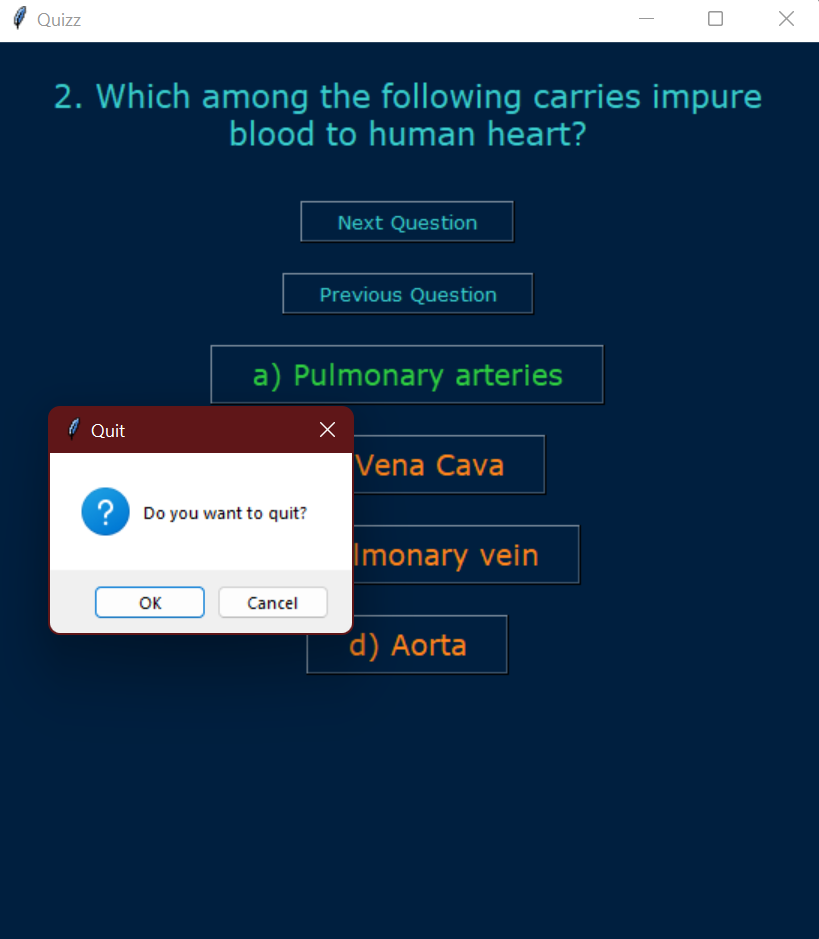
Questions.txt

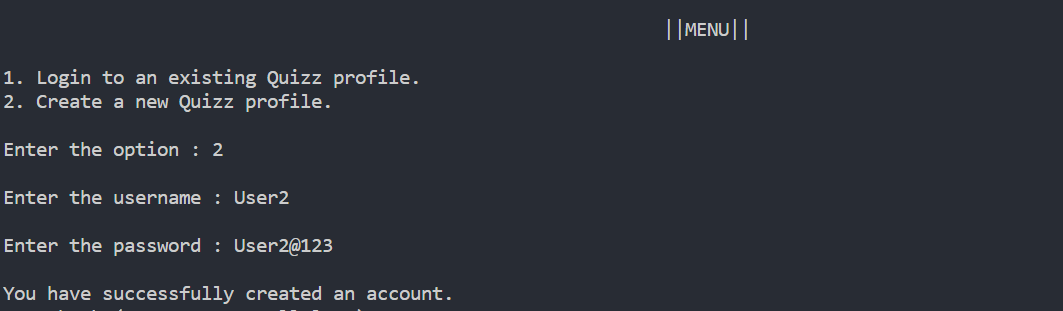


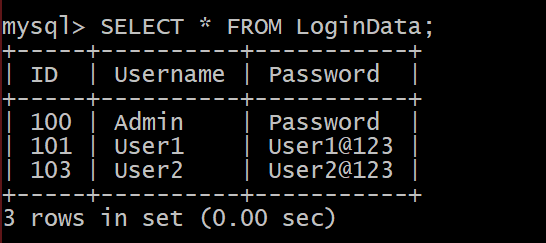
OUTPUT

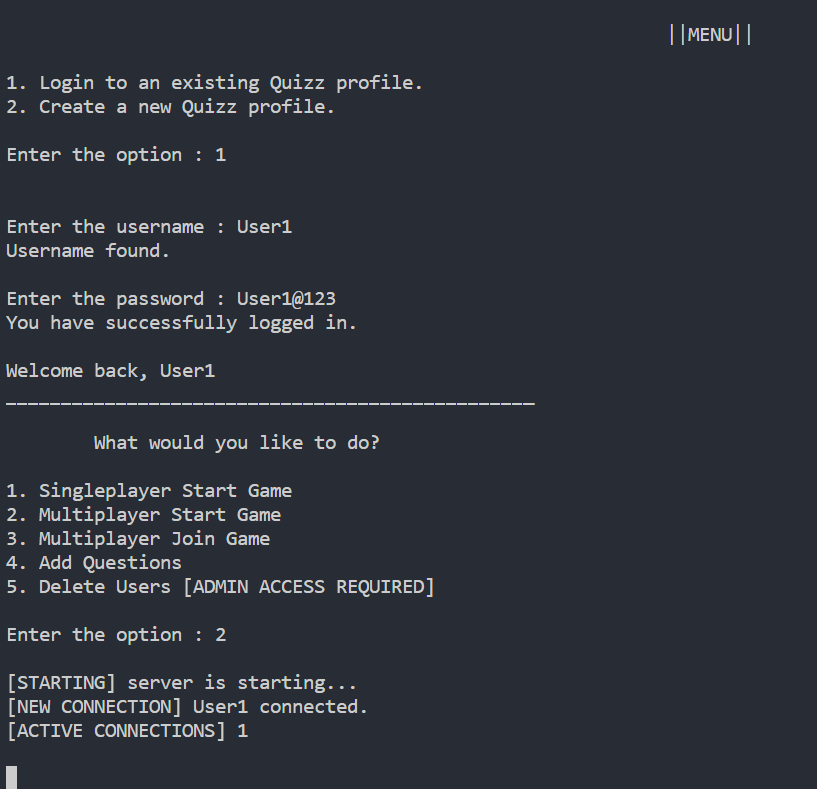


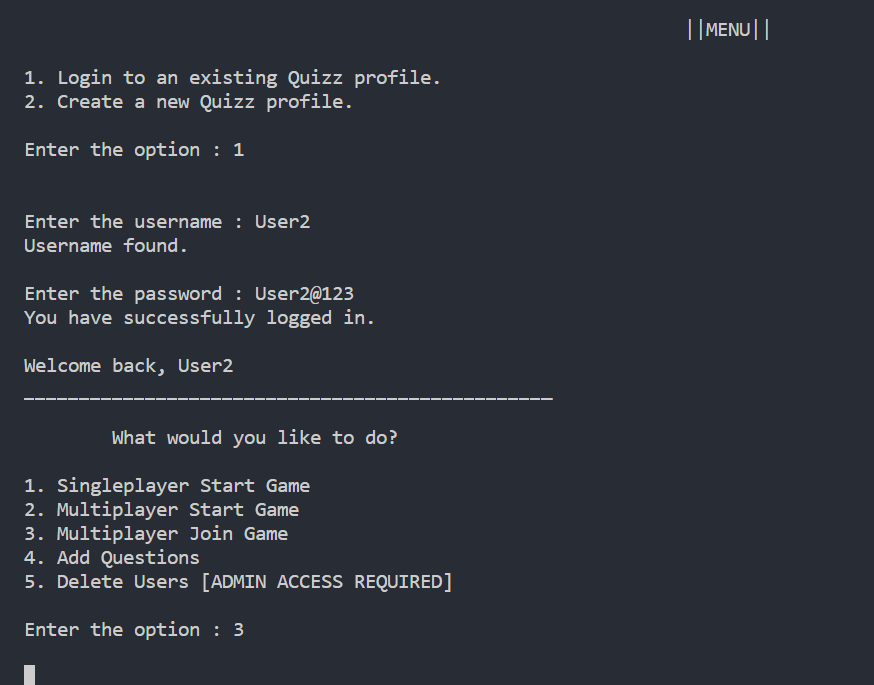


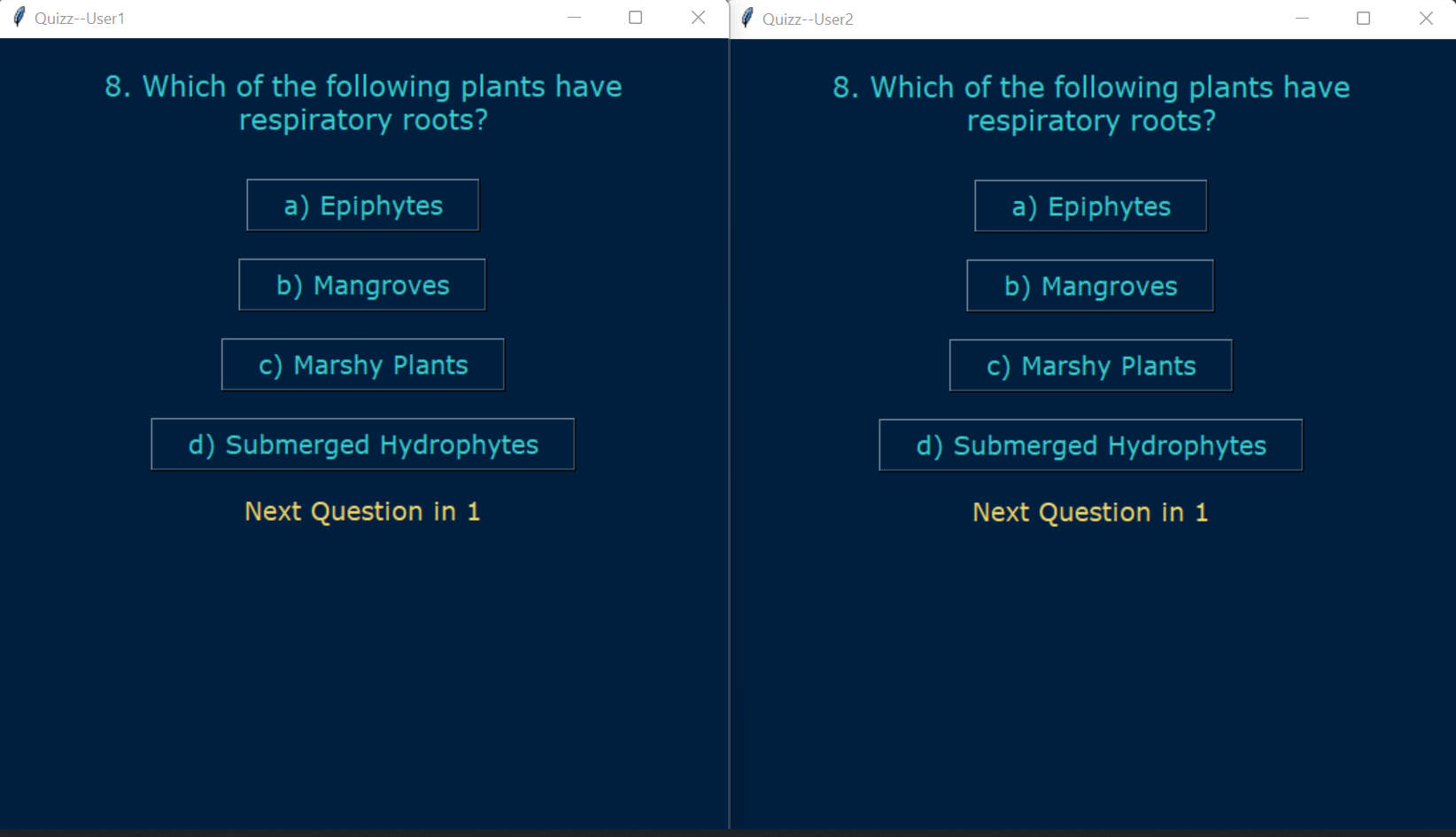


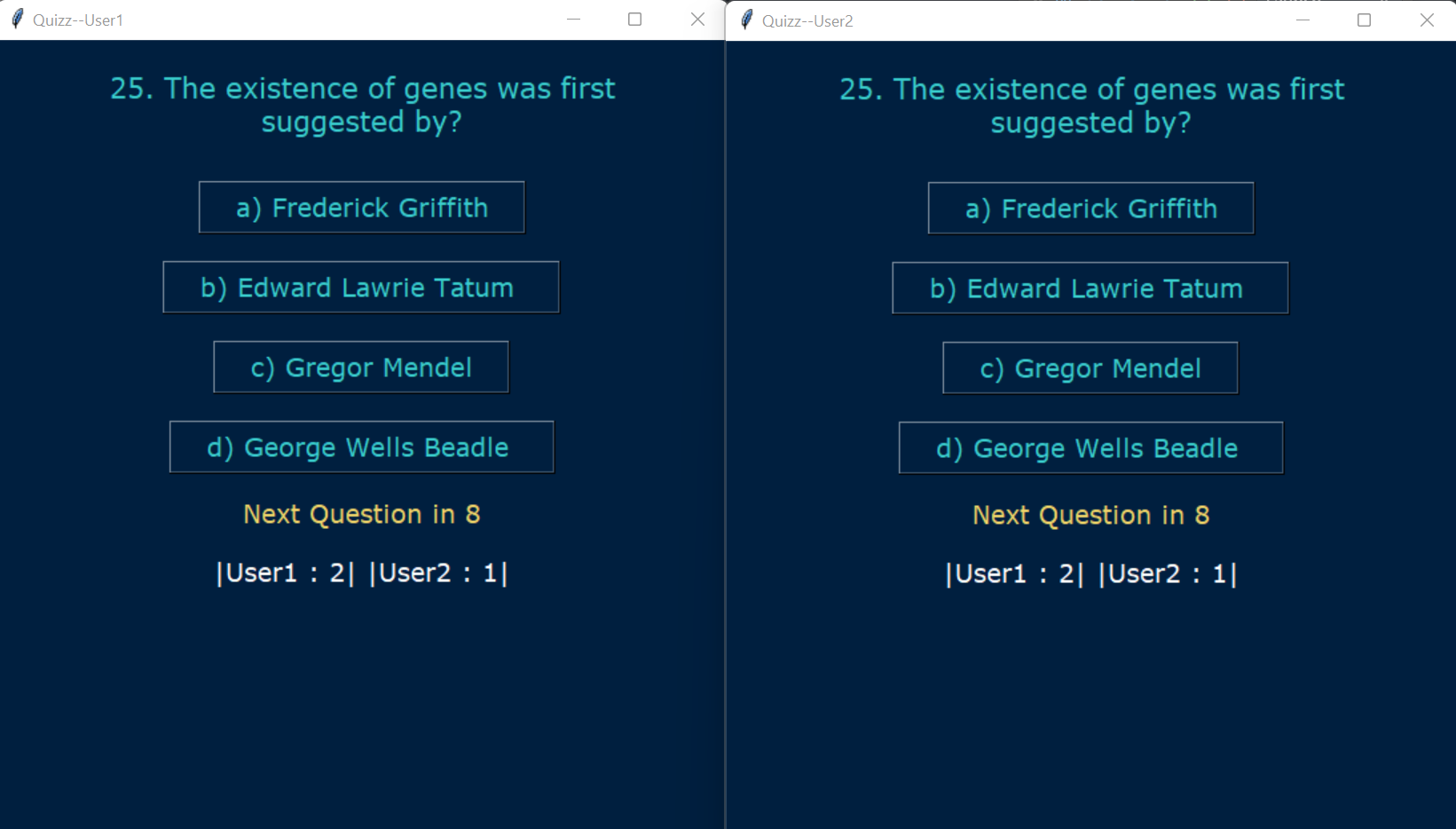


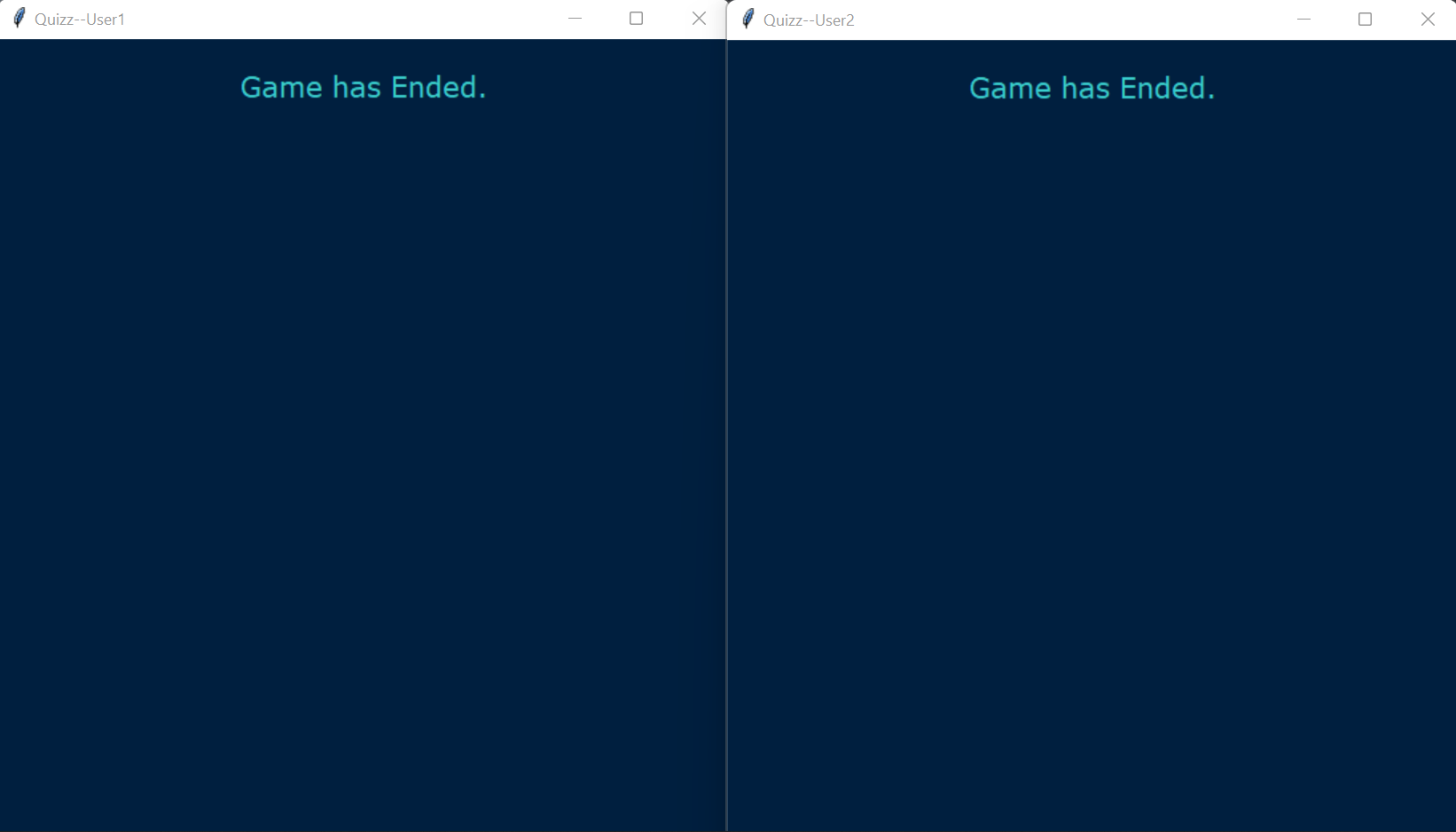


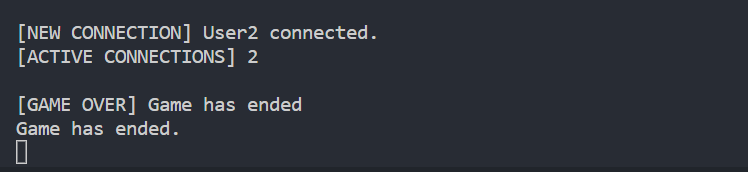


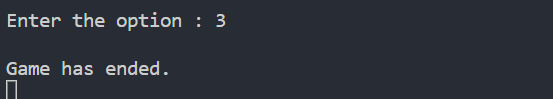


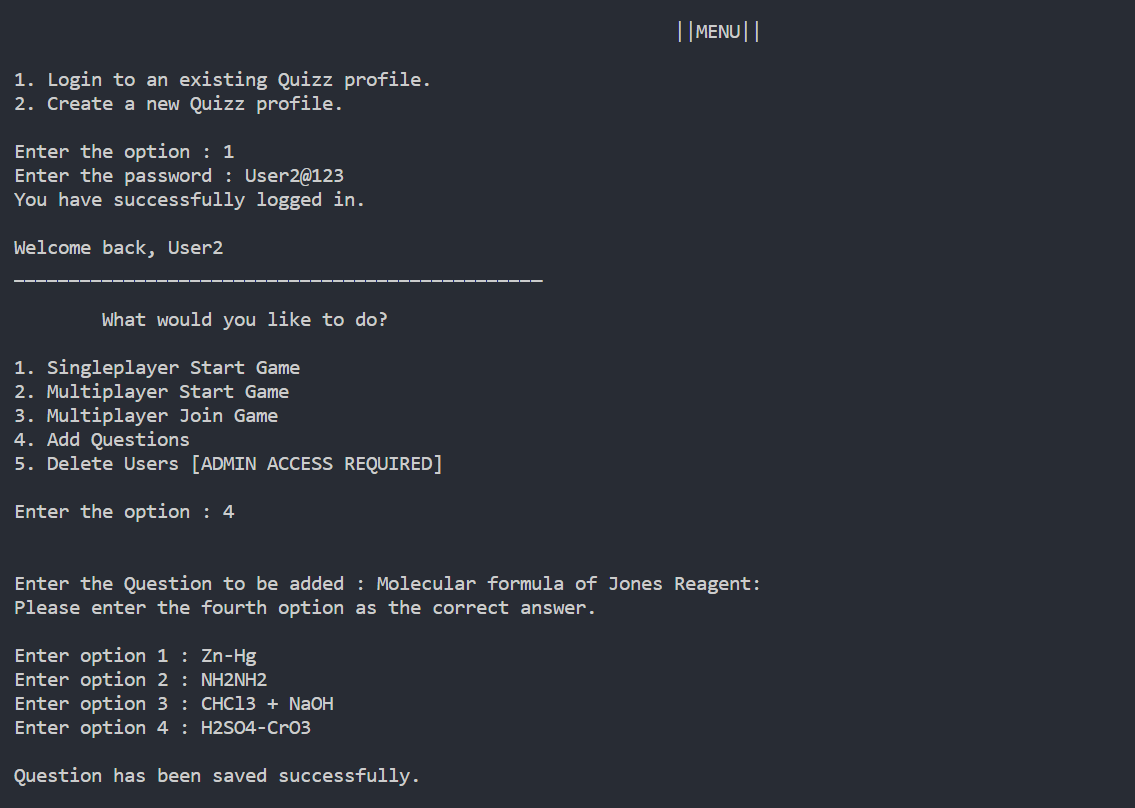


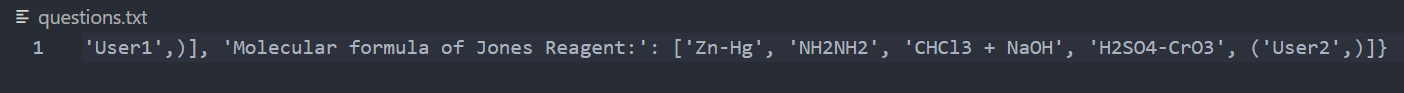


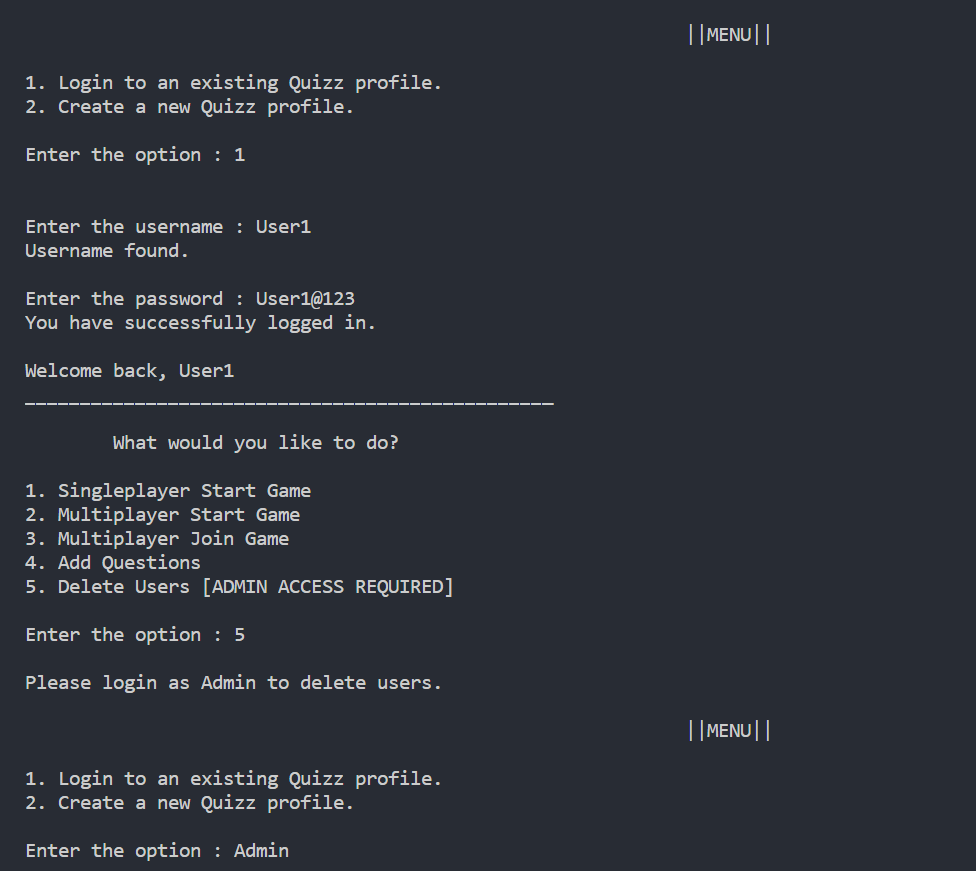


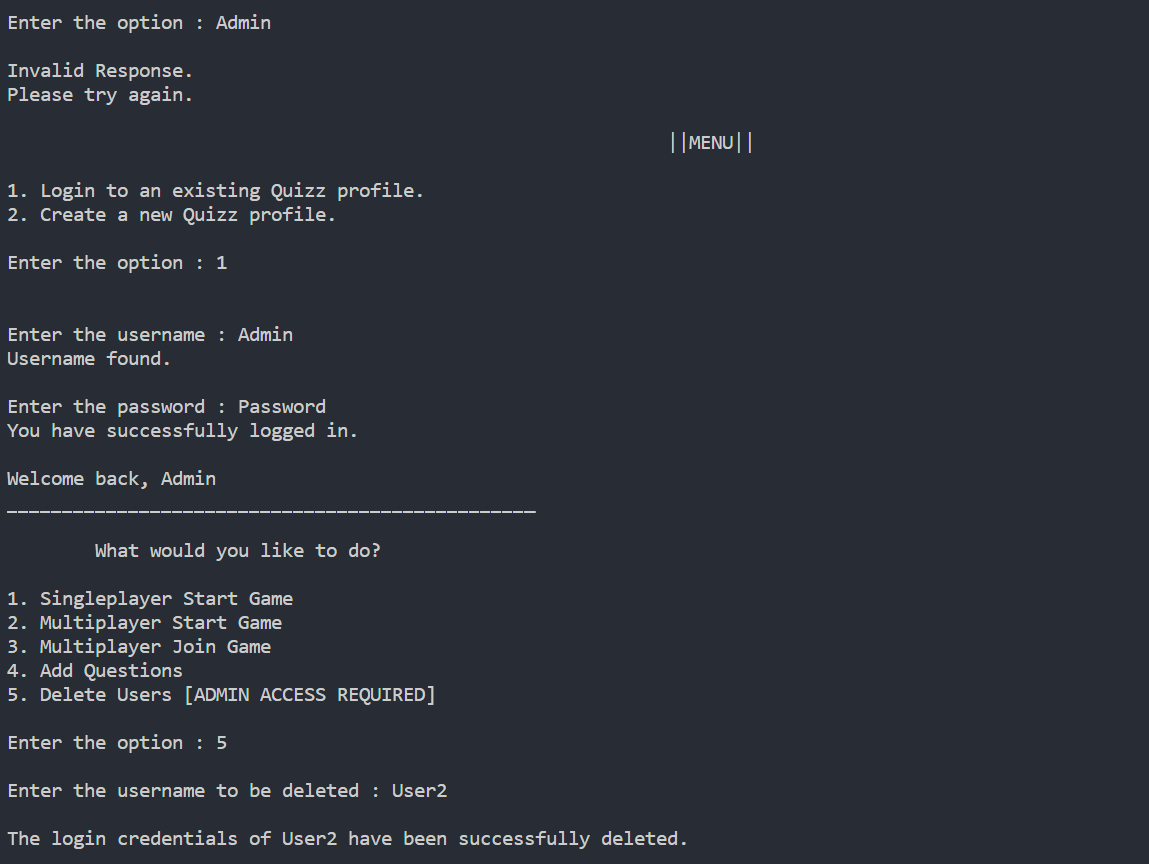


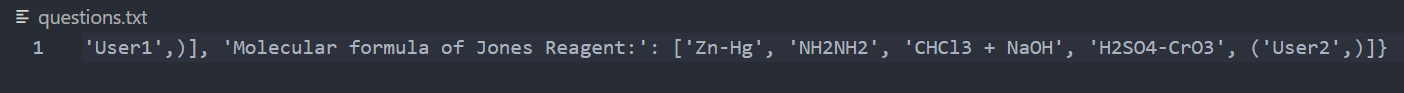












SYSTEM DESIGN

**QUIZ FILE**

**SINGLEPLAYER**

**LOCAL SERVER**

**QUESTIONS TEXT FILE**

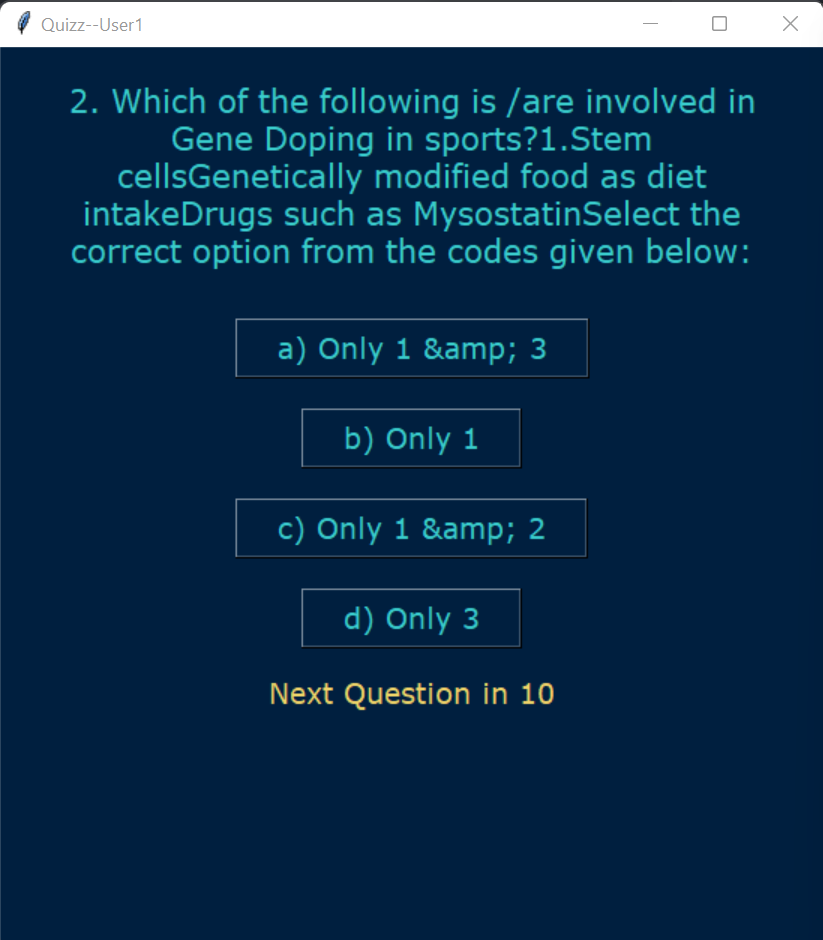
**QUIZ DATABASE**

Major segments of the quiz file are split into subfiles such as LocalServer.py, SinglePlayer.py and Questions.txt.

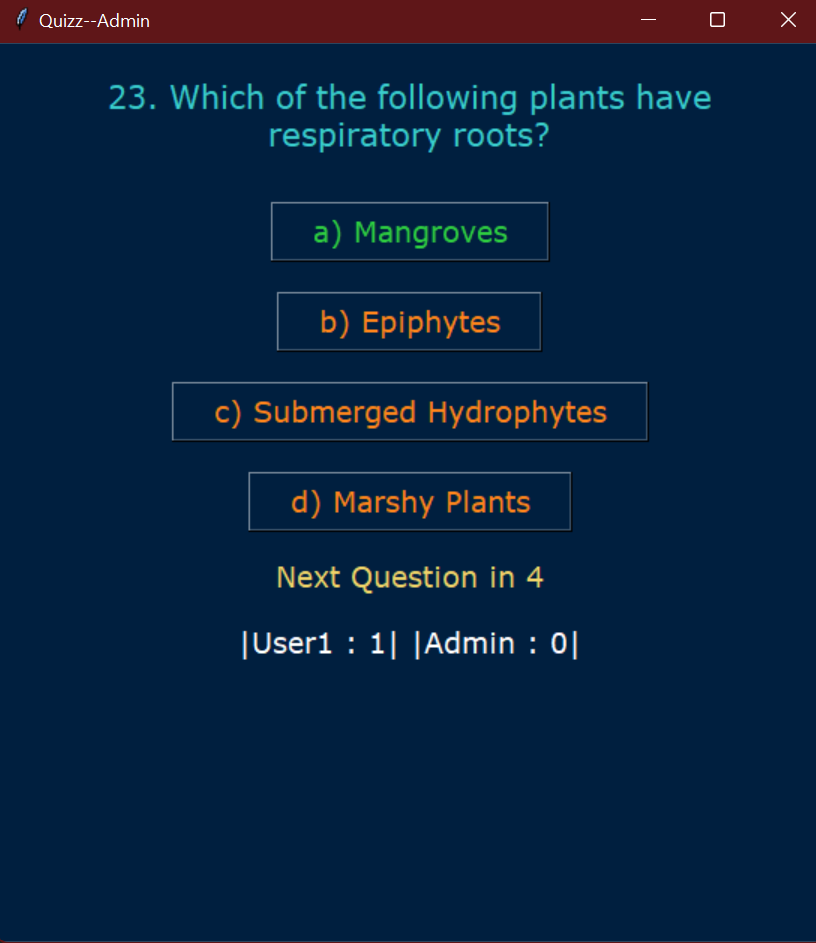
Questions.txt a text file containing a single line string of a dictionary containing questions, answers and username in a list.

The quiz file consists of the menu, all database operations such as creating new user accounts, logging into existing accounts, deleting existing accounts. It also contains the function for adding questions to the text file. The client-side aspect of the application is built into the quiz file.

When a multiplayer game is prompted, the Local Server file is imported as a separate thread, to start a local server through which clients can join the game. Once the server has started, the host joins the server and waits for the second user to join. As soon as the any client joins the game, their username is asked and a player object is created in the server holding all information about the socket object and IP address used. A thread is created in both the server and the client for handling requests and messages when connected. Once two users are connected to the server, a message is broadcasted to both the connected users stating that the game has begun and the server sends the first question. A time thread is created which counts from 10 seconds after a question is broadcasted from the server. A new tkinter window along with widgets is created on both clients. The question, options along with the timer is displayed on the window.

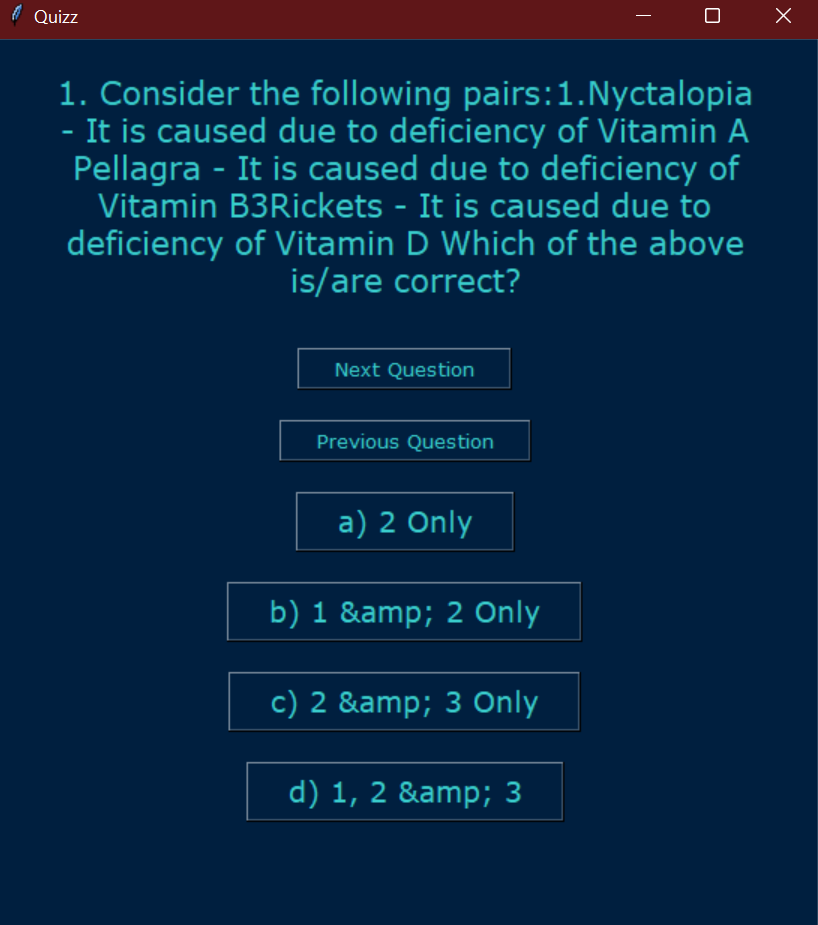


Tkinter window for “User1” in MultiPlayer



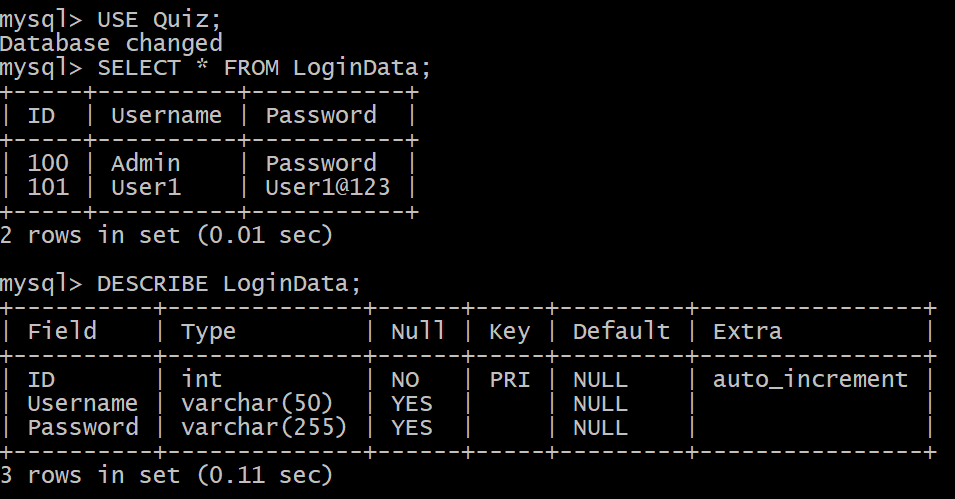
Tkinter window for “Admin” in MultiPlayer

The Single player file is imported when the user wants to learn undisturbed.



Tkinter Window for SinglePlayer

DATABASE DESIGN



CONCLUSION

I am personally looking forward to incorporating more ideas into the program. I've personally learned a lot about quiz-making, local networks and sockets during the research of this program. This project is one of my personal favourites as it focuses on our interest in the learning sector. I'm hopeful that this program will be of great use to fellow users and that it will fulfil its objectives.

BIBLIOGRAPHY

1.) Building a basic chat application in Python (Sockets and Threads) <https://www.youtube.com/watch?v=8HmKhRMlt6c>

2.) Python Object Oriented Programming (OOP) - For Beginners

<https://www.youtube.com/watch?v=JeznW_7DlB0>

3.) Tkinter Course - Create Graphic User Interfaces in Python Tutorial <https://www.youtube.com/watch?v=YXPyB4XeYLA&t=2533s>

# 4.) Python Socket Programming Tutorial

<https://www.techwithtim.net/tutorials/socket-programming/>

# 5.)Threading Tutorial #1 - Concurrency, Threading and Parallelism Explained

<https://www.youtube.com/watch?v=olYdb0DdGtM>

# 6.) Threading Tutorial #2 - Implementing Threading in Python 3 (Examples) <https://www.youtube.com/watch?v=cdPZ1pJACMI>

# 7.) Online Multiplayer Game with Python - Sockets and Networking

<https://www.youtube.com/watch?v=-3B1v-K1oXE>