**CN PROJECT REPORT**

**Introduction:**

In this project, we have developed a client-server quiz game application leveraging cloud-based networking principles. As a cloud service network provider, our goal was to design a scalable and efficient solution that allows multiple clients to participate in a quiz game hosted on a centralized server infrastructure.

**Key Features:**

* Client-Server Architecture: The project utilizes a client-server architecture where clients connect to a centralized server to participate in the quiz game.
* Scalability: The server implementation is designed to handle multiple concurrent connections, ensuring scalability to accommodate a growing number of players.
* Cloud-Based Networking: By leveraging cloud-based networking principles, our server acts as a centralized cloud service, providing reliable and high-performance connectivity to clients over the network.
* Thread Safety and Error Handling: The server script incorporates thread safety mechanisms and robust error handling to ensure smooth and reliable communication with clients.
* Graphical User Interface (GUI): The client application features a user-friendly graphical interface built using Tkinter, enhancing the overall user experience.

**WORKING IDEA OF THE CODE:**

**Client Features:**

The client script (client.py) provides a GUI interface for the player to enter their name and answer quiz questions.

It sends the player's name to the server and receives questions from the server.

After answering each question, it sends the answer to the server and displays feedback received from the server.

Once the quiz ends, it displays the final score to the player.

**Server Features:**

The server script (server.py) handles multiple clients concurrently using threading.

It loads quiz questions and answers from a predefined list.

Upon client connection, it receives the player's name, sends questions, receives answers, and calculates the score.

After the quiz ends, it prints the final scores and determines the winner or if it's a tie.

**Communication Protocol:**

JSON is used for communication between the client and server to send structured data (questions, answers, scores, etc.).

Sockets are used for network communication between the client and server.

**Client script**

import socket

import json

import tkinter.simpledialog as simpledialog

import tkinter.messagebox as messagebox

def receive\_data(sock):

    try:

        data = json.loads(sock.recv(1024).decode())

        return data

    except json.JSONDecodeError:

        print("Error: Invalid data received from the server.")

        return None

def handle\_quiz(sock):

    player\_name = ask\_name()

    sock.sendall(player\_name.encode())

    while True:

        data = receive\_data(sock)

        if not data:

            break

        if isinstance(data, dict):

            if "question" in data:

                question = data["question"]

                user\_answer = ask\_question(question)

                send\_answer(sock, user\_answer)

                feedback = sock.recv(1024).decode().strip()

                show\_feedback(feedback)

            elif "score" in data:

                final\_score = data["score"]

                print(f"Your final score is: {final\_score}")

                sock.close()

                break

        else:

            print("Error: Invalid data received from the server.")

def ask\_name():

    root = simpledialog.Tk()

    root.withdraw()

    player\_name = simpledialog.askstring("Player Name", "Enter your name:")

    root.destroy()

    return player\_name

def ask\_question(question):

    root = simpledialog.Tk()

    root.withdraw()

    user\_answer = simpledialog.askstring("Quiz Question", question)

    root.destroy()

    return user\_answer

def send\_answer(sock, answer):

    sock.sendall(answer.encode())

def show\_feedback(feedback):

    messagebox.showinfo("Quiz Feedback", feedback)

def connect\_to\_server():

    server\_address = ('192.168.18.82', 9999)

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

    client\_socket.connect(server\_address)

    handle\_quiz(client\_socket)

if \_\_name\_\_ == "\_\_main\_\_":

    connect\_to\_server()

**Server script**

import socket

import json

import threading

from threading import Barrier

def load\_questions():

*# Load quiz questions from a JSON file or database*

*# For simplicity, let's define sample questions here*

    questions = [

        {

            "question": "What is the capital of France? Choices: London, Berlin, Paris, Madrid",

            "answer": "Paris"

        },

        {

            "question": "Who wrote 'Romeo and Juliet'? Choices: Jane Austen, William Shakespeare, Charles Dickens, Mark Twain",

            "answer": "William Shakespeare"

        },

        {

            "question": "What is the largest mammal? Choices: Elephant, Giraffe, Blue Whale, Tiger",

            "answer": "Blue Whale"

        },

        {

            "question": "What is the chemical symbol for water? Choices: H, O, W, H2O",

            "answer": "H2O"

        },

        {

            "question": "Which planet is known as the Red Planet? Choices: Mars, Venus, Jupiter, Saturn",

            "answer": "Mars"

        },

        {

            "question": "What is the capital of Japan? Choices: Beijing, Seoul, Tokyo, Bangkok",

            "answer": "Tokyo"

        },

        {

            "question": "Who painted the Mona Lisa? Choices: Leonardo da Vinci, Vincent van Gogh, Pablo Picasso, Michelangelo",

            "answer": "Leonardo da Vinci"

        },

        {

            "question": "Which country is known as the Land of the Rising Sun? Choices: China, India, Japan, South Korea",

            "answer": "Japan"

        },

        {

            "question": "What is the chemical symbol for gold? Choices: G, Au, Go, Ag",

            "answer": "Au"

        },

        {

            "question": "Who discovered penicillin? Choices: Alexander Fleming, Marie Curie, Albert Einstein, Isaac Newton",

            "answer": "Alexander Fleming"

        }

*# Add more questions as needed*

    ]

    return questions

def handle\_client(conn, addr, scores, lock, names, questions, num\_clients):

    print(f"Player connected: {addr}")

    player\_name = conn.recv(1024).decode()

    names.append(player\_name)

    score = 0

    for question\_data in questions:

        question = question\_data["question"]

        answer = question\_data["answer"]

        conn.sendall(json.dumps({"question": question}).encode())

        client\_answer = conn.recv(1024).decode().strip()

        if client\_answer.lower() == answer.lower():

            score += 1

            conn.sendall(b"Correct!\n")

        else:

            conn.sendall(b"Incorrect. The correct answer is: " + answer.encode() + b"\n")

    with lock:

        scores.append((player\_name, score))

    conn.sendall(json.dumps({"score": score}).encode())  *# Send score as JSON data*

    print(f"Player {player\_name} finished with score: {score}")

    with lock:

        num\_clients[0] += 1

def start\_server():

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

    server\_socket.bind(('192.168.18.82', 9999))

    server\_socket.listen(5)

    print("Server is listening on port 9999...")

    scores = []

    names = []

    lock = threading.Lock()  *# Lock for thread-safe access to scores list*

    questions = load\_questions()

    num\_clients = [0]  *# Counter to track the number of clients that have completed the quiz*

    while num\_clients[0] < 1:

        conn, addr = server\_socket.accept()

        client\_thread = threading.Thread(target=handle\_client, args=(conn, addr, scores, lock, names, questions, num\_clients))

        client\_thread.start()

*# Wait for all threads to complete*

    for thread in threading.enumerate():

        if thread != threading.current\_thread():

            thread.join()

    server\_socket.close()

*# Print final scores and determine the winner*

    print("Quiz session ended. Final scores:")

    scores.sort(key=lambda x: x[1], reverse=True)

    print("Player scores:")

    for i, (player\_name, score) in enumerate(scores):

        print(f"Player {player\_name}: {score}")

*# Determine the winner*

    if len(scores) > 1:

        if scores[0][1] == scores[1][1]:

            print("It's a tie!")

        else:

            print(f"Player {scores[0][0]} wins!")

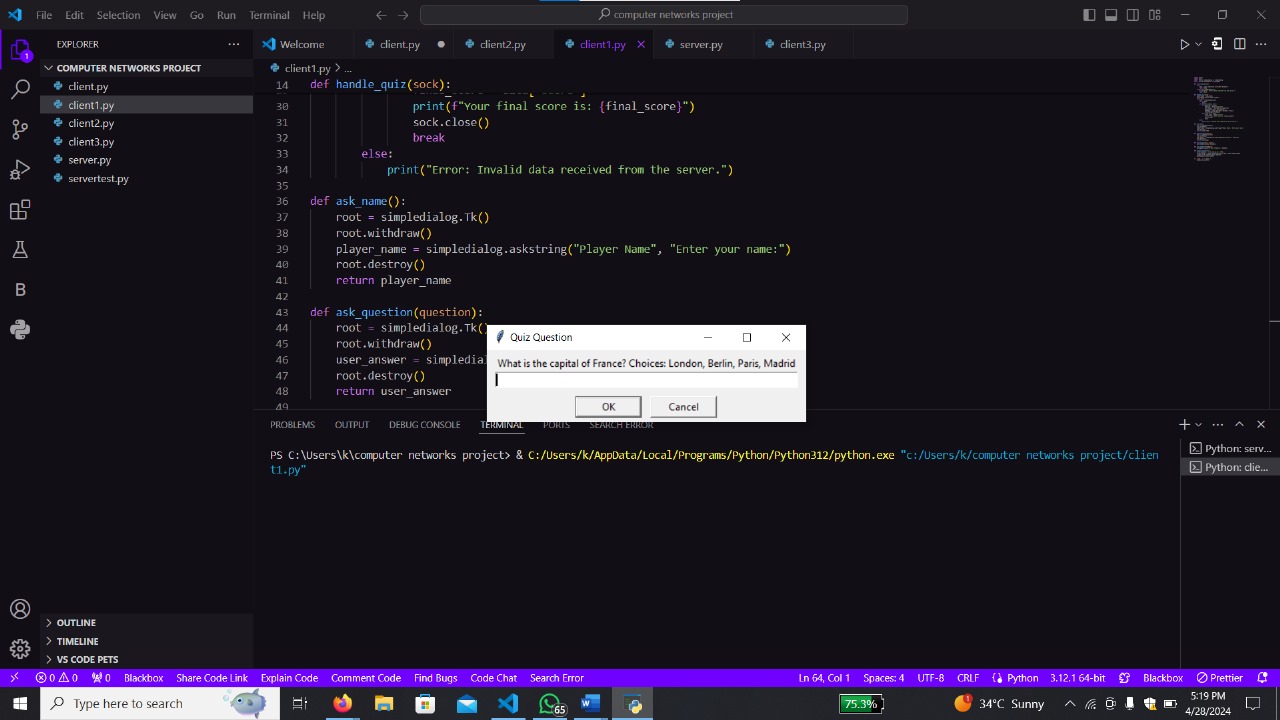
    else:

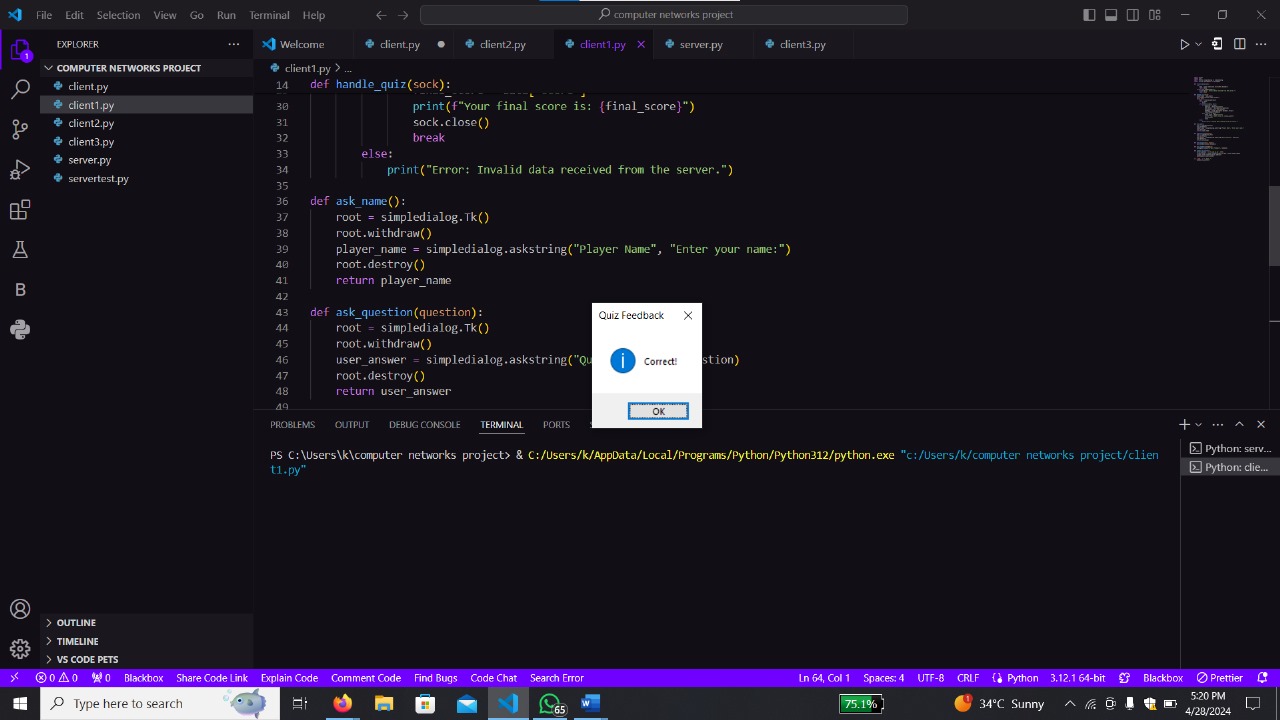
        print("There's only one player.")

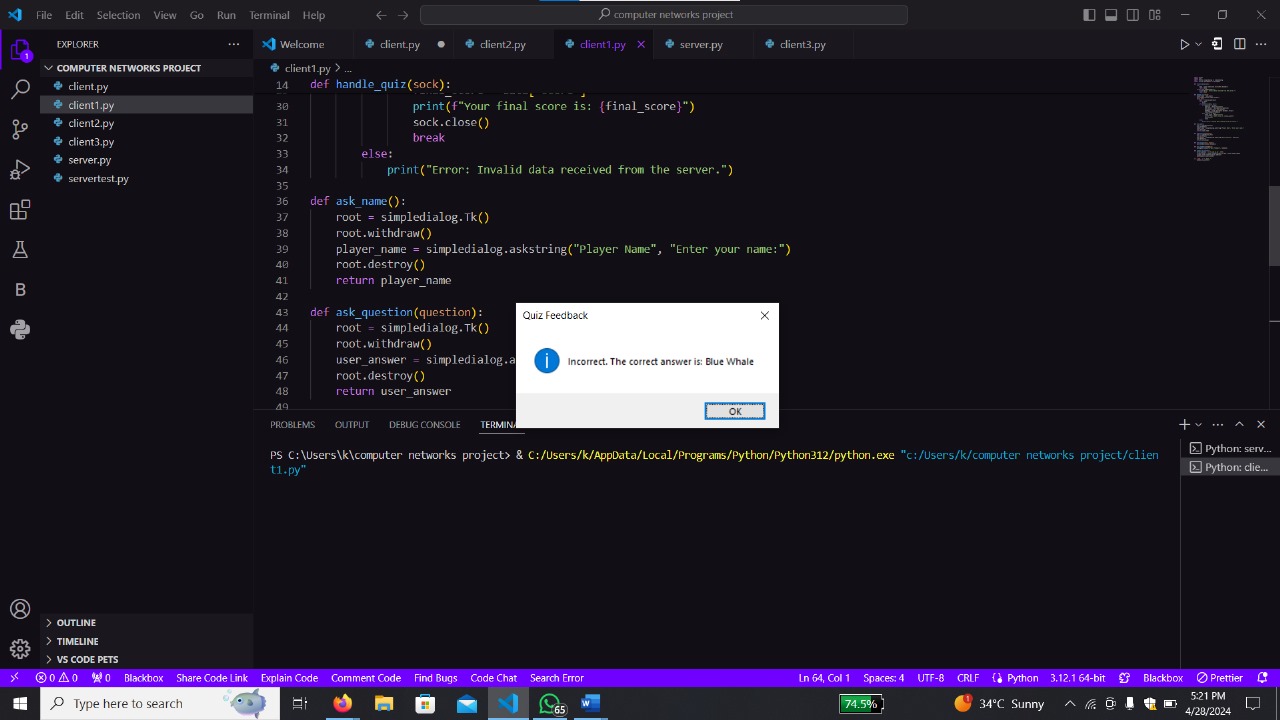
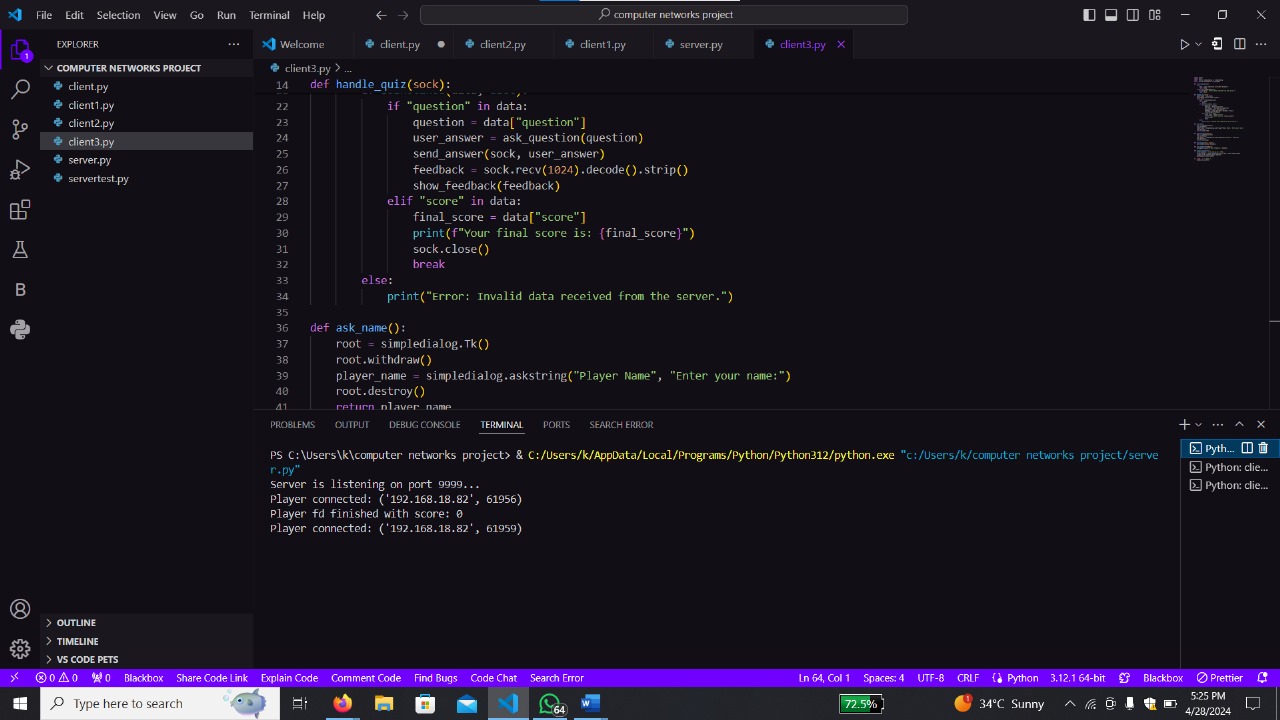
if \_\_name\_\_ == "\_\_main\_\_":

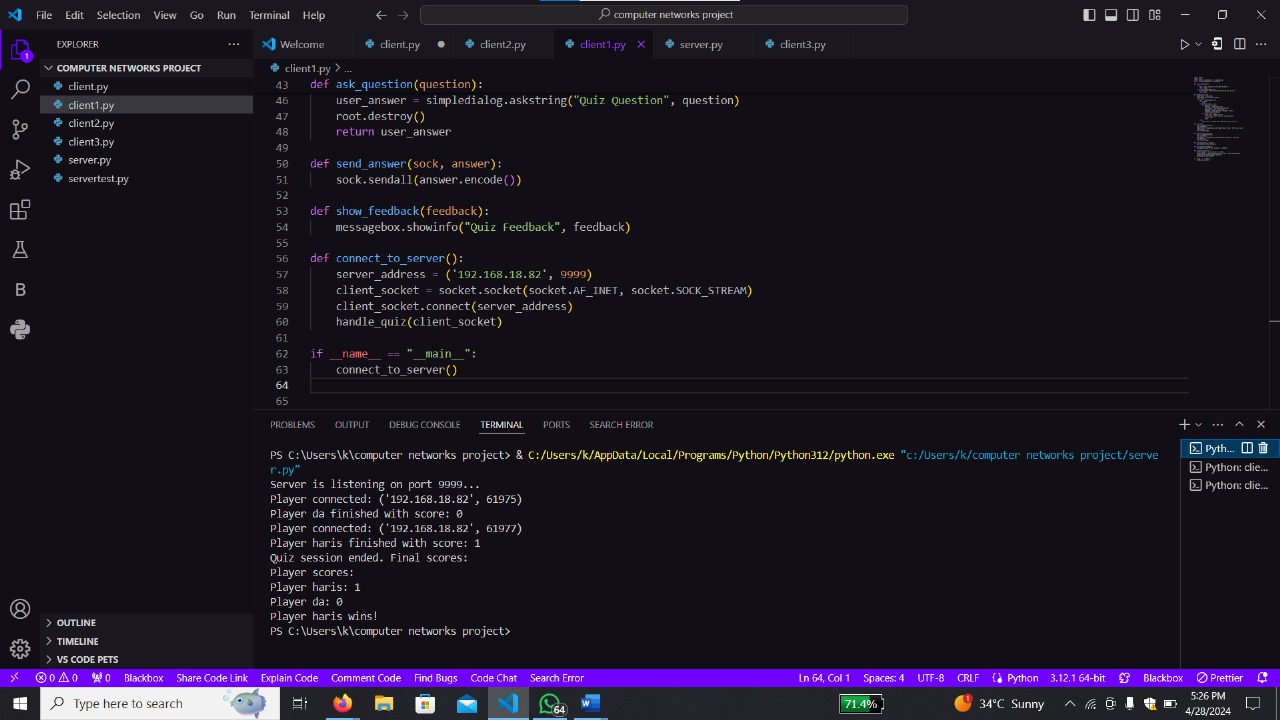
    start\_server()

**CODE OUTPUT**

****

****

****

****

**Conclusion:**

Our client-server quiz game project demonstrates the effective utilization of cloud-based networking principles to create a scalable and engaging application. By facilitating multiple clients to connect seamlessly to our cloud-based server infrastructure, we've showcased the potential of cloud services in providing reliable and efficient network connectivity. Through the integration of a user-friendly graphical interface and robust communication protocols, our project offers an enjoyable quiz experience for players while highlighting the benefits of cloud-based networking solutions.

**GROUP MEMBERS:**

**MASOOMA HASSAN 22K4749**

**MUHAMMAD HARIS 22K4752**

**MUHAMMAD ABDULLAH 22K4712**