MedTech
Mediterranean
Institute of Technology

CS411 – Computer Networks

Lab 4 – Network Socket Programming and Transport Protocol Design

BY
Ines Turki
Nour Guidara
Feriel Charrad
Oussema Sfar

Lab instructor
Iheb Hergli

Tunis, 2025-2026

Table of contents

Table of figures

1.Introduction

The goal of this mini-project is to design and implement a quiz game system that demonstrates how data transmission works at the transport layer using both TCP and UDP.
The project links directly to key **transport-layer** concepts such as:

- Connection establishment and teardown (TCP 3-way handshake)
- Reliability, sequencing, and retransmission
- Connectionless communication and packet loss behavior
- Real-time data delivery trade-offs

In this system, one computer runs **a server**, while others act as **clients**.
The **TCP** version focuses on reliable and ordered communication, while the **UDP** version prioritizes speed and simplicity.

2. Architecture Overview

2.1 Components

- **Server:**
  - Broadcast questions
  - Manage client connections
  - collects answers
  - calculating scores
  - sends feedback and announces results (from leaderboard)

- **Clients**:
  - connect to the server.
  - receive questions.
  - submit answers
  - View feedback
  - View leaderboard

2.2 TCP Architecture

The **TCP version** of the quiz system implements a **connection-oriented** model based on the **SOCK_STREAM** socket type, which ensures reliable and ordered delivery of data.

Server Side

- The TCP server listens for incoming connections on port 8888 using a stream-oriented socket.

- SOCK_STREAM specifies a TCP socket, which provides reliable, ordered, and connection-based communication.

```
#server socket that will handle TCP connections
srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
srv.bind((HOST, PORT))
srv.listen(10)
```

Figure 1:TCP server socket

=>The server uses bind(), listen(), and accept() to wait for incoming client connections, establishing a reliable TCP session for each player.

Client Side

- Each client connects to the server using a socket stream (AF_INET, SOCK_STREAM).

```
# Create and connect TCP socket
sock_obj = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Figure 2:TCP client side socket

=>Once connected, a **reliable byte stream** is established between the client and server.

Concurrency and Data Flow

- The server spawns a dedicated thread per client, allowing simultaneous communication with multiple players.
- Each thread listens for messages such as player answers or disconnects, while the main thread manages the overall game flow.
- Questions, answers, and scores are exchanged as line-based text messages (UTF-8 encoded, each ending with \n), ensuring easy parsing and human readability.

2.3 UDP Architecture

The **UDP version** of the quiz system implements a **connectionless** communication model using **SOCK_DGRAM** sockets. Unlike TCP, the server does not maintain a persistent connection with each client. Instead, all messages are exchanged as independent **datagrams**.

Server Side

- The UDP server listens on port 8888 and receives datagrams from any client that sends data to it.
- Since UDP is connectionless, there is no need to call listen() or accept().

```
# create socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((SERVER_IP, SERVER_PORT))
sock.setblocking(False)
```

Figure 3: UDP server side socket

Client Side

- Each client creates a UDP socket (AF_INET, SOCK_DGRAM) and sends a registration message to the server.

```
# create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setblocking(True)
server_addr = (server_ip, SERVER_PORT)
# send join
join_msg = f"join:{username}".encode(ENCODING)
sock.sendto(join_msg, server_addr)
```

Figure 4:UDP client side socket

=> Questions and responses are exchanged using sendto() and recvfrom(), without establishing a persistent session.

2.4 Threads

In this project, threads are used to keep the quiz running smoothly without blocking the interface. The TCP server uses multiple threads one for each client plus a host control thread while the UDP server runs mainly on a single thread with lighter communication. On the client side, both versions use a background listener thread to receive messages while the main thread handles user input and the Streamlit interface.

Aspect	TCP Server	UDP Server
Main roles	Main thread (quiz flow) + **accept thread** + **one client thread per player** + **host command thread**	Main thread (quiz flow + all network I/O) + **host command thread only**

Aspect	TCP Client	UDP Client
Background thread	Separate **listener thread** continuously reads from the server (recv() loop) and pushes messages to a queue or updates Streamlit UI	Single **listener thread** using recvfrom() to receive broadcasted data from server

3. Workflow and Implementation

3.1 Quiz Flow

1. **Joining:** Clients send join:<username> to the server. TCP connections are established per client; UDP clients are tracked by IP/port.

2. **Validation:** Server ensures each username is unique and maintains a dictionary of usernames and scores.

3. **Start Quiz:** Admin presses Enter (UDP) or types start (TCP) to begin.

4. **Question Broadcast:** Server sends question:<id>:<text> to all connected clients.

5. **Answer Collection:** Clients respond with answer:<A/B/C/D>. Server records the first correct answer.

6. **Score Update:** Scores are incremented, and server broadcasts results with broadcast:TIMEUP Correct=<correct> Winner=<username> and updated leaderboard: score:user1:3|user2:2.

7. **End Quiz:** After all questions, broadcast:QUIZ_END and final leaderboard are sent.

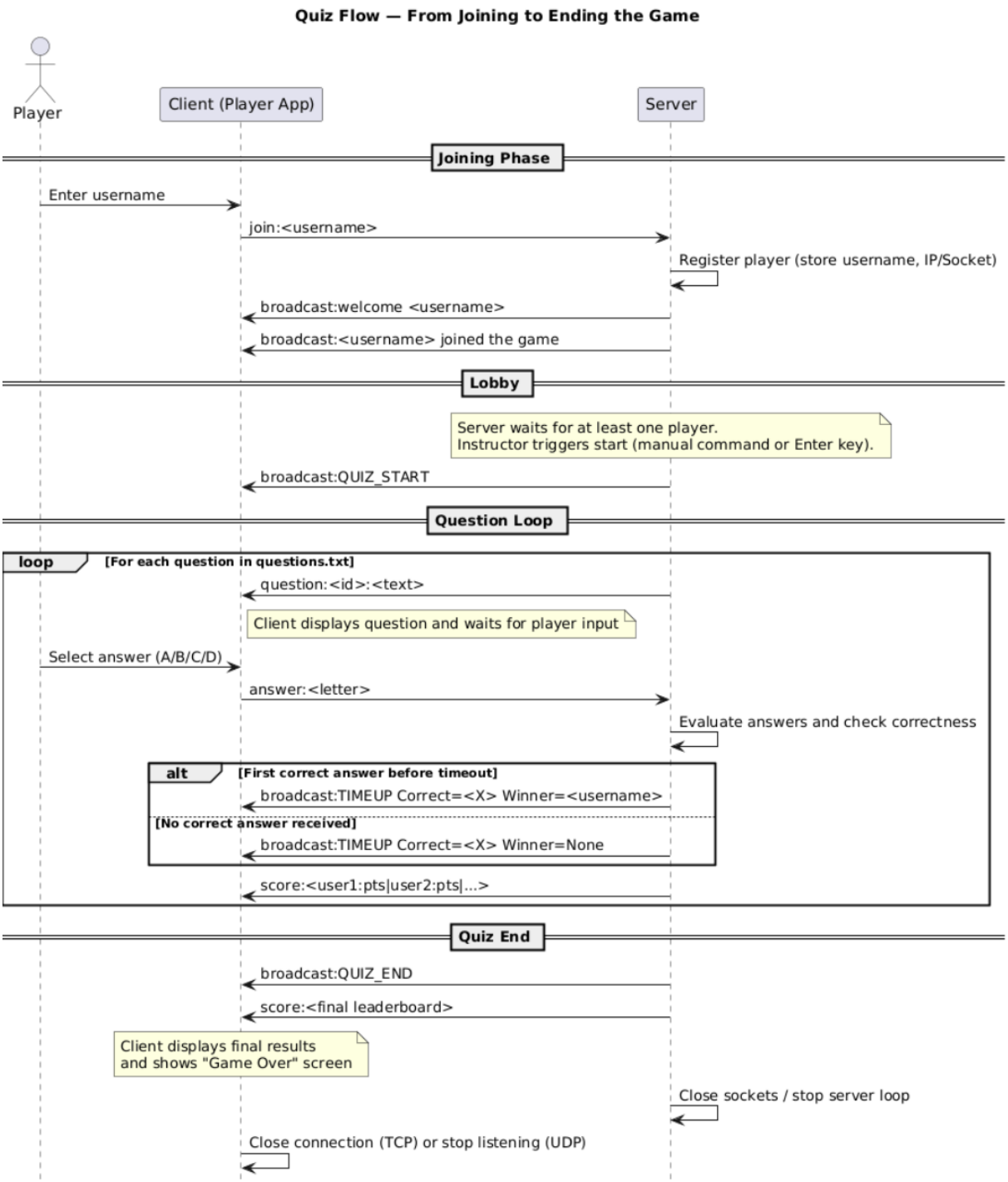

Figure 5: sequence diagram

3.2 Protocol Implementation

TCP Server:

- Listens on port 8888; accepts multiple clients.
- Dedicated thread per client handles join messages and answers.
- Uses threading.Lock to ensure shared state consistency for scores and last answers.

UDP Server:

- Listens on port 8888 using a non-blocking socket.
- Uses a single loop to handle join messages and answers.
- Tracks players via their IP/port; uses global dictionaries for usernames and scores.

3.3 Timers and Message Formats

Question Timeout

Handled entirely by the server, which broadcasts the remaining time to all clients each second and automatically skips when time runs out.

Message Formats

- **Join:** join:<username>
  Sent by a client when connecting.
- **Question:** question:<id>:<timeout>:<text>
  Broadcast by the server to all clients.
- **Answer:** answer:<A/B/C/D>
  Sent by the client in response to a question.
- **Timer:** timer:<seconds_left>
  Sent by the server during each question countdown.

- **Feedback:** feedback:<username>:<status>:<points>:<time>
  Sent to each client after a question (status = correct / wrong / timeout).

- **Broadcast:** broadcast:<message>
  Used by the server to send general updates (e.g., "QUIZ_START", "TIMEUP").

- **Scoreboard:** score:user1:1200|user2:950
  Broadcast after each round to update scores.

4. UDP vs TCP Comparison

4.1. Comparaison table

Feature	**TCP Implementation**	**UDP Implementation**	**Observed Behavior**
Connection Type	Connection oriented	connectionless	TCP Connection-oriented: requires a persistent connection. UDP no handshake; uses recvfrom() for all messages.
Reliability	Guaranteed delivery	Possible packet loss	TCP clients always received each question; UDP sometimes missed messages if network delayed
Ordering	Preserved	Not guaranteed	TCP maintained question order; UDP occasionally received out-of-order answers in tests
Latency	Slightly higher	Lower	UDP allows faster responses; TCP has negligible delay for small quizzes
Threading Model	Threads per client	Single main loop	TCP requires careful thread management; UDP simpler but requires polling for messages

4.2. Results and findings:

In the following figure, the client (left) connects to the UDP quiz server (right), sends a join request, and waits for the quiz to start. The server confirms the connection and lists active participants.

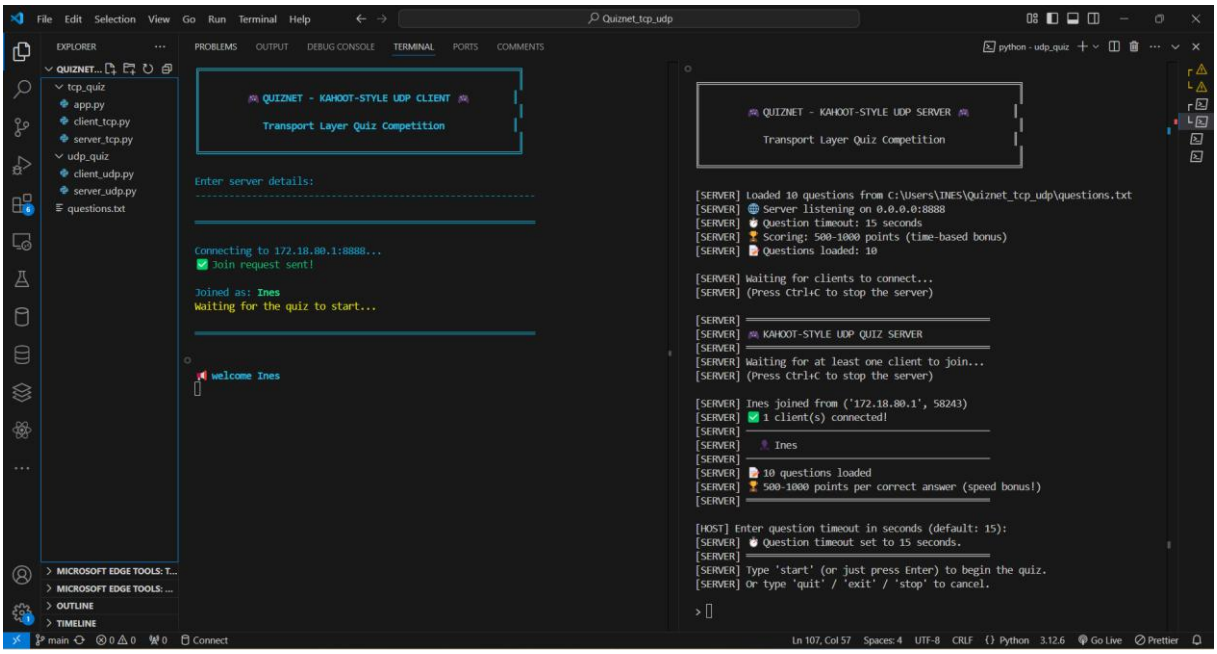

Figure 6:UDP result

=>The same logic is also implemented in the server side of TCP transport layer.

Next, the server broadcasts a question, and the client submits an answer. The server processes responses, calculates scores, and broadcasts results and leaderboard updates as shown in the Figure7.

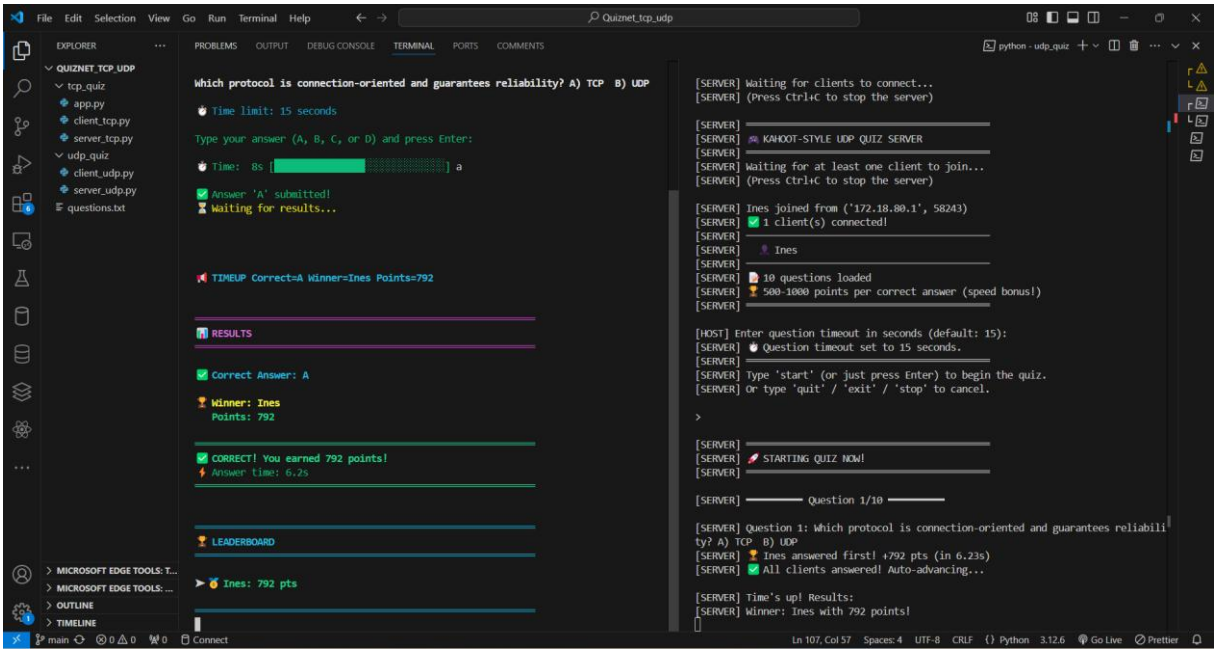

Figure 7:UDP result

In the client side of TCP transport layer version provides a graphical t interface built with Streamlit allowing players to enter the server IP and username, then join the quiz lobby before the game starts as following in the figure below:

Figure 8:TCP UI

Next step, the questions are displayed dynamically with a countdown timer and answer buttons. The leaderboard updates in real-time after each question as follows In figure9:

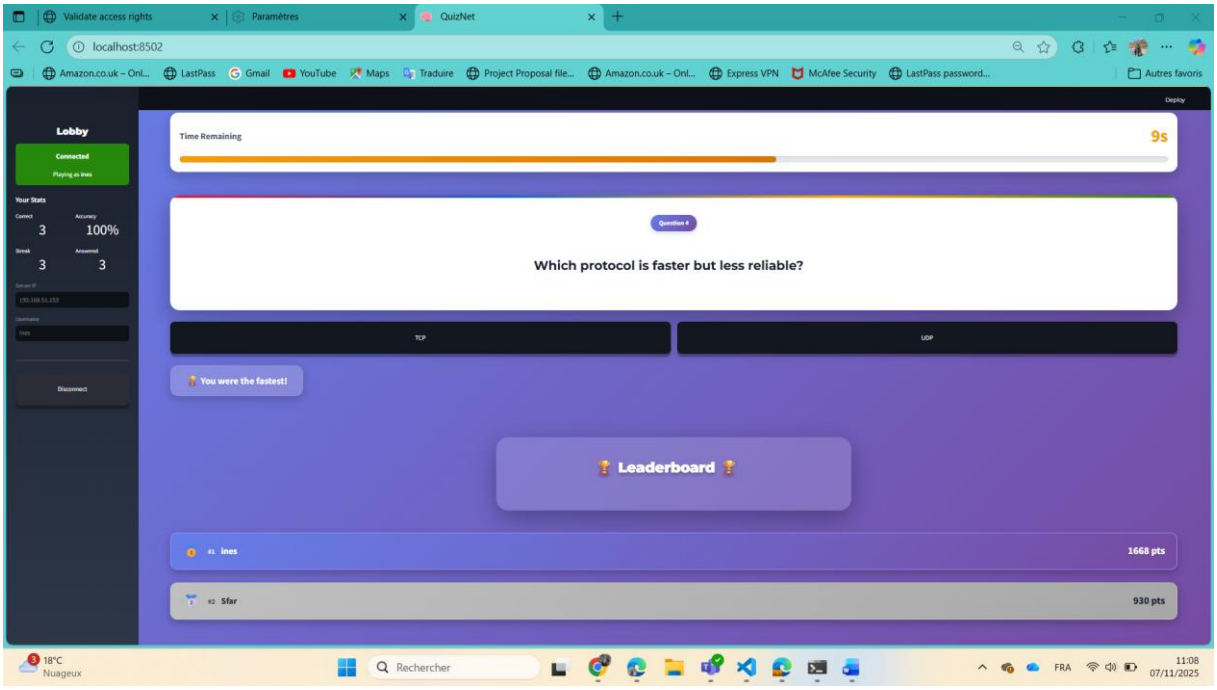

Figure 9:TCP UI

At the end of the quiz, the final standings are shown, including player ranks, scores, accuracy, and streak statistics. This figure presents the interface when the quiz is finished.

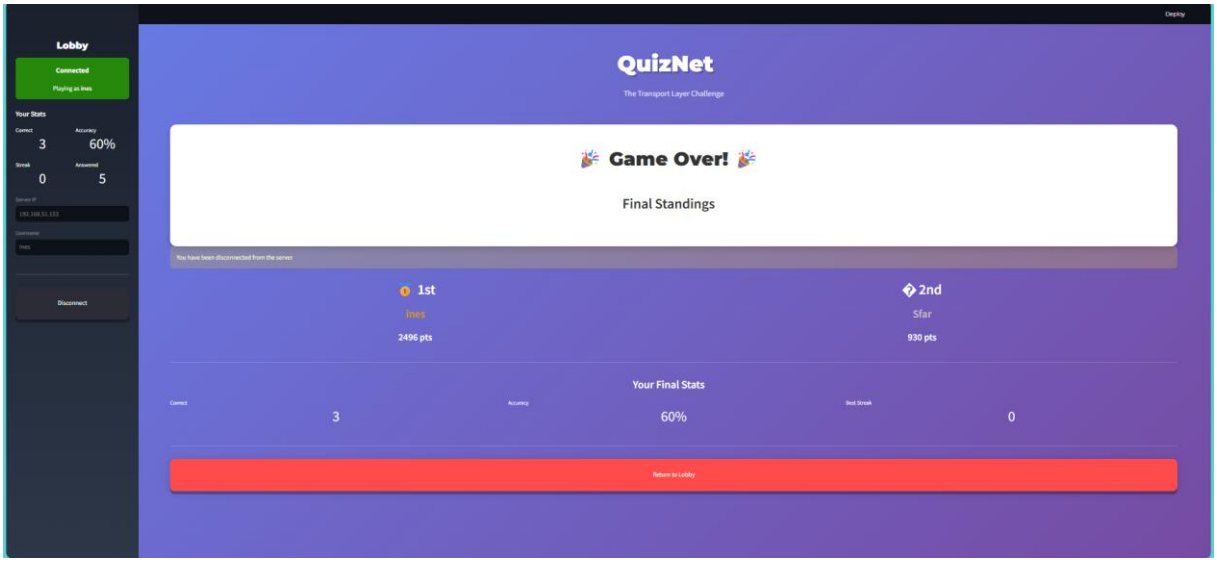

Figure 10:TCP UI

5. Challenges and Solutions

5.1 Challenges

- ### Concurrency Issues

Challenge: Multiple clients updating scores simultaneously.

Solution: Used threading.Lock in TCP to protect shared dictionaries (scores, players).

- ### Questions Not Displayed on Client UI

Challenge: Questions were only visible in the server terminal and not shown in the client interface.

Solution: A dedicated listener thread was implemented on the client side to continuously receive question: messages and update the Streamlit UI in real time.

- ### Duplicate Connections from Same Machine

Challenge: The same client could connect twice from the same IP address, causing duplicate entries.

Solution: The server now checks for existing connections with the same IP and rejects any new connection from that address.

- ### Username Validation

Challenge: Prevent duplicate usernames.

Solution: Server checks existing usernames before accepting a join request; sends error:username_taken to reject duplicates.

- ### Timer Management

Challenge: Determining first correct answer within timeout.

Solution: Server stores last_answer per player and updates the winner dynamically during the timer loop.

5.2. Individual Contributions

Member	Tasks Completed
Ines Turki	Did the **UDP server** and **UDP client**, and participated in writing the report.
Nour Guidara	Worked on the **UI** and the **TCP part**, and participated in writing the report.
Feriel Charrad	Worked on the **UI** and the **TCP part**, and participated in writing the report.
Oussama Sfar	Worked on the **UDP client**, helped **improve the UI**, and participated in writing the report.

7. Conclusion and Future Work

This project effectively demonstrated the fundamental differences between TCP and UDP at the transport layer.
Through implementation and testing, the team learned how TCP ensures reliability and order, while UDP prioritizes speed and simplicity.

Key Takeaways:

- TCP is ideal for reliable, interactive applications (like quizzes).
- UDP is best suited for real-time systems where occasional loss is acceptable.
- Multithreading and GUI integration enhance the user experience in socket-based applications.

8. References

Kurose, J. F., & Ross, K. W. Computer Networking: A Top-Down Approach, 8th Edition. Pearson, 2020.

Python socket library documentation: https://docs.python.org/3/library/socket.html

Python threading library documentation: https://docs.python.org/3/library/threading.html

Transport Layer lecture slides, [Course Name].

Project code repository (TCP/UDP quiz server and client implementations