1 Introduction

In this project you will implement online quiz game (Kahoot). The purpose of this project is to improve your programming skills and deeply expose the transport layer and the application layer (layers 4-5). The project applies the *client-server* architecture as seen at class. Note that the *client-server* architecture has two entities: **client** who consumes services, and **server** who provides services. In our case:

- The **client** is a user whose interested at participating quizzes.
- The **server** is the computer whose responsible of managing the quizzes. Its services include: access to quizzes, print new questions, etc...

Remark. In the quiz game players answer to a shared question simultaneously and get points according to how quickly they answered. After 3-5 questions, the player who achieved the highest amount of points is the winner.

2 Requirements

2.1 Server Side

1. Implement a multi-client online quiz game server using sockets.
2. Clients must connect to the server to participate in quizzes.
3. Design a quiz database with categories, questions, and correct answers.
4. The server should be capable of handling multiple concurrent quizzes with different sets of questions.
5. Implement a scoring system to track players' performance during quizzes.
6. Ensure secure communication between the server and clients.
7. Implement a time limit for each question, and handle timeout scenarios gracefully.
8. Design a leaderboard to display the scores of all participants.

2.2 Client Side

1. Clients should connect to the server and authenticate themselves to join quizzes.
2. Implement a user interface to display questions, options, and a timer.
3. Clients can choose a quiz category and participate in quizzes with other connected clients.
4. Implement a real-time update mechanism to display scores and the correct answers after each question.
5. Clients should be able to gracefully exit a quiz and disconnect from the server.

3 Technical Highlights

- Make sure to examine extreme cases. Use your own judgement to decide where a validation is necessary. Input type validation is not required – for example, if you expect to get a string you can assume it.
- You are responsible at establishing a reliable connection between the server and clients. Use the functions in *socket* library wisely.
- You should put your efforts at the following:
  - Provide a well-documented quality code.
  - Provide a documentation in the PDF file.
  - Fully understand the theoretical material.
- Implement a nice and clean GUI (Graphical User Interface). You will be graded on it also.

4 Instructions

4.1 Implementation

- You can implement your code in any operating system and any code language you wish. *Python* is recommended, as seen at class. Following is a link where you can learn python easily: https://www.w3schools.com/python
- You **must not** use any non-standard libraries for socket programming. Use only the libraries introduced at class.

4.2 Documentation

- You're required to well-document your code.
- You're required to attach a document in PDF format including the following:
  1. Full name and ID.
  2. Operating system, code language and version.
  3. An explanation about any file you created – including any classes and functions.
  4. Answers for the theoretical questions from the next section.
- Include illustration pictures (usage examples).

4.3 Submission

- Each student submits individually.
- Compress all the files in a ZIP format and submit it with your ID.
- **Final submission date: 10.03.2024**

5 Theoretical Questions

5.1 Security

1. How can you enhance the security of your application? Discuss potential vulnerabilities and propose solutions.
2. What are the risks associated with transmitting messages in plain text? How might you implement encryption to address this concern?

5.2 Scalability

1. How would you design your application to handle a large number of concurrent users? Discuss potential bottlenecks and scalability challenges.
2. What strategies could be employed to distribute the load and balance the server's resources effectively?

5.3 Reliability and Fault Tolerance

1. Describe how you would ensure the reliability of your application. What measures can be taken to handle server failures or unexpected crashes?
2. How might you implement message persistence to ensure that messages are not lost even if the server restarts?

5.4 User Authentication

1. Explain the importance of user authentication in your application. What methods could be used to authenticate users securely?
2. How would you handle user registration and password management to enhance the overall security of the system?

5.5 Concurrency and Synchronization

1. Discuss the challenges associated with handling multiple simultaneous connections in a server.
2. How might you implement thread safety and synchronization to prevent data corruption or race conditions?

5.6 Protocol Design

1. Explain the choice of communication protocol for your application (e.g., TCP, UDP). What factors influenced your decision?

2. Explain about the *socket* handshake – which protocol you used, what are the commands, are they *blocking* commands. How would you design the message format and protocol for communication between the client and server?

5.7 Message Ordering and Delivery

1. How would you address the issue of message ordering and delivery in your application? Discuss potential scenarios and solutions.

2. What mechanisms could be employed to ensure that messages are delivered in a timely and reliable manner?

5.8 Persistent Storage

1. Explain the importance of persistent storage in your server. How did you design and implement the storage on the server?

2. Discuss the trade-offs between different storage solutions in the context of your application.