"ALEXANDRU IOAN CUZA" UNIVERSITY OF IAȘI
FACULTY OF COMPUTER SCIENCE

Computer Networks Project
Quizzgame

Ioan Sava

December 2019

CONTENTS

1 Introduction

1.1 Purpose and Overview

The purpose of this paper is to present the concepts of computer networks involved in the development of a effective project. The technologies used with justification for their choice are presented. Also, use cases, project architecture and relevant pieces of code are found in this paper. Finally, some ideas on how the project can be improved are presented.

1.2 Project description

The Quizzgame project consists of the implementation of a multithreading server that supports any number of clients. The server will coordinate the clients who respond to a set of questions by turn, in the order in which they were registered. Each customer is asked a question and has a number of seconds to answer the question. The server checks the customer response and if it is correct it will retain the score for that client. The server also synchronises all clients with each other and gives each one n seconds to respond. Communication between the server and the client will be done using sockets. All the logic will be done on the server, the client just answers the questions. Questions with response variants will be stored in a SQLite database. The server will manage the situations in which one of the players leaves the game so that the game continues smoothly.[1]

2 Technologies used

2.1 TCP/IP

TCP/IP, or the Transmission Control Protocol/Internet Protocol, is a suite of communication protocols used to interconnect network devices on the internet. The entire internet protocol suite - a set of rules and procedures - is commonly referred to as TCP/IP, though others are included in the suite.

TCP/IP specifies how data is exchanged over the internet by providing end-to-end communications that identify how it should be broken into pack-

ets, addressed, transmitted, routed and received at the destination. TCP/IP requires little central management, and it is designed to make networks reliable, with the ability to recover automatically from the failure of any device on the network.

TCP/IP uses the client/server model of communication in which a user or machine (a client) is provided a service (like sending a webpage) by another computer (a server) in the network.[2]

The advantages of TCP/IP protocol and also the reasons why this protocol is implemented in the present project are: it is an industry–standard model that can be effectively deployed in practical networking problems; it is interoperable (i.e. it allows cross-platform communications among heterogeneous networks); it is an open protocol suite; It is not owned by any particular institute and so can be used by any individual or organization.[3] Also, an important factor in choosing the implementation of this protocol in the project is that it is a scalable, client-server architecture. This allows networks to be added without disrupting the current services. This advantage is crucial because project's server must support any number of clients.

A concurrent client/server model is implemented in the project, the interaction being connection oriented (TCP based) and the actors communicate through stream sockets.

2.2 Stream sockets

Connection-oriented sockets, which use Transmission Control Protocol (TCP), Stream Control Transmission Protocol (SCTP) or Datagram Congestion Control Protocol (DCCP). A stream socket provides a sequenced and unique flow of data without record boundaries, with well-defined mechanisms for creating and destroying connections and for detecting errors. A stream socket transmits data reliably, in order, and with out-of-band capabilities. On the Internet, stream sockets are typically implemented on top of TCP so that applications can run across any networks using TCP/IP protocol.[4]

2.3 POSIX THREADS

A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes. Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.[5]

In this project, the server creates threads to serve the requests of each client that connects, thus taking place the concurrency. Before the game starts, the server synchronizes the threads with a mutex. A mutex is a lock that is set before using a shared resource and released after using it. When the lock is set, no other thread can access the locked region of code. So this ensures synchronized access of shared resources in the code.[6]

By default, threads share the memory and the resources of the process to which they belong. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space.

The benefits of multithreading can be greatly increased in a multiprocessor architecture, where each thread may be running in parallel on a different processor. A single-threaded process can only run on one CPU, no matter how many are available.

Multithreading on a multi-CPU machine increases concurrency. In a single processor architecture, the CPU generally moves between each thread so quickly as to create an illusion of parallelism, but in reality, only one thread is running at a time.[7]

Also, thread creation is much faster, threads can be terminated easily and communication between threads is faster.

2.4 SQLITE3 DATABASE

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple

tables, indices, triggers, and views, is contained in a single disk file.[8]

In this project, a SQLite3 database is used to store a table with registered users and also a table with questions for the game itself.

The reasons why this database management system is used are: is a very light weighted database; it is much faster than file system; it only loads the data which is needed, rather than reading the entire file and hold it in memory; is portable across all 32-bit and 64-bit operating systems; it can be used with all programming languages without any compatibility issue.

3 Use-Case View

3.1 Purpose and overview

A use case is a software and system engineering term that describes how a user uses a system to accomplish a particular goal. A use case acts as a software modeling technique that defines the features to be implemented and the resolution of any errors that may be encountered.[9]

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Section 3.2 lists the current actors and gives a brief description of each in the overall use context of the project. In section 3.3, a series of use-cases are outlined and illustrated using use-case diagrams to clarify the interactions between components.

3.2 Actors

The actors can be human user, some internal applications or may be some external applications. In this case, the use-case actors are considered modules from the project:

Client - The client will drive all operation of the software. No distinction is made in regards to type of client. The client interacts with all available

modules(excepting the database manager).

Authentication - Through this module, the client can authenticate or create a new account, because the authentication module has access to the user table in the database.

Server - This use-case actor deals with the game itself, interacting with the eligible clients to play, (i.e. the logged in ones).

Database manager - A module through which the server can manipulate the table of questions from the database.

3.3 USE-CASE REALIZATIONS

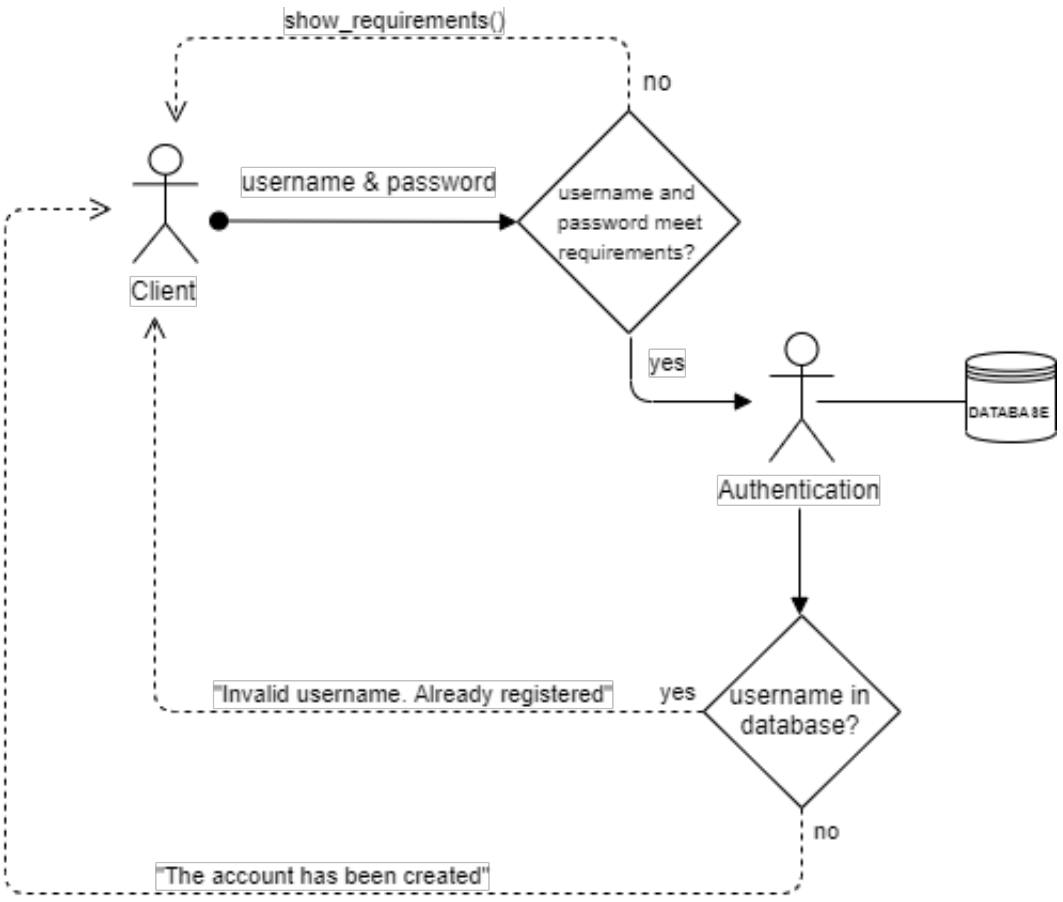

Figure 3.1: Registering Use Case

Each new user must create a new account consisting of an username and a password. These data must meet certain requirements, for example: to have a minimum length, to have at least one lowercase, one uppercase or/and one digit. Also, the username must be unique because otherwise conflicts may arise.

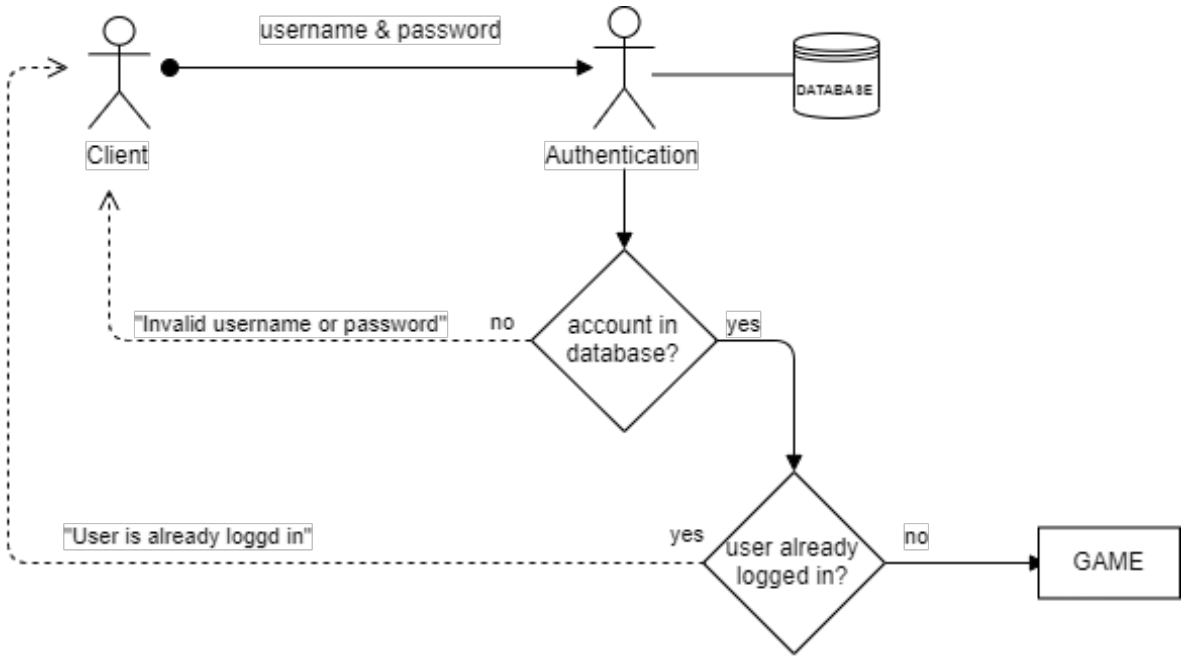

Figure 3.2: Login Use Case

A client cannot start the game until he has successfully logged in (i.e. a valid account that is not already logged in).

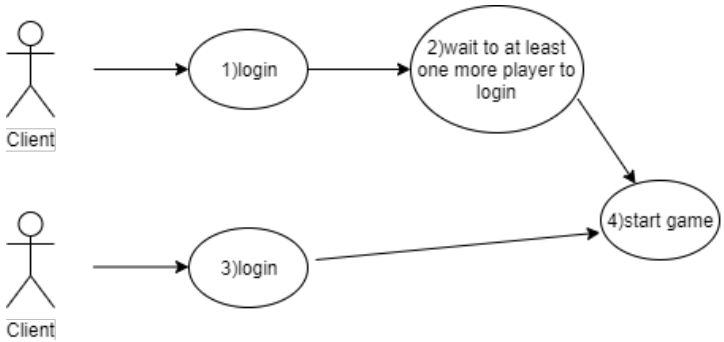

Figure 3.3: Start Game Use Case

A game cannot start until at least two clients have successfully logged in, meaning they are ready to play.

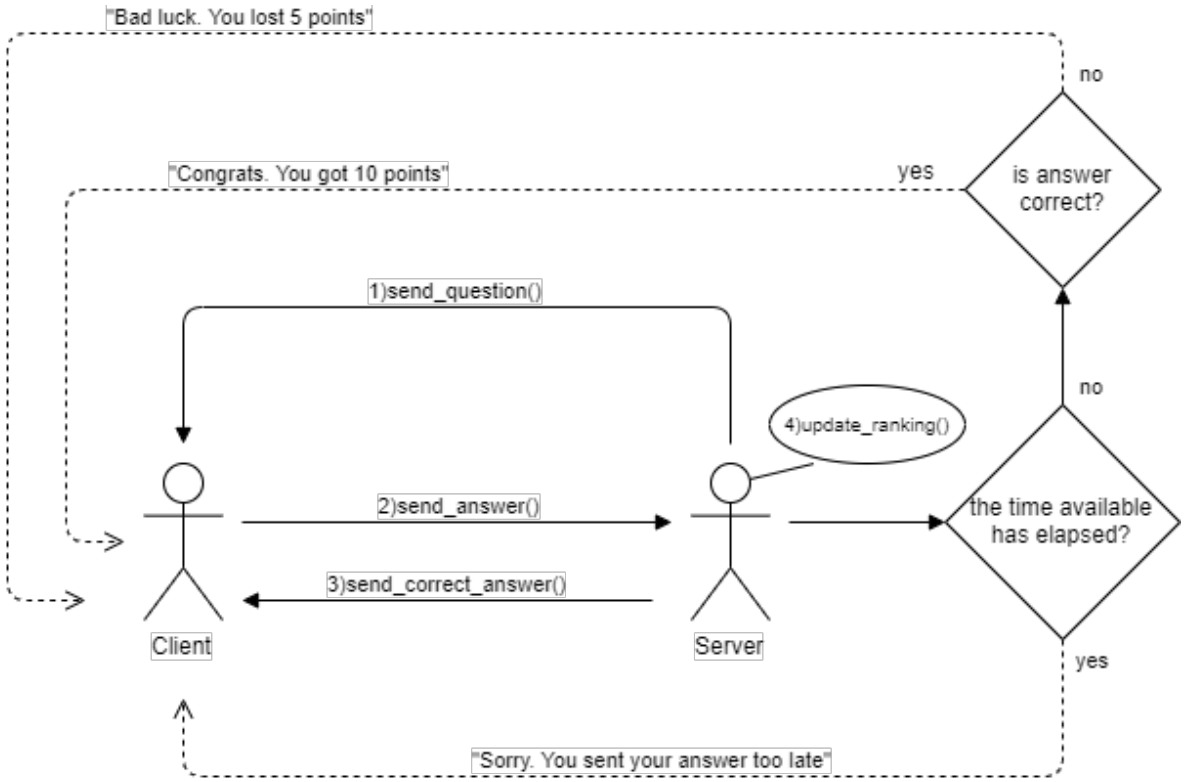

Figure 3.4: Game Use Case

Each client has a limited number of seconds to answer a question.

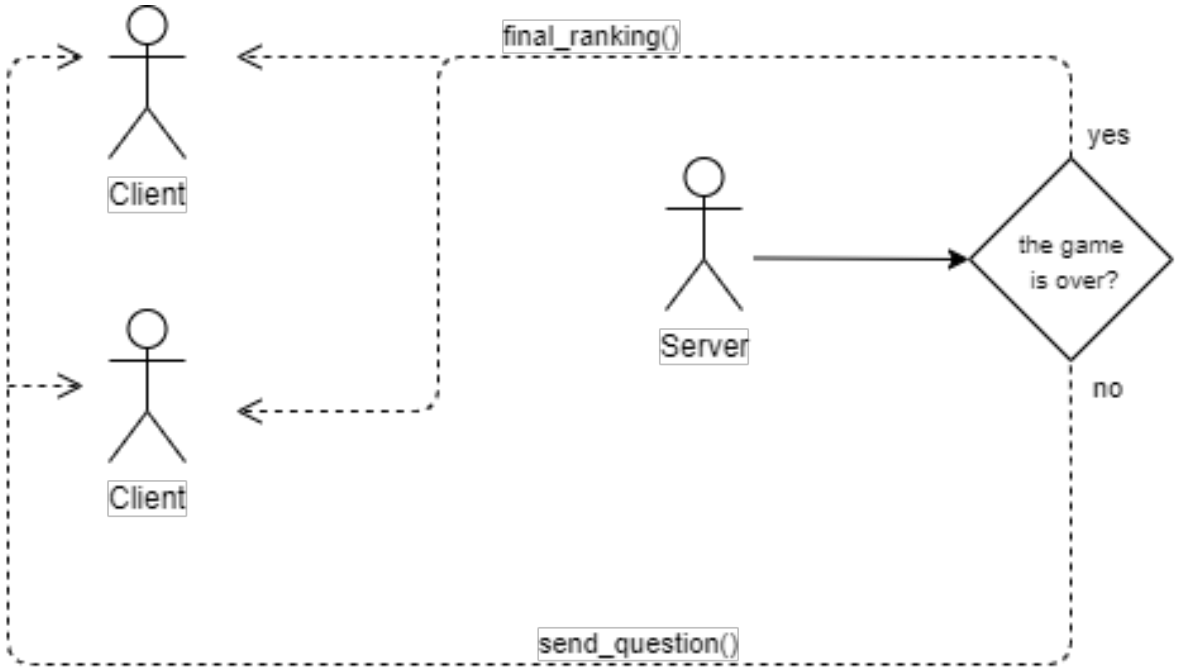

Figure 3.5: Game Over Use Case

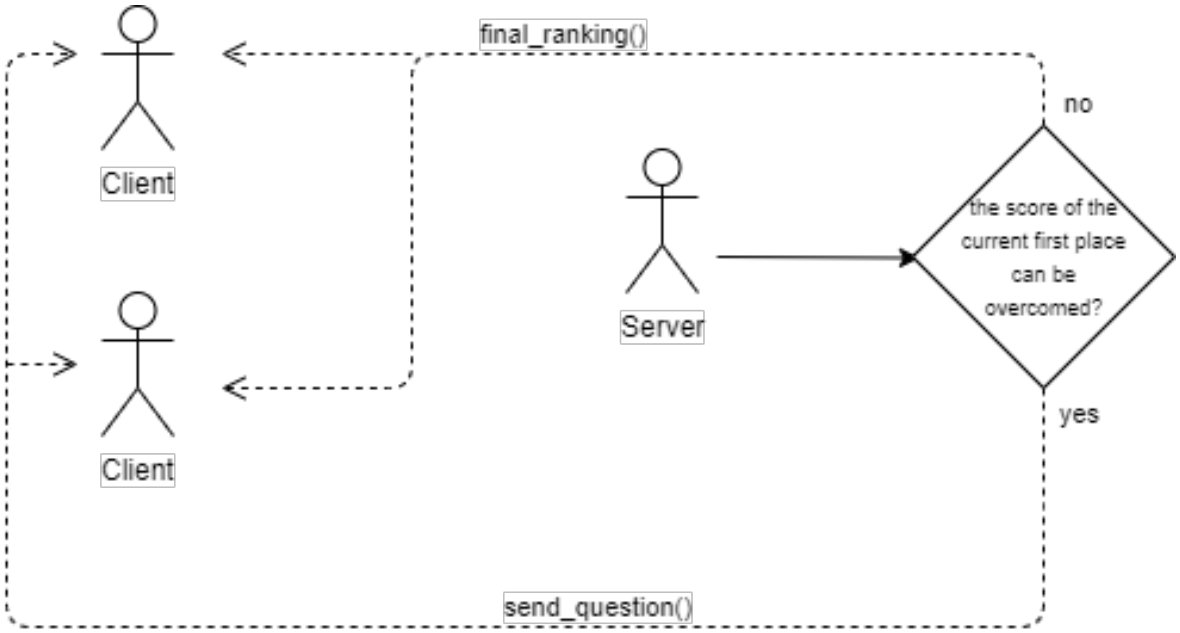

Figure 3.6: Anticipated Victory Use Case

The game ends when either the questions are completed or the player in the first place has a score that can not be surpassed until the end of the

game. At the end of the game each player will receive the ranking with the final situation.

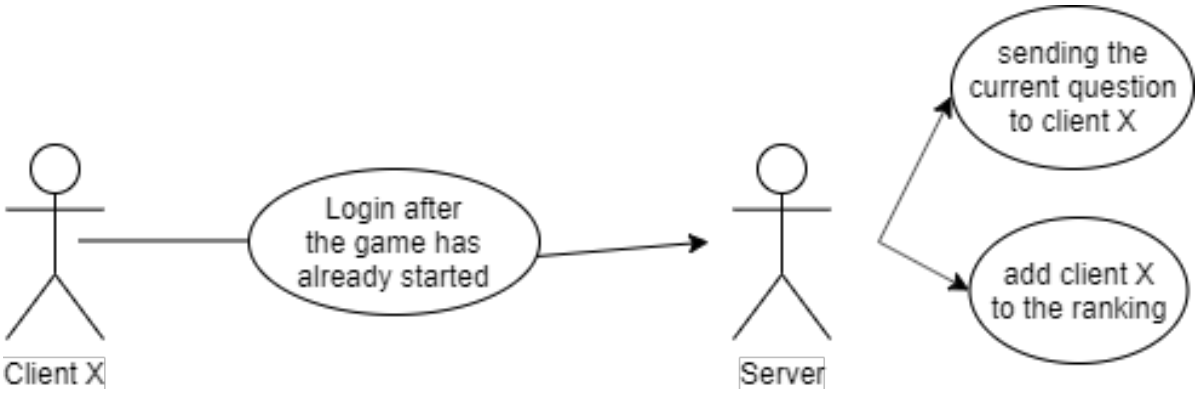

Figure 3.7: Login During Game Use Case

If a player logs in after the game has already started, he will continue from the point where all other players already connected are.

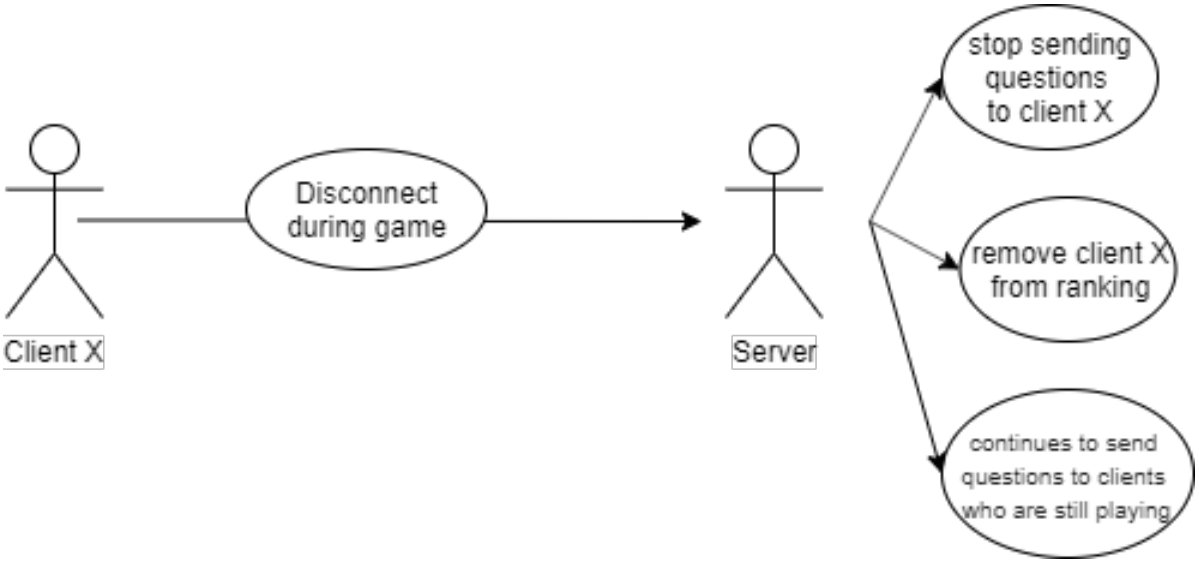

Figure 3.8: Client Quit Use Case

If a player leaves the game, the game will continue with the remaining players and the player who left the game will no longer be considered in the final ranking.

4 Project Architecture

4.1 Purpose

The software architecture of a system depicts the system's organization or structure, and provides an explanation of how it behaves. A system represents the collection of components that accomplish a specific function or set of functions. In other words, the software architecture provides a sturdy foundation on which software can be built.

A series of architecture decisions and trade-offs impact quality, performance, maintainability, and overall success of the system. Failing to consider common problems and long-term consequences can put your system at risk.[10]

4.2 Application Diagram

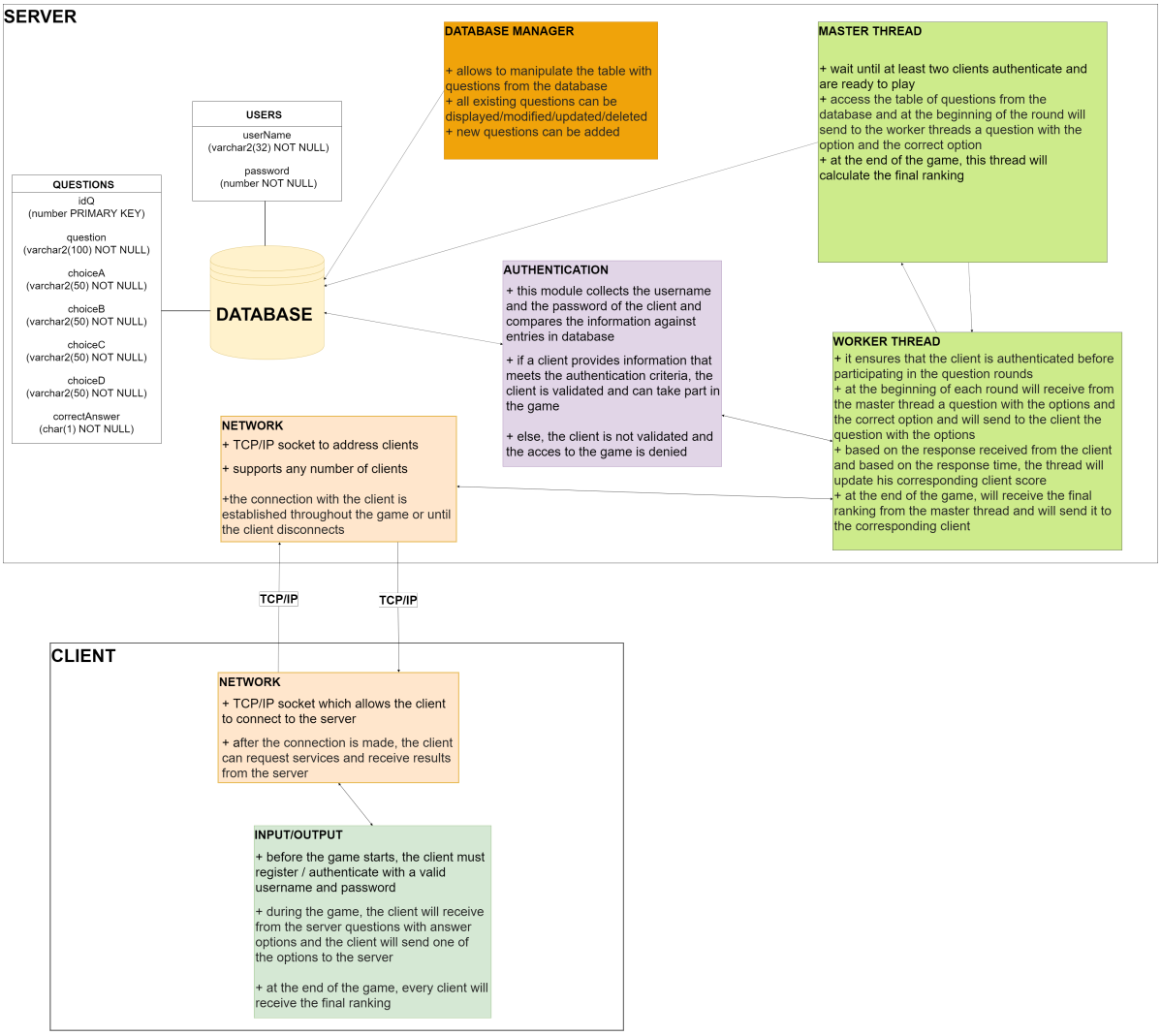

Figure 4.1: Project Architecture

5 Implementation details

Structures used:

```
#define MAX_LENGTH 100

typedef struct threadData
{
    int idThread;
    int clientDescriptor;
}threadData;

typedef struct
{
    pthread_t *array;
    int used;
    int size;
}ArrayThreads;

struct Player
{
    char userName[MAX_LENGTH];
    int score;
};

typedef struct
{
    struct Player *array;
    int used;
    int size;
}ArrayPlayers;

struct Question
{
    char question[MAX_LENGTH];
    char choiceA[MAX_LENGTH];
    char choiceB[MAX_LENGTH];
    char choiceC[MAX_LENGTH];
    char choiceD[MAX_LENGTH];
    char correctAnswer;
};
```

The concurrent server that creates a thread for each connected client:

```
// thread which manage the game
pthread_t gameId;
pthread_create(&gameId, NULL, &start_quizz, NULL);

ArrayThreads threads;
init_threads(&threads);
while (1)
{
    int client;
    threadData * td;
    int length = sizeof(from);

    printf ("Waiting at %d port\n", PORT);
    fflush(stdout);

    if ((client = accept(socketDescriptor, (struct sockaddr *) &from,
        &length)) < 0)
    {
        perror ("accept() error\n");
        continue;
    }

    td = (struct threadData*)malloc(sizeof(struct threadData));
    td->idThread = threads.used;
    td->clientDescriptor = client;

    pthread_create(&threads.array[threads.used], NULL, &treat, td);
    add_thread(&threads);
}

// close the connection
close (socketDescriptor);
free_threads(&threads);
```

The function executed by every thread:

```
int readyPlayers = 0;
// declaration of thread condition variable
pthread_cond_t conditionStartGame = PTHREAD_COND_INITIALIZER;
// declaring mutex
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

static void *treat(void * arg)
{
    struct threadData tdL;
    tdL = *((struct threadData*)arg);
    pthread_detach(pthread_self());
    struct Player currentPlayer;
    currentPlayer.score = 0;

    // postgame returns 0 if the current player logged in succesfully
    if (postgame((struct threadData*)arg, &currentPlayer) == 0)
    {
        // acquire a lock
        pthread_mutex_lock(&lock);
        ++readyPlayers;
        if (readyPlayers == 2)
        {
            // signals for first thread and game thread
            pthread_cond_signal(&conditionStartGame);
            pthread_cond_signal(&conditionStartGame);
        }
        else
        if (readyPlayers == 1)
        {
            printf("[thread %d] Waiting another player\n", tdL.idThread);
            pthread_cond_wait(&conditionStartGame, &lock);
        }
        pthread_mutex_unlock(&lock);

        printf("%s\n", currentPlayer->userName);
        game((struct threadData*)arg, currentPlayer);
    }

    printf("[thread %d] User disconnected\n", tdL.idThread);
    close((intptr_t)arg);
    return(NULL);
}
```

Game manager thread:

```
// declaration of thread condition variable
pthread_cond_t conditionStartGame = PTHREAD_COND_INITIALIZER;
// declaring mutex
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

ArrayQuestions questions;
int currentQuestion;

static void * start_quizz()
{
    pthread_mutex_lock(&lock);
    pthread_cond_wait(&conditionStartGame, &lock);
    pthread_mutex_unlock(&lock);
    init_players(&players);

    init_questions(&questions);
    add_questions_from_db(&questions);

    for (int i = 0; i < questions.used; ++i)
    {
        orderOfQuestions[i] = i;
    }

    shuffle(orderOfQuestions, questions.used);

    for (int i = 0; i < MAX_QUESTIONS_PER_ROUND; ++i)
    {
        currentQuestion = orderOfQuestions[i];
        wait_end_of_the_round();
    }

    update_final_leaderboard();

    free_questions(&questions);
    return NULL;
}
```

6 Conclusions

The solution proposed for this project can be improved and many new features can be added. Regarding the technologies used, the TCP / IP protocol can be replaced by the UDP protocol because this protocol is faster, simpler and more efficient than TCP. Also, UDP is lightweight. However, this protocol also comes with a number of disadvantages such as: there is no retransmission of lost packets, has only the basic error checking mechanism using checksums, the delivery of data to the destination cannot be guaranteed.

As for the functionalities part, the following can be taken into account: clients can choose rooms in which questions are asked only from certain topics, clients can come up with question proposals, new game modes (e.g. last man standing) or lifelines(e.g. 50/50, switch the question).

References

[1] Computer Networks UAIC 2019-2020
`https://profs.info.uaic.ro/~computernetworks/ProiecteNet2019.php`

[2] TCP/IP (Transmission Control Protocol/Internet Protocol)
Margaret Rouse
`https://searchnetworking.techtarget.com/definition/TCP-IP`

[3] Advantages and Disadvantages of the TCP/IP Model
Fendadis John
`https://www.tutorialspoint.com`

[4] Network socket
Wikipedia
`https://en.wikipedia.org/wiki/Network_socket`

[5] Multithreading in C
`https://www.geeksforgeeks.org/multithreading-c-2/`

[6] Mutex lock for Linux Thread Synchronization
`https://www.geeksforgeeks.org/`

[7] What is the advantages and disadvantages of Thread ?
https://basicittopic.com/

[8] About SQLite
https://www.sqlite.org/about.html

[9] Use Case
https://www.techopedia.com/definition/25813/use-case

[10] Software Architecture '|&' Software Security Design
https://www.synopsys.com/glossary/what-is-software-architecture.html

[11] http://man7.org/linux/man-pages/man7/pthreads.7.html

[12] https://pubs.opengroup.org/onlinepubs/009695399/functions/pthread_mutex_lock.html

[13] https://linux.die.net/man/3/pthread_cond_signal