

# QuizzGame

Balașcă Diana-Loredana<sup>1</sup>[0009–0009–7358–5488]

<sup>1</sup> Facultatea de Informatică-Universitatea "Alexandru Ioan Cuza", Iași, România

<sup>2</sup> [secretariat@info.uaic.ro](mailto:secretariat@info.uaic.ro)  
<https://www.info.uaic.ro/>

## 1 Introducere

### 1.1 Prezentarea generală a proiectului

Proiectul *QuizzGame* dorește implementarea unui joc în care fiecare concurent trebuie să răspundă la diferite întrebări cu variante de răspuns, într-un timp alocat, iar la final se va face un clasament cu toți jucătorii, pentru a stabili câștigătorul.

### 1.2 Motivarea alegerii

Am ales acest proiect deoarece consider că reprezintă un joc practic, ce poate fi pe placul oricui, deoarece poate fi aplicat pentru mai multe domenii, și totodată ajută dezvoltarea unei gândiri critice și antrenarea memoriei. Impunerea folosirii unei baze de date, am considerat-o ca o provocare, o modalitate de ieșire din zona de confort, și o oportunitate de a-mi dezvolta noi cunoștințe.

## 2 Tehnologii Aplicate

### 2.1 Protocol

Pentru realizarea proiectului am ales să folosesc modelul TCP deoarece acesta, spre deosebire de modelul UDP, asigură faptul că datele vor fi transmise în siguranță la destinatar, fără pierderi, anomalii în ordinea transmiterii sau o rearanjare a literelor.

O caracteristică importantă a modelului TCP o reprezintă controlul fluxului ce contribuie la gestionarea eficientă a transmiterii datelor pe dispozitivele conectate la rețea. Acesta este folosit pentru prevenirea supraîncărcării fluxul, TCP utilizând un mecanism de feedback între emițător și receptor pentru a ajusta ritmul de transmitere în funcție de starea actuală a rețelei.

### 2.2 Stocarea Datelor

Pentru stocarea întrebărilor, a răspunsului corect și respectiv a variantelor de răspuns am ales să folosesc o bază de date SQLite deoarece oferă posibilitatea de a-l introduce în diverse sisteme și necesită zero-configurări, fiind avantajoasă pentru gestionarea bazelor de date de dimensiuni mici.

De asemenea, am ales folosirea unei baze de date și pentru stocarea punctajului fiecărui client deoarece mi s-a părut o metodă mai sigură, mai facilă, și care mi-a permis să folosesc diverse operații asupra ei: inserare, ștergere și actualizare a diverselor câmpuri.

### 2.3 Interfață grafică


Pentru interfața grafică a clientului am preferat biblioteca GTK, deoarece oferă un mediu ușor de folosit, având multe funcții predefinite. Am ales să realizez o interfață grafică pentru ca jocul să fie mult mai interactiv și astfel logica acestuia ar putea fi înțeleasă mult mai ușor de utilizatori, oferind un mediu mai plăcut din punct de vedere vizual.

### 3 Structura Aplicației

#### 3.1 Concepte folosite în modelare

Serverul TCP pornește aplicația și oferă un timp de 30 de secunde clienților să se conecteze. La intrarea în joc a fiecărui client, serverul le trimite un set de reguli, iar clienții trebuie să trimită un nume de utilizator, care ulterior va fi inserat în baza de date.

Baza de date folosită pentru stocarea jucătorilor are următoarele entități:

| jucatori   |         |
|--|---------|
| id  | INTEGER |
| nume   | TEXT    |
| punctaj  | INTEGER |

id → identifică în mod unic clientul în baza de date  
 nume → folosit pentru afișarea clasamentului către client  
 punctaj → folosit pentru a calcula clasamentul

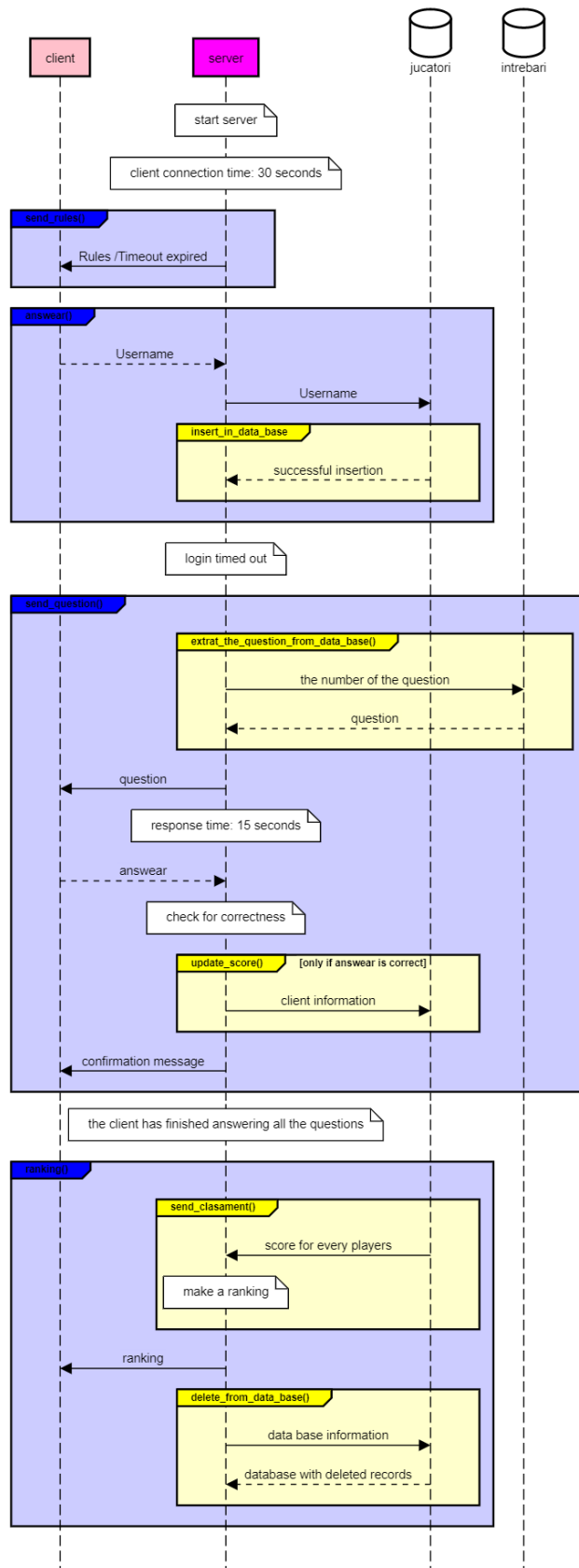
După expirarea timpului de conectare, tuturor clienților le este trimisă prima întrebare și le sunt acordate 15 secunde să răspundă.

Întrebările sunt de asemenea stocate într-o bază de date:

| intrebări  |         |
|--|---------|
| id  | INTEGER |
| intrebare  | TEXT    |
| raspuns  | TEXT    |
| varianta1  | TEXT    |
| varianta2  | TEXT    |
| varianta3  | TEXT    |
| varianta4  | TEXT    |

id → identifică în mod unic întrebările în baza de date  
 intrebare → întrebarea care va fi trimisă clientului  
 raspuns → răspuns corect la întrebarea respectivă  
 varianta1 → variantele de răspuns din care are de ales clientul  
 varianta2 → variantele de răspuns din care are de ales clientul  
 varianta3 → variantele de răspuns din care are de ales clientul  
 varianta4 → variantele de răspuns din care are de ales clientul

Pe urmă le este trimis un mesaj de înștiințare a corectitudinii răspunsului. Analog pentru toate cele 10 întrebări. Clienților li se oferă posibilitatea de a părăsi jocul prin răspunderea cu "-1" la oricare din întrebări. La sfârșit fiecărui client care a rămas în joc îi este trimis clasamentul cu toți jucătorii.

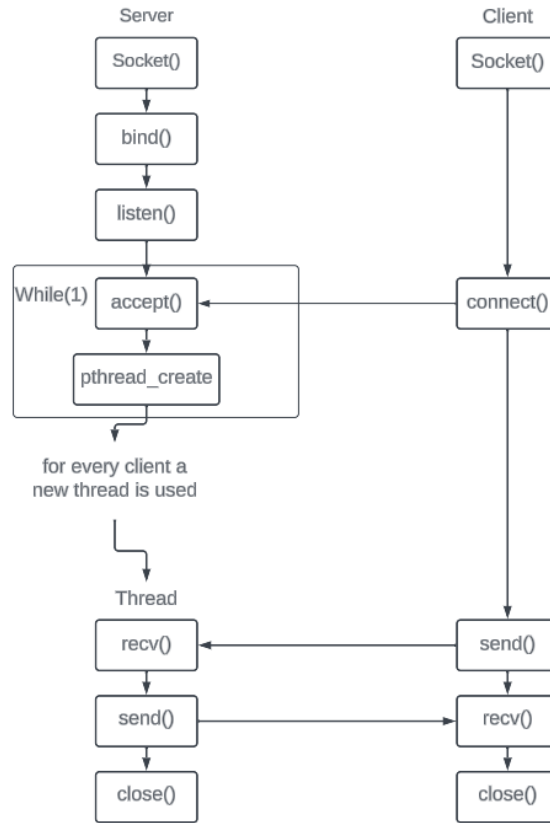


### 3.2 Explicarea aplicației din punct de vedere tehnic

Conexiunea este realizată prin intermediul unui server TCP concurrent care deservește clienții prin crearea unui fir de execuție pentru fiecare client. Pthreads (POSIX Threads) este un standard ce definește un API pentru crearea și manipularea firelor de execuție, iar primitiva pe care o folosește pentru crearea propriu-zisă a threadului este `pthread_create()` inclusă în biblioteca `<pthread.h>`.

La nivelul primitivelor I/O am ales să folosesc `send/recv` deoarece acestea sunt specifice doar descriptorilor de socket și folosesc un buffer atât pentru citirea datelor cât și pentru scrierea lor.

Arhitectura proiectului bazată pe modelul TCP:



În scrierea programului am ales să folosesc două baze de date, una pentru jucători și una pentru întrebări, și ambele folosesc o protecție mutex (mutual exclusion) care implică folosirea unui mecanism de blocare pentru a preveni accesul simultan la spațiul de stocare (baza de date). În ceea ce privește SGBD-ul, SQLite mi s-a părut o alegere potrivită deoarece oferă un mediu de lucru sigur pentru bazele de date de dimensiuni mici și un mediu ușor de manipulat și gestionat.

Pentru setarea timpului între întrebări, respectiv pentru timpul alocat conectării, am folosit structura `"timeval"` care conține două câmpuri: `tv_sec` (contorizarea în secunde) și `tv_usec` (contorizarea în microsecunde). Această structură este folosită împreună cu primitiva `"select"`, care blochează execuția programului și așteaptă până când cel puțin unul dintre FDs devine "gata" pentru o anumită operațiune (citire, scriere).

## 4 Aspecte de implementare

### 4.1 Secțiuni de cod specifice și inovative ale proiectului

Exemplu setare timer pentru conectarea clienților:

```
void start_communication()
{
    int i = 0;
    struct timeval timeout;
    fd_set set;
    int rv = 1;
    timeout.tv_sec = 30;
    timeout.tv_usec = 0;
    int client;
    thData *td;
    int aux = sd + 1;

    printf("[serverWait]We were waiting at the port %d...\n", PORT);
    fflush(stdout);

    while (1)
    {
        socklen_t length = sizeof(from);
        FD_ZERO(&set);
        FD_SET(sd, &set);
        rv = select(aux, &set, NULL, NULL, &timeout);

        if (rv == -1)
        {
            perror("Error at select");
            break;
        }
        else if (rv == 0)
        {
            printf("Timeout expired. Game started!\n");
            fflush(stdout);
            started = 1;
            nr_clients = i - 1;
            time_expired = 1;
            send_question();
        }

        if ((client = accept(sd, (struct sockaddr *)&from, &length)) < 0)
        {
            perror("[server]Error at accept().\n");
            continue;
        }
        td = (struct thData *)malloc(sizeof(struct thData));
        clients[i].cl = client;
        clients[i].idThread = i;
        clients[i].status = 2;
        all_clients = i;
        td->idThread = i++;
        td->cl = client;
        td->status = 2;
        pthread_create(&th[i], NULL, &treat, td);
    }
}
```

În această funcție din cod, setez un timer pentru permiterea conectării clienților de 30 de secunde, pe urma jocul începe primind pe rând întrebările. Am ales să folosesc această metodă deoarece voiam să realizez o sincronizare între clienți și timpul de receptare al întrebărilor. Pentru implementarea acestuia am avut nevoie de structura "timeval" din biblioteca <sys/time.h> și de de primitiva "select" din biblioteca <sys/select.h>.

Tot în această funcție se face accept-ul clienților și se face inițializarea vectorului unde rețin informațiile despre toți clienții, astfel încât să pot sincroniza trimiterea întrebărilor către toți clienții în același timp.

**Operații de inițializare si inchidere a bazei de date:**

```

void data_base_initialization(DataBase *data_base_manager, const char
*data_base_name)
{
    int ret = 0;
    ret = sqlite3_open(data_base_name, &(data_base_manager->db_ptr));
    if (ret != SQLITE_OK)
    {
        printf("Error opening database\n");
        exit(0);
    }
    if (pthread_mutex_init(&(data_base_manager->db_mutex), NULL) != 0)
    {
        printf("Error initializing mutex\n");
        exit(0);
    }
}

void close_data_base(DataBase *data_base_manager)
{
    sqlite3_close(data_base_manager->db_ptr);
    pthread_mutex_destroy(&(data_base_manager->db_mutex));
}

```

În această secțiune din cod, prezint operațiile de deschidere și închidere a bazei de date. Aceste funcții primesc ca parametru obiecte de tipul **DataBase** (o structură cu câmpurile: **sqlite3\* db\_ptr** și **pthread\_mutex\_t db\_mutex**), ce este utilă pentru a reutiliza subrutinele pentru ambele baze de date. Aceste funcții inițializează, respectiv distrug mutex-ul, pe care l-am folosit în scopul evitării accesării simultane a bazei de date de către mai mulți utilizatori.

**Diverse operații asupra bazei de date:**

Operația de inserare

```

void insert_in_data_base(void *arg, DataBase *data_base_manager, char
*buffer)
{
    struct thData tdl;
    tdl = *((struct thData *)arg);
    pthread_mutex_lock(&(data_base_manager->db_mutex));
    char sql_stmt[400];
    char *errMsg = 0;
    buffer[strlen(buffer) - 1] = '\0';
    snprintf(sql_stmt, sizeof(sql_stmt), "INSERT INTO jucatori VALUES
('%d', '%s', 0)", tdl.idThread, buffer);
    int ret = 0;
    ret = sqlite3_exec(data_base_manager->db_ptr, sql_stmt, 0, 0,
&errMsg);
    if (ret != SQLITE_OK)
    {
        printf("[Thread %d] Eroare SQL: %s\n", tdl.idThread, errMsg);
        sqlite3_free(errMsg);
        pthread_mutex_unlock(&(data_base_manager->db_mutex));
        exit(0);
    }
    printf("[Thread %d] The insertion in the database was successfully
completed.\n", tdl.idThread);
    pthread_mutex_unlock(&(data_base_manager->db_mutex));
}

```

Folosesc funcția **insert\_in\_data\_base()** pentru a insera un nou jucător în baza de date. În momentul inserării, acestuia i se atribuie un id ce este setat cheie primară și punctajul implicit este setat la 0.

### Operația de actualizare a punctajului

```
void score_update(thData tdl)

{
    char id_thread[10];
    pthread_mutex_lock(&(data_base_manager.db_mutex));
    char sql[100] = "";
    snprintf(id_thread, sizeof(id_thread), "%d", tdl.idThread);
    strcpy(sql, "UPDATE jucatori SET punctaj=punctaj+1");
    strcat(sql, " WHERE id=");
    strcat(sql, id_thread);
    sqlite3_stmt *stmt1;
    int ret = 0;
    ret = sqlite3_prepare_v2(data_base_manager.db_ptr, sql, -1, &stmt1,
0);
    if (sqlite3_step(stmt1) != SQLITE_DONE)
    {
        printf("SQL eroare: %s\n",
sqlite3_errmsg(data_base_manager.db_ptr));
        exit(0);
    }
    else
        printf("Valoarea a fost incrementata cu succes!\n");

    sqlite3_finalize(stmt1);

    pthread_mutex_unlock(&(data_base_manager.db_mutex));
}
```

Această funcție este apelată doar în cazul în care clientul a răspuns corect la întrebare. Conform id-ul clientului se stabilește cărei linie din bază îi corespunde și apoi se incrementează valoarea actuală a punctajului.

### Operația de ștergere a clienților

```
void delete_records()
{
    char *errMsg = 0;
    pthread_mutex_lock(&(data_base_manager.db_mutex));
    char sql[100];
    strcpy(sql, "DELETE FROM jucatori");

    if (sqlite3_exec(data_base_manager.db_ptr, sql, 0, 0, &errMsg) !=
SQLITE_OK)
    {
        fprintf(stderr, "Eroare SQL: %s\n", errMsg);
        sqlite3_free(errMsg);
        sqlite3_close(data_base_manager.db_ptr);
    }

    printf("Inregistrari sterse cu succes!\n");
    pthread_mutex_unlock(&(data_base_manager.db_mutex));
}
```

La sfârșitul jocului, după afișarea clasamentului, funcția face o ștergere a tuturor liniilor din baza de date pentru a pregăti jocul pentru o sesiune viitoare de conectare.

**Exemplu pentru trimiterea clasamentului:**

```

***
void ranking()
{
    char ranking[100] = "Ranking:\n";
    send_clasament(ranking);
    int size = strlen(ranking) + 1;
    for (int i = 0; i <= nr_clients; i++)
    {
        if (clients[i].status == 0)
        {
            if (send(clients[i].cl, &size, sizeof(int), 0) == -1)
            {
                perror("[server]Error send() from server.\n");
                exit(0);
            }
            if (send(clients[i].cl, ranking, size, 0) == -1)
            {
                perror("[server]Error send() from server.\n");
                exit(0);
            }
        }
    }
    delete_records();
    close_data_base(&data_base_manager);
}

```

Aceast  func ie se ocup  de trimiterea clasamentului c tre client. Ea apeleaz  la r ndul ei o alt  func ie, **send\_clasament()**, ce trimite prin intermediul parametrului referen iat toti clien ii  n ordinea punctajelor acestora.

```

***
void send_clasament(char *ranking)
{
    printf("send_clasament_sic\n");
    pthread_mutex_lock(&data_base_manager.db_mutex);
    sqlite3_stat *stat = NULL;
    char sql_stat[100];
    struct order
    {
        int score, status;
        char name[20];
    } players[100], aux;
    fprintf(sql_stat, sizeof(sql_stat), "SELECT name,punctaj FROM
jucatori");
    int j = 0;
    int ret = 0;
    ret = sqlite3_prepare_v2(data_base_manager.db_ptr, sql_stat, -1,
&stat, 0);
    while (sqlite3_step(stat) == SQLITE_ROW)
    {
        strcpy(players[j].name, (const char *)sqlite3_column_text(stat,
0));
        players[j].score = sqlite3_column_int(stat, 1);
        players[j].status = sqlite3_column_int(stat, 2);

        j++;
    }
    for (int i = 0; i < j; i++)
    for (int k = i + 1; k < j; k++)
    {
        if (players[i].score < players[k].score)
        {
            aux = players[i];
            players[i] = players[k];
            players[k] = aux;
        }
    }
    char x[10];
    for (int i = 0; i < j; i++)
    {
        if (players[i].status != 1)
        {
            strcat(ranking, players[i].name);
            strcat(ranking, " ");
            sprintf(x, "%d", players[i].score);
            strcat(ranking, x);
            strcat(ranking, "\n");
        }
    }
    pthread_mutex_unlock(&data_base_manager.db_mutex);
}

```

Func ia **send\_clasament()** extrage toti clien ii din baza de date folosind func ii specifice limbajului SQL  n C  i re ine  ntr-o structur  toate datele despre ace tia, iar apoi face o ordonare cresc toare  i o furnizeaz  func iei **ranking()**, ce se va ocupa de trimiterea c tre client.



## 4.2 Protocolul la nivelul aplicației

La fiecare conectare a unui client, serverul va trimite un set de reguli urmat de o cerere de autentificare. Serverul preia username-ul furnizat de client și va face o inserare în baza de date "jucatori", unde îi atribuie un id și îi setează punctajul în 0. După expirarea timpului de conectare, serverul va trimite clienților prima întrebare, extrasă la rândul ei din baza de date "intrebări", alături de variantele de răspuns și răspunsul corect, având setat un timer de 15 secunde pe întrebare. Clientul are obligația să răspundă la întrebare având ca variante de răspuns un număr de la 1 la 4, care corespunde indicelui variantei de răspuns, sau mai are posibilitatea de scriere a numărului "-1" ce va face o ieșire forțată din joc. Se procedează analog pentru toate cele 10 întrebări. După ce serverul termină de verificat răspunsul la ultima întrebare, va trimite către client clasamentul cu toți jucătorii ordonați în funcție de punctaj.

## 4.3 Scenarii reale de utilizare

Aceast joc are multiple utilizări în viața reală. *QuizGame* poate fi utilizat în educație și învățământ cum ar fi, pentru o lecție mai interactivă a copiilor din școala generală sau poate fi folosit pentru concursuri de admitere în care variantele de răspuns sunt de tip grilă.

Un joc ce are un concept similar de strategie cu cel al lui *QuizGame* este "Triviador", unde jucătorii primesc întrebări de cultură generală. Întrebările sunt de mai multe tipuri, atât grila cât și întrebări în care jucătorul trebuie să scrie răspunsul. Astfel, *QuizGame* poate fi adaptat și pentru strategii de joc mai complexe.

Acest joc, în principal, poate ajuta la digitalizarea mai multor concursuri în care încă se folosesc materiale fizice. În plus, cu câteva adaptări ale acestuia, se pot dezvolta și alte jocuri complexe, ce deserveșc dezvoltarea socială și culturală a utilizatorului.

## 5 Concluzii

O primă idee potențială de îmbunătățire a programului propune crearea mai multor seturi de întrebări, din mai multe domenii, astfel jocul prezintă o ofertă mai largă de subiecte, ceea ce ar putea atrage mai mulți jucători.

O alta idee ce ar putea contribui la îmbunătățirea jocului ar putea fi crearea unui cont de utilizator ce reține evoluția ta și astfel vei putea fi repartizat în joc cu alți concurenți care au aceeași experiență în joc ca și tine, astfel se menține dinamica jocului și sporește interesul jucătorilor pe termen lung.

## 6 Referințe bibliografice

- 1.Computer Networks: <https://profs.info.uaic.ro/computernetworks/cursullaboratorul.php>
- 2.Computer Networks Andrei Scutelnicu:  
<https://www.andreis.ro/teaching/computer-networks>
- 3.SQLite :<https://en.wikipedia.org/wiki/SQLite>
- 4.TCP/IP Model:<https://www.geeksforgeeks.org/tcp-ip-model/>