QuizzGame - Raport Tehnic

Onisor Maria-Patricia 2A1

January 9, 2024

1 Introducere

Viziunea generală constă în construirea unei aplicații interactive și distractive care permite testarea cunoștințelor generale ale jucătorilor și determinarea celui mai bun dintre jucători. Ca obiective aplicația își propune o comunicare eficientă cu utilizatorii: ei vor putea să se conecteze la joc prin intermediul unui username ales de ei, ca mai apoi să își testeze cunoștințele singuri sau cu prietenii.

2 Tehnologii Aplicate

2.1 TCP/IP - Transmission Control Protocol

Pentru a implementa comunicarea între client-server am preferat să utilizez protocolul TCP. Deși nu este la fel de rapid că UDP, am preferat o execuție puțin mai lentă pentru a asigura trimiterea pachetelor de informație cu succes. Avem garanția că întrebarea va ajunge la client, și răspunsul acestuia va putea ajunge la server, și astfel nu vor fi situații în care jucătorul va fi depunctat din cauza unei erori de sistem. Totodată, TCP/IP asigura transmiterea informației în ordine, ceea ce este vital când vine vorba de un joc cu întrebări-răspunsuri trimise in timp real către un număr mare de clienți sincronizați.

2.2 Baza de date SQLite

În implementarea proiectului am decis să folosesc biblioteca SQLite pentru gestionarea bazei de date. Această bibliotecă se remarcă prin trăsături semnificative, cum ar fi ușurința în utilizare și capacitatea de a organiza datele în tabele cu coloane și rânduri. Implementarea va asigura o gestionare eficientă și fiabilă a datelor.

3 Structura Aplicației

3.1 Server

Aceasta se ocupă de crearea, acceptarea și gestionarea conexiunilor. Vor fi folosite thread-uri pentru a susține concurență serverului, astfel fiecărui client îi va fi asociat un thread. De asemenea, componentă server se ocupă și de preluarea datelor și actualizarea acestora în cele două tabele sql.

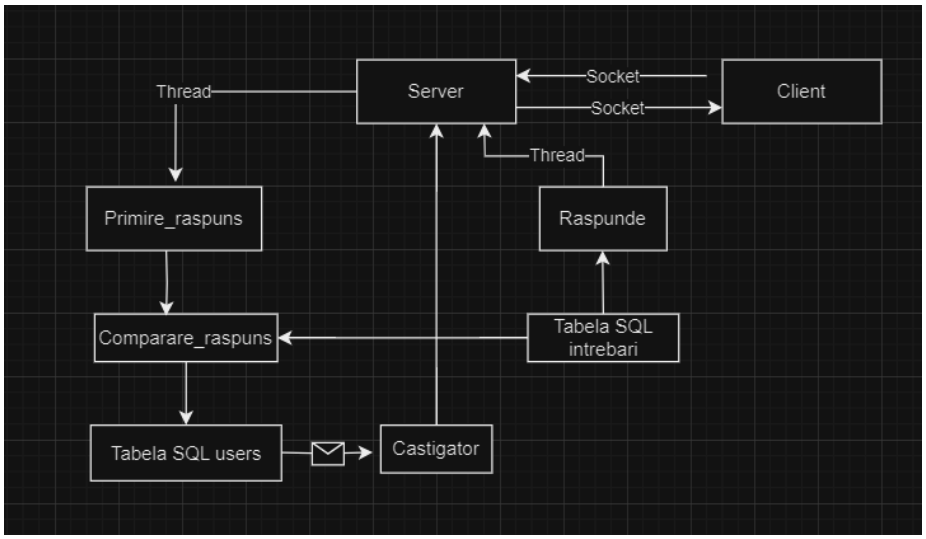
3.2 Client

Componenta client asigura comunicarea cu serverul și se ocupă de primirea și trimiterea datelor între server și jucător.

3.3 Baza de date SQL

O baza de date este implementată pentru stocarea de informații primite de la client, cum ar fi username-ul și punctajul acumulat, și pentru stocarea seturilor de întrebare-răspuns, care vor fi utilizate pe parcursul jocului.

3.4 Diagrama



4 Aspecte de Implementare

Aplicația este construită pe modelul TCP și se bazează pe comunicarea dintre client și serverul concurent, iar pentru aceasta se va folosi o baza de date SQLite pentru a se stoca informațiile primite de la client și pentru a citi întrebările și variantele de răspuns pentru quizz. Fiecare client va avea propriul sau thread, după care va fi identificat în baza de date. Clientul își poate introduce ce username dorește, și apoi, pe parcursul jocului va avea opțiunea să părăsească jocul, cu consecință că progresul sau va fi pierdut. Jucătorul va avea un timp limita pentru a răspunde la întrebare, dacă îl depășește va fi anunțat de acest lucru și va primi următoarea întrebare. Serverul preia răspunsul dat de client și îl compară cu răspunsul corect salvat în tabela întrebări. Dacă clientul a ghicit răspunsul corect, serverul îi va crește punctajul cu 1. La final, serverul îi va transmite clientului câștigătorul/câștigătorii, acesta/aceștia fiind jucatorul/jucătorii cu scorul cel mai mare.

4.1 Sectiuni specifice din cod:

Clientul trebuie să creeze socket-ul, să se conecteze la server, să trimită username-ul introdus de jucător, să afișeze întrebările, răspunsurile, și, la final, câștigătorul. Totodată, clientul se ocupă și de partea de timp, oferindu-i jucătorului 5 secunde să răspundă la întrebare.

- Citirea username-ului si conectarea la server:

```
citirea mesajului */
 printf ("[client]Introduceti un nume: ");
 fflush (stdout);
 read (0, nume, sizeof(nume));

 /* trimiterea mesajului la server */
 if (write (sd,nume,sizeof(nume)) <= 0)
   {
     perror ("[client]Eroare la write() spre server.\n");
     return errno;
   }
   fflush (stdout);
```

- Citirea raspunsului primit prin socket si analiza lui:

```c
int sd;     // descriptorul de socket
struct sockaddr_in server;  // structura folosita pentru conectare
    // mesajul trimis


/* exista toate argumentele in linia de comanda? */
if (argc != 3)
  {
    printf ("Sintaxa: %s <0> <2909>\n", argv[0]);
    return -1;
  }

/* stabilim portul */
port = atoi (argv[2]);

/* cream socketul */
if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("Eroare la socket().\n");
    return errno;
  }

/* umplem structura folosita pentru realizarea conexiunii cu serverul */
/* familia socket-ului */
server.sin_family = AF_INET;
/* adresa IP a serverului */
server.sin_addr.s_addr = inet_addr(argv[1]);
/* portul de conectare */
server.sin_port = htons (port);

/* ne conectam la server */
if (connect (sd, (struct sockaddr *) &server,sizeof (struct sockaddr)) == -1)
  {
    perror ("[client]Eroare la connect().\n");
    return errno;
  }
```

Dacă clientul s-a deconectat afișăm mesajul "Ai decis să te deconectezi." și închidem conexiunea cu server-ul.

Altfel, jocul continuă și afișăm întrebările și variantele de răspuns până la terminarea jocului, când afișăm câștigătorii și închidem conexiunea cu serverul.

```c
if (read (sd, &intrebare, 500) < 0)
  {
    perror ("[client]Eroare la read() de la server.\n");
    return errno;
  }
  if(strncmp(intrebare, " Castigatorii sunt:", strlen(" Castigatorii sunt:")) == 0)
  {
    printf("Joc terminat!%s\n", intrebare);
    break;
  }
  else if (strncmp(intrebare, "Te-ai deconectat!", strlen("Te-ai deconectat!")) == 0)
  {
    printf("Ai decis sa te deconectezi.\n");
    break;
  }
  else // afisam mesajul primit
  {  printf ("%s\n", intrebare);

    /* citirea mesajului de la tastatura*/
    printf ("Introduceti raspunsul cu litera MARE: ");
    fflush (stdout);
    timp();
    printf (" \n");
```

Mai jos este exemplificat cazul în care clientul decide să părăsească jocul:

```
Introduceti raspunsul cu litera MARE: EXIT

Ai decis sa te deconectezi.
```

Dacă jucătorul va încerca să iasă din joc el va trebui neapărat să scrie "EXIT". Dacă încearcă să iasă din joc folosind o combinație de taste, de exemplu Ctrl + C, el va fi anunțat că trebuie să scrie EXIT. Iată bucată de cod care se ocupă de asta:

```c
signal(SIGINT, semnal);
signal(SIGQUIT,semnal);
signal(SIGTERM, semnal);
```

Și ce se afișează:

```
Introduceti raspunsul cu litera MARE: ^C
 daca vrei sa iesi scrie <EXIT>
```

```
Introduceti raspunsul cu litera MARE: ^Z
 daca vrei sa iesi scrie <EXIT>
```

Iar dacă jucătorul termină quizz-ul, la final va primi username-ul clientului/clienților cu punctajul cel mai mare:

```
Joc terminat! Castigatorii sunt:
JUCATOR 2
```

• Limita de timp:

După cum am zis clientul va avea o perioada scurtă de timp pentru a răspunde la întrebare, altfel va primi 0 puncte pe acea întrebare.

```c
int timp() {
    fd_set readfds;
    struct timeval tv;
    tv.tv_sec = 5;
    tv.tv_usec = 0;
    FD_ZERO(&readfds);
    FD_SET(STDIN_FILENO, &readfds);

    int result = select(STDIN_FILENO + 1, &readfds, NULL, NULL, &tv);

    if (result == -1) {
        perror("Eroare la select");
        return -1;
    } else if (result > 0) {
        read (0, buf, sizeof(buf));
    } else {
        strcpy(buf, "pzskv");
        printf("Nu ai introdus nimic in timp util.\n");
    }
    return 0;
}
```

Serverul este responsabil de citirea username-ului trimis de client și stocarea lui împreună cu numărul thread-ului care se ocupă de acel client într-o baza de date, apoi citirea dintr-o altă tabela SQL a întrebării și posibilele răspunsuri asociate cu ea. După ce primește un răspuns de la client extrage răspunsul corect din tabela SQL și le compară. Dacă sunt la fel, crește punctajul clientului. La final, așteaptă să termine toți clienții și apoi actualizează tabela useri punând punctajul obținut de fiecare client, știind că Id-ul la care este salvat clientul este același cu numărul thread-ului care se ocupă de el. Apoi, extrage din tabela clienții cu punctaj maxim și ii trimite la clienți.

• Salvarea username-ului:

Dupa cum am zis, server-ul va salva in tabela users Id-ul, care este numarul thread-ului, username-ul trimis de client si initializeaza punctajul cu ”-1”.

```
void adaugare_sql(int id, char* username)
{
  char adaugare[500]="INSERT INTO USERS VALUES (";
  char nr[10]={ };
  sprintf(nr, "%d", id);
  strcat(adaugare, "'");
  strcat(adaugare, nr);
  strcat(adaugare, "'");
  strcat(adaugare, ", ");
  strcat(adaugare, "'");
  strcat(adaugare, username);
  strcat(adaugare, "'");
  strcat(adaugare, ", ");
  strcat(adaugare, "'");
  strcat(adaugare, "-1");
  strcat(adaugare, "'");
  strcat(adaugare, ");" );
  adaugare_users(adaugare);
```

În această funcție scriem comandă care va fi trimisă la adăugare-users care va prelua datele din SQL și le va pune în o variabilă globală.

```
void adaugare_users(char * comanda_sql)
{
    sqlite3* DB;
    int exit = 0;
    char* mesajError;
    exit = sqlite3_open(DATABASE_U, &DB);

    if (exit != SQLITE_OK)
    {
        fprintf(stderr, "Error open DB %s\n", sqlite3_errmsg(DB));
    }
    else
        printf("Opened Database Successfully!\n");

     int rc= sqlite3_exec(DB, comanda_sql, NULL, 0 , &mesajError);

    if (rc != SQLITE_OK)
        {
          fprintf(stderr, "Error SQL\n");
          sqlite3_free(mesajError);
        }
    else
        printf("Operation OK!\n");
    sqlite3_close(DB);
}
```

Apoi va deschide tabela ”întrebări” și va extrage pe rând întrebările și variantele de răspuns și le va trimite.

- Scrierea comenzii:

```
void citire_sql(int id)
{
    char citire[500]="SELECT INTREBARE, A, B, C FROM INTREBARI WHERE ID=";
    char nr[10]={ };
    sprintf(nr, "%d", id);
    strcat(citire, nr);
    strcat(citire, ";");
    executare_sql(citire);
}
```

- Executarea ei:

```
void executare_sql(char* comanda_sql)
{
  sqlite3* DB;
    int exit = 0;
    //char* mesaj_sql[MAX_LENGTH]={ };
    exit = sqlite3_open(DATABASE_I, &DB);

    char data[] = "CALLBACK FUNCTION";

    if (exit != SQLITE_OK)
    {
        fprintf(stderr, "Error open DB %s\n", sqlite3_errmsg(DB));
        //return ("ERROR");
    }
    else
        printf("Opened Database Successfully!\n");

     int rc= sqlite3_exec(DB, comanda_sql, callback, (void*)data, NULL);

    if (rc != SQLITE_OK)
        fprintf(stderr, "Error SELECT\n");
    else
        printf("Operation OK!\n");
    sqlite3_close(DB);
}
```

- Trimiterea la client:

```c
void raspunde(void *arg, int id)
{
  char intrebare[500]={ };
  citire_sql(id);
  if(strncmp(mesaj_intrare,"EXIT", strlen(mesaj_intrare))==0)
      strcpy(intrebare,"Te-ai deconectat");
  else strcpy(intrebare, mesaj_sql);

  strcpy(mesaj_intrare, " ");

  struct thData tdL;
  tdL= *((struct thData*)arg);
        /*pregatim mesajul de raspuns */
        id++;
  printf("[Thread %d]Trimitem prima intrebare...%s\n",tdL.idThread, intrebare);


        /* returnam mesajul clientului */
   if (write (tdL.cl, &intrebare, sizeof(intrebare)) <= 0)
    {
     printf("[Thread %d] ",tdL.idThread);
     perror ("[Thread]Eroare la write() catre client.\n");
    }
  else
    printf ("[Thread %d]Mesajul a fost trasmis cu succes.\n",tdL.idThread);

}
```

- Funcția de callback:

```c
static int callback(void* data, int argc, char** argv, char** azColName)
{
    for (int i = 0; i < argc; i++)
    {
        snprintf(mesaj_sql_nou + strlen(mesaj_sql_nou), MAX_LENGTH - strlen(mesaj_sql_nou), "%s : %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
    }
    strcpy(mesaj_sql, mesaj_sql_nou);
    strcpy(mesaj_sql_nou, " ");
    return 0;
}
```

Apoi așteptăm mesajul de la client. Odată primit îl comparăm cu răspunsul extras din baza de date.

- Primirea mesajului de la client:

```c
void primire_mesaj(void *arg)
{
  printf("am intrat in primire_mesaj()\n");
  struct thData tdL;
  bzero(mesaj_intrare,100);
  tdL= *((struct thData*)arg);
  if (read (tdL.cl, &mesaj_intrare,sizeof(mesaj_intrare)) <= 0)
      {
        printf("[Thread %d]\n",tdL.idThread);
        perror ("Eroare la read() de la client.\n");

      }
      mesaj_intrare[strlen(mesaj_intrare)-1]='\0';
  printf ("[Thread %d]Mesajul receptionat este...%s...\n",tdL.idThread, mesaj_intrare);
}
```

- Funcția care compară răspunsul:

```
int comparare_raspuns(int id, int scor)
{
  citire_raspuns_sql(id);
  if (strncmp(mesaj_intrare, mesaj_sql, strlen(mesaj_intrare)) == 0 )
    scor++;//daca da creste scorul
  printf("scor: %d\n", scor);
  return scor;
}
```

- Citirea răspunsului din SQL:

```
void citire_raspuns_sql(int id)
{
    char citire[500]="SELECT raspuns FROM INTREBARI WHERE ID=";
    char nr[10]={ };
    sprintf(nr, "%d", id);
    strcat(citire, nr);
    strcat(citire, ";");
    executare_sql(citire);
}
```

Dacă cumva clientul a decis să se deconecteze el va fi eliminat din tabela sql, astfel asigurându-ne că nu va fi afișat printre câștigători. Dacă nu a ieșit îi actualizăm punctajul și trecem la următoarea întrebare.

```
//verificare daca se doreste parasirea jocului
if (strncmp(mesaj_intrare, "EXIT", 4) == 0)
{
  //stergem user-ul din tabela users
  i=max+1;
  nr_clienti--;
  golim(tdL.idThread);
}
else //continuam
{
  char lala[500]={ };
  strcpy(lala, " raspuns : ");
  strcat(lala, mesaj_intrare);
  strcpy(mesaj_intrare, lala);
  nr = comparare_raspuns(i, nr);
  printf("thread:%d, punctaj:%d", tdL.idThread, nr);
}
```

- Stergerea din SQL:

```
void golim(int id)
{
  char adaugare[MAX_LENGTH], nr[10]={ };
  sprintf(nr, "%d", id);
  strcpy(adaugare, " DELETE FROM USERS WHERE ID=");
  strcat(adaugare, nr);
  strcat(adaugare, ";");
  adaugare_users(adaugare);
}
```

Dupa ce am parcurs toate întrebările așteptăm să termine toți concurenții:

```
while(1)
{
  if(castigator((struct thData*)arg))
    sleep(2);
  else break;
}
```

si apoi actualizam tabela useri cu scorul obținut de fiecare jucator.

```
void executare_castigatori()
{
    sqlite3* DB;
    int exit = 0;

    exit = sqlite3_open(DATABASE_U, &DB);

    char data[] = "CALLBACK FUNCTION";

    if (exit != SQLITE_OK)
    {
        fprintf(stderr, "Error open DB %s\n", sqlite3_errmsg(DB));
    }
    else
        printf("Opened Database Successfully!\n");

    char comanda_sql[500] = "SELECT USERNAME FROM USERS WHERE PUNCTAJ=(SELECT MAX(PUNCTAJ) FROM USERS);";

    int rc = sqlite3_exec(DB, comanda_sql, callback_c, (void*)data, NULL);

    if (rc != SQLITE_OK)
        fprintf(stderr, "Error SELECT\n");
    else
        printf("Operation OK!\n");
    printf("\n%s\n", raspuns);
    sqlite3_close(DB);
}
```

Am creat o noua funcție callback, care stochează multiple rânduri, astfel acoperind cazul când sunt mai mulți participanți cu același scor:

```c
static int callback_c(void* data, int argc, char** argv, char** azColName)
{
    int i;
  strcpy(mesaj_sql_2, "");

    for (i = 0; i < argc; i++)
    {
        snprintf(mesaj_sql_2 + strlen(mesaj_sql_2), MAX_LENGTH - strlen(mesaj_sql_2), "%s\n", argv[i] ? argv[i] : "NULL");
        strcpy(destinatie[i], mesaj_sql_2);
        strcat(raspuns, destinatie[i]);
    }
    return 0;
}
```

Caz exprimat in exemplul de mai jos:

```
Joc terminat! Castigatorii sunt:
JUCATOR 1
JUCATOR 2
```

Pe lângă faptul că apelează funcția pentru salvarea punctajelor, funcția ”câștigător”, se ocupă și de cazul în care clientul a decis să se deconecteze. Apoi trimite mesajul, fie câștigătorii, fie anunțul de deconectare, la client prin socket.

```c
int castigator(void *arg)
{
  if (strncmp(mesaj_intrare, "EXIT", strlen("EXIT")) == 0)
  {
    strcpy(raspuns,"Te-ai deconectat!");
    printf("%s\n", raspuns);
  }
  else
  {
    strcpy(raspuns, " Castigatorii sunt: \n");
    if(am_terminat(terminat)==0)
      executare_castigatori();
    else return 1;
  }
  struct thData tdL;
  tdL= *((struct thData*)arg);
        /*pregatim mesajul de raspuns */

  printf("[Thread %d]Trimitem Castigatorul\n %s\n",tdL.idThread, raspuns);


        /* returnam mesajul clientului */
   if (write (tdL.cl, &raspuns, sizeof(raspuns)) <= 0)
    {
     printf("[Thread %d] ",tdL.idThread);
     perror ("[Thread]Eroare la write() catre client.\n");
    }
  else
    printf ("[Thread %d]Mesajul a fost trasmis cu succes.\n",tdL.idThread);
    fflush (stdout);
    trimis++;
    return 0;
}
```

Apoi, serverul verifică dacă a fost trimis clasamentul la toți clienții, dacă da atunci ii șterge pe toți din tabela.

```
while(1)
{
  if(trimis==nr_clienti)
  {
    golim(tdL.idThread);
    trimis--; nr_clienti--;
    break;
  }
  else
  sleep(1);
}
```

4.2 Scenarii reale de utilizare

Acesta aplicație poate fi folosită în orice domeniu pentru desfășurarea unui test: în domeniul academic(verificarea elevilor/studenților), în formarea domeniului de afaceri(evaluarea angajaților), divertisment personal.

5 Concluzii: Potentiale imbunatatiri

• Codul poate beneficia de o împărțire în funcții mai mici și mai modulare, ceea ce poate facilita reutilizarea lor în mai multe contexte, nemaifiind necesar să fie create funcții asemănătoare, de exemplu cele două funcții callback.

• Totodată, se poate introduce un mecanism de autentificare și stocare a datelor, permițând clientului să poată vedea evoluția să, lucru necesar dacă aplicația este folosită în scop educațional. Dacă dorim extinderea audienței, se pot adaugă diferite categorii/domenii, astfel participanții putând alege din ce domeniu doresc să fie testați. De asemenea, pentru o dificultate mai mare se pot introduce întrebări fără varinte de răspuns.

• Pentru o utilizare mai simplă, se poate creea o interfață grafică, astfel fiind mai ușor pentru un client să folosească aplicația.

6 Referințe Bibliografice

Site-ul disciplinei
Site utilizat pentru intelegerea folosirii Thread-urilor
Site utilizat pentru intelegerea comunicarii cu Socket
Site utilizat pentru intelegerea lucrului cu Baze de Date
Site folosit la realizarea diagrameil