Mersul Trenurilor

Șorodoc Tudor-Cosmin
(Grupa A4)

06 Decembrie 2022

---

1 Introducere

1.1 Despre aplicație

Aplicația **Mersul trenurilor**, așa cum face referire și numele acesteia, oferă posibilitatea utilizatorilor de a afla informații despre traseul unor trenuri(orele la care acestea ajung într-o anumită stație, dacă au întârziere, dacă nu cumva mersul unui tren a fost anulat,...etc).

1.2 Utilizare

Aplicația pune la dispoziția utilizatorilor un **meniu**, prin care aceștia pot afla diverse informații, și anume :

- Orele la care ajung trenurile într-o stație "X" (specificată de utilizator) în data de "D" (de asemenea, specificată de utilizator) și în traseul acestor trenuri, trec și prin stația "Y";
- Trenurile care pleacă în urmatoarea oră, (și întârzierea cu care pleacă, dacă este cazul) dintr-o stație "X";
- Trenurile care sosesc în urmatoarea oră, (și dacă ajung mai devreme sau cu întârziere) într-o stație "X";
- De asemenea, în meniul aplicației apare și opțiunea de a alege comanda de *login*; Un utilizator se va putea loga, doar dacă acesta este Administrator(i.e. utilizatorii normali nu au acces la un username si la o parola prin care s-ar putea conecta).
  Un Administrator va avea în meniul aplicației, bineînțeles după ce s-a logat, pe lângă opțiunile pe care le are un utilizator normal, și comenzi prin care :
  - poate adăuga un tren nou, cu traseul acestuia;
  - poate introduce întârzierea pe care o are un tren "T", la sosirea/plecarea în/din stația "X"; cu alte cuvinte, Administratorii sunt singurii care pot schimba baza de date;
  - poate crea un cont nou de administrator, cont care poate fi utilizat de alt utilizator;
  - poate schimba username-ul/parola;
  - poate da logout;

2 Tehnologii utilizate

2.1 TCP/IP

La baza sistemului de comunicare al aplicației stă protocolul TCP/IP (Transmition Control Protocol/ Internet Protocol). TCP este un protocol orientat-conexiune, adica pentru transmiterea mesajelor între 2 host-uri este nevoie de realizarea unei conexiuni. Conexiunile TCP sunt full duplex și se realizează prin așa numitul **"three way handshaking"**, astfel asigurând o confirmare de primire a datelor. Protocolul TCP garantează livrarea corectă a datelor la destinatar, fără pierderi sau duplicări.

Aplicatia folosește acest protocol întrucât păstrarea integrității și a ordinii informațiilor trimise de la *Server* către *Client* este de o importanță majoră. Daca aplicația nu ar avea la bază o comunicare sigură, atunci ar exista riscul ca datele trimise de la Server la Client să fie alterate, astfel Clientul primind o dată/oră greșită cu privire la mersul unui tren. De asemenea, rapiditatea nu este esențială, cu alte cuvinte, Clientul ar prefera ca să primească un raspuns corect, într-un timp mai mare, decât unul rapid, dar posibil gresit.

2.2 MySQL

Aplicația presupune afișarea informațiilor cu privire la mersul trenurilor, deci trebuie să ținem evidența fiecărui tren, în ce stație ajunge, la ce oră, dacă are întârziere și la ce oră pleacă din stația respectivă. De asemenea, trebuie să stocăm parolele și username-urile administratorilor. Din aceste motive, este esențială utilizarea unei baze de date. MySQL este un sistem de gestiune a bazelor de date relaționale și se poate folosi impreună cu limbajul C/C++.

3 Arhitectura aplicației

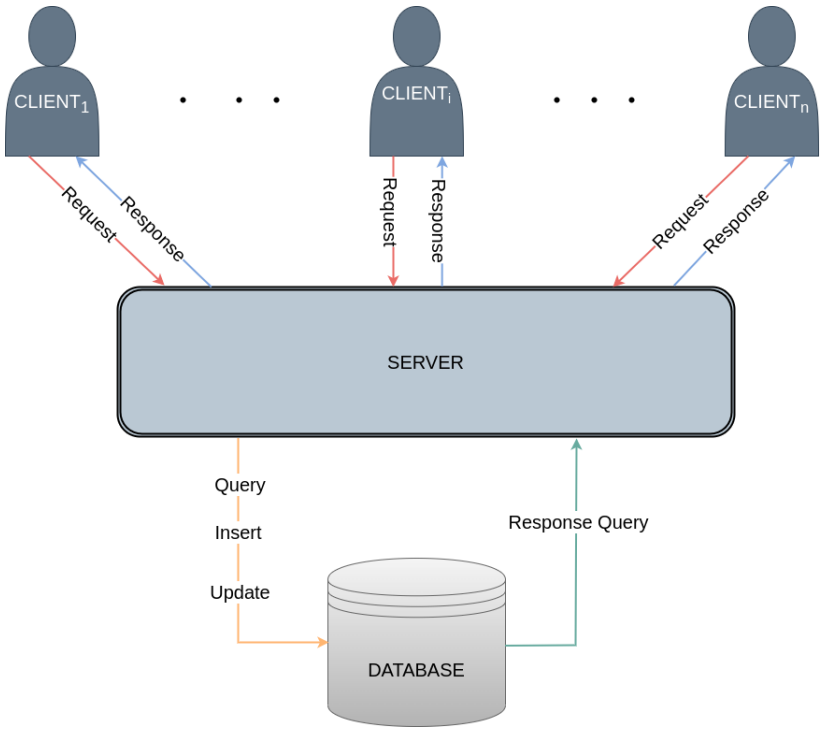

Figure 1: Diagrama generală a aplicației

După cum se observă și în imaginea de mai sus, aplicația are la bază modelul *Client − Server*. În continuare, sunt prezentate niște diagrame mai detaliate, atât pentru *Client*, cât și pentru *Server*.

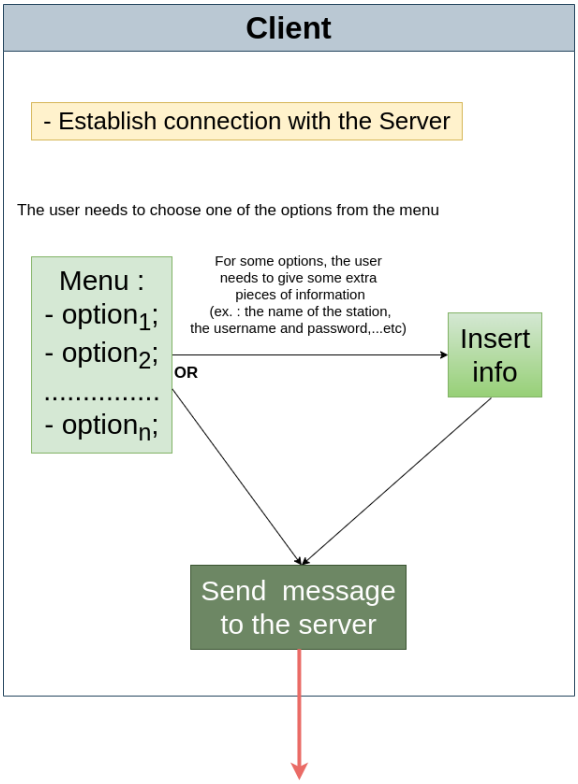

Figure 2: Diagrama - *Client*

Inițial *Clientul* încearcă să stabilească o conexiune cu *Serverul*, după care, daca s-a reușit conexiunea, îi afișează un meniu user-ului, din care poate alege diverse opțiuni. În fucție de opțiunea aleasă, user-ul trebuie să introducă date relevate pentru acea opțiune (Ex: User-ul este de fapt, un Administrator și alege opțiunea de login. Alegând acestă opțiune user-ul trebuie să introducă un *username* și o *parolă*). Toate aceste date introduse de user sunt transmise prin intermediul unui **socket**("atașat" unui anumit *PORT* și *adresa IP*) către *Server*.

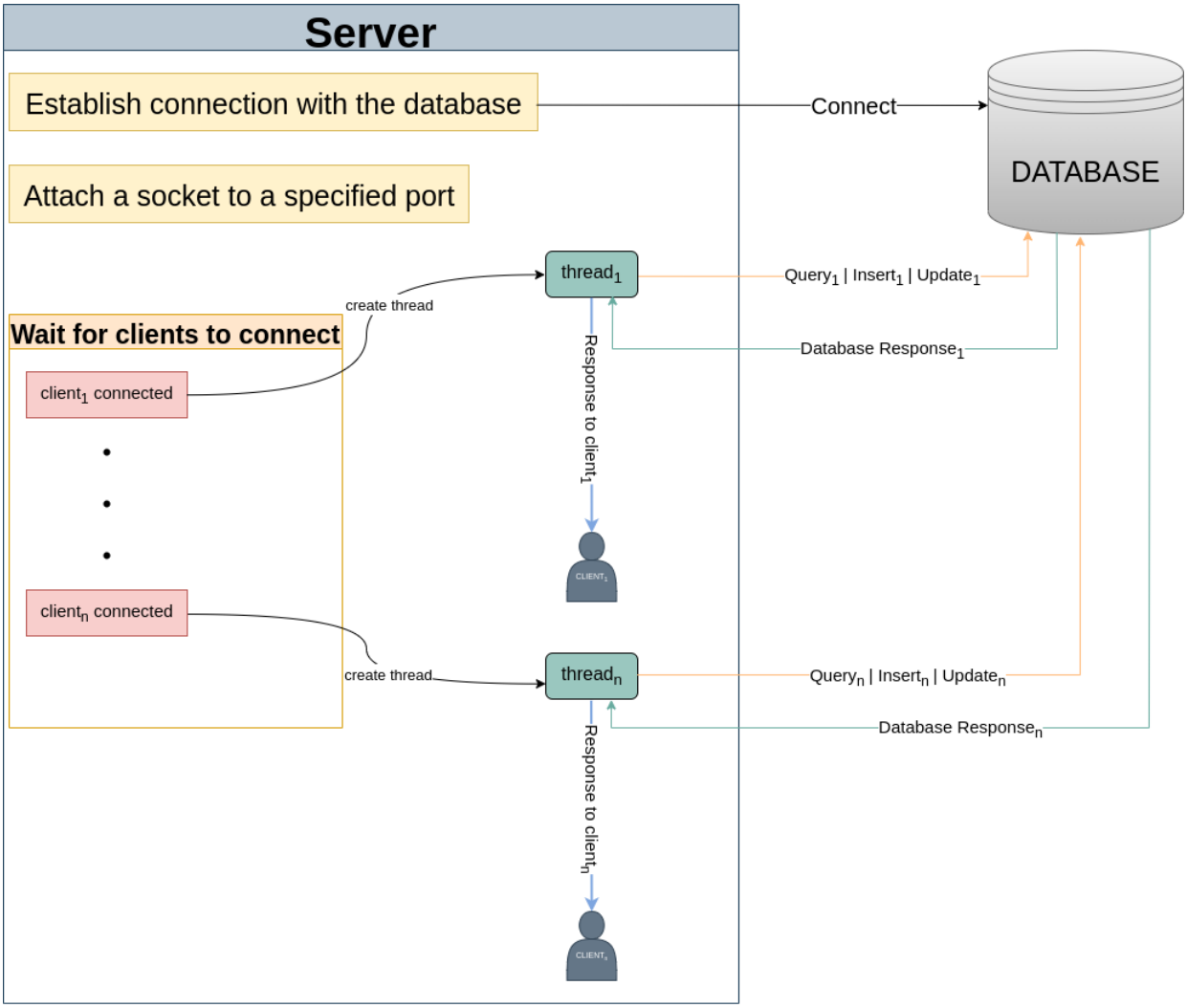

Figure 3: Diagrama - *Server*

Inițial *Serverul* se connectează la o bază de date care conține informațiile necesare pentru a răspunde cererilor Clienților. *Serverul* atașează unui socket un port specific, socket care o să fie folosit de *Client* pentru a comunica cu *Serverul*. Într-o buclă infinită Serverul asteaptă conexiuni de la *Clienti*, iar în momentul în care se conectează un Client, se creează un **thread** pentru acesta. În funcție de mesajul pe care îl primește *Serverul* de la *Client*, se apelează funcții specifice care trimit un query bazei de date si primesc răspuns (Ex: Mesajul primit de Server corespunde comenzii de login, deci se va trimite catre baza de date un query de forma "SELECT * from userTable WHERE usernames=usernameRecieved AND password=passwordRecieved"). În funcție de rezultatul interogării, se trimite răspuns către *Client*.

4 Detalii de implementare

Așa cum am preciat și mai sus, comunicarea între Clienți și Server se face prin socket-uri.

```
        /* create the socket */
if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("Socket Error.\n");
    return errno;
  }
```

AF_INET - specifică familia de protocoale cu care va comunica socket-ul, și anume IPv4;
SOCK_STREAM - tipul socket-ului va fi TCP (transmisie sigură);

4.1 Client

Clientul se conectează la *Server* prin primitiva *connect()*:

```
        /* connect to the Server */
if (connect (sd, (struct sockaddr *) &server,sizeof (struct sockaddr)) == -1)
  {
    perror ("[-]Connect() Error.\n");
    return errno;
  }
```

Bineînțeles, pentru ca această conexiune să se poată stabili, *Serverul* trebuie sa fie pornit.
User-ul trebuie să aleagă o opțiune din meniul dat, iar în funcție de opțiunea aleasă se apelează funcții specifice, care fie îi cer user-ului să mai introducă date, fie trimit *request* către *Server*.
După acest *request*, Clientul asteaptă sa primească un *response* de la *Server*, care va fi afișat pentru a fi văzut de user :

```
if ((nr_read=read (sd, buffer,sizeof(buffer))) < 0){
    perror ("[-]Read() Error, couldn't read the response from the server.\n");
    return errno;
  }

  buffer[nr_read]='\0';

printf ("[+]The response is: %s\n", buffer);
```

4.2 Server

Serverul se conecteaza la baza de date :

```
  SQL BD;

if(BD.connect_to_databse() == false){//connect the server to the databse

    printf("The server couldn't connect to the databse!\n ! Try to restart the server ! \n");
    return -1;
  }
else{
    printf("[+] The server connected to the databse successfully \n");
  }
```

Unde **SQL** este o clasă definitiă astfel :

```
1  class SQL {
2    private :
3
4    MYSQL *connection;
5    const std::string server_IP, username, password, database_name;
6
7    public :
8
9    SQL();//the constructor
10   const std::string getServer_IP();
11   const std::string getUsername();
12   const std::string getDatabaseName();
```

```
  const std::string getPassword();

  bool connect_to_databse();

  MYSQL * getConnection();//returns the connection with the database
  MYSQL_RES* execute_query (std::string cmd_query);//returns the result of the query as MYSQL_RES*
  std::string get_result_of_the_executed_query();
  //^- this functions will be called after the execute_query() one; it returns the result of the query as a string
};
```

Serverul va trata clienții concurent, folosind thread-uri :

```
    /* Accept a client (blocking state untill a client connects*/
    if ( (client = accept (sd, (struct sockaddr *) &from, (socklen_t*)&length)) < 0)
    {
        perror ("[-]Accept() Error\n");
        continue;//the client will have to try to connect again
    }

    treat_client CLIENT(client, BD.getConnection());
    CLIENT.create_thread();
```

Se poate observa faptul că instanța CLIENT, va avea in componența sa descriptorul către care trebuie să scrie *Serverul* pentru a-i răspunde acestui client și conexiunea cu baza de date. Funcția *create_thread* din cadrul clasei **treat_client** :

```
void treat_client::create_thread(){

    pthread_t t;
    pthread_create(&t,NULL,worker_thread, this);

}
```

Thread-ul creat va executa functia *worker_thread* și va avea ca parametru întreaga instață (**this**). În cadrul aceste funcții se va citi $Request - ul$ primit de la *Client* și în funcție de acesta, se vor apela unele funcții specifice. După ce se va obține $Respons - ul$, acesta va fi trimis la *Client*.

```
    if (write (ds_cl, result, sizeof(result)) <= 0){
        // where ds_cl = ((treat_client*)arg)->desciptor_client;
        perror ("[Thread]Write() Error. The response couldn't send to the client\n");
        quit=1;//if the client leaves before getting an answer,
    }
    else
        printf ("[Thread] The response was successfully sent.\n");
```

4.3 MySQL

Baza de date conține 3 tabele :

1. users; care are câmpurile :
   (a) username(VARCHAR(20));
   (b) password(VARCHAR(30)) ;
2. trains; care are campurile :
   (a) id_train(VARCHAR(5) și este cheie primară);

(b) additional_infomartion (VARCHAR(50)) unde se poate specifica daca un tren este Anulat;

3. arrivals_departures; care are câmpurile :

   (a) station_name(VARCAHR(20));
   (b) id_train(VARCHAR(5));
   (c) arrival(DATETIME);
   (d) departure(DATETIME);
   (e) delay(INT), dacă este negativ asta înseamnă ca trenul va ajunge cu *delay* minute mai devreme in stație, dacă este pozitiv, asta înseamnă ca va ajune cu *delay* minute mai târziu în stație și de asemenea, va pleca cu *delay* minute mai târziu din stație, iar daca este NULL, atunci trenul ajunge și pleacă la ora deja stabilită;

5 Bibliografie

- https://zetcode.com/db/mysqlc/
- https://dev.mysql.com/doc/c-api/8.0/en/c-api-function-descriptions.html
- https://profs.info.uaic.ro/~computernetworks/files/7rc_ProgramareaInReteaIII_Ro.pdf