

Mersul Trenurilor

Proiect realizat de

Popescu Tudor-George, grupa A1, anul 2

1. Introducere

Proiectul "**Mersul Trenurilor**" reprezinta o aplicatie client/server care transmite si actualizeaza in timp real informatiile despre mersul trenurilor. Utilizatorii vor avea acces la 2 actiuni:

1. Sa ceara informatiile despre mersul trenurilor prin urmatoarele moduri:
 - In functie de mersul trenurilor din ziua respectiva.
 - In functie de plecările din urmatoarea ora (conform cu planul, in intarziere cu x minute).
 - In functie de sosirile din urmatoarea ora (conform cu planul, in intarziere cu x minute, cu x minute mai devreme).
2. Sa trimita informatii catre server despre posibile intarzieri sau daca trenul ajunge/pleaca mai devreme.

Toata logica aplicatiei va fi realizata in partea de **server**, astfel incat **clientul** doar va cere informatii de la server despre plecari/sosiri si va putea transmite catre server informatiile despre posibile intarzieri, plecari mai devreme sau estimari ale sosirilor.

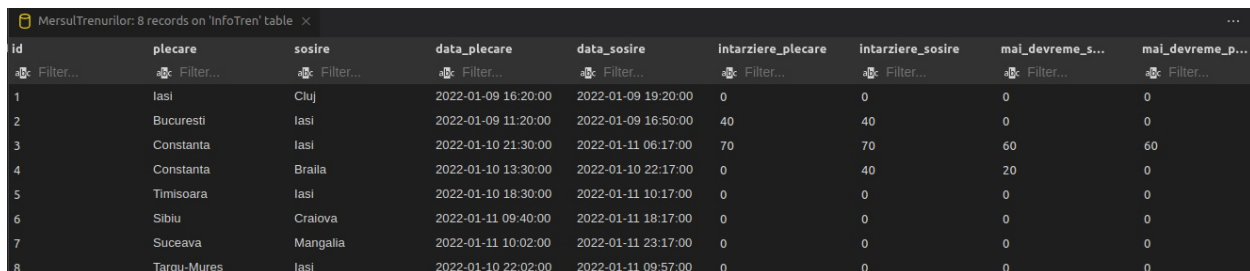
2. Tehnologii utilizate

In realizarea proiectului, am ales sa utilizez un **server TCP concurent**. TCP (Transmission Control Protocol) este un protocol de transport orientat conexiune, fara pierdere de informatii, in care acestea sunt primite in ordinea in care au fost transmise de catre server. Am ales acest protocol datorita preciziei si integralitatii datelor transportate. Intrucat este vorba despre sosirile si plecările trenurilor, informatiile trebuie sa fie ca mai exacte si sa nu se piarda, o abordare UDP nefiind convenabila in acest context.

Pentru asigurarea concurenței, am folosit **thread-uri**, intrucat reprezinta o metoda rapida si eficienta de a diviza procesele pentru clienti. Deoarece thread-urile sunt independente unul fata de celalalt, clientul poate face request-uri fara a fi afectat de actiunile altui client.

3. Arhitectura aplicatiei

Comunicarea cu baza de date se face prin intermediul librăriei **MySQL**. MySQL reprezinta un sistem de gestiune a bazelor de date relationale, bazat pe limbajul de programare SQL. Am ales sa creez o astfel de baza de date, deoarece pot gestiona si modifica foarte usor detaliile despre mersul trenurilor (cod unic, data plecare, data sosire, loc plecare, destinatie, intarzieri, etc.). Toate informatiile despre mersul trenurilor sunt stocate intr-o tabela *InfoTren*, care contine urmatoarele coloane: *id*, *plecare*, *sosire*, *data_plecure*, *data_sosire*, *intarziere_plecure*, *intarziere_sosire*, *mai_devreme_plecure*, *mai_devreme_sosire*.



id	plecare	sosire	data_plecure	data_sosire	intarziere_plecure	intarziere_sosire	mai_devreme_s...	mai_devreme_p...
1	Iasi	Cluj	2022-01-09 16:20:00	2022-01-09 19:20:00	0	0	0	0
2	Bucuresti	Iasi	2022-01-09 11:20:00	2022-01-09 16:50:00	40	40	0	0
3	Constanta	Iasi	2022-01-10 21:30:00	2022-01-11 06:17:00	70	70	60	60
4	Constanta	Braila	2022-01-10 13:30:00	2022-01-10 22:17:00	0	40	20	0
5	Timisoara	Iasi	2022-01-10 18:30:00	2022-01-11 10:17:00	0	0	0	0
6	Sibiu	Craiova	2022-01-11 09:40:00	2022-01-11 18:17:00	0	0	0	0
7	Suceava	Mangalia	2022-01-11 10:02:00	2022-01-11 23:17:00	0	0	0	0
8	Targu-Mures	Iasi	2022-01-10 22:02:00	2022-01-11 09:57:00	0	0	0	0

Tabela InfoTren din baza de date SQL

Pentru transmiterea informatii despre trenuri de la server catre client am folosit **JSON**-uri, cu ajutorul librăriei *"nlohmann/json"*, intrucat marimea datelor transmise este considerabila si trebuie organizata cat mai bine. Astfel, pentru fiecare tren, am stocat informatiile intr-un obiect de tip JSON (id, data plecare, statie plecare, etc.), in final inserand toate obiectele intr-un JSON array pe care l-am serializat si transmis catre client. In client, deserializez array-ul JSON primit de la server, il parcurg, si afisez informatiile.

Logica aplicatiei este urmatoarea:

1. Clientul face un request catre server (acesta poate sa ceara sau sa trimita informatii).

2. Server-ul proceseaza request-ul de la client si comunica cu baza de date pentru a prelua sau modifica informatiile necesare, in functie de request-ul clientului.
3. Dupa ce interogheaza baza de date, server-ul va transmite catre client informatiile cerute.
4. Daca doreste, clientul are posibilitatea de a mai face alte cereri catre server, dupa ce prima a fost procesata.

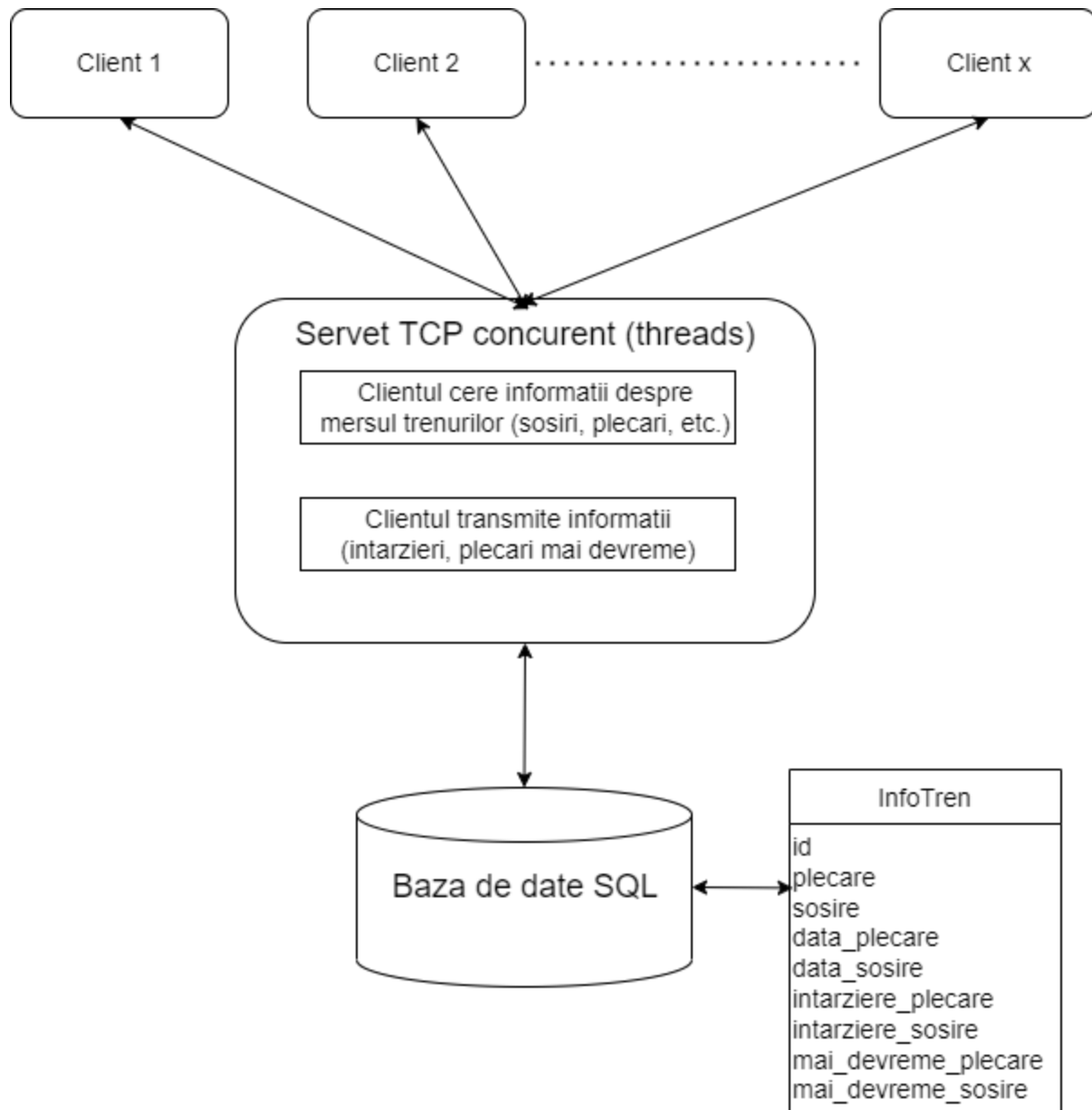


Diagrama - Arhitectura aplicatiei

4. Detalii de implementare

Pentru comunicarea dintre client si server, am folosit un **socket**, intrucat utilizarea acestuia este mai avantajoasa, dat fiind faptul ca acesta reprezinta un canal de comunicare bidirectional, pe cand pipe-urile si fifo-urile sunt unidirectionale.

Pentru a asigura concurenta aplicatiei, in implementarea serverului am folosit **thread-uri**, astfel incat, pentru fiecare client care se conecteaza, este creat un nou thread care se ocupa de cererile acestuia.

Clientul poate sa faca un numar nelimitat de cereri catre server, iar procesul client se va inchide doar atunci cand utilizatorul va folosi comanda *quit*.

In structura codului de server, am implementat functii speciale pentru interogarea bazei de date SQL, cum ar fi:

- ***mysql_connection_setup*** - creeaza conexiunea catre baza de date in care sunt stocate informatiile despre mersul trenurilor
- ***mysql_perform_query*** - creeaza interogarea necesara pentru preluarea/modificarea informatiilor din baza de date, in functie de request-ul clientului

Pentru serializarea si deserializarea JSON-urilor am folosit functiile **json.dump()** si **json::parse()** din libraria *nlohmann/json*.

```
json answer = json::array();
while ((row = mysql_fetch_row(res)) != NULL)
{
    json j;
    j["id"] = row[0];
    j["statie_plecure"] = row[1];
    j["statie_sosire"] = row[2];
    j["data_plecure"] = row[3];
    j["data_sosire"] = row[4];
    answer.push_back(j);
}
string s = answer.dump();
```

Construirea si Serializarea array-ului JSON

Totodata, in server, am creat functii care trateaza cererile diferite pe care le poate face clientul:

- **getInfoToday** - preia informatiile din baza de date despre mersul trenurilor din ziua respectiva
- **getDepartures** - ofera informatii despre plecările din urmatoarea ora (conform cu planul, in intarziere cu x minute)
- **getArrivals** - ofera informatii despre sosirile din urmatoarea ora (conform cu planul, in intarziere cu x minute, cu x minute mai devreme)
- **checkIfTrainExists** - verifica daca id-ul trenului transmis de client exista in baza de date
- **sendLateDeparture** - actualizeaza baza de date cu informatii despre intarziere plecare
- **sendLateArrival** - actualizeaza baza de date cu informatii despre intarziere sosire
- **sendEarlyDeparture** - actualizeaza baza de date cu informatii despre plecare mai devreme
- **sendEarlyArrival** - actualizeaza baza de date cu informatii despre sosire mai devreme

```
void getDepartures(MYSQL *con, struct thData tdL)
{
    MYSQL_RES *res; // the results
    MYSQL_ROW row; // the results rows (array)
    json answer = json::array();
    res = mysql_perform_query(con, "select id, plecare, sosire, data_plecare, data_sosire, intarziere_plecare, mai_devreme_plecare from InfoTren where ABS(TIMESTAMPDIFF(HOUR, TIME_TO_SEC(plecare), TIME_TO_SEC(sosire))) >= 0");
    while ((row = mysql_fetch_row(res)) != NULL)
    {
        json j;
        j["id"] = row[0];
        j["statie_plecare"] = row[1];
        j["statie_sosire"] = row[2];
        j["data_plecare"] = row[3];
        j["data_sosire"] = row[4];
        j["intarziere"] = row[5];
        j["devreme"] = row[6];
        answer.push_back(j);
    }
    string s = answer.dump();
    /* returnam mesajul clientului */
    if (write(tdL.cl, s.c_str(), 1000) <= 0)
    {
        printf("[Thread %d] ", tdL.idThread);
        perror("[Thread] Eroare la write() catre client.\n");
    }
    else
        printf("[Thread %d] Mesajul a fost transmis cu succes.\n", tdL.idThread);
    // clean up the database result
    mysql_free_result(res);
}
```

Functia getDepartures

```

MYSQL *mysql_connection_setup()
{
    struct connection_details mysql_details;
    mysql_details.server = "localhost";           // where the mysql database is
    mysql_details.user = "tudor";                 // user
    mysql_details.password = "password";          // the password for the database
    mysql_details.database = "MersulTrenurilor";  // the database
    MYSQL *connection = mysql_init(NULL);         // mysql instance

    //connect database
    if (!mysql_real_connect(connection, mysql_details.server, mysql_details.user, mysql_details.password, mysql_details.database, 0, NULL, 0))
    {
        cout << "Connection Error: " << mysql_error(connection) << endl;
        exit(1);
    }

    return connection;
}

```

Conectarea la baza de date SQL

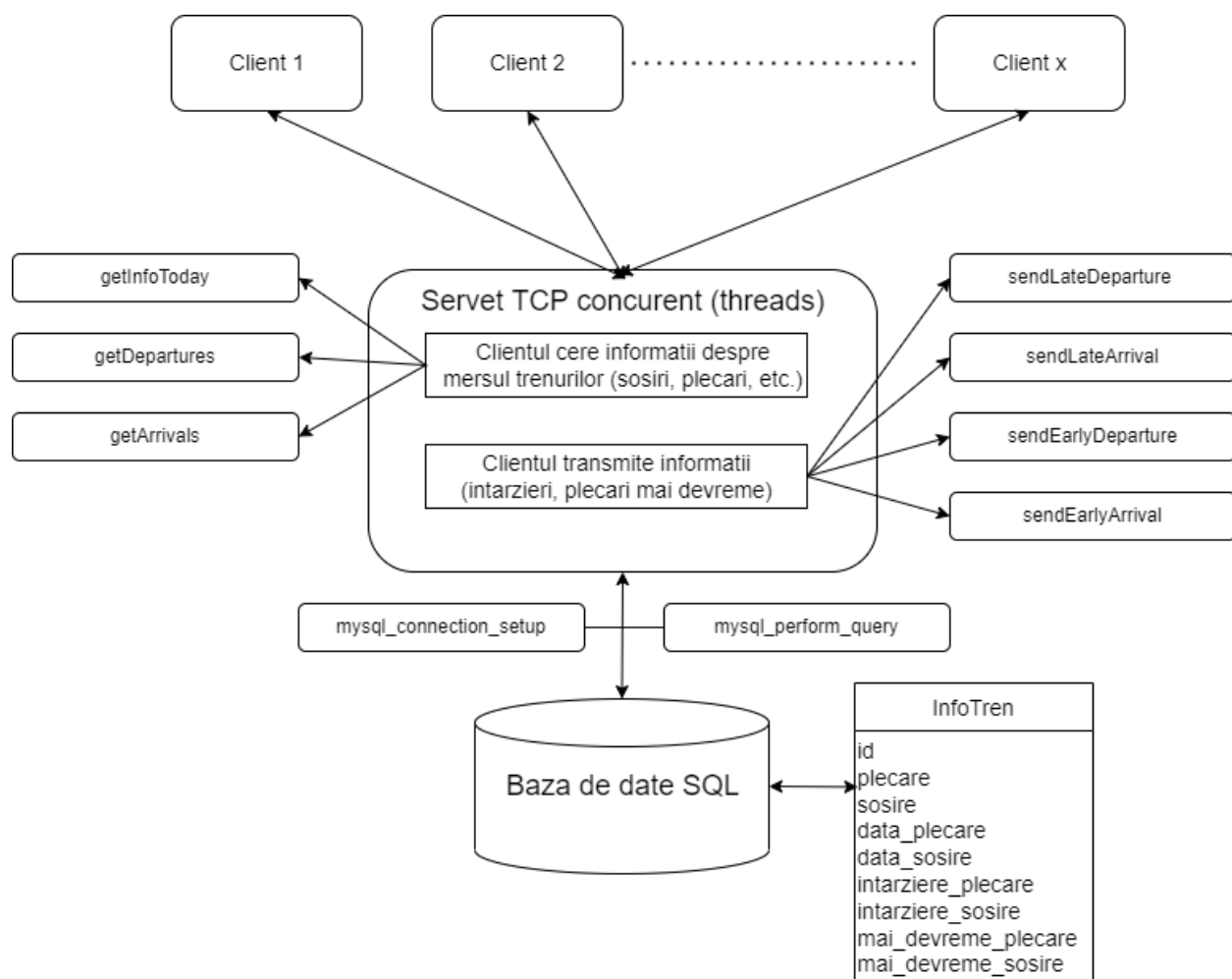


Diagrama - Arhitectura aplicatiei cu detalii de implementare

5. Concluzii si imbunatatiri

- Sa fie implementata o interfata grafica pentru comenzile pe care le poate executa clientul, pentru a face aplicatia cat mai atractiva, intuitiva si usor de utilizat.
- Sa existe roluri de user si admin:
 - Un user poate cere informatii despre mersul trenurilor si sa transmita posibile intarzieri, plecari mai devreme, estimari sosire.
 - Un admin poate adauga mersul trenurilor pentru ziua urmatoare, poate restrictiona accesul utilizatorilor care transmit informatii false, etc.
- Sa existe si alte modalitati de a cere informatii pentru un client (e.g. sa ceara informatii despre trenuri in functie de ruta, destinatie, plecare, ziua urmatoare, etc.).

6. Bibliografie

1. Model de server TCP concurent implementat cu thread-uri
<https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php>
2. Database Connection in C++ with MySQL
https://www.youtube.com/watch?v=cSZvq7Kv6_0&ab_channel=Steve'steacher
3. MySQL Database Connection + Performing Queries
<https://github.com/WeebNetsu/database-connection-cpp>
4. Librarie JSON
<https://github.com/nlohmann/json>