Mersul Trenurilor

Proiect realizat de Duluță George Denis

Grupa: B4

1.Introducere

Tema proiectului se bazează pe implementarea unui client (la care au acces mai mulți utilizatori) care va trimite comenzi către un server (signal - receive) acesta urmând să le execute și să returneze client-ului un șir de caractere bazat pe cererea facută de el (comanda trimisă).

"Mersul Trenurilor" reprezintă o aplicație de tip server/client ce oferă sau actualizează informații în timp real de la toți clienții înregistrați pentru mersul trenurilor. Utilizatorii înregistrați au acces la 2 tipuri de comenzi:

a. Comenzi prin care solicită informații despre mersul trenurilor în următoarele tipare:

- Trenurile care circulă în ziua respectivă
- Trenurile care pleacă în următoarea oră (conform cu planul, în întârziere cu x minute).
- În funcție de sosirile din următoarea oră (conform cu planul, în întârziere cu x minute, cu x minute mai devreme)

b. Comenzi prin care trimit către server informații despre posibile întârzieri sau dacă trenul ajunge/pleacă mai devreme.

2.Tehnologii utilizate

Pentru realizarea proiectului am ales să folosesc un server TCP concurent. TCP este un protocol folosit în aplicații ce au nevoie de siguranța confirmării primirii datelor, păstrarea ordinii și a integrității acestora. Alegerea acestui protocol este bazată pe fundamentul păstrării integrității și a ordinii datelor, deoarece în cadrul proiectului sunt transmise informații despre sosirile și plecările trenurilor, iar această transmitere trebuie să fie cât mai exactă și să nu avem pierderi. Dacă întreg conceptul aplicației, arhitectura, nu ar avea la bază o comunicare sigură între Server și Client, atunci informațiile ar putea fi pierdute sau alterate, fapt ce va conduce la o funcționare incorectă. În acest sens consider că un server TCP este mult mai potrivit decât unul UDP.

Pentru asigurarea concurentei, 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. Pentru informațiile despre trenuri am ales să folosesc câte un fișier xml pentru fiecare traseu al unui tren în care am folosit tag-uri precum: <Id> pentru a salva id-ul trenului, <Plecare> pentru orașul de plecare, <Sosire> pentru orașul de sosire, <Data_Plecare> pentru data de plecare a trenului, <Întârziere> pentru a informa utilizatorii că trenul x are o întârziere de y minute, etc.

Aplicația dispune și de o bază de date a utilizatorilor și un sistem de înregistrare deoarece numai utilizatorii ce au un cont pe platformă au acces la comenzile principale. În crearea acestui sistem am folosit SQLITE, un sistem de gestiune a bazelor de date relaționale. O bază de date SQLITE este mult mai ușor de utilizat și gestionat decât un fișier text, iar nivelul păstrării integrității datelor e mult mai ridicat.

3. Arhitectura aplicației

3.a Conceptele implicate

În conceperea aplicației, voi utiliza următoarele concepte: server, client, bază de date și fișiere xml. Server-ul primește cereri de conectare de la mai mulți clienți, iar acesta îi va trata pe fiecare în parte într-un mod concurent. Clientul trimite către server comenzi introduse de utilizator, iar în funcție de comanda trimisă, server-ul se folosește de baza de date pentru a vedea dacă utilizator-ul ce folosește client-ul este logat într-un cont și de fișierele xml pentru a extrage informațiile necesare răspunsului corespunzător comenzii pe care o prelucrează dacă aceasta făcea referință la mersul trenurilor.

Comunicarea cu baza de date se face prin intermediul librăriei sqlite3.h. SQLITE este un sistem de gestiune a bazelor de date relaționale bazat pe limbajul SQL. Alegerea acestei librării vine din ușurința pe care mi-o oferă în gestionarea și modificarea detaliilor utilizatorilor. Informațiile despre fiecare utilizator sunt stocate într-o tabelă internă users.db care conține următoarele coloane: ID, Nume, Parolă, Status.

	ID	Nume	Parola	Status
	Fi...	Filter	Filter	Filter
1	1	Denis	wvmrh	0
2	2	Rares	wvmrh	0
3	3	Mihai	wvm	0
4	4	Bogdan	ylt	0
5	5	Victor	erx	0

Tabela users.db din baza de date SQLITE

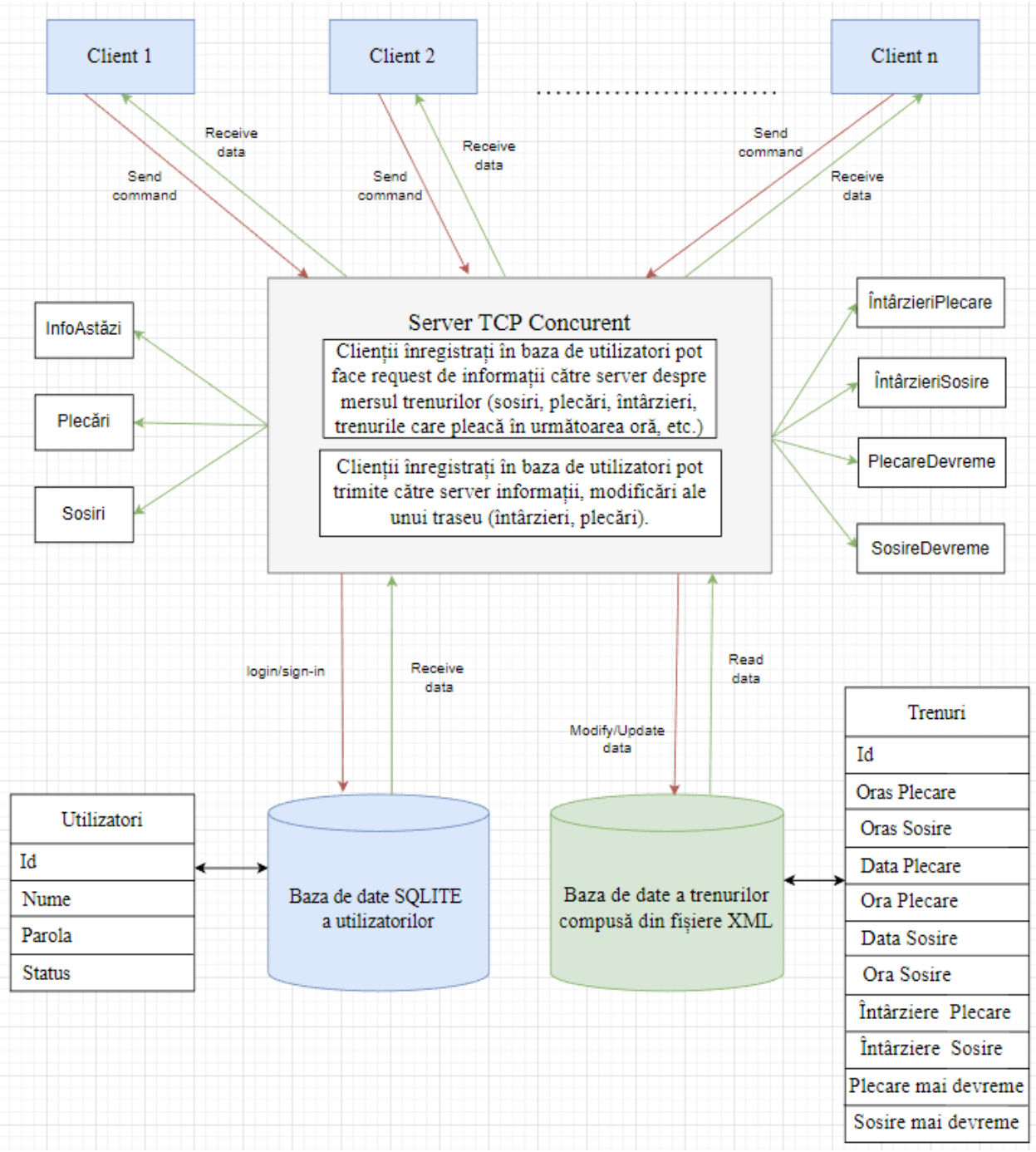
Pentru a stoca datele despre fiecare ruta a trenurilor am folosit fișiere xml cu tag-uri aferente precum <id>, <Plecare>, <Sosire>, <Data_Plecare>, <Întârziere>, etc. Pentru lucrul cu aceste fișiere am folosit librăriile “libxml/parser.h” și “limxml/tree.h”, ce îmi oferă funcționalități cu ajutorul cărora pot interpreta tag-urile, în acest scop server-ul având funcționalități precum: returnarea de informații cu privire la mersul trenurilor de astăzi, informații despre plecările din următoarea oră, posibilitatea modificării orei de plecare, actualizarea întârzierilor, etc.

```
<?xml version="1.0" encoding="UTF-8"?>
<trenuri>
  <tren numar="1">
    <oras_plecare>Iasi</oras_plecare>
    <oras_sosire>Bucuresti</oras_sosire>
    <data_plecare>06.01.2023</data_plecare>
    <ora_plecare>19:40</ora_plecare>
    <data_sosire>01.12.2022</data_sosire>
    <ora_sosire>19:50</ora_sosire>
    <intarziere_plecare>10</intarziere_plecare>
    <intarziere_sosire>20</intarziere_sosire>
    <mai_devreme>20</mai_devreme>
  </tren>
```

```
−<trenuri>
 −<tren numar="1">
    <oras_plecare>Iasi</oras_plecare>
    <oras_sosire>Bucuresti</oras_sosire>
    <data_plecare>06.01.2023</data_plecare>
    <ora_plecare>19:40</ora_plecare>
    <data_sosire>01.12.2022</data_sosire>
    <ora_sosire>19:50</ora_sosire>
    <intarziere_plecare>10</intarziere_plecare>
    <intarziere_sosire>20</intarziere_sosire>
    <mai_devreme>20</mai_devreme>
  </tren>
```

Fişier xml al unui tren ce pleacă din Iaşi spre Bucureşti

3.b Diagrama aplicației detaliată



4. Detalii de implementare

4.a Cod relevant proiectului

În comunicarea dintre server și client am folosit socket, deoarece utilizarea lui este mult mai favorabila întrucât este un canal de comunicație bidirecțional. Așa cum am spus deja, pentru a asigura concurența aplicației m-am folosit de thread-uri, iar pentru fiecare client am creat un proces copil în care rezolvăm cererile făcute de client. Fiecare client poate să facă un număr nelimitat de request-uri către server, iar client-ul se va închide corespunzător doar la tastarea comenzii quit.

Pentru a putea folosi baza de date, a prelua și modifica datele am implementat în server funcții pentru interogarea bazei de date precum:

• *setDataBase*: ce deschide baza de date aferentă și permite accesul asupra datelor;

```
void setDataBase()
{
    int rc = sqlite3_open("users.db",&db);
        if(rc != SQLITE_OK)
    {
        printf("Eroare\n");
    }
    sqlite3_exec(db,"CREATE TABLE IF NOT EXISTS Users(ID INTEGER PRIMARY KEY, Nume TEXT,Parola TEXT,Status INT);",0,0,&err);
    sqlite3_exec(db,"Update Users SET Status=0;",0,0,&err);
}
```

Funcția setDataBase

• *adaugaUser*: ce adaugă un nou user completând informațiile în câmpurile aferente: Nume, Parolă;

```
void adaugaUser(char nume[NMAX],char parola[NMAX])
{
    char sql[256];
    char *err=0;
    sql[0]='\0';
    strcpy(sql,"INSERT INTO Users(Nume,Parola) VALUES('");
    strcat(sql,nume);
    strcat(sql,"','");
    strcat(sql,parola);
    strcat(sql,"');");
    printf("%s\n",sql);
    sqlite3_exec(db,sql,0,0,&err);
    sqlite3_close_v2(db);
    ROpenBase();

}
```

Funcția adaugaUser

• *existaUser*: ce verifică în cadrul înregistrării unui user dacă nu cumva există deja un user cu același nume;

• *sign*-in: ce realizează procesul de înregistrare a unui nou user prin verificarea existenței deja a unui utilizator cu același nume;

• *logare* - prin intermediul ei se realizează verificarea username-ului introdus și dacă parolă corespunde acelui username în baza de date;

• *userLogatClientDiferit* - verifică dacă nu cumva username-ul cu care încearcă clientul să se logheze nu este deja logat într-un alt client.

```
int userLogatClientDiferit(char nume[NMAX])
{
    sqlite3_stmt* stmt;
    char sql[NMAX];
    strcpy(sql,"SELECT Status FROM Users WHERE Nume='");
    strcat(sql,nume);
    strcat(sql,"';");
    sqlite3_prepare_v2(db,sql,-1,&stmt,0);
    sqlite3_step(stmt);
    int id;
    id=sqlite3_column_int(stmt,0);
    if(id != 0)
        {
            sqlite3_finalize(stmt);
            return 1;
        }
    else
        {
            sqlite3_finalize(stmt);
            return 0;
        }
}
```

Funcția userLogatClientDiferit

Pentru a citi si modifica fișierele xml am folosit funcții ce au la bază utilizarea librăriilor “libxml/parser.h” și “limxml/tree.h”:

• *parse_xml: parsează fișierul xml și salvează într-o structură definită datele;*

• *modify_xml: modifică în fișierul xml date referitoare la trenuri;*

• *extract_field: extrage dintr-un fișier valoarea unui anumit câmp (Oraș Plecare, Oraș Sosire etc).*

Pentru a răspunde cererilor făcute de client am făcut în server funcții speciale în acest sens:

• *getInfoAstazi:* ce deschide fișierele xml și se folosește de funcția getDate pentru a vedea ce trenuri au data de plecare ca fiind ziua de astăzi urmând ca apoi să trimită informațiile despre acele trenuri către client;

```
void getInfoAstazi(char mesaj[MMAX])
{
    int i, g = 0;
    char date[NMAX];
    char msg[MMAX];
    bzero(mesaj, MMAX);
    bzero(msg, MMAX);
    getDate(date);
    sprintf(mesaj, "\e[1;32m===============Mersul trenurilor de astazi %s===============\e[0m \n", date);
    // strcpy(mesaj, "id|Plecare|Sosire|Data_plecare|Ora_Plecare|Data_Sosire|Ora_sosire|Intarziere_Plecare|Intarziere_Sosire \n");
    for (i = 0; i < numar_trenuri; i++)
    {
        if (strcmp(trenuri[i].data_plecare, date) == 0)
        {
            g = 1;
            /*sprintf(msg, "%s | %s | %s | %s | %s | %s %s | %s | %s\n",
                    trenuri[i].numar, trenuri[i].oras_plecare, trenuri[i].oras_sosire, trenuri[i].data_plecare, trenuri[i].ora_plecare,
                    trenuri[i].data_sosire, trenuri[i].ora_sosire, trenuri[i].intarziere_plecare, trenuri[i].intarziere_sosire);*/
            sprintf(msg, "Id tren : %s\nOras de plecare: %s\nOras de sosire: %s\nData plecarii : %s\nOra plecarii : %s\nData sosirii : %s\nOra sosirii : %s\nIntarziere la plecare de : %s minute\nI
                    trenuri[i].numar, trenuri[i].oras_plecare, trenuri[i].oras_sosire, trenuri[i].data_plecare, trenuri[i].ora_plecare,
                    trenuri[i].data_sosire, trenuri[i].ora_sosire, trenuri[i].intarziere_plecare, trenuri[i].intarziere_sosire, trenuri[i].mai_devreme);
            strcat(mesaj, msg);
            strcat(mesaj, "\e[1;32m================================================================\e[0m \n");
        }
    }
    if (g == 0)
    {
        sprintf(msg, "\e[1;31m[[server]Nu este niciun tren care sa plece astazi, %s\e[0m", date);
        strcpy(mesaj, msg);
    }
}
```

Funcția InfoAstăzi

• *getPlecari*: prezintă informații despre plecările din următoarea oră (conform cu planul, în întârziere cu x minute);

• *getSosiri*: prezintă informații despre sosirile din următoarea oră (conform cu planul, în întârziere cu x minute, cu x minute mai devreme;

• *setIntarziere*Plecare: actualizează fișierul corespunzător xml cu informații despre întârzieri la plecare;

• *setIntarziereSosire*: actualizează fișierul corespunzător xml cu informații despre întârzieri la sosire;

• *setPlecareDevreme*: actualizează fișierul corespunzător xml cu informații despre plecări mai devreme;

• *setSosireDevreme*: actualizează fișierul corespunzător xml cu informații despre sosiri mai devreme;

• *getOrasPlecare: prezintă trenurile care pleacă din orașul indicat de utilizator;*

• *getOrasSosire: prezintă trenurile care sosesc în orașul indicat de utilizator;*

• *getStatie: prezintă stațiile din care pleacă trenuri sau în care sosesc;*

• ruteTrenuri: prezintă trenurile care pleacă și sosesc în stațiile indicate de utilizator;

• *getTrenById: verifică daca trenul indicat prin id-ul dat de utilizator există;*

• *getDate*: returnează data curentă în format dd.ll.yyyy;

```c
void getDate(char* date)
{
    struct timeval tv;
    time_t t;
    struct tm*info;
    char buffer[64];
    gettimeofday(&tv,NULL);
    t = tv.tv_sec;
    info = localtime(&t);
    strftime(buffer,sizeof buffer,"%d.%m.%Y",info);
    strcpy(date,buffer);
}
```

Funcția getDate

• *getTime*: returnează ora și minutele curente;

• *extrageTag*: returnează valoare unui anumit tag din fișierul xml (<Plecare>, <Sosire>, <Data_Plecare>, etc.).

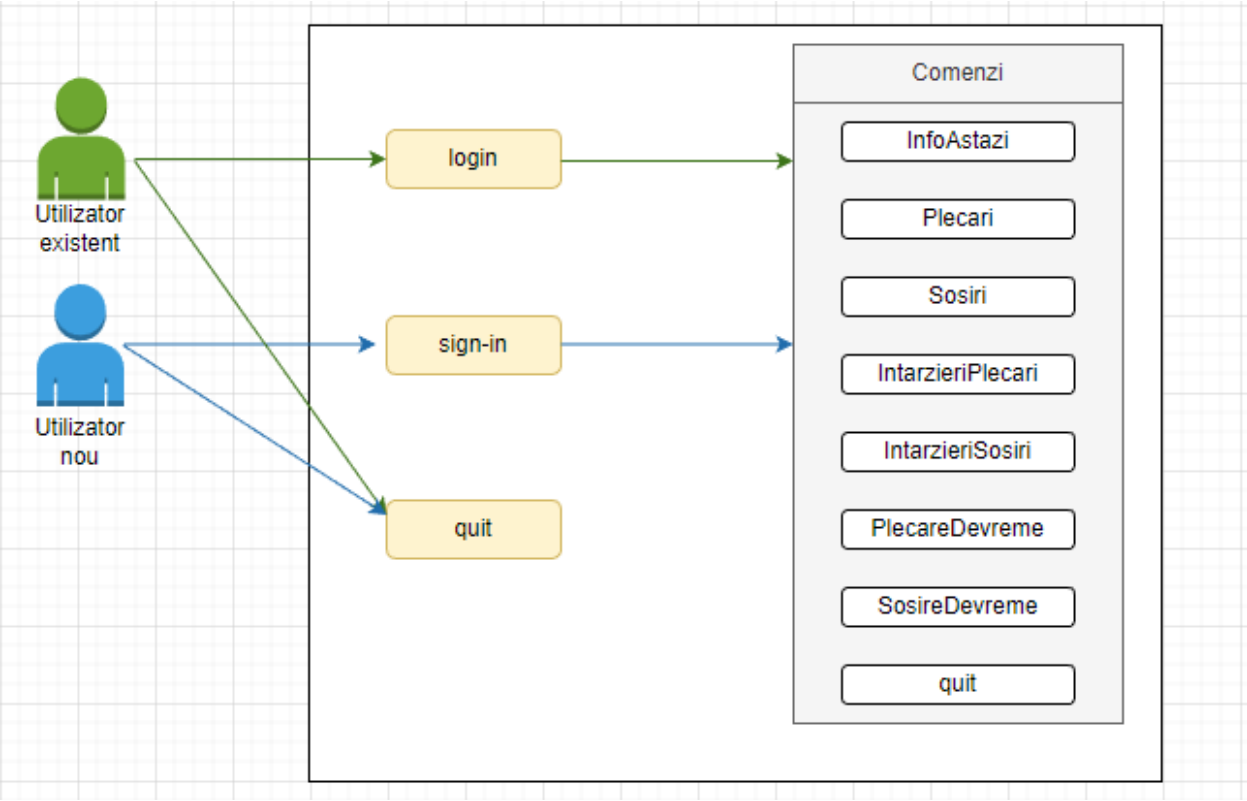
4.b Scenarii de implementare

Inițial fiecare utilizator va trebui fie să se logheze prin intermediul comenzii login într-un cont existent în baza de date, fie să se înregistreze cu ajutorul comenzii sign-in pentru a putea avea acces la comenzile referitoare la mersul trenurilor aferente aplicației, odată logați statusul lor va fi activ în baza de date. Comenzile pentru userii nelogați sunt login, sign-in și quit. După logarea sau înregistrarea acestuia în sistem, utilizatorul va avea acces la o suită de comenzi ce implică mersul trenurilor.

```
a.Comenzi pentru useri nelogati
1.login <username> <parola>
2.sign-in <user> <parola>
3.quit
b.Comenzi pentru useri logati (acestea pot fi apelate si cu <id.>):
1.getInfoAstazi - Trenurile care pleaca astazi
2.getPlecari - Trenurile care pleaca in ora urmatoare
3.getSosiri - Trenurile care sosesc in ora urmatoare
4.setIntarziere Plecare <id_tren> <minute>
5.setIntarziereSosire <id_tren> <minute>
6.setSosireDevreme <id_tren> <minute>
7.getOrasPlecare <oras_plecare>
8.getOrasSosire <oras_sosire>
9.getStatie 'plecare' | 'sosire'
10.ruteTrenuri <statie_plecare> | <statie_sosire>
11.getTrenById <id_tren>
12.logout
13.quit
```

Lista de comenzi accesibile de către utilizatori

Cu ajutorul comenzii getInfoAstazi un utilizator logat va primi informații despre mersul trenurilor din ziua curentă. Mai departe poate trimite comenzi din lista prezentată mai sus precum: Plecări pentru a primi informații despre plecările din următoarea oră, Sosiri pentru a primi informații despre sosirile din următoarea oră, IntarzieriSosiri pentru a trimite server-ului informații despre o posibilă întârziere a trenului x la sosire urmând ca server-ul să actualizeze fișierul corespunzător xml cu informații despre întârzieri la sosire, etc. Prin accesarea comenzii logout utilizator-ul se deconectează din contul pe care este logat curent, iar statusul acestuia în baza de date devine inactiv, acesta are acces în continuare la comenzile pentru useri nelogați. Prin accesarea comenzii quit, conexiunea client-ului cu server-ul se închide, iar daca utilizatorul era logat într-un cont acesta va fi delogat.



Schema diagramei use case

5. Concluzii și îmbunătățiri

Ca și îmbunătățiri m-am gândit la implementarea a două tipuri de utilizatori cu roluri diferite: **admin** și **user**. User-ul poate cere informații despre mersul trenurilor și să trimită posibile întârzieri, plecări, sosiri, iar admin-ul poate adăuga mersul trenurilor pentru zilele următoare, poate verifica și aproba/refuza informațiile trimise de user și totodată să-i trimită atenționări cu privire la răspândirea de informații false, poate chiar restricționa accesul acestuia la aplicație.

Implementarea unei interfețe grafice pentru cererile client-ului pentru a facilita atractivitatea și utilizarea aplicației.

6. Bibliografie

- https://profs.info.uaic.ro/~gcalancea/lab7/servTcpConc.c
- https://profs.info.uaic.ro/~gcalancea/lab7/cliTcpConc.c
- https://zetcode.com/db/sqlitec/
- https://www.sqlite.org/cintro.html
- https://www.w3schools.com/xml/xml_whatis.asp
- https://www.geeksforgeeks.org/differences-between-tcp-and-udp/
- https://unixism.net/2019/04/linux-applications-performance-part-ii-forking-servers/