

Mersul Trenurilor

Propunere Continental

Acasandrei Nicu-Alexandru

Universitatea "Alexandru Ioan Cuza" din Iasi, Facultatea de Informatica,
Specializarea Informatica - Limba Engleza

Abstract. Acest document prezintă o viziune generală asupra proiectului numit "Mersul Trenurilor" ce este o propunere de catre Continental și detaliază structura, tehnologiile și implementarea.

1 Introducere

"Mersul Trenurilor" este o aplicatie de tip Server/Client ce are ca scop reproducerea unei aplicatii de comunicare a informatiilor despre situatia feroviara in timp real catre clientii ce o utilizeaza. Utilizatorii vor fi capabili sa ceara informatii despre status plecari, status sosiri, intarzieri si estimare sosire. Serverul va fi hostat pe un computer ce utilizeaza Linux, si va folosi portul 2507, desi acest detaliu poate fi schimbat.

1.1 Cuvinte Cheie:

1. Command design pattern
2. Command queue
3. Threads
4. Sockets
5. XML
6. JSON
7. CMAKE

2 Tehnologii Aplicate

În cadrul proiectului, au fost utilizate următoarele tehnologii:

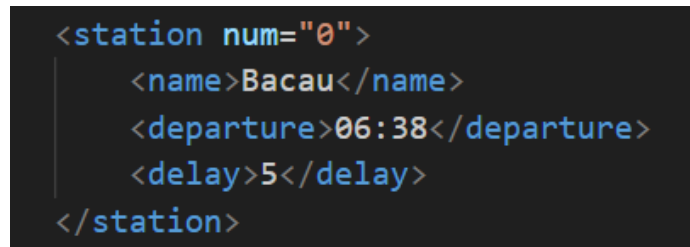
- **2.1 C++ / C:** Aplicatia este realizata atat in C++ cat si in C, combinand cele 2 limbaje de programare deoarece C este standardul pentru programare in cadrul terminalului Linux, in timp ce C++ ofera acces la o functionabilitate mai mare precum clase, templates, etc.

- **2.2 TCP/IP:** Aplicația folosește protocoale de tip TCP/IP pentru comunicarea dintre client și server. Alegerea acestui protocol s-a bazat pe cerințele aplicației legate de transmisiei datelor și evitarea pierderii informațiilor importante. Deși UDP (User Datagram Protocol) este un protocol mai rapid și mai eficient în termeni de resurse, acesta nu oferă garantarea livrării pachetelor, ceea ce îl face mai puțin potrivit pentru aplicații care necesită consistență și integritate a datelor.
- **2.3 XML:** Pentru baza de date de tip XML voi folosi biblioteca Libxml2. Aceasta la baza este scrisa in C, dar poate fi folosita si cu C++. Aceasta librarie ne ajuta sa interpretam baza de date prin intermediul unor pointers si a parcurgerii arborescente al XML-ului.
- **2.4 JSON:** Pentru baza de date ce stocheaza utilizatorii am folosit un fisier de tip JSON. Pentru acesta folosesc biblioteca "nlohmann/json" realizata de Niels Lohmann. Aceasta este special realizata pentru C++, iar modul de concepere al acesteia este astfel incat a poata parcurge vectorial informatia din fisier asemanator cu STL vector.
- **2.5 CMake:** Un CMake file este un fișier folosit de CMake, un instrument open-source pentru automatizarea procesului de construire a proiectelor software. Scopul său principal este de a genera fișiere de build specifice platformei și compilatorului, cum ar fi Makefile pentru sistemele Unix/Linux sau fișiere de soluții pentru Visual Studio pe Windows. Este foarte popular în proiectele C și C++ datorită portabilității și flexibilității sale.

3 Structura Aplicației

Conceptul general al aplicației este modelat conform următoarei structuri:

În structura se poate observa faptul ca avem o baza de date de tip XML, in aceasta sunt salvate informatiile despre trenuri, in formatul normal XML ce poate fi reprezentat ca un arbore, unde fiecarui noi ii poate fi asociat un atribut. O a doua baza de date de tip JSON este implementata in cadrul acestui proiect



```
<station num="0">
  <name>Bacau</name>
  <departure>06:38</departure>
  <delay>5</delay>
</station>
```

Fig. 1: Nod statation, cu atribut "num" si nodurile copil ale acestuia

cu scopul de a salva utilizatorii ce au campurile de nume si parola salvate in acest fisire, FARA o anumita metoda de criptare. Serverul accesează baza de date

```

"users": [
  {
    "name": "Nicu",
    "pass": "0307"
  },

```

Fig. 2: Fisier JSON, campurile "name" si "pass"

folosind thread-uri, fiecare thread fiind responsabil de gestionarea cererilor unui client specific. Acest model permite procesarea simultană a mai multor cereri din partea clienților, asigurând o creștere a eficienței și reducând timpul de așteptare pentru fiecare utilizator. Prin utilizarea thread-urilor, serverul poate gestiona în paralel conexiunile multiple, fiecare thread operând independent pentru a răspunde cererilor clientului pe care îl deservește.

Alte detalii importante despre structura:

- Server.CPP - Codul serverului;
- Client.CPP - Codul clientului;
- CMakeLists.txt - Instrument nativ CLion, folosit pentru usurarea compilarii proiectului;
- Clase - Un folder special in cadrul proiectului in care pot fi gasite clasele si headerele folosite in acest proiect;

Un ultim detaliu important in structura acestui proiect este: In acest director

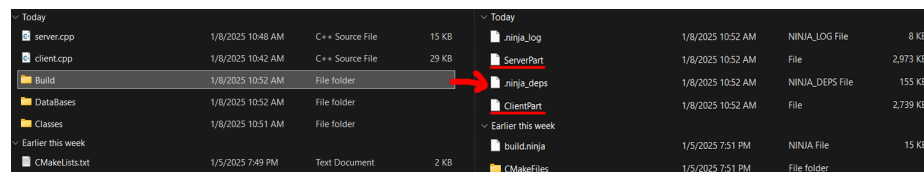


Fig. 3: Directorul build, creat prin intermediul Cmake

pot fi gasite "ServerPart" si "ClientPart", 2 executabile create de functia "build" din cadrul CLion. Functia build va compila codul si crea executabile intr-un path specific. Utilizarea functiei build este simplificarea compilarii, ce in caz normal ar trebui realizata prin comenzi precum:

- gcc -o ServerPart server.c helpers.c xmlworkaround.c reglog.c -lxml2 -pthread 'xml2-config -cflags -libs'

Aceasta comanda creand doar executabilul pentru server, fara fisier JSON asociat.

4 Aspecte de Implementare

Aceasta sectiune detaliaza codul prezentand diferite structuri de date utilizate si detalii de implementare.

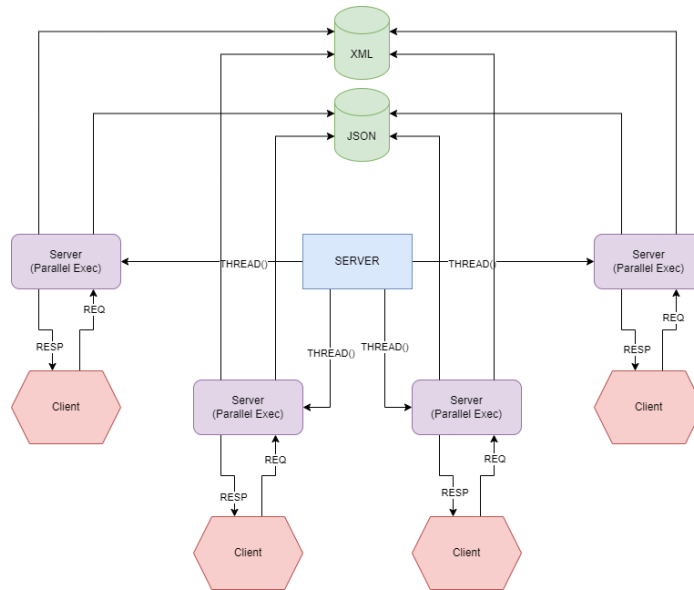


Fig. 4: Diagrama proiectului

4.1 Cod Exemplu

Figurile 5 la 16. Mai multe detalii adaugat in urmatoarea sectiune.

4.2 Detalii Implementare

Orice client are acces la urmatoarele comenzi:

1. Routes: Comanda utilizata pentru afisarea tuturor rutelor valabile in baza de date.
2. Login: Comanda utilizata pentru a te loga. Starea clientului va fi updatata la LOGGED IN daca logare este realizata cu succes.
3. Register: Comanda utilizata pentru a te inregistra. Un client se poate inregistra si apoi loga.

```

string Helpers::Menu(bool loggedIn) {
    string message;
    if (loggedIn == false) {
        cout << "Enter a command: "
        <<endl << "1. routes    -View trains."
        <<endl << "2. login     -Login user."
        <<endl << "3. register  -Register user."
        <<endl << "4. quit      -Exit application."
        <<endl; // Commands that can be used by a user that is not logged in.
        getline (cin, message); // Reading the input from the user;
    } else {
        cout << "Enter a command: "
        <<endl << "1. routes    -View trains."
        <<endl << "2. search     -Search trains."
        <<endl << "3. delay      -Signal delay of a train."
        <<endl << "4. early      -Signal early coming of a train."
        <<endl << "5. logout     -Logout user."
        <<endl << "6. quit      -Exit application."
        <<endl; // Commands that can be used by a user that is logged in.
        getline (cin, message); // Reading the input from the user;
    }

    return message;
}

```

Fig. 5: Metoda "Menu" din clasa "Helpers"

```

#define RESET "\033[0m" // Reset text color
#define RED "\033[31m" // Red text
#define GREEN "\033[32m" // Green text
#define PURPLE "\033[95m" // Purple text
#define ORANGE "\033[38;5;208m" // Orange text

```

Fig. 6: Definirea culorilor ptr textul din terminal pe baza codului ANSI

```

nlohmann::json LoadJSON();
void SaveJSON(nlohmann::json& userData);

```

Fig. 7: Enter Caption

```

nlohmann::json JSONMethods::LoadJSON() {
    //Deschidem fisierul JSON
    ifstream Jason(JSONpath); // Deschidere ptr citire;
    if (!Jason.is_open()) {
        clrMSG = "[Class - ERROR] Failed to open " + JSONpath + " for reading!";
        cmd.CheckOutput(clrMSG, true);
        return nlohmann::json(); // Returnam un "empty JSON object"
    }

    //Salvam toata informatia intr-un string
    string ContentJSON((istreambuf_iterator<char>(Jason)), (istreambuf_iterator<char>()));

    //Verificam informatia
    nlohmann::json jsonINFO;
    if (ContentJSON.empty()) {
        clrMSG = "[Class - ERROR] Couldn't save the JSON information in the string!";
        cmd.CheckOutput(clrMSG, true);
        return nlohmann::json();
    }

    //Folosim try & catch
    try {
        jsonINFO = nlohmann::json::parse(ContentJSON, nullptr, false);
        if (jsonINFO.is_discarded()) {
            clrMSG = "[Class - ERROR] Couldn't parse the JSON information!";
            cmd.CheckOutput(clrMSG, true);
            return nlohmann::json();
        }
    } catch (...) { // In cazul in care avem o eroare cu fisierul JSON, dar nu este legata de citirea acestuia;
        clrMSG = "[Class - ERROR] Unknown error! Anything else than reading JSON!";
        cmd.CheckOutput(clrMSG, true);
        return nlohmann::json();
    }

    cout << "[Class] Successfully loaded the JSON information!" << endl;
    cout << jsonINFO.dump(0) << endl; // pretty print output
    return jsonINFO;
}

```

Fig. 8: Metoda LoadJSON

```

void JSONMethods::SaveJSON(nlohmann::json& userData) {
    try {
        //Acum vom deschide fisierul pentru scriere:
        ofstream Jason(JSONpath);
        if (!Jason.is_open()) {
            clrMSG = "[Class - ERROR] Couldn't open " + JSONpath + " for writing!";
            cmd.CheckOutput(clrMSG, true);
            return;
        }
        Jason << userData.dump(4);
        Jason.close();

        cout << "[Class] Successfully updated the JSON file!" << endl;
    } catch (const exception &e) {
        clrMSG = "[Class - ERROR] Couldn't update JSON " + string(e.what()) + "!";
        cmd.CheckOutput(clrMSG, true);
    }
}

```

Fig. 9: Metoda SaveJSON()

```

bool login(string name, string pass); // Logare (Setam
bool regUser(string name, string pass); // Inregistrare
void logout(); // Delogare (Setam logged = false)
bool logCheck(); // Verificam status logged;
bool checkJSON(string name, string pass); //Verificam
bool checkUSER(string name, nlohmann::json& userData);

```

Fig. 10: Metodele clasei RegLog

```

bool RegLog::checkJSON(string name, string pass) {
    JSONMethods Jason("/home/alex/RC/TrainApp/DataBases/Users.JSON");
    nlohmann::json userData = Jason.LoadJSON();

    if (userData.contains("users") || userData["users"].is_array()) {
        clrMSG = "[Class - ERROR] JSON file doesn't have a valid 'users' array!";
        cmd.CheckOutput(clrMSG, true);
        return false;
    }

    for (auto user : userData["users"]) {
        if (user.contains("name") && user["name"] == name) {
            if (user.contains("pass") && user["pass"] == pass) {
                return true;
            }
        }
    }

    clrMSG = "[Class - ERROR] Register failed!\nPossible reasons:\nExisting username;\nCorrupted JSON";
    cmd.CheckOutput(clrMSG, true);
    return false;
}

```

Fig. 11: CheckJSON, metoda menita verificarii existentei user

```

if (existingUser == false) {
    nlohmann::json newUser = {
        {"name", name},
        {"pass", pass}
    };
    userData["users"].push_back(newUser);
    cout << "[Class] New user added: " << newUser.dump(4) << endl;

    Jason.SaveJSON(userData);
}

```

Fig. 12: Pushback() in fisirul JSON, apoi salvare, ptr register

```

enum StateListForClient { // Nu merge login cum trebuie, incerc switch case;
    LOGGED_OUT, // Default: routes, login, register, quit;
    LOGGED_IN, // Logat: routes, delay, early, logout, quit;
    LOGIN_USERNAME,
    LOGIN_PASSWORD,
    REGISTER_USERNAME,
    REGISTER_PASSWORD,
    DELAY_ID,
    DELAY_STATION,
    DELAY_DELAY,
    EARLY_ID,
    EARLY_STATION,
    EARLY_EARLY,
    SEARCHING,
    DEPARTURE_SEARCH_STATION,
    DEPARTURE_SEARCH_HOUR,
    ARRIVAL_SEARCH_STATION,
    ARRIVAL_SEARCH_HOUR,
    QUIT // Inchidere WHILE(currentState != QUIT);
};

```

Fig. 13: Starile ce pot fi accesate de un client

```

public:
    XMLWorkAround(); // Def constructor;
    XMLWorkAround(string XMLfile); // Constructor
    ~XMLWorkAround(); // Destructor

    //Fct ptr extragere informatii necesare:
    string getID(xmlNode* train);
    int getStationNum(xmlNode* station);

    // Functiile Necesare:
    bool LoadXML(); // Functie ptr a deschide fisierul
    void SaveXML(); // Functie ptr a updatea fisierul
    string PrintRoutes(); // Functie afisare rute si trenuri
    string PrintDeparture(string StationName,string hour);
    string PrintArrival(string StationName,string hour);
    void AddDelay(string trainID, int stationNum, int delay); // Functie semnalare intarziere
    void AddEarly(string trainID, int stationNum, int early); // Functie semnalare ajuns devreme
}

```

Fig. 14: Metodele clasei XMLWorkAround

```

bool XMLWorkAround::LoadXML() {
    // Primul pas e sa ne asiguram ca doc este liber;
    if (doc != nullptr) {
        xmlFreeDoc(doc);
        doc = nullptr;
    }

    // Pasul 2 este sa dam load fisierului XML in doc;
    doc = xmlReadFile(XMLfile.c_str(), NULL, 0);
    if (doc == nullptr) {
        clrMSG = "[Class - ERROR] XML could not parse the file: " + XMLfile;
        cmdHelper.CheckOutput(clrMSG, true);
        return false;
    }

    clrMSG = "[Class] XML document loaded successfully!";
    cmdHelper.CheckOutput(clrMSG, false);
    cout << "[Class] XML document loaded successfully." << endl;
    return true;
}

```

Fig. 15: Metoda LoadXML()

```

void XMLWorkAround::SaveXML() {
    if (doc == nullptr) {
        clrMSG = "[Class - ERROR] No XML file to be saved!";
        cmdHelper.CheckOutput(clrMSG, true);
        return;
    }

    // Updatea a documentului:
    int bytes = xmlSaveFileEnc(XMLfile.c_str(), doc, "UTF-8");
    if (bytes < 0) {
        clrMSG = "[Class - ERROR] XML could not be saved.";
        cmdHelper.CheckOutput(clrMSG, true);
        return;
    }

    clrMSG = "[Class] XML document saved successfully!";
    cmdHelper.CheckOutput(clrMSG, false);
}

```

Fig. 16: Metoda SaveXML()

4. Quit: Comanda pentru inchiderea clientului.

Clientii logati vor avea acces la:

1. Routes: Clientii vor primi o lista a trenurilor pentru ziua respectiva ce va include si toate informatiile despre fiecare tren.
2. PrintDeparture: Clientii vor putea cere o statie si o ora de plecare si vor primi o lista a tuturor trenurilor ce pleaca din acea statie in decursul acelei ore.
3. PrintArrival: Clientii vor putea cere o statie si o ora de sosire si vor primi o lista a tuturor trenurilor ce sosesc in acea statie in decursul acelei ore.
4. Add Early: Clientii vor semnala o sosire mai rapida a unui tren pe baza ID-ului de tren, si a statiei la care are loc sosirea. Aceasta sosire va influenta PrintArrival.
5. Add Delay: Clientii vor semnala o intarziere a unui tren pe baza ID-ului de tren, si a statiei la care are loc intarzierea. Aceasta intarziere va influenta PrintDeparture.
6. Logout: Aceasta comanda va intoarce clientul in state-ul LOGGED OUT, pierzand acces la comenzile ptr users logati.
7. Quit: Comanda pentru inchiderea clientului.

La momentul actual codul este o rescriere a celui oferit in cadrul cursului numarul 7, peste care au fost adaugate modificari majore. La baza tot conceptul de concurenta si conectare al utilizatorilor este realizat pe baza codului oferit la curs, dar pentru a realiza un sistem de logare reusit, am creat o variabila booleana. Cand aceasta este falsa, clientul are acces la un nr limitat de stari, iar cand logarea este realizata, starea se schimba pe `if(var == true)`. Starile sunt un enum ce imi permite sa fac salturi cu switch case astfel incat sa nu permit unui user logat sa se logheze din nou, sau sa mentin o ordine a logisticii de layering a codului. Cea mai importanta schimbare, nu este sectionarea clientului in doua, sau crearea de switch case pe server. Aceasta este utilizarea de clase pentru indeplinirea comenzilor intr-un mod cat mai elegant. Clasele sunt: Helpers, JSONMethods, XMLWorkAround, RegLog.

5 Concluzii

In concluzie, acest proiect tinde spre a replica o aplicatia ce urmareste mersul trenurilor precum cea de la CFR, utilizatorii avand abilitatea de a se conecta la un cont, sau a se inregistra in cazul in care nu au cont. In urma conectarii acestia au acces sa ceara informatii despre situatia trenurilor putand sa observe starea in care se afla un anumit tren. Metode de imbunatatire ale acestui proiect pot fi urmatoarele implementari:

1. Implementarea unui sistem cu numar de bilete per tren, iar clientii sa poata cumpara bilete cu un numar limitat.
2. Crearea unui rol de conductor ce sa poata modifica informatii despre tren, iar administratorul devenind doar un rol de management ce se ocupa de listele de utilizatori.

Referințe Bibliografice

References

1. Springer, LNCS Guidelines: <https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines>.
2. Clion from JetBrains: <https://www.jetbrains.com/community/education/#students>.
3. Curs Computer Networks: <https://edu.info.uaic.ro/computer-networks/index.php>.
4. Resurse Bibliografice Computer Networks: <https://profs.info.uaic.ro/sabin.buraga/teach/courses/net/net-biblio.html#web>.
5. Diagrama realizata la: <https://www.diagrams.net>.
6. Stack Overflow: <https://stackoverflow.com/>.
7. GeeksForGeeks: <https://www.geeksforgeeks.org/>.
8. CodeVault: <https://www.youtube.com/@CodeVault>.
9. CodeVault - Threads: <https://www.youtube.com/playlist?list=PLfqABt5AS4FmuQf70psXrsMLEDQXNkLq2>.
10. W3Schools XML: <https://www.w3schools.com/xml/>.
11. Libxml2: <https://gitlab.gnome.org/GNOME/libxml2/-/wikis/home>.
12. Think And Learn - Socket Tutorial: https://youtube.com/playlist?list=PLPyar5G9aNDvs6TtdpLcV043_jvxp4emI&si=K8cKZqpV0Q6rqkiu.
13. Multithreading in C: <https://www.geeksforgeeks.org/multithreading-in-c/>.
14. nLohmann Github: <https://github.com/nlohmann/json>.
15. Niels Lohmann: <https://nlohmann.me/>
16. CMake: <https://cmake.org/>
17. CMake tutorial for CLion: <https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html>
18. ANSI color code: https://en.wikipedia.org/wiki/ANSI_escape_code