

# MyFileTransferProtocol

Birulung Ioana Bianca

Universitatea Alexandru Ioan Cuza, Facultatea de Informatică

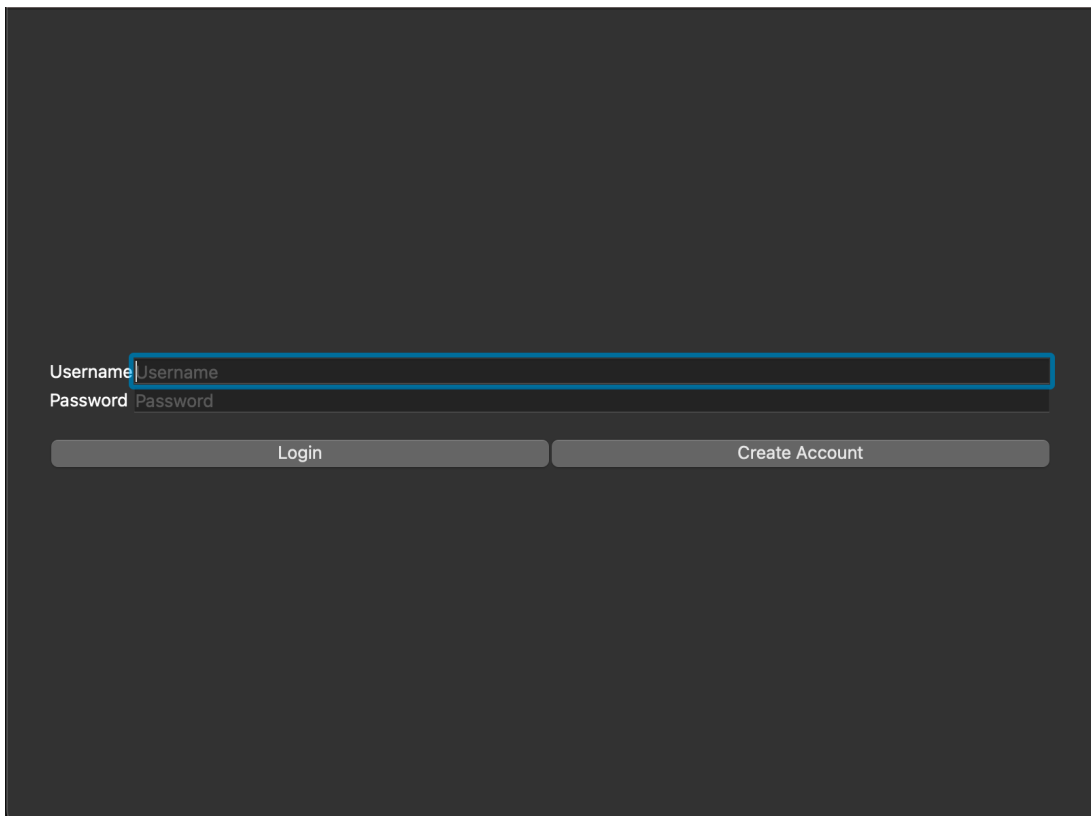
**Abstract:** Acest document reprezintă designul componentelor server/client ale aplicației myFileTransferProtocol. Conține detalii privind tehnologiile utilizate, implementarea logică, exemple de cod utilizat și referințe la documentația externă utilizată

## 1. Introducere

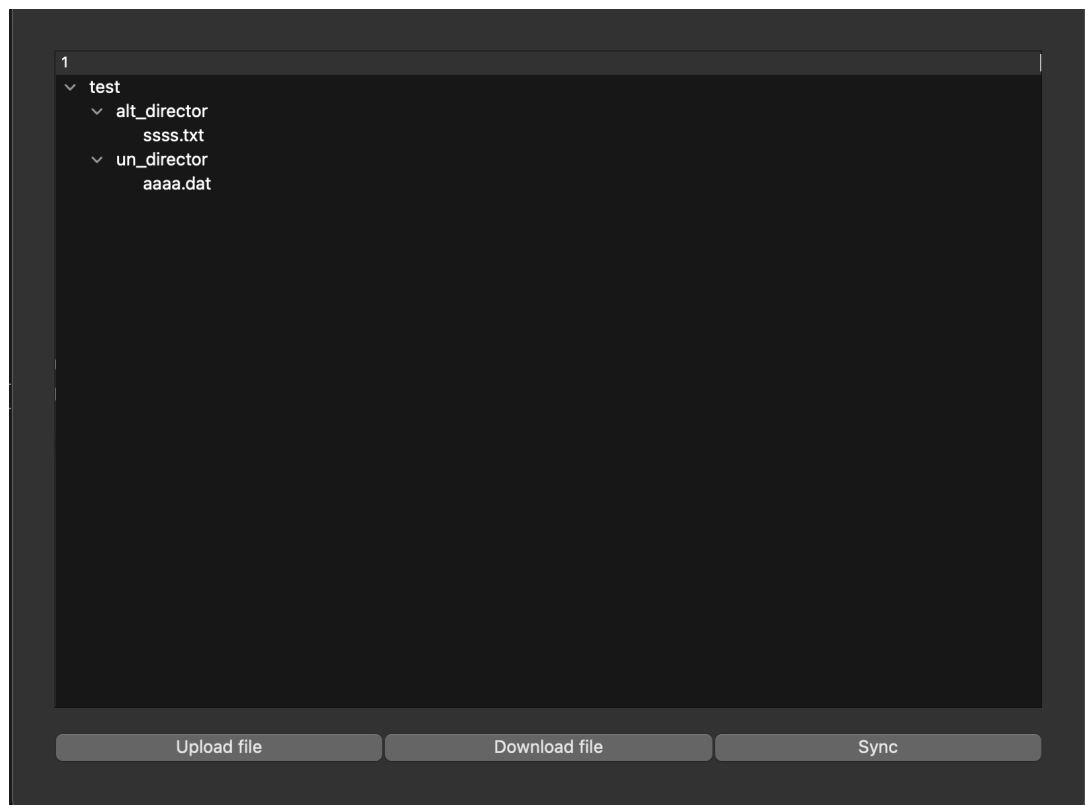
MyFileTransferProtocol este o aplicație desktop client/server, ce operează pe sistemele Linux, care permite utilizatorilor săi transferul de fișiere prin socket-uri Unix. Fiecare utilizator are propriul director pe server, unde sunt stocate fișierele acestuia. Conexiunea cu server-ul este realizată prin trimiterea unei perechi <username, password> de la client către server, cel din urmă validând credențialele primite. Protocolul de autentificare este securizat prin intermediul aplicării unui algoritm de hash peste parolă, înainte ca aceasta să fie trimisă la server.

### 1. Descrierea aplicației client

MyFileTransferProtocol oferă utilizatorilor acces la un sistem de fișiere de pe server și permite încărcarea sau descărcarea acestora. Prin intermediul aplicației client, pentru a beneficia de aceste operații, utilizatorul trebuie să se autentifice într-un cont existent, sau să creeze unul nou.

The image shows a dark-themed user interface for a file transfer protocol client. It features two input fields: 'Username' and 'Password'. The 'Username' field is highlighted with a blue border. Below the input fields are two buttons: 'Login' and 'Create Account'. The 'Create Account' button is disabled, indicated by a lighter gray color.

După ce autentificarea are loc cu succes, un sistem de fișiere va fi afișat cu fișierele utilizatorului stocate pe server.



Utilizatorul are la dispoziție 3 operații: încărcare fișier, descărcare fișier, actualizare sistem de fișiere.

## 1. Descrierea aplicației server

Pentru a accesa fișierele la distanță, aplicația client trebuie să se conecteze la un server dedicat care va procesa toate solicitările clienților. Serverul poate trata toți clienții conectați simultan datorită arhitecturii multi-threading utilizate în implementarea acestuia.

Pentru a asigura autentificarea securizată, parola este hash-uită cu algoritmul SHA256 pe partea client, înainte de a fi trimisă la server. În plus, serverul stochează detaliile de autentificare în perechi `<nume_utilizator, hashed_password>`.

Unii utilizatori sunt interzise de pe server, numele lor de utilizator sunt stocate într-un fișier blacklist, care este analizat pentru fiecare solicitare de conectare, pentru a verifica dacă utilizatorul are autoritatea necesară pentru a accesa aplicația.

## **2. Tehnologii utilizate**

### **2.1 Protocolul TCP/IP pentru comunicarea dintre server și client**

Comunicarea dintre clienți și server are la bază protocolul TCP (Transmission Control Protocol). Acesta este un protocol de transport orientat conexiune, fără pierdere de informații, fapt ce asigură acuratețea transferurilor de date ce au loc în cadrul aplicației, spre deosebire de protocolul UDP (User Datagram Protocol) ce nu asigură integritatea datelor transmise.

### **2.2 Multithreading**

Pentru a asigura concurența, a fost necesară o arhitectură bazată pe multithreading în implementarea serverului. Odată ce un client încearcă să se conecteze la server, se creează un fir nou de execuție și se va distruge atunci când clientul se deconectează de la server.

### **2.3 Qt 6 – Interfața grafică**

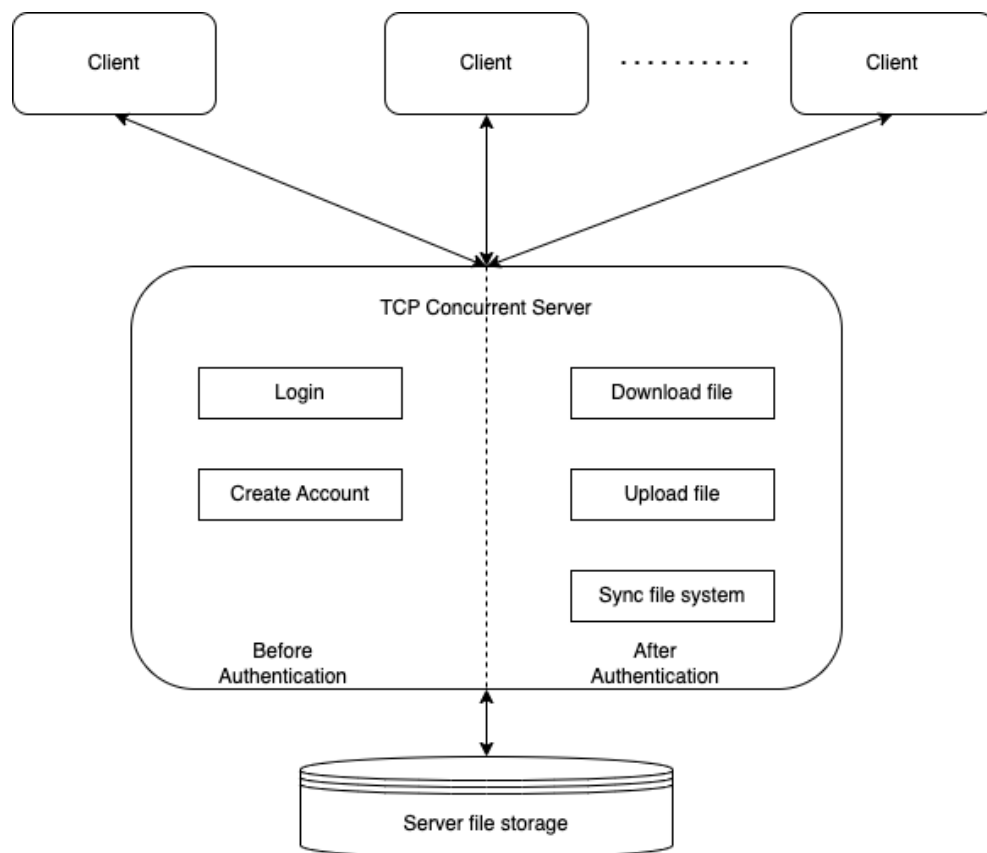
Qt Framework este utilizat pentru dezvoltarea aplicației client datorită flexibilității sale și a numărului mare de module care ar putea fi utilizate în implementarea UI. Mai mult decât atât, oferă un modul criptografic care este utilizat pentru a asigura securitatea autentificării.

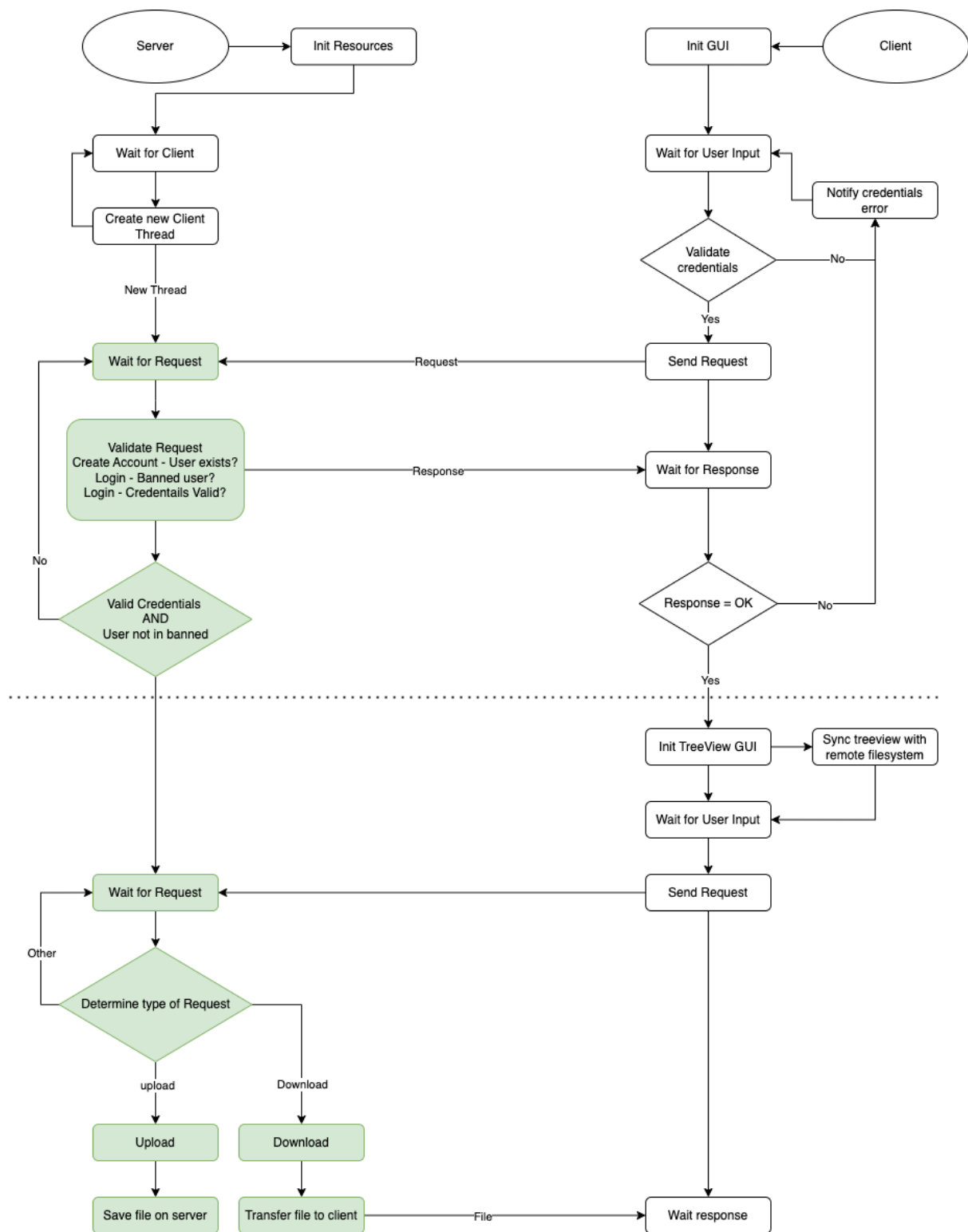
### 3. Arhitectura Aplicației

#### 3.1 Concepte utilizate în implementare

Serverul și clientul (clienții) sunt implicați în arhitectură. Unul sau mai mulți clienți pot stabili conexiunea cu serverul, care tratează cererea lor într-un mod simultan, concurent. După ce se fac solicitările, acestea sunt validate și apoi serverul răspunde cu informațiile necesare.

#### 3.2 Diagrame ale aplicației





## 4. Detalii de implementare. Cod relevant proiectului

După cum s-a menționat în secțiunile anterioare, aplicația se bazează pe un server concurrent. Pentru aceasta, a fost creată o clasă care reprezintă firul de execuție.

```
class Thread {
public:
    int socket;
    pthread_t id;

    Thread(int socket, pthread_t id);

    static void add_thread(Thread * thread);
    static void remove_thread(Thread * thread);
    static void * launch(void *a);
    static void create_thread(Thread * t);
    static void run (Thread & t);

    static std::list<Thread *> thread_list;
    static pthread_mutex_t lock_list;
};
```

Aceste metode sunt utilizate în operațiile cu fire de execuție. Logica principală a server-ului este realizată în metoda `Thread::run()`.

Așa cum am menționat anterior, securizarea autentificării se face prin aplicarea algoritmului SHA256 peste parolă, ca mai apoi criptotextul să fie trimis și stocat în server.

```
QCryptographicHash hasher( method: QCryptographicHash::Sha256);
hasher.addData( data: passwordInput->text().toStdString());
auto hashedPwd : string = QString( a: hasher.result().toHex()).toStdString();
```

Hasher-ul face parte din Qt Framework, iar pentru integrarea cu bibliotecile standard din C++, se face o conversie a rezultatului algoritmului de hash în hexazecimal. De aceea, parola este stocată pe server ca un șir de 64 de caractere simple.

În bucla principală a firului de execuție, sunt utilizate mai multe metode de service pentru a asigura modularitatea. Aceste metode returnează un șir de caractere, asemănător codurilor de stare HTTP.

```
if(m == "REQ_CREATE"){
    string usr, pwd;
    read_message( fd: a.socket, &: usr);
    read_message( fd: a.socket, &: pwd);

    switch(treatCreate(usr, pwd)){
        case 406: write_message( fd: a.socket, m: "RESP_406"); continue;
        case 201: write_message( fd: a.socket, m: "RESP_201"); continue;
        default: write_message( fd: a.socket, m: "RESP_500"); continue;
    }
}

else if(m == "REQ_LOGIN"){
    string usr, pwd;
    read_message( fd: a.socket, &: usr);
    read_message( fd: a.socket, &: pwd);

    switch(treatLogin(usr, pwd)){
        case 403: write_message( fd: a.socket, m: "RESP_403"); continue;
        case 401: write_message( fd: a.socket, m: "RESP_401"); continue;
        case 200: write_message( fd: a.socket, m: "RESP_200"); continue;
        default: write_message( fd: a.socket, m: "RESP_500"); continue;
    }
}
```

Transferul fișierelor se face prin trimiterea de blocuri de o dimensiune predefinită prin socket.

```
static void send_file(int fd, const string &path) {
    write_message(fd, m: "INIT_SEND");

    FILE *file = fopen( filename: path.c_str(), mode: "r");
    char line[256];

    while (!feof(file)) {
        fgets(line, 256, file);
        write_message(fd, m: line);
    }

    fclose(file);

    write_message(fd, m: "FIN_SEND");
}
```

## 5. Scenarii de utilizare

Un utilizator care dorește să utilizeze MyFileTransferProtocol trebuie să aibă un cont pentru a accesa aplicația, introducând apoi credențialele în câmpurile de conectare. Dacă utilizatorul nu are un cont, se poate crea unul nou și, dacă credențialele introduse sunt valide (nu mai există al cont cu același username), se oferă accesul pentru aplicație.

Dacă utilizatorul se află în blacklist sau credențialele furnizate nu se potrivesc cu baza de date, se emite un mesaj de eroare.

După autentificarea cu succes, utilizatorul poate alege între a încărca un fișier pe server, a descărca un fișier de pe server sau a actualiza sistemul de fișiere.

Dacă utilizatorul optează pentru descărcare, trebuie să facă clic pe elementul care trebuie descărcat, apoi să facă clic pe butonul de descărcare. Dacă utilizatorul face clic pe un director, se emite un mesaj de eroare, deoarece numai fișierele pot fi descărcate în aplicația MyFileTransferProtocol.

Dacă utilizatorul optează pentru încărcare, trebuie să facă clic pe directorul în care ar trebui încărcat fișierul, apoi să facă clic pe butonul de încărcare. Va apărea o fereastră de dialog în care utilizatorul trebuie să aleagă ce fișier ar trebui încărcat. Dacă utilizatorul încearcă să încarce un director, se va emite un mesaj de eroare. Dacă utilizatorul selectează calea de încărcare ca fișier, calea de încărcare efectivă va fi directorul părinte al fișierului respectiv.

## 6. Concluzie

Cerințele proiectului vor fi îndeplinite după implementarea și testarea funcționalităților prezentate. În plus, până la termenul limită, proiectul poate suferi actualizări în ceea ce privește interfața grafică cu utilizatorul, logica de implementare sau tehnologiile utilizate care vor îmbunătăți experiența utilizatorului și eficiența aplicației.

## 7. Bibliografie

- Course & Laboratory - Computer Networks 2022-2023 (<https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php>)
- Qt Documentation (<https://doc.qt.io/>)



- StackOverflow – Storing password in database (<https://stackoverflow.com/questions/1054022/best-way-to-store-password-in-database>)
- SHA-2 (<https://en.wikipedia.org/wiki/SHA-2>)
-