Universitatea Alexandru Ioan Cuza

FACULTATEA de
INFORMATICĂ

Proiect: Offline Messenger

Spiridon Vlad-Iustin

Anul II, Grupa B1

iustin.spioridon@gmail.com

2023-2024

Cuprins

Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

1.Introducere

Offline Messenger este o aplicatie de tip client/server care permite schimbul de mesaje intre utilizatorii conectati iar sistemul asigură transmiterea mesajelor către utilizatorii offline atunci când aceștia se reconectează. Funcționalitatea de răspuns specific permite utilizatorilor să selecteze un mesaj și să răspundă direct la acel mesaj, menținând astfel conversația organizată și ușor de urmărit.Aplicația păstrează un istoric detaliat al tuturor conversațiilor, permițând utilizatorilor să revadă mesajele anterioare și să urmărească evoluția discuțiilor. Aplicația gestionează eficient situațiile de conectivitate, asigurându-se că utilizatorii pot comunica fără probleme.

2.Tehnologii Utilizate

I. Server TCP Concurent

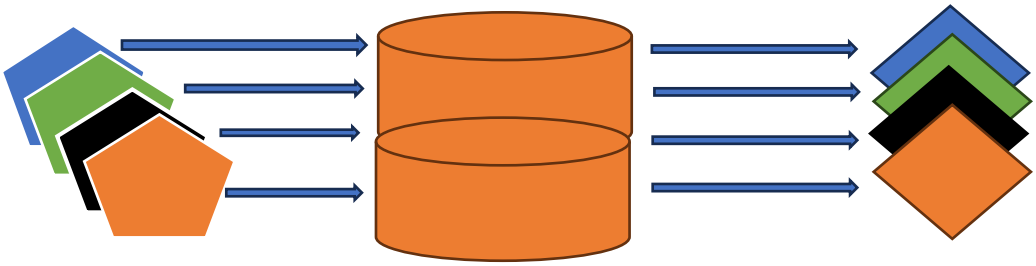
Protocolul de Control al Transmisiei (TCP) reprezintă un standard de comunicare fundamental pentru schimbul de mesaje între aplicații în cadrul unei rețele. Acest protocol este proiectat să faciliteze trimiterea și primirea pachetelor, asigurând, în același timp, livrarea cu succes a acestora. TCP se distinge prin organizarea datelor într-un mod care permite transmiterea eficientă între server și client, oferind granții privind ordinea acestora în rețea,lucru necesar pentru corectitudinea schimbului de mesaje.

II. Socketuri

Socket-urile reprezintă un element fundamental în cadrul comunicației între aplicații, furnizând o interfață pentru schimbul eficient de date între un server și un client. În contextul proiectului, unde este implementat un server TCP cu thread-uri, socket-urile sunt instrumentul principal folosit pentru a facilita această comunicare.

III. Threaduri

Thread-urile reprezintă o componentă esențială, furnizând o abordare eficientă și concurentă pentru gestionarea conexiunilor. În contextul acestei implementări, thread-urile oferă o modalitate optimă de a trata simultan multiple cereri de la clienți diferiți permitand execuția concurentă a multiplelor activități în cadrul serverului. Această paralelizare îmbunătățește semnificativ eficiența și capacitatea comunicarii utilizatorilor.



Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

3.Structura Aplicatiei

Pentru a construi o aplicatie care permite schimbul de mesaje intre utilizatori rapid,eficient si fara pierderi de date , am decis sa folosesc un server TCP concurent implementat cu Thread-uri,folosindu-ne de paralelizarea pe care o pot oferi acestea ca sa permitem utilizatorilor sa comunice in timp real.

Thread-urile sunt create la conectarea clientilor,ulterior realizandu-se comenzile si operatiile introduse de acestia in functia legata de fiecare thread in parte.

Dupa conectarea la server comunicarea dintre clienti se realizeaza pe baza unor comenzi care trebuie introduse de catre acestia(LOGIN,REGISTER,MESSAGE),ulterior ajungand sa trimita dupa bunul plac mesaje intre ei.In program utilizatorii sunt obligati sa se inregistreze sau sa se logheze cu o parola aleasa de ei, pentru a putea comunica in retea, utilizand comenzi precum LOGIN si REGISTER. Dupa ce clientul s-a conectat i se va afisa o lista de utilizatori ,online si offline,cat si de grupuri de utilizatori, , cu care acesta poate comunica.

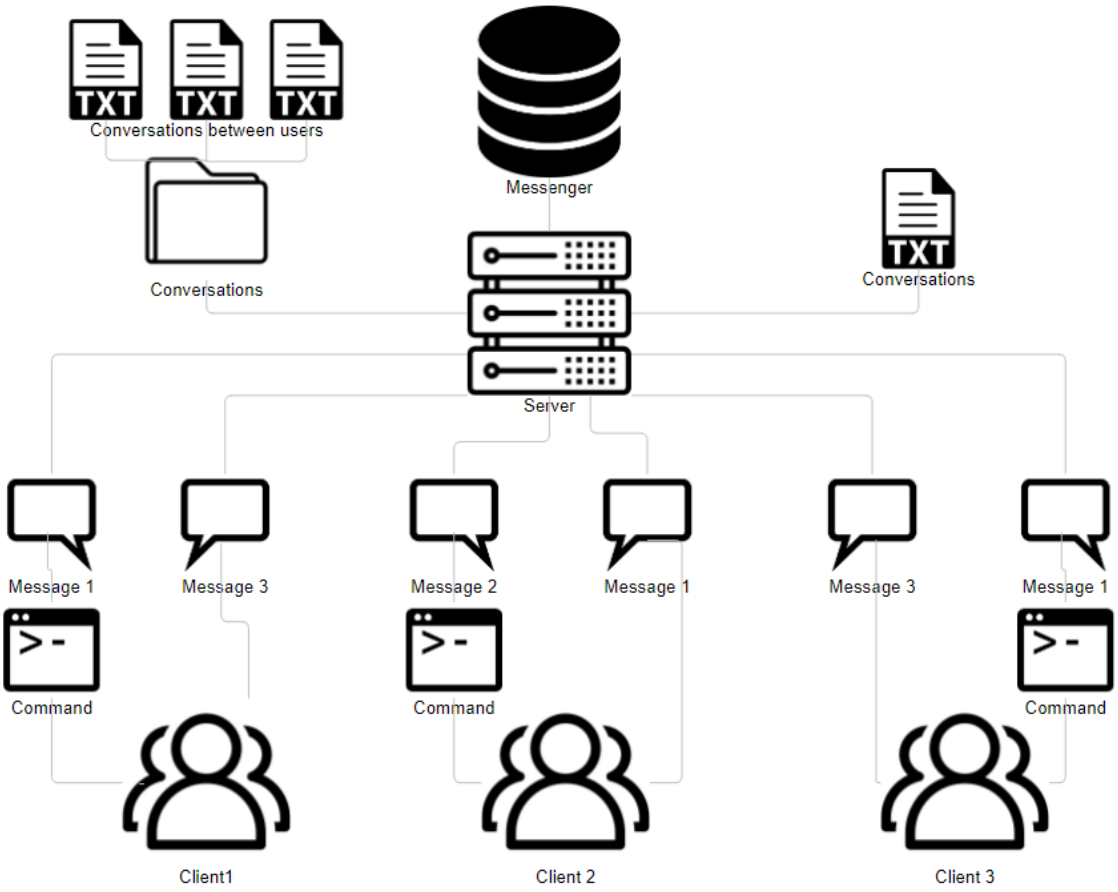
Utilizand functia MESSAGE acesta selecteaza utilizatorul si dupa restul mesajelor vor fi scrise intr un fisier pentru a se putea realiza comunicarea intre utilizatori.Acestia mai au la dispozitie in conversatie anumite comenzi pentru a schimba istoricul mesajelor sau pentru a raspunde la anumite mesaje(REPLY,ERASE,OPEN,EDIT).Comunicarea propriu-zisa se realizeaza utilizand apeluri ale functiilor read/write de la si catre un anumit client,comenzile de la client si mesajele fiind scrise intr un fisier specific conversatiei celor 2,numele conversatiei fiind retinut intr un fisier pt verificare si actualizare, aceste mesaje si comenzi fiind ulterior citite si de catre celalalt client din conversatie.

Utilizatorii sunt capabili sa creeze si grupuri cu ajutorul comenzii Group Name User1.., in aceste grupuri se pot conecta mai multi utilizatori si pot comunica intre ei dupa bunul plac,avand in aceste grupuri aceleasi functionalitati ca intr o conversatie normala,edit,erase,reply.

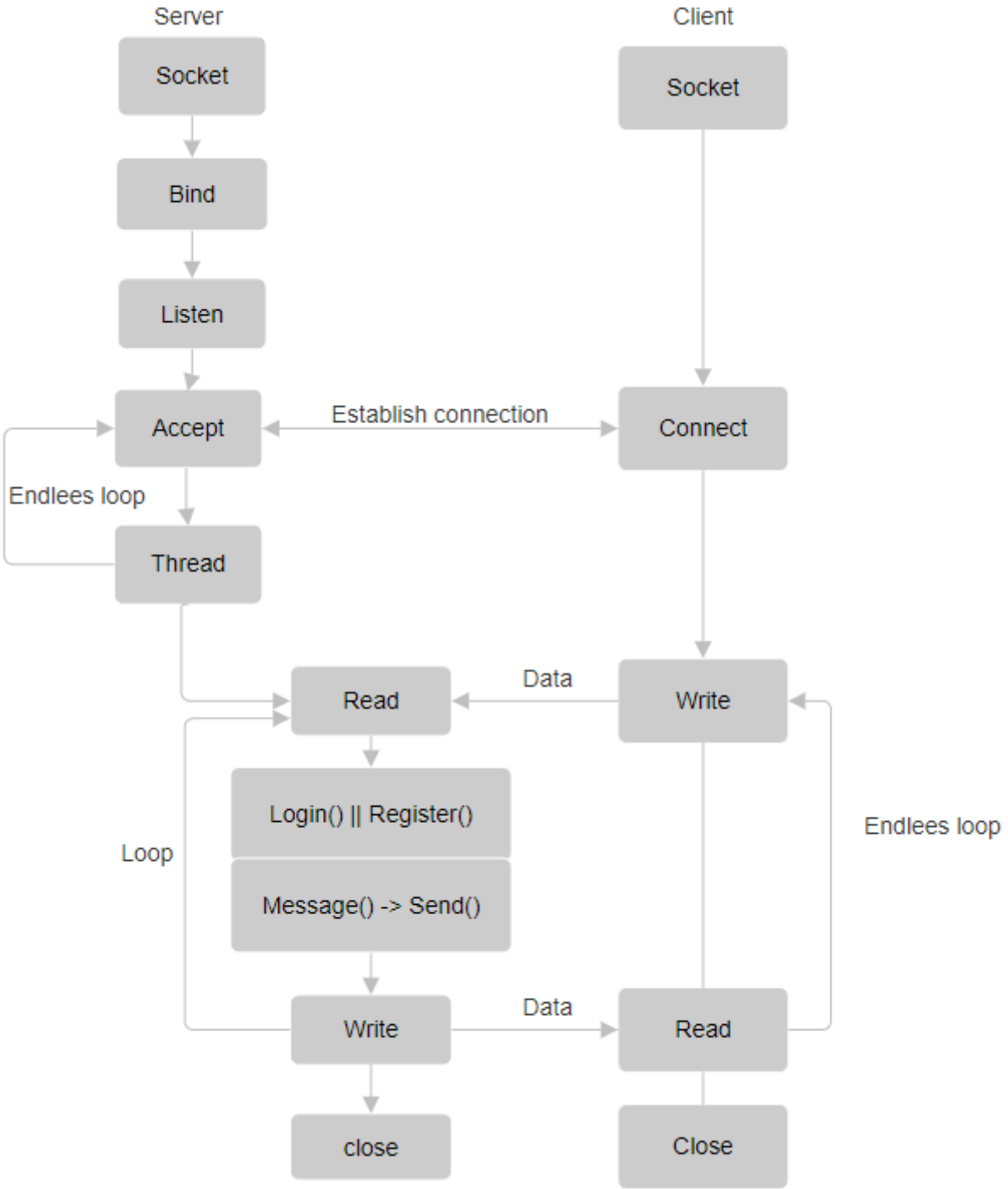
Informatiile utilizatorilor si conversatiilor sunt retinute intr un fisier din care la inceputul programului sunt preluate date de catre Clasa verificand listele de clienti si conversatiilor anterioare create de acestia.

Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

Diagrama Aplicatiei:



Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com



Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

4.Aspecte de implementare

Client:

Implementarea clientului este una tipica, clientul doar realizeaza conexiunea la server si dupa transmite mesaje si primeste mesaje inapoi. Singurul lucru diferit este implementarea unei functii care schimba numele care ii este afisat clientului atunci cand aceste se logheaza,delogheaza sau se inregistreaza.

```
class Client {
public:
    string Name = "Client";
    string Chatter = "Server";
    void ChangeName(const string& received_message) {
        string searchWord1 = "LOGGED IN";
        string searchWord2 = "REGISTERED";
        string searchWord3 = "ALREADY";
        size_t found = received_message.find(searchWord1);

        if (found != string::npos && received_message.find(searchWord3) ==
string::npos) {
            string NewName = received_message.substr(0, found - 1);
            Name = NewName;
        }
        found = received_message.find(searchWord2);
        if (found != string::npos && received_message.find(searchWord3) ==
string::npos) {
            string NewName = received_message.substr(0, found - 1);
            Name = NewName;
        }
    }
};
```

Afisarea mesajelor catre client se face cu numele si destinatarul in fata mesajelor pentru a crea o intelegere mai buna a conversatiei.

```
if (write(sd, message, sizeof(message)) <= 0)
{
    perror("[Client]Eroare la write() spre server.\n");
    return errno;
}

if (read(sd, ServerMesaje, sizeof(ServerMesaje)) < 0)
{
    perror("| [Client]Eroare la read() de la server.\n");
    return errno;
}
if (strcmp(ServerMesaje, "Quit") == 0) {
    printf("| [%s]:Serverul a inchis conexiunea\n", CLient.Chatter.c_str());
    break;
}
CLient.ChangeName(ServerMesaje);
printf("| [%s]:%s\n", CLient.Chatter.c_str(), ServerMesaje);
```


Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

Server:

Structura de baza a serverului este una normala a unui server TCP care foloseste thread-uri pentru a rezolva clientii, concurent.Creez un socket ,umplem structura folosita de server,stabilim familia de socket-uri,atasam socketul,asculta daca vin clienti sa se conecteze si ii serveste in mod concurent utilizand thread-uri si functia legata de acestea.

```
Messenger.Set_Users();
Messenger.Set_Conversations();
struct sockaddr_in server{};
struct sockaddr_in from{};
int sd;
pthread_t th[100];
int i=0;
```

Un obiect de tip Messenger este folosit ca o „baza de date”, stocand informatiile despre clientii conectati si conversatiile lor.Functiile de tip set fiind folosite pentru a pastra continuitatea conversatiilor si utilizatorilor dupa terminarea programului si la initializarea sa.Aceste functii creează mereu la deschiderea serverului vectorii de Useri si de Conversatii cu mesajele pastrate in fisier dupa terminarea programului.

```
class Messenger{
public:
    struct User{
        int id = -1;
        bool online;
        string Username;
        string Chatter;
        User(const string& Username,int id)
    };
    struct Conversation{
        string Client1;
        string Client2;
        string Name;
        int index=0;
        Conversation(string &Client1, string &Client2)
    };
    struct Group{
        string Name;
        vector<string> UserNames;
        explicit Group(string Name, vector<string> Names) {
            this->Name = std::move(Name);
            UserNames = std::move(Names);
        }
        int index=0;
    };

vector<Group> Groups;
vector<User> Users;
vector<Conversation> Conversations;

}
```


Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

O functie de Set(cealalta fiind asemanatoare doar ca initializeaza alte structuri din „baza noastra de date” clasa Messenger):

```
void Messenger::Set_Groups() {
    ifstream inputFile("/home/bolfa/CLionProjects/GROUPS.txt");

    if (!inputFile.is_open()) {
        std::cerr << "Eroare la deschiderea fisierului\n";
    }
    char separator = ' ';
    vector<string> Words;
    string Word;
    string line;
    string Name;
    string line1;
    int index;
    while (getline(inputFile, line)) {
        Words.clear();Word.clear(); Name.clear();
        index=0;
        istringstream parser(line);
        while (getline(parser, Word, separator)) {Words.push_back(Word)}
        Name = Words[0];
  string Path = "/home/bolfa/CLionProjects/CONVERSATIONS/" + Name + ".txt";
        ifstream inputFile1(Path);
        if (!inputFile1.is_open()) {
            cerr << "Eroare la deschiderea fisierului\n";
        }
        line1.clear();
        while (getline(inputFile1, line1)) { index++;}
        inputFile1.close();
        Words.erase(Words.begin());
        Group New_Group(Name,Words);
        New_Group.index = index;
        Groups.emplace_back(New_Group);
    }
```

Clasa Messenger implementeaza si alte functii care au rol in prelucrarea fisierelor de utilizatori si conversatii ,functii precum Verify_Command() care verifica mesajul trimis si returneaza tipul de comanda pe care thread-ul o va executa in functia principala, fiind folosit un switch pentru a apela restul functiilor necesare.In plussunt prezente si alte functii care faciliteaza o mai buna intelegere si utilizare a programului, cat si portabilitatea acestuia.

```
int Verify_Command(string& Command , int &client_id , vector<string>&Words);
void Set_Users();
void Set_Conversations();
void Set_Groups();
static void Create_Conversation(string& Name);
static string Create_Name(string& Name1,string& Name2);
bool Add_Conversation(string& path,string&,string&);
static bool Add_Name(string &path,string& Name);
static vector<string> ReadLastLines(string& filePath, int numLines);
static void WriteInFile(string& filePath,string& message);
static std::string ReadLineFromFile(const string& Path, int index);
static bool ChangeLineFromFile(const string& Path, int lineNumber,string
Message);
string AttachConv(thData tdL);
void CheckUserGroup(thData tdL,bool& user,bool& group);
static string GetPathForConv(User& Client, bool& exist);
```


Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

Functia Verify_Command():

```
int Messenger::Verify_Command(string& Command,int &client_id,vector<string>
&Words) {

    char separator = ' ';
    istringstream parser(Command);
    string Word;
    while (getline(parser, Word, separator)) {
        Words.push_back(Word);
    }
    string VerifyConversation = "Reply Group End Quit Check Open Erase Edit";
    static const std::unordered_map<std::string, int> Commands = {
{"Login", 1},{"Register", 2},{"Logout", 3},{"Message", 4}{"Quit", 5}{"Reply",
6},{"Group", 7},{"End", 8},{"Check", 10},{"Open", 11},{"Erase", 12},{"Edit",
13}
    };

    for (const auto& it: Users) {
        if (it.id == client_id && it.online && !it.Chatter.empty() && Verify-
Conversation.find(Words[0]) == string::npos) {
            return 9;
        }
    }

    auto it = Commands.find(Words[0]);
    if (it != Commands.end())
        return it->second;
```

Una dintre functiile pentru prelucrarea fisierelor ,deoarece majoritatea sunt destul de simple si au o implementare de baza (`ReadLineFromFile, WriteInFile`).

Functia Add_Conversation() care se apeleaza cand un utilizator deschide o conversatie, dupa o anumita comanda.Aceasta functiie verifica daca conversatia exista deja, daca nu o adauga in txt-ul cu numele conversatiilor,aceasta fiind creata ulterior in alta functiel.

```
bool Messenger::Add_Conversation(string& path,  string& Client1, string&
Client2) {

    ifstream inputFile(path);
    string line;
    for(auto &it:Conversations){
        if(it.Client1 == Client1 && it.Client2 == Client2 || it.Client1 ==
Client2 && it.Client2 == Client1){
            return false;
        }
    }
    string Name1 = Create_Name(Client1,Client2);
    string Name2 = Create_Name(Client2,Client1);
    while (std::getline(inputFile, line)) {
        if (line == Name1 || line == Name2) { inputFile.close(); return false;
    inputFile.close();
    std::ofstream outputFile(path, std::ios::app);
    if (!outputFile.is_open()) {
        cerr << "Eroare la deschiderea fisierului\n";
        return false;
    }
    outputFile << Name1 << '\n';
    outputFile.close();
```


Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

Clasa Messenger implementeaza si functiile care reprezinta prelucrarea comenzilor date de catre clienti.

```
void Login(thData tdL,string& Name);
string GroupUsers(thData tdL,vector<string> Words);
void Verify_Password(thData tdL,string& Name);
void Set_Password(thData tdL);
void Register(thData tdL,string& Name);
string Logout(thData tdL);
string Quit(thData tdL);
string Message(thData tdL,string &Name);
string End(thData tdL);
vector<string> Check_Users();
vector<string> Open(thData tdL);
void Send(thData tdL,string& Message);
void Reply(thData tdL,string& Message, string& index);
string Erase(thData tdL,string& index);
string Edit(thData tdL,string& index,const string& Message);
```

Fiecare functie se apeleaza dupa tipul returnat de functia Verify_Command() si implementeaza comanda cu numele respectiv,mai putin functia Send care se apeleaza atunci cand userul trimite mesaje dupa ce a introdus comanda Message USER.

```
string Messenger::Message(thData tdL, string &Name) {
    string Message;

    if (!Users.empty()) {
        bool exist = false;
        for (auto &it: Users) {
            if (it.Username == Name) {
                exist = true;
                break;
            }
        }
        if (exist) {
            for (auto &i: Users) {
                if (i.id == tdL.cl && i.online) {
                    i.Chatter = Name;
                    Message += i.Username + " --> " + Name;
              string Path = "/home/bolfa/CLionProjects/CONVERSATIONS.txt";
                    if(Add_Conversation(Path, i.Username,Name)){
       Conversation New_Conversation(i.Username,Name,tdL.cl,i.id_chatter);
                        Conversations.push_back(New_Conversation);
                    }
                    break;
                }
            }
        } else {
            Message += "[USER " + Name + " DOES NOT EXIST]";
        }
    } return Message; }
```


Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

Fiecare mesaj are la inceputul sau indexul din conversatie pentru a ii fi utilizatorului mai usor sa raspunda la mesaej(functiile Reply,Erase,Edit,Open,End se pot apela doar dupa ce utilizatorul a inceput conversatia cu comanda Message).

```
void Messenger::Send(thData tdL,string& Message) {
    bool exist = false;
    string Path = "/home/bolfa/CLionProjects/CONVERSATIONS/";
    string ConvName1, ConvName2, ConvName;
    string NewMessage;

    bool user = false, group = false;
    CheckUserGroup(tdL, user, group);

    if (user) {
        for (auto &i: Users) {
            if (i.id == tdL.cl && i.online && !i.Chatter.empty()) {
                Path = GetPathForConv(i, exist);
                if (exist) {
                    for (auto &it: Conversations) {
                        if (it.Name == Create_Name(i.Username, i.Chatter) ||
                            it.Name == Create_Name(i.Chatter, i.Username)) {
                            NewMessage += "[" + to_string(it.index) + "] " +
"[" + i.Username + "]: " + Message;
                            it.index++;
                            WriteInFile(Path, NewMessage);
                            break;
                        }
                    }
                }
            }
        }
    } else if (group) {
        Path = "/home/bolfa/CLionProjects/CONVERSATIONS/";
        for (auto &it: Users) {
            if (it.id == tdL.cl && it.online && !it.Chatter.empty()) {
                for (auto &i: Groups)
                    if (it.Chatter == i.Name) {
                        Path += i.Name + ".txt";
                        NewMessage = "[" + to_string(i.index) + "] " + "[" +
it.Username + "]: " + Message;
                        i.index++;
                        WriteInFile(Path, NewMessage);
                        break;
                    }

            }
        }
    }
}
```

Restul functiilor sunt asemanatoare la inceput deoarece toata cauta sa vada ce utilizator a trimis mesajul, din lista de useri .Dupa ce este gasit utilizatorul, functia va schimba ce este necesar la clasa (daca se logheaza sau se delogheaza, daca se inregistreaza,daca editeaza un mesaj, daca sterge un mesaj,daca afiseaza conversatia,daca inchide conversatia sau daca inchide de tot programul)

Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

Functia Reply pt a raspunde la anumite mesaje din fisier,aceasta construieste un mesaj intuitiv pentru intelegerea conversatiei.

```
void Messenger::Reply(thData tdL, string &Message,string& index){
    bool exist=false;
    string Path = "/home/bolfa/CLionProjects/CONVERSATIONS/";
    string ConvName1,ConvName2,ConvName;
    string NewMessage;

    bool user = false,group = false;
    CheckUserGroup(tdL,user,group);
    if(user) {
        for (auto &i: Users) {
            if (i.id == tdL.cl && i.online && !i.Chatter.empty()) {
                Path = GetPathForConv(i, exist);
                if (exist) {
                    for (auto &it: Conversations) {
                        if (it.Name == Create_Name(i.Username, i.Chatter) ||
                            it.Name == Create_Name(i.Chatter, i.Username) &&
it.index > stoi(index)){
                            NewMessage +="[" + to_string(it.index) + "] " +
"[" + i.Username + "] [REPLY] [" + index + "]: ";
                            NewMessage += Message;
                            it.index++;
                            WriteInFile(Path, NewMessage);
                            break;
                        }
                    }
                }
            }
        }
    }
    else if(group){
        Path = "/home/bolfa/CLionProjects/CONVERSATIONS/";
        for (auto &it: Users) {
            if (it.id == tdL.cl && it.online && !it.Chatter.empty()) {
                for (auto &i: Groups)
                    if (it.Chatter == i.Name && i.index > stoi(index)) {
                        Path += i.Name + ".txt";
                        NewMessage = "[" + to_string(i.index) + "] " + "[" +
it.Username + "] [REPLY] [" + index + "]: ";
                        NewMessage += Message;
                        i.index++;
                        WriteInFile(Path, NewMessage);
                        break;
                    }
            }
        }
    }
```


Spiridon Vlad-Iustin
Iustin.spiridon@gmail.com

5.Concluzii

Proiectul realizat de mine , indeplineste toate cerintele mentionate, in enuntul proiectului respectiv, si am reusit sa adaug alte functionalitati pe langa acestea precum optiunea de a edita si sterge mesaje, cat si realiza unor grupuri de utilizatori in care acestia isi pot trimite mesaje reciproc . Desi nu are interfata grafica, proiectul incearca sa fie cat mai atractiv pentru utilizatori, avand toate informatiile necesare in consola, cu indici pentru mesaje ,numele afisat al clientului s.a.

Proiectul poate fi imbunatatit atat pe partea de back-end cat si de front-end,utilizand o interfata grafica atrativa si o baza de date eficientizata pentru stocarea mesajelor si a informatiilor legate de conturi,in rest ,aplicatia este gandita sa fie eficienta si usor de utilizat.

6.Referințe Bibliografice

1. https://sites.google.com/view/fii-lab-retele-de-calculatoare/laboratoare
2. https://profs.info.uaic.ro/~computernetworks/files/6rc_ProgramareaInReteaII_En.pdf
3. https://profs.info.uaic.ro/~computernetworks/files/7rc_ProgramareaInReteaIII_En.pdf
4. https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
5. https://profs.info.uaic.ro/~computernetworks/files/NetEx/S9/cliTcp.c