OFFLINE MESSENGER

BURADA IRINA 2A6

1 INTRODUCERE

1.1 SCOPUL APLICAȚIEI

OFFLINE MESSENGER este o aplicație care respectă modelul server/client și simulează o parte dintre serviciile de mesagerie clasice.

Astfel, primele funcționalități ale aplicației vor consta în logarea unui utilizator deja existent în baza de date sau înregistrarea unui client oarecare. Realizarea acestor pași vor conduce la deblocarea unor noi utilități ale aplicației, precum trimiterea unui mesaj către un utilizator offline sau online(statusul se schimbă în momentul logării), vizualizarea istoricului de mesaje cu fiecare utilizator și pentru fiecare utilizator, iar această ultimă utilitate poate permite utilizatorului să dea reply unui anumit mesaj dintr-o conversație cu un alt utilizator.

Luând în considerare aceste funcționalități, implemetarea lor ar presupune utilizarea unui server care va comunica în mod concurent cu clienții, oferind în felul acesta atât posibilitatea comunicării în timp real, precum și a comunicării asincrone, deoarece utilizatorii offline își vor putea vedea mesajele în momentul conectării și vor putea răspunde fiecăruia în mod particular.

1.2 MOTIVAȚIE

Într-o perioadă în care comunicarea la distanță a devenit cea mai uzuală metodă de conectare a oamenilor, toate aplicațiile de mesagerie au devenit esențiale în viața omului modern. Din acest motiv, proiectul Offline Messenger mi-a atras atenția,
oferindu-mi posibilitatea de a face propria aplicație de messenger, care deși va avea utilitățile deja cunoscute, va purta amprenta mea în modul de structurare.
Pe de-o parte, având în vedere că serviciile de mesagerie se află integrate în toate rețelele de socializare actuale, am considerat că un proiect de felul acesta, deși implementat într-o manieră simplistă, mă va ajuta să înțeleg conceptele aflate la baza unei aplicații de o anvergură mai mare.
De asemenea, un alt motiv care m-a determinat să aleg acest proiect este dat de frecvența cu care utilizez eu serviciile de mesagerie în viața de zi cu zi. De aici, consider că aplicația și-ar putea găsi utilizarea în diferite arii, precum domeniul vânzarilor, al pieții de muncă sau în scop educativ, pentru a menține o comunicare mai ușoară între utilizatori cu scopuri comune.

Așadar, Offline Messenger pare un proiect potrivit pentru învățarea noțiunilor legate de comunicarea în rețea, care mi-ar putea oferi în același timp satisfacția creării unei aplicații utile și de interes în zilele noastre.

2 TEHNOLOGII UTILIZATE

2.1 C/C++

Aplicația va fi implementată in C/C++.

C este un limbaj de programare standardizat, apreciat pentru eficiența codului obiect generat de compilatoarele C si pentru portabilitate. Facilitățile adaugate C-ului au fost clase, clase derivate, verificarea tipului, inline și argumente cu valori implicite, ducând astfel la dezvoltarea limbajului C++[1]. C++ este un limbaj hibrid sau un limbaj compilat, pentru că este o combinație între un limbaj procedural și un limbaj de programare orientat pe obiecte. [2]

În anii 1990, C++ a devenit unul dintre cele mai populare limbaje de programare comerciale, rămânând astfel până azi. [3]

2.2 SQLite

SQLite este o bibliotecă compactă din C care implementează un motor de baze de date SQL tranzacțional, autonom, fără server, cu configurație zero. SQLite este cel mai răspândit motor de baze de date SQL din lume. [4]

SQLite este diferit de majoritatea altor motoare de baze de date SQL prin aceea că a fost proiectat pentru a fi simplu de administrat, de folosit, de a fi încapsulat într-un program mai mare, de întreținut și de setat.[5]

Având în vedere aceste caracteristici, voi folosi SQLite pentru a construi o bază de date în care voi reține 2 tabele (fig 1.)

1. Tabelul utilizatori: id_utilizator, nume utilizator, parola, statusul(va conține datele de conectare, precum și statusul utilizatorului, dacă este offline sau online)
2. Tabelul mesaje: id_mesaj, id_emițător, id_receptor, text_mesaj, id_reply, seen (pentru a ține evidența mesajelor trimise, de cine au fost trimise și către cine).

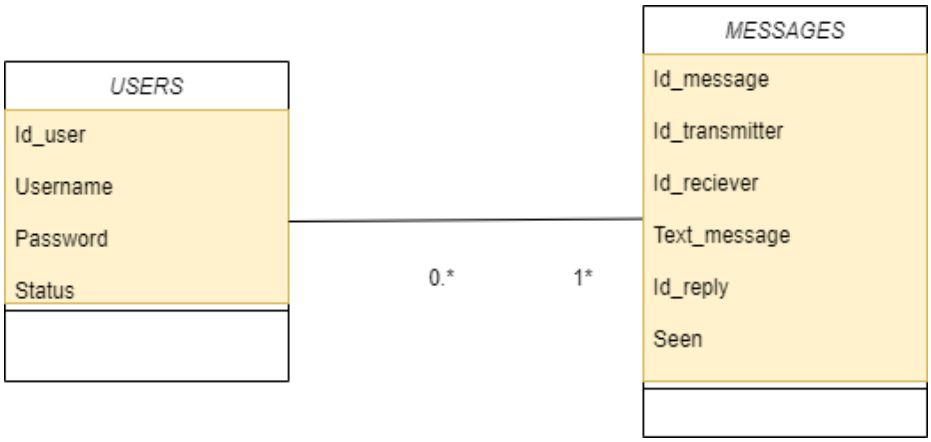

Fig. 1.

2.3 TCP

Este un protocol de comunicare la nivel transport, orientat conexiune, ceea ce înseamnă că atât expeditorul cât și destinatarul trebuie să participe la realizarea conexiunii. Abstracțiunile fundamentale sunt conexiunile, ci nu porturile, iar acestea se identifică prin perechi de tipul adresa IP:PORT(soclu – socket). Un soclu poate fi partajat de conexiuni multiple de pe aceeași mașină. [6]
Acest protocol fragmentează fluxul de octeți în mesaje discrete și pasează fiecare mesaj nivelului internet.TCP tratează totodată controlul fluxului pentru a se asigura că un emițător rapid nu inundă un receptor lent cu mai multe mesaje decât poate acesta să prelucreze. [7]

Mecanismul general de control al fluxului presupune trimiterea un pachet, urmată de așteptarea confirmării pentru a trimite următorul pachet. În cazul în care confirmarea (reprezentată prin flagul ACK) nu este trimisă în intervalul de timp prestabilit, pachetul va fi trimis din nou.[6]

Având în vedere proprietățile prezentate mai sus, am ales protocolul TCP, deoarece este sigur și garantează faptul că un pachet a ajuns la destinație, aspect elementar aplicației Offline Messenger, în cazul în care vom dori să fim convinși, de exemplu, că mesajele trimise de un utilizator au ajuns fără erori și în ordine la destinatar. Din acest motiv,TCP este mai potrivit decât UDP, protocolul în care nu se folosesc conexiuni și nu se așteaptă confirmarea primirii unui pachet.

3 ARHITECTURA APLICAȚIEI

3.1 Concepte implicate

Modelul client/server

Serverul se ocupă de acceptarea clienților, de îndeplinirea unor servicii pentru aceștia, precum și de trimiterea răspunsului, servirea clienților realizându-se în mod concurent. Clienții nu realizează operații specifice, ci inițiază conexiunea cu serverul, căruia îi transmite o anumită solicitare. Se pot conecta mai mulți clienți simultan la server și să ceară servicii precum trimiterea unui mesaj către un alt utilizator, afișarea istoricului de mesaje sau trimiterea unei replici pentru un anumit mesaj din istoric.

Procese copil pentru implementare concurentă

După realizarea conexiunilor cu clienții, serverul va crea procese copil folosind primitiva fork() pentru fiecare dintre aceștia. Aceste procese copil se vor ocupa în continuare de îndeplinirea cererilor clientului, iar încheierea perioadei lor de viață va fi în momentul în care clientul respectiv a încheiat comunicarea cu serverul.

3.2 Diagrama aplicației

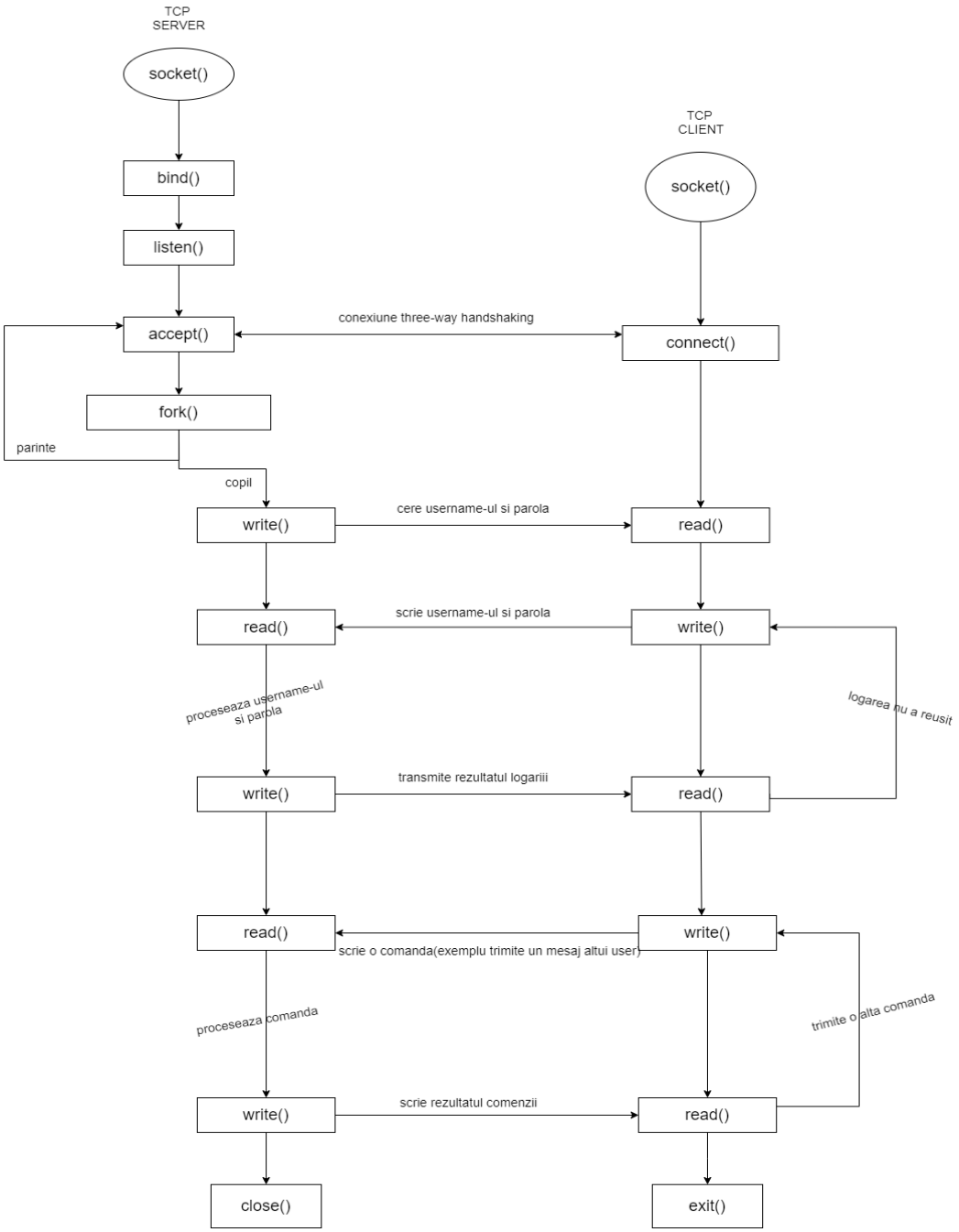

Fig. 2.

4 DETALII DE IMPLEMENTARE

4.1 Cod relevant particular

Server (demo fig. 3)

```
#define PORT 2050

extern int errno;
void registerr(char * user, char * password, int client);
void changeInfoClient(int client, char msgrasp[], char
msg[]);
void commands(int client);

void connect(int client){

    bool found=false;
    sqlite3 *db;
    sqlite3_stmt * stmt;
    char msgrasp[100] = " ";
    char user[100];

    char msg[200];

    strcat(msgrasp, "Introdu username-ul(pentru auten-
tificare sau inregistrare) : ");
    changeInfoClient(client, msgrasp, user);

    if (sqlite3_open("proiect", &db)) {
        printf("Could not open the.db\n");
        exit(-1);
    }

    sqlite3_prepare_v2(db, "select * from Users", -1,
&stmt, NULL);

    printf("Got results:\n");

    const char *pass;

    while (sqlite3_step(stmt) != SQLITE_DONE && !found) {
            const char *database_user = reinter-
pret_cast<const char*>(sqlite3_column_text(stmt, 1));
```

```
    if(strlen(user) == strlen(database_user)) {

           if (strcmp(database_user, user) == 0) {
                    found = true;
             pass = reinterpret_cast<const char
*>(sqlite3_column_text(stmt, 3));

                    strcat(msgrasp, "Am gasit utiliza-
torul. Introduceti parola: ");
                    changeInfoClient(client, msgrasp,
msg);

                    while(strcmp(pass, msg)){

                        printf("%s %s", pass, msg);
                        strcat(msgrasp, "Parola in-
corecta(introdu iar parola sau scrie 'stop': ");
                        changeInfoClient(client, msgrasp,
msg);

                        if(strcmp(msg, "stop") == 0)
                            connect(client);

                    }
                    if(strcmp(pass, msg) == 0)
                    {
                        bzero(msgrasp, 100);
                        commands(client);

                    }


                }
            }


        printf("\n");

    }



    if(!found) {
```

```
        strcat(msgrasp, "Nu am gasit utilizatorul. Doriti
sa va inregistrati cu acest user? D/N : ");

        changeInfoClient(client, msgrasp,msg);

        while(strcmp(msg,"D") && strcmp(msg,"d") &&
strcmp(msg,"N") && strcmp(msg,"n"))
        {
            strcat(msgrasp, "Comanda nerecunoscuta. In-
troduceti D/N : ");
            changeInfoClient(client, msgrasp,msg);
        }

        if(strcmp(msg, "D") == 0 || strcmp(msg, "d") ==
0) {

            char pass1[100];
            strcat(msgrasp, "Introduceti parola pentru a
va inregistra: ");
            changeInfoClient(client, msgrasp,pass1);

            strcat(msgrasp, "Confirmati parola : ");
            changeInfoClient(client, msgrasp, msg);

            if(strcmp(pass1, msg) == 0) {registerr(user,
msg, client);}
            else connect(client);


        }

        else if(strcmp(msg, "N") == 0 || strcmp(msg, "n")
== 0) {
            connect(client);
        }




        exit(2);

    }
```

```
    sqlite3_finalize(stmt);

    sqlite3_close(db);
}

void registerr(char *user, char *password, int client){

    sqlite3 *db;
    sqlite3_stmt * stmt;
    if (sqlite3_open("proiect", &db)) {
        printf("Could not open the.db\n");
        exit(-1);
    }

    char * err_msg = 0;
    string sql;
    sql = "INSERT INTO Users VALUES (1,?,null,?);";

    sqlite3_prepare(db, sql.c_str(), -1, &stmt, NULL);

    password[strlen(password)] = '\0';
    if(stmt != NULL) {
        sqlite3_bind_text(stmt,  1,  user,  strlen(user),
SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 2, password, strlen(pass-
word), SQLITE_TRANSIENT);
        sqlite3_step(stmt);
        sqlite3_finalize(stmt);
    }


    commands(client);
    sqlite3_close(db);
}

void commands(int client){

    char msgrasp[100];
    char msg[200];
    while(1)
    {
        strcat(msgrasp,    "Esti    conectat.Poti    face
urmatoarele comenzi :- 1 trimite mesaj;- 2 vezi istoric; -
3 deconectare: ");
```

```
        changeInfoClient(client, msgrasp, msg);

        if(strcmp(msg, "3") == 0) {connect(client); break;
}

    }
}

void changeInfoClient(int client, char msgrasp[], char
msg[])
{
    int length_answer = strlen(msgrasp);
    int length_comand;
    write(client, &length_answer, sizeof(int));

    if (write(client, msgrasp, length_answer) <= 0) {
        perror("[server]Eroare la write() catre cli-
ent.\n");

    }

    if (-1 == read(client, &length_comand, sizeof(int))) {
        perror("Eroare la Read:");
    }
    bzero(msg, 100);
    if (read(client, msg, length_comand) <= 0) {
        perror("[server]Eroare la read() de la cli-
ent.\n");
        close(client);
    }

    msg[strlen(msg) - 1] ='\0';
    bzero(msgrasp, 100);
}

int main () {
    struct sockaddr_in server;
    struct sockaddr_in from;
    int optval=1;
    char msg[100];

    int sd;
    bool found = false;

    /* crearea unui socket */
```

```
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("[server]Eroare la socket().\n");
        return errno;
    }

    setsockopt(sd,                                SOL_SOCKET,
SO_REUSEADDR,&optval,sizeof(optval));

    /* pregatirea structurilor de date */
    bzero(&server, sizeof(server));
    bzero(&from, sizeof(from));


    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(PORT);


    if (bind(sd, (struct sockaddr *) &server, sizeof(struct
sockaddr)) == -1) {
        perror("[server]Eroare la bind().\n");
        return errno;
    }

    /* punem serverul sa asculte daca vin clienti sa se
conecteze */
    if (listen(sd, 1) == -1) {
        perror("[server]Eroare la listen().\n");
        return errno;
    }

    /* servim in mod concurent clientii... */
    while (1) {
        int client;
        socklen_t length = sizeof(from);

        printf("[server]Asteptam   la   portul   %d...\n",
PORT);
        fflush(stdout);

        /* acceptam un client (stare blocanta pina la re-
alizarea conexiunii) */
        client  =  accept(sd,  (struct  sockaddr  *)  &from,
&length);
```

```
        /* eroare la acceptarea conexiunii de la un client
*/
        if (client < 0) {
            perror("[server]Eroare la accept().\n");
            continue;
        }

        int pid;
        if ((pid = fork()) == -1) {
            close(client);
            continue;
        } else if (pid > 0) {
            // parinte
            close(client);
            while (waitpid(-1, NULL, WNOHANG));
            continue;
        } else if (pid == 0) {
            // copil
            close(sd);

            bzero(msg, 100);

            /* citirea mesajului */


            printf("[server]Mesajul    a    fost    recep-
tionat...%s\n", msg);

            connect(client);


            close(client);
            exit(0);
        }

    }
}
```

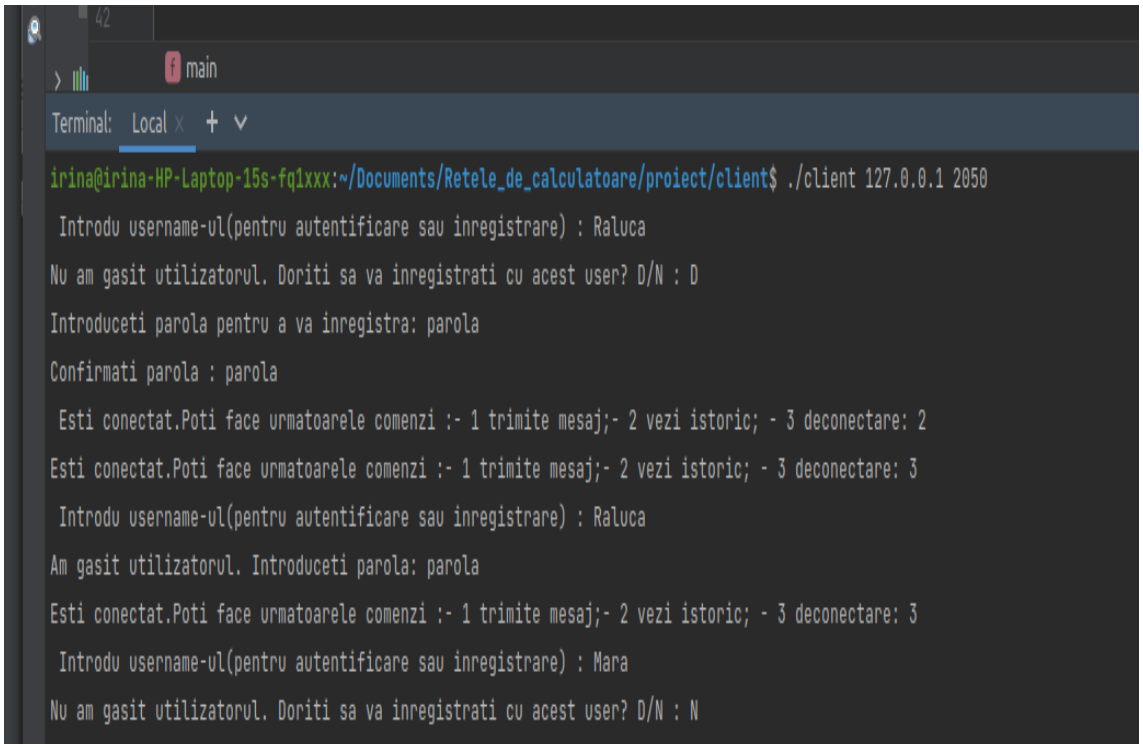

Fig. 3.

4.2 Scenarii de utilizare

Scenariu corect de utilizare

Un scenariu corect de utilizare ar presupune stabilirea unei conexiuni între client și server, acesta din urmă creând un fiu căruia îi va fi asociat clientul venit. Copilul creat se ocupă de procesarea datelor de conectare. După introducerea username-ului, acesta va căuta în baza de date dacă există un astfel de user, urmând ca în caz afirmativ să ceară și parola, iar în caz contrar să întrebe clientul dacă vrea să se înregistreze cu acel username. Dacă înregistrarea, respectiv logarea s-a realizat cu succes, atunci clientul are acces la o serie de comenzi cum ar fi trimiterea unui mesaj către un utilizator, eventual ca replică la un mesaj specific, precum și afișarea istoricului de mesaje. În cazul în care alege să trimită un mesaj către un utilizator, va avea posibilitatea să aleagă destinatarul dintr-o listă de utilizatori online sau offline, să scrie mesajul și să primească un răspuns de la server cum că mesajul a fost trimis fără erori. După servirea acestui client, fiul își încheie activitatea, însă în tot acest timp serverul a putut primi alți clienți pe care i-a predat de asemenea altor fii.(fig. 4)

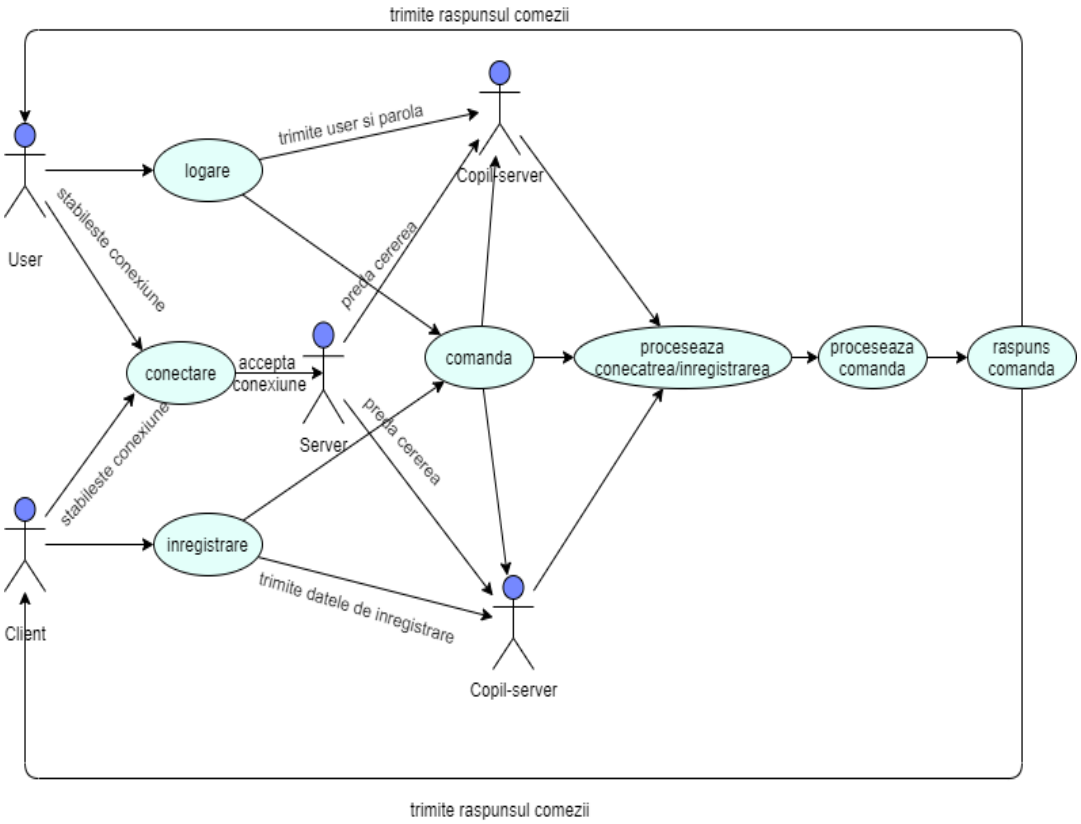

Fig. 4.

Scenariu de utilizare cu erori

Un scenariu care nu urmărește un flux corect ar putea fi reprezentat de cazul în care serverul nu poate răspunde la cererea de conexiune a clientului din cauza unor probleme tehnice ale calculatorului pe care rulează. O altă situație ar putea apărea dupa stabilirea conexiunii dintre client și server, mai exact în momentul căutării username-ul primit când baza de date nu poate fi accesată. În ambele situații serverul va returna un mesaj de eroare clientului.(fig 5).

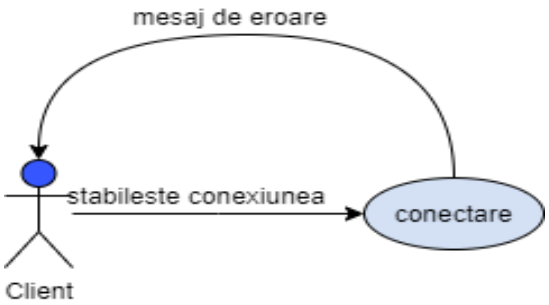

Fig. 5.

5 CONCLUZII

În final, aplicația are următoarele funcționalități: un client va avea de la început posibilitatea să se logheze sau să se înregistreze, ceea ce îi va da în continuarea fie posibilitatea să intre în lista de utilizatori, fie să se deconecteze. Odată aleasă prima opțiune, utilizatorul va putea vedea lista celorlalți ultilizatori înregistrați, statusul lor în acel moment, precum și notificările pe care le are și i se va cere id-ul utilizatorului cu care vrea să interacționeze, urmând să afișeze istoricul mesajelor cu acesta(sub forma username-Emitator : mesajul propriu-zis (id-ul mesajului, precum și id-ul mesajului pentru care este replică, în cazul în care există) – precum și dacă mesajul pe care utilizatorul curent l-a trimis a fost văzut). După deschiderea conversației, utilizatorul va putea trimite un mesaj direct, un mesaj ca replică la un alt mesaj sau să se întoarcă la lista de utilizatori. Trimiterea cu succes a oricărui mesaj îi va da posibilitatea să trimită un alt mesaj sau să aleagă alt utilizator cu care să converseze.

Îmbunătățirile aplicației ar putea consta în criptarea și decriptarea parolei, în realizarea unei interfețe ușor de folosit pentru client, în posibilitatea creării de grupuri(ceea ce ar presupune și utillizarea unor priorități a utilizatorilor), precum și trimiterea de fișiere media.

6 BIBLIOGRAFIE

1. C wiki https://ro.wikipedia.org/wiki/C%2B%2B
2. C++ https://codecool.com/ro/blog/ghid-c-incepatori/
3. C++ wiki https://ro.wikipedia.org/wiki/C%2B%2B
4. SQLITE homepage https://www.sqlite.org/about.html
5. SQLITE wiki https://ro.wikipedia.org/wiki/SQLite
6. TCP https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php
7. TCP wiki https://ro.wikipedia.org/wiki/TCP/IP