LearNet

Marin Malina 2A6

Universitatea A.I.Cuza Iasi, Facultatea de Informatica
malina.marin@info.uaic.ro

Abstract. Proiectul constă in realizarea unei aplicatii client-server care sa ajute la invatarea principalelor notiuni de Retele de Calculatoare.

Keywords: client-server · TCP · protocol.

1 Introducere

Proiectul LearNet presupune crearea unei aplicatii de tip server/client care sa ajute la asimilarea cunostintelor referitoare la Retelele de Calculatoare. Acest lucru va putea fi posibil dupa o autentificare sau inregistrare efectuata cu succes. Utilizatorii vor avea posibilitatea de a-si alege capitolul despre care doresc sa afle informatii sau de a cauta o serie de cuvinte cu scopul de a gasi capitolul/capitolele la care se afla informatiile, urmand, eventual, ca acesta sa adauge in mod public un mesaj/o intrebare dupa ce citeste lectia respectiva. Conversatiile vor fi, evident, clasificate in functie de numarul capitolului la care se afla utilizatorul. Orice utilizator va putea vedea seria de mesaje postate de catre alti utilizatori in cadrul unei anumite lectii. In plus, o parte din utilizatori vor putea fi grupati intr-o lista de prieteni.

Am ales acest proiect deoarece cred ca, la nivel inalt, ar putea fi dezvoltate aplicatii similare care sa faciliteze ajutorul intre elevi/studenti/persoane care doresc sa aprofundeze cunostinte din diferite domenii, aplicabilitatea lui fiind relevanta si in contextul actual. Dezvoltata astfel, o aplicatie de tipul LearNet ar putea reprezenta o anexa a unui plan de invatamant, un ajutor spre reconversia profesionala sau o facilitate ingenioasa oferita in cadrul unor cursuri sustinute online(studentii fiind la distanta, internationali etc), beneficiarii avand posibilitatea de a socializa, de a cunoaste persoane cu aceleasi arii de interes sau chiar de a gasi parteneri potriviti pentru realizarea unor proiecte.

2 Tehnologii utilizate

2.1 TCP

Intrucat scopul utilizatorului este acela de a acumula informatii care trebuie sa fie corecte si, in acelasi timp, de a furniza/solicita informatii suplimentare sau de a urmari discutii clasificate pe capitole(actiuni ce necesita ordine si rigoare, fiind vorba despre anumite cereri-comenzi ale clientului care presupun un raspuns

clar), este nevoie de o retea care ar putea supravietui in orice conditii, astfel ca protocolul de comunicatie ales este TCP.

TCP este un protocol orientat-conexiune ce se bazeaza pe calitatea serviciilor si pe siguranta trimiterii tuturor informatiilor intr-o ordine bine stabilita. Conexiunile se identifica prin perechi reprezentate de adresa IP: Port. TCP asigura si retransmiterea in cazul in care se pierd anumite pachete, cu ajutorul numerelor de secventa. (referinta (1))

Am ales TCP deoarece:

-nu ne putem permite ca baza de date sa fie incarcata cu informatii gresite

-ofera incredere, datele nu se vor pierde deoarece sunt implementate mecanisme de control al erorilor,asigura livrarea in ordine a pachetelor

-controleaza congestia(nu se va injecta un nou pachet in retea pana cand un pachet mai vechi nu o paraseste) si fluxul - folosind fereastra glisanta(daca se pierde vreun pachet/daca se pierde confirmarea primirii unui pachet)

Cu toate ca protocolul UDP ar avea ca avantaje : viteza foarte mare de transfer, nu ar fi avantajos sa fie utilizat in acest context deoarece acest protocol nu implementeaza un sistem de confirmare a primirii pachetelor, de prevenire a trimiterii de duplicate, proiectul ales necesitand toate aceste lucruri. Astfel, va fi sacrificata viteza mare de transfer pentru a asigura parcursul sigur al utilizatorului in cadrul aplicatiei, acesta din urma dorind ca atat materia de parcurs, cat si discutiile cu alti utilizatori sa fie corecte. (referinta (1))

2.2 SQLite

Datele despre utilizatori, discutii(inclusiv cele aparute cat timp utilizatorul a fost offline), lectii, relatii de prietenie vor fi stocate de catre server intr-o baza de date relationala. Aceasta va fi interogata de fiecare data cand clientul realizeaza o cerere de acest gen.

SQLite este o mică bibliotecă C care implementează un motor de baze de date SQL încapsulat și nu necesită configurare. O bază de date întreagă este stocată într-un singur fișier. Sursele sunt în domeniul public, SQLite poate fi folosit pentru orice scop.

Voi utiliza biblioteca SQLite pentru a gestiona baza de date, motivele fiind:

-faptul ca este mai usor si intuitiv de lucrat cu SQLite

-faptul ca SQLite ofera o rapiditate sporita comparativ cu alte tipuri de baze de date, (datele fiind stocate intr-un fisier) De exemplu, bazele de date client-server necesita un server

-simplu de a fi incapsulat intr-un program mai mare, asa cum este LearNet

Pentru SQLite, referinta(6)

2.3 Qt

In ceea ce priveste interfata grafica, am ales QT. Qt este un sistem ce cuprinde o biblioteca cu elemente de control(widgets), folosit in special pentru crearea de programe cu interfata grafica. Pentru comunicarea intre obiecte, in cadrul QT sunt folosite "signals and slots". Acest mecanism este de fapt o caracteristica principala a Qt si o alternativa a tehnicii callback(un callback este un pointer la o functie). Pentru a avea raspunsuri dinamice la input-ul dat de user, este nevoie de o comunicare clara intre elementele unei interfete grafice, iar acest lucru este simplificat mult cu paradigma "signals and slots". Am ales QT datorita usurintei cu care se poate realiza o interfata complexa, stilului intuitiv, dar si datorita numarului mare de resurse disponibile (documentatie realizata excelent).
Pentru Qt, referinta (5).

2.4 Threads

Cat despre concurenta conexiunilor client-server, am ales sa folosesc thread-uri.Create mai rapid, impartind memorie cu procesul ce le-a creat, consumand mai putine resurse, thread-urile ar fi cea mai buna optiune in cazul acestui proiect. In mod evident, utilizarea fork-ului nu ar fi adus atat de multe avantaje, cel mai mare atu al thread-urilor fiind eficienta. In cadrul fork-urilor se copiaza toata memoria apartinand procesului parinte, lucru care dureaza si, mai mult, consuma resurse. Referinta (1)

3 Arhitectura aplicatiei

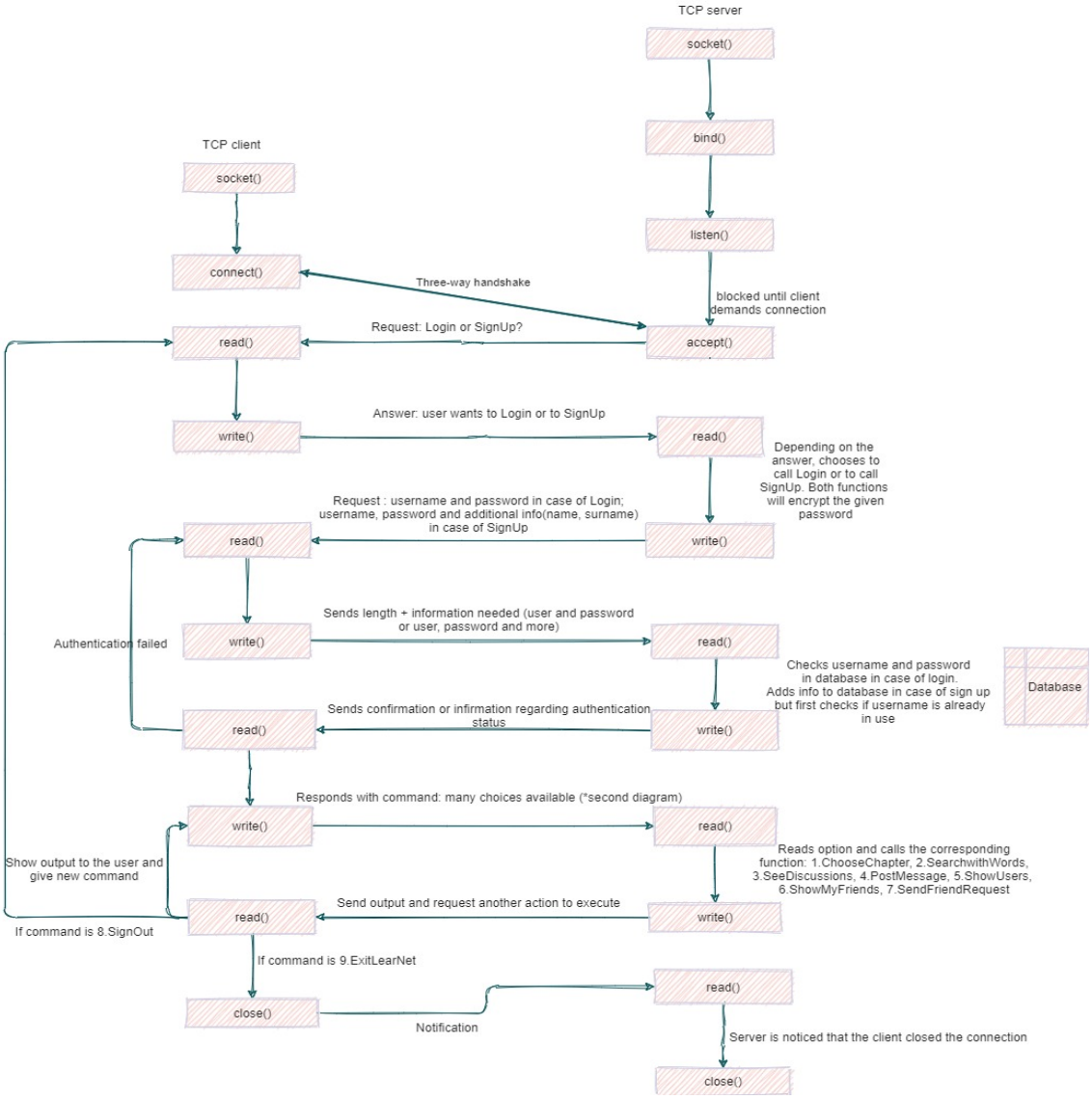

Fig. 1. Diagrama aplicatiei detaliata

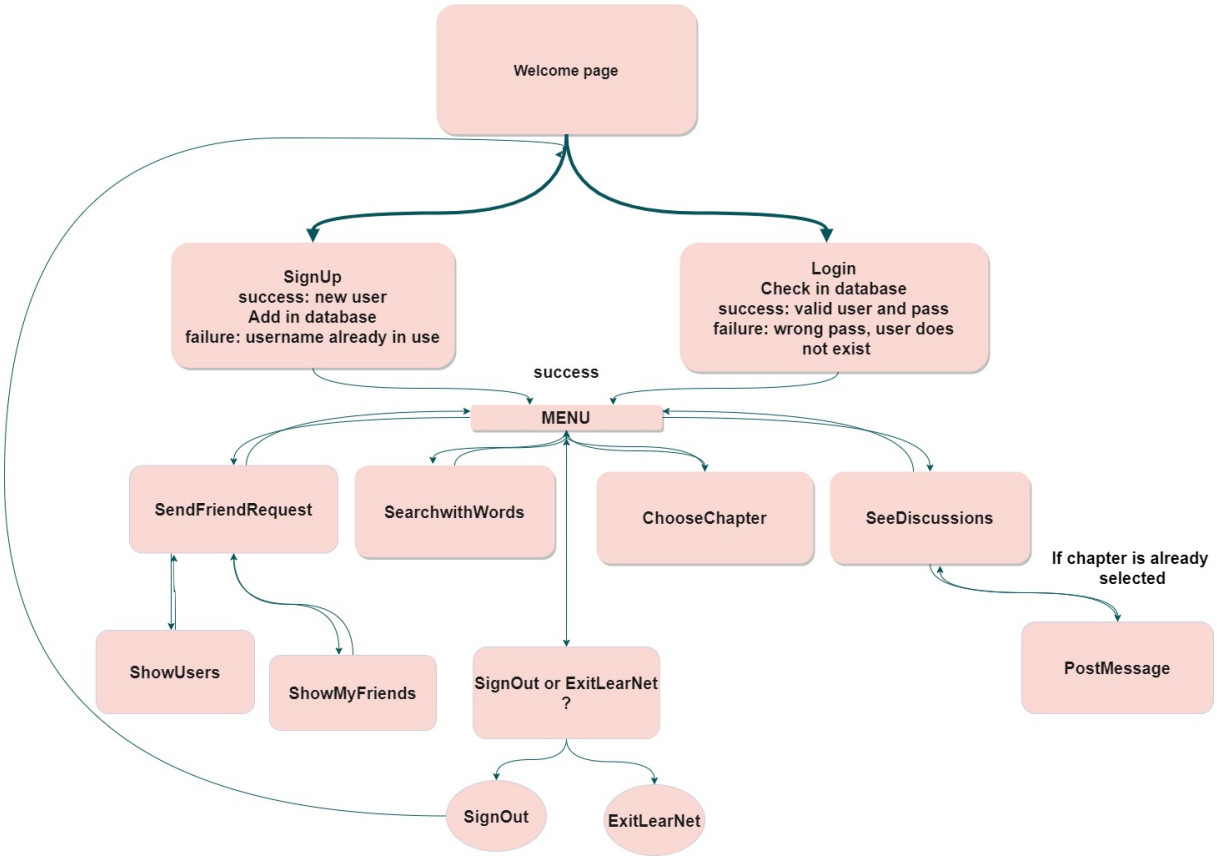

Fig. 2. Diagrama aplicatiei detaliata - functii

3.1 Conceptele implicate

Protocolul cuprinde o serie de comenzi, relatia server-client bazandu-se pe urmatoarele aspecte: clientul va trimite cereri reprezentand numarul optiunii alese sau un text(de exemplu cand doreste sa posteze un mesaj), serverul primeste numarul si astfel il va recunoaste drept una din functiile pe care trebuie sa le apeleze sau primeste textul si il stocheaza in baza de date, urmand sa execute actiuni in acest sens. Pe parcurs, in functie de optiunile alese, clientului i se vor afisa diverse ferestre care arata optiunile cu care mai poate continua din acel punct(si, evident, un buton de exit).

In ceea ce priveste **serverul**, pentru fiecare client care se va conecta, se va crea un thread ce va servi clientul acceptat. Dupa realizarea conexiunii, thread-urile apeleaza pthread detach, incepand sa asculte posibilele mesaje primite de la clientii-utilizatori. Este apelata apoi functia de raspuns care are rolul de a

identifica si de a trata comenzile date de clienti. In primul rand, se doreste a se afla daca clientul vrea sa se logheze sau sa isi faca un cont nou. In functie de raspuns, se apeleaza Login sau SignUp. Daca totul a decurs bine si clientul s-a logat/inregistrat cu succes, se va deschide o noua fereastra - meniul. Este momentul ca serverul sa citeasca de la client optiunea aleasa si, pe langa altele, sa apeleze functia corespunzatoare optiunii. Acest lucru se poate face atat timp cat utilizatorul nu a decis sa inchida aplicatia LearNet. Serverul se va ocupa si cu accesul la baza de date.

Din motive de securitate, **clientul** nu va avea posibilitatea de a accesa baza de date, fiind nevoit sa comunice exclusiv cu serverul, trimitand cereri/raspunsuri. Mentionam ca intotdeauna va exista un set de mesaje de confirmare atat din partea serverului, cat si din partea clientului.

Clientul va implementa o interfata grafica atractiva. Initial se va afisa pagina de start a aplicatiei: WELCOME PAGE. Pentru a merge mai departe, clientul va trebui sa apese pe unicul buton prezent pe pagina, "Start". In continuare, se va deschide o fereastra si clientul va avea posibilitatea de a se inregistra sau de a se loga in contul sau, daca are unul: LOGIN/SIGN UP. In cazul SIGN UP, clientul va trebui sa introduca anumite informatii: nume, prenume, username, parola, informatii ce vor fi adaugate in baza de date, parola fiind criptata cu ajutorul unei functii. Daca clientul introduce un username care exista deja in baza de date, se va afisa un mesaj de eroare corespunzator. Altfel, putem continua.
In cazul LOGIN, daca nu exista conflicte in baza de date si totul decurge normal, clientul se poate loga cu un username si o parola. In cazul in care, dupa criptare, parola nu corespunde cu cea din baza de date, se va afisa un mesaj corespunzator. Daca userul si parola sunt valide, putem continua. In caz de succes, se va deschide o noua fereastra, meniul, cu urmatoarele functionalitati:
-**ChooseChapter**, ce va permite tastarea unui numar, corespunzator capitolului dorit, urmand ca userul sa apese un buton de "Search". In caz de succes, se va deschide o fereastra ce contine textul aferent capitolului.
-**SearchwithWords**, ce va permite introducerea unui cuvant/unor cuvinte, urmand ca userul sa apese un buton de "Search". In caz de succes, se va deschide o fereastra unde vor fi afisate capitolele unde s-a facut match.
-**SendFriendRequest**, unde se va deschide o fereastra in care userul va putea sa-si vada lista de prieteni: **ShowMyFriends** sau sa vada lista tuturor userilor: **ShowUsers**. Userul va putea trimite o cerere de prietenie unui alt user care nu se afla deja in lista lui de prieteni.
-**SeeDiscussions**, ce va permite vizualizarea mesajelor postate de alti utilizatori in legatura cu un capitol anume. Daca a selectat un capitol, clientul va putea trimite mesaje la randul sau: **PostMessage**. Astfel, se face clasificarea mesajelor.
-**SignOut or ExitLearNet**, unde userul va putea alege sa se deconecteze: **Sign Out** sau sa inchida aplicatia: **ExitLearNet**

Baza de date va contine tabele pentru:
-utilizatori: username, password(encrypted), prenume, nume
-prietenii: user1, user2
-discutii: capitol, username, post
-cursuri: capitol, text

In cadrul **Qt**, o tehnica ce va fi utilizata in cadrul proiectului cu siguranta va fi modal approach. Acest approach presupune blocarea oricarei actiuni in cadrul ferestrelor deschise, cu exceptia celei curente. Cand o fereastra de tip modal este deschisa, userul trebuie sa termine mai intai interactiunea cu aceasta fereastra si sa o inchida, abia apoi va putea accesa orice alta fereastra din aplicatie. Un alt concept ce va fi utilizat este cel mentionat anterior, signals and slots. Acest mecanism ne asigura ca, daca vom conecta un signal la un slot, slot-ul va fi apelat cu parametrii signal-ului la momentul potrivit.

4 Detalii de implementare

```
void SearchWithWords(struct thData tdL){


    //citim de la client ce a tastat in bara de search + lungimea
    int length;
    int recv_length = read(tdL.cl, &length, sizeof(int));
    if(recv_length <= 0){
        qDebug()<<"Server: NU am putut citi de la client lungimea textului de cautat la Search with Words\n";
    }
    char* text_to_find = (char*) malloc(50);;
    int recv_text_to_find = read(tdL.cl, text_to_find, length);
    if(recv_text_to_find <= 0){
        qDebug()<<"Server: NU am putut citi de la client textul de cautat la Search with Words\n";
    }
    text_to_find[length] = '\0';

printf("%s\n",text_to_find);
    int count = 0; //numarul de capitole in care apare textul ce s-a cautat
int numar_aparitii;

    int ch[11], vect[12], x = 0;

    QSqlQuery interogare;
    interogare.prepare("SELECT(LENGTH(text) - LENGTH(REPLACE(text, ?, ''))) / LENGTH(?) as aparitii from cursuri");
    interogare.bindValue(0, text_to_find);
    interogare.bindValue(1, text_to_find);
    interogare.exec();
    while(interogare.next()){
        numar_aparitii = interogare.value(0).toInt();
        ch[x] = numar_aparitii;
        printf("%d\n", numar_aparitii);
        if(numar_aparitii > 0)
        {
            vect[count] = x;
            count++;

        }
         x++;
    }
```

```
int send_nr_cap = write(tdL.cl, &count, sizeof(int));
if(send_nr_cap <= 0){
    qDebug()<<"Server: NU am putut trimite la client numarul de capitole in care apare textul cautat la Search with Words\n";
}

     qDebug()<<"Server: Am putut trimite la client numarul de capitole in care apare textul cautat la Search with Words\n";


    for(int j= 0; j < count; j++) {

        int cap = vect[j] + 1;
        int send_capitol = write(tdL.cl, &cap, sizeof(int));
        if(send_capitol <= 0){
            qDebug()<<"Server: NU am putut trimite la client numarul capitolului la Search with Words\n";
        }
        printf("Server loop capitol %d\n", cap);

        qDebug()<<"Server: Am putut trimite la client numarul capitolului la Search with Words\n";
        int send_nr_ap = write(tdL.cl, &ch[vect[j]], sizeof(int));
        if(send_nr_ap <= 0){
            qDebug()<<"Server: NU am putut trimite la client numarul aparitiilor pentru capitol la Search with Words\n";
        }

        printf("Server loop numar aparitii %d\n", ch[vect[j]]);
        qDebug()<<"Server: Am putut trimite la client numarul aparitiilor pentru capitol la Search with Words\n";

    }


}
```

```
int send_friend_request(struct thData tdL){
    QSqlQuery interogare, interogare2;
    int answer = 0;
    char* username = (char*) malloc(40);
    int lg_username;
    int user_lg;
    char* username_gen = (char*) malloc(40);

    int send_user_lg = read(tdL.cl, &user_lg, sizeof(int));
    if(send_user_lg <= 0){
         qDebug()<<"Server: nu am primit lungimea username ului \n";
     }

        int send_user = read(tdL.cl, username_gen, user_lg);
        if(send_user <= 0){
            qDebug()<<"Server: nu am primit username ul\n";
        }
        username_gen[user_lg] = '\0';
        int read_lg = read(tdL.cl, &lg_username, sizeof (int));
                if(read_lg <= 0){
                    printf("Thread %d\n",tdL.idThread);
                    perror("Server: eroare, nu am putut citi de la client lungimea username-ului pentru friend_request\n");

                }
                fflush(stdout);
                int read_user = read(tdL.cl, username, lg_username);
                if(read_user <= 0){
                    printf("Thread %d\n",tdL.idThread);
                    perror("Server: eroare, nu am putut citi de la client username-ul pentru friend_request\n");

                }
                username[lg_username] = '\0';
                fflush(stdout);
                if(!strcmp(username, username_gen)){
                    answer = -1; //cazul in care clientul si-a scris propriul username
                    int send_answer = write(tdL.cl, &answer, sizeof (int));
                    if(send_answer <= 0){
                        printf("Thread  %d\n",tdL.idThread);
                        perror("Server: eroare, nu am putut trimite la client raspunsul: -1 - same username as yours pentru friend_re

                    }
                    return answer;
```

```
}
else //username-ul tastat de client este diferit de al lui
{
    interogare.prepare("select username from utilizatori where username = ?");
    interogare.bindValue(0,username);
    interogare.exec();
    if(interogare.first()){ //cazul in care am gasit username-ul tastat de client in tabela generala cu utilizatori
         QSqlQuery check_duplicate;
         check_duplicate.prepare("select * from prietenii where user1 = ? and user2 = ?");
         check_duplicate.bindValue(0, username_gen);
         check_duplicate.bindValue(1, username);
         check_duplicate.exec();
         if(check_duplicate.first()){ //cazul in care am gasit deja cererea in tabel
             answer = -5;
             interogare2.prepare("select * from prietenii where user1 = ? and user2 = ?"); //verif daca exista si invers
             interogare2.bindValue(0, username);
             interogare2.bindValue(1, username_gen);
             interogare2.exec();
             if(interogare2.first()){
                 answer = -10;
             }
             int send_answer = write(tdL.cl, &answer, sizeof (int));
             if(send_answer <= 0){
                 printf("Thread  %d\n",tdL.idThread);
                 perror("Server: eroare, nu am putut trimite la client raspunsul: -5/-10 - you have already sent a friend request to th

             }
             return answer;

         }
         else //nu am gasit cererea in tabel, dar prietenul cerut exista
         {
             answer = 1;
             QSqlQuery insert;
             insert.prepare("insert into prietenii values (?, ?)");
             insert.bindValue(0, username_gen);
             insert.bindValue(1, username);
             insert.exec();
             int send_answer = write(tdL.cl, &answer, sizeof (int));
             if(send_answer <= 0){
                 printf("Thread %d\n",tdL.idThread);
                 perror("Server: eroare, nu am putut trimite la client raspunsul: am inserat in tabela de prietenii\n");
```

```
                    }
                    return answer;

                }
            }
            else //user inexistent
            {
                answer = 0;
                //nu am gasit textul tastat de user in lista de usernames
                int send_err = write(tdL.cl, &answer, sizeof(int));
                if(send_err <= 0){
                   perror("Server: eroare, nu am putut trimite la client raspunsul 0: wrong username\n");
                }

                return answer;

            }
        }
    return answer;

}
```

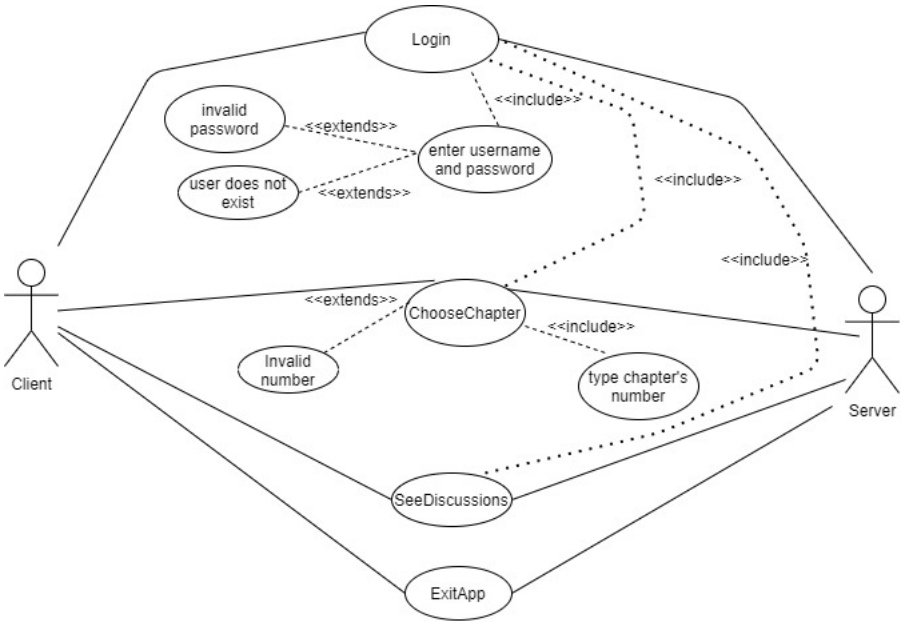

Fig. 3. Sunny day use case 1

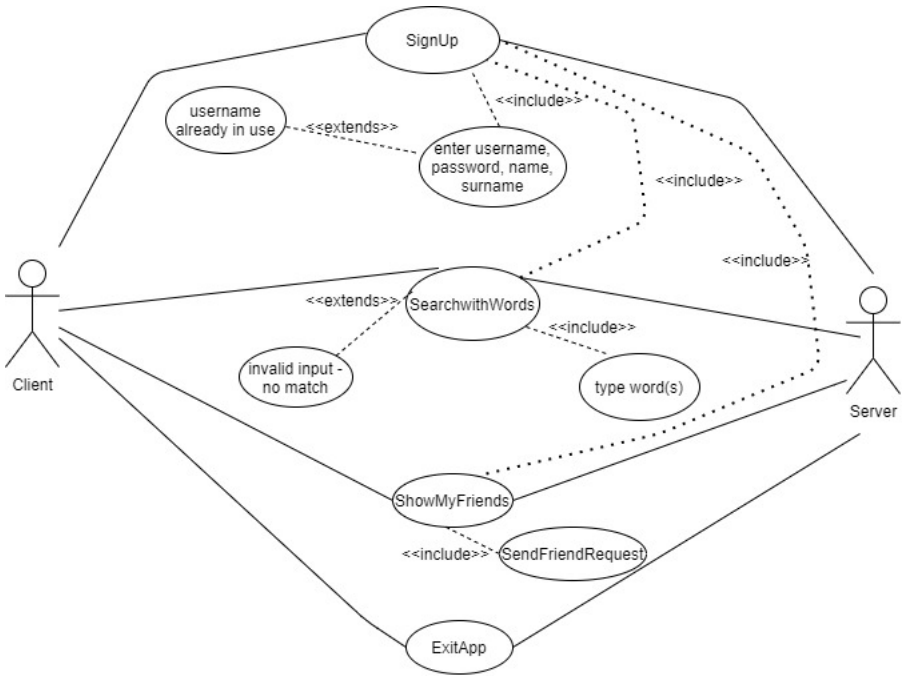

Fig. 4. Sunny day use case 2

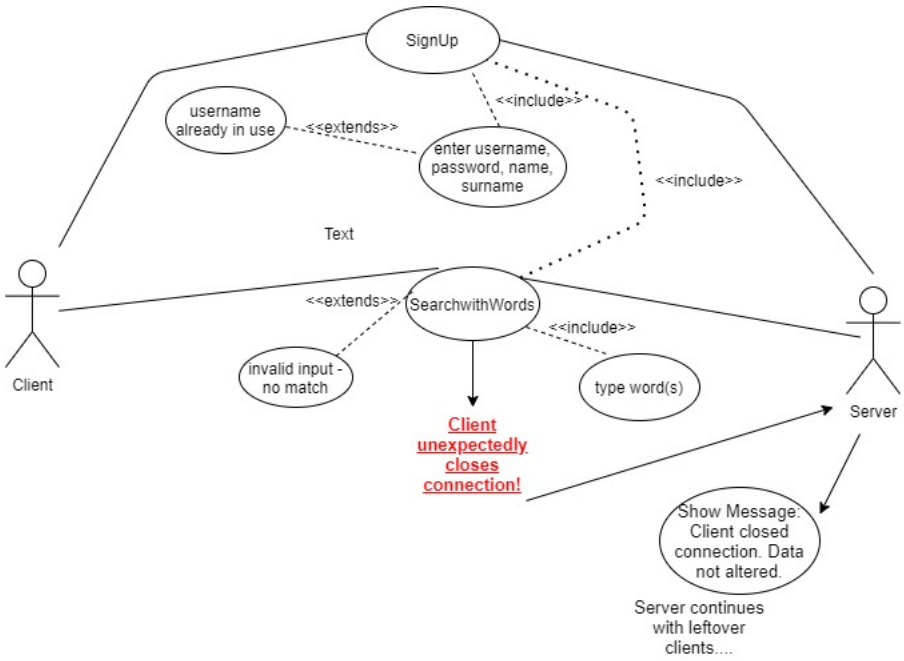

Fig. 5. Rainy day use case 1

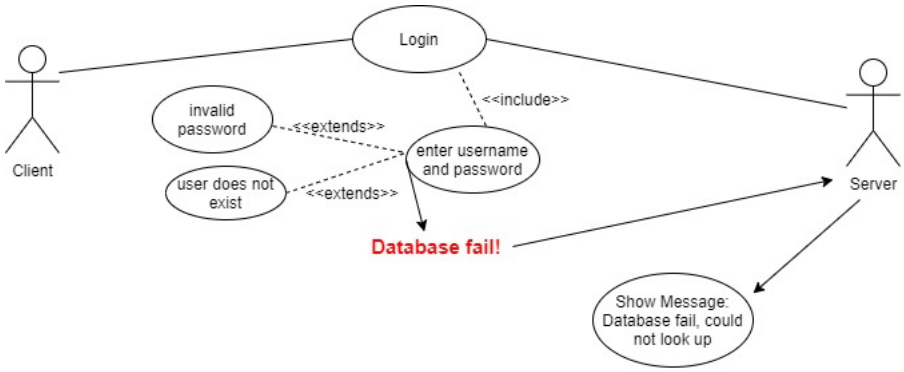

Fig. 6. Rainy day use case 2

5 Concluzii

Proiectul ar putea fi imbunatatit daca s-ar adauga urmatoarele functionalitati:
-functie de resetare a parolei, in caz ca userul doreste sa o schimbe
-adaugarea unor teste de verificare a cunostintelor in cadrul fiecarui capitol (de exemplu grila, userul putand vedea ulterior ce raspunsuri erau bune), adaugarea unor nivele pentru fiecare user, se va face update la nivel in functie de numarul de raspunsuri corecte date, fiecare utilizator va avea un profil unde va fi afisat acest nivel
-posibilitatea de a trimite un mesaj privat unui utilizator (doar daca este prieten cu el, de exemplu)
-posibilitatea unui user de a-si sterge un mesaj postat public

References

(1)https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
(2)https://doc.qt.io/qt-5/sql-connecting.html
(3)https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
(4)https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c
(5)https://doc.qt.io/
(6)https://sqlite.org/docs.html
(7)https://stackoverflow.com/questions/2523765/qstring-to-char-conversion
(8)https://doc.qt.io/qt-5/qsqlquery.html
(9)https://www.youtube.com/watch?v=EkjaiDsiM-Q&list=PLS1QulWo1RIZiBcTr5urECberTITj7gjA
(10)https://stackoverflow.com/questions/53172385/qsqlquery-prepare-and-bindvalue-not-working