# Retele de calculatoare - Proiect

Gafita Andrei A4

January 17, 2019

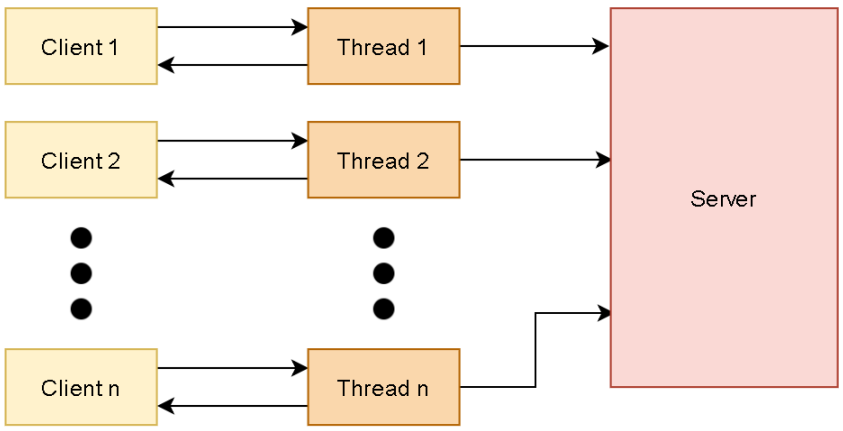
## 1 Introducere

Proiectul ales este LEARNET, acesta consta in creerea unei aplicatii client/server ce ajuta clientii ce se conecteaza sa invete principalele notiuni de Retele de Calculatoare si sa vorbeasca intre ei pe diferite teme. In acest raport voi vorbi despre tehnologiile utilizate in realizarea acestui proiect, voi prezenta si detalia arhitectura aplicatiei folosind diagrama aplicatiei si use case diagram, voi prezenta portiuni relevante din cod ce au fost deja implementate si voi concluziona prin a spune ce ar putea fi imbunatatit.

## 2 Tehnologii utilizate

### 2.1 TCPIP concurent cu thread-uri

In cadrul acestui proiect am folosit un server de tip TCP concurent realizat prin thread-uri (pthreads), adica un thread al serverului pentru fiecare client nou.



Am ales acest mod de implementare deoarece este cel mai simplu de implementat la ora actuala pentru mine si ofera posibilitatea ca mai multi clienti sa foloseasca serverul odata. Este strict necesar sa existe aceasta posibilitate pentru o experienta placuta a celor ce folosesc aplicatia, in plus unele functionalitati nu s-ar putea realiza daca serverul nu ar fi concurent, cum ar fi chat-ul live intre 2 sau mai multe persoane.

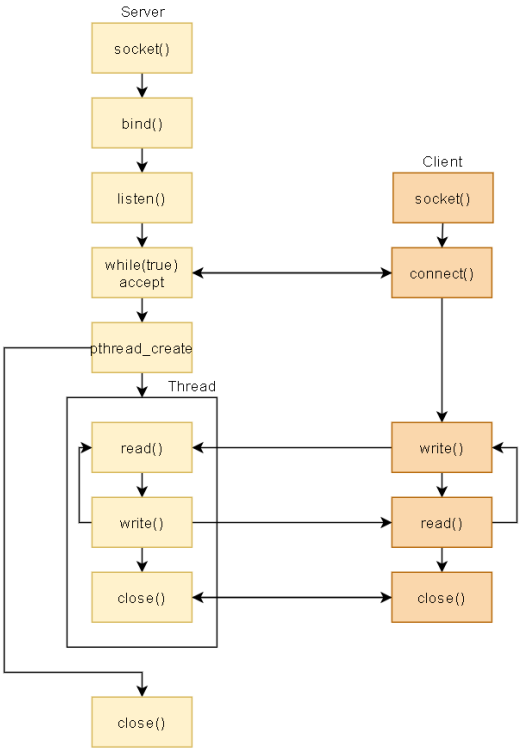
### 2.2 SQLite3

Pentru a putea folosi bazele de date in construirea aplicatiei am folosit programul public SQLite3, acesta a ajutat la proiectarea bazei de date si accesarea acesteia in cod.

### 2.3 Qt

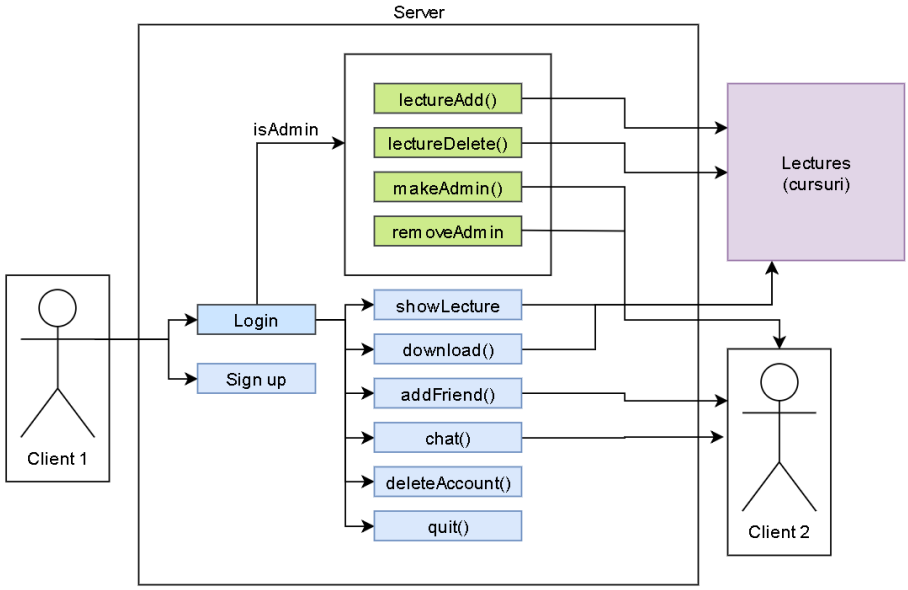
Interfata grafica nu ar fi putut fi realizata fara ajutorul IDE-ului Qt creator si toate functionalitatile lui.

## 3 Arhitectura aplicatiei

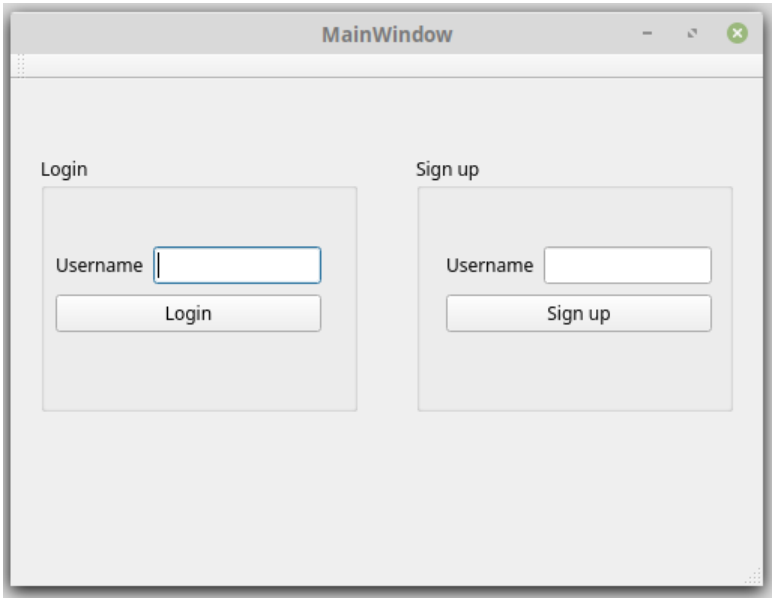



De la comanda listen() serverul incepe sa 'asculte' si intra intr-o bucla infinita

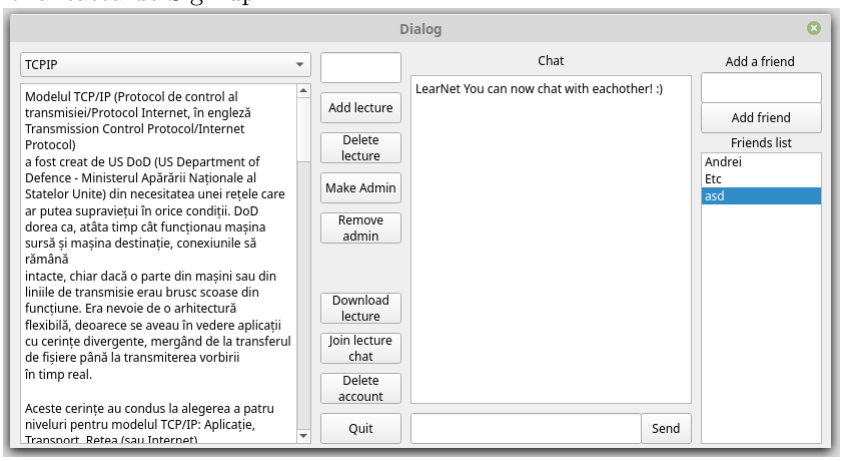
in care accepta conexiuni de la clienti. Dupa ce a fost realizata o conexiune intre server si un client, serverul executa comanda pthread_create si creeaza un thread. De aici thread-ul principal de server executa in continuare si asteapta alt client pe cand thread-ul asociat clientului are grija de acesta in calitate de server. Dupa terminarea conexiunii cu clientul, sunt eliberate resursele folosite de catre thread. In bucla read()-write() din cadrul fiului, in proiectul acesta(si in codul scris de mine) serverul citeste o comanda de la client si o executa(acum o comanda este o apasare de buton sau un mesaj cuiva), transmitand inapoi clientului datele cerute prin write, dupa care bucla se reia. Datorita interfetei grafice realizate comenzi invalide nu mai exista, astfel orice actiune realizata de catre client este tratabila de catre server.



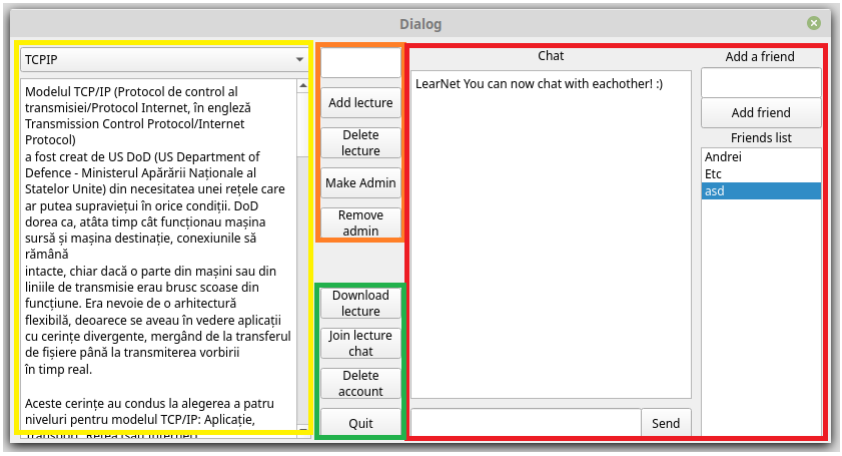

In diagrama aceasta sunt ilustrate functionalitatile actuale ale serverului si modul in care interactioneaza cu clientii. Clientul este obligat sa se autentifice pentru a folosi oricare din functionalitatile serverului sau se poate inregistra in baza de date a utilizatorilor folosind functionalitatea de sign-up. Daca user-ul este admin, acesta are privilegiul de a adauga noi cursuri in baza de date a cursurilor prin lectureAdd() si sa dea altor utilizatori privilegii de admin, altfel daca este un user normal, acesta poate doar viziona cursurile respective cu functia 'showLecture()' sau sa interactioneze cu ceilalti clienti prin chat. Prin comanda addFriend() user-ul poate sa se imprieteneasca cu un alt user existent. Prin quit() clientul termina conexiunea cu serverul.




Primele functionalitati arata astfel. Putem sa ne autentificam inserand contul in sectiunea de login sau daca nu avem unul, ne putem creea un cont folosind functionalitatea de Sign up.




Dupa ce clientul s-a autentificat acesta intra in ecranul principal. Aici se pot folosi toate functionalitatile aplicatiei, interfata poate fi vazuta mai sus cu toate functionalitatile(si cele pentru admin).

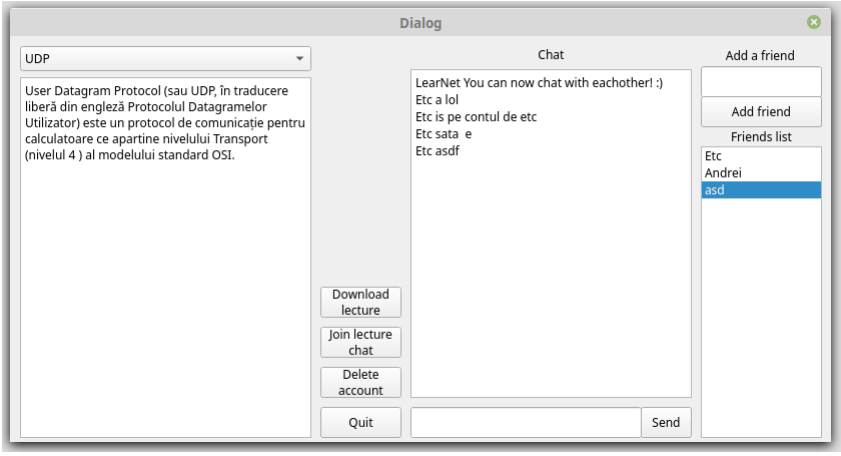



In poza de mai sus am impartit sectiunile relevante in culori pentru a le putea explica mai usor. Cu galben este marcata partea din fereastra unde clientul poate citi cursul sau sa il schimbe cu meniul unde scrie titlul cursului (TCPIP).

Cu portocaliu sunt marcate functionalitatile unui admin, un user normal nu va vedea aceste butoane si nu va avea acces la ele. Casuta de scris text serveste pentru butoanele de dedesubtul ei, in aceasta se va insera numele cursului sau al utilizatorului la care se refera adminul cand apasa butonul pentru a realiza actiunea specifica.

Cu rosu avem marcat chat-ul si lista de prieteni. Aici selectand din lista de prieteni un prieten(dand dublu click stanga pe el) incepem un chat cu respectivul. Chatul este si offline si real-time, la fel si lista de prieteni.

Cu verde sunt marcate functionalitati generale, avand cursul TCPIP selectat, daca utilizatorul apasa butonul 'Join lecture chat' acesta va intra intr-un chat pentru cursul respectiv. Utilizatorul poate si descarca o copie a cursului sub format text cu butonul 'Download lecture', acesta este descarcat in locatia in care este clientul pe disc-ul sau. Butonul 'Delete account' sterge contul si orice urma din bazele de date despre acesta, cu exceptia mesajelor scrise in chat-ul unui curs iar 'Quit' implicit inchide aplicatia.




Asa arata interfata pentru un utilizator normal (fara privilegii de admin). Pentru a proteja aplicatia de un eveniment in care sa nu mai existe admini ( adminii au apasat toti pe butonul de 'Delete account' sau un admin a dat 'Remove admin' la toti adminii cat si lui insusi) exista un cont permanent numit 'admin' care nu poate fi sters (butonul de 'Delete account' nu apare pentru contul respectiv) si nu poate sa aiba privilegiile de admin retrase cu comanda 'Remove admin'.

# 4 Detalii de implementare

```
andrei@andrei-VirtualBox:~/Desktop/Project$ sqlite3 LearNet.db
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite> .tables
AndreiAndrei  TCPIP         adminadmin    asdasd        test
EtcAndrei     UDP           asdAndrei     friendslist   users
EtcEtc        adminEtc      asdEtc        lectures
sqlite> .schema users
CREATE TABLE users ( userid INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, username TEXT NOT NULL, isadmin INTEGER NOT NULL, isonline
INTEGER NOT NULL, terminate_chat INTEGER NOT NULL);
sqlite> .schema friendslist
CREATE TABLE friendslist ( id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, username TEXT NOT NULL, friend TEXT NOT NULL );
sqlite> .schema lectures
CREATE TABLE lectures ( lectureid INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, lecturename TEXT NOT NULL, content TEXT NOT NULL);
sqlite> .schema adminadmin
CREATE TABLE adminadmin ( msgno INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, sender TEXT NOT NULL, msg TEXT NOT NULL);
sqlite>
```

Aici se poate observa baza de date aferenta proiectului. In tabela users sunt stocati userii, starea lor (daca sunt online sau nu), daca sunt admini si un indicator pentru chat.

In tabela friendslist sunt stocati pe fiecare linie relatia de prietenie de la username la friend. Daca 2 clienti se imprietenesc atunci vor aparea doua linii noi in tabel.

Tabela lectures contine cursurile si continuturile acestora.

O tabela de forma 'numenume' este un chat history intre cei doi clienti cu respectivele nume. In codul server exista o tabela unica pentru fiecare relatie de prietenie iar aceasta este codificata de catre server astfel incat sa nu poata fi confundata. Acest tip de tabela este si tipul chatului de la cursuri.

```
bzero (msg, 10000);
    printf ("[server]Asteptam mesajul...\n");
    fflush (stdout);

    int login = 0, admin = 0;
    char x;
    char* user = (char*) calloc(10000,1);

    bzero(msg,10000);
    read(tdL.client, msg, 10000);
    printf("****%s****\n",msg);
    bzero(Response,10000);
    strcat(Response,"Error! User does not exist!\n");
    strcat(Response,msg);

    if(strcmp(msg,"SIGNUP")==0){printf("EXECUTED SIGNUP\n");signUp(Response,msg,tdL.client);}
      else loginClient(Response,msg,user,tdL.client,&login,&admin);
    if(login == 0){
                    sqlite3_close(DB);
                    write (tdL.client, Response, 10000);close(tdL.client);pthread_detach(pthread_self());
                    close ((intptr_t)arg);
                    return(NULL);}
```

Aici se poate observa functionalitatea din ecranul de login. Daca clientul alege sa se inregistreze atunci e trimis la server mesajul 'SIGNUP' care apeleaza functia respectiva si il introduce pe client in baza de date de useri dupa care rupe conexiunea, chiar si asa ecranul de pe partea clientului nu este inchis, acesta cand apeleaza functia de login el se reconecteaza la server.

```
if(strcmp(subString(query,1),subString(msg,1))==0){
  memset(Response,'\0',10000);
  strcpy(Response,"Logged in as user: ");
  strcat(Response,subString(msg,1));
  strcat(Response,"\n");
  strcat(user,subString(msg,1));

  printf("[server]Trimitem mesajul inapoi...%s\n",Response);
  *login = 1;

  if(strcmp(subString(query,2),"1")==0){*admin = 1;
      memset(Response,'\0',10000);
      strcpy(Response,"Logged in as admin: ");
      strcat(Response,subString(msg,1));
      strcat(Response,"\n");
  }
  if(strcmp(user,"admin")==0){
      memset(Response,'\0',10000);
      strcpy(Response,"Logged in as superadmin: ");
      strcat(Response,subString(msg,1));
      strcat(Response,"\n");
  }
  write (client, Response, 10000);

  bzero(sql,10000);
  strcat(sql,"UPDATE users SET isonline = 1 WHERE username = '");
  strcat(sql,subString(msg,1));
  strcat(sql,"';");

  sqlite3_exec(DB, sql, callback, query, &messageError);

  sqlite3_close(DB);
```

In portiunea aceasta de cod se poate observa cum functioneaza functia de login daca utilizatorul exista. In functie de gradul pe care il are clientul (admin sau nu) serverul trimite inapoi un mesaj corespunzator de la care clientul interpreteaza si deschide interfata corespunzatoare. Serverul isi seteaza si el flagurile pentru a sti daca userul este admin, deasemenea numele clientului este marcat ca fiind online in baza de date.

```
char* subString(char* sourceString,int nrCuv){
  char* subString = (char*) calloc(100,1);
  int position = 0, positionSubString=0;

  for(int cuvant = 1; cuvant <= nrCuv;cuvant++){
    while(sourceString[position]!=' ' && sourceString[position]!='\n' && sourceString[position]!='\0')
      subString[position-positionSubString]=sourceString[position++];
    if(cuvant!=nrCuv){
      position++;
      positionSubString = position+1;
    }
}
  subString[position-positionSubString+1]='\0';
  if(nrCuv==1)
    for(int i = 0;i<100;i++)subString[i]=subString[i+1];

  return subString;
}
```

Am ales sa introduc aceasta functie in raport deoarece este vitala pentru codul proiectului meu. Este folosita pentru a sorta comenzile introduse de catre client pe cuvinte. Odata cu interfata grafica utilitatea nu mai este asa de importanta insa a devenit o portiune importanta din proiect care nu strica nimic si totusi ajuta astfel am ales ca aceasta sa ramana.

```
int testIfUserExists(char* name){
    sqlite3* DB;
  char* query = (char*) calloc(10000,1);
  char* sql = (char*) calloc(10000,1);

  strcat(sql,"SELECT DISTINCT username FROM users WHERE username = '");
  strcat(sql,subString(name,1));
  strcat(sql,"';");

  sqlite3_open("LearNet.db", &DB);
  char* messageError;
  sqlite3_exec(DB, sql, callback, query, &messageError);
  sqlite3_close(DB);
  if(strcmp(subString(query,1),subString(name,1))==0)return 1;

  return 0;
}
```

In aceasta functie este testat daca userul exista. Este accesata baza de date si se returneaza 1 daca acesta exista altfel 0.

```
char* friendDBname(char* user,char* msg){
  char* existsFriend = calloc(10000,1);

  if(strcmp(subString(user,1),subString(msg,2))>0){strcat(existsFriend,subString(user,1));strcat(existsFriend,subString(msg,2));}
    else {strcat(existsFriend,subString(msg,2));strcat(existsFriend,subString(user,1));}

  return existsFriend;
}
```

Aceasta functie este responsabila de codificarea numelui tabelei dintre doi prieteni. Aceasta codificare este destul de simpla, bazandu-se pe un strcmp intre cele doua nume si stabilirea numelui dupa.

```
void TestWindow::on_DeleteAccountButton_clicked(){
    char* deleteAccountMessage = new char[10000];bzero(deleteAccountMessage,10000);
    strcat(deleteAccountMessage,"DELETE_ACCOUNT");

    QMessageBox::StandardButton reply = QMessageBox::question(this,"Account deletion","Are you sure?", QMessageBox::Yes | QMessageBox::No);

    if(reply == QMessageBox::Yes){
        write(sd,deleteAccountMessage,10000);
        QCoreApplication::quit();
    }
    else return;
}
```

Prezint si cum arata codul pentru un buton din client. Acest cod este pentru butonul 'Delete account' si se poate vedea cum functioneaza prompt-ul ce apare dupa ce a fost accesat butonul. Apare un 'message box' care intreaba clientul daca e sigur de actiunea aleasa, daca acesta apasa 'yes' atunci se va continua actiunea, stergand contul, altfel actiunea va fi anulata.

# 5 Concluzii

Proiectul in stadiul actual este functional din punctul meu de vedere. Desigur ar putea fi si imbunatatit, spre exemplu sa se adauge feature-ul de cerere de prietenie, in stadiul actual daca doi clienti se adauga la prieteni acestia devin prieteni pur si simplu. Sau un buton de 'Unfriend' pentru a rupe prietenia cu un prieten actual. Ceea ce ar mai putea fi adaugat ar fi actualizarea in real time a cursurilor, daca un admin adauga un curs nou, lista de cursuri nu va fi actualizata pana cand utilizatorul nu se va reloga.

Cu toate acestea, aplicatia este complet functionala iar feature-urile mentionate mai sus nu ar modifica cu extrem de mult experienta.

# 6 Bibliografie


https://profs.info.uaic.ro/ computernetworks/
https://sites.google.com/view/fii-rc
https://www.draw.io/
https://stackoverflow.com/
https://linux.die.net/man/
https://www.youtube.com/
https://images.google.com/
http://doc.qt.io/
https://www.tutorialspoint.com/sqlite