# VirtualSoc

Matcovici Ştefan

Facultatea de Informatică, Universitatea Alexandru Ioan Cuza, Iaşi
stefan.matcovici@info.uaic.ro

**Abstract.** Proiectul constă în realizarea unei aplicaţii client-server care simulează funcţionalitatea unei reţele sociale. Pentru a facilita transmisia datelor între server şi client va fi dezvoltat şi un protocol de comunicare ce va permite trimiterea şi răspunsul la cereri în mod corespunzător.

**Keywords:** reţea socială, protocol de comunicare, client-server, TCP

## 1 Introducere

Reţelele sociale au cunoscut în ultimii ani (în special odată cu apariţia Facebook-ului) o creştere în popularitate, ajungând în anul 2015 la peste 1,4 miliarde de utilizatori la nivel global care au cont măcar un cont [12]. Această lucrare prezintă dezvoltarea unei aplicaţii server-client care permite clientului obişnuit posibilitatea de înregistrare, de a posta şi a se descrie prin cadrul profilului în mod public sau privat, de a avea anumite categorii de prieteni şi de a putea comunica în mod privat cu 2 sau mai multe persoane. Aplicaţia va suporta şi useri administratori care vor putea adăuga sau şterge useri şi posturi. Comunicaţia dintre server şi client se va realiza prin schimbul de mesaje minimaliste. Datele vor fi stocate de server într-o bază de date pe care o va interoga ori de câte ori clientul va trimite cereri.

## 2 Tehnologii utilizate

### 2.1 TCP

TCP (Transmission Control Protocol) este un protocol de transport orientat conexiune, oferă calitate maximă serviciilor, fără pierdere de informaţii, integrând mecanisme de stabilire şi eliberare a conexiunii. Aceste conexiuni, utilizate ca abstracţiuni fundamentale, se identifică în mod unic prin perechi reprezentate de adresa ip:port. Adresa IP este o etichetă asignată unei entităţi într-o reţea de calculatoare în timp ce portul identifică procesele ce rulează pe un sistem de calcul.

Serverul şi clientul iau parte deopotrivă la realizarea conexiunii. Serverul oferă o deschidere pasivă, aşteptând apariţia unei cereri din partea clientului care realizează o deschidere activă. Iniţializarea conexiunii are loc prin parcurgerea unor paşi specifici, metodă cunoscută sub numele de "three-way handshaking" [1].

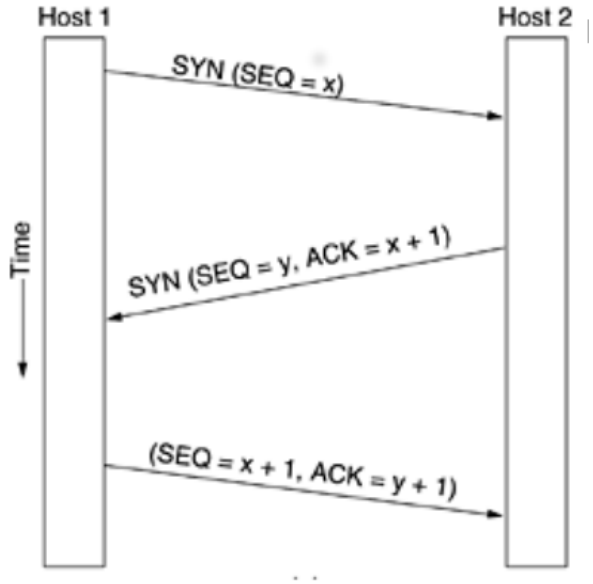


Fig. 1: three-way handshaking

Am ales acest protocol deoarece în comparaţie cu UDP oferă siguranţă în transmisia datelor, prin implementarea mecanismelor de control al erorilor. Acurateţea informaţiilor şi calitatea transmisiei datelor sunt vitale pentru aplicaţie de acest tip deoarece nu se poate permite stocarea de informaţii eronate şi inconsistente în baza de date. Deşi aplicaţia rulează cu o viteză nu la fel de mare ca în cazul UDP, clienţii reţelei ar fi mai degrabă deranjaţi, spre exemplu, de o postare scrisă incorect, decât de o oarecare întârziere în înregistrarea postării.

### 2.2 Posix threads

Un thread (fir de execuţie) este folosit pentru a eficientiza execuţia programelor. Spre deosebire de procese, thread-urile partajează spațiul de adrese al procesului de care aparțin, folosirea lor are o serie de avantaje:

- crearea/distrugerea unui fir de execuție durează mai puțin decât crearea/distrugerea unui proces
- durata context-switch-ului între firele de execuție aceluiași proces este foarte mică, întrucât nu e necesar să se "comute" și spațiul de adrese
- comunicarea între firele de execuție are un overhead mai mic (realizată prin modificarea unor zone de memorie din spațiul comun de adrese)

În cazul aplicaţiei VirtualSoc, se folosesc thread-uri pentru a putea deservi clienţii în mod concurent şi rapid.

### 2.3 JSON

JSON (JavaScript Object Notation - Notația Obiect JavaScript) este un format text de reprezentare și interschimb de date. JSON este autodescriptiv, uşor de înţeles şi inteligibil pentru oameni. Deşi foloseşte sintaxa JavaScript pentru a descrie obiecte de date, formatul este independent de limbaj şi platformă.[13]

Pentru a manipula obiectele de tip JSON aplicaţia utilizează biblioteca Jansson deoarece oferă o interfaţă de lucru intuitivă, o documentaţie bine pusă la punct şi nu are dependenţe în alte biblioteci. Am ales formatul JSON pentru a transmite datele deoarece o utilitate esenţială a acestuia este faptul că datele pot fi transmise în format text (într-un singur şir) la alte entităţi unde pot fi uşor decodificate.

### 2.4 SQLite3

Pentru a stoca datele referitoare la useri, posturi, grupuri, mesaje, serverul apelează la o bază de date relaţională prin intermediul bibliotecii SQLite3. Ca şi avantaje ale acestei biblioteci se pot menţiona următoarele:

- nu este necesar un server ca în cazul Oracle sau MicrosoftServer, toate datele sunt stocate doar într-un fişier pe disc de aceea SQLite este mai rapidă decât bazele de date bazate pe arhitectura client-server
- suportă baze de date cu dimensiuni de ordinul teraocteţilor (pâna la 2 teraocteţi)
- API simplu, intuitiv, uşor de folosit
- sursele sunt domeniu public, putând fi folosite de orice

### 2.5 QT

Interfaţa grafică a clientului este realizată în QT 5.7.0, în mediul de dezvoltare pus la dispoziţie de această bibliotecă gratuită. Pe lângă C++, QT utilizează

paradigma "signals and slots" [4] pentru comunicarea între obiecte ceea ce simplifică modul de a manipula evenimentele. Elementele unei interfeţe grafice trebuie să comunice între ele pentru a răspunde efectiv şi dinamic la input-ul oferit de utilizator. Până la QT acest lucru era realizat prin intermediul callback-ului care este de fapt un pointer către o funcţie. QT implementează o metodă de generare de cod denumită MOC, care generează un alt fişier .cpp, oferind o introspecţie a obiectelor în timpul rulării. Altfel spus, se poate şti la runtime metodele şi alte informaţii despre o instanţă a unei clase.

# 3 Arhitectura aplicaţiei

## 3.1 Diagrama aplicaţiei

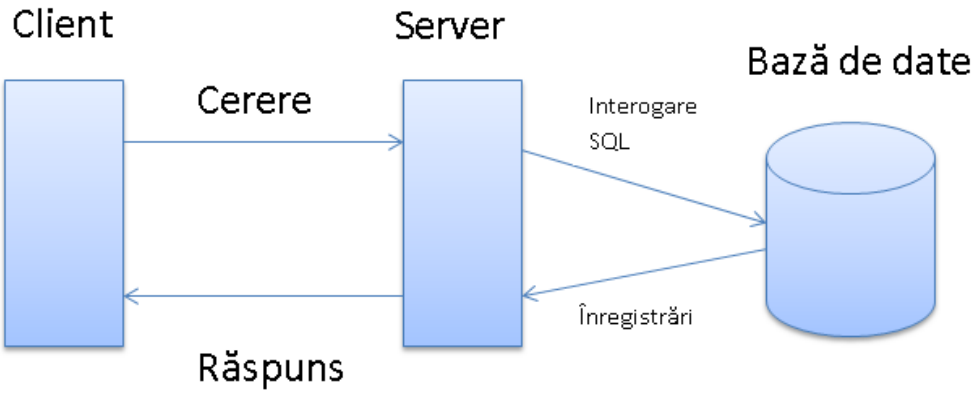


Fig. 2: schema generală

## 3.2 Concepte implicate

Serverul este **multi-threaded** cu câte un thread pentru fiecare client. Threadul principal este blocat la apelul accept() şi de fiecare dată când este acceptat un client, se va crea un thread care va servi clientul acceptat. Această metodă

este mai eficientă decât cea mai eficientă versiune de server preforked dar nu depăşeşte nicio implementare de server prethreaded. După realizarea conexiunii, thread-urile apelează pthread_detach şi încep să asculte mesaje venite de la clienţii aferenţi. Am asigurat prin intermediul funcţiilor de citire şi scriere folosind socket-uri că imediat ce clientul îşi închide capătul său de comunicare serverul iese din bucla de ascultare şi închide thread-ul corespunzător. Astfel nu există riscul ca serverul să aibă deschis vreun thread care să ruleze la infinit, utilizând inutil resurse, fără să deservească vreun client activ.

Totodată, serverul este responsabil şi cu **accesul la baza de date** stocate local. Clientul nu va avea posibilitatea de a trimite efectiv interogări SQL la baza de date ci va delega serverul din raţiuni de securitate si acces controlat la informaţii. Serverul îşi stochează local pentru fiecare thread creat detalii despre user: id-ul corespunzător din baza de date şi username-ul dacă userul s-a logat deja într-un map partajat de toate threadurile. Accesul la resursa partajată este restricţionat de un mecanism de blocare, mutex lock, astfel încât doar un singur thread din server să poată modifica sau verifica datele despre utilizatorii deja logaţi. Comenzile SQL executate folosesc informaţiile din map, aşadar accesul la orice date confidenţiale din baza de date sunt restricţionate de logare chiar şi în cazul unei alte implementări de client pentru acest server.

Clientul va implementa o interfaţă grafică atractivă şi uşor de utilizat. Un utilizator are posibilitatea de a alege serverul la care se conectează (prin IP si port). În continuare poate vedea postările publice, se poate înregistra (ca user normal sau ca administrator) sau loga cu un cont deja existent. În cazul userului normal dacă nu există conflicte în baza de date, userul se poate loga imediat. Pentru a se loga ca administrator, cererea trebuie acceptată de măcar unul dintre administratorii deja existenţi În funcţie de tipul contului, clientului i se vor pune la dispoziţie diferite acţiuni.

Utilizatorilor normali le furnizate următoarele funcţionalităţi:

- vizualizarea posturilor puse de prieteni, colegi sau alţii care sunt destinate grupului din care fac parte
- vizualizarea posturilor şi posibilitatea de modificare şi ştergere
- vizualizarea profilurilor care nu sunt private şi a userilor care sunt adăugaţi în vreun grup
- posibilitatea de a începe, de a participa şi de a părăsi o conversaţie
- vizualizarea cererilor de participare la conversaţie şi a userilor participanţi la o conversaţie activă
- posibilitatea de delogare

Iar administratorilor reţelei:

- vizualizarea tuturor posturilor şi userilor, şi posibilitatea de a îi şterge
- vizualizarea tuturor cererilor pentru a fi administrator şi posibilitatea de a îi accepta sau respinge

Pentru mai multe detalii despre modul de utilizare al interfeţei grafice se poate consulta Appendix A.

## 4 Detalii de implementare

Ca design am hotărât să despart implementarea clientului de interfaţa grafică deoarece astfel facilitez dezoltarea ulterioară a unui posibil alt client cu altă interfaţă grafică sau fară.

Pentru a compila fişierele sursă am folosit tehnica "shadow build" [11]. Aceasta presupune stocarea fişierelor executabile şi intermediare într-un alt folder decât cel al surselor. În final am realizat un fişier makefile care construieşte executabilele pentru client şi server. Pentru a crea executabilele deci este nevoie doar de o comandă make dată în folderul aplicaţiei, executabilele aferente clientului şi serverului găsindu-se în subfolderele build.

Comunicarea între server şi client se face prin intermediul socket-urilor BSD. Serverul şi clientul trimit un obiect JSON prefixat de dimensiunea lui. Structura unui JSON depinde de tipul de cerere şi tipul de răspuns. În cazul clientului câteva exemple de cereri sunt:

- Cererea de logare:
  {"MSG":"LOGIN","Username":"<username>","Password":"<password>"}
- Cererea de vizualizare useri şi posturi publice:
  {"MSG":"FINGER"}
- Cererea de înregistrare ca user:
  {"MSG":"REGISTER","Name":"<name>","Password":"<password>"}
- Cererea de postare:
  {"MSG":"POST","Type":"<groupType>","Text":"<text>"}
- Cererea de adăugare a unui user la lista de prieteni:
  {"MSG":"ADD","List":"<listType>","User":"<username>"}

Pe partea de server mesajele vor de două tipuri:

- A apărut o eroare:
  {"MSG":"Error","Error":"<text explicativ>""}
- S-a reuşit extragerea de informaţii dorite din baza de date:
  {"MSG":"OK":"DATA":
  [{"R1Col1":"<R1Val1>","R1Col2":"<R1Val2>",...},
  {"R2Col1":"<R2Val1>","R2Col2":"<R2Val2>",...},
  ...
  ]}
- S-a reuşit inserarea, actualizarea sau ştergerea din baza de date:
  {"MSG":"OK"}

Baza de date este relaţională creată cu ajutorul librăriei sqlite3. Pentru a-mi simplifica munca am folosit "DB Browser for SQLite", un utilitar open-source care facilitează crearea, designul şi editarea unei baze de date compatibile cu SQLite. Ca design am folosit cunoştinţele dobândite la cursul de baze de date unde am învăţat despre normalizări, dependenţe funcţionale, multivaluate, constrângeri, triggers, etc. Schema bazei de date este următoarea:

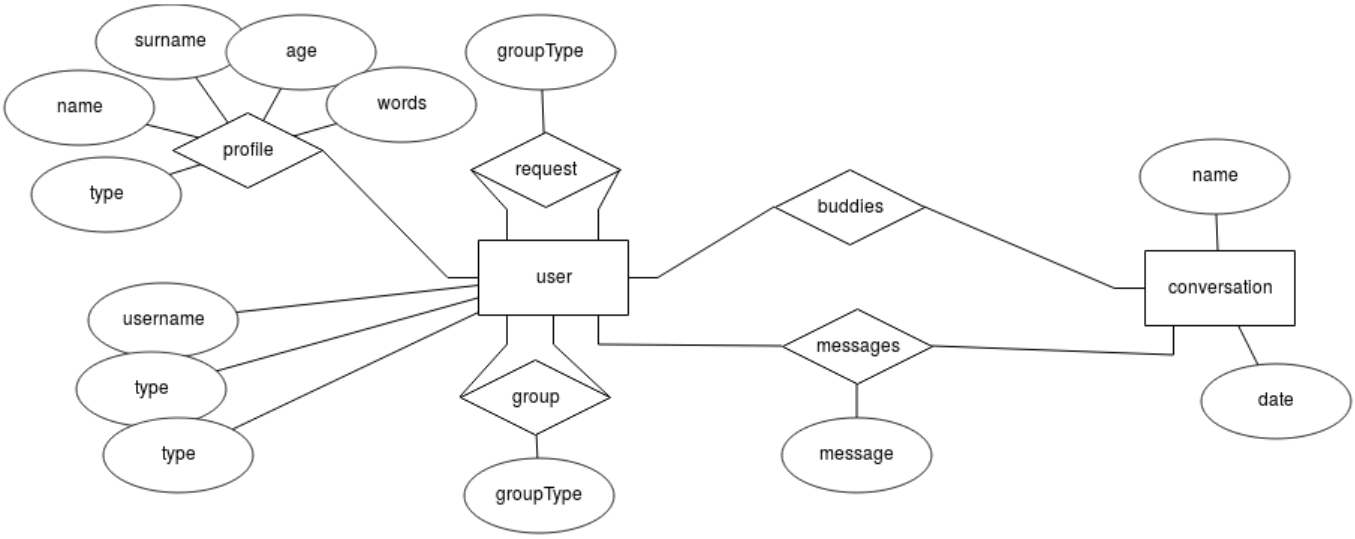


Fig. 3: schema bazei de date

### 4.1 Cod relevant

**Funcţia de trimitere JSON**

```
int Communication::sendMessage(int socket,json_t* json)
{
    char*  sendBuffer = NULL;
    char*  jsonBuffer = NULL;
    int  jsonSize;

    //get char* from json_t
    jsonBuffer  = json_dumps(json,JSON_PRESERVE_ORDER);
    if (jsonBuffer == NULL)
    {
```

```
        fprintf(stderr, "Error dumping json.");
        return -1;
    }
    jsonSize  = strlen(jsonBuffer);

    //construct send buffer, adding a header with size in bytes of JSON and * terminator
    sendBuffer = (char*)calloc( (10 + jsonSize + 1), sizeof(char));
    sprintf(sendBuffer,"%d%c%s",jsonSize,DELIMITER,jsonBuffer);

    //send buffer through socket
    if ( (send(socket,sendBuffer,strlen(sendBuffer),0)) < 0 )
    {
        free(sendBuffer);
        fprintf(stderr,"Error sending message:%s",strerror(errno));
        return -1;
    }

    free(sendBuffer);
    return 0;
}
```

**Funcţia de execuţie de interogări SQL şi serializarea rezultatelor în JSON**

```
    //compiling query
    sqlite3_prepare_v2(db, query, strlen(query), &stmt, NULL);

    //looping through results
    while (sqlite3_step(stmt) == SQLITE_ROW)
    {
        if (error)
            break;

        noColumns = sqlite3_data_count(stmt);
        if (noColumns == 0)
        {
            error = 1;
            strcpy(errMessage,"Couldn't get number of columns");
            break;
        }

        json_t* row = json_object();
        if (row == NULL)
        {
            strcpy(errMessage,"Couldn't create json object");
            error = 1;
```

```
            break;
        }

        //add object to array of data acordingly with their type
        for (int i=0;i<noColumns;i++)
        {
            switch(sqlite3_column_type(stmt,i))
            {
                case SQLITE_INTEGER:
                    if ( (error = json_object_set(row,sqlite3_column_name(stmt,i),
                    json_integer(sqlite3_column_int(stmt,i)))) == -1 )
                        strcpy(errMessage,"Couldn't create json object");
                break;
                case SQLITE_TEXT:
                    if ( (error = json_object_set(row,sqlite3_column_name(stmt,i),
                    json_string((const char*)sqlite3_column_text(stmt,i)))) == -1 )
                        strcpy(errMessage,"Couldn't create json object");
                break;
            }
        }

        if (!error)
            if ( (error = json_array_append(data,row)) == -1)
                strcpy(errMessage,"Couldn't add to array");
    }

    sqlite3_finalize(stmt);

    if (error)
    {
         json_object_set(responseJson,"ERROR",json_string(errMessage));
    }
    else
    {
        json_object_set(responseJson,"MSG",json_string("OK"));
        json_object_set(responseJson,"Data",data);
    }

    return responseJson;
```

### 4.2 Use-case-uri

Diagrama use-case:

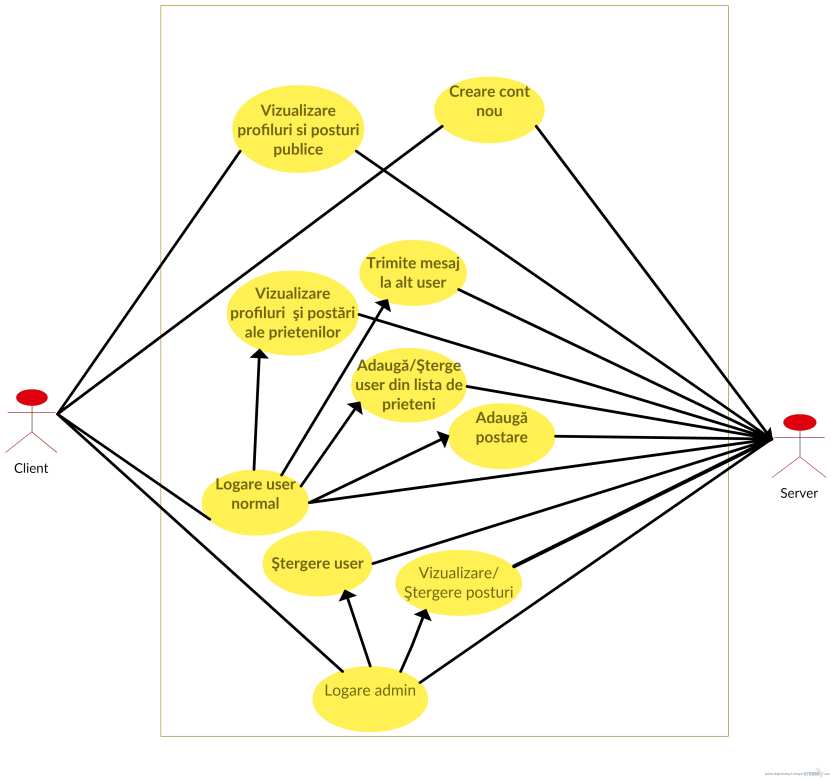


Fig. 4: diagrama use-case

## 5 Concluzii

Scopul proiectul, pe lângă cel de a implementa o reţea socială cu diverse funcţionalităţi, este de a înţelege modul de comunicare în reţea, arhitectura client-server, serverul concurent, modul de utilizare al thread-urilor şi al unei baze de date. Consider că proiectul VirtualSoc constituie un punct de plecare în dezvoltarea unei aplicaţii de socilizare în mediul virtual.

### 5.1 Idei de îmbunătaţire ale aplicaţiei

Dezvoltarea aplicaţiei din acest punct va fi simplu de realizat deoarece codul este unul reutilizabil iar unui posibil contributor îi va fi foarte uşor să se folosească de ceea ce deja a fost implementat si tratat. Iată câteva idei:

– Nu am reuşit realizarea comunicării instantanee între clienţi. Mesajele trimise sunt stocate în baza de date iar pentru a vedea mesajele trimise de

ceilalţi useri, un client trebuie să facă o cerere de a vedea toate mesajele unei conversaţii ceea ce poate fi costisitor. O idee de dezvoltare ulterioară ar fi inserarea în interiorul protocolului creat de mine a unui protocol de instant-messaging precum IRC sau XMPP
- Trimiterea de mesaje prin reţea să se facă criptat pentru a face aplicaţia invulnerabilă la programe maliţioase
- Oferirea de funcţionalităţi de vizualizare şi controlare a fluxului de date pentru un administrator pentru a putea eventual optimiza acele părţi din aplicaţie folosite intens
- Implementarea serverului prethreaded pentru o rapiditate şi mai mare
- Oferirea userilor posibilitatea de a posta fotografii, fişiere, video-uri (poate chiar şi live)

# Appendix A Scheme cu funcţionalităţile interfeţei grafice

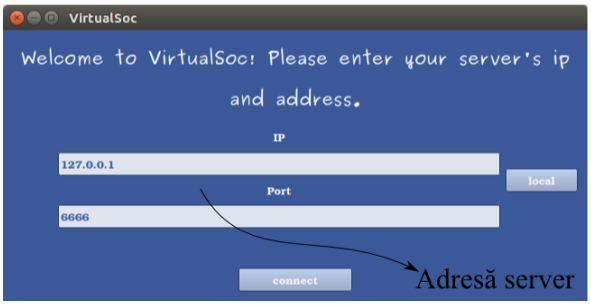


Fig. 5: conectare la un server prin ip si port


Fig. 6: fereastra de logare, înregistrare, şi vizualizare doar posturi publice

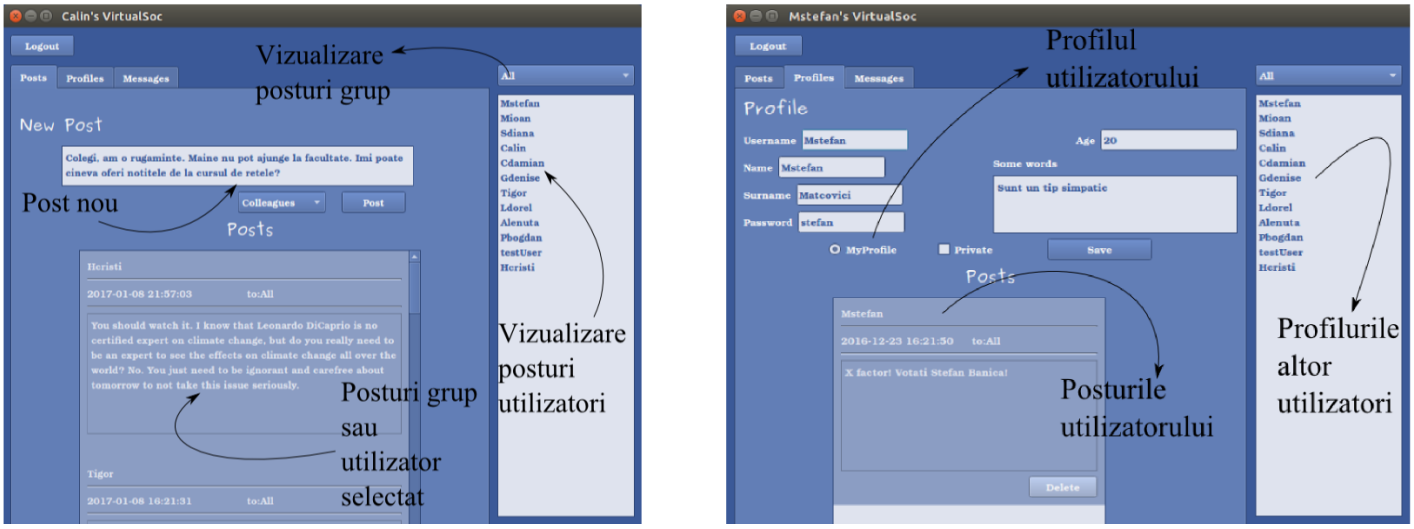


Fig. 7: tab-urile Posts şi Profiles

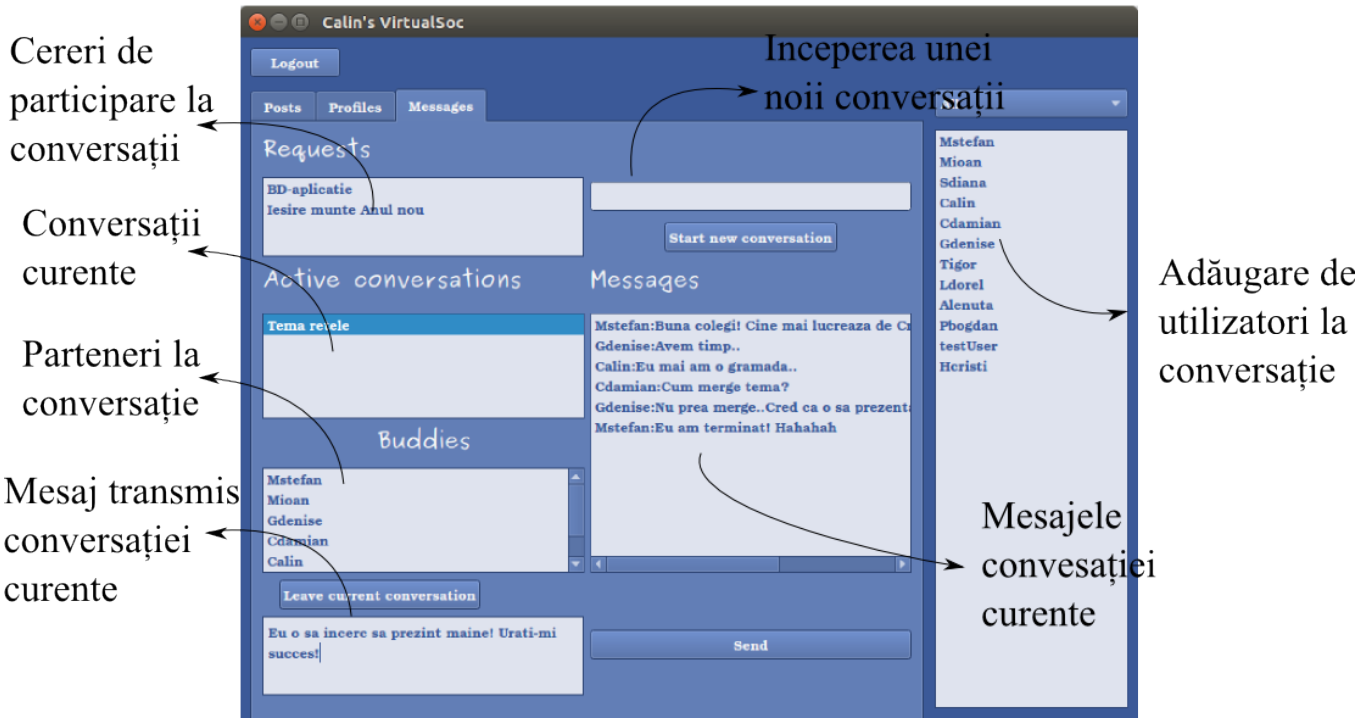


Fig. 8: tab-ul Messages

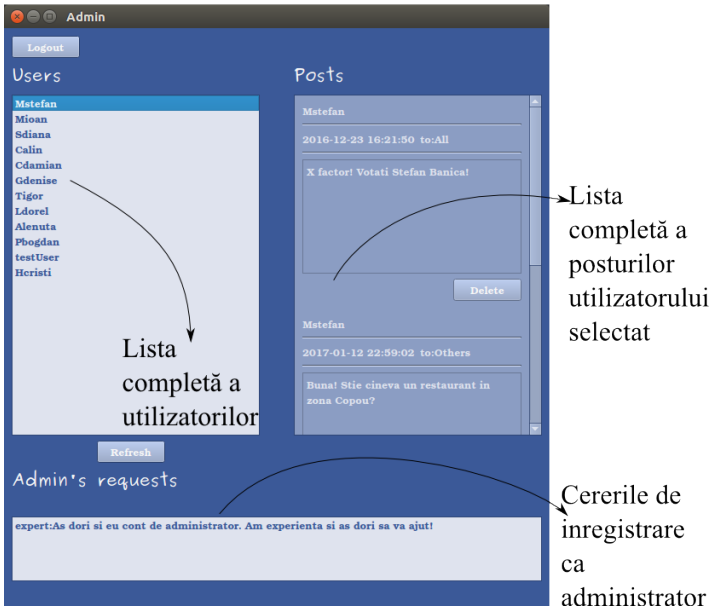


Fig. 9: fereastra administratorului

## References


1. Andrew S. Tanenbaum, David J. Wetherall: Computer Networks 5th Edition. (2010)
2. S. Buraga, G. Ciobanu: Atelier de programare în reţele de calculatoare Polirom Iaşi (2001)
3. A. Silberschatz, P. Galvin, G. Gagn: Operating System Concepts Essentials 8th edition. Willey and Sons (2011)
4. Ray Rischpater: Application Development with Qt Creator Packt Publishing (2013)
5. Michael Owens: The Definitive Guide to SQLite Springer-Verlag New York (2006)
6. Jay A. Kreibich: Using SQLite O'Reilly Media (2010)
7. http://www.digip.org/jansson/
8. https://www.tutorialspoint.com/sqlite/sqlite_tutorial.pdf
9. http://doc.qt.io/
10. http://profs.info.uaic.ro/ adria/teach/courses/net/cursullaboratorul.php
11. http://doc.qt.io/qtcreator/creator-glossary.html#glossary-shadow-build
12. www.internetlivestats.com/
13. http://www.json.org/