# Rețele de calculatoare. Raportul tehnic.
## VirtualSoc

Bursuc E. Eduarda

Universitatea A.I.Cuza

## 1 Introducere

Proiectul ales, VirtualSoc, propune simularea unei rețele sociale cu următoarele funcții implementate: înregistrarea unui cont nou public sau privat de tip user (client) sau admin (administrator), autentificarea prin nume de utilizator și parolă, adăugarea prietenilor în grupuri speciale (prieten sau prieten apropiat), publicarea noutăților atât public cât și pentru grupurile speciale și opțiunea de chat ce permite comunicarea între doi sau mai mulți utilizatori activi.

## 2 Tehnologii Aplicate

Rețelele sociale implică un schimb constant de date între client și server unde integritatea acestora are o importanță vitală, motiv pentru care am ales protocolul de comunicare TCP, care se ocupă bine de gestionarea conexiunilor, asigurându-ne că datele utilizatorilor sunt transferate în mod corect și complet. Iar pentru asigurarea comunicării între utilizatori și utilizarea paralelă a rețelei de către mai mulți clienți, am implementat un server TCP cu multiplexare I/O prin primitiva select() care permite gestionarea și prelucrarea clienților simultană și interacționarea în timp real.

De asemenea, pentru stocarea datelor despre utilizatori, profiluri, postări și relații, am optat pentru sistemul de gestionare a bazelor de date SQLite3 ce oferă o soluție ușor de implementat și fără server pentru gestionarea bazelor de date în cadrul aplicațiilor, fiind portabil și eficient în citire.

## 3 Structura Aplicației

Aplicația utilizează o arhitectură client-server, unde clientul transmite cereri către server prin intermediul unui set de comenzi specifice. Pentru a gestiona eficient un flux crescut de utilizatori, se folosește un protocol de comunicare bazat pe TCP concurent. Acest aspect permite serverului să gestioneze simultan multiple conexiuni, ceea ce optimizează accesul și răspunsul către un număr mai mare de utilizatori. Pentru a gestiona concurența într-un mod eficient, aplicația beneficiază de funcționalitatea select() a TCP, permițând o gestionare non-blocantă a comunicării între clienți și server. În cadrul acestui protocol, se utilizează un

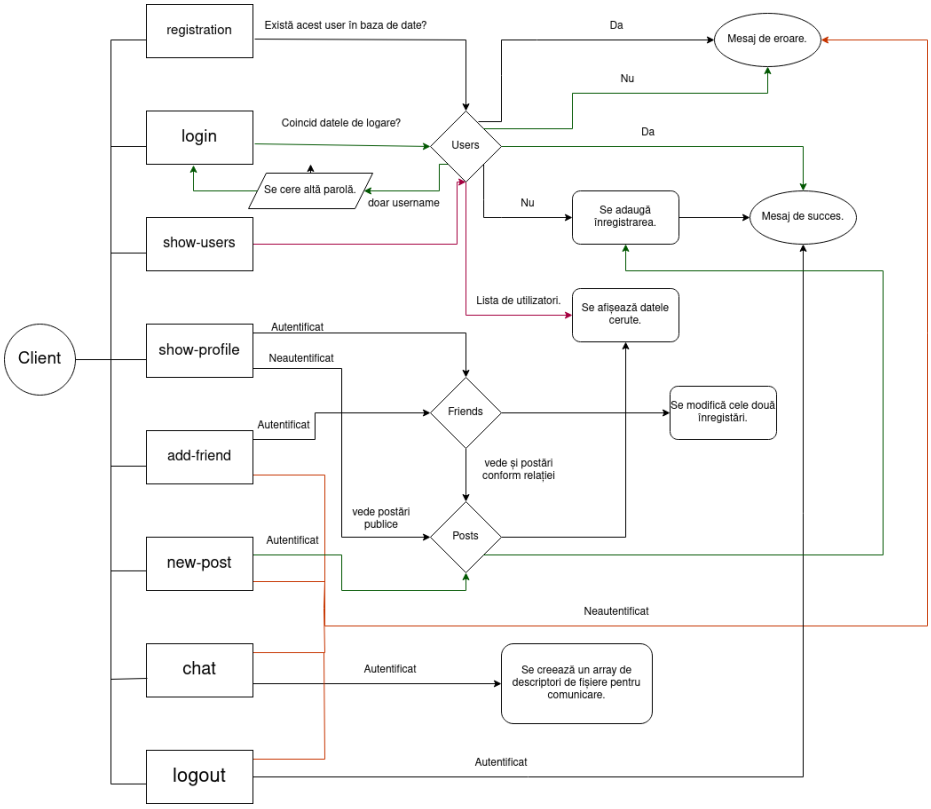


**Fig. 1.** Diagrama detaliată

set bine definit de comenzi: registration, login, logout, show-users, show-profile, add-friend, new-post și chat care facilitează comunicarea de la client la server.

Diagrama detaliată prezintă o construcție aproximativă a programului incluzând comenzile specifice și tabelele din baza de date necesare.

## 4 Aspecte de Implementare

Implementarea constituie un set de comenzi pe care le poate introduce un utilizator pentru a introduce date sau vizualiza conținut. Toate comenzile vor fi inițial recunoscute în funcția principală și redericționate funcțiilor corespunzatoare cu parametri potriviți.

1. Comanda **registration** este preluată de o funcție specifică în cazul unul utilizator neutentificat, care primește descriptorul de fișier al utilizatorului și cere un set de informații, cum ar fi tipul de cont, tipul de utilizator, un nume de utilizator și o parolă. În momentul primirii unui nume de utilizator, funcția va accesa baza de date Users și va verifica dacă există un cont cu același nume, în

caz afirmativ se produce un mesaj de eroare, iar în caz contrar se alocă informația în baza de date și se va trimite un mesaj de succes.

De asemenea, funcția principală va redirecționa clientul spre funcția login.

2. Comanda **login** trimite ca parametru o structură User a căror componente sunt fd (file descriptor) și username(inițial cu valoare nulă) unei funcții corespunzătoare ce va cere un nume de utilizator și va verifica în baza de date Users existența acestuia. În caz afirmativ se va cere o parola până când aceasta nu va coincide cu parola de pe aceeași linie în baza de date. În caz contrar se va produce un mesaj de eroare. De asemenea, în cazul în care autentificare a avut succes, se va actualiza în structura User componenta username și se va produce un text de succes.

3. Comanda **logout** este executată în funcția principală și verifică inițial dacă structura User specifică utilizatorlui are un username atașat (lucru ce ne spune că utilizatorul este autentificat) și în acest caz o va șterge, iar în caz contrar - trimite un mesaj utilizatorului că nu este autentificat.

4. Comanda **show-users** transmite doar descriptorul de fișier funcției ce accesează baza de date Users și colectează toți utilizatorii existenți pentru a-i afișa utilizatorului.

5. Comanda **show-profile : username** transmite funcției structura User și numele utilizatorului ai cărui profil vrea să-l viziteze. În cazul unui utilizator neautentificat, se accesează baza de date Posts și se selectează linia utilizatorului cerut și coloana cu postări de tip public, iar în caz contrar se verifică tipul de relație a celor doi utilizatori dacă există (friends, close friends) în baza de date Friends și în funcția acesteia se cer postări din Posts.

6. Comanda **add-friend : username relationship** este prelucrată de o funcșie ce primește structura User, numele utilizatorului ce trebuie adăugat, relația dorită și modifcă sau adaugă o nouă înregistrare în baza de date Friends. De asemenea, trimite un mesaj de succes. În cazul unui utilizator neautenficat, se va trimite doar un mesaj de eroare.

7. Comanda **new-post** este redirecționată funcșiei ce primește structura User și cere tipul postării (public, friends, close friends) și conținutul acesteia, urmând ca uletrior să introducă o nouă înregistrare în baza de date Posts pe linia utilizatorului și coloana tipului cerut. În cazul unui utilizator neautenficat, se va trimite doar un mesaj de eroare.

8. Comanda **chat : username ... username** este autentifictă în funcția principală și se creează un vector cu descriptorii de fișiere corespunzatoare numelor de utilizatori activi și se transmite unei funcții ce va avea proprietatea unui broadcast. La generarea unui mesaj specific un utilizator poate ieși din conversație. În cazul unui utilizator neautenficat, se va trimite doar un mesaj de eroare.

În următoarele figuri se pot observa secțiuni de cod specifice lucrului cu baze de date și implementarea structurii User menționată mai sus.

**Fig. 2** și **3** ilustrează crearea celor trei tabele Users, Posts și Friends menționate mai sus cu coloanele bine definite:

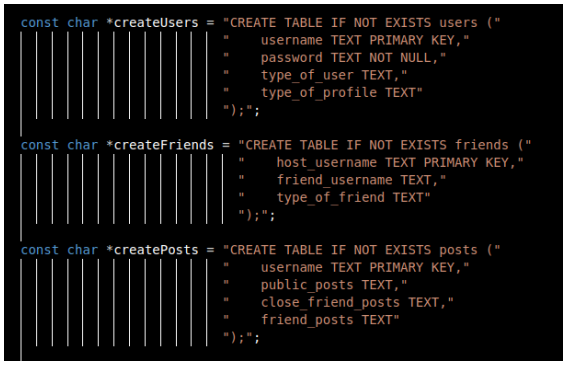


**Fig. 2.** Crearea interogărilor

Iar pe măsură ce datele de inserție a clientului sunt verificate inițial în aplicația client și ulterior în unele funcții ale aplicației server, nu este necesar să condiționăm coloanele cu NOT NULL.

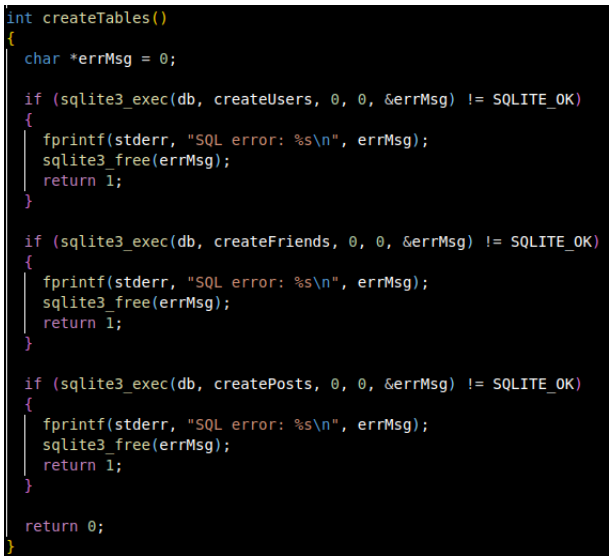


**Fig. 3.** Execuția și crearea propriu-zisă a tabelelor din baza de date

**Fig. 4** reprezintă inserarea datelor specifice unui utilizator în tabela Users în urma primirii și validării acestora, folosind funcția sqlite3_bind_text(sqlite3_stmt*, int, const char*, int, void(*)(void*)) care este utilizată în SQLite pentru a lega o valoare de tip text la o instrucțiune SQL pregătită:

```
const char *selectDataSQL = "insert into users values (?, ?, ?, ?);";
if (sqlite3_prepare_v2(db, selectDataSQL, -1, &statement, 0) == SQLITE_OK)
{
  sqlite3_bind_text(statement, 1, username, -1, SQLITE_STATIC);
  sqlite3_bind_text(statement, 2, passwd, -1, SQLITE_STATIC);
  sqlite3_bind_text(statement, 3, type_usr, -1, SQLITE_STATIC);
  sqlite3_bind_text(statement, 4, type_acc, -1, SQLITE_STATIC);

  if (sqlite3_step(statement) != SQLITE_DONE)
  {
    strcpy(answ, "Datele introduse sunt invalide.");
    write(fd, answ, strlen(answ));
  }
  else
  {
    strcpy(answ, "Inregistrare cu succes.");
    write(fd, answ, strlen(answ));
  }
}
```

**Fig. 4.** Înserarea datelor specifice în baza de date

**Fig. 5** prezintă definirea structurii User despre care am vorbit în partea de implementare:

```
struct User
{
  int fd;
  char username[101];
};
```

**Fig. 5.** Definirea structurii User

În **Fig. 6** este reprezentată inițializarea cu șir de caractere nul a componentei username pentru a facilita validarea corectă a datelor în funcții ce au funcționalitate diferită în conformitate cu statutul utilizatorului (autentificat sau nu). De asemenea, prezintă alocarea descriptorilor de fișiere activi în tabloul unidimensional users de tip User.

```
for (int j = 0; j < 10001; j++)
  strcpy(users[j].username, "");
```

```
FD_SET(client, &actfds);
users[i].fd = client;
i++;
```

**Fig. 6.** Inițializarea componentelor

Modificarea numelui de utlizator a structurii User se realizează în funcția login care în urma validării datelor va aloca numele specificat în componenta specifică:

```
strcpy(user->username, username);
strcpy(answ, "Logare cu succes.");
write(user->fd, answ, strlen(answ));
```

**Fig. 7.** Modificarea componentei username

Așa cum gestionarea utilizatorilor se face cu ajutorul descriptorului de fișier al acestuia, în situațiile în care avem nevoie să validăm dacă un utilizator este sau nu autentificat sau dacă primim pentru comanda chat o înșiruire de nume de utilizatori și trebuie în server să selectăm doar acei descriptori de fișier corespunzători acestor nume, avem nevoie de o astfel de structură ca să identificăm corespondența între file descriptor și username. Primul caz este ilustrat în **Fig. 8** de mai jos:

```
if (strcmp(command, "registration") == 0)
{
  for (j = 0; j < i; j++)
  {
    if (users[j].fd == fd)
    {
      if (strlen(users[j].username) > 0)
      {
        strcpy(answ, "Sunteti autentificat.");
        write(fd, answ, strlen(answ));
      }
      else
      {
        registration(fd);
      }
    }
  }
  // strcpy(answ, "Cerere de registrare.");
}
```

**Fig. 8.** Funcționalitatea structurii User

Pentru a preveni erori și a îmbunătăți aplicația la nivelul memoriei de alocare, în momentul deconectării unui utilizator, structura acestuia în tabloul users va fi înlocuită cu ultima înregistrare făcută, întrucât ordinea descriptorilor nu are nici o importanță în implementare:

```c
ssize_t bytes_received = recv(fd, command, sizeof(command), 0);

if (bytes_received <= 0)
{
  for( j = 0; j < i; j++ ){
    if( fd == users[j].fd ) {
      users[j] = users[i-1];
      i--;
      break;
    }
  }
  close(fd);
  FD_CLR(fd, &actfds);
  printf("[server] Client with descriptor %d disconnected.\n", fd);
  continue;
}
```

**Fig. 9.** Eliminarea unui utilizator din tabloul users

## 5 Concluzii

Aplicația poate suferi și mici modificări pe parcurs ce vor imbunătăți soluția de implementare descrisă, de exemplu modificarea aplicației client pentru a adapta comunicarea atât pentru comenzi simple cât și pentru chat, de asemenea, pot fi adăugate noi funcții și comenzi pentru o simulare mai bună a unei rețele sociale familiare, de exemplu o comanda pentru modificare a tipului de cont.

## Referințe bibliografice

1. www.sqlite.org/c3ref/intro.html