

# 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

### 2.1 TCP

Rețelele sociale implică un schimb constant de date între client și server, unde integritatea și ordinea acestora este foarte importantă, motiv pentru care am optat pentru un protocol de comunicare TCP (Transmission Control Protocol). Acesta spre deosebire de UDP se ocupă bine de gestionarea conexiunilor, asigurând că datele utilizatorilor sunt transferate în mod corect și complet. De asemenea, pentru implementarea concurenței ce asigură utilizarea paralelă a rețelei de către mai mulți clienți, am folosit un server TCP cu multiplexare I/O prin primitiva `select()` care permite gestionarea și prelucrarea clienților simultană și interacționarea în timp real.

### 2.2 SQLite

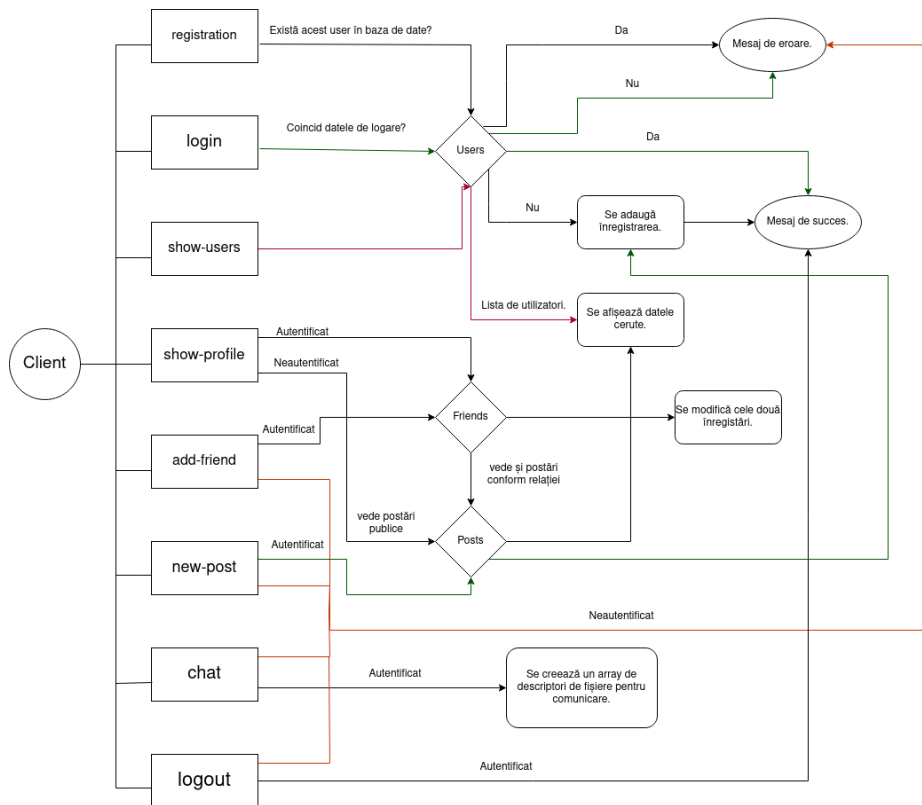
Pentru stocarea datelor despre utilizatori, profiluri, postări și relații, am optat pentru sistemul SQLite3 ce oferă o soluție ușor de implementat și fără server pentru gestionarea bazelor de date în cadrul aplicațiilor, fiind portabil și eficient în citire.

### 2.3 Socket

Utilizarea socket-urilor în cadrul TCP oferă o soluție robustă și eficientă pentru dezvoltarea aplicațiilor de rețea care necesită o comunicare fiabilă și bidirecțională. Unele din beneficiile socket-ului importante pentru proiectul dat: comunicarea bidirecțională, fiabilitatea, reasamblarea pachetelor, reconectarea automată, adresarea și menținerea unor conexiuni persistente.

### 3 Structura Aplicației

Aplicația utilizează o arhitectură client-server, unde clientul transmite cereri către server prin intermediul unui set de comenzi bine definit: registration, login, logout, show-users, show-profile, add-friend, new-post și chat cu parametri potriviți. În urma primirii unei cereri, serverul analizează comanda și cere, dacă este nevoie, informații suplimentare, după care se ocupă de prelucrarea datelor: verificarea acestora în baza de date, înregistrarea unor date noi, colectarea conținutului dorit etc. În urma procesării cererii, serverul trimite clientului un mesaj de succes sau conținutul cerut de acesta în conformitate cu specificații și datele deja cunoscute din baza de date.



**Fig. 1.** Diagrama detaliată

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

## 4 Aspecte de Implementare

### 4.1

Inițial am definit o structură `User` ce va diferenția utilizatorii după două componente: `username` (nume de utilizator) și `fd` (descriptor de fișier), aceasta va fi utilizată ulterior pentru determinarea statutului de autentificare a unui utilizator și recunoașterea acestuia în lista de descriptori de fișier de citire.

Implementarea propriu-zisă 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 corespunzătoare cu parametri potriviți.

1. Comanda **registration** este preluată de o funcție specifică în cazul unui utilizator neautentificat, 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.

2. Comanda **login username/password** este prelucrată inițial prin verificarea componentei `username` a structurii `User` care ne indică statutul de autentificare a utilizatorului. În cazul în care utilizatorul este autentificat se va afișa un mesaj specific, în caz contrar se va verifica numele de utilizator introdus în baza de date `Users` și în cazul existenței unui astfel de utilizator se vor compara cele două parole. Pentru o parolă greșită se va afișa un mesaj specific, iar în cazul autentificării cu succes, componenta `username` în structura `User` va fi modificată.

3. Comanda **logout** este executată în funcția principală și verifică inițial dacă structura `User` specifică utilizatorului 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** este prelucrată de o funcție ce primește structura `User`, numele utilizatorului ce trebuie adăugat, și cere relația dorită între ei (prieteni sau prieteni apropiați) și modifică sau adaugă o nouă înregistrare în baza de date `Friends`. De asemenea, trimite un mesaj de succes. În cazul unui utilizator neautentificat, se va trimite doar un mesaj de eroare.

7. Comanda **new-post** este redirecționată funcției ce primește structura `User` și cere tipul postării (`public`, `friends`, `close friends`) și conținutul acesteia, urmând

ca ulterior să introducă o nouă înregistrare în baza de date Posts pe linia utilizatorului și coloana tipului cerut. În cazul unui utilizator neautentificat, se va trimite doar un mesaj de eroare.

8. Comanda **chat : username ... username** este autentificată în funcția principală și se creează un vector cu descriptorii de fișiere corespunzătoare 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 neautentificat, se va trimite doar un mesaj de eroare.

De asemenea, prin avansarea în implementare este posibilă adăugarea sau modificarea unor comenzi ce vor apropia proiectul de o rețea socială familiară și vor îmbunătăți utilizarea aplicației în paralel.

## 4.2

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:

```
const char *createUsers = "CREATE TABLE IF NOT EXISTS users ("
    "    username TEXT PRIMARY KEY,"
    "    password TEXT NOT NULL,"
    "    type_of_user TEXT,"
    "    type_of_profile TEXT"
    ");";

const char *createFriends = "CREATE TABLE IF NOT EXISTS friends ("
    "    host_username TEXT PRIMARY KEY,"
    "    friend_username TEXT,"
    "    type_of_friend TEXT"
    ");";

const char *createPosts = "CREATE TABLE IF NOT EXISTS posts ("
    "    username TEXT PRIMARY KEY,"
    "    public_posts TEXT,"
    "    close_friend_posts TEXT,"
    "    friend_posts TEXT"
    ");";
```

**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 specifice în interogarea de creare cu NOT NULL.

```

int createTables()
{
    char *errMsg = 0;

    if (sqlite3_exec(db, createUsers, 0, 0, &errMsg) != SQLITE_OK)
    {
        fprintf(stderr, "SQL error: %s\n", errMsg);
        sqlite3_free(errMsg);
        return 1;
    }

    if (sqlite3_exec(db, createFriends, 0, 0, &errMsg) != SQLITE_OK)
    {
        fprintf(stderr, "SQL error: %s\n", errMsg);
        sqlite3_free(errMsg);
        return 1;
    }

    if (sqlite3_exec(db, createPosts, 0, 0, &errMsg) != SQLITE_OK)
    {
        fprintf(stderr, "SQL error: %s\n", errMsg);
        sqlite3_free(errMsg);
        return 1;
    }

    return 0;
}

```

Fig. 3. Execuția și crearea propriu-zisă a tabelor 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. Această funcție va fi întâlnită de asemenea la interogări select, update etc.

```

const char *insertDataSQL = "insert into users values (?, ?, ?, ?)";
if (sqlite3_prepare_v2(db, insertDataSQL, -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. Înscrierea 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 utilizator a structurii User se realizează în funcția login care în urma validării datelor va alocă 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

## 5 Concluzii

Aplicația poate suferi și mici modificări pe parcurs ce vor îmbunătăți soluția de implementare descrisă, de exemplu pot fi adăugate noi funcții și comenzi pentru o simulare mai bună a unei rețele sociale, de exemplu o comanda pentru modificarea tipului de cont.

De asemenea, protocolul va suferi schimbări în conformitate cu funcționalul acestuia. Spre exemplu așa cum un chat nu ar trebui să forțeze un timp de așteptare pentru alți utilizatori, o soluție ar fi crearea unui proces copil ce va colecta descriptorii de fișier a utilizatorilor unui chat și le va șterge temporar din componenta descriptorilor activi, creând una noua care va conține doar particiții conversației, urmând ca după ieșire din chat să fie din nou adăugați descriptorilor activi global.

## Referințe bibliografice

1. [www.sqlite.org/c3ref/intro.html](http://www.sqlite.org/c3ref/intro.html)