VirtualSoc

Isac Lucian-Constantin

January 3, 2022

1 Introducere

Scopul proiectului este de a simula o retea sociala in cadrul careia clientul poate realiza postarea de mesaje vizibile publice sau private si permite comunicare intre 2 clienti. Informatiile sunt memorate intr-o baza de date, fiind accesate prin intermediul unui server.

2 Tehnologii utilizate

2.1 TCP

TCP este un protocol de transport orientat conexiune care ofera siguranta in transportul de date. Serverul si clientul participa la transferul de date: serverul asteapta aparitia unui client pentru a incepe comunicarea.
Am ales TCP in loc de UDP pentru siguranta oferita in comunicare in schimbul vitezei. Nu sunt permise erori in transferul de mesaje (baza de date trebuie sa memoreze datele transmise de client).

2.2 Thread

Threadul (firul de executie) este folosit pentru a putea optimiza executia programelor concurent. Threadurile vor fi create odata cu conectarea unui client la server.
Am ales sa folosesc Threaduri in loc de procese datorita posibilitatii de a crea mai multe fire de executie intr-un timp mai scurt comparativ cu procesele.

2.3 Qt

Qt este folosit pentru realizarea interfetei grafice a acestui proiect (clientul genereaza interfata). Qt ofera un IDE (Qt creator) foarte util pentru realizarea unei interfete intrucat nu este necesara crearii acesteia in mod programatic.

2.4 Sqlite3

Sqlite3 este folosit pentru stocarea informatiilor in baza de date (useri, postari, mesaje). Sqlite3 are o performanta foarte buna, este flexibila, datele sunt stocate intr-un fisier pe disc iar API-ul este destul de intuitiv.

3 Arhitectura aplicatiei

Serverul este multi-threaded, adica va fi folosit un thread pentru fiecare client. Dupa realizarea conexiunii si a threadului, acesta asteapta sa primeasca informatii de la client. Tot aici se face si comunicarea cu baza de date. Mesajele clientului vor fi trimise astfel incat se pot identifica tabelul si tipul operatiei ce trebuie executata. modelul implementat este asemanator cu un MVC deoarece clientul trimite un mesaj de forma: header mesaj, unde headerul este de forma tabel@functie#id (id-ul poate lipsi) iar mesajul poate fi null, reprezinta campurile ce trebuie inserate sau updatate. Modelul tabelului, mesajul si id-ul (daca exista) sunt trimise in functie (poate fi select, insert, etc.).
Clientul implementeaza interfata grafica, cu ajutorul careia va trimite informatia spre server. Prin interactiunea cu campurile text si a butoanelor clientul trimite informatia spre server, ea fiind structurata ca mai sus. Un user normal se poate loga, deloga (doar daca este logat), vizualiza postarile (daca nu este logat doar pe cele publice) si poate comunica cu alti useri (daca este logat). In plus un user care este si administrator poate sterge postari sau conturi.

Figure 1: Diagrama aplicatiei

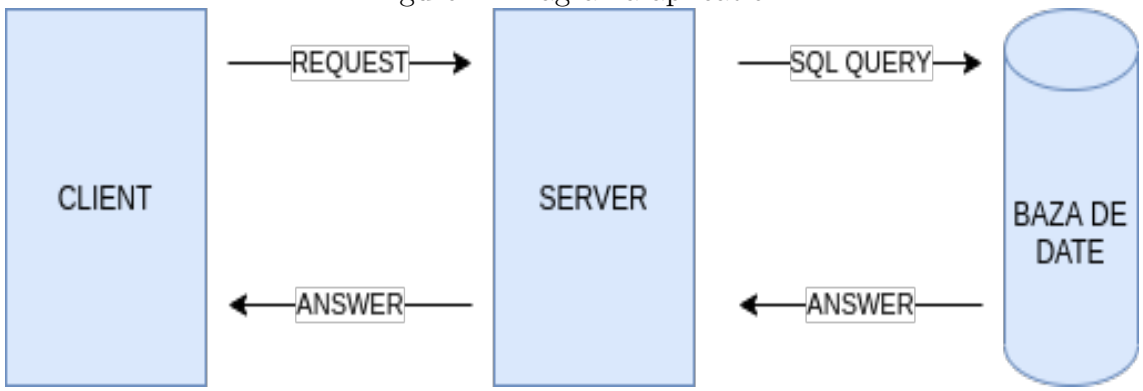
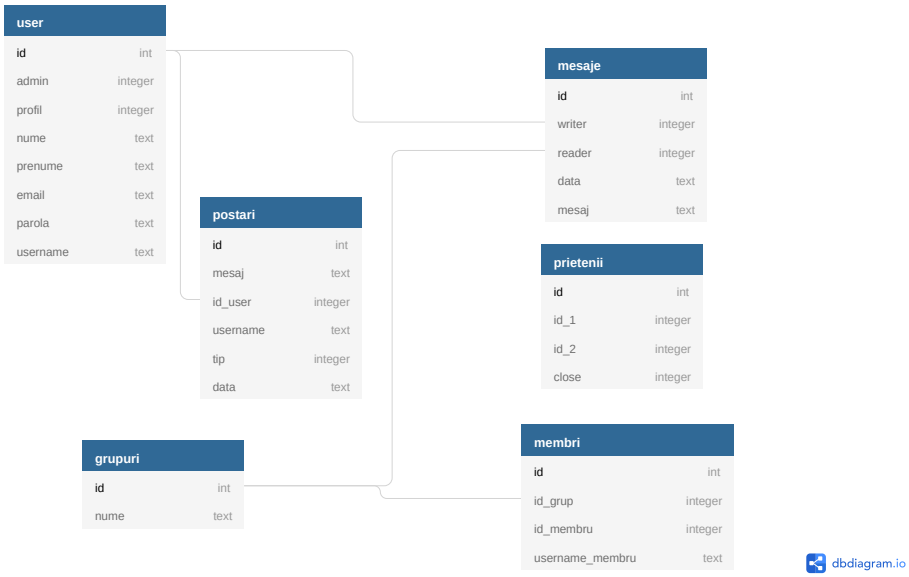


Figure 2: Baza de date:



4 Detalii de implementare

In cadrul serverului am organizat socketul si crearea threadurilor intr-o structura cu urmatoarele metode: CreateSoc (creaza socketul , realizeaza bind si listen), Accept (realizeaza conexiunea cu un client), CreateThread (creaza threadul) si treat(realizeaza pthread_detach si apeleaza functia Handle).
Functia de handle va astepta clientul sa trimita stringuri sub forma header mesaj.

```
void Handle() {
    int sd;
    pthread_t self = pthread_self();
    Thread info = threads[self];

    if(sqlite3_open(Database, &db) != SQLITE_OK)
    {
        perror ("Eroare la sqlite3_open().\n");
        return;
    }

    int id, ind, i;
    char *pch;
    string functie;
    vector <string> parametri;
    char buffer[size], header[hsize], mes[size];

    while(1)
    {
```

```
memset(mes, 0, sizeof(mes));
if((sd = read(info.client, mes, sizeof(mes))) <= 0)
{
    if(sd == -1)
        perror ("Eroare la read().\n");
    break;
}

memset(header, 0, sizeof(header));
memset(buffer, 0, sizeof(buffer));

for(ind = 0; mes[ind] != '\n'; ind++)
    header[ind] = mes[ind];

if(strcmp(header, "exit") == 0)
    break;

if(strcmp(header, "logout") == 0)
{
    threads[self].userId = -1;
    threads[self].username = "";
    continue;
}

ind++;
i = 0;
for(;mes[ind]; ind++)
    buffer[i++] = mes[ind];

cout << header << "\n" << buffer << "\n";

// header = tabel@functie#id (id poate sa nu existe)
id = -1;
string aux(header);
parametri.clear();
int pos = aux.find("@");
parametri.push_back(aux.substr(0, pos));     // parametri[0] = numel
functie = aux.substr(pos + 1);

if((pos = functie.find("#")) != string::npos) // am id
{
    id = stoi(functie.substr(pos + 1));
    functie = functie.substr(0, pos);
}
pch = strtok(buffer, "\n");
while(pch != NULL)
{
```

```
            string aux(pch);
            parametri.push_back(aux);
            pch = strtok(NULL, "\n");
        }

        string ans = db_requests[functie](parametri, id);

        if(write(info.client, ans.c_str(), strlen(ans.c_str())) < 0)
        {
            perror ("Eroare la write().\n");
            break;
        }

    }

    sqlite3_close(db);
    close(info.client);
    pthread_mutex_lock(&mlock);
    threads.erase(self);     //sectiune critica?
    pthread_mutex_unlock(&mlock);

    pthread_exit(0);
}
```

Stringul ans va retine outputul dorit obtinut dupa accesul lui db_request care este un map de forma:

```
map <string, function<string(vector<string>, int)> > db_requests = {
    {"select", Select},
    {"insert", Insert},
    {"update", Update},
    {"delete", Delete}
};
```

Am folosit acest map pentru a apela functiile respective folosind stringuri. Exemplu: daca headerul are forma user@select#1 (vreau sa selectez din tabela user, userul cu id=1). Pentru asta apelez db_request["select"](<user >, 1) (intotdeauna vectorul info va avea pe pozitia 0 numele tabelei asupra careia se efectueaza operatia). Acest apel va redirectiona parametri transmisi la o functie de select implementata.

```
string Select(vector<string> info, int id) {
    string model = info[0];
    string query = "SELECT * FROM " + model;

    if(!(id == -1 && info.size() == 1))
    {
        query = query + " WHERE";
        if(id != -1)
            query = query + " id IN (" + to_string(id) + ") AND";
```

```
    if(info.size() > 1)
    {
        for(int i = 1; i < (tabel[model]).size(); i++)
        {
            if(info[i] == null)
                continue;

            auto at = (tabel[model])[i].find('@');
            query = query + " " + (tabel[model])[i].substr(0,
            at) + " IN (";

            if(at != string::npos)
            {
                size_t pos = 0;
                while((pos = info[i].find(",", pos)) != string::npos)
                {
                    info[i].replace(pos, string(",").length(),
                    string("','"));
                    pos += string("','").length();
                }
                info[i].insert(0, 1, '\'');
                info[i].push_back('\'');
            }

            query = query + info[i];

            query = query + ") AND";


        }
    }
    query.erase(query.length() - 4);
}

sqlite3_stmt* stmt;
sqlite3_prepare_v2(db, query.c_str(), -1, &stmt, 0);

string response("");
while(sqlite3_step(stmt) != SQLITE_DONE)
{
    response = response + "{\n";
    for(int i = 0; i < (tabel[model]).size(); i++)
    {
        auto at = (tabel[model])[i].find('@');
        response = response + (tabel[model])[i].substr(0, at)
                    + ":" +
```

```
                    string(reinterpret_cast
                    <const char*>(sqlite3_column_text(stmt, i)));

        // (e const unsigned char* rezultatul si nu merge
        convertit la string)
            response = response + "\n";
        }
        response = response + "}\n";
    }
    return response;
}
string Insert(vector<string> info, int id) {
    string model = info[0];
    ...
}
string Update(vector<string> info, int id) {
    string model = info[0];
    ...
}
string Delete(vector<string> info, int id) {
    string model = info[0];
    ...
}
```

Stiind tabelul pe care trebuie facuta operatia respectiva si parametrii se poate executa interogarea si returnarea raspunsului. Raspunsul primit in urma selectului v-a fi un string de forma:

```
{
camp_1:valoare_item_1_camp_1
camp_2:valoare_item_1_camp_2
...
camp_n:valoare_item_1_camp_n
}
{
camp_1:valoare_item_2_camp_1
camp_2:valoare_item_2_camp_2
...
camp_n:valoare_item_2_camp_n
}
...
```

Raspunsul este trimis catre client si este procesat. Un exemplu de comunicare client-server in cadrul operatiei de login (se observa ca daca un camp nu este utilizat in select este introdus un null in loc):

```
    std::string header = "user@select";
    std::string buffer = null + null + null + null + email + "\n"
```

```
+ parola + "\n" + null;

char ans[Size];
write(sd, (header + "\n" + buffer).c_str(), strlen((header +
"\n" + buffer).c_str()));

memset(ans, 0, sizeof(ans));
read(sd, ans, sizeof(ans));

if((std::string(ans) + "\n") == null)
{
    ui->errLabel->setText("Email sau parola gresita");
    return;
}
else if(std::string(ans) == "Found")
{
    ui->errLabel->setText("Contul este deja autentificat");
    return;
}

isLogged = 1;
char* p = strtok(ans, "\n");

user.email = email;
while(p)
{
    std::string x(p);
    if(strncmp(p, "username", strlen("username")) == 0)
        user.username = x.substr(x.find(':') + 1);

    if(strncmp(p, "admin", strlen("admin")) == 0)
        user.admin = std::stoi(x.substr(x.find(':') + 1));

    if(strncmp(p, "profil", strlen("profil")) == 0)
        user.profil = std::stoi(x.substr(x.find(':') + 1));

    if(strncmp(p, "nume", strlen("nume")) == 0)
        user.nume = x.substr(x.find(':') + 1);

    if(strncmp(p, "prenume", strlen("prenume")) == 0)
        user.prenume = x.substr(x.find(':') + 1);

    if(strncmp(p, "id", strlen("id")) == 0)
        user.id = std::stol(x.substr(x.find(':') + 1));

    p = strtok(NULL, "\n");
```

}

User este o structura ce contine date despre utilizatorul autentificat in momentul respectiv.

5 Concluzii

Solutia ar putea fi imbunatatita prin folosirea json pentru comunicarea intre client si server datorita structurii utile (ar simplifica lucrul cu datele in comunicare). Alta imbunatatire ar fi prethreading in loc de crearea unui thread de fiecare data cand apare un client. In sectiunea de mesaje, acestea sunt reincarcate odata la 1 secunda lucru nu foarte optim din punct de vedere al timpului.

References

[1] https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php

[2] https://doc.qt.io

[3] https://www.sqlite.org/docs.html