Offline Messenger (B)

Ciripescu Teodor
teodor.ciripescu@info.uaic.ro

Facultatea de Informatica, UAIC

Abstract. Acest document contine planul de implementare pentru proiectul "Offline Messenger".

1 Introducere

Aceasta abordare a proiectului Offline Messenger va avea in prim plan dezvoltarea unor sisteme de: autentificare/inregistrare a utilizatorilor, "prietenie" intre utilizatori, stocare/procesare a conversatiilor dintre utilizatori, notificari din partea serverului in legatura cu anumite evenimente. Vom utiliza un server TCP concurent cu threaduri. Pentru fiecare client va fi folosit un thread ce va fi eliminat odata cu deconectarea clientului de la server.
Pentru stocarea datelor va fi folosita o baza de date sqlite.
Pentru facilitarea de dezvoltare a unui client web, va fi prezenta si o abordare a raspunsurilor de tip JSON din partea serverului.

2 Autentificarea/Inregistrarea utilizatorilor

La inregistrare clientul isi alege un username (unic) si o parola. Acestea vor fi stocate in baza de date sub tabela users (parola va fi hash-uita).
La autentificare se va cere ca input un username si o parola, daca username-ul exista si daca hash-ul parolei date ca input coincide cu cel prezent in baza de date, clientul este autentificat. Acum are acces la o serie de comenzi specifice utilizatorilor.
In cazul unor impedimente precum: cineva incearca sa se inregistreze cu un username deja existent, parola scrisa la autentificare este gresita etc ..., clientul va fi atentionat de catre server.

3 Sistem de "Prietenie" intre utilizatori

Fiecare utilizator autentificat va putea trimite cereri de prietenie catre alti utilizatori. In cazul in care aceste cereri sunt acceptate, vor putea incepe conversatii intre utilizatorii respectivi.

Cand un client se autentifica vor fi cerute de la server si salvate in memorie listele de prieteni, conversatiile si cererile de prietenie. Aceste liste vor fi modificate pe parcursul rularii in functie de evenimentele petrecute. Stocarea informatiilor este vitala pentru a evita comunicarea nenecesara cu serverul.
Utilizatorii vor putea sa isi vada lista de prieteni, sa trimita cereri de prietenie, sa isi vada lista cererilor de prietenie primite si sa accepte sau sa refuze cererile de prietenie.

4 Procesarea/Stocarea Conversatiilor

Un client autentificat poate trimite altor clienti din lista sa de prieteni mesaje. Aceste mesaje vor avea asociate in baza de date campuri precum time, seen si reply_to pentru a determina cand a fost trimis un mesaj, daca a fost vazut destinatar si daca reprezinta o replica la alt mesaj. Mesajele vor avea optiunea de a fi sterse (atat pentru o singura persoana cat si pentru ambele).

5 Notificari

Utilizatorii vor fi notificati de catre server cand au loc anumite evenimente: mesaj nou primit, cerere noua de prietenie. De asemenea, in momentul autentificarii, clientii vor fi atentionati de mesajele si cererile de prietenie pe care le-au primit pe perioada in care au fost offline.

6 Protocol

Fiecare client va avea asociat threadul lui pe server si o structura in care va fi memorat descriptorul si ID-ul de utilizator obtinut la autentificare. Threadul va inceta sa mai existe in momentul in care clientul se deconecteaza de la server.
Mai intai se vor trimite 2 bytes pentru a semnala lungimea comenzii.
Apoi va fi trimisa comanda propriu-zisa, formata din 1 byte pentru codul comenzii (comenzile vor fi indexate printr-un numar), iar restul de bytes vor fi folositi pentru lungimea parametrilor si parametri.
Raspunsul primit de client va fi format asemanator, 2 bytes pentru lungimea raspunsului, 1 byte pentru comanda folosita, iar restul pentru mesajul propriu-zis.
Va fi folosit un thread nou in care clientul asteapta mesaje noi de la server. Astfel va fi evitat un blocaj in care incercam sa scriem si se citim in acelasi timp.
Am ales folosirea threadurilor pentru ca sunt mai eficiente, mai lightweight decat o abordare cu procese copil. Mai mult, natura proiectului implica prezenta multor clienti conectati la server in acelasi timp, cat si citiri/scrieri constante la server. Taskurile date serverului sunt in general simple si multe, deci se preteaza

mai bine pe modelul cu threaduri.

```
//server_boilerplate.h
struct thData{
    int idUser; //user's id extracted from the DB at authentication
    int descriptor; //descriptor returned by accept
};

class server_boilerplate {
     static void *treatClient(void *arg);
     static void addThreadData(void *arg);
     static void talkToClient(void *arg);
public:
    int createServer(int PORT);

    static char *handleCommand(short len, char *command, thData *thread_data);
```

```
while(1){
    if (read (thread_data.descriptor, &command_len, sizeof(short int)) <= 0)
    {
        printf("[User %d]\n", thread_data.idUser);
        perror ("Error at reading command length from client.\n");

    }
    printf ("[User %d]Command length = %d.\n", thread_data.idUser, command_len);


    if (read (thread_data.descriptor, &command, command_len + 1) <= 0)
    {
        printf("[User %d]\n", thread_data.idUser);perror ("Error at reading command from client.\n");

    }
    handleCommand(command_len, command, &thread_data);
```

Comenzi si evenimente ce au loc in urma executarii lor de catre utilizatori

Fiecare client ce se conecteaza la server poate folosi una dintre urmatoarele comenzi:
-register(username, password) → Daca a avut succes sau nu;
-authenticate(username, password) → daca are loc cu succes, clientul capata titlul de utilizator si va avea drepturi extra

Fiecare utilizator va putea folosi urmatoarele comenzi:
getFriendList() → Lista cu ID-urile prietenilor;
sendFriendRequest(username) → cererea (nu) a fost trimisa;
getFriendRequestList() → Lista cu ID-urile cererilor care nu au fost solutionate;
acceptFriendRequest(request_id);
rejectFriendRequest(request_id);
getConversations() → Lista cu ID-urile conversatiilor;
getMessages(id_conversation, messages_amount) → ultimele *messages_amount* din conversatia *id_conversation*;
getAllMessages(params: id_coversation) →toate mesajele din converastia *id_conversation*;
sendMessage(id_conversation, content) → mesajul (nu) a fost trimis;
replyMessage(id_conversation, id_message_toReply, content) → mesajul (nu) a

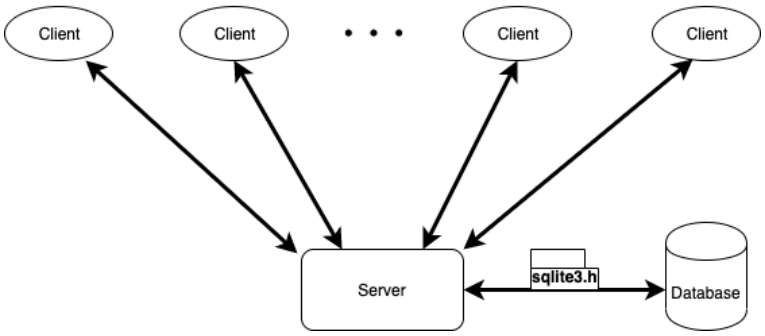
fost trimis;
deleteMessage(id_message, for_me/for_all) → mesajul (nu) a fost sters;
logout();

Exemplu:

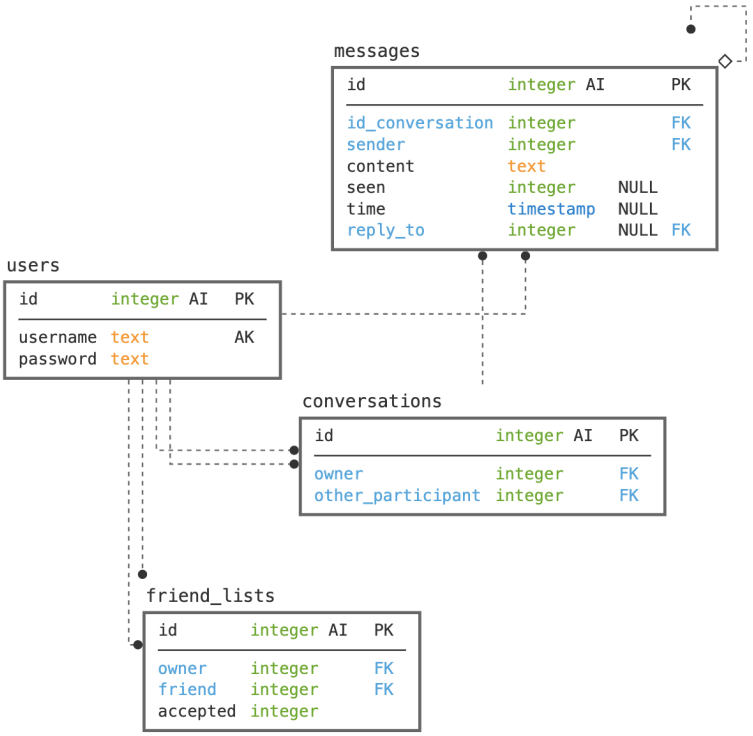
```
char *server_boilerplate::handleCommand(short len, char *command, thData *thread_data) {
    int command_id = (int)command[0];
    if(command_id==1){ //AUTH
        int username_len = (int)command[1];
        int password_len = (int)command[2];
        char *username = new char[username_len+2];
        char *password = new char[password_len+2];
        memcpy(username, src: &command[3], username_len);
        username[username_len] = '\0';
        memcpy(password, src: &command[3+username_len], password_len);
        password[password_len] = '\0';
        std::cout<<"COMMAND: AUTH("<<"username: "<<username<< ", password: "<<
                        password<<") descript: "<<thread_data->descriptor<<"\n";

        bool authenticated = dc.auth(username, password, &thread_data->idUser, &thread_data->descriptor);
        if(authenticated){
            addThreadData(thread_data);
            std::cout<<"Authenticated! userId = "<<thread_data->idUser<<std::endl;
        }
        delete[] username;
        delete[] password;
    }
```

Evenimente ce se petrec automat
Serverul va notifica in momentul autentificarii pe utilizator in legatura cu: conversatii ce contin mesaje noi/necitite, noi cereri de prietenie.
Mesajele cu statutul de "necitit" vor avea statutul de "citit" daca acestea sunt continute in raspunsul intors de functiile "getMessages" sau "getAllMessages".
Utilizatorii vor fi notificati de noi cereri de prietenie si noi mesaje.
Daca 2 utilizatori nu sunt "prieteni" nu va fi posibila crearea unei conversatii intre cei 2.

7 Baza de date



Vom folosi o baza de date sqlite pe care o vom accesa prin intermediul bibliotecii sqlite3.h . Toate apelurile catre baza de date vor fi facute de catre server in urma comenzilor date de catre clienti.
Baza de date va respecta urmatoarea schema:



In lista de prieteni va fi inregistrata prietenia din ambele sensuri (poti sterge pe cineva ca prieten si el sa te aiba in continuare in lista);'
La fel si la conversatii, se poate pastra o conversatie cu cineva dar acel cineva sa nu o pastreze la randul sau.
Si mai mult, pe acelasi principiu se vor baza si mesajele, va fi o copie amesajului

pentrut fiecare conversatie(respectiv utilizator) in parte. Astfel, va fi posibila stergerea mesajelor pentru fiecare client, fara sa fie nevoie sa se stearga mesajul si pt cealalta persoana.
Procedura de interogare a bazei de date pentru autentificare:

```
bool database_commands::auth(const char *username, const char *password, int* idUser, int* descriptor) {
     std::string sqlstatement = "SELECT id FROM users WHERE username='"+(string)username+"' AND password='"+(string)
  password+"'";
     p=idUser;
     d.executeSQLstatement(sqlstatement.c_str(), &auth_callback);
    return *idUser > 0;
}

int database_commands::auth_callback(void *data, int argc, char **argv, char **azColName) {
    if(argv[0])
        *p=atoi(argv[0]);
    return 0;
}
```

8 Concluzii si posibile aditii

Luand in considerare cele de mai sus putem intelege ca orice request al unui client catre server are urmatorulu drum. In prima instanta, clientul se conecteaza la server si un thread ii este alocat pentru managementul requesturilor. Detalii referitoare la client vor fi retinute intr-o structura de tipul thData (descriptorul si id-ul care initial este -1 din lipsa de autentificare). Acum singurele comenzi ce pot fi facute de catre client sunt register si authenticate. In urma unei autentificari cu succes (ce implica o interogare a bazei de date cu parametrii username si password), campului idUser din structura thData ii este atribuita valoarea id-ului asociat utilizatorului din tabela users a bazei de date. Clientul capata mai multe drepturi, si anume accesul la mai multe comenzi(prezentate mai sus). Proaspat autentificat, utilizatorul este notificat de ce a ratat in perioada in care a fost offline (mesaje noi, cereri de prieteni). Noi cereri de prietenie si mesaje noi vor aparea in continuare pe parcursul rularii.
In momentul in care un client doreste sa trimita un mesaj altui client, acesta va folosi una dintre functiile de trimitere mesaj catre server. Serverul va cauta in vectorul de structuri thData descriptorul destinatarului si ii va transmite mesajul prin acesta.

```
Opened database successfully...
[server]Waiting at port 2908...
[thread]- -1 - New thread, waiting for client...
[User -1]Command length = 15.
COMMAND: AUTH(username: teodor, password: parola) descript: 5
Query executed successfully!
Authenticated! userId = 1
```

O abordare a notificarilor poate fi crearea unui nou thread care sa sa astepte mereu un mesaje de la server.
Poate fi implementata si o solutie de criptare a mesajelor. Va trebui folosita o

metoda de a securiza schimbul de chei (Diffie Hellman key exchange).

References

1. https://www.sqlite.org/whentouse.html
2. https://www.sqlite.org/capi3ref.html
3. https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
4. https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c
5. https://www.geeksforgeeks.org/sql-using-c-c-and-sqlite/