

Offline Messenger

Rusu Daniela, 2A1

Facultatea de Informatică, Universitatea Alexandru Ioan Cuza, Iași

1 Introducere

Proiectul *Offline Messenger* este o aplicație client/server cu interfața grafică realizată folosind framework-ul Qt Creator, care permite schimbul de mesaje între utilizatori, cu următoarele opțiuni:

- Înregistrarea/Logarea utilizatorilor în cadrul aplicației, cu posibilitatea de a afișa istoricul conversațiilor cu fiecare alt utilizator
- Trimiterea mesajelor între utilizatori online și offline
- Posibilitatea de a răspunde la un anumit mesaj (*reply*) și de a vedea istoricul conversațiilor

2 Tehnologii Aplicate

În cadrul acestei aplicații, voi utiliza următoarele tehnologii:

- un model client/server TCP/IP concurent, bazat pe multiplexare, pentru asigurarea conectării a mai multor clienți simultan și pentru monitorizarea celui care trimite cereri într-un moment dat, fără blocarea procesului. Avantaje:
 - gestionarea concurenței la resursele partajate (baza de date).
 - simplificarea și claritatea codului prin furnizarea unui mecanism centralizat pentru coordonarea eficientă a operațiunilor I/O (multiplexare).
 - asigurarea livrării fiabile și ordonate a mesajelor conform protocolului TCP.
- o bază de date MySQL care stochează informații despre Utilizatori (tabelul **User**) și informații despre mesaje (tabelul **Messages**)
- interfața grafică bazată pe framework-ul Qt Creator

3 Structura Aplicației

(**Fig. 1**) Aplicația realizează conexiunea dintre client și server prin intermediul unui *socket*. Prin intermediul funcției *select()* în server, actualizăm mereu lista descriptorilor de clienți pregătiți pentru interacțiune și apoi monitorizăm clasa de descriptori *readfds*, așteptând cel puțin un descriptor să fie în starea *ready* pentru citire. După preluarea datelor din descriptorul *ready*, serverul procesează

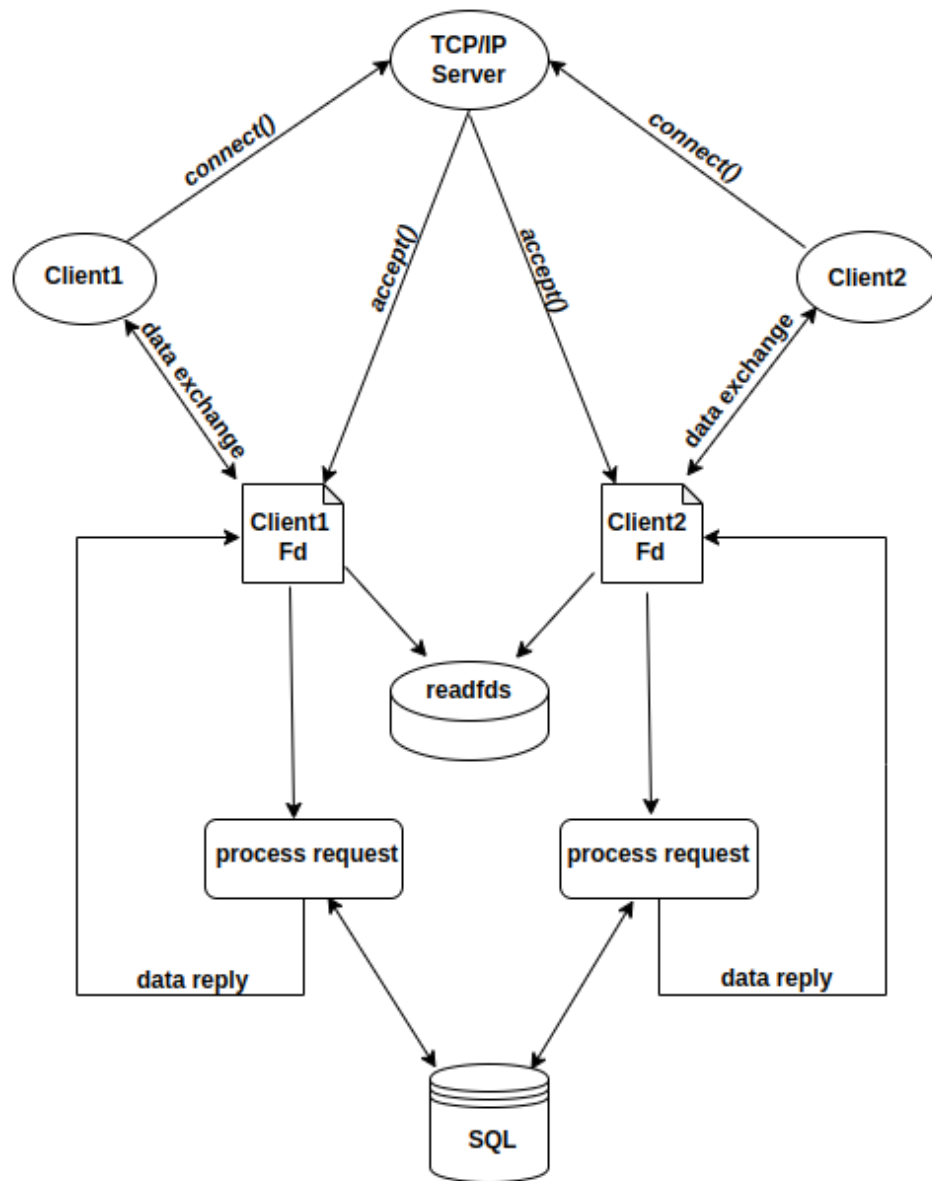


Figura 1. Diagrama aplicației client/server TCP concurent cu multiplexare

informația, făcând legătura cu baza de date, și apoi scrie înapoi în descriptor rezultatul.

(**Fig. 2**) Tabelele folosite în baza de date sunt **User** și **Messages**. Diagrama UML descrie că un *utilizator* poate trimite 0 sau mai multe mesaje, iar un mesaj poate fi trimis de către un *utilizator*. Atributul *timestamp* din tabelul **Messages**, descrie data și ora la care a fost creat mesajul, iar *ancestor_message*, mesajul la care s-a dat reply.

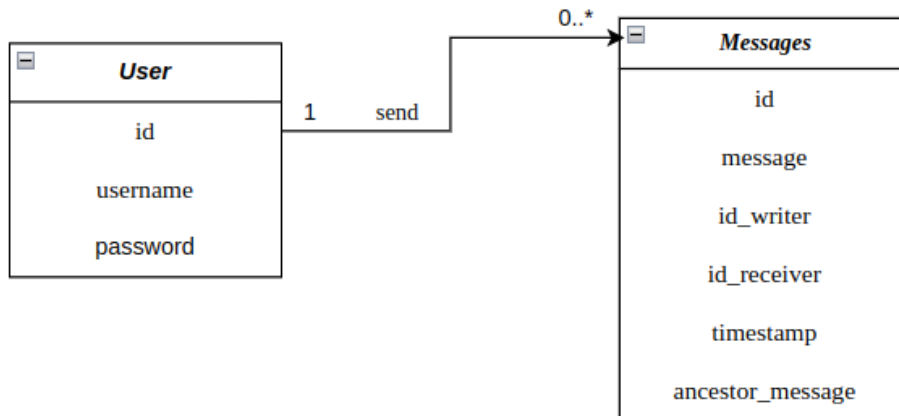


Figura 2. Diagrama UML a bazei de date

Fig. 3 descrie funcționalitățile sistemului din perspectiva utilizatorilor. Diagrama Use Case evidențiază situațiile în care utilizatorii interacționează cu sistemul pentru a realiza anumite comenzi.

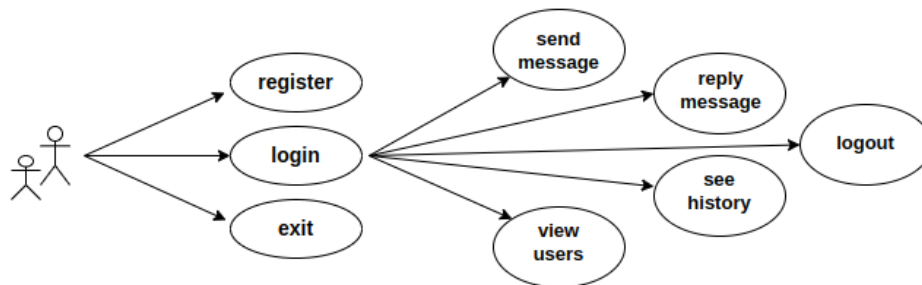


Figura 3. Diagrama Use Case

4 Aspecte de Implementare

4.1 Implementarea clientului

Clientul și serverul partajează același socket. În cadrul acestei interacțiuni, clientul (**main.cpp**), transmite toate cererile către server prin intermediul funcției *send()*. Prin intermediul unui *while()*, clientul verifică variabila globală **MessageToServer**, care este actualizată la fiecare introducere de date din partea utilizatorului și la efectuarea unei cereri (login, sign up, reply, send, logout). Această abordare asigură un flux de informații eficient și un răspuns coordonat la cererile utilizatorului. Toată funcționalitatea clientului este implementată în mai multe fișiere care alcătuiesc câte o fereastră specifică aplicației.

```
while(1){
    ...
    if (send(sd,MessageToServer,strlen(MessageToServer),0)<=0){
        perror("[CLIENT]Eroare la send() spre server.\n");
        return;
    }
    ...
    if (-1==(Read_Code=recv(sd,MessageFromServer,
                           sizeof(MessageFromServer),0)))

    ...
}
}
```

Un aspect important în aplicație este primirea mesajelor în timp real între 2 utilizatori online, logica care este implementată în fișierele **chatwindowdialog.cpp** și **chatmanager.cpp**. După ce sunt afișate mesajele din istoric în constructorul *ChatWindowDialog*, acesta creează o instanță nouă a clasei *ChatManager*, care creează un thread separat ce verifică în mod constant (interval de 2 secunde) prin interacțiunea cu serverul dacă există mesaje noi, iar în caz afirmativ, le afișează pe ecran la momentul respectiv. Threadul se oprește în momentul în care fereastra cu chatul este închisă de către utilizator.

```
void ChatManager::run(){
    sleep(2);
    do {
        QMutex mutex;
        mutex.lock();
        if(this->Stop) break;
        mutex.unlock();
        //get all the new messages in real time
        ....
        emit aNewMessage(Message);
        this->sleep(2);
    }while(!Stop);
}
```

4.2 Implementarea serverului

Din diverse motive, un client poate trimite datele foarte încet, prin urmare, procesul server ar rămâne blocat așteptând date de la clientul respectiv. Astfel, serverul folosește multiplexarea pentru procesarea requesturilor de la clienți.

În bucla *while*, actualizăm mereu lista de descriptori activi (*actfds*) în clasa de descriptori de citire *readfds*. Apelul *select()* așteaptă în interval de maxim 1 secundă ca măcar unul dintre descriptorii din setul *readfds* să devină gata pentru citire. Acesta este un mod eficient de a gestiona comunicarea cu clienții fără a bloca execuția. În blocul *FD_ISSET(sd, &readfds)* realizăm o conexiune nouă cu un client, prin funcția *accept()*, și se scrie un mesaj de bun-venit către noul client. Evident, descriptorul acestuia este adăugat în mulțimea *actfds*.

Apoi serverul iterează prin toți descriptorii conectați și când detectează unul pregătit pentru citire, îl procesează prin intermediul funcției *Message_Work()*.

```
FD_SET(sd, &actfds); //setam socketul in actfds(descriptori activi)
while(1) {
    bcopy((char *)&actfds, (char *)&readfds, sizeof(readfds));
    if (select(nfds + 1, &readfds, NULL, NULL, &timev) < 0)
        error("[SERVER] Error select\n", errno);
    if (FD_ISSET(sd, &readfds)) {
        ...
        client = accept(sd, (struct sockaddr *)&from, &len);
        //write welcome message to new client
        ...
        FD_SET(client, &actfds);
    }
    for (fd = 0; fd <= nfds; fd++) {
        if (fd != sd && FD_ISSET(fd, &readfds)) {
            if (Message_Work(fd) == 0) { //client disconnected }
        }
    }
}
```

Pentru gestionarea informațiilor despre utilizatori și mesaje, am optat pentru o bază de date în loc de fișiere simple, deoarece baza de date oferă o organizare structurată a informațiilor și manipulare mai ușoară a datelor prin intermediul interogărilor. Conexiunea cu baza de date se face în cadrul constructorului clasei *UserDataAccess* și al clasei *MessagesHandler*, unde se gestionează informațiile din tabelul **User** și respectiv, tabelul **Messages**.

```
connection = mysql_init(NULL);
if (!mysql_real_connect(connection, "localhost", "daniela",
    "123450", "offline_messenger", 0, NULL, 0)) {
    fprintf(stderr, "ERROR %s\n", mysql_error(connection));
    exit(1);
}
```

4.3 Scenarii de utilizare

Scenariul 1. Utilizatorul dorește să se logheze în aplicație, astfel clientul preia datele (**username** și **password**) și face un request spre server sub forma "LOGIN:username:password". Serverul apelează funcția *login*(*Username*, *Password*) din clasa *UserHandler*, care verifică dacă clientul există în baza de date și returnează clientului un mesaj relevant în urma logării. (**Fig.4**)

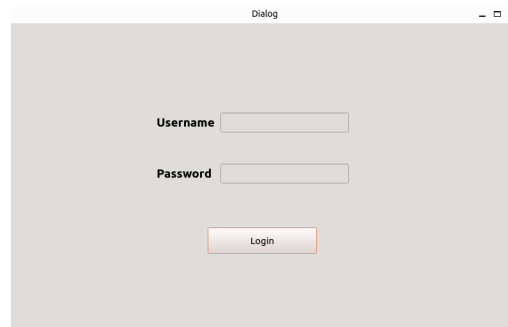
A screenshot of a software dialog box titled "Dialog". It contains two text input fields: the first is labeled "Username" and the second is labeled "Password". Below these fields is a single button labeled "Login". The dialog box has a standard Windows-style title bar with minimize, maximize, and close buttons.

Figura 4. Fereastra de logare a utilizatorului

Scenariul 2. Utilizatorul dorește să se înregistreze în aplicație, făcând click pe butonul **Sign up**, astfel clientul preia datele și face un request sub forma "REGISTER:<username>:<password>:<verification password>". Serverul apelează *signup*(*Username*, *Password*) din clasa *UserHandler*, care inserează în baza de date user-ul nou(cu verificările necesare) și returnează clientului un mesaj prin care confirmă sau nu înregistrarea lui cu succes.(**Fig.5**)

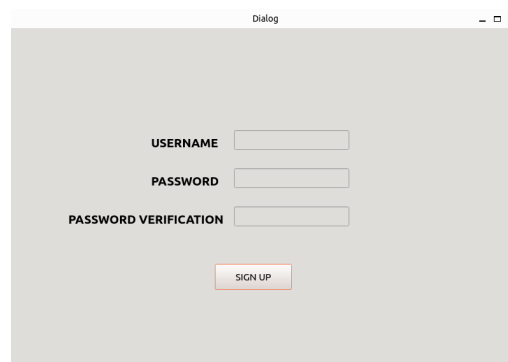
A screenshot of a software dialog box titled "Dialog". It contains three text input fields: the first is labeled "USERNAME", the second is labeled "PASSWORD", and the third is labeled "PASSWORD VERIFICATION". Below these fields is a single button labeled "SIGN UP". The dialog box has a standard Windows-style title bar with minimize, maximize, and close buttons.

Figura 5. Fereastra de înregistrare a unui nou utilizator

Scenariul 3. După logarea cu succes, clientul face o cerere către server de forma "LIST_ALL_USERS", serverul apelează funcția *getAllUsers()* din clasa *UserHandler*, trimite înapoi clientului răspuns, iar clientul afișează o fereastră cu toți utilizatorii înregistrați în aplicație, permițându-i utilizatorului să aleagă chat-ul cu un anumit user.

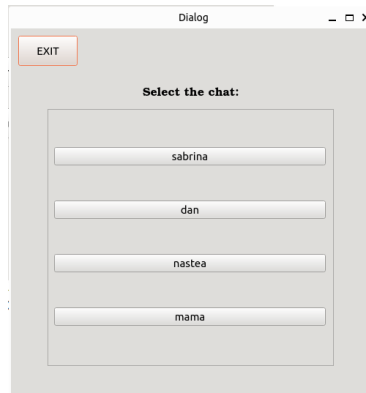


Figura 6. Fereastra de afișare a tuturor utilizatorilor

Scenariul 4. În fereastra chat, clientul face un request spre server de forma "HISTORY:<User_Current>:<To_User>", iar serverul apelează funcția *getChatMessages(UserCurr, ToUser)* din clasa *MessagesHandler*, care preia toate mesajele din baza de date dintre cei 2 utilizatori. Mesajele vor fi afișate pe ecran.

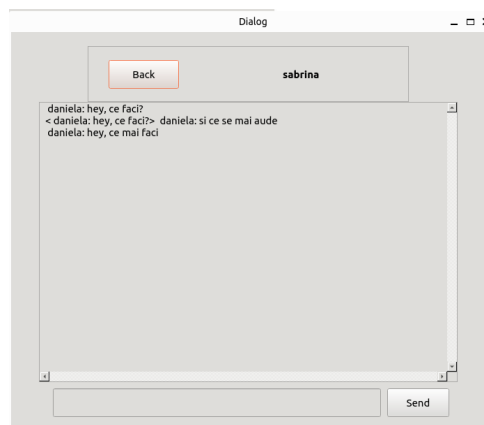


Figura 7. Fereastra de chat a utilizatorului

Scenariul 5. Dacă utilizatorul vrea să dea reply la un anumit mesaj din chat, trebuie să dea click pe acel mesaj, clientul va apela funcția *ChatWindowDialog::on_MessageList_itemClicked*, care actualizează într-o variabilă globală **AncestorMessage**, mesajul la care s-a dat click. Apoi utilizatorul introduce mesajul dorit, dă click *Send* și face un request spre server sub forma "SEND/message/UserCurrent/ToUser/ReplyStatus/PreviousMessage". Mesajul va fi afișat pe ecran astfel:

Figura 8. Exemplu de afișare a unui mesaj cu reply

5 Concluzii

Proiectul ar putea fi îmbunătățit prin dezvoltarea structurii grafice în ceea ce privește partea de comunicare (adăugarea emoji-urilor, reacții la mesaje). Astfel, s-ar îmbunătăți experiența unui utilizator cu aplicația.

Bibliografie

1. Rețele de Calculatoare, pagina cursului
<https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php>
2. Rețele de Calculatoare, pagina laboratorului
<https://www.andreis.ro/teaching/computer-networks>
3. MySQL C API programming
https://zetcode.com/db/mysqlc/?utm_content=cmp-true
4. Diagramming Application
<https://app.diagrams.net/>
5. QT Creator tutorial
https://www.youtube.com/watch?v=g6fw5n9Gt-E&list=PL010_mIqDDFUaZe7H9kY6vWbSVrtwFv4M
6. QT Creator documentation
<https://doc.qt.io/>