Documentație proiect Offline Messenger

# Pîrlac Andreea-Diana, grupa A5

# Introducere

Proiectul „Offline Messenger” are scopul de a dezvolta o aplicație de tip client/server care permite schimbul de mesaje între utilizatori care își creează sau dețin un cont în aceasta, fiind utila pentru oricine are nevoie și vrea sa comunice cu alți useri. Se va putea face acest lucru cat timp utilizatorii sunt conectați la server, dar se oferă funcționalitatea trimiterii mesajelor si către utilizatorii offline, acestora din urma apărându-le mesajele atunci când se vor conecta din nou la server. De asemenea, utilizatorii vor avea posibilitatea de a trimite un răspuns (reply) in mod specific la anumite mesaje primite. Aplicația va permite si vizualizarea istoricul conversațiilor pentru si cu fiecare utilizator in parte.

# Tehnologii Aplicate:

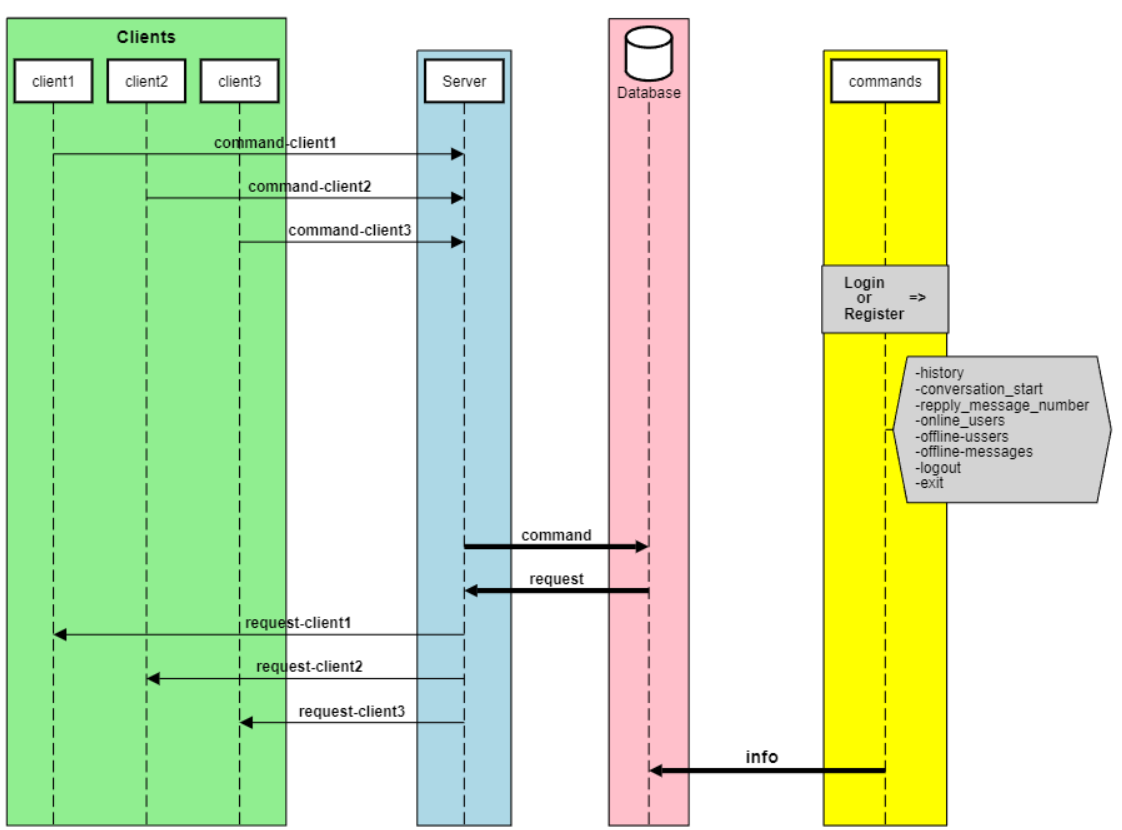
Protocolul de comunicare ales la acesta aplicație este TCP (”Transmission Control Protocol”). TCP este un protocol orientat pe conexiuni folosit de aplicații care au nevoie de confirmare la primire a datelor, sa ajungă fără erori de la sursa la destinație (in cazul nostru, de la client la server). Efectuează o conectare virtuala full duplex intre doua puncte terminale, fiecare punct fiind definit de către o adresa IP si de către un port TCP. Acest protocol detectează probleme precum pierderea pachetelor, duplicarea sau livrarea acestora într-o alta ordine si solicita retransmiterea pachetelor pierdute, rearanjează pachetele in ordine si ajuta la minimizarea traficului din rețea in vederea reducerii apariției altor probleme. Așadar, TCP este un protocol ideal pentru aceasta aplicație, fiind un serviciu reliable care garantează livrarea unui flux de date trimis de la un client la altul, spre deosebire de UDP unde datele pot ajunge la destinație incomplete, neordonate sau chiar duplicate din cauza congestiilor din rețea sau prin încărcarea traficului.

Astfel vom folosi un server TCP concurent care deservește clienții prin crearea unui thread pentru fiecare client. Thread-urile sunt procese ce rulează in paralel, in comparație cu fork care creează o copie a procesului. Multithreading-ul este capacitatea unui program sau a unui sistem de operare de a permite existenta mai multor utilizatori simultan, fără a necesita mai multe copii ale programului care rulează pe computer, in timp ce fork-urile sunt procese independente. Fiecare cerere de utilizator pentru un program sau serviciu de sistem este urmărita ca un fir de execuție cu o identitate separata.

Pentru stocarea datelor (nume de utilizatori, parole, conversație, mesaje etc.) se folosește SQLite. SQLite este un sistem popular de gestionare a bazelor de date ce are ca avantaje: scrierea si citirea sunt mai eficiente într-o baza de date decât folosirea unor fișiere aflate pe disk, modul eficient de stocare a datelor in tabele si conținutul poate fi accesat si actualizat prin interogari SQL.

# Structura Aplicației:

Arhitectura aplicației este prezentata detaliat in diagrama de mai jos.



După cum se poate vedea in diagrama, in implementarea aplicatiei, am realizat conexiunile dintre clienți folosind un server concurent, deoarece o aplicației de tip messenger pune la dispoziție conectarea mai multor clienți in același timp. Astfel, daca client1 va trimite un mesaj către client2 ,dar si client3 va trimite mesaj tot la client2, atunci atât si client1, cat si client3 vor trimite mesajele in același timp, fără ca unul dintre clienți sa aștepte. Comunicare intre clienți si conectarea la server se va face cu ajutorul unor funcții(comenzi) ce își vor lua datele necesare din baza de date:

### login & register

Dacă utilizatorul deține deja un cont, nu-i rămâne decât sa introducă numele de utilizator si parola pentru a se conecta. Altfel, utilizatorii vor fi nevoiți să-și creeze propriul cont înainte pentru a putea folosi aplicația, acest lucru constă in setarea unui username si a unei parolei.

### history

Această comandă permite vizualizarea mesajelor trimise anterior dintre doi clienți, va extrage din baza de date informațiile necesare pentru a putea fi revizualizate.

### conversation\_start & repply\_message\_number

Utilizatorul va începe sesiunea de mesaje dintre el si alți clienți, iar clientul va putea răspunde explicit la unul dintre mesajele primite anterior folosindu-se de un număr, ce reprezintă al câtelea mesaj este in conversație de când a început sesiunea.

### online\_users

Se vor afișa utilizatorii conectați in acel moment la server, informații luate dintr-un fișier .txt (unde vor fi trecuți utilizatori când se vor conecta).

### offline\_users & offline\_messages

Afișarea utilizatorilor deconectați de la server la acel moment se va face prin folosirea funcției offline\_users, informații luate tot dintr-un fișier .txt . Mesajele primite atunci când userul a fost deconectat de la server vor fi afișate in momentul in care se va reconecta, utilizând offline\_messages.

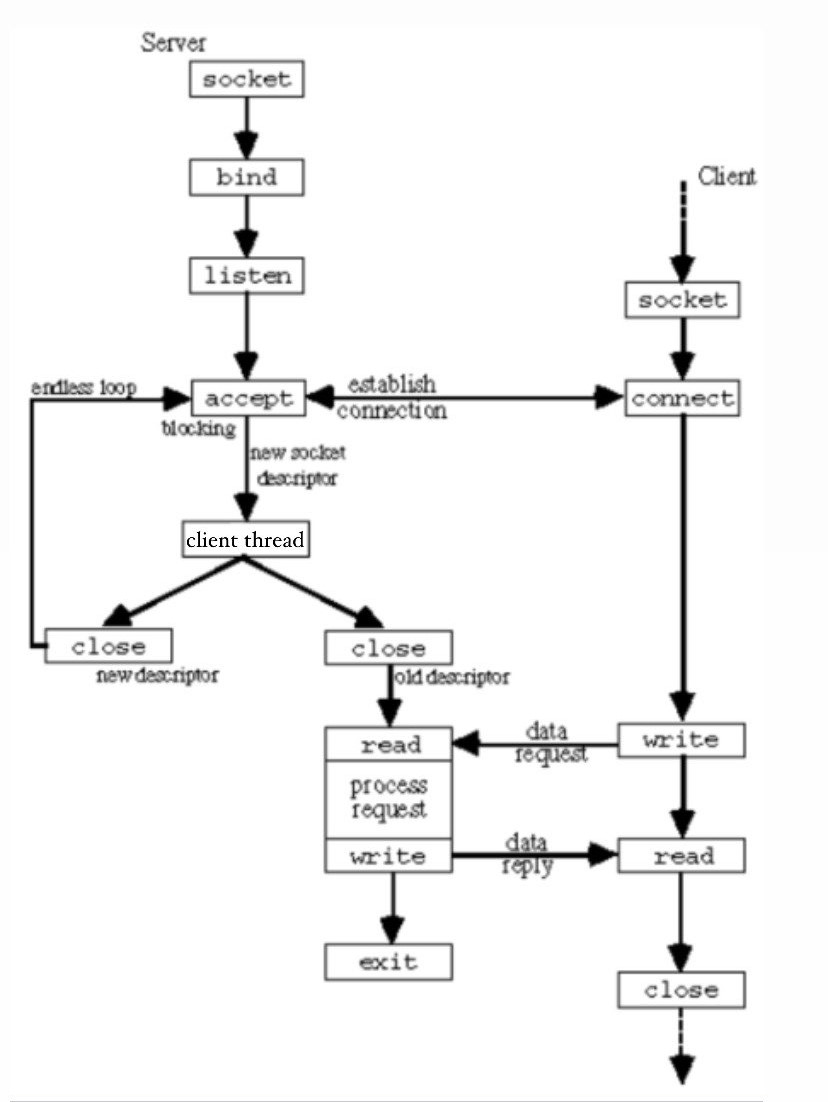
### logout & exit

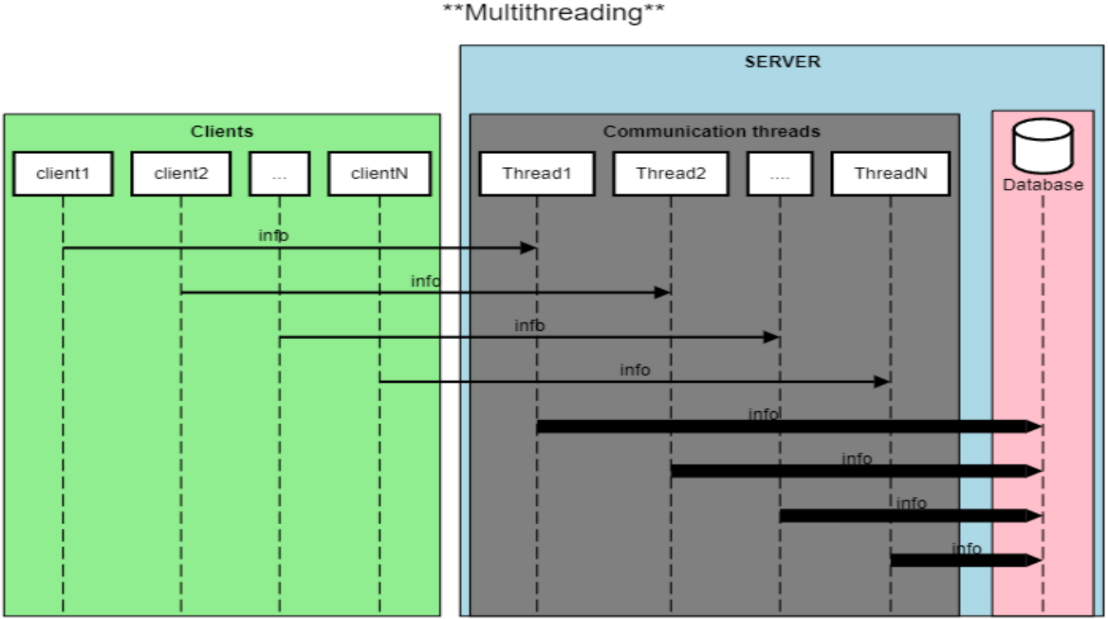
Utilizatorul daca va folosi comanda logout, se va deconecta de la server, dar daca va folosi exit, aplicația se va închide.

# Aspecte de Implementare:

## Implementare Client/Server (explicatii):

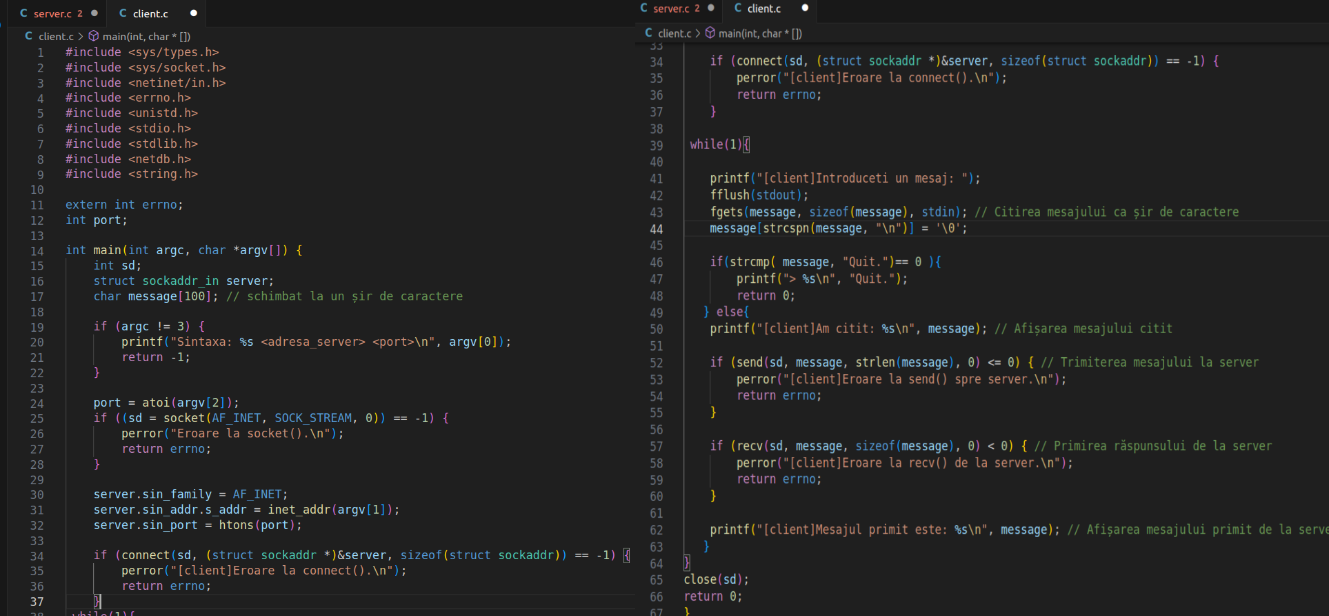
Se va face conexiunea intre un client TCP si un server TCP concurent care deservește clienții prin crearea unui thread pentru fiecare client. Clientul creează un socket , îi aloca o zona de memorie, apoi începe conectarea la IP-ul si PORT-ul la care poate fi găsit serverul. Serverul creează un socket care va fi folosit in continuare pentru a face bind(), pentru a-i asocia un port prestabilit oferind anumite servicii la acel port. Serverul este pregătit să accepte conexiuni de la clienți (listen()). Thread-ul principal este blocat la apelul lui accept() si de fiecare data când este acceptat un client se creează un thread care îl va servi. S-a realizat conexiunea între server si client, iar comunicarea efectivă se va face prin read()/write()/close().



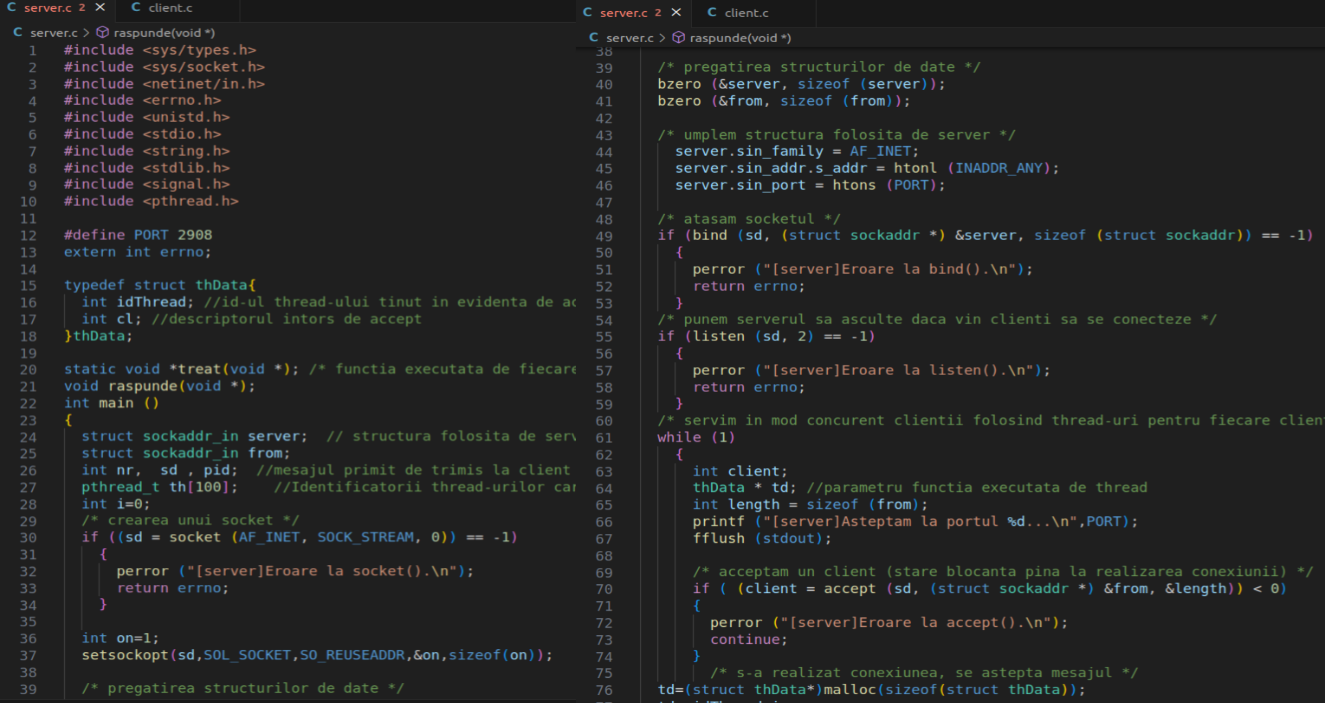


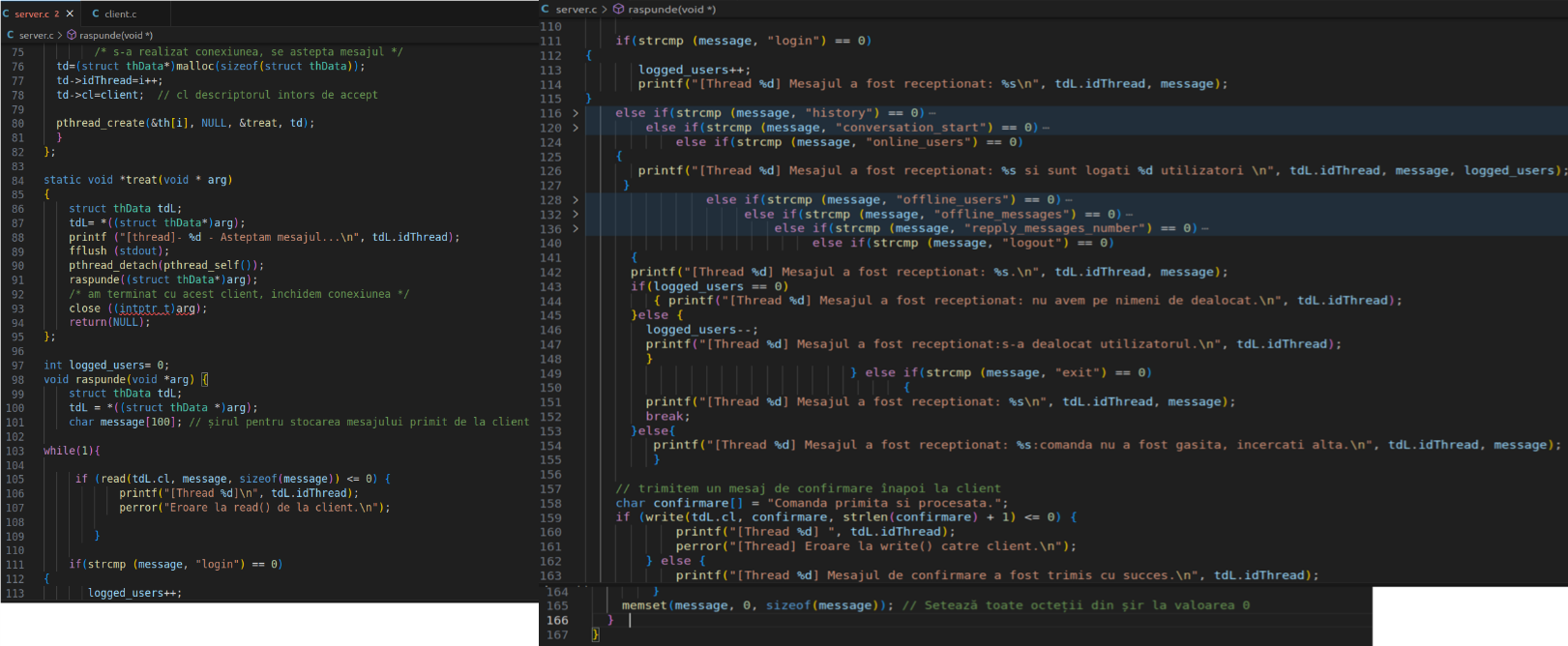
## Implementare Client/Server (cod):

* client.c



* server.c





## Implementarea bazei de date (explicații):

Baza de date facilitează stocarea conversațiilor între utilizatori, oferind garanția ca datele nu se vor pierde dacă serverul se va întrerupe. Aceasta este utilă atunci când apelăm funcția ”history”, deoarece ne permite vizualizarea istoricul conversațiilor pentru și cu fiecare utilizator în parte. Funcția afișează pe ecran liniile specifice ID-ului cu toate înregistrările ce reprezintă de fapt, mesaje pe care le-a trimis, dar și mesajele pe care acesta le-a primit de la alți utilizatori. Tabelele ce se regăsesc in baza de date a server-ului sunt următoarele:

**Tabelul "Utilizatori"**:

|  |  |  |  |
| --- | --- | --- | --- |
| ID\_Utilizator | Nume\_Utilizator | Email | Parola |
| 1 | User1 | [User1@yahoo.com](mailto:User1@yahoo.com) | Parola1 |
| 2 | User2 | [User2@yahoo.com](mailto:User2@yahoo.com) | Parola2 |
| 3 | User3 | [User3@yahoo.com](mailto:User3@yahoo.com) | Parola3 |
| 4 | User4 | [User4@yahoo.com](mailto:User4@yahoo.com) | Parola4 |

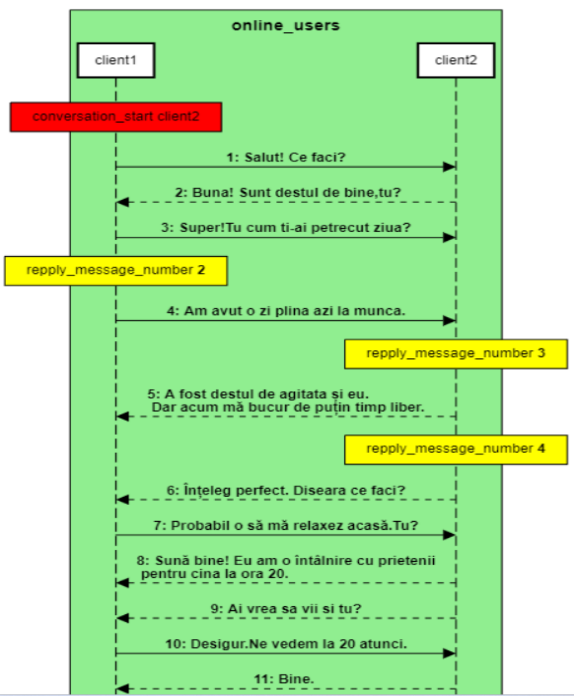
**Tabelul "Conversatii"**

|  |  |  |
| --- | --- | --- |
| ID\_Conversatie | ID\_Utilizator\_1 | ID\_Utilizator\_2 |
| 1 | **1** | **2** |
| 2 | **3** | **4** |

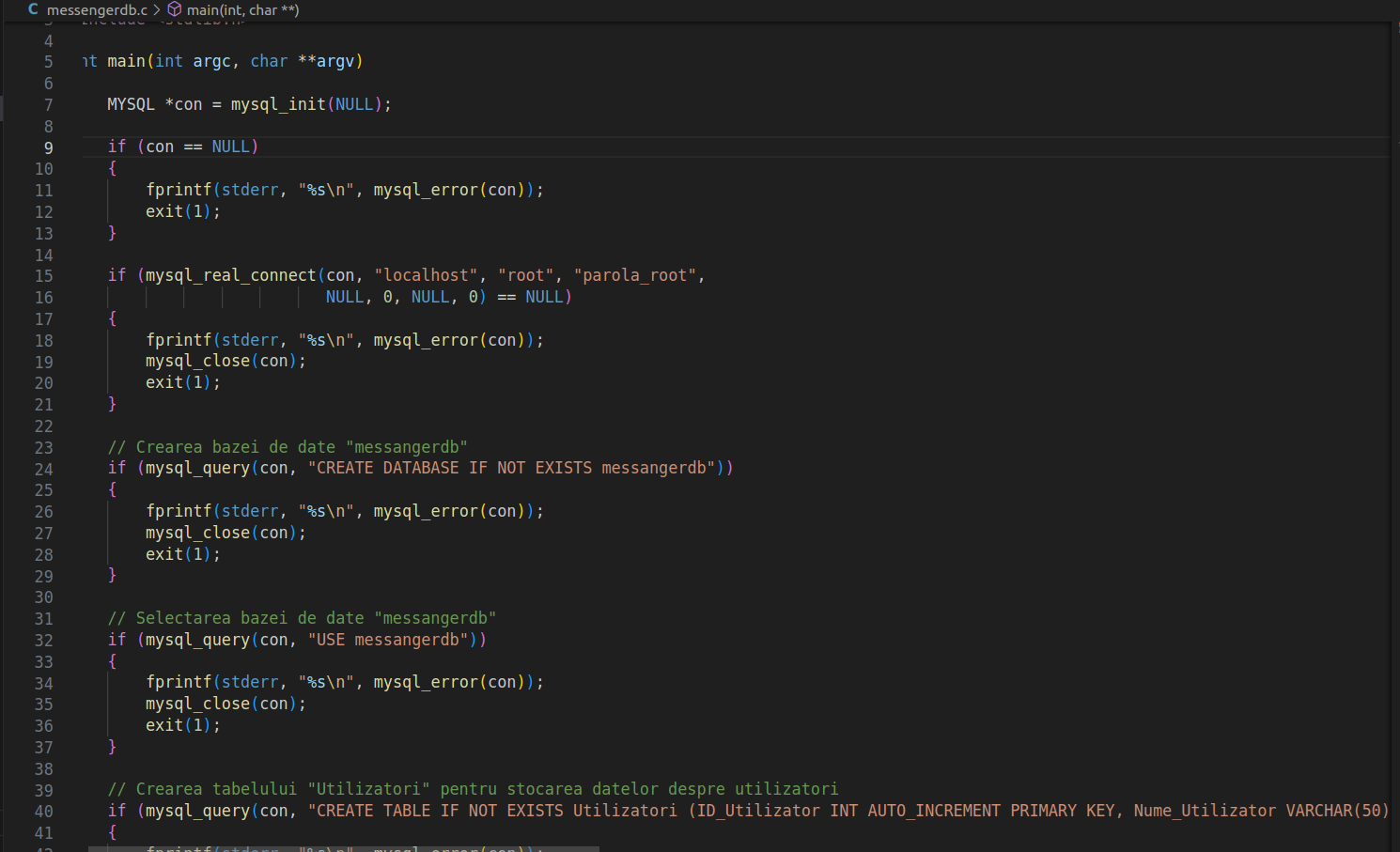
**Tabelul "Mesaje"**:

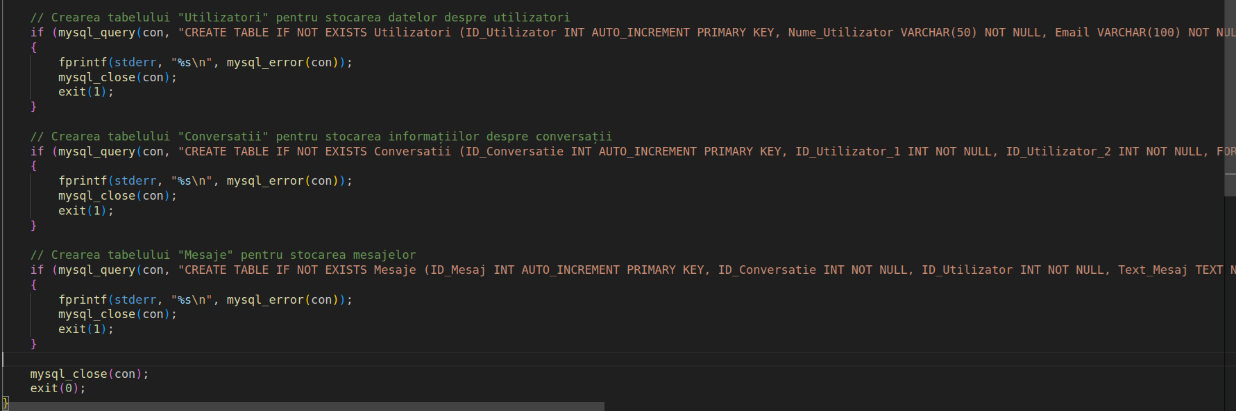
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID\_Mesaj | ID\_Conversatie | ID\_Utilizator | Text\_Mesaj | Data\_ora |
| 1 | 1 | 1 | Salut!Ce faci? | 09/12/2023 17:00 |
| 2 | 1 | 2 | Buna!Sunt destul de bine, tu? | 09/12/2023 17:05 |
| 3 | 1 | 1 | Super!Tu cum ti-ai petrecut ziua? | 09/12/2023 17:44 |
| 4 | 1 | 1 | Am avut o zi plina azi la munca. | 09/12/2023 18:30 |

Observăm din următoarea simulare a unei conversații între doi utilizatori modul în care se va realiza comunicarea. Mesajele au fost indexate cu numere de la 1 la 11 , această numerotare este realizată pentru momentul în care unul dintre utilizatori dorește să răspundă în mod explicit la unul dintre mesaje, prin funcția reply\_message\_number urmat de numărul mesajului. În exemplu, client1 răspunde explicit la mesajul 2 al client2. Această numerotare se realizează automat în momentul în care a fost apelată funcția conversation\_start ,dar indexarea se va șterge când este apelată comanda ”exit” sau ”logout”.



## Implementarea bazei de date (cod)





Codul furnizat va crea o bază de date numită messangerdb.c și trei tabele: "Utilizatori", "Conversatii" și "Mesaje". Aceste tabele sunt create cu coloane corespunzătoare pentru a stoca informațiile despre utilizatori, conversații și mesaje în cadrul aplicației. Pentru crearea bazei de date in limbajul C++ se va folosi biblioteca ”< mysql.h>”. Vom explica funcțiile folosite :

• mysql\_init() inițializează un obiect MYSQL ce va fi folosit ulterior pentru conexiunea la server. Daca parametrul ”mysql” este NULL, atunci se aloca un nou obiect MYSQL înainte de inițializare si se întoarce un pointer la el. Daca parametrul ”mysql” nu este NULL, atunci obiectul se reinițializează și se întoarce un pointer la același obiect.

• mysql\_real\_connect() realizează conexiunea dintre server și baza de date. Se conectează la serverul MySQL care rulează pe "localhost" folosind numele de utilizator "root" și parola "password\_root".

• mysql\_query() creează tabelele. În caz de eroare se va afișa un mesaj și atât programul cât și legătura cu baza de date se vor închide, dar datele vor fi păstrate.

• mysql\_close() închide legătura dintre server si baza de date.

# Concluzii:

Proiectul Offline Messenger se bazează in principal pe creare de socket-uri intre clienți si server folosindu-se de comunicarea TCP concurent , având ca model de referința o baza de date.

Îmbunătățiri :

* Utilizatorul să aibă posibilitatea de reconfigurare a contului în cazul în care își uita parola pentru a nu fii nevoit să își creeze alt cont.
* Pentru informațiile legate de activitatea unui utilizator (online/offline) informațiile sunt

stocate in fișiere .txt, dar putem îmbunătăți acest lucru prin adăugarea unei coloane in tabela „Utilizatori” numită „Status” ce va avea ca valori online sau offline. In acest mod, renunțam la fișiere si introducem toate informațiile in baza de date, iar programul nu va mai fi nevoit sa citească si alte zone de memorie si se va asigura o comunicare fluidă și securizată între utilizatori.

* Ar mai putea fi îmbunătățit și prin adăugarea unor noi opțiuni: ștergere utilizator, trimiterea aceluiași mesaj mai multor utilizatori, posibilitate de creare a unor grupuri(sa poata conversa mai mulți utilizatori in același loc), să apară data și ora la care a fost trimis un mesaj și vizualizat de celălalt user .

# 6. Bibliografie

Bibliografie utilizată în scopul realizării acestei documentații:

<https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php> (pentru codul server si client )

<https://profs.info.uaic.ro/~ioana.bogdan/>

[FII - Retele de calculatoare - Laboratoare (google.com)](https://sites.google.com/view/fii-lab-retele-de-calculatoare/laboratoare)

https://sequencediagram.org/

https://en.wikipedia.org/wiki/Transmission\_Control\_Protocol

[What is multithreading? (techtarget.com)](https://www.techtarget.com/whatis/definition/multithreading)

[MySQL C API programming (zetcode.com)](https://zetcode.com/db/mysqlc/)(pentru codul de la implementarea bazei de date)

[Advantages of SQLite – W3schools](https://www.w3schools.blog/advantages-sqlite#google_vignette)