Offline Messenger

Popa Ioana-Alexia, an II, grupa A3

January 17, 2024

1 Introducere

"Offline Messenger" este un proiect ce implementează o aplicație client/server ce facilitează comunicarea prin mesaje. Scopul principal este asigurarea unei comunicări utilizatorii conectați în timp real, precum și posibilitatea transmiterii și vizualizării mesajelor pentru utlizatorii offline. De asemenea, aplicatia dispune si de funcționalitatea ca utlizatorii să poată raspundă specific la anumite mesaje, precum si vizualizarea istoricului conversațiilor pe care le-a avut cu ceilalți utlizatori în parte. Prin aceste funcționalități, proiectul urmărește să ofere o experiență complexă și organizată a interacțiunilor dintre utilizatori.

Obiectivele proiectului sunt concepute pentru a îndeplini nevoile esențiale ale utilizatorilor în ceea ce privește comunicarea eficientă și flexibilă.

2 Tehnologii Aplicate

2.1 Protocol TCP

Pentru a realiza comunicarea dintre server si client am folosit protocolul TCP, mai exact un server TCP concurent.

Alegerea folosirii acestui protocol in detrimentul UDP este justificată de faptul ca TCP asigură fiabilitate in comunicare, garantează ordinea și livrarea fiecărui pachet de date, eliminând pierderile sau reordonările acestora, aceasta fiind o caracteristică crucială pentru un acest tip de proiect. Datorită caracterului critic al mesajelor într-o aplicație de mesagerie, sunt de parere că TCP furnizează un canal de comunicare stabil și consecvent, deoarece asigură totodată și mecanisme de control al fluxului de date în cazul în care sunt trimise foarte multe mesaje într-un timp foarte scurt.

Pentru a gestiona eficient mai multe conexiuni simultane și a asigura o experiență de comunicare fluidă, am adoptat o strategie de TCP concurent. Această tehnică permite serverului să răspundă rapid la cererile multiple, evitând blocarea intregului sistem în timpul procesării unei singure cereri.

Asigurarea concurenței la nivelul serverului se va realiza prin threaduri. Acestea partajează implicit majoritatea resurselor unui proces, permițând modificarea vizibilă a acestora in cadrul tuturor threadurilor. Alegerea threadurilor pentru această implementare contribuie la sincronizarea eficientă a resurselor si la crearea unei aplicații scalabile și receptive la interacțiunile multiple.

2.2 Stocarea datelor

Pentru stocarea eficientă și pentru a simplifica dezvoltarea sistemului de gestionare a datelor am implementat un motor de baze de date SQL încapsulat prin intermediul bibliotecii SQLLite. Am ales să utilizez acest tip de bibliotecă datorită naturii sale usor de integrat, a unei performanțe foarte bune atunci când se lucrează cu baze de date de dimensiuni mici, fiind rapidă dar și ușor de folosit și a capacității de a oferi un mecanism consistent pentru operațiunile de citire și scriere.

3 Structura Aplicatiei

Diagrama aplicației care prezinta comunicarea serverului cu clientul si cu baza de date este urmatoarea:

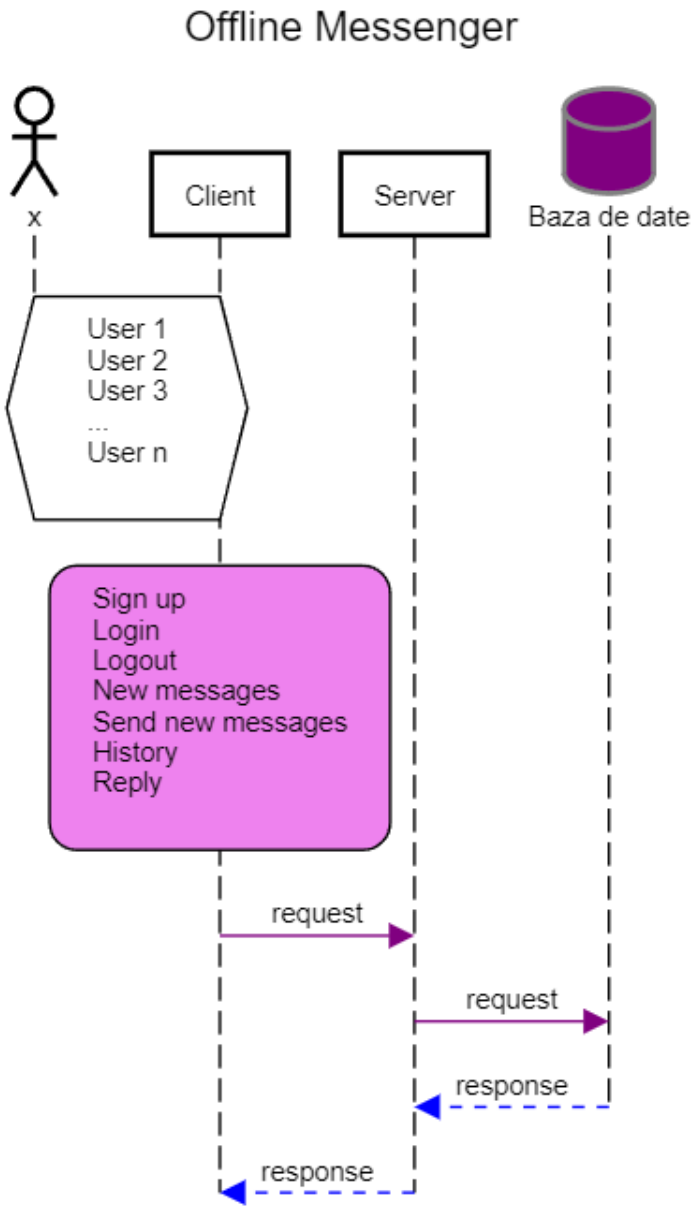

Figure 1: Diagrama aplicației

3.1 Conceptele folosite in modelare

Aplicația este împărțită în 3 componente principale: client, server si baza de date.

În cadrul serverului se pot inregistra si conecta simultan un număr finit de utilizatori. Procesul implică transmiterea cererilor de inregistrare, autentificare si celelalte enumerate in diagramă de către client catre server. Serverul la randul lui acceptă aceste cereri

si comunică cu baza de date pentru a valida informațiile simultan. Baza de date este alcătuită din 3 tabele Utilizatori, Mesaje si Conversații. Aceastea gestionează stocarea numelor de utilizatori și a mesajelor trimise sau primite de fiecare utilizator. Totodata aceste mesaje sunt grupate intr-o tabela separată de numită Conversații, pe baza idului expeditorului si a idului destinatarului.

Conceptele folosite in modelarea aplicației sunt urmatoarele:

1 .Utilizator: entitatea interactivă cu un cont in aplicație, capabilă să trimită și să primească mesaje.

2. Mesaj: unitatea de comunicare, conținând text sau alte tipuri de conținut, facilitând schimbul de informații între utlizatori.

3. Conversație: gruparea mesajelor între utilizatori, oferind acces la istoricul complet al interacțiunilor.

Pe langă acestea specifice unei aplicatii de mesagerie, restul sunt legate în mare parte de implementarea serverului ***TCP concurent***. Pentru a face posibilă comunicarea cu mai multi clienți simultan am folosit conceptul de ***threaduri*** - fire de execuție. Conectarea la server a mai multor utilizatori este posibilă datorită unui apel accept() care blochează serverul.

Urmatoarea diagramă reprezintă prezentarea detaliată a aplicației de Offline Messenger, precum si comportamentul funcțiilor.

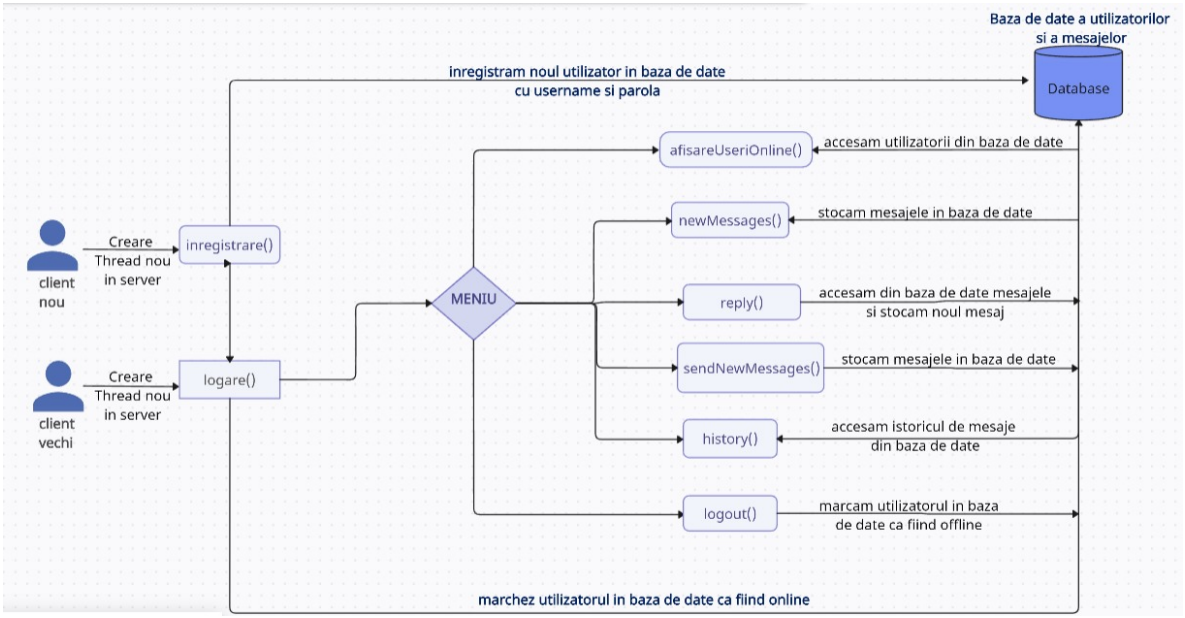

Figure 2: Diagrama aplicației

4 Aspecte de Implementare

4.1 Cod relevant proiectului

Comunicarea este realizată prin intermediul unui server TCP concurent. La implementarea acestuia am folosit threaduri pentru a facilita comunicarea cu mai multi clienți deodată.

Pentru threaduri am utilizat biblioteca "pthread" si am folosit functia detach() pentru a detașa threadul curent, după cum se poate observa și in codul de mai jos.

```c
static void *treat(void * arg){ //functia executata de fiecare thread ce realizeaza comunicarea cu clientii
  struct thDate tdL;
  tdL= *((struct thDate*)arg);
  printf("[thread] - %d- Asteptam mesajul...\n", tdL.IdThread);
  fflush(stdout);
  pthread_detach(pthread_self());
  mesajClient((struct thData*)arg, db);
  /*am terminat cu acest client, inchidem conexiunea*/
  close(tdL.cl);
  free(arg);
  return (NULL);
```

Figure 3: Lucrul cu threaduri

Inițial serverul creează un socket cu ajutorul funcții socket(). Socketul este bidirecțional ceea ce facilitează comunicarea fără a creea mai multe canale de comunicare separate.

```c
/* crearea unui socket */
descr_sk= socket(AF_INET, SOCK_STREAM, 0);
if (descr_sk== -1){
    perror ("[server]Eroare la socket().\n");
    return errno;
}
```

Figure 4: Crearea socketului

Socketul va fi atasat la server prin functia bind().

```c
/* atasam socketul */
if (bind (descr_sk, (struct sockaddr *) &server, sizeof (struct sockaddr)) == -1)
  {
    perror ("[server]Eroare la bind().\n");
    return errno;
  }
```

Figure 5: Funcția bind()

Serverul va ”asculta” cererile de conectare prin funcția listen().

```
/* punem serverul sa asculte daca vin clienti sa se conecteze */
if (listen (descr_sk, 2) == -1)
  {
    perror ("[server]Eroare la listen().\n");
    return errno;
  }
```

Figure 6: Funcția listen()

În cele din urmă se va realiza acceptarea clienților prin funcția accept().

```
 /* acceptam un client (stare blocanta pana la realizarea conexiunii) */
 if ( (client = accept (descr_sk, (struct sockaddr *) &from, &lungime)) < 0){
   perror ("[server]Eroare la accept().\n");
   continue;
 }
```

Figure 7: Funcția accept()

O funcție importantă din server este funcția trimiteRaspuns() care facilitează transmiterea prin descriptor a lungimii precum si raspunsul pe care serverul il va trimite clientului, dupa ce utlizatorul va alege comanda dorita

```
void trimiteRaspuns(int descr_sk, const char *raspuns, int idThread){
  int lungime = strlen(raspuns)+1;

  if(write(descr_sk, &lungime, sizeof(lungime)) <=0){
    perror("Eroare la trimiterea lungimii mesajului.");
    return errno;
  }
  if(write(descr_sk, raspuns, strlen(raspuns) + 1)<=0){
    perror("Eroare la trimiterea raspunsului.");
  }

  printf("Lungimea mesajului primit: %d\n", lungime);
  printf("[Thread %d] Mesajul trimis este: %s\n", idThread, raspuns);
}
```

Figure 8: Funcția trimiteRaspuns()

Clientul va citi mesajul de la server prin aceasta functie:

```c
int primireRaspuns(int descr_sk, char **raspuns){
  int len;

  if (read(descr_sk, &len, sizeof(len)) <0 ){
    printf("[client] Eroare le citirea lungimii.\n");
  }
  //printf("Lungimea mesajului primit: %d\n", len);

  *raspuns=(char*)malloc(len*sizeof(char));
  if(*raspuns == NULL){
      perror("[client] Eroare la alocarea memoriei\n");
      return errno;
  }

  memset(*raspuns, 0, len*sizeof(char));
  if(read(descr_sk, *raspuns, len) < 0){
    perror("[client] Eroare la read de la server.\n");
    free(*raspuns);
    return errno;
  }

  return 0;
}
```

Figure 9: Funcția primireRaspuns()

Exemplu din funcția login unde accesez baza de date și unde setez utilizatorul ca fiind online.

```c
int bd = sqlite3_exec(db, sql, verificareLogin, &dateLogin, &err_msg);

if(bd != SQLITE_OK){
  snprintf(td->raspuns, RESPONSE_SIZE, "[server]Logarea a esuat.\n");
}else{
  if(dateLogin.gasit){
    td->logat = 1;
    td->IdUser = dateLogin.td->IdUser;

    snprintf(td->raspuns, RESPONSE_SIZE, "[server] V-ati logat cu succes! ID utilizator: %d\n", td->IdUser);

    char sqlOnline[512];
    sprintf(sqlOnline, "UPDATE Utilizatori SET online = 1 WHERE IdUser = %d;", td->IdUser);
    char *err_msg_online = 0;
    sqlite3_exec(db, sqlOnline, NULL, 0, &err_msg_online);
    if(err_msg_online != NULL){
      fprintf(stderr, "Eroare la actualizarea starii online: %s.\n", err_msg_online);
      sqlite3_free(err_msg_online);
    }

    conexiuniActive[nrConexiuni].sock = td->cl;
    conexiuniActive[nrConexiuni].IdUser = td->IdUser;
    strncpy(conexiuniActive[nrConexiuni].Username, user.username, sizeof(conexiuniActive[nrConexiuni].Username));
    nrConexiuni++;

  }else{
    snprintf(td->raspuns, RESPONSE_SIZE, "[server] Username sau parola gresita. Incercati din nou!\n");
  }
}
```

Figure 10: Exemplu accesare baza de date

Funcția de verificareLogin() este urmatoarea:

```c
/// fucntia de verificare pt login ///
int verificareLogin(void *data, int argc, char **argv, char **aznumeCol){
    DateLogin *dateLogin = (DateLogin*)data;
    if(argc>0){
      dateLogin->gasit = 1;
      dateLogin->td->IdUser = atoi(argv[0]);
    }else{
      dateLogin->gasit = 0;
    }
    return 0;
}
```

Figure 11: Functie verificareLogin()

Funcția de marcare a mesajelor:

```c
/// functia care marcheaza mesajele ca citite sau necitite ///
void marcareMesaje(int IdUser, sqlite3 *db){
  char sql[256];
  snprintf(sql, sizeof(sql), "UPDATE Mesaje SET Citit = 1 WHERE IdDestinatar = %d AND Citit = 0;", IdUser);
  char *err_msg = 0;
  int bd = sqlite3_exec(db, sql, 0, 0, &err_msg);
  if(bd != SQLITE_OK){
    fprintf(stderr, "Eroarea la baza de date: %s\n", err_msg);
    sqlite3_free(err_msg);
  }
}
```

Figure 12: Funcție marcareMesaje()

Schema bazei de date este aceasta:

```
CREATE TABLE Utilizatori(
IdUser INTEGER PRIMARY KEY AUTOINCREMENT,
Username TEXT NOT NULL UNIQUE,
Parola TEXT NOT NULL,
LastLoginTime TEXT
, online INTEGER);
CREATE TABLE Mesaje(
IdMesaj INTEGER PRIMARY KEY AUTOINCREMENT,
IdExpeditor INTEGER,
IdDestinatar INTEGER,
TextMesaj TEXT,
Timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
Citit INTEGER DEFAULT 0,
IdRaspunsMesaj INTEGER,
FOREIGN KEY(IdExpeditor) REFERENCES Utilizatori(IdUser),
FOREIGN KEY(IdDestinatar) REFERENCES Utilizatori(IdUser),
FOREIGN KEY(IdRaspunsMesaj) REFERENCES Mesaje(IdMesaj)
);
CREATE TABLE Conversatii(
IdConversatie INTEGER PRIMARY KEY AUTOINCREMENT,
IdUtilizator1 INTEGER,
IdUtilizator2 INTEGER,
IdUltimulMesaj INTEGER,
FOREIGN KEY(IdUtilizator1) REFERENCES Utilizatori(IdUser),
FOREIGN KEY(IdUtilizator2) REFERENCES Utilizatori(IdUser),
FOREIGN KEY(IdUltimulMesaj) REFERENCES Mesaje(IdMesaj)
);
```

Figure 13: schema Baza de Date

4.2 Prezentarea protocolului la nivelul aplicatie

- *inregistrare()*: prin intermediul acestei funcții utilizatorul îi va cere serverului să se poată înregistra. Dacă username-ul lui nu se găsește printre ceilalți utilizatori acesta va putea să introduca o parolă pentru contul creat. Apoi, acesta va fi directionat spre logare.

-*login()*: după ce clientul este înregistrat, acesta se va putea loga cu contul sau. Dacă utilizatorul nu este inregistrat si incearcă să se logheze acesta va primi un mesaj de eroare care îl va înștiința că mai intai trebuie să se inregistreze. Dacă utilizatorul își

greșeste parola pe ecran va aparea mesajul "Username sau parola gresit. Incercati din nou!". Odată ce utilizatorul este logat statusul său din baza de date este modificat ca fiind "online".

-*afisareUseriOnline()*: prin intermediul acestei comenzi utilizatorul îi cere serverului să-i afișeze lista de utilizatori care sunt online in același timp cu el.

-*newMessages()*: prin intermediul acestei comenzi utilizatorul îi cere serverului să afișeze lista de mesaje noi pe care le-a primit.. Serverul la randul lui va accesa prin intermediul bazei de date mesajele către utilizator curent care sunt marcate ca necitite și le va afișa. Odată vizualizate acestea sunt marcate in baza de date ca citite.

-*reply()*: prin intermediul acestei comenzi utilizatorul îi cere serverului să afiseze istoricul cu un anumit utlizator pe care il introduce acesta. Fiecare mesaj din istoric va fi insotit de idul acestuia, pentru a facilita alegerea mesajului căruia utilizatorul vrea să răspundă. Dacă nu există mesaje in conversația cu respectivul utilizator i se va afișa mesajul "Nu există mesaje in conversația cu " urmat de numele utilizatorului cerut.

-*sendNewMessages()*: prin intermediul acestei comenzi utilizatorul îi cere serverului să poată trimite un mesaj oricărui alt user din sistem. La selecția acestei comenzi utilizatorului îi este cerut să introducă utilizatorul căruiă vrea să-i trimtită un mesaj. Dacă utilizatorul introdus nu există i se va afișa mesajul "Eroare, destinatarul nu există.", iar daca mesajul a fost trimis i se va afisa un mesaj de confirmare.

-*history()*: prin intermediul acestei comenzi utilizatorul îi cere serverului să poată să vizualizeze istoricul conversațiilor cu toți utilizatorii.

-*logout()*: prin intermediul acestei comenzi utilizatorul îi cere serverului să îl deconecteze de la sesiunea curentă și să-l marcheze in baza de date drept "offline".

4.3 Scenarii de utilizare

În client vom avea initial un meniu de utilizare unde clientul va selecta ce opțiune dorește dintre "Inregistrare" si "Logare" aceasta fiind transmisă mai departe in server, pentru ca acesta la randul lui să trimită înapoi spre client răspunsul corespunzător. După ce s-a facut logarea va fi afișat și restul meniului alcatuit din restul optiunilor.

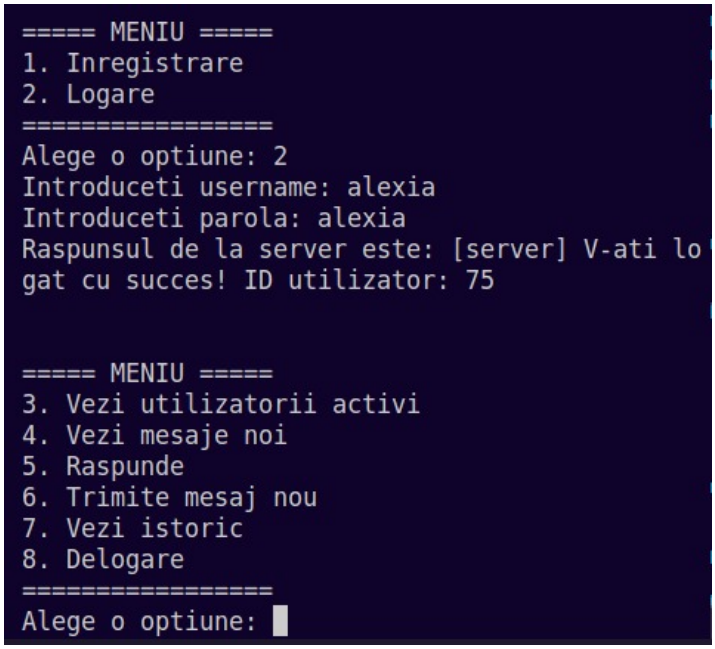

Figure 14: Meniu client

Atunci când un utilizator vrea să se conecteze acesta trebuie să se înregistreze sau să se logheze la sistem prin intermediul unui username și a unei parole. În acest mod

clientul înștiințează serverul că se dorește conectarea și astfel serverul creează un nou thread - fir de execuție și va trimite raspuns la baza de date că este cineva online in momentul respectiv.

Utilizatorul poate alege acum dacă vrea să vizualizeze ce utilizatori mai sunt online în momentul respectiv, sa vizualizeze mesajele noi, să raspundă la unele mesaje, să vizualizeze istoricul conversațiilor cu alți utilizatori sau să înceapă o nouă conversație. De asemenea, acesta se poate deconecta de la server chiar dacă nu a trimis niciun mesaj încă, acest fapt fiind transmis la baza de date pentru ca utilizatorul să fie marcat ca fiind offline.

Vizualizarea utilizatorilor online si a mesajelor noi sau a istoricului conversației se face accesând baza de date unde sunt stocate toate conversațiile dintre toți utilizatorii.

5 Concluzii

Îmbunătățirea majoră pe care aș putea să o aduc proiectului este realizarea unei interfațe grafice. Totodată, alte îmbunătățiri ar fi posibilitatea de a reacționa la mesaje cu emoji sau de a raspunde cu un mesaj vocal. Introducerea unor funcționalități avansate, precum filtrarea istoricului conversațiilor, de exemplu căutarea unor mesaje după anumite cuvinte ar aduce un plus de remarcat aplicației.

6 Referințe Bibliografice

1. https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
2. https://www.andreis.ro/teaching/computer-networks
3. https://ro.wikipedia.org/wiki/TransmissionControlProtocol
4. https://www.geeksforgeeks.org/tcp-ip-model/?ref=headersearch
5. https://ro.wikipedia.org/wiki/SQLite
6. https://app.creately.com/d/k3blDlotD0w/edit
7. https://sequencediagram.org/