Local Marketplace Platform

Dima Gabriel

January 7, 2025

1. Introducere

Proiectul **Local Marketplace Platform** este o platformă locală care permite utilizatorilor să cumpere și să vândă produse în comunitatea lor, folosind un server pentru gestionarea tranzacțiilor și a produselor. Această platformă oferă oportunitatea de a cumpăra și de a vinde orice tip de produs, din partea oricărui utilizator cu un cont valid. Platforma facilitează utilizatorii cu opțiunea de gestionare a soldului și o opțiune de căutare a oricărui obiect dorit, după nume.

2. Tehnologii aplicate

Pentru comunicare între server şi client am decis să utilizez **socket-uri**. Proiectul se va baza pe o **tehnologie de tip TCP**, care va crea un thread separat pentru fiecare client care dorește să se conecteze la server. Această implementare permite conectarea mai multor clienți simultan, fiecare având opțiunea de a vizualiza produsele adăugate pe platforma în timp real. Datele sunt salvate într-o baza de date de tip **SQLite**, avantajul fiind accesul și interogarea eficientă a acesteia.

3. Structura aplicatiei

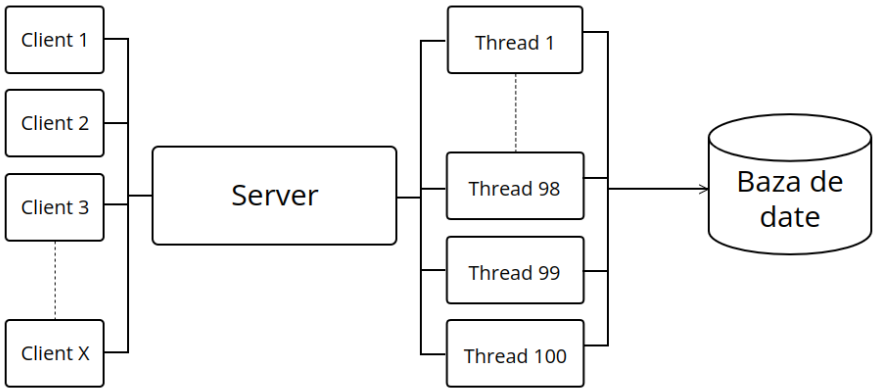

Figure 1: Arhitectura aplicației **Local Marketplace Platform**

Platforma este alcătuită dintr-un server, un număr de clienți care se pot conecta simultan și o bază de date în care se vor reține datele despre fiecare client, precum și informații despre produsele puse la vânzare.

4. Aspecte de implementare

4.1. Prezentare generală:

Client-ul primește de la utilizator (se citește de la tastatură) o serie de comenzi, pe rând, fiecare fiind trimisă ulterior server-ului.

După ce s-a realizat conexiunea dintre client și server (cu ajutorul socket-urilor), în cadrul server-ului se verifică dacă există un thread nefolosit. Dacă un astfel de thread este disponibil, acestuia i se va asigna comunicarea cu clientul și va recepționa mesajul trimis de client.

În cadrul aceluiași thread, comanda primită de la client va fi ”descifrată”, adică verificăm ce comandă am primit și dacă este corect scrisă. Daca această verificare este afirmativă, comanda respectivă se execută și, în funcție de output, se va trimite un răspuns clientului și se vor aplica interogări bazei de date.

4.2. Comenzile disponibile

- login : <username> <parola>
- vizualizare comenzi
- exit
- vizualizare sold
- retragere sold : <suma> <parola>
- adaugare sold : <suma> <parola>
- logout
- creare cont : <username> <parola>
- vizualizare produse
- vizualizare produse proprii
- cumparare produs : <id>
- listare produs : <nume> <pret>
- cautare produs : <id>

4.3. Implementare cod

O îmbunătățire pe care am implemntat-o este reutilizarea thread-urilor. Acest principiu are în vedere fixarea unui număr fix de thread-uri, având astfel un număr fix de clienți cu care se pot conecta(ex. maxim 100 clienți). Am reușit implementarea folosind un vector declarat global **folosit**, în care reținem dacă thread-ul cu indicele **i** este asignat unui client sau nu. În cazul în care nu este asignat, **folosit[i] = 1**. După ce terminăm comunicarea cu clientul, **folosit[i] = 0** (pentru a putea reutiliza thread-ul ulterior).

Am luat în considerare și posibilitatea de **race condition** (două thread-uri separate accesează memoria alocată vectorului **folosit** simultan, având pierderi de memorie). După cum se poate observa și în imaginea de mai jos, am utilizat un mecanism de sincronizare **(mutex)** care rezolvă această problemă și permite accesarea în ordine a memoriei vectorului:

```
/// aici refolosim thread-urile
pthread_mutex_lock(&mutex);
for (i = 0; i < 100; i++)
{
  printf("am gasit thread-ul %d cu valoarea %d\n", i, folosit[i]);
  if (folosit[i] == 0)
  {
    td->idThread = i;
    folosit[i] = 1;
    td->cl = client;
    break;
  }
}
pthread_mutex_unlock(&mutex);

if (i == 100)
{
  printf("No threads available, rejecting client.\n");
  close(client);
}
else
{
  printf("\nId-ul clientului este: %d\n", client);
  pthread_create(&th[i], NULL, &treat, td); // Create thread to handle client
}
```

Crearea server-ului și conectarea cu clienții se va realiza la fel ca în exemplele oferite în cadrul orelor de laborator (**port** va fi adresa portului la care se așteaptă mesajul, iar **host** va fi adresa **IP** a serverului):

```c
struct sockaddr_in server; // structura folosita de server
struct sockaddr_in from;
/* crearea unui socket */
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
  perror("[server]Eroare la socket().\n");
  return errno;
}
/* utilizarea optiunii SO_REUSEADDR */
int on = 1;
setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

/* pregatirea structurilor de date */
bzero(&server, sizeof(server));
bzero(&from, sizeof(from));

server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(PORT);

/* atasam socketul */
if (bind(sd, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1)
{
  perror("[server]Eroare la bind().\n");
  return errno;
}
if (listen(sd, 2) == -1)
{
  perror("[server]Eroare la listen().\n");
  return errno;
}
```

Verificarea și executarea comenzilor va fi relizată cu ajutorul unor funcții în care se realizează interogarile bazei de date. În cadrul funcției principale se verifică corectitudinea instrucțiunilor primite de la client, dacă aceastea pot fi executate și dacă sunt scrise corect. Dacă aceste criterii sunt îndeplinite, se apelează funcția care permite executarea instrucțiunii.

```
if (i != strlen(mesaj))
{
  strcpy(raspuns, "Comanda nu este corecta.\n");
}
else
{
  if (check_login(db, username, parola))
  {
    strcpy(raspuns, "Contul respectiv este valid");
    strcpy(username_logged, username);
    logged_in = 1;
  }
  else
    strcpy(raspuns, "Contul respectiv NU ESTE VALID");
}
```

Baza de date este alcătuită din două tabele, **Users** și **Produse**. Tabela **Users** reține informații despre utilizatori (**username-ul**, **password** și **soldul** fiecăruia), permite crearea unor conturi noi care pot fi accesate ulterior și admite administrarea soldului (se pot adauga sau retrage bani). Tabela **Produse** reține informații despre produsele listate (**numele**, **prețul**, **ID-ul** fiecărui produs și **ID-ul** utilizatorului care vinde produsul). Prin intermediul acestei tabele avem acces la vizualizarea produselor, achiziționarea sau listarea altor obiecte, precum și căutarea unui produs în funcție de nume.

```c
rc = sqlite3_open("users.db", &db);
if (rc != SQLITE_OK)
{
  fprintf(stderr, "Nu se deschide baza de date: %s\n", sqlite3_errmsg(db));
  return 1;
}

const char *create_users_table_sql =
    "CREATE TABLE IF NOT EXISTS Users ("
    "ID INTEGER PRIMARY KEY AUTOINCREMENT, "
    "Username TEXT NOT NULL, "
    "Password TEXT NOT NULL, "
    "Sold REAL NOT NULL DEFAULT 0.0);";

rc = sqlite3_exec(db, create_users_table_sql, 0, 0, 0);
if (rc != SQLITE_OK)
{
  fprintf(stderr, "Eroare la crearea tabelului Users: %s\n", sqlite3_errmsg(db));
  sqlite3_close(db);
  return 1;
}
printf("Tabelul Users a fost creat sau exista deja.\n");
```

Am creat funcții care implementează comenzile citite de la tastatură, fiecare având la baza o comandă **SQL** (de inserare/interogare/ștergere în baza de date).
Funcția următoare execută partea de *login* a utilizatorului, adică verifică dacă parolă și username-ul introduse sunt corecte și se regăsesc în tabela **Useri**.

```
int add_user(sqlite3 *db, const char *username, const char *password)
{
  sqlite3_stmt *stmt;
  const char *sql = "INSERT INTO Users (Username, Password, Sold) VALUES (?, ?, 0.0);";

  int rc = sqlite3_prepare_v2(db, sql, -1, &stmt, 0);
  if (rc != SQLITE_OK)
  {
    fprintf(stderr, "Eroare la interogare: %s\n", sqlite3_errmsg(db));
    return 0;
  }

  sqlite3_bind_text(stmt, 1, username, -1, SQLITE_STATIC);
  sqlite3_bind_text(stmt, 2, password, -1, SQLITE_STATIC);

  rc = sqlite3_step(stmt);
  if (rc != SQLITE_DONE)
  {
    fprintf(stderr, "Eroare la creare user: %s\n", sqlite3_errmsg(db));
    sqlite3_finalize(stmt);
    return 0;
  }
  sqlite3_finalize(stmt);
  return 1;
}
```

4.4. Exemple de utilizare

- Un utilizator care are cont dorește să vândă un produs nou.
- Un utilizator nou dorește să-și facă un cont și să adauge fonduri în acel cont
- Un utilizator a vândut un produs și dorește să-și scoată banii din cont pentru a-i folosi în alt context
- Un utilizator are nevoie de un produs specific (ex. canapea). Poate folosi funcția de căutare a produsului și apoi poate alege oferta cea mai convenabilă.
- Fiecare utilizator poate vedea ofertele sale.
- Un utilizator dorește să cumpere un obiect, având opțiunea de a adauga fonduri în cont pentru a realiza tranzacția.

5. Concluzie

În concluzie, aplicația creează o platforma unde utilizatorii pot cumpăra și vinde produsele listate de ceilalți utilizatori, având o suma de bani disponibilă, pe care o pot modifica.
O îmbunătățire care ar putea fi adusă aplicației ar fi implementarea mai multor comenzi care contribuie la confortul utilizatorului.

6. Bibliografie

- https://diacritice.opa.ro/
- https://edu.info.uaic.ro/computer-networks/index.php
- https://youtu.be/FY9livorrJI?si=PxMs5F0RQELT1Vfi
- https://youtu.be/oq29KUy29iQ?si=wqMKMJ65DC-xyN31
- https://online.visual-paradigm.com/drive/proj=0diagram=list