

Documentație proiect

Maria Tărăboanță

Ianuarie 2025

1 Introducere

Proiectul se numește **LocalMarketplacePlatform**. Acesta are ca scop dezvoltarea unei aplicații client-server ce permite utilizatorilor să comunice cu un marketplace local. Aplicația facilitează următoarele operații pe care le poate efectua clientul:

1. Creare cont (*create_account*)
2. Autentificare (*login*)
3. Vizualizarea produselor ce pot fi cumpărate (*view_products*)
4. Căutarea unui produs (*search_product*)
5. Ordonarea produselor alfabetic (*sort_products*)
6. Vizualizarea prețului unui produs (*view_price*)
7. Cumpărarea unui produs (*buy_product*)
8. Vizualizarea istoricului achizițiilor (*view_history*)
9. Adăugarea unui produs spre vânzare (*sell_product*)
10. Deconectare (*logout*)

2 Tehnologii Aplicate

Pentru dezvoltarea aplicației am utilizat protocolul **TCP** într-un model bazat pe **fork**, din următoarele motive:

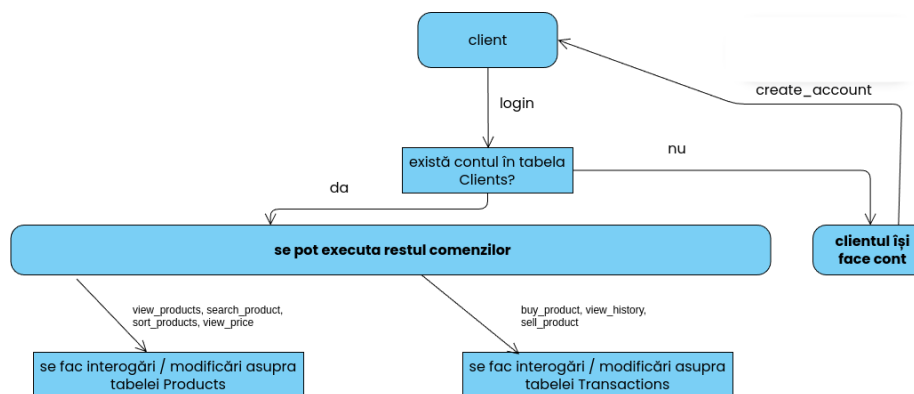
1. **TCP** garantează livrarea mesajelor în aceeași ordine în care au fost trimise, ceea ce este esențial pentru o aplicație de tip marketplace;
2. **TCP** permite utilizatorilor să efectueze operațiuni succesive (ex. căutarea produselor, cumpărare, vizualizarea istoricului) fără întreruperi;

3. Modelul concurent utilizând **fork** permite serverului să proceseze cererile mai multor clienți simultan, alocând fiecărei cereri un proces dedicat.

De asemenea, voi utiliza și baza de date relațională **SQLite**. Aceasta va gestiona informații despre utilizatori, produse și istoricul tranzacțiilor, oferind interogări rapide.

3 Structura Aplicației

Serverul ascultă cererile clientului și răspunde prin procese copil independente, fiind de asemenea conectat la o bază de date pentru interogarea și actualizarea informațiilor, în timp ce clientul trimite cereri către server și primește răspunsuri.



După cum se poate observa în diagrama de mai sus, un client poate executa comenzile specifice aplicației de tip marketplace doar în cazul în care are creat un cont și este conectat.

Structura bazei de date este următoarea.



Când un client va face o ofertă, produsul va fi adăugat în tabela **Products**. Asemănător, când clientul va dori să cumpere un produs, acesta va trebui să îl identifice după codul său unic *product_id*. Tranzacția va fi înregistrată în tabela **Transactions**, și produsul va fi eliminat din tabela **Products**. Astfel, clientul va putea vizualiza și istoricul achizițiilor sale, acesta fiind obținut prin selectarea liniilor din tabela **Transactions** ce au în câmpul *client_id* id-ul clientului ce execută comanda *view_history*.

4 Aspecte de Implementare

În server, se apelează funcția **fork()** pentru a crea un proces independent pentru fiecare client conectat. Astfel se asigură izolarea completă a datelor transmise de mulți clienți și faptul că acestea nu se vor suprapune.

```

132 int main()
133 {
134     while (1)
135     {
136         client = accept(sd, (struct sockaddr *)&from, &length);
137         if (client < 0)
138         {
139             perror("[server]Eroare la accept().\n");
140             continue;
141         }
142         int pid;
143         if ((pid = fork()) == -1)
144         {
145             close(client);
146             continue;
147         }
148         else if (pid > 0)
149         {
150             // parinte
151             close(client);
152             while (waitpid(-1, NULL, WNOHANG));
153             continue;
154         }
155         else if (pid == 0)
156         {
157             // copil
158             close(sd);
159             handle_command(client, db);
160             exit(0);
161         }
162     }
163 }

```

Datorită buclei infinite, clientul poate introduce câte comenzi dorește, acesta oprindu-se utilizând comanda *logout*. Acest tip de buclă se regăsește și în codul pentru client.

```

60 menu();
61 while (1)
62 {
63     bzero(command, 100);
64     printf("[client]Introduceti comanda: ");
65     fflush(stdout);
66     read(0, command, 100);
67     if (strcmp(command, "logout", 0) == 0)
68     {
69         printf("[client] Deconectare...\n");
70         break;
71     }
72     if (write(sd, command, 100) <= 0)
73     {
74         perror("[client]Eroare la write() spre server.\n");
75         return errno;
76     }
77     char command_ans[100];
78     bzero(command_ans, 100);
79     int bytesRead = 0;
80     while ((bytesRead = read(sd, command_ans, sizeof(command_ans) - 1)) > 0)
81     {
82         command_ans[bytesRead] = '\0';
83         printf("[client] Mesajul primit de la server: %s\n", command_ans);
84     }
85 }

```

În server am implementat câte o funcție pentru a gestiona comenzile.

```

12 #define PORT 2024
13
14 extern int errno;
15
16 void database(sqlite3 *db) {
17     // ...
18 }
19
20 void handle_create_account(int client, sqlite3 *db, int *is_logged_in) {
21     // ...
22 }
23
24 void handle_login(int client, sqlite3 *db, int *is_logged_in, int *logged_in_user_id) {
25     // ...
26 }
27
28 void handle_sell_product(int client, sqlite3 *db) {
29     // ...
30 }
31
32 void handle_view_products(int client, sqlite3 *db) {
33     // ...
34 }
35
36 void handle_sort_products(int client, sqlite3 *db) {
37     // ...
38 }
39
40 void handle_search_product(int client, sqlite3 *db) {
41     // ...
42 }
43
44 void handle_view_price(int client, sqlite3 *db) {
45     // ...
46 }
47
48 void handle_buy_product(int client, sqlite3 *db, int logged_in_user_id) {
49     // ...
50 }
51
52 void handle_view_history(int client, sqlite3 *db, int logged_in_user_id) {
53     // ...
54 }
55
56 void handle_command(int client, sqlite3 *db) {
57     // ...
58 }
59
60 int main() {
61     struct sockaddr_in server;
62     struct sockaddr_in from;
63     int sd;
64     // ...
65 }

```

Comenzile *create_account* și *login* vor cere de la utilizator, pe rând, numele de utilizator și parola. Din acest motiv am implementat și în client funcții pentru procesarea datelor, fiind o comunicare în mai multe etape între server și client:

1. clientul transmite serverului numele comenzii, pe care l-a citit de la tastatură
2. serverul apelează funcția *handle* corespunzătoare și transmite un mesaj către client, așteptând numele de utilizator
3. utilizatorul introduce datele, ele fiind ulterior transmise înapoi la server
4. serverul verifică validitatea datelor primite și trimite răspunsul înapoi către client
5. în cazul unui mesaj de eroare, clientul îl afișează și oprește execuția comenzii, altfel continuă cu introducerea parolei, verificarea acesteia făcându-se similar

```

72 char username[100], password[100], message[100];
73 bzero(username, 100);
74 bzero(password, 100);
75 bzero(message, 100);
76
77 struct message {
78     int type;
79     char *data;
80 };
81
82 if (read(client, username, sizeof(username)) < 0) {
83     perror("server: Eroare la citirea numelui de utilizator.\n");
84     close(client);
85     return;
86 }
87
88 if (strlen(username) < 1) {
89     username[strlen(username)] = '\0';
90     printf("server: Numele de utilizator primit: %s\n", username);
91 }
92
93 bzero(message, 100);
94 struct message msg;
95 msg.type = 1;
96 msg.data = "Introduceti parola:\n";
97 write(client, message, strlen(message));
98
99 if (read(client, password, sizeof(password)) < 0) {
100     perror("server: Eroare la citirea parolei.\n");
101     close(client);
102     return;
103 }
104
105 if (strlen(password) < 1) {
106     password[strlen(password)] = '\0';
107     printf("server: Parola primita: %s\n", password);
108 }

```

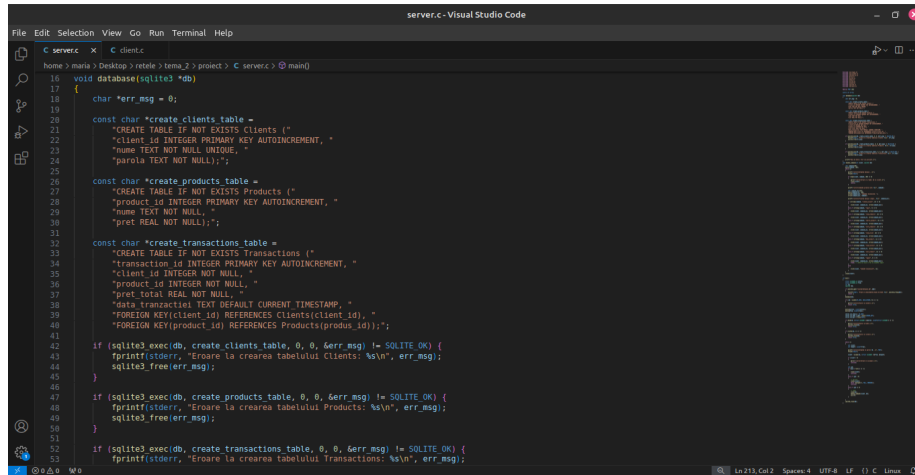
```

41 bzero(ans, 100);
42 while ((bytesRead = read(sd, ans, sizeof(ans) - 1)) > 0) {
43     ans[bytesRead] = '\0';
44     printf("client: %s\n", ans);
45     if (strcmp(ans, "Eroare") < 0) {
46         return;
47     }
48     if (bytesRead < sizeof(ans) - 1) {
49         break;
50     }
51 }
52
53 if (bytesRead < 0) {
54     perror("client: Eroare la read() de la server.\n");
55     return;
56 }
57
58 bzero(username, 100);
59 read(0, username, 100);
60 username[strlen(username)] = '\0';
61 if (write(sd, username, strlen(username)) < 0) {
62     perror("client: Eroare la trimiterea numelui de utilizator.\n");
63     return;
64 }
65
66 bzero(ans, 100);
67 while ((bytesRead = read(sd, ans, sizeof(ans) - 1)) > 0) {
68     ans[bytesRead] = '\0';
69 }

```

Comanda *sell_product* funcționează similar, dar utilizatorul va trebui să introducă numele produsului și prețul acestuia. La comenzile *view_price* și *buy_product* este necesar ca utilizatorul să introducă doar id-ul produsului dorit, respectiv numele produsului la comanda *search_product*, deci se va face o singură verificare. Pentru comenzile *view_products*, *sort_products* și *view_history* nu sunt necesare date suplimentare, serverul doar execută interogări și transmite răspunsul clientului.

De asemenea, serverul creează și tabelele necesare.

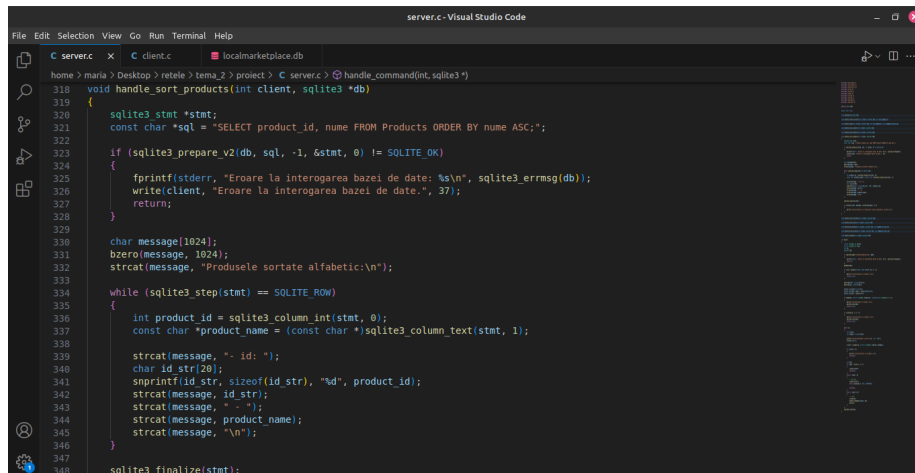


```

server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
C server.c x C client.c
home > maria > Desktop > retele > tema_2 > proiect > C server.c > @ main()
16 void database(sqlite3 *db)
17 {
18     char *err_msg = 0;
19
20     const char *create_clients_table =
21         "CREATE TABLE IF NOT EXISTS Clients ("
22         "client_id INTEGER PRIMARY KEY AUTOINCREMENT, "
23         "nume TEXT NOT NULL UNIQUE, "
24         "parola TEXT NOT NULL);";
25
26     const char *create_products_table =
27         "CREATE TABLE IF NOT EXISTS Products ("
28         "product_id INTEGER PRIMARY KEY AUTOINCREMENT, "
29         "nume TEXT NOT NULL, "
30         "pret REAL NOT NULL);";
31
32     const char *create_transactions_table =
33         "CREATE TABLE IF NOT EXISTS Transactions ("
34         "transaction_id INTEGER PRIMARY KEY AUTOINCREMENT, "
35         "client_id INTEGER NOT NULL, "
36         "product_id INTEGER NOT NULL, "
37         "pret_total REAL NOT NULL, "
38         "data_transactiei TEXT DEFAULT CURRENT_TIMESTAMP, "
39         "FOREIGN KEY(client_id) REFERENCES Clients(client_id), "
40         "FOREIGN KEY(product_id) REFERENCES Products(product_id));";
41
42     if (sqlite3_exec(db, create_clients_table, 0, 0, &err_msg) != SQLITE_OK) {
43         fprintf(stderr, "Eroare la crearea tabelului Clients: %s\n", err_msg);
44         sqlite3_free(err_msg);
45     }
46
47     if (sqlite3_exec(db, create_products_table, 0, 0, &err_msg) != SQLITE_OK) {
48         fprintf(stderr, "Eroare la crearea tabelului Products: %s\n", err_msg);
49         sqlite3_free(err_msg);
50     }
51
52     if (sqlite3_exec(db, create_transactions_table, 0, 0, &err_msg) != SQLITE_OK) {
53         fprintf(stderr, "Eroare la crearea tabelului Transactions: %s\n", err_msg);
54     }

```

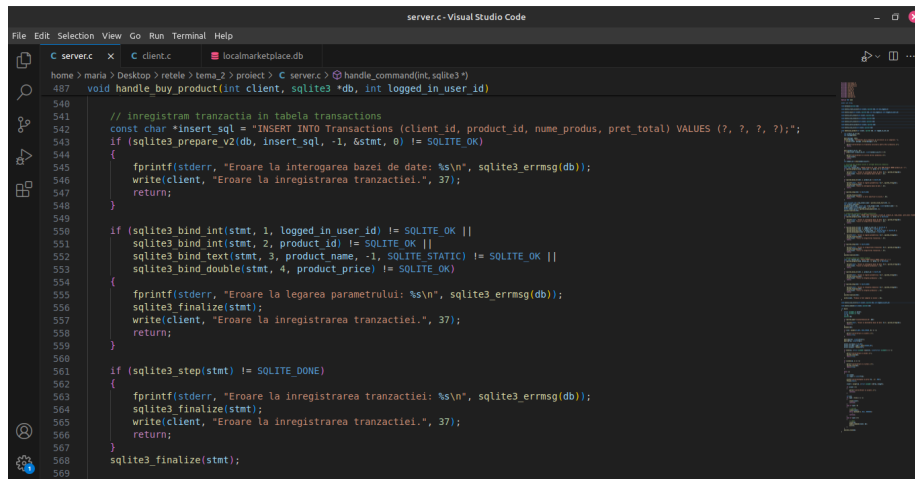
Toate funcțiile descrise mai sus conțin o parte de interogări/modificări asupra bazei de date, acest lucru fiind esențial pentru funcționarea corectă.



```

server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
C server.c x C client.c x localmarketplace.db
home > maria > Desktop > retele > tema_2 > proiect > C server.c > @ handle_command(int,sqlite3 *)
318 void handle_sort_products(int client, sqlite3 *db)
319 {
320     sqlite3_stmt *stmt;
321     const char *sql = "SELECT product_id, nume FROM Products ORDER BY nume ASC;";
322
323     if (sqlite3_prepare_v2(db, sql, -1, &stmt, 0) != SQLITE_OK)
324     {
325         fprintf(stderr, "Eroare la interogarea bazei de date: %s\n", sqlite3_errmsg(db));
326         write(client, "Eroare la interogarea bazei de date.", 37);
327         return;
328     }
329
330     char message[1024];
331     bzero(message, 1024);
332     strcat(message, "Produsele sortate alfabetic:\n");
333
334     while (sqlite3_step(stmt) == SQLITE_ROW)
335     {
336         int product_id = sqlite3_column_int(stmt, 0);
337         const char *product_name = (const char *)sqlite3_column_text(stmt, 1);
338
339         strcat(message, "- id: ");
340         char id_str[20];
341         sprintf(id_str, "%d", product_id);
342         strcat(message, id_str);
343         strcat(message, " - ");
344         strcat(message, product_name);
345         strcat(message, "\n");
346     }
347
348     sqlite3_finalize(stmt);

```



```
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help

C: server.c x C: client.c localmarketplace.db

home > marta > Desktop > retele > tema.2 > proiect > C server.c @ handle_command(int,sqlite3*)
487 void handle_buy_product(int client, sqlite3 *db, int logged_in_user_id)
{
    // inregistram tranzactia in tabela transactions
    const char *insert_sql = "INSERT INTO Transactions (client_id, product_id, nume_produs, pret_total) VALUES (?, ?, ?, ?);";
    if (sqlite3_prepare_v2(db, insert_sql, -1, &stmt, 0) != SQLITE_OK)
    {
        fprintf(stderr, "Eroare la interogarea bazei de date: %s\n", sqlite3_errmsg(db));
        write(client, "Eroare la inregistrarea tranzactiei.", 37);
        return;
    }

    if (sqlite3_bind_int(stmt, 1, logged_in_user_id) != SQLITE_OK ||
        sqlite3_bind_int(stmt, 2, product_id) != SQLITE_OK ||
        sqlite3_bind_text(stmt, 3, product_name, -1, SQLITE_STATIC) != SQLITE_OK ||
        sqlite3_bind_double(stmt, 4, product_price) != SQLITE_OK)
    {
        fprintf(stderr, "Eroare la legarea parametrului: %s\n", sqlite3_errmsg(db));
        sqlite3_finalize(stmt);
        write(client, "Eroare la inregistrarea tranzactiei.", 37);
        return;
    }

    if (sqlite3_step(stmt) != SQLITE_DONE)
    {
        fprintf(stderr, "Eroare la inregistrarea tranzactiei: %s\n", sqlite3_errmsg(db));
        sqlite3_finalize(stmt);
        write(client, "Eroare la inregistrarea tranzactiei.", 37);
        return;
    }

    sqlite3_finalize(stmt);
}
```

Protocolul implementat (scenariu de utilizare):

- clientul trimite o cerere sub formă de text, după ce este logat în contul său (*view_products*);
- serverul interpretează cererea și efectuează interogarea corespunzătoare în baza de date;
- serverul trimite înapoi lista produselor sub formă de text simplu;
- clientul afișează lista produselor primită.

5 Concluzii

Potențiale îmbunătățiri:

- introducerea unui stoc pentru produse (momentan acesta este 1 atât pentru vânzare cât și pentru cumpărare);
- vizualizarea istoricului de vânzări;
- introducerea unei descrieri pentru produse;
- căutarea produselor după cuvinte-cheie (nu doar după nume).

6 Referințe Bibliografice

- <https://www.overleaf.com/learn>;
- <https://www.sqlite.org/>;
- <https://www.visual-paradigm.com/> (pentru diagrame);
- <https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php> (pentru structura implementării serverului / clientului TCP concurent).