Documentatie aplicatie VISIT CARD

Istrate Gheorghe-Iustin,B4

1. INTRODUCERE

Proiectul Visit implementeaza o aplicatie client-server pentru gestionarea si accesarea electronica a cartilor de vizita. Prin utilizarea tehnologiei client-server, aplicatia aduce accesul rapid la informatii si permita utilizatorilor sa mentina actualizate aceste carti de vizita. Vom putea stoca si accesa aceste carti de vizita, avand posibilitatea de a adauga noi carti de vizita,de a modifica sau sterge aceste carti de vizita.

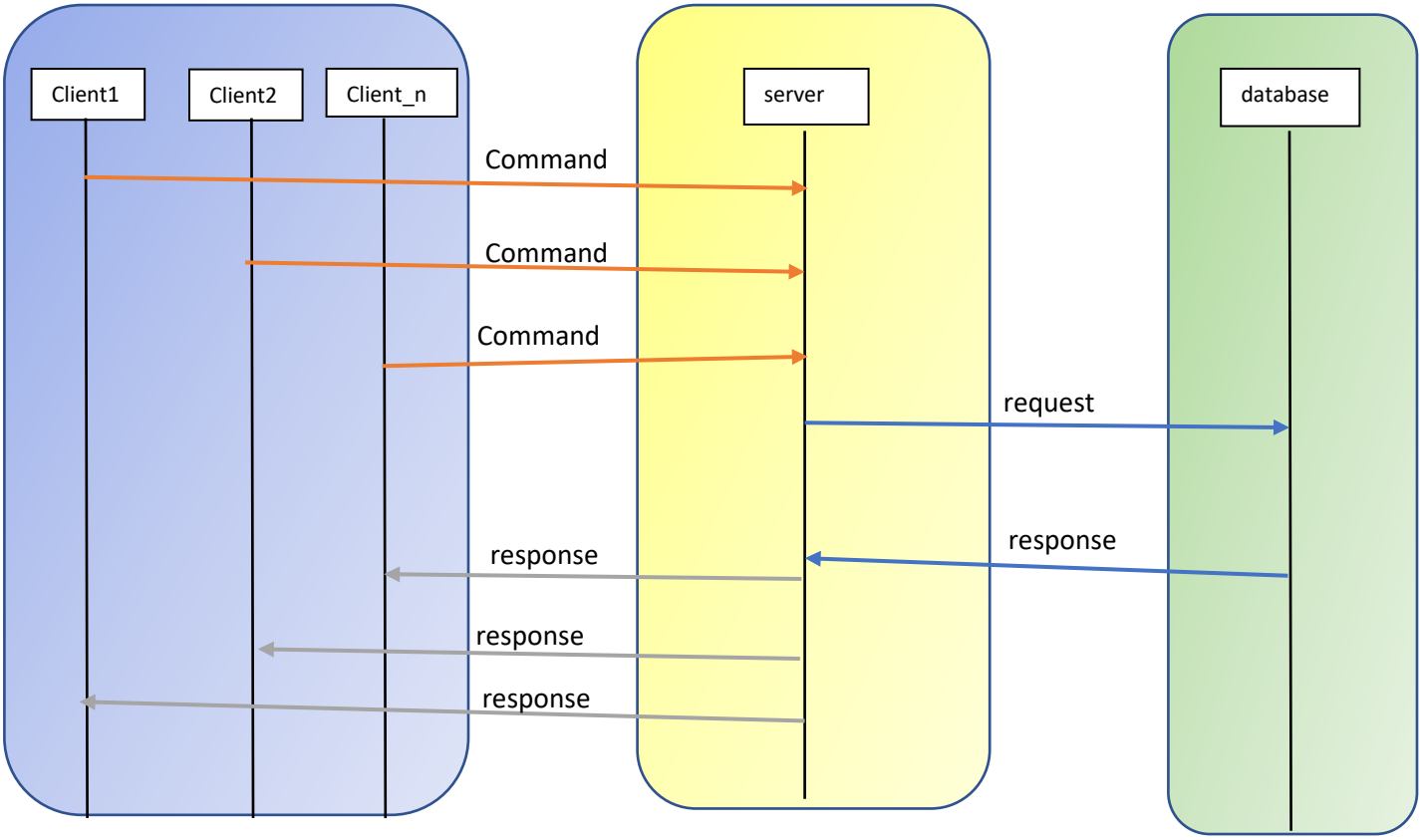
2. TEHNOLOGII APLICATE

Aplicatia va folosi protocolul TCP(Transmission Control Protocol). TCP este un standard care defineste cum stabilim si mentinem conexiunea intre o sursa si o destinatie pana cand transferal de date este complet (in cazul nostru, il folosim pentru conexiunea client/server). Acest protocol asigura ca toate datele vor ajunge la destinatie in ordinea corecta. Prin multiplexare, permite mai multor aplicatii sa utilizeze reteaua simultan, in cazul nostru, sa avem mai multi client conectati la server. In contextul aplicatiilor web,TCP este adesea folosit impreuna cu IP formand TCP/IP, acest lucru permitand transmiterea fiabila si ordonata a datelor intre calculatoare pe o retea,inclusive pe Internet.

De asemenea vom folosi baza de date SQLite pentru stocarea si gestionarea eficienta a cartilor de vizita.

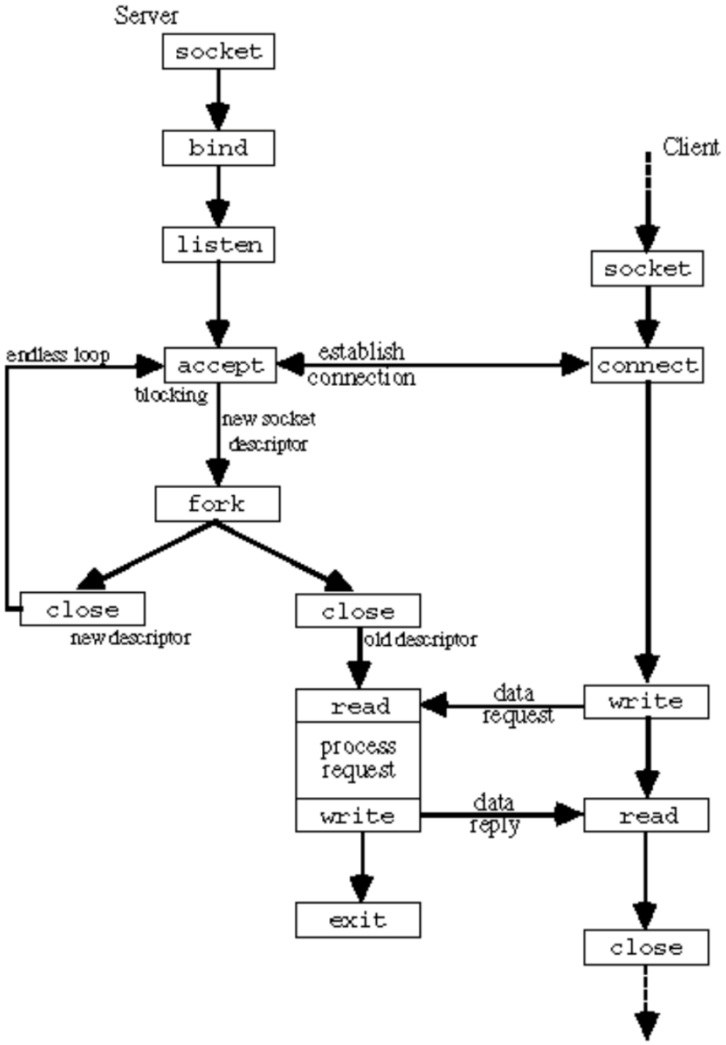
3. ARHITECTURA APLICATIEI

Sistemul este structurat in două componente principale, serverul si clientii. Serverul var ula continuu asteptand cereri de la client si procesand aceste cereri in functie de tipul fiecareia,adica cerere de vizualizare a cartilor de vizita, actualizarea datelor a unei carti de vizita, adaugarea sau stergerea uneia.Comunicarea se bazeaza cum am spus si mai sus pe protocolul TCP/IP.



4. IMPLEMENTARE

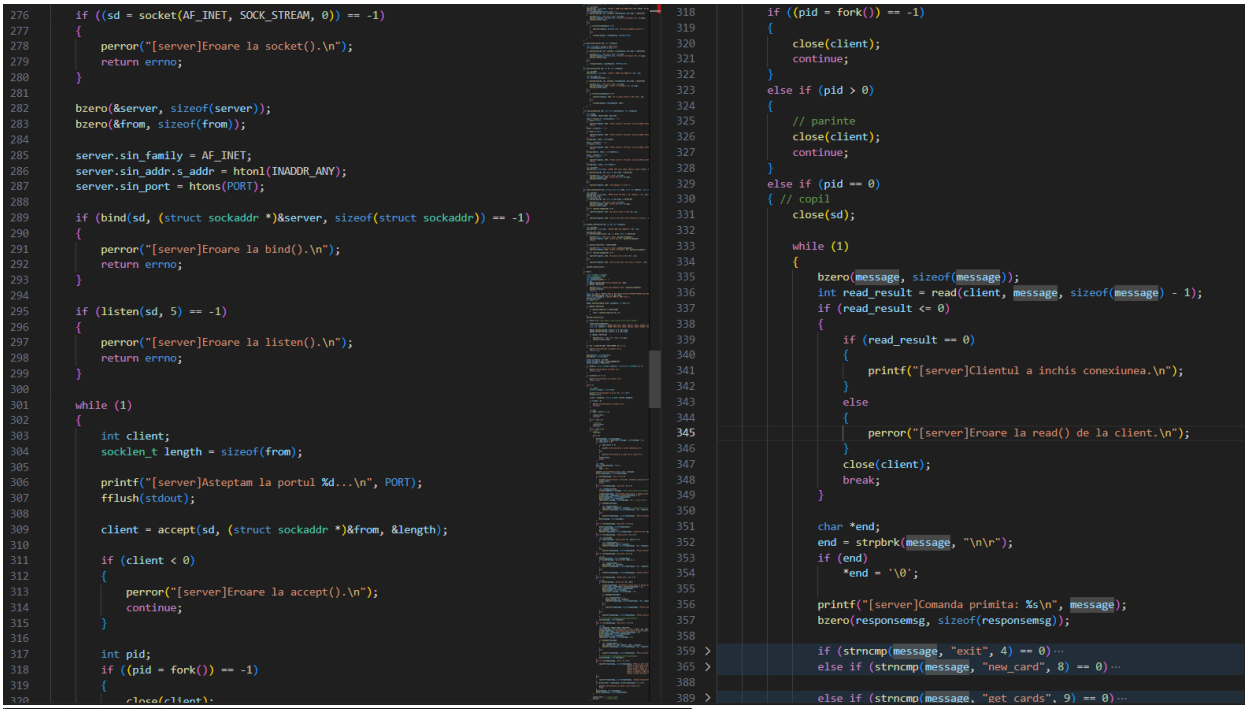
Facem conexiunea inre un client TCP si un server TCP concurent. Clientul creeaza un socket si ii aloca o zona de memorie iar serverul deschide socketul, ataseaza un port pentru furnizarea serviciului si asteapta conectarea clientilor(listen() (acestia se conecteaza la server folosind IP ul si PORT ul la care se afla serverul iar serverul primeste acest client prin accept()). Serverul creeaza un process copil prin fork() specific pentru clientul respectiv. In cadrul acestui proces serverul realizeaza transferul de date cu read() si write() spre clienti si inapoi, dupa care finalizeaza conexiunea (close()).



Implementare cod:

- server.c

  Partea de comunicare cu clientul



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

    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);

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

    if (listen(sd, 5) == -1)
    {
        perror("[server]Eroare la listen().\n");
        return errno;
    }

    while (1)
    {
        int client;
        socklen_t length = sizeof(from);

        printf("[server]Asteptam la portul %d...\n", PORT);
        fflush(stdout);

        client = accept(sd, (struct sockaddr *)&from, &length);

        if (client < 0)
        {
            perror("[server]Eroare la accept().\n");
            continue;
        }

        int pid;
        if ((pid = fork()) == -1)
        {
            close(client);
            continue;
        }
        else if (pid > 0)
        {
            // parinte
            close(client);
            continue;
        }
        else if (pid == 0)
        { // copil
            close(sd);

            while (1)
            {
                bzero(message, sizeof(message));
                int read_result = read(client, message, sizeof(message) - 1);
                if (read_result <= 0)
                {
                    if (read_result == 0)
                    {
                        printf("[server]Clientul a inchis conexiunea.\n");
                    }
                    else
                    {
                        perror("[server]Eroare la read() de la client.\n");
                    }
                    close(client);
                    break;
                }

                char *end;
                end = strpbrk(message, "\n\r");
                if (end)
                    *end = '\0';

                printf("[server]Comanda primita: %s\n", message);
                bzero(responsemsg, sizeof(responsemsg));

                if (strncmp(message, "exit", 4) == 0)...
                else if (strncmp(message, "new_card", 8) == 0)...

                else if (strncmp(message, "get_cards", 9) == 0)...
                else if (strncmp(message, "search_cards", 12) == 0)...
                else if (strncmp(message, "get_card", 8) == 0)...
                else if (strncmp(message, "delete_card", 11) == 0)...
                else if (strncmp(message, "edit_card", 9) == 0)...
                else if (strncmp(message, "help", 4) == 0)...
                else...
                if (write(client, responsemsg, strlen(responsemsg) + 1) <= 0)
                {
                    perror("[server]Eroare la write() catre client.\n");
                    break;
                }
                bzero(message, sizeof(message));
                bzero(responsemsg, sizeof(responsemsg));
            }

            close(client); // inchid socketu
            exit(0);       // inchid copilu
        }
    }
    sqlite3_close(db);
}
```

Conectarea la baza de date:

Deschid baza de date, creez tabelul daca nu există,iar daca acesta este gol,voi insera două valori default date de mine (linia 251 la if(count==0).

```
231     struct sockaddr_in server;
232     struct sockaddr_in from;
233     char message[1024];
234     char responsemsg[1024] = " ";
235     int sd;
236     int dbmsg= sqlite3_open("server_database.db", &db);
237     if (dbmsg!= SQLITE_OK)
238     {
239         fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
240         sqlite3_close(db);
241         return 1;
242     }
243     const char *sql = "CREATE TABLE IF NOT EXISTS Cards(ID INTEGER PRIMARY KEY AUTOINCREMENT, Name TEXT, Address TEXT, Email TEXT);";
244     dbmsg= sqlite3_exec(db, sql, 0, 0, &err_msg);
245     const char *countQuery = "SELECT COUNT(*) FROM Cards;";
246     sqlite3_stmt *res;
247     int count = 0;
248 
249     dbmsg= sqlite3_prepare_v2(db, countQuery, -1, &res, 0);
250 
251     if (dbmsg== SQLITE_OK)
252     {
253         if (sqlite3_step(res) == SQLITE_ROW)
254         {
255             count = sqlite3_column_int(res, 0);
256         }
257     }
258     sqlite3_finalize(res);
259 
260     if (count == 0) //daca tabelu i gol inserez niste valori default
261     {
262         reset_autoincrement(db);
263         const char *insert1 = "INSERT INTO Cards (Name, Address, Email) VALUES ('Ana Maria', 'Dimitrie Ralet', 'anamaria@gmail.com');";
264         const char *insert2 = "INSERT INTO Cards (Name, Address, Email) VALUES ('Damian', 'strada x', 'damian@gmail.com');";
265 
266         dbmsg= sqlite3_exec(db, insert1, 0, 0, &err_msg);
267         dbmsg= sqlite3_exec(db, insert2, 0, 0, &err_msg);
268 
269         if (dbmsg!= SQLITE_OK)
270         {
271             fprintf(stderr, "SQL error: %s\n", err_msg);
272             sqlite3_free(err_msg);
273         }
274     }
```

Implementarea comenzilor new_card,get_card,get_cards etc.

Voi pune doar cele trei comenzi exit,new_card si get_cards deoarece celelalte sunt asemanatoare:

```
if (strncmp(message, "exit", 4) == 0)
{
    printf("[server]Clientul a solicitat închiderea conexiunii.\n");
    close(client);
    break;
}
else if (strncmp(message, "new_card", 8) == 0)
{
    char cardDetails[1024];
    strcpy(cardDetails, message); //aici salvez detaliile cardului ca sa nu se piarda cand o sa citesc parola

    strcpy(responsemsg, "Introduceti parola pentru a adauga cardul:");
    write(client, responsemsg, strlen(responsemsg) + 1);
    bzero(responsemsg, sizeof(responsemsg));
    bzero(message, sizeof(message));
    read(client, message, sizeof(message) - 1); // citesc parola

    if (checkpass(message))
    {
        char response[1024] = "";
        new_card(db, cardDetails, response); // adaug cardu nou cu detaliile de le am salvat mai sus
        snprintf(responsemsg, sizeof(responsemsg), "%s", response);
    }
    else
    {
        snprintf(responsemsg, sizeof(responsemsg), "Parola incorecta. Cardul nu a fost adaugat.");
    }
    bzero(message, sizeof(message));
}

else if (strncmp(message, "get_cards", 9) == 0)
{
    bzero(responsemsg, sizeof(responsemsg));
    char response[1024] = "";
    get_cards(db, response);
    snprintf(responsemsg, sizeof(responsemsg), "Cardurile sunt:\n%s", response);
}
```

Pentru comenzile new_card,delete_card si edit_card cer o parola pentru a putea folosi aceste comenzi doar cei care detin parola(parola este declarata global constanta ca fiind “rc2024”). Cand clientul primeste o comanda din cele mentionate, acesta face needsPassword=1 pentru ca va fi nevoie de un schimb de mesaje intre server si client care nu are loc pentru celelalte comenzi care nu necesita parola. Dupa cum se vede in poza de mai sus, cand suntem pe una din comenzile respective,serverul trimite mesaj catre client si asteapta un raspuns cu parola, iar acesta verifica daca parola este correcta in if(checkpass(message)). Daca parola este corecta, executam comanda,in caz contrar trimitem catre client mesajul “Parola incorecta…”.Voi atasa mai jos si codul din client pentru gestionarea situatiei cu parola:

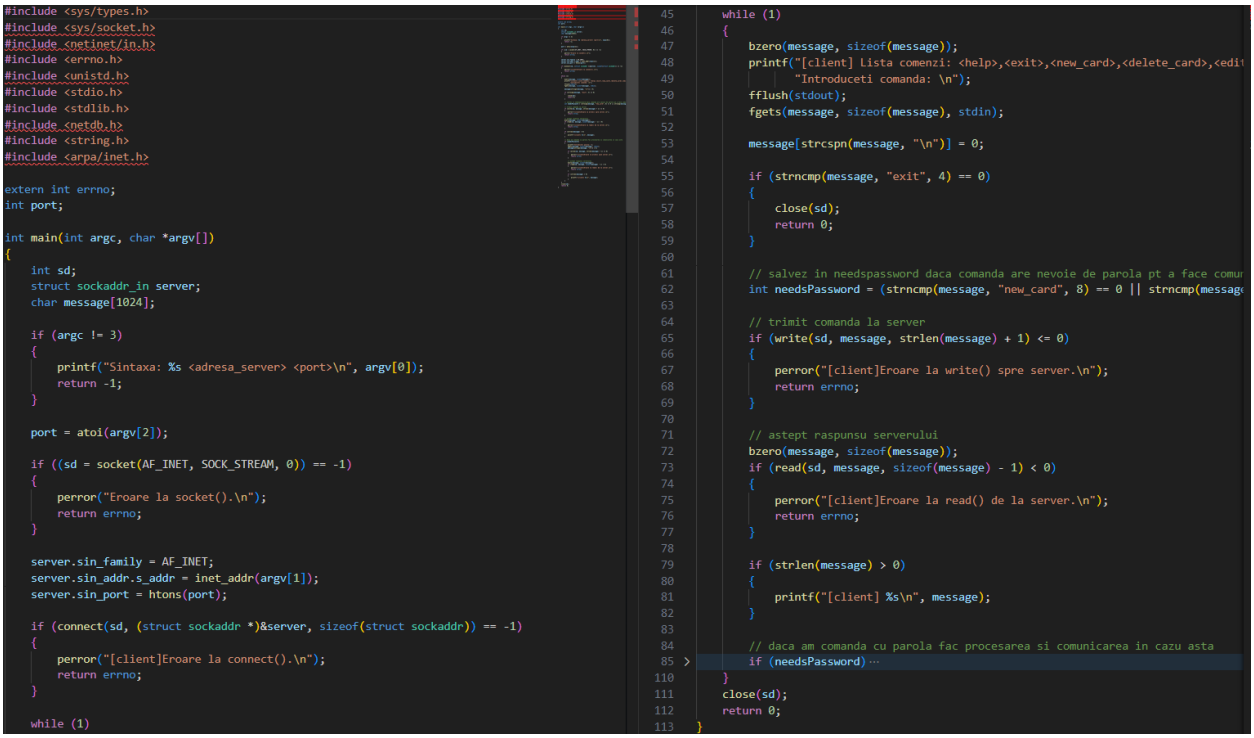
```
85          if (needsPassword)
86          {
87              printf("Introduceti parola: ");
88              fgets(message, sizeof(message), stdin);
89              message[strcspn(message, "\n")] = 0;
90
91              if (write(sd, message, strlen(message) + 1) <= 0)
92              {
93                  perror("[client]Eroare la write() spre server.\n");
94                  return errno;
95              }
96
97              // astept raspunsu serverului
98              bzero(message, sizeof(message));
99              if (read(sd, message, sizeof(message) - 1) < 0)
100             {
101                 perror("[client]Eroare la read() de la server.\n");
102                 return errno;
103             }
104
105             if (strlen(message) > 0)
106             {
107                 printf("[client] %s\n", message);
108             }
109         }
```

Pentru toate comenzile in afara de help si exit apelez functia fiecarei comenzi pentru a lucra cu baza de date.In exemplul de mai jos de afla functia new_card care va primi detaliile cardului separate prin virgula. In cazul in care nu am folosit formatul Nume,Adresa,Email afisam mesajul format incorrect si iesim din functie. In caz contrar vom insera in tabel cu “INSERT….”,text care va fi stocat in variabila sql, iar apoi vom folosi functia sqlite3_exec care va executa mesajul din variabila sql pentru baza de date db.

```
void new_card(sqlite3 *db, const char *card_details, char *response)
{
    char *token;
    char name[50], address[100], email[50];

    token = strtok((char *)card_details, ",");
    if (token == NULL)
    {
        snprintf(response, 1024, "Format incorect. Utilizati: new_card,<Nume>,<Adresa>,<Email>");
        return;
    }
    token = strtok(NULL, ",");

    if (token == NULL)
    {
        snprintf(response, 1024, "Format incorect. Utilizati: new_card,<Nume>,<Adresa>,<Email>");
        return;
    }
    strncpy(name, token, sizeof(name));

    token = strtok(NULL, ",");
    if (token == NULL)
    {
        snprintf(response, 1024, "Format incorect. Utilizati: new_card,<Nume>,<Adresa>,<Email>");
        return;
    }
    strncpy(address, token, sizeof(address));

    token = strtok(NULL, ",");
    if (token == NULL)
    {
        snprintf(response, 1024, "Format incorect. Utilizati: new_card,<Nume>,<Adresa>,<Email>");
        return;
    }
    strncpy(email, token, sizeof(email));

    char sql[1024];
    snprintf(sql, sizeof(sql), "INSERT INTO Cards (Name, Address, Email) VALUES ('%s', '%s', '%s');", name, address, email);

    if (sqlite3_exec(db, sql, NULL, 0, &err_msg) != SQLITE_OK)
    {
        fprintf(stderr, "SQL error: %s\n", err_msg);
        snprintf(response, 1024, "Eroare SQL: %s", err_msg);
        sqlite3_free(err_msg);
    }
    else
    {
        snprintf(response, 1024, "Card adaugat cu succes.");
    }
}
```

- client.c

  Partea de comunicare cu serverul:



```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <string.h>
#include <arpa/inet.h>

extern int errno;
int port;

int main(int argc, char *argv[])
{
    int sd;
    struct sockaddr_in server;
    char message[1024];

    if (argc != 3)
    {
        printf("Sintaxa: %s <adresa_server> <port>\n", argv[0]);
        return -1;
    }

    port = atoi(argv[2]);

    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("Eroare la socket().\n");
        return errno;
    }

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr(argv[1]);
    server.sin_port = htons(port);

    if (connect(sd, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1)
    {
        perror("[client]Eroare la connect().\n");
        return errno;
    }

    while (1)
    {
        bzero(message, sizeof(message));
        printf("[client] Lista comenzi: <help>,<exit>,<new_card>,<delete_card>,<edit
               "Introduceti comanda: \n");
        fflush(stdout);
        fgets(message, sizeof(message), stdin);

        message[strcspn(message, "\n")] = 0;

        if (strncmp(message, "exit", 4) == 0)
        {
            close(sd);
            return 0;
        }

        // salvez in needspassword daca comanda are nevoie de parola pt a face comu
        int needsPassword = (strncmp(message, "new_card", 8) == 0 || strncmp(messag

        // trimit comanda la server
        if (write(sd, message, strlen(message) + 1) <= 0)
        {
            perror("[client]Eroare la write() spre server.\n");
            return errno;
        }

        // astept raspunsu serverului
        bzero(message, sizeof(message));
        if (read(sd, message, sizeof(message) - 1) < 0)
        {
            perror("[client]Eroare la read() de la server.\n");
            return errno;
        }

        if (strlen(message) > 0)
        {
            printf("[client] %s\n", message);
        }

        // daca am comanda cu parola fac procesarea si comunicarea in cazu asta
        if (needsPassword) ···
    }
    close(sd);
    return 0;
}
```

Lista de comenzi si functionalitatea acestora:

- help – va afisa lista cu comenzi si formatul acestora
- exit – deconecteaza clientul
- new_card,<Nume>,<Adresa>,<Email> – adauga un nou card
- get_card <id> - va afisa cardul cu id-ul dat
- get_card – va afisa toate cardurile
- search_cards <text> - punem dupa ce vrem sa cautam, cumva functioneaza ca motorul de cautare pe google atunci cand apasam tasta F3, adica daca pun search_cards gmail,va afisa cardurile ce contin “gmail”, search_cards Ana va afisa toate cardurile cu numele Ana
- delete_card <id> -sterge cardul cu id respectiv daca există
- edit_card,<id>,<nume>,<adresa>,<email>- modifica dupa id cardul si inlocuieste cu ce nume,adresa si email am pus noi

Implementarea bazei de date

Baza de date gestioneaza eficient datele noastre despre cartile de vizita si ne asigura o stocare safe a acestora. Pozele si explicatia cum implementez baza de date au fost afisate mai sus,acum voi atasa o poza cum arata baza de date dupa cateva adaugari si modificari:

	ID	Name	Address	Email
	Filter	Filter	Filter	Filter
1	1	Ana Maria	Dimitrie Ralet	anamaria@gmail.com
2	2	Damian Mihai	Bulevardul 1 Mai	damian.mihai@yahoo.com
3	3	Anghel Marius	Strada Brancusi	anghel.marius@gmail.com
4	4	Bianca CT	Strada Iasilor	bianca.ct@gmail.com
5	5	Damian Mihai Marius	Strada Brasov	damian.marius@icloud.com
6	6	Bianca Tabacaru	Bulevardul Decebal	bianca.tabacaru@gmail.com
7	7	Serban Trimbitasu	Bulevardul cu flori	serban.trimbitasu@icloud.com
8	9	Adrian Elicopter	Strada Budeasa	adrian.elicopter@yahoo.com

5. CONCLUZII

In proiectul Visit Card dezvoltam un server TCP stabil si responsive capabil sa gestioneze multimple cereri de la clienti simultan avand “legata” o baza de date la acest server care asigura stocarea sigura si accesul rapid la date.

6. BIBLIOGRAFIE

- https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php
- https://docs.google.com/presentation/d/1reUzYxEYVd1WjqvNNy-wKNurJsS57_Oxn7ciMss6KU8/edit#slide=id.g261900c4cb6_0_125
- https://www.techtarget.com/searchnetworking/definition/TCP
- https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/
- https://nikhilroxtomar.medium.com/tcp-client-server-implementation-in-c-idiot-developer-52509a6c1f59
- https://zetcode.com/db/sqlitec/ (pentru baza de date)
- https://stackoverflow.com/questions/12984248/sqlite-in-c-c-sqlite3-exec-parameter-set-in-callback-function-is-pointing-to
- https://www.tutorialspoint.com/sql-using-c-cplusplus-and-sqlite