MyFileTransferProtocol

Balteanu A Andrei X4

January 11, 2023

Abstract

FTP Commands Project isi propune sa puna in evidenta functionalitatea unei structuri de tip server-client cu File Transfer Protocol (FTP). Aplicatia pune la dispozitie posibilitatea utilizatorului de a accesa, transfera si vizualiza diferite fisiere din cadrul serverului. Acest lucru este conditionat de existenta unei baze de date, in care sunt trecute persoanele care au acces la aplicatie(whitelist) sau sunt blocate(blacklist). Parola este criptata cu o cheie, care este trimisa odata cu credentialele la server.

1 Introducere

Aplicatia pune la dispozitia utilizatorilor, cu un cont existent, mai multe actiuni, cum ar fi accesarea contului, vizualizarea structurii de fisiere din cadrul serverului, accesarea si respectiv transferul de fisiere de catre utilizator prin intermediul aplicatiei client.

2 Tehnologii folosite

- **FTP (File Transfer Protocol)** pentru transfer de fisiere intre server si aplicatia client;
- **TCP/IP (Transmission Control Protocol/Internet Protocol)** pentru realizarea conexiunii propriu-zise intre server si client;
- **MySQL Server** pentru a retine in siguranta utilizatorii;
- **MySQL** pentru gestionarea bazei de date ;
- **C++ Programming Language;**(in cadrul sistemului de operare Ubuntu)
- **Visual Studio Code;**

2.1 TCP/IP

Conexiunea intre doua masini diferite se realizeaza de cele mai multe ori prin intermediul Internetului. Pentru a putea crea o conexiune stabila intre doua masini, a aparut protocolul standard pentru transmisie de date prin internet, acesta fiind TPC/IP. Acesta este prezent pe mai multe niveluri, dupa cum se poate observa in diagrama de mai jos.

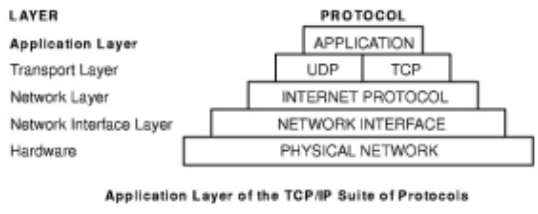

Figure 1: Nivelurile TCP

2.2 FTP

FTP (file transfer protocol) este un protocol pentru accesul la fisiere aflate pe servere din retele de calculatoare particulare sau din internet. El foloseste o retea clasica de tip TCP/IP.Primele metode de utilizare ale FTP constau practic in instructiuni in linie de comanda. Interfetele grafice, cunoscute si sub numele de clienti FTP au fost dezvoltate ulterior, astfel incat astazi le regasim chiar instalate default pe anumite sisteme de operare.

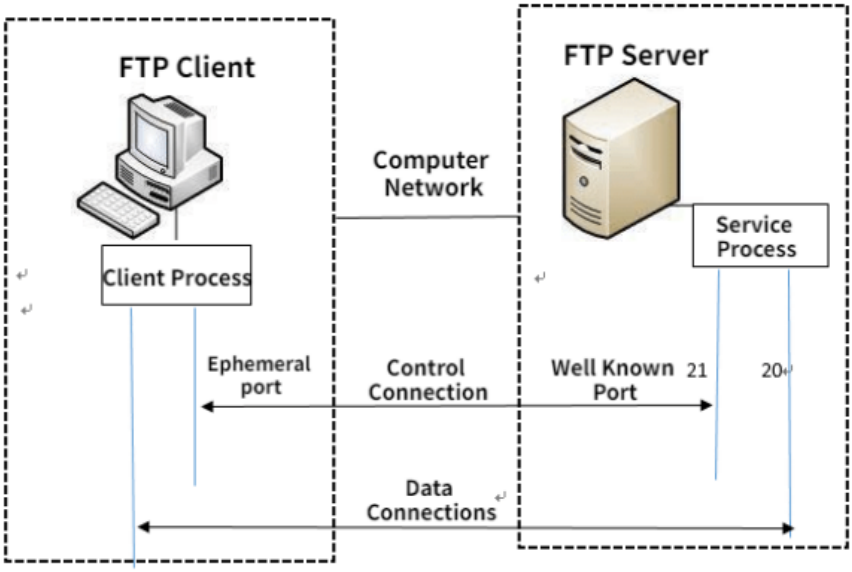

Figure 2: FTP Ilustrare

2.3 MySQL and MySQL Server

MySQL este un sistem de gestiune a bazelor de date relationale, fiiund o componenta cheie a grupului LAMP(Linux, Apache, MySQL, PHP).
MySQL Server este folosit pentru adaugare, stocare si procesare a datelor stocate intr-o baza de date.

Figure 3: My Sql logo

3 Arhitectura aplicatiei

Arhitectura cuprinde o baza de date pentru conturile utilizatorilor, multitudinea de fisiere de pe server, serverul propriu-zis si aplicatia client cu interfata grafica care face legatura dintre utilizatori si datele/actiunile pe care doresc sa le acceseze.
Datele sensibile de conectare aferente conturilor (parolele) vor fi criptate la nivelul clientului si decriptate la nivelul serverului pentru a spori securitatea. Astfel, odata ajunsa parola criptata la server,

acesta va permite sau nu accesul utilizatorului in baza mecanismului 'whitelist-blacklist', acesta va fi descris in sectiunea 'Baza de date'.

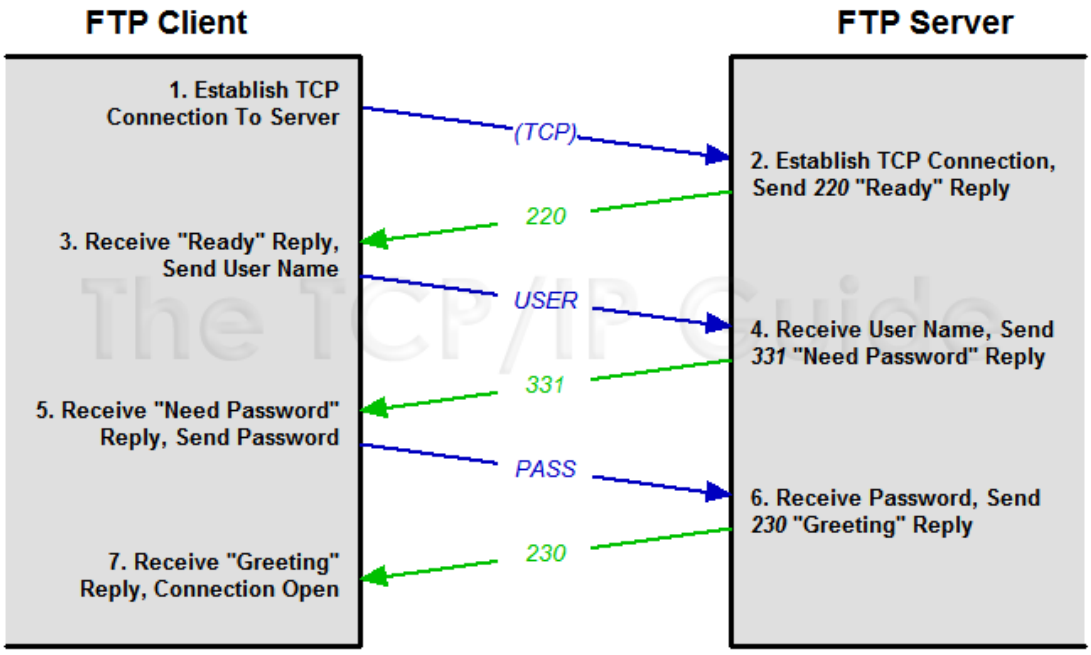

Figure 4: Arhitectura aplicatiei

3.1 Baza de date

Folosim o baza de data constituita dintr-o tabela, users, cu 4 campuri:
-**user-id(INT)**: este un identificator unic a unui cont detinut de catre un client
-**username(TEXT)**: numele folosita de utilizator pentru folosirea contului
-**password(TEXT)**: parola folosita de utilizator pentru folosirea contului
-**blacklisted(INT)**: 1 daca contul este blocat(blacklisted), 0 altfel.
Utilizatorul isi va introduce id-ul si parola, acestea fiind stocate in baza de date numita users.Pentru a sporii siguranta aplicatiei, atunci cand utilizatorii vor incerca sa se conecteze, parola va fi trimisa catre server criptata. Daca accesul ii este permis celui ce incearca sa se conecteze, un mesaj precum "Conectat cu succes" va fi primit de catre utilziator, in caz contrar un mesaj "Cont blocat sau neexistent".

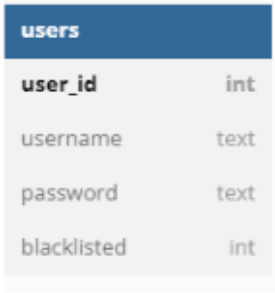

Figure 5: Diagrama tabelei

4 Use case

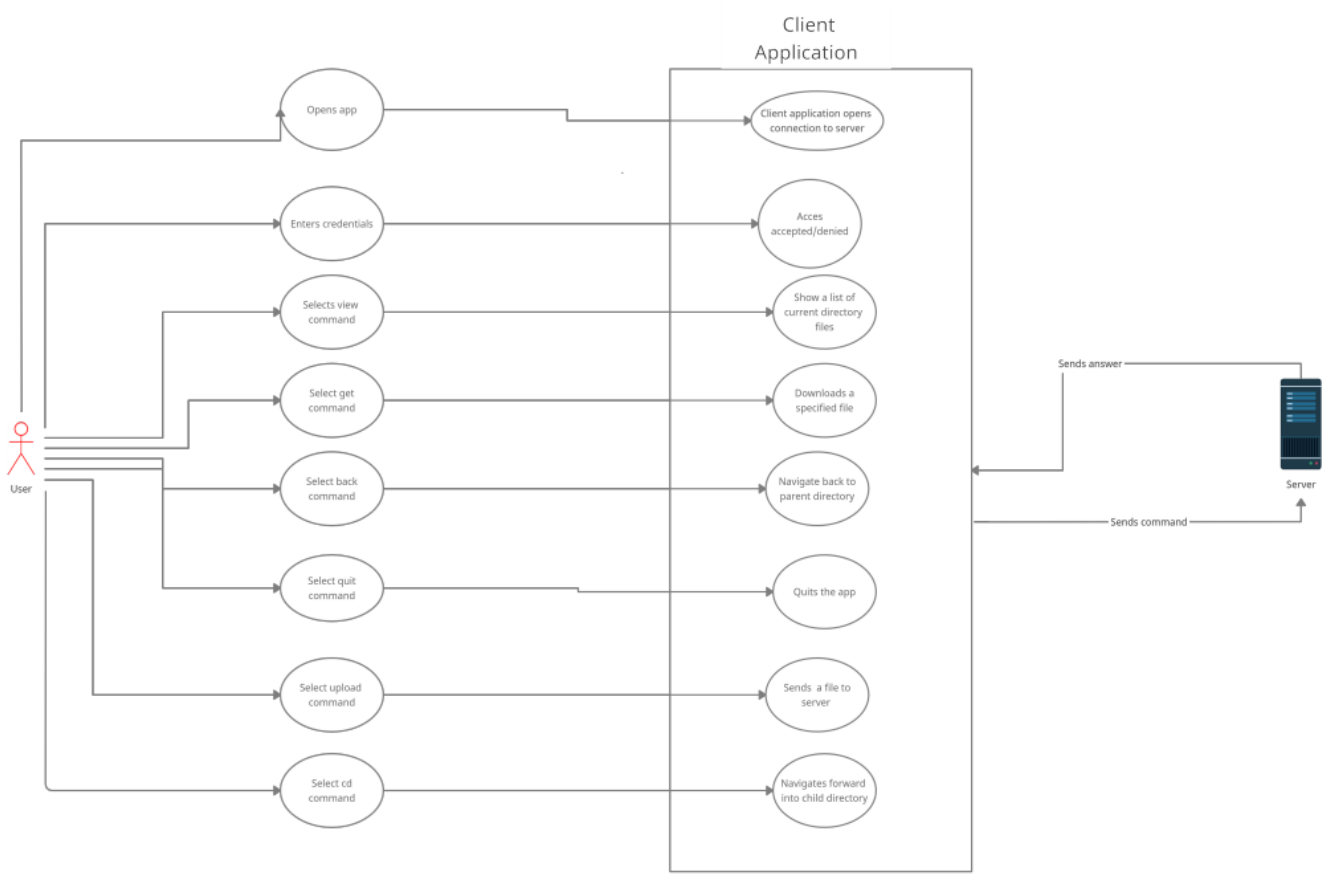

Figure 6: Use case diagram

5 Detalii de implementare

Aplicatia, inainte de toate, pentru a putea functiona trebuia sa faca legatura dintre client si server. Acest lucru se poate face creand un socket in server.

```
// Creates a TCP socket id from IPV4 family
sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Error handling if socket id is not valid
if (sockfd < 0)
{
        printf("Error in connection.\n");
        exit(1);
}

printf("Server Socket is created.\n");
```

Figure 7: Cod pentru creare socket-ului in server

Dupa crearea socket-ului in server, vom incerca sa ne connectam din client la el.

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

Figure 8: Cod din client pentru connectarea la socket-ul serverului

Daca conexiunea s-a realizat cu succes:
-**server**:un proces copil este creat (cu ajutorul primitevei fork) pentru a gestiona independent clientul. Serverul asteapta credentialele utilizatorului.
-**client**: utilizatorul primeste un mesaj care il indruma sa introduca credentialele pentru a folosi aplicatia. Apoi username-ul, parola criptata si cheia de cripatare sunt trimise catre server intr-un singur string.

```
char *format_credentials(char *username, char *password)
{

  printf("Username:");
  scanf("%s", username);
  printf("You entered the username:%s\n", username);

  printf("Password:");
  scanf("%s", password);
  printf("You entered the password:%s\n", password);

  char *temporary;
  temporary = (char *)malloc((strlen(username) + strlen(password) * 2) * sizeof(char));

  strcpy(temporary, username);
  strcat(temporary, " ");

  int key = rand() % 21;
  char int_str[20];
  sprintf(int_str, "%d", key);

  char *crypted_password;
  crypted_password = (char *)malloc(strlen(password) * sizeof(char));
  strcpy(crypted_password, crypt_password(password, key));

  strcat(temporary, crypted_password);
  strcat(temporary, " ");

  strcat(temporary, int_str);
  return temporary;
}
```

Figure 9: Cod din client pentru prelucrarae username-ului, parolei criptate si a cheii de criptare

```
char *crypt_password(char *password, int key)
{
  for (int i = 0; i < strlen(password); i++)
  {
    password[i] -= key;
  }
  return password;
}
```

Figure 10: Cod din client pentru criptarea parolei cu o cheie random

In server, se prelucreaza datele si se verifica daca , username-ul si parola trimise de client, sunt asociate unui cont care are acces la aplicatie.
Daca credentialele sunt ok, un mesaj aferent este trimis clientului. Apoi clientului ii este afisata lista de comenzi pe care le poate folosi in aplicatie.

Figure 11: Cod din server pentru prelucrarea datelor primite de la client

```
int check_credentials(char **credentials)
{
        MYSQL *conn;
        MYSQL_RES *res;
        MYSQL_ROW row;
        char *server_db = "localhost";
        char *user = "andrei";
        char *password_db = "andreib1234";
        char *database = "accounts";
        conn = mysql_init(NULL);

        if (!mysql_real_connect(conn, server_db, user, password_db,
                                                database, 0, NULL, 0))
        {
                fprintf(stderr, "%s\n", mysql_error(conn));
                exit(1);
        }

        char query[100];
        strcpy(query, "select count(*) from account where username='");
        strcat(query, credentials[0]);
        strcat(query, "' and password ='");
        strcat(query, credentials[1]);
        strcat(query, "';");
        if (mysql_query(conn, query))
        {
                fprintf(stderr, "%s\n", mysql_error(conn));
                exit(1);
        }
        res = mysql_use_result(conn);
        row = mysql_fetch_row(res);
        if (strcmp(row[0], "1") == 0)
        {
                printf("User %s logged in succesfully!\n", credentials[0]);
                return 1;
        }
        else
        {
                printf("User tried and failed to login with username %s and password %s!\n", credentials[0], credentials[1]);
                return 0;
        }
}
```

Figure 12: Cod din server pentru verificare credentialelor

6 Comenzi

```
void display_meniu()
{
  printf("Choose your command:\n");
  printf("1.List subdirectories(ls)\n");
  printf("2.Change directory(cd)\n");
  printf("3.Download\n");
  printf("4.Upload\n");
  printf("5.Print working directory(pwd)\n");
  printf("6.Quit\n");
  printf("Your command:");
}
```

Figure 13: Lista de comenzi pe care clientul le poate folosi.

6.1 Comanda ls

Cu aceasta comanda clientul poate sa vada toate fisierele dint-un anumit folder aflat pe server.Aceasta comanda este utila cand utilizatorul nu stie exact numele unui fisier, si pentru a nu descarca fisiere de care nu este interesat, cu ajutorul comenzii preview el va gasi mult mai usor fisierul de care are nevoie.

```
if (strcmp(message_from_client, "1") == 0)
{
        char *filename;
        filename = (char *)malloc(256 * sizeof(char));
        strcpy(filename, get_ls_filename_from_username(credentials[0]));

        system(filename);

        memset(message_to_client, 0, 256);
        strcpy(message_to_client, get_content_of_ls(filename));
        write(clientSocket, message_to_client, 256);
}
```

Figure 14: Cod server pentru primirea comenzii ls

```
char *get_content_of_ls(char *filename)
{
        char *temporary;
        temporary = (char *)malloc(256 * sizeof(char));
        strcpy(temporary, &filename[4]);

        char *content_of_file = read_file(temporary);
        remove(temporary);

        return content_of_file;
}
```

Figure 15: Procesarea comenzii ls

6.2 Comanda cd

Aceata comanda este folosita pentru ca utilizatorul sa poate naviga mai usor prin directoarele serverului.Cum in toate cazurile pe server sunt mai mult decat un director, clientul trebuie sa aiba posibilitatea sa navigheze prin toate pentru a ajunge la fisierul dorit.

```
if (strcmp(message_from_client, "2") == 0)
{
        memset(message_from_client, 0, 256);
        read(clientSocket, message_from_client, 256);
        memset(message_to_client, 0, 256);
        if (chdir(message_from_client) == 0)
        {
                strcpy(message_to_client, "1");
        }
        else
        {
                strcpy(message_to_client, "0");
        }
        write(clientSocket, message_to_client, 256);
}
```

Figure 16: Cod server pentru primirea si procesarea comenzii cd

6.3 Comanda download

Clientul poate sa descarce un fisier de pe server folosind comanda get. El trebuie sa introduca numele fisierului la care doreste sa aiba acces.Fisierul va fi descarcat in folderul unde se afla codul pentru client.

```
if (strcmp(message_from_client, "3") == 0)
{
        memset(message_from_client, 0, 256);
        read(clientSocket, message_from_client, 256);
        char *file_to_send;
        file_to_send = (char *)malloc(256 * sizeof(char));

        strcpy(file_to_send, message_from_client);
        fp = fopen(file_to_send, "r");
        if (fp == NULL)
        {
                perror("[-]Error in reading file.");
                exit(1);
        }
        printf("%s\n", message_from_client);
        send_file(fp, clientSocket);
        fclose(fp);
        //  printf("File sended!");
}
```

Figure 17: Cod server pentru primirea comenzii download

```
void send_file(FILE *fp, int clientSocket)
{
        int n;
        char data[SIZE] = {0};
        int count = 0;
        while (1)
        {
                if (fgets(data, SIZE, fp) != NULL)
                {
                        if (write(clientSocket, data, sizeof(data)) == -1)
                        {
                                perror("[-]Error in sending file.");
                                break;
                        }
                        printf("%d-%s", count, data);
                        bzero(data, SIZE);
                        count += 1;
                }
                else
                {
                        printf("\nend of file\n");
                        bzero(data, SIZE);
                        strcpy(data, "stop");
                        if (write(clientSocket, data, sizeof(data)) == -1)
                        {
                                perror("[-]Error in sending file.");
                                break;
                        }
                        break;
                }
        }
}
```

Figure 18: Cod server pentru procesarea comenzii download

```
void write_file(int sd,char* file_name)
{
  int n;
  FILE *fp;
  char buffer[SIZE];

  fp = fopen(file_name, "a+");
  if (fp == NULL)
  {
    perror("[-]Error in reading file.");
    exit(1);
  }
  int count = 0;
  while (1)
  {
    if (read(sd, buffer, sizeof(buffer)) <= 0)
    {
      perror("[-]Error in receiving file.");
      break;
    }
    if (strcmp(buffer, "stop") == 0)
    {
      printf("\ngotta stop..\n");
      break;
    }
    fprintf(fp, "%s", buffer);
    printf("%d-%s", count, buffer);
    count += 1;
    bzero(buffer, SIZE);
  }
  fclose(fp);
}
```

Figure 19: Cod client pentru crearea fisierului primit de la server

6.4 Comanda pwd

Pentru ca utilizatorul sa poata vedea path-ul unde se afla, comanda pwd va afisa pe ecran path-ul curent.

```
if (strcmp(message_from_client, "5") == 0)
{
        printf("%s", message_from_client);
        char *file_name;
        file_name = (char *)malloc(256 * sizeof(char));

        strcpy(file_name, get_filename_for_pwd(credentials[0]));
        system(file_name);

        memset(message_to_client, 0, 256);
        strcpy(message_to_client, get_content_of_pwd(file_name));

        write(clientSocket, message_to_client, 256);
}
```

Figure 20: Cod server pentru primirea comenzii pwd

6.5 Comanda quit

Clientul poate inchide conexiunea cu serverul trimitand numarul comenzii quit.

```
if (strcmp(message_from_client, "6") == 0)
{
        logged_in = 0;
        memset(message_to_client, 0, 256);
        printf("Client %s disconnected succesfully!", credentials[0]);
        strcpy(message_to_client, "Quit succesfully!");
        write(clientSocket, message_to_client, 256);
        break;
}
```

Figure 21: Cod server pentru primirea comenzii quit

6.6 Comanda create directory

Una dintre comenzile esentiale pentru un FTP, cea de creeare de fisiere.

```
else if (strcmp(message_from_client, "7") == 0)
{
    //reading the folder name from client
    memset(message_from_client,0,256);
    read(clientSocket,message_from_client,256);
    char *name_of_directory;
    name_of_directory = (char *)malloc(256 * sizeof(char));

    strcpy(name_of_directory, message_from_client);

    //creating the folder
    //check the result of mkdir
    int check;
    check = mkdir(name_of_directory,0777);
    memset(message_to_client,0,256);
    //sending the message to client
    if (!check){
        strcpy(message_to_client,"Directory created succesfully!");
    }
    else {
        strcpy(message_to_client,"Directory created unsuccesfully!");
    }
    write(clientSocket,message_to_client,256);

}
```

Figure 22: Cod server pentru primirea comenzii quit

6.7 Comanda delete directory

Ca si in cazul comenzii create directory, comanda delete directory este esentiala pt un FTP.

```
else if (strcmp(message_from_client, "8") == 0)
{
    //reading the name of the folder wants to delete
    memset(message_from_client,0,256);
    read(clientSocket,message_from_client,256);

    //formating the path string
    // in order to contain the path to the folder to be deleted
    char *name_of_directory;
    name_of_directory = (char *)malloc(256 * sizeof(char));
    strcpy(name_of_directory,message_from_client);



    char* path;
    path = (char *)malloc(256 * sizeof(char));
    memset(path,0,256);

    char *file_name;
    file_name = (char *)malloc(256 * sizeof(char));
    memset(file_name,0,256);

    strcpy(file_name, get_filename_for_pwd(credentials[0]));
    system(file_name);
    strcpy(path, get_content_of_pwd(file_name));

    path[strlen(path)-1]='\0';
    strcat(path,"/");
    strcat(path,name_of_directory);
    printf("%s\n",path);

    //remove_directory functions needs the full path
    //in order to delete the folder
    int check = remove_directory(path);

    printf("%d\n",check);
```

Figure 23: Cod server pentru primirea comenzii quit

7 Posibile imbunatatiri

1)O interfata grafica ar face folosirea aplicatiei mult mai usoara si mai user-friendly.
2)In caz in care un client nu are cont, sa existe posibilitatea creării unui cont direct din cadrul aplicatiei.
3)In caz ca un utilizator a uitat parola, sa existe posibilitatea resetarii acesteia. 4)Posibilitatea de a

comunica cu admnisistratorii serverului pentru a intelege de ce contul este blocat.

8 Concluzii

In concluzie, proiecul consta intr-o aplicatie client/server ce permite transferul de fisiere intre clienti si server. Accesul la server se face pe baza unui cont si este verificat intro baza de data creata in MySql. Pentru a sporii siguranta conturilor, parolele vor fi trimise criptat catre server, si acestea vor fi decriptate la nivelul serverului dupa o cheia specifica. Aplicatia pune la dispozitie un minim de comenzii pentru gestionarea fisierelor cum ar fi: upload,get,cd, cd .. (sau back), ls, PWD.

9 References

[1] Pagina curs
[2] Pagina seminar
[3] Documentatie MySQL
[4] Template Overleaf
[5] Template Raport
[6] Informatii FTP
[7] Informatii FTP(youtube)
[8] Informatii fork
[9] Informatii pid
[10] Informatii baze de date
[11] Informatii proces copil/parinte
[12] Informatii proces copil/parinte(youtube)
[13] Informatii conectare MySQL in C