

Personal Cloud Storage

Barbărașă Sebastian-George
FII, UAIC Iași

December 2023

1 Introduction

Personal Cloud Storage is a simple and fast tool to store your data in a secure cloud system. It provides its users with a login system which connects them via a unique username and a password. Once signed up and logged in, users can upload their files (or even folders) directly into the cloud. Similarly, they can list and download their data anywhere and anytime.

The software provides a user-friendly interface which helps people see their saved files and maneuver them as fast as possible. An advantage of storing your data in the cloud is that now you have access to it among all your devices that support the application. Privacy for your data is guaranteed since everything saved within the cloud is encrypted with a Vigenère key, a powerful and hard to crack encrypting method. The application also has redundancy implemented to protect the stored information at all costs.

2 Technologies Used

• TCP

Client-server communication is done through concurrent TCP/IP (Transmission Control Protocol/Internet Protocol), which is a connection oriented protocol responsible for data transfer. The information is split into independent packages and then reassembled back once arrived at the destination. In such manner no piece of information is lost during the transfer and when brought together, they will respect the original order. This is very important for a cloud system since losing even the smallest data package could lead to a disaster in the files tree of a user.

• SFML

SFML (Simple and Fast Multimedia Library) provides application programming interface for C++ and many more languages. I chose it because it delivers simple functionalities for creating and managing windows, handling input events (keyboard, mouse, etc.), and controlling the application loop.

• SQLite

I opted for SQLite as the database solution for my application due to its lightweight and efficient nature. It is a self-contained, serverless, and zero-configuration database engine, making it seamless to integrate into my application without the need for complex setup procedures. Its simplicity doesn't compromise on performance; SQLite is known for its speed and reliability, making it an ideal choice for applications where resources are a concern.

3 Application Structure

The application is composed out of 2 main programs, one which implements the server and one for the client. They are both connected to a socket which provides information interchange between them. The server is not limited to serving one client at a time since it creates a child process destined for each client. After creating a child process, the server goes on with accepting incoming connect request.

Once connected a client has access to a limited series of commands:

- **help**: provides general guidance on how to use the application and the syntax of the other commands
- **help [command name]**: provides guidance on how to use a particular command
- **register**: creates a new user account; unique username and a password with minimum 5 characters from which at least one numeric and password confirmation are needed
- **login [username]**: if username is existent, password is asked; if both are correct, client is logged in
- **logout**: logout from current connected user account
- **exit**: closes current session (implicit logout)

After logging in, additional commands, such as files management are available:

- **upload [filepath]**: uploads specified local file to user's personal cloud
- **download [filepath]**: downloads specified file from cloud to cloudDownloads folder
- **list/ls**: shows the total number of files stored in user's cloud and enumerates them
- **rename [old filename] [new filename]**: renames old file to new file if old file exists and no file with new file's name already exists
- **delete [filename]**: deletes specified file from cloud if it exists; confirmation is required

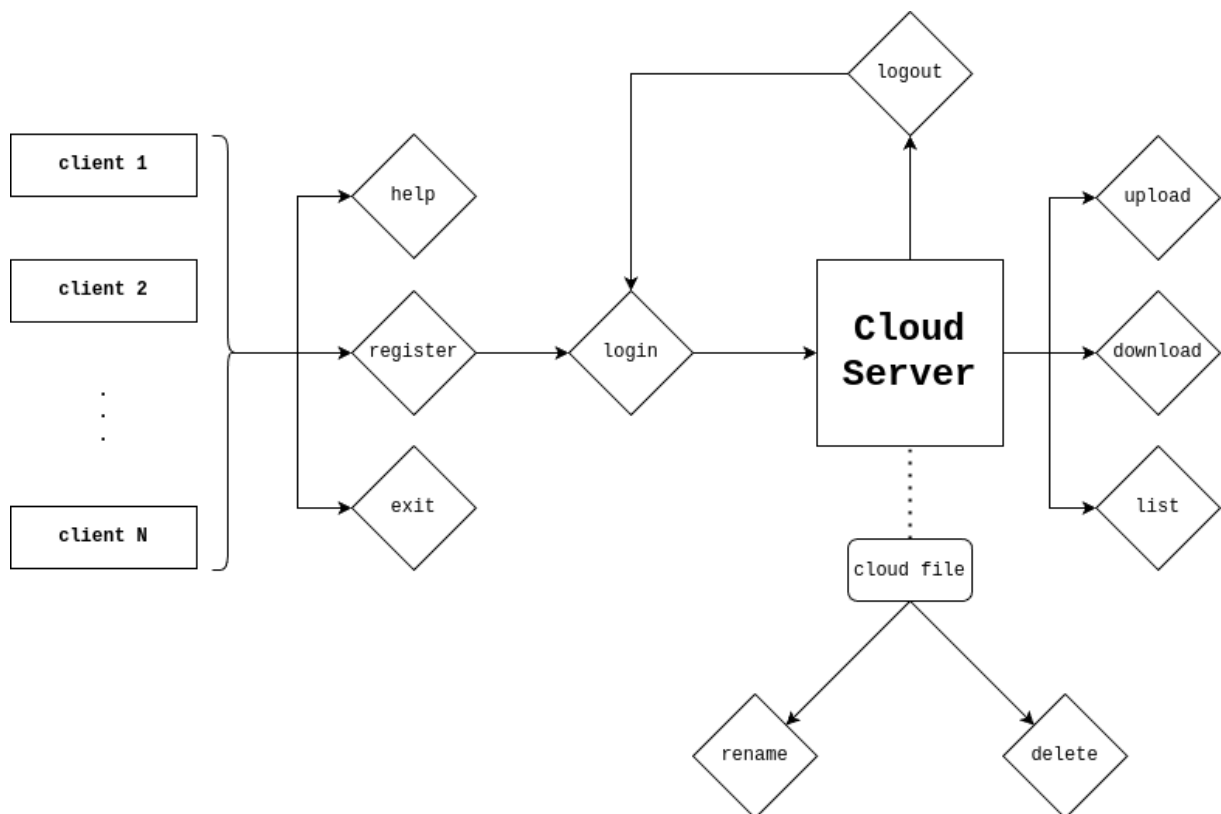


Figure 1: Application structure diagram

4 Implementation Aspects

- Transferring a file through a socket

```
//both parties are ready
char fileLocation[] = "cloud/";
strcat(fileLocation, basename(filename));
FILE *cloudFile = fopen(fileLocation, "w");
char sectionOfFile[SECTION_MAX + 1]; //+1 for null terminator

struct stat fileinfo;
if (read(clientFd, &fileinfo.st_size, sizeof(fileinfo.st_size)) == -1) {
    perror("file size read error");
    return;
}

int totalBytesRead = 0, i = 1;

while (totalBytesRead < fileinfo.st_size) {
    while ((bytesRead = read(clientFd, sectionOfFile, SECTION_MAX)) <= 0);
    sectionOfFile[bytesRead] = 0;

    //debug
    printf("<Section %d [%d bytes]: ", i++, bytesRead);
    for (int i = 0; i <= 20; i++)
        if (sectionOfFile[i])
            printf("%c", sectionOfFile[i]);
        else printf("?");
    printf(">\n");

    fwrite(sectionOfFile, 1, bytesRead, cloudFile);

    totalBytesRead += bytesRead;
}
fflush(NULL);
```

Figure 2: Snippet of server's upload part

Because of the immense sizes a file can have, it cannot be transmitted through a socket as a whole. To fix this, the application portions each file in smaller sections (`sectionOfFile`) which are then transferred one by one through the socket (first `while`). All these sections are not reassembled together at the end of the transmission; they are appended in the output file during the transmission (`fwrite` function) for more time and space efficiency. As a side note, an important thing that the server needs to do is to first read the file size (`fileinfo.st_size`), otherwise it couldn't know how much time to read sections from the client.

- Concurrently processing clients' commands

```
case 0:
    while (clientIsConnected(clientFd, (struct sockaddr*)&server, clientSize)) {
        printf("waiting for a command...\n");
        fflush(stdout);

        char command[COMMAND_MAX];
        int bytesRead;

        if ((bytesRead = read(clientFd, command, COMMAND_MAX - 1)) <= 0)
        {
            perror("read from client error / empty buffer");
            close(clientFd);
            continue;
        }
        command[bytesRead] = 0;
        printf("[Debug] bytes read: %d, received command: %s\n", bytesRead, command);
        fflush(stdout);

        char *whichCommand = strtok(command, " ");

        if (strcmp(whichCommand, "upload") == 0) uploadCommand(command, clientFd);
        else if (strcmp(whichCommand, "get") == 0) getCommand(command, clientFd);
        else unknownCommand(command, clientFd);
        printf("-----\n");
    }

    close(clientFd);
    exit(EXIT_SUCCESS);
default:
    close(clientFd);
    while (waitpid(-1, NULL, WNOHANG)); //WNOHANG: does not wait for children, used for getting the exit status
    continue;
```

Figure 3: Server's commands processor

As previously said, the server does not wait for any child program to finish its execution (WNOHANG parameter for the `waitpid` primitive). Each child processes its client inputted string via one `strtok` call to determine the command's name, then calls the function associated to that command which then checks for syntax, access or other errors and process it. To ensure no child processes are left hanging, client connectivity is checked before every input process (`clientIsConnected` function), as in the following snippet:

```
bool clientIsConnected(int clientFd, struct sockaddr* server, socklen_t clientSize) {
    if (getpeername(clientFd, server, &clientSize) == -1) {
        printf("connection terminated\n");
        printf("-----\n");
        return 0;
    }
    return 1;
}
```

Figure 4: Function which checks client connectivity





USERNAME  	PASSWORD  
<input type="text" value="Search column..."/>	<input type="text" value="Search column..."/>

Figure 5: Server database: LOGIN table

With the help of SQLite, Cloud Storage can save and go through its data in a very short time. For example managing users' accounts is now just a matter of querying the LOGIN table. It has 2 columns, from which one stores the primary key, which is the username. Because of that, it is granted that the username will be unique, so no more checkings are required when inserting a new username. The second column stores the password of the corresponding user and it cannot be null.

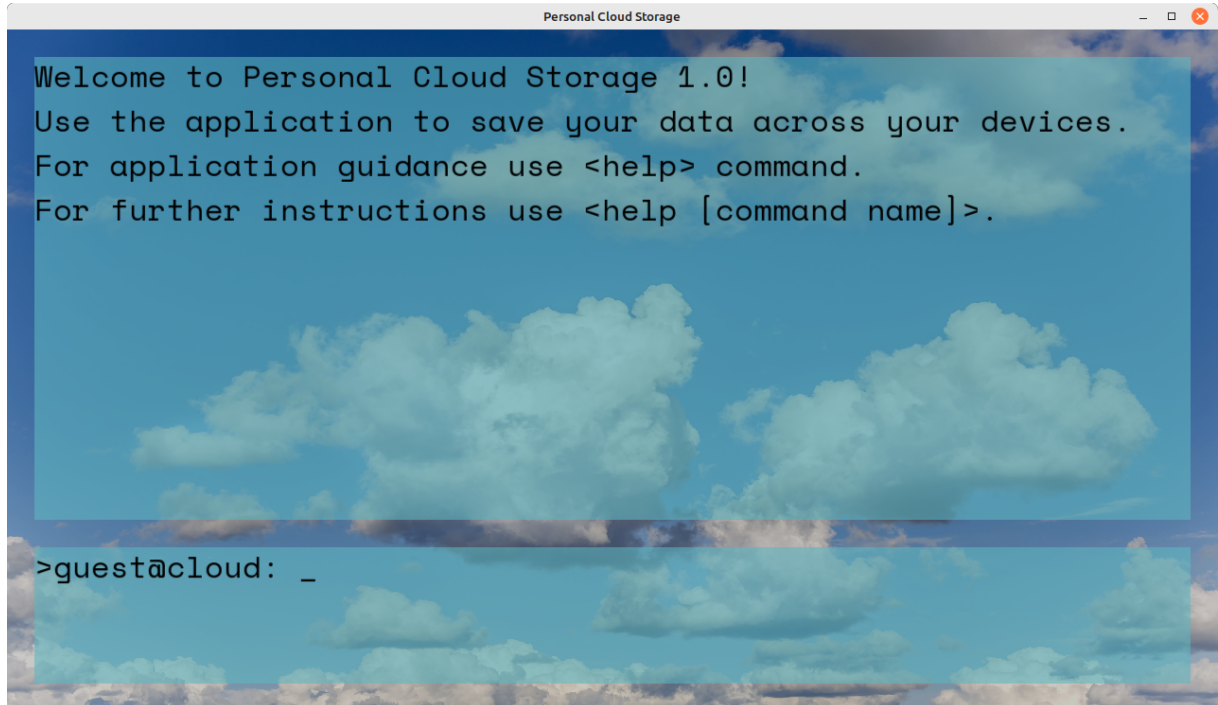


Figure 6: Server interface

The application provides a simple to use interface, made using SFML. It has 2 workspaces, one for users' input and one for the server answer. Shortcuts like CTRL + V (paste) and arrow keys (up / down) to access commands history can be used.

5 Conclusions

Having a fast way to secure your data in a manner that you have access to it across many devices is essential in our era. Personal Cloud System provides a solution designed purposely to help people manage their files faster. There are though some additions or improvements the application could use:

- ability to send/receive friend requests and share files along groups of friends
- an email verification system for each user account to provide a back-up plan if credentials are forgotten

6 Bibliography

- My computer networks laboratory supervisor's (Hrib Ecaterina) presentations; lots of thanks to her amazing work (and patience)
- Linux Users Manual
- Computer Networks course, FII, UAIC Iași
- Object Oriented Programming course, FII, UAIC Iași

- [Databases course, FII, UAIC Iași](#)
- [SFML documentation](#)
- [SQLite documentation](#)