Networks Assignment

Racquel, Simba, Tapera

March 2023

1 Introduction

The aim of the given assignment was to implement a client-server model in python that makes use of the TCP transportation link to ensure reliable transfer of data over the sockets created.

2 Functionality and Design of the application

The development of the file transfer app was quite integrate.The application made use of both local and imported functions that allowed the server to meet the specified requirements in the outlined assignment specification.

The implementation was developed with the following hierarchy:

```
CSC3002 Assignment
├── downloads
├── server_files
├── server.py
├── files.json
├── client.py
├── serv_utils.py
├── users.bin
├── filekeys.key
└── requirements.txt
```

2.1 Directory Components

- The downloads represents the download directory that will store all the files that are downloaded from the server onto the clients local machine.
- The server files directory has all the files that are currently uploaded onto the server. This directory will be the central directory that has the files that the clients have uploaded onto the server.
- The files.json file consists of the files names and their associated passwords.
- The script server.py is the script that contains the required code to start the server.
- The script client.py is the script that contains the required code to start the client and allow the client to interact with the server.

- The script serv_utils.py is a util file that contains the functions that will be called by the server in order to execute the tasks that the client calls for.
- The filekey.key file contains a unique hashing key that is used for when users are created. They have this specified key that allows them to be created as users.
- The users.bin has the entire directory file of users that are created.
- The requirements.txt file contains the prerequisites needed in order to run the chat application. It contains the modules needed for the application.

2.2 Structure of the Application

The application is based on the client-server model that allows the client to initiate a connection with the server that is ready to take connection from clients. The server is able to host and accommodate interactions with multiple clients at the same time. This is achieved through the use of multi threading. The transportation protocol that was implemented was the TCP protocol. This requires the two services to initiate a handshake protocol to confirm the connection.

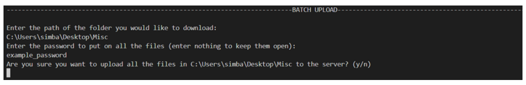
2.2.1 Server Implementation

- When the server is started, a socket connection is created and then bind-ed to the specified IP address and port number. The socket is then called to listen and the server is set up to service in coming connection calls.
  - The server also allows the user the option to specify the port number that it will be running on.
- To allow the user to shut down the server at any given time, a separate thread was created that will target the treading_pointer function. This function will run a while loop that accepts in coming connections from clients.
- When a client connects to the server, the server will create connection sockets for each client and then a thread will be created that will target the file_handling function.
  - The file_handling function will handle all the incoming requests from the client server and will send out the appropriate responds following a specified protocol that will be outline later on in the report. Essentially, this function handles all incoming requests that the client might make based on the features available from the server.
- If an anytime when the server is running and the user would like to close the server and disconnect all clients from the server, they can simply type "exit()" in the console and the server will disconnect all users and shut down. This will also terminate all running threads that are children to the main thread that the while loop is running on.

2.2.2 Client Implementation

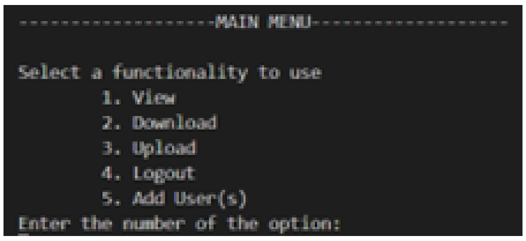
- <u>Simba</u> :

  I have implemented the client as a command line interface (CLI) with the core functionality of the file transfer app. When the user opens the app, they are immediately met with prompts to enter the IP address of the server and the port number it has error checking to make sure that



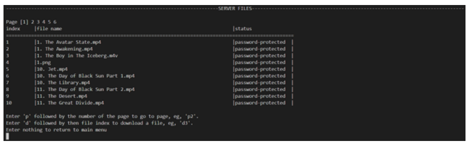
```
-----------------------------------------------------------------BATCH UPLOAD-----------------------------------------------------------------

Enter the path of the folder you would like to download:
C:\Users\simba\Desktop\Misc
Enter the password to put on all the files (enter nothing to keep them open):
example_password
Are you sure you want to upload all the files in C:\Users\simba\Desktop\Misc to the server? (y/n)
```

(a) Figure A.



```
------------------MAIN MENU------------------

Select a functionality to use
        1. View
        2. Download
        3. Upload
        4. Logout
        5. Add User(s)
Enter the number of the option:
```

(b) Figure B.



```
----------------------------------------------------SERVER FILES----------------------------------------------------

Page [1] 2 3 4 5 6
index      |file name                                                   |status
=================================================================================================
1          |1. The Avatar State.mp4                                     |password-protected  |
2          |1. The Awakening.mp4                                        |password-protected  |
3          |1. The Boy in The Iceberg.m4v                               |password-protected  |
4          |1.png                                                       |password-protected  |
5          |10. Jet.mp4                                                 |password-protected  |
6          |10. The Day of Black Sun Part 1.mp4                         |password-protected  |
7          |10. The Library.mp4                                         |password-protected  |
8          |11. The Day of Black Sun Part 2.mp4                         |password-protected  |
9          |11. The Desert.mp4                                          |password-protected  |
10         |11. The Great Divide.mp4                                    |password-protected  |

Enter 'p' followed by the number of the page to go to page, eg, 'p2'.
Enter 'd' followed by then file index to download a file, eg, 'd3'.
Enter nothing to return to main menu
```

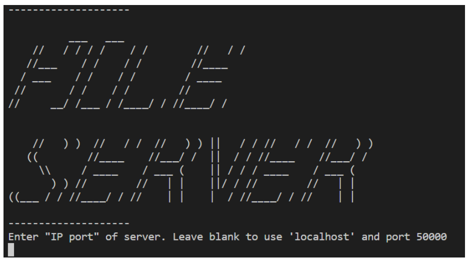
(a) Figure C.

Figure 1: Simba's Client.

the user enters viable input. From log in, users are presented with a main menu from which they can select the 5 main menu functions: "View files", "Download file(s)", "Upload file(s)", "Logout", and "Add user(s)", the last one of which only shows for admin users. The View files option gets all the files from the server and uses pagination to display the files in pages of 10, ordered alphabetically. The user is given the commands to switch between the pages that they must enter to view files on other pages, e.g., entering "p3" to go to the 3rd page of the file list. Users also get a chance to download files straight from the directory by giving a command which is carefully explained on the interface, e.g., entering "d12" to download the 12th file on the list. From there they also have an option to return to the main menu. The Download files option allows them to type the filename and download it. The Upload files option has two modes which make are meant to give the user options: they can opt for the Interactive Upload function in which they can upload one file at a time with its password (if they want to) or Batch Upload function which allows them to upload all the files from folder using a single password for all of them. The Batch Upload function uploads the file with the same name that they are saved as on the client machine. Batch Upload also has an option to upload files in a directory recursively meaning that if they are sub-directories in the folder, it also uploads the contents of those sub-directories. The interfaces have a title at the top of the command line so that the user knows where they are at every point in the interaction. The app also clears the page after every interaction to make visibility easier. A CLI was chosen since it is simple to implement and understand for regular users. This implementation has error checking at every point to make sure that the user always enters correct input. Features like the View and download also allow the user to use the application with ease.

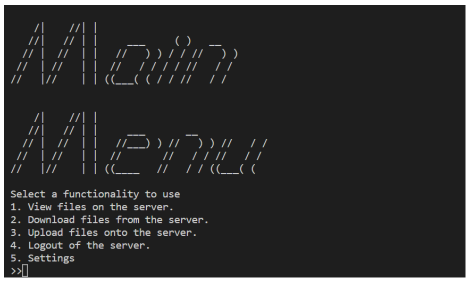
- **Racquel** :

  The client that was implemented allows the user to access all the functionality offered by the server. The interface in which the client interacts with is the command line. This was implemented for ease of use and universal use on all platforms. When the client connects to the server, they are allowed to specify the IP address and port number that they want to connect to, as seen in Figure One .



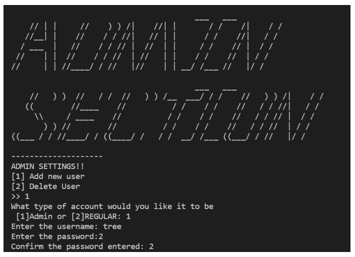
```
--------------------
--------------------
Enter "IP port" of server. Leave blank to use 'localhost' and port 50000
```

(a) Figure A.



```
Select a functionality to use
1. View files on the server.
2. Download files from the server.
3. Upload files onto the server.
4. Logout of the server.
5. Settings
>>
```

(b) Figure B.



```
--------------------
ADMIN SETTINGS!!
[1] Add new user
[2] Delete User
>> 1
What type of account would you like it to be
 [1]Admin or [2]REGULAR: 1
Enter the username: tree
Enter the password:2
Confirm the password entered: 2
```

(a) Figure C.

Figure 2: Racquel's Client.

In order to abstract and encapsulate the code, the client is separated into functions that are called when specific events take place when the application is operating. The main method acts as a driver that uses the functions defined to server the client interaction.

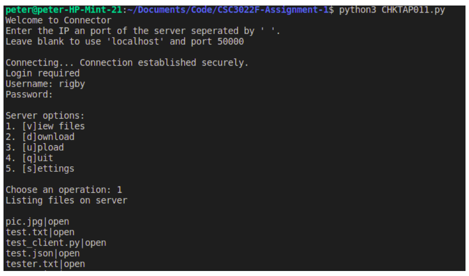
There is administrative functionality added into the client. This allows the administrative user the ability to add users and delete users. When the user logs into the application, they are prompted to enter their username and password which was registered by the admin user.The use of the admin function can be seen in Figure C.

The main functionality of the client allows users to upload and download files onto the server. The user that is interacting with the client will get a menu displayed that will allow them to choose how they would like to interact with the server. As seen in Figure B. The users then choose their options through typing on the CMD.

To add some graphics to the client, the application makes use of python Figlet. This adds an illustrative graphic to the CMD and makes it more interactive with the user.

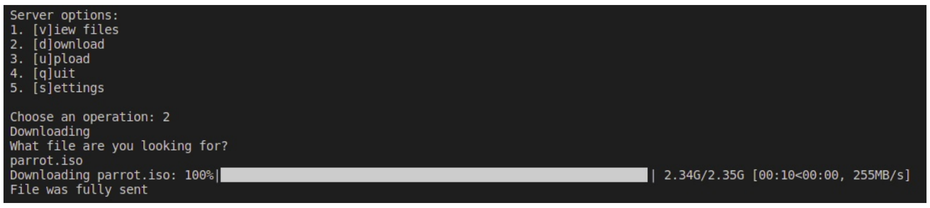
The files that are displayed to the client when the view function is called is organised in a clear way that allows the user to easily read what is stored on the server. This allows for easy of navigation through the application.

The client was extensively tested on different file sizes. That varied in different sizes and were of different media types. A try, catch block was also implemented into order to catch for errors that may arise. This try, catch will simply break from the server and inform that client as to what error had occurred.



```
peter@peter-HP-Mint-21:~/Documents/Code/CSC3022F-Assignment-1$ python3 CHKTAP011.py
Welcome to Connector
Enter the IP an port of the server seperated by ' '.
Leave blank to use 'localhost' and port 50000

Connecting... Connection established securely.
Login required
Username: rigby
Password:

Server options:
1. [v]iew files
2. [d]ownload
3. [u]pload
4. [q]uit
5. [s]ettings

Choose an operation: 1
Listing files on server

pic.jpg|open
test.txt|open
test_client.py|open
test.json|open
tester.txt|open
```

(a) Figure A.



```
Server options:
1. [v]iew files
2. [d]ownload
3. [u]pload
4. [q]uit
5. [s]ettings

Choose an operation: 2
Downloading
What file are you looking for?
parrot.iso
Downloading parrot.iso: 100%|██████████████████████████████████████████| 2.34G/2.35G [00:10<00:00, 255MB/s]
File was fully sent
```

(b) Figure B.

Figure 3: Tapera's Client.

- **Tapera**:

  A command-line client was favoured for its ease of use and compatibility with different interfaces. A modular approach was adopted in development. Most client operations are defined as functions with the main method serving as a driver. These use try-except blocks for error handling to prevent unusual behaviour.

  This client provides the user with access to the main server functions of uploading, downloading and listing files. User management is also provided as admins are able to add more users. All options are accessed with simple terminal input, typically the first letter of the option.

2.2.3 Features of the Application

The application has a few features that were both requirements and added on features. The one feature added was the file permissions feature. When a client uploads a file, they can state if the file is protected or not. An open file will require no password to access and download, however, a protected file will require a password for users to enter. The protection on a file will be specified when a client uploads a file onto the server and the password will be required when a user downloads a file from the server.

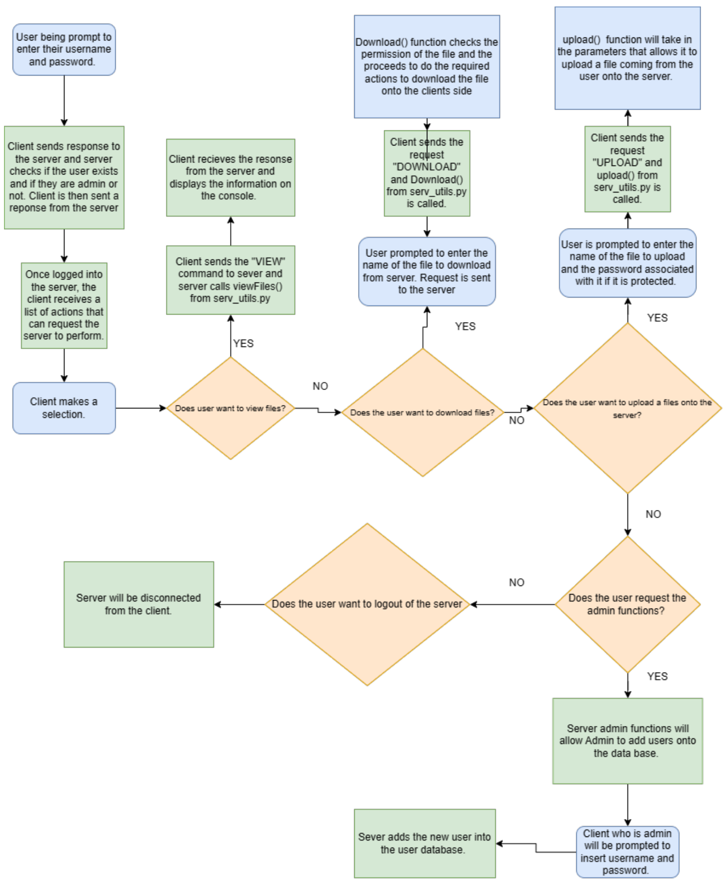
Another aspect that the application has, is the ability to log into and out of the application with a username and password. A user can either be an admin user or a regular user, where admin users are given special access to add users and delete users.

To further add to the security features of the model. There was file encryption and hashing used when users where created on the database. This was used with various python models. Hashing was also used to validate the transfer of files from the server to the client. While file encryption is used to handle user data.

2.2.4 Functionality Implementation of File_Handling Function

The function file_Handilng is where the main functionality of the server is based. Within this function, it will take connection requests from the client and respond to them calling the relevant files from the serv_utils.py function.

The following flow chart shows the interactions that the client will face with the server.



User being prompt to enter their username and password.
Client sends response to the server and server checks if the user exists and if they are admin or not. Client is then sent a reponse from the server
Once logged into the server, the client receives a list of actions that can request the server to perform.
Client makes a selection.
Does user want to view files?
YES
Client sends the "VIEW" command to sever and server calls viewFiles() from serv_utils.py
Client recieves the resonse from the server and displays the information on the console.
NO
Does the user want to download files?
YES
User prompted to enter the name of the file to download from server. Request is sent to the server
Client sends the request "DOWNLOAD" and Download() from serv_utils.py is called.
Download() function checks the permission of the file and the proceeds to do the required actions to download the file onto the clients side
NO
Does the user want to upload a files onto the server?
YES
User is prompted to enter the name of the file to upload and the password associated with it if it is protected.
Client sends the request "UPLOAD" and upload() from serv_utils.py is called.
upload() function will take in the parameters that allows it to upload a file coming from the user onto the server.
NO
Does the user request the admin functions?
NO
Does the user want to logout of the server
Server will be disconnected from the client.
YES
Server admin functions will allow Admin to add users onto the data base.
Client who is admin will be prompted to insert username and password.
Sever adds the new user into the user database.

Figure 4: Flow chart of chat application

2.2.5 Protocol of the application

A protocol is a set of rules that describes how an application should format and process data. The application has been implemented using the TCP protocol. This allows for the use of reliable data delivery. When implementing the chat application, our group followed the structure below:

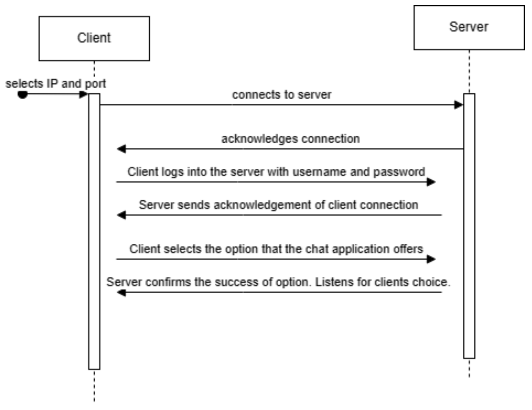
Table 1: Protocol Outline

Command	**Message information**	**other relevant information**

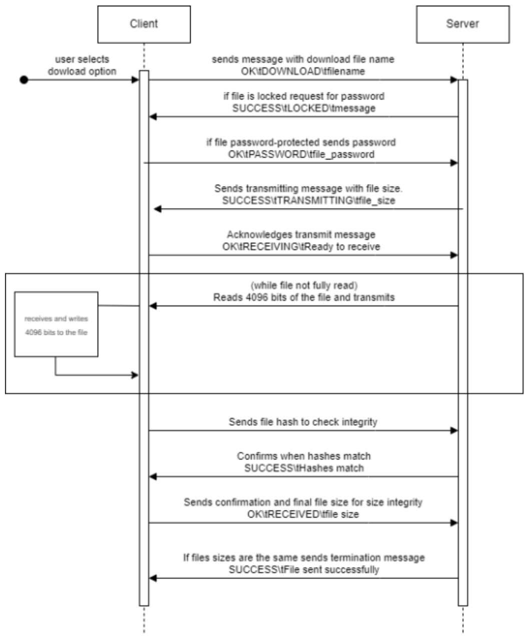
The commands component is sent to the server from the client to indicate the desired functionality that the client wants the chat application to execute. Refer to table to that explains the protocol commands.

Table 2: Explanation of Commands

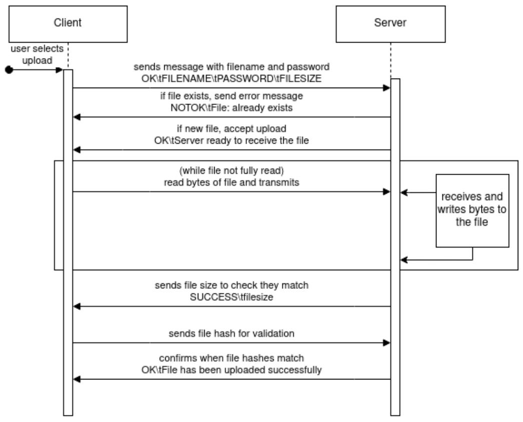
Command	**Explanation**
HANDSHAKE	This is associated with the HANDSHAKE between the client and server.
ERROR	An error has occurred in processing the request.
OK	The request was processed.
SUCCESS	The server executed the command successfully.
FAILURE	The server failed to execute the command.
VIEW	The client is requesting to view files in the server.
DOWNLOAD	The client is requesting to download files from the server.
UPLOAD	The client is requesting to upload a file onto the server.
LOGOUT	The client is requesting to logout from the server.
ADMIN	Admin functions are requested.


Client
Server
selects IP and port
connects to server
acknowledges connection
Client logs into the server with username and password
Server sends acknowledgement of client connection
Client selects the option that the chat application offers
Server confirms the success of option. Listens for clients choice.

(a) General Implementation.


Client
Server
user selects dowload option
sends message with download file name OK\tDOWNLOAD\tfilename
if file is locked request for password SUCCESS\tLOCKED\tmessage
if file password-protected sends password OK\tPASSWORD\tfile_password
Sends transmitting message with file size. SUCCESS\tTRANSMITTING\tfile_size
Acknowledges transmit message OK\tRECEIVING\tReady to receive
(while file not fully read) Reads 4096 bits of the file and transmits
receives and writes 4096 bits to the file
Sends file hash to check integrity
Confirms when hashes match SUCCESS\tHashes match
Sends confirmation and final file size for size integrity OK\tRECEIVED\tfile size
If files sizes are the same sends termination message SUCCESS\tFile sent successfully

(b) Download Implementation.


Client
Server
user selects upload
sends message with filename and password OK\tFILENAME\tPASSWORD\tFILESIZE
if file exists, send error message NOTOK\tFile: already exists
if new file, accept upload OK\tServer ready to receive the file
(while file not fully read) read bytes of file and transmits
receives and writes bytes to the file
sends file size to check they match SUCCESS\tfilesize
sends file hash for validation
confirms when file hashes match OK\tFile has been uploaded successfully

(a) Upload Implementation.

Figure 5: Sequence Diagrams.

2.2.6 Sequence diagrams of the chat application

The diagrams above show how the interaction between the client and the server will take place. Figure A gives an overall idea of what is expected from each functionality within the application. Figure B outlines the download functionality and Figure C outlines the upload functionality.