CLIENT-SERVER PROTOCOL

Group 24

Mbali Myeza (MYZMBA002)

Prosper Mambambo (MMBPRO002)

Sisipho Tukushe (TKSSIS001)

University of Cape Town
CSC3002F

Overview

The basic model of computer networks is composed of routers, devices, and links to enable the sharing of information and/or data between different hosts through a wireless system. At the network’s edge is an array of end systems consisting of various hosts, such as desktop computers and servers, on which data is stored. In many cases various devices may want to exchange or access data that exists in another host in the same network or in other networks. This exchange of data between devices is made possible through the communication links of the internet, that transport data from one device to another. The access that a network has to a host's data is enabled through a set of protocols defined by the specific host, which are necessary to allow easy communication between different devices which may have different internal processes.

The internet's layered protocol stack is organized into a series of five layers, namely the application, transport, network, link, and physical layers, and they all perform specific services that all assist in the transfer of data across the network. This layered approach is used as an efficient means of error detection and productivity control, if one of the layers happened to require maintenance. The layers are interconnected, and each layer is reliant on the seamless execution of the services belonging to the layer below it.

The focus of this study is on the functionalities of the application layer. This layer is responsible for supporting the implementation of application processes and uses message protocols to communicate between applications that are run on different hosts. The study focuses on the client server paradigm, which can be understood as a model for distributed computer protocols. The client-server model employs a request and response application, where a client requests a connection from a server prior to the connection being made, and they can begin sharing files between each other in the network.

In this study, a protocol was created to manage and define the order of messages sent and received by the different network applications, and to determine what action must be taken on each message transmission. To elaborate how the implemented protocol works, a single server and two differing clients were created to interact with each other using the defined protocol. The client applications are built differently from each other, but the way they communicate with the server is the same, showing that a set protocol was correctly defined and implemented.

The Protocol

The implemented protocol uses a TCP connection, which means that before the client and server can communicate and send file to each other, they need to first form a connection. The server application is always running and listening to requests of connections from clients. When the client runs, it first forms a connection with the server. Once the server accepts the connection, the client can then send a message to the server, which specifies what the user wants to do on the server. The client acts as an interface or application that a user interacts with, and the user uses it to specify the actions or commands it wants to perform on the server through the network. The client then sends these commands to the server, which performs a specified action, and sends the message back to the client.

The user can either list files that are available for download in the server, upload files to the server (specifying the confidentiality), download files from the server, delete files that are in the server, request to see the Help function, which helps them to see which commands they should use for each action they want to do, and they can logout from the server. All these actions that a user performs through the client can be done within one client-server connection. This means that the user can download, delete, and upload files on the server without having to disconnect every time they have performed an action.

Features

Upload

The upload function is responsible for allowing users to upload files from their client to the server. The feature offers extra capabilities where it can encrypt files and protect them using the users email address and surname. Here is a sequence diagram depicting the flow of messages between the client and server in Figure 1 below:

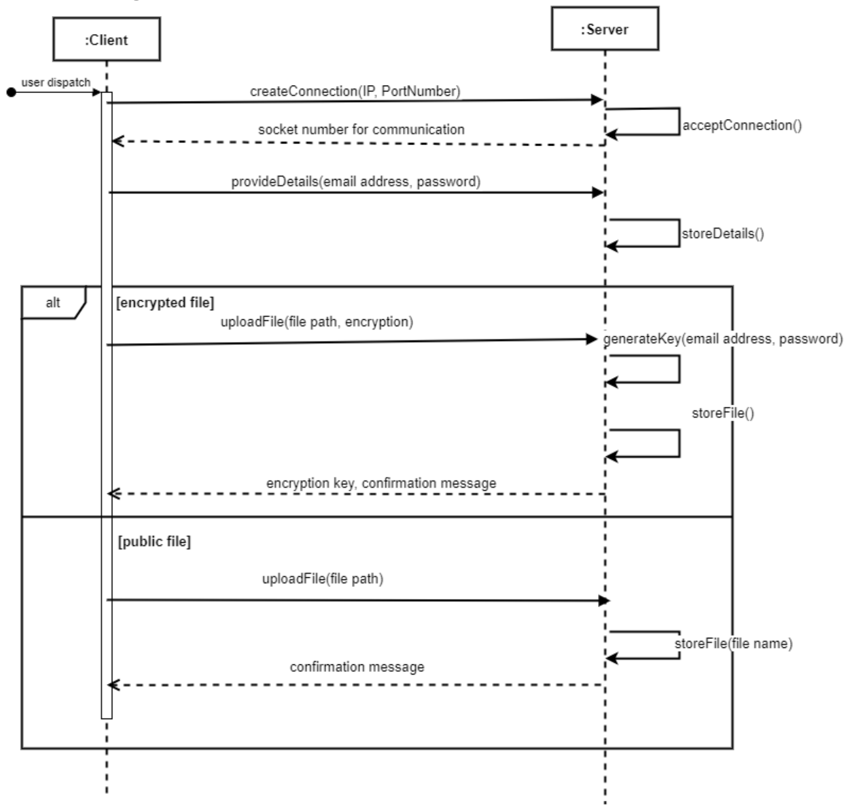

:Client
:Server
user dispatch
createConnection(IP, PortNumber)
acceptConnection()
socket number for communication
provideDetails(email address, password)
storeDetails()
alt
[encrypted file]
uploadFile(file path, encryption)
generateKey(email address, password)
storeFile()
encryption key, confirmation message
[public file]
uploadFile(file path)
storeFile(file name)
confirmation message

Figure 1

Download

This allows the client/user to be able to download files from the server to the local machine. Depending on whether a file was private or public each will have a separate process. If the file was a public file. The server will receive a request and send all the bytes relating to that file the client requested where the bytes will be written to a new file on the client-side producing a copy of the file on the server. If file is private, it will be first decrypted before bytes are extracted and sent to the client side. Figure 2 below depicts how messages flow between the server and client during the downloading process:

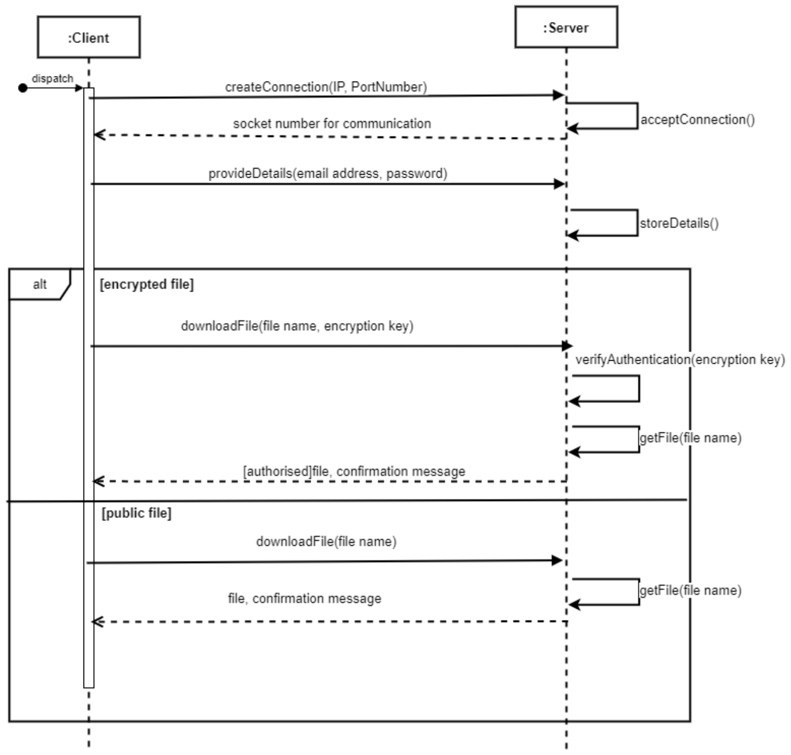

:Client
:Server
dispatch
createConnection(IP, PortNumber)
acceptConnection()
socket number for communication
provideDetails(email address, password)
storeDetails()
alt
[encrypted file]
downloadFile(file name, encryption key)
verifyAuthentication(encryption key)
getFile(file name)
[authorised]file, confirmation message
[public file]
downloadFile(file name)
getFile(file name)
file, confirmation message

Figure 2

Delete

Users are given rights to be able to delete any of the public files that are uploaded on the server. The process of deleting the files requires the file name the user wants to delete and makes use of the python OS library 's remove function to delete the file. Figure 3 below shows how messages flow between the client and server when the user wants to delete a file:

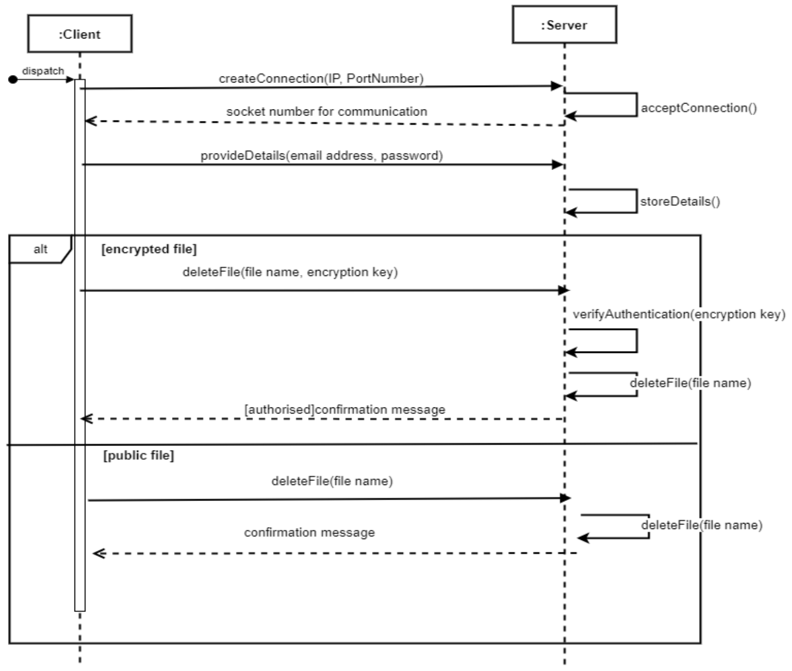

:Client
:Server
dispatch
createConnection(IP, PortNumber)
acceptConnection()
socket number for communication
provideDetails(email address, password)
storeDetails()
alt
[encrypted file]
deleteFile(file name, encryption key)
verifyAuthentication(encryption key)
deleteFile(file name)
[authorised]confirmation message
[public file]
deleteFile(file name)
deleteFile(file name)
confirmation message

Figure 3

Help

Help function was created to allow users struggling to use the application to get detailed explanations on how to use the application. Here is how the client and server interact for this functionality in figure 4 below:

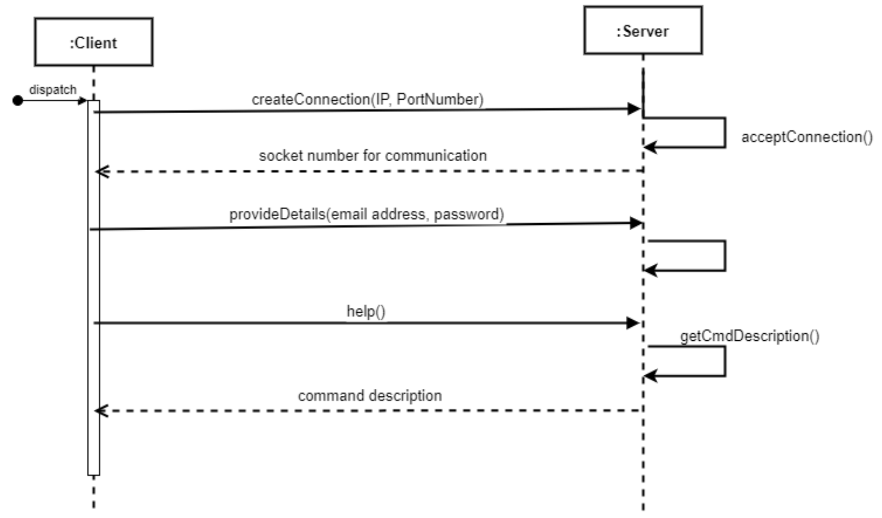

:Client
:Server
dispatch
createConnection(IP, PortNumber)
acceptConnection()
socket number for communication
provideDetails(email address, password)
help()
getCmdDescription()
command description

Figure 4

List

This allows the user to be able to view a list of public files available on the server. Figure 5 below shows how the client and server interact for this functionality:

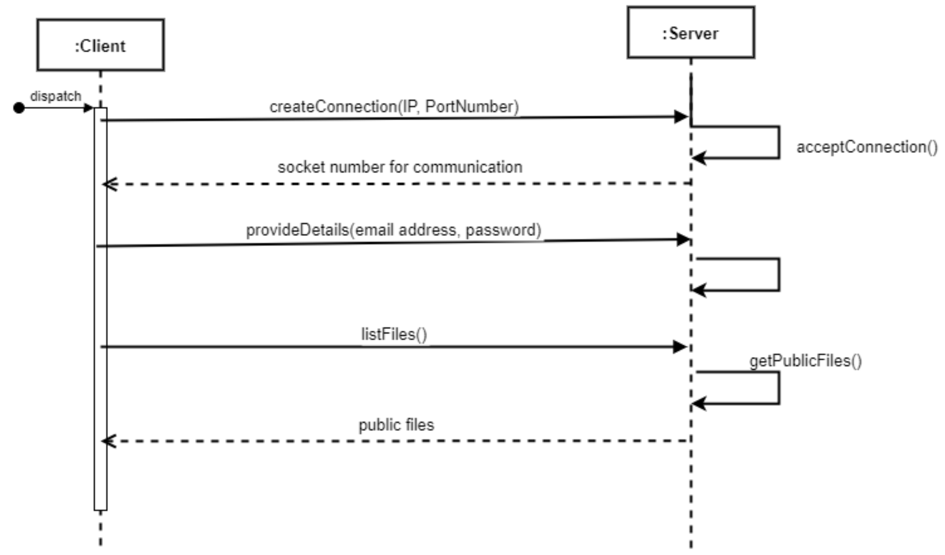

:Client
:Server
dispatch
createConnection(IP, PortNumber)
acceptConnection()
socket number for communication
provideDetails(email address, password)
listFiles()
getPublicFiles()
public files

Figure 5

Encryption and Decryption

Used to control access to the files. the files are encrypted and decrypted for security purposes and for managing access to specific files. The encryption feature is used when the user wants to upload a secured/private file. The email address and password of the user is taken to generate a key, which is used to store the file in the server, and the user receives that key. The decryption feature is used when a user wants to download private file from the server. The server first checks if the user's details are correct, and whether the key matches the generated key for the file. If so, the file is decrypted and sent to the client from the server side, making it available for download to the user.

Multithreading

Used to allow multiple request and multiple responses to different clients. Allows the server to be able to receive a lot of requests at any given time.

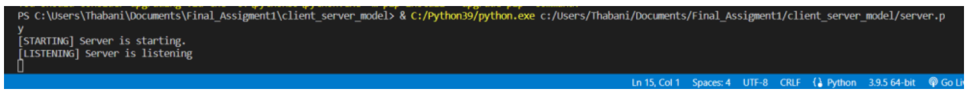
It is important to note that all actions and features depicted in the sequence diagrams from Figure 1 to Figure 5 start with forming a connection between the client and the server, to show that this is a TCP connection, but once a connection is already made, and the user has not logged out of the client, they can continue performing other actions/features without forming the connection first. It is also important to note that the sequence diagrams show a happy-day scenario, and not what happens when the feature cannot be implemented.

The Client Models

First client: Sisipho and Mbali

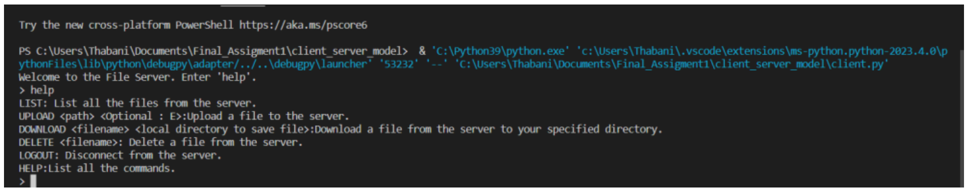
The first client uses simple, command line inputs and outputs to interact with the user. Once the client runs, it displays this message to the user:

1.First run the Server on Terminal, which prompts the server to start listening to any potential client connections.



```
PS C:\Users\Thabani\Documents\Final_Assigment1\client_server_model> & C:/Python39/python.exe c:/Users/Thabani/Documents/Final_Assigment1/client_server_model/server.p
y
[STARTING] Server is starting.
[LISTENING] Server is listening
```

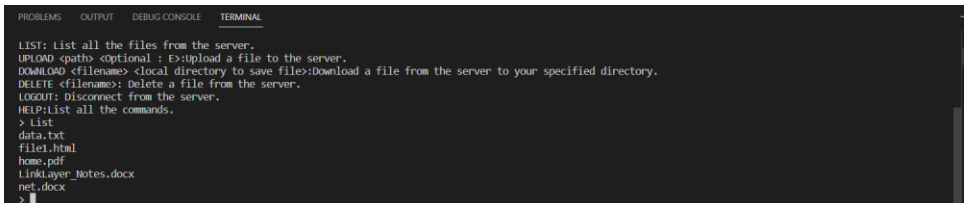
2.Client is started, and a connection is established with the server and message is sent from Server displaying an array of commands available to the client.



```
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Thabani\Documents\Final_Assigment1\client_server_model>  & 'C:\Python39\python.exe' 'c:\Users\Thabani\.vscode\extensions\ms-python.python-2023.4.0\p
ythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '53232' '--' 'C:\Users\Thabani\Documents\Final_Assigment1\client_server_model\client.py'
Welcome to the File Server. Enter 'help'.
> help
LIST: List all the files from the server.
UPLOAD <path> <Optional : E>:Upload a file to the server.
DOWNLOAD <filename> <local directory to save file>:Download a file from the server to your specified directory.
DELETE <filename>: Delete a file from the server.
LOGOUT: Disconnect from the server.
HELP:List all the commands.
>
```

3.User Types in a command and message are communicated back to the Server. In this instance the user selected the" list" command which returns a list of all file documents on the Server that are accessible to the user.



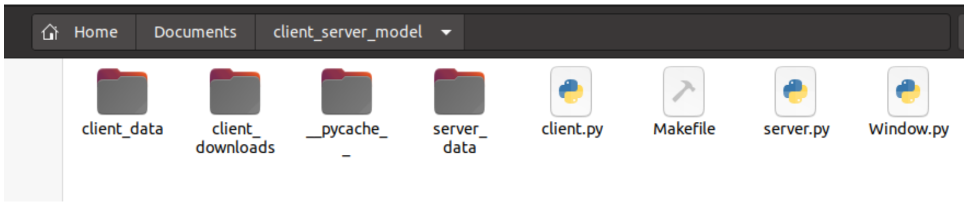
```
LIST: List all the files from the server.
UPLOAD <path> <Optional : E>:Upload a file to the server.
DOWNLOAD <filename> <local directory to save file>:Download a file from the server to your specified directory.
DELETE <filename>: Delete a file from the server.
LOGOUT: Disconnect from the server.
HELP:List all the commands.
> List
data.txt
file1.html
home.pdf
LinkLayer_Notes.docx
net.docx
>
```

4. On requesting a file ”upload/ download” the client has to also provide on which to get or store the file too, as predefined in the command prompt outline.

Second client: Prosper

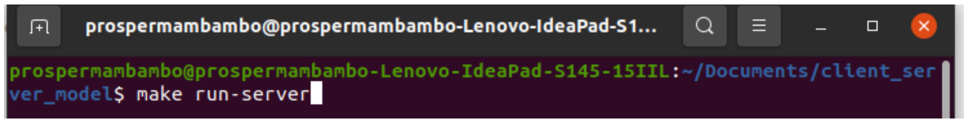
Instruction On How to run Project:

Navigate to the following:



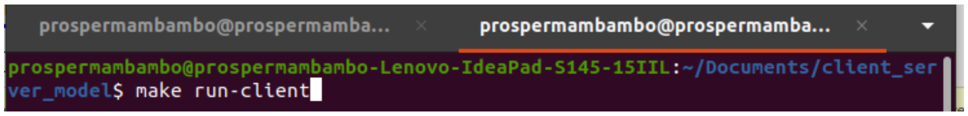
1.open the terminal

2.type the following and click enter



```
prospermambambo@prospermambambo-Lenovo-IdeaPad-S1...
prospermambambo@prospermambambo-Lenovo-IdeaPad-S145-15IIL:~/Documents/client_ser
ver_model$ make run-server
```

3. open another terminal tab and type:

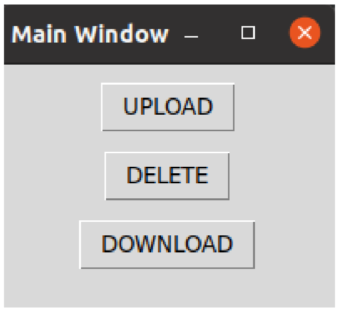


```
prospermambambo@prospermamba...    prospermambambo@prospermamba...
prospermambambo@prospermambambo-Lenovo-IdeaPad-S145-15IIL:~/Documents/client_ser
ver_model$ make run-client
```

Interface Details

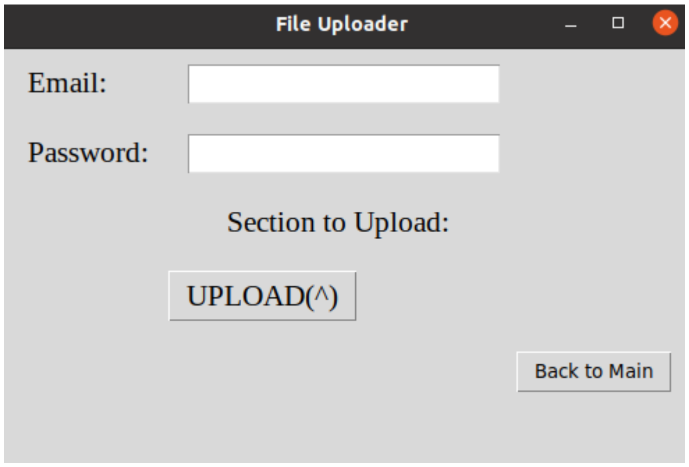
To Simplify the application and allow users as well as Tutors to be able to use the application Easly I opted to adding a Tkinter GUI. The Gui makes use of the same Server we implemented as a group as well as uses the same client code but modified to work together with the GUI components.

1. Main Screen Of the Client:



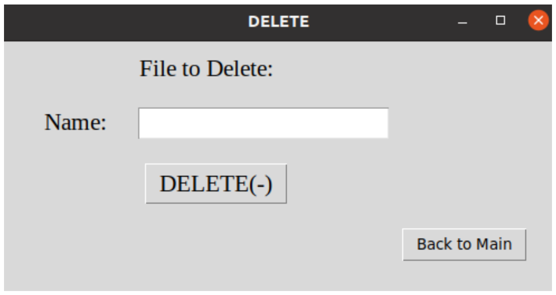
The three main functions of the Main Screen is to allow the user to be able to UPLOAD, DOWNLOAD and Delete a File. Based on what selection the user makes they will be directed to the appropriate screen

2. UPLOADER



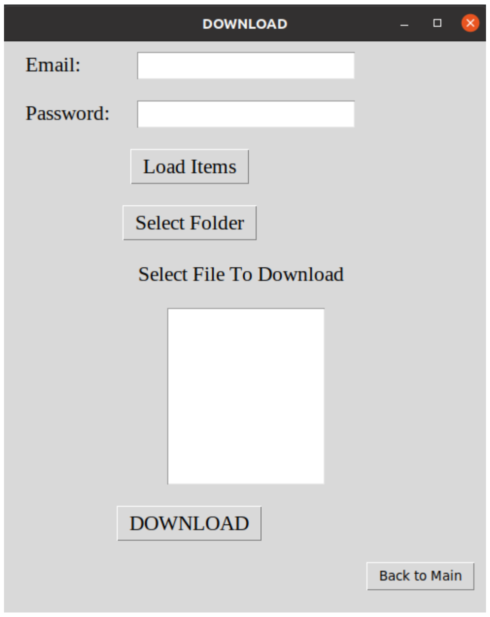
The file uploader allows users to be able to Upload files, all files the user uploads without expressing if the files are private are stored in the file path : server_data/public_files. Files that have been specified as private shall be encrypted and stored in the file folder: server_data/protected_data. This window needs the user email and password and will mainly use their info to search if the user has any private files and makes those files public to the user.

3. File Delete



Allows user to be able to delete any of the public files just by entering the file names e.g “data.txt”. Users have the ability to delete files they would have uploaded by mistake or files that contain malware

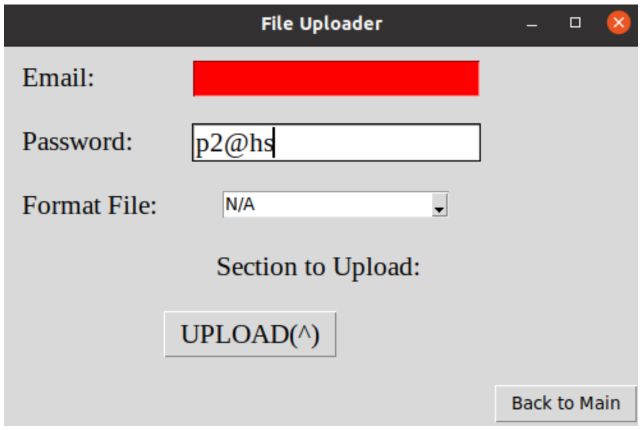
4. File Downloader



This allows users to be able to download files that are in the server. You select the filename you want to download, the folder you want to download after inputting your email and password. once download is clicked it download the files from the server

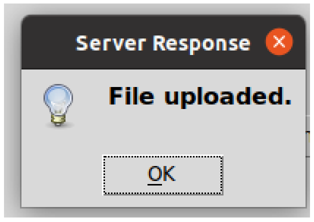
Creativity

Validations were added to the program to prevent users from performing actions without required data. Added a combobox which will allow users to be able to specify if they want to save the file encrypted and hidden privately or just uploaded without any alteration



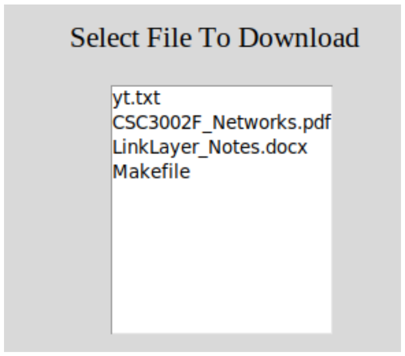
Password: p2@hs

Added feedback to allow users to receive feedback on actions that they perform on the system



File uploaded.

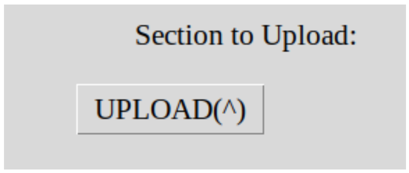
Created a listbox that's automatically populated on click of a button that allows a user to view files on the server as well as to select which file they want to download

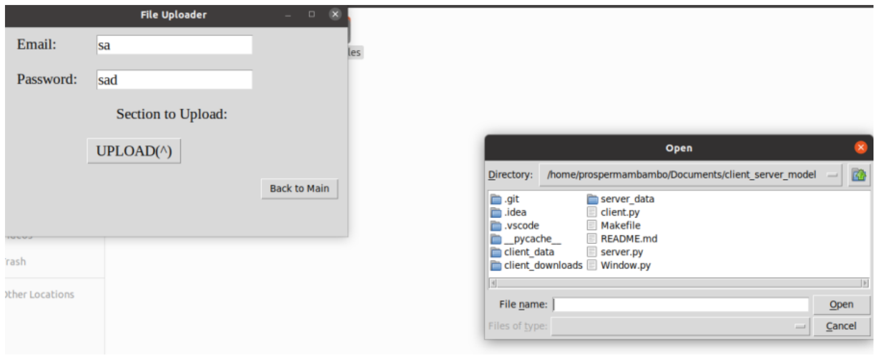


yt.txt
CSC3002F_Networks.pdf
LinkLayer_Notes.docx
Makefile

User does not need to enter file paths manually as I have supplied appropriate windows to deal with that that allow user to select files in their system using a dialog and retrieves the path

On click of upload





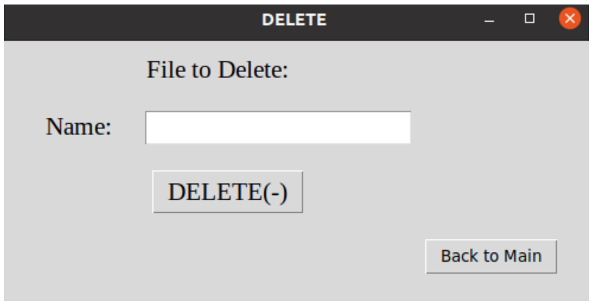
Email: sa
Password: sad

Directory: /home/prospermambambo/Documents/client_server_model

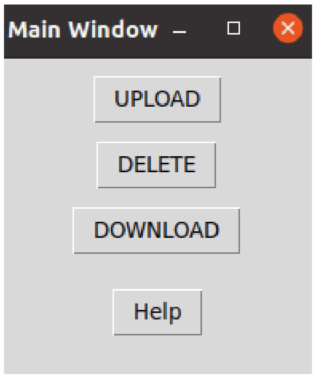
.git
.idea
.vscode
__pycache__
client_data
client_downloads
server_data
client.py
Makefile
README.md
server.py
Window.py

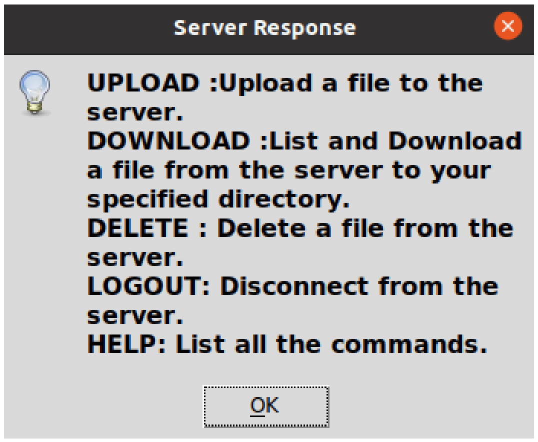
File name:
Files of type:

Have a delete function which allows users to specify the name of the file they want to delete and it will delete it from the server.



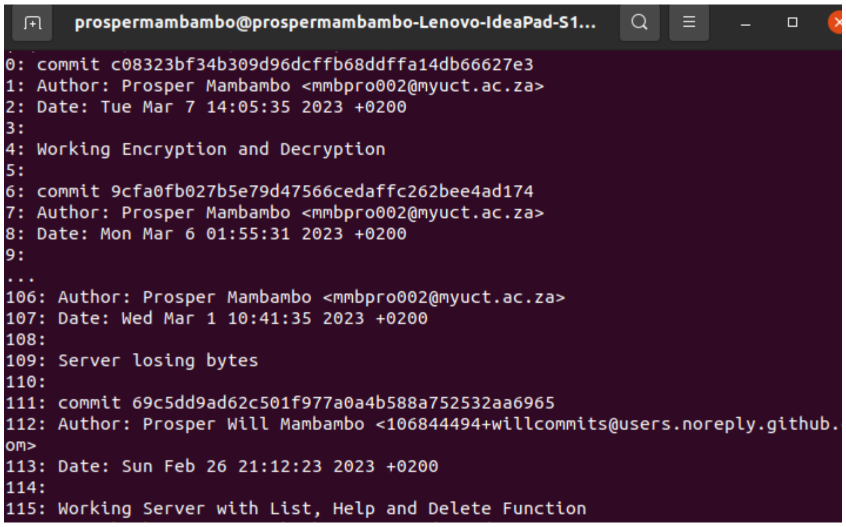
You can be able to click the help function and get a description of each function





UPLOAD :Upload a file to the server.
DOWNLOAD :List and Download a file from the server to your specified directory.
DELETE : Delete a file from the server.
LOGOUT: Disconnect from the server.
HELP: List all the commands.

Gitlog:



```
0: commit c08323bf34b309d96dcffb68ddffa14db66627e3
1: Author: Prosper Mambambo <mmbpro002@myuct.ac.za>
2: Date: Tue Mar 7 14:05:35 2023 +0200
3:
4: Working Encryption and Decryption
5:
6: commit 9cfa0fb027b5e79d47566cedaffc262bee4ad174
7: Author: Prosper Mambambo <mmbpro002@myuct.ac.za>
8: Date: Mon Mar 6 01:55:31 2023 +0200
9:
...
106: Author: Prosper Mambambo <mmbpro002@myuct.ac.za>
107: Date: Wed Mar 1 10:41:35 2023 +0200
108:
109: Server losing bytes
110:
111: commit 69c5dd9ad62c501f977a0a4b588a752532aa6965
112: Author: Prosper Will Mambambo <106844494+willcommits@users.noreply.github.
om>
113: Date: Sun Feb 26 21:12:23 2023 +0200
114:
115: Working Server with List, Help and Delete Function
```

Reliability constraints considered in protocol design/implementation

In protocol design/implementation, reliability is an important consideration to ensure that data is transmitted accurately and efficiently between communicating entities. Here are some common reliability constraints considered we considered when designing our Protocol

1. Error Detection and Correction: One of the most important considerations for reliability is error detection and correction. Error detection techniques like checksum which will included to be able to check if a file was corrupted during the process of being sent, we used when uploading our files
2. Acknowledgment and Negative Acknowledgment: Acknowledgment is an essential component of reliable data transfer. It is a message sent by the receiver to indicate successful receipt of data. In some cases, negative acknowledgment is used to indicate that the received data is erroneous. We included this to be able to tell when the 2 devices connecting on the TPC connection have established a handshake
3. Flow Control: Flow control ensures that the sender does not overwhelm the receiver with data. The receiver sends a message to the sender indicating the amount of data it can receive at a time, which the sender must obey. This we ensure by making sure we get the size of the file before we upload it to the client and allow the client to wait to receive the specific amount of data from the client. This also helps the server be able to receive all the bytes being sent by the client without leaving or not obtaining any bytes being transmitted by the client
4. Congestion Control: Congestion control ensures that the network is not overloaded with traffic. It involves monitoring the network for congestion and regulating the rate of transmission accordingly. Depending on the size of the data we shared we always sent 1024 bytes to the server until all bytes have been successfully sent from the client to the server and the server will have a while loop to receive all the data received.

Stress Testing

Stress testing in programming is a type of software testing that evaluates the ability of a system or application to function properly under heavy loads or extreme conditions. The main objective of stress testing is to identify the limitations of the system or application under test, such as memory usage, processing power, network traffic, and other resources. The testing process involves simulating a high volume of traffic or data inputs to the system or application to determine how it responds.

Tests were conducted to assess the compatibility of the Client-server system when faced with differing scenarios as listed below, and the system's ability to maintain a TCP connection and execute with correctness.

Multiple Connections Above Server Threshold/ Multithreading

To allow multiple connections to take place concurrently at the same time on the server side, every time a request is made to the server the program creates a separate thread to deal with the requests related to the client. This was achieved by using multithreading on the main method of the server responsible for listening and responding to requests from the client side.
The Server has no limitations as to how many connections it can handle simultaneously at the same time. The thread class was imported, and an instance implemented within the server's method main. At given time, the server was made to listen to an uncapped amount of potential connection requests, allowing different clients to establish a connection with the server concurrently.

Uploading of large files

The Server has the ability to receive any file extension and file size. There are no limitations or restrictions to specific files. When uploading large files, the time taken for the bytes/binary being received on the server-side increases before the file is uploaded on the server. The smaller the file , the faster the process.

Initially, the file being uploaded is located and its byte size is read. On completion the encoded bytes are sent to the server socket via the TCP connection established at the beginning. In increments of 1024 bytes, to generate sizable packets of data, the server receives the data sent and on completion a new file is generated, and all data is written to it.
In addition, to ensure the consistency in file conversion Hashlib library was used. The hash function was used to generate hexadecimal sums on the user side prior to the data being sent to the socket. When the data is received on the server, the generated hexadecimals are compared for correctness and detect file corruption.

Uploading a lot of files at the same time (creating a lot of traffic)

Due to the nature of the application of being capable of having multiple requests being taken care of by the server at the same time, the Application can deal with multiple requests where it be UPLOAD, DOWNLOAD AND DELETE where each request 's individual thread will give the appropriate response. Each request is assigned a thread when it's made, and the life cycle of the thread will end when the required tasks have been accomplished.

Conclusion

The aim was to implement a Client-Server model which Includes Features such as downloading, uploading as well as managing access and visibility to certain files in the server which was accomplished. Also, paramount to the implementation was the ability for the Server to interact with different clients at the same time without error. As seen, three individual clients were created, and all were able to interact with the server using the defined Protocol on separate instances. Additional features were created to give more functionality or add simplicity to the users (using Gui) to navigate the client while interacting with the server.