COMP3331

**ASSIGNMENT**

**REPORT**

Richard Quisumbing

z5330506

1. **Program Design**
   1. **Client**

The client design is simplistic, purposefully done to delegate most of the command functionality to the server. Specifically, the client handles user input and parses the command and its associated arguments which forms the data of the client request. The request is then sent to the server via a TCP connection. Resulting responses from the server are displayed to the user in the client.

As for the UDP command which defines a peer-to-peer interaction utilising a UDP connection, the client implements a server thread (if they are the audience) called UDPServerThread which remains open throughout the user session. On the other hand, if the client is acting as the presenter in the peer-to-peer connection, a new client thread called UDPClientThread is created that sends the desired file to the audience client, which ends as soon as all the bytes of the file has been sent.

* 1. **Server**

The server is responsible for identifying the type of command inside incoming requests from clients inside its threading class ClientThread, capable of handling multiple clients simultaneously. It is also directly linked to the server data structure defined in the ServerData class in server\_data.py. When requests are received, the server identifies the command, and handles the majority of the command functionality including user login. Each command is handled within its own function, usually with the naming scheme that looks like *handle\_command\_BCM()*. Inside these functions, error checks are first implemented, then interacting with the server data structure to ensure data is preserved, and finally sending the response back to the relevant client.

* 1. **Server Data Structure**

The server data is a class defined in server\_data.py, which is instantiated as soon as the server is up and running, and is responsible for preserving the data sent by clients. Additionally, the data structure automatically loads the users inside credentials.txt, classifying them as registered users as soon as the ServerData class is instantiated. This means that if the server owner wants new users to be able to login, the server would need to be restarted with the new users added to the credentials.txt file.

* 1. **Validate.py**

Validate.py contains simple validation functions that verify certain aspects of command requests, such as verifying command argument syntax and certain error scenarios.

1. **Message Format**

Messages are mainly formatted as key-value pairs inside python dictionaries.

* 1. **Client Requests**

Text

Description automatically generated

Type: Specifies the type of request being made (usually just ‘command’), which lets the server know how the request is going to be handled.

User: Specifies the username of the user that made the request to the server.

**Data**

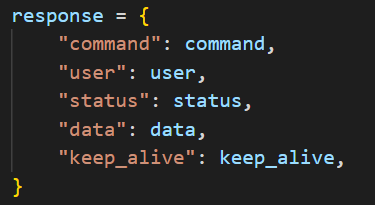
The data for a command request has its own dictionary format:

Graphical user interface, text

Description automatically generated

The “command” key contains the input command supplied by the user, and the “command\_args” contains a list of the command arguments that were supplied alongside the command.

* 1. **Server Response**



Command: The command for which the request was initially made for.

User: The username of the user that made the request.

Status: A Boolean value indicating whether the command was successfully executed.

Data: Contains the requested data.

Keep-alive: Contains a Boolean value of whether the connection should still be kept alive after the command has been executed.

1. **Program Description**
   1. **Authentication**

Authentication functionality is handled as soon as a connection is established between the server and the client. The server first requests the client to provide a username.

1. **Potential Improvements**