Comp Eng 4DN4 Lab #2

Alex Szebrag
1148394

Sever class modifications (from the EchoClientServer.py file provided by Dr. Todd)

Instance variables: Added an instance variable "GET_AVERAGES_CMD" and set it equal to the string "GETA" so that the server can compare "GETA" encoding from client with its own encoding of "GETA" (i.e.: to see if the client sent the GAC command).

Constructor: Added a call to "get_csv_file" method prior (which reads and prints out the CSV file on the server terminal) to creating the socket, binding it to a port/IP address and putting it into listen mode. Like before, the socket is created, bind to a port/IP address and then continuously accepting connections, receiving encoded bytes, decoding them, and then sending back responses to the client (through "process_connections_forever" and "connection_handler" methods.

get_csv_file method: Added this method to read the CSV file properly, loop through line by line to append to an array, and count the amount of total rows there are. This method then creates an instance variable copy of that array, and finds the column size of the array as well.

create_listen_socket method: No changes. Still creates socket, binds the socket and sets it into listen mode (in order to accept connections).

process_connections_forever method: No changes. Still accepts client connections and calls "connection handler" method.

connection_handler method: modified to (assuming we know the string fields we are looking for and assuming we know that averages proceed directly after the row of empty strings, and that the last student comes before the row of empty strings): 1) find the row index of the array the 1st student and last student is on, find the row index of the headers (i.e: Lab 1), find the student ID # column index, find student ID password column index, find the last name column index, find the first name column index, find the midterm column index, find lab 1 column index, find lab 2 column index, find lab 3 column index, find lab 4 column index, find the column index that states "averages", find the rows that contain the averages. This is done by manually looping through the 2D array and finding either the desired string value or finding the index based on assumptions mentioned above. 2) receives bytes from the client and checks if the received bytes match the GAC command ("GETA" string). **If so:** the server will print out that it received "GAC" command from client, create a string which contains all the averages, encode it and send it back to the client. **If not:** the server will assume it received a password + ID from the client and it will print out the hashed value of the ID/password sent from the client. The server will then loop through the rows of the array starting at the first student index until the last student index creating a new hash object each time, and adding the current student ID + password to the new hash and comparing it with the received hashed bytes. If there's a match, we break out of the loop and we can see that the matching student ID is the same index as the looping variable. If there's no match then the looping variable will be the last student index row

+1. In the case that there's a match, the server will create a string of the student's record, encode it and send it back to the client. If there is no match, the server will create a string of the error message, encode it and send it back to the client.

Client class modifications (from the EchoClientServer.py file provided by Dr. Todd)

Instance variables: Added an instance variable "GET_AVERAGES_CMD" and set it equal to the string "GETA" so that the client can compare the client input command to the string "GETA".

Constructor: Added the following (looped through continuously): 1) call to "get_client_command" method to get client input from command prompt (not accepting empty strings), 2) A check to see if the command entered by the user is the GAC ("GETA" string). **If so:** establish a TCP connection with the server, encode and send the GAC command to the server and receive the encoded bytes of the averages from the server + decode and display it on the client terminal. **If not:** get student ID input from the client (not accepting empty strings), and password input from the client terminal (using getpass method from getpass module), print the student ID + password, encode both in UTF-8, the create a hash object to store both the student ID and password (in SHA256). A TCP connection is then established with the server and this hash encoded byte stream is then sent to the server over the TCP connection. The server then returns either an error message or the student record encoded in UTF-8, the client receives and decodes this and displays it on the client terminal.

get_socket method: No modifications. Creates a socket object like before.

get_client_command method: Does the same thing as provided function: get_console_input. Gets user input on the client terminal not accepting any empty strings.

connect_to_server method: No modifications. Establishes a TCP connection with the server.

connection_send_HASH method: prints out the hash value of the student ID and password sent from the client to the server, then sends all the hashed bytes to the server via the TCP connection.

connection_send_GAC method: Encodes (in UTF-8) and sends the byte stream to the server via the TCP connection.

connection_receive_grade_avgs method: Receives the bytes stream of the string containing the averages (encoded in UTF-8) from the server, and prints the decoded version of the bytes object.

connection_receive method: No modifications made. Used to receive an encoded UTF-8 bytes object (containing the student's record or error message from the server). Receives and prints the decoded bytes on the client terminal.