KTN - Chat client/server

Henrik Olsvik Navjot Singh Peter Holm
Philip Puente Halvard Hummel

March 13, 2016

1 Client

For the client there are two sets of python files which are almost identical, the difference being that the one labeled Linux handles formatting the input in case of receiving a response from the server at the same time that you write a request. As this is easily done in Linux, opposed to Windows and OSX, we decided to include it in the client. The Linux code does not run in Windows and has some problems with formatting in OSX, for all other OSes than Linux use the other Client files.

1.1 Client.py

1.1.1 Functions

run() - Handles logic at runtime, this includes taking input from the user and parsing the input into the send_payload() method in the client object. This function also prints out a short message at startup with information about the client.

1.1.2 Variables

client - The client object created at startup.

message_parser - A MessageParser object, for encoding messages.

1.1.3 Classes

Client

Handles client logic, e.g. creating a connection between the server and the

client and sending messages to the server.

Variables

connection - The connection to the server.

server_port - The port for the connection on the server.

host - The host address for the server.

Methods

Client(self, host : string, server_port : int) - Setup connection variables and calls run method for connecting the client to the server.

disconnect(self) - Handles logout and disconnect from the server, stops the MessageReceiver thread.

run(self) - Connects the client to the server, starts a MessageReceiver thread for receiving messages from the server.

send_payload(self, data: string(json)) - Handles sending of requests from client to server.

1.2 MessageReceiver.py

1.2.1 Classes

MessageReceiver(Thread)

Handles message receiving for the client, runs as a own thread so the client can receive and send messages at the same time.

Variables

client - The Client object connected to the server

connection - The connection to the server from the client

message_parser - A MessageParser object for decoding and handling the content of incoming messages.

is_running - Variable used to check if the MessageReceiver object should continue running.

Methods

MessageReciever(self, client : Client, connection : connection) - Creates a MessageReciever object. Setup of variables.

run(self) - Listens to the connection for incoming messages from the server. Sends all messages to the parse() method in the MessageParser object.

1.3 MessageParser.py

1.3.1 Functions

print_formatted_message(timestamp : string, response_type : string, content : string) - Prints formatted message/error to the console.

1.3.2 Classes

MessageParser

Encodes requests to json. Handles and decodes responses from the server.

Variables

possible_responses - A dictionary with possible response codes from the server as keys and the methods for parsing these a values.

Methods

MessageParser(self) - Creates a MessageParser object with a dictionary for possible response codes.

parse(self, payload : string) - Parses and handles the json payload. Calls the appropriate method for handling the response from the server.

encode(self, request : string, content : string) - Encodes request and content to json string.

parse_error(payload : ?json) - Handles an error response and prints the error message using print_formatted_message().

parse_info(payload : ?json) - Handles an info respone and prints the info message using print_formatted_message().

parse_message(payload : ?json) - Handles a message response and prints the message using print_formatted_message().

parse_history (payload : ?json) - Handles a history response, calls parse_message() for each message.

2 Server

2.1 Server.py

2.1.1 Functions

username_available(username : string) - Returns boolean for availability of username.

valid_username(username : string) - Returns if username is in the format [A-z0-9]+

user_logged_in(user : ClientHandler) - Returns boolean if the user is logged in

get_username(user: ClientHandler) - Returns the username of the client as a string, this string is empty if the user is not logged in.

current_timestamp() - Returns the current date as a string in the format, "MM.DD HH:MM:SS".

encode(sender : string, response : string, content : string) - Returns a json string in the servers response format with the current time as timestamp.

parse_request(payload : string, user : ClientHandler) - Parses the json object, checks if the user has access to the request and call on the appropriate function for handling the request (request_codes).

parse_login(username : string, user : ClientHandler) - Checks if the user is logged in, username is in wrong format or username is taken and sends error message if necessary, if not register user and send info response. Sends any message history the server has.

parse_logout(content : string, user : ClientHandler) - Disconnects the user, removes user from user dictionary.

parse_message(message : string, user : ClientHandler) - Adds message object to history and sends message to all other users logged in to the server.

parse_help(content : string, user : ClientHandler) - Sends the user a response to the user with a help text.

parse_names(content : string, user : ClientHandler) - Sends the user a response with the username of all logged in users.

2.1.2 Variables

history - List containing message objects (2.2.1) for all messages sent while the server has been running.

users - Dictionary with username as key and ClientHandler object as value, for all users who have logged in.

unlogged_users - List of all clients that have not logged in.

request_codes - Dictionary with all request codes supported by the server as keys and the functions for handling these as values.

2.1.3 Classes

ThreadedTCPServer(ThreadingMixIn, TCPServer)

Creates threads for server

- Builds on the **socketserver** packet and will not be changed.

ClientHandler(BaseRequestHandler)

Handles connection between server and client, listens for requests from client and sends responses to client.

Variables

ip - IP address of client.

port - Port for the connection at the client.

connection - The connection between the client and the server.

run - Indicates if the object should continue to run

Methods

handle(self) - Handles the connection between the client and the server and waits for messages from the client

close(self) - Closes the connection between the client and the server.

self, payload : string(json) - Sends a message to the client.

2.2 Message.py

2.2.1 Classes

Message

Creates nessage objects, user for history

Variables

message_text - Text content of the message.

user - Username of the message sender.

timestamp - Timestamp for when the message was received at the server

Methods

Message(self, message_text : string, username : string, timestamp : string) - Creates a message object containing message text, username and timestamp.

to_JSON(self) - Returns a json string in the servers response format with "Message" as the response code and fills the other fields with the fields of the object.