COMP 4958: Assignment 1

Submission deadline will be announced. You will need to submit a zip file containing the project folder named `chat` (excluding the directory `_build`). Be sure to comment your source files. You will need to set up and demonstrate your "chat" system as well as explain, and possibly answer questions about, your code.

For this assignment, you are asked to implement a system in Elixir that allows users to send messages to one another.

The user connects to the system via TCP and issues commands. There are basically four supported commands:

- /NCK <nickname>
- /LST
- /MSG <recipients> <message>
- /GRP <groupname> <users>

The names of the commands are case-insensitive and each command is terminated by the end-of-line character sequence. Any command different from these 4 is invalid.

The /NCK command is used to set a nickname. It takes one word as argument; any extra words are ignored. A /NCK command with no following words is invalid. For example, `/NCK homer` asks the server to use `homer` as the nickname for the user who issues that command. The server needs to ensure that the requested nickname is not in use before granting the request. The client receives a response message indicating whether the operation is successful. There are restrictions on what constitutes a valid nickname: it must start with an alphabet, optionally followed by characters that are either alphanumeric or the underscore character, up to a maximum of 10 characters. *A user of the system must succeed in setting a nickname before they can send or receive messages.* However, the user is allowed to use the /LST command without setting a nickname. Note that a user can also use the /NCK command to change their existing nickname. In the following, if necessary, we will refer to a user that has acquired a nickname as a registered user.

The /LST command does not take any arguments: words after it are ignored. It is used to get a list of the nicknames currently in use.

The /MSG command is used to send a message to a specific user or a list of users. `/MSG homer hello world` sends the message "hello world" to the user with nickname `homer`. `/MSG homer,bart hello world` sends the message to users `homer` and `bart`. Note that the nicknames are separated by commas. Clearly, the /MSG command must be followed by at least two words: the first word specifies the recipient(s) and the rest the actual message. A /MSG command followed by fewer than two words is invalid. Note that only registered users can send (and receive) messages.

The /GRP command is used to assign a group name to a list of (one or more) users. After `/GRP #simpsons homer,bart,lisa`, we can use, e.g., `/MSG #simpsons hello there` to send the message "hello there" to the three users: homer, bart and lisa. A group name must start with the hash symbol (`#`) followed by an alphabet, and optionally followed by characters that are either alphanumeric or the underscore character, up to a maximum of 11 characters (including the hash symbol).

For the design, there must be one globally-registered supervised server that provides the main behaviour of dispatching messages and handling nicknames, i.e., it receives requests from other processes whenever messages need to be sent and whenever a user needs to set or change their nickname. This server must use one or more ETS tables to keep track of its state. We'll refer to this server as the chat server. This server does not communicate with external clients (which may be implemented in a language different from Elixir and which use TCP).

There must also be a second type of Elixir servers responsible for accepting external clients that connect via TCP. Such a server creates one Elixir process to handle each TCP connection. This "spawned" (i.e., created) process is responsible for parsing and validating commands sent by the client, remembering and handling group names for the client, and, if necessary, requests the chat server for services (e.g., to send a message to users). It also sends messages from the chat server, as well as its own error messages (e.g., for invalid commands), to its connected TCP client. From the point of view of the TCP client, it is a "proxy" for the chat server. We call the server that creates these proxies the proxy server, and its "spawned" processes proxies. The proxy server is not registered and we can run different instances of it on different nodes.

Note that proxies are responsible for handling group names; the chat server has no notion of groups. Note also that command validation is part of the responsibilities of a proxy as well.

Your mix project should be named `chat`. From the description above, it is clear that there must be at least two modules. Call them `Chat.Server` and `Chat.ProxyServer`. (Proxies can be in a third module if so desired. In that case, name it `Chat.Proxy`). `Chat.Server` implements the globally-registered chat server. It uses GenServer. `Chat.ProxyServer` implements the proxy server that listens at a particular port (defaults to 6666) and creates a proxy process whenever an external client connects to the system via TCP. In this case, it is up to you to use GenServer or not.

You will also need to implement a suitable client either in Elixir or in Java. It takes a host and a port number as command-line arguments. (The host defaults to "localhost" and the port to 6666.) Basically, the program needs to connect to a proxy server, sends commands the user types to the proxy that is created, as well as receiving and displaying replies from that proxy concurrently. It is a "dumb" client and does not do command validation. (Command validation is performed by the Elixir proxy). The program terminates when the user enters the end-of-file key at the beginning of a input line. The system needs to remove the nickname when a client terminates.

To faciliate testing, the system must print "debugging" information. More details may be provided. You may need to provide documentation on how to test your system.