COMP 4958: Assignment 1

Information on how to submit this assignment as well as submission deadline will be provided. 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. The are basically three supported commands:

- /NICK <nickname>
- /LIST
- /MSG <recipients> <message>

The names of the commands are case-sensitive and each command is terminated by the end-of-line character sequence. Also, /NICK can be abbreviated to /N, /LIST to /L, and /MSG to /M. Any command different from these 6 variants is invalid.

The /NICK command is used to set a nickname. It takes one word as argument; any extra words are ignored. A /NICK command with no following words is invalid. For example, `/NICK 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 followed by characters that are either alphanumeric or the underscore character and it can have a maximum length of 12 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 /LIST command without setting a nickname. Note that a user can also use the /NICK 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 /LIST 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 user, a list of users, or all registered 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 separted by commas. Finally, `/MSG * hello world` "broadcasts" the message to all registered users. Clearly, the /MSG command must be followed by at least two words: the first word specifies the recipients 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.

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 so that it can recover from (brief) crashes. We'll refer to this server as the chat server. Note that this server does not directly communicate with external clients using 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 process parses and validates the commands the client sends and requests the chat server for services when needed. This reduces the workload on the chat server as it does not need to deal directly with external clients. The spawned process also sends messages from the chat server, as well as its own error messages (for invalid commands), to its connected client. From the point of view of a connected external client, this spawned process is a proxy for the chat server. We call the server that spawns these proxies the proxy server. It is not registered and we can run different instances of it on different hosts or nodes.

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`. `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 spawns a proxy process whenever an external client connects to the system via TCP. It does not need to be a GenServer (although it can be).

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 spawned by the proxy server.) 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 servers must print "debugging" information. More details may be provided. You may need to provide documentation on how to test your system.