Systems Programming Chat Application

Isaac Sutherland | c2014926

https://www.youtube.com/watch?v=4Q7ajgXOnTs - Video Presentation

The Architecture

Entry point Program.cs, provides selection to start either a Client or a Server (Static classes).

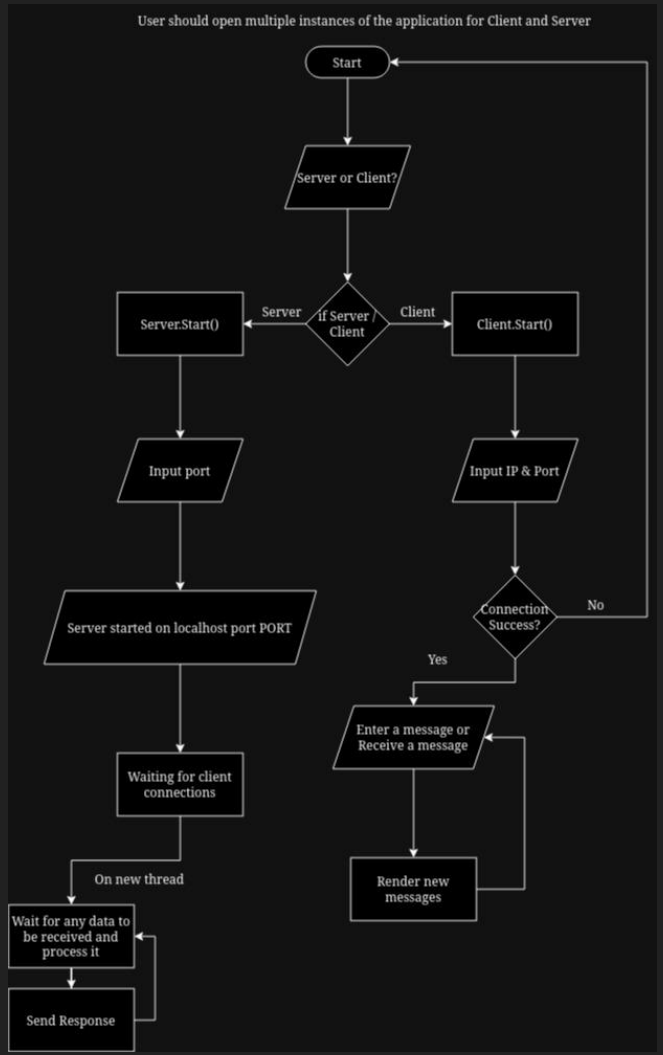
Static Client & Server classes to do any work related to either the client or server-side of the application including Socket connecting / sending and receiving data.

Static UI class that is used in Client to render a console UI for the user to interact with based on public properties of Client.

Message and User classes that are serializable and sent between the Client and Server to communicate using JSON serialization & deserialization as well as UTF8 byte encoding into a buffer.

Command Class and Classes that inherit the CommandImplementation interface for defining Commands with a concrete execute implementation.

Program Flow



Dealing With Data - The Server

I have used a HashSet to store data that should not have duplicates but has the possibility of a duplicate being added during runtime. And Lists to store any data that does not have the possibility of a duplicate being added during runtime e.g. Connections due to the connected callback only running once per client and Commands which are pre-defined and stored in a list with a predetermined size.

I have used IEnumerable for Users by running Select over the connections list to produce an IEnumerable of Users objects on those connections, I have not converted this back into a list due to it using lazy evaluation meaning the response is always up to date with the connections list.

I have also queried the collections using LINQ methods (Where, Any and First) to retrieve data such as which command matches the received message and, do any users have the same username.

Dealing With Data - The Client

Used List collection to store any message objects received from the Server so that the UI can render them. Lists are a good fit due to them being dynamically resizable so it can store any amount of messages as well as duplicate objects such as cached command responses such as the help command.

Command Line Arguments

I have used the string array args passed to the Main function of Program.cs to create a few command line arguments:

- s / server: Prompts for port then starts a new server instance.
- c / client: Prompts for IP and Port then attempts to connect to the server at that address.
- -v / –version: Outputs the current version of the program.
- Any other argument, outputs a help message

Robustness

Without any error handling the program becomes volatile and crashes easily due to Socket Errors that occur when clients disconnect forcefully or crash. I have resolved this through catching both specific and non-specific errors as well as, performing different actions based on the exceptions error code. For example, when catching a SocketError I may want to remove the client if the error is that the pipe is broken, however, do not want to do that if any other code occurs.

Object Oriented Programming (OOP)

I have used Object Oriented practices in my development of this application, such as, encapsulation and inheritance. I used encapsulation to stop any aliases that already exist on other commands from being added to a new command. Inheritance has been used to ensure that any concrete implementations of an interface adhere to its structure. However, rather than using polymorphism I have implemented a strategy pattern to define the execution of a Command. This is better for the maintainability of the application as it will allow any future commands to use the same execution with any new polymorphed functions and reduce the amount of boilerplate needed.

Data Persistence

I do not persistently store any data in my application to reduce privacy concerns over the encryption of messages that are stored. However, I do use JSON serialization and deserialization to turn objects into strings that can be encoded into a UTF8 byte array, as well as encode and decode strings into bytes and vice-versa using UTF8.

Writing Fast Code

I have not put a major emphasis on the speed of this application. However, I do cache any static responses from commands e.g. the help command menu.

Added Complexity

- Thread Blocking / Unblocking using ManualResetEvent to ensure that only one action is occurring at a time.
- Having the UI update when a message is received while asking the user for input ended up needing CancellationTokens to dispose of the Spectre.Console prompt and clean up the thread that the iteration of the UI was running on
- Communication between Client & Server and the use of AsyncCallbacks to determine the result of Sending / Receiving data.

Thanks For Listening!