Network Program Design

Crab Chat

# Project Members

* 1. Evan Binkley, in charge of server and project manager
  2. Austin Swartley, in charge of server
  3. Peter Schaefer, in charge of client and library

# Communication Plan

* 1. Snapchat and GitHub will be used for general communicating and collaboration.
  2. Snapchat and IRL Meetings will be the main method of conducting team meetings and focused collaboration efforts.

# Task Management

## Program Design (Theorizing)

* + 1. JSON packet formatting and organization idea (Create format)
    2. Create possible schedules and due dates.
    3. Have consistent programming format practices for readability.
    4. Create a sequence diagram.

## Application Design (Writing)

### Client

* + - 1. Basic nickname sending
      2. Colored name specs storing
      3. JSON packet management and sending
      4. Server-to-Client push receiving and printing

### Server

* + - 1. Basic server user setup (Executable startup process)
      2. Thread management for user connections
      3. Stream handling per thread and JSON packet handling
      4. Server-to-Client message pushing and updating
      5. Signal Handling and Graceful closing

### Library

* + - 1. Library (lib.rs) will be updated as per sufficient needs (When applicable)

# Programming Languages

* 1. The Client-side program for our project will be written and running in Rust.
  2. The Server-side program for our project will be written and running in Rust.
  3. The Library for our project will be written in rust, and shared between both programs for what is needed.

# Project Scope

* 1. **Client-side**
     1. Taking Ip/Port #’s in command line arguments
     2. Nickname sending
     3. Nickname color selection
     4. message sending
     5. SIGINT handling
  2. **Server-side**
     1. nickname checking/handling
     2. Message receiving
     3. Message management and message pushing / updating.
     4. Message and event logging
     5. Graceful program termination
     6. Taking Port # in command line arguments.

# User Interaction

## Client-side

* + 1. User inputs username / nickname (UTF-8 text restrictions)
    2. User inputs color of nickname (UTF-8 text restrictions)
    3. User sends messages to terminal for processing (UTF-8 text restrictions)
    4. Client updates board of messages as server pushes and receives them

(Arbitrary non-printable characters are not expected, and although the program is not based on ASCII characters, it is expected that all messages and nicknames be restrained to ASCII characters.)

## Server-side

* + 1. Server awaits port number for socket.
    2. Server processes information and pushes packet based data to clients based on receiving. (Packets received can be UTF-8 but are expected to be ASCII in terms of content.)
    3. Users can input termination signal for graceful close.

# Shared Functionalities

* 1. JSON packet formatting
  2. Encoding and decoding packets sent and received

# Application Protocol

### JSON Packet Makeup

| **Component** | **Explanation** | **Type** |
| --- | --- | --- |
| '**author**' | Nickname (previously accepted) of client sending message, in string format. | Type String in rust |
| '**time**' | Time at the packets creation in (Hours:Minutes:Seconds), in string format. | Type String in rust |
| '**message**' | Contents of the message in the form of a string. | Type str in rust |
| '**color**' | Three consecutive numbers in the range [0, 255] in a single string, separated by spaces. | Type String in rust |
| '**kind**' | (Optional, sent under specific circumstances) Contains specific packet type, signaling special behavior by either server or client. | Type str in rust |

### JSON Usage Outline

* When a client connects, its address is saved to a list of active clients in a scanning thread's memory. This removes the need for a "hello" message to be received.
* When a client disconnects, it will send a JSON object with type 'kind' of "disconnection", signaling its removal from the active list of clients.
* After the client successfully connects, they will select a username. This username is set as the "author" type of a JSON object, and the type 'kind' of "nick".
  + The server will verify the uniqueness of the nickname it has received.
    - If unique, the server will send a JSON object back to the client with the type 'kind' of "okay". The client is now able to freely send and receive messages.
    - If not unique, the server will send a JSON object back to the client with the type 'kind' of "retry". This will start a loop that will make the client select a nickname not in use.
    - When the client selects a unique nickname, the client's nickname, IP address, and active timestamp is recorded in a log file.
* **Sending messages**
  + The user will enter a message through the client, which will then be sent to the server.
    - The nickname of the client will be in the type 'kind' of "author".
    - The message data will be in type 'kind' of "message".
    - Color data will be included in the type 'kind' of "color", formatted in the format above.
    - Time data will be included in type 'kind' of "time", as formatted above.
  + The user's messages will be recorded to a log file, which includes nickname, timestamp, and message content.
* **Receiving messages**
  + The client has an active listening thread that will print all messages received.
    - This thread will also check for other types of things, such as:
      * Not printing messages with no type 'kind' of "message".
      * Other object types (Server disconnect, etc.)
    - This makes the messages a push-based service.

# Sequence Diagram

A diagram of a flowchart

Description automatically generated

# Test Plan

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Number** | **Brief Description** | **Expected Results** | **Tester Name** | **Date Tested** | **Actual Results** | **Success or fail** | **Correction** |
| 1 | Nickname taken | Error message stating nickname is taken and then ask for a new name. | Evan | 12/8/23 | Message states nickname is taken. Asks for new nickname | Success | N/A |
| 2 | Send a message | Push a message from the server to all connected clients | Austin | 12/8/23 | Message received by all active clients. | Success | N/A |
| 3 | Close a client | User inputs the close command and then their client session will be closed and cleaned up on server side, also should work with Ctrl+C | Evan | 12/8/23 | Terminal closes and cleans up server-side. Functional with SIGINT | Success | N/A |
| 4 | Close server | Should inform all clients upon closing, then close after X time. | Evan | 12/8/23 | Server sends message to all active clients upon receiving SIGINT, waits X amount of time, then closes. (server-only) | Success | N/A |
| 5 | Empty Message | If a user sends an empty message, then the message should still send to the server and all the other users. | Austin | 12/8/23 | An empty message is sent to active clients. | Success | N/A |
| 6 | Username color selection | User can select color for their nickname to display to clients when messaging. | Evan | 12/8/23 | Users can see colored nicknames in terminals when receiving other clients’ messages (based on their color choice), and if the choice is invalid, the program will simply shut down. | Success | N/A |
| 7 | Large amount of users all inputting messages. | If there are many users all sending messages at once, we would like to see no felt drop in performance. | Evan | 12/7/23 | Attempted to access multiple terminals at same time and send messages rapidly. Saw no difference in performance. | Success | N/A |
| 8 | Server logs messages and events | Server should print events and user messages to a log file. | Evan | 12/8/23 | Log file is created and appended to upon new messages or events caught by server. | Success | N/A |
| 9 | Client message sending | Client should accept (expected) characters and send to server. | Austin | 12/8/23 | Expected characters received by server and pushed to other active clients. | Success | N/A |
| 10 | Server-side nickname handling | Server should be able to access a table of active nicknames and refuse requests for taken ones | Evan | 12/8/23 | Server recognizes active nicknames from file table of nicknames and refuses new requests for ones already taken. | Success | N/A |
| 11 | Server-side message receiving | Server receives messages sent by any active client. | Austin | 12/8/23 | Server receives all messages. Evident by fact that multiple clients can receive clients message | Success | N/A |
| 12 | Client-side message receiving | Client able to receive messages sent by other clients. | Austin | 12/6/23 | Client is able to receive and print all messages received by active clients connect to same server | Success | N/A |
| 13 | Client taking bad IP addresses or ports | If bad IP or port is given, program refuses connection and exits, giving reason | Austin | 12/6/23 | Client refuses connection because of rust unwrap, error printed | Success | N/A |
| 14 | Server taking bad port # | If bad port is given in COMMAND LINE ARGUMENTS, server refuses to start, prints error for reason | Austin | 12/6/23 | Server refuses to start because of rust unwrap error checking, error printed | Success | N/A |