

Specification

The chat program will be written in Rust. The information flow is mainly handled by channels between threads, and as such the arrows in the sequence diagrams represent dataflow, rather than function calls.

Server

The server is what contains most of (if not all) the logic. On startup it will create a `TcpListener` which will listen for, and accept incoming connection requests. For every established connection a `ClientHandler` will be made, and it will run in a separate thread. The `ClientHandler` will receive requests from a `TcpStream`, parse them, and hand them to the `ChatServer` via a channel. Likewise the `ChatServer` will hand responses through a channel to the `ClientHandler`. When the `ChatServer` receives a request it will perform the necessary actions. A request will be on the form

```
struct Request {
    request: RequestType,
    content: Option<String>,
}
```

Where the enum `RequestType` will have the definition

```
enum RequestType {
    login,
    logout,
    help,
    names
    msg,
}
```

The `ChatServer` replies to all requests with a response on the form

```
struct Response {
    timestamp: String,
    sender: String,
    response: ResponseType,
    content: Option<String>,
}

enum ResponseType {
    error,
    info,
    message,
    history,
}
```

The only requests that is valid for a user that is not yet logged in is `login` and `help`. All other

responses will be answered with an error response.

Client

The client will get an IP address and port of a listening server during startup. It then connects to the server, and pass messages between the user and the server. When the user writes a command on the form `<command> <data>` it will be parsed by the client, and sent to the server. Likewise when the client receives a response from the server it will parse this response and show the relevant information to the user.