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 recieve requests from a `TcpStream`, parse them, and hand them to the `ChatServer` via a channel. Likwise the `ChatServer` will hand responses through a channel to the `ClientHandler`. When the `ChatServer` recieves 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

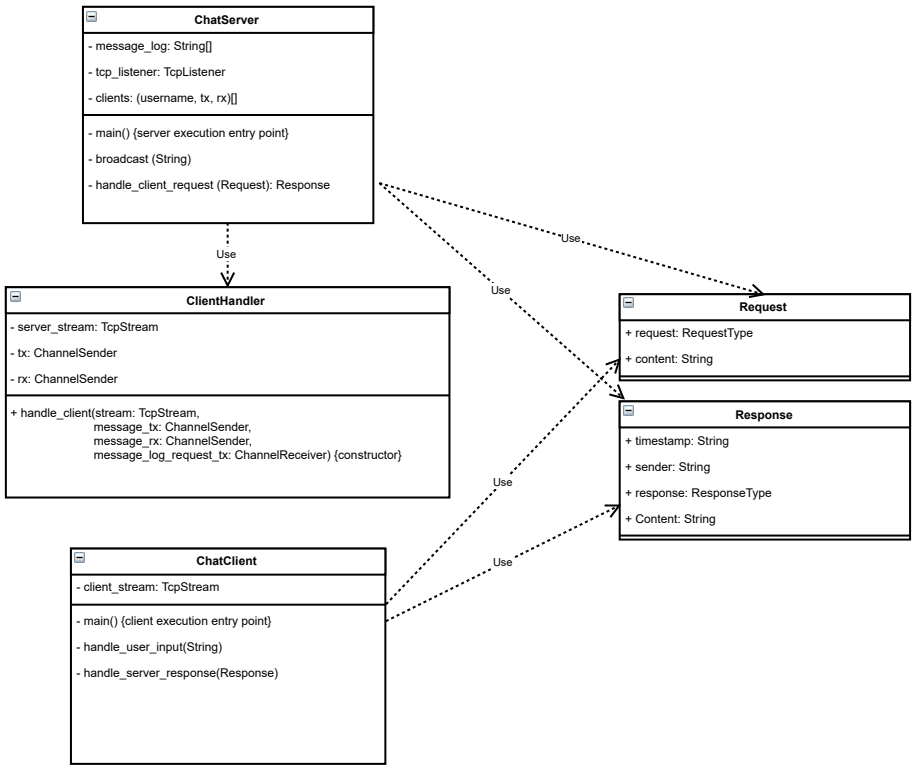
```
struct Response {
    timestamp: String,
    sender: String,
    response: ResponseType,
    content: Option<String>,
}

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

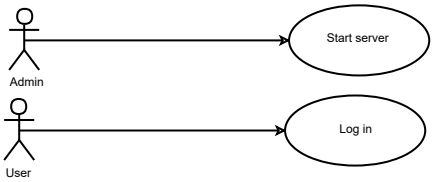
The only requests that is valid for a user that is not yet logged in is `login` and `help`. All other

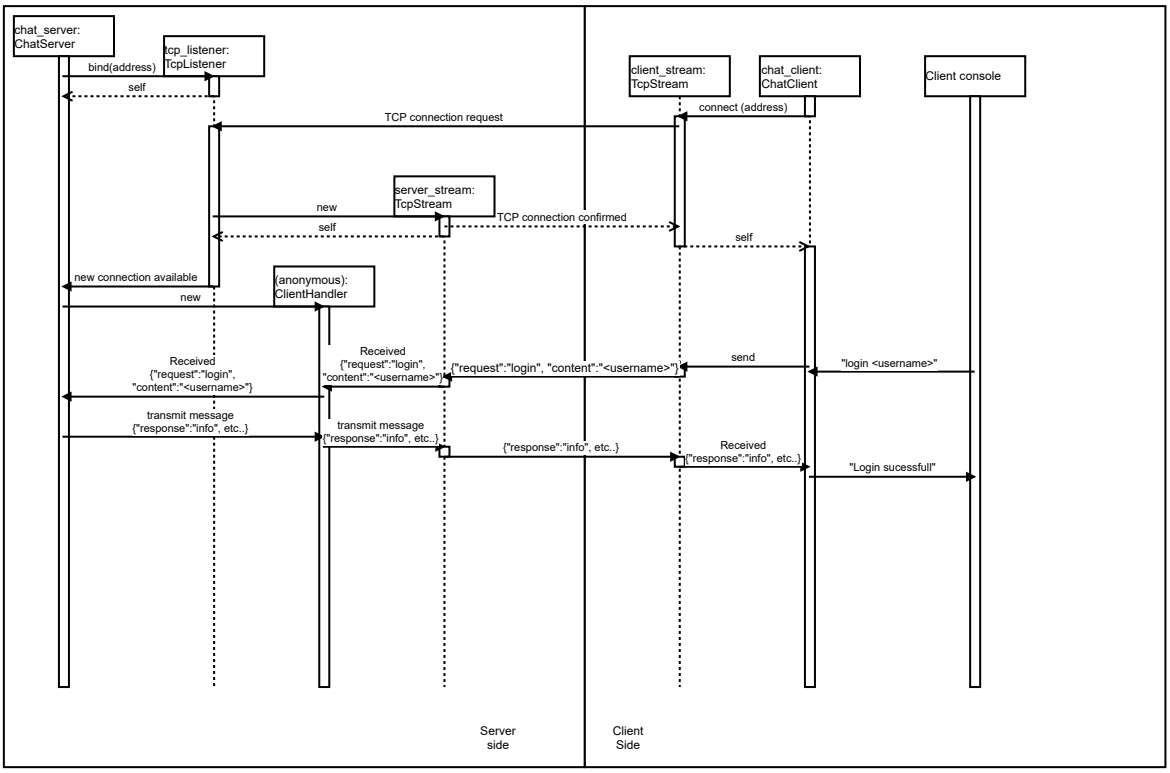
responses will be answered with an error response.

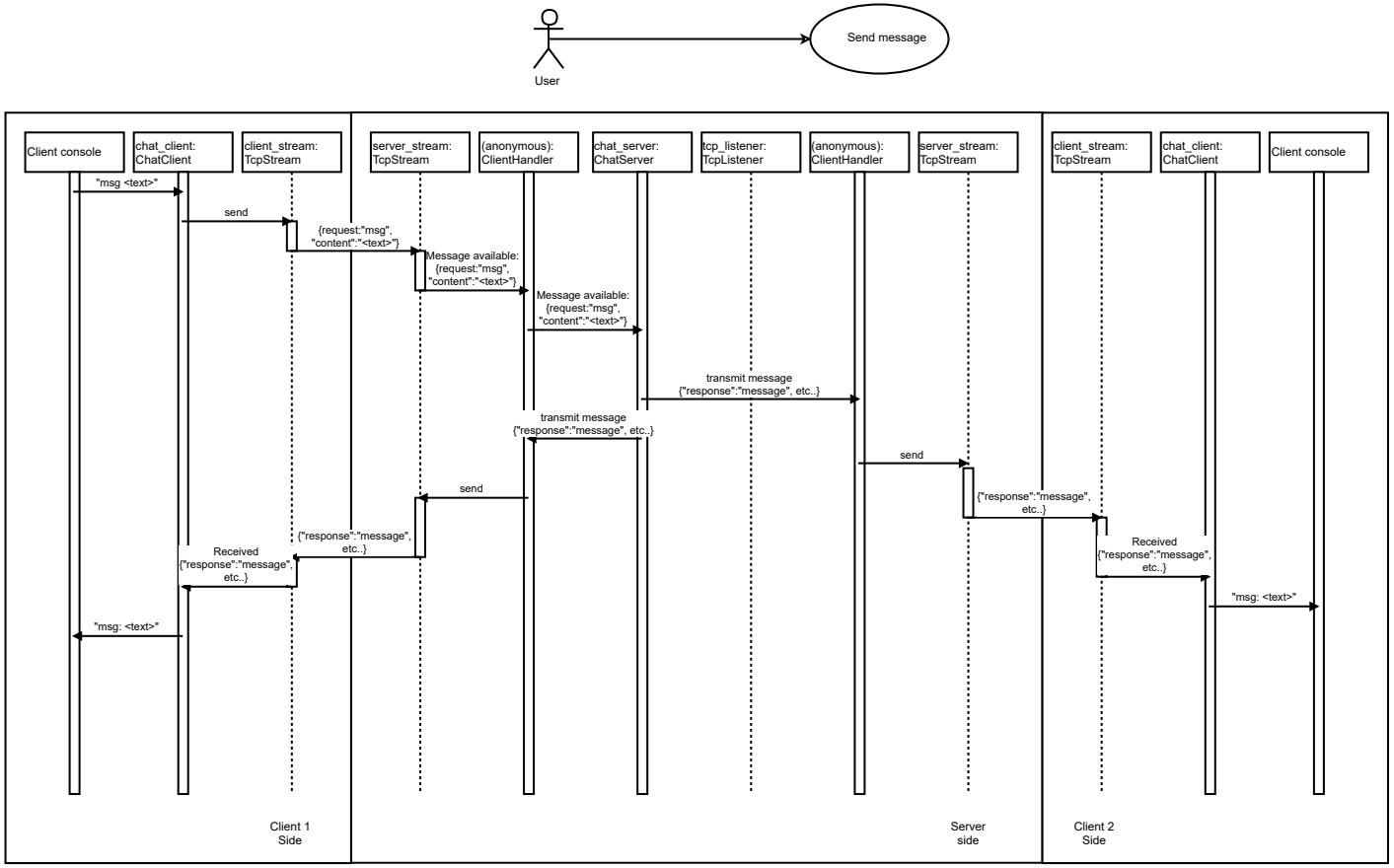
Client

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 recieves a response form the server it will parse this response and show the relevant information to the user.


ChatServer
- message_log: String[]
- tcp_listener: TcpListener
- clients: (username, tx, rx)[]
- main() {server execution entry point}
- broadcast (String)
- handle_client_request (Request): Response
Use
Use
Use
ClientHandler
- server_stream: TcpStream
- tx: ChannelSender
- rx: ChannelSender
+ handle_client(stream: TcpStream,
message_tx: ChannelSender,
message_rx: ChannelSender,
message_log_request_tx: ChannelReceiver) {constructor}
Request
+ request: RequestType
+ content: String
Response
+ timestamp: String
+ sender: String
+ response: ResponseType
+ Content: String
Use
Use
ChatClient
- client_stream: TcpStream
- main() {client execution entry point}
- handle_user_input(String)
- handle_server_response(Response)


Admin
Start server
User
Log in


chat_server:
ChatServer
tcp_listener:
TcpListener
client_stream:
TcpStream
chat_client:
ChatClient
Client console
bind(address)
self
connect (address)
TCP connection request
server_stream:
TcpStream
new
self
TCP connection confirmed
self
new connection available
(anonymous):
ClientHandler
new
Received
{"request":"login",
"content":"<username>"}
Received
{"request":"login",
"content":"<username>"}
{"request":"login", "content":"<username>"}
send
"login <username>"
transmit message
{"response":"info", etc..}
transmit message
{"response":"info", etc..}
{"response":"info", etc..}
Received
{"response":"info", etc..}
"Login sucessfull"
Server
side
Client
Side


User
Send message
Client console
chat_client: ChatClient
client_stream: TcpStream
server_stream: TcpStream
(anonymous): ClientHandler
chat_server: ChatServer
tcp_listener: TcpListener
(anonymous): ClientHandler
server_stream: TcpStream
client_stream: TcpStream
chat_client: ChatClient
Client console
"msg <text>"
send
{request:"msg", "content":"<text>"}
Message available: {request:"msg", "content":"<text>"}
Message available: {request:"msg", "content":"<text>"}
transmit message {"response":"message", etc..}
transmit message {"response":"message", etc..}
send
send
{"response":"message", etc..}
Received {"response":"message", etc..}
{"response":"message", etc..}
Received {"response":"message", etc..}
"msg: <text>"
"msg: <text>"
Client 1 Side
Server side
Client 2 Side