**CHAT SERVER CONSOLE APPLICATION   
TECHNICAL DOCUMENT**

**Scope:**

This application will act as a server console application to create communication between clients.

**Features:**

* In-thread multiplexing
* Multithreading
* UDP broadcast
* User Registration
* User Authentication
* Hashtable
* Commands Execution
* Messaging
* Logging
* Program Exit

**Command Character:**

~ will be used to signal the beginning of a command  
  
Other characters can be specified when starting the server.

**Command Format:**

For any command you would simply use ~ before the command then its options shortly thereafter.

**Example:**   
  
Command structure: ~register username password

Command in use: ~register johnsmith P@ssw0rd

**Command Table:**

**The table below will list the commands and their parameters:**

***Note: \* means optional parameter***

|  |  |  |
| --- | --- | --- |
| **Command** | **Parameters** | **Description** |
| help |  | Provides list of commands available |
| register | username password | Registers a user on the server. Users are stored in a hashtable once registered. Clients must be logged out for this command |
| login | username password | Attempts to login a Client. Provides feedback if successful, user not found, or incorrect password |
| logout |  | Logs the user out gracefully and initiates FIN/ACK/FIN/ACK to disconnect |
| getlist |  | Provides list of active clients currently logged in to the Client |
| send | username\* message | The default without username sends the same message to all active clients, with the username we send to the specific user.  For example:  **~send client1 test message** will send client1 a message  **~send test message**  Will send all active clients a message |
| getlog | user  public | user parameter will retrieve user logs for user commands public parameter will retrieve public messages |

**System Architecture:**

|  |  |
| --- | --- |
| **Component** | **Description** |
| **messageHandler** | handles multiplexing and messages between clients |
| **chatServer** | core application and connects all classes together to facilitate communication between clients. chatServer will include a clientHandler, messageHandler, and a public log |
| **clientHandler** | manages client connections including registration, and authentication. |
| **user** | handles user data such as username, password, and a personal log. |
| **logger** | handles functionality for a single log |

**Design Diagram:**

A diagram of a computer network

Description automatically generated

**Implementation Approach:**

Object-oriented, classes are encapsulated and handle their own features, and functionality.

**Project Map:**

A diagram of a chat server

Description automatically generated

**Sequence Diagrams:**

A diagram of a chat server

Description automatically generatedA diagram of a login

Description automatically generated

A diagram of a chat server

Description automatically generated

A diagram of a chat server

Description automatically generated

A diagram of a chat service

Description automatically generated A diagram of a chat service

Description automatically generated

A diagram of a chat

Description automatically generated

**Error Handling:**

|  |  |
| --- | --- |
| **Error** | **Description** |
| **ClientDisconnected** | Client connection was disrupted |
| **IncorrectUNorPW** | User input incorrect credentials |
| **AlreadyRegistered** | User is already registered to the hashtable |
| **RegistrationFailed** | Generic error message for multiple errors:   1. Username or Password length over 20 characters 2. User registration did not save to hashtable |
| **FailedToSend** | Failed to send message between clients |
| **CharLimitReached** | Message was over 255 characters |
| **LogRetrievalFailed** | Unable to retrieve log |
| **IncorrectCommand** | Command structure was not recognizable |
| **IncorrectPort** | Port Number is invalid |
| **IncorrectIP** | IP address is invalid |
| **NotLoggedIn** | There is no client logged in. This message is received when attempting to use a command when user is not logged in. |
| **InternalError** | Generic Error |
| **CapacityReached** | Server capacity has been reached |