# Chat Application Design Document

## Overview

This project implements a terminal-based group chat application using Python sockets and threading. It includes a server and multiple client instances that communicate over TCP. Clients feature a user interface with emoji support, clickable links, direct messaging and color-coded messages.

## Architecture

- **Language**: Python 3
- **Networking**: TCP sockets
- **Concurrency**: threading.Thread
- **UI**: Tkinter with ScrolledText widget
- **Host Setup**:
    - Server runs on: eos20.cis.gvsu.edu
    - Client connects via SSH tunnel on localhost:5000

---

## Server Design

### Socket Setup

- Host: 0.0.0.0
- Port: 5000
- SO_REUSEADDR allows immediate reuse after shutdown.

### Responsibilities

- Accept incoming TCP connections
- Receive username upon connection
- Maintain a list of active clients and usernames
- Spawn a thread per client connection
- Relay incoming messages to all other connected clients
- Support @username direct messages (DMs)

### Data Structures

- clients: List of active client sockets
- usernames: Dict mapping socket → username
- clients_lock: Threading lock for synchronized access

### Message Handling

- Global messages: [username] message
- Direct messages: @username message
    - Delivered only to specified user

---

## Client Design

### Startup

- Prompts for username
- Sends username to server
- Starts background thread to receive messages

### Socket Behavior

- Connects to localhost:5000 (via SSH tunnel)
- Receives and displays messages using a queue

### UI Features (Tkinter)

- ScrolledText display area (read-only)
- Input box (multi-line)
- Send button and emoji picker

### Message Display

- Color-coded sender names
- Highlight if @mention targets current user
- Auto-scroll to bottom on new messages

### Extra Features

- Emoji shortcut replacement (e.g. :smile: → 😊)
- Emoji picker GUI
- Clickable hyperlinks using webbrowser.open
- Tag-based color styling (DMs, mentions, etc.)

---

## Threading Model

- Server:
    - One thread for accepting connections
    - One thread per client to receive and relay messages
- Client:
    - Main thread runs UI loop
    - Background thread receives messages from socket and posts to message queue

---

## Testing and Demo

### Local Testing

- Run server and multiple clients on the same machine using different terminals.

### Remote Testing

- Server: SSH into eos20.cis.gvsu.edu and run python3 server.py
- Client:
    - Use SSH tunnel: ssh -N -L 5000:localhost:5000 yourid@eos20.cis.gvsu.edu
    - Run python Client.py locally

### Demo Steps

- Show emoji picker
- Show clickable link
- Use @username to send DM
- Open multiple clients and demonstrate broadcast

---

## Git and Collaboration

- Code maintained in a Git repository
- Branches used for major features: ui-features, server-logic, dm-support
- Code reviews and merge approvals coordinated by team

---

## Final Notes

This chat application demonstrates real-time, multi-user communication using Python's socket and threading libraries. The UI supports useful features while remaining light and responsive, fulfilling all minimum and extra credit requirements.