PingPal Project Proposal

Ping Pal: A Real-Time, Terminal-Based Multi-User Chat Application Built w/ TCP Sockets in C.

Team Members & Roles

Name	Role
Keaton Rowley	Systems architecture, protocol design, socket programming
Álvaro Salvador	Client interface, command parsing, terminal input/output
Godbless Amankwah	Server-side session handling, user/channel state management

Project Overview

PingPal is a terminal-based, multi-user chat application that enables real-time text communication over a network using TCP sockets. It follows a centralized server/client model and allows for multiple users to connect simultaneously. Multiple users can connect simultaneously and exchange messages in public or private channels within a lightweight terminal-based UI. This project is designed to simulate and demonstrate application-layer protocols, socket-based communication, message framing, multi-threaded concurrency, and command parsing.

Core Features

Feature	Description
Multi-client server	Accepts multiple concurrent client connections using select() or threads.
Username support	Users register a unique name when joining.
Public channels	Joinable rooms (#general, #help, etc.) to group conversations.
Private messaging	Direct messages between users using commands like /msg @username.
Command-based interaction	Commands such as /join, /leave, /list, /who, /quit.
Message broadcasting	Messages are sent to appropriate channel members only.

Stretch Features (Time-Permitting)

- Persistent message logs (using local file system)
- Basic user authentication (username/password)
- Colored output per user/channel (terminal-friendly)
- Admin commands (/kick, /mute)
- Rate limiting or spam protection
- Server configuration via a .conf file

System Architecture

Components:

- Chat Server
    - Accepts TCP connections on a designated port.
    - Spawns a thread or uses select() to manage each client.
    - Parses incoming commands and routes messages.
    - Manages shared state (connected users, active channels, message history).
- Chat Client
    - Connects to the server and handles user input/output.
    - Sends user commands or messages via socket.
    - Asynchronously listens for server broadcasts.

Proposed Message Format:

makefile
CopyEdit
<COMMAND> <TARGET> <MESSAGE>\n
Examples:
MSG #general Hello everyone!
MSG @alice Hey, how are you?
JOIN #random
QUIT

Technology Stack

Component	Choice
Language	C (C99 or newer)
Networking	POSIX Sockets (BSD sockets API)
Concurrency	POSIX Threads (pthread)
I/O Multiplexing	select() or poll() (optional for scalability)
Terminal I/O	Standard ANSI codes
OS Compatibility	Unix/Linux

Development Timeline

Week	Tasks & Milestones
1	Finalize design. Set up Git repo. Divide responsibilities. Test socket setup.
2	Implement basic server/client connection; Manual message exchange
3	Support multiple clients. Add usernames and basic broadcasting.
4	Implement a channel system. Add private messaging and command handlin.
5	Add message history. Polish user interface. Robust error handling.
6	Final testing. Add documentation and demo. Prepare a presentation/report.

Testing Strategy

- Manual Testing: Multiple clients run in separate terminal sessions.
- Automated Stress Test: Use scripts to simulate 10+ clients sending messages.
- Unit Testing: For parsing logic (command/token validation).
- Edge Case Handling: Test malformed inputs, network disconnections, duplicate usernames.

Risks and Mitigation

Risk	Mitigation Strategy
Race conditions or deadlocks	Design with mutex-protected shared state. Extensive testing.
Message parsing bugs	Strict command format with token validation and logging.
Too many features	Core chat system completed by Week 4. Stretch goals only after core stability
Terminal compatibility issues	Stick to ANSI escape codes. Test on standard Linux terminals

Deliverables

- Fully functioning chat server and client code (in C)
- README (build instructions, usage, architecture overview)
- Final PDF report
- Code documentation (inline comments, header file documentation)
- Live presentation