Relatório Final - Sistema de Chat Multiusuário (Cliente/Servidor TCP)

1. Diagrama de Sequência Cliente-Servidor

Cliente ---> Servidor: conexão TCP

Cliente ---> Servidor: envia mensagem

Servidor ---> Logger: registra mensagem (TSLogger)

Servidor ---> Clientes: broadcast da mensagem

Cliente <--- Servidor: recebe mensagens de outros clientes

Cliente ---> Servidor: encerra conexão

Servidor ---> Logger: registra desconexão

2. Mapeamento: Requisitos → Código

RF01: Servidor deve aceitar múltiplos clientes simultâneos → Implementado em *server.cpp* com *accept()* em loop e *std::thread(handle_client, ...)*

RF02: Cada cliente deve ser atendido em thread separada → Implementado em *server.cpp*, função *handle_client* executada em threads destacadas

RF03: Mensagens enviadas por um cliente devem ser retransmitidas a todos os outros (broadcast) → Implementado em *server.cpp*, função *broadcast* protegida por *std::mutex*

RF04: Logging concorrente das mensagens → Implementado em *tslog.cpp/h* (classe TSLogger), integrado em *server.cpp* via *logger.info(...)*

RF05: Cliente CLI deve permitir enviar e receber mensagens → Implementado em *client.cpp* com threads (*receive_messages* para recepção e loop *stdin* para envio)

RF06: Estruturas compartilhadas devem ter proteção contra corrida → Uso de *std::mutex* em clients (*server.cpp*) e em *queue* do logger (*tslog.cpp*).

3. Relatório de Análise com IA

O sistema desenvolvido segue os princípios de programação concorrente e comunicação em rede TCP.

Pontos fortes identificados pela análise automática:

- Uso adequado de threads: servidor cria uma thread por cliente, garantindo concorrência.

- Logger thread-safe: separação clara entre produtores (threads clientes) e consumidor (worker thread).

- Proteção de dados compartilhados: uso consistente de *std::mutex* e *condition_variable*.

- Estrutura modular: *server.cpp*, *client.cpp* e *tslog.cpp* são bem separados e favorecem manutenção.

Riscos e pontos de melhoria:
- Não há tratamento sofisticado de erros (ex.: queda inesperada de cliente, falha de socket)
- Não há autenticação ou segurança nas mensagens (poderia ser considerado em versões futuras)
- Logging atual escreve apenas em arquivo local.

Conclusão: O protótipo cumpre os requisitos definidos, é funcional e fornece base sólida para evolução. A arquitetura é clara e escalável, podendo ser expandida com segurança, autenticação e métricas de desempenho.