UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

CAMPUS CHAPECÓ

CURSO DE CIÊNCIA DA COMPUTAÇÃO

COMPONENTE CURRICULAR: GEX635 - TÓPICOS ESPECIAIS EM COMPUTAÇÃO XIII

Prof. Marco Aurélio Spohn

RIAN BORGES BARBOSA

Aplicação de Chat Utilizando MQTT com Typescript

CHAPECÓ

2025

Sumário

1 Resumo

Este projeto consiste no desenvolvimento de uma aplicação de mensagens instantâneas ("chat") utilizando estritamente o protocolo MQTT (Message Queuing Telemetry Transport). A aplicação foi desenvolvida para o ambiente Linux, utilizando a linguagem TypeScript/Node.js e a biblioteca paho-mqtt sobre WebSockets.

O objetivo principal foi implementar um sistema de comunicação robusto que suporte mensagens um-para-um e em grupo, garantindo a persistência de informações de estado (usuários online e grupos existentes) mesmo para clientes que se conectem tardiamente.

2 Arquitetura do Sistema

A arquitetura do sistema baseia-se no modelo *Publish-Subscribe*, mediada por um *Message Broker* Mosquitto. O sistema foi projetado para ser distribuído, onde cada cliente mantém uma cópia local do estado global (usuários e grupos), sincronizada através de mensagens retidas no servidor MQTT.

2.1 Infraestrutura

A infraestrutura está orquestrada via Docker Compose, conforme ilustrado na Figura 1, e é composta por dois componentes principais:

- **Broker MQTT (Mosquitto):** Configurado para aceitar conexões via WebSockets na porta 9001. Utiliza persistência em disco (`mosquitto.db`) para manter o estado de mensagens retidas e assinaturas duráveis, garantindo que grupos e usuários não "desapareçam"em caso de reinício do serviço.
- **Clientes (Node.js):** Aplicações TypeScript que atuam como clientes, conectando-se ao broker via biblioteca `paho-mqtt` (sobre um *wrapper* de WebSocket).

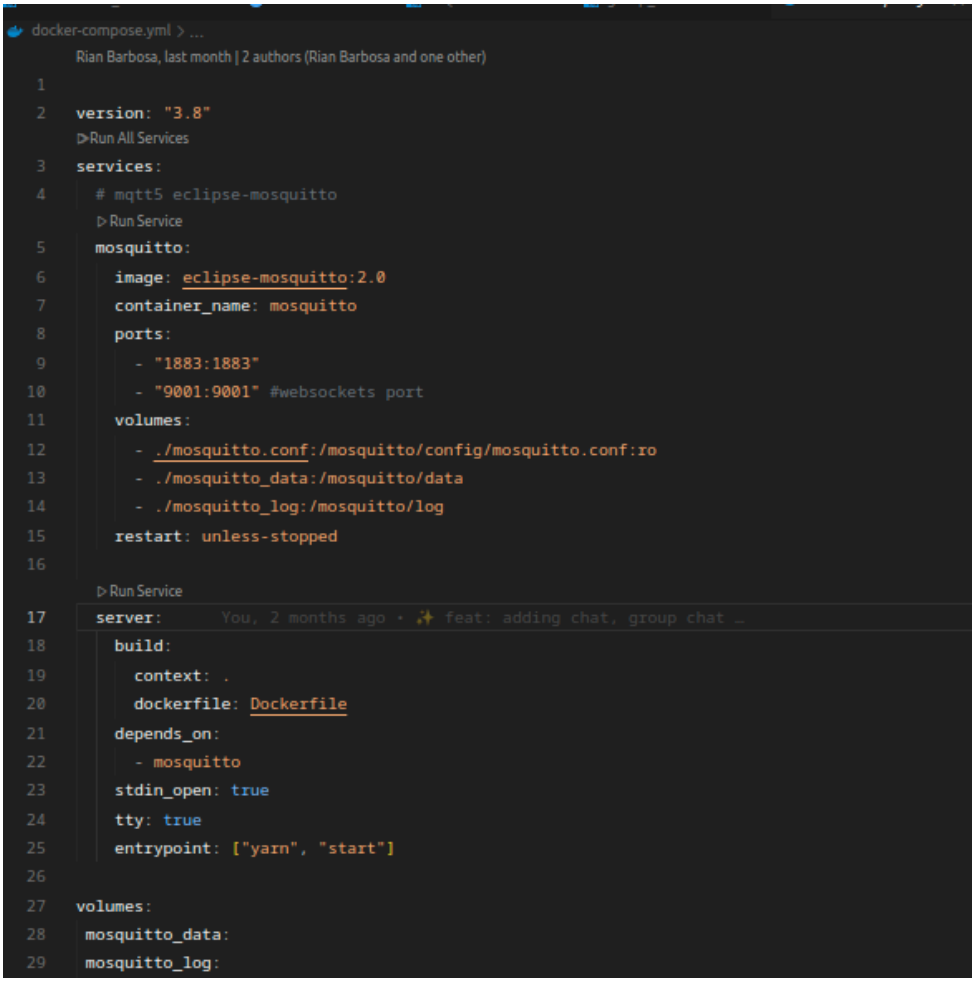

Figura 1: Estrutura Docker Compose.

2.2 Estrutura de Tópicos

A estrutura é organizada por meio de uma hierarquia de tópicos para separar o plano de controle do plano de dados (mensagens de chat).

1. **Descoberta de Usuários (`USERS/+`)**: Cada cliente publica uma mensagem retida no tópico `USERS/{userID}` contendo seu status (*online/offline*) e a data da última atividade. Isso permite que novos clientes descubram quem está online imediatamente após a conexão.

2. **Gerenciamento de Grupos (`GROUPS`)**: Um único tópico global utilizado para sincronização de metadados de grupos (criação, atualização de membros, exclusão). As mensagens são retidas para garantir consistência eventual.

3. **Sinalização de Controle (`{userID}_Control`)**: Cada usuário subscreve ao seu próprio tópico de controle. Este canal é utilizado para receber solicitações de conversa privada,

convites de grupo e negociação de tópicos (Handshake).

2.3 Canais de Chat

- **Privado:** Tópicos dinâmicos no formato `UserA_UserB_Timestamp`.
- **Grupo:** Tópicos no formato `GROUP_NomeDoGrupo`.

3 Implementação

A aplicação foi desenvolvida utilizando a linguagem TypeScript para garantir a tipagem estática das interfaces de mensagens (definidas em `interface_config.ts`). O código segue o padrão de projeto *Service-Controller*, separando a lógica de conexão da lógica de negócios.

3.1 Diagrama de Classes Simplificado

A Figura 2 demonstra como o serviço MQTT centraliza a conexão e instancia os controladores responsáveis pela lógica de negócio.

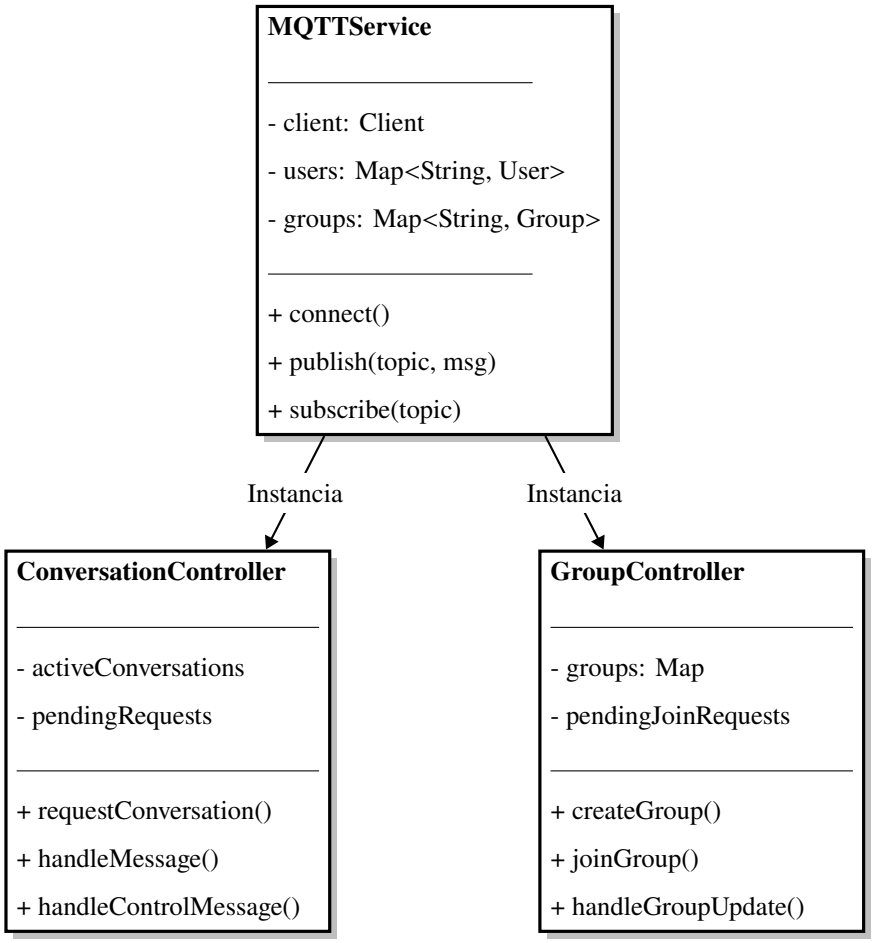

Figura 2: Diagrama de classes simplificado da aplicação cliente.

3.2 Serviço MQTT Central e Adaptação Paho-MQTT

O projeto utiliza a biblioteca `paho-mqtt`, nativa para navegadores. Para execução em Node.js, foi necessário implementar um *polyfill* de WebSockets injetando a biblioteca `ws` no escopo global em `MQTTService.ts`:

```
1 global.WebSocket = require('ws');
2 import paho, { Client } from 'paho-mqtt';
```

Essa classe gerencia a conexão (QoS 1), o roteamento de mensagens para os controladores baseando-se no tópico e os *Heartbeats* para detecção de usuários offline.

3.3 Lógica de Negócio (Controladores)

3.3.1 ConversationController

Gerencia o ciclo de vida de chats *one-to-one*. Implementa um protocolo de *handshake* de três vias, conforme detalhado no diagrama de sequência da Figura 3:

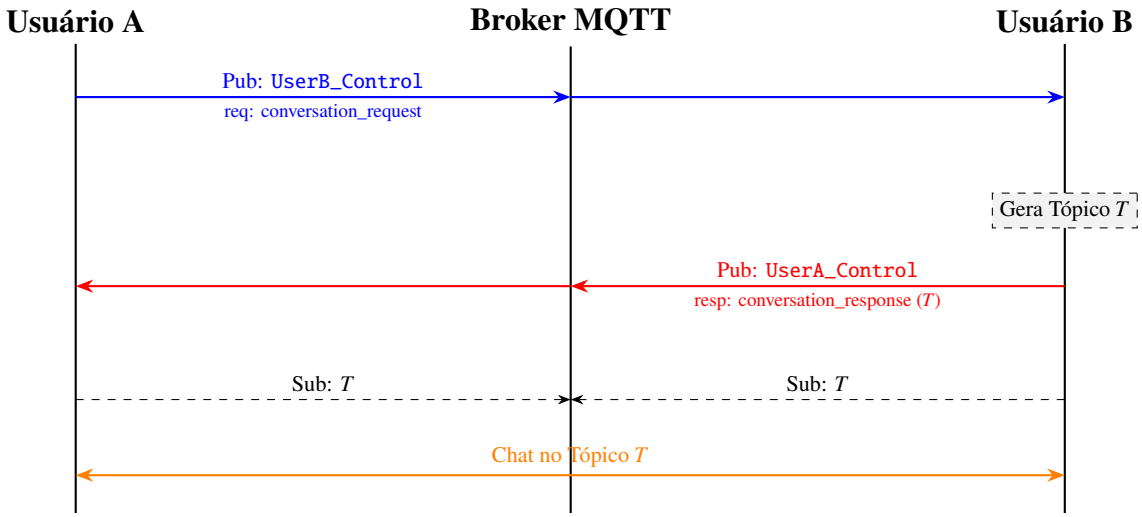

Figura 3: Diagrama de sequência do protocolo de Handshake.

3.3.2 GroupController

Gerencia a lógica de grupos, onde apenas o líder pode excluir o grupo ou aprovar a entrada de novos membros. A sincronização ocorre enviando um objeto JSON com o array completo de membros para o tópico `GROUPS`, garantindo que todos os clientes tenham a lista atualizada localmente.

3.4 Interface de Usuário (CLI)

A interface é baseada em linha de comando, utilizando a biblioteca `console-table-printer` para exibição tabular de dados e códigos de escape ANSI para colorização dos nomes dos usuários no chat em grupo.

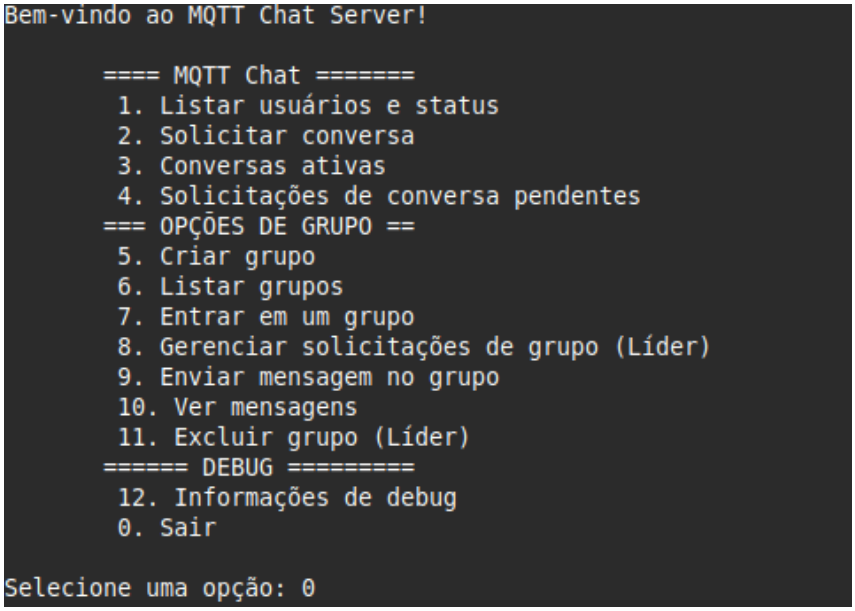

Figura 4: Interface da aplicação CLI.

4 Conclusão

O projeto cumpriu o objetivo de demonstrar a viabilidade do protocolo MQTT para aplicações de chat além do escopo tradicional de IoT. A arquitetura baseada em tópicos hierárquicos mostrou-se flexível o suficiente para emular funcionalidades complexas de mensageria moderna.

A principal vantagem da abordagem foi a descentralização da lógica de estado: o broker atua apenas como transportador e repositório de chaves-valor (mensagens retidas), enquanto a inteligência da aplicação reside nos clientes (Edge Computing). Para trabalhos futuros, sugere-se a implementação de criptografia ponta-a-ponta no *payload* das mensagens.