**UNIVERSIDADE FEDERAL DA FRONTEIRA SUL**
**CAMPUS CHAPECÓ - SC**
**CURSO DE CIÊNCIAS DA COMPUTAÇÃO**

**LUCAS MAHLE**
**RODRIGO RECH**
**ELVIS DE SOUZA MACHADO**

# CHAT BASEADO NO PROTOCOLO MQTT

**CHAPECÓ - SC**
**2022**

**LUCAS MAHLE**
**RODRIGO RECH**
**ELVIS DE SOUZA MACHADO**

# CHAT BASEADO NO PROTOCOLO MQTT

Trabalho apresentado ao Curso Ciências da Computação da Universidade Federal da Fronteira Sul (UFFS) como requisito parcial para aprovação na disciplina de Tópicos especiais em computação XIII

Professor: Marco Aurélio Spohn

**CHAPECÓ - SC**
**2022**

# Introdução

Desenvolvimento de uma aplicação com um bate-papo baseado no protocolo MQTT junto a isso será utilizada a linguagem NODE.js. Disponibilizando um front-end para passar informações para ser feito login com alguns usuários pré-definidos sendo possível abrir conversas, criar grupos e ter troca de mensagens entre os usuários logados na aplicação.

# 1.0: Startar aplicação

Suba o broker Mosquitto com a configuração do nosso projeto utilizando o comando abaixo
$ mosquitto -c ./assets/mosquitto.conf

Em seguida basta subir a aplicação javascript e rodá-la em seu navegador. Nosso grupo fez isso a partir da extensão Live Server do Visual Studio Code. Com essa extensão instalada, basta clicar no botão "Go Live" que aparece na IDE, e o projeto sobe e é aberto no navegador.

# 2.0: Funcionalidades

## 2.1: Login

Primeiramente informe o id do seu usuário, ele pode ser 1, 2 ou 3 (valores que definimos no arquivo users.json). Após isso, você abrirá a tela principal da aplicação. Quando você se identifica como usuário, o valor fica salvo no navegador, então você sempre estará logado com esse usuário a menos que limpe o LocalStorage do navegador. Você pode entrar com mais de um usuário utilizando a janela anônima do navegador ou utilizando outro navegador.

## 2.2: Chat de usuários

Após o login, o nome dos usuários presentes no arquivo users.json aparecem na tela, ao lado de um círculo vermelho ou verde que identifica se eles estão online ou offline. O usuário pode clicar no nome dos usuários que estão online e enviar uma solicitação de conversa, caso seja aceita, o chat fica disponível para ambos trocarem mensagens.

## 2.3: Grupos

Após o login, é possível criar grupos, pedir permissão para fazer parte de um grupo onde é enviado a solicitação para o líder do grupo, que pode aceitar ou recusar o pedido para participar do grupo, após o lider aceitar a solicitação é possivel os usuarios enviarem mensagens dentro do grupo, pode ser feita a exclusão do grupo pelo líder do grupo.

# 3.0: Funcionamento interno da aplicação

## 3.1: Mosquitto

O broker Eclipse Mosquitto configurado pelo projeto em assets/mosquitto.conf

- Ip e Porta: Configuramos a conexão como websocket e reservamos a porta 9001 e não explicítamos o ip pois por default ele é 127.0.0.1.
- Autenticação: Ignora conceitos fundamentais de segurança, como a autenticação dos clientes via usuário e senha pois contém a configuração allow_anonymous valorada como true.

- Web socket: Faz conexão web socket (conexão full duplex)
- ACL: Ignora restrição de postagem por usuário ou assinatura de tópicos por usuário

## 3.2: Paho

Utilizamos a versão 1.1.0 da biblioteca Eclipse Paho para Javascript. Com ela, conseguimos publicar e assinar no broker Eclipse Mosquitto.

- QoS: Ao publicarmos as mensagens ou assinarmos os tópicos não estamos definindo o QoS pois por default do Paho ele é 0, logo as conexões não são persistentes (Clean Sessions) em toda nossa aplicação.
- Id do Usuário no Mosquitto: Ao estabelecermos a conexão com o Mosquitto estamos definindo o id do usuário no broker como o id do usuário informado por ele mesmo no início da aplicação.
- Path do Broker: Não estamos definindo o path ao conectarmos ao broker pois o path definido por default pelo Pahot é /mqtt.
- Last Will Message: Não utilizamos o recurso Last Will Message. Logo, caso o publisher fique incomunicável com o broker devido a problemas de rede, ele não é programado para enviar nenhuma mensagem de aviso aos assinantes.
- Retain Flag: Não utilizamos a retain flag em nenhum momento, que guarda o estado da mensagem, ou seja, apenas a última publicação do tópico e sempre encaminhá-la quando um novo assinante for acionado.

## 3.3: Tópico

Conforme enunciado do trabalho, criamos um conjunto de tópicos para utilização do chat. Abaixo segue a lista de tópicos. Contudo essa documentação aprofunda quando e como cada tópico é utilizado nas seções que virão a seguir, como 3.4, 3.5, etc...

- Tópico USERS: referente ao status de online ou offline de cada usuário
- Tópico {idUsuario}_Control: {idUsuario} no caso é uma variável, variando conforme id do usuário da aplicação. Caso outro usuário deseje iniciar uma conversa, a aplicação dele enviará uma publicação do tipo CONVERSATION para esse tópico.
- Tópico do Chat: o nome do tópico será o id de ambos e o timestamp. Nele são publicadas as mensagens do chat.
- Tópico Groups: tópico que gerencia grupos existentes.
- Tópico de Chat do Grupo: o tópico usado para envio de mensagem entre os grupos

## 3.4: Login

A primeira tela exibida ao usuário é a tela de "Login", na qual o usuário deve informar seu ID. Os IDs de usuários válidos estão armazenados no arquivo users.json. Ao receber um ID válido a aplicação exibe a tela de chat e usuários disponíveis no arquivo users.json.

## 3.5: Usuários online/offline

Após o Login, a aplicação se conecta com o broker Eclipse Mosquitto. Quando a conexão é feita, a aplicação passa a assinar o tópico USERS, referente ao status de online ou offline dos usuários e pública que está ONLINE, enviando um pacote com seu ID, tipo CONTROL

com status TRUE conforme mostrado no [Capítulo 4.2]. As outras aplicações que assinaram o tópico, ao receber uma publicação nesse formato, atualizam na tela o usuário como online e fazem uma publicação com seu ID e tipo UPDATE[Capítulo 4.4], os assinantes colocam todos que publicaram como online. Dessa forma o novo usuário sabe quais estão online desde antes de ele assinar o tópico. Também é publicado um pacote com ID, tipo CONTROL com status FALSE caso o usuário feche a aplicação no navegador[Capítulo 4.3], um callback foi criado para monitorar caso a janela ou guia seja fechada e faz o envio antes da ação acontecer. Com isso os outros assinantes marcam o usuário como offline.

## 3.6: Iniciação de conversa

Após a conexão com o broker ser realizada, também é criado o tópico com o nome "{idUsuario}_Control"[Capítulo 4.1] e o usuário se inscreve nesse tópico. Caso outro usuário deseje iniciar uma conversa, a aplicação dele enviará uma publicação do tipo CONVERSATION[Capítulo 4.5]. A aplicação do usuário que recebeu a solicitação abrirá uma informará que há um solicitação de conversa na qual ele pode aceitar ou recusar, sua resposta resulta em uma publicação do tipo Conversation_Response [Capítulo 4.5]. Caso seja aceita e criado um novo tópico e ambos os usuários entram nele, o nome do tópico será uma união do id de ambos e o timestamp. Quando um dos usuários desloga, o outro usuário sabe por conta do tópico USERS[Capítulo 4.3] e sai do tópico da conversa. Caso o usuário clique para iniciar a conversa com outro usuário, porém o tópico já existe, as mensagens são exibidas e o tópico não precisa ser recriado.

## 3.7: Troca de mensagens

Após o tópico do chat entre dois usuários ser criado, quando qualquer um deles submete a mensagem a aplicação pública no tópico, no formato [Capítulo 4.6] e salva em um array de histórico, armazenando o histórico para caso o usuário abra outro chat e depois reabra o chat que foi ocultado. Essa feature também é útil para caso o usuário receba a mensagem mas o chat não esteja aberto, por exemplo, quando ele está conversando com outra pessoa, nesse caso a conversa aparecerá quando o usuário voltar a abrir o chat cuja mensagem foi recebida. Quando o usuário desloga o chat ativo e a conversa do usuário, a tela é limpa o mesmo acontece com o grupo

## 3.8: Grupos

Ao ser criado um grupo a aplicação pública no tópico GROUPS, no formato [Capítulo 4.7] assim ficando visivel para todos usuarios, onde um usuário pode fazer a solicitação de entrada no grupo no formato [Capítulo 4.8] onde é feita uma requisicao de entrada no grupo para o líder/criador do grupo que responde a requisição feita no formato [Capítulo 4.9] ao aceitar a entrada do usuário no grupo é feita a atualização da informação, onde é enviado para o tópico GROUPS pelo líder no formato [Capítulo 4.10], feito isso é possivel enviar mensagens para chegar em todos os usuarios que participam do grupo onde envia com o formato [Capítulo 4.11], para a exclusão do grupo o lider envia um objeto no formato [Capítulo 4.12] para o tópico GROUPS informando que o grupo foi apagado.

# 4.0: Tópicos e Publicações

Este capítulo explica e exemplifica os tipos de publicações e tópicos utilizados para completar o entendimento do capítulo anterior

## 4.1: Instância

Quando uma nova instância é criada, é definido o tópico que controla o identificador da instância, que será {{id}}_control. Por ex: id: 9 tópico: 9_control

## 4.2: Tópico Online

Assim que a instância é criada, é enviado para o tópico users o seu id. Então, todas as instâncias já criadas recebem a informação que o usuário está online. O padrão de envio dessa informação é um json com o seguinte formato:

```
{
   "id": "9_control",
   "status": true,
   "type": "control"
}
```

Sendo status true para ONLINE e false para OFFLINE.
A instância quando recebe essa informação, altera a lista de usuários onlines com a informação do usuário que logou.

## 4.3: Tópico Offline

Quando o usuário clica no botão de SAIR, é enviado para o tópico users a informação de logout. O padrão de envio dessa informação é um json com o seguinte formato:

```
{
   "id": "9_control",
   "status": false,
   "type": "control"
}
```

Sendo status true para ONLINE e false para OFFLINE.
A instância quando recebe essa informação, altera a lista de usuários onlines com a informação do usuário que deslogou.

## 4.4: Processando usuário online

Quando receber a informação de usuário online, todas as instâncias online, enviam para o tópico users um sinal, para que o usuário que acabou de se conectar, consiga saber quem está online. Sendo assim, todos enviam a seguinte informação:

```
{
   "id": "9_control",
   "type": "update"
}
```

Essa informação é relevante somente ao usuário que acabou de se conectar, para os demais, a informação é ignorada. O mesmo acontece com grupos, todas instâncias cujo são líderes de algum grupo, enviam para o tópico groups, da mesma maneira que acontece na seção "Atualizando informações do grupos" deste documento.

```
{
   "type": "update",
   "id": "20211226150122",
   "name": "Grupo do Zequinha",
   "leader": "foo_control",
   "members": ["bar_control"]
}
```

## 4.5: Início conversa

Quando um usuário clica em outro usuário para iniciar uma conversa, é enviado para a instância alvo uma requisição para iniciar uma conversa. Após o usuário alvo aceitar ou recusar a conversa, é enviado a informação ao usuário solicitante, e caso seja uma resposta positiva, é definido um tópico para a conversa entre ambos, no formato {{solicitante}}_{{alvo}}_{{timestamp}} Ex: O usuário cujo possui tópico de controle foo_control deseja iniciar com um usuário cujo o tópico de controle é bar_control. A instância do solicitante envia uma mensagem para a instância do alvo com o seguinte conteúdo:

```
{
   "from": "foo_control",
   "type": "conversation"
}
```

A instância do alvo processa e exibe o modal para o usuário confirmar ou recusar, quando o usuário aceita/recusa, a instância do alvo manda uma mensagem para a instância solicitante com o seguinte conteúdo:

```
{
   "from": "bar_control",
   "type": "conversation_response",
   "status": true
}
```

Sendo status true para ACEITAR e false para RECUSAR.
Caso seja aceita a conversa, é criado o tópico entre ambos, considerando como timestamp a hora e minuto do momento. Nesse caso seria foo_bar_1534, sendo foo o id do solicitante, bar o id do alvo e 1534 o timestamp do momento.

Quando um dos dois usuário deslogar, é enviado para o tópico users essa informação, logo, todos que possuem alguma conversa ativa, removem a informação. Tornando necessário uma nova requisição é aceita quando o usuário logar novamente

## 4.6: Envio de mensagem

Uma vez a conversa ativa, quando um usuário envia uma mensagem, a instância envia para o alvo a seguinte informação:

```
{
   "from": "foo_control",
   "type": "message",
   "payload": "Hello World!",
   "time": "2021-12-23T20:33:13.471Z"
}
```

## 4.7 Criando novo grupo

Quando um usuário decide criar um grupo, a aplicação irá requisitar o nome do grupo. Após isso, o grupo será criado e será replicado a todos as instâncias pelo tópico groups. Também é gerado um id para identificar o grupo, esse id será formado pelo timestamp no formato {{id do líder}}_{{ id do grupo }}.

```
{
   "type": "new",
   "id": "20211226150122",
   "name": "Grupo do Zequinha",
   "leader": "foo_control",
   "members": []
}
```

Todos que receberem essa informação, atualizaram a lista de grupos, considerando o id para evitar duplicidade na atualização de informações.

## 4.8 Requisitando entrada no grupo

Quando um usuário desejar ingressar em um grupo, um pedido será enviado ao líder, com a seguinte informação:

```
{
   "type": "request",
   "id": "20211226150122",
   "from": "bar_control",
}
```

Sendo id, o identificador do grupo.

## 4.9 Respondendo requisição para entrar no grupo

Assim que o líder receber a informação de requisição para entrada no grupo, ele pode aceitar ou recusar a entrada, após essa ação, será enviada para o usuário solicitante a seguinte informação:

```
{
   "type": "response",
   "id": "20211226150122",
   "leader": "foo_control",
   "status": true,
}
```

Sendo status true para ACEITE e false para RECUSADO.

## 4.10 Atualizando informações do grupos

Se um novo usuário entra em um grupo, o líder envia para o tópico groups as nova informações do grupo:

```
{
   "type": "update",
   "id": "20211226150122",
   "name": "Grupo do Zequinha",
   "leader": "foo_control",
   "members": ["bar_control"]
}
```

Todos que receberem a informação, vão conseguir atualizar a lista de grupos.

## 4.11 Enviando mensagens no grupos

Quando um usuário manda uma mensagem em um grupo, envia para o tópico groups:

```
{
   "from": "foo_control", -> usuário que mandou
   "group": "123123", -> grupo destino
   "type": "group_message", -> tipo do pacote
   "payload": "Hello World!", -> conteúdo do pacote
   "time": "2021-12-23T20:33:13.471Z" -> tempo
}
```

Todos os integrantes recebem a informação.

## 4.12 Apagando o grupo

Quando o líder resolve apagar o grupo, ele envia para o tópico groups um objeto com a informação que o grupo foi apagado:

```
{
   "type": "delete",
   "id": "20211226150122",
   "leader": "foo_control",
}
```

# 5.0 Arquitetura

## 5.1 index.html

O index.html contém o front-end da aplicação e injeta os arquivos javascript.

## 5.1 bootstrap.js:

É injetado no index.html, responsável por iniciar a aplicação, chamando o app.js

## 5.2 app.js

É a classe controladora da aplicação,

- Lê e salva os usuários presentes no users.json
- Instancia objetos das classes auth.js, user.js e group.js
- Define callback para iniciar o fluxo do Capítulo 3.2 que ocorre após o login bem sucedido comentado no Capítulo 3.1
- Define callback para enviar públicas no tópico USERS caso saia da aplicação

## 5.3 auth.js

- Armazena o fluxo de login do Capítulo 3.2 suas interações com o index.html estão no arquivo user.template.js.
- Ao login ser bem sucedido, armazena id do usuário no localStorage da aplicação. Sempre que as outras classes precisarem do id do usuário logado, consultarão o método da auth.js que acessa ao localStorage.

## 5.4 socket.js

- Fornece função para criação de tópicos
- Fornece funções para escutar tópicos

## 5.5 users.js

- Funções para identificar o tipo de publicações pertinentes aos chats com um único usuário, informados no capítulo 4.0 e como reagir a cada tipo de publicação. Conforme explicado no capítulo 3.0 e capítulo 4.0
- As funções que envolvem interações com a tela foram implementadas no arquivo user.template.js. Como por exemplo, quando a solicitação de conversa chega a um

usuário, o user.js chama o método do user.template.js que faz a solicitação ser exibida na tela do usuário por meio de interações com o index.html

## 5.6 groups.js

- Funções para identificar o tipo de publicações pertinentes aos grupos, informados no capítulo 4.0 e como reagir a cada tipo de publicação. Conforme explicado no capítulo 3.0 e capítulo 4.0
- As funções que envolvem interações com a tela foram implementadas no arquivo group.template.js. Como por exemplo, quando o groups.js reconhece que um usuário deseja criar, requisitar entrada ou excluir um grupo, ele chama o groups.template.js que é responsável por fazer interações no html para exibir as ações feitas na tela dos usuários