# Universidade Federal do ABC
# MCTA025-13 - Sistemas Distribuídos - 2024.Q2

—

**Projeto 01 - MeuQoelhoMQ**

—

Emilio Francesquini
e.francesquini@ufabc.edu.br

17 de julho de 2024

## 1 O projeto

Neste projeto vamos implementar um serviço de mensageria[1], a exemplo do RabbitMQ, Kafka, ZeroMQ entre outros.

Nestes serviços há produtores de mensagens e consumidores. Os produtores podem publicar mensagens em canais específicos enquanto os consumidores podem assinar esses canais para receber as mensagens. O serviço de mensageria é responsável por distribuir essas mensagens entre todos os clientes conforme os seus interesses estabelecidos através de inscrições em canais. O seu servidor de mensagens deve ser capaz de atender **todos** os pontos abaixo:

- Toda a comunicação entre o servidor e os seus clientes (sejam produtores ou consumidores) deve ser feita utilizando gRPC sobre Protocol Buffers.
- Você é responsável por definir o seu próprio protocolo de comunicação (usando um ou mais arquivos `.proto`).
- O seu servidor deve ter suporte a:
  - Criação de canais/filas de mensagens identificadas por nome
  - Remoção de canais/filas de mensagens identificadas por nome
  - Listagem dos canais disponíveis incluindo o seu tipo e o número de mensagens pendentes de entrega

[1] Também conhecidos como *messaging broker*, *streaming broker*, *event streaming platform*, …

- Publicação de novas mensagens em uma fila especificada pelo seu nome. O conteúdo dessas mensagens pode ser um texto ou uma sequência de bytes de tamanho arbitrário
- Assinatura de canais para recebimento das mensagens publicadas.
- Salvamento das mensagens em disco para que, em caso de falha de energia por exemplo, o servidor seja capaz de retomar a execução com as mensagens das filas intactas.

Abaixo mais detalhes sobre a publicação e assinatura são dados. Em caso de dúvidas, use o Discord da disciplina para buscar esclarecimentos.

## 2 Funcionamento do servidor

**Canais** são filas independentes de mensagens administradas pelo servidor. Eles são identificados por um nome (texto) e são classificados como:

- **Simples**: quando uma mensagem é publicada em um canal simples, apenas um dos assinantes daquele canal será notificado e receberá uma cópia da mensagem. A escolha dos assinantes pode ser feita a seu critério e possíveis estratégias incluem aleatório, *round-robin*, LRU, etc.
- **Múltiplo**: Por outro lado, quando uma mensagem é publicada em um canal múltiplo, uma cópia de cada mensagem deve ser entregue para cada um dos assinantes daquele canal.

Em ambos os casos as mensagens que já tiverem sido entregues para os destinatários são descartadas pelo servidor.

Os canais devem ser criados com um nome (string) e com o seu tipo (simples/múltiplo). O tipo deve ser definido como uma `enum` no seu serviço dentro do `.proto`.

Atenção, o seu servidor deve ser capaz de lidar de maneira transparente com clientes que não estão mais disponíveis, por exemplo, sem que haja qualquer interrupção do serviço. Em outras palavras o seu servidor não pode explodir por mal comportamento (deliberado ou não) de um dos clientes. Como tratar a lista de clientes não mais disponíveis fica a seu cargo.

Cuidado também com concorrência por acesso às estruturas de dados compartilhadas. Lembrem-se que possivelmente milhares de clientes simultâneos podem estar acessando a mesma fila e isso causa concorrência por acesso.

O salvamento das mensagens em disco deve ser feito periodicamente e automaticamente pelo servidor. Pode-se fazer isso a cada nova mensagem recebida ou periodicamente (sabendo-se que, neste caso, algumas mensagens

podem ser perdidas em caso de falhas). Mensagens já entregues devem ser apagadas do disco imediatamente após a sua entrega.

### 2.1 Assinaturas

Para a implementação da assinatura, os clientes devem ter a opção de utilizar uma chamada gRPC no modo `stream` (Server streaming RPCs) enquanto aguardam por novas mensagens ou simplesmente fazer uma chamada independente para receber apenas uma das mensagens enfileiradas em um dado canal (Unary RPC). Caso não haja mensagens disponíveis, o servidor deve dar a opção de um timeout para o cliente não ficar esperando indefinidamente.

A publicação, de modo semelhante, deve permitir a publicação de uma única mensagem, ou uma lista (ambos como *unary RPC*, e não um *stream*) de mensagens em uma chamada única.

## 3 O projeto

O projeto tem como **prazo de entrega 04/08/2024**. A entrega deverá ser feita via GitHub Classroom. Não serão aceitas entregas por outros meios. O link para entrega será disponibilizado na página da disciplina.

### 3.1 Linguagem

Você pode implementar o seu código na sua linguagem de programação favorita, seja ela qual for. Contudo tenha em mente que:

- O ambiente de correção é uma máquina Linux, logo é importante que você se assegure que o seu código funciona neste ambiente ainda que você o desenvolva no Windows, MacOS, etc.
- Códigos que não compilam ou que não contiverem instruções para sua compilação receberão nota ZERO.
- Junto com o seu servidor você deverá entregar dois clientes que exemplifiquem o seu uso. Estes clientes podem ser equivalentes (i.e., fazerem as mesmas operações) contudo devem ser escritos em duas linguagens de programação diferentes (uma pode ser a mesma utilizada pelo servidor).

### 3.2 Equipe

O projeto pode ser feito em grupos de no máximo 3 pessoas

- Apenas um dos integrantes do grupo deve fazer o upload do projeto para o GitHub e deve estar bem claro quais são os integrantes da equipe no relatório.
- Todos os *commits* e o *push* final devem ter sido feitos até no máximo às **23h59 do dia 04/08/2024**.
  - Atenção, caso ocorram/commits/pushes após o prazo, será considerada apenas a última versão para avaliação. Caso o último esteja com atraso $> 3$ dias, então será considerado o último commit até 3 dias (valendo no máximo 5).
  - Entregas em atraso sofrerão descontos conforme a tabela abaixo.

| Dias em atraso | Nota máxima |
|---|---|
| 1 dia | 7 |
| 2 dias | 6 |
| 3 dias | 5 |
| >3 dias | 0 |

### 3.3 Entregáveis

Juntamente com o código para o seu servidor, você deverá entregar o código de dois clientes que mostrem o servidor em funcionamento. Estes dois clientes devem ser implementados em linguagens de programação distintas, contudo um deles pode utilizar a mesma linguagem do servidor.

No mesmo repositório do seu código, deverá haver um relatório de no máximo 1 páginas (em PDF! nada de `.md`, `.docx`, `.odt`, ou qualquer outro formato da moda…) descrevendo:

- O seu projeto
- Como utilizar o seu código
- Dificuldades, surpresas e destaques do seu código

O seu relatório também deverá conter **obrigatoriamente** um link para um vídeo que:

- Esteja abrigado no Youtube (ou qualquer serviço semelhante)
- Seja privado/não listado e disponível apenas àqueles com o link
- Tenha no máximo 3 minutos (ou seja, <= 180 segundos). **Sem exceções**.
- Mostre a compilação, o uso e uma apresentação do código do projeto
- Durante a correção o professor pode pedir por esclarecimentos adicionais.

### 3.4 Dúvidas

Todas as dúvidas devem ser sanadas diretamente com o professor através do Discord da disciplina.