Sistema de Gestão de Pedidos – POC TMB

Henry Lírio Rufino

Documentação Oficial do Projeto – TMB Orders

🎯 Objetivo do Desafio

Desenvolver um sistema simples de gestão de pedidos, onde será possível criar, listar e visualizar pedidos. Sempre que um pedido for criado, o sistema deve enviar uma mensagem para o Azure Service Bus, simulando um processamento assíncrono. Um worker será responsável por consumir as mensagens, processar o pedido e atualizar seu status.

1. Visão Geral do Projeto

O **TMB Orders** é um sistema completo de gerenciamento de pedidos com:

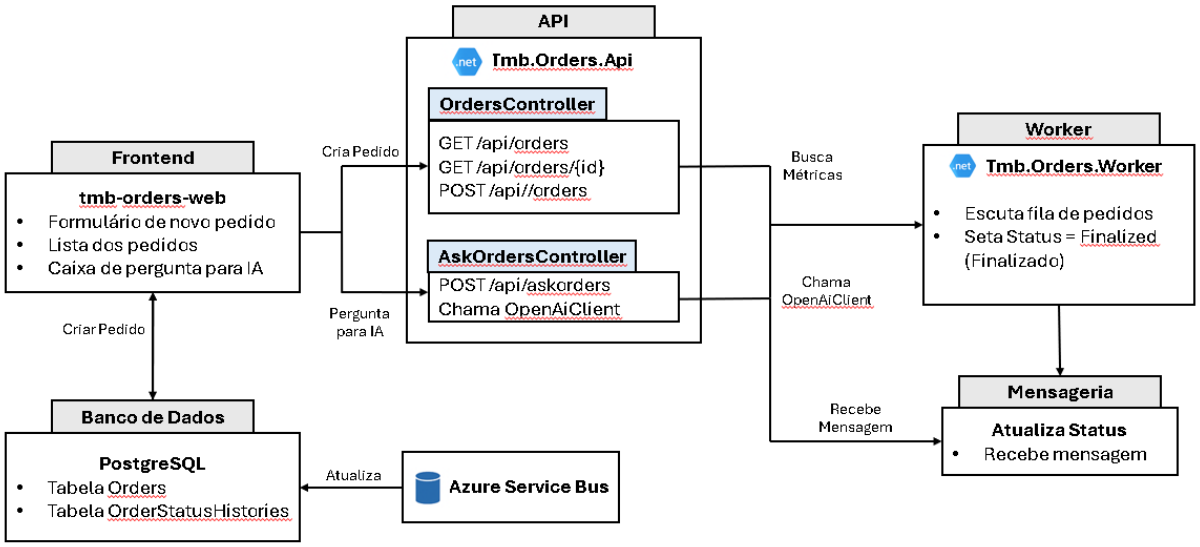
- Frontend em **React + Vite.**
- Backend em **.NET 8.**
- Banco **PostgreSQL.**
- Mensageria **Azure Service Bus.**
- Worker para processamento assíncrono.
- Módulo de IA usando **Groq (Api).**

Fluxo principal:

1. Usuário cria um pedido no frontend.
2. A API salva o pedido no banco e envia uma mensagem para o Service Bus.
3. O Worker recebe a mensagem e finaliza o pedido.
4. O Frontend atualiza automaticamente até o pedido estar finalizado.
5. O usuário pode consultar métricas via IA.

2. Arquitetura Geral

Siga o diagrama geral do projeto abaixo:



3. Componentes de Responsabilidades

3.1. Frontend – tmb-orders-web (React + Vite):

- SPA (Sigle Page Application) em React/TS.
- Consome exclusivamente a API:
  - GET / api/ orders
  - GET / api/ orders{id}
  - POST / api/ orders
  - POST / api/ askorders
- Auto-Refresh enquanto houver pedidos em “Processando”.
- UI composta por:
  - Formulário de novo pedido.
  - Lista de pedidos realizados.
  - Caixa de pergunta para IA.

3.2. API – Tmb.Orders.Api (.NET 8):

- **OrdersController**

Endpoint	Descrição
GET / api/ orders	Lista todos os pedidos
GET / api/ orders{id}	Retorna um pedido específico
POST / api/ askorders	Cria um novo pedido e envia mensagem para o Service Bus

- **AskOrdersController**

Endpoint	Descrição
POST / api/ askorders	Gera resposta inteligente baseada nas métricas do banco

- **OrderCreatedPublisher**

Envia mensagem para o Azure Service Bus usando:

  - CorrelationId = OrderId
  - EventType = “OrderCreated”
  - Payload {“OrderId”;”<guid>”}

- **OpenAiClient**
  - Conecta na API Groq compatível.
  - Usa env:
    - GROQ__ApiKey
    - GROQ__Model

- **Healthchecks**
  - /health testando:
    - Conexão DB
    - Fila (AzureServiceBusQueue)
  - Usado pelo Docker Compose.

3.3. Database – PostgreSQL

- Orders

Campo	Tipo	Descrição
Id	guid	Identificador
Cliente	text	Nome do cliente
Produto	text	Nome do produto
Valor	numeric	Valor do pedido
Status	int	Pending/Processing/Finalized
DataCriacao	timestamp	Data/Hora
LastProcessedMessageId	text	Idenpotência

- OrderStatusHistories

Campo	Tipo
Id	guid
OrderId	guid
Status	int
ChangedAt	timestamp

3.4. Azure Service Bus

- Uma única fila.
- Recebe mensagens geradas ao criar pedido.
- Todas com:
    - CorrelationId = OrderId
    - EventType = “OrderCreated”

3.5. Worker – Tmb.Orders.Worker

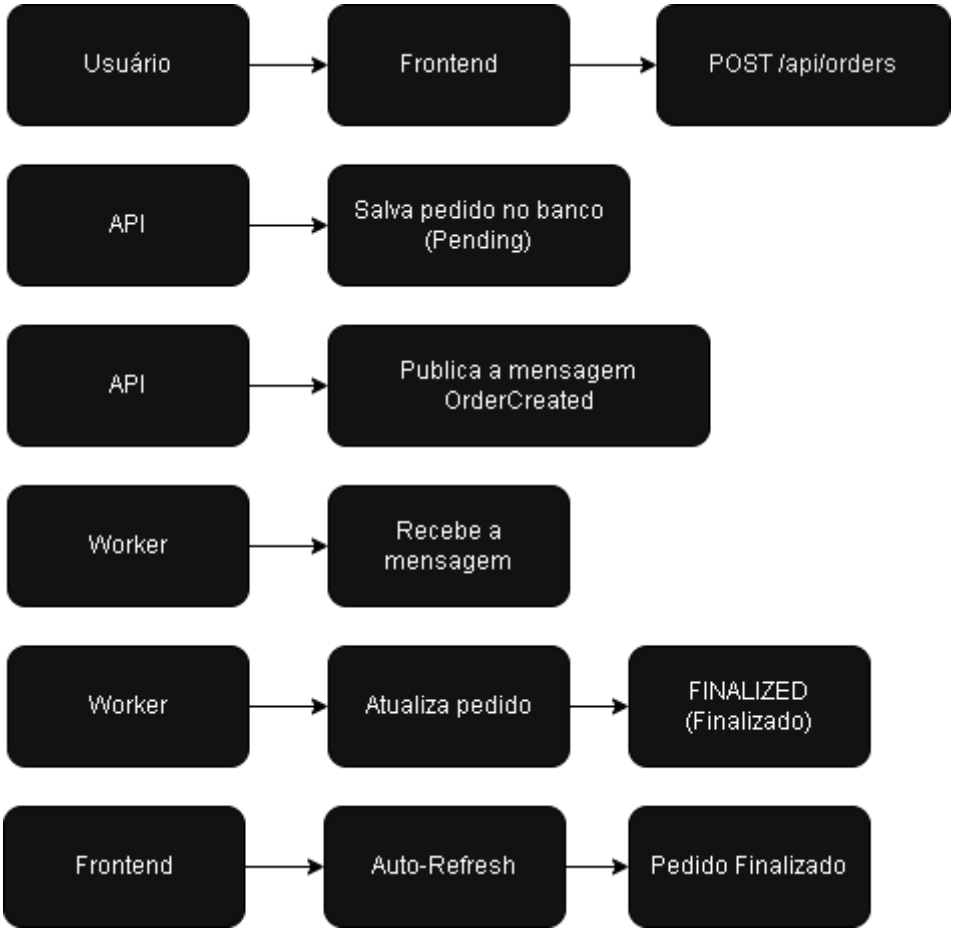
Processo assíncrono:

1. Recebe mensagem.
2. Lê OrdeId + MessageId.
3. Verifica idempotência:
    - Se LastProcessedMessageId == MessageId -> ignora.
4. Atualiza Status -> Finalized (Finalizado)
5. Insere OrderSatusHistory.
6. Atualiza LastProcessedMessageId.
7. Completa mensagem.

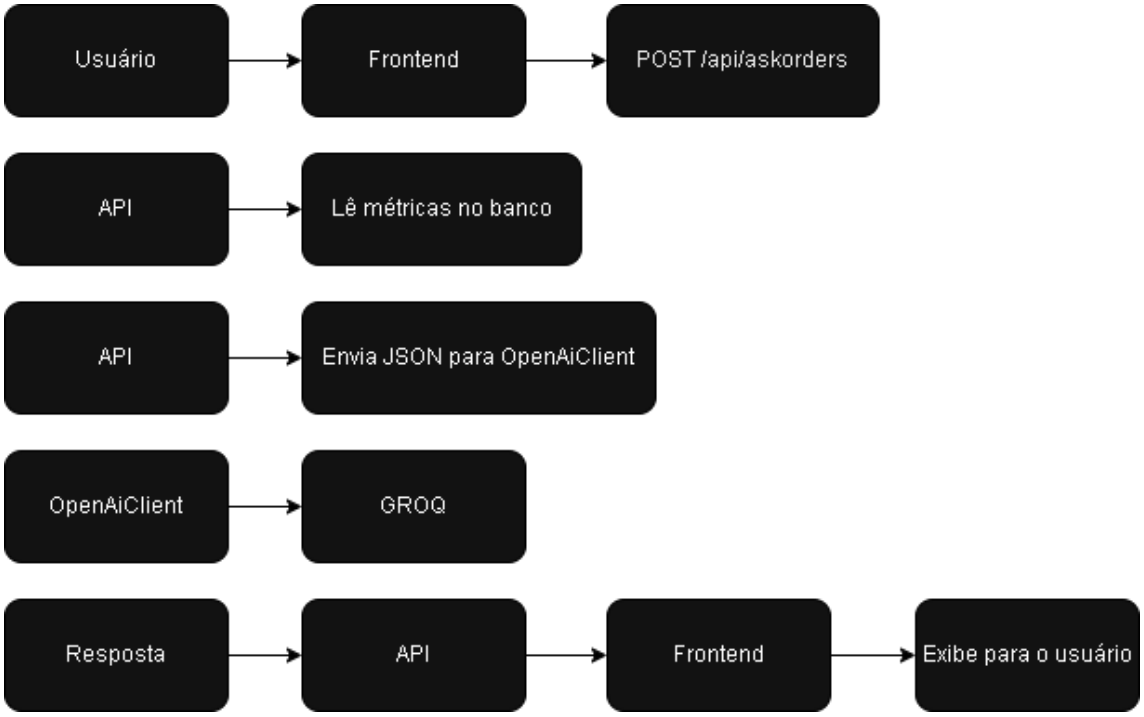
Simula processamento usando delay de 5s.

4. Fluxos Detalhados

4.1. Criar Pedido



4.2. Perguntar para IA



5.Variáveis de Ambiente

- **API**

```
ASPNETCORE_ENVIRONMENT=Development
ConnectionStrings__DefaultConnection=...
ServiceBus__ConnectionString=...
ServiceBus__QueueName=orders
GROQ__ApiKey=...
GROQ__Model=llama3-8b
FrontendUrl=http://localhost:3000
```

- **Worker**

```
ConnectionStrings__DefaultConnection=...
ServiceBus__ConnectionString=...
ServiceBus__QueueName=orders
```

- **Frontend**

```
VITE_API_URL=http://localhost:8080
```

6. Como Rodar

Via Docker Compose

```
docker compose up --build
```

Serviços disponíveis:

- **API -> http://localhost:8080**
- **Frontend -> http://localhost:3000**
- **pgAdmin -> http://localhost:8081**
- **PostgreSQL -> localhost:5432**