RELATÓRIO

Desafio Engenheiro de software

Processar pedidos

NÍCOLAS PAIUCA BUSCARINI

São Paulo, 2024

Sumário

1. Plano de Trabalho

Atividades

Configuração Inicial do Ambiente

- **Task:** Configurar o ambiente de desenvolvimento com Docker para incluir RabbitMQ, MySQL, e a aplicação .NET.
- **Estimativa:** 4 horas

Desenvolvimento de Arquitetura

- **Task:** Definir a arquitetura do sistema, incluindo a criação de diagramas de arquitetura, modelagem do banco de dados, diagrama de implantação e infra.
- **Estimativa:** 6 horas

Modelagem do Banco de Dados

- **Task:** Modelar e implementar a base de dados em MySQL utilizando Entity Framework.
- **Estimativa:** 4 horas

Implementação do Microserviço

- **Task:** Criar o microserviço em .NET que consome dados de uma fila RabbitMQ e grava no banco de dados.
- **Estimativa:** 8 horas

Implementação da API REST

- **Task:** Criar uma API REST para consultar as informações solicitadas: valor total do pedido, quantidade de pedidos por cliente, e lista de pedidos realizados por cliente.
- **Estimativa:** 6 horas

Testes e Documentação

- **Task:** Testar a aplicação, criar o relatório técnico, incluindo os diagramas (arquitetura, banco de dados, implantação) e publicar no GitHub.
- **Estimativa:** 8 horas

Buffer e Revisões

- **Task:** Revisar o trabalho, ajustar eventuais problemas, e garantir que a aplicação esteja rodando conforme esperado.
- **Estimativa:** 4 horas

Total Estimado: 40 horas

Execução

O trabalho foi executado conforme o planejado, sem desvios significativos em relação às estimativas iniciais. O processo ocorreu de forma eficiente devido à familiaridade com as tecnologias e ferramentas envolvidas, como .NET, Docker, RabbitMQ e MySQL. Já estou acostumado a configurar ambientes semelhantes e a desenvolver microserviços e APIs REST em arquiteturas baseadas em containers, o que contribuiu para que todas as etapas fossem concluídas dentro do prazo esperado.

Essa experiência prévia garantiu que o fluxo de trabalho fosse suave e que não houvesse complicações técnicas inesperadas, resultando em uma execução precisa, com todos os serviços interagindo conforme esperado.

2. Tecnologias Utilizadas

O desenvolvimento deste projeto foi realizado utilizando a linguagem .NET 8, uma plataforma robusta e de alto desempenho para o desenvolvimento de APIs e microserviços. A IDE escolhida foi o Visual Studio 2022, que oferece ferramentas avançadas de depuração, suporte a containers e integração com bibliotecas de terceiros, facilitando o desenvolvimento e o gerenciamento do projeto.

Para gerenciamento de containers, utilizou-se o Docker, uma tecnologia amplamente reconhecida por sua capacidade de criar ambientes de desenvolvimento isolados e consistentes. No projeto, o Docker foi utilizado para rodar o banco de dados MySQL e o sistema de mensagens RabbitMQ, garantindo fácil replicação do ambiente de desenvolvimento. A orquestração dos serviços dentro do ambiente Docker foi feita com Docker Compose, permitindo configurar e iniciar múltiplos containers com apenas um comando.

O sistema operacional utilizado foi o Windows 10, configurado com suporte completo para containers Docker, oferecendo um ambiente de desenvolvimento estável e otimizado para a execução dos serviços necessários ao projeto.

Em termos de bibliotecas, utilizamos a RabbitMQ.Client para lidar com a comunicação com as filas do RabbitMQ. Esta biblioteca fornece uma interface eficiente para o envio e recebimento de mensagens entre os componentes do sistema.

Para a persistência dos dados no MySQL, foi utilizado o Entity Framework Core, uma poderosa ferramenta ORM (Mapeamento Objeto-Relacional) que simplifica as operações de banco de dados. Mais especificamente, a biblioteca Pomelo.EntityFrameworkCore.MySql foi escolhida para suportar a conexão e as operações com o banco de dados MySQL, garantindo uma integração estável e eficiente.

Na API, também utilizamos o Swashbuckle.AspNetCore para gerar a documentação Swagger, facilitando o entendimento e a interação com a API por parte de outros desenvolvedores. A biblioteca Newtonsoft.Json foi utilizada para lidar com a serialização e desserialização de dados em formato JSON, sendo especialmente útil para manipulação de mensagens e conversão de objetos complexos.

Além disso, utilizamos o Microsoft.Extensions.Configuration.Abstractions, uma biblioteca essencial para a configuração do aplicativo, permitindo a integração de configurações externas, como variáveis de ambiente e arquivos de configuração, para um gerenciamento centralizado das conexões e credenciais.

Essas tecnologias e bibliotecas foram escolhidas de maneira estratégica, garantindo que a aplicação fosse desenvolvida de forma modular, eficiente e pronta para a escalabilidade e manutenção.

3. Diagrama de Arquitetura

A arquitetura do sistema foi projetada para garantir a escalabilidade, a modularidade e o desempenho no processamento de pedidos. Ela é composta por cinco principais componentes que trabalham em conjunto para o correto funcionamento da aplicação:

Banco de Dados MySQL

O banco de dados MySQL é utilizado para armazenar as informações dos pedidos, como dados de clientes, itens do pedido e status. A escolha do MySQL foi motivada por sua robustez, alta performance e ampla compatibilidade com o Entity Framework, permitindo uma fácil integração com a aplicação e fornecendo suporte para consultas complexas e eficientes.

RabbitMQ (Serviço de Mensageria)

O RabbitMQ atua como o serviço de mensageria responsável pelo gerenciamento de filas de pedidos. Ele garante que as mensagens, que representam os pedidos realizados, sejam enviadas e consumidas de forma assíncrona, permitindo que a aplicação processe grandes volumes de pedidos sem sobrecarregar os sistemas. Além disso, o RabbitMQ possibilita o desacoplamento entre os produtores de mensagens (como um serviço de pedidos) e os consumidores, melhorando a flexibilidade da arquitetura.

Camada de Persistência

A camada de persistência é responsável por interagir diretamente com o banco de dados MySQL, utilizando o Entity Framework Core para realizar operações CRUD (Create, Read, Update, Delete) nos dados. Essa camada se encarrega de salvar as informações dos pedidos consumidos das filas do RabbitMQ e garantir a integridade dos dados. Além disso, ela oferece abstração do banco de dados, facilitando a migração ou modificação do banco de dados no futuro, se necessário.

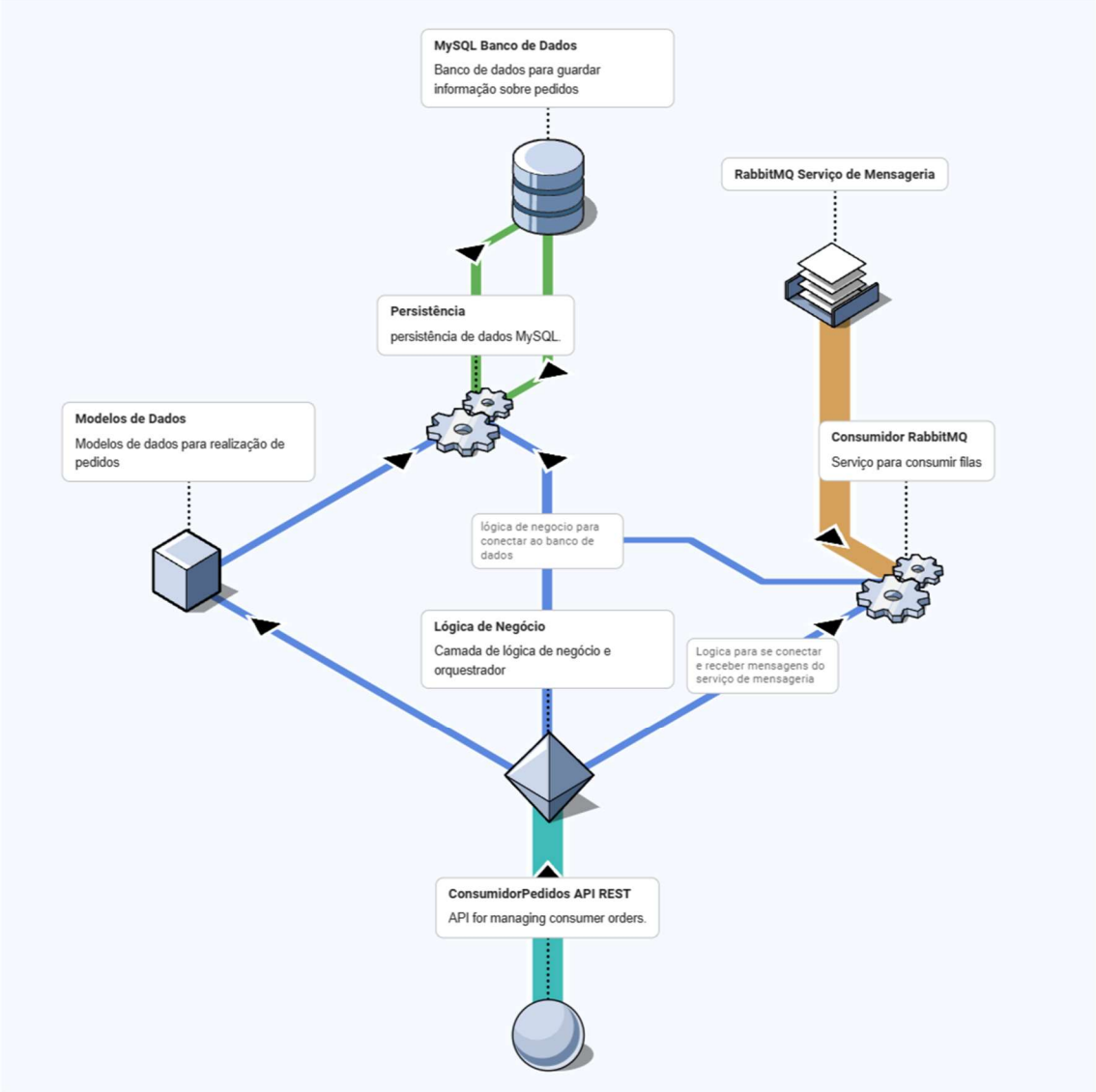
Camada de Lógica de Negócio

Esta é a camada central do sistema, onde ocorre o processamento dos pedidos e a implementação das regras de negócio. A camada de lógica de negócio orquestra a interação entre a camada de persistência e o Consumidor RabbitMQ. Ela garante que as mensagens de pedidos recebidas pelas filas sejam processadas corretamente, aplicando validações e cálculos necessários, e repassa os dados à camada de persistência para que sejam gravados no banco de dados. Essa camada também pode incluir a lógica de envio de mensagens para outras filas, caso seja necessário comunicar a outros serviços.

API REST (ConsumidorPedidos)

A API REST, exposta pelo microserviço ConsumidorPedidos, é responsável por permitir a interação externa com o sistema. Através de endpoints REST, outros serviços ou sistemas podem consultar informações sobre os pedidos, como o valor total de um pedido, a quantidade de pedidos por cliente e a lista de pedidos realizados.

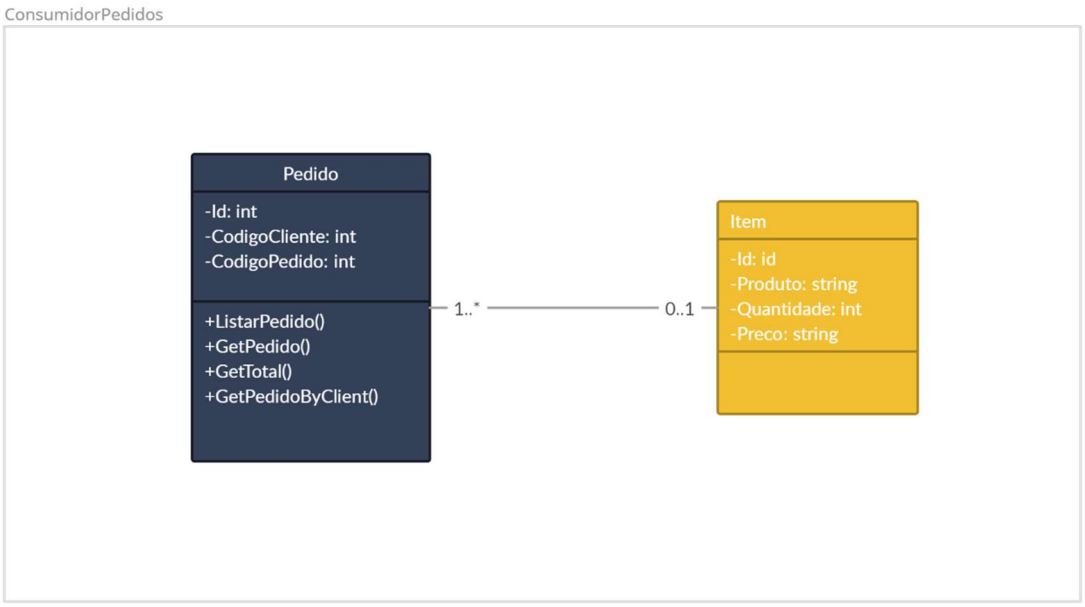
Esta API facilita a comunicação entre o sistema de pedidos e outros módulos ou aplicações, garantindo que os dados estejam disponíveis em tempo real para consultas.



O uso de Docker para gerenciar o ambiente de execução, com containers para o banco de dados e o serviço de mensageria, permite que a aplicação seja facilmente escalável e configurada em diferentes ambientes de produção e desenvolvimento.

4. Modelagem da Base de Dados

A modelagem do banco de dados do **ConsumidorPedidos** foi projetada com uma abordagem relacional e modular, contemplando duas principais entidades: **Pedido** e **Item**. Essa modelagem visa suportar o fluxo de informações de pedidos realizados por clientes, mantendo a consistência e a flexibilidade necessárias para futuras expansões. A seguir, descrevemos a estrutura de cada entidade e seus relacionamentos:



1. Entidade Pedido

A entidade **Pedido** é responsável por armazenar as informações referentes aos pedidos realizados por clientes. Seus principais atributos são:

- **Id (int)**: Identificador único do pedido.
- **CodigoCliente (int)**: Código que identifica o cliente que realizou o pedido.
- **CodigoPedido (int)**: Código associado ao pedido em questão.

A entidade **Pedido** também contém quatro métodos principais:

- **ListarPedido()**: Retorna a lista de todos os pedidos registrados.
- **GetPedido()**: Recupera os detalhes de um pedido específico.
- **GetTotal()**: Calcula e retorna o valor total do pedido.
- **GetPedidoByClient()**: Filtra e recupera pedidos de um determinado cliente.

2. Entidade Item

A entidade **Item** armazena os detalhes de cada item pertencente a um pedido. Seus principais atributos são:

- **Id (int)**: Identificador único do item.
- **Produto (string)**: Nome do produto.
- **Quantidade (int)**: Quantidade do produto no pedido.
- **Preco (string)**: Preço unitário do produto.

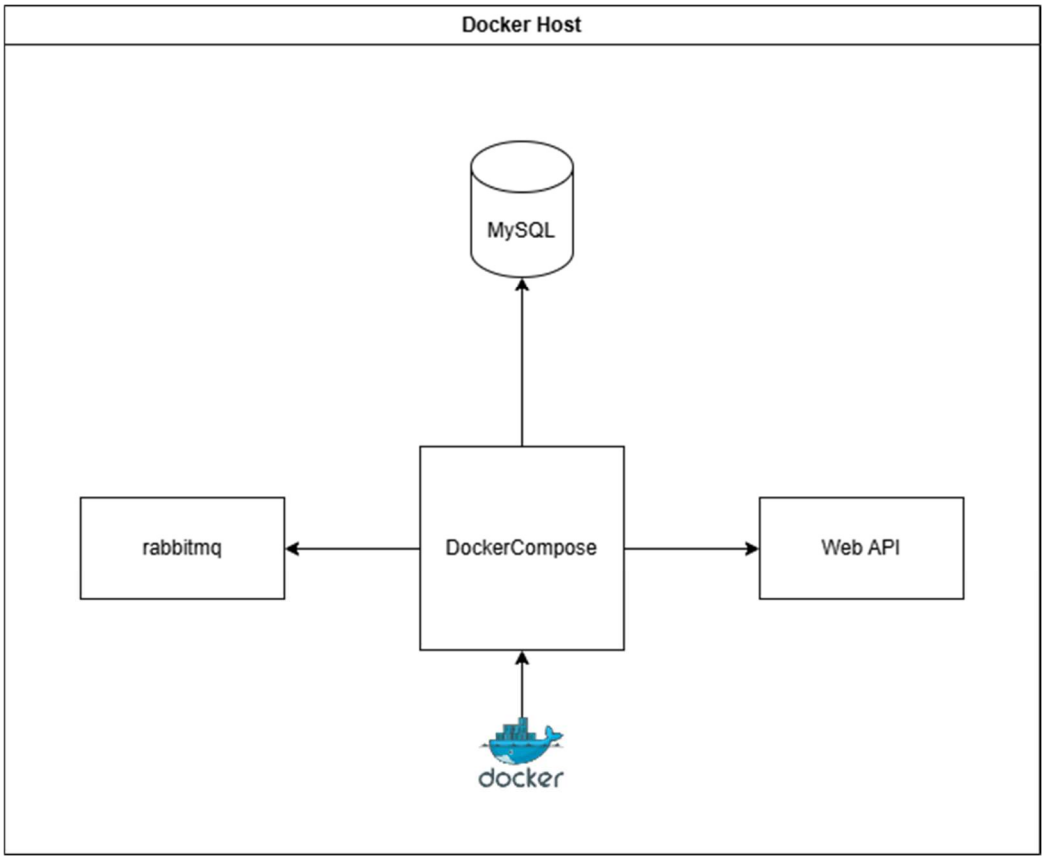
Relacionamento

Há um relacionamento de **1 para muitos** entre a entidade **Pedido** e **Item**. Ou seja, um **Pedido** pode conter vários **Itens**, mas um **Item** pode pertencer a, no máximo, um

Pedido. Este relacionamento garante a integridade referencial entre os pedidos e os itens, possibilitando consultas e manipulações eficientes de dados entre essas tabelas.

5. Diagrama de Implantação

A seguir, vou descrever a estrutura básica do diagrama e como ele poderia ser montado:



Descrição do Diagrama de Implantação

- **Docker Compose**: A ferramenta que orquestra e gerencia os serviços.
- **Contêiner MySQL**:
  - Responsável pelo armazenamento de dados do sistema.
  - Utiliza volumes para persistência de dados.
  - Porta: 3306 (expondo para a porta 3307).
- **Contêiner RabbitMQ**:
  - Responsável pelo sistema de mensageria, que permite a comunicação assíncrona entre os serviços.
  - Pode utilizar o dashboard do RabbitMQ para monitoramento (acessível via porta 15672).
  - Porta: 5672 para mensagens e 15672 para interface de administração.
- **Contêiner Web API (.NET 8)**:

- Contém a aplicação desenvolvida em .NET 8.
- Esta API consome os serviços de MySQL e RabbitMQ.
- Porta: 8080 (Expondo 5005 para evitar portas comumente usadas)

6. Diagrama de Infraestrutura

O projeto ConsumidorPedidos foi desenvolvido localmente com o objetivo de ser um desafio técnico, sem a intenção inicial de ser integrado a ambientes de produção na nuvem. Toda a infraestrutura foi montada em uma máquina local com as seguintes especificações:

Embora o projeto tenha sido idealizado apenas como um exercício técnico, sem a previsão de integração com a nuvem, sua implementação foi feita de forma que a migração para um ambiente de cloud seria extremamente simples. Graças à utilização do Docker para a orquestração dos serviços, o processo de conteinerização dos componentes permite que todo o ambiente seja facilmente replicado em qualquer provedor de nuvem (como AWS, Azure ou Google Cloud).

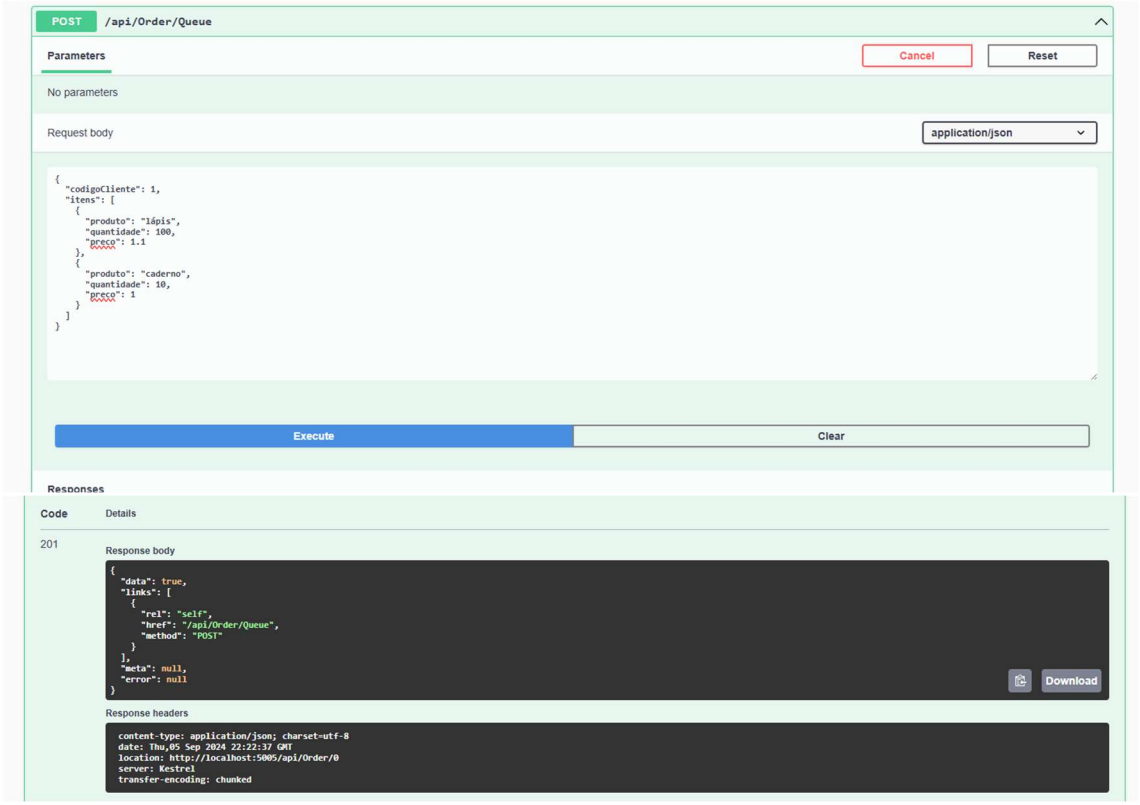
Com a infraestrutura já separada em contêineres, a transição para a nuvem se resumiria a pequenos ajustes na configuração de rede e autenticação, além da escolha de um serviço de orquestração, como Kubernetes, para gerenciar os contêineres. Isso torna o projeto versátil e pronto para ser escalado em ambientes maiores, caso necessário.

7. Evidência de Testes Funcionais

Testes funcionais executados para:

1. Subir os Serviços com Docker Compose

2. Enfileirar Pedido
Para maior abstração, foi decidido criar um endpoint na API para enfileirar no RabbitMQ apenas demonstrativo, pois a aplicação deve apenas consumir a fila.



Logs:

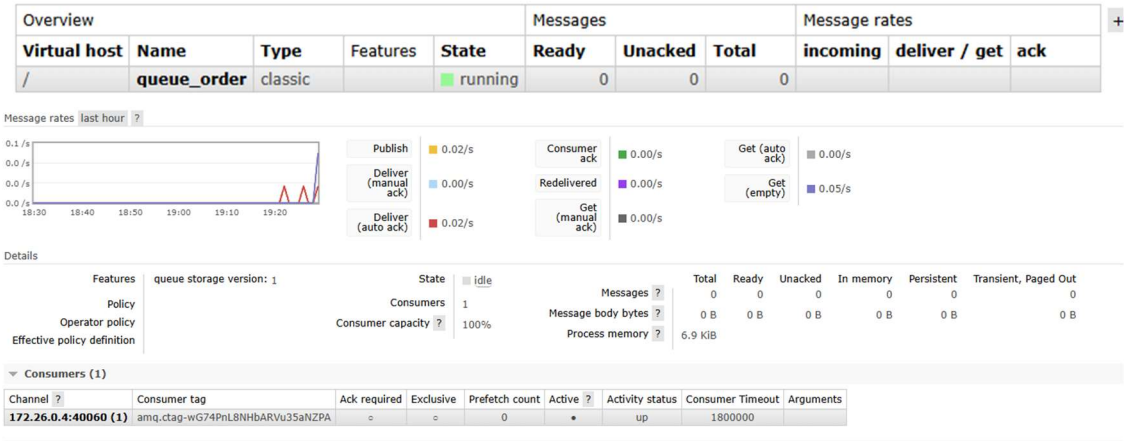
```
2024-09-05 19:25:51       Queueing a new order
2024-09-05 19:25:51  [x] Sent '{"Id":0,"ClientCode":1,"Items":[{"Id":0,"Product":"lápis","Quantity":100,"Price":1.1},{"Id":0,"Product":"caderno","Quantity":10,"Price":1.0}],"Total":120.0}' to queue 'queue_order'
2024-09-05 19:25:51 info: ConsumidorPedidos.Controllers.OrderController[0]
2024-09-05 19:25:51       Order successfully queued
```

2. Consumo de mensagens do RabbitMQ e persistência no MySQL.

Logs:

```
2024-09-05 19:25:51       [x] Received '{"Id":0,"ClientCode":1,"Items":[{"Id":0,"Product":"lápis","Quantity":100,"Price":1.1},{"Id":0,"Product":"caderno","Quantity":10,"Price":1.0}],"Total":120.0}' from queue 'queue_order'
2024-09-05 19:25:51 info: ConsumidorPedidos.Core.Consumer.MessageConsumer[0]
2024-09-05 19:25:51       [x] Processing message: {"Id":0,"ClientCode":1,"Items":[{"Id":0,"Product":"lápis","Quantity":100,"Price":1.1},{"Id":0,"Product":"caderno","Quantity":10,"Price":1.0}],"Total":120.0}
2024-09-05 19:25:51 info: ConsumidorPedidos.Core.Service.OrderService[0]
2024-09-05 19:25:51       Attempting to create a new order with ClientCode: 1
2024-09-05 19:25:51 info: ConsumidorPedidos.Core.Repository.OrderRepository[0]
2024-09-05 19:25:51       Creating a new entity.
```

Filas:



Consultas para verificar banco de dados:

```
1 •  SELECT * FROM root.Order where Id = 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Id	ClientCode
1	1
NULL	NULL

Query 1 | Order | Item

Limit to 1000 rows

```
1 •  SELECT * FROM root.Item where OrderId = 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Id	Product	Quantity	Price	OrderId
1	lápis	100	1.1	1
2	caderno	10	1	1
NULL	NULL	NULL	NULL	NULL

3. Listagem de pedidos via API REST com filtros e paginação.

**curl -X 'GET' **

**'http://localhost:5005/api/Order?pageNumber=1&pageSize=10' **

-H 'accept: text/plain'

response.json

note: neste response conseguimos identificar o número de pedidos e o total gasto em cada pedido

8. Publicação dos Códigos

GitHub: NicolasBuscarini/consumidor-pedidos: ConsumidorPedidos é uma aplicação .NET que processa pedidos, consumindo mensagens de uma fila RabbitMQ, armazenando os dados em um banco de dados MySQL e expondo uma API REST para consulta de informações sobre pedidos. (github.com)

IP: http://168.138.148.110:5005/swagger rodando em uma VM da Oracle

Board (com histórico de atividades): Backlog · Plano de Trabalho - Consumidor de Pedidos (github.com)