Relatório Técnico - Desafio BTG

Autor: Reinaldo Tenório dos Santos

Projeto: Processamento de Pedidos com Microsserviço + API REST

Repositórios:  
<https://github.com/reinaldots>  
<https://hub.docker.com/r/reinaldots/desafiobtg-api>  
<https://hub.docker.cddom/r/reinaldots/desafiobtg-ms>

# 1. Estrutura da Solução

A solução é composta por dois serviços principais:  
  
1. Microsserviço (DesafioBTG.MS)  
 - Consome pedidos da fila RabbitMQ e armazena no MongoDB.  
 - Executado via console.  
  
2. API REST (DesafioBTG.API)  
 - Expõe endpoints para consulta de pedidos.  
 - Se comunica com MongoDB para leitura.  
 - Acessível via Swagger UI.

# 2. Tecnologias e Ferramentas

| Tecnologia | Finalidade |
| --- | --- |
| Visual Studio 2022 / .NET C# 8.0 | Desenvolvimento da API e microsserviço |
| RabbitMQ | Comunicação assíncrona via fila |
| MongoDB | Persistência dos dados dos pedidos |
| Docker | Container de serviços |
| Docker Compose | Orquestração local |
| Swagger | Documentação da API |
| Git + GitHub | Versionamento e hospedagem do código |

# 3. Funcionalidades Implementadas

- Microsserviço consome mensagens da fila 'pedidos' no RabbitMQ.  
- Dados são salvos corretamente no MongoDB.  
- API REST com os seguintes endpoints:  
 - GET /api  
 - POST /api/pedidos  
 - GET /api/pedidos/{codigo\_cliente}  
 - GET /api/pedidos/{codigo\_cliente}/quantidade  
 - GET /api/pedidos/{codigo\_pedido}/valortotal  
- Validações básicas e mensagens de erro amigáveis.  
- Testes funcionais documentados.

# 4. Validação e Testes

- Testes manuais realizados com:  
 - Swagger UI  
 - RabbitMQ (mensagens)  
 - MongoDB Compass  
- Testes negativos incluídos (ex: JSON inválido, cliente inexistente)  
- Plano de testes documentado em 'TestesFuncionais.docx'

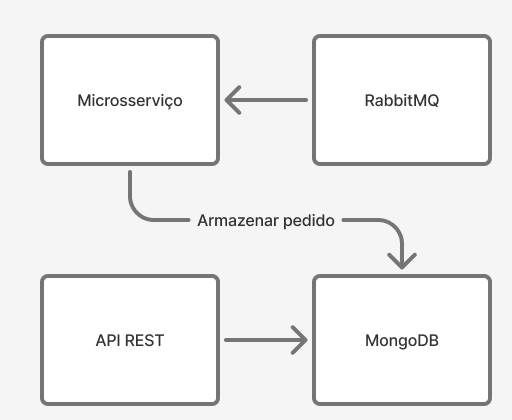
# 5. Organização do Repositório

| Pasta / Arquivo | Conteúdo |
| --- | --- |
| solution/ | Código-fonte da API e microsserviço |
| docker-compose.yml | Configuração dos serviços |
| README.md | Documentação geral |
| PLANO\_DE\_TRABALHO.md | Plano de trabalho com as atividades estimadas |
| TestesFuncionais.docx | Plano de testes |
| RelatorioTecnico.docx | Relatório técnico (este documento) |

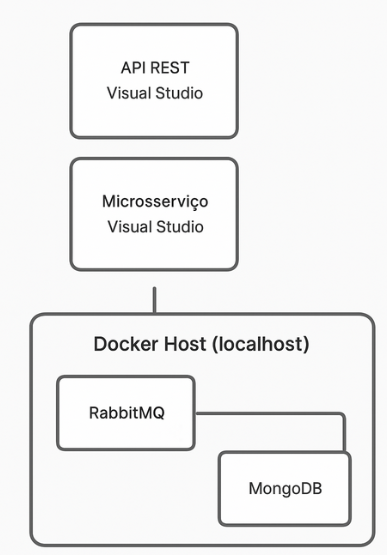
# 6. Diagramas da Solução

## 6.1 Diagrama de Arquitetura

Representa a comunicação entre API, RabbitMQ, Microsserviço e MongoDB.



## 6.2 Diagrama de Implantação



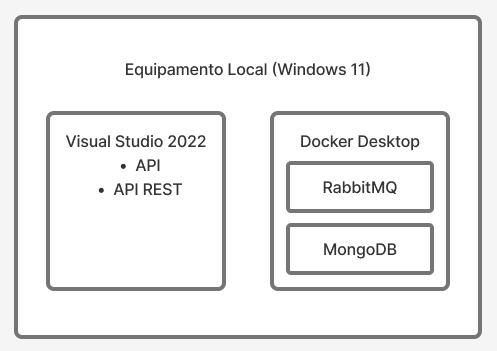
## 6.3 Modelagem da Base de Dados

Modelo de documento MongoDB armazenado no collection de pedidos.  
{  
 "codigoPedido": 1001,  
 "codigoCliente": 1,  
 "itens": [  
 { "produto": "lápis", "quantidade": 100, "preco": 1.10 },  
 { "produto": "caderno", "quantidade": 10, "preco": 1.00 }  
 ]  
}

| **Campo** | **Tipo de Dado** | **Obrigatório?** | **Descrição** |
| --- | --- | --- | --- |
| \_id | ObjectId | Sim (padrão) | Identificador único do documento, gerado automaticamente pelo MongoDB. |
| codigoPedido | Number | Sim | Identificador do pedido. |
| codigoCliente | Number | Sim | Identificador do cliente que realizou o pedido. |
| itens | Array of Documents | Sim | Lista contendo os itens que compõem o pedido. |
| ↳ produto | String | Sim | Nome do produto. |
| ↳ quantidade | Number | Sim | Quantidade do produto. |
| ↳ preco | Number | Sim | Preço unitário do produto. |

## 6.4 Diagrama de Infraestrutura

Infraestrutura local com Docker, sem dependências externas.



# 7. Plano de Trabalho (previsto / realizado)

O plano de trabalho foi seguido conforme o previsto, sem desvios relevantes. A divisão inicial das tarefas entre estruturação da solução, desenvolvimento dos componentes (API e microsserviço), testes funcionais e documentação mostrou-se adequada ao escopo do desafio.

Esse resultado foi possível pelos seguintes motivos:

* Clareza nos requisitos e objetivos do desafio desde o início.
* Organização antecipada da estrutura do projeto, o que facilitou a divisão lógica das etapas.
* Implementação de componentes simples, objetivos e alinhados diretamente ao escopo do desafio.
* Utilização de ferramentas como Docker, Postman e Swagger que aceleraram a criação do ambiente e os testes.

# Considerações Finais

O projeto foi desenvolvido com foco em clareza, organização e uso de boas práticas. A arquitetura simula um cenário moderno de microsserviços e mensageria. Toda a solução está documentada e validada com testes funcionais.