Projeto: Sistema de Gerenciamento de Pedidos Integrado com Spring e Microsserviços- Pós Tech - 4ª Fase - Grupo 17

Equipe:

Alfredo Hipólito - 359702

Fabio Batista – 358686

Ricardo Amaro – 359608

Victor André Salles – 360257

Vinicius Souza – 358319

1. Introdução

1.1 Objetivo do Projeto

Necessidade de um sistema de gerenciamento de pedidos altamente eficiente, que explore profundamente a arquitetura de microsserviços utilizando o ecossistema Spring. Este sistema deverá abranger desde a gestão de clientes e produtos até o processamento e entrega de pedidos, enfatizando a autonomia dos serviços, comunicação eficaz e persistência de dados isolada.

O objetivo é criar um sistema modular, onde cada microsserviço desempenha um papel no gerenciamento de pedidos. Este sistema não apenas facilitará a gestão eficiente de pedidos, mas também servirá como um exemplo prático do uso de tecnologias de ponta em um cenário realista de desenvolvimento de software.

2. Arquitetura do Projeto

2.1 Descrição Geral da Arquitetura

A arquitetura do projeto segue o padrão de microsserviços em Spring e Quarkus, com cada serviço representando uma responsabilidade no contexto de gestão dos pedidos. Essa separação facilita o desenvolvimento desacoplado, a escalabilidade e a manutenção contínua.

Os serviços são:

- **Clientes**: Centraliza a criação, alteração, consulta e exclusão de clientes, desenvolvido em Quarkus.
- **Produtos**: Centraliza a criação, alteração, consulta e exclusão de produtos
- **Pedidos**: Centraliza a criação, alteração, consulta e exclusão de pedidos, colocando-os em fila para posterior processamento no pedido receiver.
- **Estoque**: Centraliza a criação, alteração, consulta e exclusão de estoque
- **Pagamentos**: Centraliza a criação, alteração, consulta e exclusão de pagamentos, utiliza um modulo “mock” para retorno de informações.
- **Pedido Receiver**: Responsável por receber a solicitação do pedido e processa-la.

Todos os serviços são desenvolvidos com Spring Boot (exceto Cliente que está em Quarkus) e empacotados em containers Docker.

2.2 Tecnologias Utilizadas

Spring Boot, Quarkus, JUnit, Jacoco, Sonar, Docker, Docker Compose, Lombok, Maven.

2.3 Organização em Camadas

Controller / Resolver, Service, Repository, Entity/Model, DTOs.

2.4 Docker

Utilizar o docker no projeto trouxe ganho de produtividade, visto que criamos a estrutura somente uma vez e foi utilizada por todos do time de maneira simples e rápida durante o desenvolvimento. Criamos um Dockerfile para montar a imagem da aplicação e utilizamos o Docker Compose para orquestrar o contêiner da aplicação e o banco de dados.

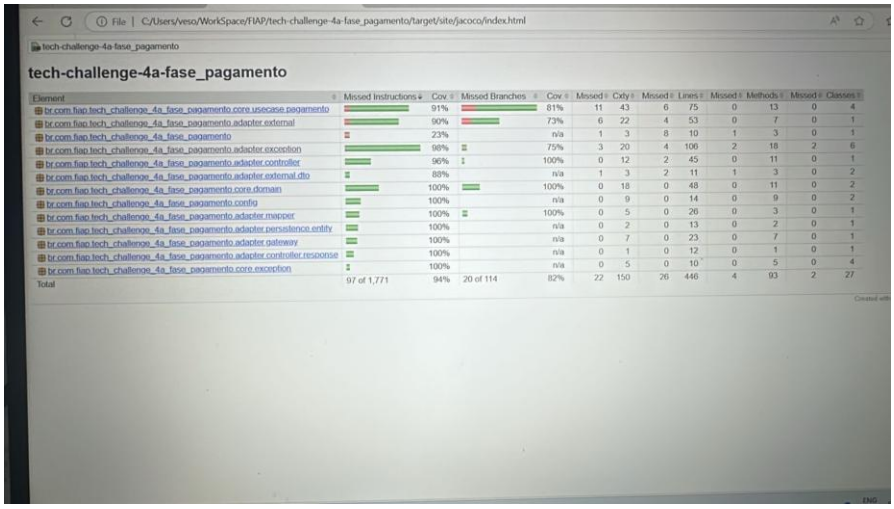
2.5 Banco de dados (MySql)

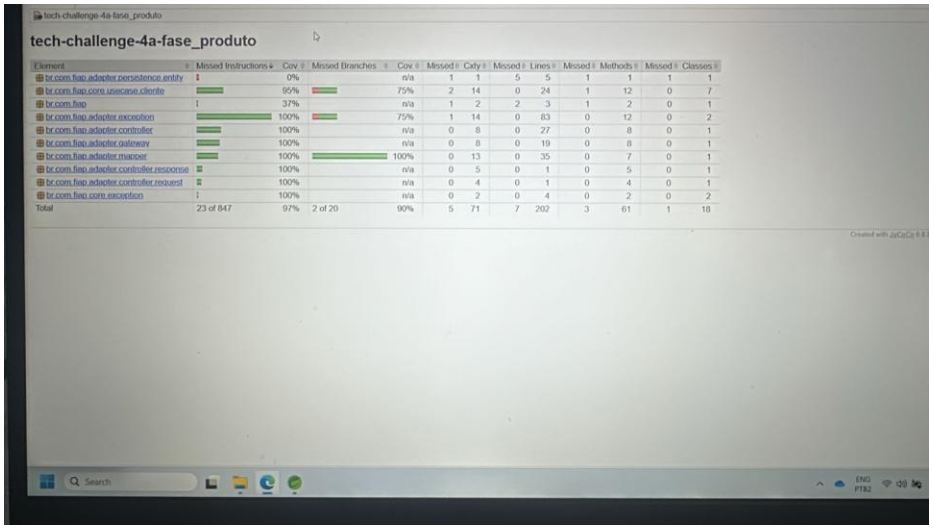
Visando o desenvolvimento profissional da equipe, optamos pelo MySQL como banco de dados por ser amplamente conhecido e utilizado pela equipe, o que facilitou o desenvolvimento e a manutenção do projeto. Além disso, o MySQL é uma opção confiável para sistemas que requerem alto desempenho em operações de leitura e escrita, possui vasta documentação e suporte da comunidade, e é facilmente integrado com as ferramentas e tecnologias utilizadas no projeto.

Essa estruturação, combinada com o uso de containers e um banco de dados robusto, garantiu que o projeto fosse desenvolvido de forma eficiente e escalável, alinhado às melhores práticas de desenvolvimento.

3. Relatorios Jacoco

Utilizado o Jacoco para cobertura de código.

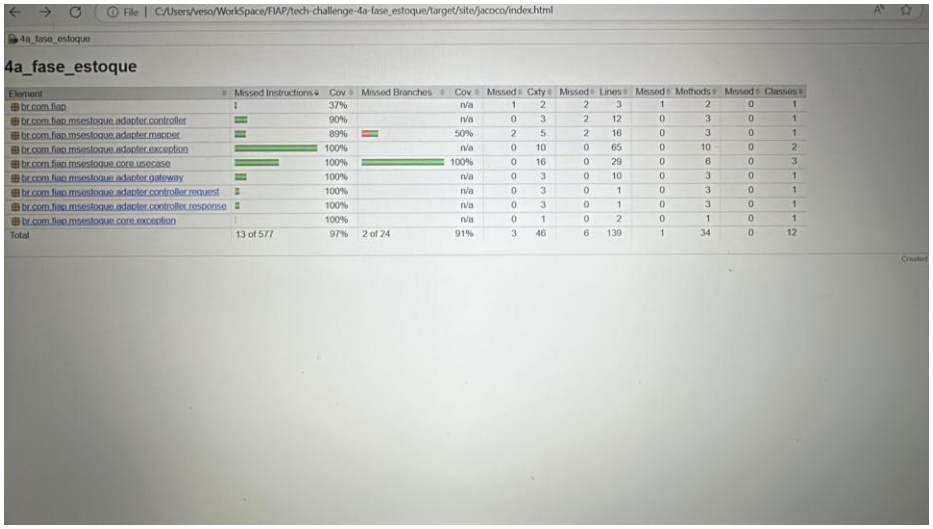




tech-challenge-4a-fase_produto

tech-challenge-4a-fase_produto

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
br.com.fiap.adapter.persistence.entity		0%		n/a	1	1	5	5	1	1	1	1
br.com.fiap.core.usecase.cliente		95%		75%	2	14	0	24	1	12	0	7
br.com.fiap		37%		n/a	1	2	2	3	1	2	0	1
br.com.fiap.adapter.exception		100%		75%	1	14	0	83	0	12	0	2
br.com.fiap.adapter.controller		100%		n/a	0	8	0	27	0	8	0	1
br.com.fiap.adapter.gateway		100%		n/a	0	8	0	19	0	8	0	1
br.com.fiap.adapter.mapper		100%		100%	0	13	0	35	0	7	0	1
br.com.fiap.adapter.controller.response		100%		n/a	0	5	0	1	0	5	0	1
br.com.fiap.adapter.controller.request		100%		n/a	0	4	0	1	0	4	0	1
br.com.fiap.core.exception		100%		n/a	0	2	0	4	0	2	0	2
Total	23 of 847	97%	2 of 20	90%	5	71	7	202	3	61	1	18

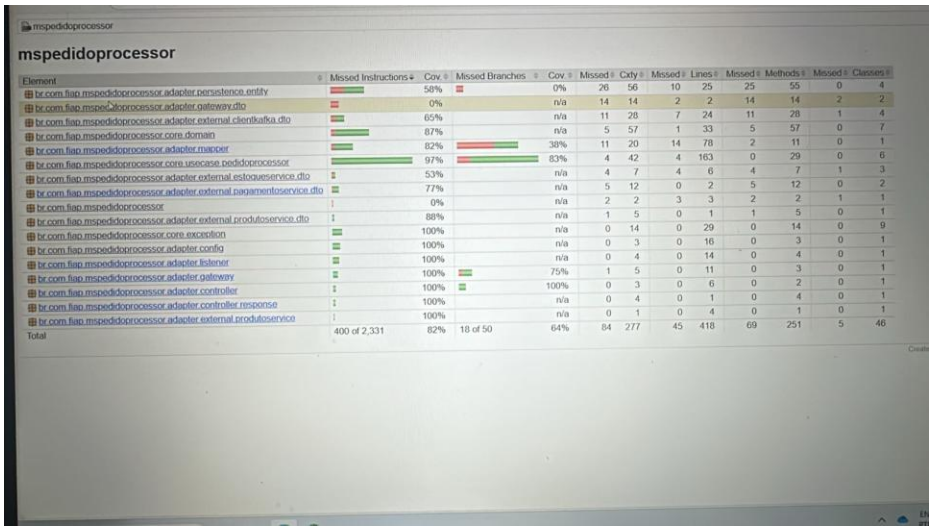


File | C:/Users/veso/WorkSpace/FIAP/tech-challenge-4a-fase_estoque/target/site/jacoco/index.html

4a_fase_estoque

4a_fase_estoque

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
br.com.fiap		37%		n/a	1	2	2	3	1	2	0	1
br.com.fiap.msestoque.adapter.controller		90%		n/a	0	3	2	12	0	3	0	1
br.com.fiap.msestoque.adapter.mapper		89%		50%	2	5	2	16	0	3	0	1
br.com.fiap.msestoque.adapter.exception		100%		n/a	0	10	0	65	0	10	0	2
br.com.fiap.msestoque.core.usecase		100%		100%	0	16	0	29	0	6	0	3
br.com.fiap.msestoque.adapter.gateway		100%		n/a	0	3	0	10	0	3	0	1
br.com.fiap.msestoque.adapter.controller.request		100%		n/a	0	3	0	1	0	3	0	1
br.com.fiap.msestoque.adapter.controller.response		100%		n/a	0	3	0	1	0	3	0	1
br.com.fiap.msestoque.core.exception		100%		n/a	0	1	0	2	0	1	0	1
Total	13 of 577	97%	2 of 24	91%	3	46	6	139	1	34	0	12



mspedidoprocessor

mspedidoprocessor

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
br.com.fiap.mspedidoprocessor.adapter.persistence.entity		58%		0%	26	56	10	25	25	55	0	4
br.com.fiap.mspedidoprocessor.adapter.gateway.dto		0%		n/a	14	14	2	2	14	14	2	2
br.com.fiap.mspedidoprocessor.adapter.external.clientkafka.dto		65%		n/a	11	28	7	24	11	28	1	4
br.com.fiap.mspedidoprocessor.core.domain		87%		n/a	5	57	1	33	5	57	0	7
br.com.fiap.mspedidoprocessor.adapter.mapper		82%		38%	11	20	14	78	2	11	0	1
br.com.fiap.mspedidoprocessor.core.usecase.pedidoprocessor		97%		83%	4	42	4	163	0	29	0	6
br.com.fiap.mspedidoprocessor.adapter.external.estoqueservice.dto		53%		n/a	4	7	4	6	4	7	1	3
br.com.fiap.mspedidoprocessor.adapter.external.pagamentoservice.dto		77%		n/a	5	12	0	2	5	12	0	2
br.com.fiap.mspedidoprocessor		0%		n/a	2	2	3	3	2	2	1	1
br.com.fiap.mspedidoprocessor.adapter.external.produtoservice.dto		88%		n/a	1	5	0	1	1	5	0	1
br.com.fiap.mspedidoprocessor.core.exception		100%		n/a	0	14	0	29	0	14	0	9
br.com.fiap.mspedidoprocessor.adapter.config		100%		n/a	0	3	0	16	0	3	0	1
br.com.fiap.mspedidoprocessor.adapter.listener		100%		n/a	0	4	0	14	0	4	0	1
br.com.fiap.mspedidoprocessor.adapter.gateway		100%		75%	1	5	0	11	0	3	0	1
br.com.fiap.mspedidoprocessor.adapter.controller		100%		100%	0	3	0	6	0	2	0	1
br.com.fiap.mspedidoprocessor.adapter.controller.response		100%		n/a	0	4	0	1	0	4	0	1
br.com.fiap.mspedidoprocessor.adapter.external.produtoservice		100%		n/a	0	1	0	4	0	1	0	1
Total	400 of 2,331	82%	18 of 50	64%	84	277	45	418	69	251	5	46

4. Endpoints da API

4.1 Tabela de Endpoints

Método	Endpoint	Descrição
POST	/clientes	Criar Cliente
GET	/clientes/{id}	Buscar Cliente por ID
PUT	/clientes/{id}	Atualizar Cliente
DELETE	/clientes/{id}	Deletar Cliente
GET	/clientes	Listar Todos os Clientes

POST	/produtos	Cadastrar Produto
GET	/produtos/{id}	Buscar por ID
GET	/produtos/sku/{id}	Buscar por SKU
GET	/produtos/nome/{id}	Buscar por Nome
GET	/produtos	Listar Todos
PUT	/produtos/{id}	Atualizar Produto
DELETE	/produtos/{id}	Deletar Produto

PUT	/estoque/baixa	Baixa
GET	/estoque/consulta/{id}	Consulta Produto Estoque
PUT	/estoque/estorno	Estorno

POST	/pagamentos	Pagamento
GET	/pagamentos/{id}	Consulta

POST	/pedidos-reciver	Pedido Receiver
GET	/pagamento/consultar/{id}	Pedido Processor

5. Boas Práticas Utilizadas

5.1 Aplicação de Clean Code no Projeto

Este projeto adota práticas de Clean Code, garantindo que o código seja legível, compreensível e sustentável. Abaixo estão as principais práticas adotadas, acompanhadas de onde elas aparecem no código:

1. Nomes Significativos

Todas as classes, métodos e variáveis usam nomes claros e descritivos, facilitando a leitura.

• ClienteAtualizar, ClienteBuscarporId, etc

Esses nomes deixam claro o que a função ou classe faz, sem a necessidade de comentários adicionais.

2. Funções Pequenas e Focadas

Os métodos realizam apenas uma tarefa e são coesos.

Exemplo aplicado: • O método ClienteBuscarPorIdResource apenas para buscar os clientes e retornar a resposta, delegando toda a lógica ao serviço

Separação de Responsabilidades

O projeto respeita a divisão entre camadas: • Controllers lidam com a entrada da requisição HTTP. • Services contêm as regras de negócio. • Repositories acessam os dados.

Exemplo: • O ConsultaController apenas recebe a requisição e chama consultaService.salvarConsulta(...), sem conter lógica de negócio.

4. Reutilização e Eliminação de Código Duplicado

A lógica de conversão, validação e persistência foi encapsulada em métodos e serviços reutilizáveis.

5. Tratamento de Erros Claro

Exceções específicas e tratadas são utilizadas para facilitar a identificação de falhas.

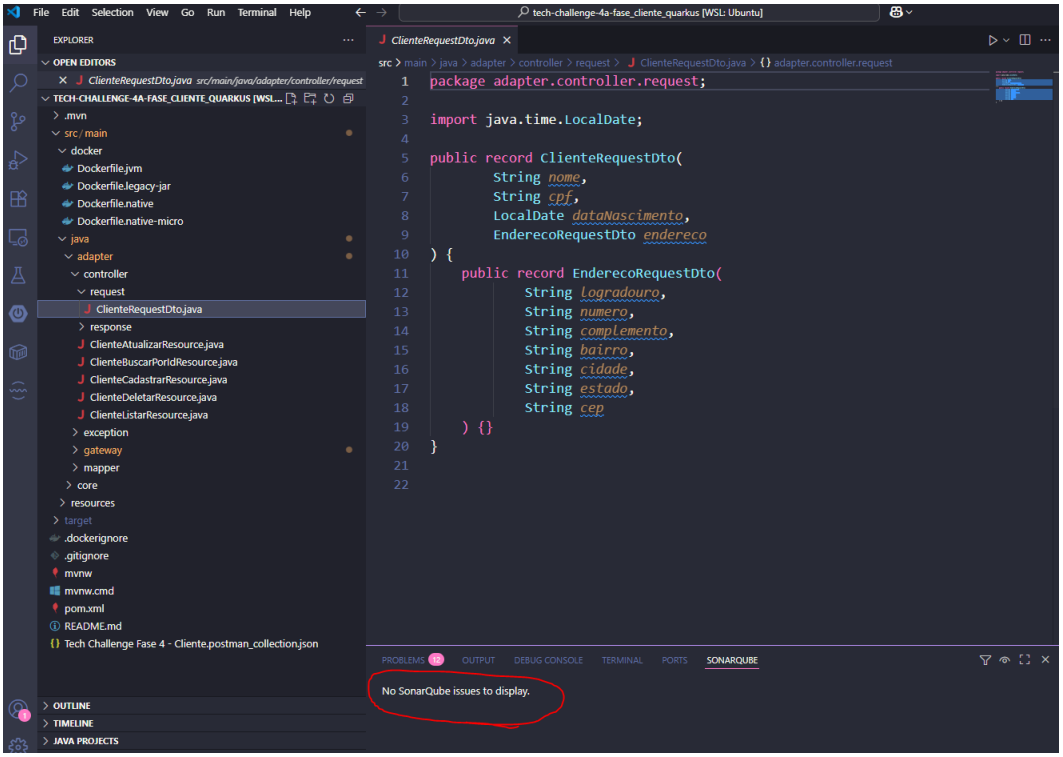
Exemplo: • EntityNotFoundException é usada para retornar 404 quando uma consulta não é encontrada. • Logs com logger.error(...) ou logger.warn(...) facilitam o rastreamento dos erros.

5.2 SONAR

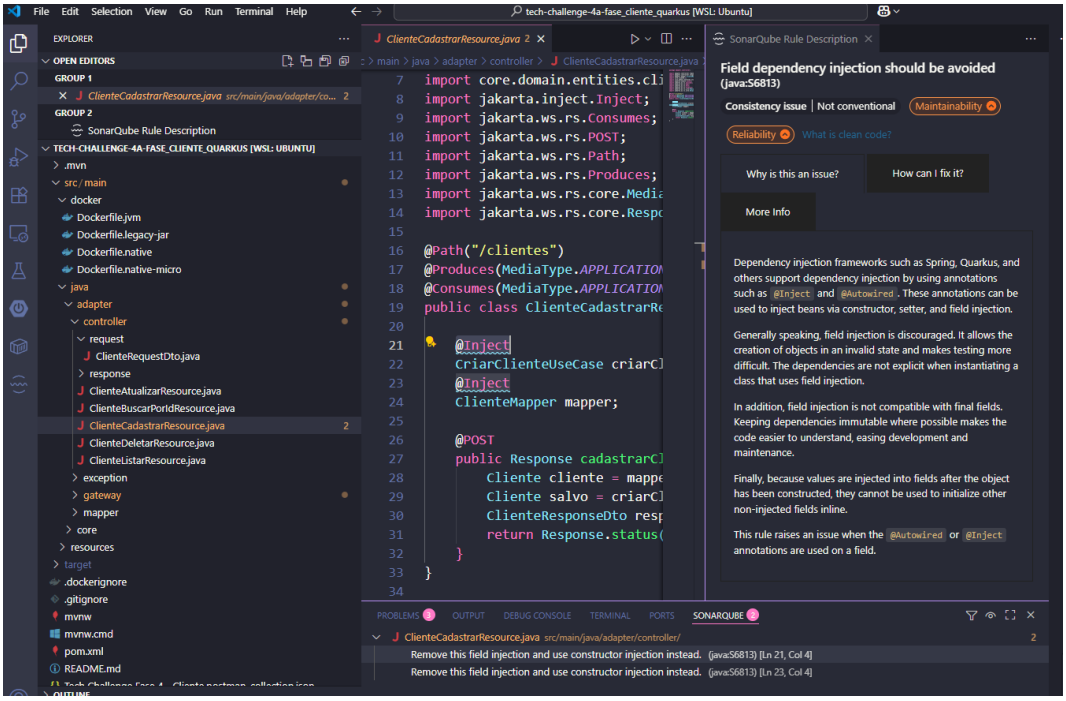
Utilizado a ferramenta Sonar para verificação do código desenvolvido, garantindo a qualidade e padrão do sistema.

Exemplos no código:

Códigos com apontamentos OK no sonar:



Apontamento deixado para testes Sonar:



7. Postman

7.1 link para collections

https://github.com/fabiopinheirobatista/tech-challenge-4a-fase_orquestrador/blob/feature/iniciando/FIAP%20-%20Meu%20TechChalange%20-%20FASE%204.postman_collection.json

7.2 Descriçao dos testes manuais

- Acessar as Collections nos repositories GITHUB:
    - Realizar o download das collections do link fornecido no GITHUB e abrir o arquivo no ambiente local, se necessário.
- Importar a Collection no Postman:
    - Abrir o Postman e, na barra lateral esquerda, clicar em "Import".
    - Selecionar a opção para importar uma coleção e carregar os arquivos que foram baixados do Google Drive.
- Visualizar a Collection:
    - Uma vez importada, verificar a collection listada no painel do Postman. Clicar nela para expandir e visualizar as requisições incluídas.
- Revisar as Requisições:
    - Clicar em cada requisição para revisar as seguintes informações:
        - Método HTTP (GET, POST, PUT, DELETE).
        - Endpoint URL.
        - Cabeçalhos (Headers) configurados, como autenticação ou tipo de conteúdo.
        - Corpo da requisição (Body), se aplicável.
- Executar os Testes:
    - Para cada requisição que necessitar de teste:
        - Clicar em "Send" para enviar a requisição e aguardar a resposta do servidor.
        - Verificar a resposta:
            - Conferir se o código de status retornado está de acordo com o esperado (ex.: 200, 201, 404).
            - Observar o formato da resposta, confirmando se está em JSON (ou outro formato, conforme esperado) e se contém os dados corretos.
            - Validar se o conteúdo da resposta corresponde ao que deve ser retornado, conforme as especificações da API.

8. Repositório do código

8.1. URL do Repositório

https://github.com/fabiopinheirobatista/tech-challenge-4a-fase_cliente_quarkus
https://github.com/fabiopinheirobatista/tech-challenge-4a-fase_pagamento
https://github.com/fabiopinheirobatista/tech-challenge-4a-fase_estoque
https://github.com/fabiopinheirobatista/tech-challenge-4a-fase_pedido_receiver
https://github.com/fabiopinheirobatista/tech-challenge-4a-fase_pedido_processor
https://github.com/fabiopinheirobatista/tech-challenge-4a-fase_cliente
https://github.com/fabiopinheirobatista/tech-challenge-4a-fase_produto
https://github.com/fabiopinheirobatista/tech-challenge-4a-fase_orquestrador

9. Vídeo explicando o código e sua estrutura

https://drive.google.com/file/d/1HhbgODslD_qomniqWPLra56EM3h2hBWL/view?usp=drivesdk