Tech Challenge – Fase 1

Grupo 2 - Fernanda T. Piva – rm354516

Introdução

Este documento contém algumas informações importantes a respeito do trabalho da fase 1 realizado para o curso de Software Architecture.

Um ponto a destacar é que a aplicação foi feita em python, dado o contexto atual de minha atuação de dados. Com isso existem algumas limitações, principalmente na aplicação da arquitetura hexagonal, mais detalhes na sessão de detalhamento dos entregáveis.

Links

Miro – https://miro.com/app/board/uXjVKFny-xg=/?share_link_id=324313442023

GitHub – https://github.com/FeTPiva/tech_challenge

Qualquer problema de acesso ou build da aplicação entrar em contato comigo!

@Fernanda Piva – rm354516 ou direto pelo meu user do discord @aelin18

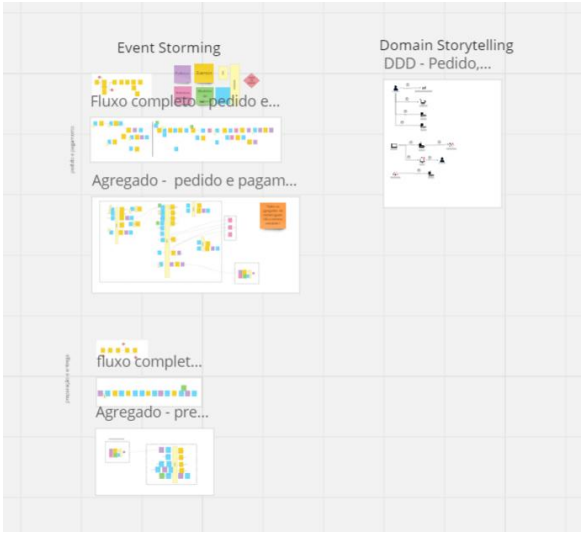
Detalhamento dos entregáveis

DDD e Event Storming

Ambos fluxos estão documentados no Miro . Fluxo de pedido e pagamento está separado de preparação e entrega pelos frames, com as etapas até a versão agregada final.

Uma observação é que devido a múltiplos caminhos possíveis da aplicação, principalmente em pedido e pagamento, dividi em vários quadros o agregado ‘Pedido’, porém a nível de arquitetura todos os pedidos são a mesma entidade.

Na parte de DDD mantive os fluxos juntos, pensando no ponto que ficaria mais simples para uma pessoa de negócios entender, do que algo apartado por mais que no futuro a aplicação possa vir a ser separada em microserviços.

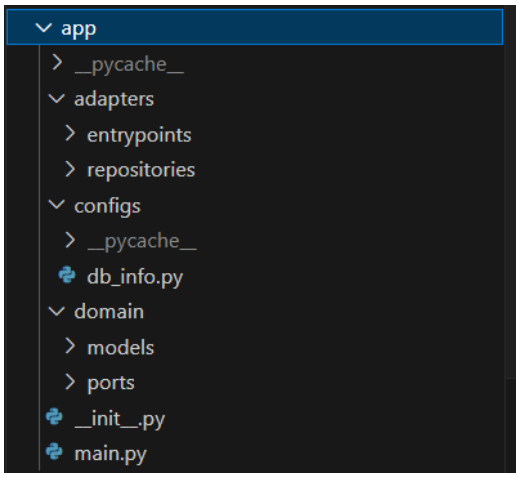


https://miro.com/app/board/uXjVKFny-xg=/?share_link_id=324313442023

API

A API foi feita em python usando o framework FastAPI, adaptando à arquitetura hexagonal, se conectando a uma estancia do MySQL localmente, com um script populando de forma automática em seu build com as tabelas essenciais para a aplicação.

Por ser uma aplicação python, existem algumas limitações como o uso de interfaces. Mas busquei manter as boas práticas da arquitetura hexagonal o máximo possível, na separação de pastas com seus objetivos específicos.



Domain/Models - aqui está a declaração das entidades utilizando o BaseModel da lib pydantic.

Domain/Ports - tentando abstrair a ideia de ports & adapters, aqui estão as classes com métodos abstratos de cada entidade.

Configs - Configurações de conexão do banco de dados MySQL

Adapters/Repositories - Para cada entidade foi utilizada herança para implementação dos métodos, integrando ao banco de dados.

Adapters/Entrypoints - definição de rotas de entrada utilizadas pela API

Como buildar o app?

Depois do clone do código via git (https://github.com/FeTPiva/tech_challenge) basta buildar o docker compose pelo comando ‘ *docker compose –d --build* ’ dentro da pasta onde foi baixado.

O swagger pode ser acessado em http://localhost:8000/docs

Rotas

I - Cadastro do cliente
rota POST http://localhost:8000/client/

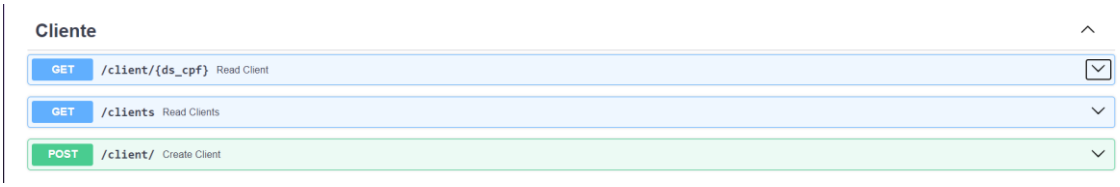
Detalhes do body necessário vide documentação do swagger

II - Identificação do cliente via CPF

O cliente pode ser identificado por CPF no campo ds_cpf ou id_cliente. Pensando em boas práticas de privacidade, tabelas que interagem com cliente utilizam id_cliente.

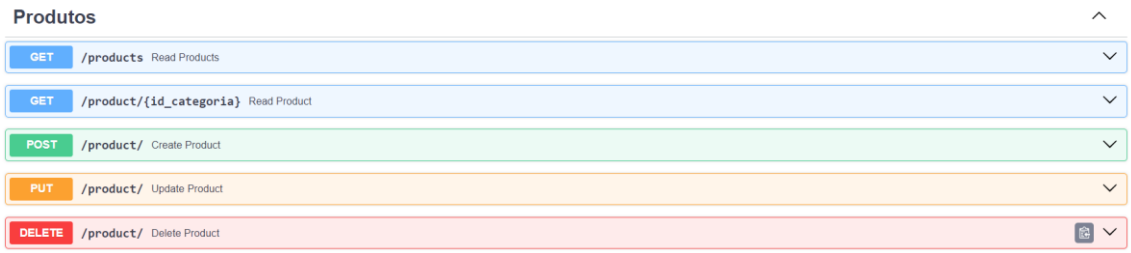
Para consultar a estrutura da tabela é possível usar a rota GET http://localhost:8000/clients ou vide swagger

Existe também uma rota onde é possível resgatar as informações do cliente via CPF, http://localhost:8000/client/{ds_cpf}



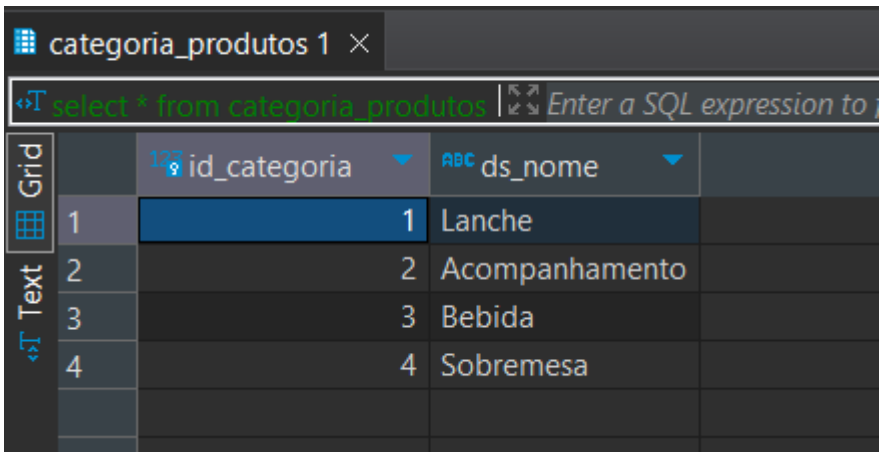
III - CRUD de produtos

Na sessão ‘Produtos’ do swagger é possível ver todos os detalhes das rotas



IV - Buscar produtos por categoria

Existe a rota GET http://localhost:8000/product/{id_categoria} onde é possível realizar essa consulta, sendo os valores de id_categoria conforme consta na tabela ‘categoria_produtos’ do MySQL:



V - Fake checkout

Tendo em vista que o fake checkout é o 'envio' de informações para a cozinha, ignorando a etapa de pagamento, existe a rota POST http://localhost:8000/order/ responsável por pegar as informações do cliente e dos produtos escolhidos, e salvá-las no banco de dados.

VI - Listar pedidos

Buscando manter boas práticas de um banco relacional como o MySQL, existe a tabela 'pedido' contendo informações do pedido em si e a tabela 'itens_pedido' com os dados de id_pedido relacionados a um ou mais itens escolhidos pelo cliente.

Para trazer as informações de forma mais clara na listagem de pedido, criei quatro rotas, onde é possível ler os pedidos (orders) ou ler pedidos com detalhamento dos produtos escolhidos pelo cliente, podendo ser filtrado também por id_pedido