**Bootcamp: 2025-2A - POS - Bootcamp Arquiteto(a) de Software**

**Aluno(a): Leonardo Barcelos Marques**

**Relatório de Entrega do Desafio Final**

Introdução

O desafio final é uma atividade realizada pelo(a) aluno(a), **individualmente**, visando **levar à prática o conteúdo do bootcamp na totalidade**, utilizando as principais ferramentas, conteúdos, dinâmicas e modalidades orientadas pelo coordenador do curso.

O desafio final é **composto por um enunciado**, com viés de atividade prática (laboratório guiado, desafio a resolver, case a analisar etc.) com os principais tópicos vistos ao longo dos quatro módulos do bootcamp.

Quando necessário, o enunciado do desafio final pode ser acompanhado de uma videoaula gravada de orientação, onde o professor explica qual é a proposta do desafio final do bootcamp em questão, detalhando a dinâmica e elencando ferramentas/pré-requisitos para a resolução do enunciado.

Os principais passos, as escolhas e decisões tomadas para a resolução do enunciado do desafio final, bem como o resultado final, devem ser registrados pelo(a) aluno(a) neste documento, observando-se as **instruções de preenchimento**. Importante também observar os critérios no **protocolo avaliativo** da entrega.

Expirado o prazo de entrega, será disponibilizada uma **videoaula gravada contendo a resolução** da prática/problema/case proposto no enunciado do desafio final, bem como as devidas explicações acerca das decisões tomadas/ações executadas pelo professor para chegar no resultado.

**Protocolo Avaliativo**

O valor total do desafio final é de **95 pontos**. Ele será avaliado pelo professor do módulo em questão, com base no relatório do desafio final enviado pelo(a) aluno(a) e seguindo-se o protocolo avaliativo abaixo:

1. Todos os itens solicitados pelo(a) professor(a) no enunciado do desafio final foram entregues?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

1. Os desafios/requisitos/funcionalidades foram resolvidos/considerados/implementados corretamente pelo(a) aluno(a)?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

1. O(a) aluno(a) utilizou corretamente as ferramentas, conceitos e tecnologias apresentadas ao longo do bootcamp?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

1. O(a) aluno(a) forneceu comentários detalhados e documentação completa, facilitando a compreensão da sua proposta de resolução do desafio?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

1. O(a) aluno(a) entregou a solução com qualidade e zelo, demonstrando dedicação e empenho na solução do desafio?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

**Instruções de Preenchimento do Relatório do Desafio Final**

O preenchimento do relatório deve ser realizado com base nas diretrizes abaixo, para assegurar uma avaliação justa e detalhada de cada item solicitado.

1. **Entrega Completa dos Itens Solicitados**

Verifique se todos os itens mencionados no enunciado do desafio final foram devidamente contemplados na sua entrega, pois a entrega deles é fundamental para a avaliação do desafio final na totalidade.

1. **Resolução dos Desafios e Implementação dos Requisitos**

Avalie se você abordou corretamente o(s) desafio(s) proposto(s), considerando e implementando todas as funcionalidades e requisitos apresentados no enunciado. O objetivo é verificar se a solução atende às expectativas técnicas e funcionais solicitadas.

1. **Utilização das Ferramentas, Conceitos e Tecnologias**

Certifique-se de que você aplicou corretamente as ferramentas, conceitos e tecnologias discutidas ao longo do bootcamp. A adequação e o uso eficaz dos recursos disponíveis são elementos essenciais para a solução do desafio.

1. **Qualidade dos Comentários e Documentação**

Verifique se você forneceu os comentários necessários e detalhados acerca dos passos e decisões tomadas na resolução do desafio. Esses comentários servem para facilitar a compreensão da proposta de solução, demonstrando clareza e organização.

1. **Qualidade e Capricho na Entrega Final**

Observe se o seu relatório do desafio final apresenta um nível adequado de qualidade, zelo e capricho, de forma que evidencie sua dedicação, empenho e atenção aos detalhes durante o desenvolvimento da solução.

O preenchimento deve ser realizado com base em evidências concretas encontradas no trabalho entregue, assegurando que todos os aspectos mencionados acima sejam considerados.

**Desenvolvimento da Solução do Desafio Final**

1. **Escolha de Plataforma e Linguagem:** O framework/linguagem escolhidos foram: .NET 9 com a linguagenm C#, eu trabalho com este framework e linguagem no dia-a-dia e é o ideal para que eu consiga apresentar um projeto exemplificando a arquitetura funcional.
2. **Funcionalidades da API:** A API será dividida em 3 endpoints base [produtos, clientes, pedido]:

/produtos [GET] => Retorna todos produtos cadastrados

/produtos/count [GET] => Retorna a quantidade de produtos cadastrada

/produtos/nome/{nome} [GET] => Retorna todos produtos cadastrados filtrando pelo nome (LIKE)

/produtos/{id} [GET] => Retorna um produto por id

/produtos [POST] => Cadastra um novo produto

/produtos/{id} [PUT] => Atualiza um produto

/produtos/{id} [DELETE] => Remove um produto

/clientes [GET] => Retorna todos clientes cadastrados

/clientes/count [GET] => Retorna a quantidade de clientes cadastrada

/clientes/nome/{nome} [GET] => Retorna todos clientes cadastrados filtrando pelo nome (LIKE)

/clientes/{id} [GET] => Retorna um cliente por id

/ clientes [POST] => Cadastra um novo cliente

/clientes/{id} [PUT] => Atualiza um cliente

/clientes/{id} [DELETE] => Remove um cliente

/pedidos/cliente/{id} [GET] => Retorna todos os pedidos de um clientes cadastrados

/pedidos/count [GET] => Retorna a quantidade de pedidos cadastrada

/pedidos/{id} [GET] => Retorna um pedido por id

/pedidos [POST] => Cadastra um novo pedido

1. **Arquitetura:** A arquitetura utilizada será a MVC conforme solicitado no enunciado:

\* - A camada Model (M do MVC) será composta pelo projeto lib MVC-XP.Model que contem a seguinte estrutura de diretórios e as classes/entidades:

**MVC-XP.Model**

[DTO]

ClienteDtos => Classes para comunicação entre a API e os serviços do modelo relativas a clientes

PedidoDtos => Classes para comunicação entre a API e os serviços do modelo relativas a pedidos e itens de pedidos

ProdutoDtos => Classes para comunicação entre a API e os serviços do modelo relativas a produtos

[Interfaces]

IClienteService => Interface de serviço com regras para clientes

IProdutoService => Interface de serviço com regras para produtos

IPedidoService => Interface de serviço com regras para pedidos e itens de pedidos

IRepository => Interface de repositório para acesso as entidades no database

[Entidades]

Cliente (Classe que guarda os dados do cliente com as propriedades Id, Nome)

Produto (Classe que guarda os dados do produto com as propriedades Id, Nome, Valor)

Pedido (Classe que guarda os dados principais do pedido com as propriedades Id, Data, ClienteId, ValorTotal)

PedidoItem (Classe que guarda os dados dos itens do pedido com as propriedades Id, PedidoId, ProdutoId, Quantidade)

[Services] (Serviços com as operações de negócio que usam os repositórios para interagir com os dados)

ClienteService

ProdutoService

PedidoService

\* A camada Controller (C do MVC) será composta pelo projeto lib MVC-XP.Api que contem os controladores que chama os serviços da camada model para interagir com os dados e possui os diretórios / classes:

**MVC-XP.Api**

[Controllers]

ClienteController (Configura e disponibiliza os dados para interação com clientes)

ProdutoController (Configura e disponibiliza os dados para interação com produtos)

PedidoController (Configura e disponibiliza os dados para interação com pedidos)

\* - A camada View (V do MVC) não está presente, mais ela interagirá com a Api através de requisições REST para recuperar e inserir dados.

\* - Existe uma camada que implementa a "infra" ou seja, as tecnologias usadas pela camada Model para interagir com dados:

**MVC-XP.Infra**

MVCXPRepository => Classe com as funções base do CRUD

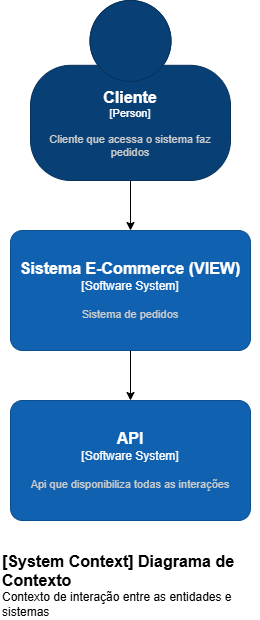
MVCXPContext => Classe que mapeia as entidades para serem usadas em um contexto de banco de dados

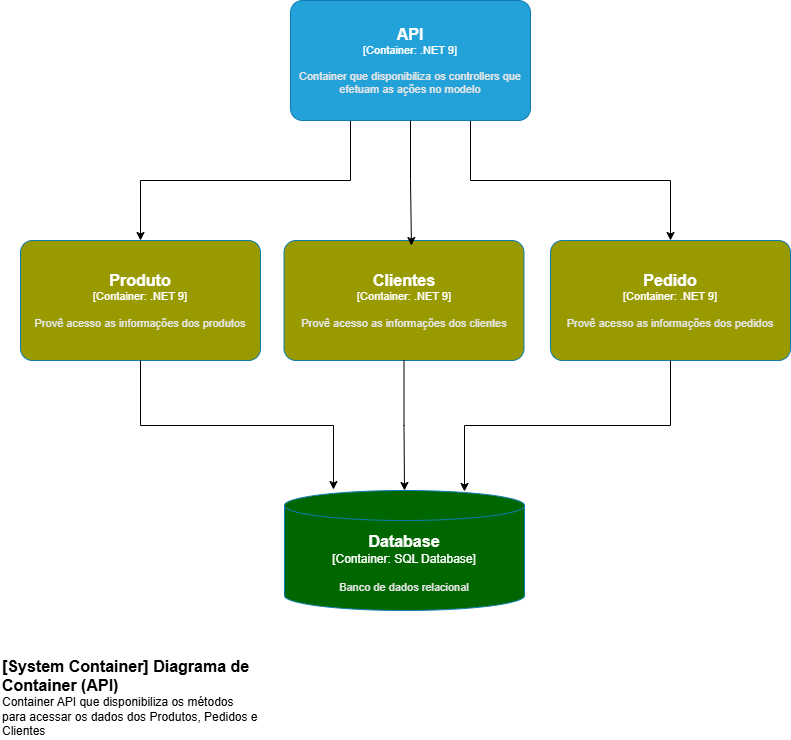
\* - A camada de testes de unidade que são usados para garantir a integridade do código e a consistência nas alterações:

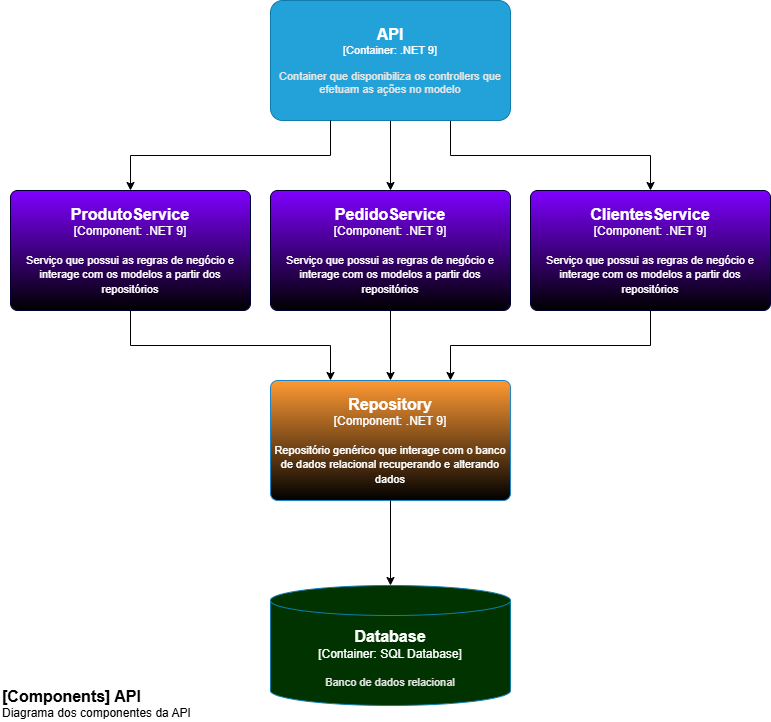
**MVC-XP.Tests**

1. **Entrega do Exercício:**

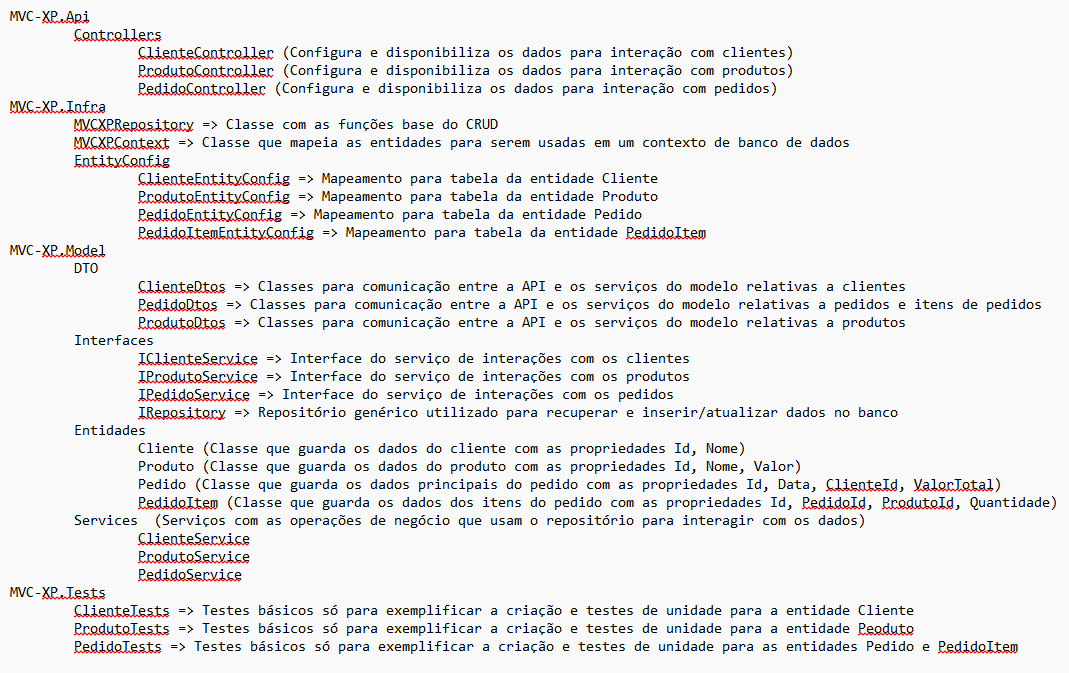
**Desenho arquitetural do software:**

****

****

****

**Diretórios e classes:**

****

**Entrega do Código:**

**https://github.com/LeoBMarques/MVC-XP.git**

**Conclusão**

Nesta seção, você deve resumir o seu desempenho ao longo do desafio final, destacando os principais pontos da sua experiência. A conclusão serve como uma oportunidade para refletir sobre o processo de resolução do desafio, abordando aspectos como o que foi aprendido, as dificuldades enfrentadas e as soluções implementadas.

Ao elaborar sua conclusão, considere os seguintes tópicos:

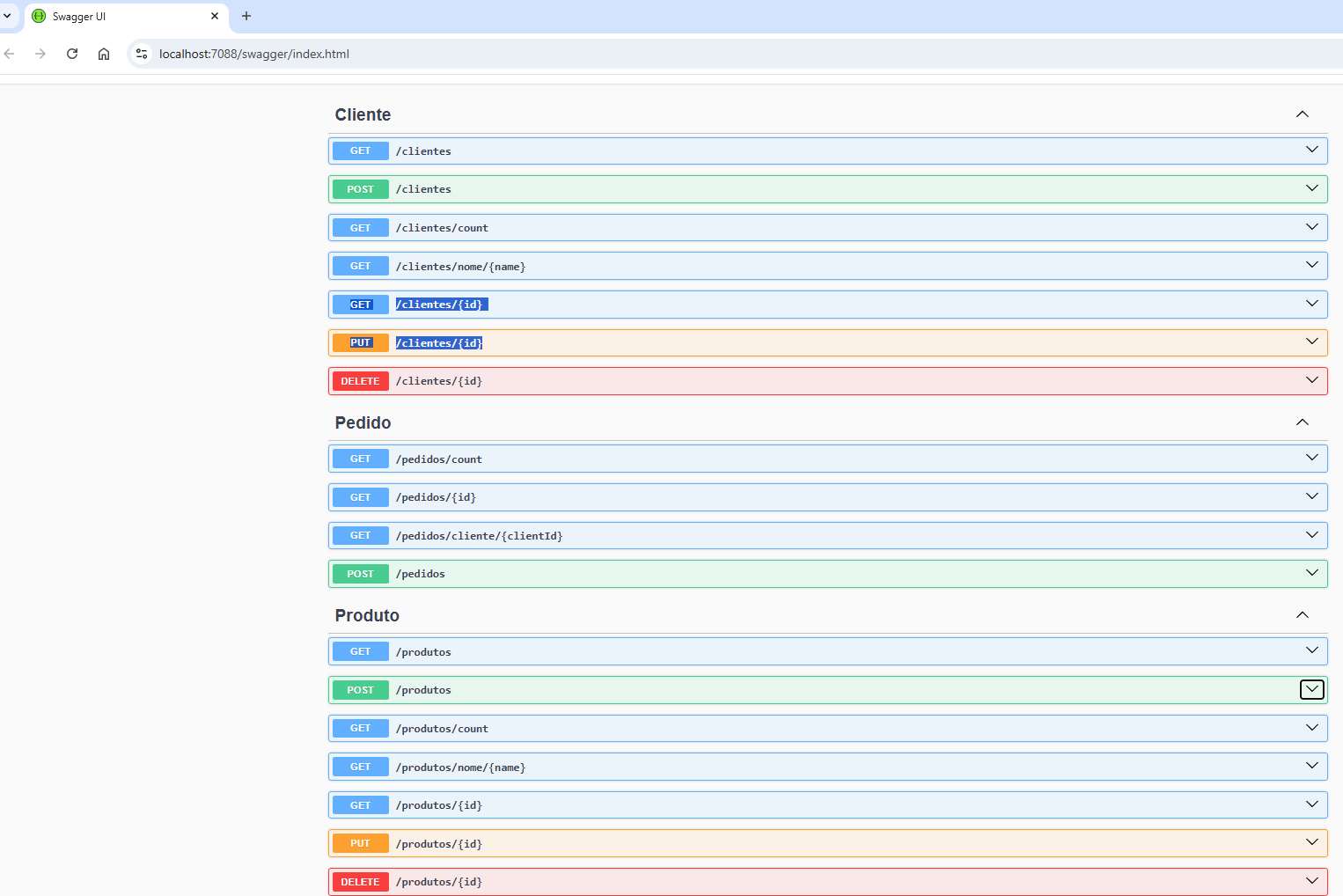
1. **Aplicação dos Conhecimentos:** durante o decorrer do bootcamp foram expostos todos os processos que permeiam a criação da arquitetura de um sistema, no módulo 2 por exemplo, fomos introduzidos a levantamento de requisitos onde pude aprende a escolher quais recursos eu precisaria utilizar no projeto final como por exemplo, a interação entre as camadas "superiores" com as "inferiores", especificação de necessidades de persistência e modelagem de dados, no módulo 3 aprendi sobre os design patterns e estilos/padrões arquiteturais onde consegui entender o uso / função tanto do C4 que usei no modelo da arquitetura com o drawio (exemplificado nas aulas) quanto no modelo arquitetural do código MVC, e nas melhorias que apresentarei em seguida, falo da necessidade de "migração" do modelo MVC para um modelo mais dinâmico e abrangente como clean arquiteture por exemplo.
2. **Principais Dificuldades e Superações:** a maior dificuldade que tive foi tentar utilizar o modelo MVC que considero um modelo ultrapassado, limitar a arquitetura a este modelo restringe muito o que costumo usar no dia-a-dia onde trabalho, com a divisão de domínios, contextos, camadas, más revendo e entendendo o MVC acredito que consegui dividir corretamente as "camadas" e criar o projeto conforme solicitado.
3. **Resultados Obtidos:** a api que criei possui os métodos solicitados em que podemos recuperar os dados e fazer inserções / atualizações, assim como pesquisas específicas por nome ou id e exclusão, e para seguir a convenção ainda mantive a lógica REST onde em muitos casos o endpoint é o mesmo, o que determina a ação é o "verbo" REST em que a chamada é feita [GET, POST, PUT, DELETE].
4. **Lições Aprendidas:** este desafio proporcionou um aprendizado diferente na minha carreira, eu nunca havia pensado no sistema e planejado ele completamente com diagramas, casos de uso, estruturas antes de codificar, sempre atuei diretamente na construção do código, foi algo diferenciado e que trouxe uma nova perspectiva, eu percebi na pratica como é mais simples a construção do código depois de ter planejado anteriormente o modelo arquitetural, os diagramas e a comunicação entre os componentes da arquitetura.
5. **Melhorias Futuras:** o primeiro ponto a ser melhorado é o da segurança, deve-se criar uma entidade de usuário que possua senha e que gere um token jtw para interação nas chamadas REST, além de "fechar" as requisições para não permitir que qualquer pessoa faça interações (somente usuários logados) traz mais integridade nas transações porque os dados do usuário cliente estarão sempre coerentes com o usuário logado.

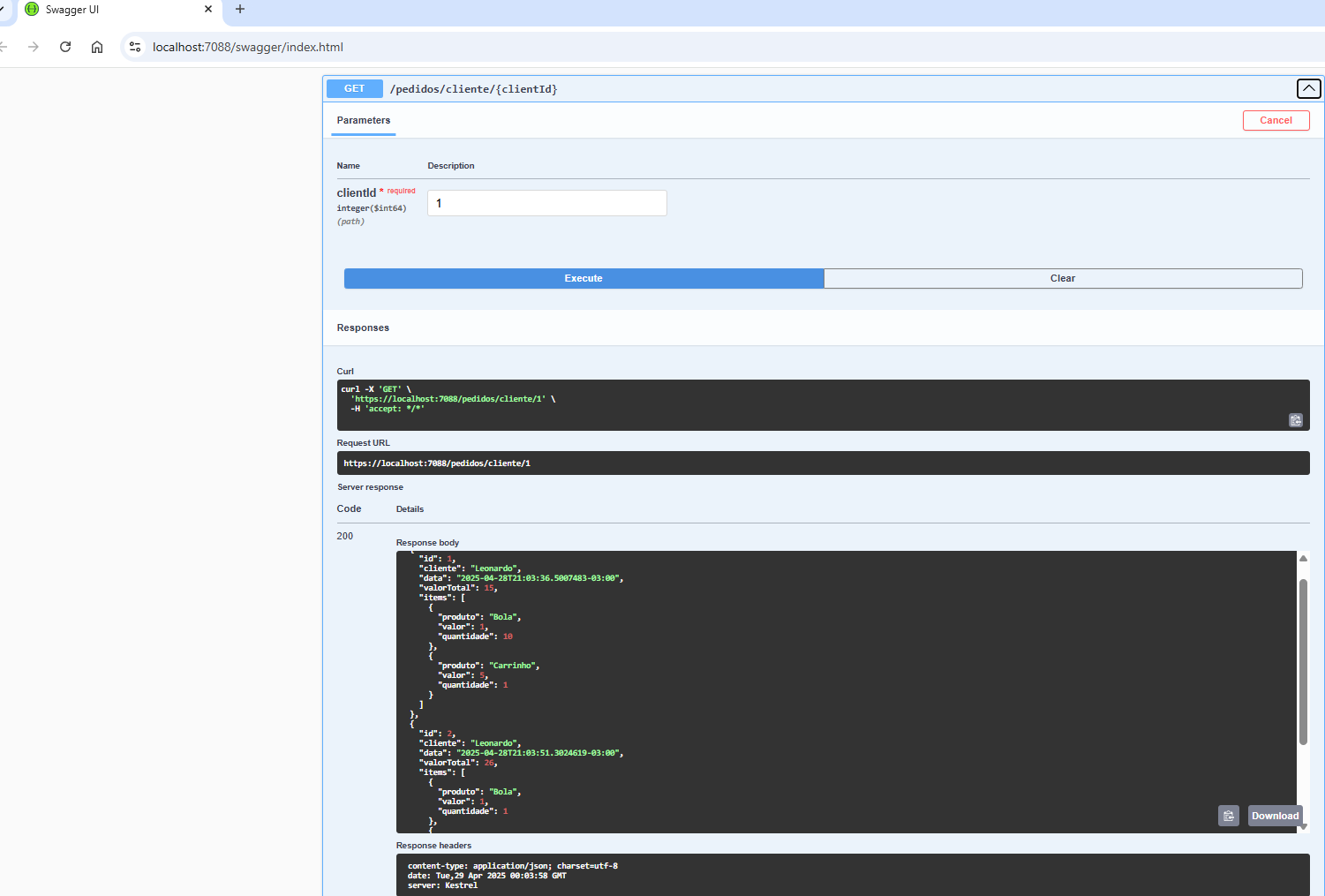
Um outro ponto a ser melhorado é a criação de notificações de eventos que ocorrem nas requisições como por exemplo, cadastro de novos clientes, alterações em produtos, novos pedidos feitos e etc…

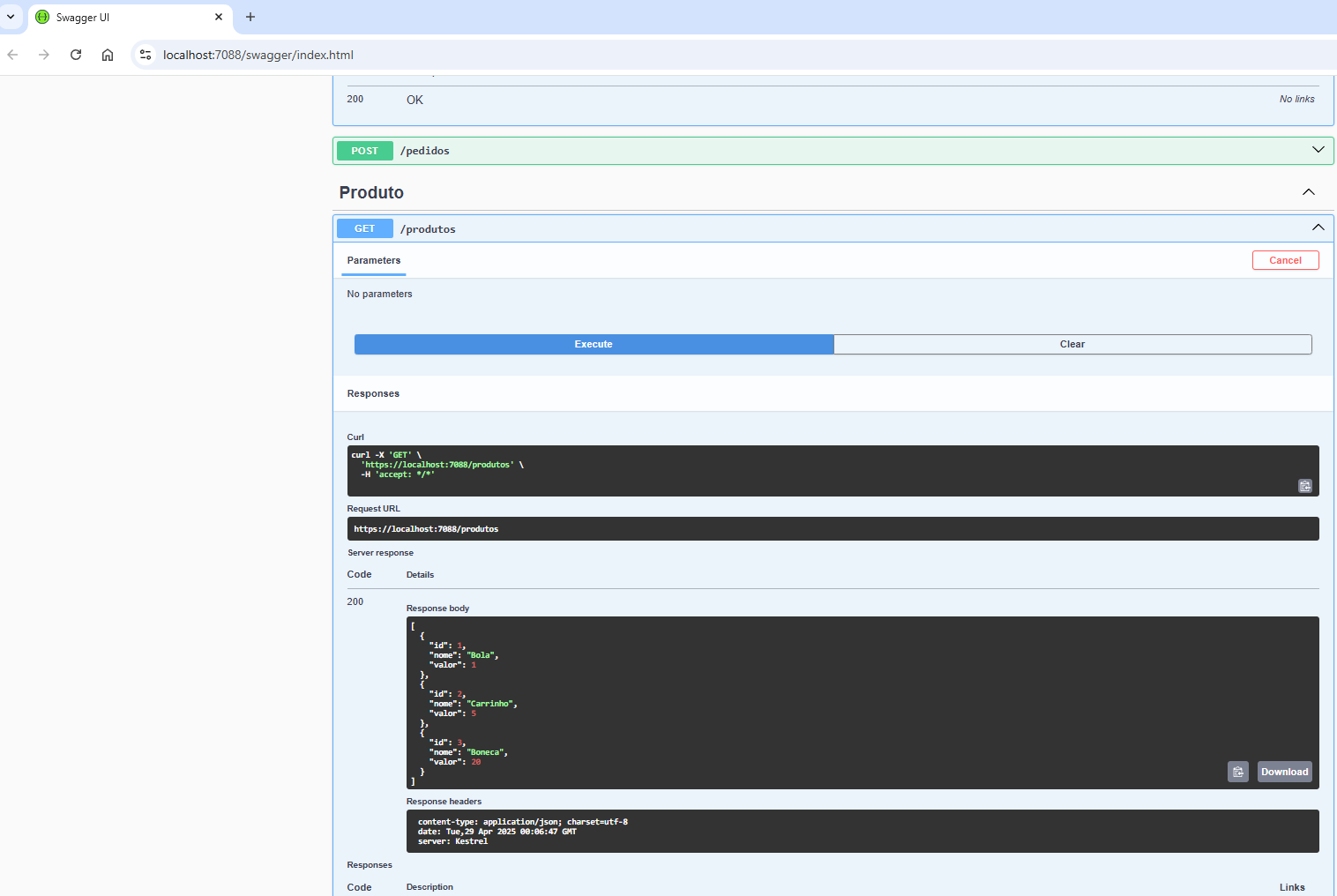
Agora o principal ponto seria evoluir a arquitetura para um modelo clean + DDD onde teremos uma melhoria na dinâmica e manuteniblidade do sistema determinando contextos delimitados e específicos o que centralizará as mudanças por contexto das entidades diminuido os chamados “efeitos colaterais” de alterações e evoluções em entidades que afetam outras que não tem relação alguma.

**Print do runtime da API:**

**Swagger com todos endpoints:**

**Swagger de pesquisa de pedidos de um cliente:**

**Swagger pesquisa de todos produtos cadastrados:**

****