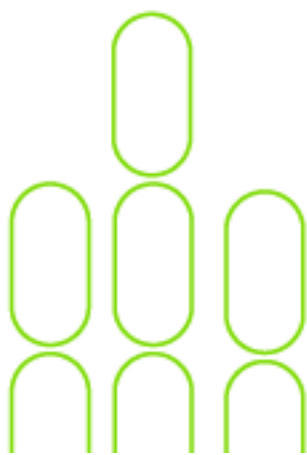




# **Bootcamp: Arquitetura de Software**

**Aluno(a): Natanael Felipe Pereira Nunes**

**Relatório de Entrega do Desafio Final**



<b>Introdução.....</b>	<b>3</b>
<b>Visão Geral.....</b>	<b>4</b>
<b>Objetivos.....</b>	<b>4</b>
<b>Arquitetura da API.....</b>	<b>4</b>
<b>Desenvolvimento da Solução do Desafio Final.....</b>	<b>5</b>
<b>Camadas do projeto (Modelo C4).....</b>	<b>5</b>
<b>Diagrama de Classes.....</b>	<b>7</b>
<b>Estrutura de pastas.....</b>	<b>8</b>
<b>Estrutura na prática.....</b>	<b>10</b>
VendasOnline.sln.....	10
VendasOnline.API.....	10
Controllers.....	11
Swagger UI.....	11
Configurações.....	12
Middlewares.....	12
Validadores.....	12
Program.cs.....	12
appsettings.json.....	13
VendasOnline.Dominio.....	13
Entidades.....	13
Interfaces.....	13
Enums.....	13
DTOs.....	13
VendasOnline.Infraestrutura.....	14
Contexto.....	14
Repositorios.....	14
Migrations.....	14
VendasOnline.Servico.....	14
Serviços.....	14
<b>Fluxo de Operação na arquitetura.....</b>	<b>15</b>
<b>Benefícios dessa arquitetura.....</b>	<b>15</b>
<b>Possíveis melhorias.....</b>	<b>15</b>
<b>Github do projeto.....</b>	<b>15</b>
<b>Conclusão.....</b>	<b>16</b>
Resumo do Desempenho.....	16
Aplicação dos Conhecimentos.....	16
Principais Dificuldades e Superações.....	16
Resultados Obtidos.....	17
Lições Aprendidas.....	17



## Introdução

O Desafio Final do Bootcamp de Arquitetura de Software foi concebido como uma atividade prática individual para consolidar os conhecimentos adquiridos ao longo dos quatro módulos do curso, com foco na aplicação de conceitos, ferramentas e práticas de desenvolvimento de software. Este desafio propôs a criação de uma API RESTful para um sistema de vendas online, utilizando o framework .NET e seguindo os princípios de arquitetura de software, como o padrão MVC, separação de responsabilidades e os princípios SOLID. A atividade abrange tópicos como fundamentos de arquitetura, padrões de design, persistência de dados, validação, documentação e testes, exigindo a integração de diversas tecnologias e metodologias apresentadas durante o bootcamp.

A solução desenvolvida, intitulada **API de Vendas Online**, é uma aplicação robusta que gerencia produtos, clientes e pedidos, oferecendo endpoints RESTful para operações CRUD, validação de dados, tratamento de erros e documentação interativa via Swagger. O projeto foi estruturado em camadas (Apresentação, Domínio, Serviço e Infraestrutura), utilizando ferramentas como Entity Framework Core, AutoMapper, FluentValidation e ASP.NET Core. Este relatório documenta o processo de desenvolvimento, detalhando os passos executados, as decisões tomadas e os resultados alcançados, com o objetivo de demonstrar a aplicação prática dos conceitos aprendidos e a entrega de uma solução funcional, bem documentada e alinhada com as melhores práticas de arquitetura de software.

O desafio foi uma oportunidade para exercitar habilidades técnicas, como modelagem de domínio, implementação de APIs e configuração de bancos de dados, além de competências como planejamento, resolução de problemas e documentação clara. A seguir, apresento os passos de desenvolvimento, as evidências de implementação e uma reflexão sobre o processo.





## Visão Geral

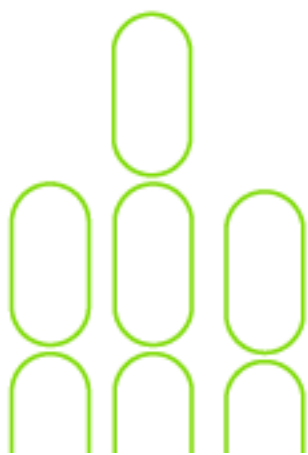
A API de Vendas Online é uma solução RESTful desenvolvida para gerenciar operações de e-commerce, incluindo o gerenciamento de produtos, clientes e pedidos. O sistema foi projetado utilizando uma arquitetura em camadas, seguindo o padrão MVC (Model-View-Controller), com foco em separação de responsabilidades, manutenibilidade e extensibilidade.

## Objetivos

- Criar uma API para operações de vendas online
- Implementar um sistema que siga as melhores práticas de desenvolvimento
- Fornecer endpoints RESTful para gerenciamento de produtos, clientes e pedidos
- Garantir a manutenibilidade e extensibilidade do código através de uma arquitetura em camadas
- Estruturar as pastas do projeto de forma pertinente a separação em camadas e responsabilidades
- Facilitar a integração com sistemas frontend e/ou outros serviços externos
- Permitir o desenvolvimento, teste e manutenção independentes de cada camada do sistema

## Arquitetura da API

O projeto foi desenvolvido utilizando arquitetura em camadas, seguindo o padrão arquitetural MVC (Model-View-Controller), complementado por uma abordagem de separação de responsabilidades.





## Desenvolvimento da Solução do Desafio Final

### 1. Pré-requisitos:

- **.NET SDK:** .NET 9.0.
- **SQL Server:** Banco de dados SQL Server (LocalDB).
- **Visual Studio 2022:** Para desenvolvimento e execução.
- **Entity Framework Core.**

### 2. Criação do projeto

### 3. Configurar a Connection String

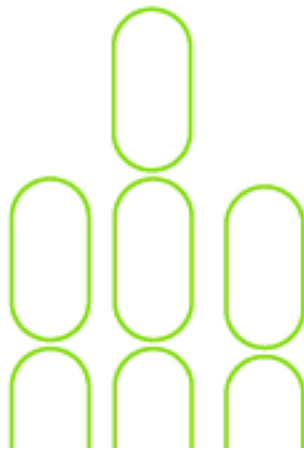
### 4. Aplicar Migrações ao Banco de Dados:

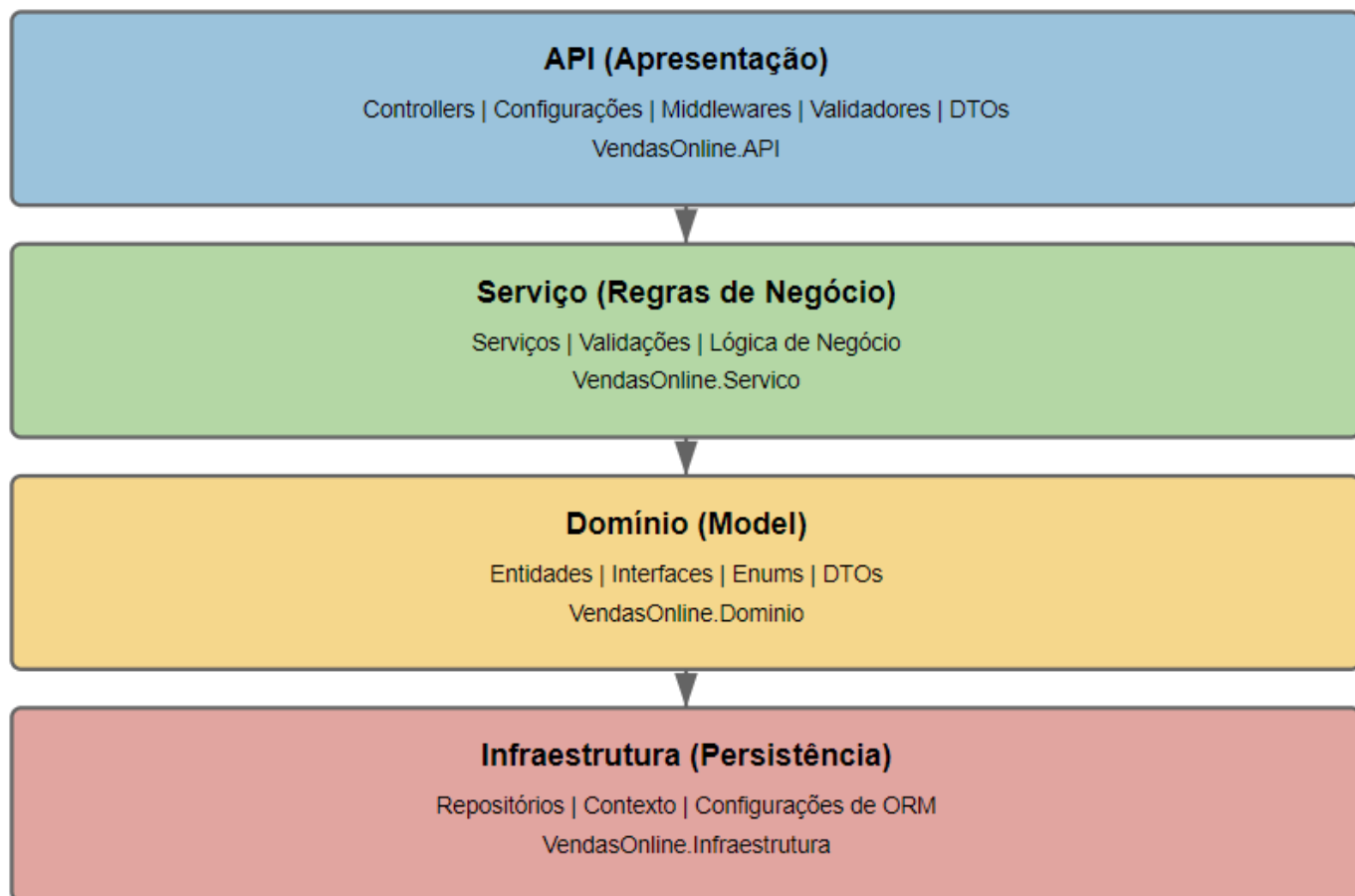
### 5. Executar o Projeto

### 6. Acessar a API

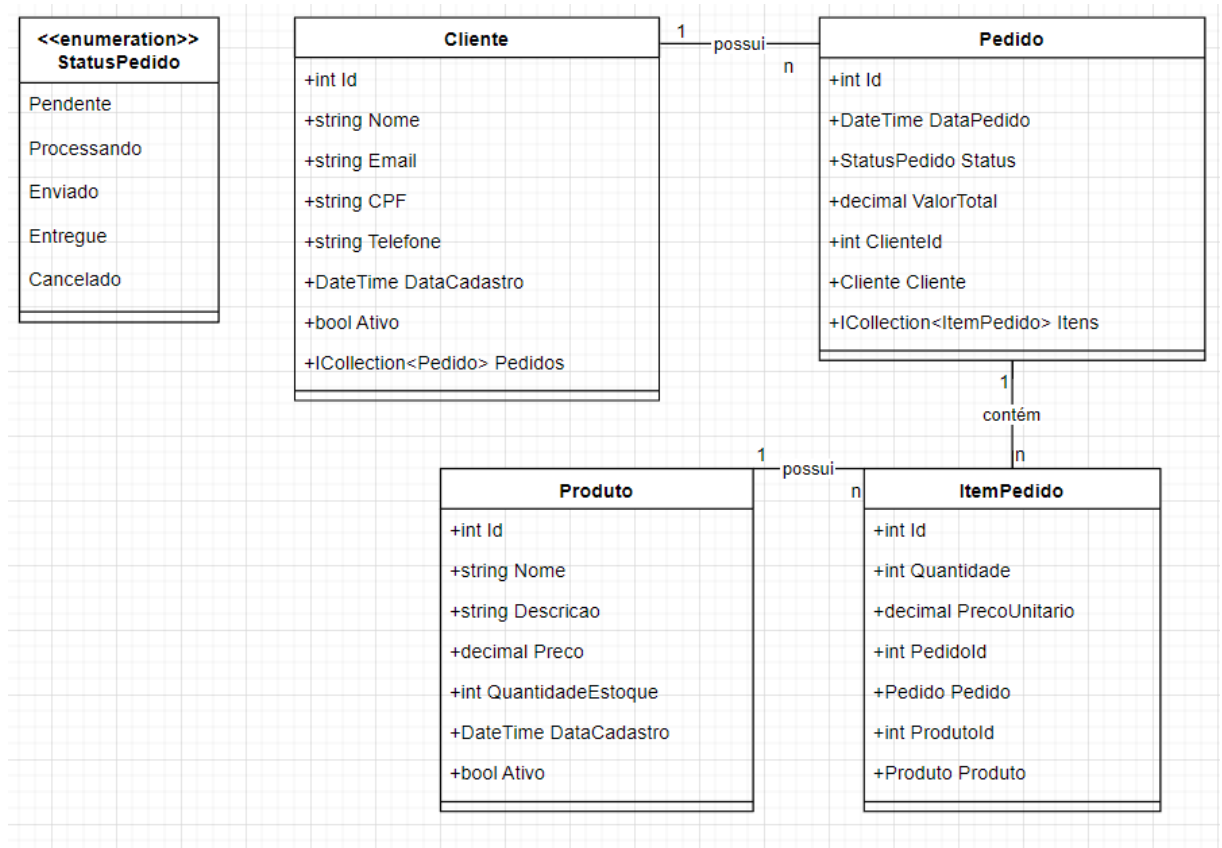
### 7. Testar os Endpoints

## Camadas do projeto (Modelo C4)

- **Camada de Apresentação (VendasOnline.API)**
    - Expõe endpoints RESTful
    - Recebe requisições dos clientes
    - Delega o processamento para a camada de serviço
    - Retorna respostas adequada
  - **Camada de Aplicação (VendasOnline.Servico)**
    - Implementa regras de negócio complexas
    - Orquestra operações entre as camadas
    - Valida dados
  - **Camada de Domínio (VendasOnline.Dominio)**
    - Contém as entidades, contratos e regras de negócio
    - Define as estruturas de dados e interfaces usadas em toda a aplicação
  - **Camada de Infraestrutura (VendasOnline.Infraestrutura)**
    - Implementa o acesso a dados e serviços externos
    - Contém repositórios e contexto de banco de dados
- 



## Diagrama de Classes



## Estrutura de pastas

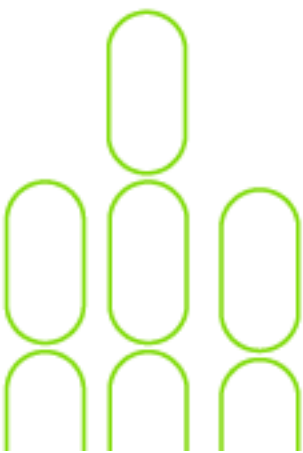
```
VendasOnline/
├── VendasOnline.sln
├── VendasOnline.API/
│   ├── Controllers/
│   │   ├── ProdutosController.cs
│   │   ├── ClientesController.cs
│   │   └── PedidosController.cs
│   ├── Configuracoes/
│   │   ├── AutoMapperConfig.cs
│   │   ├── DependencyInjectionConfig.cs
│   │   └── SwaggerConfig.cs
│   ├── Middlewares/
│   │   └── ExceptionMiddleware.cs
│   ├── Validadores/
│   │   └── ProdutoValidator.cs
│   ├── Program.cs
│   ├── appsettings.json
│   └── appsettings.Development.json
├── VendasOnline.Dominio/
│   ├── Entidades/
│   │   ├── Produto.cs
│   │   ├── Cliente.cs
│   │   ├── Pedido.cs
│   │   └── ItemPedido.cs
│   ├── Interfaces/
│   │   ├── IRepositoryBase.cs
│   │   ├── IRepositoryProduto.cs
│   │   ├── IRepositoryCliente.cs
│   │   ├── IRepositoryPedido.cs
│   │   ├── IServicoBase.cs
│   │   ├── IServicoProduto.cs
│   │   ├── IServicoCliente.cs
│   │   └── IServicoPedido.cs
```



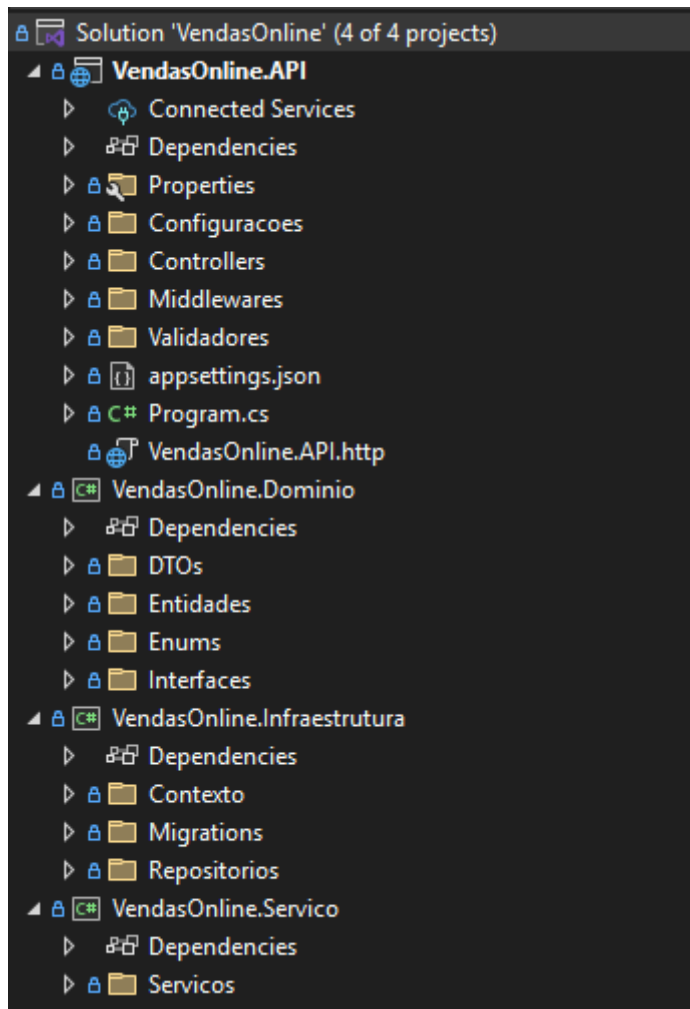


```
├── Enums/
│   └── StatusPedido.cs
├── DTOs/
│   ├── ProdutoDTO.cs
│   ├── ClienteDTO.cs
│   └── PedidoDTO.cs
├── VendasOnline.Infraestrutura/
│   ├── Contexto/
│   │   └── VendasOnlineDbContext.cs
│   ├── Repositorios/
│   │   ├── RepositorioBase.cs
│   │   ├── RepositorioProduto.cs
│   │   ├── RepositorioCliente.cs
│   │   └── RepositorioPedido.cs
│   └── Migrations/
│       └── [arquivos de migração gerados pelo EF Core]
└── VendasOnline.Servico/
    └── Servicos/
        ├── ServicoBase.cs
        ├── ServicoProduto.cs
        ├── ServicoCliente.cs
        └── ServicoPedido.cs
```

**Não possui view tradicional por ser uma API REST, mas os DTOs podem ser considerados como "views" dos dados.**



## Estrutura na prática



### VendasOnline.sln

Arquivo de solução do Visual Studio que agrupa todos os projetos relacionados. Ele mantém referências entre os projetos e é usado para gerenciar a solução como um todo.

### VendasOnline.API

Este é o projeto de entrada da aplicação que hospeda a API REST e expõe os endpoints HTTP.

## Controllers


Contém os controladores que recebem as requisições HTTP, processam os dados e retornam as respostas:

- **ProdutosController.cs:** Gerência operações CRUD para produtos
- **CientesController.cs:** Gerência operações CRUD para clientes
- **PedidosController.cs:** Gerência operações CRUD para pedidos

## Swagger UI

1. Clientes
2. Pedidos
3. Produtos





Pedidos	
GET	/api/Pedidos
POST	/api/Pedidos
GET	/api/Pedidos/{id}
PUT	/api/Pedidos/{id}
DELETE	/api/Pedidos/{id}
GET	/api/Pedidos/nome/{nome}
GET	/api/Pedidos/contar

Produtos	
GET	/api/Produtos
POST	/api/Produtos
GET	/api/Produtos/{id}
PUT	/api/Produtos/{id}
DELETE	/api/Produtos/{id}
GET	/api/Produtos/nome/{nome}
GET	/api/Produtos/contar

## Configurações

Contém classes de configuração da aplicação:

- **AutoMapperConfig.cs:** Configura o mapeamento entre entidades e DTOs
- **DependencyInjectionConfig.cs:** Registra serviços no container de injeção de dependência
- **SwaggerConfig.cs:** Configura a documentação Swagger da API

## Middlewares

Contém componentes que processam requisições HTTP:

- **ExceptionMiddleware.cs:** Intercepta exceções não tratadas e retorna respostas de erro padronizadas

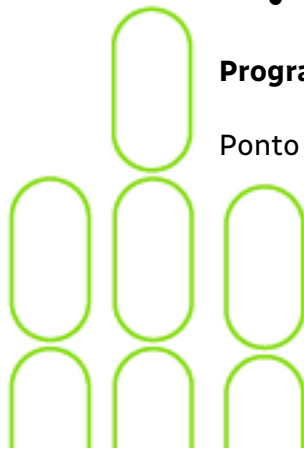
## Validadores

Contém classes de validação para os DTOs:

- **ProdutoValidator.cs:** Define regras de validação para produtos

## Program.cs

Ponto de entrada da aplicação que configura serviços e o pipeline HTTP.





## appsettings.json

Arquivo de configuração que contém strings de conexão e outras configurações.

## VendasOnline.Dominio

Este projeto contém as entidades de domínio, interfaces e DTOs, representando o núcleo da aplicação e suas regras de negócio.

### Entidades

Classes que representam os objetos de negócio:

- **Produto.cs:** Entidade que representa um produto para venda
- **Cliente.cs:** Entidade que representa um cliente
- **Pedido.cs:** Entidade que representa um pedido com itens
- **ItemPedido.cs:** Entidade que representa um item dentro de um pedido

### Interfaces

Contratos que definem comportamentos:

- **IRepositorioBase.cs:** Interface genérica para operações de repositório
- **IRepositorioProduto.cs, IRepositorioCliente.cs, IRepositorioPedido.cs:** Interfaces específicas para cada entidade
- **IServicoBase.cs:** Interface genérica para serviços
- **IServicoProduto.cs, IServicoCliente.cs, IServicoPedido.cs:** Interfaces específicas para cada serviço

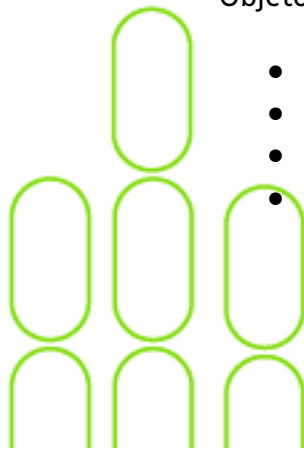
### Enums

Enumerações utilizadas no domínio:

- **StatusPedido.cs:** Define os possíveis estados de um pedido (Pendente, Processando, etc.)

### DTOs

Objetos de transferência de dados usados para comunicação entre camadas:

- **ProdutoDTO.cs:** DTO para transferência de dados de produto
  - **ClienteDTO.cs:** DTO para transferência de dados de cliente
  - **PedidoDTO.cs:** DTO para transferência de dados de pedido
  -
- 



## VendasOnline.Infraestrutura

Este projeto implementa o acesso a dados e serviços externos, isolando a camada de persistência.

### Contexto

Configuração de acesso ao banco de dados:

- **VendasOnlineDbContext.cs:** Classe que configura o Entity Framework e mapeia as entidades para tabelas

### Repositorios

Implementa o acesso a dados para as entidades:

- **RepositorioBase.cs:** Implementação genérica para operações CRUD básicas
- **RepositorioProduto.cs, RepositorioCliente.cs, RepositorioPedido.cs:** Implementações específicas para cada entidade

### Migrations

Contém arquivos de migração gerados pelo Entity Framework Core para controle de versão do banco de dados.

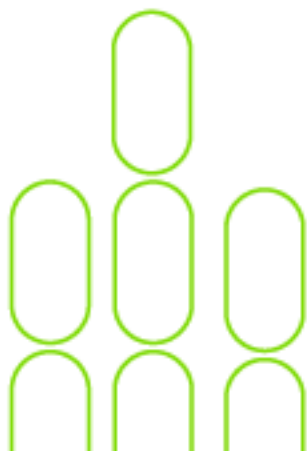
## VendasOnline.Servico

Este projeto implementa a lógica de negócio da aplicação, orquestrando chamadas entre repositórios e aplicando regras.

### Serviços

Implementa a lógica de negócio:

- **ServicoBase.cs:** Implementação genérica para operações de serviço comuns
- **ServicoProduto.cs, ServicoCliente.cs, ServicoPedido.cs:** Implementações específicas para cada entidade





## Fluxo de Operação na arquitetura

- Uma requisição HTTP chega ao controlador apropriado na **VendasOnline.API**
- O controlador valida os dados de entrada usando validadores
- O controlador chama o serviço apropriado em **VendasOnline.Servico**
- O serviço aplica regras de negócio e chama o repositório em **VendasOnline.Infraestrutura**
- O repositório interage com o banco de dados através do **VendasOnline.DbContext**
- Os dados são retornados através da cadeia: Repositório → Serviço → Controlador
- O controlador retorna uma resposta HTTP formatada para o cliente

## Benefícios dessa arquitetura

- **Separação de Responsabilidades:** Cada projeto tem uma responsabilidade clara
- **Testabilidade:** Facilidade para criar testes unitários para cada camada
- **Manutenibilidade:** Mudanças em uma camada não afetam diretamente outras camadas
- **Escalabilidade:** É possível escalar partes específicas da aplicação
- **Desacoplamento:** Uso de interfaces e injeção de dependência permite trocar implementações

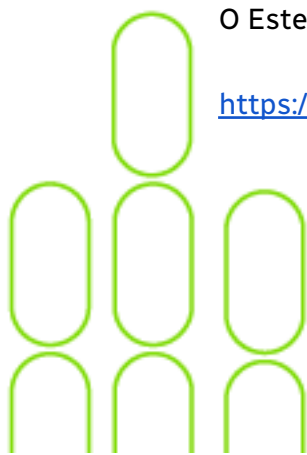
## Possíveis melhorias

- **Implementação de CQRS:** Para melhor separação entre operações de leitura e escrita
- **Autenticação e Autorização:** Implementar JWT ou OAuth2 para segurança
- **Caching:** Adicionar cache para consultas frequentes
- **Logging detalhado:** Implementar sistema de logging mais robusto
- **Paginação:** Adicionar suporte a paginação para listas grandes
- **Versionamento da API:** Implementar versionamento para evolução da API
- **Testes automatizados:** Adicionar testes unitários e de integração

## Github do projeto

O Este projeto foi desenvolvido e guardado no seguinte repositório Github:

<https://github.com/NFPN/xpe-desafio-final-arquitetura-software>





## Conclusão

### Resumo do Desempenho

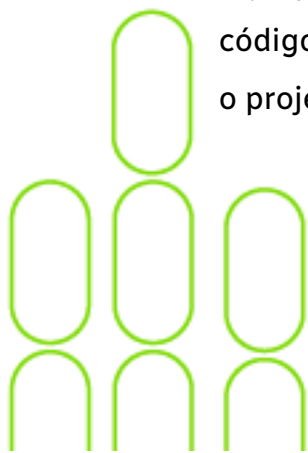
O desenvolvimento do desafio final da pós-graduação em Arquitetura de Software foi uma oportunidade valiosa para aplicar os conhecimentos adquiridos ao longo do curso, culminando na criação de uma API RESTful robusta para gerenciar operações de vendas online. A solução foi implementada utilizando .NET 9, seguindo o padrão MVC e uma arquitetura em camadas, com foco em manutenibilidade, escalabilidade e separação de responsabilidades. Todos os requisitos do enunciado foram atendidos, incluindo a implementação de endpoints CRUD para produtos, clientes e pedidos, validação de dados, tratamento de exceções, documentação via Swagger e persistência de dados com Entity Framework Core.

### Aplicação dos Conhecimentos

Os conceitos fundamentais de arquitetura de software foram aplicados de forma abrangente. O padrão MVC foi utilizado para organizar a camada de apresentação, enquanto os princípios SOLID (como Injeção de Dependência e Segregação de Interfaces) guiaram o design do código. Ferramentas como AutoMapper, FluentValidation e Entity Framework Core foram empregadas para otimizar o mapeamento de dados, validação de entrada e acesso ao banco de dados, respectivamente.

### Principais Dificuldades e Superações

Uma das principais dificuldades foi construir a ideia antes de implementar de fato o projeto, como desenvolvedor que está se aprofundando no assunto, a parte do código em si foi menos desafiadora. Porém consegui apesar de simples desenhar o projeto.







### **Resultados Obtidos**

A solução final atendeu a todos os requisitos funcionais e não funcionais do enunciado. A API suporta operações CRUD completas para as entidades Produto, Cliente e Pedido, com validação robusta e tratamento de erros padronizado. A documentação gerada pelo Swagger facilita a integração com sistemas externos, e a arquitetura em camadas permite fácil manutenção e extensão. Testes realizados com ferramentas como Postman confirmaram o funcionamento correto dos endpoints, com respostas HTTP apropriadas e dados persistidos no banco de dados.

### **Lições Aprendidas**

O desafio proporcionou um bom aprendizado, tanto técnicos quanto em termos de organização. A implementação de uma arquitetura em camadas reforçou a importância do desacoplamento e da modularidade no desenvolvimento de software. E a integração de ferramentas modernas destacou a necessidade de planejamento detalhado e documentação clara, habilidades essenciais para projetos complexos.

