PÓS-GRADUAÇÃO MIT ENGENHARIA DE SOFTWARE COM .NET

Projeto de Bloco: Engenharia de Software e Modelagem

Integrantes:

ALUNO: ALEXANDER SILVA

ALUNO: YURI ALVES

ALUNO: PEDRO NOVAES

Projeto de Bloco: Engenharia de Software e Modelagem

MODELAGEM ESTRATÉGICA

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

https://www.eduardopires.net.br/2016/08/ddd-nao-e-arquitetura-em-camadas

Entender a fundo o negócio

Qual problema vai resolver?;
O que vai implementar?;
Para que serve?;
Quem vai atender?
Quando implementar o DDD?

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Pontos Chave

Uma linguagem onipresente, universal de um negócio dentro da empresa, compartilhada por todas as partes envolvidas no projeto, ou seja, termos específicos do domínio.

É uma parte da empresa ou do domínio do negócio que possui elementos ou conceitos com significados bem definidos, com linguagem própria, arquitetura e implementação específica.

É um mapa que representa todos os contextos mapeados através dos elementos da linguagem ubíqua, bem como seus subdomínios e relacionamentos.

Projeto de Bloco: Engenharia de Software e Modelagem

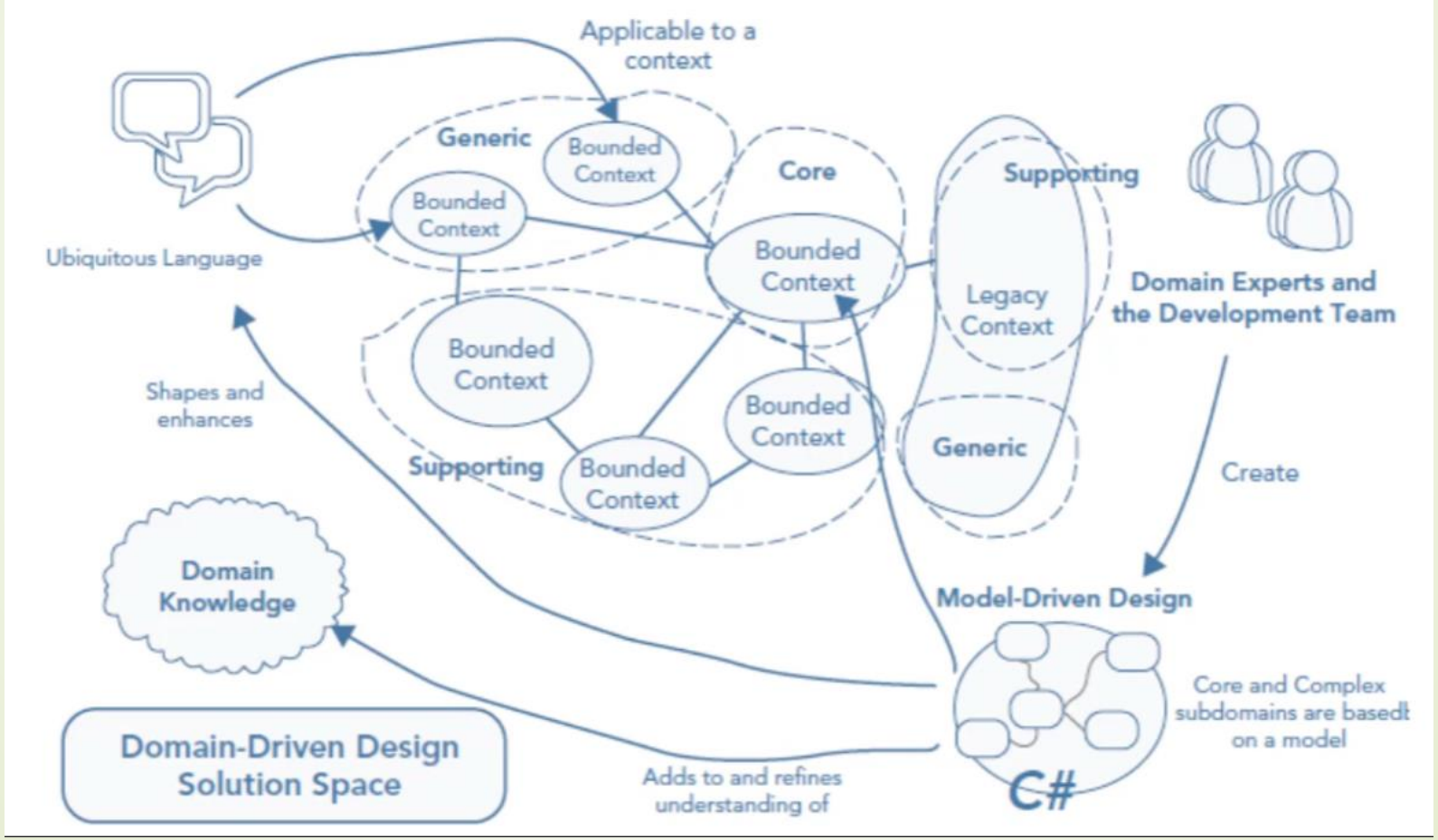
Aplicar os conceitos de DDD

Atenção!!!
Deixar de extrair a linguagem ubíqua é um dos piores erros.

Primeiro passo extrair a linguagem ubíqua

Projeto de Bloco: Engenharia de Software e Modelagem

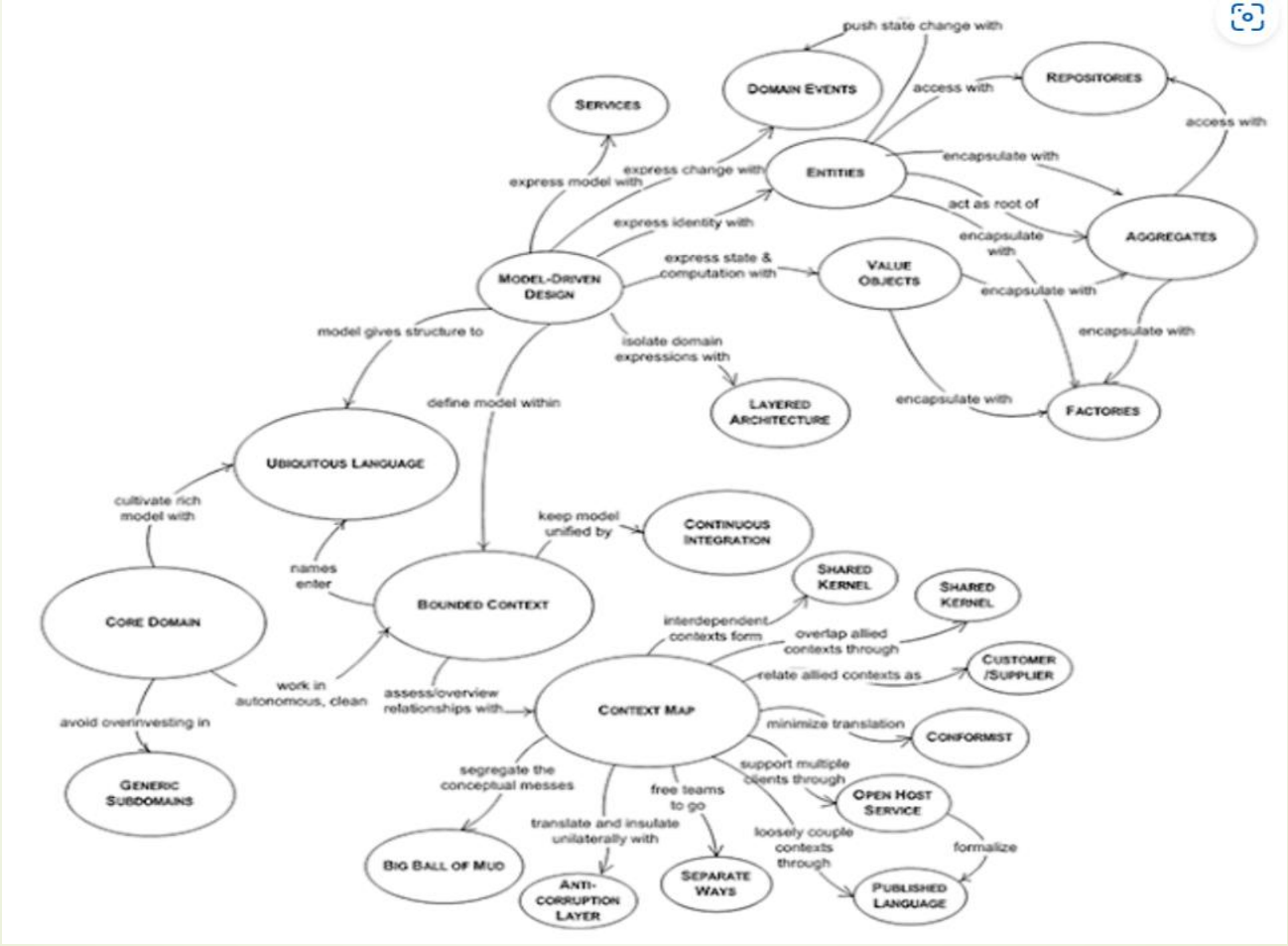
Aplicar os conceitos de DDD



Projeto de Bloco: Engenharia de Software e Modelagem

Mapa de Aprendizado

Aplicar os conceitos de DDD



Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

- A loja virtual exibirá um catálogo de produtos de diversas categorias.
- Um cliente pode realizar um pedido contendo 1 ou N produtos.
- A loja realizará as vendas através de pagamento por cartão de crédito.
- O cliente irá realizar o seu cadastro para poder fazer pedidos.
- O cliente irá confirmar o pedido, endereço de entrega, escolher o tipo de frete e realizar o pagamento.
- Após o pagamento o pedido mudará de status conforme resposta da transação via cartão.
- Ocorrerá a emissão da nota fiscal logo após a confirmação de pagamento do pedido.

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

- A loja virtual exibirá um catálogo de produtos de diversas categorias.
- Um cliente pode realizar um pedido contendo 1 ou N produtos.
- A loja realizará as vendas através de pagamento por cartão de crédito.
- O cliente irá realizar o seu cadastro para poder fazer pedidos.
- O cliente irá confirmar o pedido, endereço de entrega, escolher o tipo de frete e realizar o pagamento.
- Após o pagamento o pedido mudará de status conforme resposta da transação via cartão.
- Ocorrerá a emissão da nota fiscal logo após a confirmação de pagamento do pedido.

Projeto de Bloco: Engenharia de Software e Modelagem

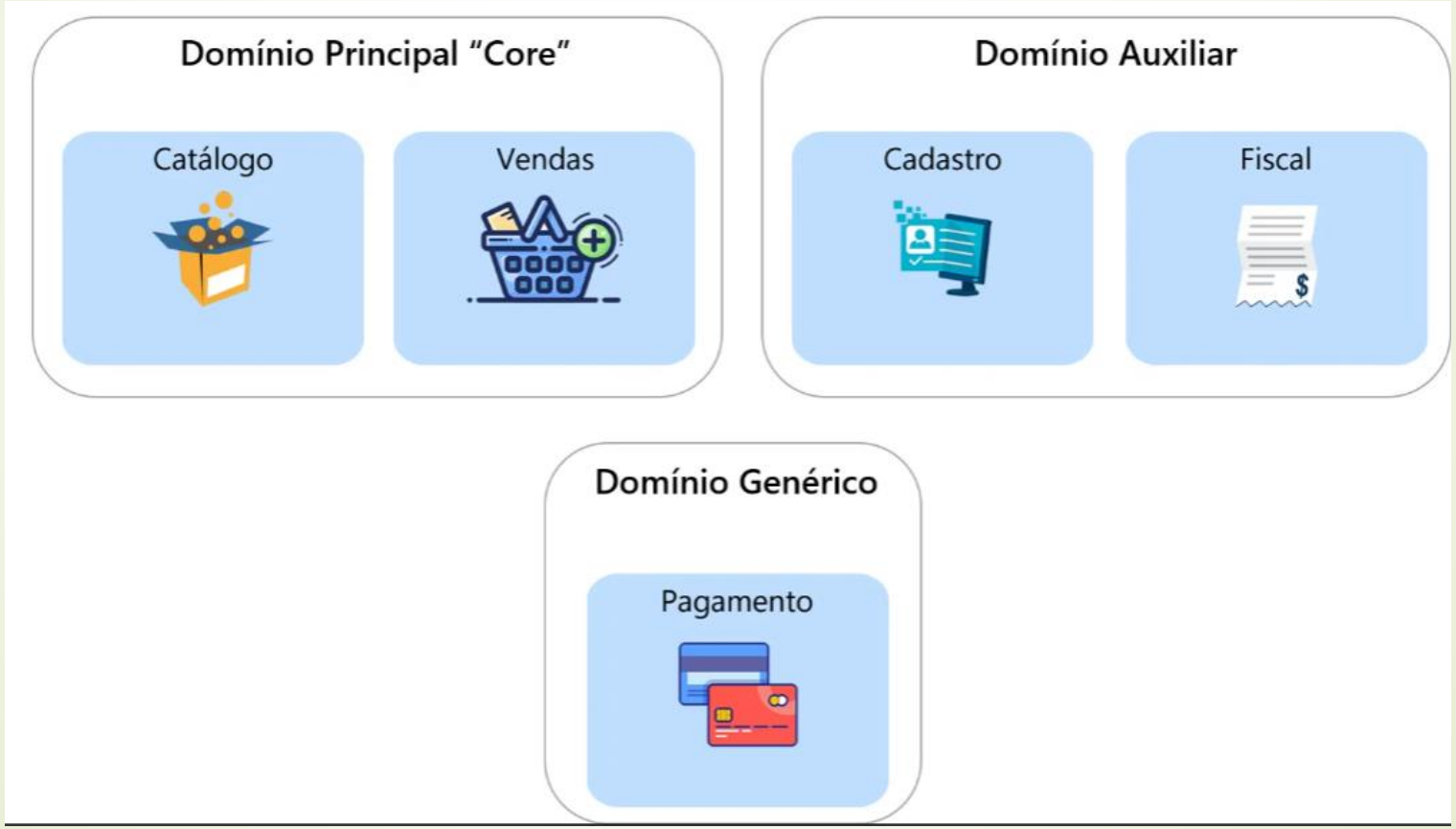
Aplicar os conceitos de DDD

- A loja virtual exibirá um catálogo de produtos de diversas categorias.
- Um cliente pode realizar um pedido contendo 1 ou N produtos.
- A loja realizará as vendas através de pagamento por cartão de crédito.
- O cliente irá realizar o seu cadastro para poder fazer pedidos.
- O cliente irá confirmar o pedido, endereço de entrega, escolher o tipo de frete e realizar o pagamento.
- Após o pagamento o pedido mudará de status conforme resposta da transação via cartão.
- Ocorrerá a emissão da nota fiscal logo após a confirmação de pagamento do pedido.

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Tipos de Domínios



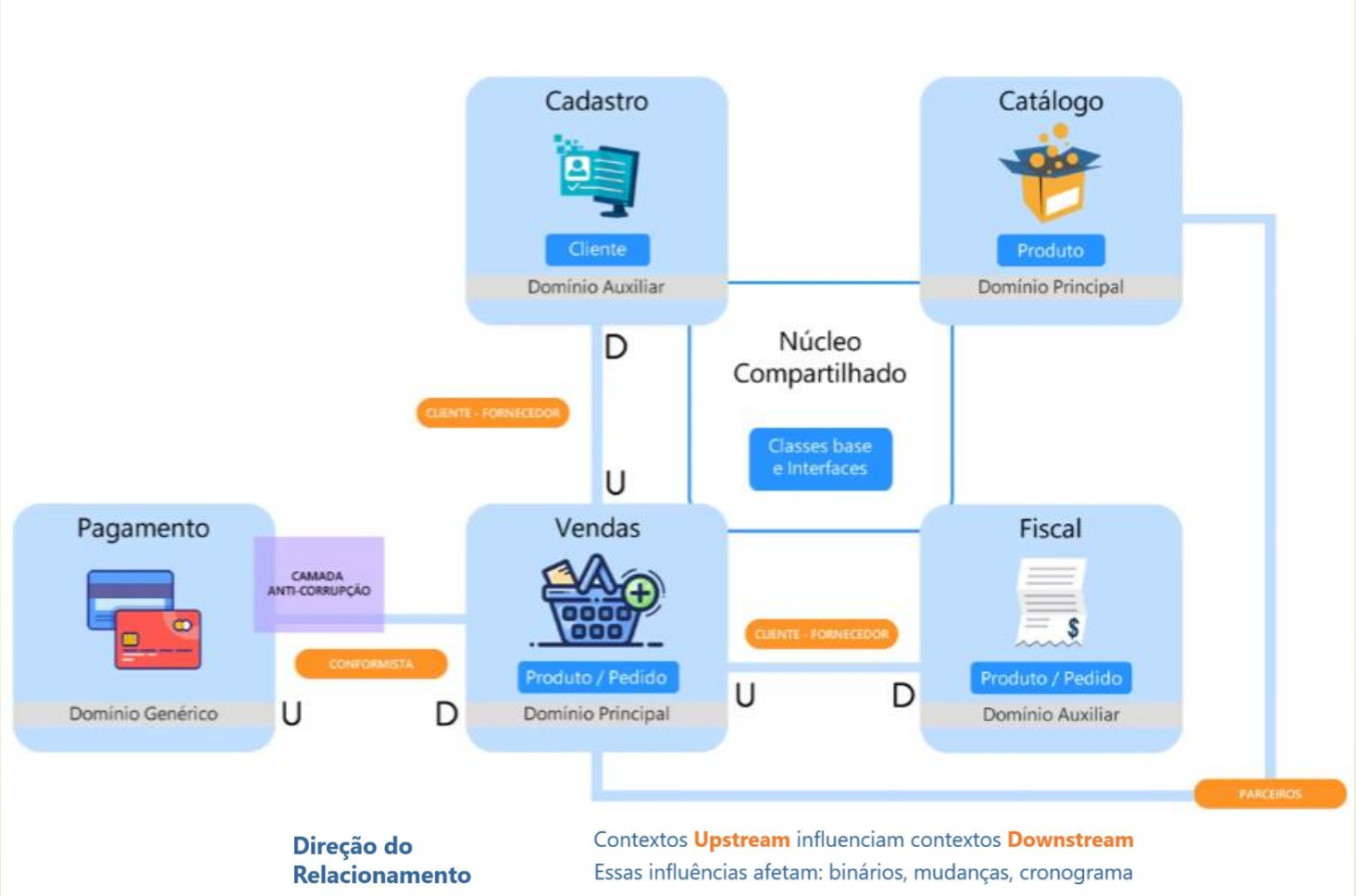
Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

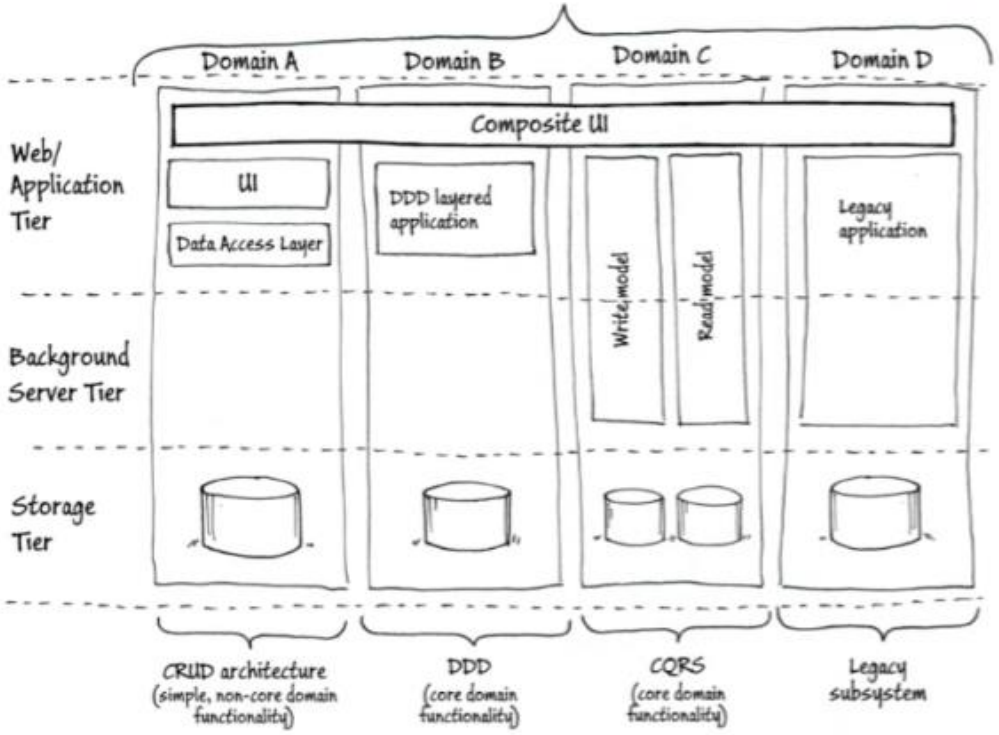
Mapa de Contextos

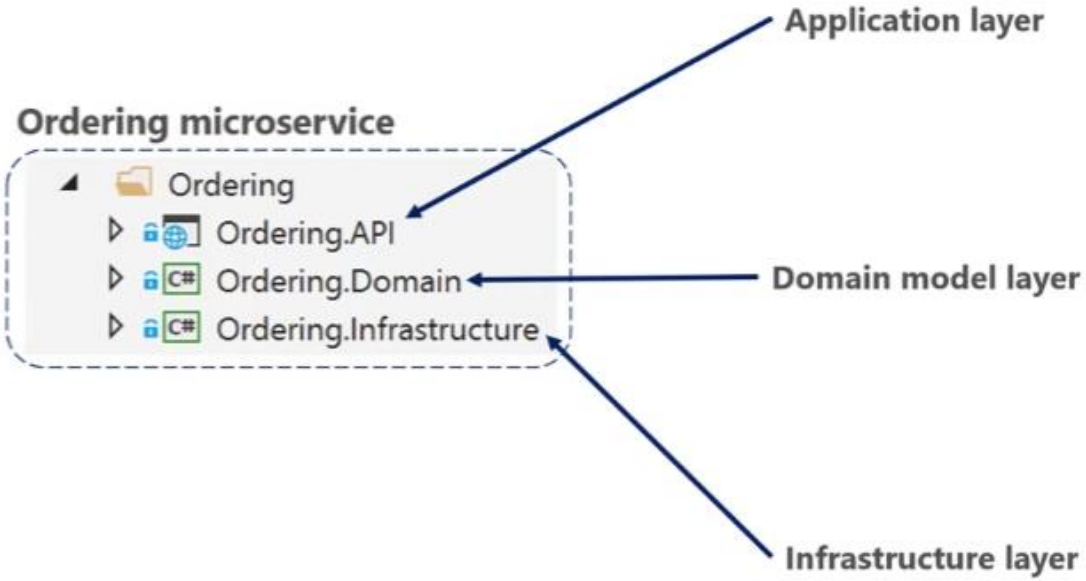


Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Definir arquitetura dos contextos

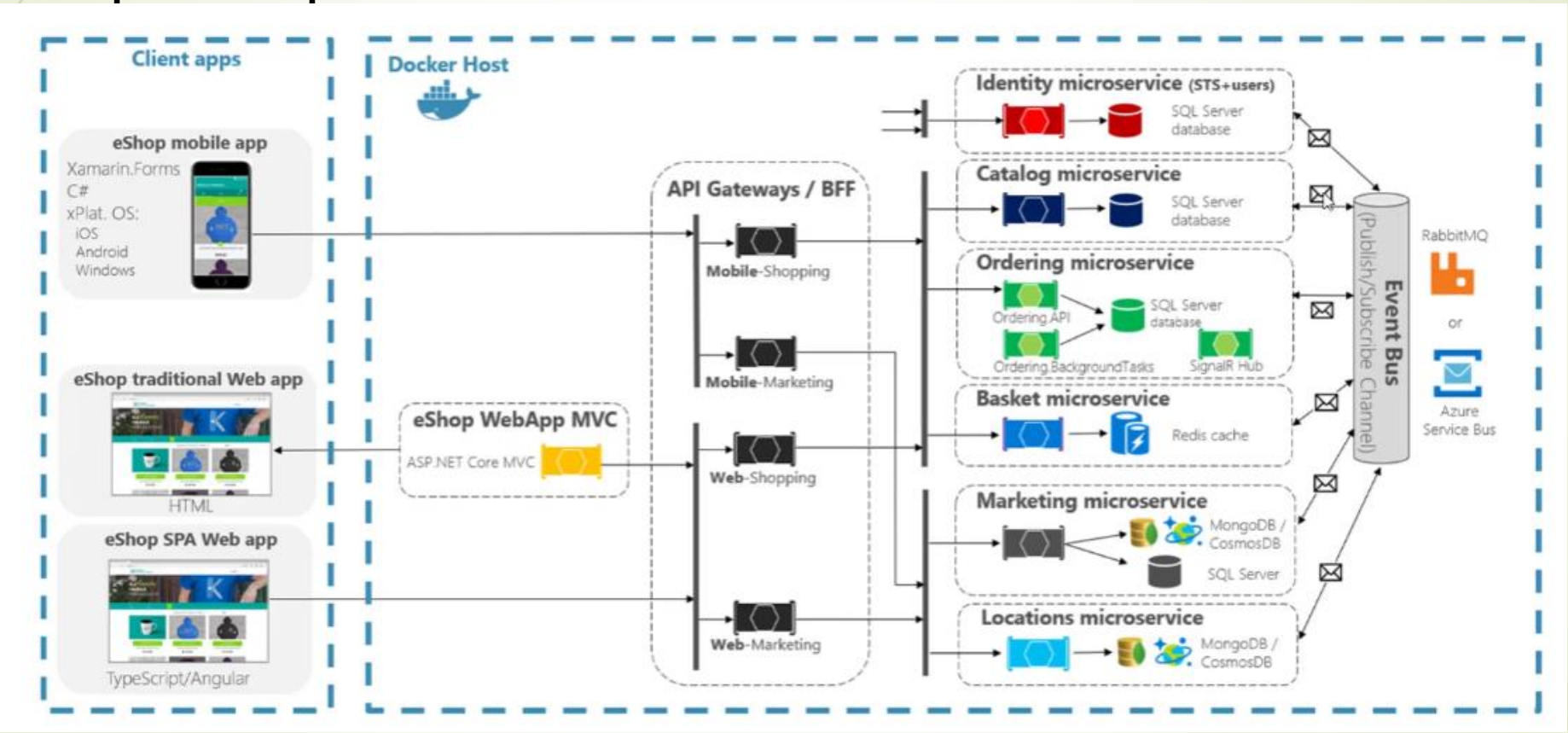




Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Exemplo de arquitetura dos contextos



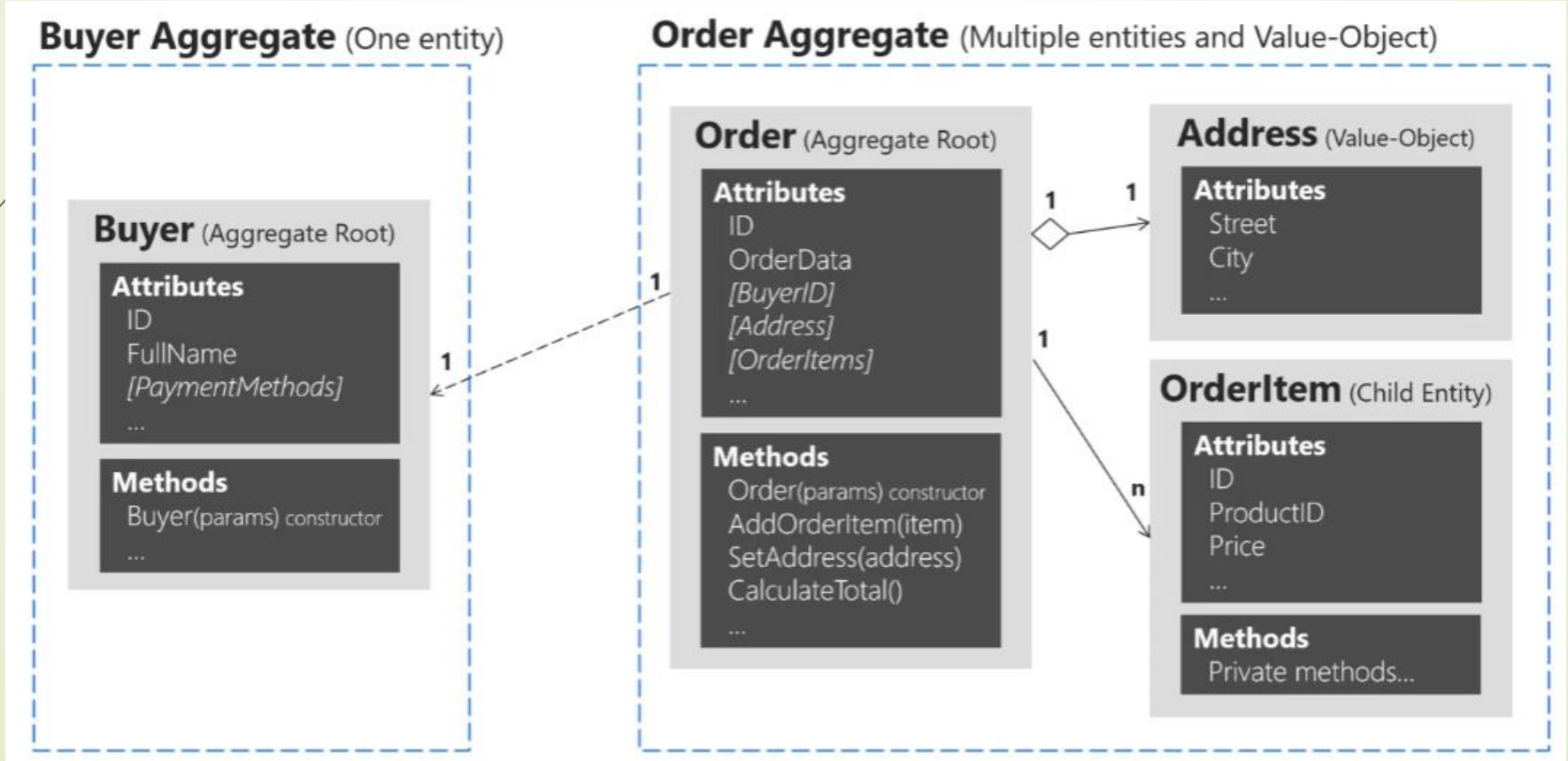
Projeto de Bloco: Engenharia de Software e Modelagem

MODELAGEM TÁTICA

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Agregação



Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

Interface de marcação

```
1  namespace NerdStore.Core.DomainObjects
2  {
       4 referências
3      public interface IAggregateRoot { }
4  }
```

```
4  namespace NerdStore.Core.Data
5  {
       3 referências
6      public interface IRepository<T> : IDisposable where T : IAggregateRoot
7      {
           18 referências
8          IUnitOfWork UnitOfWork { get; }
9      }
10 }
```

Raiz de agregação

```
7  namespace NerdStore.Vendas.Domain
8  {
       21 referências
9      public class Pedido : Entity, IAggregateRoot
10     {
           3 referências
11         public int Codigo { get; private set; }
           8 referências
12         public Guid ClienteId { get; private set; }
           3 referências
13         public Guid? VoucherId { get; private set; }
           3 referências
14         public bool VoucherUtilizado { get; private set; }
           4 referências
15         public decimal Desconto { get; private set; }
           8 referências
16         public decimal ValorTotal { get; private set; }
           1 referência
17         public DateTime DataCadastro { get; private set; }
           8 referências
18         public PedidoStatus PedidoStatus { get; private set; }
19
20         private readonly List<PedidoItem> _pedidoItems;
           10 referências
21         public IReadOnlyCollection<PedidoItem> PedidoItems => _pedidoItems;
22
23         // EF Rel.
           9 referências
24         public Voucher Voucher { get; private set; }
25
```

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

```
using System;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Catalogo.Domain
{
    28 referências
    public class Produto : Entity, IAggregateRoot
    {
        4 referências
        public Guid CategoriaId { get; private set; }
        4 referências
        public string Nome { get; private set; }
        4 referências
        public string Descricao { get; private set; }
        3 referências
        public bool Ativo { get; private set; }
        2 referências
        public decimal Valor { get; private set; }
        1 referência
        public DateTime DataCadastro { get; private set; }
        3 referências
        public string Imagem { get; private set; }
        5 referências
        public int QuantidadeEstoque { get; private set; }
        5 referências
        public Dimensoes Dimensoes { get; private set; }
        4 referências
        public Categoria Categoria { get; private set; }
```

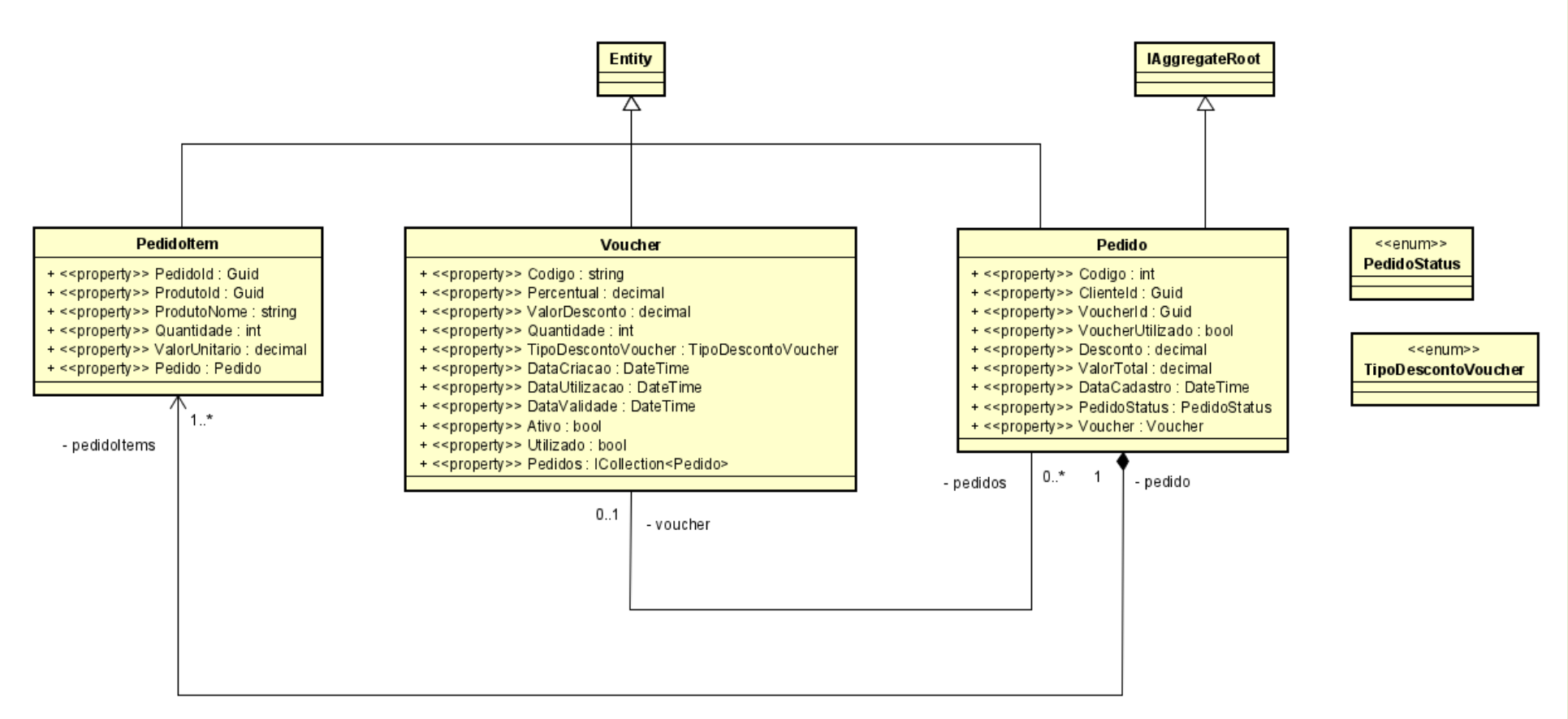
Objeto de valor

```
using NerdStore.Core.DomainObjects;

namespace NerdStore.Catalogo.Domain
{
    10 referências
    public class Dimensoes
    {
        4 referências
        public decimal Altura { get; private set; }
        4 referências
        public decimal Largura { get; private set; }
        4 referências
        public decimal Profundidade { get; private set; }

        7 referências
        public Dimensoes(decimal altura, decimal largura, decimal profundidade)
        {
            Validacoes.ValidarSeMenorQue(valor: altura, minimo: 1, mensagem: "O campo Altura não pode ser menor ou igual a 0");
            Validacoes.ValidarSeMenorQue(valor: largura, minimo: 1, mensagem: "O campo Largura não pode ser menor ou igual a 0");
            Validacoes.ValidarSeMenorQue(valor: profundidade, minimo: 1, mensagem: "O campo Profundidade não pode ser menor ou igual a 0");

            Altura = altura;
            Largura = largura;
            Profundidade = profundidade;
        }

        1 referência
        public string DescricaoFormatada()
        {
            return $"LxAxP: {Largura} x {Altura} x {Profundidade}";
        }

        0 referências
        public override string ToString()
        {
            return DescricaoFormatada();
        }
    }
}
```

Projeto de Bloco: Engenharia de Software e Modelagem

Aplicar os conceitos de DDD

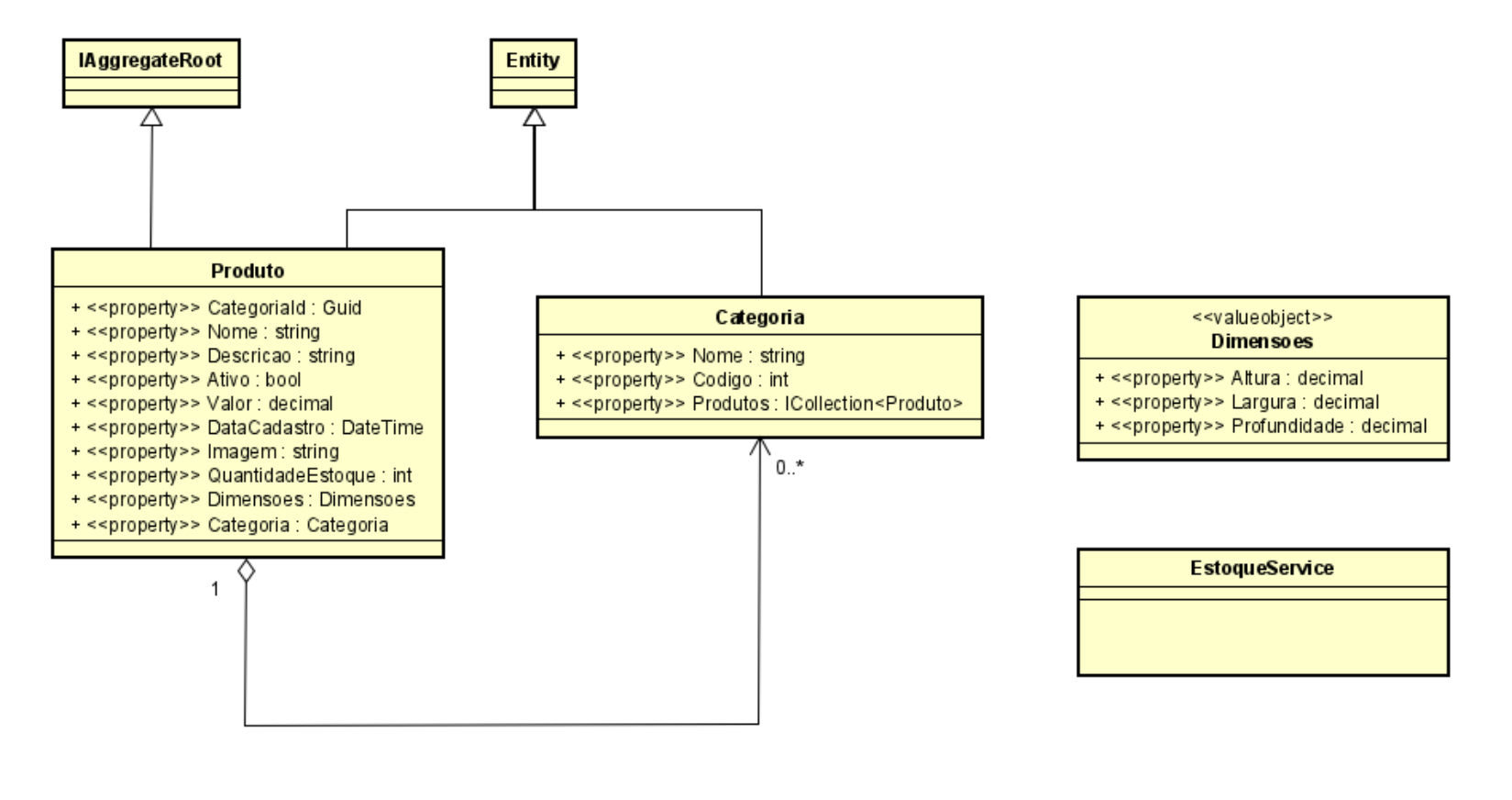
Modelo Conceitual – Contexto Vendas



Projeto de Bloco: Engenharia de Software e Modelagem

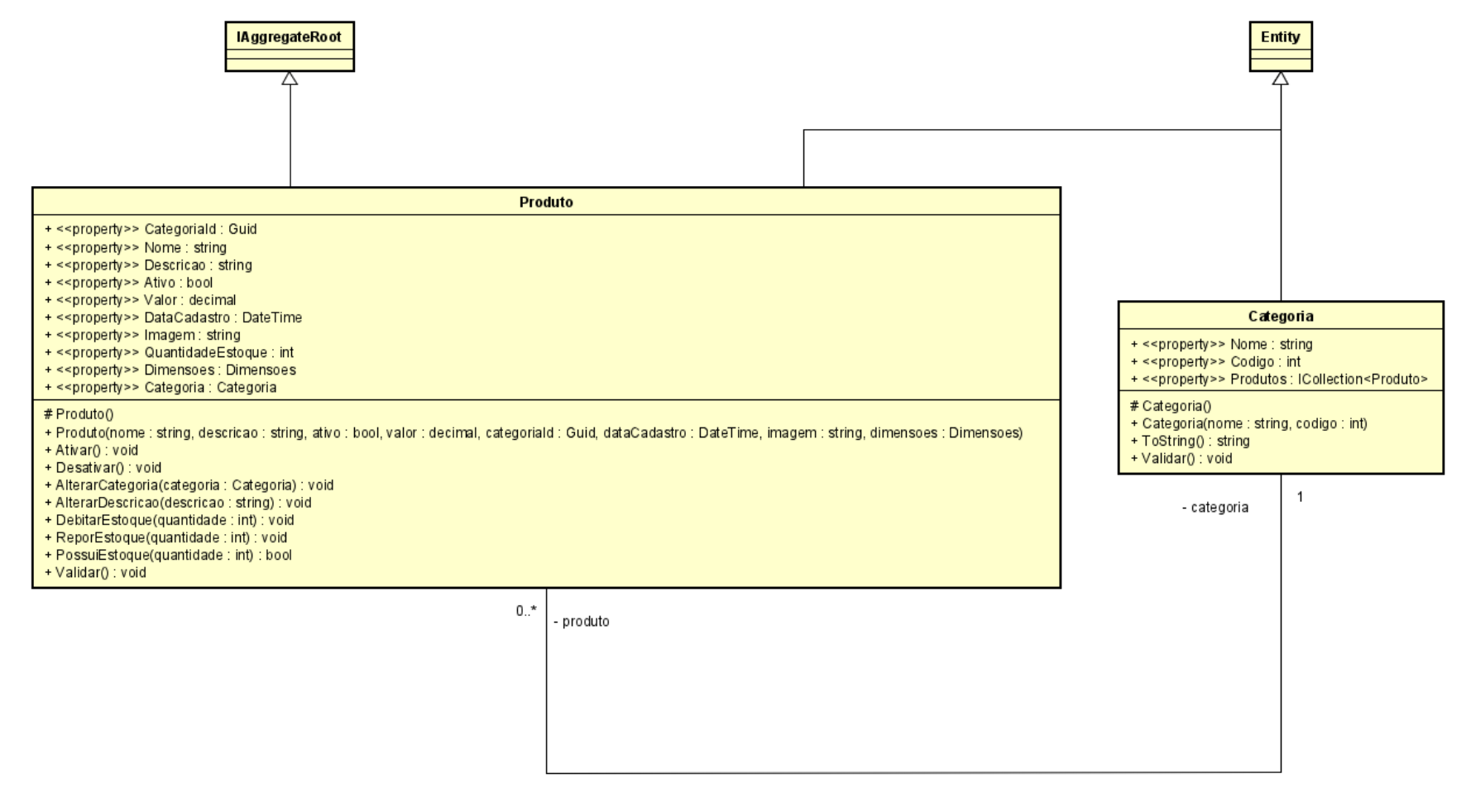
Aplicar os conceitos de DDD

Modelo Conceitual – Contexto Catálogo



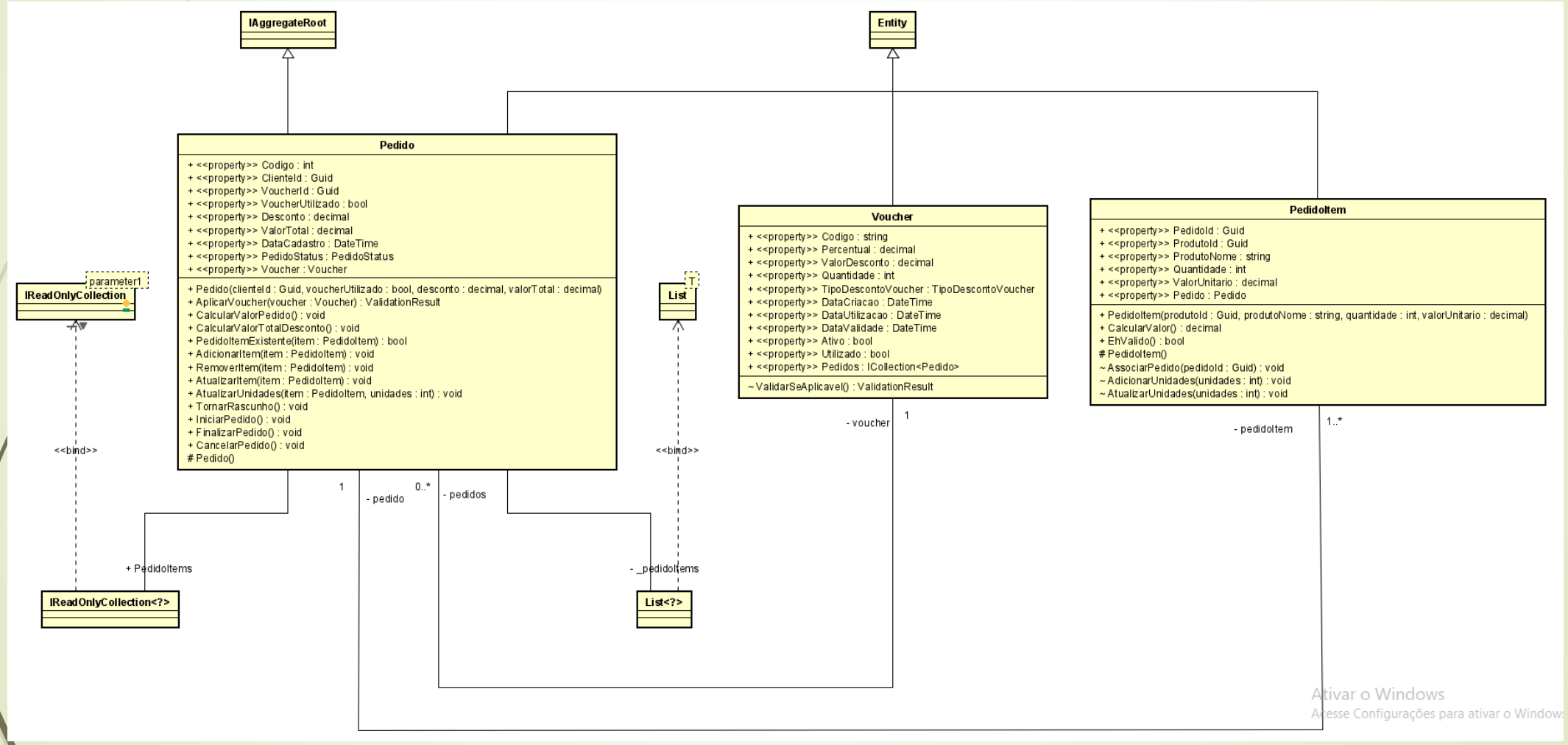
Projeto de Bloco: Engenharia de Software e Modelagem

Diagrama de Classe – Contexto Catálogo



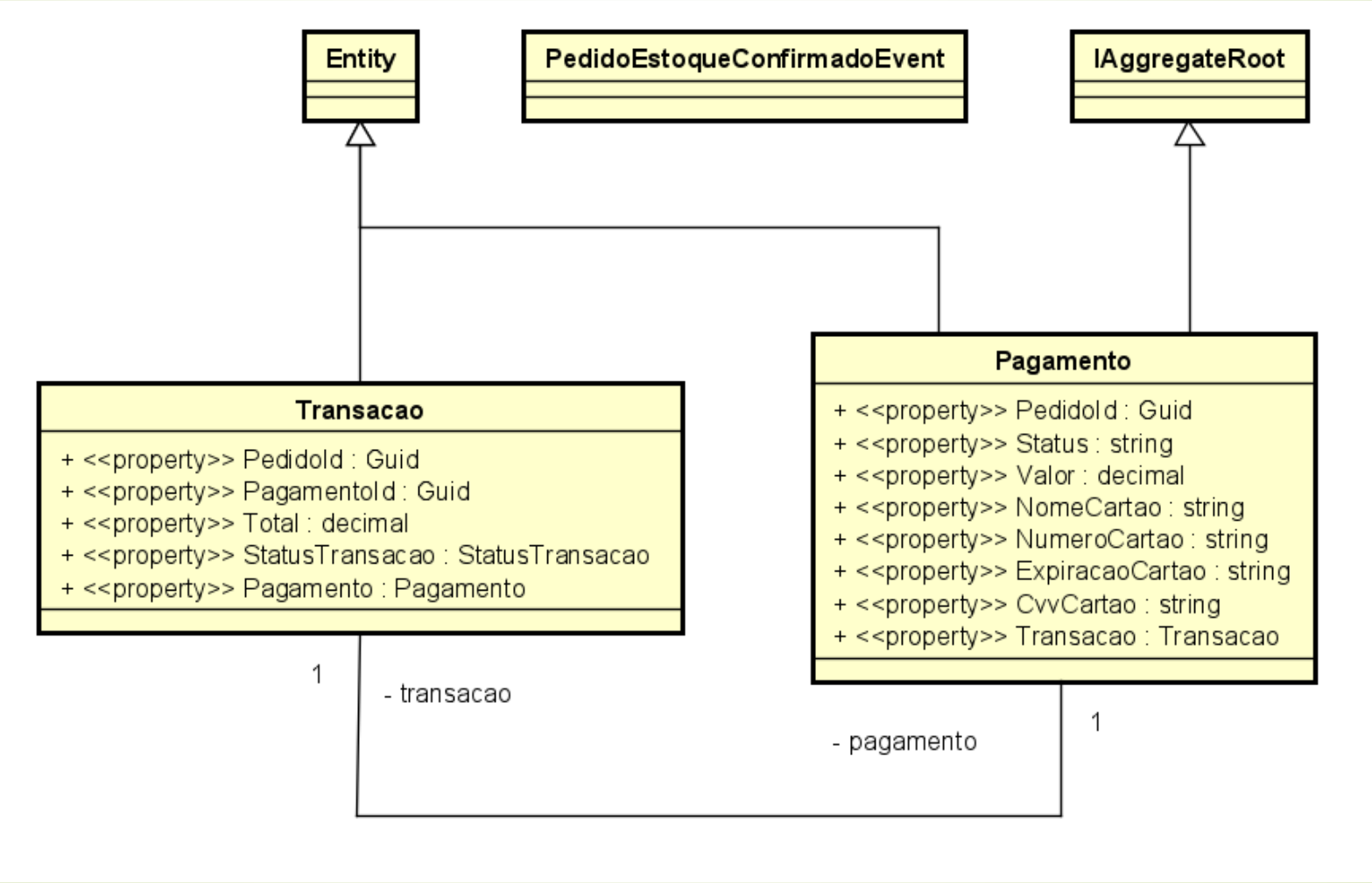
Projeto de Bloco: Engenharia de Software e Modelagem

Diagrama de Classe – Contexto Vendas



Projeto de Bloco: Engenharia de Software e Modelagem

Diagrama de Classe – Contexto Pagamentos



Projeto de Bloco: Engenharia de Software e Modelagem

Rel. OneToMany – Contexto Catálogo

```csharp
using System;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Catalogo.Domain
{
    public class Produto : Entity, IAggregateRoot
    {
        public Guid CategoriaId { get; private set; }
        public string Nome { get; private set; }
        public string Descricao { get; private set; }
        public bool Ativo { get; private set; }
        public decimal Valor { get; private set; }
        public DateTime DataCadastro { get; private set; }
        public string Imagem { get; private set; }
        public int QuantidadeEstoque { get; private set; }
        public Dimensoes Dimensoes { get; private set; }
        public Categoria Categoria { get; private set; }

        protected Produto() { }
        public Produto(string nome,
                       string descricao,
                       bool ativo,
                       decimal valor,
                       Guid categoriaId,
                       DateTime dataCadastro,
                       string imagem,
                       Dimensoes dimensoes)
        {
            CategoriaId = categoriaId;
            Nome = nome;
            Descricao = descricao;
            Ativo = ativo;
            Valor = valor;
            DataCadastro = dataCadastro;
            Imagem = imagem;
            Dimensoes = dimensoes;

            Validar();
        }
```

```csharp
using System.Collections.Generic;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Catalogo.Domain
{
    public class Categoria : Entity
    {
        public string Nome { get; private set; }
        public int Codigo { get; private set; }

        // EF Relation
        public ICollection<Produto> Produtos { get; set; }

        protected Categoria() { }

        public Categoria(string nome, int codigo)
        {
            Nome = nome;
            Codigo = codigo;

            Validar();
        }

        public override string ToString()
        {
            return $"{Nome} - {Codigo}";
        }

        public void Validar()
        {
            Validacoes.ValidarSeVazio(Nome, "O campo Nome da categoria não pode estar vazio");
            Validacoes.ValidarSeIgual(Codigo, 0, "O campo Codigo não pode ser 0");
        }
    }
}
```

Projeto de Bloco: Engenharia de Software e Modelagem

Rel. OneToMany – Contexto Vendas

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using FluentValidation.Results;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Vendas.Domain
{
    public class Pedido : Entity, IAggregateRoot
    {
        public int Codigo { get; private set; }
        public Guid ClienteId { get; private set; }
        public Guid? VoucherId { get; private set; }
        public bool VoucherUtilizado { get; private set; }
        public decimal Desconto { get; private set; }
        public decimal ValorTotal { get; private set; }
        public DateTime DataCadastro { get; private set; }
        public PedidoStatus PedidoStatus { get; private set; }

        private readonly List<PedidoItem> _pedidoItems;
        public IReadOnlyCollection<PedidoItem> PedidoItems => _pedidoItems;

        // EF Rel.
        public Voucher Voucher { get; private set; }

        public Pedido(Guid clienteId,
                      bool voucherUtilizado,
                      decimal desconto,
                      decimal valorTotal)
        {
            ClienteId = clienteId;
            VoucherUtilizado = voucherUtilizado;
            Desconto = desconto;
            ValorTotal = valorTotal;
            _pedidoItems = new List<PedidoItem>();
        }

        protected Pedido()
        {
            _pedidoItems = new List<PedidoItem>();
        }
```

```cs
using System;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Vendas.Domain
{
    public class PedidoItem : Entity
    {
        public Guid PedidoId { get; private set; }
        public Guid ProdutoId { get; private set; }
        public string ProdutoNome { get; private set; }
        public int Quantidade { get; private set; }
        public decimal ValorUnitario { get; private set; }

        // EF Rel.
        public Pedido Pedido { get; set; }

        public PedidoItem(Guid produtoId,
                          string produtoNome,
                          int quantidade,
                          decimal valorUnitario)
        {
            ProdutoId = produtoId;
            ProdutoNome = produtoNome;
            Quantidade = quantidade;
            ValorUnitario = valorUnitario;
        }

        protected PedidoItem() { }

        internal void AssociarPedido(Guid pedidoId)
        {
            PedidoId = pedidoId;
        }

        public decimal CalcularValor()
        {
            return Quantidade * ValorUnitario;
        }
```

```cs
using System;
using System.Collections.Generic;
using FluentValidation;
using FluentValidation.Results;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Vendas.Domain
{
    public class Voucher : Entity
    {
        public string Codigo { get; private set; }
        public decimal? Percentual { get; private set; }
        public decimal? ValorDesconto { get; private set; }
        public int Quantidade { get; private set; }
        public TipoDescontoVoucher TipoDescontoVoucher { get; private set; }
        public DateTime DataCriacao { get; private set; }
        public DateTime? DataUtilizacao { get; private set; }
        public DateTime DataValidade { get; private set; }
        public bool Ativo { get; private set; }
        public bool Utilizado { get; private set; }

        // EF Rel.
        public ICollection<Pedido> Pedidos { get; set; }

        internal ValidationResult ValidarSeAplicavel()
        {
            return new VoucherAplicavelValidation().Validate(this);
        }
    }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Rel. OneToOne – Contexto Pagamentos

```
1  using System;
2  using NerdStore.Core.DomainObjects;
3
4  namespace NerdStore.Pagamentos.Business
5  {
       – referências
6      public class Pagamento : Entity, IAggregateRoot
7      {
           – referências
8          public Guid PedidoId { get; set; }
           – referências
9          public string Status { get; set; }
           – referências
10         public decimal Valor { get; set; }
11
           – referências
12         public string NomeCartao { get; set; }
           – referências
13         public string NumeroCartao { get; set; }
           – referências
14         public string ExpiracaoCartao { get; set; }
           – referências
15         public string CvvCartao { get; set; }
16
17         // EF. Rel.
           – referências
18         public Transacao Transacao { get; set; }
19     }
20 }
21
```

```
1  using System;
2  using NerdStore.Core.DomainObjects;
3
4  namespace NerdStore.Pagamentos.Business
5  {
       – referências
6      public class Transacao : Entity
7      {
           – referências
8          public Guid PedidoId { get; set; }
           – referências
9          public Guid PagamentoId { get; set; }
           – referências
10         public decimal Total { get; set; }
           – referências
11         public StatusTransacao StatusTransacao { get; set; }
12
13         // EF. Rel.
           – referências
14         public Pagamento Pagamento { get; set; }
15     }
16 }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Relacionamento Extends

Projeto de Bloco: Engenharia de Software e Modelagem

Herança – Contexto Catálogo

```
using System;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Catalogo.Domain
{
    public class Produto : Entity, IAggregateRoot
    {
        public Guid CategoriaId { get; private set; }
        public string Nome { get; private set; }
        public string Descricao { get; private set; }
        public bool Ativo { get; private set; }
        public decimal Valor { get; private set; }
        public DateTime DataCadastro { get; private set; }
        public string Imagem { get; private set; }
        public int QuantidadeEstoque { get; private set; }
        public Dimensoes Dimensoes { get; private set; }
        public Categoria Categoria { get; private set; }
```

```
using System;
using System.Collections.Generic;
using NerdStore.Core.Messages;

namespace NerdStore.Core.DomainObjects
{
    public abstract class Entity
    {
        public Guid Id { get; set; }

        private List<Event> _notificacoes;
        public IReadOnlyCollection<Event> Notificacoes => _notificacoes?.AsReadOnly();

        protected Entity()
        {
            Id = Guid.NewGuid();
        }

        public void AdicionarEvento(Event evento)
        {
            _notificacoes = _notificacoes ?? new List<Event>();
            _notificacoes.Add(evento);
        }

        public void RemoverEvento(Event eventItem)
        {
            _notificacoes?.Remove(eventItem);
        }

        public void LimparEventos()
        {
            _notificacoes?.Clear();
        }

        public override bool Equals(object obj)
        {
            var compareTo = obj as Entity;

            if (ReferenceEquals(this, compareTo)) return true;
            if (ReferenceEquals(null, compareTo)) return false;

            return Id.Equals(compareTo.Id);
        }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Encapsulamento – Contexto Vendas

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using FluentValidation.Results;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Vendas.Domain
{
    public class Pedido : Entity, IAggregateRoot
    {
        public int Codigo { get; private set; }
        public Guid ClienteId { get; private set; }
        public Guid? VoucherId { get; private set; }
        public bool VoucherUtilizado { get; private set; }
        public decimal Desconto { get; private set; }
        public decimal ValorTotal { get; private set; }
        public DateTime DataCadastro { get; private set; }
        public PedidoStatus PedidoStatus { get; private set; }

        private readonly List<PedidoItem> _pedidoItems;
        public IReadOnlyCollection<PedidoItem> PedidoItems => _pedidoItems;

        // EF Rel.
        public Voucher Voucher { get; private set; }

        public Pedido(Guid clienteId,
                      bool voucherUtilizado,
                      decimal desconto,
                      decimal valorTotal)
        {
            ClienteId = clienteId;
            VoucherUtilizado = voucherUtilizado;
            Desconto = desconto;
            ValorTotal = valorTotal;
            _pedidoItems = new List<PedidoItem>();
        }

        protected Pedido()
        {
            _pedidoItems = new List<PedidoItem>();
        }
```

```csharp
        public ValidationResult AplicarVoucher(Voucher voucher)
        {
            var validationResult = voucher.ValidarSeAplicavel();
            if (!validationResult.IsValid) return validationResult;

            Voucher = voucher;
            VoucherUtilizado = true;
            CalcularValorPedido();

            return validationResult;
        }

        public void CalcularValorPedido()
        {
            ValorTotal = PedidoItems.Sum(p => p.CalcularValor());
            CalcularValorTotalDesconto();
        }

        public void CalcularValorTotalDesconto()
        {
            if (!VoucherUtilizado) return;

            decimal desconto = 0;
            var valor = ValorTotal;

            if (Voucher.TipoDescontoVoucher == TipoDescontoVoucher.Porcentagem)
            {
                if (Voucher.Percentual.HasValue)
                {
                    desconto = (valor * Voucher.Percentual.Value) / 100;
                    valor -= desconto;
                }
            }
            else
            {
                if (Voucher.ValorDesconto.HasValue)
                {
                    desconto = Voucher.ValorDesconto.Value;
                    valor -= desconto;
                }
            }

            ValorTotal = valor < 0 ? 0 : valor;
            Desconto = desconto;
        }
```

```csharp
        public bool PedidoItemExistente(PedidoItem item)
        {
            return _pedidoItems.Any(p => p.ProdutoId == item.ProdutoId);
        }

        public void AdicionarItem(PedidoItem item)
        {
            if (!item.EhValido()) return;

            item.AssociarPedido(Id);

            if (PedidoItemExistente(item))
            {
                var itemExistente = _pedidoItems.FirstOrDefault(p => p.ProdutoId == item.ProdutoId);
                itemExistente.AdicionarUnidades(item.Quantidade);
                item = itemExistente;

                _pedidoItems.Remove(itemExistente);
            }

            item.CalcularValor();
            _pedidoItems.Add(item);

            CalcularValorPedido();
        }

        public void RemoverItem(PedidoItem item)
        {
            if (!item.EhValido()) return;

            var itemExistente = PedidoItems.FirstOrDefault(p => p.ProdutoId == item.ProdutoId);

            if (itemExistente == null) throw new DomainException("O item não pertence ao pedido");
            _pedidoItems.Remove(itemExistente);

            CalcularValorPedido();
        }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Polimorfismo – Contexto Catálogo e Vendas

Projeto de Bloco: Engenharia de Software e Modelagem

Interfaces – Contexto Catálogo e Vendas

```
1   using System;
2   using System.Collections.Generic;
3   using System.Threading.Tasks;
4   using NerdStore.Core.Data;
5
6   namespace NerdStore.Catalogo.Domain
7   {
        – referências
8       public interface IProdutoRepository : IRepository<Produto>
9       {
            – referências
10          Task<IEnumerable<Produto>> ObterTodos();
            – referências
11          Task<Produto> ObterPorId(Guid id);
            – referências
12          Task<IEnumerable<Produto>> ObterPorCategoria(int codigo);
            – referências
13          Task<IEnumerable<Categoria>> ObterCategorias();
14
            – referências
15          void Adicionar(Produto produto);
            – referências
16          void Atualizar(Produto produto);
17
            – referências
18          void Adicionar(Categoria categoria);
            – referências
19          void Atualizar(Categoria categoria);
20      }
21  }
```

```
1   using System;
2   using System.Collections.Generic;
3   using System.Threading.Tasks;
4   using NerdStore.Core.Data;
5
6   namespace NerdStore.Vendas.Domain
7   {
        – referências
8       public interface IPedidoRepository : IRepository<Pedido>
9       {
            – referências
10          Task<Pedido> ObterPorId(Guid id);
            – referências
11          Task<IEnumerable<Pedido>> ObterListaPorClienteId(Guid clienteId);
            – referências
12          Task<Pedido> ObterPedidoRascunhoPorClienteId(Guid clienteId);
            – referências
13          void Adicionar(Pedido pedido);
            – referências
14          void Atualizar(Pedido pedido);
15
            – referências
16          Task<PedidoItem> ObterItemPorId(Guid id);
            – referências
17          Task<PedidoItem> ObterItemPorPedido(Guid pedidoId, Guid produtoId);
            – referências
18          void AdicionarItem(PedidoItem pedidoItem);
            – referências
19          void AtualizarItem(PedidoItem pedidoItem);
            – referências
20          void RemoverItem(PedidoItem pedidoItem);
21
            – referências
22          Task<Voucher> ObterVoucherPorCodigo(string codigo);
23      }
24  }
```

Projeto de Bloco: Engenharia de Software e Modelagem

API Collections – Contexto Catálogo e Vendas

```csharp
using System.Collections.Generic;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Catalogo.Domain
{
    public class Categoria : Entity
    {
        public string Nome { get; private set; }
        public int Codigo { get; private set; }

        // EF Relation
        public ICollection<Produto> Produtos { get; set; }

        protected Categoria() { }

        public Categoria(string nome, int codigo)
        {
            Nome = nome;
            Codigo = codigo;

            Validar();
        }

        public override string ToString()
        {
            return $"{Nome} - {Codigo}";
        }

        public void Validar()
        {
            Validacoes.ValidarSeVazio(Nome, "O campo Nome da categoria não pode estar vazio");
            Validacoes.ValidarSeIgual(Codigo, 0, "O campo Codigo não pode ser 0");
        }
    }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using FluentValidation.Results;
using NerdStore.Core.DomainObjects;

namespace NerdStore.Vendas.Domain
{
    public class Pedido : Entity, IAggregateRoot
    {
        public int Codigo { get; private set; }
        public Guid ClienteId { get; private set; }
        public Guid? VoucherId { get; private set; }
        public bool VoucherUtilizado { get; private set; }
        public decimal Desconto { get; private set; }
        public decimal ValorTotal { get; private set; }
        public DateTime DataCadastro { get; private set; }
        public PedidoStatus PedidoStatus { get; private set; }

        private readonly List<PedidoItem> _pedidoItems;
        public IReadOnlyCollection<PedidoItem> PedidoItems => _pedidoItems;

        // EF Rel.
        public Voucher Voucher { get; private set; }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Expressões Lambda – Contexto Catálogo e Vendas

```
10  {
        – referências
11      public class ProdutoRepository : IProdutoRepository
12      {
13          private readonly CatalogoContext _context;
14
            – referências
15          public ProdutoRepository(CatalogoContext context)
16          {
17              _context = context;
18          }
            – referências
19          public IUnitOfWork UnitOfWork => _context;
20
            – referências
21          public async Task<IEnumerable<Produto>> ObterTodos()
22          {
23              return await _context.Produtos.AsNoTracking().ToListAsync();
24          }
25
            – referências
26          public async Task<Produto> ObterPorId(Guid id)
27          {
28              //return await _context.Produtos.AsNoTracking().FirstOrDefaultAsync(p
29              return await _context.Produtos.FindAsync(id);
30          }
31
            – referências
32          public async Task<IEnumerable<Produto>> ObterPorCategoria(int codigo)
33          {
34              return await _context
35                              .Produtos
36                              .AsNoTracking()
37                              .Include(p => p.Categoria)
38                              .Where(c => c.Categoria.Codigo == codigo)
39                              .ToListAsync();
40          }
41
```

```
            – referências
11      public class PedidoRepository : IPedidoRepository
12      {
13          private readonly VendasContext _context;
14
            – referências
15          public PedidoRepository(VendasContext context)
16          {
17              _context = context;
18          }
19
            – referências
20          public IUnitOfWork UnitOfWork => _context;
21
            – referências
22          public async Task<Pedido> ObterPorId(Guid id)
23          {
24              return await _context.Pedidos.FindAsync(id);
25          }
26
            – referências
27          public async Task<IEnumerable<Pedido>> ObterListaPorClienteId(Guid clienteId)
28          {
29              return await _context
30                              .Pedidos
31                              .AsNoTracking()
32                              .Where(p => p.ClienteId == clienteId)
33                              .ToListAsync();
34          }
35
            – referências
36          public async Task<Pedido> ObterPedidoRascunhoPorClienteId(Guid clienteId)
37          {
38              var pedido = await _context
39                                  .Pedidos
40                                  .FirstOrDefaultAsync(p =>
41                                      p.ClienteId == clienteId && p.PedidoStatus == PedidoStatus.Rascunho);
42              if (pedido == null) return null;
```

Projeto de Bloco: Engenharia de Software e Modelagem

Validações – Contexto Catálogo

```
4   namespace NerdStore.Catalogo.Domain
5   {
        – referências
6       public class Produto : Entity, IAggregateRoot
7       {
            – referências
8           public Guid CategoriaId { get; private set; }
            – referências
9           public string Nome { get; private set; }
            – referências
10          public string Descricao { get; private set; }
            – referências
11          public bool Ativo { get; private set; }
            – referências
12          public decimal Valor { get; private set; }
            – referências
13          public DateTime DataCadastro { get; private set; }
            – referências
14          public string Imagem { get; private set; }
            – referências
15          public int QuantidadeEstoque { get; private set; }
            – referências
16          public Dimensoes Dimensoes { get; private set; }
            – referências
17          public Categoria Categoria { get; private set; }
18
            – referências
19          protected Produto() { }
            – referências
20          public Produto(string nome,
21                         string descricao,
22                         bool ativo,
23                         decimal valor,
24                         Guid categoriaId,
25                         DateTime dataCadastro,
26                         string imagem,
27                         Dimensoes dimensoes)
28          {
29              CategoriaId = categoriaId;
30              Nome = nome;
31              Descricao = descricao;
32              Ativo = ativo;
33              Valor = valor;
34              DataCadastro = dataCadastro;
35              Imagem = imagem;
36              Dimensoes = dimensoes;
37
38              Validar();
39          }
40
```

```
51          public void AlterarDescricao(string descricao)
52          {
53              Validacoes.ValidarSeVazio(descricao, "O campo Descricao do produto não pode estar vazio");
54              Descricao = descricao;
55          }
56
            – referências
57          public void DebitarEstoque(int quantidade)
58          {
59              if (quantidade < 0) quantidade *= -1;
60              if (!PossuiEstoque(quantidade)) throw new DomainException("Estoque insuficiente");
61              QuantidadeEstoque -= quantidade;
62          }
63
            – referências
64          public void ReporEstoque(int quantidade)
65          {
66              QuantidadeEstoque += quantidade;
67          }
68
            – referências
69          public bool PossuiEstoque(int quantidade)
70          {
71              return QuantidadeEstoque >= quantidade;
72          }
73
            – referências
74          public void Validar()
75          {
76              Validacoes.ValidarSeVazio(Nome, "O campo Nome do produto não pode estar vazio");
77              Validacoes.ValidarSeVazio(Descricao, "O campo Descricao do produto não pode estar vazio");
78              Validacoes.ValidarSeIgual(CategoriaId, Guid.Empty, "O campo CategoriaId do produto não pode estar vazio");
79              Validacoes.ValidarSeMenorQue(Valor, 1, "O campo Valor do produto não pode se menor igual a 0");
80              Validacoes.ValidarSeVazio(Imagem, "O campo Imagem do produto não pode estar vazio");
81          }
82      }
83  }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Validações – Contexto Vendas

```csharp
using System;
using FluentValidation;
using NerdStore.Core.Messages;

namespace NerdStore.Vendas.Application.Commands
{
    public class AdicionarItemPedidoCommand : Command
    {
        public Guid ClienteId { get; private set; }
        public Guid ProdutoId { get; private set; }
        public string Nome { get; private set; }
        public int Quantidade { get; private set; }
        public decimal ValorUnitario { get; private set; }

        public AdicionarItemPedidoCommand(Guid clienteId,
                                          Guid produtoId,
                                          string nome,
                                          int quantidade,
                                          decimal valorUnitario)
        {
            ClienteId = clienteId;
            ProdutoId = produtoId;
            Nome = nome;
            Quantidade = quantidade;
            ValorUnitario = valorUnitario;
        }

        public override bool EhValido()
        {
            ValidationResult = new AdicionarItemPedidoValidation().Validate(this);
            return ValidationResult.IsValid;
        }
    }
```

```csharp
    public class AdicionarItemPedidoValidation : AbstractValidator<AdicionarItemPedidoCommand>
    {
        public AdicionarItemPedidoValidation()
        {
            RuleFor(c => c.ClienteId)
                .NotEqual(Guid.Empty)
                .WithMessage("Id do cliente inválido");

            RuleFor(c => c.ProdutoId)
                .NotEqual(Guid.Empty)
                .WithMessage("Id do produto inválido");

            RuleFor(c => c.Nome)
                .NotEmpty()
                .WithMessage("O nome do produto não foi informado");

            RuleFor(c => c.Quantidade)
                .GreaterThan(0)
                .WithMessage("A quantidade miníma de um item é 1");

            RuleFor(c => c.Quantidade)
                .LessThan(15)
                .WithMessage("A quantidade máxima de um item é 15");

            RuleFor(c => c.ValorUnitario)
                .GreaterThan(0)
                .WithMessage("O valor do item precisa ser maior que 0");
        }
    }
}
```

Projeto de Bloco: Engenharia de Software e Modelagem

Testes de Unidade – Contexto Catálogo

```
4
5   namespace NerdStore.Catalogo.Domain.Tests
6   {
        – referências
7       public class ProdutoTests
8       {
9           [Fact]
            – referências
10          public void Produto_Validar_ValidacoesDevemRetornarExceptions()
11          {
12
13              // Arrange & Act & Assert
14
15              var ex = Assert.Throws<DomainException>(() =>
16                  new Produto(string.Empty, "Descricao", false, 100, Guid.NewGuid(), DateTime.Now, "Imagem", new Dimensoes(1, 1, 1))
17              );
18
19              Assert.Equal("O campo Nome do produto não pode estar vazio", ex.Message);
20
21              ex = Assert.Throws<DomainException>(() =>
22                  new Produto("Nome", string.Empty, false, 100, Guid.NewGuid(), DateTime.Now, "Imagem", new Dimensoes(1, 1, 1))
23              );
24
25              Assert.Equal("O campo Descricao do produto não pode estar vazio", ex.Message);
26
27              ex = Assert.Throws<DomainException>(() =>
28                  new Produto("Nome", "Descricao", false, 0, Guid.NewGuid(), DateTime.Now, "Imagem", new Dimensoes(1, 1, 1))
29              );
30
31              Assert.Equal("O campo Valor do produto não pode se menor igual a 0", ex.Message);
32
33              ex = Assert.Throws<DomainException>(() =>
34                  new Produto("Nome", "Descricao", false, 100, Guid.Empty, DateTime.Now, "Imagem", new Dimensoes(1, 1, 1))
35              );
36
37              Assert.Equal("O campo CategoriaId do produto não pode estar vazio", ex.Message);
38
39              ex = Assert.Throws<DomainException>(() =>
40                  new Produto("Nome", "Descricao", false, 100, Guid.NewGuid(), DateTime.Now, string.Empty, new Dimensoes(1, 1, 1))
41              );
42
43              Assert.Equal("O campo Imagem do produto não pode estar vazio", ex.Message);
44
45              ex = Assert.Throws<DomainException>(() =>
46                  new Produto("Nome", "Descricao", false, 100, Guid.NewGuid(), DateTime.Now, "Imagem", new Dimensoes(0, 1, 1))
47              );
48
49              Assert.Equal("O campo Altura não pode ser menor ou igual a 0", ex.Message);
50          }
51      }
52  }
```

Projeto de Bloco: Engenharia de Software e Modelagem

CRUD Classes de Domínio – Contexto Catálogo

```
1   using System;
2   using System.Collections.Generic;
3   using System.Threading.Tasks;
4   using NerdStore.Core.Data;
5
6   namespace NerdStore.Catalogo.Domain
7   {
        0 referências
8       public interface IProdutoRepository : IRepository<Produto>
9       {
            0 referências
10          Task<IEnumerable<Produto>> ObterTodos();
            0 referências
11          Task<Produto> ObterPorId(Guid id);
            0 referências
12          Task<IEnumerable<Produto>> ObterPorCategoria(int codigo);
            0 referências
13          Task<IEnumerable<Categoria>> ObterCategorias();
14
            0 referências
15          void Adicionar(Produto produto);
            0 referências
16          void Atualizar(Produto produto);
17
            0 referências
18          void Adicionar(Categoria categoria);
            0 referências
19          void Atualizar(Categoria categoria);
20      }
21  }
```

Projeto de Bloco: Engenharia de Software e Modelagem

CRUD Classes de Domínio – Contexto Vendas / Pagamento

```
1   using System;
2   using System.Collections.Generic;
3   using System.Threading.Tasks;
4   using NerdStore.Core.Data;
5
6   namespace NerdStore.Vendas.Domain
7   {
        – referências
8       public interface IPedidoRepository : IRepository<Pedido>
9       {
            – referências
10          Task<Pedido> ObterPorId(Guid id);
            – referências
11          Task<IEnumerable<Pedido>> ObterListaPorClienteId(Guid clienteId);
            – referências
12          Task<Pedido> ObterPedidoRascunhoPorClienteId(Guid clienteId);
            – referências
13          void Adicionar(Pedido pedido);
            – referências
14          void Atualizar(Pedido pedido);
15
            – referências
16          Task<PedidoItem> ObterItemPorId(Guid id);
            – referências
17          Task<PedidoItem> ObterItemPorPedido(Guid pedidoId, Guid produtoId);
            – referências
18          void AdicionarItem(PedidoItem pedidoItem);
            – referências
19          void AtualizarItem(PedidoItem pedidoItem);
            – referências
20          void RemoverItem(PedidoItem pedidoItem);
21
            – referências
22          Task<Voucher> ObterVoucherPorCodigo(string codigo);
23      }
24  }
```

```
1   using NerdStore.Core.Data;
2
3   namespace NerdStore.Pagamentos.Business
4   {
        – referências
5       public interface IPagamentoRepository : IRepository<Pagamento>
6       {
            – referências
7           void Adicionar(Pagamento pagamento);
8
            – referências
9           void AdicionarTransacao(Transacao transacao);
10      }
11  }
```

Projeto de Bloco: Engenharia de Software e Modelagem

Consultar e Manipular usando LINQ

Projeto de Bloco: Engenharia de Software e Modelagem

Mapeamento EF Core – Contexto Catálogo

```csharp
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using NerdStore.Catalogo.Domain;

namespace NerdStore.Catalogo.Data.Mappings
{
    public class ProdutoMapping : IEntityTypeConfiguration<Produto>
    {
        public void Configure(EntityTypeBuilder<Produto> builder)
        {
            builder.HasKey(c => c.Id);

            builder.Property(c => c.Nome)
                .IsRequired()
                .HasColumnType("varchar(250)");

            builder.Property(c => c.Descricao)
                .IsRequired()
                .HasColumnType("varchar(500)");

            builder.Property(c => c.Imagem)
                .IsRequired()
                .HasColumnType("varchar(250)");

            builder.OwnsOne(c => c.Dimensoes, cm =>
            {
                cm.Property(c => c.Altura)
                    .HasColumnName("Altura")
                    .HasColumnType("int");

                cm.Property(c => c.Largura)
                    .HasColumnName("Largura")
                    .HasColumnType("int");

                cm.Property(c => c.Profundidade)
                    .HasColumnName("Profundidade")
                    .HasColumnType("int");
            });

            builder.ToTable("Produtos");
        }
    }
}
```

```csharp
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using NerdStore.Catalogo.Domain;

namespace NerdStore.Catalogo.Data.Mappings
{
    public class CategoriaMapping : IEntityTypeConfiguration<Categoria>
    {
        public void Configure(EntityTypeBuilder<Categoria> builder)
        {

            builder.HasKey(c => c.Id);

            builder.Property(c => c.Nome)
                .IsRequired()
                .HasColumnType("varchar(250)");

            // 1 : N => Categorias : Produtos
            builder.HasMany(c => c.Produtos)
                .WithOne(p => p.Categoria)
                .HasForeignKey(p => p.CategoriaId);

            builder.ToTable("Categorias");
        }
    }
}
```

Projeto de Bloco: Engenharia de Software e Modelagem

Mapeamento EF Core – Contexto Vendas

```csharp
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using NerdStore.Vendas.Domain;

namespace NerdStore.Catalogo.Data.Mappings
{
    public class PedidoMapping : IEntityTypeConfiguration<Pedido>
    {
        public void Configure(EntityTypeBuilder<Pedido> builder)
        {
            builder.HasKey(c => c.Id);

            builder.Property(c => c.Codigo)
                .HasDefaultValueSql("NEXT VALUE FOR MinhaSequencia");

            // 1 : N => Pedido : PedidoItems
            builder.HasMany(c => c.PedidoItems)
                .WithOne(c => c.Pedido)
                .HasForeignKey(c => c.PedidoId);

            builder.ToTable("Pedidos");
        }
    }
}
```

```csharp
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using NerdStore.Vendas.Domain;

namespace NerdStore.Catalogo.Data.Mappings
{
    public class PedidoItemMapping : IEntityTypeConfiguration<PedidoItem>
    {
        public void Configure(EntityTypeBuilder<PedidoItem> builder)
        {
            builder.HasKey(c => c.Id);


            builder.Property(c => c.ProdutoNome)
                .IsRequired()
                .HasColumnType("varchar(250)");

            // 1 : N => Pedido : Pagamento
            builder.HasOne(c => c.Pedido)
                .WithMany(c => c.PedidoItems);

            builder.ToTable("PedidoItems");
        }
    }
}
```

Projeto de Bloco: Engenharia de Software e Modelagem

Desacoplar Camada de Persistência com MVC

```csharp
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NerdStore.Catalogo.Application.Services;

namespace NerdStore.WebApp.MVC.Controllers
{
    public class VitrineController : Controller
    {
        private readonly IProdutoAppService _produtoAppService;

        public VitrineController(IProdutoAppService produtoAppService)
        {
            _produtoAppService = produtoAppService;
        }

        [HttpGet]
        [Route("")]
        [Route("vitrine")]
        public async Task<IActionResult> Index()
        {
            return View(await _produtoAppService.ObterTodos());
        }

        [HttpGet]
        [Route("produto-detalhe/{id}")]
        public async Task<IActionResult> ProdutoDetalhe(Guid id)
        {
            return View(await _produtoAppService.ObterPorId(id));
        }
    }
}
```

```csharp
using System.Threading.Tasks;
using MediatR;
using Microsoft.AspNetCore.Mvc;
using NerdStore.Core.Communication.Mediator;
using NerdStore.Core.Messages.CommonMessages.Notifications;
using NerdStore.Vendas.Application.Queries;

namespace NerdStore.WebApp.MVC.Controllers
{
    public class PedidoController : ControllerBase
    {
        private readonly IPedidoQueries _pedidoQueries;

        public PedidoController(IPedidoQueries pedidoQueries,
            INotificationHandler<DomainNotification> notifications,
            IMediatorHandler mediatorHandler) : base(notifications, mediatorHandler)
        {
            _pedidoQueries = pedidoQueries;
        }

        [Route("meus-pedidos")]
        public async Task<IActionResult> Index()
        {
            return View(await _pedidoQueries.ObterPedidosCliente(ClienteId));
        }
    }
}
```

Projeto de Bloco: Engenharia de Software e Modelagem

Desacoplar Camada de Persistência com Web API

Projeto de Bloco: Engenharia de Software e Modelagem

Documentação com Swagger/OpenAPI

Projeto de Bloco: Engenharia de Software e Modelagem

Requisitos de Alto Nível - RANs

Projeto de Bloco: Engenharia de Software e Modelagem

Os RANs são descrições do que o sistema/produto deve ou não fazer. Eles não definem como deve ser feito, pois isto já compete a área técnica. Eles geralmente se concentram nos resultados desejados e nas funcionalidades gerais que o sistema ou produto deve ter para atender às necessidades dos usuários e das partes interessadas.

Projeto de Bloco: Engenharia de Software e Modelagem

Projeto de Bloco: Engenharia de Software e Modelagem

1. O sistema deve permitir que os usuários façam login e gerenciem suas contas.
2. O produto deve ser fácil de usar para usuários com habilidades limitadas em tecnologia.
3. O sistema deve ser capaz de processar um grande volume de transações em tempo real.
4. O produto deve ser seguro e proteger os dados confidenciais dos usuários.
5. O sistema deve ser escalável e capaz de lidar com o crescimento futuro da empresa.
6. O Produto enviado deverá ser rastreável.

Projeto de Bloco: Engenharia de Software e Modelagem

Requisitos Funcionais

Projeto de Bloco: Engenharia de Software e Modelagem

Requisitos Funcionais	
[RF-001]	Manter Produto
[RF-002]	Manter Estoque
[RF-003]	Manter Categoria
[RF-004]	Gerir venda
[RF-005]	Aplicar voucher pedido
[RF-006]	Estornar estoque

Requisito	Regra de negócio
RF 01	QUANTIDADE: produto é considerado disponível quando está ativo e tem estoque > 0
RF 01	PRODUTO: tem que ter obrigatoriamente id, nome, descrição, valor e imagem
RF 01	PRODUTO: dimensões devem ter altura, largura e profundidade maior ou igual a zero
RF 01	PRODUTO: a formatação das dimensões deve ser "L x A x P"
RF 01	PRODUTO: Sistema deve exibir detalhes de produto
RF 02	ESTOQUE: Ao concluir uma compra, sistema deve debitar quantidade de itens de estoque
RF 03	PRODUTO: Sistema deve listar todos os produtos disponíveis
RF 03	PRODUTO: Sistema deve listar produtos disponíveis filtrados por categoria
RF 03	CATEGORIA: deve ter obrigatoriamente Nome e código
RF 04	PAGAMENTO: Pagamento pode ser realizado por cartão de crédito e boleto
RF 04	CARRINHO: Precisa ter 1 ou mais itens para ser fechado
RF 04	PAGAMENTO: Status de transação pode ser Pago ou Recusado
RF 04	COMPRA: Sistema deve recuperar resumo de compra desejado
RF 05	VOUCHER: Carrinho de compras - Desconto absoluto e/ou percentual.
RF 06	PAGAMENTO: Se o pagamento não for realizado com sucesso, a compra não deve ser concluída
RF 06	ESTORNO: Em casos de devolução de produto, estoque deve ser acrescido da quantidade devolvida

Projeto de Bloco: Engenharia de Software e Modelagem

Requisitos Não Funcionais

Projeto de Bloco: Engenharia de Software e Modelagem

Código	Requisitos Não Funcionais	Indicadores
RNF 01	Desempenho	Tempo de Resposta das requisições(1 a cada 100 ms)
RNF 02	Disponibilidade	Funcional em tempo integral (24h/7 dias por semana)
RNF 03	Escalabilidade	picos de consumo de até 1k acessos/min
RNF 04	Segurança	Autenticação JWT / criptografia Char 256
RNF 05	Privacidade	Cumprimento da LGPD

Projeto de Bloco: Engenharia de Software e Modelagem

Matriz de Rastreabilidade

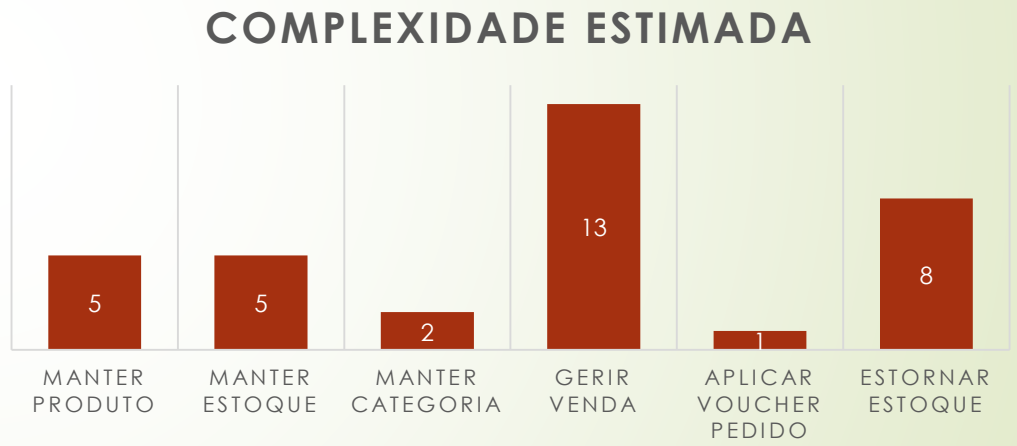
Requesitos X Casos de uso		[UC-001] Obter Produto por Categoria	[UC-002] Obter Produto por Id	[UC-003] Obter Todos os Produtos	[UC-004] Adicionar Produto	[UC-005] Atualizar Produto	[UC-006] Debitar Produto em Estoque	[UC-007] Repor Produto em Estoque	[UC-008] Debitar Estoque	[UC-009] Debitar Lista Produtos Pedido	[UC-010] Repor Estoque	[UC-011] Repor Lista Produtos Pedido	[UC-012] Obter Categorias	[UC-013] Adicionar item pedido	[UC-014] Cancelar processamento pedido	[UC-015] Finalizar pedido	[UC-015] Iniciar pedido	[UC-015] Remove item pedido	[UC-015] Obtém pagamento
[RF-001]	Manter Produto	■	■	■	■	■													
[RF-002]	Manter Estoque						■	■	■	■	■	■							
[RF-003]	Manter Categoria												■						
[RF-004]	Gerir venda													■					
[RF-005]	Aplicar voucher pedido														■				
[RF-006]	Estornar estoque															■	■	■	■

Projeto de Bloco: Engenharia de Software e Modelagem

Processo de estimativa

Projeto de Bloco: Engenharia de Software e Modelagem

código	Requisito Funcional	Complexidade Estimada
[RF-001]	Manter Produto	5
[RF-002]	Manter Estoque	5
[RF-003]	Manter Categoria	2
[RF-004]	Gerir venda	13
[RF-005]	Aplicar voucher pedido	1
[RF-006]	Estornar estoque	8



Projeto de Bloco: Engenharia de Software e Modelagem

Cronograma de implementação

Requisitos	Semanas							
	1	2	3	4	5	6	7	8
Manter Produto	■							
Manter Estoque		■						
Manter Categoria			■					
Gerir venda			■	■	■			
Aplicar voucher pedido						■		
Estornar estoque						■	■	■

Cronograma de lançamento

Requisitos	Releases			
	1	2	3	Final
Manter Produto	■			
Manter Estoque	■			
Manter Categoria		■		
Gerir venda			■	
Aplicar voucher pedido			■	
Estornar estoque				■

Projeto de Bloco: Engenharia de Software e Modelagem

Proposta de implantação

- AKS (Serviço de Kubernetes do Azure)
- Registro de containers do Azure
- Aplicativo Web para Contêiners
- Banco de dados SQL do Azure
- Cache do Redis para Azure
- Armazenamento Blobs do Azure
- Key Vault
- API Gateway
- Firewall

Projeto de Bloco: Engenharia de Software e Modelagem

Cronograma geral

Fase do projeto	Meses					
	1	2	3	4	5	..
Implementação	■	■				
Implantação	■	■				
Manuentanção			■	■	■	
Sustentação						■

Projeto de Bloco: Engenharia de Software e Modelagem

Custo do software

Manutenção	R$ 3.750,00
Implantação	R$ 6.000,00
Implementação	R$ 15.454,55
Gerenciamento	20%
Margem de lucro	40%
CUSTO TOTAL	**R$ 30.246,86**
Manutenção extra	*R$ 1.250,00 / mês*

Projeto de Bloco: Engenharia de Software e Modelagem

- **Referências**

MODELAGEM de Domínios Ricos. desenvolvedor.io/. Disponível em: https://desenvolvedor.io/curso/modelagem-de-dominios-ricos . Acesso em: 02 jun. 2022.

MODELAGEM de Software. lms.infnet.edu.br. Disponível em: https://lms.infnet.edu.br/moodle/course/view.php?id=5928 . Acesso em: 10 fev. 2022.

ENGENHARIA de Software Aplicada. lms.infnet.edu.br. Disponível em: https://lms.infnet.edu.br/moodle/course/view.php?id=5927 . Acesso em: 11 dez. 2022.

MODELAGEM de Dados UML. udemy.com. Disponível em: https://www.udemy.com/course/uml-diagrama-de-classes/learn/lecture/9881660?start=255#overview . Acesso em: 17 abr. 2022.