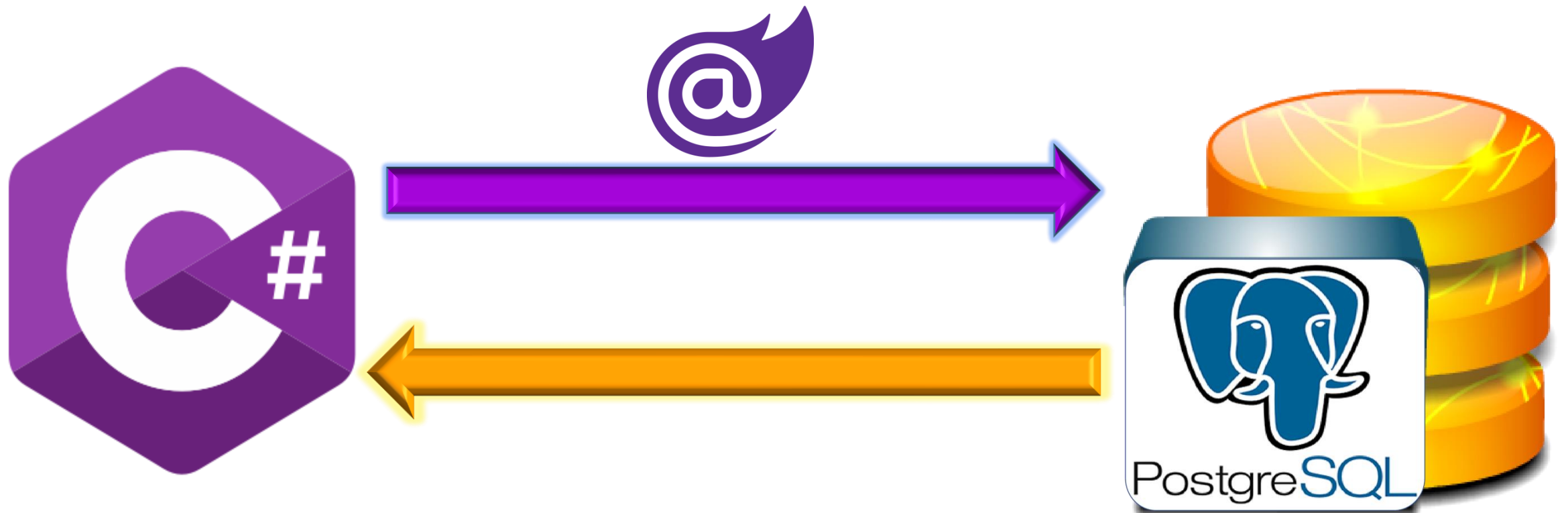


Programação com Acesso a Banco de Dados



AULA 5

Implementação do Sistema de Inventários





PROPOSTA DO SISTEMA



Vamos implementar um sistema de inventário que realizará as seguintes funções:

Primeiro Cadastro: A pessoa será cadastrada pela primeira vez. Neste módulo, além do cadastro dos dados da pessoa, também serão cadastrados os bens.

Segundo Cadastro: A pessoa já está cadastrada e, neste caso, novos bens serão acrescentados a ela, que será selecionada.

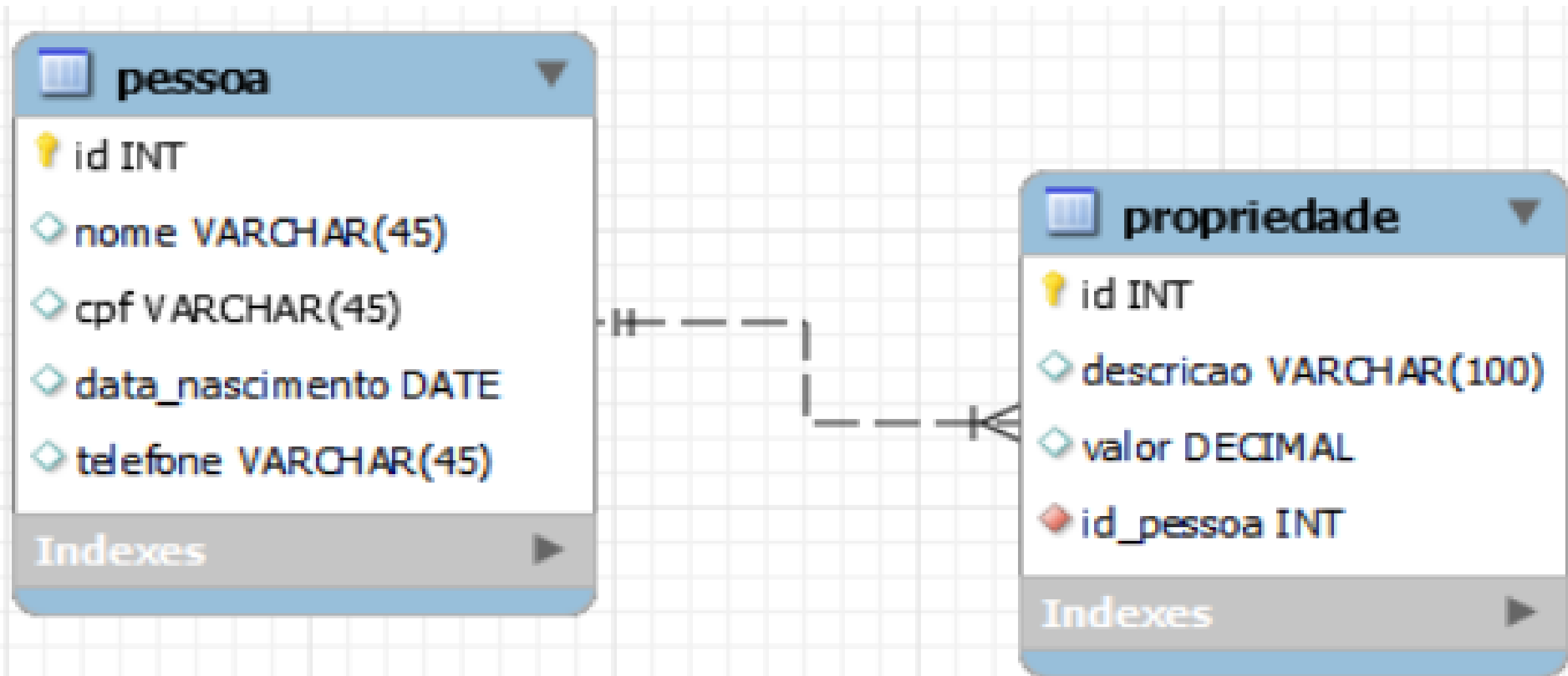
Consulta: Selecionará uma pessoa e listar todos os bens, juntamente com a soma total dos bens registrados.



BANCO DE DADOS MYSQL



Utilize o arquivo “bd_inventario.mwb” disponibilizado com os Slides para criar o banco de dados em MySql





CONFIGURAR O SISTEMA COM ENTITY FRAMEWORK




Clique com o botão direito do mouse sobre o nome do projeto ([AppInventario](#)) e depois vá na opção [Gerenciar Pacotes do NuGET](#). Depois selecione a opção conforme imagem abaixo:


Procurar Instalado Atualizações


Gerenciador de Pacotes do NuGet: AppInventario


Pomelo.EntityFrameworkCore.MySql x ↻ ☐ Incluir pré-lançamento


Origem do pacote: nuget.org ⚙

**Pomelo.EntityFrameworkCore.MySql** por Laurents Meyer, Caleb Lloyd, Yuko Zheng, **59,5M** downloads 8.0.2
Pomelo's MySQL database provider for Entity Framework Core.

**Pomelo.EntityFrameworkCore.MySql.Design** por Pomelo.EntityFrameworkCore.MySql.Design, **2,29M** downloads 1.1.2
Package Description

**Pomelo.JsonObject** por Pomelo.JsonObject, **24,4M** downloads 2.2.1
MySQL provider for Entity Framework Core

**Pomelo.EntityFrameworkCore.MySql.NetTopologySuite** por Laurents Meyer, Caleb Lloyd, Yuko Zheng, **8.0.2**
NetTopologySuite support for Pomelo's MySQL provider for Entity Framework Core.

**Pomelo.EntityFrameworkCore.MySql** nuget.org

Versão: 7.0.0

Opções

☒ Mostrar janela de visualização

Descrição

Pomelo's MySQL database provider for Entity Framework Core.

Versão:

7.0.0

Autor(es):

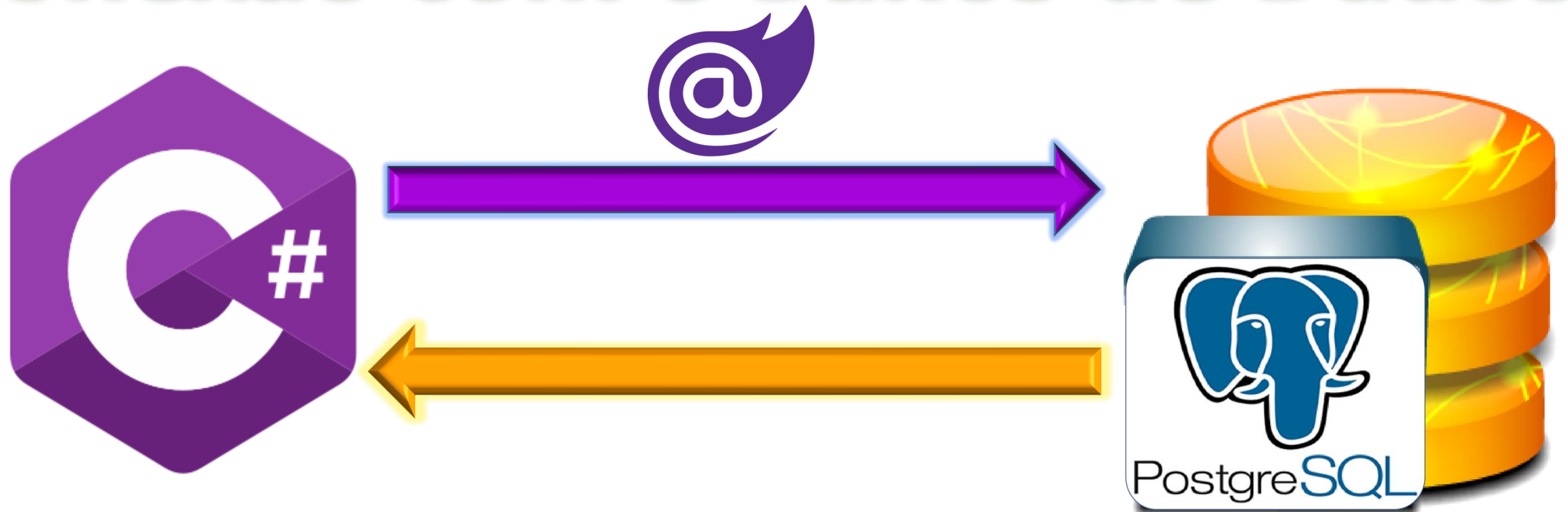
Laurents Meyer, Caleb Lloyd, Yuko Zheng

Licença:

MIT

1ª Fase da implementação

Conexão com o Banco de Dados





LEGENDA DO FLUXOGRAMA



O próximo slide tem o objetivo de demonstrar a visão geral do **sistema de conexão com o banco de dados**. Logo abaixo, segue legenda e as observações para a correta leitura do fluxograma:



Indica qual o próximo item que você deve criar.



Indica a conexão que existe entre as classes

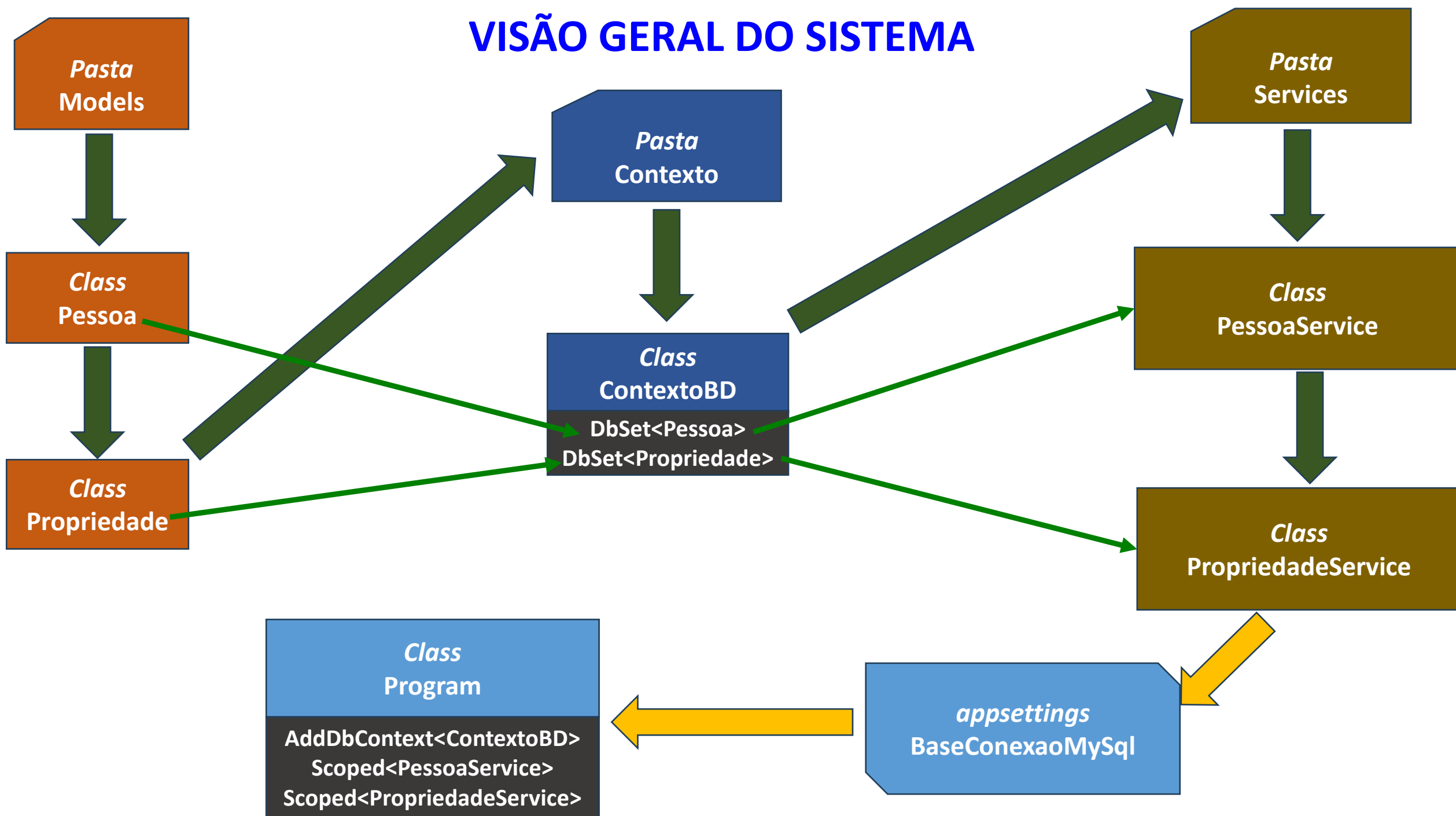


Indica qual o próximo item existente que será editado.

Obs1: A classe tem a mesma cor da pasta a qual ela pertence.

Obs2: O fluxograma começa com a criação da pasta Models.

VISÃO GERAL DO SISTEMA





Implementação da classe “Pessoa” com mapeamento



```
using System.ComponentModel.DataAnnotations.Schema;

namespace AppInventario.Models
{
    [Table("pessoa")]
    public class Pessoa
    {
        [Column("id")]
        public int Id { get; set; }

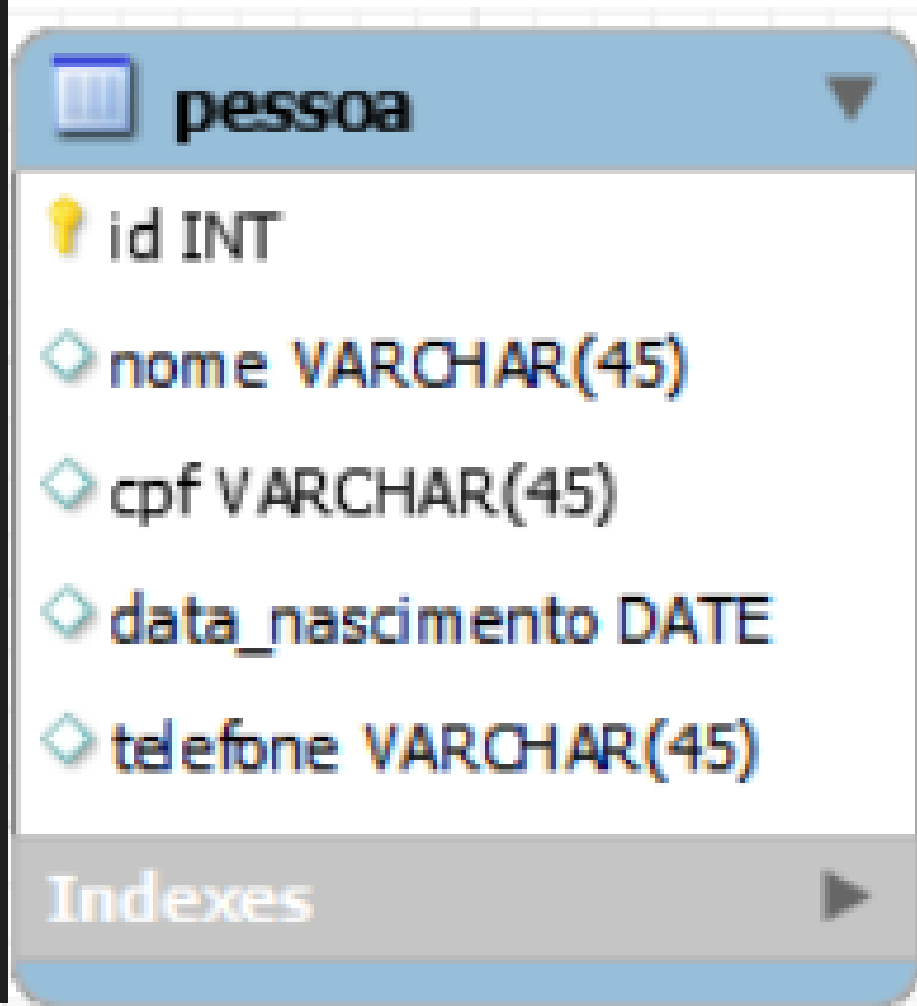
        [Column("nome")]
        public string? Nome { get; set; }

        [Column("cpf")]
        public string? Cpf { get; set; }

        [Column("data_nascimento")]
        public DateTime? DataNasc { get; set; }

        [Column("telefone")]
        public string? Telefone { get; set; }
    }
}
```

Obs.: Como a conexão é para MySQL, não é necessário informar o Schema = public





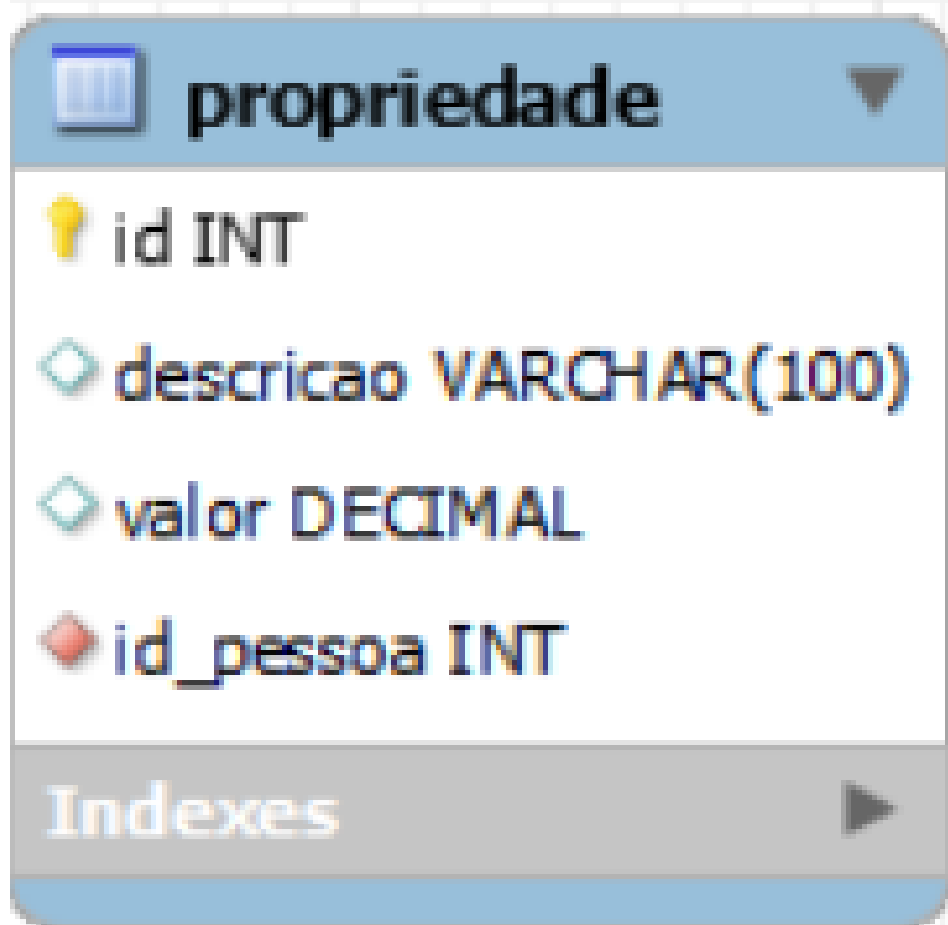
Implementação da classe “Propriedade” com mapeamento



```
using System.ComponentModel.DataAnnotations.Schema;

namespace AppInventario.Models
{
    [Table("propriedade")]
    public class Propriedade
    {
        [Column("id")]
        public int Id { get; set; }
        [Column("descricao")]
        public string? Descricao { get; set; }
        [Column("valor")]
        public double? Valor { get; set; }
        [Column("id_pessoa")]
        public int? IdPessoa { get; set; }
    }
}
```

Obs.: Como a conexão é para MySQL, não é necessário informar o Schema = public

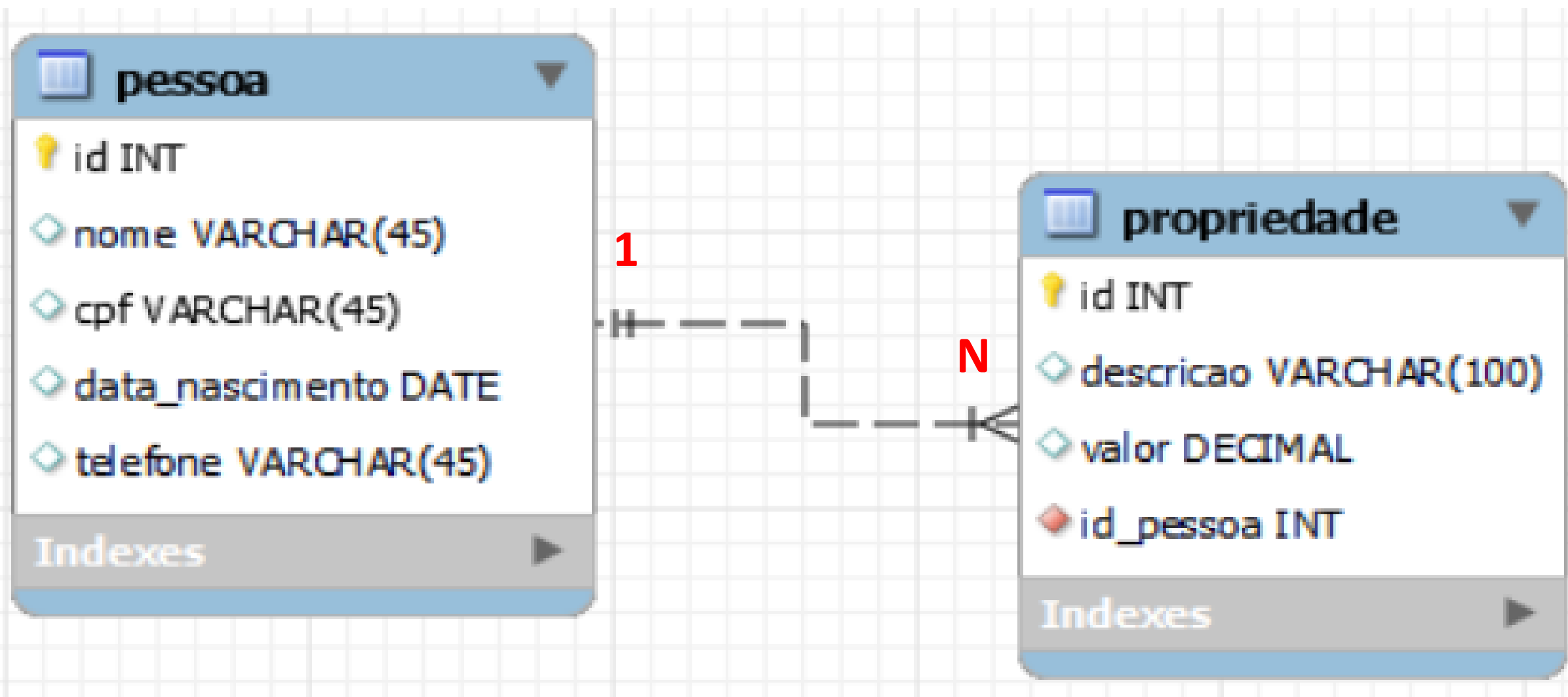




Mapeamento da relação das tabelas no banco de dados



Para o correto funcionamento, será necessário mapear a relação (1:N)





Pessoa tem muitas propriedades



```
[Table("pessoa")]
public class Pessoa
{
    [Column("id")]
    public int Id { get; set; }

    [Column("nome")]
    public string? Nome { get; set; }

    [Column("cpf")]
    public string? Cpf { get; set; }

    [Column("data_nascimento")]
    public DateTime? DataNasc { get; set; }

    [Column("telefone")]
    public string? Telefone { get; set; }

    //lista de propriedades da pessoa
    public List<Propriedade>? Propriedades { get; set; }
}
```

Acrescentar o código





Propriedade tem uma Pessoa

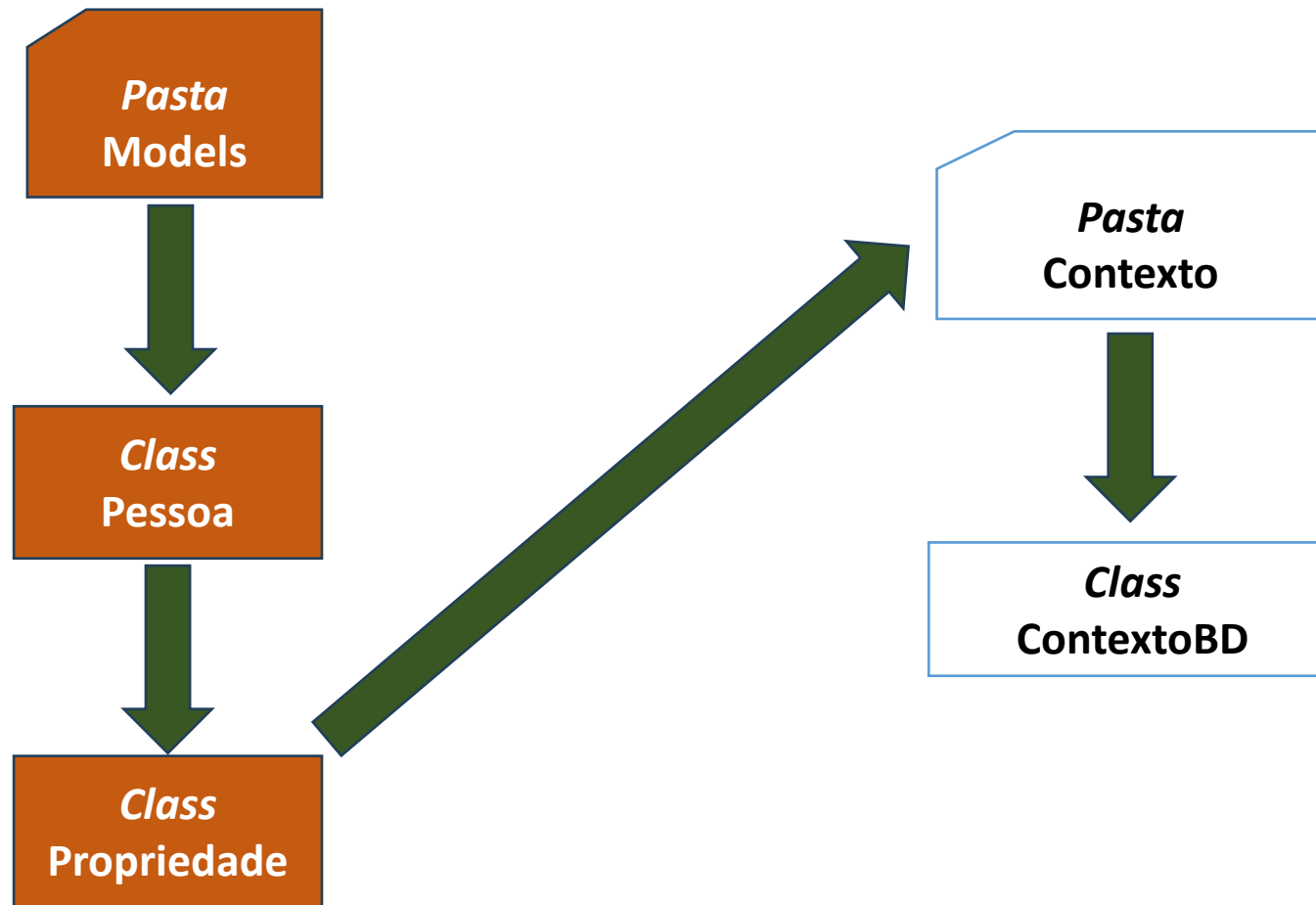


```
[Table("propriedade")]
public class Propriedade
{
    [Column("id")]
    public int Id { get; set; }
    [Column("descricao")]
    public string? Descricao { get; set; }
    [Column("valor")]
    public double? Valor { get; set; }
    [Column("id_pessoa")]
    public int? IdPessoa { get; set; }
    [ForeignKey("IdPessoa")]
    public Pessoa? Pessoa { get; set; }
}
```

Informa qual o atributo da classe vai armazenar a FK

Indica o dono da propriedade

PROGRESSO DA IMPLEMENTAÇÃO



As etapas da criação da pasta **Models** e suas respectivas classes já foram concluídas, conforme o fluxograma. Próximo passo será a criação da pasta **Contexto** e da classe **ContextoBD**.



Implementação da classe “ContextoBD” para conexão com BD

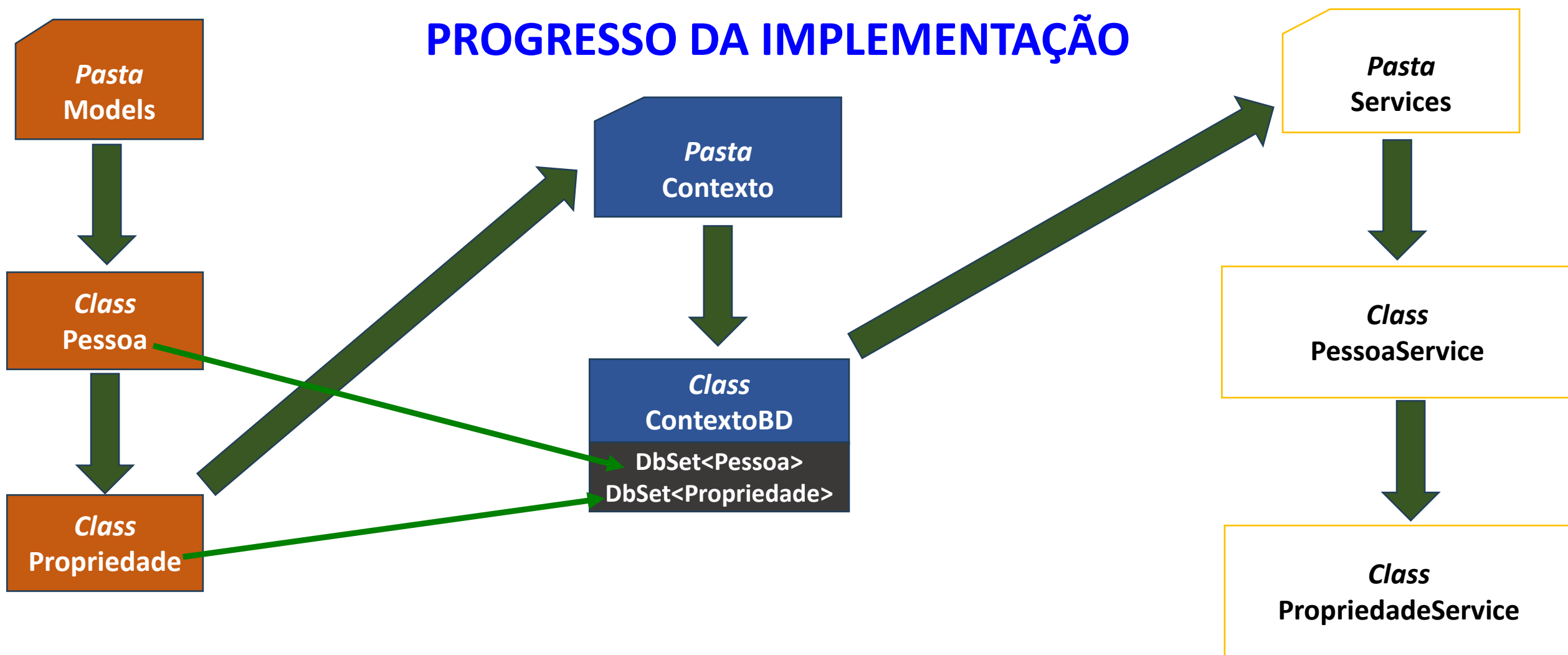


```
using AppInventario.Models;
using Microsoft.EntityFrameworkCore;

namespace AppInventario.Contexto
{
    public class ContextoBD : DbContext
    {
        public ContextoBD(DbContextOptions<ContextoBD> options) : base(options)
        {
        }

        public DbSet<Pessoa>? Pessoas { get; set; }
        public DbSet<Propriedade> Propriedades { get; set; }
    }
}
```

PROGRESSO DA IMPLEMENTAÇÃO



As etapas da criação da pasta **Contexto** e classe **ContextoBD** já foram concluídas conforme o fluxograma. Próximo passo será a criação da pasta **Services** e das suas respectivas classes **PessoaService** e **PropriedadeService**.



Implementação da Classe PessoaService e seus serviços



```
using AppInventario.Contexto;  
using AppInventario.Models;  
using Microsoft.EntityFrameworkCore;  
  
namespace AppInventario.Services  
{  
    public class PessoaService  
    {  
        private readonly ContextoBD _context;  
  
        public PessoaService(ContextoBD con)  
        {  
            _context = con;  
        }  
  
        public async Task<List<Pessoa>>? Pessoas()  
        {  
            var pessoas = await _context.Pessoas.Include(p=>p.Propriedades).ToListAsync();  
            return pessoas;  
        }  
  
        public async Task<Pessoa>? GetPessoa(int id)  
        {  
            var pessoa = await _context.Pessoas.Include(p=>p.Propriedades).Where(p => p.Id == id).FirstOrDefaultAsync();  
            return pessoa;  
        }  
    }  
}
```

Continuação da Implementação da Classe PessoaService



```
public async Task<Pessoa> GetPessoa(string cpf)
{
    var pessoa = await _context.Pessoas.Include(p => p.Propriedades).Where(p => p.Cpf == cpf).FirstOrDefaultAsync();
    return pessoa;
}

public async Task Add(Pessoa pessoa)
{
    if (pessoa != null)
    {
        await _context.Pessoas.AddAsync(pessoa);
    }
}

public async Task Salvar()
{
    await _context.SaveChangesAsync();
}
}
```



Implementação da Classe PropriedadeService e seus serviços



```
using AppInventario.Contexto;  
using AppInventario.Models;  
  
namespace AppInventario.Services  
{  
    public class PropriedadeService  
    {  
        private readonly ContextoBD _context;  
  
        public PropriedadeService(ContextoBD con)  
        {  
            _context = con;  
        }  
  
        public async Task Add(List<Propriedade> bens)  
        {  
            if (bens != null)  
            {  
                await _context.Propriedades.AddRangeAsync(bens);  
            }  
        }  
    }  
}
```



Continuação da Implementação da Classe PropriedadeService

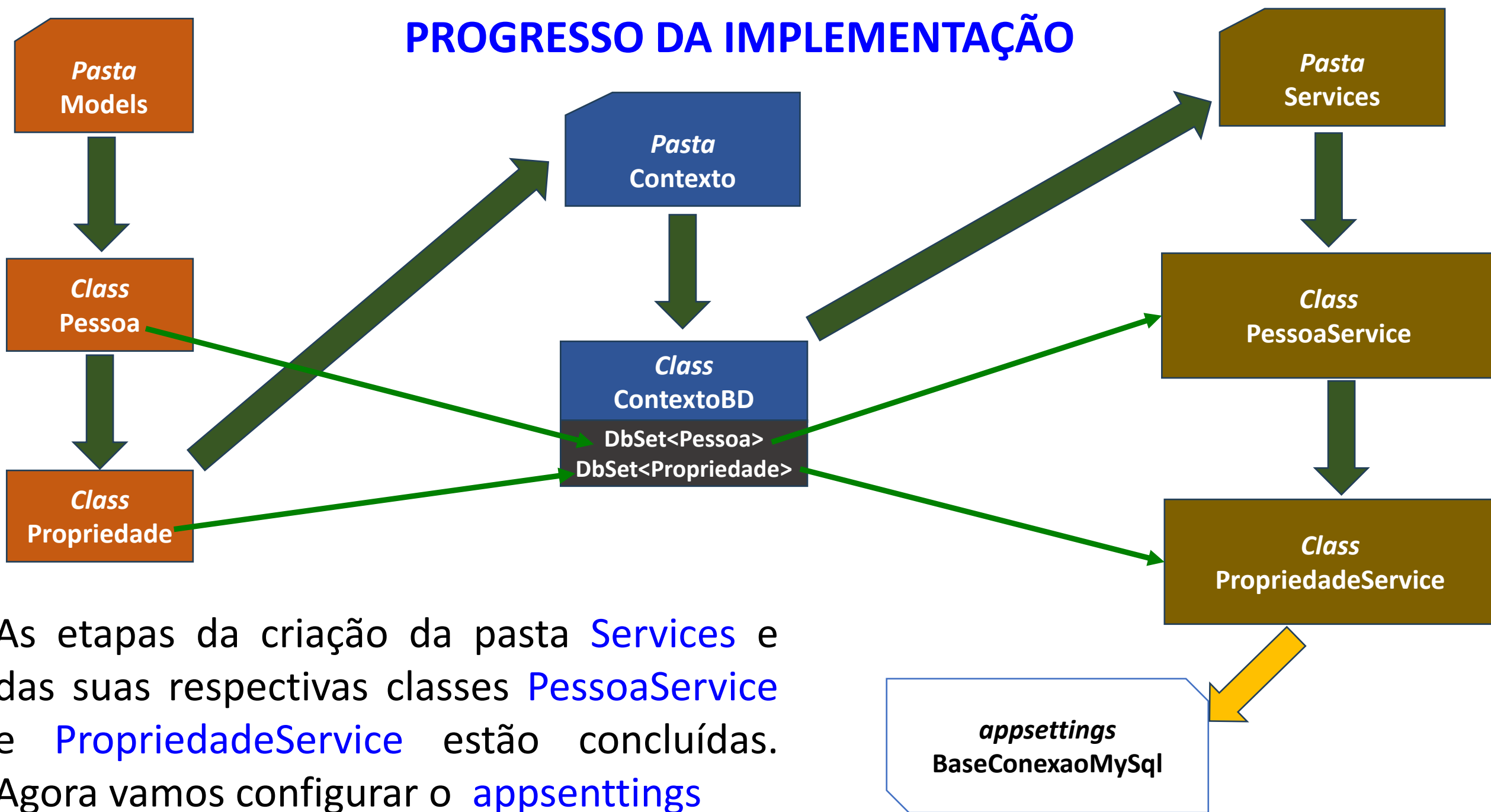


```
public async Task Add(Propriedade bens)
{
    if (bens != null)
    {
        await _context.Propriedades.AddAsync(bens);
    }
}

public async Task Salvar()
{
    await _context.SaveChangesAsync();
}

//Com Include, o nome do dono da propriedade também será carregado
public async Task<List<Propriedade>> Propriedades()
{
    var p = await _context.Propriedades.Include(p => p.Pessoa).ToListAsync();
    return p;
}
```

PROGRESSO DA IMPLEMENTAÇÃO



As etapas da criação da pasta **Services** e das suas respectivas classes **PessoaService** e **PropriedadeService** estão concluídas. Agora vamos configurar o **appsettings**



Configuração da Conexão com o Mysql “BaseConexaoMySQL”

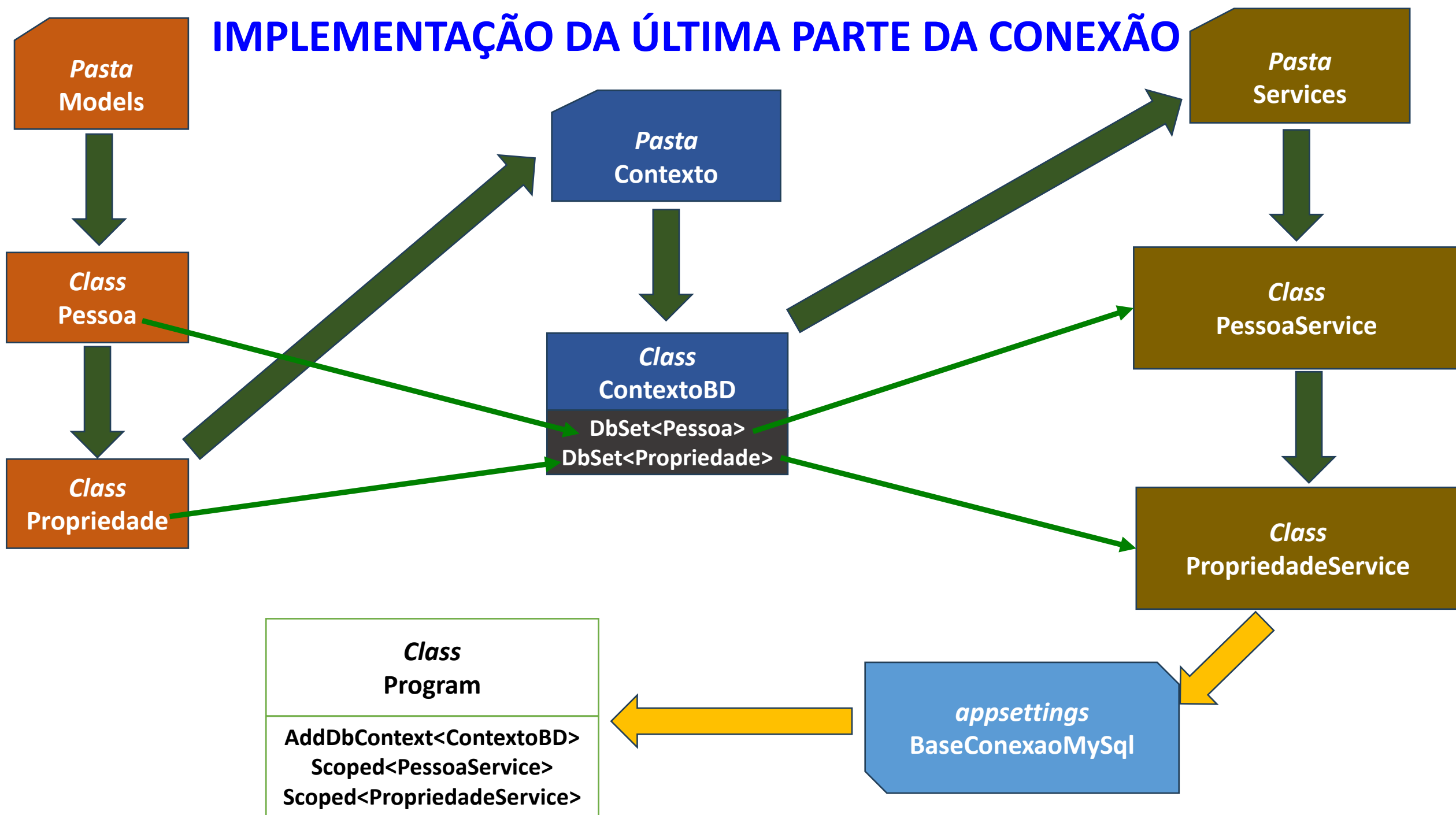


```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "BaseConexaoMySQL": "server=localhost;port=3306;database=bd_inventario;user=root;password=root;Persist Security Info=False;Connect Timeout=300;"
  },
  "AllowedHosts": "*"
}
```

Utilize o arquivo [Conexão MySQL](#) disponibilizado com os Slides para configurar a conexão.

Obs.: Sempre verifique a port = 3306 (padrão).
Nos laboratórios, a port = 3360.

IMPLEMENTAÇÃO DA ÚLTIMA PARTE DA CONEXÃO



Configuração da classe Program



```
using AppInventario.Contexto;
using AppInventario.Data;
using AppInventario.Services;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSingleton<WeatherForecastService>();

builder.Services.AddScoped<PessoaService>();
builder.Services.AddScoped<PropriedadeService>();

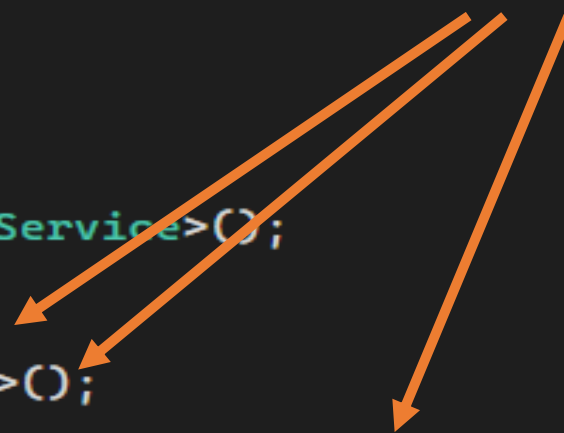
string mySqlConexao = builder.Configuration.GetConnectionString("BaseConexaoMySQL");
builder.Services.AddDbContextPool<ContextoBD>(options =>
options.UseMySQL(mySqlConexao, ServerVersion.AutoDetect(mySqlConexao)));

//##### abaixo a conexao para postgresQL #####
//builder.Services.AddEntityFrameworkNpgsql().AddDbContext<ContextoBD>(options =>
//options.UseNpgsql(builder.Configuration.GetConnectionString("BaseConexaoPostgreSQL")));

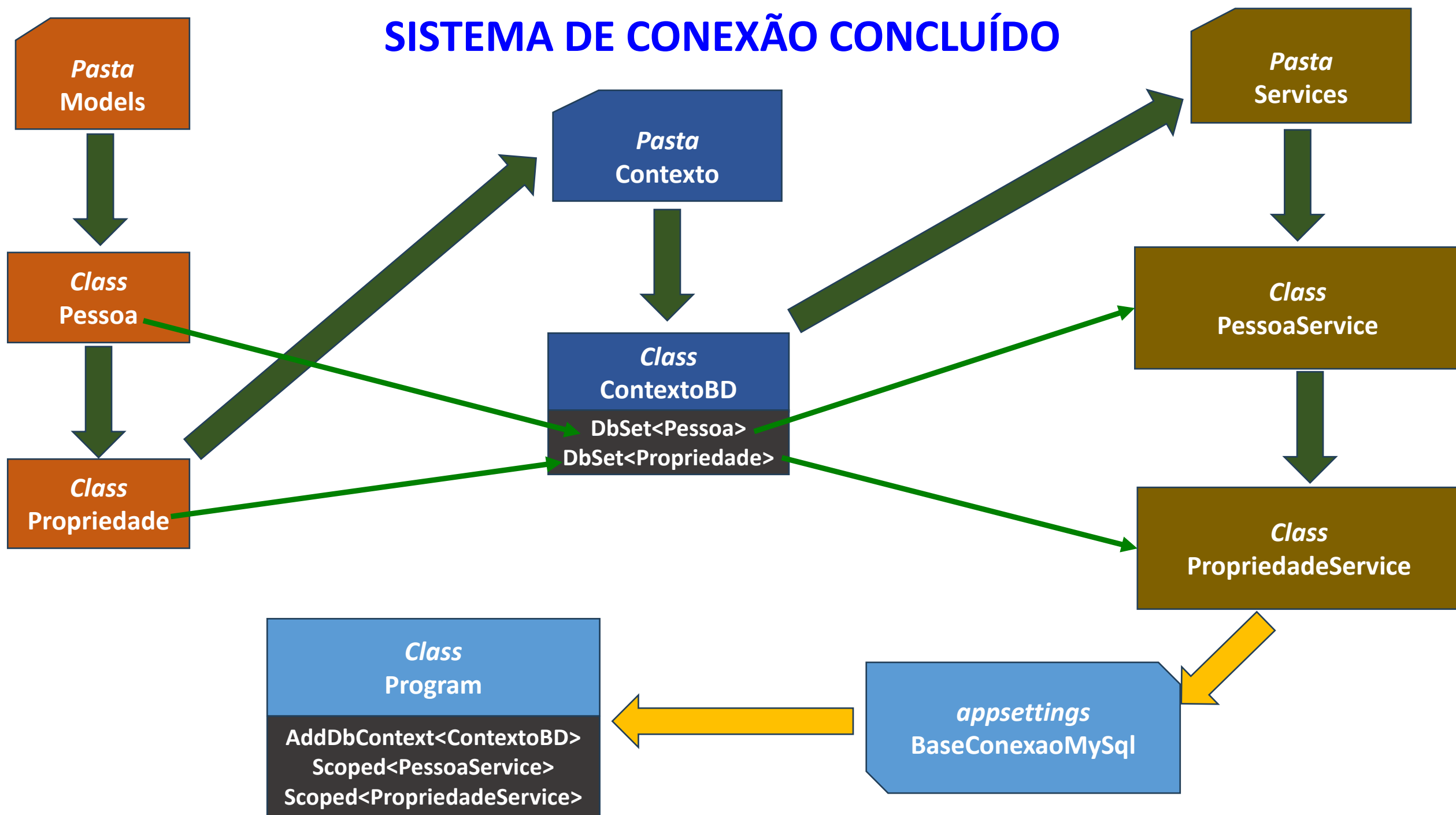
var app = builder.Build();

// Configure the HTTP request pipeline.
```

Implementações

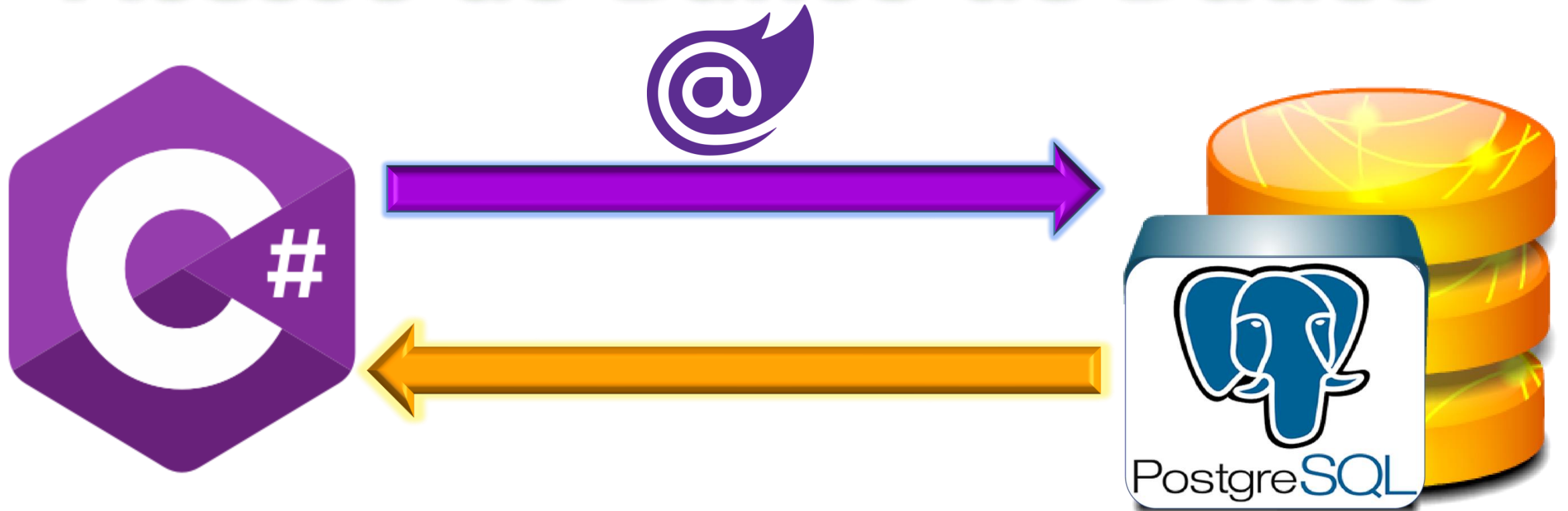


SISTEMA DE CONEXÃO CONCLUÍDO



2ª Fase da implementação

Acesso ao Banco de Dados





LEGENDA DO FLUXOGRAMA



O próximo slide tem o objetivo de demonstrar a visão geral do **sistema com acesso ao banco de dados**. Logo abaixo, segue legenda e as observações para a correta leitura do fluxograma:



Indica qual o próximo item que você deve criar.



Indica a conexão por injeção (**@inject**)



Indica a conexão que existe entre as classes

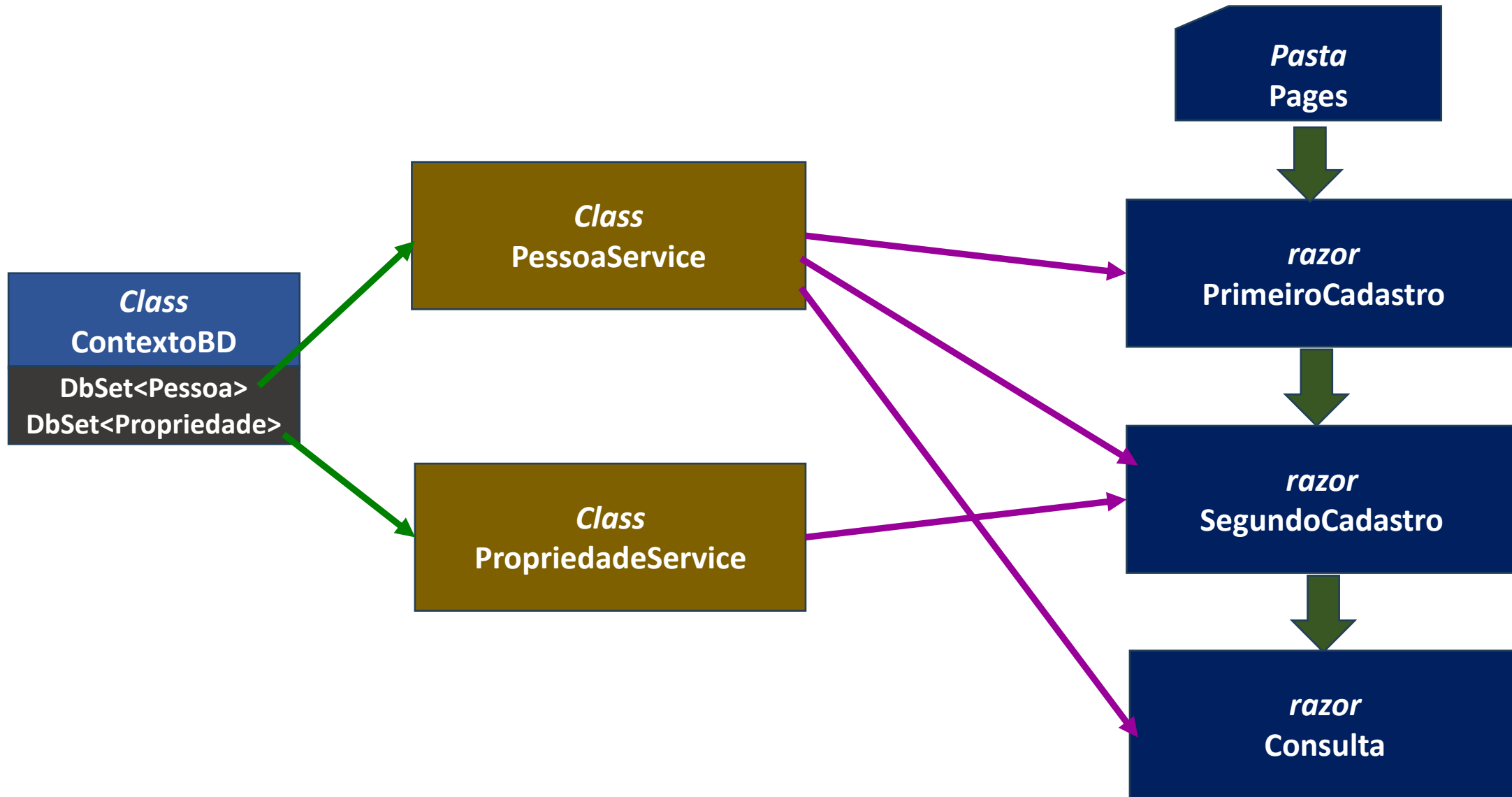


Indica qual o próximo item existente que será editado.

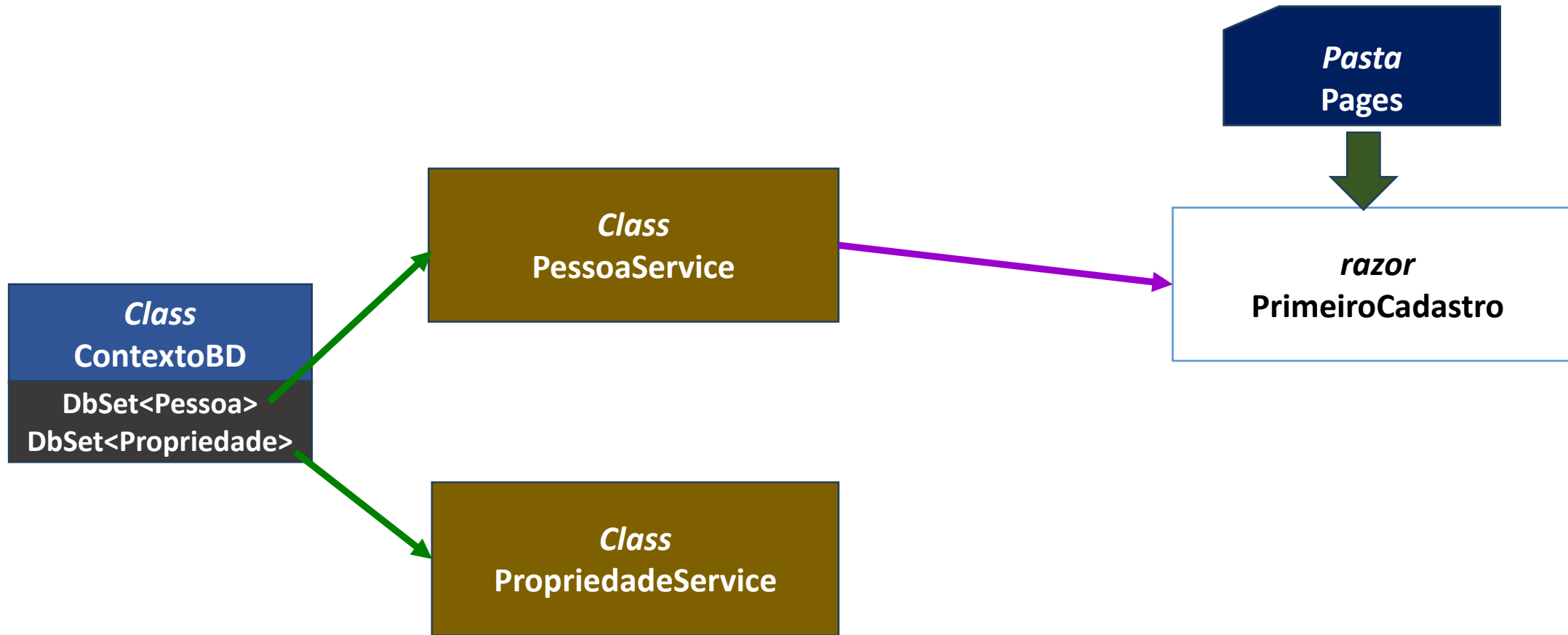
Obs1: A classe tem a mesma cor da pasta a qual ela pertence.

Obs2: O fluxograma começa com a criação da pasta Models.

VISÃO GERAL DO SISTEMA DO ACESSO AO BANCO DE DADOS



Dentro da pasta Pages, Implemente “PrimeiroCadastro.razor”





```
@page "/primeiroCadastro"
```

```
@using AppInventario.Models;
```

```
@using AppInventario.Services;
```

```
@inject PessoaService pessoaService;
```

```
@inject NavigationManager navegacao;
```

Acessa o banco de dados

```
<h3>PRIMEIRO REGISTRO DE PESSOA COM PROPRIEDADES</h3>
```



Continuação da implementação do “PrimeiroCadastro.razor”



```
= @code {  
    private string? Descricao { get; set; }  
    private double? Valor { get; set; }  
    private bool novo = false;  
    private bool campoBloqueado = false;  
    private Pessoa? pessoa = new Pessoa();  
    private List<Propriedade> bensRegistrados = new List<Propriedade> ();  
  
    private void AddBens()  
    {  
        Propriedade bem = new Propriedade();  
        bem.Descricao = Descricao;  
        bem.Valor = Valor;  
        bensRegistrados.Add(bem);  
        Descricao = null; Valor = null;  
    }  
}
```



Continuação da implementação do “PrimeiroCadastro.razor”



```
bensRegistrados.Add(bem);  
Descricao = null; Valor = null;
```

```
public async Task Salvar()
```

```
{  
    pessoa.DataNasc = pessoa.DataNasc.Value.ToUniversalTime();  
    pessoa.Propriedades = bensRegistrados;  
    await pessoaService.Add(pessoa);  
    await pessoaService.Salvar();  
    novo = true;  
    campoBloqueado = true;  
}
```

```
public void NovoRegistro()
```

```
{  
    novo = false;  
    //navegacao.NavigateTo("/cadastro");  
    navegacao.NavigateTo("/primeiroCadastro", forceLoad: true);  
}
```


Use o código Html disponibilizado para implementar o “PrimeiroCadastro”,

```
<h3>PRIMEIRO REGISTRO DE PESSOA COM PROPRIEDADES</h3>
```

```
<div class="container">
  <nav class="navbar navbar-light" style="background-color: darkgreen"></nav>

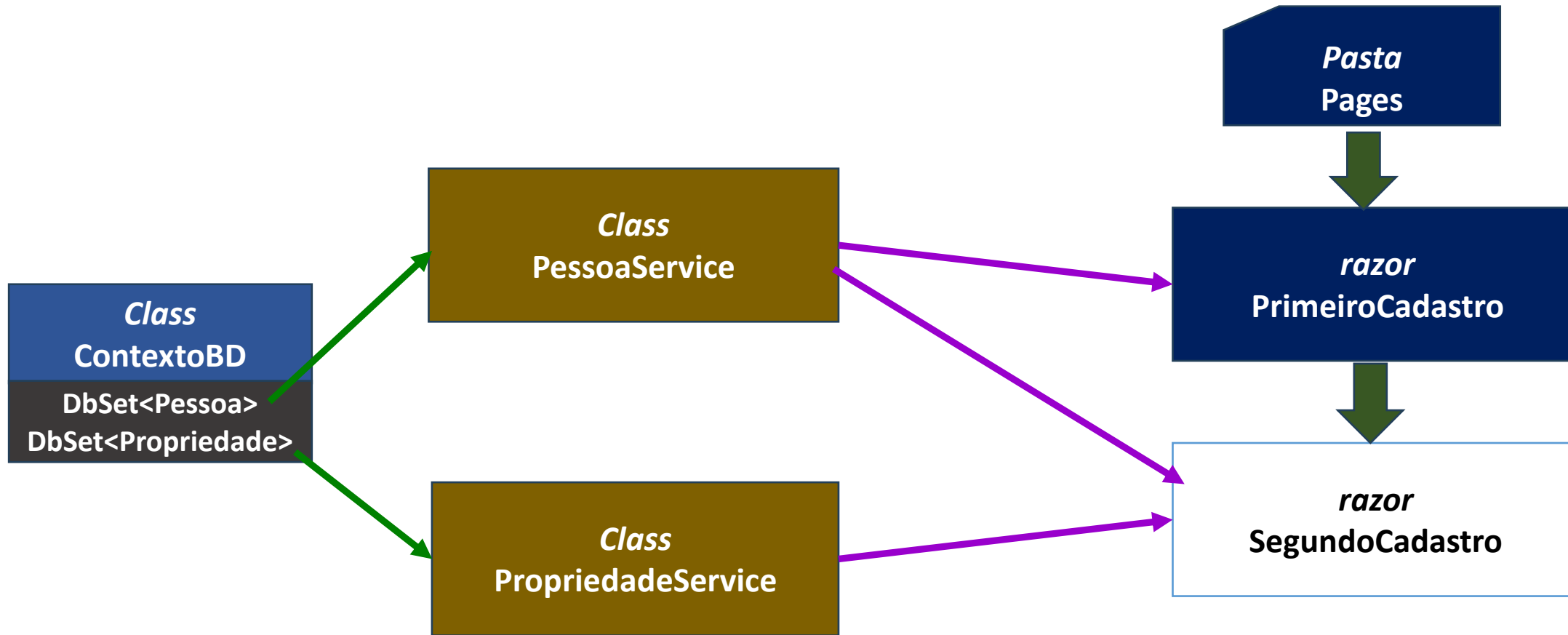
  <div class="mb-3">
    <label for="exampleFormControlInput1" class="form-label">NOME:</label>
    <input @bind="pessoa.Nome" type="text" class="form-control" id="exampleFormControlInput1" disabled="@campoBloqueado">
  </div>
  <div class="row align-items-start">...

  <nav class="navbar navbar-light" style="background-color: white"></nav>
  @if (novo == false)...
```

```
<nav class="navbar navbar-light" style="background-color: white"></nav>
@if (novo)
{
  <div class="alert alert-success" role="alert">
    Salvo com Sucesso;
  </div>
  <button type="button" class="btn btn-primary" @onclick="NovoRegistro">NOVO REGISTRO</button>
}
```

```
<h4 style="text-align:center">Bens Cadastrados</h4>
<table class="table">
  <thead>
    <tr>
      <th>Descrição</th>
      <th>Valor</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var item in bensRegistrados)
    {
      <tr>
        <td>@item.Descricao</td>
        <td>@item.Valor</td>
      </tr>
    }
  </tbody>
</table>
```

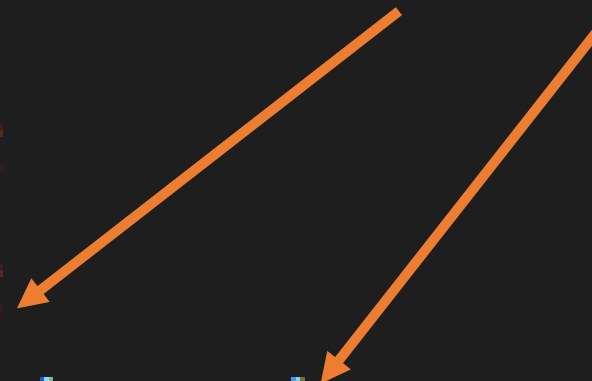
Dentro da pasta Pages, Implemente “SegundoCadastro.razor”





```
@page "/segundoCadastro"  
@using AppInventario.Models;  
@using AppInventario.Services;  
@inject NavigationManager navegacao;  
@inject PessoaService pessoaService;  
@inject PropriedadeService propriedadeService;
```

Acessa o banco de dados



```
<h3>REGISTRO DE PROPRIEDADES DE PESSOAS COM CADASTRO</h3>
```



```
@code {  
    private string? Descricao { get; set; }  
    private double? Valor { get; set; }  
    private bool novo = false;  
    private List<Pessoa> Pessoas { get; set; } //lista de pessoas  
    private Pessoa? pessoa = new Pessoa();  
    private bool campoBloqueado = false;  
    private List<Propriedade> bensRegistrados = new List<Propriedade>();  
  
    //Carrega a lista assim que a página acaba de abrir  
    protected override async Task OnInitializedAsync()  
    {  
        Pessoas = new List<Pessoa>();  
        var lista = await pessoaService?.Pessoas(); //buscando no banco  
        Pessoas = lista.ToList(); //lista de pessoas carregadas  
    }  
}
```



Continuação da implementação do “SegundoCadastro.razor”



```
var lista = await pessoaService?.Pessoas(); //buscando no banco
Pessoas = lista.ToList(); //lista de pessoas carregadas
```

```
public void SelecionarPessoas(EventArgs e)
{
    string cpf = e.Value.ToString();
    //seleciona uma pessoa específica para registrar os bens
    pessoa = Pessoas.Where(p => p.Cpf == cpf).FirstOrDefault();
}
```

```
private void AddBens() //método para colocar os bens na lista
{
    Propriedade bem = new Propriedade();
    bem.Descricao = Descricao;
    bem.Valor = Valor;
    bem.Pessoa = pessoa; ← Vínculo do bem com a pessoa (FK)
    bensRegistrados.Add(bem);
    Descricao = null; Valor = null;
}
```

Continuação da implementação do “SegundoCadastro.razor”



```
bensRegistrados.Add(bem);  
Descricao = null; Valor = null;
```

```
public async Task Salvar()
```

```
{  
    //salvar na tabela propriedade do banco de dados  
    await propriedadeService.Add(bensRegistrados);  
    await propriedadeService.Salvar();  
    novo = true;  
    campoBloqueado = true;  
}
```

```
public void NovoRegistro()
```

```
{  
    novo = false;  
    //navegacao.NavigateTo("/cadastro");  
    navegacao.NavigateTo("/segundoCadastro", forceLoad: true);  
}
```

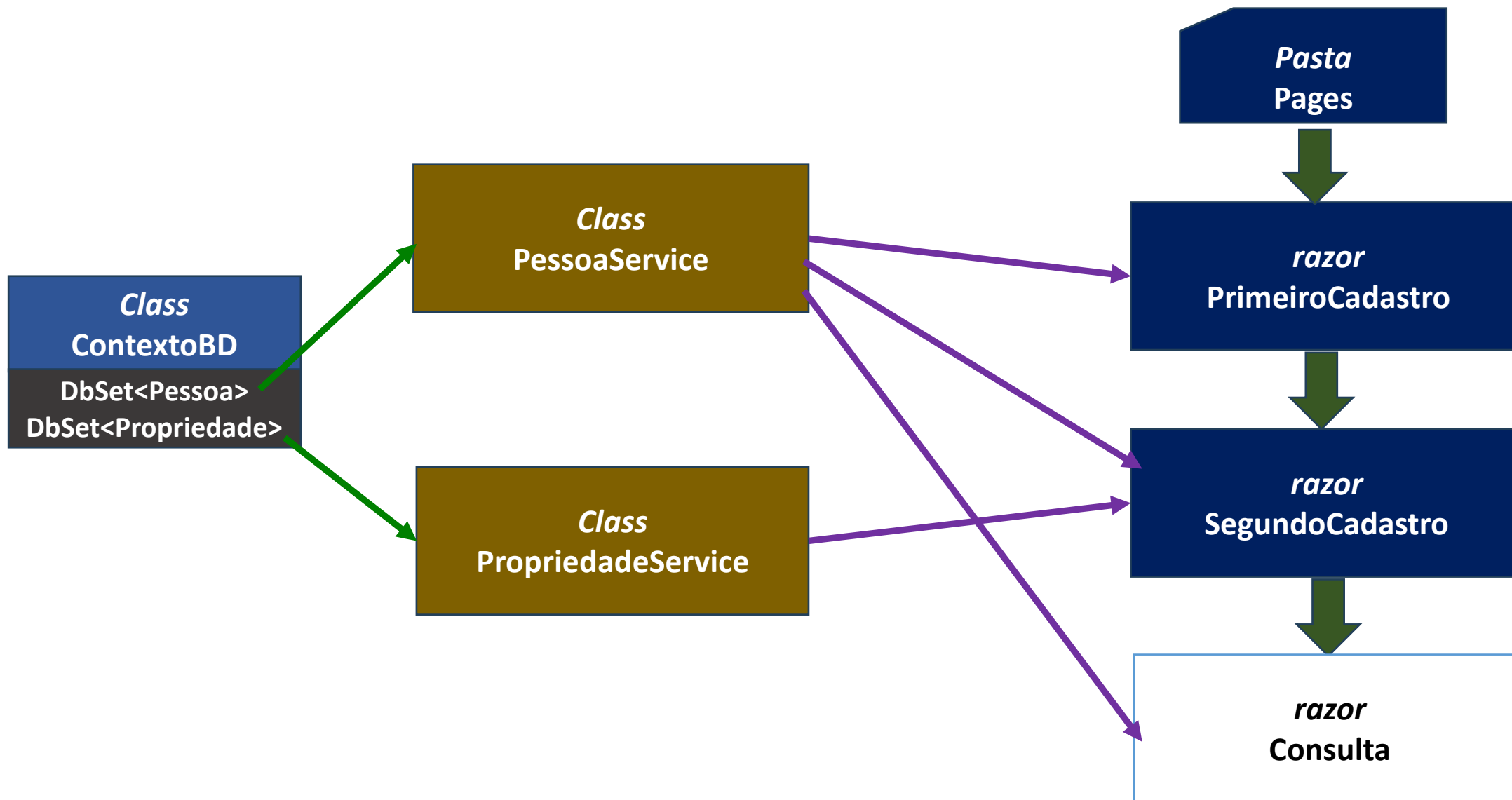
Use o código Html disponibilizado para implementar o “SegundoCadastro”,

```
<div class="container">
  <nav class="navbar navbar-light" style="background-color: ■darkgreen"></nav>
  <div class="form-group">
    <label for="input-datalist">Buscar</label>
    <input type="text" class="form-control" placeholder="Buscar" list="lista-pessoas" @onchange="SelecionarPessoas">
    <datalist id="lista-pessoas">
      @foreach (var item in Pessoas)
      {
        <option value="@item.Cpf">@item.Nome</option>
      }
    </datalist>
  </div>
  <nav class="navbar navbar-light" style="background-color: ■white"></nav>
  <nav class="navbar navbar-light" style="background-color: ■darkgreen"></nav>
  <nav class="navbar navbar-light" style="background-color: ■white"></nav>
  <div class="mb-3">
    <label for="exampleFormControlInput1" class="form-label">NOME:</label>
    <input @bind="pessoa.Nome" type="text" class="form-control" id="exampleFormControlInput1" disabled="true">
  </div>
</div>
```

```
<nav class="navbar navbar-light" style="background-color: ■white"></nav>
@if (novo)
{
  <div class="alert alert-success" role="alert">
    Salvo com Sucesso;
  </div>
  <button type="button" class="btn btn-primary" @onclick="NovoRegistro">NOVO REGISTRO</button>
}
</div>
```

```
<h4 style="text-align:center">Bens Cadastrados</h4>
<table class="table">
  <thead>
    <tr>
      <th>Descrição</th>
      <th>Valor</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var item in bensRegistrados)
    {
      <tr>
        <td>@item.Descricao</td>
        <td>@item.Valor</td>
      </tr>
    }
  </tbody>
</table>
</div>
```

Dentro da pasta Pages, Implemente “Consulta.razor”





Dentro da pasta Pages, Implemente “Consulta.razor”



```
@page "/consulta"
@using AppInventario.Models;
@using AppInventario.Services;
@inject PessoaService pessoaService;
@inject NavigationManager navegacao;

<h3>CONSULTA DE PESSOAS</h3>
```

Acessa o banco de dados





Continuação da implementação da “Consulta.razor”



```
@code {  
    private string? Descricao { get; set; }  
    private double? Valor { get; set; }  
    private bool novo = false;  
    private List<Pessoa>? Pessoas { get; set; }  
    private Pessoa? pessoa = new Pessoa();  
    private bool campoBloqueado = false;  
    private List<Propriedade> bensRegistrados = new List<Propriedade>();  
  
    protected override async Task OnInitializedAsync()  
    {  
        Pessoas = new List<Pessoa>();  
        var lista = await pessoaService?.Pessoas();  
        Pessoas = lista.ToList();  
    }  
}
```

Continuação da implementação da “Consulta.razor”



```
Pessoas = lista.ToList();

public void SelecionarPessoas(ChangeEventArgs e)
{
    string cpf = e.Value.ToString();
    var p = Pessoas.Where(p => p.Cpf == cpf).FirstOrDefault();
    if (p != null)
    {
        pessoa = p;
        bensRegistrados = pessoa.Propriedades;
    }
}
```

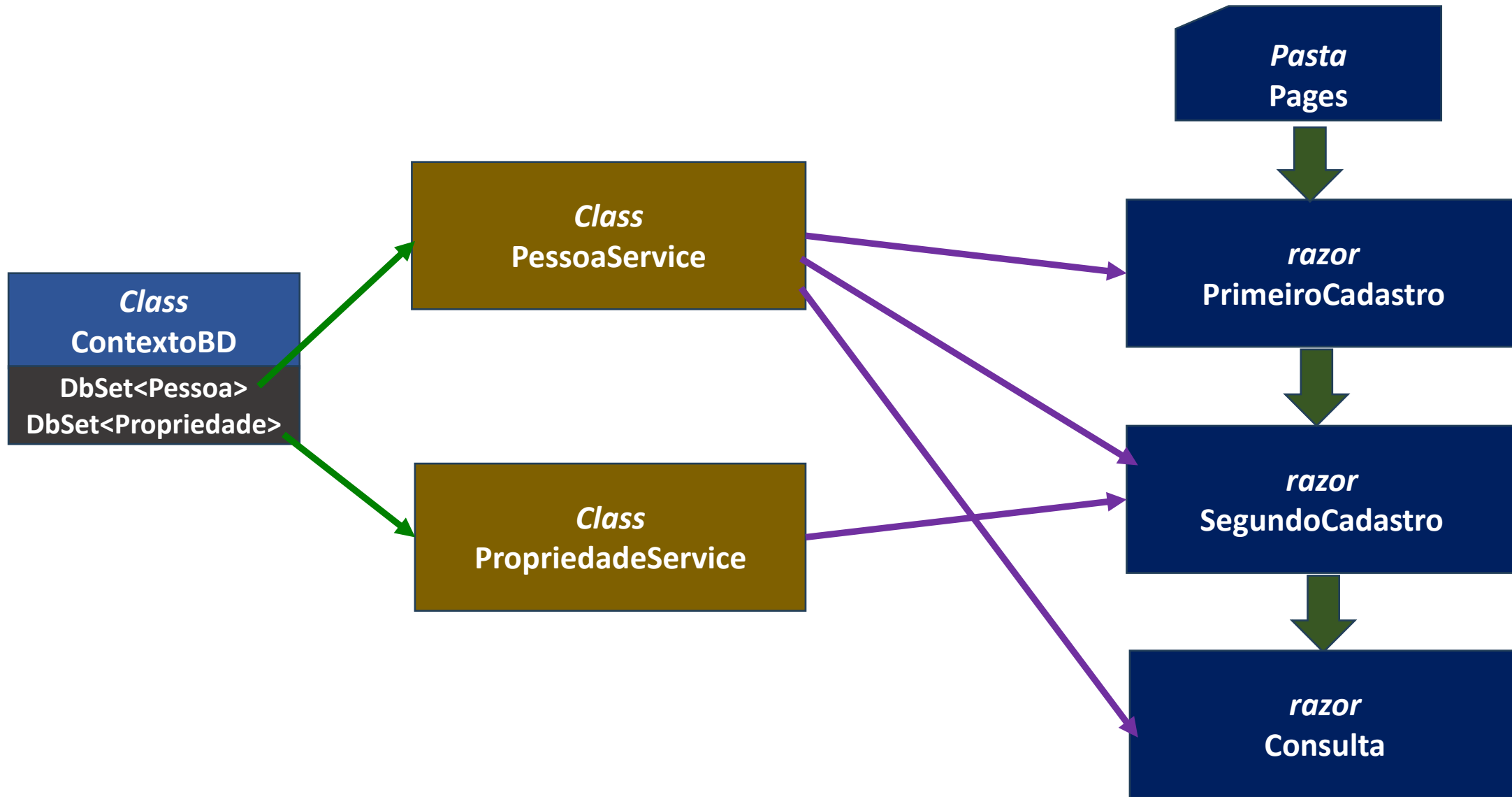
Observe que não foi preciso usar o [PropriedadeService](#) uma vez que o comando [Include](#) no [PessoaService](#) incluiu as propriedades conforme imagem abaixo:

```
public async Task<List<Pessoa>>? Pessoas()
{
    var pessoas = await _context.Pessoas.Include(p=>p.Propriedades).ToListAsync();
    return pessoas;
}
```

Use o código Html disponibilizado para implementar a “Consulta”,

```
<div class="container">
  <nav class="navbar navbar-light" style="background-color: ■darkgreen"></nav>
  <div class="form-group">
    <label for="input-datalist">Buscar</label>
    <input type="text" class="form-control" placeholder="Buscar" list="lista-pessoas" @onchange="SelecionarPessoas">
    <datalist id="lista-pessoas">
      @foreach (var item in Pessoas)
      {
        <option value="@item.Cpf">@item.Nome</option>
      }
    </datalist>
  </div>
  <nav class="navbar navbar-light" style="background-color: ■white"></nav>
  <nav class="navbar navbar-light" style="background-color: ■darkgreen"></nav>
  <nav class="navbar navbar-light" style="background-color: ■white"></nav>
  <div class="mb-3">
    <label for="exampleFormControlInput1" class="form-label">NOME:</label>
    <input @bind="pessoa.Nome" type="text" class="form-control" id="exampleFormControlInput1" disabled="true">
  </div>
```

CONCLUSÃO DO ACESSO AO BANCO DE DADOS





ATIVIDADE



1 – Mostrar a soma total de todos bens registrados de uma pessoa selecionada na página da consulta.

2 - Criar uma página de consulta onde permita que o usuário possa listar todos os bens de um pessoa selecionada, mas que também permita:

2.1 – Filtrar os bens acima de uma valor informado pelo usuário

2.2 – Filtrar os bens por uma descrição informada pelo usuário

2.3 – Mostrar o total dos bens

2.4 – Mostrar o total dos bens que foram filtrados. (não utilizar o mesmo filtro do item anterior)

3 – Criar uma página para listar os nomes das pessoas, Cpf's e a descrição da propriedade de maior valor de cada pessoa listada.