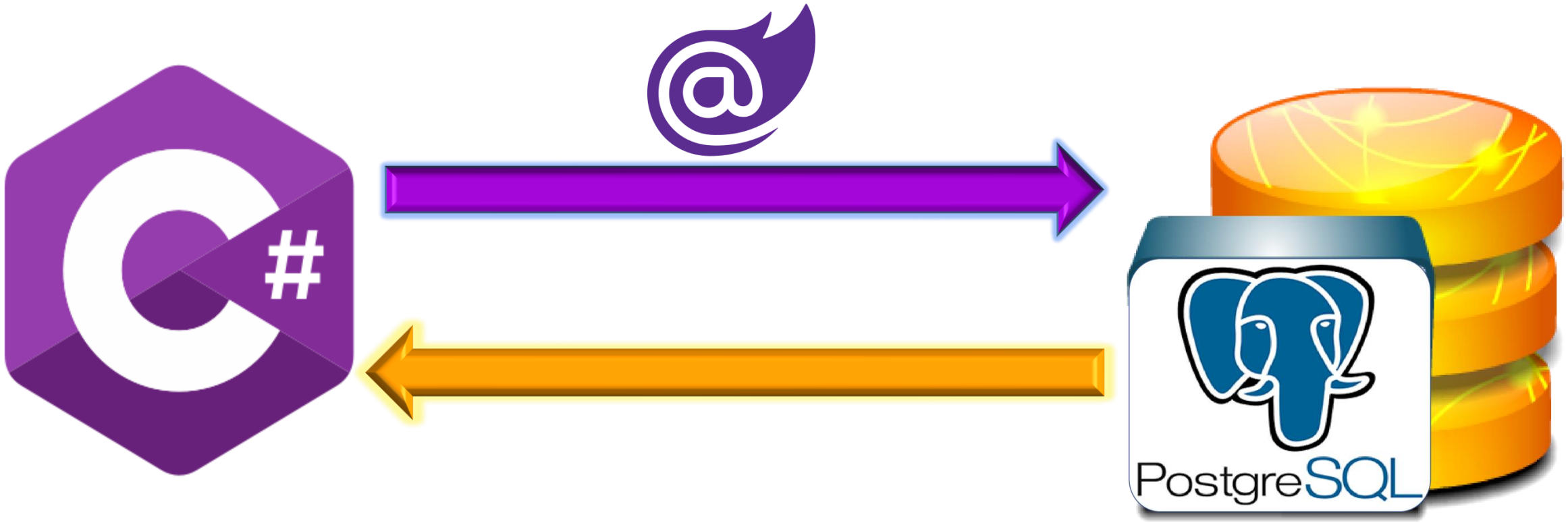
# Programação com Acesso a Banco de Dados



# AULA 6
# Implementação do Sistema de Multas

# PROPOSTA DO SISTEMA

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

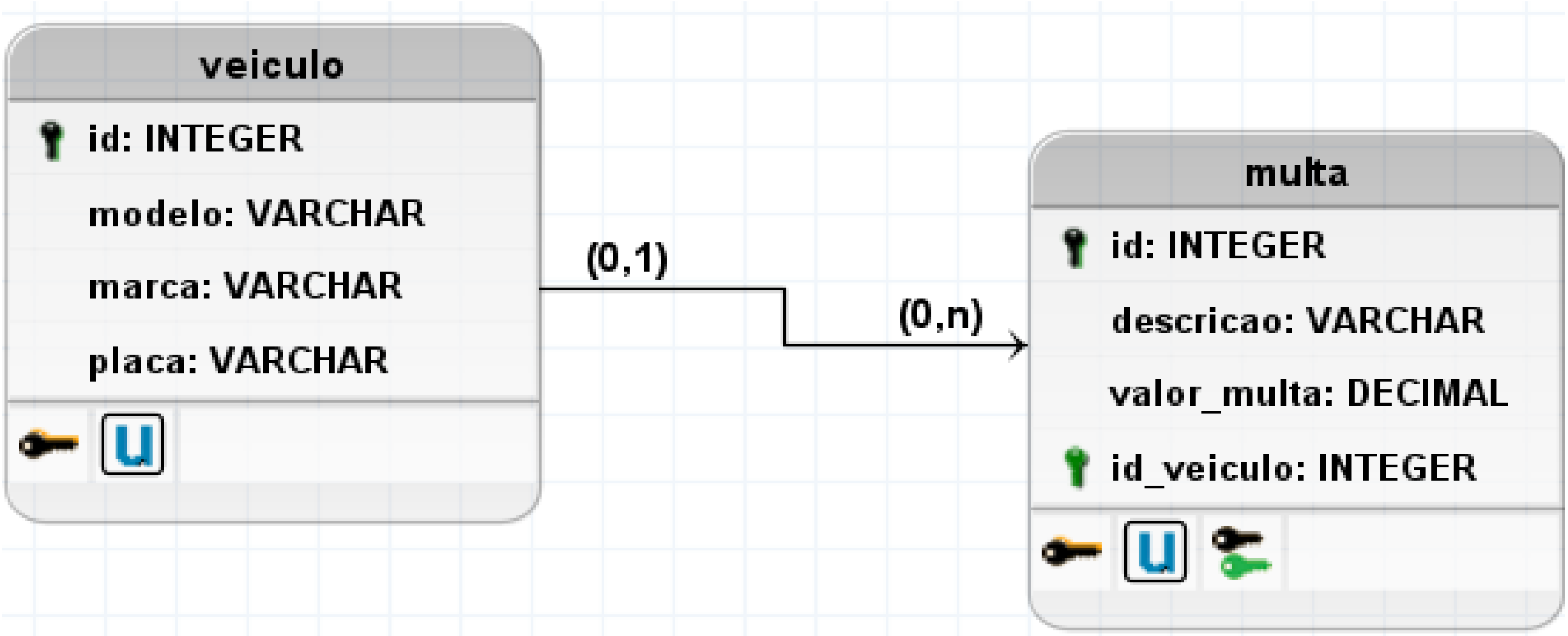
**Primeiro Cadastro**: O veículo será cadastrado pela primeira vez. Neste módulo, além do cadastro dos dados do veículo, também serão cadastrados as multas.

**Segundo Cadastro:** O veículo já está cadastrado e, neste caso, novas multas serão acrescentadas, e acordo com o veículo selecionado.

**Consulta:** Selecionará um veículo e listar todas as multas, juntamente com a soma total das multas.

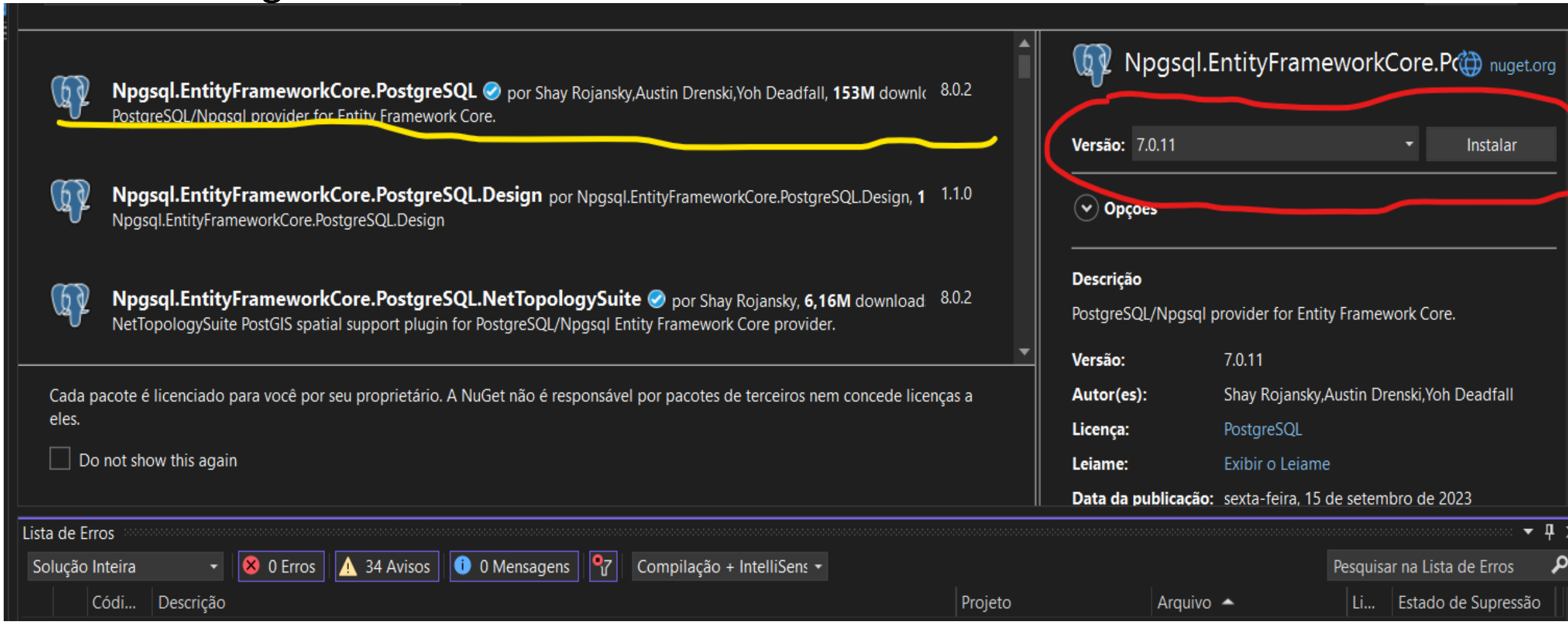
# BANCO DE DADOS MYSQL

Utilize o arquivo “bdMultaSQL.Text” disponibilizado com os Slides para criar o banco de dados **BDMultа** em PostgreSQL

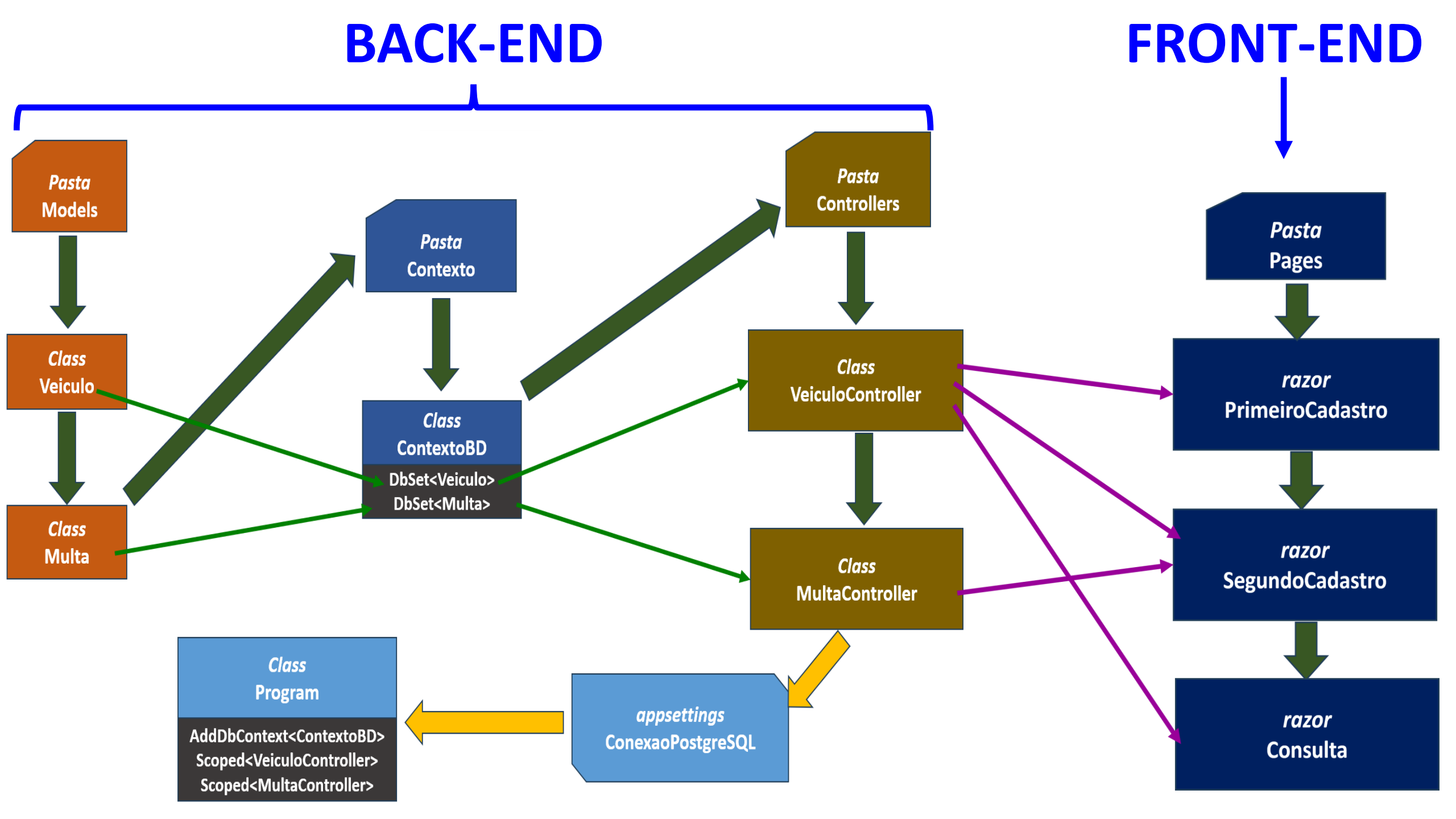


# CONFIGURAR O SISTEMA COM ENTITY FRAMEWORK

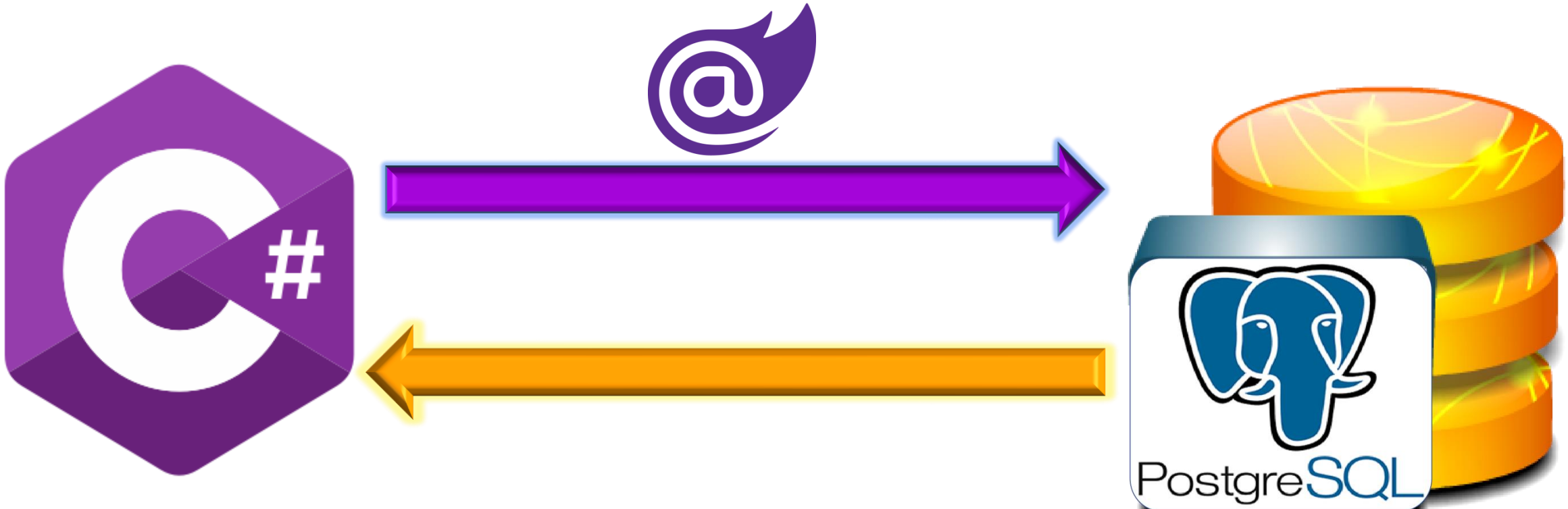
Clique com o botão direito do mouse sobre o nome do projeto (AppMulta) e depois vá na opção Gerenciar Pacotes do NuGET. Depois selecione a opção conforme imagem abaixo:



# Visão geral do Sistema (back-end) => (front-end)


BACK-END
FRONT-END
Pasta
Models
Pasta
Contexto
Pasta
Controllers
Pasta
Pages
Class
Veiculo
Class
ContextoBD
DbSet<Veiculo>
DbSet<Multa>
Class
VeiculoController
razor
PrimeiroCadastro
Class
Multa
Class
MultaController
razor
SegundoCadastro
Class
Program
AddDbContext<ContextoBD>
Scoped<VeiculoController>
Scoped<MultaController>
appsettings
ConexaoPostgreSQL
razor
Consulta

# 1ª Fase da implementação
# Conexão com o Banco de Dados
# (back-end)



# LEGENDA DO FLUXOGRAMA

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

Indica qual o próximo item que você deve criar.

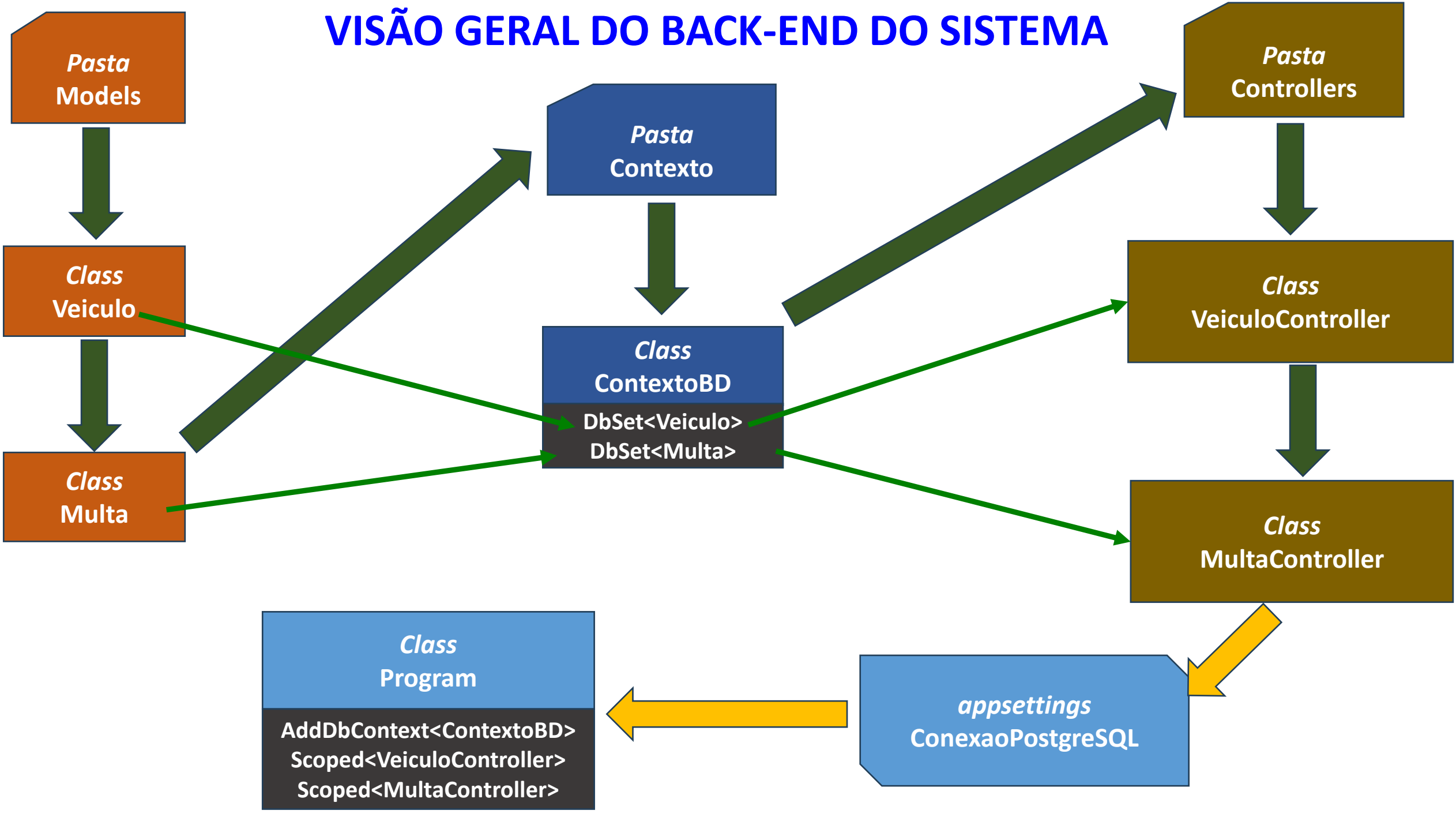
Indica a conexão que existe entre as classes

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 BACK-END DO SISTEMA



# Implementação da classe “Veiculo” com mapeamento

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

namespace AppMulta.Models
{
    [Table("veiculo", Schema = "public")]
    public class Veiculo
    {
        [Column("id")]
        public int Id { get; set; }
        [Column("modelo")]
        public string? Modelo { get; set; }
        [Column("marca")]
        public string? Marca { get; set; }
        [Column("placa")]
        public string? Placa { get; set; }
    }
}
```

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

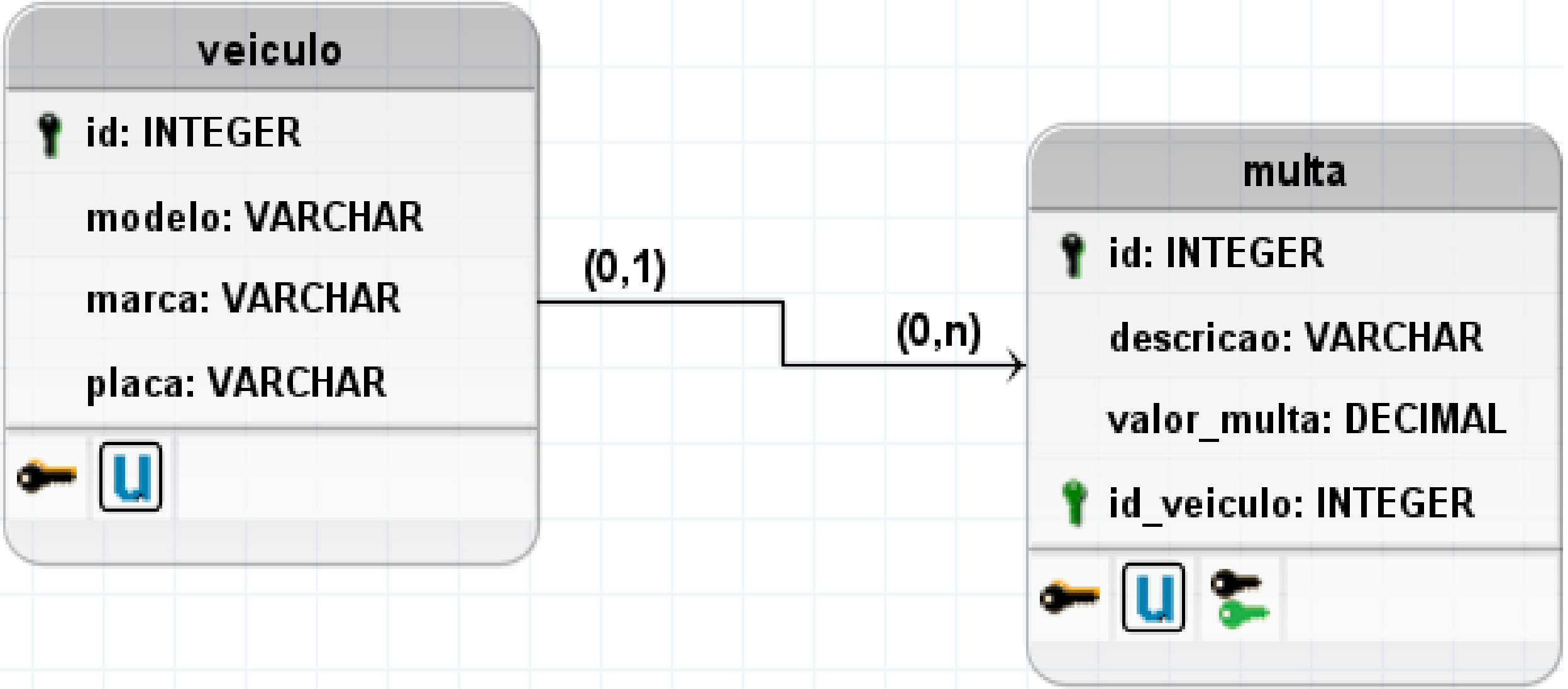
# Implementação da classe “Multa” com mapeamento

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

namespace AppMulta.Models
{
    [Table("multa", Schema = "public")]
    public class Multa
    {
        [Column("id")]
        public int Id { get; set; }
        [Column("descricao")]
        public string? Descricao { get; set; }
        [Column("valor_multa")]
        public decimal? ValorMulta { get; set; }
        [Column("id_veiculo")]
        public int IdVeiculo { get; set; }
    }
}
```

Obs.: Como a conexão é para Postgre, 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)



# Veiculo tem muitas Multas

```
[Table("veiculo", Schema = "public")]
public class Veiculo
{
    [Column("id")]
    public int Id { get; set; }
    [Column("modelo")]
    public string? Modelo { get; set; }
    [Column("marca")]
    public string? Marca { get; set; }
    [Column("placa")]
    public string? Placa { get; set; }

    public List<Multa>? Multas { get; set; }
}
```

Acrescentar o código

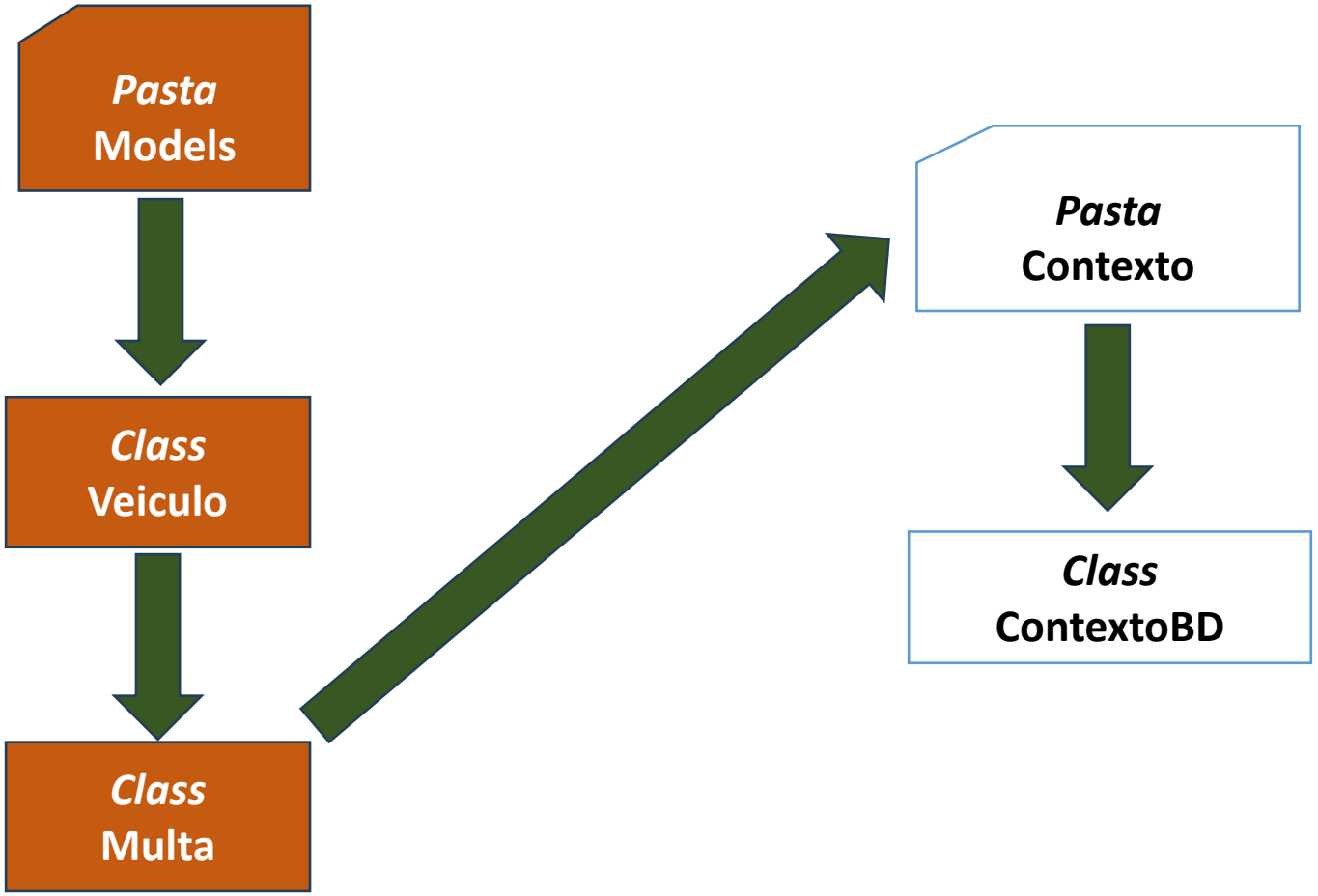
# Multa tem um Veículo

```
[Table("multa", Schema = "public")]
public class Multa
{
    [Column("id")]
    public int Id { get; set; }
    [Column("descricao")]
    public string? Descricao { get; set; }
    [Column("valor_multa")]
    public decimal? ValorMulta { get; set; }
    [Column("id_veiculo")]
    public int IdVeiculo { get; set; }

    [ForeignKey("IdVeiculo")]
    public Veiculo? Veiculo { get; set; }
}
```

Informa qual o atributo da classe vai armazenar a FK

Indica o dono da propriedade

# PROGRESSO DA IMPLEMENTAÇÃO

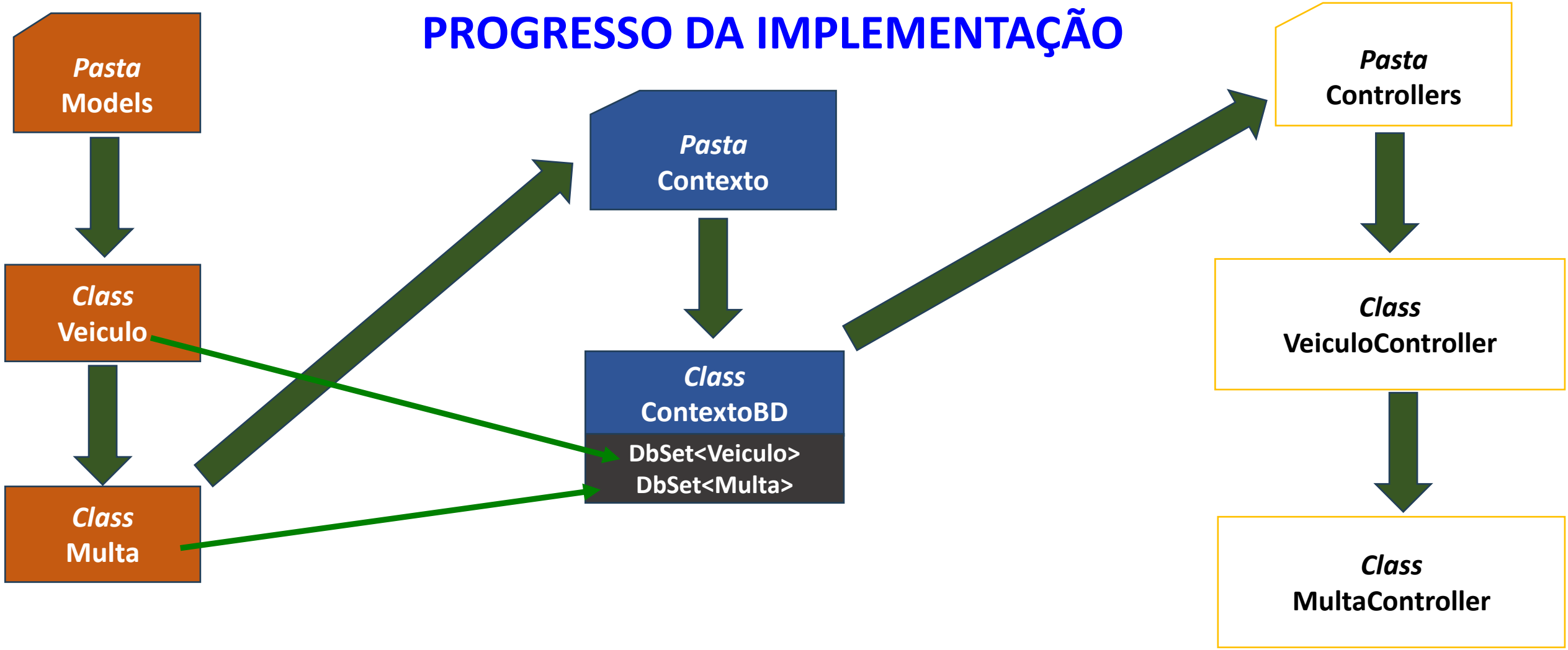



As etapas da criação da pasta Models e suas respectivas classes já foram concluídas, conforme o fluxograma. Próximo passo será a criação da pasta Contexto e da classe ContextoBD.

# Implementação da classe “ContextoBD” para conexão com BD

```
using AppMulta.Models;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;

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

        public DbSet<Veiculo> Veiculos { get; set; }
        public DbSet<Multa> Multas { 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 VeiculoController e MultaController.

# Implementação da Classe VeiculoController e seus controles

```
using AppMulta.Contexto;
using AppMulta.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace AppMulta.Controllers
{
    public class VeiculoController : Controller
    {
        private readonly ContextoBD _context;

        public VeiculoController(ContextoBD context)
        {
            _context = context;
        }

        public async Task<List<Veiculo>> ListaDeVeiculos()
        {
            var veiculos = await _context.Veiculos.Include(x => x.Multas).ToListAsync();
            return veiculos;
        }
```

# Continuação da Implementação da Classe VeiculoController

```
        public async Task<Veiculo> GetVeiculo(string placa)
        {
            var veiculo = await _context.Veiculos.Include(m => m.Multas).Where(v => v.Placa == placa).FirstOrDefaultAsync();
            return veiculo;
        }

        //método para cadastrar  veículos
        public async Task Add(Veiculo veiculo)
        {
            await _context.Veiculos.AddAsync(veiculo);
        }

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

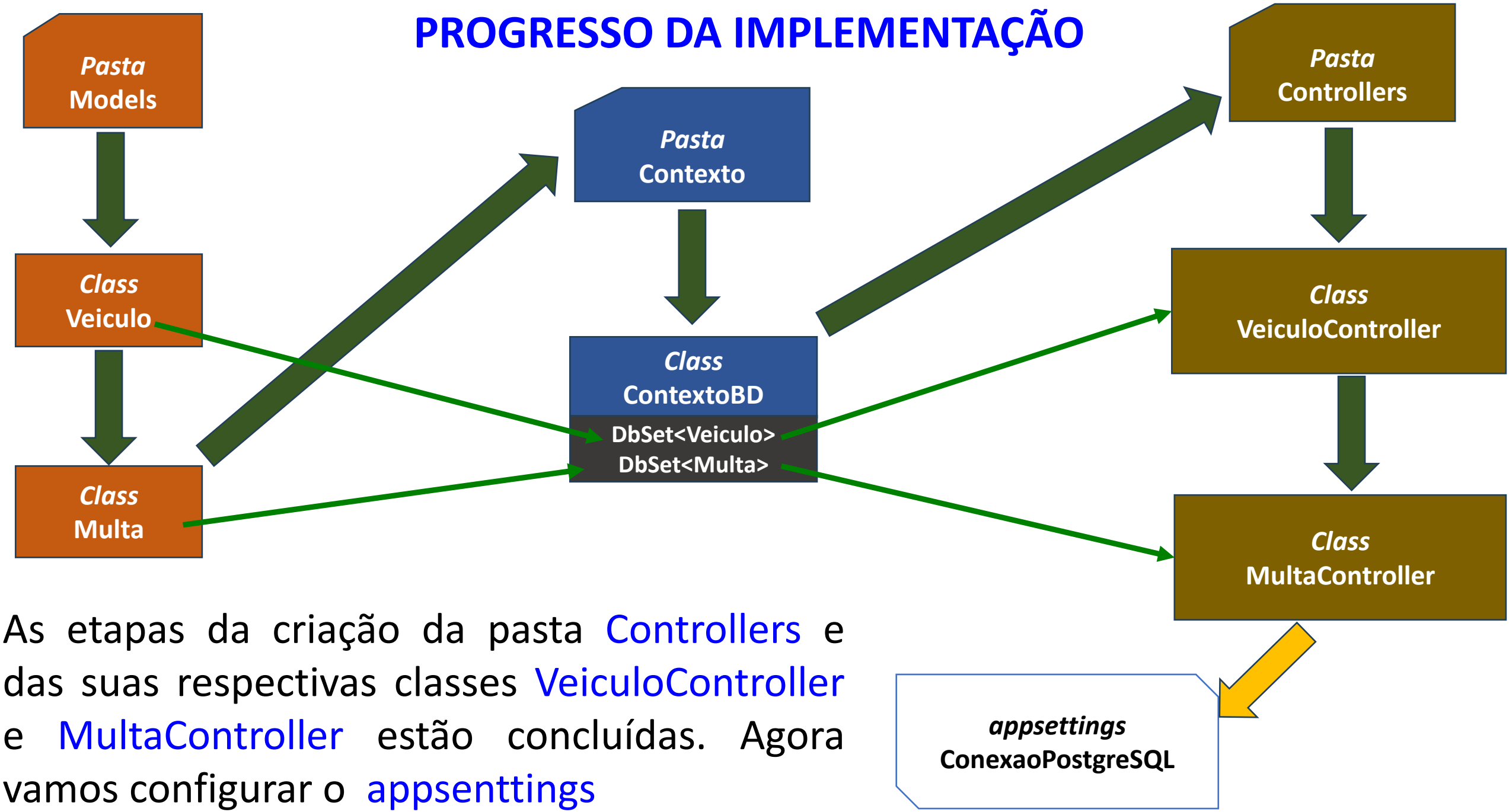
# Implementação da Classe MultaController e seus controles

```
using AppMulta.Contexto;
using AppMulta.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace AppMulta.Controllers
{
    public class MultaController : Controller
    {

        private readonly ContextoBD _context;

        public MultaController(ContextoBD context)
        {
            _context = context;
        }

        public async Task<List<Multa>> ListaDeMultas()
        {
            var multas = await _context.Multas.Include(v=>v.Veiculo).ToListAsync();
            return multas;
        }
```

# Continuação da Implementação da Classe MultaController

```
        }

        public async Task Add(Multa multa)
        {
            await _context.Multas.AddAsync(multa);
        }

        public async Task AddRanger(List<Multa> multas) //adicionar uma lista de multas
        {
            await _context.Multas.AddRangeAsync(multas);
        }

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

# PROGRESSO DA IMPLEMENTAÇÃO



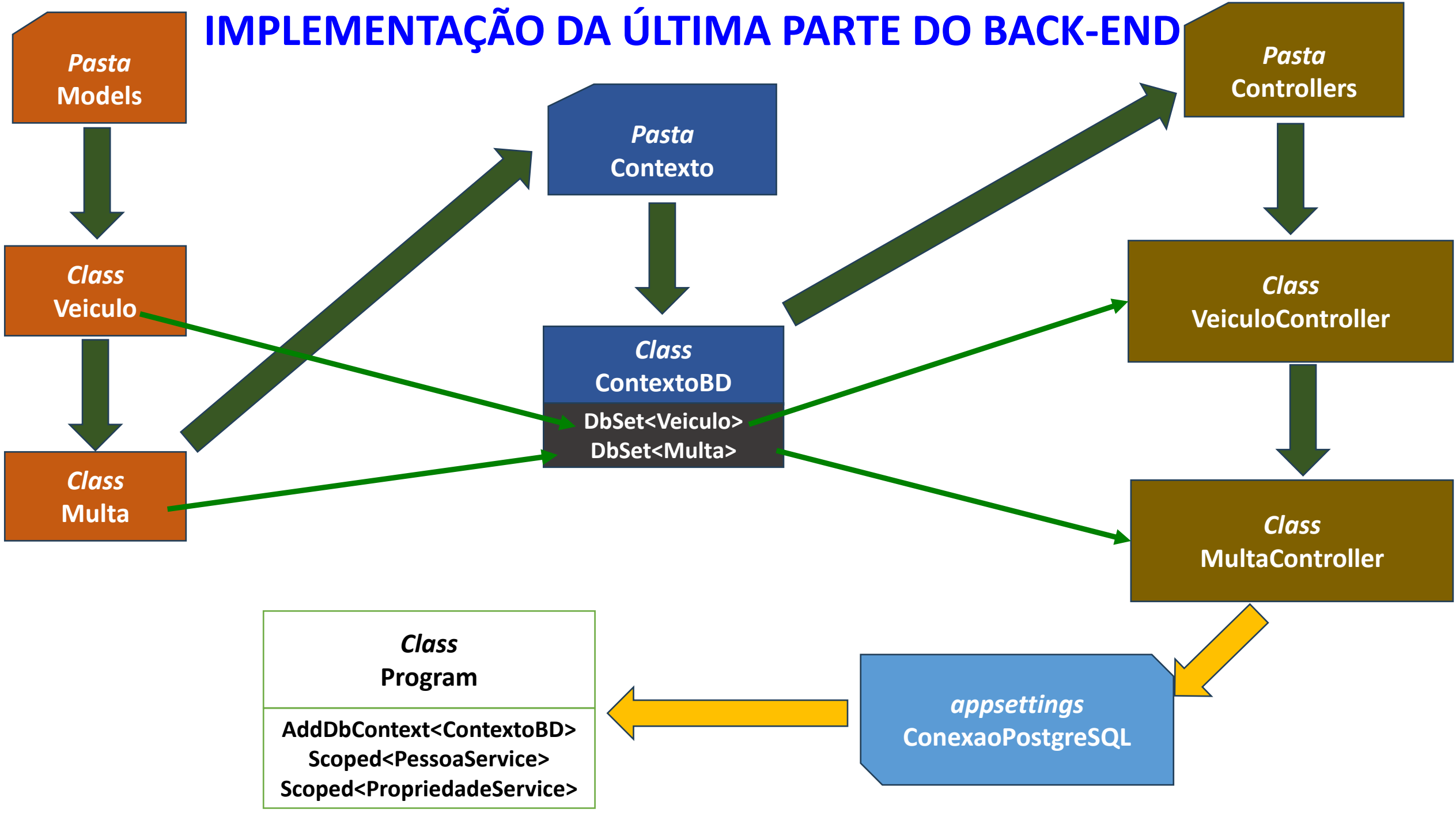

As etapas da criação da pasta Controllers e das suas respectivas classes VeiculoController e MultaController estão concluídas. Agora vamos configurar o appsenttings

# Configuração da Conexão com o Mysql “ConexaoPostgreSQL”

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "ConexaoPostgreSQL": "Host=localhost;Pooling=true;Port=5432;Database=BDMultas;User Id=postgres;Password=root;"
  },
  "AllowedHosts": "*"
}
```

Utilize o arquivo Conexão PostgreSQL disponibilizado com os Slides para configurar a conexão.

# IMPLEMENTAÇÃO DA ÚLTIMA PARTE DO BACK-END

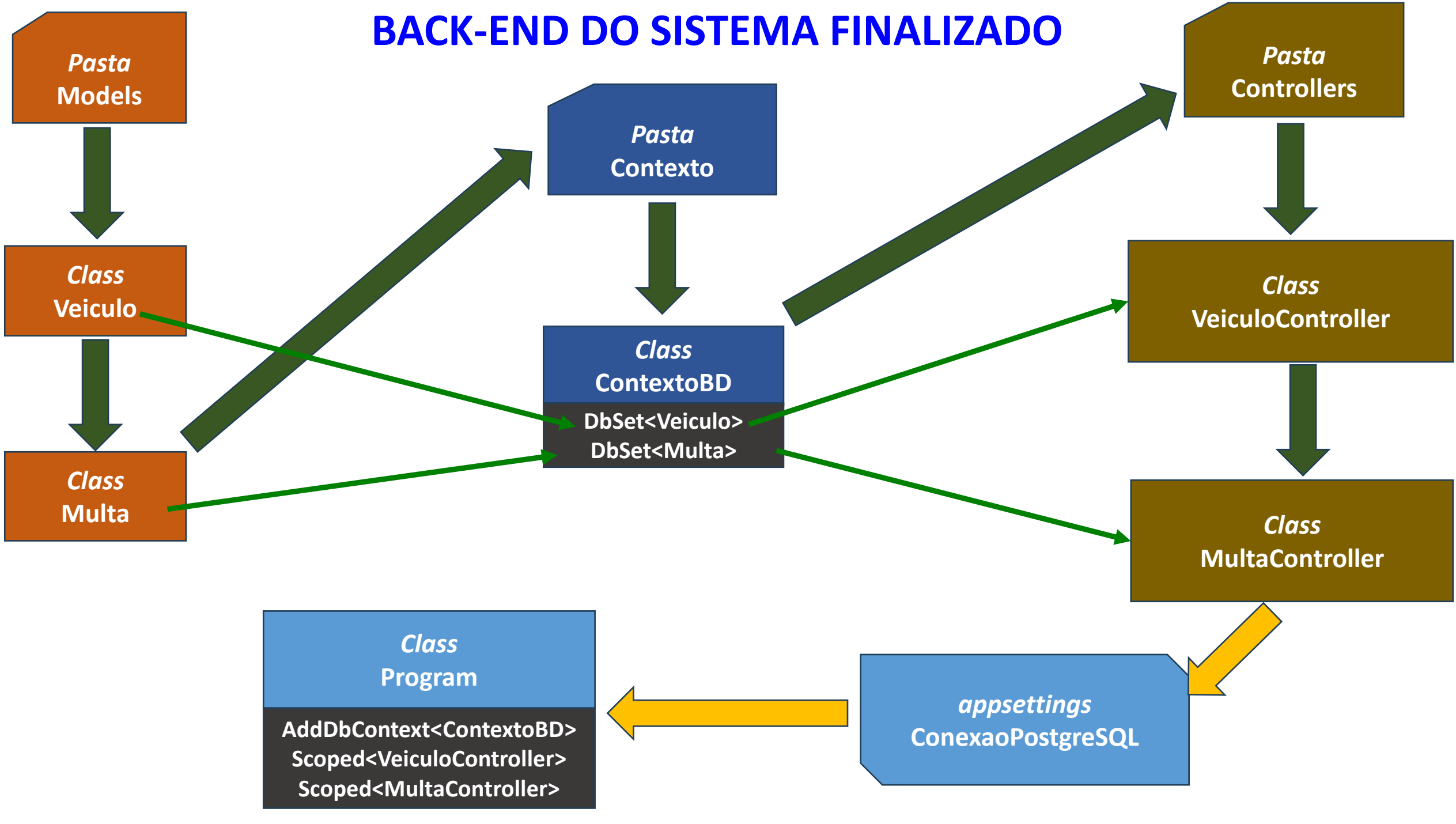


# Configuração da classe Program

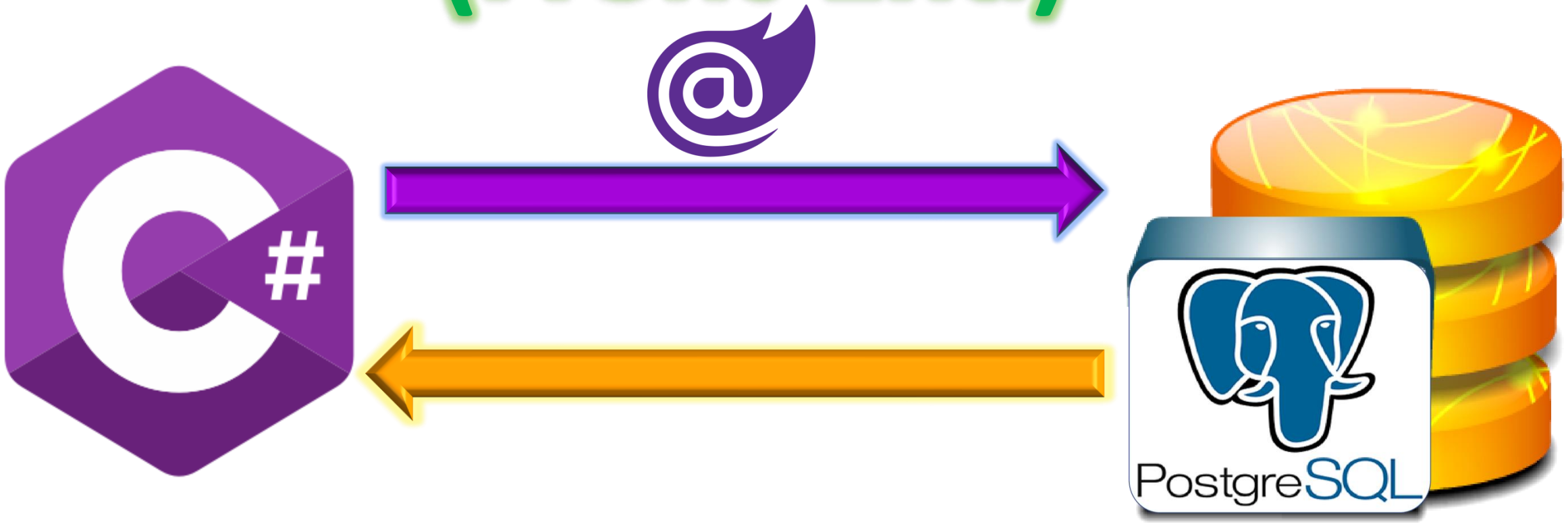
```
using AppMulta.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

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

builder.Services.AddScoped<VeiculoController>();
builder.Services.AddScoped<MultaController>();

//conexão com banco de dados
builder.Services.AddEntityFrameworkNpgsql().AddDbContext<ContextoBD>(options =>
options.UseNpgsql(builder.Configuration.GetConnectionString("ConexaoPostgreSQL")));


var app = builder.Build();
```

Implementações

# BACK-END DO SISTEMA FINALIZADO



# 2ª Fase da implementação
# Acesso ao Banco de Dados
# (Front-End)



# LEGENDA DO FLUXOGRAMA

O próximo slide tem o objetivo de demostrar 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**)

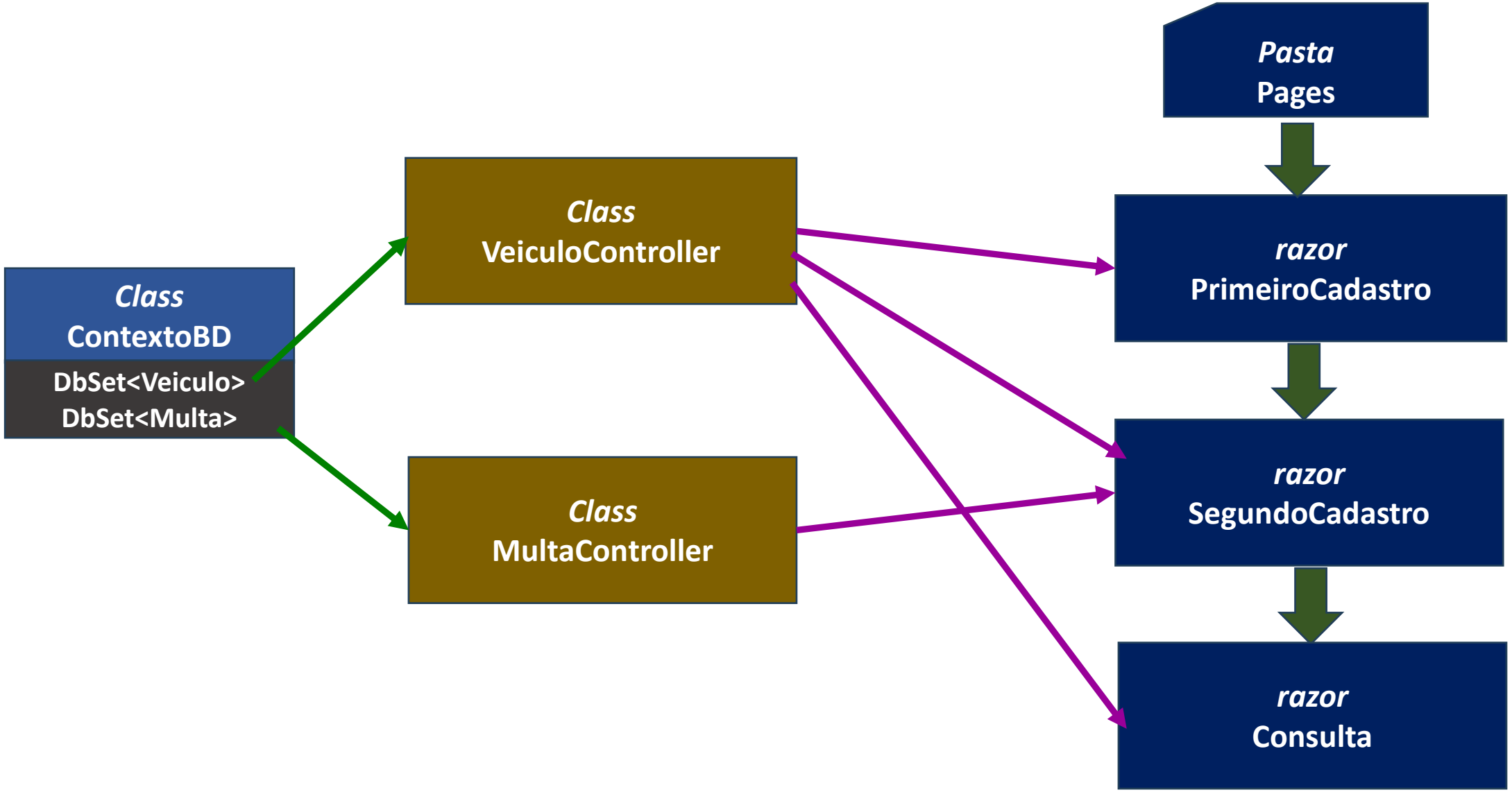
Indica a conexão que existe entre as classes

Indica qual o próximo item existente que será editado.

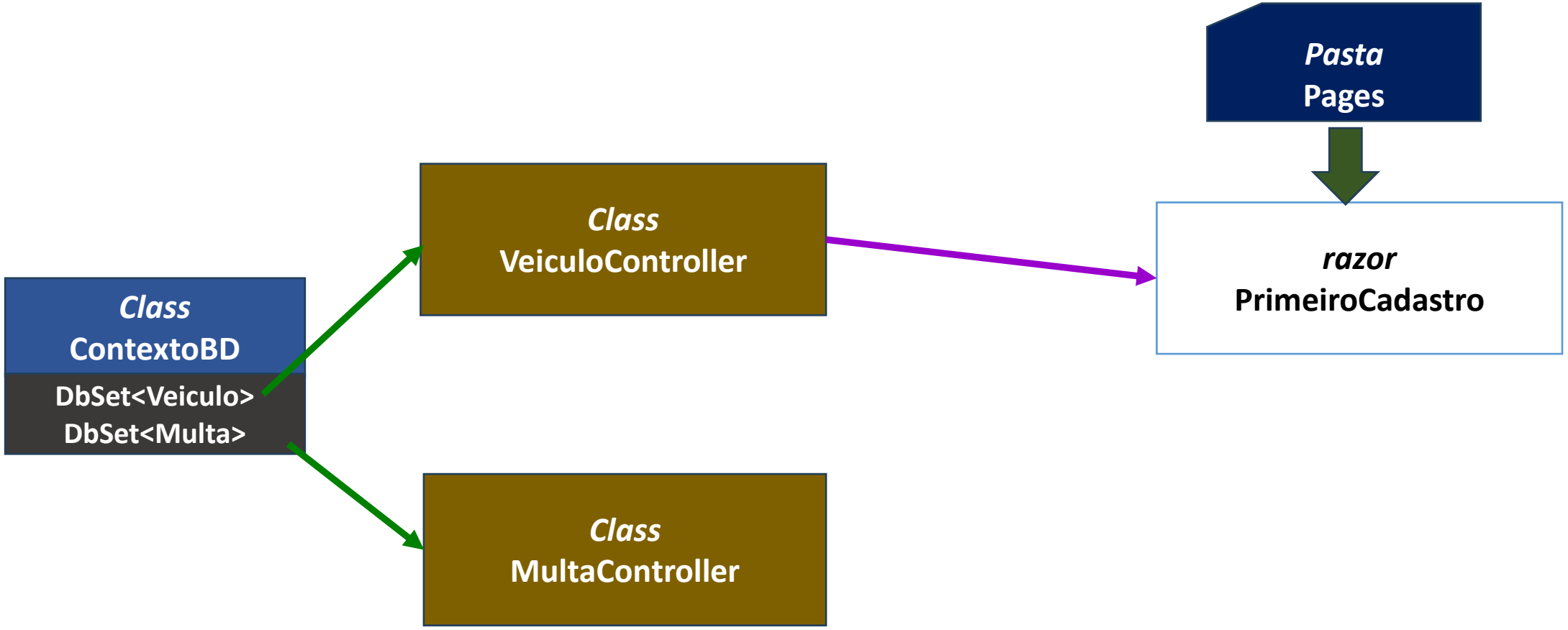
**Obs1: A classe tem a mesma cor da pasta a qual ela pertence.**

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

# VISÃO GERAL DO SISTEMA (FRONT-END)



# Dentro da pasta Pages, Implemente “PrimeiroCadastro.razor”



# Dentro da pasta Pages, Implemente “PrimeiroCadastro.razor”

```
@page "/primeiroCadastro"
@using AppMulta.Models;
@using AppMulta.Controllers;
@inject VeiculoController veiculoController
@inject NavigationManager navegacao


<h3>PRIMEIRO REGISTRO DE MULTA DO VEÍCULO</h3>
```

Acessa o banco de dados

# Continuação da implementação do “PrimeiroCadastro.razor”

```
@code {
    public List<Multa> listaMultas = new List<Multa>(); //Lista de multas
    public Veiculo? veiculo = new Veiculo();
    private bool enabledButton = true;
    private bool campoBloqueado = false;
    public string? Descricao { get; set; }
    public decimal? ValorMulta { get; set; }


    public void AddMulta()
    {
        Multa multa = new Multa();
        multa.Descricao = Descricao;
        multa.ValorMulta = ValorMulta;
        listaMultas.Add(multa);
        Descricao = null;
        ValorMulta = null;
    }
```

# Continuação da implementação do “PrimeiroCadastro.razor”

```
    private async Task Salvar()
    {
        veiculo.Multas = listaMultas;
        await veiculoController.Add(veiculo);
        await veiculoController.Salvar();
        enabledButton = false;
        campoBloqueado = true;
    }

    public void Cancelar()
    {
        navegacao.NavigateTo("/primeiroCadastro", forceLoad: true);
    }

    public void NovoRegistro()
    {
        navegacao.NavigateTo("/primeiroCadastro", forceLoad: true);
    }
}
```

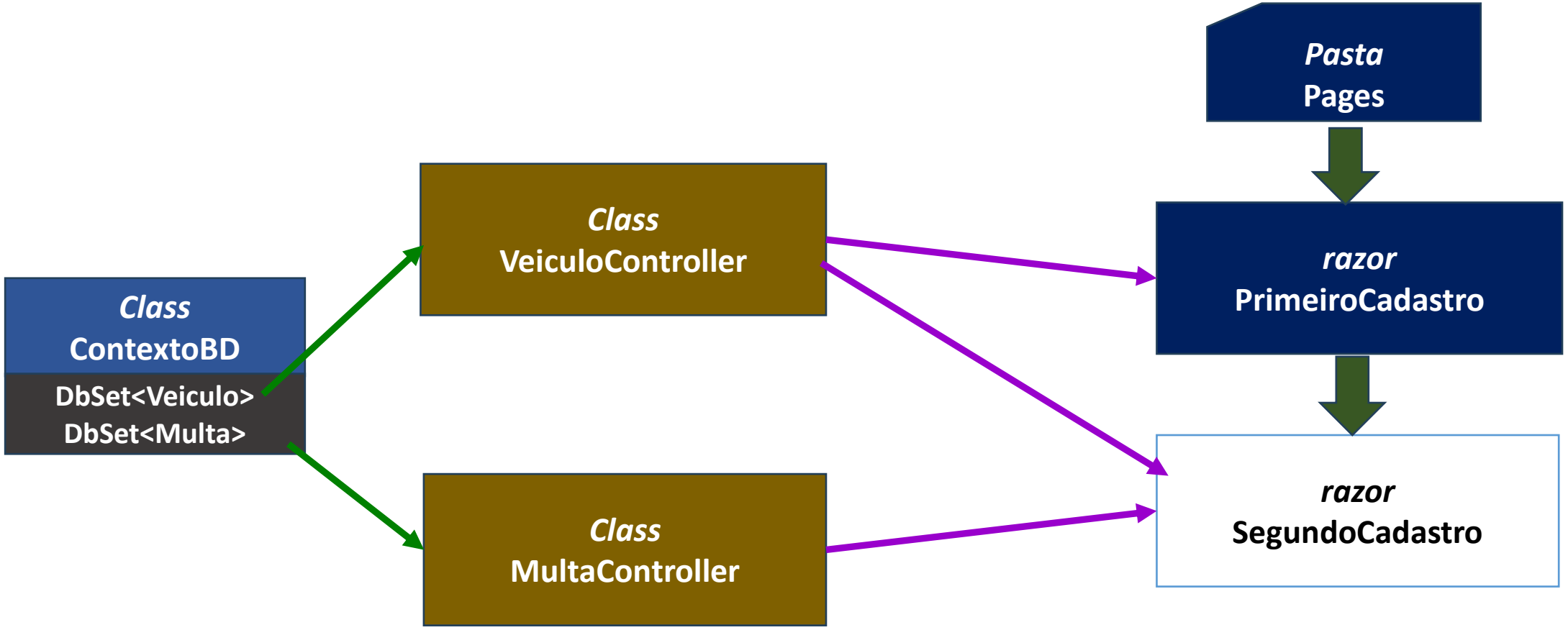
# Use o código Html disponibilizado para implementar o “PrimeiroCadastro”,

```
<h3>PRIMEIRO REGISTRO DE MULTA DO VEÍCULO</h3>

<div class="container">


    <div class="mb-3">
        <label for="exampleFormControlInput1" class="form-label">MODELO:</label>
        <input @bind="veiculo.Modelo" type="text" class="form-control" disabled="@campoBloqueado">
    </div>

    <div class="mb-3">
        <label for="exampleFormControlInput1" class="form-label">MARCA:</label>
        <input @bind="veiculo.Marca" type="text" class="form-control" disabled="@campoBloqueado">
    </div>

    <div class="mb-3">
        <label for="exampleFormControlInput1" class="form-label">PLACA:</label>
        <input @bind="veiculo.Placa" type="text" class="form-control" disabled="@campoBloqueado">
    </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>
}
```

```
<nav class="navbar navbar-light" style="background-color

<table class="table">
    <thead>
        <tr>
            <th>Descrição:</th>
            <th>Valor:</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var multa in listaMultas)
        {
            <tr>
                <td>@multa.Descricao</td>
                <td>@multa.ValorMulta</td>
            </tr>
        }
    </tbody>
</table>
</div>
```

# Dentro da pasta Pages, Implemente “SegundoCadastro.razor”



# Dentro da pasta Pages, Implemente “SegundoCadastro.razor”

```
@page "/segundoCadastro"
@using AppMulta.Models;
@using AppMulta.Controllers;
@inject VeiculoController veiculoController
@inject MultaController multaController
@inject NavigationManager navegacao


<h3>REGISTRO DE MULTAS DE VEÍCULOS COM CADASTRO</h3>
```

Acessa o banco de dados

# Continuação da implementação do “SegundoCadastro.razor”

```
<h3>REGISTRO DE MULTAS DE VEÍCULOS COM CADASTRO</h3>


@code {
    public List<Multa> multas = new List<Multa>(); //Lista de multas
    public List<Veiculo> veiculos = new List<Veiculo>(); //Lista de veículos
    public Veiculo? veiculo = new Veiculo();
    private bool enabledButton = true;
    private bool campoBloqueado = false;
    public string? Descricao { get; set; }
    public decimal? ValorMulta { get; set; }

    protected override async Task OnInitializedAsync()
    {
        var listaVeiculos = await veiculoController.ListaDeVeiculos();
        veiculos = listaVeiculos;
    }
}
```

# Continuação da implementação do “SegundoCadastro.razor”

```
        listaVeiculos = lista;
    }

    public void SelecionarVeiculo(ChangeEventArgs e)
    {
        int id = Convert.ToInt32(e.Value);
        veiculo = listaVeiculos.FirstOrDefault(v => v.Id == id);
    }

    public void AddMulta()
    {
        Multa multa = new Multa();
        multa.Descricao = Descricao;
        multa.ValorMulta = ValorMulta;
        multa.Veiculo = veiculo;
        listaMultas.Add(multa);
        Descricao = null;ValorMulta = null;
    }
```

Vínculo da Multa com a pessoa (FK)

# Continuação da implementação do “SegundoCadastro.razor”

```
    private async Task Salvar()
    {
        await multaController.AddRanger(listaMultas);
        await multaController.Salvar();
        enabledButton = false; campoBloqueado = true;
    }

    public void Cancelar()
    {
        navegacao.NavigateTo("/segundoCadastro", forceLoad: true);
    }

    public void NovoRegistro()
    {
        navegacao.NavigateTo("/primeiroCadastro", forceLoad: true);
    }
}
```

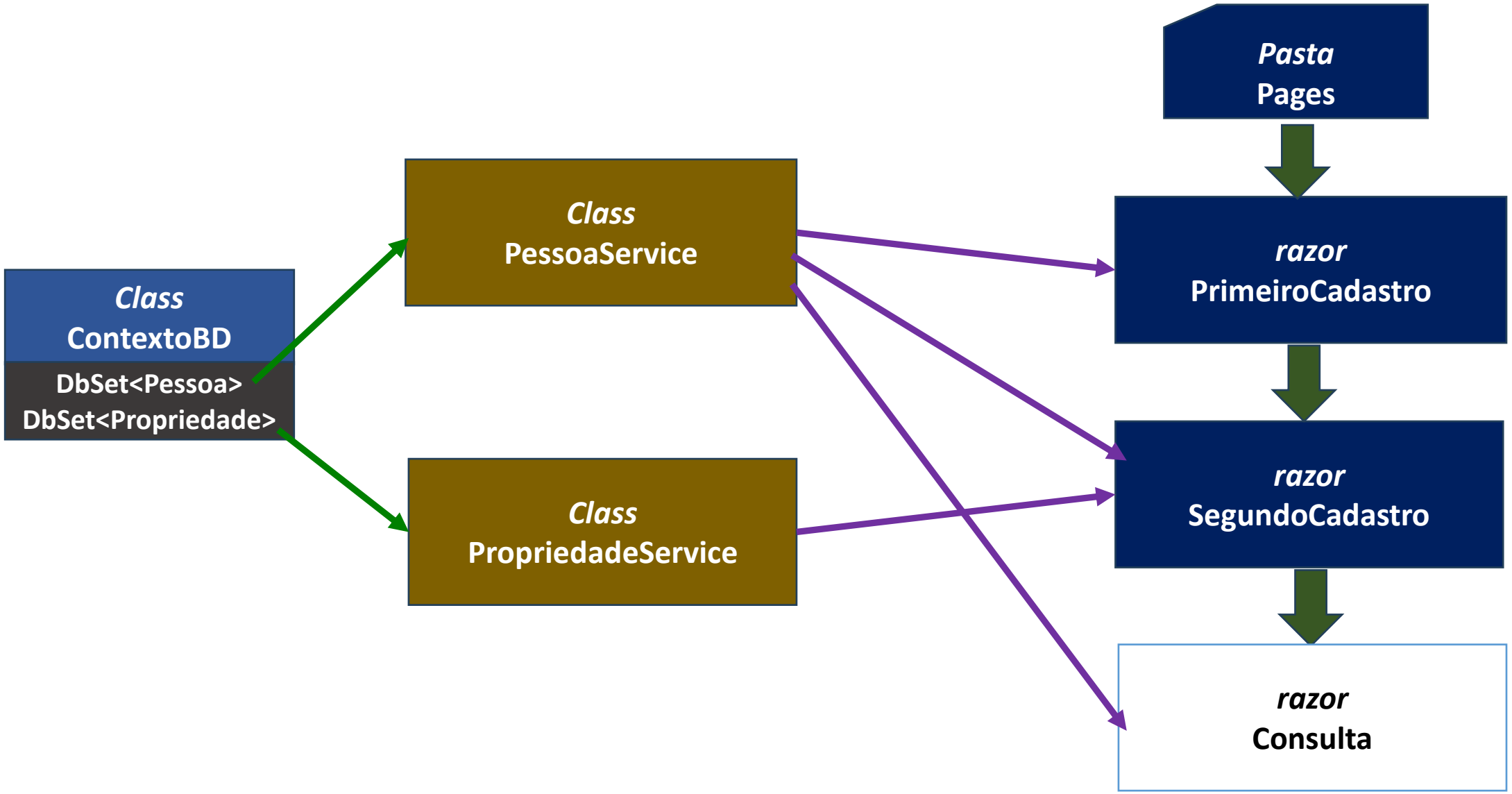
# Use o código Html disponibilizado para implementar o “SegundoCadastro”,

```
<h3>REGISTRO DE MULTAS DE VEÍCULOS COM CADASTRO</h3>

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

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

    <label for="veiculo" class="form-label">CARROS CADASTRADOS:</label>
    <select  @onchange="SelecionarVeiculo" class="form-select" aria-label="Selecione um carro">
        <option selected>Selecione uma placa</option>
        @foreach (var item in listaVeiculos)
        {
                <option value=@item.Id>@item.Placa</option>
        }
    </select>

    <nav class="navbar navbar-light" style="background-color: white"></nav>

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

```
<div class="mb-3">
    <label for="exampleFormControlInput1" class="form-label">VALOR :</label>
    <input @bind="ValorMulta" type="number" class="form-control" id="exampleFormControlInput1">
</div>
  @if(enabledButton){
    <button type="button" class="btn btn-primary" @onclick="AddMulta">ADICIONAR</button>
    <button type="button" class="btn btn-warning" @onclick="Cancelar">CANCELAR</button>
    <button type="button" class="btn btn-success" @onclick="Salvar">SALVAR</button>
}else{
    <div class="alert alert-success" role="alert">
       Salvo com Sucesso;
    </div>
    <button type="button" class="btn btn-primary" @onclick="NovoRegistro">NOVO REGISTRO</button>
}

<nav class="navbar navbar-light" style="background-color: white"></nav>
```

```
    <table class="table">
        <thead>
            <tr>
                <th>Descrição:</th>
                <th>Valor:</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var multa in listaMultas)
            {
                <tr>
                    <td>@multa.Descricao</td>
                    <td>@multa.ValorMulta</td>
                </tr>
            }
        </tbody>
    </table>
</div>
```

# Dentro da pasta Pages, Implemente "Consulta.razor"



# Dentro da pasta Pages, Implemente “Consulta.razor”

```
@page "/consulta"
@using AppMulta.Models;
@using AppMulta.Controllers;
@inject VeiculoController veiculoController;
@inject NavigationManager navegacao;



<h3>CONSULTA DE MULTAS</h3>
```

Acessa o banco de dados

# Continuação da implementação da “Consulta.razor”

```
@code {
    public List<Multa> listaMultas = new List<Multa>();
    public List<Veiculo> listaVeiculos = new List<Veiculo>();
    public Veiculo? veiculo { get; set; } //veículos


    protected override async Task OnInitializedAsync()
    {

        var lista = await veiculoController.ListaDeVeiculos();
        listaVeiculos = lista;
    }


    public void SelecionarVeiculo(ChangeEventArgs e)
    {
        int id = Convert.ToInt32(e.Value);
        veiculo = listaVeiculos.FirstOrDefault(v => v.Id == id);
        listaMultas = veiculo.Multas;
    }
}
```

# Continuação da implementação da “Consulta.razor”

```
public void SelecionarVeiculo(ChangeEventArgs e)
{
    int id = Convert.ToInt32(e.Value);
    veiculo = listaVeiculos.FirstOrDefault(v => v.Id == id);
    listaMultas = veiculo.Multas;
}
}
```

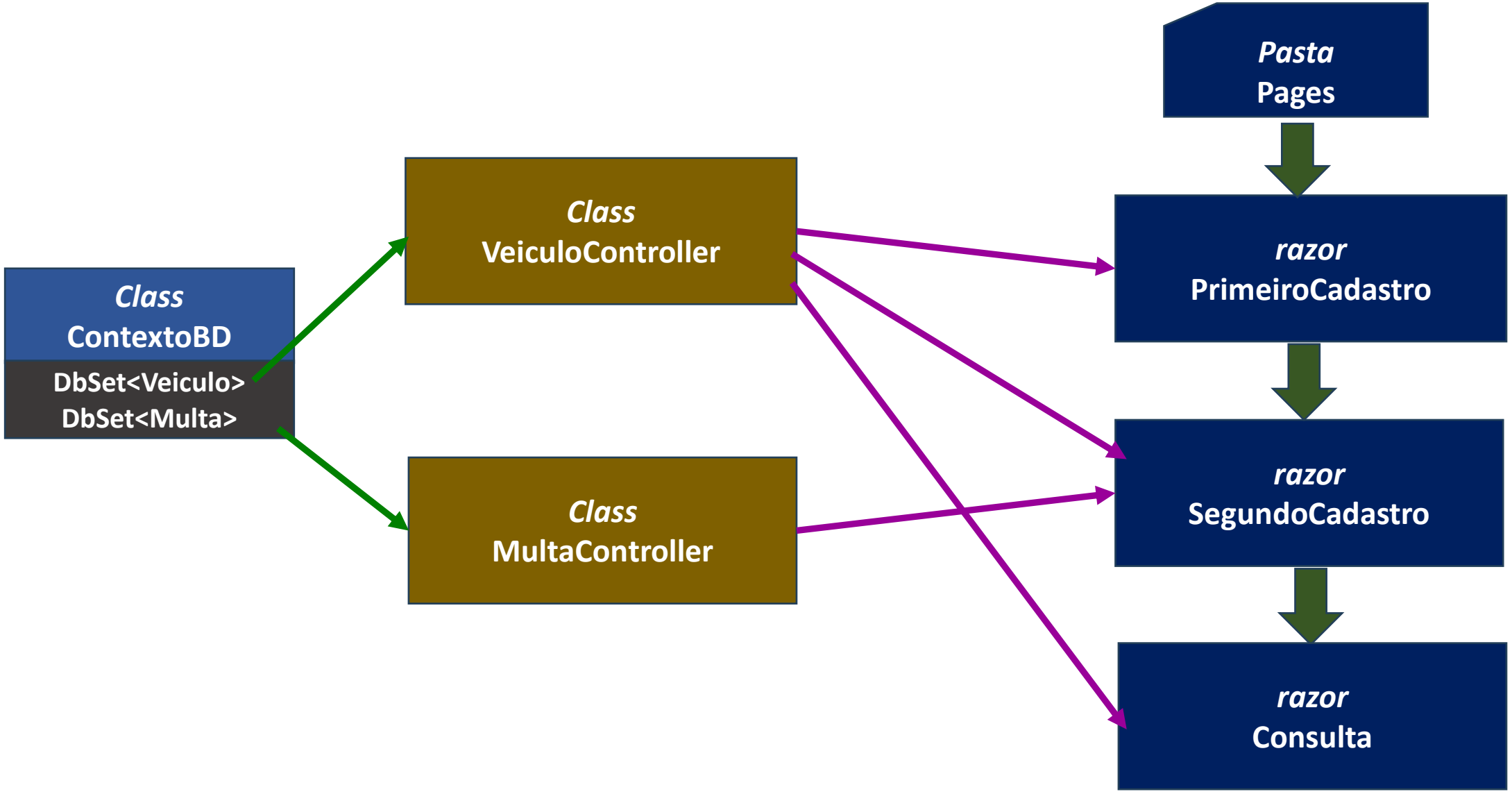
Observe que não foi preciso usar o MultaController uma vez que o comando Include no VeiculoController incluiu as multas conforme imagem abaixo:

```
public async Task<List<Veiculo>> ListaDeVeiculos()
{
    var veiculos = await _context.Veiculos.Include(x => x.Multas).ToListAsync();
    return veiculos;
}
```

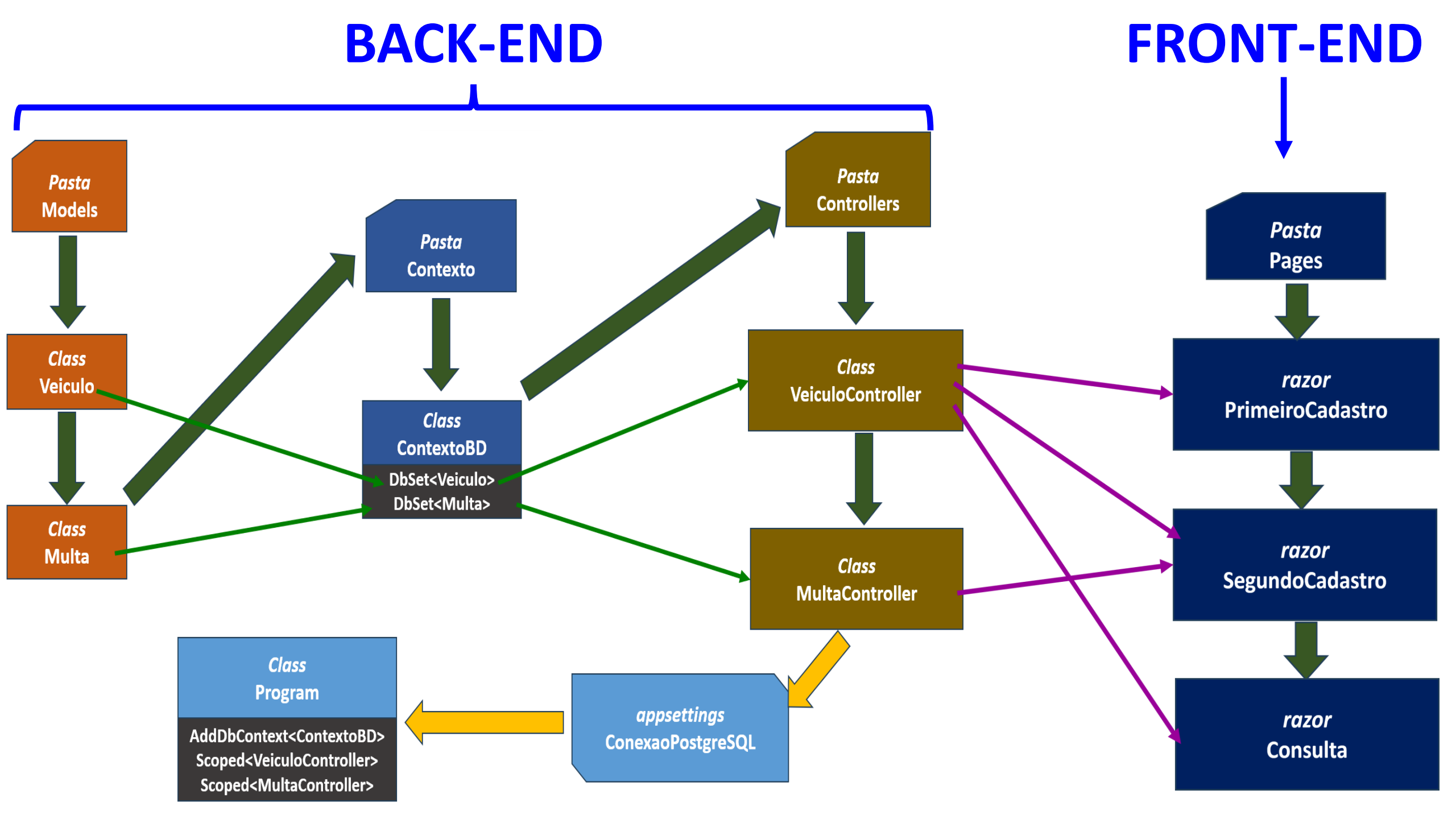
# Use o código Html disponibilizado para implementar a “Consulta”,

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

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

    <label for="veiculo" class="form-label">CARROS CADASTRADOS:</label>
    <select @onchange="SelecionarVeiculo" class="form-select" aria-label="Selecione um carro">
        <option selected>Selecione uma placa</option>
        @foreach (var item in listaVeiculos)
        {
            <option value=@item.Id>@item.Placa</option>
        }
    </select>

    <nav class="navbar navbar-light" style="background-color: white"></nav>

    <nav class="navbar navbar-light" style="background-color: darkgreen"></nav>
    <div class="row align-items-start">
        <div class="col">
            <div class="mb-3">
                <label for="exampleFormControlInput1" class="form-label">MODELO:</label>
                <input @bind="veiculo.Modelo" type="text" class="form-control" id="exampleFormControlInput1">
```

# CONCLUSÃO DO (FRONT-END)



# Visão geral do Sistema (back-end) => (front-end)


BACK-END
FRONT-END
Pasta
Models
Pasta
Contexto
Pasta
Controllers
Pasta
Pages
Class
Veiculo
Class
ContextoBD
DbSet<Veiculo>
DbSet<Multa>
Class
VeiculoController
razor
PrimeiroCadastro
Class
Multa
Class
MultaController
razor
SegundoCadastro
Class
Program
AddDbContext<ContextoBD>
Scoped<VeiculoController>
Scoped<MultaController>
appsettings
ConexaoPostgreSQL
razor
Consulta

# ATIVIDADE

1 – Mostrar a soma total de todas as multas registrados de um veículo selecionado na página da consulta.

2 - Criar uma página de consulta onde permita que o usuário possa listar todas as multas de um pessoa selecionada, mas que também permita:

2.1 – Filtrar as multas acima de uma valor informado pelo usuário

2.2 – Filtrar as multas por uma descrição informada pelo usuário

2.3 – Mostrar o total das multas

2.4 – Mostrar o total das multas que foram filtradas. (não utilizar o mesmo filtro do item anterior)

3 – Criar uma página para listar os veículos e a descrição da multa de maior valor de cada veículo listado.