**Criação da App Completo**

## **1 - Setup da aplicação completa e Criação das Entidades**

1 - Criar uma “Blank Solution”

2 - Criar as pastas (sql, src, tests) na pasta da Solution

3 - Criar o arquivo “global.json” (Define qual versão do seu SDK vi ser utilizado)

Verificando a versão no CMD com o comando dotnet - - info

{

"projects": [ "", "" ],

"sdk": {

"version": "2.2.107"

}

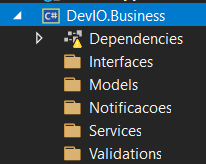
}

4 – (CAMADA DE APRESENTAÇÂO “DevIO.App”) Add um novo projeto na Solution (ASP.NET Core Web Aplicattion (MVC)) dentro da pasta src 

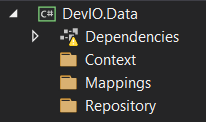
5 – (CAMADA DE NEGOCIO(Business) “DevIO.Business”) Add um novo projeto na Solution (Class Library .Net Core) dentro da pasta src 

6 – (CAMADA DE DADOS(Data) “DevIO.Data”)Add um novo projeto na Solution (Class Library .Net Core) dentro da pasta src 

7 – Criar as pastas (Interdaces, Models, Notificacoes, Services, Validations) dentro da camada (Business)



8– Criar as pastas (Context, Mappings, Repository) dentro da camada (Data)



9 - Criação das classes (Entity,Endereco,Fornecedor,Produto,TipoFornecedor) dentro (Business\Models)

using System;

namespace DevIO.App.Models

{

public abstract class Entity

{

protected Entity()

{

Id = Guid.NewGuid();

}

public Guid Id { get; set; }

}

}

using System;

namespace DevIO.App.Models

{

public class Endereco : Entity

{

public Guid FornecedorId { get; set; }

public string Logradouro { get; set; }

public string Numero { get; set; }

public string Complemento { get; set; }

public string Cep { get; set; }

public string Bairro { get; set; }

public string Cidade { get; set; }

public string Estado { get; set; }

/\* EF Relation \*/

public Fornecedor Fornecedor { get; set; }

}

}

using System.Collections.Generic;

namespace DevIO.App.Models

{

public class Fornecedor : Entity

{

public string Nome { get; set; }

public string Documento { get; set; }

public TipoFornecedor TipoFornecedor { get; set; }

public bool Ativo { get; set; }

/\* EF Relations \*/

public Endereco Endereco { get; set; }

public IEnumerable<Produto> Produtos { get; set; }

}

}

using System;

namespace DevIO.App.Models

{

public class Produto : Entity

{

public Guid FornecedorId { get; set; }

public string Nome { get; set; }

public string Descricao { get; set; }

public string Imagem { get; set; }

public decimal Valor { get; set; }

public DateTime DataCadastro { get; set; }

public bool Ativo { get; set; }

/\* EF Relations \*/

public Fornecedor Fornecedor { get; set; }

}

}

namespace DevIO.App.Models

{

public enum TipoFornecedor

{

PessoaFisica = 1,

PessoaJuridica

}

}

## **2 - Mapeando as entidades para o banco de dados**

1 – Criar a classe “MeuDbContext” na pasta Context na camada DavIO.Date

2 – Executar o comando Install-Package Microsoft.EntityFrameworkCore na camada “DavIO.Date”

3 - Após a instalação é possível relaciona a classe MeuDebContext “: DbContext”

4 – Criar um construtor (Public MeuDbContext(DbContextOptions options) : base(options))

using System;

using System.Collections.Generic;

using System.Text;

using DevIO.App.Models;

using Microsoft.EntityFrameworkCore;

namespace DevIO.App.Data

{

public class MeuDbContext : DbContext

{

public MeuDbContext(DbContextOptions<MeuDbContext> options)

: base(options)

{

}

public DbSet<Produto> Produtos { get; set; }

public DbSet<Fornecedor> Fornecedores { get; set; }

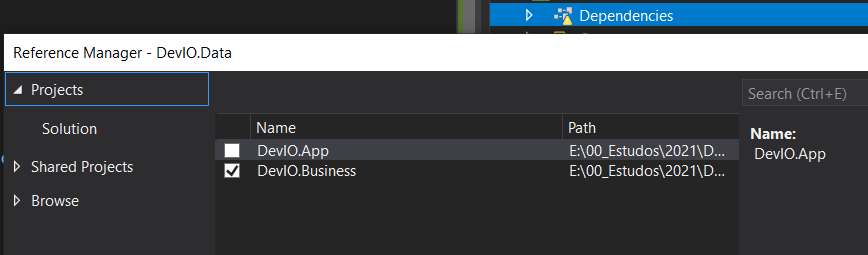
public DbSet<Endereco> Enderecos { get; set; }

}

}

5 – Mapear os DbSet para (produtos, Enderecos, Fornecedores)

Add as referências em “Dependencies” e add a camada de “DevIO.Business”



6 – Criar a classe “ProdutoMapping” e herda de : IEntityTypeConfiguration<Produto>

7 – Dentro da classe é necessário implementar a interfece “public void Configure(EntityTypeBuilder<Produto> builder)” e dentro é feito o mapeamento dos campos e seus relacionamentos

**Obs.: é importante instalar o Microsoft.EntityFrameworkCore.Relational**

using DevIO.App.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace DevIO.Data.Mappings

{

public class ProdutoMapping : IEntityTypeConfiguration<Produto>

{

public void Configure(EntityTypeBuilder<Produto> builder)

{

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

builder.Property(p => p.Nome)

.IsRequired()

.HasColumnType("varchar(200)");

builder.Property(p => p.Descricao)

.IsRequired()

.HasColumnType("varchar(1000)");

builder.Property(p => p.Imagem)

.IsRequired()

.HasColumnType("varchar(100)");

builder.ToTable("Produtos");

}

}

}

8 – As classes “EnderecoMapping” e “FornecedorMapping” devem ser criadas seguindo os passos das etapas 6,7

using DevIO.App.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace DevIO.Data.Mappings

{

public class FornecedorMapping : IEntityTypeConfiguration<Fornecedor>

{

public void Configure(EntityTypeBuilder<Fornecedor> builder)

{

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

builder.Property(p => p.Nome)

.IsRequired()

.HasColumnType("varchar(200)");

builder.Property(p => p.Documento)

.IsRequired()

.HasColumnType("varchar(14)");

// 1 : 1 => Fornecedor : Endereco

builder.HasOne(f => f.Endereco)

.WithOne(e => e.Fornecedor);

// 1 : N => Fornecedor : Produtos

builder.HasMany(f => f.Produtos)

.WithOne(p => p.Fornecedor)

.HasForeignKey(p => p.FornecedorId);

builder.ToTable("Fornecedores");

}

}

}

using DevIO.App.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace DevIO.Data.Mappings

{

public class ProdutoMapping : IEntityTypeConfiguration<Produto>

{

public void Configure(EntityTypeBuilder<Produto> builder)

{

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

builder.Property(p => p.Nome)

.IsRequired()

.HasColumnType("varchar(200)");

builder.Property(p => p.Descricao)

.IsRequired()

.HasColumnType("varchar(1000)");

builder.Property(p => p.Imagem)

.IsRequired()

.HasColumnType("varchar(100)");

builder.ToTable("Produtos");

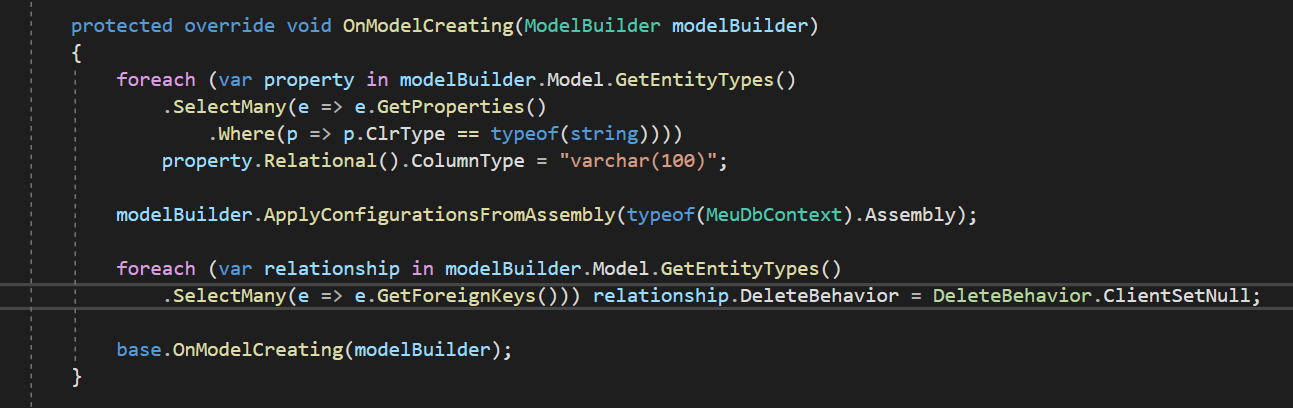
}

}

}

**Obs.: Que configura as dependências é a classe mãe na hora do Mapping**

9 – Mapeamento no arquivo de contexto “MeuDbContext” realizar os registros dos arquivos de Mappings (ProdutoMapping, EnderecoMapping e FornecedorMapping)



modelBuilder.ApplyConfigurationsFromAssembly(typeof(MeuDbContext).Assembly);

10 - Desabilitação do cascade delete (Isso impe que ao excluir, exclua seus filhos juntos)

foreach (var relationship in modelBuilder.Model.GetEntityTypes()

.SelectMany(e => e.GetForeignKeys())) relationship.DeleteBehavior = DeleteBehavior.ClientSetNull;

11 - Garantir que as colunas do tipo string esquecidas de ser mapeadas fique varchar(100) e não varchar(MAX)

foreach (var property in modelBuilder.Model.GetEntityTypes()

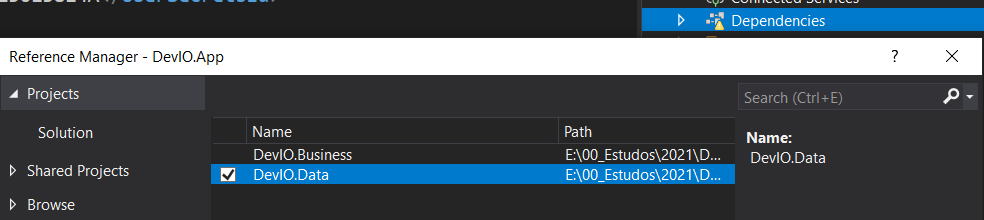
.SelectMany(e => e.GetProperties()

.Where(p => p.ClrType == typeof(string))))

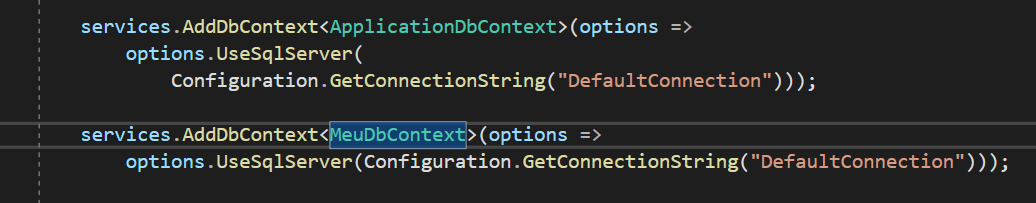
property.Relational().ColumnType="varchar(100)";

12 - Configuração do DbContext na aplicação,

Ir nas dependencies no DevIO.App e add a reference DavIO.Data



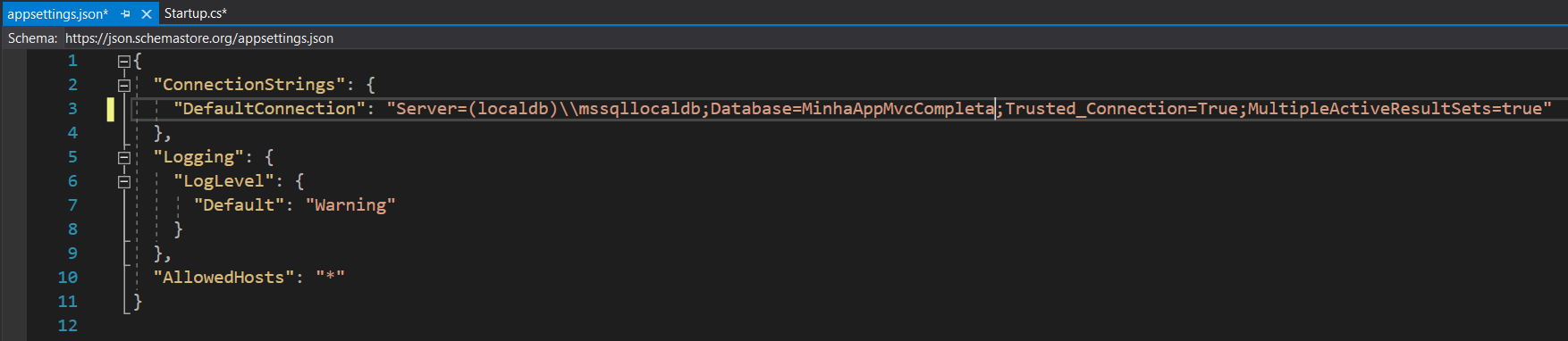
13 – Depois ir no Startup e add um novo **services.AddDbContext**



services.AddDbContext<MeuDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

14 – Alterar o nome do Database para MinhaAppMvcCompleta no arquivo appsettings.json da camada **DevIO.App**



15 – Criar uma Migration: (É necessário selecionar a camada (DATA) e só do ser executada em apenas um contexto)

PM> Add-Migration Initial –Verbose –Context MeuDbContext

16 – Instalar PM> Install-Package Microsoft.EntityFrameworkCore.SqlServer

17 - Criação dos scripts das tabelas

PM> Script-Migration -Context MeuDbContext (Bases da Aplicação)

PM> Script-Migration -Context ApplicationDbContext (Bases do Identity)

18 – Salvar o arquivo de Script de banco de dados na base sql da aplicação

19 - PM> Update-Database –Context MeuDbContext

PM> Update-Database –Context ApplicationDbContext

## **3 - Acessando o banco via repositórios**

1 – Criação da interface **IRepository** dentro DevIO.Business.Interfaces

public interface IRepository<TEntity> : IDisposable where TEntity : Entity

<TEntity> - Repositorio Genérico

IDisposable – Implementa para liberar a memoria

where TEntity : Entity – É especifico par classes filha de Entity

Sempre é bom trabalhar com métodos assíncrono (Task) para garantir a melhor performance e saúde do servido.

* 1. Criar o método Adicionar
  2. Criar o método ObterPorId
  3. Criar o método ObterTodos
  4. Criar o método Atualizar
  5. Criar o método REMOVER
  6. Criar o método Buscar
  7. Criar o método SaveChanges

using System;

using System.Collections.Generic;

using System.Linq.Expressions;

using System.Threading.Tasks;

using DevIO.App.Models;

namespace DevIO.Business.Intefaces

{

public interface IRepository<TEntity> : IDisposable where TEntity : Entity

{

Task Adicionar(TEntity entity);

Task<TEntity> ObterPorId(Guid id);

Task<List<TEntity>> ObterTodos();

Task Atualizar(TEntity entity);

Task Remover(Guid id);

Task<IEnumerable<TEntity>> Buscar(Expression<Func<TEntity, bool>> predicate);

Task<int> SaveChanges();

}

}

O repositório genérico serve para qualquer entidade, ele esta na camada de negocio pq a camada de negócio não vai conhecer a camada de acesso a dados “MeuDbContext”, quem conhece a camada de negocio e a camada de acesso a dados, a única forma da camada de negócio conversa é através dessa interface “**IRepository**” que será injetada por dependência fazendo referência a uma classe de repositório que está na camada de dados

2 - Criação da interface responsável pela classe de fornecedor

using System;

using System.Threading.Tasks;

using DevIO.App.Models;

namespace DevIO.Business.Intefaces

{

public interface IFornecedorRepository : IRepository<Fornecedor>

{

Task<Fornecedor> ObterFornecedorEndereco(Guid id);

Task<Fornecedor> ObterFornecedorProdutosEndereco(Guid id);

}

}

3 - Criação da interface responsável pela classe de Produtos

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using DevIO.App.Models;

namespace DevIO.Business.Intefaces

{

public interface IProdutoRepository : IRepository<Produto>

{

Task<IEnumerable<Produto>> ObterProdutosPorFornecedor(Guid fornecedorId);

Task<IEnumerable<Produto>> ObterProdutosFornecedores();

Task<Produto> ObterProdutoFornecedor(Guid id);

}

}

IEnumerable -> significa que o método traz uma lista

4 - Criação da interface responsável pela classe de Endereço

using System;

using System.Threading.Tasks;

using DevIO.App.Models;

namespace DevIO.Business.Intefaces

{

public interface IEnderecoRepository : IRepository<Endereco>

{

Task<Endereco> ObterEnderecoPorFornecedor(Guid fornecedorId);

}

}

5 – Implementar as Interfaces como classes completas, a primeira a ser implementada é no DevIO.Data.Repository é a Repository

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using System.Threading.Tasks;

using DevIO.App.Data;

using DevIO.App.Models;

using DevIO.Business.Intefaces;

using Microsoft.EntityFrameworkCore;

namespace DevIO.Data.Repository

{

public abstract class Repository<TEntity> : IRepository<TEntity> where TEntity : Entity, new()

{

protected readonly MeuDbContext Db;

protected readonly DbSet<TEntity> DbSet;

protected Repository(DbSet<TEntity> dbSet)

{

DbSet = dbSet;

}

protected Repository(MeuDbContext db)

{

Db = db;

DbSet = db.Set<TEntity>();

}

public async Task<IEnumerable<TEntity>> Buscar(Expression<Func<TEntity, bool>> predicate)

{

return await DbSet.AsNoTracking().Where(predicate).ToListAsync();

}

public virtual async Task<TEntity> ObterPorId(Guid id)

{

return await DbSet.FindAsync(id);

}

public virtual async Task<List<TEntity>> ObterTodos()

{

return await DbSet.ToListAsync();

}

public virtual async Task Adicionar(TEntity entity)

{

DbSet.Add(entity);

await SaveChanges();

}

public virtual async Task Atualizar(TEntity entity)

{

DbSet.Update(entity);

await SaveChanges();

}

public virtual async Task Remover(Guid id)

{

DbSet.Remove(new TEntity { Id = id });

await SaveChanges();

}

public async Task<int> SaveChanges()

{

return await Db.SaveChangesAsync();

}

public void Dispose()

{

Db?.Dispose();

}

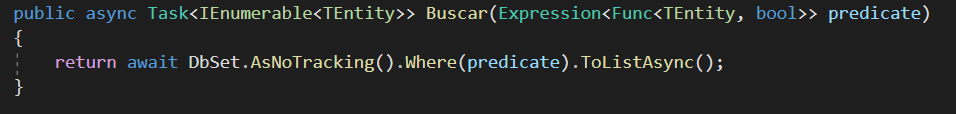
}

}

Ela é uma classe abstracta só pode ser herdada não pode ser instanciada.

E é necessário implementa a interface IRepository de <TEntity> mas essa é uma filha de **Entity**

É obrigatório implementar todos os métodos desses contratos.

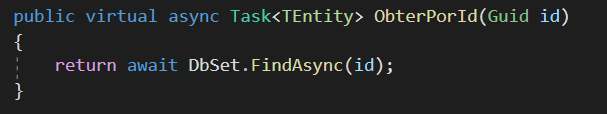


Vamos ter que devolver uma consulta utilizando o predicado.

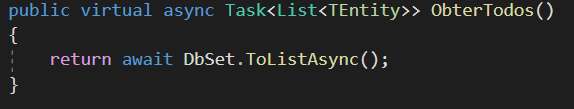
Sempre será um **await** por ser métodos assíncronos

É importante usar o **AsNotracking** pois ele fica acompanhado as mudanças de estados, retorna a resposta com mais performance.

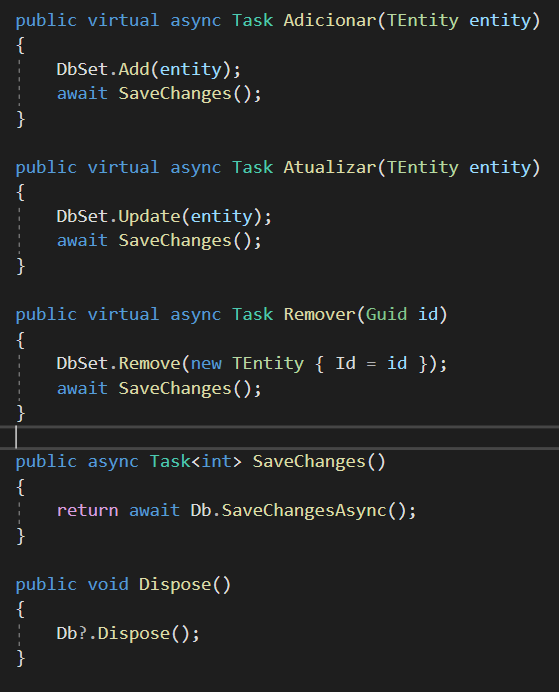
Obs: Vá até o banco de dados busque na entidade onde a expressão que vc passar retorne uma lista de forma assíncrona.



Obter por Id.



Obter todos.



**Virtual** é usada quando você quer em algum momento sobrescrever o método.

6 – Implementar as Interfaces **ProdutoRepository** na DevIO.Data.Repository

public class ProdutoRepository : Repository<Produto>, IProdutoRepository

É obrigatório implementar todos os métodos desses contratos (IProdutoRepository):

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using DevIO.App.Data;

using DevIO.App.Models;

using DevIO.Business.Intefaces;

using Microsoft.EntityFrameworkCore;

namespace DevIO.Data.Repository

{

public class ProdutoRepository : Repository<Produto>, IProdutoRepository

{

public ProdutoRepository(MeuDbContext context) : base(context) { }

public async Task<Produto> ObterProdutoFornecedor(Guid id)

{

return await Db.Produtos.AsNoTracking().Include(f => f.Fornecedor)

.FirstOrDefaultAsync(p => p.Id == id);

}

public async Task<IEnumerable<Produto>> ObterProdutosFornecedores()

{

return await Db.Produtos.AsNoTracking().Include(f => f.Fornecedor)

.OrderBy(p => p.Nome).ToListAsync();

}

public async Task<IEnumerable<Produto>> ObterProdutosPorFornecedor(Guid fornecedorId)

{

return await Buscar(p => p.FornecedorId == fornecedorId);

}

}

}

7 – Implementar as Interfaces **FornecedorRepository** na DevIO.Data.Repository

public class FornecedorRepository : Repository<Fornecedor>,

É obrigatório implementar todos os métodos desses contratos (IFornecedorRepository):

using System;

using System.Threading.Tasks;

using DevIO.App.Data;

using DevIO.App.Models;

using DevIO.Business.Intefaces;

using Microsoft.EntityFrameworkCore;

namespace DevIO.Data.Repository

{

public class FornecedorRepository : Repository<Fornecedor>, IFornecedorRepository

{

public FornecedorRepository(MeuDbContext context) : base(context)

{

}

public async Task<Fornecedor> ObterFornecedorEndereco(Guid id)

{

return await Db.Fornecedores.AsNoTracking()

.Include(c => c.Endereco)

.FirstOrDefaultAsync(c => c.Id == id);

}

public async Task<Fornecedor> ObterFornecedorProdutosEndereco(Guid id)

{

return await Db.Fornecedores.AsNoTracking()

.Include(c => c.Produtos)

.Include(c => c.Endereco)

.FirstOrDefaultAsync(c => c.Id == id);

}

}

}

8 – Implementar as Interfaces **EnderecoRepository** na DevIO.Data.Repository

public class EnderecoRepository : Repository<Endereco>, IEnderecoRepository

É obrigatório implementar todos os métodos desses contratos (IEnderecoRepository):

using System;

using System.Threading.Tasks;

using DevIO.App.Data;

using DevIO.App.Models;

using DevIO.Business.Intefaces;

using Microsoft.EntityFrameworkCore;

namespace DevIO.Data.Repository

{

public class EnderecoRepository : Repository<Endereco>, IEnderecoRepository

{

public EnderecoRepository(MeuDbContext context) : base(context) { }

public async Task<Endereco> ObterEnderecoPorFornecedor(Guid fornecedorId)

{

return await Db.Enderecos.AsNoTracking()

.FirstOrDefaultAsync(f => f.FornecedorId == fornecedorId);

}

}

}

9 – Realizar a resolução da injeção de dependência no **Startup** na parte do public void ConfigureServices(IServiceCollection services) inserindo os:

services.AddScoped<MeuDbContext>();

services.AddScoped<IProdutoRepository, ProdutoRepository>();

services.AddScoped<IFornecedorRepository, FornecedorRepository>();

services.AddScoped<IEnderecoRepository, EnderecoRepository>();

Agora é possível injetar onde quisermos tanto o **contexto** quanto o **repositório**.

## 4 - Mapeando as entidades em ViewModels com Automapper

1 – Antes de tudo renomear a pasta **Models** para **ViewModels.**

2 – Para mapear as **ViewModels** é necessário acrescentar na as classes **ProdutoViewModel, FornecedorViewModel** e **EnderecoViewModel.**

Depois devemos usar as devidas **ComponentModel.DataAnnotations** par cada situação especifica.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

namespace DevIO.App.ViewModels

{

public class FornecedorViewModel

{

[Key]

public Guid Id { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(100, ErrorMessage = "O campo {0} precisa ter entre {2} e {1} caracteres", MinimumLength = 2)]

public string Nome { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(14, ErrorMessage = "O campo {0} precisa ter entre {2} e {1} caracteres", MinimumLength = 11)]

public string Documento { get; set; }

[DisplayName("Tipo")]

public int TipoFornecedor { get; set; }

public EnderecoViewModel Endereco { get; set; }

[DisplayName("Ativo?")]

public bool Ativo { get; set; }

public IEnumerable<ProdutoViewModel> Produtos { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Http;

namespace DevIO.App.ViewModels

{

public class ProdutoViewModel

{

[Key]

public Guid Id { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[DisplayName("Fornecedor")]

public Guid FornecedorId { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(200, ErrorMessage = "O campo {0} precisa ter entre {2} e {1} caracteres", MinimumLength = 2)]

public string Nome { get; set; }

[DisplayName("Descrição")]

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(1000, ErrorMessage = "O campo {0} precisa ter entre {2} e {1} caracteres", MinimumLength = 2)]

public string Descricao { get; set; }

[DisplayName("Imagem do Produto")]

public IFormFile ImagemUpload { get; set; }

public string Imagem { get; set; }

[Moeda]

[Required(ErrorMessage = "O campo {0} é obrigatório")]

public decimal Valor { get; set; }

[ScaffoldColumn(false)]

public DateTime DataCadastro { get; set; }

[DisplayName("Ativo?")]

public bool Ativo { get; set; }

public FornecedorViewModel Fornecedor { get; set; }

public IEnumerable<FornecedorViewModel> Fornecedores { get; set; }

}

}

[ScaffoldColumn(false)] não quero que esse cara seja criado como campo.

O Campo Imagem será duplicado, ele tem propriedades tamanho, nome do arquivo etc.

using System;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Mvc;

namespace DevIO.App.ViewModels

{

public class EnderecoViewModel

{

[Key]

public Guid Id { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(200, ErrorMessage = "O campo {0} precisa ter entre {2} e {1} caracteres", MinimumLength = 2)]

public string Logradouro { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(50, ErrorMessage = "O campo {0} precisa ter entre {2} e {1} caracteres", MinimumLength = 1)]

public string Numero { get; set; }

public string Complemento { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(100, ErrorMessage = "O campo {0} precisa ter entre {2} e {1} caracteres", MinimumLength = 2)]

public string Bairro { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(8, ErrorMessage = "O campo {0} precisa ter {1} caracteres", MinimumLength = 8)]

public string Cep { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(100, ErrorMessage = "O campo {0} precisa ter entre {2} e {1} caracteres", MinimumLength = 2)]

public string Cidade { get; set; }

[Required(ErrorMessage = "O campo {0} é obrigatório")]

[StringLength(50, ErrorMessage = "O campo {0} precisa ter entre {2} e {1} caracteres", MinimumLength = 2)]

public string Estado { get; set; }

[HiddenInput]

public Guid FornecedorId { get; set; }

}

}

3 – Instalar o automapper via comando na solução **DevIO.App**

**PM>** Install-Package automapper

Ou

**PM>** Install-Package automapper.extensions.microsoft.dependencyinjection

4 - Realizar na **Startup** na parte do public void ConfigureServices(IServiceCollection services) a configuração do **automapper,** inserindo a linha de código:

services.AddAutoMapper(typeof(Startup));

5 – Para criar a configuração do **automapper** é necessário criaruma pasta denominada de **AutoMapper** dentro da solution **DevIO.App**

6 – Dentro da pasta **AutoMapper** criar a classeAutoMapperConfig

using AutoMapper;

using DevIO.App.Models;

using DevIO.App.ViewModels;

namespace DevIO.App.AutoMapper

{

public class AutoMapperConfig : Profile

{

public AutoMapperConfig()

{

CreateMap<Fornecedor, FornecedorViewModel>().ReverseMap();

CreateMap<Endereco, EnderecoViewModel>().ReverseMap();

CreateMap<Produto, ProdutoViewModel>().ReverseMap();

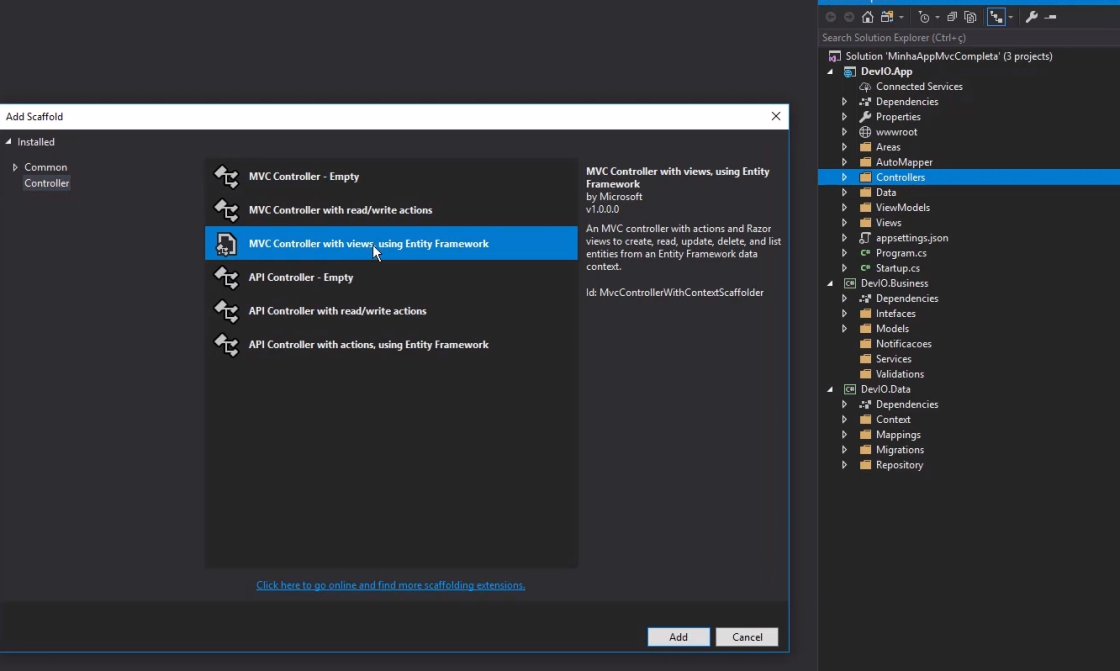
}

}

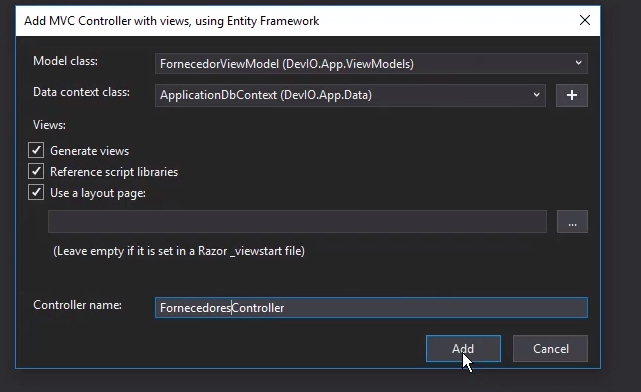
}

## 5 - Scaffolding das Controllers e Views

1 – Criar a controller de ForncedoresController



Nesse caso podemos escolher qualquer um **Data contexto class.**

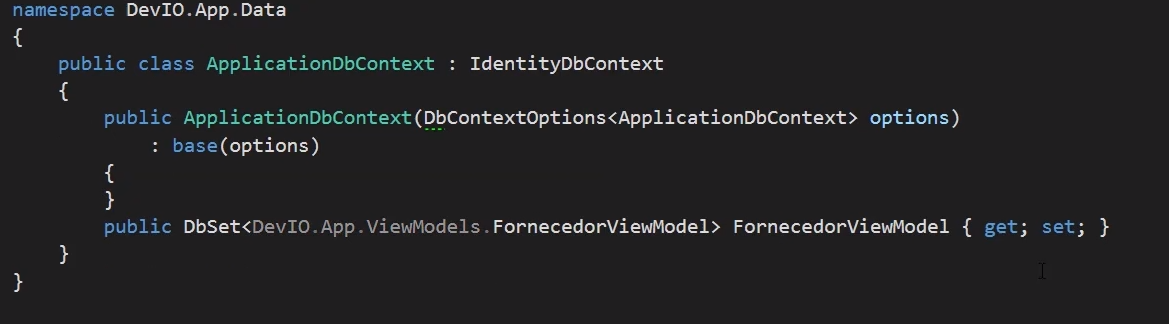


Obs.: Nesse caso irar da um erro pela questão do campo imagem duplicado esse momento iremos exclui-lo para gerar o **Scaffolding.**

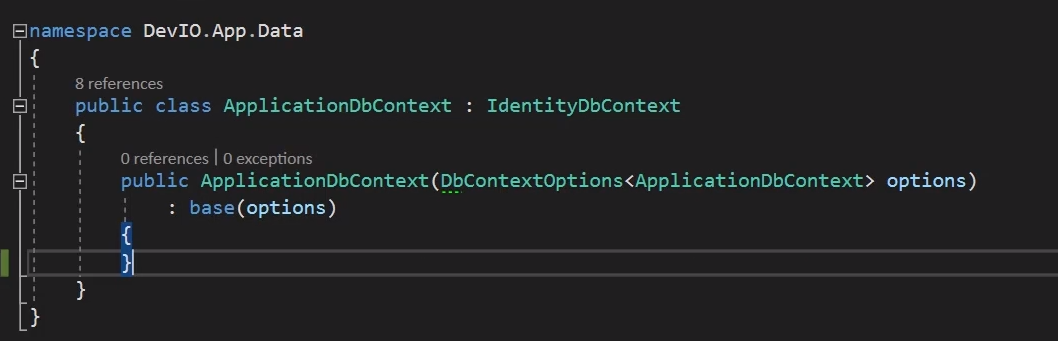


É importante retirar o DbSet do ApplicationDbContext.

Antes:



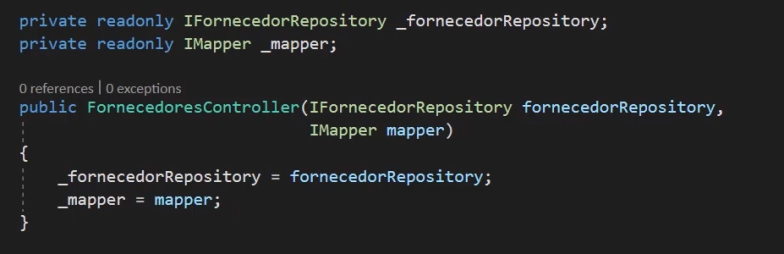
Depois:



Primeiro passo na ForncedoresController é substituir o contexto do Entity.



Por nosso IfornecedorRepository e injetar nosso construtor junto o Imapper:



Vamos construir 2 metodos privados para **ObterFornecedorEndereco** e **ObterFornecedorProdutosEndereco**.

Isso vai evitar o do **mapper** em toda a controle, deixando o código mais limpo.

private async Task<FornecedorViewModel> ObterFornecedorEndereco(Guid id)

{

return \_mapper.Map<FornecedorViewModel>(await \_fornecedorRepository.ObterFornecedorEndereco(id));

}

private async Task<FornecedorViewModel> ObterFornecedorProdutosEndereco(Guid id)

{

return \_mapper.Map<FornecedorViewModel>(await \_fornecedorRepository.ObterFornecedorProdutosEndereco(id));

}

**Index()**

public async Task<IActionResult> Index()

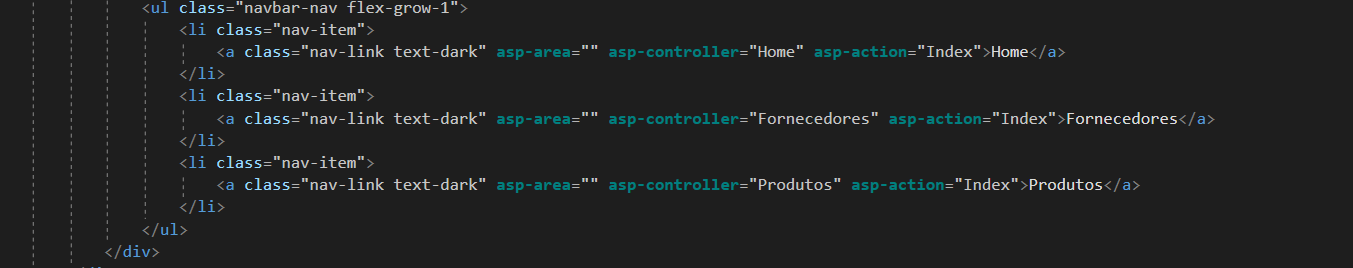
{

return View(\_mapper.Map<IEnumerable<FornecedorViewModel>>(await \_fornecedorRepository.ObterTodos()));

}



Configure os Link na **\_Layout.cshtml**



## 6 - Customização das Views

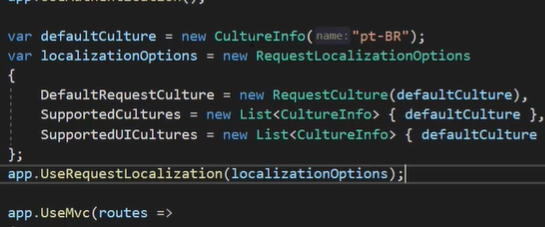
Customizado igual app fácil.

## 7 - Globalizando a aplicação em pt-BR

1 – É possível verificar a cultura inserindo a linha de código no começo de alguma view.



2 – Implementação na Startup



3- Alterar o arquivo **\_ValidationScriptsPartial.cshtml** adicionando o código abaixo.

<script>

$.validator.methods.range = function (value, element, param) {

var globalizedValue = value.replace(",", ".");

return this.optional(element) || (globalizedValue >= param[0] && globalizedValue <= param[1]);

};

$.validator.methods.number = function (value, element) {

return this.optional(element) || /-?(?:\d+|\d{1,3}(?:[\s\.,]\d{3})+)(?:[\.,]\d+)?$/.test(value);

};

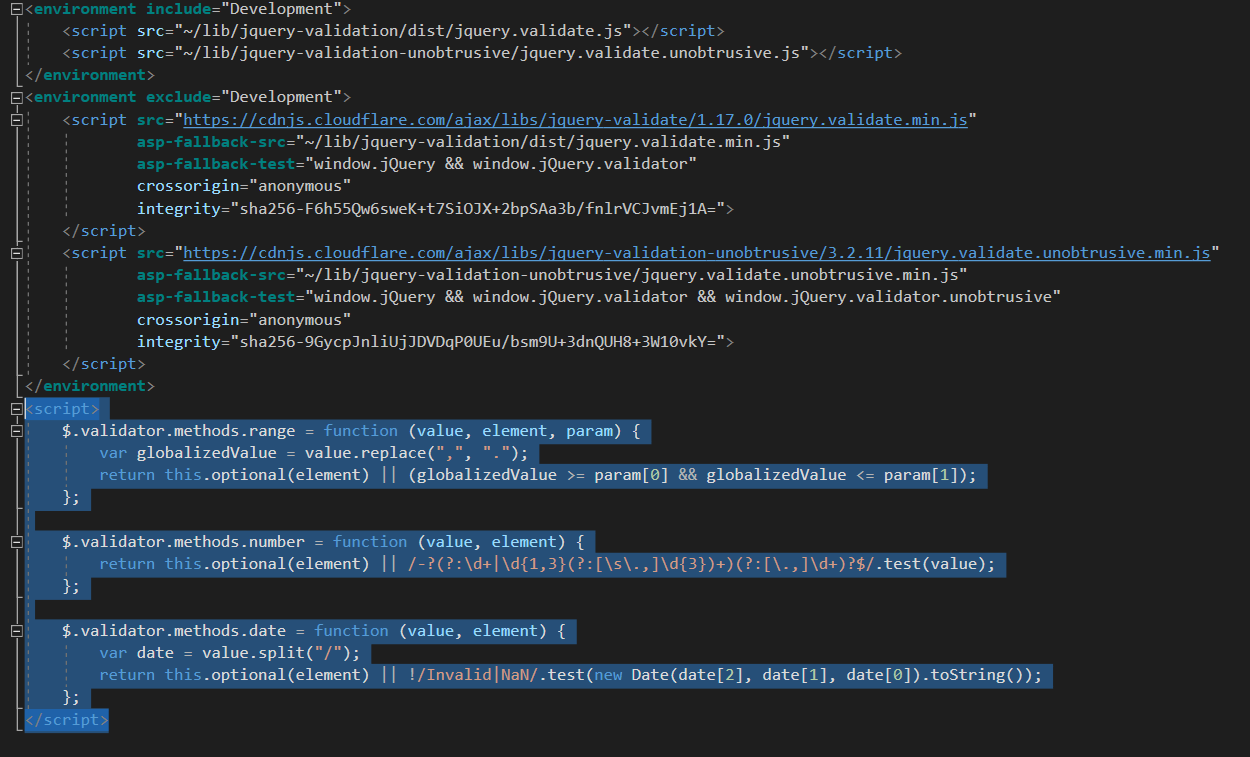
$.validator.methods.date = function (value, element) {

var date = value.split("/");

return this.optional(element) || !/Invalid|NaN/.test(new Date(date[2], date[1], date[0]).toString());

};

</script>



3 - Ajuste as mensagens de erros, alterar o services.AddMvc na stutup.

services.AddMvc(o =>

{

o.ModelBindingMessageProvider.SetAttemptedValueIsInvalidAccessor((x, y) => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetMissingBindRequiredValueAccessor(x => "Este campo precisa ser preenchido.");

o.ModelBindingMessageProvider.SetMissingKeyOrValueAccessor(() => "Este campo precisa ser preenchido.");

o.ModelBindingMessageProvider.SetMissingRequestBodyRequiredValueAccessor(() => "É necessário que o body na requisição não esteja vazio.");

o.ModelBindingMessageProvider.SetNonPropertyAttemptedValueIsInvalidAccessor(x => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetNonPropertyUnknownValueIsInvalidAccessor(() => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetNonPropertyValueMustBeANumberAccessor(() => "O campo deve ser numérico");

o.ModelBindingMessageProvider.SetUnknownValueIsInvalidAccessor(x => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetValueIsInvalidAccessor(x => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetValueMustBeANumberAccessor(x => "O campo deve ser numérico.");

o.ModelBindingMessageProvider.SetValueMustNotBeNullAccessor(x => "Este campo precisa ser preenchido.");

o.Filters.Add(new AutoValidateAntiforgeryTokenAttribute());

}).SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);

4 – Criar validação de moeda.

É necessário criar uma pasta Extensions e dentro criar a classe MoedaAttribute.cs 

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

try

{

var moeda = Convert.ToDecimal(value, new CultureInfo("pt-BR"));

}

catch (Exception)

{

return new ValidationResult("Moeda em formato inválido");

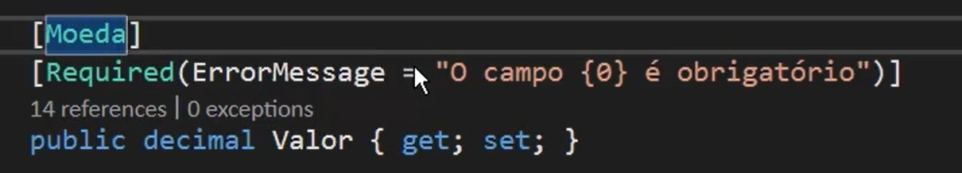
}

return ValidationResult.Success;

}

}

Agora temos um **DataAnnotations** customizado (Mas isso so resolve a validação do lado do server).



Para adaptar no cliente precisamos criar um adapter.

public class MoedaAttributeAdapter : AttributeAdapterBase<MoedaAttribute>

{

public MoedaAttributeAdapter(MoedaAttribute attribute, IStringLocalizer stringLocalizer) : base(attribute, stringLocalizer)

{

}

public override void AddValidation(ClientModelValidationContext context)

{

if (context == null) throw new ArgumentNullException(nameof(context));

MergeAttribute(context.Attributes, "data-val", "true");

MergeAttribute(context.Attributes, "data-val-moeda", GetErrorMessage(context));

MergeAttribute(context.Attributes, "data-val-number", GetErrorMessage(context));

}

public override string GetErrorMessage(ModelValidationContextBase validationContext)

{

return "Moeda em formato inválido";

}

}

Mas para usar MoedaAttributeAdapter precsamos criar um provide.

public class MoedaValidationAttributeAdapterProvider : IValidationAttributeAdapterProvider

{

private readonly IValidationAttributeAdapterProvider \_baseProvider = new ValidationAttributeAdapterProvider();

public IAttributeAdapter GetAttributeAdapter(ValidationAttribute attribute, IStringLocalizer stringLocalizer)

{

if (attribute is MoedaAttribute moedaAttribute)

{

return new MoedaAttributeAdapter(moedaAttribute, stringLocalizer);

}

return \_baseProvider.GetAttributeAdapter(attribute, stringLocalizer);

}

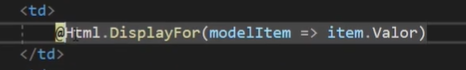
}

Agora é necessário injeta-lo na Startup.

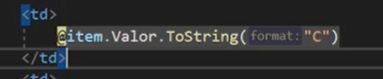


Para Fazer a decoração com R$, para ficar mais claro qual a moeda.

Nas views onde teremos os valores: **100,00**



Precisamos moda para: **R$ 100,00**





## 8 – Ajustando as View

Formatando campo de CPF ou CNPJ, na pasta Extensionscriar a classe RazorExtensions.

using System;

using Microsoft.AspNetCore.Mvc.Razor;

namespace DevIO.App.Extensions

{

public static class RazorExtensions

{

public static string FormataDocumento(this RazorPage page, int tipoPessoa, string documento)

{

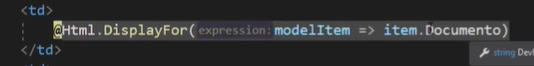
return tipoPessoa == 1 ? Convert.ToUInt64(documento).ToString(@"000\.000\.000\-00") : Convert.ToUInt64(documento).ToString(@"00\.000\.000\/0000\-00");

}

}

}

Nas view vamos alterar o código de:



Para:



## 9 – Utilizando Model Window de edição (Ajax)

## 10 – Utilizando Ajax para preenchimento do CEP

## 11 – Organizado e otimizando a estrutura da App

1 – Organizando a Startup o primeiro passo é criar a pasta **Configurations**

2 – Dentro da pasta **Configurations** será criado a classe DependencyInjectionConfig.cs para colocar todas as injeção de dependências.

using DevIO.App.Extensions;

using DevIO.Business.Intefaces;

using DevIO.Business.Notificacoes;

using DevIO.Business.Services;

using DevIO.Data.Context;

using DevIO.Data.Repository;

using Microsoft.AspNetCore.Mvc.DataAnnotations;

using Microsoft.Extensions.DependencyInjection;

namespace DevIO.App.Configurations

{

public static class DependencyInjectionConfig

{

public static IServiceCollection ResolveDependencies(this IServiceCollection services)

{

services.AddScoped<MeuDbContext>();

services.AddScoped<IProdutoRepository, ProdutoRepository>();

services.AddScoped<IFornecedorRepository, FornecedorRepository>();

services.AddScoped<IEnderecoRepository, EnderecoRepository>();

services.AddSingleton<IValidationAttributeAdapterProvider, MoedaValidationAttributeAdapterProvider>();

services.AddScoped<INotificador, Notificador>();

services.AddScoped<IFornecedorService, FornecedorService>();

services.AddScoped<IProdutoService, ProdutoService>();

return services;

}

}

}

E depois colocamos na Startup apenas o codigo:

services.ResolveDependencies();

3 – Fazer a mesma coisa para o MVC criando a classe MvcConfig.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.DependencyInjection;

namespace DevIO.App.Configurations

{

public static class MvcConfig

{

public static IServiceCollection AddMvcConfiguration(this IServiceCollection services)

{

services.AddMvc(o =>

{

o.ModelBindingMessageProvider.SetAttemptedValueIsInvalidAccessor((x, y) => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetMissingBindRequiredValueAccessor(x => "Este campo precisa ser preenchido.");

o.ModelBindingMessageProvider.SetMissingKeyOrValueAccessor(() => "Este campo precisa ser preenchido.");

o.ModelBindingMessageProvider.SetMissingRequestBodyRequiredValueAccessor(() => "É necessário que o body na requisição não esteja vazio.");

o.ModelBindingMessageProvider.SetNonPropertyAttemptedValueIsInvalidAccessor(x => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetNonPropertyUnknownValueIsInvalidAccessor(() => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetNonPropertyValueMustBeANumberAccessor(() => "O campo deve ser numérico");

o.ModelBindingMessageProvider.SetUnknownValueIsInvalidAccessor(x => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetValueIsInvalidAccessor(x => "O valor preenchido é inválido para este campo.");

o.ModelBindingMessageProvider.SetValueMustBeANumberAccessor(x => "O campo deve ser numérico.");

o.ModelBindingMessageProvider.SetValueMustNotBeNullAccessor(x => "Este campo precisa ser preenchido.");

o.Filters.Add(new AutoValidateAntiforgeryTokenAttribute());

}).SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);

return services;

}

}

}

E depois colocamos na Startup apenas o codigo:

services.AddMvcConfiguration();

4 – Fazer a mesma coisa para o Identity criando a classe IdentityConfig.cs

using DevIO.App.Data;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.UI;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

namespace DevIO.App.Configurations

{

public static class IdentityConfig

{

public static IServiceCollection AddIdentityConfiguration(this IServiceCollection services, IConfiguration configuration)

{

services.Configure<CookiePolicyOptions>(options =>

{

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));

services.AddDefaultIdentity<IdentityUser>()

.AddDefaultUI(UIFramework.Bootstrap4)

.AddEntityFrameworkStores<ApplicationDbContext>();

return services;

}

}

}

E depois colocamos na Startup apenas o código:

services.AddIdentityConfiguration(Configuration);

5 – Fazer a mesma coisa para o GlobalizationConfig criando a classe GlobalizationConfig.cs

using System.Collections.Generic;

using System.Globalization;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Localization;

namespace DevIO.App.Configurations

{

public static class GlobalizationConfig

{

public static IApplicationBuilder UseGlobalizationConfig(this IApplicationBuilder app)

{

var defaultCulture = new CultureInfo("pt-BR");

var localizationOptions = new RequestLocalizationOptions

{

DefaultRequestCulture = new RequestCulture(defaultCulture),

SupportedCultures = new List<CultureInfo> { defaultCulture },

SupportedUICultures = new List<CultureInfo> { defaultCulture }

};

app.UseRequestLocalization(localizationOptions);

return app;

}

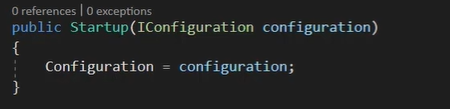
}

}

E depois colocamos na Startup apenas o código:

app.UseGlobalizationConfig();

5 – Alterar os construtor de Configuration, para uma mais complexo.



Para:

public Startup(IHostingEnvironment hostEnvironment)

{

var builder = new ConfigurationBuilder()

.SetBasePath(hostEnvironment.ContentRootPath)

.AddJsonFile("appsettings.json", true, true)

.AddJsonFile($"appsettings.{hostEnvironment.EnvironmentName}.json", true, true)

.AddEnvironmentVariables();

if (hostEnvironment.IsDevelopment())

{

builder.AddUserSecrets<Startup>();

}

Configuration = builder.Build();

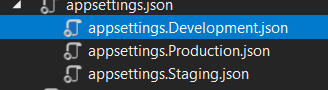
}

Isso permitirá a configuração para cada tipo de ambiente.

Para isso devemos duplicar o arquivo appsettings.Development.json e criar os arquivos:

appsettings.Production.json

appsettings.Staging.json

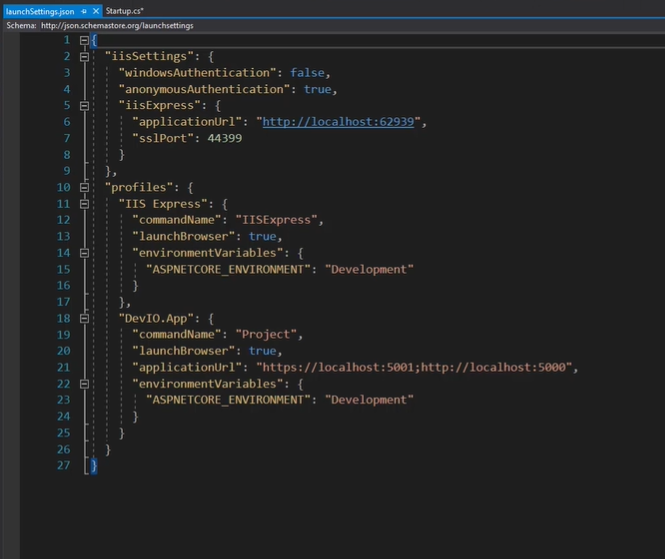




Para que isso funcione precisamos configurar no arquivo launchSettings.json



De:



Para:



5 – Cria na pasta Extensions a classe EmailTagHelper.cs

using System.Threading.Tasks;

using Microsoft.AspNetCore.Razor.TagHelpers;

namespace DevIO.App.Extensions

{

public class EmailTagHelper : TagHelper

{

public string EmailDomain { get; set; } = "desenvolvedor.io";

public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)

{

output.TagName = "a";

var content = await output.GetChildContentAsync();

var target = content.GetContent() + "@" + EmailDomain;

output.Attributes.SetAttribute("href", "mailto:" + target);

output.Content.SetContent(target);

}

}

}

Para funcionar é só colar esse código na view \_Layout.cshtml.

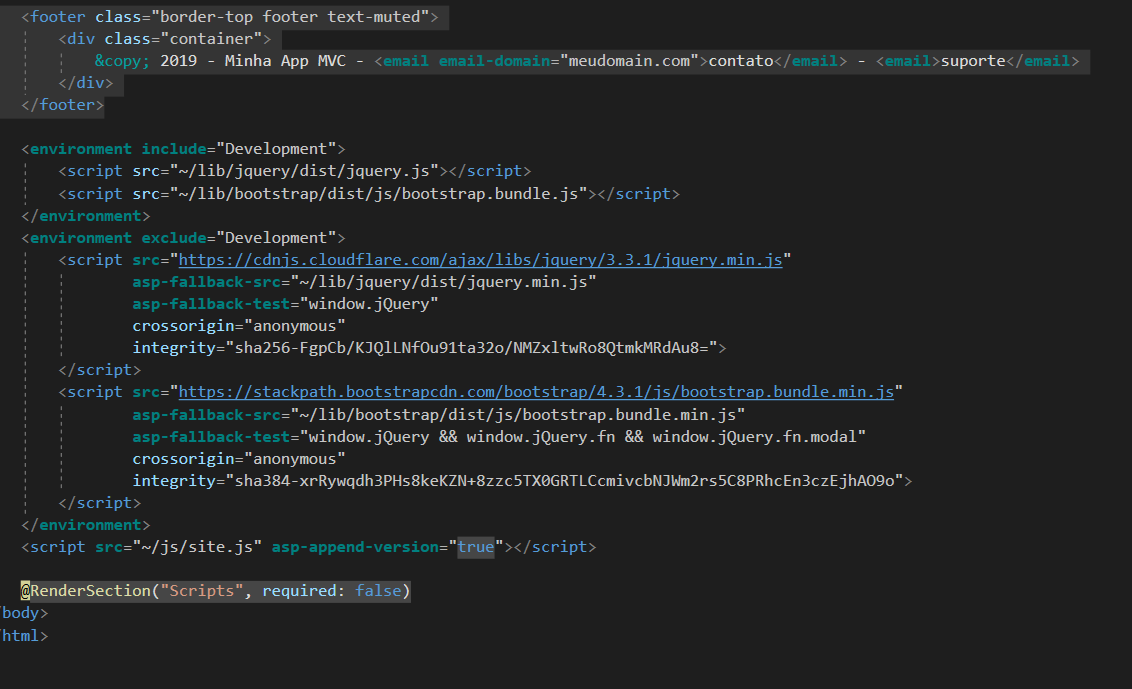
<footer class="border-top footer text-muted">

<div class="container">

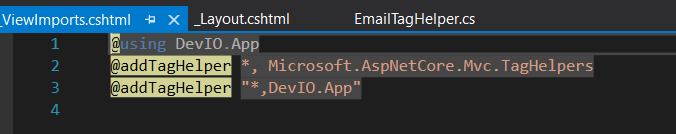
&copy; 2019 - Minha App MVC - <email email-domain="meudomain.com">contato</email> - <email>suporte</email>

</div>

</footer>



Mas e importante lembrar que temos que configurar na \_ViewImports.cshtml



## 12 – Roteamento inteligente

Rota para mostrar uma lista de fornecedores.



Rota para mostrar os dados de um fornecedor especifico.

É necessário passar o parâmetro, nesse caso é o id igual está no método, e nesse caso forçamos para que o parâmetro seja colocado com o mesmo formato do método (Guid).



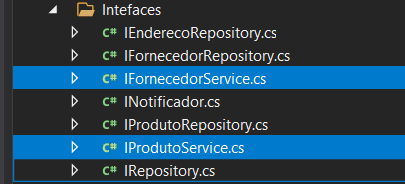
## 13 – Validando as entidades de negocio

É importante valida a informação na camada de negocio (DevIO.Business) e não só na camada de apresentação (DevIO.App) pelos DataAnnotations.

É importante validar no front e no back-end.

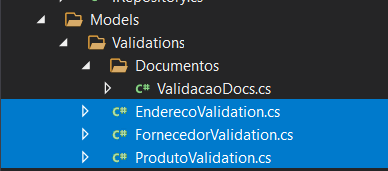
Os métodos de modificação não da informação não devem ser feitos direto no banco via repositório.

1 – Criar as interfaces com seus métodos de validação na camada de negócio (DevIO.Business).





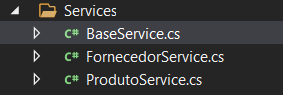
2 – Na camada de negócio (DevIO.Business) na pasta Validations dentro da Models, vamos cria as classes:





3 – Criar as classes abaixo na pasta services na camada de negócio (DevIO.Business).

Onde iremos colocar três classes de serviço:



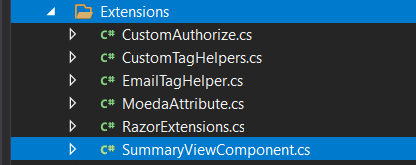
Essas classes vão usar a biblioteca do <https://fluentvalidation.net/>;

using DevIO.Business.Models.Validations;

Para instalação: Install-Package FluentValidation

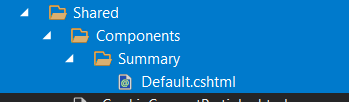


4 – Para mostrar os erro na tela é necessário criar uma intenção.





Adiciona as pastas e cria uma view Default.





Para mostrar as msn de validação na tela é necessário substituir nas views:

De:



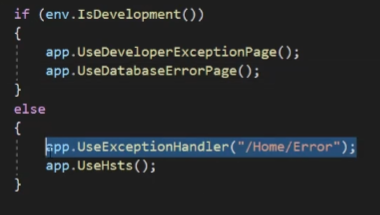
Para:



## 14 – Tratamento de erros

1 – Ajustar os ambientes com seus roteamentos na Startup.

De:



Para:

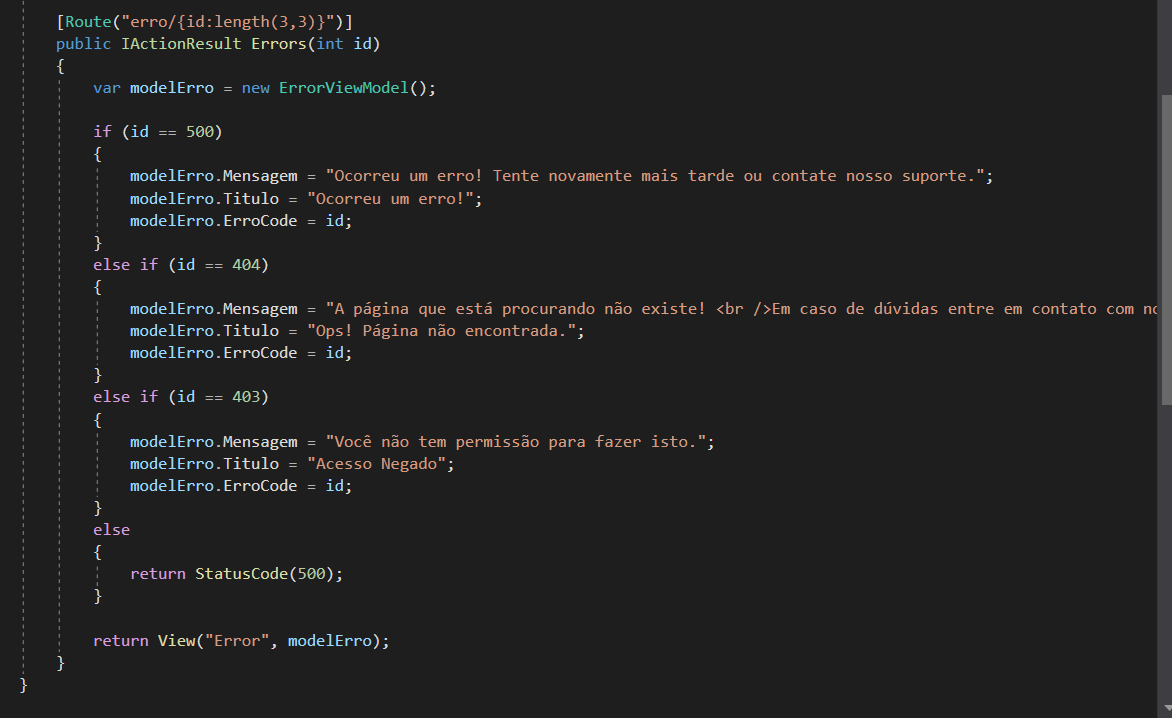


2 – Agora na HomeController precisamos fazer o seguinte ajuste:

De:



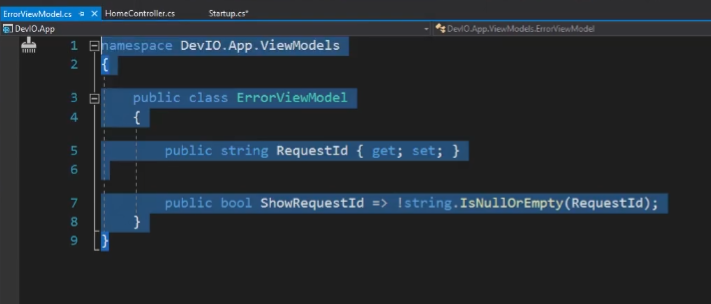
Para:



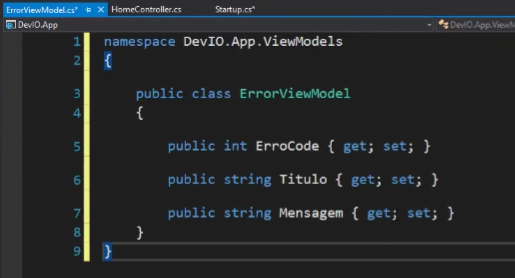


3 – Precisamos ajustar a classe ErrorViewModel.cs

De:



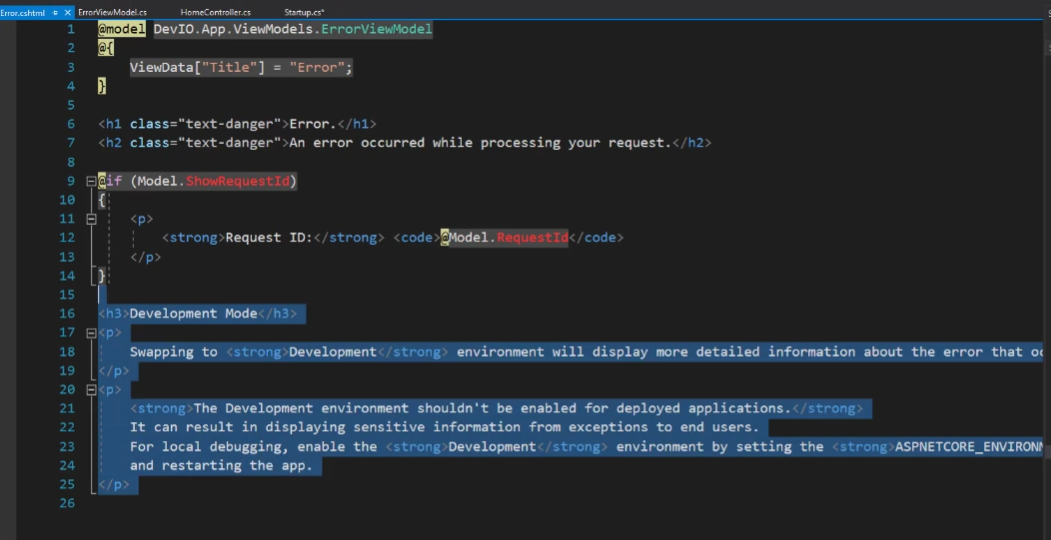
Para:





4 – Precisamos ajustar a classe ErrorViewModel.cs

De:



Para:

