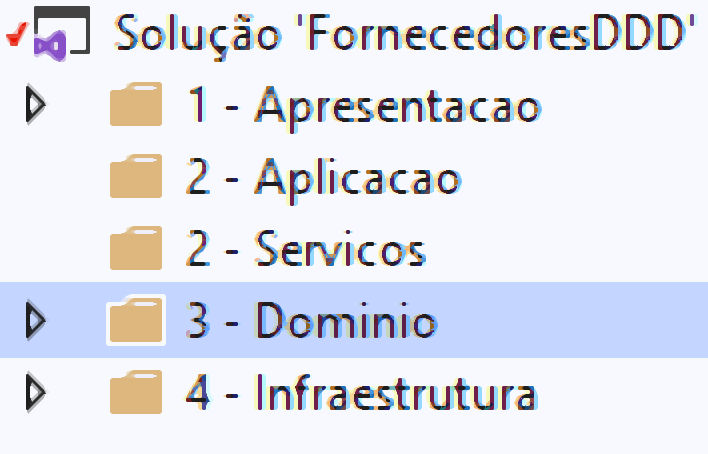
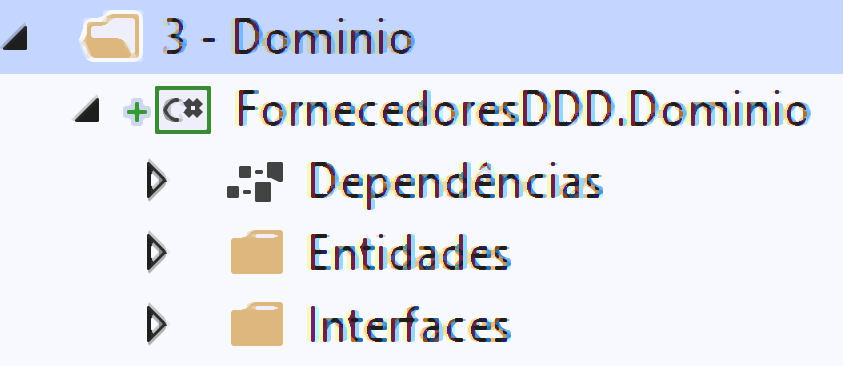
**🏯 Aplicação com arquitetura DDD**



### A camada de Domínio

Na pasta **3 - Dominio**, criar um projeto Biblioteca de Classes C# (.NET Core) e manter a estrutura:



Em Entidades, criar **Produto.cs**:

|  |
| --- |
| **namespace FornecedoresDDD.Dominio.Entidades {  public class Produto  {  public Guid Id { get; set; }  public string Nome { get; set; }  public decimal Preco { get; set; }  } }** |

A seguir, criar na pasta **Interfaces** uma interface **IBase.cs**.

A função desta interface é estabelecer contratos de CRUD padrão. Conforme o crescimento da complexidade do domínio, esta ação se paga rapidamente, pois padroniza fortemente a ampliação horizontal do sistema:

|  |
| --- |
| **namespace** **FornecedoresDDD.Dominio.Interfaces** {  **public** **interface** **IBase**<**T**> **where** **T** : **class**  {  **void** **Adicionar**(T obj);    **T** **RecuperarPorId**(Guid id);    **void** **Atualizar**(T obj);     **void** **Excluir**(T obj);    IList<T> **Listar**();  } } |

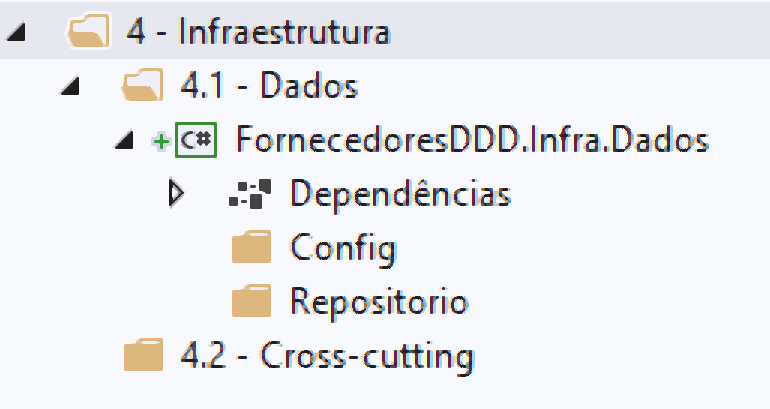
A seguir, criaremos, ainda dentro de **Interfaces**, a interface **IProduto.cs**:

|  |
| --- |
| using FornecedoresDDD.Dominio.Entidades;  using System;  using System.Collections.Generic;  using System.Text;  namespace FornecedoresDDD.Dominio.Interfaces  {  public interface IProduto : IBase<Produto>  {  //Herdar todos os métodos de IBase  //TODO: Outros contratos relevantes para produto  //Exemplo: RecuperarPorNome()  Produto RecuperarPorNome(string nome);  }  } |

Por ora, vamos encerrar o trabalho na pasta **Dominio.**

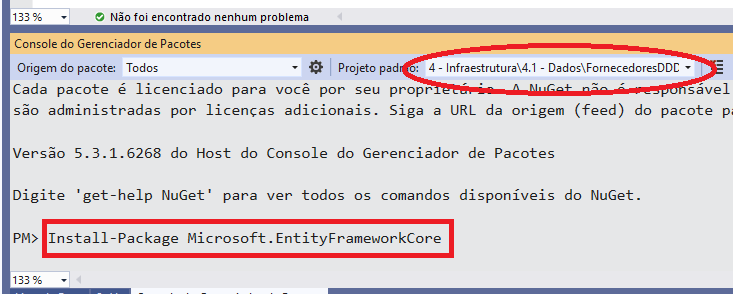
### Trabalhando a Infraestrutura.

Mantenha a seguinte estrutura, dentro de um projeto **Biblioteca de Classes C# (.NET Core)**:



A pasta **Config** abrigará o **contexto** do **Entity Framework**.

Para tal, o **Entity Framework Core** precisa estar configurado dentro do projeto de Infra.



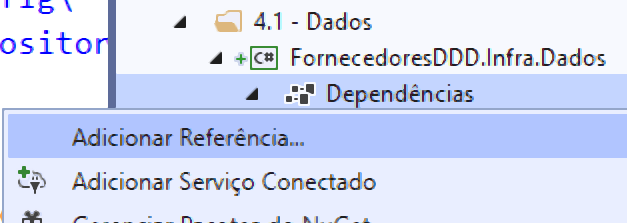
Em apresentação, crie um projeto web MVC.

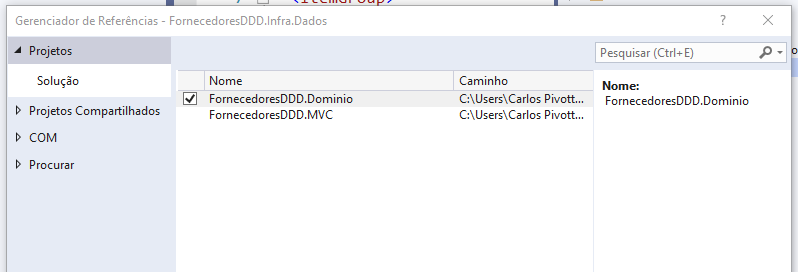
(?) Instale o pacote também no seu projeto MVC: Por ser o projeto inicializador da solução, ele precisará das referências de manipulação de banco de dados via ORM.

Instale também em infra.dados os pacotes: **Microsoft.EntityFrameworkCore.SqlServer**, **Microsoft.EntityFrameworkCore.Design** e **Microsoft.EntityFrameworkCore.Tools** com

|  |
| --- |
| Install-Package Microsoft.EntityFrameworkCore.SqlServer  Install-Package Microsoft.EntityFrameworkCore.Tools  Install-Package Microsoft.EntityFrameworkCore.Design |

Como característica do DDD, o domínio é isolado, mas os demais projetos envolvidos na solução podem - e devem, claro - conhecê-lo. Assim, para trabalhar o projeto de infraestrutura para nosso domínio, devemos adicionar uma referência entre os projetos - da infra para o domínio, mas não o inverso.





Em Config, criamos a classe **FornecedoresDbContext**\*.

\*Nome da solução, FornecedoresDDD + nome da superclasse, DbContext

Esta classe mapeará, com objetos do tipo DbSet, as classes que serão manipuladas no banco de dados.

|  |
| --- |
| using FornecedoresDDD.Dominio.Entidades;  using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.Text;  **namespace** **FornecedoresDDD.Infra.Dados.Config** {  **public** **class** **FornecedoresDbContext** : **DbContext**  {  **public** DbSet<Produto> Produtos { **get**; **set**; }  } } |

E vamos sobrescrever o método OnConfiguring:

|  |
| --- |
| **namespace** **FornecedoresDDD.Infra.Dados.Config** {  **public** **class** **FornecedoresDbContext** : **DbContext**  {  **public** DbSet<Produto> Produtos { **get**; **set**; }    **protected** **override** **void** **OnConfiguring**(DbContextOptionsBuilder optionsBuilder)  {  **if** (!optionsBuilder.IsConfigured)  optionsBuilder.UseSqlServer(GetConnectionString());    **base**.OnConfiguring(optionsBuilder);  }    **private** **string** **GetConnectionString**()  {  *//Alterar para configurações desejadas:*  **return** "Server=(localdb)\\MSSQLLocalDB;Database=FornecedoresDDD;Trusted\_Connection=True;MultipleActiveResultSets=True;";  }  } } |

O método **GetConnectionString** trata apenas de retornar uma string de conexão válida com o banco de dados.

Server=(localdb)\\MSSQLLocalDB;

Configura o servidor como (localdb)\\MSSQLLocalDB

Database=FornecedoresDDD;

Configura o nome do banco de dados como FornecedoresDDD

Trusted\_Connection=True;

[Especifica como um usuário se conecta](https://docs.microsoft.com/en-us/openspecs/sql_server_protocols/ms-odbcstr/6c134f58-30b6-48bd-be5b-ed3a8492d870" \l "Appendix_A_2).

MultipleActiveResultSets=True;

[Habilita múltiplos conjuntos de resultados (ResultSets) ativos](https://docs.microsoft.com/pt-br/dotnet/framework/data/adonet/sql/enabling-multiple-active-result-sets)

Também personalizaremos o construtor:

|  |
| --- |
| **namespace** **FornecedoresDDD.Infra.Dados.Config** {  **public** **class** **FornecedoresDbContext** : **DbContext**  {  **public** DbSet<Produto> Produtos { **get**; **set**; }  **public** FornecedoresDbContext()  {  }  **public** **FornecedoresDbContext**(DbContextOptions<FornecedoresDbContext> options) : **base**(options)  {    } *//Resto do código...* |
| **}** |

Agora, voltamos nossa atenção para a pasta **Repositorio**. Nela, criaremos **RepositorioBase.cs**, de maneira semelhante ao que fizemos com a interface base no domínio:

|  |
| --- |
| **using** FornecedoresDDD.Dominio.Interfaces;  using FornecedoresDDD.Infra.Dados.Config;  using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  **namespace** **FornecedoresDDD.Infra.Dados.Repositorio.RepositorioBase** {  **public** **class** **RepositorioBase**<**T**> : **IBase**<**T**>, **IDisposable** **where** **T** : **class**  {  **public** **void** **Adicionar**(T obj)  {  **throw** **new** NotImplementedException();  }    **public** **T** **RecuperarPorId**(Guid id)  {  **throw** **new** NotImplementedException();  }    **public** **void** **Atualizar**(T obj)  {  **throw** **new** NotImplementedException();  }    **public** **void** **Excluir**(T obj)  {  **throw** **new** NotImplementedException();  }    **public** IList<T> **Listar**()  {  **throw** **new** NotImplementedException();  }    **public** **void** **Dispose**()  {  **throw** **new** NotImplementedException();  }  } } |

Em seguida, criamos o construtor:

|  |
| --- |
| **namespace** **FornecedoresDDD.Infra.Dados.Repositorio.Genericos** {  **public** **class** **RepositorioBase**<**T**> : **IBase**<**T**>, **IDisposable** **where** **T** : **class**  {  **private** **readonly** DbContextOptions<FornecedoresDbContext> \_optionsBuilder;    **public** **RepositorioBase**()  {  \_optionsBuilder = **new** DbContextOptions<FornecedoresDbContext>();  }    *//Resto do código* } |

Assim, o método Adicionar, fica:

|  |
| --- |
| **public** **void** **Adicionar**(T obj) {  **using** (**var** db = **new** FornecedoresDbContext(\_optionsBuilder))  {  db.Set<T>().Add(obj);  db.SaveChanges();  } } |

Usamos a estrutura da classe DbContext do Entity Framework Core para evocar os métodos **Add** (adicionar), que adiciona o objeto criado ao conjunto de objetos rastreados pelo Entity Framework; em seguida, usamos **SaveChanges**, que persiste os dados.

Analogamente, os outros métodos de escrita, Atualizar e Excluir, ficam:

|  |
| --- |
| **public** **void** **Atualizar**(T obj) {  **using** (**var** db = **new** FornecedoresDbContext(\_optionsBuilder))  {  db.Set<T>().Update(obj);  db.SaveChanges();  } }   **public** **void** **Excluir**(T obj) {  **using** (**var** db = **new** FornecedoresDbContext(\_optionsBuilder))  {  db.Set<T>().Remove(obj);  db.SaveChanges();  } } |

E os métodos de leitura, **RecuperarPorId** e **Listar**:

|  |
| --- |
| **public** T **RecuperarPorId**(Guid id) {  **using** (**var** db = **new** FornecedoresDbContext(\_optionsBuilder))  {  **return** db.Set<T>().Find(id);  } }  **public** IList<T> **Listar**() {  **using** (**var** db = **new** FornecedoresDbContext(\_optionsBuilder))  {  **return** db.Set<T>().AsNoTracking().ToList();  } } |

Estes não usam **SaveChanges** por não realizarem persistência no banco.

*Mais sobre DbContext e Entity Framework Core* [*aqui*](https://drive.google.com/file/d/1HJjaXqy-NDlb8yF_BA8yv5JIoqgR7foU/view?usp=sharing)*.*

### Criando a camada de Aplicação: (A camada parece repetir a camada de domínio)

IAppBase.cs:

|  |
| --- |
| **public** **interface** **IAppBase**<**T**> **where** **T** : **class**  {  **void** **Adicionar**(T obj);   T **RecuperarPorId**(Guid id);   **void** **Atualizar**(T obj);   **void** **Excluir**(T obj);   IList<T> **Listar**();  } |

Em FornecedoresDDD.Aplicacao, adicionar a referência do Domínio e de Infra.Dados (da mesma forma que foi adicionado a referência de Domínio em Infra.Dados no exemplo mais acima)

IProdutoApp.cs:

|  |
| --- |
| using FornecedoresDDD.Dominio.Entidades;  using System;  using System.Collections.Generic;  using System.Text;  namespace FornecedoresDDD.Aplicacao.Interfaces  {  public interface IProdutoApp : IAppBase<Produto>  {  }  } |

ProdutoApp.cs

|  |
| --- |
| using FornecedoresDDD.Aplicacao.Interfaces;  using FornecedoresDDD.Dominio.Entidades;  using FornecedoresDDD.Dominio.Interfaces;  using System;  using System.Collections.Generic;  using System.Text;  namespace FornecedoresDDD.Aplicacao  {  public class ProdutoApp : IProdutoApp  {  private readonly IProduto \_produtoInterface;  public ProdutoApp(IProduto produtoInterface)  {  \_produtoInterface = produtoInterface;  }  public void Adicionar(Produto obj)  {  //Validações das Regras de negócio  //Chamar métodos de domínio  \_produtoInterface.Adicionar(obj);  }  public void Atualizar(Produto obj)  {  throw new NotImplementedException();  }  public void Excluir(Produto obj)  {  throw new NotImplementedException();  }  public IList<Produto> Listar()  {  return \_produtoInterface.Listar();  }  public Produto RecuperarPorId(Guid id)  {  throw new NotImplementedException();  }  }  } |

### Consolidando o banco de dados

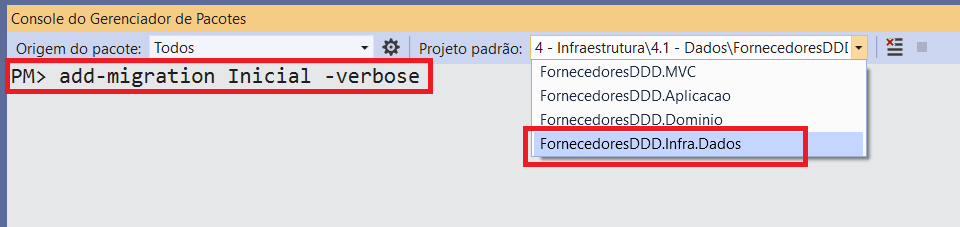
Utilize o comando:

Pelo console, selecionando o projeto padrão Domínio e execute:

PM> Install-Package Microsoft.EntityFrameworkCore.Design

Deixe selecionado o projeto Infra.Dados e selecione também no console Infra.Dados, depois execute execute:

|  |
| --- |
| Add-Migration Inicial |

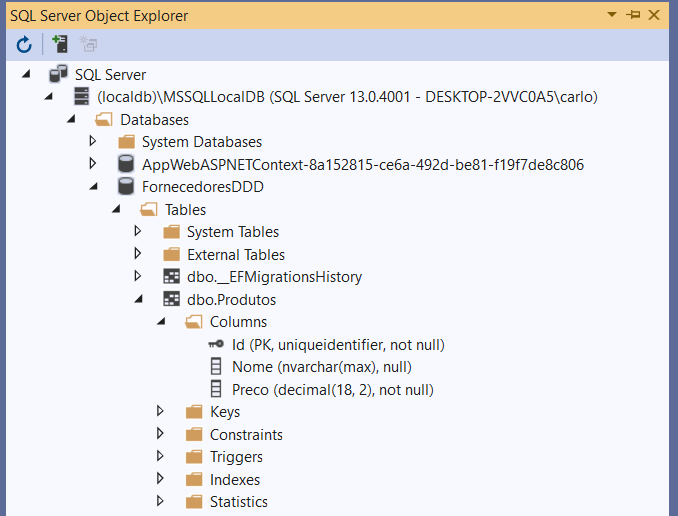


A seguir, podemos usar:

|  |
| --- |
| Update-Database |

Para consolidar o banco.

### A camada de Apresentação



Primeiro, adicionamos as referências na camada de apresentacao:

Aplicacao, Dominio e Infra.Dados.

No projeto web, podemos criar o controlador para Produto e nomeá-lo ProdutoMVCController:

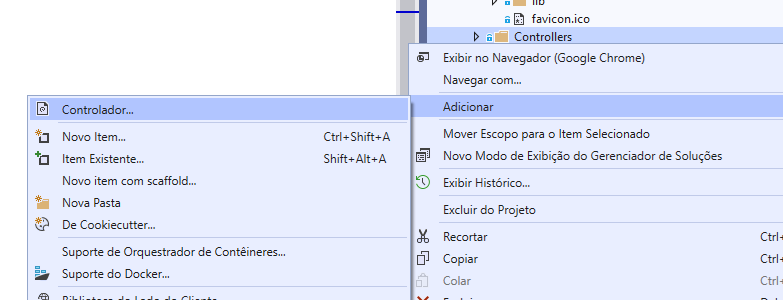
|  |
| --- |
| **~~public~~****~~class~~****~~ProdutoMVCController~~** ~~:~~ **~~Controller~~** ~~{~~ **~~private~~****~~readonly~~** ~~IProdutoApp \_IProdutoApp;~~ **~~public~~****~~ProdutoMVCController~~**~~(IProdutoApp IProdutoApp)  {  \_IProdutoApp = IProdutoApp;  }~~ *~~// GET: ProdutoMVC~~***~~public~~** ~~ActionResult~~ **~~Index~~**~~()  {~~ **~~return~~** ~~View(\_IProdutoApp.Listar());  }~~ *~~// Resto do código~~* ~~}~~ |

Executar neste ponto renderá erros pois várias dependências não foram resolvidas ao longo do caminho. Vamos resolvê-las em Startup.cs.

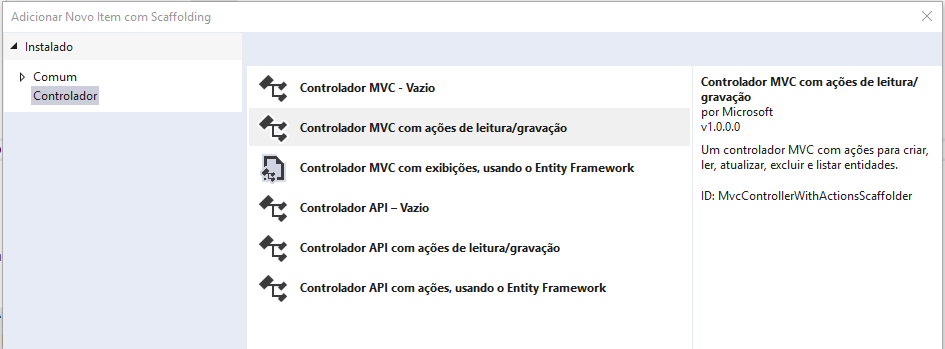
### Modelo, Visão e Controle

|  |
| --- |
| Neste primeiro exemplo, ainda não usaremos AutoMapper, pois nossa classe de modelo MVC é exatamente idêntica à classe do Domínio. Ainda estamos apenas modelando a arquitetura DDD, e não nos debruçaremos ainda sobre riqueza x anemia do modelo. |

Vamos criar uma classe de controle chamada **ProdutoMVCController**. Usaremos este nome para deixar em aberto a ideia da criação de um controlador API.



Usaremos um controlador MVC com ações de leitura e gravação:



ProdutoMVCController.cs

|  |
| --- |
| **public** **class** **ProdutoMVCController** : **Controller** {  **private** **readonly** IProdutoApp \_produtoApp;    **public** **ProdutoMVCController**(IProdutoApp produtoApp)  {  \_produtoApp = produtoApp;  }    *//...* } |

O controlador MVC saberá acessar o domínio por intermédio da camada de aplicação (lembrando que ainda não usaremos AutoMapper neste exemplo, mas que isto é o correto)

Assim, temos a propriedade:

|  |
| --- |
| **private** **readonly** IProdutoApp \_produtoApp; |

Ela receberá valor oriundo de parâmetro no construtor:

|  |
| --- |
| **public** **ProdutoMVCController**(IProdutoApp produtoApp) {  \_produtoApp = produtoApp; } |

Este parâmetro, por sua vez, será resolvido como dependência no Startup.cs.

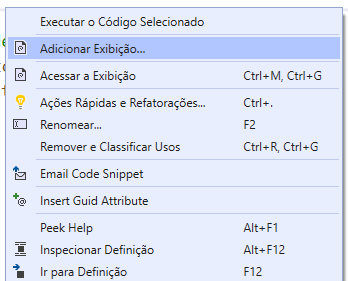
#### Adição

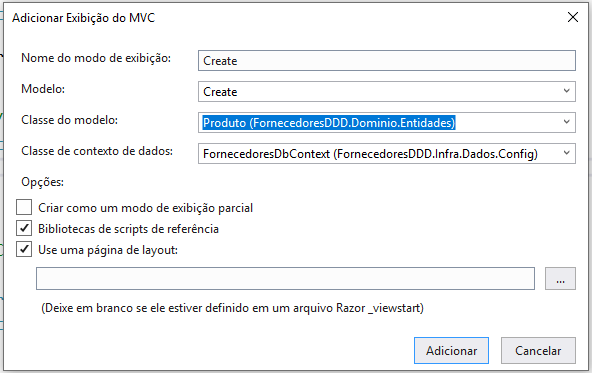
|  |
| --- |
| *// GET: ProdutoMVC/Create* **public** ActionResult **Create**() {  **return** View(); } |

|  |
| --- |
| *// POST: ProdutoMVC/Create* [**HttpPost**] [**ValidateAntiForgeryToken**] **public** ActionResult **Create**(Produto produto) {  **try**  {  produto.Id = Guid.NewGuid();  \_produtoApp.Adicionar(produto);    **return** RedirectToAction(**nameof**(Index));  }  **catch** (Exception e)  {  Console.WriteLine(e.Message);  **return** View();  } } |

Usamos produto.Id = Guid.NewGuid(); para assumir responsabilidade, na aplicação, em tempo de execução, de gerar um id para o objeto. Não se trata de um campo inteiro, que o banco cria através de uma regra simples de identidade no SQL Server.

Neste caso e nos demais, podemos, ao solicitar o menu com o botão direito do mouse, dentro do corpo do método, acessar a opção **Adicionar Exibição**:





Ao testar a solução no momento, tudo funciona - quase.

Caso tentemos usar valores reais para o preço - como, por exemplo, 999,99, haverá erro de validação. Ao serem exibidos em listagens e detalhamentos, porém, os preços serão mostrados usando corretamente a vírgula como separador decimal. Este problema, porém, não será nosso foco.

|  |
| --- |
| Para uma solução para validação de uso de vírgulas X pontos em ASP.NET, veja:  [Problema com decimais, vírgulas e validação do lado do cliente](https://stackoverflow.com/questions/20263762/issue-with-decimals-commas-and-client-side-validation)  [Model binding personalizado no ASP.NET Core](https://docs.microsoft.com/pt-br/aspnet/core/mvc/advanced/custom-model-binding?view=aspnetcore-3.1) |

Opcionalmente, podemos alterar o Startup.cs para explicitamente definir a cultura padrão da aplicação para o português brasileiro:

Startup.cs

|  |
| --- |
| **using** System.Globalization;  using Microsoft.AspNetCore.Localization;  **//....**  **public** **void** **Configure**(IApplicationBuilder app, IWebHostEnvironment env) {  *// Definir cultura padrão para pt-BR*  **var** supportedCultures = **new**[] { **new** CultureInfo("pt-BR") };  app.UseRequestLocalization(**new** RequestLocalizationOptions  {  DefaultRequestCulture = **new** RequestCulture(culture: "pt-BR", uiCulture: "pt-BR"),  SupportedCultures = supportedCultures,  SupportedUICultures = supportedCultures  });    *//....* } |

Isto, porém, ainda não resolve os problemas de validação no cliente.

#### Listagem

|  |
| --- |
| *// GET: ProdutoMVC* **public** ActionResult **Index**() {  **return** View(\_produtoApp.Listar()); } |

#### Detalhamento

|  |
| --- |
| *// GET: ProdutoMVC/Details/5* **public** ActionResult **Details**(Guid id) {  **return** View(\_produtoApp.RecuperarPorId(id)); } |

#### Edição

|  |
| --- |
| *// GET: ProdutoMVC/Edit/5* **public** ActionResult **Edit**(Guid id) {  **return** View(\_produtoApp.RecuperarPorId(id)); } |

|  |
| --- |
| *// POST: ProdutoMVC/Edit/5* [**HttpPost**] [**ValidateAntiForgeryToken**] **public** ActionResult **Edit**(Produto produto) {  **try**  {  *// TODO: Add update logic here*  \_produtoApp.Atualizar(produto);  **return** RedirectToAction(**nameof**(Index));  }  **catch**  {  **return** View();  } } |

#### Exclusão

|  |
| --- |
| *// GET: ProdutoMVC/Delete/5* **public** ActionResult **Delete**(Guid id) {  **return** View(\_produtoApp.RecuperarPorId(id)); } |

|  |
| --- |
| *// POST: ProdutoMVC/Delete/5* [**HttpPost**] [**ValidateAntiForgeryToken**] **public** ActionResult **Delete**(Produto produto) {  **try**  {  *// TODO: Add delete logic here*  \_produtoApp.Excluir(produto);  **return** RedirectToAction(**nameof**(Index));  }  **catch**  {  **return** View();  } } |

### Resolvendo dependências

Em **Startup.cs**, vamos realizar as injeções de dependência:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using FornecedoresDDD.Aplicacao;  using FornecedoresDDD.Aplicacao.Interfaces;  using FornecedoresDDD.Dominio.Interfaces;  using FornecedoresDDD.Infra.Dados.Repositorio;  using Microsoft.AspNetCore.Builder;  using Microsoft.AspNetCore.Hosting;  using Microsoft.AspNetCore.HttpsPolicy;  using Microsoft.Extensions.Configuration;  using Microsoft.Extensions.DependencyInjection;  using Microsoft.Extensions.Hosting;  ///....  **public** **void** **ConfigureServices**(IServiceCollection services)  {  services.AddControllersWithViews();   services.AddSingleton(**typeof**(IBase<>), **typeof**(RepositorioBase<>));  services.AddSingleton<IProduto, ProdutoRepositorio>();  services.AddSingleton<IProdutoApp, ProdutoApp>();  } |