ETAPAS DE DESENVOLVIMENTO

Fonte: [Assistir Aula - Setup da aplicação completa - Curso - Dominando o ASP.NET MVC Core | desenvolvedor.io](https://desenvolvedor.io/curso/dominando-o-asp-net-mvc-core/desenvolvendo-uma-aplicacao-mvc-core-completa/setup-da-aplicacao-completa#status)

**Criando um projeto**

1. Criar a "Solution"
2. Criar as pastas básicas "src", "test", "sql", "doc"
3. Criar o arquivo global.json
4. Criar um projeto dentro da Solution
   1. ASP.NET Core Web App (Model-View\_Controller)

**Criando a camada de negócios**

1. Criar um Project do tipo “Class Library”
   1. Class Library (A Project for creating a class library that targets .NET or .NET Standard)
      1. DevIO.Business (Dentro da pasta “src”)
      2. DevIO.Data (Dentro da pasta “src”)
2. Na pasta **Business** teremos algumas responsabilidades
   1. Criar as pastas
      1. Interfaces
      2. Models
      3. Notificacoes
      4. Services
      5. Validations
3. Na pasta **Data** teremos
   1. Criar as pastas
      1. Context
      2. Mappings
      3. Repository

**Adicionando o conteúdo**

1. Na pasta **Business** teremos algumas responsabilidades
   1. Criar as pastas
      1. Interfaces[[1]](#footnote-1)
         1. DevIO.Business > Add > Class e escolher Interface
         2. Criar um repositório genérico “IRepository.cs”
         3. Criar um repositório para o Fornecedor “IFornecedorRespository.cs”
      2. Models
         1. Entity.cs
         2. Fornecedor.cs
         3. TipoFornecedor.cs
         4. Endereco.cs
         5. Produto.cs
      3. Notificacoes
      4. Services
      5. Validations
2. Na pasta **Data** teremos
   1. Criar as pastas
      1. Context
         1. MeuDbContext.cs[[2]](#footnote-2)
            * Instalar o pacote Install-Package Microsoft.EntityFrameworkCore
            * Criar o construtor
            * Adicionar os DbSet<>
            * Depois de mapeado conforme o item 9.1.2 devo chamar no meu DbContext
            * Garantir que um fornecedor não seja excluído caso exista produtos vinculados a ele
      2. Mappings[[3]](#footnote-3)
         1. ProdutoMapping.cs
            * Realizar o mapeamento dos campos(propriedades)
            * Para utilizar o **HasColumnType** é necessário instalar o pacote Install-Package Microsoft.EntityFrameworkCore.Relational[[4]](#footnote-4)
            * Mais utilizados:

IsRequired()

HasColumnType("varchar(200)”)

ToTable(“Produtos”)

* + - 1. FornecedorMapping.cs
         * Criando a relação 1 : 1 => Fornecedor : Endereco
         * Criando a relação 1 : N => Fornecedor : Produtos
      2. EnderecoMapping.cs
    1. Repository[[5]](#footnote-5)
       1. Criar o repositório genérico “Repository.cs”[[6]](#footnote-6)
       2. Criar o repositório ProdutoRepository.cs
       3. Criar o repositório FornecedorRepository.cs
       4. Criar o repositório EnderecoRepository.cs
  1. Configurar o nosso DbContext
     1. Realizar a configuração de dependências
        1. Botão direito em “Dependencies” > Add Project Reference > Data
     2. Configurar no arquivo “Startup.cs”



* + 1. Configurar o acesso ao banco de dados
  1. Criar uma Migration [[7]](#footnote-7)(Selecionar DevIO.Data no Package Manager Console)
     1. Caso não queira deixar para executar no banco depois será preciso criar um script migration[[8]](#footnote-8).
        1. Executa no Package Manager Console o comando “Script-Migration”
        2. Salvar este arquivo na pasta /sql
        3. Criar agora o script [[9]](#footnote-9)para o acesso aos dados do Identity
  2. Pattern
     1. Vamos utilizar o Repository, para isso, vamos criar uma interface na pasta interface conforme o item 8.1.1.
  3. Renomear a pasta Models para ViewModels
     1. Criar o arquivo EnderecoViewModel.cs
     2. Criar o arquivo FornecedorViewModel.cs
     3. Criar o arquivo ProdutoViewModel.cs
     4. Criar as annotations [[10]](#footnote-10)
     5. Configurar no arquivo “Sturtup.cs”[[11]](#footnote-11)

1. Criar uma pasta “AutoMapper”
   1. Verificar se já está configurado conforme as anotações do item 9.5.5
   2. Criar uma nova class “AutoMapperConfig” que irá herdar de “Profile”
   3. Criar o construtor responsável pela transformação
   4. Aplicar o .ReverseMap()[[12]](#footnote-12)
2. Criar os Controllers[[13]](#footnote-13)
   1. FornecedoresController.rs
      1. Acessar o arquivo Data > “ApplicationDbContext.cs” e excluir um comando[[14]](#footnote-14)
      2. Substituir o contexto desse framework pelo nosso repositório [[15]](#footnote-15)
      3. Criar o construtor desse repositório
      4. Excluir os comentários desnecessários
      5. Realizar as correções utilizando o IMapper
      6. Criar o método ObterFornecedorEndereco()[[16]](#footnote-16)
      7. Configurar para trazer a lista de fornecedores[[17]](#footnote-17)

**Ferramentas úteis**

1. Marketplace Visual Studio
   1. Comment Remover

É preciso instalar o Microsoft.EntityFrameworkCore.Relational para usar o .HasColumnName()

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: <https://desenvolvedor.io/curso/dominando-o-asp-net-mvc-core/desenvolvendo-uma-aplicacao-mvc-core-completa/mapeando-as-entidades-para-o-banco-de-dados>

1. Tendo a Interface implementada agora podemos criar a classe em “DevIO.Data > Repository” Item 9.1.3 [↑](#footnote-ref-1)
2. Lembrar de escolher a camada certa, neste caso, DevIO.Data

    [↑](#footnote-ref-2)
3. Após a finalização do item 9.5 vamos estar utilizando o automapper para fazer esse mapeamento funcionar instalando o pacote na camada de apresentação “Install-Package automapper.extensions.microsoft.dependencyinjection” [↑](#footnote-ref-3)
4. Não esqueça de escolher a camada DevIO.Data, se der erro procure especificar a versão correta. Se a versão do seu .Net for 3, então, seus pacotes precisam começar com 3.X. Exemplo: Install-Package Microsoft.EntityFrameworkCore.Relational -Version 3.1.26 [↑](#footnote-ref-4)
5. Para funcionar é preciso ir ao arquivo Startup.cs e em ConfigureServices fazer a resolução dessa injeção de dependência. [↑](#footnote-ref-5)
6. Se é uma class abstract alguém vai ter que passar alguma coisa para ela, então, devemos ir ao ProdutoRepository.cs e informar o que será passado [↑](#footnote-ref-6)
7. Add-Migration Initial (nome) -Verbose (para ver detalhes) -Context MeuDbContext (para especificar com o contexto, pois, uma migration só pode executar em um context (banco). [↑](#footnote-ref-7)
8. Irá precisar instalar mais um pacote “Install-Package Microsoft.EntityFrameworkCore.SqlServer [↑](#footnote-ref-8)
9. Script-Migration -Context ApplicationDbContext [↑](#footnote-ref-9)
10. O annotation [ScaffoldColumn(false)] na hora de fazer o scaffold não considerar a coluna. Como imagem não pode ser string e precisa ser mapeada, preciso duplicar e alterar o tipo para IFormFile. [↑](#footnote-ref-10)
11. Antes do MVC adicionar o código “services.AddAutoMapper(typeof(Startup));” [↑](#footnote-ref-11)
12. Antes era preciso criar um caminho para transformar do Model > ViewModel e ViewModel > Model, porém, agora não precisa mais, desde que não há troca de parâmetros, os dois façam as mesmas coisas é possível utilizar o comando .ReverseMap(); [↑](#footnote-ref-12)
13. Na pasta Controllers > Add > Controller > “MVC Controller with views, using Entity Framework” [↑](#footnote-ref-13)
14. “public DbSet<DevIO.App.ViewModels.FornecedorViewModel> FornecedorViewModel { get; set; }” [↑](#footnote-ref-14)
15. “private readonly IFornecedorRepository \_fornecedorRepository;” [↑](#footnote-ref-15)
16. Para usarmos sempre que for necessário trazer o endereço de um fornecedor [↑](#footnote-ref-16)
17. “return View(\_mapper.Map<IEnumerable<FornecedorViewModel>>( await \_fornecedorRepository.ObterTodos()));” [↑](#footnote-ref-17)