Autor: Rafael Brasil

Data: 15/01/2021

Utilizando Entity Framework Core CodeFirst com MySql

Quer plugar sua aplicação .NET Core num banco de dados sem escrever uma Query sequer? Há quem prefira usar uma aplicação conectando ao banco e fazendo SELECTs, UPDATEs e DELETEs. Se você sempre fez assim ou se já usa outros ORMs, quero mostrar a você como fazer com Entity Framework Core e MySQL, permitindo que você não fique preso a ambientes Windows e reduza seus custos de infraestutura.

# Introdução

Nos tempos de .NET Framework, quando apenas windows era integralmente suportado, a primeira alternativa dos desenvolvedores de sistemas com .NET era utilizar um banco de dados MSSQL Server pela afinidade entre os produtos Microsoft.

Com a popularização do .NET Core, a possibilidade de se desprender de ambientes Windows e utilizar Linux ou MacOS já é realidade. Ainda assim, para os desenvolvedores que costumam utilizar SQL Server e se perguntam “Como usar MSSQL Server no Linux?”, aqui vai uma alternativa simples e gratuita para usar o EntityFramework com CodeFirst sem necessariamente usar o banco carro-chefe da Microsoft.

Para o tutorial a seguir, adotamos o MySQL pela sua populalidade e pela sua compatibilidade total com as distribuições Linux, além de sua fácil configuração em ambientes windows para quem usa este Sistema Operacional e não quer depender de uma máquina virtual para fazer uma prova de conceito. Contudo, antes de irmos para a partes mais técnica, é importante comentarmos um pouco sobre a teoria por trás do assunto.

## ORM

Um ORM é, basicamente, um tipo de ferramenta de desenvolvimento utilizado para reduzir a diferença de impedância da programação orientada a objetos no uso bancos de dados relacionais. Se você se pergunta o que seria essa “diferença de impedância”, podemos dizer que um banco de dados relacional armazena estruturas de dados e não objetos de negócio; a diferença de impedância consiste nessa diferença entre os sistemas de tipos distintos.

Mas por que usar um ORM no desenvolvimento de um projeto/produto? O principal motivo é o ganho de produtividade, dado que normalmente os ORMs reduzem a quantidade de código necessário para se usar o banco de dados. É possível ter um modelo de dados orientado a objetos e ainda assim conseguir fazer queries rápidas e funcionais no banco de dados.

Exemplo de um possível código sem uso de um ORM:

|  |
| --- |
| var id = 5;  var sql = "SELECT id, firstName, lastName, phoneNumber, birthDate, gender FROM person WHERE id = {0}";  var result = connection.Query(string.Format(sql, id)).ToList();  var name = result[0]["firstName"].ToString(); |

Exemplo de um possível código usando os princípios de um ORM:

|  |
| --- |
| var id = 5;  var person = context.People.GetById(id);  var name = person.FirstName; |

Outro motivo relevante é o fato de os ORMs normalmente permitirem uma escrita mais natural, facilitando o entendimento do código sem necessidade de documentação ou comentários de código.

Alguns exemplos de ORM compatíveis com C# são Entity Framework, Nhibernate, Dapper, Venflow, RepoDb e SQLite-net.

## Entity Framework

O Entity Framework é um ORM compatível com C# muito popular, focado em produtividade, que permite escrever consultas utilizando LINQ. Surgiu com o .NET Framework 3.5 Service Pack 1 e foi incorporado ao .NET Framework 4, mas atualmente a instalação e uso são feitos através de pacotes NuGet. Desde seu lançamento, já recebeu inúmeros recursos e atualizações. É *open source* e é suportado pela Microsoft.

Através de pacotes complementares, conhecidos como *database providers*, é possível utilizar o EF com dezenas de bancos de dados, como SQL Server, MySQL, PostgreSQL, SQLite, Cosmos DB, etc.

O Entity Framework Core (ou EF Core) é uma versão mais leve, de software livre, do popular Entity Framework, compatível com .NET core. A seguir, vamos usar o EF Core para um exemplo prático de uso com um banco de dados MySQL. Adotaremos a abordagem CodeFirst, onde fazemos o código e ele faz a parte do banco de forma transparente para o desenvolvedor.

Obs: Por não ser o foco deste trabalho, não será dada ênfase a padrões de projeto ou organização da solução dentro do Visual Studio. A atenção será voltada ao uso do Entity Framework em si.

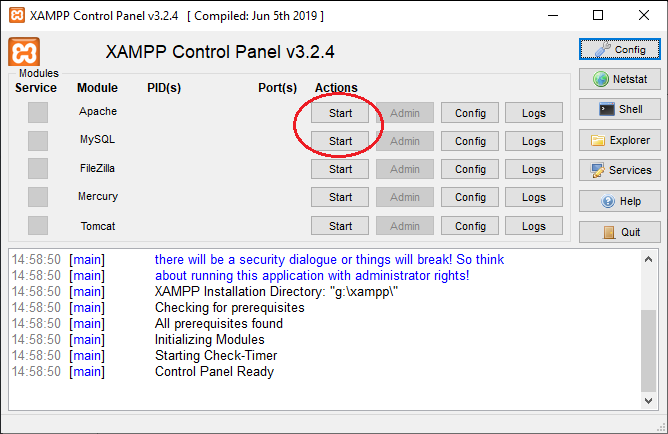
# Requisitos necessários

Será necessário:

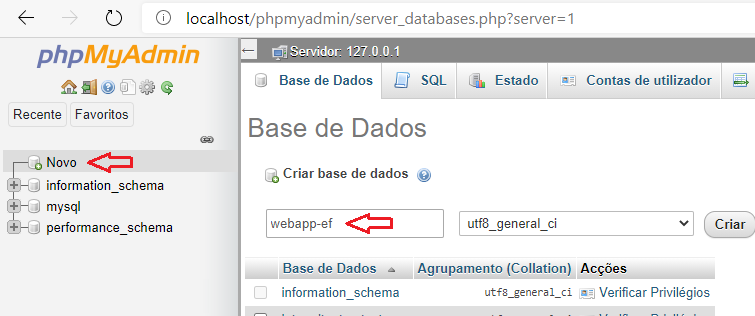
* Visual Studio 2019, qualquer distribuição, preferencialmente em inglês.
* Dotnet core SDK 3.1 ou superior - recomendo 3.1 para ficar igual ao tutorial e por ser a última release LTS até o momento da publicação deste artigo
* [Xampp](https://www.apachefriends.org) completo, com MySQL (ou MariaDB), Apache e PhpMyAdmin

# Configuração do ambiente

1. Abra o xampp, e inicie os serviços Apache e MySQL clicando nos respectivos botões Start.

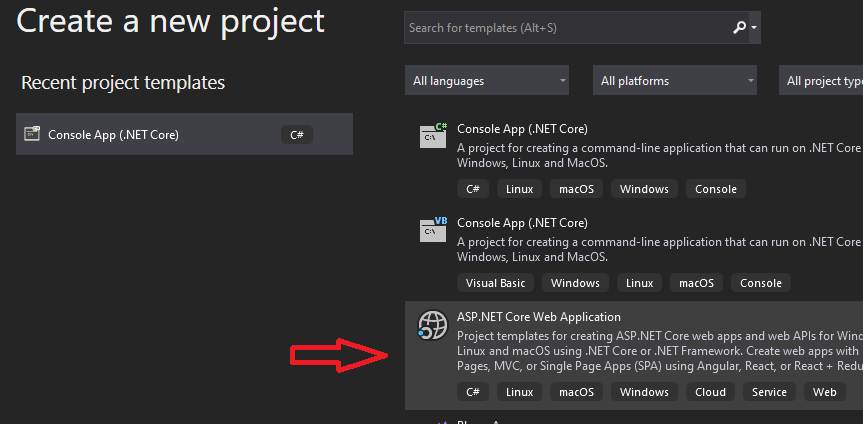


1. Clique no botão Admin do MySQL para abrir o PhpMyAdmin. Uma janela do seu navegador padrão (normalmente o Chrome ou o Edge) irá se abrir.
2. No canto esquerdo da janela do navegador, já no PhpMyAdmin, clique na primeira opção, chamada “Novo”, para criar um novo banco de dados. Em seguida, informe um nome (recomendo webapp-ef para facilitar os próximos passos) e clique em criar. Apesar de eu usar a *collation* **utf8\_general\_ci**, você pode deixar o que vier por padrão. Para o nosso exemplo, a *collation* não fará diferença.



# Criação do projeto

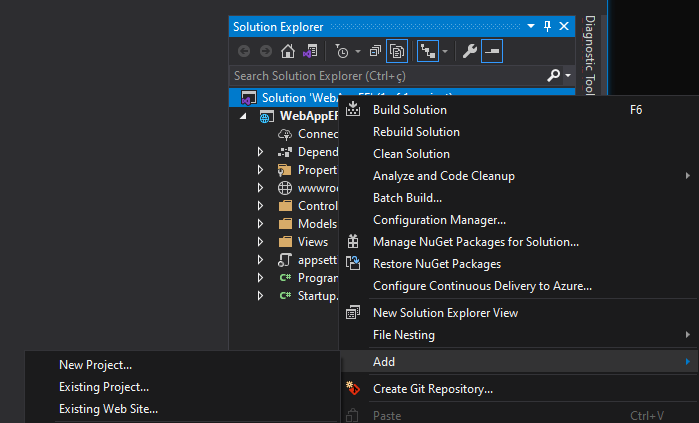
1. Abrir Visual Studio clicar em **Create New Project**.
2. Selecione **ASP.NET Core Web Application** e clique em **Next**.



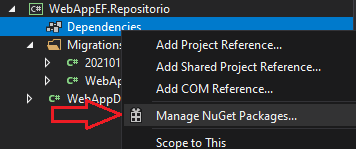
1. Dê um nome para sua aplicação (ex.: WebAppEF) e clique no botão Create.
2. Na lista de templates, selecione **Web Application (Model-View-Controller)** , certifique-se de que a versão do .NET core selecionada é a 3.1 e clique novamente em Create.
3. Aguarde o carregamento. O VS irá abrir sua solution pronta para executar.

Recomendo dar um play (F5) e executar a aplicação para ver se está tudo ok. Se aparecer um alerta de SSL, pode clicar em “Sim” para confiar no certificado SSL autoassinado do IIS. Essa configuração também pode ser desativada nas propriedades de Debug do projeto.

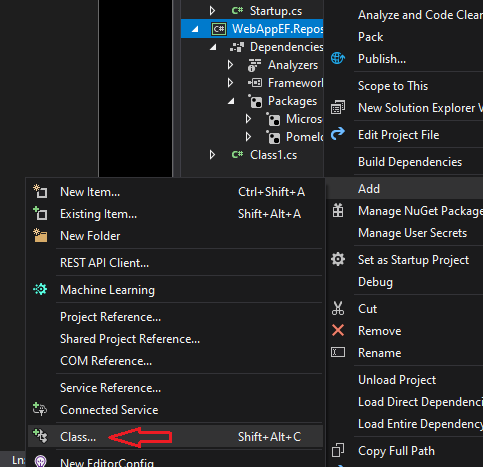
1. Clique com o botão direito na solução WebAppEF e vá em Add > New Project.



1. Selecione **Class Library (.Net Core)** na lista.
2. Forneça um nome para o projeto (por exemplo: WebAppEF.Repositorio) e clique em Create.
3. Dentro do projeto web WebAppEF, clique com o botão direito do mouse em **Dependencies** do, clique em **Add Project Reference**. Selecione o projeto Repositório recém criado na lista e dê OK.
4. Agora no projeto WebAppEF.Repositorio, clique com botão direito do mouse nas dependências do e abra o Nuget Package manager do Repositório clicando em **Manage NuGet Packages**.



1. Na aba Browse, busque e instale os pacotes a seguir
   1. Microsoft.EntityFrameworkCore.Tools – versão 3.1.8 (ferramentas para linha de comando do EF). Outros pacotes do EF Core serão instalados automaticamente para satisfazer dependências.
   2. Pomelo.EntityFrameworkCore.MySql – versão 3.2.4 (provedor específico do MySQL).
2. De volta ao projeto web WebAppEF, clique com botão direito do mouse nas dependências e abra o Nuget Package manager do Repositório clicando em **Manage NuGet Packages**.
3. Na aba **Browse**, busque e instale o pacote Microsoft.EntityFrameworkCore.Design –versão 3.1.8 (complementos para funcionamento correto do CodeFirst).
4. Clique com o botão direito do mouse no projeto Repositório e vá em **Add** > **Class** para criar uma nova classe.



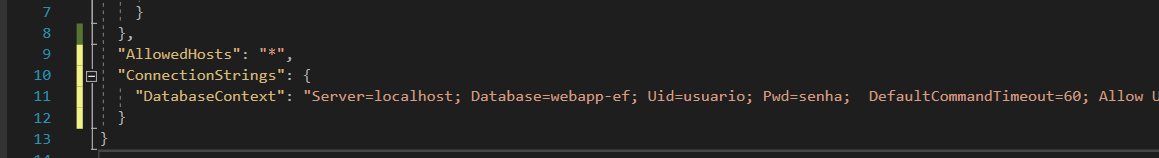
1. Adicione o conteúdo abaixo no novo arquivo:

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  namespace WebAppEF.Repositorio  {  public class WebAppDbContext : DbContext  {  public WebAppDbContext(DbContextOptions<WebAppDbContext> options) : base(options)  {  }  protected override void OnModelCreating(ModelBuilder modelBuilder)  {  }  }  } |

1. Adicione a conection string abaixo em seu appsettings.json no projeto web. Lembre-se de manter o json válido, verificando as vírgulas, chaves e aspas. Para simplificar, não usaremos autenticação no banco. Por isso o usuário na connection string é o root do MySQL.

|  |
| --- |
| "ConnectionStrings": {  "WebAppDbContext": "Server=localhost; Database=webapp-ef; Uid=root; DefaultCommandTimeout=60; Allow User Variables=true"  } |

O resultado deve ficar como na imagem:



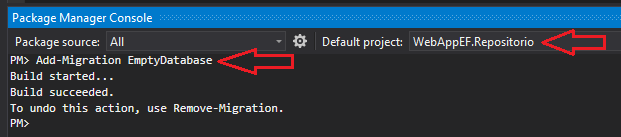
1. Adicione as duas linhas abaixo no topo do arquivo Startup.cs do projeto web.

|  |
| --- |
| using WebAppEF.Repositorio;  using Microsoft.EntityFrameworkCore;  using Microsoft.EntityFrameworkCore.Design; |

1. Inicialize a connection string junto ao contexto. A inicialização é feita no momento da inicialização das dependências, ainda na classe **Startup** do projeto web, dentro do método **ConfigureServices**.

|  |
| --- |
| public void ConfigureServices(IServiceCollection services)  {  // adicione as linhas abaixo dentro da função que já existe  var connection = Configuration.GetConnectionString(nameof(WebAppDbContext));  services.AddDbContext<WebAppDbContext>(options => options.UseMySql(connection));  services.AddScoped<DbContext, WebAppDbContext>();  // manter a linha abaixo que já existe na função  services.AddControllersWithViews();  } |

1. No menu superior do Visual Studio, vá em **Tools** > **NuGet Package Manager** > **Package Manager Console** para abrir o console do NuGet. No console, selecione o projeto Repositório como na imagem abaixo e execute a seguinte linha no console para verificar a configuração do EF:



|  |
| --- |
| Add-Migration EmptyDatabase |

Se funcionou corretamente, aparecerá uma pasta Migrations dentro do projeto Repositório. Nessa pasta ficará todo o versionamento da estrutura do nosso banco de dados em ordem cronológica.

Se ocorreu algum erro, dê uma repassada nos passos anteriores para garantir que está com tudo certo antes de continuar.

# Adicionando estrutura ao banco de dados

1. No projeto Repositório, crie uma classe (assim como fizemos no passo 17) chamada **Usuario** com o conteúdo abaixo.

|  |
| --- |
| namespace WebAppEF.Repositorio  {  public class Usuario  {  public int Id { get; set; }  public string Nome { get; set; }  public string Email { get; set; }  }  } |

1. Adicione uma pasta Mappings e crie uma classe UsuarioMapping com o conteúdo abaixo ainda no projeto Repositório.

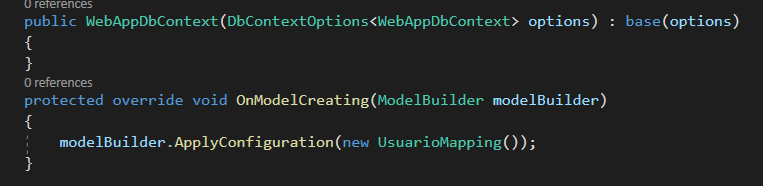
|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using Microsoft.EntityFrameworkCore.Metadata.Builders;  namespace WebAppEF.Repositorio  {  internal class UsuarioMapping : IEntityTypeConfiguration<Usuario>  {  public void Configure(EntityTypeBuilder<Usuario> builder)  {  builder.ToTable(nameof(Usuario))  .HasKey(p => p.Id); // Chave primária da tabela  builder.Property(p => p.Nome)  .HasMaxLength(180); // texto de tamanho 180  builder.Property(p => p.Email)  .HasMaxLength(255); // texto de tamanho 255  }  }  } |

1. Vamos agora configurar o nosso modelo para executar esse mapeamento no banco adicionando a linha abaixo no método OnModelCreating da classe **WebAppDbContext**. Além disso, coloque um *using* na primeira linha do mesmo arquivo.

|  |
| --- |
| modelBuilder.ApplyConfiguration(new UsuarioMapping()); |

|  |
| --- |
| using WebAppEF.Repositorio.Mappings; |

Ficará assim:

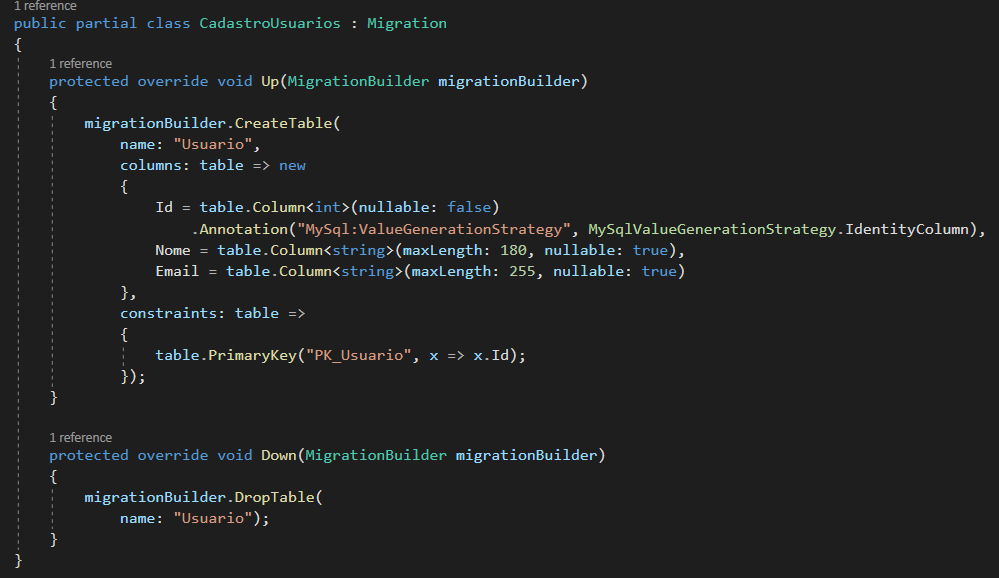


1. Execute o comando no Package Manager Console para adicionar uma nova versão do modelo:

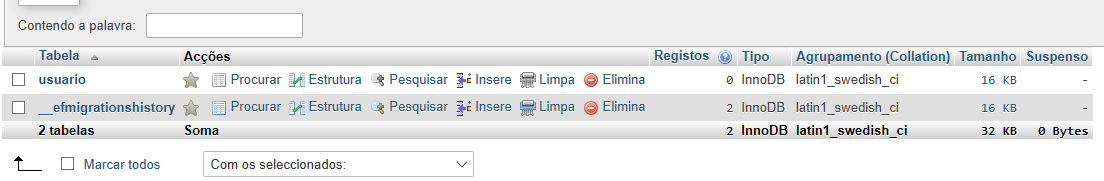
|  |
| --- |
| Add-Migration CadastroUsuarios |

Dica: recomendo não usar somente “Usuario”, pois isso criará uma nova classe com nome Usuario que não terá nenhuma ligação com sua entidade inicial, gerando confusões e conflitos.

Será criada automaticamente uma classe como a abaixo. Ela contém todas as informações da nova tabela e está pronta para ser criada no banco de dados.



1. Execute o comando Update-Database no console para aplicar o migration.
2. Volte ao PhpMyAdmin, selecione o nosso banco webapp-ef e veja sua nova tabela já criada a pronta pra uso.



Uma tabela é criada pelo Entity Framework para gerenciar as versões do seu banco de dados. Com essa única tabela, ele é capaz de aplicar mudanças e fazer rollback com apenas um Update-Database no Package Manager Console.

# Adicionando registros ao banco de dados

O banco já está pronto para receber e prover dados. Vamos agora adicionar um pequeno trecho de código na tela para permitir que usemos nossa conexão com o banco. Para começar, abra, no projeto web, o arquivo WebAppEF/Controllers/HomeController.cs.

1. Adicione um using para o funcionamento do banco de dados e receba o DbContext no contrutor.

|  |
| --- |
| // manter usings anteriores e adicionar os seguintes:  using Microsoft.EntityFrameworkCore;  using WebAppEF.Repositorio;  using System.Linq;  namespace WebAppEF.Controllers  {  public class HomeController : Controller  {  DbContext \_dbContext;  public HomeController(DbContext dbContext)  {  \_dbContext = dbContext;  }  // restante da classe continua aqui em baixo |

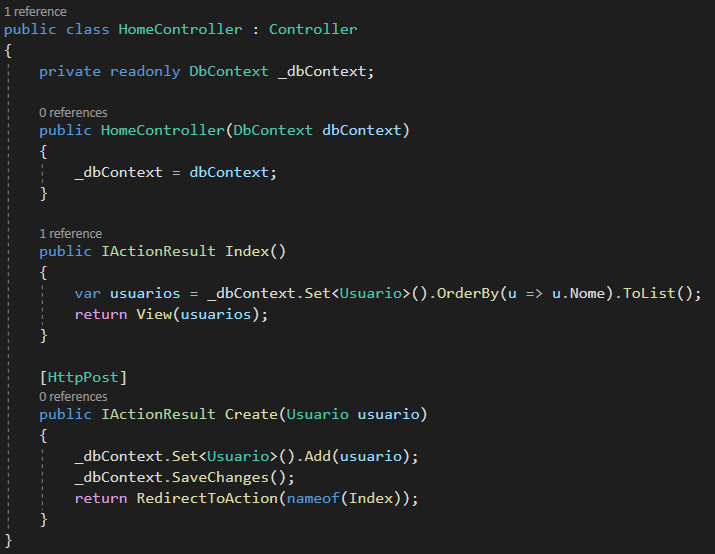
1. Na mesma classe, adicione uma action chamada Create no MVC do projeto web para criar um usuário.

|  |
| --- |
| [HttpPost]  public IActionResult Create(Usuario usuario)  {  \_dbContext.Set<Usuario>().Add(usuario);  \_dbContext.SaveChanges();  return RedirectToAction(nameof(Index));  } |

1. Adicione também um trecho de código para listar os dados. Para esse exemplo, sugiro usar a própria action Index que já existe.

|  |
| --- |
| public IActionResult Index()  {  var usuarios = \_dbContext.Set<Usuario>().OrderBy(u => u.Nome).ToList();  return View(usuarios);  } |

Lembre-se de inicializar o contexto \_dbContext no construtor do controller. A classe inteira deve ficar como assim:



Se o Visual Studio criou outros métodos, você pode removê-los sem problema.

1. Abra o arquivo WebAppEF/Views/Home/Index.cshtml e substitua o conteúdo pelo seguinte:

|  |
| --- |
| @model IEnumerable<WebAppEF.Repositorio.Usuario>  @{  ViewData["Title"] = "Cadastro de usuários";  }  <form action="@Url.Action("Create", "Home")" method="post">  Nome: <input name="nome" /> <br />  Email: <input name="email" /> <br />  <input type="submit" />  </form>  <table>  <thead>  <tr>  <td>Id</td>  <td>Nome</td>  <td>Email</td>  </tr>  </thead>  <tbody>  @foreach(var item in Model)  {  <tr>  <td>@item.Id</td>  <td>@item.Nome</td>  <td>@item.Email</td>  </tr>  }  </tbody>  </table> |

1. Basta executar seu projeto com F5 e utilizar a tela para brincar com os dados.

Caso queira alterar a estrutura do banco de dados, colocando/removendo tabelas, índices, colunas, etc, basta fazê-lo como foi feito para a entidade Usuario, repetindo o processo de Add-Migration e Update-Database antes de executar o projeto para testar.

Se tiver alguma dificuldade e quiser ver como ficou o projeto final, veja em

<https://github.com/rafaelhbrasil/hands-on-efcore>