Autor: Rafael Brasil

Data: 27/02/2020

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

Hands-on EntityFramework Core

# Requisitos necessários

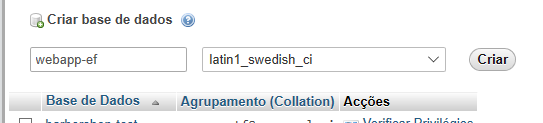
Será necessário:

* Visual Studio 2019, qualquer distribuição
* Dotnet core SDK 3.0 ou superior (recomendo 3.0 para ficar igual ao tutorial)
* Xampp completo, com MySQL, Apache e PhpMyAdmin

# Configuração do ambiente

## Criação do banco de dados

1. Abra o xampp, e inicie os serviços Apache e MySQL clicando nos respectivos botões Start
2. Clique no botão Admin do MySQL para abrir o PhpMyAdmin. Uma janela do seu navegador padrão irá se abrir.
3. No canto esquerdo da janela, clique na primeira opção, chamada “Novo”, para criar um novo banco de dados.

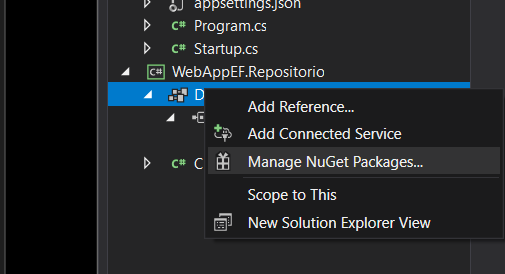


# Criação do projeto

1. Abrir Visual Studio clicar em **Create New Project**
2. Selecione ASP.NET Core Web Application
3. Dê um nome para sua aplicação (ex.: WebAppEF) e clique no botão Create
4. Na lista de templates, selecione Web Application (Model-View-Controller) e clique novamente em Create
5. 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
2. Selecione **Class Library (.Net Core)** na lista
3. Forneça um nome para o projeto (Ex. WebAppEF.Repositorio) e clique em Create
4. No primeiro projeto, clique com o direito do mouse em Dependencies, clique em Add e selecione o projeto recém criado na lista.
5. Clique com direito do mouse nas dependências do repositório e abra o Nuget Package manager

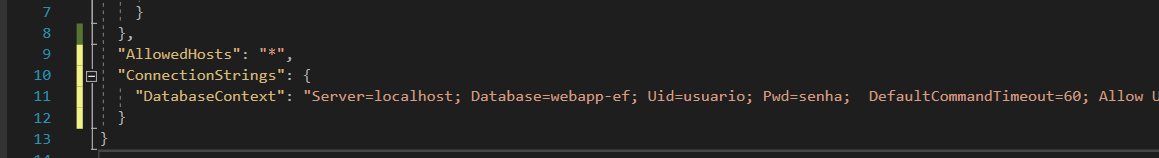


1. Instale os pacotes
   1. Microsoft.EntityFrameworkCore (comum a qualquer conector do EF)
   2. Microsoft.EntityFrameworkCore.Relational (comum a qualquer conector do EF de BD relacional)
   3. Microsoft.EntityFrameworkCore.Tools (ferramentas para linha de comando do EF)
   4. Pomelo.EntityFrameworkCore.MySql (conector específico do MySQL)
   5. Microsoft.EntityFrameworkCore.Design **no projeto web** (complementos para funcionamento correto do CodeFirst)
2. Crie uma nova classe no projeto Repositorio com o conteúdo abaixo:

|  |
| --- |
| 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.

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



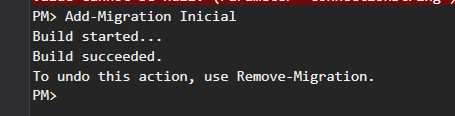
1. Inicialize a connection string junto ao contexto. A inicialização é feita no momento da inicialização das dependências, na classe Startup do projeto web

|  |
| --- |
| public void ConfigureServices(IServiceCollection services)  {  var connection = Configuration.GetConnectionString(nameof(WebAppDbContext));  services.AddDbContext<WebAppDbContext>(options => options.UseMySql(connection));  services.AddScoped<DbContext, WebAppDbContext>();  // manter a linha abaixo  services.AddControllersWithViews();  } |

1. Execute a seguinte linha no Package Manager Console para verificar a configuração do EF:

Add-Migration InitialDatabase

Se tudo ocorrer bem, você verá uma mensagem como abaixo:



Se ocorrer o erro “Value cannot be null. (Parameter 'connectionString')”, verifique sua connection string no appsettings.json. Ela deve ter o mesmo nome da classe de contexto que criamos

# Adicionando estrutura ao banco de dados

1. Adicione uma pasta Models e crie uma classe Usuario com o conteúdo abaixo no projeto Repositório

|  |
| --- |
| public class Usuario  {  public int Id { get; set; }  public string Nome { get; set; }  public string Email { get; set; }  } |

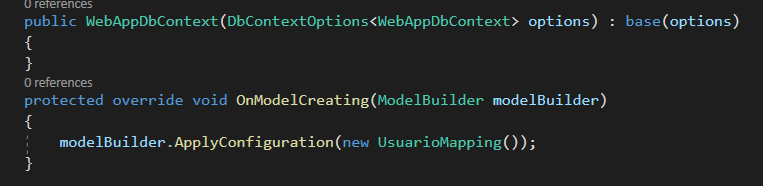
1. Adicione uma pasta Mappings e crie uma classe UsuarioMapping com o conteúdo abaixo no projeto Repositório

|  |
| --- |
| internal class UsuarioMapping : IEntityTypeConfiguration<Usuario>  {  public void Configure(EntityTypeBuilder<Usuario> builder)  {  builder.ToTable(nameof(Usuario))  .HasKey(p => p.Id);  builder.Property(p => p.Nome)  .HasMaxLength(180);  builder.Property(p => p.Email)  .HasMaxLength(255);  }  } |

1. Vamos agora configurar o nosso modelo para executar esse mapeamento no banco adicionando a linha abaixo no método OnModelCreating do nosso Context

modelBuilder.ApplyConfiguration(new UsuarioMapping());

Ficará assim:

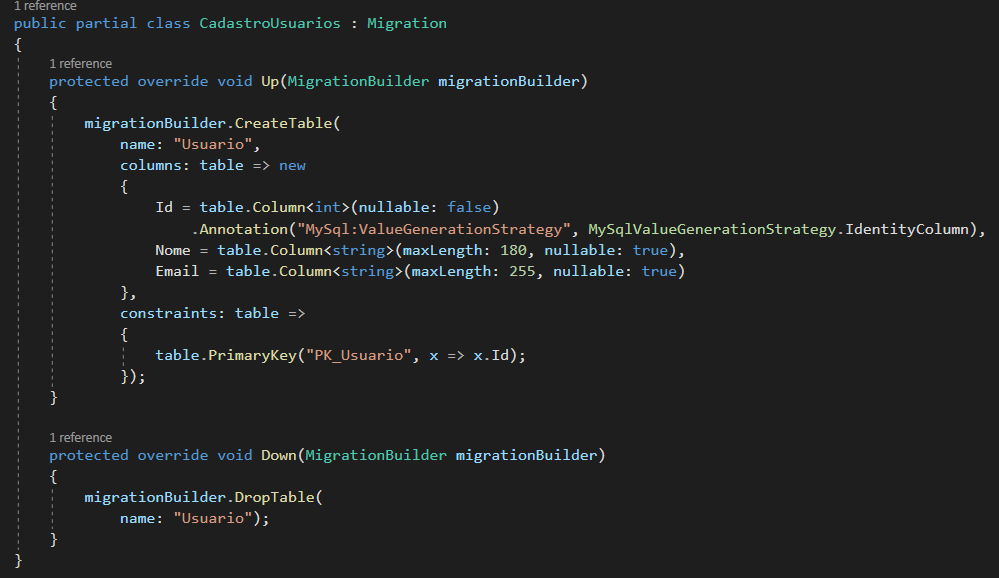


1. Execute o comando no console para adicionar uma nova versão do modelo:

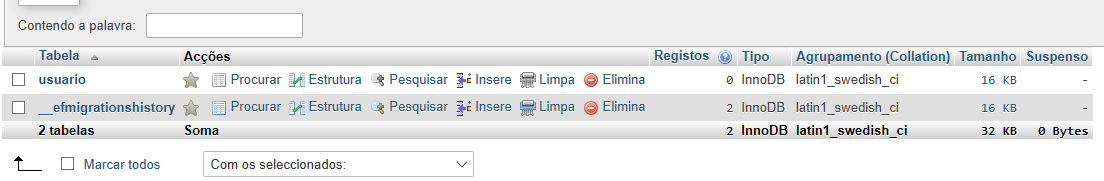
Add-Migration CadastroUsuarios

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

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 e veja sua nova tabela já criada a pronta pra uso.



Uma tabela é criada pelo EntityFramework 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.

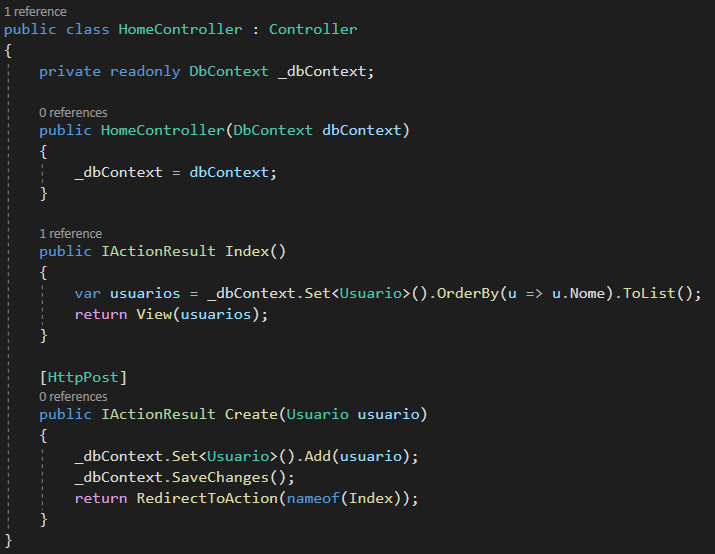
1. Adicione uma action no MVC para criar um usuário. O controller HomeController pode ser usado para adiantar.

|  |
| --- |
| [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 sabemos que funciona.

|  |
| --- |
| 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:



1. Na primeira linha do arquivo WebAppEF/Views/Home/Index.cshtml , adicione a linha

@model IEnumerable<WebAppEF.Repositorio.Models.Usuario>

1. Adicione ao final do arquivo um trecho de HTML + Razor para listar dados e popular o banco:

|  |
| --- |
| <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 a aplicação.