Criando CRUD=== CODIGAO DIA 17

1)Criar Classe em Models

2)Criar o Contexto para as migrations(codigo abaixo)

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ValidacaoRemota.Models;

-namespace ValidacaoDados.Models

{

public class Contexto : DbContext

{

public DbSet<Pessoa> Pessoas { get; set; }

public Contexto(DbContextOptions<Contexto> opcoes) : base(opcoes)

{

}

}

}

3)Conexao com o banco no appsettings.json

-"ConnectionStrings": {

"Conexao": "Server=(localdb)\\mssqllocaldb;Database=ValidacaoRemotaDB;"

},

4)Criar services para o banco no Startup

-services.AddDbContext<Contexto>(opcoes => opcoes.UseSqlServer(Configuration.GetConnectionString("Conexao")));

5)Entrar no PowerShell

-add-migration/update-database

6)Criar um controllador com entity fremework

7)startup e trocar o template do controller

-template: "{controller=Pessoas}/{action=Index}/{id?}");

8)trocar o asp-controller de \_Layout

-<a asp-area="" asp-controller="Pessoas" asp-action="Index" class="navbar-brand">ValidacaoRemota</a>

Material

1. Validação Remota\*

A validação remota é uma regra de negócio que impedi o usuário a cadastrar mais de uma vez o mesmo nome,email ,cpf e etc.

Exemplo:

(Controllador)

public async Task<JsonResult> UsuarioExisteAsync(string nome)

{

if (await \_context.Pessoas.AnyAsync(x => x.Nome == nome))

return Json("Usuario já existe");

return Json(true);

}

(Models)

[Remote("UsuarioExisteAsync","Pessoas")]

public string Nome { get; set; }

1. Validação Personalizada\*

A Validação personalizada são herdadas de ValidationAttribute, é necessário criar uma pasta de validação

Exemplo: Criando uma validação para somente pessoas maiores de idade pode se cadastrar.

(Validação)

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

using ValidacaoPersonalizada.Models;

namespace ValidacaoPersonalizada.Validação

{

public class AdultoAttribute : ValidationAttribute

{

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

Pessoa pessoa = (Pessoa)validationContext.ObjectInstance;

if (pessoa.Idade < 18)

return new ValidationResult("Apenas adultos podem se cadastrar");

return ValidationResult.Success;

}

}

}

(Models)

[Adulto]

public int Idade { get; set; }

1. Mais Algumas anotações de Validação \*

-Table muda o nome da variável do banco(Person para Pessoa) = [Table("Pessoa")]

-Collumn também serve para mudar a variável do banco =

[Key]

[Column("PessoaId", TypeName = "int")]

-DatabaseGenerated = serve para fornecer o valor para o usuário, ou seja, não sera um auto-incremento!!! = [DatabaseGenerated(DatabaseGeneratedOption.None)]

-Display mostra para o usuário o texto = [Display(Name = "Seu Nome")]

-DisplayFormat: é a formatação de data = [DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:dd/MM/yyyy}")]

(models)

[Table("Pessoa")]

public class Person

{

[Key]

[Column("PessoaId", TypeName = "int")]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int PersonId { get; set; }

[Column("Nome", TypeName = "ntext")]

[Display(Name = "Seu Nome")]

public string Name { get; set; }

[Column("CPF", TypeName = "ntext")]

[Display(Name = "Cadastro de pessoa física")]

public string CPF { get; set; }

[Column("Nascimento", TypeName = "date")]

[Display(Name = "Sua data de nascimento")]

[DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:dd/MM/yyyy}")]

public DateTime DateOfBirth { get; set; }

[Column("Idade", TypeName = "int")]

[Display(Name = "Sua Idade")]

public int Age { get; set; }

1. TempData\*

- Usado para armazenamento de dados temporários

-TempData com o método Kepp ele mante os dados

Exemplo se eu ir para pagina about nome ira ser Thiago e idade ira ser 29 com o TempData.Keep(); os dados de “Thiago” e “idade” ira sempre retornar 29 e Thiago

(Controllador)

TempData["nome"] = "Thiago";

TempData["idade"] = "29";

string nome;

int idade;

if (TempData.ContainsKey("nome"))

nome = TempData["nome"].ToString();

if (TempData.ContainsKey("idade"))

idade = int.Parse(TempData["idade"].ToString());

TempData.Keep();

1. ViewData\*

-Transfire dados entre controller e view que é uma lista de valores que se atribui quando o programa é executado.

Exemplo:

(Controller)

IList<Produto> produtos = new List<Produto>();

produtos.Add(new Produto { Nome = "Café", Preco = 5.25 });

produtos.Add(new Produto { Nome = "Leite", Preco = 1.75 });

produtos.Add(new Produto { Nome = "Açucar", Preco = 3.75 });

ViewData["produtos"] = produtos;

(View)

@foreach (var p in ViewData["produtos"] as List<Produto>)

{

<ul>

<li>@p.Nome</li>

<li>@p.Preco</li>

</ul>

}

-View Data com chave-valor

(Controller)

ViewData.Add(new KeyValuePair<string, object>("Nome", "Thiago"));

ViewData.Add(new KeyValuePair<string, object>("Idade", "29"));

(View)

<h2>@ViewData["Nome"]</h2>

<h2>@ViewData["Idade"]</h2>

1. ViewBag\*

São Obejetos dinâmicos, usando para passar do controller para a view

Exemplo: Adicionaremos uma Lista de produtos em Controller e retornara na View

(Controller)

IList<Produto> produtos = new List<Produto>();

produtos.Add(new Produto { Nome = "Café", Preco = 3.75 });

produtos.Add(new Produto { Nome = "Leite", Preco = 1.75 });

produtos.Add(new Produto { Nome = "Açucar", Preco = 5.25 });

ViewBag.produtos = produtos;

(View)

@foreach (var p in ViewBag.produtos)

{

<ul>

<li>@p.Nome</li>

<li>@p.Preco</li>

</ul>

}

7. Modelagem Relacional\*

Relação de tabelas com coleção de tabelas

HasKey -> Defiine a chave primaria

HasColumnName -> define o nome da coluna que será criada no banco de dados

HasMaxLength -> Define quantidade máxima de caractere em um atributo varchar

IsRequired -> Define se o atributo pode ter valor nulo ou não

taTable -> Define da tabela que será criada no banco de dados

(Models:Pessoa) Interrogação somente quando valores Null “isRequired”

public int PessoaId { get; set; }

public string Nome { get; set; }

public int? Idade { get; set; }

public double? Peso { get; set; }

(Models:Context)o método “OnModelCreating” TEM QUE SER DESSE NOME SE NÃO VAI DAR MERDA

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Pessoa>().HasKey(x => x.PessoaId);

modelBuilder.Entity<Pessoa>().Property(x => x.Nome).HasColumnName("Nome").HasColumnType("ntext").HasMaxLength(50).IsRequired();

modelBuilder.Entity<Pessoa>().Property(x => x.Idade).HasColumnName("Idade").HasColumnType("int").IsRequired(false);

modelBuilder.Entity<Pessoa>().Property(x => x.Peso).HasColumnName("Peso").HasColumnType("float").IsRequired(false);

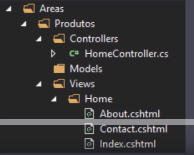
modelBuilder.Entity<Pessoa>().ToTable("Pessoas");

}

8.Areas\*

São usados para organizar funcionalidades relacionadas em grupos, namespaces(Para roteamento) e estruturas de pastas diferentes

É necessário informar a área a qual ele está associado



(Controller:Areas)

[Area("Produtos")]

public class HomeController : Controller

(Startup)

routes.MapAreaRoute(

"RotaUtilizadoArea", "Produtos", "ProdutosUtilizandoArea/{controller}/{action}/{id?}");

(\_Layout)

<li><a asp-area="Produtos" asp-controller="Home" asp-action="Index">Home</a></li>

<li><a asp-area="Produtos" asp-controller="Home" asp-action="About">About</a></li>

<li><a asp-area="Produtos" asp-controller="Home" asp-action="Contact">Contact</a></li>

9.Relacionamento 0...1 e 1...1

.O Mapeamento de 0...1 é quando uma tabela pode ter 0 ou 1 chave estrangeira, ou seja , é opcional o relacionamento.

Exemplo: um relacionamento de pessoa com endereço a pessoa pode cadastrar sem nem um endereço só não pode cadastrar com o mesmo endereço(Chave Primaria)]

.O Mapeamento de 1...1 é quando uma tabela é obrigatório se relacionar com a outra e também não pode se relacionar com a mesma chave primaria

Exemplo: Endereco não pode ter várias Pessoas e vice-versa

Mapeamento de Pessoas= relacionamento 1...1

(Context)

modelBuilder.Entity<Pessoa>()

.HasOne(x => x.Endereco)

.WithOne(x => x.Pessoa)

.HasForeignKey<Pessoa>(x => x.EnderecoId);

Mapeamento de Endereço

(Context)

modelBuilder.Entity<Endereco>().

HasOne(x => x.Pessoa).

WithOne(x => x.Endereco);

\*Atenção: Necessario criar 2 controladores um para Endereço outro par Pessoas

10. Relacionamento 1..N

.Vamos fazer um relacionamento de Empregado para trabalhador: Um empregado tem um trabalhador enquanto o trabalhador tem vários empregados

(Models:Trabalho) é uma Icollection por ter vários Empregados

public int TrabalhoId { get; set; }

(Models:Empregado) è necessário do ID de trabalho

public int TrabalhoId { get; set; }

Mapeamento Empregado(1)

(Context)

modelBuilder.Entity<Empregado>()

.HasOne(e => e.Trabalho)

.WithMany(e => e.Empregados)

.HasForeignKey(e => e.TrabalhoId);

Mapeamento Trabalho(N)

(Context)

modelBuilder.Entity<Trabalho>()

.HasMany(t => t.Empregados)

.WithOne(t => t.Trabalho);

Atenção:Trocar o TrabalhoID para Nome no IACtionResutl

(EmpregadosController:Create)

ViewData["TrabalhoId"] = new SelectList(\_context.Trabalhos, "TrabalhoId", "Nome");

(EmpregadosController:CreateAsync)

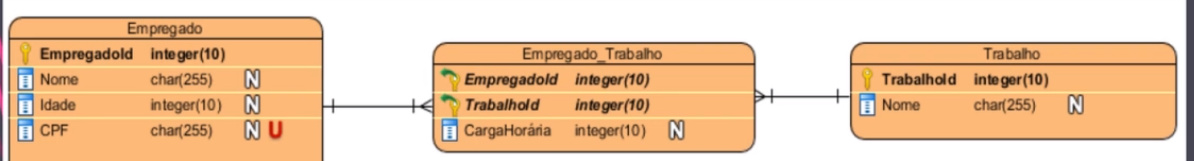
ViewData["TrabalhoId"] = new SelectList(\_context.Trabalhos, "TrabalhoId", "Nome", empregado.TrabalhoId);

(View:Index:Foreach)

@Html.DisplayFor(modelItem => item.Trabalho.Nome)

11.Relacionamento N...N

Para o relacionamento n...n é necessário criar outras tabelas(Tanto para as tabelas quanto para o MVC).



(Models:Empregado) = É necessário o ICollection por ser N EmpregadoTrabalhos

public ICollection<EmpregadoTrabalho> EmpregadoTrabalhos { get; set; }

(Models:EmpregadoTrabalho)

(Models:EmpregadoTrabalho) =

public int EmpregadoId { get; set; }

public Empregado Empregado { get; set; }

public int TrabalhoId { get; set; }

public Trabalho Trabalho { get; set; }

public int CargaHoraria { get; set; }

(Context)= No contexto o Entity é somente para empregadotrabalho,Indetificação de chave primaria

modelBuilder.Entity<EmpregadoTrabalho>().HasKey(x => new { x.EmpregadoId, x.TrabalhoId });

(Context)= Chaves estrangeiras que possui um empregado em que esta relacionado a muitos da tabela EmpregadoTrabalhos e a chave Estrangeira.

modelBuilder.Entity<EmpregadoTrabalho>().HasOne(x => x.Empregado).WithMany(x => x.EmpregadoTrabalhos).HasForeignKey(x => x.EmpregadoId);

(Context)= Chaves Estrangeiras, tem apensa um trabalho relacionado,com muitos empregados trabalhos e com a chave estrangeira de trabalhoID

modelBuilder.Entity<EmpregadoTrabalho>().HasOne(x => x.Trabalho).WithMany(x => x.EmpregadoTrabalhos).HasForeignKey(x => x.TrabalhoId);

-Atenção!!!:Mudar o nome no Iactionresult de Home para EmpregadoTrabalhos

12.Procurar Registro

É uma caixa de texto que filtra o que você quer.

(View:Index) = Caixa de texto para pesquisar e um button para procurar

<**form** **asp-controller**="Pessoas" **asp-action**="Index" method="post">

<input class="form-control" name="txtProcurar"/><button type="submit">Procurar</button>

</**form**>

(Models:PessoaController) = método para procurar

[HttpPost]

public async Task<IActionResult> Index(string txtProcurar)

{

if(!String.IsNullOrEmpty(txtProcurar))

{

return View(await \_context.Pessoas.Where(x => x.Nome.ToUpper().Contains(txtProcurar.ToUpper())).ToListAsync());

}

return View(await \_context.Pessoas.ToListAsync());

}

13.Paginação

Ajuda na otimização do site, utilização de paginação é mostrar a quantidade n de registros de uma pagina

Atenção!!!: Necessario instalação do Nuget “x.PagedList.Mvc.Core” – 7.6.0

(Models:PessoasController)

using X.PagedList;

// GET: Pessoas

public async Task<IActionResult> Index(int? pagina)

{

const int itensPorPagina = 5;

int numeroPagina = (pagina ?? 1);

return View(await \_context.Pessoas.ToPagedListAsync(numeroPagina, itensPorPagina));

}

(Views:Pessoas:Index) = Obrigatorio mudar a model e htmlHelper para os imports

@model X.PagedList.IPagedList<Paginacao.Models.Pessoa>

@using X.PagedList.Mvc.Core

@Html.PagedListPager(Model, pagina => Url.Action("Index", new { pagina }))

14.View Models

É um objeto que tem o objetivo de apensa receber ou exibir todos dados ao usuario