# Projeto Vitrine

**SETUP DO PROJETO**

**Passo 01: Criar a base de dados**

create database vitrine\_csharp;

use vitrine\_csharp;

create table usuarios(

usuarioid int primary key auto\_increment,

nome varchar(180),

email varchar(180),

senha text

)engine=innoDb;

create table produtos(

produtoid int primary key auto\_increment,

nome varchar(180),

descricao text,

preco decimal(10,2),

ativo tinyint,

usuarioid int not null,

foreign key (usuarioid) references usuarios(usuarioid)

)engine=innoDb;

create table imagens(

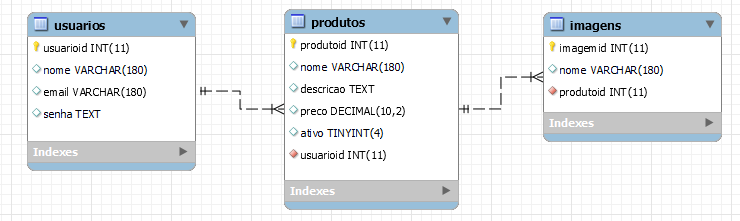
imagemid int primary key auto\_increment,

nome varchar(180),

produtoid int not null,

foreign key(produtoid) references produtos(produtoid)

)engine = InnoDB;



**Passo 02: Create a new project**

**ASP.NET Core Web App (Model-View-Controller)**

Applicativo Web do Asp.NET Core (Model-View-Controller)

**Na Tela: Confire your new project**

Project Name: nome\_do\_projeto

Location: escolha o local para salvar

**Na Tela: Additional information**

Framework: .NET 8.0

Authentication type: none

[x] Configure for HTTPS

Create

Execute a aplicação para testar

**Passo 02: Instalar Pacotes**

Clique com o botão direito no mouse no projeto.

- Manage NuGet Packages

Microsoft.EntityFrameworkCore [8.0.13]

Microsoft.EntityFrameworkCore.Design [8.0.13]

Pomelo.EntityFrameworkCore.MySql [8.03]

[Cuidado com conflitos entre os pacotes e a versão do banco]

**Passo 04: Criação da Classe de Modelo**

Na pasta Models, clique com o botão direito → Add → Class → **NomeDaClasseModelo.cs**:

Vamos criar as classes Usuario, Produto e Imagem.

**Classe Usuario**

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Vitrine.Models {

[Table("usuarios")]

public class Usuario {

[Key]

public int UsuarioId { get; set; }

[Required(ErrorMessage ="Nome é um campo obrigatório")]

[StringLength(180, MinimumLength = 3 ,

ErrorMessage ="O campo deve ter entre 3 e 180 caracteres")]

[Display(Name = "Nome do usuário:")]

public string Nome { get; set; }

[Required(ErrorMessage = "O e-mail é um campo obrigatório")]

[EmailAddress(ErrorMessage = "Informe um e-mail válido")]

[Display(Name = "Seu melhor e-mail:")]

public string Email { get; set; }

[Required(ErrorMessage = "A senha é um campo obrigatório")]

[MinLength(6, ErrorMessage ="A senha deve ter no mínimo 6 caracteres")]

[DataType(DataType.Password)]

[Display(Name = "Criar sua senha:")]

public string Senha { get; set; }

// propriedade de navegação - Possui uma coleção de Produtos

public ICollection<Produto> Produtos { get; set; } = new List<Produto>();

}

}

**Classe Produto**

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Vitrine.Models {

[Table("produtos")]

public class Produto {

[Key]

public int ProdutoId { get; set; }

[Required(ErrorMessage ="O campo nome é obrigatório")]

[StringLength(180, MinimumLength =3,

ErrorMessage ="O nome deve ter entre 3 e 180 caracteres")]

[Display(Name ="Nome do produto:")]

public string Nome { get; set; }

[Display(Name ="Descrição do produto:")]

public string? Descricao { get; set; }

[Column(TypeName ="decimal(10,2)")]

[Display(Name ="Preço do produto:")]

public decimal? Preco { get; set; }

[Display(Name="Produto ativo:")]

public bool Ativo { get; set; }

[ForeignKey("Usuario")]

public int UsuarioId { get; set; } // FK

public Usuario? Usuario { get; set; } // Navegação para Usuário

// propriedade de navegação - Possui uma coleção de Imagens

public ICollection<Imagem> Imagens { get; set; } = new List<Imagem>();

}

}

**Classe Imagem**

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Vitrine.Models {

[Table("imagens")]

public class Imagem {

[Key]

public int ImagemId { get; set; }

[Required, StringLength(180)]

public string Nome { get; set; }

[NotMapped]

// Usado apenas para upload no formulário

public IFormFile? ArquivoImagem { get; set; }

[ForeignKey("Produto")]

public int ProdutoId { get; set; } // FK

public Produto? Produto { get; set; } // Navegação para Produto

// Método auxiliar para gerar nome único com base em timestamp

public static string GerarNomeArquivo(string nomeOriginal) {

var extensao = Path.GetExtension(nomeOriginal);

var timestamp = DateTime.Now.ToString("yyyyMMddHHmmssfff");

return $"{timestamp}{extensao}";

}

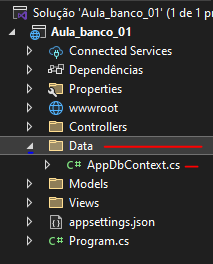
}

}

**Passo 05: Configuração do Contexto**

Crie uma pasta chamada **Data**

Adicione uma nova classe: **AppDbContext.cs**



// Importa o namespace onde está definida a classe Cadastro

using Microsoft.EntityFrameworkCore;

using Vitrine.Models;

namespace Vitrine.Data {

public class AppDbContext : DbContext {

// Construtor que recebe as opções de configuração do contexto

public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)

{ }

// Define as tabelas no banco de dados

// representada pela entidade Usuario, Produto e Imagem

public DbSet<Usuario> Usuarios { get; set; }

public DbSet<Produto> Produtos { get; set; }

public DbSet<Imagem> Imagens { get; set; }

}

}

**Passo 06: Configurar a String de Conexão no appsettings.json**



Adicione a string de conexão:

{

**"ConnectionStrings": {**

**"MySqlConnection":**

**"server=localhost;port=3306;database=vitrine\_csharp;user=root;password="**

**},**

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

**Passo 07: Registrar o Contexto no Program.cs**

**Registra as informações de conexão e o MapControllerRoute**

**using Microsoft.EntityFrameworkCore;**

**using Vitrine.Data;**

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

**//---------------------------------------------------------------------**

**// Lê a string de conexão do appsettings.json**

**var connectionString = builder.Configuration.GetConnectionString("MySqlConnection");**

**builder.Services.AddDbContext<AppDbContext>(**

**// auto detecta a versão do MySQL**

**options => options.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString))**

**);**

**//---------------------------------------------------------------------**

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

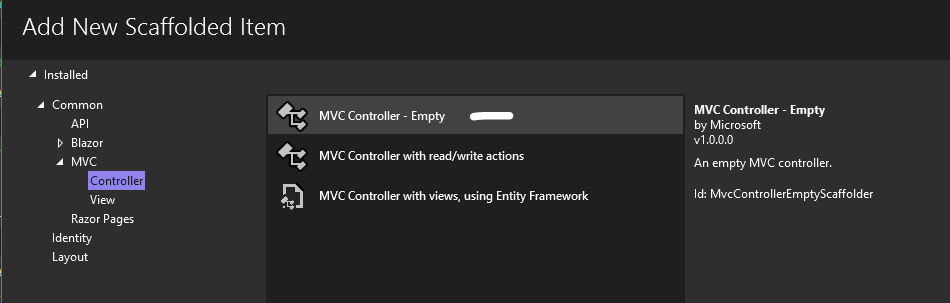
**FIM DO SETUP DO PROJETO**

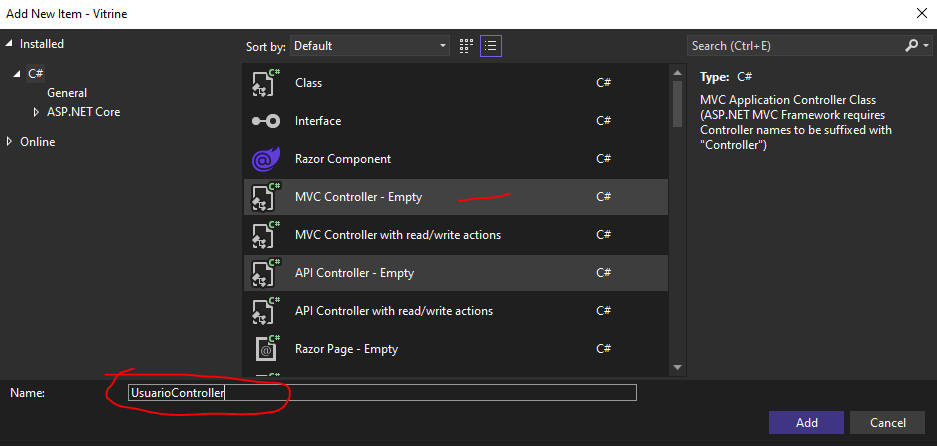
**Controllers**

**Observação: Em nosso controller vamos assumir que a Action Index sempre será o formulário de cadastro.**

**Controller Usuario**

Na pasta Controllers, clique com o botão direito → Add → Controller → **UsuarioController.cs.**





**Cadastrando o Usuário - UsuarioController - V1**

using Microsoft.AspNetCore.Mvc;

namespace Vitrine.Controllers {

public class UsuarioController : Controller {

//---------------------------------------------------------------------------------

// Index - vai exibir o formulário de cadastro

public IActionResult Index() {

return View();

}

//---------------------------------------------------------------------------------

// CadastrarUsuario - Vai inserir os dados do usuario na base de dados

public IActionResult CadastrarUsuario() {

return View();

}

//---------------------------------------------------------------------------------

public IActionResult Sucesso() {

return View();

}

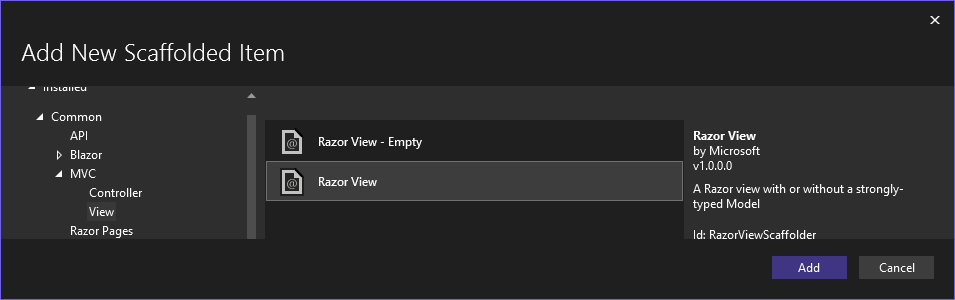
}

}

**Criando a View Index**

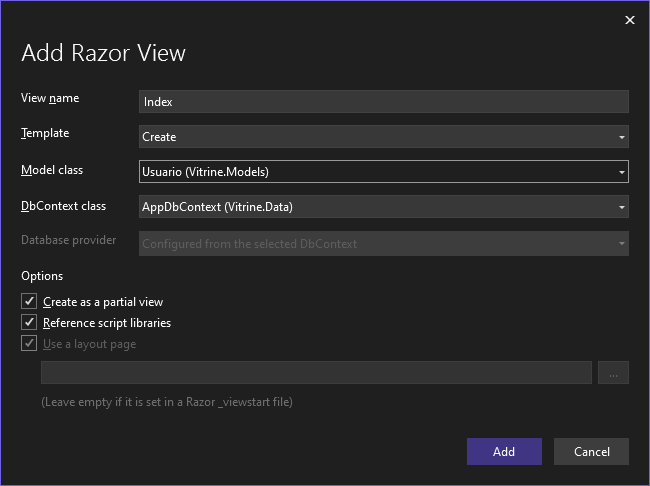
- Clique com o Botão Direito sobre a IActionResult Index -> Add View

- Escolha Razor View



- Selecione as opções corretas

- Add



**Código da View Index - Formulário**

@model Vitrine.Models.Usuario

<h4>Usuario</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="CadastrarUsuario" method="post">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-group">

<**label** **asp-for**="Nome" class="control-label"></**label**>

<**input** **asp-for**="Nome" class="form-control" />

<**span** **asp-validation-for**="Nome" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Email" class="control-label"></**label**>

<**input** **asp-for**="Email" class="form-control" />

<**span** **asp-validation-for**="Email" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Senha" class="control-label"></**label**>

<**input** **asp-for**="Senha" class="form-control" />

<**span** **asp-validation-for**="Senha" class="text-danger"></**span**>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

**Código da View Sucesso**

@{

}

<h2>Usuário cadastrado com sucesso!</h2>

<p>O usuário foi inserido na base de dados.</p>

<**a** **asp-controller**="Usuario" **asp-action**="Index" class="btn btn-success">Cadastrar novo usuário</**a**>

**Cadastrando o Usuário - UsuarioController - V2**

using Microsoft.AspNetCore.Mvc;

using Vitrine.Data;

using Vitrine.Models;

namespace Vitrine.Controllers {

public class UsuarioController : Controller {

**private readonly AppDbContext \_context;**

//-----------------------------------------------

// Injeção de dependência do DbContext

**public UsuarioController(AppDbContext context) {**

**\_context = context;**

**}**

//-----------------------------------------------

// GET: Exibe formulário de cadastro

public IActionResult Index() {

return View();

}

//-----------------------------------------------

// CadastrarUsuario - Vai inserir os dados do usuario na base de dados

// POST: Recebe dados do formulário e salva no banco

**[HttpPost]**

**[ValidateAntiForgeryToken]**

**public IActionResult CadastrarUsuario(Usuario usuario) {**

**if (ModelState.IsValid) {**

**\_context.Usuarios.Add(usuario);**

**\_context.SaveChanges();**

**return RedirectToAction("Sucesso");**

**}**

**// Se os dados forem inválidos, volta para o formulário mostrando os erros.**

**return View(usuario);**

}

//-----------------------------------------------

// Página simples de confirmação

public IActionResult Sucesso() {

return View();

}

}

}

**Observação:**

Até aqui o cadastro está funcionando mas a senha não está protegida.

Em “**Models**” vamos criar uma classe “**Criptografia**” para proteger a senha.

**Classe Criptografia**

using System.Security.Cryptography;

using System.Text;

namespace Vitrine.Models {

public static class Criptografia {

// Chave secreta de 32 bytes (256 bits) para criptografia AES

// deve ter 32 caracteres

private static readonly string chaveSecreta = "ChaveUltraSecreta123456789012345";

// Vetor de inicialização (IV) de 16 bytes

// deve ter 16 caracteres

private static readonly string vetorInicializacao = "1234567890123456";

// Método para criptografar uma string e retornar o texto criptografado em Base64

public static string Criptografar(string texto) {

// Verifica se o texto é nulo ou vazio

if (string.IsNullOrEmpty(texto))

return string.Empty;

// Converte a chave e o IV em arrays de bytes

byte[] chave = Encoding.UTF8.GetBytes(chaveSecreta);

byte[] iv = Encoding.UTF8.GetBytes(vetorInicializacao);

// Cria um objeto AES

using (Aes aesAlg = Aes.Create()) {

aesAlg.Key = chave; // define a chave

aesAlg.IV = iv; // define o IV

// Cria um objeto para criptografar os dados

ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

// Usa um stream para escrever os dados criptografados

using (MemoryStream msEncrypt = new MemoryStream()) {

// Cria o stream de criptografia

using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write)) {

// Converte o texto em bytes

byte[] textoBytes = Encoding.UTF8.GetBytes(texto);

// Escreve os dados no stream criptografado

csEncrypt.Write(textoBytes, 0, textoBytes.Length);

csEncrypt.FlushFinalBlock();

// Retorna o conteúdo criptografado em formato Base64

return Convert.ToBase64String(msEncrypt.ToArray());

}

}

}

}

// Método para decriptografar uma string criptografada e retornar o texto original

public static string Decriptografar(string textoCriptografado) {

// Verifica se o texto é nulo ou vazio

if (string.IsNullOrEmpty(textoCriptografado))

return string.Empty;

// Converte a chave e o IV em arrays de bytes

byte[] chave = Encoding.UTF8.GetBytes(chaveSecreta);

byte[] iv = Encoding.UTF8.GetBytes(vetorInicializacao);

// Converte o texto criptografado de Base64 para array de bytes

byte[] textoBytes = Convert.FromBase64String(textoCriptografado);

// Cria um objeto AES

using (Aes aesAlg = Aes.Create()) {

aesAlg.Key = chave; // define a chave

aesAlg.IV = iv; // define o IV

// Cria um objeto para descriptografar os dados

ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

// Usa um stream para ler os dados descriptografados

using (MemoryStream msDecrypt = new MemoryStream(textoBytes)) {

// Cria o stream de descriptografia

using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read)) {

// Lê os dados do stream e retorna como string

using (StreamReader srDecrypt = new StreamReader(csDecrypt)) {

return srDecrypt.ReadToEnd();

}

}

}

}

}

}

}

//string original = "minhaSenha123";

//string criptografado = Criptografia.Criptografar(original);

//string decriptografado = Criptografia.Decriptografar(criptografado);

//Console.WriteLine($"Original: {original}");

//Console.WriteLine($"Criptografado: {criptografado}");

//Console.WriteLine($"Decriptografado: {decriptografado}");

**Finalizando o Cadastro**

Refatore a ActionResult CadastrarUsuario

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult CadastrarUsuario(Usuario usuario) {

if (ModelState.IsValid) {

**usuario.Senha = Criptografia.Criptografar(usuario.Senha);**

\_context.Usuarios.Add(usuario);

\_context.SaveChanges();

return RedirectToAction("Sucesso");

}

return View(usuario);

}