**APLICAÇÃO WEB COM ASP.NET CORE BLAZOR**

Projeto inicial: <https://github.com/alura-cursos/ScreenSound-BlazorWeb/tree/master>

Criar um projeto, na solução, que vai ser o front-end, com Blazor

Botão direito na solução-> Adicionar projeto-> APlicativo Autônomo Blazor WebAssembly

Nome: ScreenSound.Web

Estrutura: .NET 8.0

**CRIAÇÃO DE UMA PÁGINA DE ARTISTAS**

Pasta Pages -> Adicionar -> Razor page -> Artistas.razor

@page "/Artistas"

<h3>Artistas</h3>

<h4>@Mensagem</h4>

@code {

public string? Mensagem { get; set; } = "Olá !";

/\*

DEFINIÇÃO DA ROTA

@page "/Artistas"

CORPO DA PÁGINA

<h3>Artistas</h3>

<h4>@Mensagem</h4>

CÓDIGO C#

@code {

public string? Mensagem { get; set; } = "Olá !";

\*/

}

**CONFIGURAR ScreenSound.Web PARA CONSUMIR OS CLIENTES DA API**

***Pacotes necessários: Microsoft.Extensions.Http - v8.0.0***

Criar uma classe ArtistaAPI numa pasta Services, que vai ser a classe contendo os métodos desse serviço de consumo de Clientes da ScreenSound.API

Services/ArtistaAPI

public class ArtistaAPI

{

//PROPRIEDADES - CAMPOS

private readonly HttpClient \_httpClient;

//CONSTRUTOR

public ArtistaAPI(IHttpClientFactory factory)

{

\_httpClient = factory.CreateClient("API");

}

//MÉTODO QUE RETORNA UMA COLEÇÃO DE ArtistaResponse DA API

public async Task<ICollection<ArtistaResponse>?> GetArtistasAsync()

{

return await

\_httpClient.GetFromJsonAsync<ICollection<ArtistaResponse>>("artistas");

}

}

Para que a classe ArtistaAPI funcione, ela precisa das pastas Request e Response, que estão no projeto ScreenSound.API, então, copiamos ela para o projeto atual, sincronizando os namespaces. (*Porquê copiar ao invés de incluir nas dependências?*)

Copiar as pastas Response e Request;

Botão direito em ScreenSound.Web->Sincronizar namespaces;

No program, injetamos as dependências para que o ASP.NETCore nativamente gerencie o ciclo de vida dos objetos gerados pelos serviços ArtistaAPI e HttpClient

//ADICIONANDO SERVIÇO DE ArtistaAPI - CONSUMO DE Artistas da API

builder.Services.AddTransient<ArtistaAPI>();

//ADICIONANDO SERVIÇO DE HttpClient

builder.Services.AddHttpClient("API", client =>

{

client.BaseAddress = new Uri(builder.Configuration["APIServer:Url"]!);

client.DefaultRequestHeaders.Add("Accept", "application/json");

});

Observando a injeção do HttpClient, temos que client.BaseAdress recebe uma Uri que é gerada em builder.Configuration[“APIService:Url”], ou seja, esse serviço httpclient vai ser acessado na Uri que estiver configurada no campo Url dentro do campo APIService, que no caso de projetos razor, ficam em appsettings.json, dento de wwwroot

*Botão direito em wwwroot -> Adicionar item -> Arquivo de configurações do aplicativo*

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"APIServer": {

"Url": "https://localhost:7089"

}

}

<https://localhost:7089> é a url onde a API é executada localmente

**CONFIGURANDO UMA PÁGINA RAZOR PARA CONSUMIR DA API - LISTAGEM DE CLIENTES**

Mudar a página Pages/Artistas.razor

@page "/Artistas"

@inject ArtistaAPI artistaApi

<h3>Artistas</h3>

@if(artistas is not null)

{

foreach(var artista in artistas)

{

<p>@artista.Nome</p>

}

}

@code {

//ATRIBUTOS - CAMPOS

private ICollection<ArtistaResponse>? artistas;

//DEMAIS MÉTODOS

//SOBRECARGA DO MÉTO OnInitializedAsync PARA QUE, SEMPRE QUE A ROTA Artistas

//FOR CONSULTADA, A COLEÇÃO artistas VAI RECEBER O RESULTADO DE

// artistaAPI.GetArtistasAsync(), QUE FOI DEFINIDA EM ArtistaAPI.cs

protected override async Task OnInitializedAsync()

{

artistas = await artistaApi.GetArtistasAsync();

}

//OU SEJA, SEMPRE QUE ACESSADO A ROTA Artistas

//É MOSTRADO NA TELA A LISTAGEM DE ARTISTAS - MÉTODO app.MapGet("Artistas") DA API

}

Destacando que, as dependências usadas nas páginas razor podem ser definidas na própria página, ou usando uma alternativa mais elegante: inseridas em \_Imports.razor

No nosso caso até o momento, adicionamos as seguintes dependências

@using ScreenSound.Web.Response

@using ScreenSound.Web.Services

E para que efetivamente o projeto ScreenSound.Web (páginas razor) possa consumir o projeto ScreenSound.API, precisamos que a API esteja rodando no momento que acessarmos a rota /Artistas nas páginas razor.

Para que esses dois projetos sempre executem ao mesmo tempo

-Configurar projetos de Inicialização;

-Marcar os dois projetos como inicializar;





Assim, a execução fica marcada como “**Novo perfil**” e esse perfil faz a execução em paralelo dos dois projetos

Mas, para que ambos projetos rodem ao mesmo tempo, eles **não podem estar configurados para poder escutar nas mesmas portas**, e no momento, os dois estão configurados para poder escutar na porta [http://localhost:5241](http://localhost:5241,)

Para mudar isso, defini que o projeto API não vai escutar nessa porta. Isso pode ser mudado em ScreenSound.API/Properties/launchSettings.json, nos campos https e https-production

"https": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"launchUrl": "Swagger/index.html",

"applicationUrl": "https://localhost:7089",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"https-production": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"launchUrl": "Swagger/index.html",

"applicationUrl": "https://localhost:7089",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Production"

}

**BIBLIOTECA MUDBLAZOR - APLICAÇÃO DE LAYOUTS PRONTOS**

Até então foi criada apenas uma página que mostra o nome de todos artistas cadastrados na base de dados, mas sem nenhum estilo. Para as demais páginas, e também em substituição a essa, será utilizada a biblioteca MudBlazor, que traz diversos layouts prontos para aplicar nas páginas.

**Biblioteca necessária: MudBlazor - v.6.11.1**

Imports necessários

Em \_Imports.razor

@using MudBlazor

@using MudBlazor.Utilities

Cadastrar o serviço MudBlazor em program.cs

builder.Services.AddMudServices();

Trocar em wwwroot/Index.html

<link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />

Por

<link href="\_content/MudBlazor/MudBlazor.min.css" rel="stylesheet" />

E adicionar abaixo da linha 29

<script src="\_content/MudBlazor/MudBlazor.min.js"></script>

Em Layout/MainLayout.razor

@inherits LayoutComponentBase

<**MudThemeProvider** />

<**MudDialogProvider** />

<**MudSnackbarProvider** />

<**MudLayout**>

<**MudAppBar** **Color**="Color.Surface" **Fixed**="true" **Elevation**="2">

<**MudImage** **Src**="images/screensound-logo.png"></**MudImage**>

</**MudAppBar**>

<**MudDrawer** **Open**="true" **ClipMode**="DrawerClipMode.Always" **Elevation**="2">

<**NavMenu**></**NavMenu**>

</**MudDrawer**>

<**MudMainContent**>

@Body

</**MudMainContent**>

</**MudLayout**>

Em wwwroot, adicionar a pasta images, com as imagens de favicon e da logo (arquivos no projeto do curso)

Em Layout/NavMenu - configurar o menu lateral

﻿﻿<**MudNavMenu** **Class**="mud-width-full mt-4">

<**MudNavLink** **Href**="/" **Icon**="@Icons.Material.Filled.Home">Home</**MudNavLink**>

<**MudNavGroup** **Title**="Artistas" **Icon**="@Icons.Material.Filled.People" **Expanded**="true">

<**MudNavLink** **Href**="/Artistas">Exibir</**MudNavLink**>

<**MudNavLink** **Href**="/CadastrarArtista">Cadastrar</**MudNavLink**>

</**MudNavGroup**>

<**MudNavGroup** **Title**="Músicas" **Icon**="@Icons.Material.Filled.QueueMusic" **Expanded**="true">

<**MudNavLink** **Href**="/MusicasPorArtista">Músicas por artista</**MudNavLink**>

<**MudNavLink** **Href**="/MusicasPorGenero">Músicas por gênero</**MudNavLink**>

<**MudNavLink** **Href**="/CadastrarMusica">Cadastrar</**MudNavLink**>

</**MudNavGroup**>

</**MudNavMenu**>

Assim, a aplicação tem um layout mais amigável, utilizando componentes prontos do MudBlazor.

**MUDANÇAS NO TEMA DO MUDBLAZOR - MÓDULO 2, AULA 4**

Ainda neste componente, certifique-se que a sessão de código esteja da seguinte maneira:

@code {

private MudTheme currentTheme = new MudTheme()

{

Palette = ScreendsoundPallete.CreatePallete

};

}

**CRIAÇÃO DE UM COMPONENTE CardArtista E SEU USO NA PÁGINA Artistas**

Para melhorar a estética de apresentação de cada artista, na página de listagem dos artistas, será usado um componente CardArtista

Adicionar a pasta Componentes ao projeto ScreenSound.Web, com o comonente razor

<**MudCard**>

<**MudCardMedia** **Image**="images/cardArtista.png"></**MudCardMedia**>

<**MudCardHeader**>

<**CardHeaderContent**>

<**MudText** **Typo**="Typo.h6">@Artista!.Nome</**MudText**>

</**CardHeaderContent**>

</**MudCardHeader**>

<**MudCardContent**>

<**MudText** **Typo**="Typo.body2">@Artista!.Bio</**MudText**>

</**MudCardContent**>

</**MudCard**>

@code {

[Parameter]

public ArtistaResponse? Artista{ get; set; }

/\*

NESSE COMPONENTE, O ATRIBUTO Artista É DEFINIDO COMO UM PARÂMETRO

PQ ELE É USADO EM Pages/Artistas.razor ASSUMINDO PARA CADA ITERAÇÃO

DO LAÇO UM VALOR DIFERENTE NO COMPONENTE MudItem

CardArtista.razor - COMPONENTE COM O PARÂMETRO Artista

Artistas.razor - PÁGINA RAZOR QUE USA O PARÂMETRO Artista DENTRO DE

SEU COMPONENTE CardArtista EM MudItem

\*/

}

E as mudanças na página Artistas

@page "/Artistas"

@inject ArtistaAPI artistaAPI

<h3>Artistas</h3>

@if(artistas is not null)

{

<**MudGrid**>

@foreach(var artista in artistas)

{

<**MudItem** **xs**="3">

<**CardArtista** **Artista**="artista"/>

</**MudItem**>

}

</**MudGrid**>

}

@code {

//ATRIBUTOS - CAMPOS

private ICollection<ArtistaResponse>? artistas;

//DEMAIS MÉTODOS

//SOBRECARGA DO MÉTO OnInitializedAsync PARA QUE, SEMPRE QUE A ROTA Artistas

//FOR CONSULTADA, A COLEÇÃO artistas VAI RECEBER O RESULTADO DE

// artistaAPI.GetArtistasAsync(), QUE FOI DEFINIDA EM ArtistaAPI.cs

protected override async Task OnInitializedAsync()

{

artistas = await artistaAPI.GetArtistasAsync();

}

//OU SEJA, SEMPRE QUE ACESSADO A ROTA Artistas

//É MOSTRADO NA TELA A LISTAGEM DE ARTISTAS - MÉTODO app.MapGet("Artistas") DA API

//<MudItem xs="3"> - DEFINE QUE PAR CADA LINHA, HAVERÁ 3 ESPAÇOS ENTRE, OS ITENS,

//OU SEJA, 4 ITENS

//<CardArtista Artista="artista"/> - O ITEM CardArtista RECEBE A VARIÁVEL

//artista, USANDO PARÂMETRO Artista EM CADA REPETIÇÃO

}

**CADASTRO DE ARTISTA NOVO PELO USUÁRIO**

Utilizar na página CadastrarArtista um MudForm, que liga os campos do formulário às variáveis nome e biografia, no trecho code.

@page "/CadastrarArtista"

@inject ArtistaAPI artistasAPI

@inject NavigationManager navigationManager

<**MudPaper** **Class**="px-8 pt-2 pb-4 mx-12 my-8" Justify="Justify.Center">

<**MudText** **Class**="mt-8" **Typo**="Typo.h4">Cadastro do Artista</**MudText**>

<**MudForm**>

<**MudTextField** **Class**="mt-4" **T**="string" **Placeholder**="Nome do Artista"

**Variant**="Variant.Outlined"

@bind-Value="nome"

**Required**="true"

**RequiredError**="Campo obrigatório." />

<**MudTextField** **Class**="mt-4" **T**="string" **Placeholder**="Biografia do artista"

**Variant**="Variant.Outlined"

@bind-Value="biografia"

**Lines**="4"

**Required**="true"

**RequiredError**="Campo obrigatório." />

<div class="d-flex align-center justify-space-between mt-4">

<**MudButton** **Variant**="Variant.Filled"

@onclick="Cadastrar"

**Color**="Color.Primary"

**Class**="ml-auto">

Cadastrar

</**MudButton**>

</div>

</**MudForm**>

</**MudPaper**>

@code {

private string? nome;

private string? biografia;

private async Task Cadastrar()

{

var request = new ArtistaRequest(nome!, biografia!);

await artistasAPI.AddArtistaAsync(request);

navigationManager.NavigateTo("/Artistas");

}

/\*

@bind-Value="nome" - RELACIONA O CAMPO COM A VARIÁVEL nome

@onclick="Cadastrar" - RELACIONA O EVENTO DE CLICAR COM A FUNÇÃO Cadastrar

SENDO QUE ESSA FUNÇÃO INSTANCIA UM OBJETO ArtistaRequest e CADASTRA ELE

NA BASE DE DADOS DA API, POR MEIO DO SERVIÇO artistasAPI

\*/

}

Pelo que foi determinado, ao clicar em Cadastrar, o método Cadastar() é chamado, usando a função AddArtistaAsync, que é definida em Servicos/ArtistaAPI, fazendo assim o cadastro na base de dados.

//MÉTODO QUE FAZ UM CADASTRO DE ARTISTA NA API A PARTIR DE UM OBJETO

//ArtistaRequest

public async Task AddArtistaAsync(ArtistaRequest artista)

{

await \_httpClient.PostAsJsonAsync("artistas", artista);

}

Ao final do formulário, é usado um objeto NavigationManager para redirecionar a aplicação para a página de listagem de clientes, já atualizada.

navigationManager.NavigateTo("/Artistas");

**EXCLUSÃO DE ARTISTA NO CARD**

Buscando ter uma página que mostre as informações detalhadas de cada artista para exclusão, cria-se um botão Detalhes, em cada card, isso é feito em Componentes/CardArtista

<**MudCardActions**>

<**MudButton** **Color**="Color.Warning"

**Variant**="Variant.Outlined"

**Href**=@($"/EditarArtista/{Artista!.Nome}")>

Detalhes

</**MudButton**>

</**MudCardActions**>

Sendo definido que esse botão detalhes leva para a página EditarArtista/{Artista!.Nome}, Temos que essa rota já é acessada com o parâmetro Artista.Nome na sua URL, e vamos usar esse parâmetro na página para pesquisar na API pelo Artista e já carregar a página com as informações preenchidas.

Pages/EditarArtista

@page "/EditarArtista/{NomeArtista}"

@inject ArtistaAPI artistasAPI

@inject NavigationManager navigationManager

<**MudPaper** **Class**="px-8 pt-2 pb-4 mx-12 my-8" Justify="Justify.Center">

<**MudText** **Class**="mt-8" **Typo**="Typo.h4">Detalhes do Artista</**MudText**>

<**MudForm**>

<**MudTextField** **Class**="mt-4" **T**="string" **Placeholder**="Nome do Artista"

**Variant**="Variant.Outlined"

@bind-Value="nome"

**Required**="true"

**RequiredError**="Campo obrigatório." />

<**MudTextField** **Class**="mt-4" **T**="string" **Placeholder**="Biografia do artista"

**Variant**="Variant.Outlined"

@bind-Value="biografia"

**Lines**="4"

**Required**="true"

**RequiredError**="Campo obrigatório." />

<div class="d-flex align-center justify-space-between mt-4">

<**MudButton** **Variant**="Variant.Filled"

@onclick="Deletar"

**Color**="Color.Secondary"

**Class**="ml-auto">

Deletar

</**MudButton**>

<**MudButton** **Variant**="Variant.Filled"

@onclick="Voltar"

**Color**="Color.Default"

**class**="ml-auto">

Voltar

</**MudButton**>

</div>

</**MudForm**>

</**MudPaper**>

@code {

private string? nome;

private string? biografia;

[Parameter]

public string? NomeArtista{ get; set; }//VEM DA URL

//OBJETO USADO PARA CONSULTAR NA API O ARTISTA DE NOME NomeArtista

public ArtistaResponse Artista { get; set; }

//AO CARREGAR A PÁGINA, OS CAMPOS DE ID nome e biografia

//SÃO PREENCHIDOS COM AS INFORMAÇÕES CARREGADAS DA API

protected override async Task OnInitializedAsync()

{

Artista = await artistasAPI.GetArtistaPorNomeAsync(NomeArtista!);

//CAMPOS DO FORMULÁRIO RECEBEM AS INFORMAÇÕES DA API

nome = Artista!.Nome;

biografia = Artista!.Bio;

}

private async Task Deletar()

{

await artistasAPI.DeleteArtistaAsync(Artista!.Id);

navigationManager.NavigateTo("/Artistas");

}

private void Voltar()

{

navigationManager.NavigateTo("..", true);//PÁGINA ANTERIOR

}

/\*

OBSERVAÇÕES

PARÂMETRO NA URL É RENOMEADO PARA NomeArtista

@page "/EditarArtista/{NomeArtista}"

E UMA VARIÁVEL DE MESMO NOME ASSUME SEU VALOR (POR TER O MESMO NOME)

[Parameter]

public string? NomeArtista{ get; set; }//VEM DA URL

-------------

AO CARREGAR A PÁGINA, OS CAMPOS DE ID nome e biografia

SÃO PREENCHIDOS COM AS INFORMAÇÕES CARREGADAS DA API

public ArtistaResponse Artista { get; set; }

protected override async Task OnInitializedAsync()

{

Artista = await artistasAPI.GetArtistaPorNomeAsync(NomeArtista!);

//CAMPOS DO FORMULÁRIO RECEBEM AS INFORMAÇÕES DA API

nome = Artista!.Nome;

biografia = Artista!.Bio;

}

\*/

}

Importante destacar dois aspectos:

**-O nome da página razor é “EditarArtista”, mas não faz edição, somente mostra os detalhes e dá a opção de deletar.**

**-Se for tentado deletar uma banda/artista que tem músicas relacionadas a ela, o entity framework impede por “restrição de chave estrangeira” - *como definir essa política de exclusão? Cascade?***

*Erro ao tentar deletar uma banda/artista que está ligada a alguma música*

***Microsoft.EntityFrameworkCore.DbUpdateException:*** *'An error occurred while saving the entity changes. See the inner exception for details.'*

*SqlException: The DELETE statement conflicted with the REFERENCE constraint "FK\_Musicas\_Artistas\_ArtistaId". The conflict occurred in database "ScreenSoundV0", table "dbo.Musicas", column 'ArtistaId'.*

**PAGINAÇÃO DA LISTAGEM DE ARTISTAS**

Para fazer a lógica de paginação, será necessário usar componentes MudStack e MudPagination em Pages/Artistas, da seguinte forma:

@page "/Artistas"

@inject ArtistaAPI artistaAPI

<**MudStack** **Class**="mt-4 mb-4" **Row**="true" **Justify**="Justify.SpaceBetween">

<**MudText** **Class**="mb-4" **Typo**="Typo.h4">

Artistas cadastrados

</**MudText**>

<**MudPagination** **Count**="@totalPaginas" **SelectedChanged**="PaginaMudou">

</**MudPagination**>

</**MudStack**>

@if (artistasPorPagina is not null)

{

<**MudGrid**>

@foreach (var artista in artistasPorPagina)

{

<**MudItem** **xs**="3">

<**CardArtista** **Artista**="artista"/>

</**MudItem**>

}

</**MudGrid**>

}

@code {

//ATRIBUTOS - CAMPOS

private ICollection<ArtistaResponse>? artistas;

//PARA PAGINAÇÃO

private int tamanho = 8;//ITENS POR PÁGINA

private int totalItens;

private int totalPaginas;

private ICollection<ArtistaResponse>? artistasPorPagina;

//DEMAIS MÉTODOS

//SOBRECARGA DO MÉTODO OnInitializedAsync PARA QUE, SEMPRE QUE A ROTA Artistas

//FOR CONSULTADA, A COLEÇÃO artistas VAI RECEBER O RESULTADO DE

// artistaAPI.GetArtistasAsync(), QUE FOI DEFINIDA EM ArtistaAPI.cs

//PAGINADO

protected override async Task OnInitializedAsync()

{

artistas = await artistaAPI.GetArtistasAsync();

//PARA PAGINAÇÃO

if(artistas is not null)

{

//ORDENAÇÃO PELOS MAIS RECENTES

artistas = artistas.OrderByDescending(a => a.Id).ToList();

totalItens = artistas.Count();

totalPaginas = Convert.ToInt32(Math.Ceiling((totalItens \* 1.0)/tamanho));

this.PaginaMudou(1);

}

}

private void PaginaMudou(int pageNumber)

{

var indice = pageNumber - 1;

artistasPorPagina = artistas!

.Skip(tamanho \* indice)

.Take(tamanho)

.ToList();

}

/\*

OBSERVAÇÕES

OU SEJA, SEMPRE QUE ACESSADO A ROTA Artistas

É MOSTRADO NA TELA A LISTAGEM DE ARTISTAS - MÉTODO app.MapGet("Artistas") DA API

<MudItem xs="3"> - DEFINE QUE PAR CADA LINHA, HAVERÁ 3 ESPAÇOS ENTRE, OS ITENS,

OU SEJA, 4 ITENS

<CardArtista Artista="artista"/> - O ITEM CardArtista RECEBE A VARIÁVEL

artista, USANDO PARÂMETRO Artista EM CADA REPETIÇÃO

\*/

}

EDIÇÃO DE ARTISTA - NO CARD

Se aproveitando da página que apresenta todas as informações do artista num formulário editável, podemos criar um botão editar, que vai construir um objeto ArtistaRequestEdit com as informações da página e por meio da API, atualizar as informações daquele ertista que está sendo consultado

EditarArtista.razor

.

.

<**MudButton** **Variant**="Variant.Filled"

@onclick="Editar"

**Color**="Color.Secondary"

**Class**="ml-auto">

Salvar

</**MudButton**>

.

.

private async Task Editar()

{

var requestEdit = new ArtistaRequestEdit(Artista!.Id, nome!, biografia!);

await artistasAPI.UpdateArtistaAsync(requestEdit);

navigationManager.NavigateTo("/Artistas");

}

Services/ArtistaApi

.

.

public async Task UpdateArtistaAsync (ArtistaRequestEdit artista)

{

await \_httpClient

.PutAsJsonAsync($"artistas", artista);

}

.

.

CADASTRO DE ARTISTA COM FOTO

Primeiro, vamos editar o formulário de cadastro de artistas, para que ele tenha um botão de upload de imagens, e um componente MudImage, que já mostra uma prévia dessa imagem

CadastrarArtista.razor

.

.

<**MudImage** **Class**="mt-4" **src**="@fileImage"/>

<**MudFileUpload** **T**="IBrowserFile" **Accept**=".jpeg" **FilesChanged**="UploadFile">

<**ButtonTemplate**>

<**MudButton** **HtmlTag**="label"

**Variant**="Variant.Filled"

**Color**="Color.Primary"

**StartIcon**="@Icons.Material.Filled.PhotoCamera"

for="@context">

Foto de Perfil

</**MudButton**>

</**ButtonTemplate**>

</**MudFileUpload**>

E no trecho @code, a função UploadFile

.

.

private async Task UploadFile(IBrowserFile file)

{

long maxFileSize = 1024 \* 1024 \* 15;//ARQUIVO DE NO MÁXIMO 15MB

var format = "image/jpeg";

//IMAGEM QUE APARECE NO FORMULÁRIO - FORMATO JPEG E TAMANHO MENOR

var resizedImage = await file.RequestImageFileAsync(format, 200, 200);

//FileStream e MemoryStream PARA SALVAR A IMAGEM COMO UM JPEG EM BASE64 (STRING)

using var fileStream = resizedImage.OpenReadStream();

using var memoryStream = new MemoryStream();

await fileStream.CopyToAsync(memoryStream);

fileImage = $"data:{format};base64,{Convert.ToBase64String(memoryStream.ToArray())}";

}