**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**