Instructions to implement DsK.AuthServer in a new Blazor Webassembly/WebApi project

Contents

[Requirements 2](#_Toc158667324)

[Create Blazor WebAssembly Standalone App Project 2](#_Toc158667325)

[Create ASP.NET Core Web API Project 4](#_Toc158667326)

[Create Class Library Project 6](#_Toc158667327)

[Install Required Nuget Packages 8](#_Toc158667328)

[WebAssembly Project 8](#_Toc158667329)

# Requirements

DsK.AuthServer is used and tested in .Net 8. The following structure needs to be in place to implement DsK.AuthServer in a new .Net 8 BlazorWebAssembly and WebAPI projects.

Visual Studio 2022

Microsoft SQL Server 2019 or higher.

Microsoft SQL Server Management Studio

## Create Blazor WebAssembly Standalone App Project

A screenshot of a phone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Create ASP.NET Core Web API Project

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Create Class Library Project

A screenshot of a phone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Install Required Nuget Packages

## WebAssembly Project

* Blazored.LocalStorage --version 4.4.0
* Microsoft.AspNetCore.Components.Authorization --version 8.0.1

## ASP.NET Core Web API Project

* Microsoft.AspNetCore.Authentication.JwtBearer --version 8.0.1

# Create classes

## Shared class project

Create the following classes in its own file.

public class TokenModel

{

public string Token { get; set; }

public string RefreshToken { get; set; }

public TokenModel(string token, string refreshToken)

{

Token = token;

RefreshToken = refreshToken;

}

}

public class TokenSettingsModel

{

public string? Issuer { get; set; }

public string? Audience { get; set; }

public string? Key { get; set; }

}

public class ValidateLoginTokenDto

{

public string LoginToken { get; set; } = string.Empty;

public string TokenKey { get; set; } = string.Empty;

}

## ASP.NET Core Web API Project

Create folder called HttpClients

Create class AuthorizarionServerAPIHttpClient

public class AuthorizarionServerAPIHttpClient{

public AuthorizarionServerAPIHttpClient(HttpClient client)

{

Client = client;

}

public HttpClient Client { get; }

}

Create SecurityController in Controller folder

[ApiController]

[Route("[controller]")]

public class SecurityController : ControllerBase

{

HttpClient \_Http;

private readonly TokenSettingsModel \_tokenSettings;

public SecurityController(AuthorizarionServerAPIHttpClient authorizarionServerAPIHttpClient, IOptions<TokenSettingsModel> tokenSettings)

{

\_Http = authorizarionServerAPIHttpClient.Client;

\_tokenSettings = tokenSettings.Value;

}

[HttpPost]

[Route("ValidateLoginToken")]

public async Task<IActionResult> ValidateLoginToken(ValidateLoginTokenDto model)

{

//todo : fix this line

model.TokenKey = \_tokenSettings.Key;

var response = await \_Http.PostAsJsonAsync($"https://localhost:7045/api/authentication/ValidateLoginToken", model);

if (!response.IsSuccessStatusCode) return NotFound();

var result = await response.Content.ReadFromJsonAsync<TokenModel>();

if (result == null) return NotFound();

return Ok(result);

}

}

In the SecurityController class, add reference and *using* statement to Shared Class Project. Also add using statement to Server.HttpClients.

Modify the line <https://localhost:7045/api/authentication/ValidateLoginToken> to the correct hostname or IP where the DsK.AuthServer is located.

Add the following lines of code in the Program.cs

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddScoped<AuthorizarionServerAPIHttpClient>();

builder.Services.AddHttpClient<AuthorizarionServerAPIHttpClient>("AuthorizarionServerAPI", c =>

{

c.BaseAddress = new System.Uri("https://localhost:7045");

});

builder.Services.Configure<TokenSettingsModel>(builder.Configuration.GetSection("TokenSettings"));

var IssuerSigningKey = builder.Configuration.GetSection("TokenSettings").GetValue<string>("Key") ?? "";

if (IssuerSigningKey == "")

{

return; //Exit app if IssuerSigningKey is not found

}

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidIssuer = builder.Configuration.GetSection("TokenSettings").GetValue<string>("Issuer"),

ValidateIssuer = true,

ValidAudience = builder.Configuration.GetSection("TokenSettings").GetValue<string>("Audience"),

ValidateAudience = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(IssuerSigningKey)),

ValidateIssuerSigningKey = true,

ValidateLifetime = true,

};

});

Add using statement to Server.HttpClients.

Add using statement to Shared Project.

Add using statement “using Microsoft.AspNetCore.Authentication.JwtBearer;”

Add using statement “using Microsoft.IdentityModel.Tokens; ”

Add using statement “using System.Text; ”

Modify the line https://localhost:7045/ to the correct hostname or IP where the DsK.AuthServer is located.

## WebAssembly Project

Add reference to Shared Class Project.

Create a folder called Services

Create Class TokenHelpers

public static class TokenHelpers

{

public static bool IsTokenExpired(string token)

{

List<Claim> claims = ParseClaimsFromJwt(token).ToList();

if (claims?.Count == 0)

return true;

string expirationSeconds = claims.Where(\_ => \_.Type.ToLower() == "exp").Select(\_ => \_.Value).FirstOrDefault();

if (string.IsNullOrEmpty(expirationSeconds))

return true;

var expirationDate = DateTimeOffset.FromUnixTimeSeconds(Convert.ToInt64(expirationSeconds));

if (expirationDate < DateTime.UtcNow)

return true;

return false;

}

public static IEnumerable<Claim> ParseClaimsFromJwt(string jwt)

{

var claims = new List<Claim>();

var payload = jwt.Split('.')[1];

var jsonBytes = ParseBase64WithoutPadding(payload);

var keyValuePairs = JsonSerializer.Deserialize<Dictionary<string, object>>(jsonBytes);

if (keyValuePairs != null)

{

claims.AddRange(keyValuePairs.Select(kvp => new Claim(kvp.Key, kvp.Value.ToString() ?? "")));

}

return claims;

}

private static byte[] ParseBase64WithoutPadding(string base64)

{

switch (base64.Length % 4)

{

case 2: base64 += "=="; break;

case 3: base64 += "="; break;

}

return Convert.FromBase64String(base64);

}

}

Create Class SecurityServiceClient

Create Class CustomAuthenticationStateProvider

public class CustomAuthenticationStateProvider : AuthenticationStateProvider

{

private readonly ILocalStorageService \_localStorageService;

public CustomAuthenticationStateProvider(ILocalStorageService localStorageService,

HttpClient httpClient)

{

\_localStorageService = localStorageService;

}

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

string token = await \_localStorageService.GetItemAsync<string>("token");

if (string.IsNullOrEmpty(token) || TokenHelpers.IsTokenExpired(token))

{

var anonymous = new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity() { }));

return anonymous;

}

var userClaimPrincipal = new ClaimsPrincipal(new ClaimsIdentity(TokenHelpers.ParseClaimsFromJwt(token), "jwt"));

var loginUser = new AuthenticationState(userClaimPrincipal);

return loginUser;

}

public void Notify()

{

NotifyAuthenticationStateChanged(GetAuthenticationStateAsync());

}

}

Add using statement “using Microsoft.AspNetCore.Components.Authorization;”

Add using statement “using Blazored.LocalStorage;”

Add using statement “using System.Security.Claims;”

Add this code to the Program.cs

//Add Authorization Core - To be able to use [CascadingAuthenticationState, AuthorizeRouteView, Authorizing], [AuthorizeView, NotAuthorized, Authorized], @attribute [Authorize]

builder.Services.AddAuthorizationCore();

//The CustomAuthenticationStateProvider is to be able to use tokens as the mode of authentication.

builder.Services.AddScoped<AuthenticationStateProvider, CustomAuthenticationStateProvider>();

builder.Services.AddScoped<SecurityServiceClient>();

/\* ---Manages saving to local storage--- \*/

builder.Services.AddBlazoredLocalStorage();

await builder.Build().RunAsync();

}

Add using statement “using Microsoft.AspNetCore.Components.Authorization;”

Add using statement “using Blazored.LocalStorage;”

Add using statement Client.Services

Edit \_Imports.razor and add:

@using Blazored.LocalStorage;

@using Microsoft.AspNetCore.Components.Authorization;

@inject NavigationManager \_navigationManager

Edit App.Razor and Encapsulate all with the tag <**CascadingAuthenticationState**> and change the RouteView tag to **AuthorizeRouteView**.

## Create pages

Create Login.razor

@page "/login"

@inject HttpClient Http;

<**PageTitle**>Login</**PageTitle**>

<h1>Login</h1>

<button @onclick="LoginLocal">Click here to login locally</button>

@code {

public async Task LoginLocal()

{

string loginurl = "https://localhost:7190/login/9EBA0CCD-FF5B-42AB-B6FB-861D18BD68D3";

\_navigationManager.NavigateTo(loginurl);

}

}

Change the URL to the correct one and change the API ID to the correct one for you application.

Create Callback.razor

@page "/callback/{LoginToken}"

<**PageTitle**>Validating Login</**PageTitle**>

<h1>Validating Login</h1>

@code {

[Parameter] public string LoginToken { get; set; }

protected override async Task OnInitializedAsync()

{

if (LoginToken != null)

{

var model = new ValidateLoginTokenDto() { LoginToken = LoginToken };

var response = await Http.PostAsJsonAsync("https://localhost:7298/Security/ValidateLoginToken", model);

if (!response.IsSuccessStatusCode)

\_navigationManager.NavigateTo("/noaccess");

var result = await response.Content.ReadFromJsonAsync<TokenModel>();

if (result == null)

\_navigationManager.NavigateTo("/noaccess");

await \_localStorageService.SetItemAsync("token", result.Token);

await \_localStorageService.SetItemAsync("refreshToken", result.RefreshToken);

(\_authenticationStateProvider as CustomAuthenticationStateProvider).Notify();

\_navigationManager.NavigateTo("/");

}

}

}