Instructions to implement DsK.AuthServer in a new or existing Blazor WebAssembly/Web API project

Contents

[Introduction 2](#_Toc158724891)

[Configure App in the DsK.AuthServer GUI 2](#_Toc158724892)

[New Projects 5](#_Toc158724893)

[Create Blazor WebAssembly Standalone App Project 5](#_Toc158724894)

[Create ASP.NET Core Web API Project 7](#_Toc158724895)

[Create Class Library Project 9](#_Toc158724896)

[Install Required Nuget Packages 11](#_Toc158724897)

[WebAssembly Project 11](#_Toc158724898)

[ASP.NET Core Web API Project 11](#_Toc158724899)

[Create supporting classes 11](#_Toc158724900)

[Shared class project 11](#_Toc158724901)

[ASP.NET Core Web API Project 12](#_Toc158724902)

[WebAssembly Project 14](#_Toc158724903)

[Create pages 18](#_Toc158724904)

[Implementation 19](#_Toc158724905)

[Per page 19](#_Toc158724906)

# Introduction

DsK.AuthServer is an authentication and authorization server that can be easily integrated into new or already existing Blazor WebAssembly / Web API projects.

It has the built-in capacity to have local users mixed with OnPremise Active Directory. With some modifications, other Authentication Providers can be added.

The following structure needs to be in place to implement DsK.AuthServer in Blazor WebAssembly and Web API projects.

# Configure App in the DsK.AuthServer GUI

Go to Applications and click the CREATE APPLICATION button.

A screenshot of a computer

Description automatically generated

Enter the application name and Callback URL.

A screenshot of a computer

Description automatically generated

The Callback URL can be found in the launchSettings.json of the Blazor WebAssembly project.

A screen shot of a computer code

Description automatically generated

Enable the Admin User.

A close up of a message

Description automatically generated

Create the Application Permissions.

A search box with words

Description automatically generated

For this example, we are creating a permission that is going to be used in the Counter.page of the Template Blazor WebAssembly project.

A screenshot of a computer application

Description automatically generated

Every application is created with a disabled Local Authentication Provider. Edit the Local Authentication Provider and enable it.

A screenshot of a search engine

Description automatically generated

A screenshot of a computer

Description automatically generated

# New Projects

If DsK.AuthServer is going to be implemented in a new project, these are the templates that were used for this test.

## 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

These packages need to be installed in their corresponding projects.

## 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 supporting 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;

}

public static class Access

{

public const string Admin = "Admin";

[DisplayName("Counter")]

[Description("Counter Permissions")]

public static class CounterPage

{

public const string CounterFunction = "TestApp2.Counter";

}

/// <summary>

/// Returns a list of Permissions.

/// </summary>

/// <returns></returns>

public static List<string> GetRegisteredPermissions()

{

var permissions = new List<string>();

foreach (var prop in typeof(Access).GetNestedTypes().SelectMany(c => c.GetFields(BindingFlags.Public | BindingFlags.Static | BindingFlags.FlattenHierarchy)))

{

var propertyValue = prop.GetValue(null);

if (propertyValue is not null)

permissions.Add(propertyValue.ToString());

}

return permissions;

}

}

## 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,

};

});

The following using statements may be added automatically to the Program.cs when you copy the code, if not, add them manually.

* using DsK.AuthServer.TestApp.Server.HttpClients;
* using DsK.AuthServer.TestApp.Shared;
* using Microsoft.AspNetCore.Authentication.JwtBearer;
* using Microsoft.IdentityModel.Tokens;
* using System.Text;

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

Modify your appsettins.json after the allowed hosts line. Enter the Application Id in the audience value and the API Key in the Key value.

"AllowedHosts": "\*",

"TokenSettings": {

"Issuer": "DsK.AuthServer",

"Audience": "",

"Key": ""

}

## 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

public partial class SecurityServiceClient

{

private readonly ILocalStorageService \_localStorageService;

private readonly HttpClient \_httpClient;

private readonly AuthenticationStateProvider \_authenticationStateProvider;

public SecurityServiceClient(ILocalStorageService localStorageService,

HttpClient httpClient, AuthenticationStateProvider authenticationStateProvider)

{

\_localStorageService = localStorageService;

\_httpClient = httpClient;

\_authenticationStateProvider = authenticationStateProvider;

}

private async Task PrepareBearerToken()

{

var token = await GetTokenAsync();

\_httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer", token);

}

public async Task<string> GetTokenAsync()

{

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

if (string.IsNullOrEmpty(token))

return string.Empty;

if (TokenHelpers.IsTokenExpired(token))

token = await TryRefreshToken();

return token;

}

private async Task<string> TryRefreshToken()

{

//TODO : FIX

return "";

}

public bool HasPermission(ClaimsPrincipal user, string permission)

{

//ClaimsPrincipal newuser = user;

// && (x.Value.Contains(permission)|| x.Value.Contains("Admin")

var permissions = user.Claims.Where(x => x.Type == ClaimTypes.Role).FirstOrDefault();

if (UserHasPermission(permissions, permission) || UserHasPermission(permissions, "Admin"))

return true;

else

return false;

}

public int GetUserId(ClaimsPrincipal user)

{

string userId = user.Claims.Where(\_ => \_.Type == "UserId").Select(\_ => \_.Value).FirstOrDefault();

int userIdParsed = 0;

int.TryParse(userId, out userIdParsed);

return userIdParsed;

}

private bool UserHasPermission(Claim permissions, string permission)

{

if (permissions != null)

{

var schema = "http://schemas.microsoft.com/ws/2008/06/identity/claims/role : ";

var parsedPermissions = permissions.Value.ToString().Replace(schema, "").Trim().TrimStart('[').TrimEnd(']').Replace("\"", "").Split(',');

foreach (var parsedPermission in parsedPermissions)

{

if (parsedPermission == permission) return true;

}

}

return false;

}

}

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());

}

}

The following using statements may be added automatically to the CustomAuthenticationStateProvider.cs when you copy the code, if not, add them manually.

* using Blazored.LocalStorage;
* using Microsoft.AspNetCore.Components.Authorization;
* using System.Security.Claims;

Add this code to the Program.cs before the “await builder.Build().RunAsync();”

//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();

The following using statements may be added automatically to the Add this code to the Program.cs when you copy the code, if not, add them manually.

* using Blazored.LocalStorage;
* using DsK.AuthServer.TestApp.Client.Services;
* using Microsoft.AspNetCore.Components.Authorization;

Edit \_Imports.razor and add:

@using TestApp2.Client.Services;

@using TestApp2.Shared;

@using Blazored.LocalStorage;

@using Microsoft.AspNetCore.Components.Authorization;

@using Microsoft.AspNetCore.Authorization;

@inject NavigationManager \_navigationManager

@inject HttpClient Http;

@inject ILocalStorageService \_localStorageService;

@inject AuthenticationStateProvider \_authenticationStateProvider;

@inject SecurityServiceClient securityService;

@inject IJSRuntime JSRuntime;

Note: You may have to replace TestApp2 with the Namespace of your project.

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

# Create pages

## Login.razor

@page "/login"

<**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 GUID to the Authentication Provider GUID you want to use for your application in DsK.AuthServer.

A screenshot of a computer

Description automatically generated

## 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("/");

}

}

}

# Implementation

## NavMenu

Make the following modifications to the NavMenu.razor.

Add this code at the end.

private async Task Logout()

{

await \_localStorageService.RemoveItemAsync("token");

await \_localStorageService.RemoveItemAsync("refreshToken");

(\_authenticationStateProvider as CustomAuthenticationStateProvider).Notify();

}

public async Task MyProfile()

{

string loginurl = "https://localhost:7190/MyProfile";

//\_navigationManager.NavigateTo(loginurl);

await JSRuntime.InvokeVoidAsync("open", loginurl, "\_blank");

}

Replace the Home, Counter and Weather menu options for this:

<AuthorizeView>

<NotAuthorized>

<div class="nav-item px-3">

<NavLink class="nav-link" href="login">

<span class="oi oi-plus" aria-hidden="true"></span> Login

</NavLink>

</div>

</NotAuthorized>

<Authorized>

<div class="nav-item px-3">

<NavLink class="nav-link" @onclick="MyProfile">

<span class="oi oi-account-login" aria-hidden="true"></span> My Profile

</NavLink>

</div>

<div class="nav-item px-3">

<NavLink class="nav-link" href="counter">

<span class="oi oi-plus" aria-hidden="true"></span> Counter

</NavLink>

</div>

<div class="nav-item px-3">

<NavLink class="nav-link" href="fetchdata">

<span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data

</NavLink>

</div>

<div class="nav-item px-3">

<NavLink class="nav-link" @onclick="Logout">

<span class="oi oi-plus" aria-hidden="true"></span> Logout

</NavLink>

</div>

</Authorized>

</AuthorizeView>

## Per page

Add the “@attribute [Authorize]” after @page directive in every page you want to implement security.

Add the following to the Counter.razor Page to test.

@if (!\_Access)

{

<h1>You dont have access to this page.</h1>

}

else

{

<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

}

Add the following code to the Code Section.

[CascadingParameter] protected Task<AuthenticationState> AuthStat { get; set; }

private bool \_Access;

protected async override Task OnInitializedAsync()

{

base.OnInitialized();

var user = (await AuthStat).User;

if (!user.Identity.IsAuthenticated)

{

\_navigationManager.NavigateTo($"login?returnUrl={Uri.EscapeDataString(\_navigationManager.Uri)}");

}

else

{

\_Access = securityService.HasPermission(user, Access.CounterPage.CounterFunction);

}

}

In this example we are verifying if the user has the Counter Permission and mapping the result to the \_Access variable. Then we can use this variable to restrict page functionality.

At this point you should be able to run your program and have authentication and authorization enabled.