If you disable the launch browser the it runs behind the scene but no web page.

<<<<<need to understand this completely by myself>>>>>>>>>>>>>>>

@using Microsoft.AspNetCore.Authentication

@using Microsoft.AspNetCore.Authentication.OpenIdConnect

@{

ViewData["Title"] = "Claims";

}

<h2>Claims</h2>

<dl>

@foreach (var claim in User.Claims)

{

<dt>@claim.Type</dt>

<dd>@claim.Value</dd>

}

</dl>

<h2>Properties</h2>

<dl>

@foreach (var prop in (await Context.AuthenticateAsync(OpenIdConnectDefaults.AuthenticationScheme)).Properties.Items)

{

<dt>@prop.Key</dt>

<dd>@prop.Value</dd>

}

</dl>

#### Use Identity server authentication

.AddIdentityServerAuthentication("Bearer", options =>

{

options.Authority = "https://localhost:44378";

options.ApiName = "orders.api";

})

The above code needs to be added to the client home controller view

Logout functionality in the client project:

public async Task Logout()

{

//With an authentication service like IdentityServer, it is not enough to clear the local application cookies.

//In addition you also need to make a roundtrip to IdentityServer to clear the central single sign-on session.

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme); // this signs out just from the client

await HttpContext.SignOutAsync(OpenIdConnectDefaults.AuthenticationScheme);

//this from the auth server as well.

}

Token endpoint,

Discovery Endpoint,

End session endpoint – implemented by the Authorization server.

Introspection endpoint – can be used to check the validity of the access token.

4 links to be added to the index page and they are shown only when the user is logged in:

If user.identity.isauthenticated then only show the links.

[Authorize]

public async Task<IActionResult> Weather()

{

HttpClient client = new HttpClient();

// access the tokens in the session

string accessToken = await HttpContext.GetTokenAsync("access\_token");

//assign the access token to the "Authorization" header

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

//invoke the web api

string content = await client.GetStringAsync("https://localhost:44381/weatherForecast");

string jsonResult = JArray.Parse(content).ToString();

return Json(jsonResult);

}

Yesterday we saw the Order details on the post man. Client was postman. Today we will show via the web page -> Client page.

Also the await HttpContext.GetTokenAsync("access\_token");

Gets hold of the token named “Access token” and we are looking for the Bearer token to be specific.

In the scope of the access token, we need to specify the orders API.

In the allowed scope property of the client (the new client that you had added to the IndentityServerConfig, you need to add the AllowedScopes = {“orders.API”}

Client cannot request for more than what auth server allows access to.

Client project also should have the scope of Orders API added to it. in the startup class.

Implicit – auth flow.

In case of a SPA – then how do I store the information. In a .net code, we can save in user secrets, env variables, azure vault.

In the SPA, implicit flow because where will you store it. SPA is only on the client side. So in implicit, the token is not to be viewed. Simplified version of authorized code flow. Lesser secure version of the suthorized flow.

There is no authorization code that is returned from the auth server.

In SPAs and Javascript client.

Authorization code with PKCE – proof key for coede exchange (between auth code flow and implicit)

Meant to be used for native application (like windows application, mobile application – that are installed on the device).

Link hijacking is an issue you should be aware in the mobile application.

------------------------------------Working with Tokens

Reduce the lifetime of Access token and explicitly request for the refresh token.

Reference token is a pointer to where the access token is sitting. In this case the JWT token (access token) is not sitting on the client end. The access token is instead sitting on the auth server

Client (Access token) -> API ->

Client (Reference token) –> API -> now API is supposed to hit the introspect the Auth server to get the access token -> Auth server returns the Access token to the API.

No new code needs to be written apart from the

AccessTokenType = AccessTokenType.Reference

In the list API resources the new secret is added.

AddDefaultIdentity will help creating a view on the fly to do all the user management, if you are adding on the fly then you can’t edit the pages.

Google authentication – register your website there. Give the link to the authentication server URI.

Google nuget package. In the identity server project.

The services need to be registered in the same project.

Check external authenticator code.

There the clientid and client secret must be added.

If we don’t have the auth server then we can configure the google authorization on the clinet application itself.