[Robin van der Knaap](https://dev.to/robinvanderknaap)

[Robin van der Knaap](https://dev.to/robinvanderknaap)

Posted on Dec 11, 2020 • Updated on Dec 15, 2020

**Setting up an Authorization Server with OpenIddict - Part IV - Authorization Code Flow**

[#dotnet](https://dev.to/t/dotnet)[#oauth2](https://dev.to/t/oauth2)[#openiddict](https://dev.to/t/openiddict)[#aspnet](https://dev.to/t/aspnet)

This article is part of a series called **Setting up an Authorization Server with OpenIddict**. The articles in this series will guide you through the process of setting up an OAuth2 + OpenID Connect authorization server on the the [ASPNET Core](https://dotnet.microsoft.com/apps/aspnet) platform using [OpenIddict](https://github.com/openiddict/openiddict-core).

* [Part I: Introduction](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-i-introduction-4jid)
* [Part II: Create ASPNET project](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949)
* [Part III: Client Credentials Flow](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iii-client-credentials-flow-55lp)
* **Part IV: Authorization Code Flow**
* [Part V: OpenID Connect](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-v-openid-connect-a8j)
* [Part VI: Refresh tokens](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-vi-refresh-tokens-5669)

GitHub logo [robinvanderknaap](https://github.com/robinvanderknaap)/ [**authorization-server-openiddict**](https://github.com/robinvanderknaap/authorization-server-openiddict)

Authorization Server implemented with OpenIddict.

In this part we will implement the [Authorization Code Flow with PKCE extension](https://oauth.net/2/grant-types/authorization-code/). This flow is the recommended flow for Single Page Applications (SPA's) and native/mobile applications.

**Configure OpenIddict**

First we need to enable the Authorization Code Flow in Startup.cs:

public void ConfigureServices(IServiceCollection services)

{

...

services.AddOpenIddict()

...

.AddServer(options =>

{

options.AllowAuthorizationCodeFlow().RequireProofKeyForCodeExchange();

...

options

.SetAuthorizationEndpointUris("/connect/authorize")

.SetTokenEndpointUris("/connect/token");

...

options

.UseAspNetCore()

.EnableTokenEndpointPassthrough()

.EnableAuthorizationEndpointPassthrough();

});

...

}

The call AllowAuthorizationCodeFlow enables the flow, RequireProofKeyForCodeExchange is called directly after that, this makes sure all clients are required to use PKCE (Proof Key for Code Exchange).

The authorization code flow dictates that the user first **authorizes** the client to make requests in the user's behalf. Therefore, we need to implement an authorization endpoint which returns an **authorization code** to the client when the user allows it.

The client can exchange the authorization code for an access token by calling the token endpoint we already created for the client credentials flow.

First, we will create the authorization endpoint, the call to EnableAuthorizationEndpointPassthrough in Startup.cs allows us to implement the endpoint within a controller.

After that we will make some minor adjustments to our token endpoint to allow clients to exchange authorization codes for access tokens.

**Authorization endpoint**

We'll implement the authorization endpoint in the Authorization Controller, just like the token endpoint:

[HttpGet("~/connect/authorize")]

[HttpPost("~/connect/authorize")]

[IgnoreAntiforgeryToken]

public async Task<IActionResult> Authorize()

{

var request = HttpContext.GetOpenIddictServerRequest() ??

throw new InvalidOperationException("The OpenID Connect request cannot be retrieved.");

// Retrieve the user principal stored in the authentication cookie.

var result = await HttpContext.AuthenticateAsync(CookieAuthenticationDefaults.AuthenticationScheme);

// If the user principal can't be extracted, redirect the user to the login page.

if (!result.Succeeded)

{

return Challenge(

authenticationSchemes: CookieAuthenticationDefaults.AuthenticationScheme,

properties: new AuthenticationProperties

{

RedirectUri = Request.PathBase + Request.Path + QueryString.Create(

Request.HasFormContentType ? Request.Form.ToList() : Request.Query.ToList())

});

}

// Create a new claims principal

var claims = new List<Claim>

{

// 'subject' claim which is required

new Claim(OpenIddictConstants.Claims.Subject, result.Principal.Identity.Name),

new Claim("some claim", "some value").SetDestinations(OpenIddictConstants.Destinations.AccessToken)

};

var claimsIdentity = new ClaimsIdentity(claims, OpenIddictServerAspNetCoreDefaults.AuthenticationScheme);

var claimsPrincipal = new ClaimsPrincipal(claimsIdentity);

// Set requested scopes (this is not done automatically)

claimsPrincipal.SetScopes(request.GetScopes());

// Signing in with the OpenIddict authentiction scheme trigger OpenIddict to issue a code (which can be exchanged for an access token)

return SignIn(claimsPrincipal, OpenIddictServerAspNetCoreDefaults.AuthenticationScheme);

}

Unlike the Clients Credentials Flow, the Authorization Code Flow involves the end user for approval. Remember that we already implemented authentication in our project. So, in the authorize method we just determine if the user is already logged in, if not, we redirect the user to the login page.

When the user is authenticated, a new claims principal is created which is used to sign in with OpenIddict authentication scheme.

You can add claims to principal which will be added to the access token if the destination is set to AccessToken. The subject claim is required and is always added to the access token, you don't have to specify a destination for this claim.

The scopes requested by the client are all given with the call claimsPrincipal.SetScopes(request.GetScopes()), because we don't implement a consent screen in this example to keep things simple. When you do implement consent, this would be the place to filter the requested scopes.

The SignIn call triggers the OpenIddict middleware to send an authorization code which the client can exchange for an access token by calling the token endpoint.

We need to alter the token endpoint also since we now support the Authorization Code Flow:

[HttpPost("~/connect/token"), Produces("application/json")]

public async Task<IActionResult> Exchange()

{

...

if (request.IsClientCredentialsGrantType())

{

...

}

else if (request.IsAuthorizationCodeGrantType())

{

// Retrieve the claims principal stored in the authorization code

claimsPrincipal = (await HttpContext.AuthenticateAsync(OpenIddictServerAspNetCoreDefaults.AuthenticationScheme)).Principal;

}

else

{

throw new InvalidOperationException("The specified grant type is not supported.");

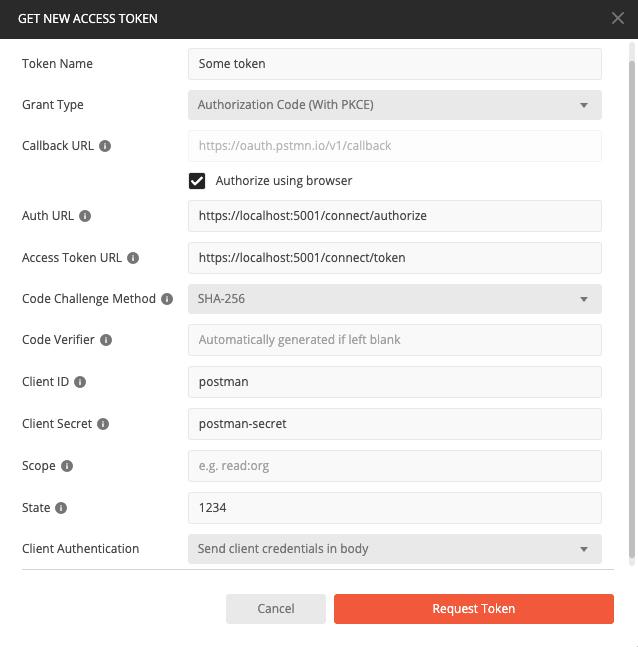
}

...

}

The claims principal we created in the authorize method is stored in the authorization code, so we only need to grab the claims principal from the request and pass it to the SignIn method. OpenIddict will respond with an access token.

Now, let's see if we can request an access token with the Authorization Code Flow using Postman:

[](https://res.cloudinary.com/practicaldev/image/fetch/s--685wAOQS--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/dev-to-uploads.s3.amazonaws.com/i/igbagmw9zi4l1bfanz2s.png)

This won't work because the client is not allowed to use the Authorization Code Flow and also we did not specify the RedirectUris of the client.

The redirect URI in the case of Postman is https://oauth.pstmn.io/v1/callback. The authorization code is sent here after successful authentication.

After updating, the Postman client in TestData.cs should look like this:

ClientId = "postman",

ClientSecret = "postman-secret",

DisplayName = "Postman",

RedirectUris = { new Uri("https://oauth.pstmn.io/v1/callback") },

Permissions =

{

OpenIddictConstants.Permissions.Endpoints.Authorization,

OpenIddictConstants.Permissions.Endpoints.Token,

OpenIddictConstants.Permissions.GrantTypes.AuthorizationCode,

OpenIddictConstants.Permissions.GrantTypes.ClientCredentials,

OpenIddictConstants.Permissions.Prefixes.Scope + "api",

OpenIddictConstants.Permissions.ResponseTypes.Code

}

If everything is working correctly, you should be able to obtain an access token with Postman after authenticating yourself.

Note. A client secret is optional when configuring a client with OpenIddict, this is useful for public clients which aren't able to securely store a client secret. When the client secret is omitted from the configuration, you can also omit it from the request.  
The PKCE-enhanced Authorization Code Flow introduces a secret created by the calling application that can be verified by the authorization server ([source](https://auth0.com/docs/flows/authorization-code-flow-with-proof-key-for-code-exchange-pkce)). However, PKCE doesn't replace client secrets. PKCE and client secrets are complementary and you should use them both when possible (typically, for server-side apps). ([Explained to me by the author of OpenIddict](https://twitter.com/kevin_chalet/status/1338130301739036673))

**Next**

Congratulations, you have implemented the Authentication Code Flow with PKCE with OpenIddict! [Next up](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-v-openid-connect-a8j), we will demonstrate how to leverage the OpenID Connect protocol.

**Discussion (7)**

Subscribe

Top of Form





Bottom of Form

[borisgr04 profile image](https://dev.to/borisgr04)

borisgr04

•[Apr 16 '21](https://dev.to/borisgr04/comment/1dg8n)

Very good solution now that identityserver is paid.  
Question. As handling of various scopes.

4 likes[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8/comments/new/1dg8n)

[zizounetgit profile image](https://dev.to/zizounetgit)

zizounetgit

•[Dec 30 '21](https://dev.to/zizounetgit/comment/1ko3b)

When i call the userinfo i got this :  
  
The userinfo request was rejected because the mandatory 'access\_token' parameter was missing.  
info: OpenIddict.Server.OpenIddictServerDispatcher[0]  
The response was successfully returned as a challenge response: {  
"error": "missing\_token",  
"error\_description": "The mandatory 'access\_token' parameter is missing.",  
"error\_uri": "[documentation.openiddict.com/error...](https://documentation.openiddict.com/errors/ID2029)"  
}.

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8/comments/new/1ko3b)

[nonsenseless profile image](https://dev.to/nonsenseless)

nonsenseless

•[Oct 26 '21](https://dev.to/nonsenseless/comment/1j7pg)

I'm still going through the series and picking out how things will apply to my own setup, but I wanted to pause and thank you for a very well put together series on authentication. This may be the single best walkthrough I've found for configuring API authentication in core.

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8/comments/new/1j7pg)

[skini82 profile image](https://dev.to/skini82)

Dario Fraschini

•[Mar 21](https://dev.to/skini82/comment/1n00b)

Hello guys!  
I followed this guide but I noticed that the PKCE flow needs a client\_secret to be accomplished. I was thinking the PKCE flow is just done to avoid exchange of client secret from a SPA to the Auth Server. Am I right? What can I do to avoid openiddict to ask for a client\_secret?

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8/comments/new/1n00b)

[hypervtechnics profile image](https://dev.to/hypervtechnics)

hypervtechnics

•[Jun 23 '21](https://dev.to/hypervtechnics/comment/1fjf5)

Very good article!

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8/comments/new/1fjf5)

[dubik profile image](https://dev.to/dubik)

dubik

•[May 8 '21](https://dev.to/dubik/comment/1e9d5)

Registered just to say thank you for such an amazing article!

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8/comments/new/1e9d5)

[robinvanderknaap profile image](https://dev.to/robinvanderknaap)

Robin van der Knaap

•[May 8 '21](https://dev.to/robinvanderknaap/comment/1e9f2)

Thank you, very nice to hear that!

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8/comments/new/1e9f2)

[Code of Conduct](https://dev.to/code-of-conduct) • [Report abuse](https://dev.to/report-abuse)