[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 II - Create ASPNET project**

[#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**
* [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](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iv-authorization-code-flow-3eh8)
* [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 create an ASPNET Core project which serves as a minimal setup for our authorization server. We will use MVC to serve pages and we will add authentication to the project, including a basic login form.

**Create a new empty ASPNET project.**

As was said in the previous article, an authorization server is just another web application. The following content will guide you through setting up an ASPNET Core application with a username-password login. I choose not to use [ASPNET Core Identity](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity) to keep things really simple. Basically every username-password combination will work.

Let's start with creating a new web application called AuthorizationServer using the [ASP.NET Core Empty template](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-new#web):

dotnet new web --name AuthorizationServer

We will work with just this project, and we will not add a solution file in this guide.

OpenIddict requires us to work with the https protocol, even when developing locally. To make sure the local certificate is trusted, you will have to run the following command:

dotnet dev-certs https --trust

On Windows the certificate will be added to the certificate store and on OSX to the keychain. On Linux there isn't a standard way across distros to trust the certificate. Read more about this subject in [Hanselman's blog article](https://www.hanselman.com/blog/developing-locally-with-aspnet-core-under-https-ssl-and-selfsigned-certs).

Start the application to see if everything works as expected

dotnet run --project AuthorizationServer

Visit [https://localhost:5001](https://localhost:5001/). You should see the Hello World! in your browser.

**MVC**

We have created a project based on the ASPNET Core Empty template. This is a very minimal template. I have done this intentionally, because I like to have as little 'noise' in my project as possible to keep things clear and simple.

Downside of using this template is we have to add [MVC](https://docs.microsoft.com/en-us/aspnet/core/mvc/overview) ourselves. First, we need to enable MVC by altering the Startup.cs class:

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

namespace AuthorizationServer

{

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

services.AddControllersWithViews();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseStaticFiles();

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapDefaultControllerRoute();

});

}

}

}

MVC is configured by calling services.AddControllersWithViews(). Endpoints are setup to use default routing. We also enabled the serving of static files, we need this to serve our style sheets out of the wwwroot folder.

Now, let's create the controllers, views and view models. Start with adding the following folder structure inside the project folder (mind the casing):

/Controllers

/Views

/Views/Home

/Views/Shared

/ViewModels

/wwwroot

/wwwroot/css

**Layout**

The first item we add is a layout file called \_Layout.cshtml to the Views/Shared folder. This file defines the general layout of the application, and also loads [Bootstrap](https://getbootstrap.com/) and jQuery from a CDN. jQuery is a dependency of Bootstrap.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8"/>

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no"/>

<title>OpenIddict - Authorization Server</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" crossorigin="anonymous">

<link rel="stylesheet" href="~/css/site.css"/>

</head>

<body>

<div class="container-sm mt-3">

<div class="row mb-3">

<div class="col text-center">

<h1>

Authorization Server

</h1>

</div>

</div>

<div class="row">

<div class="col-xs-12 col-md-8 col-xl-4 offset-md-2 offset-xl-4 text-center mb-3">

@RenderBody()

</div>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js" crossorigin="anonymous"></script>

</body>

</html>

For the layout and views to work, we need to add two files to the \Views folder:

\_ViewStart.cshtml

@{

Layout = "\_Layout";

}

\_ViewImports.cshtml

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

**Home page**

Add a basic HomeController to the /Controllers folder, which has the sole purpose of serving our home page:

using Microsoft.AspNetCore.Mvc;

namespace AuthorizationServer.Controllers

{

public class HomeController : Controller

{

public IActionResult Index()

{

return View();

}

}

}

Add Index.cshtml to the Views/Home folder, which is served by the HomeController:

<h2>MVC is working</h2>

**Style sheet**

Last but not least, we need some styling. Add a style sheet called site.css to the wwwroot\css folder:

:focus {

outline: 0 !important;

}

.input-validation-error {

border: 1px solid darkred;

}

form {

width: 100%;

}

.form-control {

border:0;

border-radius: 0;

border-bottom: 1px solid lightgray;

font-size:0.9rem;

}

.form-control:focus{

border-bottom-color: lightgray;

box-shadow: none;

}

.form-control.form-control-last {

border-bottom: 0;

}

.form-control::placeholder {

opacity: 0.6;

}

.form-control.input-validation-error {

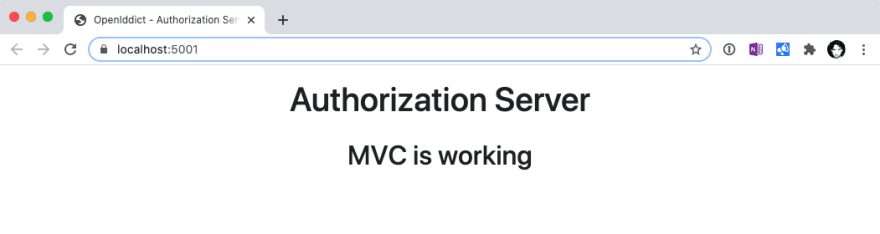
border: 1px solid darkred;

}

Some style rules are already added to the style sheet anticipating the login form we are going to create later.

If you want to use SASS, or customize Bootstrap with SASS, check [my article](https://dev.to/robinvanderknaap/setting-up-bootstrap-4-sass-with-aspnet-core-4ff6) about setting up Bootstrap SASS with ASPNET.

Let's run the application and see if everything is working, you should see something like this in your browser:

[](https://res.cloudinary.com/practicaldev/image/fetch/s--YsPPboZC--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/dev-to-uploads.s3.amazonaws.com/i/nkgftn4plz7typ4mjtf2.png)

**Enable authentication**

In ASP.NET Core, [authentication](https://docs.microsoft.com/en-us/aspnet/core/security/authentication) is handled by the IAuthenticationService. The authentication service uses authentication handlers to complete authentication-related actions.

The authentication handlers are registered during startup and their configuration options are called "schemes". Authentication schemes are specified by registering authentication services in Startup.ConfigureServices.

For this project we will use [cookie authentication](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie), so we need to register the Cookie authentication scheme in the ConfigureServices method in Startup.cs:

public void ConfigureServices(IServiceCollection services)

{

services.AddControllersWithViews();

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie(CookieAuthenticationDefaults.AuthenticationScheme, options =>

{

options.LoginPath = "/account/login";

});

}

The login path is set to /account/login, we will implement this endpoint shortly.

The authentication middleware, which uses the registered authentication schemes, is added by calling the UseAuthentication extension method on the app's IApplicationBuilder:

app.UseRouting();

app.UseAuthentication();

app.UseEndpoints(endpoints =>

{

endpoints.MapDefaultControllerRoute();

});

The call to UseAuthentication is made after the call to UseRouting, so that route information is available for authentication decisions, but before UseEndpoints, so that users are authenticated before accessing the endpoints.

**Login page**

Now that we have authentication enabled, we will need a login page to authenticate users.

First, create the Login view model containing the information we need to authenticate the user. Make sure to put this file in the ViewModels folder:

using System.ComponentModel.DataAnnotations;

namespace AuthorizationServer.ViewModels

{

public class LoginViewModel

{

[Required]

public string Username { get; set; }

[Required]

public string Password { get; set; }

public string ReturnUrl { get; set; }

}

}

Create a folder named Account in the Views folder and add the login view, Login.cshtml, containing the login form:

@model AuthorizationServer.ViewModels.LoginViewModel

<form autocomplete="off" asp-route="Login">

<input type="hidden" asp-for="ReturnUrl"/>

<div class="card">

<input type="text" class="form-control form-control-lg" placeholder="Username" asp-for="Username" autofocus>

<input type="password" class="form-control form-control-lg form-control-last" placeholder="Password" asp-for="Password">

</div>

<p>

<button type="submit" class="btn btn-dark btn-block mt-3">Login</button>

</p>

</form>

Finally we add the AccountController:

using System.Collections.Generic;

using System.Security.Claims;

using System.Threading.Tasks;

using AuthorizationServer.ViewModels;

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace AuthorizationServer.Controllers

{

public class AccountController : Controller

{

[HttpGet]

[AllowAnonymous]

public IActionResult Login(string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginViewModel model)

{

ViewData["ReturnUrl"] = model.ReturnUrl;

if (ModelState.IsValid)

{

var claims = new List<Claim>

{

new Claim(ClaimTypes.Name, model.Username)

};

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

await HttpContext.SignInAsync(new ClaimsPrincipal(claimsIdentity));

if (Url.IsLocalUrl(model.ReturnUrl))

{

return Redirect(model.ReturnUrl);

}

return RedirectToAction(nameof(HomeController.Index), "Home");

}

return View(model);

}

public async Task<IActionResult> Logout()

{

await HttpContext.SignOutAsync();

return RedirectToAction(nameof(HomeController.Index), "Home");

}

}

}

So, what happens here?  
We have two login actions (GET and POST) on the account controller, both allow anonymous requests, otherwise nobody would be able to login.

The GET action serves the login form we just created. We have an optional query parameter returlUrl which we store in ViewData, so we can use this to redirect the user after a successful login.

The POST action is more interesting. First the ModelState is validated. That means that a username and a password are required. We do not check the credentials here, any combination is valid in this example. Normally, this would be the place where you check the credentials against your database.

When the ModelState is valid, a claims identity is constructed. We add one claim, the name of the user. Beware, we specify the cookie authentication scheme (CookieAuthenticationDefaults.AuthenticationScheme) when creating the claims identity. This is basically a string, and maps to the authentication scheme we defined in the Startup.cs class when setting up the cookie authentication.

The SignInAsync method is an extension method which calls the AuthenticationService which calls the CookieAuthenticationHandler because that's the scheme we specified when creating the claims identity.

After signing in we need to redirect the user. If a return url is specified, we check if it's a local url to [prevent open redirect attacks](https://docs.microsoft.com/en-us/aspnet/core/security/preventing-open-redirects) before redirecting. Otherwise the user is redirected to the home page.

The last action, Logout, calls the authentication service to sign out the user. The authentication service will call the authentication middleware, in our case the cookie authentication middleware, to sign out the user.

**Update home page**

Update the home page (Views/Home/Index.cshtml):

@using Microsoft.AspNetCore.Authentication

@if (User.Identity.IsAuthenticated)

{

var authenticationResult = await Context.AuthenticateAsync();

var issued = authenticationResult.Properties.Items[".issued"];

var expires = authenticationResult.Properties.Items[".expires"];

<div>

<p>You are signed in as</p>

<h2>@User.Identity.Name</h2>

<hr/>

<dl>

<dt>Issued</dt>

<dd>@issued</dd>

<dt>Expires</dt>

<dd>@expires</dd>

</dl>

<hr/>

<p><a class="btn btn-dark" asp-controller="Account" asp-action="Logout">Sign out</a></p>

</div>

}

@if (!User.Identity.IsAuthenticated)

{

<div>

<p>You are not signed in</p>

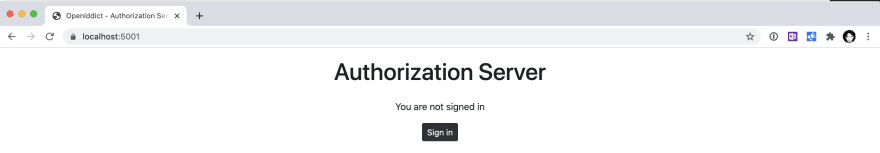
<p><a class="btn btn-sm btn-dark" asp-controller="Account" asp-action="Login">Sign in</a></p>

</div>

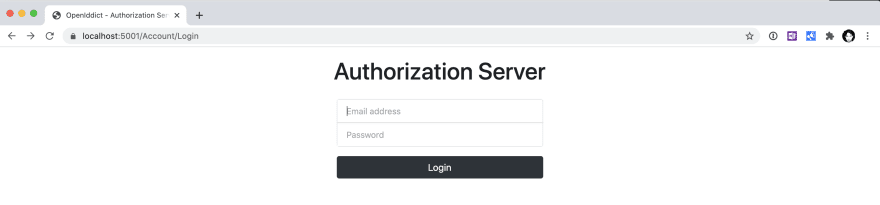
}

If the user is authenticated we display the user name, information about the current session and a sign-out button. When the user is not authenticated, we show a sign-in button which navigates the user to the login form.

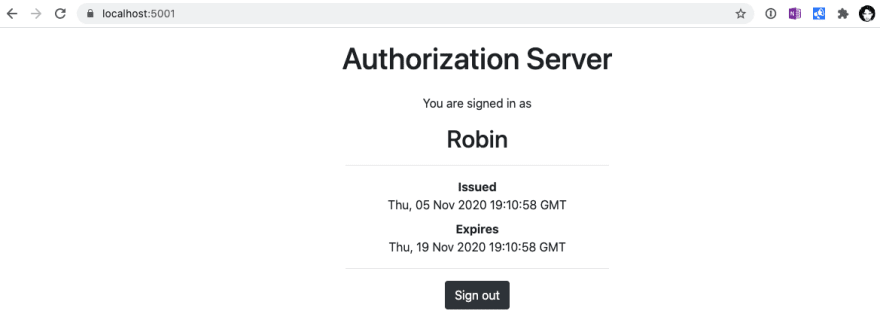
Start the application, you should see a sign-in button on the home page:

[](https://res.cloudinary.com/practicaldev/image/fetch/s--FwO4--n8--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/dev-to-uploads.s3.amazonaws.com/i/35a01y8defub4u2ina7l.png)

When you click Sign in you should navigate to the login form:

[](https://res.cloudinary.com/practicaldev/image/fetch/s--gABHALeq--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/dev-to-uploads.s3.amazonaws.com/i/lghiwqsrfwwn7gqss16p.png)

To sign in, just fill in random credentials, everything but an empty value is fine.  
If everything works correctly you should be redirected to the home page which shows you are signed in:

[](https://res.cloudinary.com/practicaldev/image/fetch/s--UldJaNLn--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/dev-to-uploads.s3.amazonaws.com/i/yqzbu8i2m2dtu5brjcq9.png)

**Next**

Currently, we have a basic ASPNET Core project running with authentication implemented, nothing fancy so far. [Next up](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-iii-client-credentials-flow-55lp), we will add OpenIddict to the project and implement the Client Credentials Flow.

**Discussion (7)**

Subscribe

Top of Form





Bottom of Form

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

Thada Wangthammang

•[Jun 6 '21](https://dev.to/mildronize/comment/1f69i)

Amazing articles thank you a lot.

You may forget to add this like line in file Startup.cs

using Microsoft.AspNetCore.Authentication.Cookies;

I think it should include in [this section](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949#enable-authentication)

2 likes[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949/comments/new/1f69i)

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

Robin van der Knaap

•[Jun 6 '21](https://dev.to/robinvanderknaap/comment/1f6b5)

Yes, that is correct. Thanx.

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949/comments/new/1f6b5)

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

Jeffrey Hartmann

•[Dec 28 '21](https://dev.to/lefty/comment/1kmkc)

I needed to change the definition of ReturnUrl in LoginViewModel to be nullable in order for the ModelState to be valid if a ReturnUrl wasn't supplied.

public string? ReturnUrl { get; set; }

2 likes[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949/comments/new/1kmkc)

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

Chrlol

•[Feb 23 • Edited on Feb 23](https://dev.to/chrlol/comment/1ma8n)

Or you can remove  
< Nullable >enable</ Nullable >  
from the csproj file

Like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949/comments/new/1ma8n)

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

codingsoldiers

•[Jan 16](https://dev.to/codingsoldiers/comment/1la9c)

Robin, Thank you for such great content explaining the implementation of OpenIddict. This would be the most detailed tutorial I have been able to find, but I am struggling with how to roll out my implementation using ASP.NET Identity. How does the application flow in the situation where users use the Identity Register/Login Pages to gain access to backend services. Being fairly new to OpenIddict, I see a lot of areas where things just don't make sense and I have a desire to understand. Thank you for any additional information you can provide. Take Care and Have a Great Day.

Charles

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949/comments/new/1la9c)

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

Tim Dowd

•[Jul 28 '21](https://dev.to/timothydowd/comment/1gkgp)

Thanks for the great tutorial. Does the auth ui need to be part of the auth server or can the ui be separate?

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949/comments/new/1gkgp)

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

Robin van der Knaap

•[Jul 31 '21](https://dev.to/robinvanderknaap/comment/1gmpi)

As far as I know, you should be able to separate the UI from the auth server.

1 like[Reply](https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949#/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-ii-create-aspnet-project-4949/comments/new/1gmpi)

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