



# HangFire integration with SSO Identity Server 4 .net core 2.0



Wilson Santos · [Follow](#)

5 min read · Sep 25, 2018



Share



More

## History:

A web application throughout its life cycle accumulates many processes that can be Asynchronous, especially longer processes, in a completely independent execution pipeline, thus relieving your main application.

Various implementations can be made with the idea of Asynchrony, outside the main application process, for this, we commonly use the concepts of Queues RabittMQ is very famous MessageBroker and takes care of this type of demand, in addition, Azure also offers messaging services Service Bus, in the PAAS model, a MAAS in fact .

All of these solutions are a little complicated and require infrastructure and more specialized knowledge.

A really cool option is HangFire , which can perform this role with minimal infrastructure and configuration, using its SQL-Server for example and a web application we can start separating processes into another execution pipeline.

## What problem does it solve?

Solution for executing asynchronous processes, relieving your application pipeline integrated with your SSO solution.

## Implementation:

hangfire runs on an MVC application, a web application, so create an empty Web Application project and install the hangfire package via nuget

### Install-Package Hangfire

After that, hangfire configuration is mostly concentrated in the web application's Startup.cs file.

in the ConfigureServices method, we need to add Hangfire and configure its storage as per the example below;

```
var connectionString = Configuration.GetSection("ConfigConnectionString:Default").Value;  
services.AddHangfire(x => x.UseSqlServerStorage(connectionString));
```

here we provide the connection string for hangfire to access its database, it will use an SQLServer to store the information necessary to manage Queues, Schedules, etc. (you can use mongo too).

Everything will be created automatically as long as the user has permission.

In the Configure method, we initialize the server and the Dashboard, the interface where we can manage HangFire

```
app.UseHangfireServer();  
app.UseHangfireDashboard();
```

In short, here we already have HangFire configured, and we can now use it

Example;

```
//Sample HTTP JOB  
var apiEndPoint = configSettingsBase.Value.ApiEndPoint;  
var authorityEndPoint = configSettingsBase.Value.AuthorityEndPoint;  
RecurringJob.AddOrUpdate(() => ExecuteCallHttpProcess(authorityEndPoint, apiEndPoint), Cron.Daily);
```

Here we will create a recurring service, which runs every day, basically we have a method called ExecuteCallHttpProcess that executes something on the defined

schedule. ( [here we can see more examples of Scheduling queues etc.](#) )

**ExecuteCallHttpProcess:** It is an implementation used to make http calls from an API, so we can automate backend processes in a simple way, but be careful, this practice does not separate the processing on the Stack from hangfire as it only makes an HTTP call that actually executes the operation in the target API, on the other hand, if we write the process in the hangfire stack, you place this processing separately from the stack of your main application.

Once this is done, simply configure the basic middleware for an MVC application, and run it, the dashboard's Default url is /hangfire .

[Here you have the complete startup file](#)

## Configuring the Authorization process

Note that this tool is open and if you publish it on the internet, anyone can access it, which is why hangFire offers a simple way to implement the Security process of this Dashboard.

Still in the Configure method, we will improve the Dashboard configuration, we will pass a custom URL and a filter for the Authorization process, see:

```
app.UseHangfireDashboard("/jobs", new DashboardOptions
{
    Authorization = new[] { new AuthoritionDashboardFilter() }
});
```

the AuthoritionDashboardFilter class is responsible for checking whether the application is Authenticated or not

```
public class AuthoritionDashboardFilter : IDashboardAuthorizationFilter
{
    public bool Authorize(DashboardContext context)
    {
        var httpContext = context.GetHttpContext();

        // Allow all authenticated users to see the Dashboard (potentially dangerous).
        return httpContext.User.Identity.IsAuthenticated;
        //return true;
    }
}
```

realize that basically the Authorize method returns can or cannot access the application, return true or false manually to test, and next we will see how to use an SSO Identity Server 4 to set the main user of this application as Authenticated, with it we will be able use the Implicit Flow flow and direct the user to log in to the SSO and then return to the application Logged in, our Hangfire will be just another SSO Client.

for this we need a Configured Identity Server server, this article can help you configure an [ASP.NET Core 2.1 with IdentityServer4](#)

and after configuring the Server we return to Startup.cs of the HangFire application  
In it, in the ConfigureServices method, we will configure the Authentication process

```
var authorityEndPoint = Configuration.GetSection("ConfigSettings:AuthorityEndPoint").Value;
services.AddAuthentication(options =>
{
    options.DefaultScheme = "Cookies";
    options.DefaultChallengeScheme = "oidc";
})
.AddCookie("Cookies")
.AddOpenIdConnect("oidc", options =>
{
    options.SignInScheme = "Cookies";

    options.Authority = authorityEndPoint;
    options.RequireHttpsMetadata = false;

    options.ClientId = "hangfire-dash";
    options.SaveTokens = true;
});
```

basically we pass the SSO URL that is in appsettings.json, and define the client that must be configured in SSO as below

```

new Client
{
    ClientId = "hangfire-dash",
    ClientName = "HangFire Dashboard",
    ClientSecrets = { new Secret("segredo".Sha256()) },

    AllowedGrantTypes = GrantTypes.Implicit,
    AllowAccessTokensViaBrowser = true,
    RedirectUri = {
        "http://localhost:8123/signin-oidc",
    },

    AllowedScopes = {
        StandardScopes.OpenId,
        StandardScopes.Profile,
        StandardScopes.Email,
        "ssosa"
    },
    RequireConsent = false,
    AccessTokenLifetime = 43200
},

```

In this client we have the hangfire application address that will be used after the user logs into SSO when returning to the application.

The rest are authorization settings for OpenId Connect or (OIDC) a layer that implements an Identity server over the OAuth2 protocol, [in this video we have Heber Pereira presenting the Identity Server](#), (I've already taken his course, I recommend this subject, it's extensive, complicated and dense we need a lot of study to avoid messing up)

let's add the Configure Authentication Middleare method

```
app.UseAuthentication();
```

and we're basically done.

**Toast :** to avoid problems with Dependency Injection in the methods that HangFire will execute, it is important to configure this in the Configure method

```
// Configure hangfire to use the new JobActivator we defined.  
GlobalConfiguration.Configuration.UseActivator(new HangfireActivator(serviceProvider));
```

The HangfireActivator class is not git.

I abstracted some implementations and encapsulated them for reuse (you can find them on my [github](#) )

1. [Seed do HangFire](#)
2. [Seed do Basic SSO](#)
3. [AuthoritionFilter](#)
4. [HangfireActivator](#)

Hangfire

Dotnet Core

Sso

Jobs



Follow

## Written by Wilson Santos

88 Followers

For the last 15 years, I have been developing, perfecting and integrating systems, I am passionate about

Open in app ↗



Search



more from Wilson Santos



## Configure Default Porta do Angular



```
Your global Angular CLI version (9.0.2) is greater than your local
version (6.0.1). The local Angular CLI version is used.

To disable this warning use "ng config --cli.warnings.versionMismatch false".
12% building modules 19/22 modules 3 active ...ffice.Spa.Ui\src\assets\scss\style.scss
<--- Last few GCs --->
[7828:00000212829FD630] 98851 ms: Scavenge 1331.7 (1425.2) -> 1331.1 (1425.7) MB, 4.4 / 0.0 ms (average mu = 0.146, current mu = 0.011) allocation failure
[7828:00000212829FD630] 99918 ms: Mark-sweep 1331.8 (1425.7) -> 1331.4 (1425.2) MB, 1059.5 / 0.0 ms (average mu = 0.093, current mu = 0.037) allocation failure
[7828:00000212829FD630] 99930 ms: Scavenge 1332.1 (1425.2) -> 1331.8 (1426.2) MB, 6.5 / 0.0 ms (average mu = 0.093, current mu = 0.037) allocation failure
<--- JS stacktrace --->

==== JS stack trace =====
    0: ExitFrame [pc: 00000180B35DC5C1]
Security context: 0x01c479B1e6e9 <JSObject>
    1: createSynthesizedNode(aka createSynthesizedNode) [000002FADF1450A9] [C:\Projeto\Score.Platform.Features\Score.Platform.Features.Backoffice.Spa.Ui\node_modules\typescript\lib\typescript.js:~48720] [pc=00000180B4EF455F](this=0x01d27c7026f1 <undefined>,kind=247)
    2: createExportSpecifier [000002FADF1478A9] [C:\Projeto\Score.Platform.Features\Score...
FATAL ERROR: Ineffective mark-compacts near heap limit Allocation failed - JavaScript heap out of memory
 1: 00007FF68298008A v8::internal::GCIdleTimeHandler::GCIdleTimeHandler+4506
 2: 00007FF6828E8336 node::MakeCallback+4534
 3: 00007FF6828E9280 node::module_register+2B32
 4: 00007FF682C0340E v8::internal::FatalProcessOutOfMemory+846
 5: 00007FF682C0340F v8::internal::FatalProcessOutOfMemory+639
 6: 00007FF682DE9384 v8::internal::Heap::MaxHeapGrowingFactor+962B
 7: 00007FF682DE97E6 v8::internal::ScavengeJob::operator+=+24550
 8: 00007FF682D0E13C v8::internal::ScavengeJob::operator+=+17980
 9: 00007FF682DE7887 v8::internal::Heap::MaxHeapGrowingFactor+2327
10: 00007FF682DE7C06 v8::internal::Heap::MaxHeapGrowingFactor+2454
11: 00007FF682F11EA7 v8::internal::Factory::NewFillerObject+55
12: 00007FF682F8F096 v8::internal::operator<+73494
13: 00000180B35DC5C1
```



Wilson Santos

## Increase-memory-limit angular

compilation of very large projects

1 min read · Feb 19, 2021







Wilson Santos

## Entity Framework EnableSensitiveDataLogging

Quick tip for an annoying error in EF, regarding context state control, if for some reason there are two entities with the...

2 min read · Aug 14, 2019




8



See all from Wilson Santos

Recommended from Medium



 Juan Spain in ByteHide

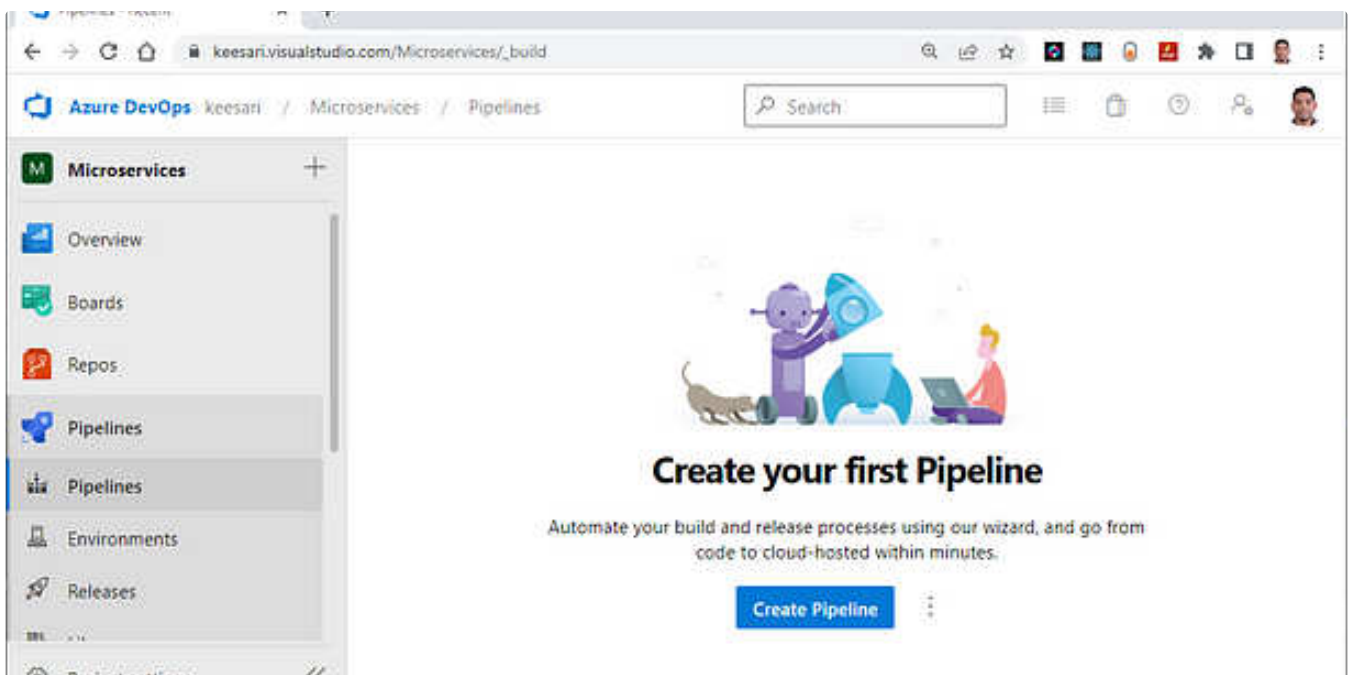
## HashMap in C#: Detailed Guide

Understanding HashMap in C#

🌟 · 9 min read · Dec 22

 73 



 Anji Keesari

## Create Azure DevOps pipeline—for .NET Core Web API

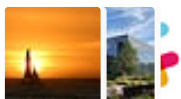
Introduction:

8 min read · Sep 6

 25 

## Lists



### Stories to Help You Level-Up at Work

19 stories · 386 saves



### Work 101

26 stories · 79saves



### Productivity

234 stories · 251 saves



### Staff Picks

544 stories · 577saves



 Anish Bilas Panta 

## Configuring Duende Identity Server on .NET: Unlock the Secrets of Secure Identity Management

Boost User Engagement and Protect Your Application with Cutting-Edge Authentication and Authorization Techniques

4 min read · Jul 15

 151 




 Niraj Ranasinghe

## From Slow to Swift : Accelerating Applications with .NET 7 Caching Methods

Optimizing application speed is essential for providing outstanding user experiences in today's fast-paced digital environment. Caching is...

13 min read · Jul 10

 2 



# .http Files in Visual Studio



Henrique Siebert Domareski

## .http Files in Visual Studio

The Visual Studio 2022 .http file editor provides an easy way to test Web APIs directly on Visual Studio, without needing to switch to...

5 min read · 5 days ago



22



Dr. Ernesto Lee



# Building a .NET 7 Microservice with Monitoring and Centralized Logging on Gitpod

Introduction

4 min read · Sep 13



66



---

See more recommendations