# Using Background Processing to Build Scalable Applications with Hangfire

## Lab 8

### Goal

As with any library, Hangfire has some not so well documented features as well as other goodies that you may want to take advantage of as you learn the ins and outs. In this lab we will take a look at two of these features.

The first feature we will take a look at is the implementation of authorization for the dashboard.

*Reminder: Hangfire wants to be as secure as possible out of the box so the dashboard will only respond to requests when running on localhost.*

1. Add a new folder to the Web project called Security
2. Implement the HangfireDashboardAuthorizationFilter class

public class HangfireDashboardAuthorizationFilter : IAuthorizationFilter

{

public bool Authorize(IDictionary<string, object> owinEnvironment)

{

var context = new OwinContext(owinEnvironment);

var user = context.Authentication.User;

if (user?.Identity == null || !user.Identity.IsAuthenticated)

return false;

return true; // Consider adding in a custom check here for a specific user role in your application

}

}

1. Add the following using statements to your new class

using System.Collections.Generic;

using Hangfire.Dashboard;

using Microsoft.Owin;

1. Note that we are allowing anyone that is authenticated to access the dashboard. In a production application you should implement some custom checking of AspNetIdentity Roles or other permissions if you have implemented something custom to restrict access to the dashboard.
2. Update the Startup.Hangfire file in the Web.App\_Start folder

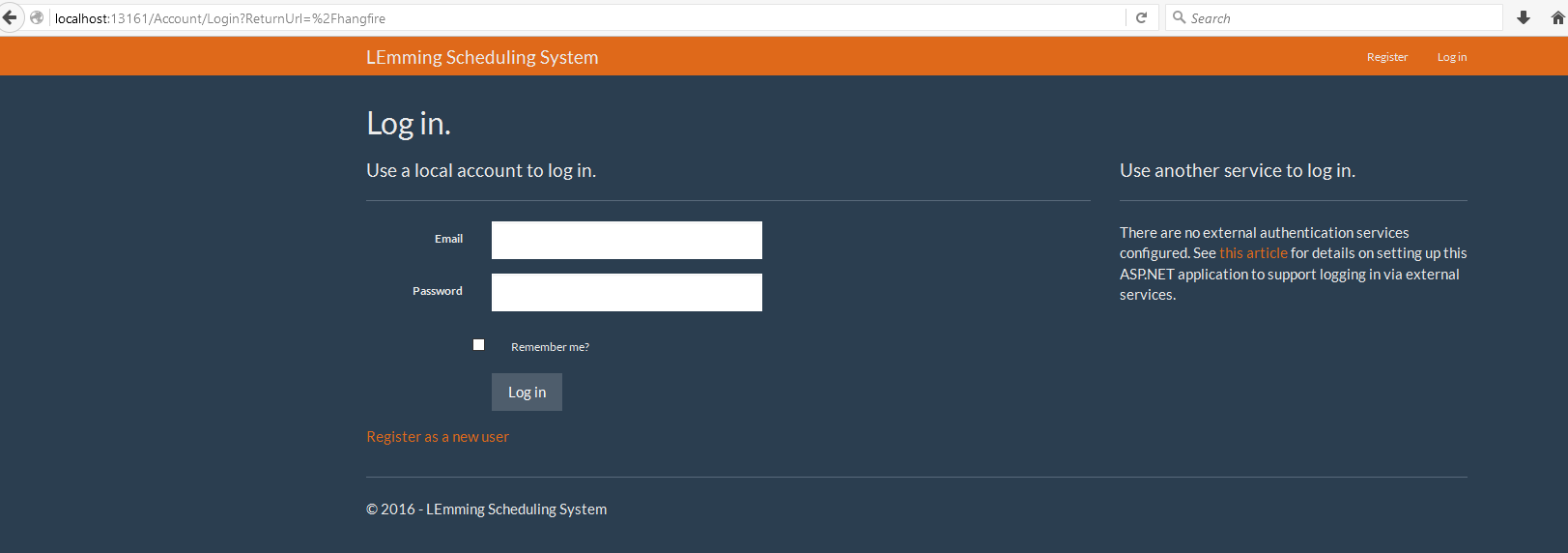
app.UseHangfireDashboard("/hangfire", new DashboardOptions()

{

AuthorizationFilters = new[] { new HangfireDashboardAuthorizationFilter() }

}); // This will configure the current application to run the Hangfire Dashboard with the configured connection.

1. Run your project by pressing F5,navigate to <http://localhost:13161/hangfire> and take note that your authorization filter is now being applied and you will now be able to access the dashboard when your application is deployed to a different environment. (Note: There is no need to create an account as we are just demonstrating how to create the authorization filter.)



1. Stop running your project and go back to the HangfireDashboardAuthorizationFilter class that you created and comment out the checks in the filter and just return true. This will allow you to continue accessing the dashboard without the need to login for the remainder of the labs.

public class HangfireDashboardAuthorizationFilter : IAuthorizationFilter

{

public bool Authorize(IDictionary<string, object> owinEnvironment)

{

//var context = new OwinContext(owinEnvironment);

//var user = context.Authentication.User;

//if (user?.Identity == null || !user.Identity.IsAuthenticated)

// return false;

return true; // Consider adding in a custom check here for a specific user role in your application

}

}

In the next part of the lab we are going to take advantage of the DisplayName “easter egg”.

1. Go to the BuildHouseJob in Core.Jobs and add the following DisplayName attribute to the BuildHouseWithFriendWithName method:

[DisplayName("BuildHouseWithFriendWithName")]

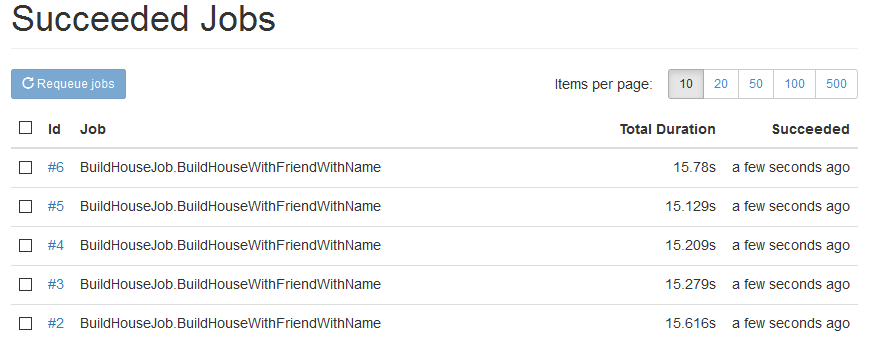
1. Add the following using statement to the file:

using System.ComponentModel;

1. Run your project by pressing F5 and navigate to <http://localhost:13161/hangfire/jobs/succeeded> and take a look at the difference in the job names now

Figure 1 - Before

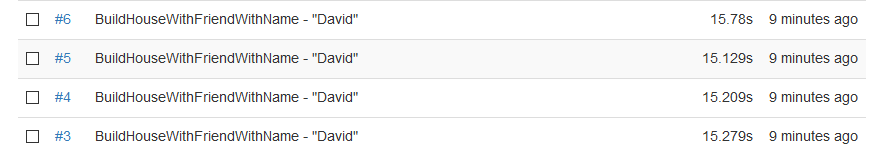
Figure 2 - After



1. Update the DisplayName attribute you added in step 8 to the following:

[DisplayName("BuildHouseWithFriendWithName - {0}")]

1. Run your project by pressing F5 and navigate to <http://localhost:13161/hangfire/jobs/succeeded> and take a look at the difference in the job names now



1. Take note that if we had more parameters on our method we could utilize common string formatting techniques to add them to our display name however we like (i.e. {1}, {2}, etc.)

This completes Lab 08.