Using Background Processing to Build Scalable Applications with Hangfire

Lab 2

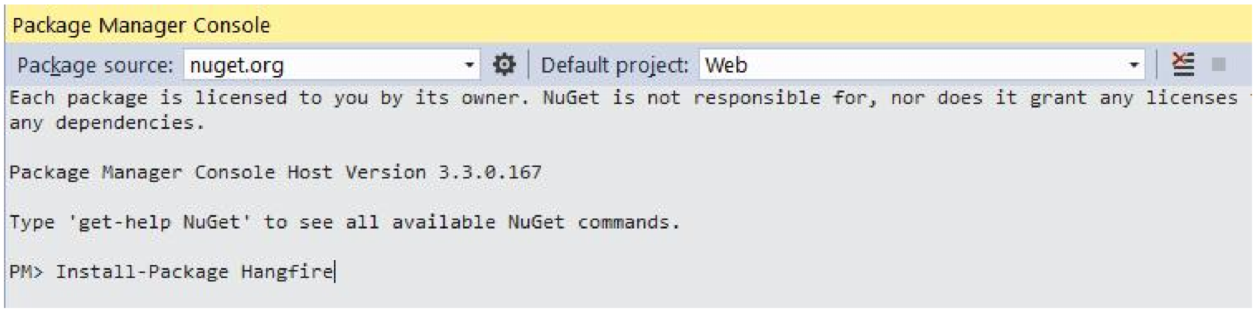
Goal

In this exercise we will install Hangfire, configure our application to run a Hangfire Server, and refactor our Lemmings to perform their work using Background Jobs.

Part I

1. Hangfire is installed as a NuGet package. To install Hangfire, open the Package Manager Console from Visual Studio (Tools \ NuGet Package Manager \ Package Manager Console).
2. Verify that the Default Project is "Web".
3. Type the following command at the PM> prompt:

```
PM> Install-Package Hangfire
```



Package Manager Console

Package source: nuget.org — Default project: Web

```
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses
any dependencies.

Package Manager Console Host Version 3.3.0.167

Type 'get-help NuGet' to see all available NuGet commands.

PM> Install-Package Hangfire
```

4. In the App_Start folder (of the Web Project), add a new class called Startup.Hangfire.cs.

5. Copy the following into the Startup.Hangfire.cs file:

   ```
   using Hangfire;
   using Owin;

   namespace Web
   {
       public partial class Startup
       {
           public void ConfigureHangfire(IAppBuilder app)
           {
               GlobalConfiguration
                   .Configuration
                   .UseSqlServerStorage("DefaultConnection");

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

   In this file, we are creating a partial class to handle our Hangfire configuration. First, we are configuring Hangfire to use SQL Server for persistent storage and to use the database referenced by the *DefaultConnection* connection string in our web.config. For the purposes of this workshop, we will share our application database with Hangfire.

   Next, we tell Hangfire that we want to host the Hangfire Dashboard in our application. The Dashboard provides a management and monitoring interface which we will use throughout the workshop.

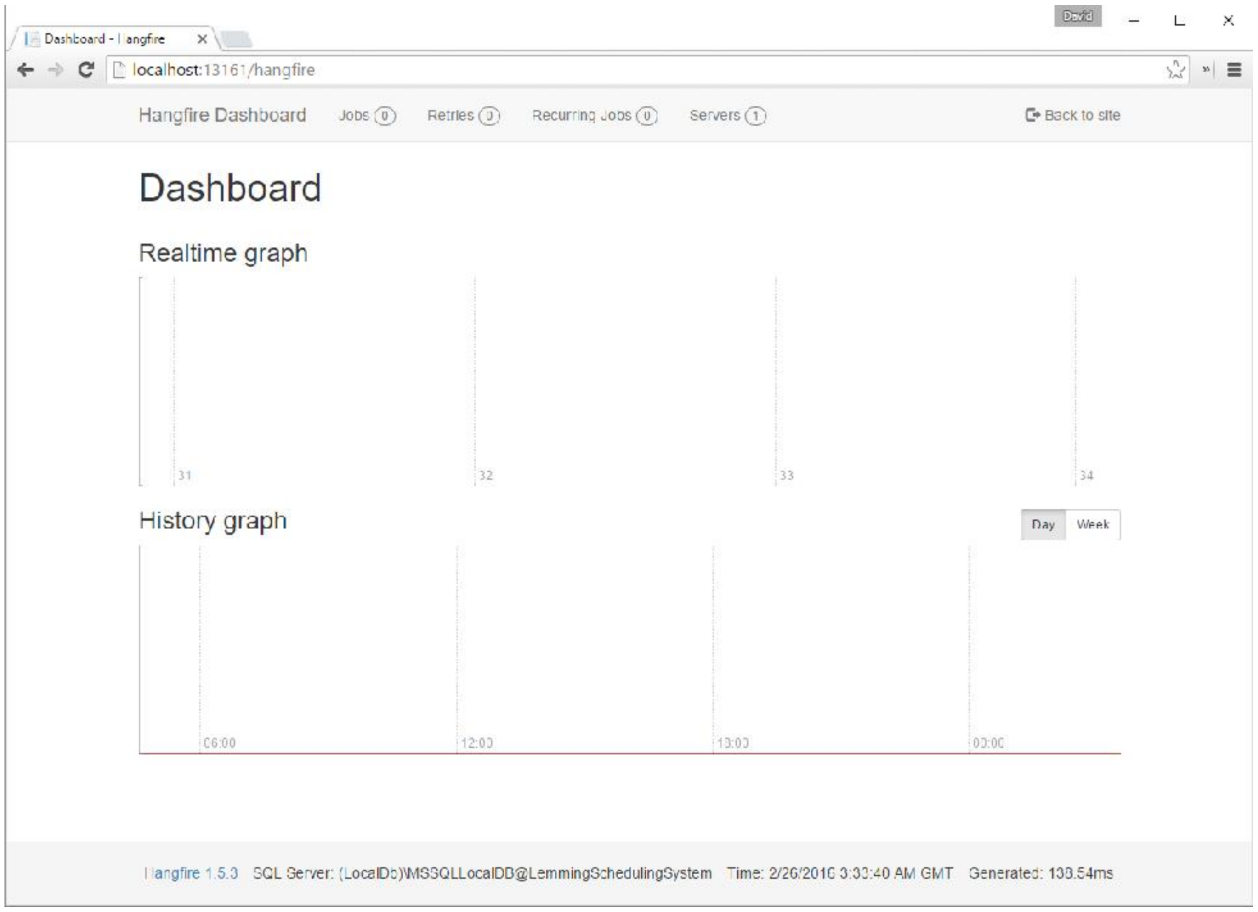
   Lastly, we start up our Hangfire Server, hosted inside of our application.

6. Open the Startup.cs file under the root of the Web project.
7. Modify the Configure method as follows:

   ```
   public void Configuration(IAppBuilder app)
   {
       ConfigureAuth(app);
       ConfigureHangfire(app);
   }
   ```

8. Now that Hangfire is installed and configured, let’s run our application and verify that everything is configured properly. Press F5 to start debugging.
9. The LEmming Scheduling System should launch just as before.

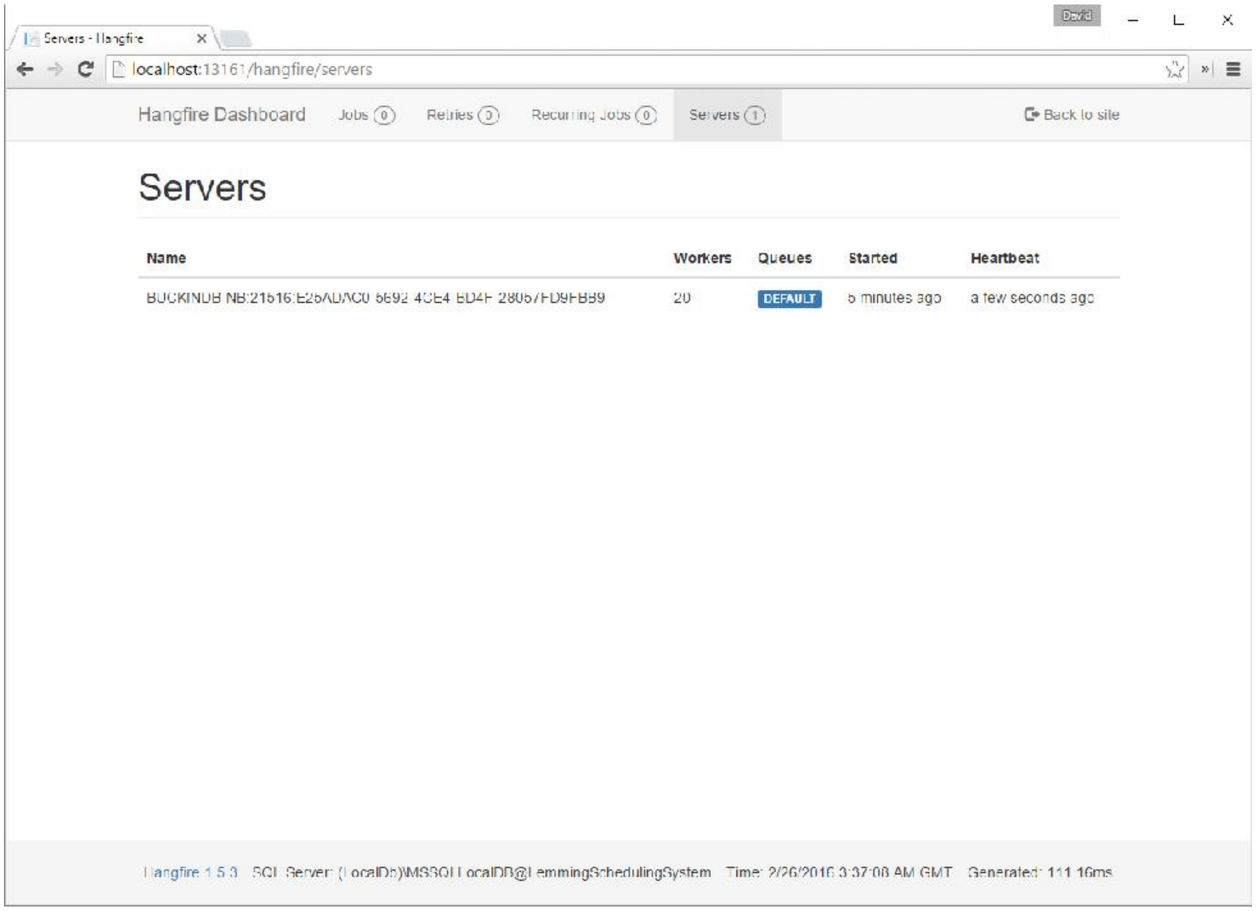
10. Browse to http://localhost:13161/hangfire. You should see the Hangfire Dashboard.



Hangfire 1.5.3 SQL Server: (LocalDb)\MSSQLLocalDB@LemmingSchedulingSystem Time: 2/26/2016 3:33:40 AM GMT Generated: 138.54ms

11. Notice on the bottom of the Dashboard, we can find the Hangfire Version and the current storage configuration (SQL Server in our case).

12. In the navigation bar, click Servers. You should see a single server listed. This is a reference to the server that we configured to run inside of our application.



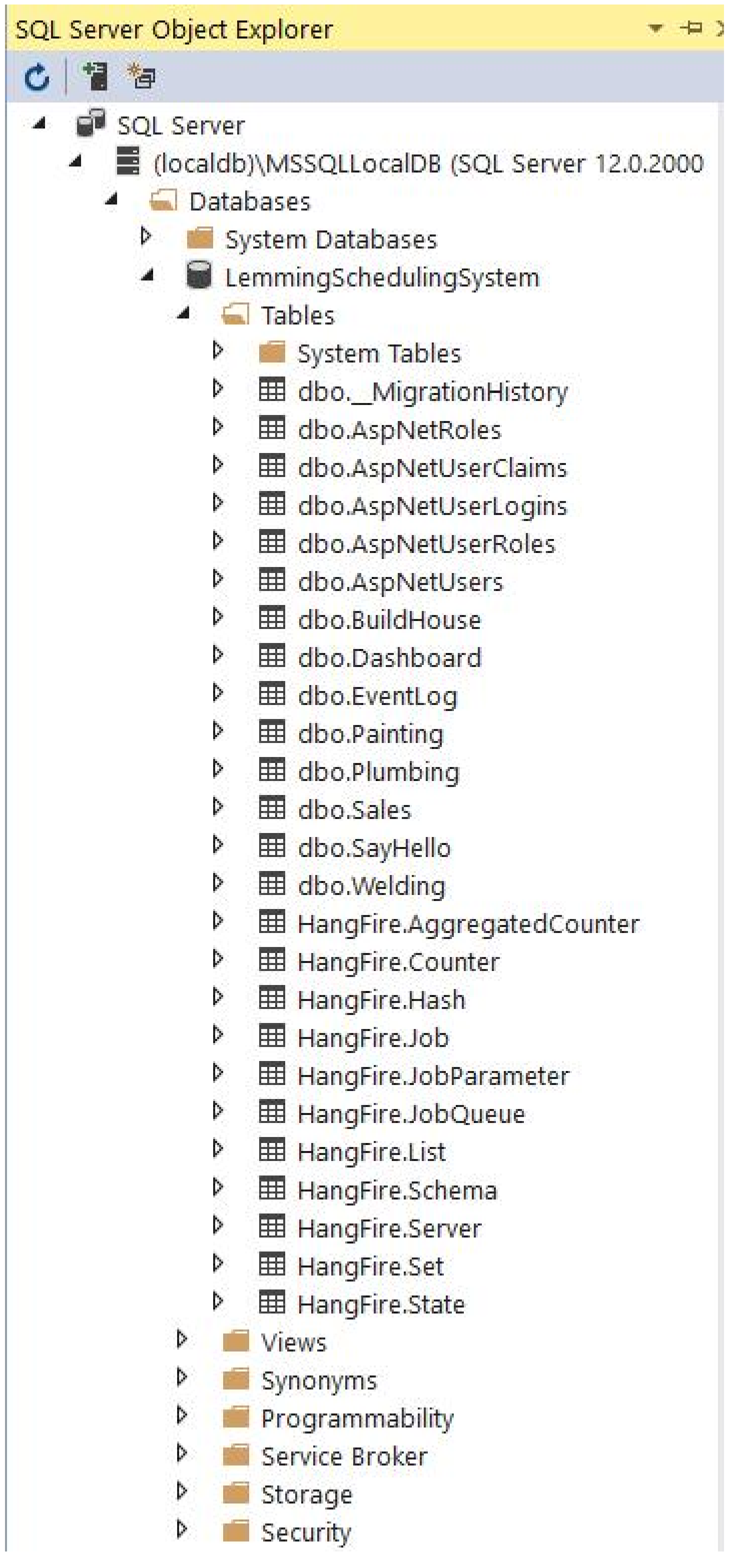
Servers

Name	Workers	Queues	Started	Heartbeat
BUCKINDB NB:21616:E25AD/AC0 6692 4CE4 BD4F 28057FD9FBB9	20	DEFAULT	5 minutes ago	a few seconds ago

Hangfire 1.5.3 SQL Server: (LocalDb)\MSSQLLocalDB@LemmingSchedulingSystem Time: 2/26/2016 3:37:08 AM GMT Generated: 111.16ms

13. Return to Visual Studio.
14. Open SQL Server Object Explorer (View \ SQL Server Object Explorer).
    a. Alternately, you can use SQL Server Management Studio (if you prefer)

15. Navigate to (localdb)\MSSQLLocalDB \ Databases \ LemmingSchedulingSystem. Expand the list of Tables.



Notice that Hangfire has installed all of the necessary tables into its own schema, neatly separating itself from our application tables.

Part II

Goal

In this part of the exercise we will be refactoring application so that our Lemmings can work asynchronously in the background to complete their work. We will do this by forcing our Lemmings to perform their by implementing a BaseLemmingJob which we will derive other jobs from.

1. Stop debugging by pressing SHIFT + F5. With Hangfire properly installed and configured, we can now begin refactoring our Lemmings to perform their work in the background.
2. In the Core project, create a new folder called Jobs.
3. Add a new class called BaseLemmingJob.

```
using System;
using System.Linq;
using System.Threading;
using Core.Context;
using Core.Data;

namespace Core.Jobs
{
    public abstract class BaseLemmingJob
    {
        private readonly ApplicationDbContext _dbContext = new ApplicationDbCo
ntext();

        public void Run()
        {
            System.Diagnostics.Debug.WriteLine($"Running job of type {GetType(
).Name}.");
            SleepyTime();
            DoWork();
        }

        protected abstract void DoWork();

        protected void UpdateDashboard<T>() where T : BaseEntity, IDashboard,
new()
        {
            _dbContext.Set<T>().Add(new T() { Created = DateTime.Now, Dashboar
d = _dbContext.Set<Dashboard>().FirstOrDefault() ?? new Dashboard() { Created
= DateTime.Now } });
            _dbContext.SaveChanges();
        }

        // This helps us simulate a task taking a while to run
        private void SleepyTime()
        {
            var random = new Random();

            Thread.Sleep(random.Next(1000, 10000));
        }
    }
}
```

The BaseLemmingJob is an abstract class that will simulates work that each of our Lemmings can perform.

4. Add a new class to the Jobs folder, called SayHelloJob.

```
using Core.Data;

namespace Core.Jobs
{
    public class SayHelloJob : BaseLemmingJob
    {
        protected override void DoWork()
        {
            System.Diagnostics.Debug.WriteLine($"Hello!");
            UpdateDashboard<SayHello>();
        }
    }
}
```

   The SayHelloJob class is a concrete implementation of our BaseLemmingJob.

5. Add a new class to the Jobs folder, called BuildHouseJob.

```
using Core.Data;

namespace Core.Jobs
{
    public class BuildHouseJob : BaseLemmingJob
    {
        protected override void DoWork()
        {
            System.Diagnostics.Debug.WriteLine($"Building a house...");
            UpdateDashboard<BuildHouse>();
        }
    }
}
```

6. Add a new class to the Jobs folder, called PaintingJob.

```
using Core.Data;

namespace Core.Jobs
{
    public class PaintingJob : BaseLemmingJob
    {
        protected override void DoWork()
        {
            System.Diagnostics.Debug.WriteLine($"Painting some walls...");
            UpdateDashboard<Painting>();
        }
    }
}
```

7. Add a new class to the Jobs folder, called PlumbingJob.

```
using Core.Data;

namespace Core.Jobs
{
    public class PlumbingJob : BaseLemmingJob
    {
        protected override void DoWork()
        {
            System.Diagnostics.Debug.WriteLine($"Unclogging some toilets...");
            UpdateDashboard<Plumbing>();
        }
    }
}
```

8. Add a new class to the Jobs folder, called SalesJob.

```
using Core.Data;

namespace Core.Jobs
{
    public class SalesJob : BaseLemmingJob
    {
        protected override void DoWork()
        {
            System.Diagnostics.Debug.WriteLine($"ABC...Always be closing...");
            UpdateDashboard<Sales>();
        }
    }
}
```

9. Add a new class to the Jobs folder, called WeldingJob.

```
using Core.Data;

namespace Core.Jobs
{
    public class WeldingJob : BaseLemmingJob
    {
        protected override void DoWork()
        {
            System.Diagnostics.Debug.WriteLine($"Welding some metal...");
            UpdateDashboard<Welding>();
        }
    }
}
```

10. Add the following method to the HomeController class.

```
private void RunAsync<T>(Expression<Action<T>> job, int? quantity) where T : B
aseLemmingJob
{
    Task.Run(() =>
    {
        var timesToRun = quantity ?? 1;
        for (var i = 0; i < timesToRun; i++)
        {
            BackgroundJob.Enqueue(job);
        }
    });
}
```

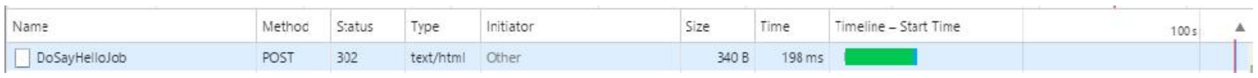
11. Update each Do___Job Action (such as DoSayHelloJob) as follows, using the appropriate BaseLemmingJob type for each Action:

```
[HttpPost]
public ActionResult DoSayHelloJob(int? quantity)
{
    RunAsync<SayHelloJob>(j => j.Run(), quantity);

    return RedirectToAction("Index");
}
```

12. Update the Lemming Dashboard so that it will update the metrics by opening the Home Index view (Views / Home / Index) and uncommenting the javascript block at the end of the file

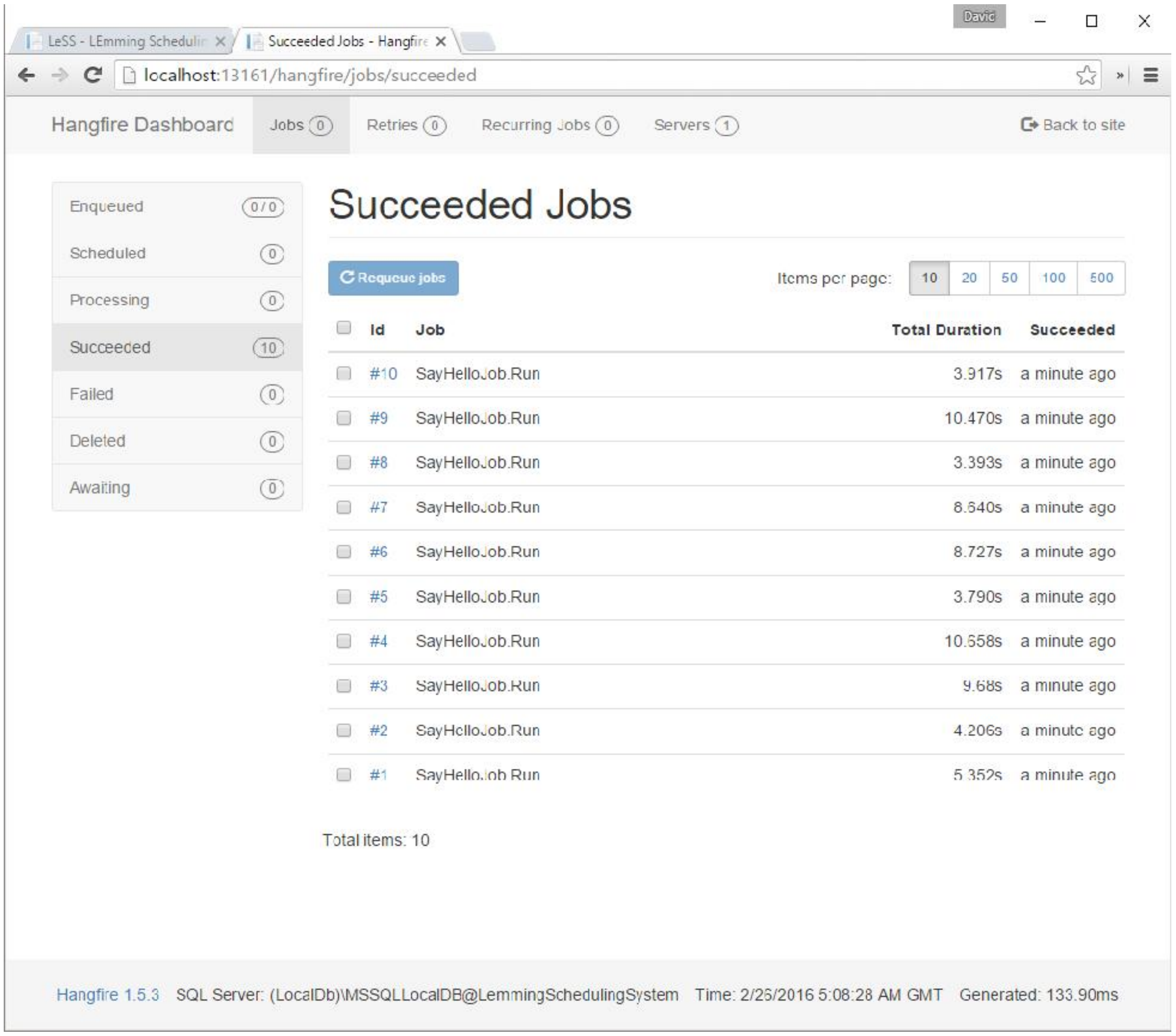
```
@*@section scripts
{
    <script>
        function refreshDashboard() {
            setTimeout(function () {
                $.get('@Url.Action("GetDashboard", "Home")', function
(data) {
                    $('#dashboard').html(data);
                }).always(function() {
                    refreshDashboard();
                });
            }, 10000);
        }

        refreshDashboard();
    </script>
}*@
```

13. Once each Action has been updated, press F5 to Build and Run our application again.
14. With the Developer Tools open, and the Network tab visible, schedule 10 Hello Lemmings.
15. In the Network tab of the Developer Tools, observe that the POST to the DoSayHelloJob job is nearly instantaneous.



Name	Method	Status	Type	Initiator	Size	Time	Timeline – Start Time
DoSayHelloJob	POST	302	text/html	Other	340 B	198 ms	

16. Open a new tab in your browser, and browse to the Hangfire Dashboard (http://localhost:13161/hangfire).



Succeeded Jobs

Id	Job	Total Duration	Succeeded
#10	SayHelloJob.Run	3.917s	a minute ago
#9	SayHelloJob.Run	10.470s	a minute ago
#8	SayHelloJob.Run	3.393s	a minute ago
#7	SayHelloJob.Run	8.640s	a minute ago
#6	SayHelloJob.Run	8.727s	a minute ago
#5	SayHelloJob.Run	3.790s	a minute ago
#4	SayHelloJob.Run	10.658s	a minute ago
#3	SayHelloJob.Run	9.68s	a minute ago
#2	SayHelloJob.Run	4.206s	a minute ago
#1	SayHelloJob.Run	5.352s	a minute ago

Total items: 10

Hangfire 1.5.3 SQL Server: (LocalDb)\MSSQLLocalDB@LemmingSchedulingSystem Time: 2/25/2016 5:08:28 AM GMT Generated: 133.90ms

17. Observe the progress of the SayHelloJob.
18. Try loading different quantities of the different Lemming jobs. Notice the how responsive the application is when scheduling large quantities of our Lemmings. Follow the various Lemming jobs as they are processed by using the Hangfire Dashboard.

This completes Lab 02.