Exercises in .NET with Andras Nemes

Tips and tricks in C# .NET

Introduction to WebSockets with SignalR in .NET Part 2: code basics

MAY 19, 2014 1 COMMENT (HTTPS://DOTNETCODR.COM/2014/05/19/INTRODUCTION-TO-WEBSOCKETS-WITH-SIGNALR-IN-NET-PART-2-CODE-BASICS/#COMMENTS)

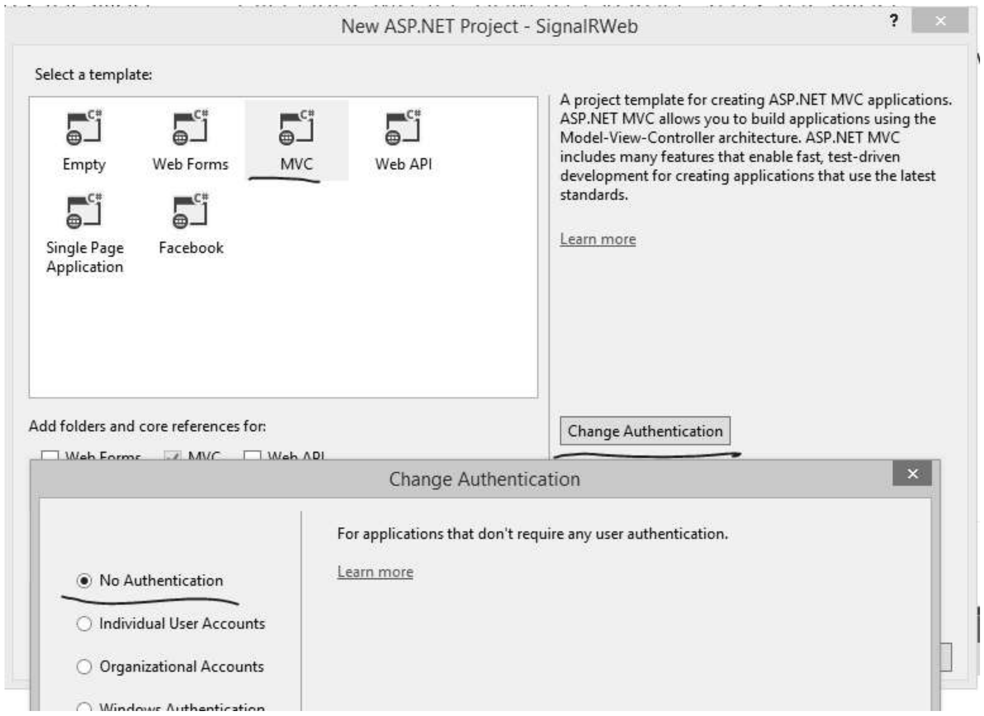
Introduction

In the previous post (https://dotnetcodr.com/2014/05/15/introduction-to-websockets-with-signalr-in-net-part-1-the-basics/) of this series we looked at the foundations of WebSockets and SignalR. We're now ready to look at some action.

Demo

I'll build this demo on Visual Studio 2013 Express for Web. If you have the full suite of VS 2013 Pro then you'll be fine too. The goal is to build two model applications with SignalR: a rudimentary chat site and an equally rudimentary stock ticker site where the current stock price is updated real-time based on what the server is reporting to it. Hopefully we'll learn enough so that you can build your own scenario: show real-time data on performance data, sport scores, the number of people on Earth etc.

Open VS 2013 and select the ASP.NET Web Application template in the New Project window with .NET 4.5 as the underlying framework. Call it SignalRWeb, click OK and then pick the MVC template. On the same window click "Change Authentication" and select the No Authentication option:

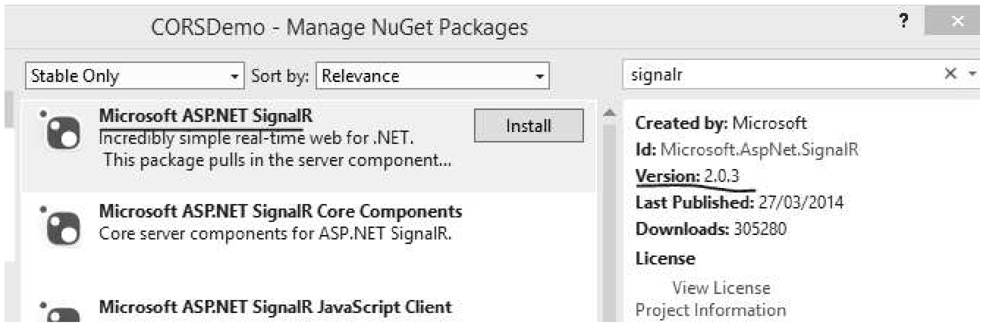


A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

For applications that don't require any user authentication.

(https://dotnetcodr.files.wordpress.com/2014/04/createwebproject1.png)

We don't want to deal with authentication issues as it would only divert us from the main topic. Click OK to create the MVC app. SignalR is not added automatically to the project. Add the Microsoft.AspNet.SignalR NuGet package to the solution:


CORSDemo - Manage NuGet Packages

Stable Only | Sort by: Relevance | signalr

Microsoft ASP.NET SignalR — Incredibly simple real-time web for .NET. This package pulls in the server component... [Install]

Microsoft ASP.NET SignalR Core Components — Core server components for ASP.NET SignalR.

Microsoft ASP.NET SignalR JavaScript Client

Created by: Microsoft
Id: Microsoft.AspNet.SignalR
Version: 2.0.3
Last Published: 27/03/2014
Downloads: 305280
License
View License
Project Information

(https://dotnetcodr.files.wordpress.com/2014/04/signalr-nuget.png)

This will install a number of other NuGet packages and dependencies. You will be greeted with a readme.txt file in Visual Studio giving you hints about the initial steps. If you followed along the introductory posts on Owin (https://dotnetcodr.com/2014/04/14/owin-and-katana-part-1-the-basics/) then you'll recognise the short sample code:

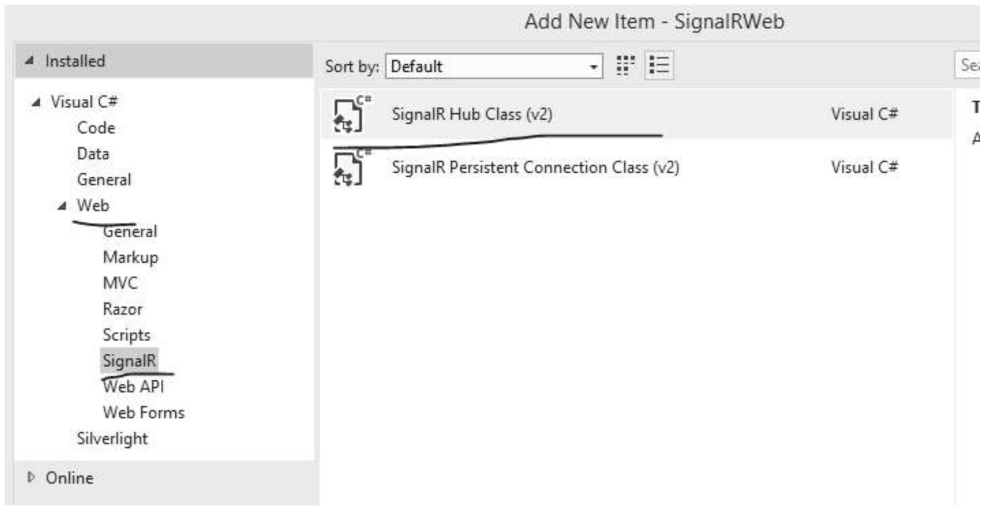
```
public class Startup
{
     public void Configuration(IAppBuilder app)
     {
         app.MapSignalR();
     }
}
```

We have the Startup class with the Configuration method that accepts an IAppBuilder object. We call an extension method on this app builder. If you don't understand what this means then it's best if you at least skim through the posts on OWIN first.

Normally if you create an MVC app with the default "Individual user Accounts" authentication option then the project starting point will already have Katana components in it that are related to authentication: OAuth – Facebook, Google, etc., or authentication using some other user store. However, we have no authentication at all in our project and there's no OWIN Startup entry point class either. In fact, the Owin libraries were only added when we downloaded the SignalR NuGet package and its dependencies. This means we'll need to create it ourselves, but that's OK. Add a new class to the project called Startup.cs. Be sure to call it "Startup" and not "startup" or "StartUp" so that we follow the naming conventions. Let's do exactly as it says in readme.txt and add the following code to Startup.cs:

```
public void Configuration(IAppBuilder app)
{
     app.MapSignalR();
}
```

You'll need to import the Owin namespace. This extension will make sure that our app starts listening to requests. You'll recall from the previous post that we'll need to create at least one Hub that clients can connect to. Add a new folder called Hubs. In that folder add a new class. In the Add New Item window you'll notice that there are some new template types under the Web/SignalR category:



(https://dotnetcodr.files.wordpress.com/2014/04/signalr-hub-class-template.png)

Select the Hub Class template. The Persistent Connection template is similar but you'll need to deal with connections at a lower level whereas the Hub Class represents a higher level of abstraction. I've never used the Persistent Connection template but I believe it's rather for experienced programmers who may want to control connections at a lower level for customisation purposes.

Call the class "ResultsHub.cs". You'll be provided a very short class implementation that derives from Hub:

```
public class ResultsHub : Hub
{
     public void Hello()
     {
         Clients.All.hello();
     }
}
```

Let's explore this a little before we continue with our main goal. Clients.All represents all clients that have connected to this hub which has a simple hello() method. "All" is a dynamic property so you can attach any method name to it: elvisPresley(), mickeyMouse() etc., so don't expect any IntelliSense there. hello() is therefore a dynamic method, you won't find it through IntelliSense either. This bit of code means that each client should be contacted and a JavaScript function called "hello" should be invoked on each of them. Keep in mind that we're programming a lot against JavaScript in SignalR hence the need for dynamic methods. IntelliSense cannot expect the exact JavaScript method names, right?

You don't have to contact every caller. Start typing "Clients." within the Hello function and you'll see other options, e.g.:

- Contact certain groups
- Contact every client except some with certain connection IDs
- One specific user

So you have some control over who will receive updates from this hub. We can send a greeting to the method if there's a matching JavaScript method on the client side:

```
public class ResultsHub : Hub
{
     public void Hello()
     {
         Clients.All.hello("Welcome from SignalR!");
     }
}
```

Let's see how this can be used in a view. Locate the Views/Index.cshtml view. Erase all code from it except:

```
@{
    ViewBag.Title = "Home Page";
}
```

We'll need some JavaScript on this page. The SignalR NuGet package has installed a couple of jQuery libraries specific to SignalR. Insert the following script section stub to Index.cshtml:

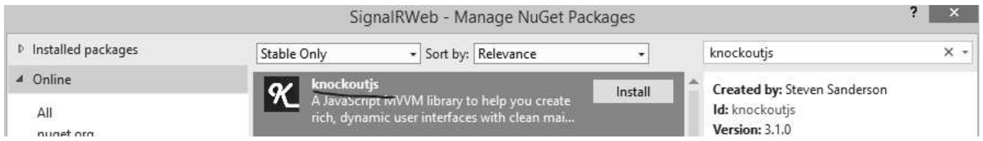
```
@section scripts
{
<script src="~/Scripts/jquery.signalR-2.0.3.js"></script>
}
```

The easiest way to achieve this is to locate the jquery.signalR-2.x.x file in the Scripts folder which you can drag and drop within the curly braces of the scripts section. That will create the script tag for you automatically. This library has a dependency on jQuery but that's already imported in Shared/_Layout.cshtml in a bundle (https://dotnetcodr.com/2013/01/14/web-optimisation-resource-bundling-and-minification-in-net4-5-mvc4-with-c/). You'll also find the scripts section declared in that common layout file which is why we can extend it in Index.cshtml.

We'll also need some custom javascript. Add a new JS file into the Scripts folder called results.js. Reference it already now on the Index page:

```
@section scripts
{
    <script src="~/Scripts/jquery.signalR-2.0.3.js"></script>
    <script src="~/Scripts/results.js"></script>
}
```

We'll also need knockout.js (http://knockoutjs.com/) to make the updates on the page very responsive and fluent. It's not currently available in our project so let's add the following NuGet package:



knockoutjs
A JavaScript MVVM library to help you create rich, dynamic user interfaces with clean mai...

Created by: Steven Sanderson
Id: knockoutjs
Version: 3.1.0

(https://dotnetcodr.files.wordpress.com/2014/04/knockout-js-nuget-package.png)

Add it to the scripts section:

```
@section scripts
{
    <script src="~/Scripts/jquery.signalR-2.0.3.js"></script>
    <script src="~/Scripts/knockout-3.1.0.js"></script>
    <script src="~/Scripts/results.js"></script>
}
```

We've laid the foundations for our SignalR app. We'll continue in the next post (https://dotnetcodr.com/2014/05/22/introduction-to-websockets-with-signalr-in-net-part-3/).

View the list of posts on Messaging here (https://dotnetcodr.com/messaging/).
FILED UNDER .NET, MESSAGING TAGGED WITH C#, SIGNALR

About Andras Nemes
I'm a .NET/Java developer living and working in Stockholm, Sweden.

One Response to *Introduction to WebSockets with SignalR in .NET Part 2: code basics*

Jesse Liu says:
May 19, 2014 at 10:00 am
Nice Post! Waiting for the section of self hosting with Owin.

Reply