SignalR with ASP.NET MVC5

Rajiv Gogoi

20 Aug 2014 CPOL

Introduction to SignalR using ASP.NET MVC 5

Introduction

In this article I would be trying to explain what SignalR is and would try to give a walkthrough using an ASP.NET MVC5 example. So even if you don't have any previous knowledge on SignalR, its fine as long as you know the basics of ASP.NET MVC, C# and Javascript.

SignalR is an web based real time bi-directional communication framework. By real time, we mean the clients get the messages being sent in real time as and when the server has something to send without the client requesting for it. And bi-directional because both the client and server can send messages to each other.

In the simplest terms, the way it works is the client is able to call methods on the server and the server likewise is able to call methods in the client.

Typical uses

Lets see some cases where SignalR can be used so that we can better appreciate the technology.

1. Chat Room application

This is an example where users keep sending messages to each other in a ChatRoom.

2. Broadcasting

This is an example where the server needs to broadcast messages to the clients.

E.g. Sports website where we keep getting live updated scores, Stock applications or News websites or even Facebook, Twitter updates.

3. Internet Games

This is where many players play some game online where each player does some action turn by turn.

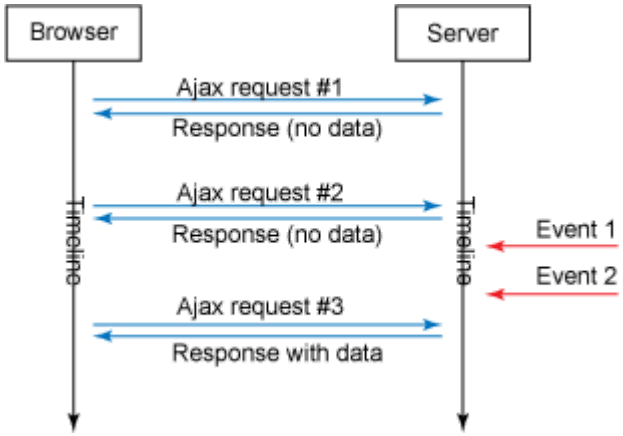
History of Real time communication and challenges

In the first liner introduction to SignalR, we used the term real time and bi-directional.

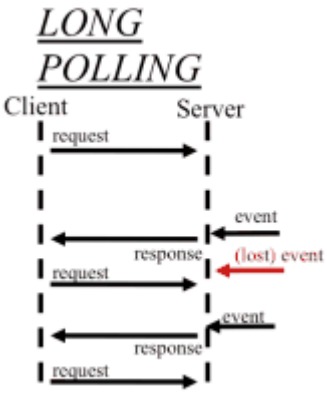
Now why do we need an entirely new technology for implementing real time communication. The reason is because of the way HTTP protocol works. So those of you who have some experience with web technologies know that HTTP works on a request/response mechanism. So a client(typically a web browser) makes an HTTP request to the Web Server and the Web Server sends an HTTP response back to the client. Unless the client makes a request to the server, the server has no knowledge of who its clients are and so it can't send a message to the client. We must have heard that HTTP is a stateless protocol, which essentially means that it doesn't remember which client made a request to it and how many times. For HTTP every request to the Web Server is an independant one. So for these reasons a bi-directional real time communication was a challenge before SignalR.

But this doesn't mean that we didn't had such applications like a sports news update site or a chat site before SignalR. Lets try to look at some techniques to achieve such real time communication before SignalR.

AJAX polling



Here as we can see the client makes some periodic requests to the server at some periodic intervals of time. In this way we keep the client updated with the latest data. As we can see from the image, when the client makes a request to the server, we don't know for sure that the server has something new to send so in a way those requests might be unnecessary. On the other hand when the server has something new to serve the client it can't send it unless the client makes a request, so we can't term this communication exactly real time.



Another technique which is a bit better than AJAX polling is long polling as shown above in the image. Here too the client makes a request to the server but it doesn't do so in periodic intervals of time. The client makes a request but the server doesnt respond immediately, rather it keeps the clients request open unless it has something new to serve. The client makes the new request after it gets a response back from the server. So this technique eliminates some of the drawbacks of the previous one and sounds a bit real time. But this technique needs some customization both on the client and server side to manage the client connections to the server and also managing timeouts.

Plugins

Plugins like flash or silverlight are some other technologies which can be used to implement real time communication. However in this world of mobility and HTML5 some mobile devices doesn't support plugin's so we have to be mindful of that. Speaking of mobility and HTML5, HTML5 actually brings in a concept of Web Sockets which also addresses the need of bi-directional communication. However we have to be using a modern browser for that.

As we can see that we have a number of options to implement real time bi-directional communication between client and server lets see how SignalR address these needs by creating a tunnel between client and server which is bi-directional in nature so that server can happily send messages to its attached clients whenever it wants to.

Lets get started

Now lets try to see some introduction concepts of SignalR.

WebSocket

WebSocket is a bi-directional TCP socket connection in which the client can send messages to the server and also the server can send messages to the client. Only IIS8+ webserver supports WebSockets.

SignalR Hub

The SignalR Hub is the server side part of the technology which exposes public methods to the connected clients.

Client

The client which is a web browser(also can be mobile native apps) uses Javascript to send and receive messages from the server.

Connectivity

SignalR uses a couple of different technologies to support real time communication between client and server.

- WebSockets

The first approach it would try for connection is WebSockets because its the most efficient one.

- Server Sent Events

If this doesn't work(say because of browser compatibility issues), then it would try falling back to "Server Sent Events(SSE)". SSE is an HTML5 specification where the server can sent updates to the client using an HTTP connection. http://en.wikipedia.org/wiki/Server-sent_events

- Iframe

This method uses a hidden iframe in the browser and server keeps sending script blocks to the iframe. The scripts have knowledge to maintain the connection to the server.

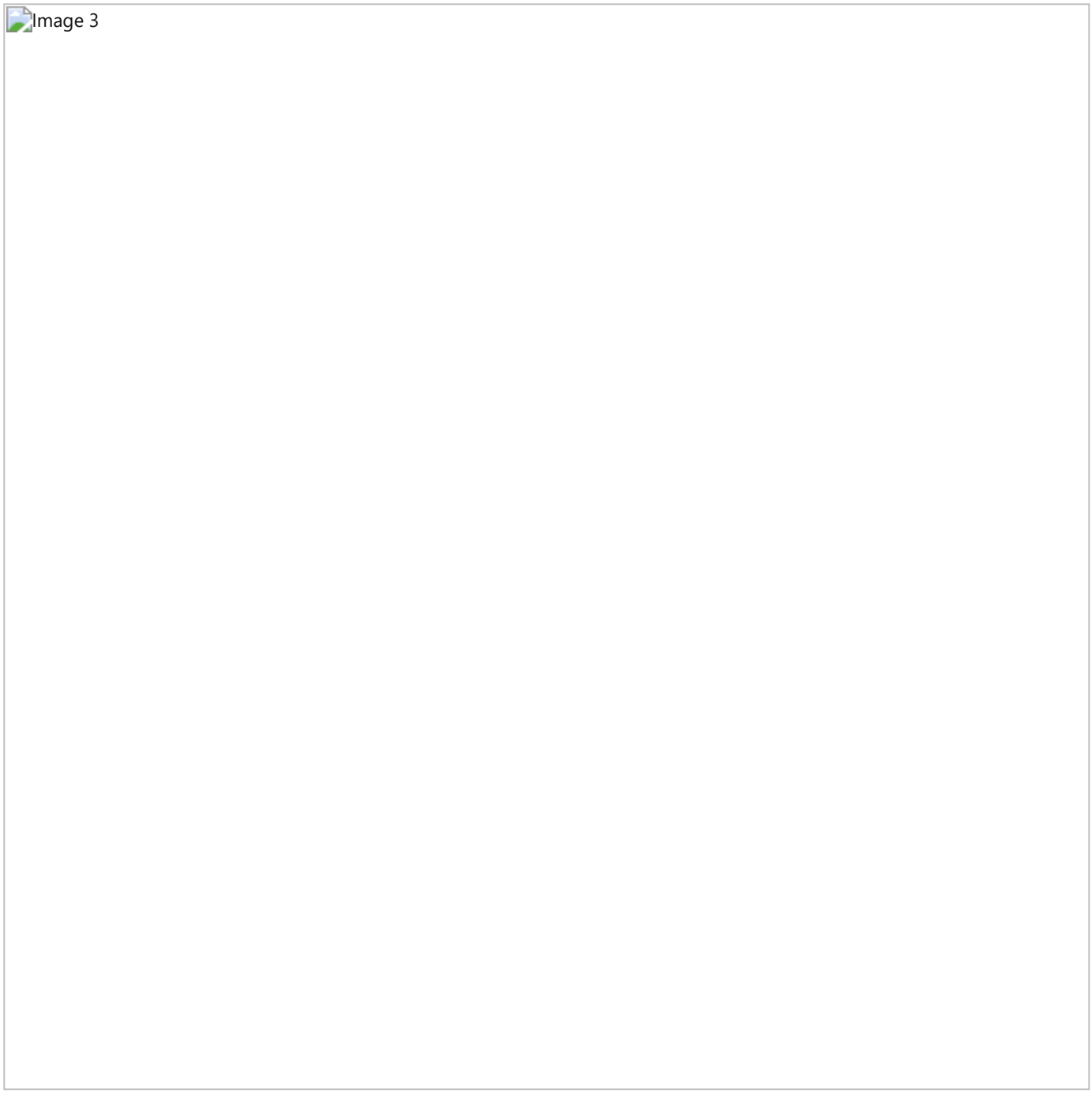
Read about Hidden iframe here http://en.wikipedia.org/wiki/Comet_(programming)

- Long polling

We have already read about Long polling mechanism in this history section above.

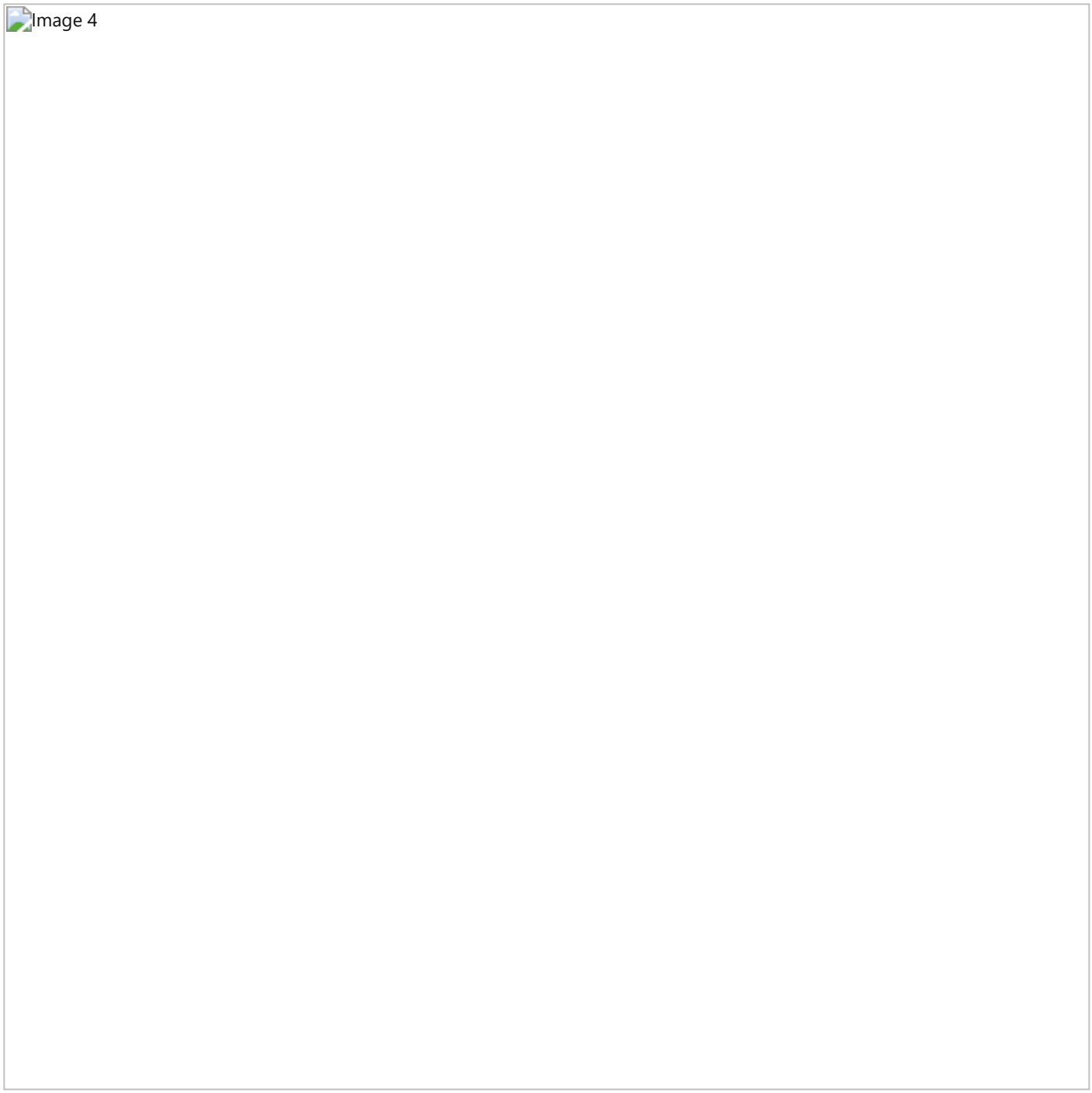
So in this way SignalR based on the infrastructure will try to achive real time communication in the most efficient way and fallback to another one if needed.

Web Browser requirements -- taken from http://www.asp.net/signalr/overview/signalr-20/getting-started-with-signalr-20/supported-platforms



Lets get practical

1. Create a new ASP.NET MVC5 web project. Select the MVC template and set Authentication to "Individual User Accounts". You are actually free to use any authentication you want but selecting "Individual User Accounts" gives us a Startup class which otherwise you would have to write yourself.



2. Install Nuget package Microsoft.AspNet.SignalR.

This nuget package contains both the server and client side dependencies needed to run SignalR.

3. After installing the Nuget package, it shows us a readme.txt file which mentions that to enable SignalR in our application, we have to do some OWIN/Katana configuration. If you are new to OWIN/Katana, you might want to read my article http://www.codeproject.com/Articles/805304/ASP-NET-MVC-Features

So basically, we have a Startup class in out project and a Configuration(IAppBuilder app) method. Now inside the ConfigureAuth class add this line of code to enable SignalR as shown in the code snippet end.

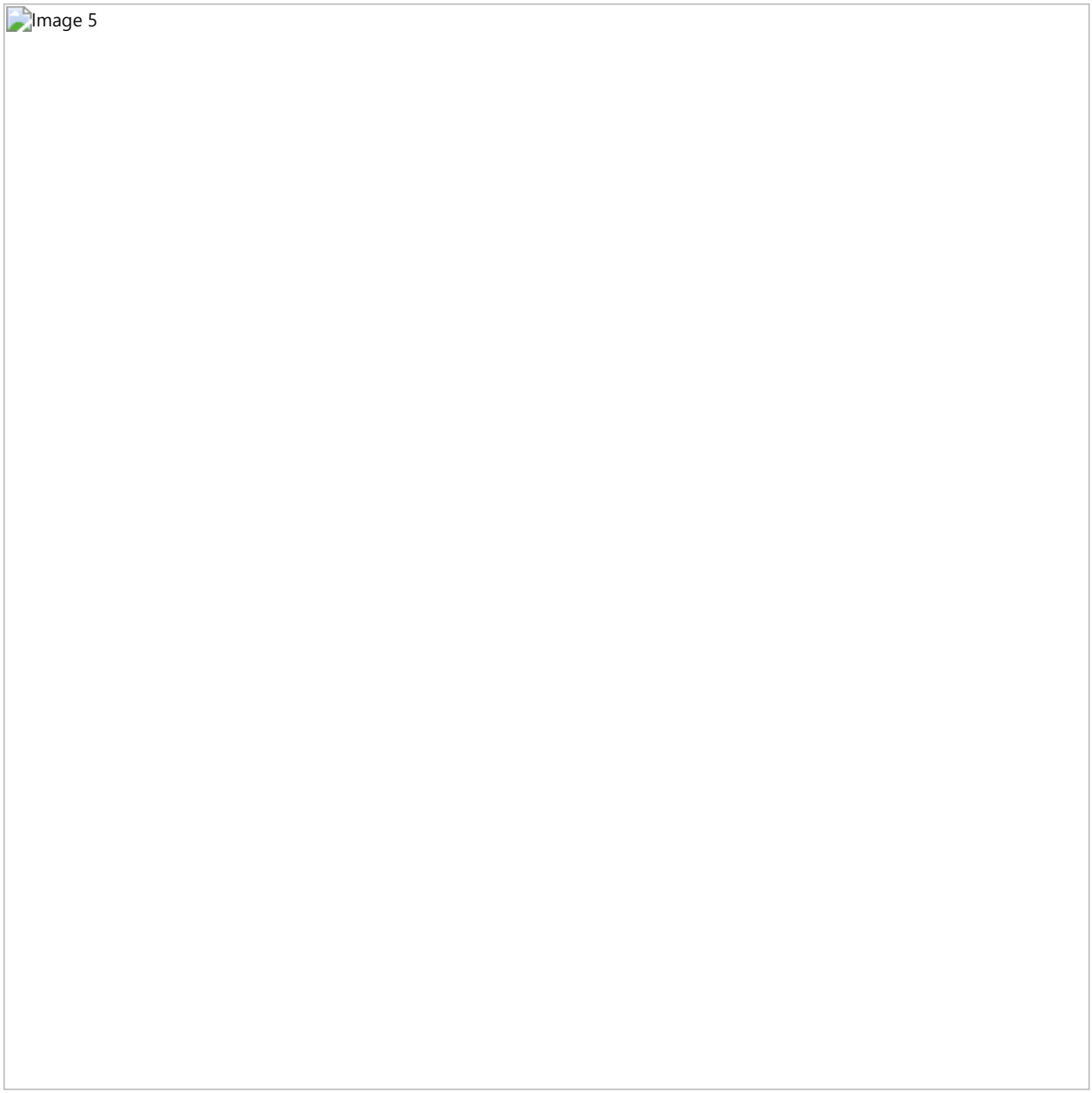
```
app.MapSignalR();
```

```
public partial class Startup
{
    // For more information on configuring authentication, please visit
```

```
http://go.microsoft.com/fwlink/?LinkId=301864
    public void ConfigureAuth(IAppBuilder app)
    {
        // Enable the application to use a cookie to store information for the signed in user
        app.UseCookieAuthentication(new CookieAuthenticationOptions
        {
            AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
            LoginPath = new PathString("/Account/Login")
        });
        // Use a cookie to temporarily store information about a user logging in with a third party
login provider
        app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

        //Enables SignalR
        app.MapSignalR();
```

4. In our project, Add a New item and select SignalR tab. We have 2 templates here. One SignalR hub class which we would select here. There is another template SingalR Persistent Connection Class which we can use if we want to do something at a connection level(a bit lower level)



So this MyHub class derives from the SignalR "Hub" class which exposes a public method Hello() which the clients can call to using WebSockets etc.

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

Inside the Hub's Hello method, we are doing Clients.All.hello(); which means the for all clients that are connected, call the hello() method on the client(a Javascript method).

Please note that "All" is a dynamic property of Clients and so we can call any method we want, e.g. in this case instead of hello() method we can call any other method say helloworld().

Clients.All.helloworld();

Ok, so now lets write some SignalR code which will broadcast the server time every 3 seconds to the clients, a simple example.

For that, in the above MyHub class, lets write a constructor and create a long running task in an infinite loop and send the server time to all the connected clients by calling the client method(a Javascript method on the client called from C# server side) after a delay of every 3 seconds. I am using TPL for this to make it asynchronous. If you are new to TPL and would like to know more, please visit the following link.

http://msdn.microsoft.com/en-us/library/dd537609(v=vs.110).aspx

```
        public MyHub()
        {
      // Create a Long running task to do an infinite loop which will keep sending the server time
            // to the clients every 3 seconds.
            var taskTimer = Task.Factory.StartNew(async () =>
                {
                    while(true)
                    {
                        string timeNow = DateTime.Now.ToString();
      //Sending the server time to all the connected clients on the client method SendServerTime()
                        Clients.All.SendServerTime(timeNow);
                        //Delaying by 3 seconds.
                        await Task.Delay(3000);
                    }
                }, TaskCreationOptions.LongRunning
                );
        }
```

Now that we have a hub defined and wrote the server code to send the server time to the clients, lets try writing some client side code.

We'll need a view to display the server time. I'll use the Contact view from the default ASP.NET MVC5 template project and delete its contents to keep it clean. Please feel free to use any view :).

cshtml View code

```
@section scripts{
    <script src="~/Scripts/jquery.signalR-2.1.1.js"></script>
    <script src="~/SignalR/hubs"></script>
    <script src="~/Scripts/Custom/timer.js"></script>
}
<span id="newTime"></span>
```

In the view snippet above, we have referenced the signalR jquery library jquery.signalR-2.1.1.js and a custom javascript file timer.js which we'll use to write our own custom code to define the connection to the hub and the client side methods.

The script in the middle <script src="~/SignalR/hubs"> </script> is not really a script but an endpoint exposed by the hub for the clients to consume. Its a kind of proxy on the client similar to a WCF proxy for those who have used WCF. We also need the jquery library but i haven't included it here as its alreay there in the _layout.cshtml view which this view inherits.

And we have a span defined with an id so that we can use Jquery to write the server send time there in the span which we'll see later.

Lets browse the url http://localhost:29511/SignalR/hubs to have a look at the proxy contents.

Here in the proxy code, we can see that, we have the myHub proxy which we defined as a hub in the server side.

We also see the hello() function defined on the server side here.

The last piece of code left to be written is the timer.js Javscript code.

```
(function () {
    // Defining a connection to the server hub.
    var myHub = $.connection.myHub;
    // Setting logging to true so that we can see whats happening in the browser console log.
[OPTIONAL]
    $.connection.hub.logging = true;
    // Start the hub
    $.connection.hub.start();

    // This is the client method which is being called inside the MyHub constructor method every 3
seconds
```

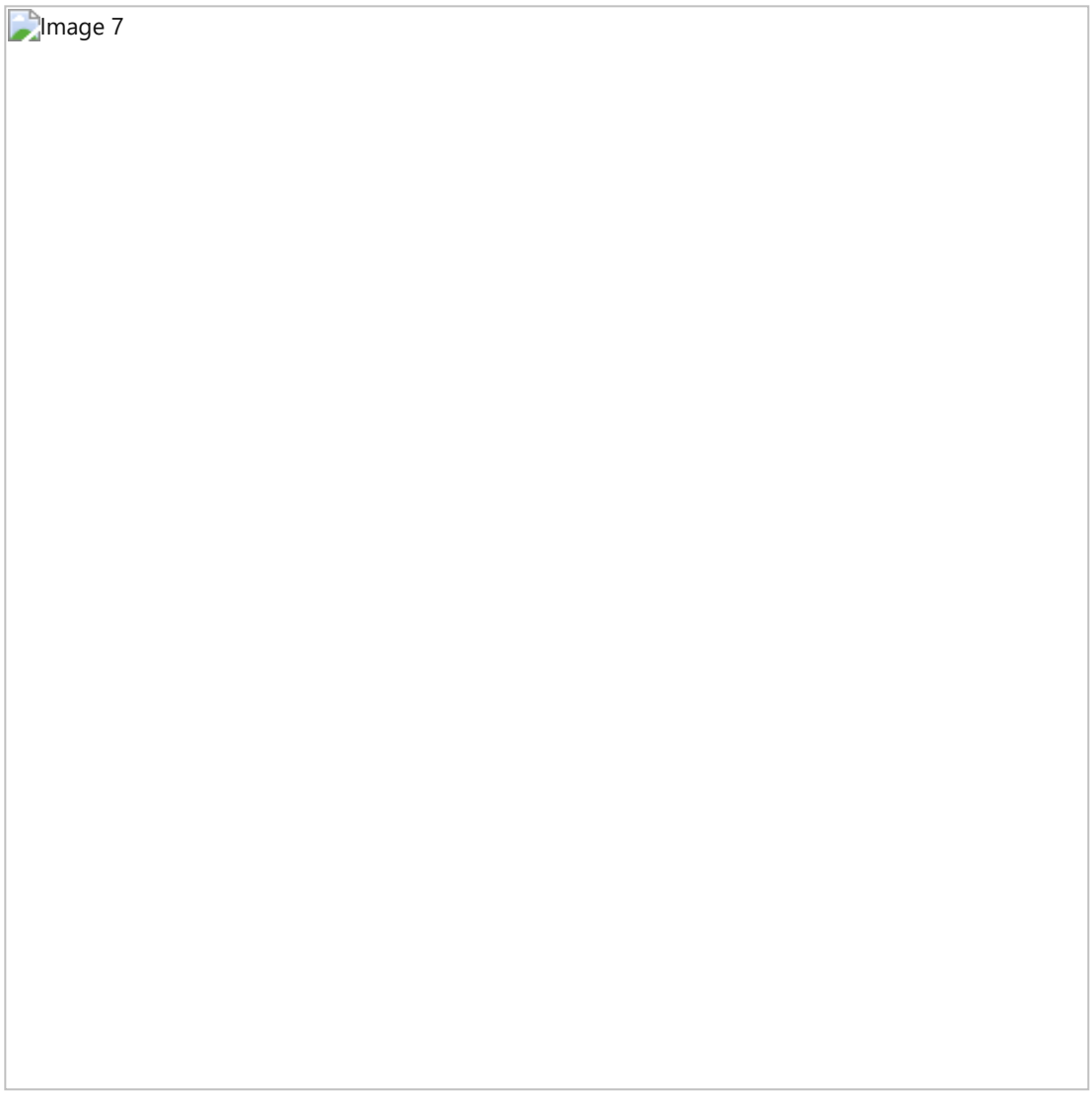
```
    myHub.client.SendServerTime = function (serverTime) {
        // Set the received serverTime in the span to show in browser
        $("#newTime").text(serverTime);
    };
}());
```

In the Javascript code snippet above, we have written an anonymous self executing Javascript method(self executing means we don't have to call it explicitly) where we defined the connection to the hub, started the hub and also wrote the client side method which the server hub calls.

To know more about Javascript self executing function(technically termed IIFE-Immediately-invoked function expression), please visit http://en.wikipedia.org/wiki/Immediately-invoked_function_expression

We are using the connection property on the $ object. Yes $ sounds like JQuery and the signalR client side library is actually a jquery plugin which makes this possible. Using the connection property we can access to the server side hubs.

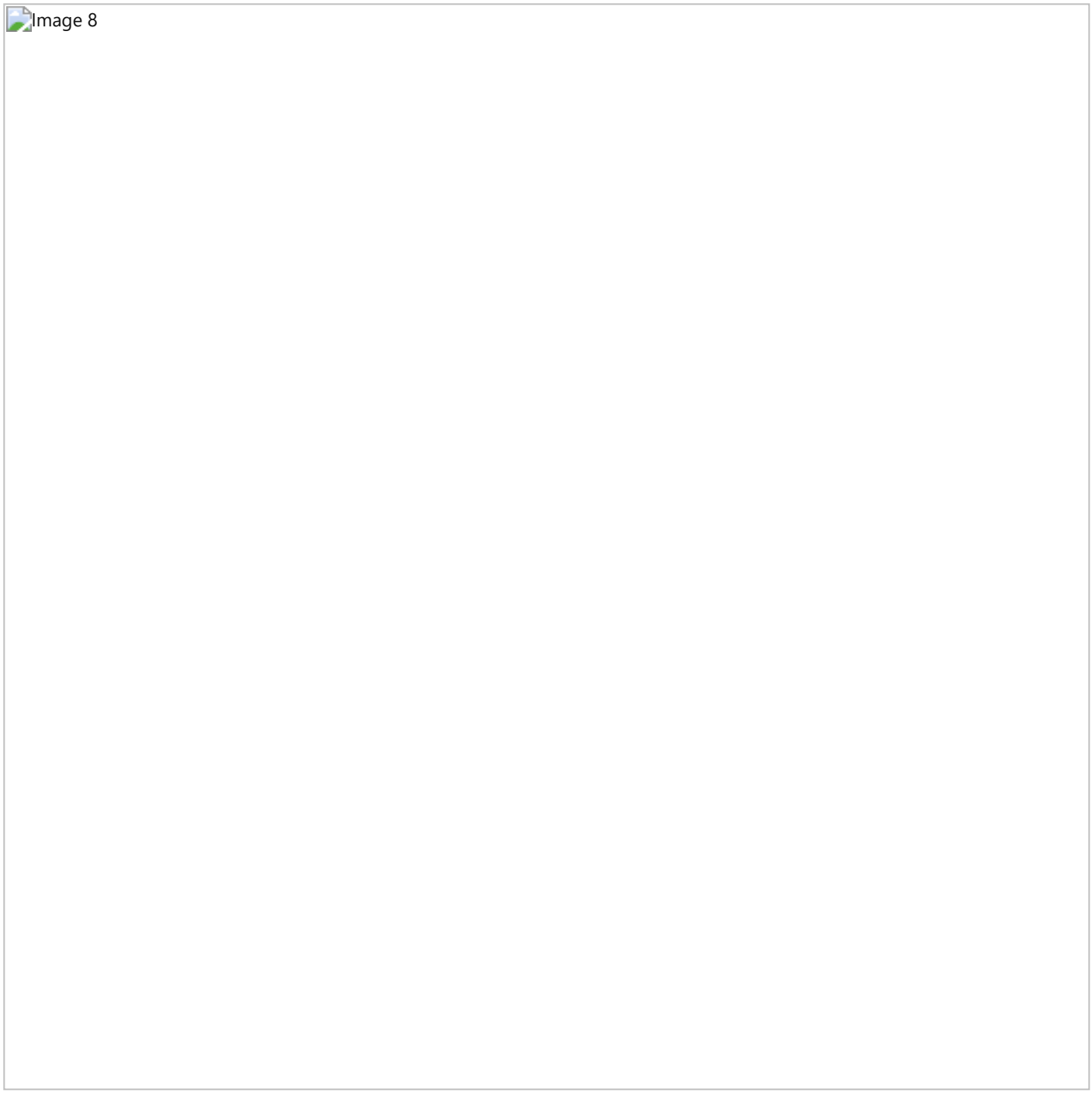
Now we are ready to run our program, excited!!!



Here we can see that the time keeps updating every 3 seconds. Try spinning up multiple browsers and see that the time gets updated for all the browsers. So its SignalR broadcasting the server time as a message to all connected clients.

If you remember that we had set logging to true in our client side javascript $.connection.hub.logging = true;.

So if you open up the Console window of the browser(F12 key for most of the ones) we will see the communication as shown in the snapshot below. In this case, its using WebSockets for communication as I am using the latest version of Chrome, but if you use an older browser, the communication may actually fall back to something else.



The last piece of thing I wanted to show is the client making a call to a server function defined in the Hub. This would show the true piece of bi-directional communication between the client and server.

We'll try sending a message to all connected clients on a button click. So a button click method on the client will call a server hub method which inturn will call a client method to all its connected clients. So one client will essentially call the server method which will broadcast to all the clients.

```
public class MyHub : Hub
{
    public void HelloServer()
    {
        Clients.All.hello("Hello message to all clients");
    }
}
```

So here we have a HelloServer() method defined in our hub which sends a message to all the clients.

Lets add a button and another span to our view. Button to click and span to display the message.

```
@section scripts{
    <script src="~/Scripts/jquery.signalR-2.1.1.js"></script>
    <script src="~/SignalR/hubs"></script>
    <script src="~/Scripts/Custom/timer.js"></script>
}
<span id="newTime"></span><br />
<input type="button" id="btnClick" value="Send Message" /><br />
<span id="message"></span>
```

Now lets change our timer.js file to include 2 more functions.

1. Client method hello() which is called from server hub as shown below.

Clients.All.hello("Hello message to all clients");

2. Button click handler which calls server hub method helloServer().

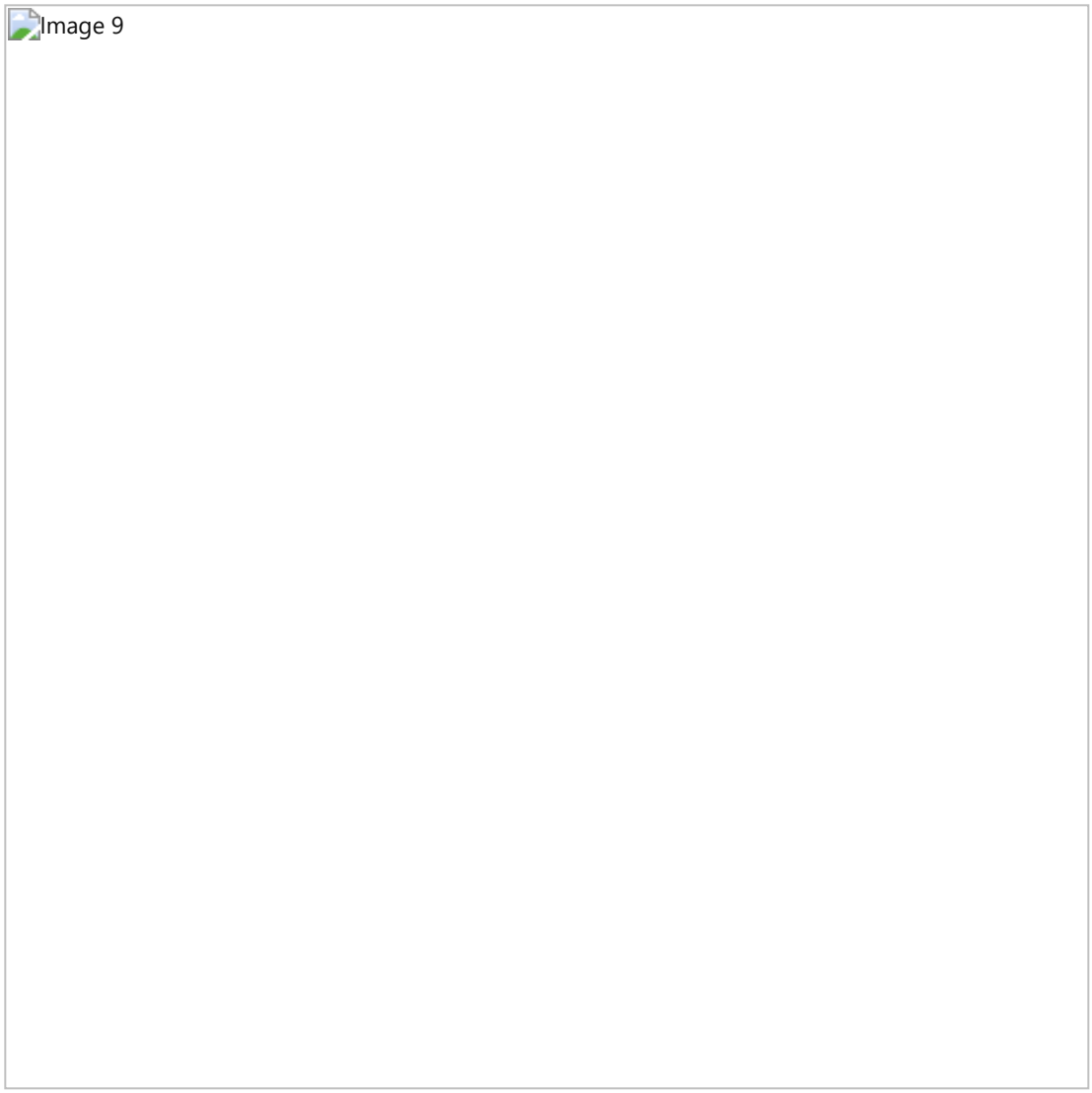
```
(function () {
    // Defining a connection to the server hub.
    var myHub = $.connection.myHub;
    // Setting logging to true so that we can see whats happening in the browser console log.
[OPTIONAL]
    $.connection.hub.logging = true;
    // Start the hub
    $.connection.hub.start();

    // This is the client method which is being called inside the MyHub constructor method every 3
seconds
    myHub.client.SendServerTime = function (serverTime) {
        // Set the received serverTime in the span to show in browser
        $("#newTime").text(serverTime);
    };

    // Client method to broadcast the message
    myHub.client.hello = function (message) {
        $("#message").text(message);
    };

    //Button click jquery handler
    $("#btnClick").click(function () {
        // Call SignalR hub method
        myHub.server.helloServer();
    });
}());
```

With this code in place now if we open up 2 or more browser windows and click on the button on one browser, all the browsers will receive the message real time. We can easily tweak this code to actually make a chat room by sending a proper message written by one client instead of a hardcoded message sent by server. So this is a brief introduction of the possibilities of SignalR.



History

Some of my other articles which you might be interested.

http://www.codeproject.com/Articles/805931/ASP-NET-MVC-Features-WebAPI

http://www.codeproject.com/Articles/805304/ASP-NET-MVC-Features

Download sample

License

About the Author

Rajiv Gogoi

Technical Lead Wipro Technologies

India

Currently working with Wipro Technologies as a Technical Consultant. I am primarily a .NET developer/consultant with experience in ASP.NET, ASP.NET MVC 3/4/5, WCF, Windows Azure,Knockout, AngularJS, NodeJS etc. I am interested in keeping myself updated with emerging technologies.