Lab SignalR

Andrea Dottor – Microsoft MVP ASP.NET/IIS – www.dottor.net – andrea@dottor.net
XeDotNet 07 Novembre 2013

Indice

Introduzione

Questo lab è composto da 4 esercitazioni che possono essere svolte in modo del tutto indipendente l'una dall'altra.

Si dovrà comunque iniziare dalla sezione "Codice comune a tutti gli esercizi" e passare successivamente all'esercitazione di proprio interesse.

Il lab comprende:

- ESERCIZIO 1
  Chat semplice. Ogni messaggio viene inviato a tutti gli utenti collegati.
- ESERCIZIO 2
  Chat con stanze. Ogni messaggio viene inviato ai soli utenti presenti in una determinata stanza/gruppo
- ESERCIZIO 3
  Invio di messaggi tra browser dello stesso utente. Invio di un oggetto complesso che identifica mittente, destinatario e messaggio.
- ESERCIZIO 4
  Invio di messaggi da server verso client. Al variare delle informazioni recuperate da un thread nel server, i client vengono aggiornati con i nuovi dati.

Il codice contenuto nello zip **Dottor.LabSignalR.Web____step-0-iniziale.zip** comprende già i passaggi elencati nella sezione "codice comune a tutti gli esercizi"

Nel file **Dottor.LabSignalR.Web____step-4-finale.zip** è presente il codice completo di questo lab, con tutti gli esercizi completi e funzionanti.

Codice comune a tutti gli esercizi

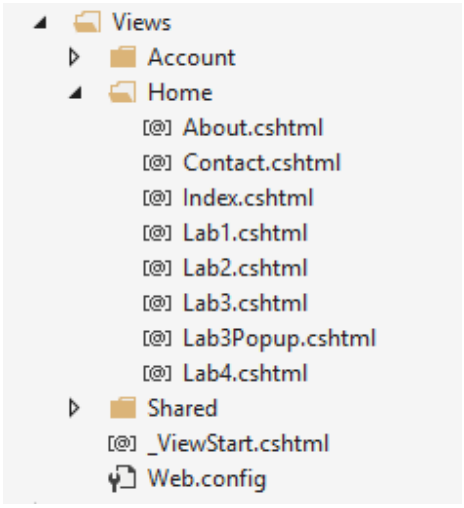
1. Creare un progetto ASP.NET MVC 4 o 5.

2. Aggiungere tramite NuGet il package di SignalR.
   Se l'applicativo utilizza il framework 4.5 è possibile installare l'ultima versione di SignalR direttamente dall'interfaccia grafica oppure digitando nella Package Manager Console "**Install-Package Microsoft.AspNet.SignalR**".
   Per chi invece utilizza Visual Studio 2010 oppure ha scelto una versione precedente del framework, dovrà referenziare la versione 1.1.3 di SignalR tramite il comando "**Install-Package Microsoft.AspNet.SignalR -Version 1.1.3**"

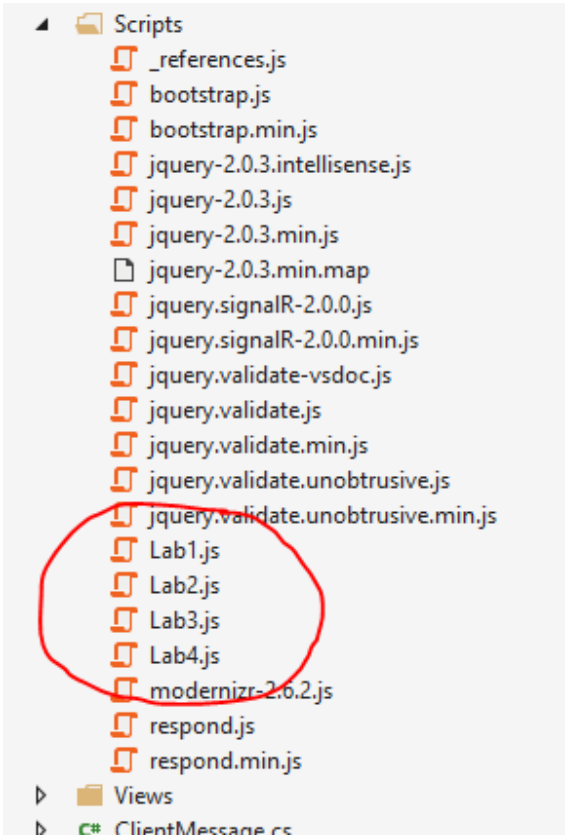
3. Nel HomeController aggiungere i metodi: Lab1, Lab2, Lab3, Lab3Popup e Lab4

```
public ActionResult Lab1()
{
    return View();
}

public ActionResult Lab2()
{
    return View();
}

public ActionResult Lab3()
{
    return View();
}

public ActionResult Lab3Popup()
{
    return View();
}

public ActionResult Lab4()
{
    return View();
}
```

4. All'interno della cartella Views/Home creare le view con nome Lab1, Lab2, Lab3, Lab3Popup e Lab4



5. Nella cartella Scripts creare un file JavaScript per ogni esercizio.



6. All'interno della View di layout, aggiungiamo nel menu della pagina i link alle action corrispondenti ai vari esercizi.
   La View di layout è presente in Views/Shared/_Layout.cshtml.
   Per creare i link:
   @Html.ActionLink("LAB 1", "Lab1", "Home")

7. Referenziare nelle View il rispettivo file JavaScript, la libreria di SignalR e la libreria degli hub.
   Per fare in modo che i file JavaScript dell'esercizio vengano caricati successivamente a quello di jQuery, li andremo ad inserire nelle sezione "scripts".
   [NOTE: Nella view Lab3Popup referenziare il file Lab3.js]

```
@section scripts {
    @Scripts.Render("~/Scripts/jquery.signalR-2.0.0.js")
    @Scripts.Render("~/signalr/hubs")
    @Scripts.Render("~/Scripts/Lab1.js")
}
```

8. Nella classe Startup.cs richiamare il metodo che si occupa di configurare SignalR (e le relative regole di routing, andando a modificare la classe nel seguente modo:

```
using Microsoft.Owin;
using Owin;

[assembly: OwinStartupAttribute(typeof(Dottor.LabSignalR.Web.Startup))]
namespace Dottor.LabSignalR.Web
{
    public partial class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            ConfigureAuth(app);
            app.MapSignalR();
        }
    }
}
```

   Nel caso si trattasse di un progetto ASP.NET MVC 4 o non fosse presente la classe Startup.cs, modificare il metodo Application_Start nel Global.asax nel seguente modo:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace Dottor.LabSignalR.Web
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            RouteTable.Routes.MapHubs();
```

```
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

9. Creare in root all'applicazione una cartella **Hubs** che andremo ad utilizzare per inserire tutti gli hub di SignalR che creeremo nel corso del laboratorio.

ESERCIZIO 1

Nel seguente esercizio realizzeremo una semplice chat dove i messaggi di ogni utente verranno inviati a tutti gli utenti collegati.

In input avremo due campi di testo: il nome dell'utente ed il messaggio che si vuole inviare.

L'elenco dei messaggi verrà renderizzato tramite JQuery all'interno di un apposito elemento ul.

1. All'interno della cartella Hubs creiamo un nuovo elemento di tipo "SignalR Hub Class (v2)", oppure nel caso non fosse presente il templace, creiamo una classe e la nominiamo **ChatHub.cs**.

2. Creiamo un metodo Send che si occupi di inviare il messaggio a tutti gli utenti collegati:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;
using System.Threading.Tasks;

namespace Dottor.LabSignalR.Web.Hubs
{
    public class ChatHub : Hub
    {
        /// <summary>
        /// Invio di un messaggio a tutti gli utenti collegati
        /// </summary>
        /// <param name="name">Nome dell'utente</param>
        /// <param name="message">Messaggio da inviare</param>
        public void Send(string name, string message)
        {
            Clients.All.broadcastMessage(name, message);
        }
    }
}
```

3. Nella view Lab1.cshtml creo il codice necessario per recuperare dall'utente il nome ed il messaggio da inviare, un pulsante ed un elemento "ul" che conterrà i messaggi della chat.

```
<div class="container">
    Nome utente: <br />
    <input type="text" id="displayname" /><br />
    Messaggio: <br />
    <input type="text" id="message" />
    <input type="button" id="sendmessage" value="Invia" />

    <ul id="chatMessages"></ul>
</div>
```

4. Nel file JavaScript Lab1.js che abbiamo precedentemente collegato alla view, andiamo ad aggiungere il codice necessario all'invio dei messaggi verso l'hub:

```
$(function () {
    // Referenza all'hub 'chat'
    var chat = $.connection.chatHub;

    // Avvio la connessione verso il server
    $.connection.hub.start().done(function () {
        $('#sendmessage').click(function () {
            // Chiamo il metodo Send dell'hub per inviare un messaggio
            chat.server.send($('#displayname').val(), $('#message').val());
            // Svuoto la casella di testo
            $('#message').val('').focus();
        });
    });
});
```

5. Nel file JavaScript Lab1.js, all'interno della function eseguita al caricamento della pagina, andiamo ad aggiungere il codice necessario alla gestione della ricezione dei messaggi (dopo aver recuperato il riferimento all'hub corrente):

```
    // Funzione che viene invocata alla ricezione di un messaggio
    chat.client.broadcastMessage = function (name, message) {
        // Html encode display name and message.
        var encodedName = $('<div />').text(name).html();
        var encodedMsg = $('<div />').text(message).html();
        // Add the message to the page.
        $('#chatMessages').append('<li><strong>' + encodedName
            + '</strong>:  ' + encodedMsg + '</li>');
    };
```

6. Avviare l'applicazione e cliccare alla voce LAB 1 del menu per eseguire l'esercizio appena concluso.

ESERCIZIO 2

L'esercizio 2 è una versione più avanzata della chat creata nell'esercizio precedente, in quanto permette il raggruppamento degli utenti a seconda della stanza della chat che selezionano.

1. Modificare il ChatHub andando ad aggiungere i metodi che permettono l'ingresso e l'uscita degli utenti dalle stanze, ed il metodo di invio dei messaggi ai soli utenti appartenenti ad una precisa stanza.

```
        /// <summary>
        /// Ingresso di un utente in una stanza
        /// </summary>
        /// <param name="roomName">Nome della stanza</param>
        /// <returns></returns>
        public Task JoinRoom(string roomName)
        {
            return Groups.Add(Context.ConnectionId, roomName);
        }

        /// <summary>
        /// Uscita di un utente da una stanza
        /// </summary>
        /// <param name="roomName">Nome della stanza</param>
        /// <returns></returns>
        public Task LeaveRoom(string roomName)
        {
            return Groups.Remove(Context.ConnectionId, roomName);
        }

        /// <summary>
        /// Invio di un messaggio a tutti gli utenti presenti nella stanza
        /// </summary>
        /// <param name="roomName">Nome della stanza</param>
        /// <param name="name">Nome dell'utente</param>
        /// <param name="message">Messaggio da inviare</param>
        public void SendToRoom(string roomName, string name, string message)
        {
            Clients.Group(roomName).addChatMessage(name, message);
        }
```

2. Nella view Lab2.cshtml creare del codice html simile a quello dell'esercizio precedente, ma con l'aggiunta della possibilità di scelta della stanza di appartenenza.

```
<div class="container">
    Nome utente:<br />
    <input type="text" id="displayname" /><br />

    Stanza:<br />
    <select id="chatroom">
        <option value="xedotnet" selected="selected">XeDotNet</option>
        <option value="aspnet">ASP.NET</option>
        <option value="cs">C#</option>
    </select><br />

    Messaggio:<br />
    <input type="text" id="message" />
    <input type="button" id="sendmessage" value="Invia" />

    <ul id="chatMessages"></ul>
</div>
```

3. Nel file JavaScript Lab2.js inserire il seguente codice:
   Il metodo collegato al "addCahtMessage" è identico a quello dell'esercizio 1. A cambiare è la gestione dell'evento "change" della select che permette la gestione dell'ingresso ed uscita dalle stanze.
   A differenza dell'esercizio precedente, vediamo che nel metodo di invio del messaggio, andiamo a passare anche l'informazione relativa alla stanza corrente (variabile groupName), informazione che viene passata anche all'avvio della chat, recuperando il valore di default della select.

```
$(function () {
    // Referenza all'hub 'chat'
    var chat = $.connection.chatHub;
    var groupName;
    // Funzione che viene invocata alla ricezione di un messaggio
    chat.client.addChatMessage = function (name, message) {
        // Html encode display name and message.
        var encodedName = $('<div />').text(name).html();
        var encodedMsg = $('<div />').text(message).html();
        // Aggiungo il messaggio alla pagina
        $('#chatMessages').append('<li><strong>' + encodedName
            + '</strong>:  ' + encodedMsg + '</li>');
    };

    // Entro nella stanza e lo notifico all'hub
    $.connection.hub.start(function () {
        groupName = $('#chatroom').val();
        chat.server.joinRoom(groupName);
    });

    // Gestione del cambio di stanza
    $('#chatroom').on('change', function () {
        // Lascio la vecchia stanza
        chat.server.leaveRoom(groupName);
        groupName = $('#chatroom').val();
        // Entro nella nuova stanza
```

```
        chat.server.joinRoom($('#chatroom').val());
    });

    // Avvio la connessione verso il server
    $.connection.hub.start().done(function () {
        $('#sendmessage').click(function () {
            // Chiamo il metodo Send dell'hub per inviare un messaggio
            chat.server.sendToRoom(
                groupName,
                $('#displayname').val(),
                $('#message').val());
            // Svuoto la casella di testo
            $('#message').val('').focus();
        });
    });
});
```

4. Avviare l'applicazione e cliccare alla voce LAB 2 del menu per eseguire l'esercizio appena concluso.

ESERCIZIO 3

In questo esercizio vedremo come utilizzare SignalR per inviare messaggi tra browser dello stesso utente.
Quello che faremo sarà memorizzare lato server tutte le ConnectioId associandole allo username dell'utente collegato. Ogni finestra di ogni utente ha una ConnectionId differente. Dovremo quindi occuparci manualmente della gestione delle connessioni, e del relativo inoltro dei messaggi.

1. All'interno della cartella Hubs creiamo un nuovo elemento di tipo "SignalR Hub Class (v2)", oppure nel caso non fosse presente il templace, creiamo una classe e la nominiamo **BrowserMessagesHub.cs**.
2. Marcare il BrowserMessagesHub con l'attributo [Authorize] in modo da forzare che possa venir chiamato dai soli utenti autenticati
3. Marcare il metodo Lab3 nell'HomeController con l'attributo [Authorize] in modo che la pagina sia accessibile dai soli utenti autenticati
4. In root all'applicazione creiamo una nuova classe chiamata UserConnections.cs che ci permetterà di memorizzare tutte le ConnectionId di un preciso utente.

```
namespace Dottor.LabSignalR.Web
{
	using System.Collections.Generic;

	/// <summary>
	/// Connessioni attive per singolo utente.
	/// La chiave dell'oggetto è il SessionId
	/// </summary>
	public class UserConnections
	{
		/// <summary>
		/// Nome dell'utente.
		/// Utile per inviare i messaggi a tutti i browser dell'utente.
		/// </summary>
		public string UserName { get; set; }

		/// <summary>
		/// Elenco delle connessioni attive per SessionId corrente
		/// </summary>
		public HashSet<string> ConnectionIds { get; set; }
	}
}
```

5. In root all'applicazione creiamo una nuova classe chiamata ClientMessage.cs che rappresenta il messaggio che transiterà tra le varie finestre dei browser.

```
namespace Dottor.LabSignalR.Web
{
	/// <summary>
	/// Messaggio che viene inviato da client a client oppure da server a client.
	/// </summary>
	/// <remarks>
	/// Viene inviato/ricevuto tramite SignalR
	/// </remarks>
	public class ClientMessage
	{
		/// <summary>
		/// Nome o chiave di chi invia il messaggio
		/// </summary>
		public string from { get; set; }

		/// <summary>
		/// Nome o chiave di chi riceve il messaggio
		/// </summary>
		public string to { get; set; }

		/// <summary>
		/// Tipologia di messaggio/azione
		/// </summary>
		public string action { get; set; }

		/// <summary>
		/// Corpo del messaggio
		/// </summary>
		public string message { get; set; }
	}
}
```

6. Nel hub BrowserMessagesHub inserire il codice necessario alla gestione delle connessioni e all'invio ed inoltro dei messaggi.
   Nei metodi OnConnected ed OnDisconnected vengono gestite le associazioni tra username e ConnectionId.
   Nel metodo SendToPages viene gestito l'inoltro dei messaggi verso le ConnectionId dello stesso utente.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;
using System.Threading.Tasks;
using System.Collections.Concurrent;

namespace Dottor.LabSignalR.Web.Hubs
{
    //[Authorize]
    public class BrowserMessagesHub : Hub
    {
        /// <summary>
```

```
        /// Elenco degli utenti con connessioni attive
        /// </summary>
        private static readonly ConcurrentDictionary<string, UserConnections> Users = new
ConcurrentDictionary<string, UserConnections>();

        /// <summary>
        /// Metodo invocato alla connessione di un client.
        /// Viene utilizzato per salvare il connectionId di tutte le connessioni aperte
        /// </summary>
        /// <returns></returns>
        public override Task OnConnected()
        {
            string userName = Context.User.Identity.Name;
            string connectionId = Context.ConnectionId;

            UserConnections user = Users.GetOrAdd(userName, _ => new UserConnections
            {
                UserName = userName,
                ConnectionIds = new HashSet<string>()
            });

            lock (user.ConnectionIds)
            {
                if (!user.ConnectionIds.Contains(connectionId))
                    user.ConnectionIds.Add(connectionId);
            }

            return base.OnConnected();
        }

        /// <summary>
        /// Metodo invocato alla discnonessione dei client.
        /// Utilizzato per rimuovere i connectionId non più utilizzati.
        /// </summary>
        /// <returns></returns>
        public override Task OnDisconnected()
        {
            string userName = Context.User.Identity.Name;
            string connectionId = Context.ConnectionId;

            UserConnections user;
            if (Users.TryGetValue(userName, out user))
            {
                lock (user.ConnectionIds)
                {
                    user.ConnectionIds.RemoveWhere(cid => cid.Equals(connectionId));

                    if (!user.ConnectionIds.Any())
                    {
                        UserConnections removedUser;
                        Users.TryRemove(userName, out removedUser);
                    }
                }
            }

            return base.OnDisconnected();
        }

        /// <summary>
        /// Riceve il messaggio da una pagina, e si occupa di inoltrarlo
```

```
        /// alle evenutali altre pagine/browser aperti dallo stesso utente
        /// </summary>
        /// <param name="message">Messaggio da inviare</param>
        public void SendToPages(ClientMessage message)
        {
            string userName = Context.User.Identity.Name;
            string connectionId = Context.ConnectionId;

            UserConnections sender;
            if (Users.TryGetValue(userName, out sender))
            {
                IEnumerable<string> connections;

                lock (sender.ConnectionIds)
                {
                    // Invio da tutte le connessioni tranne a chi ha inviato il
messaggio,
                    // quindi rimuovo la connessione corrente
                    //
                    connections = sender.ConnectionIds.Except(new[] { connectionId
}).ToList();
                }
                // Invio il messaggio solo alle altre finestre
                //
                foreach (var cid in connections)
                    Clients.Client(cid).pageNewMessage(message);
            }
        }


    }
}
```

7. Nella view Lab3.cshtml inserire il seguente codice:

```
<div class="container">
    Azione: <br />
    <select id="ddlAction">
        <option value="color" selected="selected">Cambia colore</option>
        <option value="background-color">Cambia colore di sfondo</option>
        <option value="border">Modifica bordi</option>
    </select><br />

    Valore: <br />
    <input type="text" id="txtValue" /><br />

    <input type="button" id="sendmessage" value="Esegui" />
    <br />
    <br />
    <input type="button" id="openpopup" value="Apri finestra in popup" />
</div>
```

8. Nella view Lab3Popup.cshtml inserire il seguente codice:

```
<div id="pnlDestination">
    <p>
        Lorem ipsum dolor sit amet, et eum prima dictas, ad idque labore copiosae duo.
Eam prima similique reformidans eu. Essent similique disputationi sea te, no per mollis
perfecto. Pri adhuc mazim dignissim at, nam ornatus vituperata reprehendunt eu.
    </p>
    <p>
        Nam ei veniam laoreet persecuti, sumo agam fuisset an mel. Quo deserunt torquatos
ad, pri ad debet mandamus comprehensam, quem iudicabit dissentiet his cu. Sed alii causae
et, quod elaboraret mel eu, ex tale nostrum expetenda vel. Ius erat aliquam tacimates et,
vis quem theophrastus necessitatibus at. Noster tractatos nec cu. Et iudico exerci
invenire sit, ex eam dicam invenire reprehendunt.
    </p>
</div>
```

9. Nel file JavaScript Lab3.js inserire il seguente codice:

```
$(function () {
    // Referenza all'hub 'browserMessages'
    var hub = $.connection.browserMessagesHub;

    // Funzione che viene invocata alla ricezione di un messaggio
    hub.client.pageNewMessage = function (message) {
        $('#' + message.to).css(message.action, message.message);
    };

    // Avvio della connessione verso l'hub
    $.connection.hub.start().done(function () {
        console.log("connection started!");
    });

    // Invio di un messaggio verso il server
    $('#sendmessage').click(function () {
        var msg = new Object();
        msg.from = 'Lab3';
        msg.to = 'pnlDestination';
        msg.action = $('#ddlAction').val();
        msg.message = $('#txtValue').val();

        hub.server.sendToPages(msg);
    });

    // Apertura di una finestra di popup
    $('#openpopup').click(function(){
        window.open('Lab3Popup');
    })

});
```

10. Avviare l'applicazione e cliccare alla voce LAB 3 del menu per eseguire l'esercizio appena concluso.
    a. Cliccare sul pulsante "Apri finestra in popup" per far aprire così la view Lab3Popup che sarà la destinataria dei messaggi.

b. Nell finestra del Lab3, selezionare uno stile da voler cambiare, ed impostare nel campo in input un valore.
   Es, per il “Cambia colore di sfondo” o il “Cambia colore” è possibile impostare un colore in esadecimale come #FF0000
   per il “Modifica bordi”, settare per esempio “10px solid #00FF00”
c. Cliccare nel pulsante Esegui per inviare il messaggio, e guardare il risultato nella pagina Lab3Popup

ESERCIZIO 4

Questo esercizio dimostra come far visualizzare sul client messaggi/informazioni in realtime recuperandole lato server in modo temporizzato. In questa esercitazione visualizzeremo i dati relativi ai processi in esecuzione nel server con i relativi dati di utilizzo della memoria.
Un esempio simile può essere eseguito recuperando e visualizzando i dati di titoli di borsa, oppure informazioni recuperate da feed rss (es: terremoti, blog, ….)

1. In root all'applicazione creare la classe **ServerProcess.cs** che rappresenta i dati di un processo che verranno inviati con SignalR verso i client.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Dottor.LabSignalR.Web
{
	public class ServerProcess
	{
		public int Id { get; set; }

		public string Name { get; set; }

		public long PrivateMemorySize { get; set; }

		public long VirtualMemorySize { get; set; }
	}
}
```

2. All'interno della cartella Hubs creiamo un nuovo elemento di tipo "SignalR Hub Class (v2)", oppure nel caso non fosse presente il templace, creiamo una classe e la nominiamo **ServerProcessHub.cs**.
3. In root all'applicazione creare la classe **ServerProcessesTicker.cs** che si occupa di gestire il recupero delle informazioni dei processi, e si occupa di notificare i nuovi dati ai client. All'interno della classe viene utilizzato un singleton per fare in modo di aver un unico timer che aggiorna tutti i client.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNet.SignalR;
using Microsoft.AspNet.SignalR.Hubs;
using System.Threading;
using System.Collections.Concurrent;
using System.Diagnostics;
using Dottor.LabSignalR.Web.Hubs;

namespace Dottor.LabSignalR.Web
{
	public class ServerProcessesTicker
	{
```

```
		// Singleton instance
		//
		private readonly static Lazy<ServerProcessesTicker> instance = new
Lazy<ServerProcessesTicker>(() => new
ServerProcessesTicker(GlobalHost.ConnectionManager.GetHubContext<ServerProcessHub>().Clie
nts));

		// Elenco di processi
		//
		private readonly ConcurrentDictionary<string, ServerProcess> processes =
new ConcurrentDictionary<string, ServerProcess>();

		// Intervallo di tick del timer
		//
		private readonly TimeSpan timerUpdateInterval =
TimeSpan.FromMilliseconds(500);
		private readonly Timer timer;

		// Oggetti utili per la gestione thread-safe delle operazioni
		//
		private volatile bool updatingProcesses = false;
		private readonly object updateProcessesLock = new object();

		private ServerProcessesTicker(IHubConnectionContext clients)
		{
			Clients = clients;

			RefreshData();

			timer = new Timer(UpdateProcesses, null, timerUpdateInterval,
timerUpdateInterval);
		}

		/// <summary>
		/// Singleton di ServerProcessesTicker
		/// </summary>
		public static ServerProcessesTicker Instance
		{
			get { return instance.Value; }
		}

		/// <summary>
		/// Clients collegati tramite SignalR
		/// </summary>
		private IHubConnectionContext Clients { get; set; }

		/// <summary>
		/// Ritorna i processi allo stato dell'ultimo aggiornamento
		/// </summary>
		/// <returns></returns>
		public IEnumerable<ServerProcess> GetRefreshedData()
		{
			return processes.Values;
		}

		/// <summary>
		/// Invio aggiornato dei dati sui client
		/// </summary>
		/// <param name="state"></param>
		private void UpdateProcesses(object state)
```

```
            {
                  lock (updateProcessesLock)
                  {
                        if (!updatingProcesses)
                        {
                              updatingProcesses = true;

                              RefreshData();

                              BroadcastProcesses(processes.Select(p =>
p.Value).OrderByDescending(p => p.VirtualMemorySize).ToArray());

                              updatingProcesses = false;
                        }
                  }
            }

            /// <summary>
            /// Recupero dei dati aggiornati a riguardo dei processi
            /// </summary>
            private void RefreshData()
            {
                  processes.Clear();

                  var currentProcesses = Process.GetProcesses();
                  foreach (var process in currentProcesses)
                  {
                        processes.TryAdd(
                              process.ProcessName,
                              new ServerProcess
                              {
                                    Name = process.ProcessName,
                                    PrivateMemorySize = process.PrivateMemorySize64,
                                    VirtualMemorySize = process.VirtualMemorySize64,
                                    Id = process.Id
                              });
                  }
            }

            /// <summary>
            /// Notifica dei dati aggiornati a tutti i client
            /// </summary>
            /// <param name="serverProcesses">Processi correnti</param>
            private void BroadcastProcesses(ServerProcess[] serverProcesses)
            {
                  Clients.All.updateProcesses(serverProcesses);
            }
      }
}
```

4. Nell'hub ServerProcessHub istanziare il timer appena creato all'interno del costruttore dell'hub, in modo da avviare l'aggiornamento dei client in modo automatico.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;

namespace Dottor.LabSignalR.Web.Hubs
{

	public class ServerProcessHub : Hub
	{
		private readonly ServerProcessesTicker ticker;

		public ServerProcessHub() : this(ServerProcessesTicker.Instance) { }

		public ServerProcessHub(ServerProcessesTicker processesTicker)
		{
			ticker = processesTicker;
		}

	}
}
```

5. All'interno della view Lab4.cshtml aggiungere il seguente codice html, che farà da placeholder per i dati che arriveranno da SignalR

```
<table id="tblProcesses" style="width:100%;">
    <thead>
        <tr>
            <th>Id </th>
            <th>Nome processo </th>
            <th>Private memory size [bytes] </th>
            <th>Virtual memory size [bytes] </th>
        </tr>
    </thead>
    <tbody>
        <tr class="loading">
            <td colspan="4">
                caricamento in corso...
            </td>
        </tr>
    </tbody>
</table>
```

6. Nel file JavaScript Lab4.js verranno gestiti i messaggi ricevuti dal server:

```
// A simple templating method for replacing placeholders enclosed in curly braces.
if (!String.prototype.supplant) {
    String.prototype.supplant = function (o) {
        return this.replace(/{([^{}]*)}/g,
                    function (a, b) {
                        var r = o[b];
                        return typeof r === 'string' || typeof r === 'number' ? r : a;
                    });
    };
}

$(document).ready(function () {
    var hub = $.connection.serverProcessHub;
    var rowTemplate =
'<tr><td>{Id}</td><td>{Name}</td><td>{PrivateMemorySize}</td><td>{VirtualMemorySize}</td>
</tr>';
    var table = $('#tblProcesses tbody');

    // Add a client-side hub method that the server will call
    hub.client.updateProcesses = function (processes) {
        table.empty();
        $.each(processes, function () {
            table.append(rowTemplate.supplant(this));
        });
    };

    // Start the connection
    $.connection.hub.start().done(function () {
        console.log('hub started');
    });
});
```

7. Avviare l'applicazione e cliccare alla voce LAB 4 del menu per eseguire l'esercizio appena concluso.