1. Đăng ký SignalR Middleware

using Microsoft.Owin;

using Owin;

[assembly: OwinStartup(typeof(MyApplication.Startup))]

namespace MyApplication

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

// Any connection or hub wire up and configuration should go here

app.MapSignalR();

}

}

}

**The /signalr URL**

**Server code that specifies the URL**

app.MapSignalR("/signalr", new HubConfiguration());

**JavaScript client code that specifies the URL (with the generated proxy)**

$.connection.hub.url = "/signalr"

$.connection.hub.start().done(init);

**JavaScript client code that specifies the URL (without the generated proxy)**

var connection = $.hubConnection("/signalr", { useDefaultPath: false });

**.NET client code that specifies the URL**

var hubConnection = new HubConnection("http://contoso.com/signalr", useDefaultUrl: false);

### Configuring SignalR Options

* Enable cross-domain calls using CORS or JSONP from browser clients.

Typically if the browser loads a page from http://contoso.com, the SignalR connection is in the same domain, at http://contoso.com/signalr. If the page from http://contoso.com makes a connection to http://fabrikam.com/signalr, that is a cross-domain connection. For security reasons, cross-domain connections are disabled by default. For more information, see [ASP.NET SignalR Hubs API Guide - JavaScript Client - How to establish a cross-domain connection](https://docs.microsoft.com/en-us/aspnet/signalr/overview/guide-to-the-api/hubs-api-guide-javascript-client#crossdomain).

* Enable detailed error messages.

When errors occur, the default behavior of SignalR is to send to clients a notification message without details about what happened. Sending detailed error information to clients is not recommended in production, because malicious users might be able to use the information in attacks against your application. For troubleshooting, you can use this option to temporarily enable more informative error reporting.

* Disable automatically generated JavaScript proxy files.

By default, a JavaScript file with proxies for your Hub classes is generated in response to the URL "/signalr/hubs". If you don't want to use the JavaScript proxies, or if you want to generate this file manually and refer to a physical file in your clients, you can use this option to disable proxy generation. For more information, see [SignalR Hubs API Guide - JavaScript Client - How to create a physical file for the SignalR generated proxy](https://docs.microsoft.com/en-us/aspnet/signalr/overview/guide-to-the-api/hubs-api-guide-javascript-client#manualproxy).

### Vd: trình diễn cách chỉ định một url kết nối signalR và những tùy chọn của nó trong một cuộc gọi đến phương thức MapSignalR. Để chỉ định một custom URL thay thế “/signalr” bằng một URL

var hubConfiguration = new HubConfiguration();

hubConfiguration.EnableDetailedErrors = true;

hubConfiguration.EnableJavaScriptProxies = false;

### app.MapSignalR("/signalr", hubConfiguration);

1. How to create and use Hub classes

Vd: đơn giản cho 1 Hub class chat

public class ContosoChatHub : Hub

{

public async Task NewContosoChatMessage(string name, string message)

{

await Clients.All.addNewMessageToPage(name, message);

}

}

* Một client được kết nối có thể gọi NewContosoChatMessage method -> khi nó thự thi -> dữ liệu nhận được được broadcast đến tất cả các client được kết nối.

**Hub object lifetime**

You don't instantiate the Hub class or call its methods from your own code on the server; all that is done for you by the SignalR Hubs pipeline. SignalR creates a new instance of your Hub class each time it needs to handle a Hub operation such as when a client connects, disconnects, or makes a method call to the server.

Vì các biến đối tượng của Hub class là tạm thời -> Không thể duy trì trạng thái từ một phương thức gọi đến lần gọi kế tiếp. Each time the server receives a method call from a client, a new instance of your Hub class processes the message. Để duy trì trạng thái thông qua nhiều kết nối và các cuộc gọi, sử dụng một số phương thức như DB, biến static trên Hub class, hoặc một class khác mà không xuất phát từ Hub. If you persist data in memory, using a method such as a static variable on the Hub class, the data will be lost when the app domain recycles.

If you want to send messages to clients from your own code that runs outside the Hub class, you can't do it by instantiating a Hub class instance, but you can do it by getting a reference to the SignalR context object for your Hub class. For more information, see [How to call client methods and manage groups from outside the Hub class](https://docs.microsoft.com/en-us/aspnet/signalr/overview/guide-to-the-api/hubs-api-guide-server#callfromoutsidehub) later in this topic.

### Camel-casing of Hub names in JavaScript clients

Bằng việc mặc định, Js client tham chiếu đến Hubs bằng việc sử dụng camel-cased version of the class name. SignalR tự động tạo sự thay đối này mà Js code có thể theo Js quy ước.

VD:

**Server**

public class ContosoChatHub : Hub

**JavaScript client using generated proxy**

var contosoChatHubProxy = $.connection.contosoChatHub;

If you want to specify a different name for clients to use, add the HubName attribute. When you use a HubName attribute, there is no name change to camel case on JavaScript clients.

**Server**

[HubName("PascalCaseContosoChatHub")]

public class ContosoChatHub : Hub

**JavaScript client using generated proxy**

var contosoChatHubProxy = $.connection.PascalCaseContosoChatHub;

### Multiple Hubs

Có thể định nghĩa nhiều Hub classes trong 1 ứng dụng. Khi thực hiện điều đó thì kết nối được chia sẻ nhưng các nhóm thì chia tách:

+ Tất cả các client sẽ sử dụng cùng URL để thiết lập một kết nối SignalR với thiết bị của bạn ("/signalr" or your custom URL if you specified one), và kết nối đó được sử dụng cho tất cả các hubs được định nghĩa bởi dịch vụ.

Không có sự khác nhau về performance cho sự so sánh giữa các Hubs đến việc định nghĩa cho tất cả chức năng của Hub trong 1 single class.

+ All Hubs get the same HTTP request information.

Vì tất cả các Hub đều có chung một kết nối, thông tin yêu cầu HTTP duy nhất mà máy chủ nhận được là những gì có trong yêu cầu HTTP gốc thiết lập kết nối SignalR. Nếu bạn sử dụng yêu cầu kết nối để truyền thông tin từ máy khách đến máy chủ bằng cách chỉ định chuỗi truy vấn, bạn không thể cung cấp các chuỗi truy vấn khác nhau cho các Hub khác nhau. Tất cả các Hub sẽ nhận được thông tin giống nhau.

+ File các proxies Js được tạo sẽ chứa các proxies cho tất cả các Hub trong cùng 1 file.

For information about JavaScript proxies, see [SignalR Hubs API Guide - JavaScript Client - The generated proxy and what it does for you](https://docs.microsoft.com/en-us/aspnet/signalr/overview/guide-to-the-api/hubs-api-guide-javascript-client#genproxy).

+ Groups are defined within Hubs.

Trong SignalR có thể định nghĩa tên các group để broadcast đến một subsets các clients được kết nối. Group duy trì sự chia tách cho mỗi Hub.

For example, a group named "Administrators" would include one set of clients for your ContosoChatHub class, and the same group name would refer to a different set of clients for your StockTickerHub class.

### Strongly-Typed Hubs

To define an interface for your hub methods that your client can reference (and enable Intellisense on your hub methods), derive your hub from Hub<T> (introduced in SignalR 2.1) rather than Hub:

public class StrongHub : Hub<IClient>

{

public async Task Send(string message)

{

await Clients.All.NewMessage(message);

}

}

public interface IClient

{

Task NewMessage(string message);

}

1. How to define methods in the Hub class that clients can call (Định nghĩa các phương thức trong Hub class mà client có thể gọi)

Để show ra một phương thức trên Hub -> client có thể gọi vào -> public method

public class ContosoChatHub : Hub

{

public async Task NewContosoChatMessage(string name, string message)

{

await Clients.All.addNewMessageToPage(name, message);

}

}

public class StockTickerHub : Hub

{

public IEnumerable<Stock> GetAllStocks()

{

return \_stockTicker.GetAllStocks();

}

}

### Camel-casing of method names in JavaScript clients

**Server**

public void NewContosoChatMessage(string userName, string message)

**JavaScript client using generated proxy**

contosoChatHubProxy.server.newContosoChatMessage(userName, message);

If you want to specify a different name for clients to use, add the HubMethodName attribute.

**Server**

[HubMethodName("PascalCaseNewContosoChatMessage")]

public void NewContosoChatMessage(string userName, string message)

**JavaScript client using generated proxy**

contosoChatHubProxy.server.PascalCaseNewContosoChatMessage(userName, message);

### When to execute asynchronously

Making a Hub method asynchronous avoids blocking the connection when it uses the WebSocket transport. When a Hub method executes synchronously and the transport is WebSocket, subsequent invocations of methods on the Hub from the same client are blocked until the Hub method completes.

**Synchronous**

public IEnumerable<Stock> GetAllStocks()

{

// Returns data from memory.

return \_stockTicker.GetAllStocks();

}

**Asynchronous**

public async Task<IEnumerable<Stock>> GetAllStocks()

{

// Returns data from a web service.

var uri = Util.getServiceUri("Stocks");

using (HttpClient httpClient = new HttpClient())

{

var response = await httpClient.GetAsync(uri);

return (await response.Content.ReadAsAsync<IEnumerable<Stock>>());

}

}

**JavaScript client using generated proxy**

stockTickerHubProxy.server.getAllStocks().done(function (stocks) {

$.each(stocks, function () {

alert(this.Symbol + ' ' + this.Price);

});

});

For more information about how to use asynchronous methods in ASP.NET 4.5, see [Using Asynchronous Methods in ASP.NET MVC 4](https://docs.microsoft.com/en-us/aspnet/mvc/overview/performance/using-asynchronous-methods-in-aspnet-mvc-4).

### Defining Overloads

## How to call client methods from the Hub class

Để gọi các phương thức từ server, sử dụng tính chất Clients trong một phương thức trong Hub class.

The following example shows server code that calls addNewMessageToPage on all connected clients, and client code that defines the method in a JavaScript client

**Server**

public class ContosoChatHub : Hub

{

public async Task NewContosoChatMessage(string name, string message)

{

await Clients.All.addNewMessageToPage(name, message);

}

}

Thực thi phương thức client là một hoạt động bất đồng bộ (Task và await)

* To ensure the message is sent without error.
* To enable catching and handling errors in a try-catch block.

**JavaScript client using generated proxy**

contosoChatHubProxy.client.addNewMessageToPage = function (name, message) {

// Add the message to the page.

$('#discussion').append('<li><strong>' + htmlEncode(name)

+ '</strong>: ' + htmlEncode(message) + '<li>');

};

You can't get a return value from a client method; syntax such as int x = Clients.All.add(1,1) does not work.

You can specify complex types and arrays for the parameters. The following example passes a complex type to the client in a method parameter.

**Server code that calls a client method using a complex object**

public async Task SendMessage(string name, string message)

{

await Clients.All.addContosoChatMessageToPage(new ContosoChatMessage() { UserName = name, Message = message });

}

public class ContosoChatMessage

{

public string UserName { get; set; }

public string Message { get; set; }

}

**JavaScript client using generated proxy**

var contosoChatHubProxy = $.connection.contosoChatHub;// tạo một biến kết nối đến Hub

contosoChatHubProxy.client.addMessageToPage = function (message) {

console.log(message.UserName + ' ' + message.Message);

});