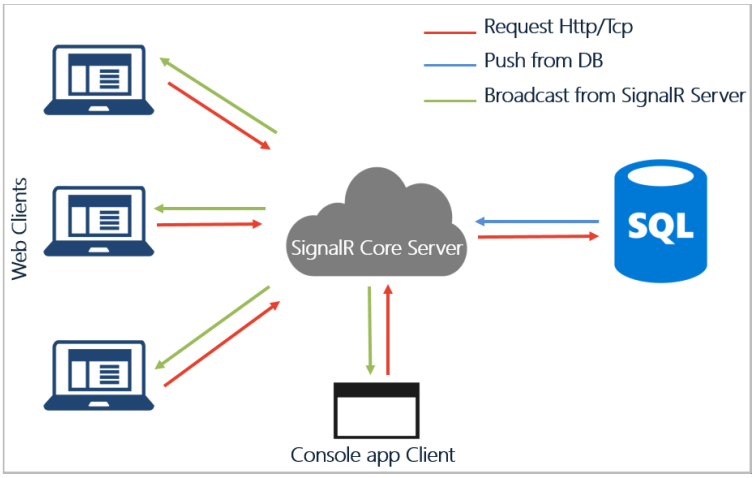
Agenda: **SignalR**

- Introduction and Terminology.
- Buiding Broadcast / Chat Application.
- Building One to One Chat Application.

## Introduction an Terminology

**SignalR** is an open-source library that simplifies adding real-time web functionality to apps. Real-time web functionality enables server-side code to push content to clients instantly.

It provides an API for creating server-to-client **remote procedure calls** (RPC). The RPCs call JavaScript functions on clients from server-side




**Features of SignalR**

- Handles connection management automatically.
- Sends messages to all connected clients simultaneously. For example, a chat room.
- Sends messages to specific clients or groups of clients.
- Scales to handle increasing traffic.

**Transports**:

It supports several techniques for handling real-time communications. SignalR automatically chooses the best transport method that is within the capabilities of the server and client.

- Web Sockets
- Server-Sent Events
- Long Polling

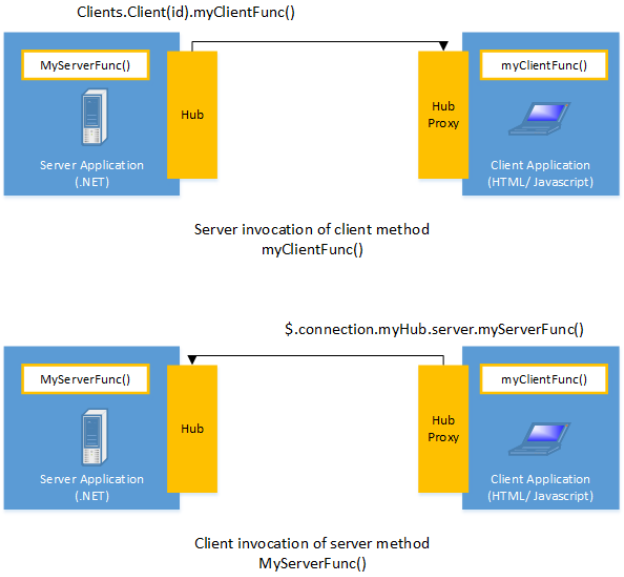
**HUBS:**

SignalR uses Hubs to communicate between Clients and Servers.

- A hub is a **high-level pipeline** that allows a client and server to call methods on each other.
- SignalR handles the dispatching across machine boundaries automatically, **allowing clients to call methods on the server and vice versa.**
- We can pass **strongly-typed parameters to methods**, which enables model binding. SignalR provides two built-in hub protocols:
    - A text protocol based on JSON.
    - A binary protocol based on Message Pack.
- Hubs call client-side code by sending messages that contain the name and parameters of the client-side method.
- Objects sent as method parameters are desterilized using the configured protocol. The client tries to match the name to a method in the client-side code. When the client finds a match, it calls the method and passes to it the desterilized parameter data

**SignalR Hub:**

The SignalR Hubs API enables you to call methods on connected clients from the server. In the server code, you define methods that are called by client. In the client code, you define methods that are called from the server.




## Example 1: Chat Application

**Step1:** Create Asp.net MVC Project

1. Create an ASP.NET Core Application**.**

**Step2:** Add SignalR client Library

2. Right Click on Project → Add → Client-Side Library.

3. Select Provier = "unpkg", Library = "@aspnet/signalr@" → Select the latest version.
4. Select **Choose specific files**, expand **dist/browser** and select **signalr.js** and **signalr.min.js**

**Step3: Create a SignalR Hub**

5. Add New Folder - **Hubs**
6. Add Hubs/ChatHub.cs

```
public class ChatHub : Hub
{
   public async Task SendMessage(string user, string message)
   {
      await Clients.All.SendAsync("ReceiveMessage", user, message);
   }
}
```

**Step4: Configure SignalR**

7. Add the following to ConfigureServices in Startup class (last line)

```
services.AddSignalR();
```

8. Add the following to Configure method in Startup class (Just above app.UseMVC())

```
app.UseSignalR(routes =>
{
            routes.MapHub<ChatHub>("/chatHub");
});
```

**Step5: Add SignalR client code**

9. Edit Views\Home\Index.cshtml

```
<div class="container">
   <div class="row"> </div>
   <div class="row">
      <div class="col-6"> </div>
      <div class="col-6">
         User.....<input type="text" id="userInput" />
         <br />
         Message...<input type="text" id="messageInput" />
         <input type="button" id="sendButton" value="Send Message" />
```

```
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <hr />
    </div>
  </div>
  <div class="row">
    <div class="col-6"> </div>
    <div class="col-6">
      <ul id="messagesList"></ul>
    </div>
  </div>
</div>
<script src="~/lib/signalr/dist/browser/signalr.js"></script>
<script src="~/js/chat.js"></script>
```

10. Create wwwroot/js/chat.js file

```
"use strict";

var connection = new signalR.HubConnectionBuilder().withUrl("/chatHub").build();

//Disable send button until connection is established
document.getElementById("sendButton").disabled = true;

connection.on("ReceiveMessage", function (user, message) {
  var msg = message.replace(/&/g, "&amp;").replace(/</g, "&lt;").replace(/>/g, "&gt;");
  var encodedMsg = user + " says " + msg;
  var li = document.createElement("li");
  li.textContent = encodedMsg;
  document.getElementById("messagesList").appendChild(li);
});

connection.start().then(function(){
  document.getElementById("sendButton").disabled = false;
```

```
}).catch(function (err) {
    return console.error(err.toString());
});

document.getElementById("sendButton").addEventListener("click", function (event) {
    var user = document.getElementById("userInput").value;
    var message = document.getElementById("messageInput").value;
    connection.invoke("SendMessage", user, message).catch(function (err) {
        return console.error(err.toString());
    });
    event.preventDefault();
});
```

The preceding code:

- Creates and starts a connection.
- Adds to the submit button a handler that sends messages to the hub.
- Adds to the connection object a handler that receives messages from the hub and adds them to the list.

**Step6: Run the application.**

11. Open multiple instance of browser and chat.

## Example2: One on One Chat

**ChatHub.cs**

```
public class ChatHub : Hub
{
    static Dictionary<string, string>  clientIds = new Dictionary<string, string>();
    public async Task RegisterMe(string user)
    {
        if (!clientIds.Keys.Contains(user))
        {
            clientIds.Add(user, base.Context.ConnectionId);
            await Clients.All.SendAsync("NewUserRegistered", clientIds.Keys.ToList());
        }
    }
    public async Task SendMessage(string fromUser, string toUser, string message)
    {
        List<string> lst = new List<string>() { clientIds[toUser] };
```

```
        IReadOnlyList<string> connectionIds = lst.ToList();
        await Clients.Clients(connectionIds).SendAsync("ReceiveMessage", fromUser, message);
    }
}
```

**Index.html**

```
<div>
    <div>
        Enter your Name:<input type="text" id="fromUserName" />
        <input type="button" id="registerButton" value="Register" />
        <hr />
        Your Message:<input type="text" id="messageInput" /><br />
        To User: <select id="lstUsers"></select><br />
        <input type="button" id="sendButton" value="Send Message" />
    </div>
    <div><hr /></div>
    <div>
        <ul id="messagesList"></ul>
    </div>
</div>
<script src="~/lib/signalr/dist/browser/signalr.js"></script>
<script src="~/js/chat.js"></script>
```

**Chat.js:**

```
"use strict";

var connection = new signalR.HubConnectionBuilder().withUrl("/chatHub").build();

//Disable send button until connection is established
document.getElementById("sendButton").disabled = true;

connection.on("ReceiveMessage", function (fromUser, message) {
    var msg = message.replace(/&/g, "&amp;").replace(/</g, "&lt;").replace(/>/g, "&gt;");
    var encodedMsg = fromUser + " says " + msg;
    var li = document.createElement("li");
```

```
    li.textContent = encodedMsg;
    document.getElementById("messagesList").appendChild(li);
});

connection.on("NewUserRegistered", function (lstUsers) {
    var lst = document.getElementById("lstUsers");
    for (var user in lstUsers) {
        if (lstUsers[user] !== document.getElementById("fromUserName").value) {
            var option = document.createElement("option");
            option.text = lstUsers[user];
            lst.options.add(option);
        }
    }
});

connection.start().then(function () {
    document.getElementById("sendButton").disabled = false;
}).catch(function (err) {
    return console.error(err.toString());
});

document.getElementById("registerButton").addEventListener("click", function (event) {
    var user = document.getElementById("fromUserName").value;
    connection.invoke("RegisterMe", user).catch(function (err) {
        return console.error(err.toString());
    });
    event.preventDefault();
});

document.getElementById("sendButton").addEventListener("click", function (event) {
    var fromUser = document.getElementById("fromUserName").value;
    var toUser = document.getElementById("lstUsers").value;
    var message = document.getElementById("messageInput").value;
    connection.invoke("SendMessage", fromUser, toUser, message).catch(function (err) {
        return console.error(err.toString());
```

```
    });
    event.preventDefault();
});
```