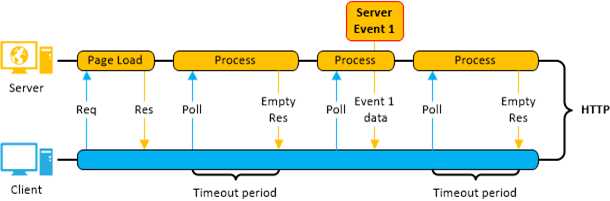
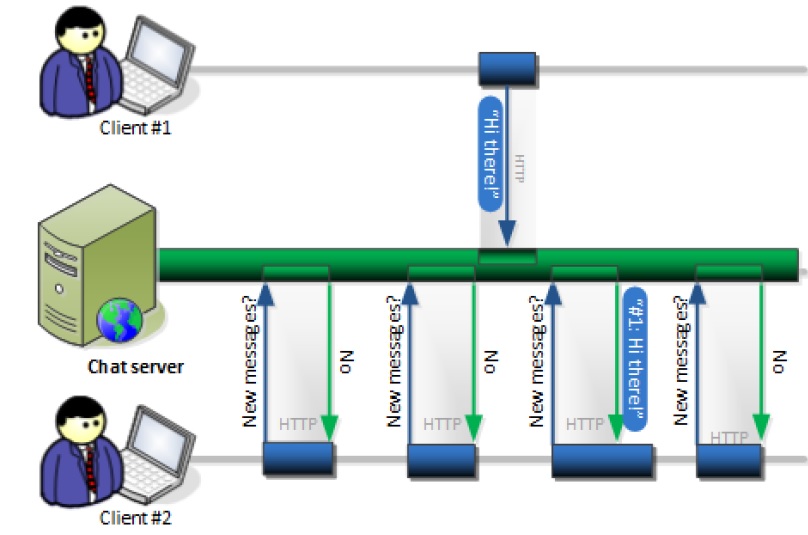
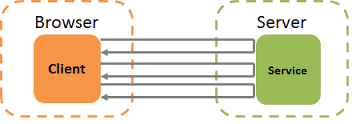
**SignalR**SignalR is a new developer's API provided for ASP.NET Web Applications by Microsoft. It used to add "real time" Web functionality to ASP.NET Applications. "Real Time" Web functionality is the ability of a Server code to push the contents to the connected clients. In this article, we will focus on the following things in detail-

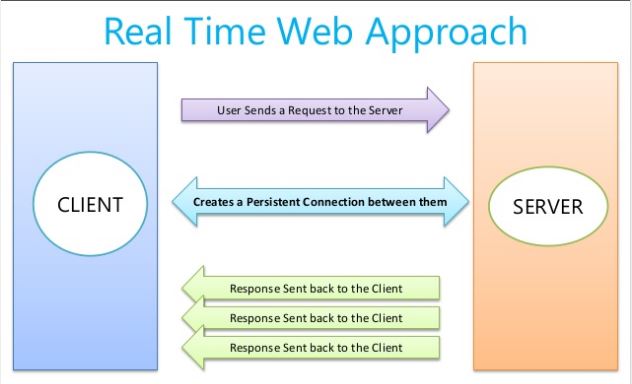
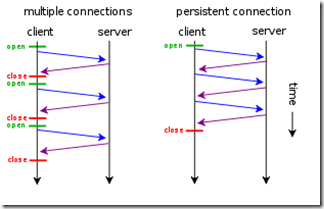
1. Introduction to SignalR
2. Traditional Pooling vs Pushing
3. SignalR clients
4. Working with Query client
5. Working with .NET client.
6. Self hosting of SignalR Application.
7. Understanding SignalR Methods

Before we start digging into SignalR, let's check the Wiki of SignalR.  
   
SignalR is a library for ASP.NET developers used to develop real time Web Applications (which makes use of Push Technology).

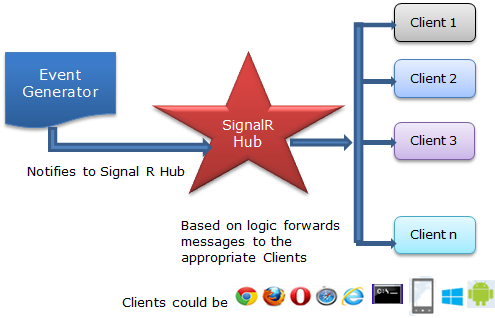
If we compare the traditional Web Application with current SignalR Application, we will understand why we should prefer SignalR.  
  
Let's see how traditional Pulling Web Applications work.  
  
**Scenario1**Previously, when we were creating an Application like chat app or something diffrent, the first thing we would think is how to get the latest chats from the user, so we usually put a timer inside our page in ASP.NET client, which calls the Web Service method or the Data access logic in **every 5 seconds (depends on timer)** and updates the chat data on the client side.  
  
This kind of Application works perfectly for us but it has too many problems.

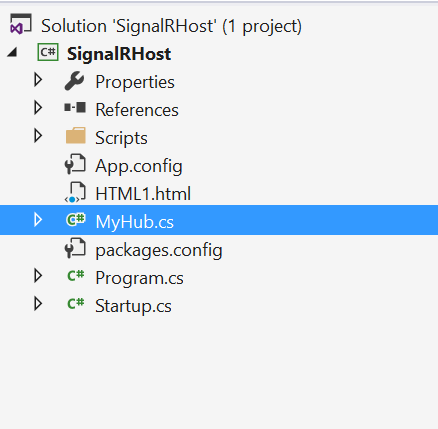
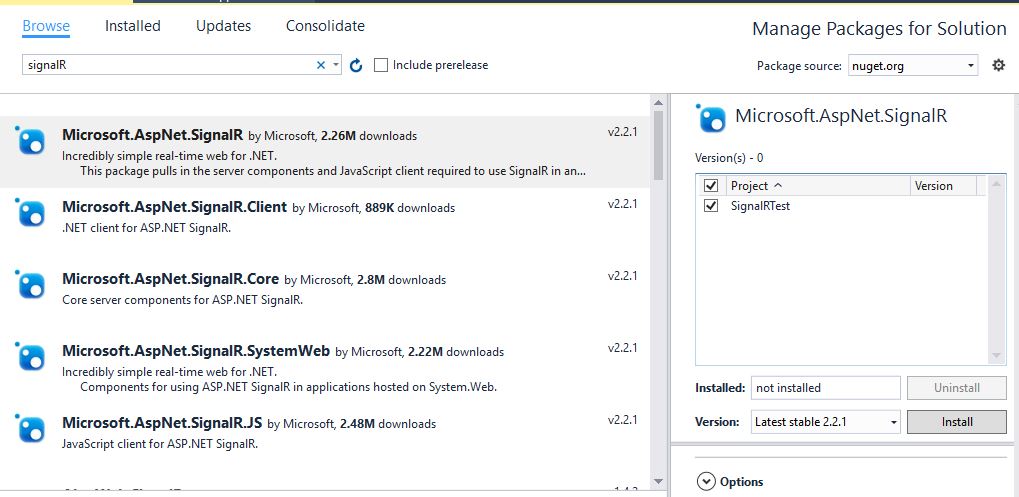
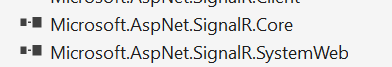
1. Increase network traffic (In each particular time, the  communication happens, there is a request to the Server and gets back the response from the Server.)
2. HTTP connections for the client-Server communication connection is re-established for each request.
3. Delay in reciving the data due to pooling time period.  
     
     
     
   **In real time, lots of request from the client  are wasted when there is no update data in the Server and returns null in the response.If you are not maintaining your client app correctly, your whole app refreshes in each round trip also.**  
     
     
     
   When you need a continous update from the Server, you will call the Server at a particular period of time, so in every request, a connection is established with the Server and checks for the update. **If there is any update, it will talk to the client, so this is called Pulling.**  
     
   

**Real Time Pushing**SignalR support "server Push" is a functionality in which the Server code can call out client code in the Browser, using **RPC**. Server push means if any update occurs in the database, it just pushes the data to the client instead of creating connection checking for the update etc.  
  
In SignalR, we have persistant connection, so the connection is established once in the beginning and remains.

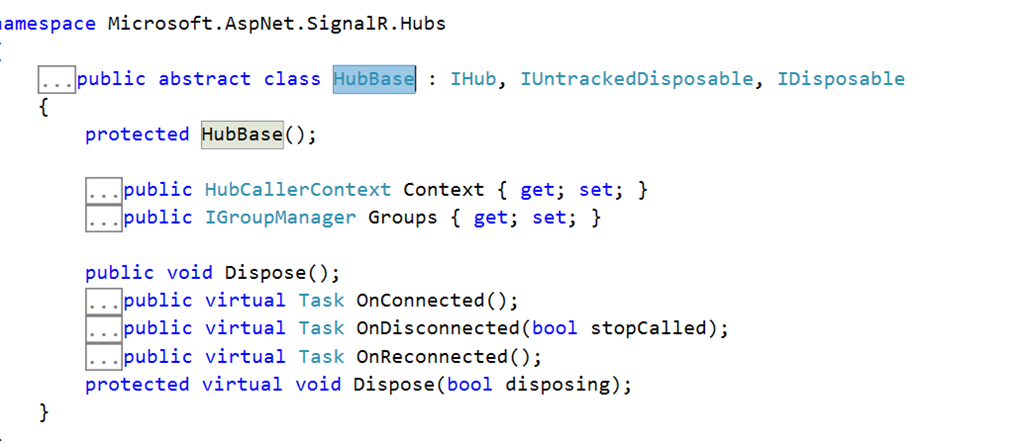
   
 **Persistant Connection**  
As per Wiki "**HTTP persistent connection**, also called **HTTP keep-alive**, or **HTTP connection reuse**, is the idea of using a **single TCP connection** to send and receive multiple **HTTP requests/responses**, **as opposed to opening a new connection for every single request/response pair.** The newer HTTP/2 protocol uses the same idea and takes it further to allow multiple concurrent requests/responses to be multiplexed over a single connection."  
  
  
  
To develop SignalR Application, we need 2 things in our Application.

1. HUB Class(Server side)
2. SignalR Clients(Client side)

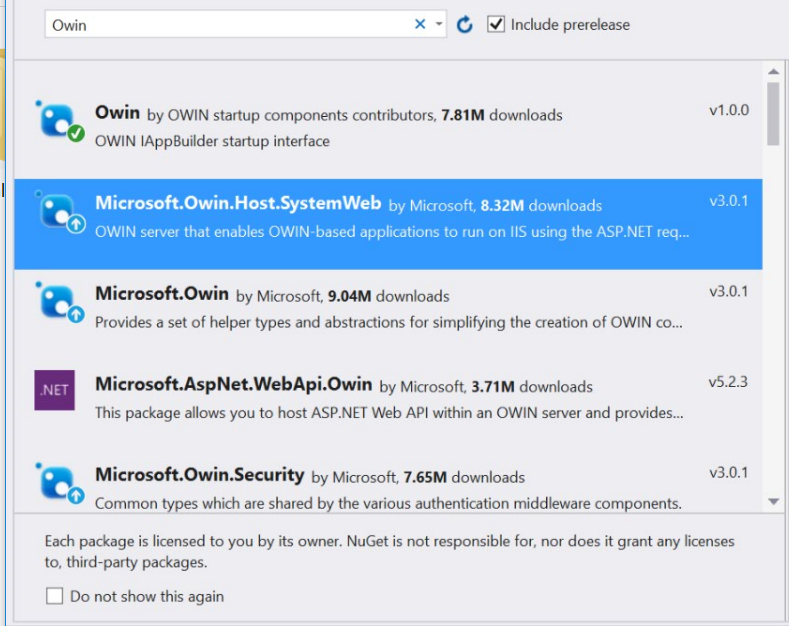
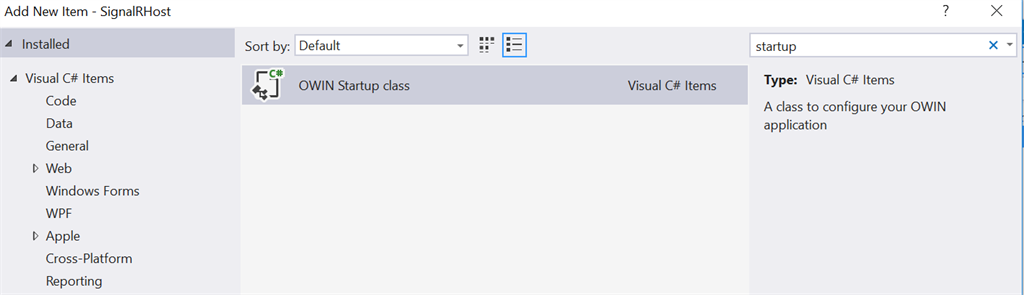
SignalR almost supports clients for all technologies. If we have a look at this, it has a client for Web, Andrid, iPhone etc.  
  
   
  
Before working on signalR, let's check some points.

1. Any class that is Inherited from HUB class is a SignalR Hub.Hub is a Server side material and needs to be hosted.
2. For .NET,  we have 2 clients that is (jQuery client and  have ASP.NET client).  
   So from here we will check how we can work with signalR.  
     
   Open vs and Create a new Web Project in Console Application.  
     
     
     
   Now, go to the Tool and install SignalR from NuGet Package.  
     
     
   Now, you will find the References given below in the project.  
     
     
     
   You will find a Hub class also in the project.
   1. **using** System;
   2. **using** System.Collections.Generic;
   3. **using** System.Linq;
   4. **using** System.Text;
   5. **using** System.Threading.Tasks;
   6. **using** Microsoft.AspNet.SignalR;
   7. **using** Microsoft.AspNet.SignalR.Hubs;
   9. **namespace** SignalRHost
   10. {
   11. [HubName("MyHub")]
   12. **public** **class** MyHub:Hub
   13. {

   16. }
   17. }

If we go to the definition of Hub, we will find the things given below inside the Hub.  
  
  
  
Now, we will see how can we call the SignalR Hub methods from .NET client. Now, I have created a method and I want to consume this method in my .NET client.

* 1. **using** System;
  2. **using** System.Collections.Generic;
  3. **using** System.Linq;
  4. **using** System.Text;
  5. **using** System.Threading.Tasks;
  6. **using** Microsoft.AspNet.SignalR;
  7. **using** Microsoft.AspNet.SignalR.Hubs;
  9. **namespace** SignalRHost
  10. {
  11. [HubName("MyHub")]
  12. **public** **class** MyHub:Hub
  13. {
  15. **public** string getdetails( string s)
  16. {
  17. **return** "Hi" + s;
  19. }
  21. }
  22. }

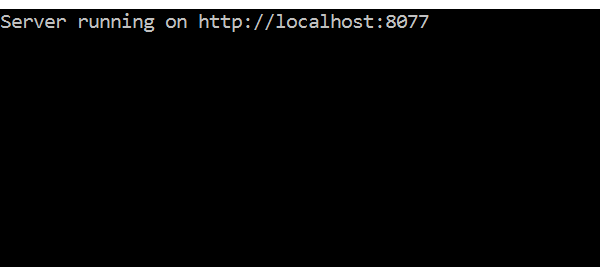
Now, we need to host the signalR Hub. Hence, we need OWIN to host this.  
  
  
  
  
It will add a startup.cs class or you can add a startup class and configure the SignalR hub, as shown below.  
  
  
  
Now, open the startup class and write the line given below.     
  
**Mapping the Hubs connection**  
To enable SignalR in your Application, create a class called Startup with the code given below.

* 1. **using** System;
  2. **using** System.Threading.Tasks;
  3. **using** Microsoft.Owin;
  4. **using** Owin;
  6. [assembly: OwinStartup(**typeof**(SignalRHost.Startup))]
  8. **namespace** SignalRHost
  9. {
  10. **public** **class** Startup
  11. {
  12. **public** **void** Configuration(IAppBuilder app)
  13. {
  15. app.MapSignalR();
  17. // For more information on how to configure your application, visit http://go.microsoft.com/fwlink/?LinkID=316888
  18. }
  19. }
  20. }

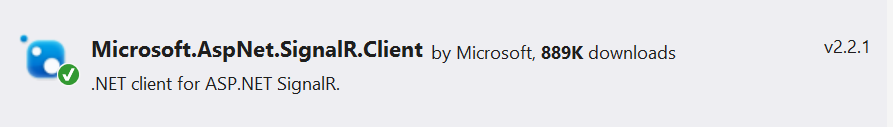
Now, open the main method of the console Application and write the code given below to host the Hub.

* 1. **using** System;
  2. **using** System.Collections.Generic;
  3. **using** System.Linq;
  4. **using** System.Text;
  5. **using** System.Threading.Tasks;
  6. **using** Microsoft.AspNet.SignalR;
  7. **using** Microsoft.AspNet.SignalR.Hubs;
  8. **using** Microsoft.AspNet.SignalR.Client;
  9. **using** Microsoft.Owin.Hosting;
  11. **namespace** SignalRHost
  12. {
  13. **class** Program
  14. {
  15. **static** **void** Main(**string**[] args)
  16. {
  18. **string** url = "http://localhost:8077";
  19. **using** (WebApp.Start(url))
  20. {
  21. Console.WriteLine("Server running on {0}", url);
  22. Console.ReadLine();
  23. }
  25. System.Console.Read();
  26. }
  27. }
  28. }

Here, we are hosting the the Hub on "*http://localhost:8077*" URL.

        Now, run the console Application, as shown below.  
          
  
In this way, we can host our Application, using self Hosting. Now, SignalR Service is available in "http://localhost:8077".

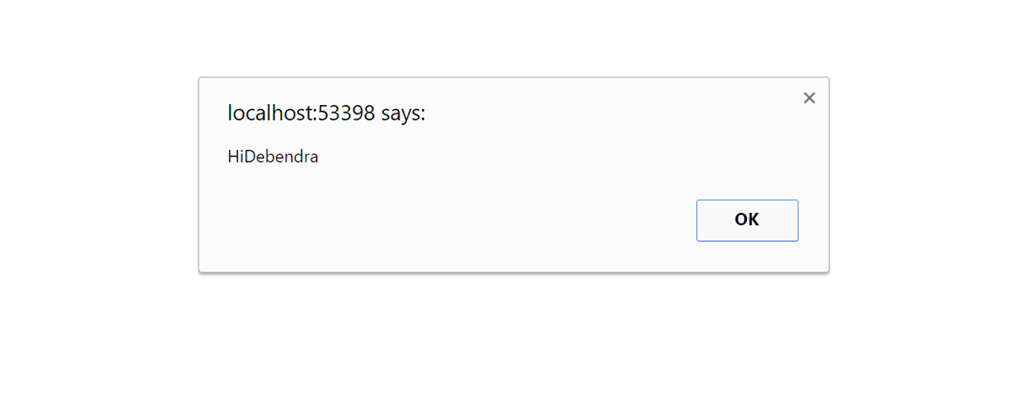
Now, let's create another Web client Application and try to consume the Hub Method. Create another application and add signalR .NET client, as shown below.

  
  
Now, there are simple 5 steps to consume the method hosted in the hub.

1. Establish a connection with the URL, where the Hub is hosted.
2. Create a proxy of the Hub.
3. Open the connection.
4. Using the proxy object, call the method, which you want to invoke.
5. Save the result and display in the client Application.
   1. **using** Microsoft.AspNet.SignalR.Client;
   2. **using** System;
   3. **using** System.Collections.Generic;
   4. **using** System.Linq;
   5. **using** System.Web;
   6. **using** System.Web.UI;
   7. **using** System.Web.UI.WebControls;
   9. **namespace** SignalRChat
   10. {
   11. **public** partial **class** chat : System.Web.UI.Page
   12. {
   13. **public**  HubConnection hubConnection = **null**;
   14. **public**  IHubProxy HubProxy=**null**;
   15. **protected** **void** Page\_Load(**object** sender, EventArgs e)
   16. {
   18. }
   20. **protected** **void** Button\_Click(**object** sender, EventArgs e)
   21. {
   22. hubConnection = **new** HubConnection("http://localhost:8077/");
   24. HubProxy = hubConnection.CreateHubProxy("MyHub");
   25. hubConnection.Start();
   26. Execute();
   28. var p = HubProxy.Invoke<**string**>("getdetails", "Debendra").Result;
   29. ClientScript.RegisterStartupScript(**this**.GetType(), "myalert", "alert('" + p + "');", **true**);


   33. }

   36. **private** **void** Execute()
   37. {
   39. hubConnection.Start().ContinueWith(task =>
   40. {
   41. **if** (task.IsFaulted)
   42. {
   43. Console.WriteLine("There was an error opening the connection:{0}",
   44. task.Exception.GetBaseException());
   46. **return**;
   47. }
   48. **else**
   49. {
   50. Console.WriteLine("Connected to Server.The ConnectionID is:" + hubConnection.ConnectionId);
   52. }
   54. }).Wait();
   55. }
   56. }
   57. }

The result is given below.  
  
Now, let's check how can we call a Hub method, using JavaScript client. Let's have the method given below in the HUB.

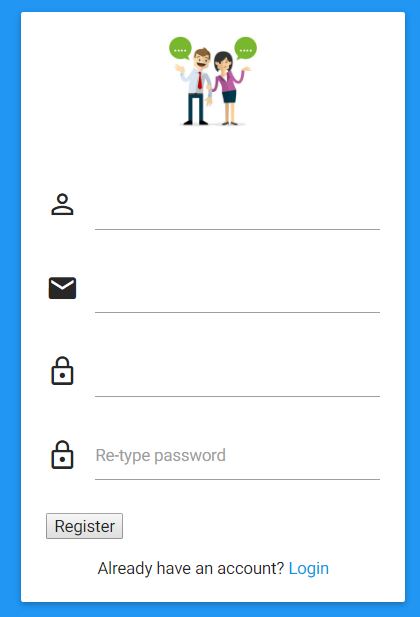
* 1. **public** **class** ChatHub : Hub
  2. {
  3. #region Data Members
  5. **static** List<UserDetail> ConnectedUsers = **new** List<UserDetail>();
  6. **static** List<MessageDetail> CurrentMessage = **new** List<MessageDetail>();
  7. SqlConnection con;
  8. DataSet ds;
  9. DataTable dt;
  10. SignalREntities \_signalRDB;
  12. #endregion
  14. #region Methods
  16. **public** **void** Connect(string userName, string email,string pass)
  17. {
  18. \_signalRDB = **new** SignalREntities();
  19. Registration reg = **new** Registration();
  20. var id= Context.ConnectionId;
  21. reg.ConnectionId= Context.ConnectionId;
  22. reg.NickName= userName;
  23. reg.EmailId = email;
  24. reg.password = pass;
  25. \_signalRDB.Registrations.Add(reg);
  26. \_signalRDB.SaveChanges();
  28. }

I have have simply created a Create method in my Chat Hub. Now, I want to call this method from Javascript client. First, add JavaScript references given below on the page.

* 1. <script src="/Scripts/jquery-1.8.2.min.js"></script>
  2. <!--Reference the SignalR library. -->
  3. <script src="/Scripts/jquery.signalR-1.0.0.js"></script>
  5. <!--Reference the autogenerated SignalR hub script. -->
  6. <script src="/signalr/hubs"></script>

Now, my JavaScript client code is given below.

* 1. <script type="text/javascript">
  3. $(**function** () {
  5. // Declare a proxy to reference the hub.
  6. **var** chatHub = $.connection.chatHub;
  8. registerClientMethods(chatHub);
  10. // Start Hub
  11. $.connection.hub.start().done(**function** () {
  13. registerEvents(chatHub)
  15. });
  17. });
  18. **function** registerEvents(chatHub) {
  20. $("#btn\_register").click(**function** () {
  22. **var** name = $("#txt\_NickName").val();
  23. **var** email = $("#txt\_email").val();
  24. **var** password = $("#txt\_password").val();
  26. **if** (name.length > 0) {
  27. chatHub.server.connect(name,email,password);  **//It will call the connect method on Hub**
  28. }
  29. **else** {
  30. alert("Please enter name");
  31. }
  33. });
  35. </script>

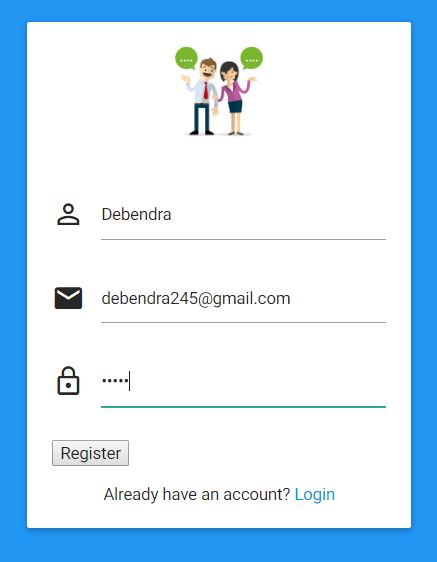
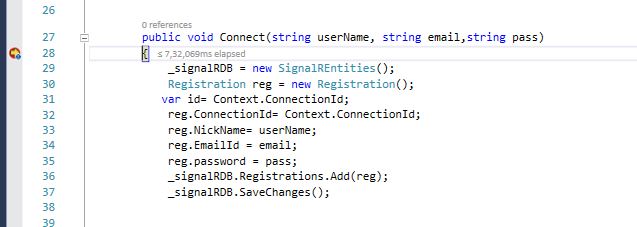
Hence, my UI is given for for the code given below.  
  


* 1. <body **class**="blue">

  4. <div id="login-page" **class**="row">
  5. <div **class**="col s12 z-depth-6 card-panel">
  6. <form **class**="login-form" runat="server">
  7. <div **class**="row">
  8. <div **class**="input-field col s12 center">
  10. <img src="Images/240\_F\_98129226\_SNcpf7qMgTgJikjE4xdskEHEgV465gnN.jpg" width="120" />
  11. </div>
  12. </div>
  13. <div **class**="row margin">
  14. <div **class**="input-field col s12">
  15. <i **class**="mdi-social-person-outline prefix"></i>
  17. <asp:TextBox ID="txt\_NickName" runat="server"></asp:TextBox>
  19. </div>
  20. </div>
  21. <div **class**="row margin">
  22. <div **class**="input-field col s12">
  23. <i **class**="mdi-communication-email prefix"></i>
  25. <asp:TextBox ID="txt\_email" runat="server"></asp:TextBox>
  27. </div>
  28. </div>
  29. <div **class**="row margin">
  30. <div **class**="input-field col s12">
  31. <i **class**="mdi-action-lock-outline prefix"></i>
  32. <input id="txt\_password" type="password" **class**="validate">
  34. </div>
  35. </div>

  38. <div **class**="row">
  39. <div **class**="input-field col s12">
  41. <asp:Button ID="btn\_register" Text="Register" runat="server" />
  42. </div>
  43. <div **class**="input-field col s12">
  44. <p **class**="margin center medium-small sign-up">Already have an account? <a href="login.aspx">Login</a></p>
  45. </div>
  46. </div>
  47. </form>
  48. </div>
  49. </div>


  53. </body>

Now, lets enter the details and click on register the button.  
  
  
  
As per the logic, it will execute the Hub connect method.  
  
  
  
In this way in JavaScript, we can call the Server Hub method.

I hope you understood these basics of SignalR. If you have any doubts about this, please let me know, so that I can try to explain it more clearly.  
  
In my next article, we will work on a SignalR project.