Instructor Notes:

Add instructor notes here.

Instructor Notes:

Add instructor notes here.

Lesson Objectives

- In this lesson we will cover the following
- Traditional Web Application Development
- Real Time Application
- SignalR

Instructor Notes:

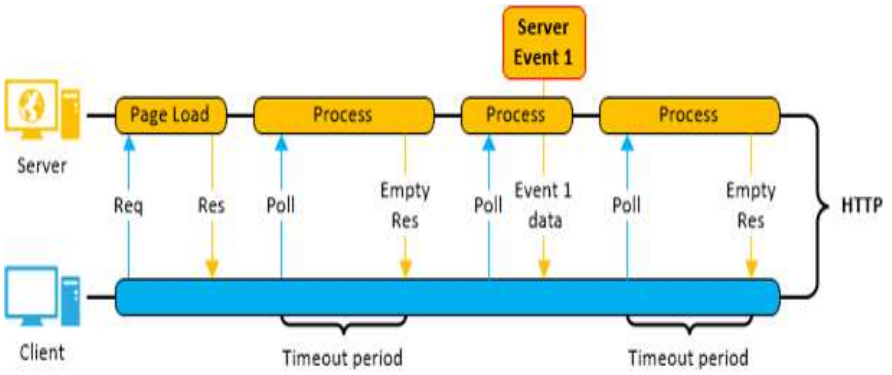
Traditional Web Application Development

- 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 usally 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.
  - 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.)
  - HTTP connections for the client-Server communication connection is re-established for each request.
  - Delay in reciving the data due to pooling time period.

Instructor Notes:

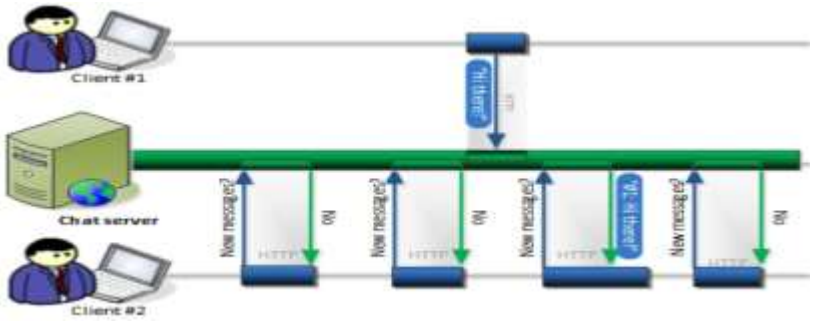
Working of Tradition Pulling Web Application



Instructor Notes:

Real Time Scenario

- 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.



Instructor Notes:

Real Time Application

- A real-time application is an application program that functions within a time frame that the user senses as immediate or current.
- The latency must be less than a defined value, usually measured in seconds.
- The real-time web is a network web using technologies and practices that enable users to receive information as soon as it is published by its authors, rather than requiring that they or their application check a source periodically for updates.
- Example of Real Time Application
  - Facebook
  - Twitter
  - Google Docs

The term *real-time software* refers to the type of software that is subject to a soft or hard time constraint. By the nature of its business domain, this type of software must complete its processing within a particular timeframe. This time constraint can be strict to make it hard real-time software or flexible to make it soft real-time software. For example, aerospace software can be hard real-time software because it must finish its execution within a defined time interval; otherwise, operations might fail, and people's lives can be in danger. In contrast, video players can be soft real-time software because it is not mission-critical if the processing does not finish on time (although it is important to process the videos in time to display them to users).

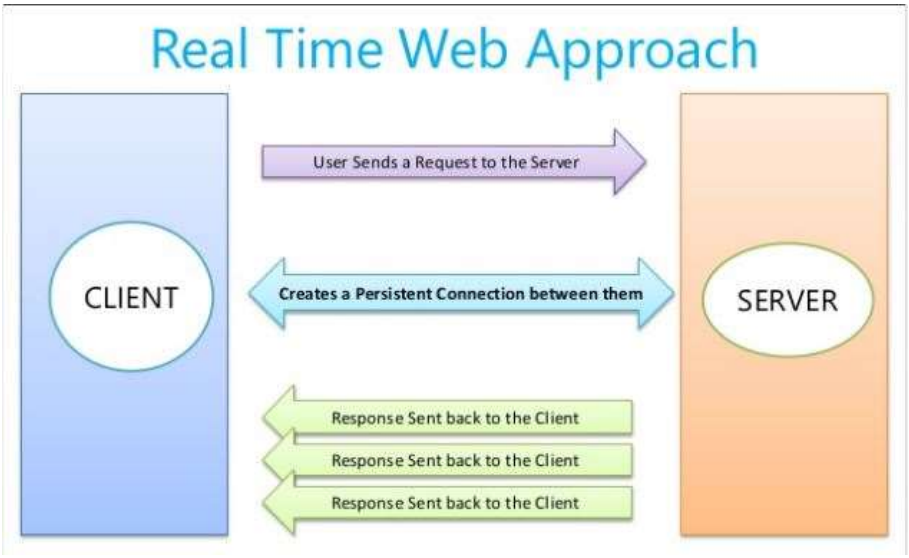
Although real-time software and real-time computing have been around for a long time, the term *real-time web* is relatively new. This concept, which was introduced in the past few years, focuses on real-time delivery of content to clients as soon as it is available.

The real-time web is similar to soft real-time software because the delivery of content from the information source to consumers should occur in a short period of time to be considered real time (from a few milliseconds up to 1 second).

The real-time web was embraced by social networks and their need to update users with frequent status updates and content changes by friends and peers. Facebook and Twitter were among the first major sites on the Internet that pioneered in this area and implemented real-time web features.

Instructor Notes:

Real Time Web Application



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.

Instructor Notes:

SignalR

- ASP.NET SignalR is a library for ASP.NET developers that simplifies the process of adding real-time web functionality to applications.
- Real-time web functionality is the ability to have server code push content to connected clients instantly as it becomes available, rather than having the server wait for a client to request new data.
- SignalR takes advantage of several transports, automatically selecting the best available transport given the client's and server's best available transport.
- SignalR takes advantage of WebSocket, an HTML5 API that enables bi-directional communication between the browser and server.

ASP.NET SignalR is a library for ASP.NET developers that simplifies the process of adding real-time web functionality to applications. Real-time web functionality is the ability to have server code push content to connected clients instantly as it becomes available, rather than having the server wait for a client to request new data.

SignalR can be used to add any sort of "real-time" web functionality to your ASP.NET application. While chat is often used as an example, you can do a whole lot more. Any time a user refreshes a web page to see new data, or the page implements long polling to retrieve new data, it is a candidate for using SignalR. Examples include dashboards and monitoring applications, collaborative applications (such as simultaneous editing of documents), job progress updates, and real-time forms.

SignalR also enables completely new types of web applications that require high frequency updates from the server, for example, real-time gaming. For a great example of this, see the ShootR game.

SignalR provides a simple API for creating server-to-client remote procedure calls (RPC) that call JavaScript functions in client browsers (and other client platforms) from server-side .NET code. SignalR also includes API for connection management (for instance, connect and disconnect events), and grouping connections.

Instructor Notes:

SignalR (contd..)

- SignalR will use WebSockets under the covers when it's available, and gracefully fall back to other techniques and technologies when it isn't, while the application code remains the same
- SignalR also provides a simple, high-level API for doing server-to-client RPC in an ASP.NET application, as well as adding useful hooks for connection management, such as connect/disconnect events, grouping connections, authorization.
- SignalR can be used to add any sort of "real-time" web functionality to your ASP.NET application.

SignalR handles connection management automatically, and lets you broadcast messages to all connected clients simultaneously, like a chat room. You can also send messages to specific clients. The connection between the client and server is persistent, unlike a classic HTTP connection, which is re-established for each communication.

SignalR supports "server push" functionality, in which server code can call out to client code in the browser using Remote Procedure Calls (RPC), rather than the request-response model common on the web today.

Instructor Notes:

Characteristic of SignalR

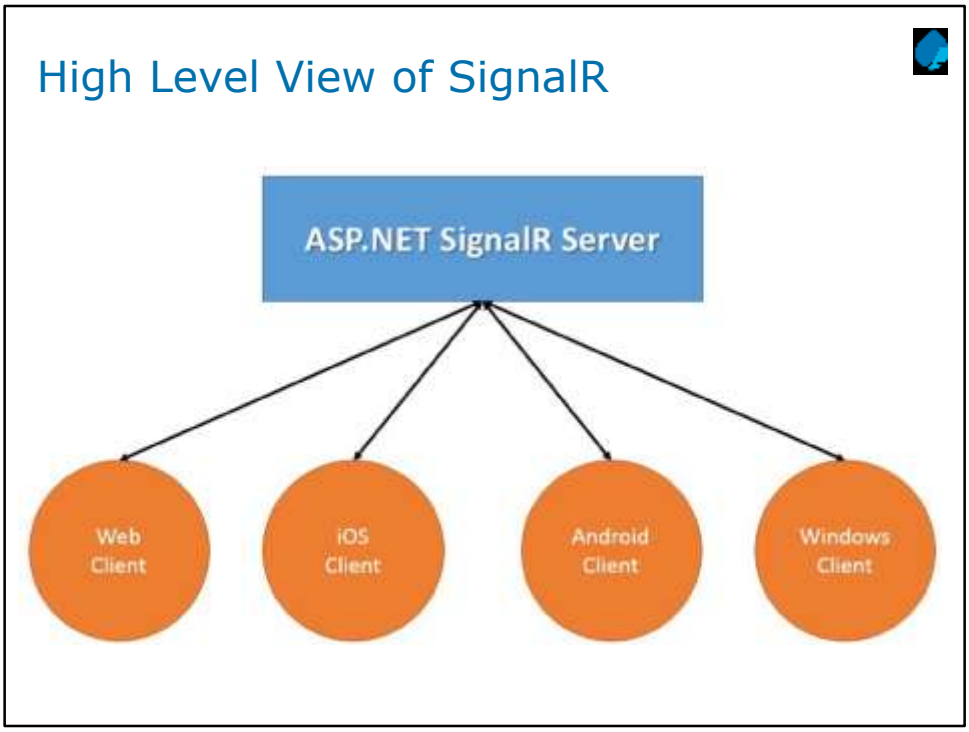
- Flexible
- Extensible
- Scalable

Flexible :- It provides different layers of tools to allow developers to build their custom applications. On one hand, ASP.NET SignalR offers hubs, which area simple and quick way to build a real–time web application and hide some of the details to facilitate the job of web developers. On the other hand, persistent connections are a more fundamental tool for building applications that give more flexibility and power to developers, yet require more effort to handle certain things that are taken care of by hubs.

Extensible :- Many components in ASP.NET SignalR are designed to be easily replaced by a custom implementation if necessary. ASP.NET SignalR has integrated dependency injection into its internal structure to offer such a good level of extensibility. You usually do not need to replace these components, but it is a straightforward task if you do.

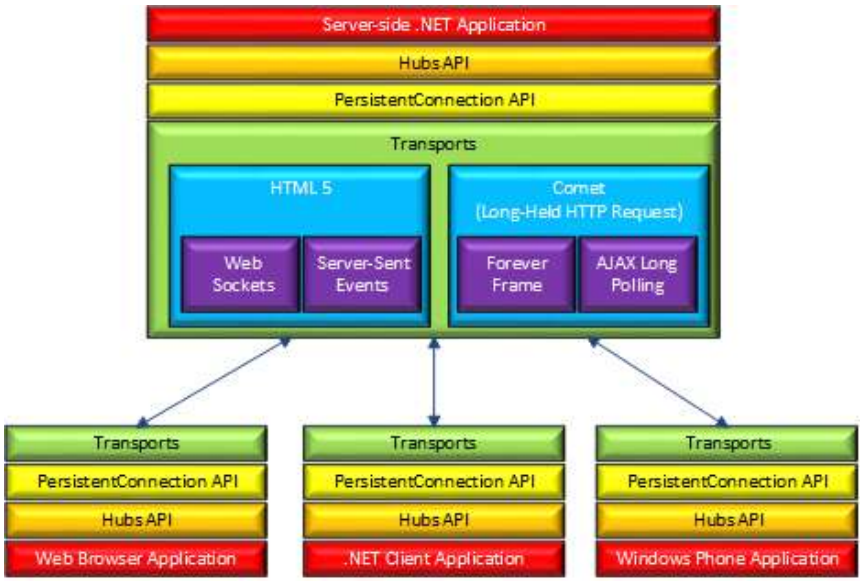
Scalable :- ASP.NET SignalR provides some built-in mechanisms to enable web developers to scale it up and out easily. Hosting a SignalR server application on multiple servers can introduce a set of common challenges, but these challenges are addressed by providing a set of extensible features in SignalR.

Instructor Notes:



Instructor Notes:

SignalR Architecture



SignalR library consists of a stacked architecture that uses one of the four common transport option i.e Long Polling, Forever Frame, Server Sent Event, Web Sockets

Depending on the network infrastructure and availability,one of these four transports is used in order of priority. If hub APIs are used, they are the points of connection for clients to simplify the logic and call the underlying persistent connection APIs in SignalR. If persistent connection APIs are used directly, developers need to take care of receiving the raw data from clients and extracting metadata to respond to these requests.

Instructor Notes:

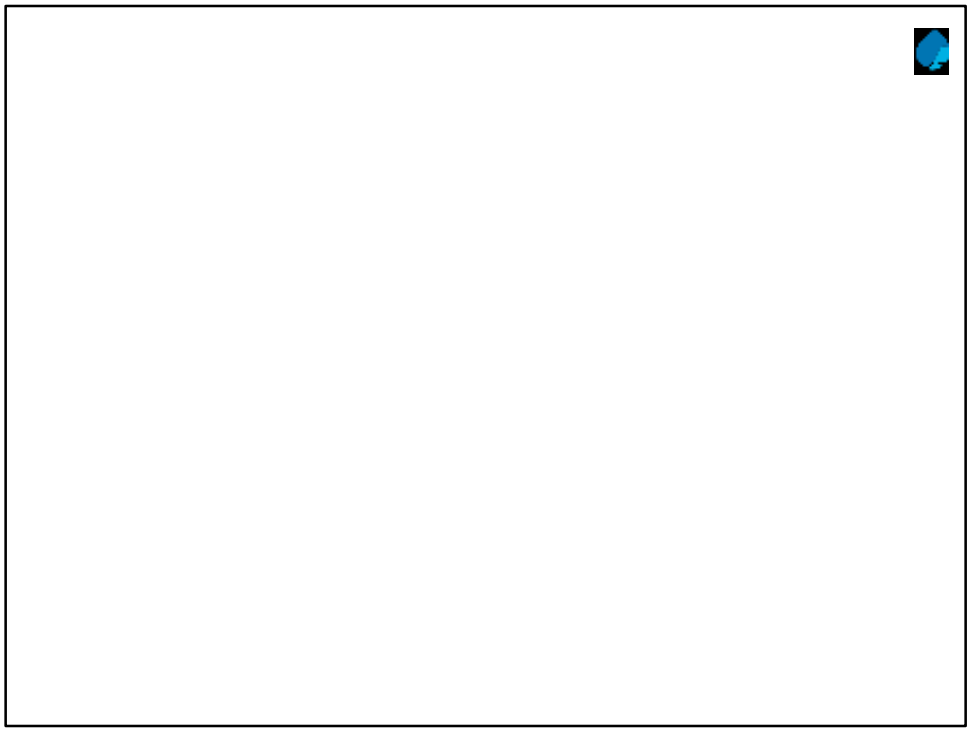
Transport Options

- ASP.NET SignalR relies on transport layers to make communication possible between clients and servers.
- They are categorized as
  - Comet Approaches
    - Long Pooling
    - Forever Frame
  - HTML5 Approaches
    - WebSockets
    - Server Sent Events

Long Polling
This famous Comet approach is one of the most common ways to achieve real–time web development in today's Internet. It relies on using JavaScript (or other technologies and techniques) to make a lightweight HTTP connection to a server and keep it open for a period of time (e.g., 30 seconds). In this interval, if new data becomes available on the server, it puts the data in the currently open HTTP connection and then closes it. The client receives this data and opens a new long poll connection immediately. If there is no data in that period of time, the client establishes a new long poll connection and continues this as long as the page is open and active in a web browser. Long polling follows this simple model and reduces the overhead of opening several connections to servers that could be introduced by an interval polling approach. Interval polling is the traditional approach in which the server is checked at regular intervals (e.g., every 10 seconds) for new content. This approach is not efficient; opening and closing HTTP connections to web servers create overhead. Long polling tries to reduce this overhead.

Forever Frame
Another Comet approach that is less common than long polling is *Forever Frame* (also known as *hidden iframe*), which is an approach specific to Internet Explorer. In this approach, a hidden iframe element is attached to each web page that is kept open for the whole duration of the request (the period of time that a user stays on the page). As data becomes available on the server, it is filled with JavaScript codes in a stacked manner, and because browsers execute these scripts in order, they can provide the desired behavior needed.

Instructor Notes:



Server-Sent Event
This approach is a modern HTML5 transport that enables web browsers to receive events from servers through an
HTTP connection. The main limitation of server-sent events (like any other approach based on HTML5) is browser support.
Although the most recent versions of common browsers support server-sent events, Internet Explorer does not support them, which leaves a big gap for those interested in using these events for real–time web development.

WebSockets
The other HTML5 approach that has better browser support is WebSockets, which provides two-way communication channels over a TCP connection. WebSockets is not limited to HTTP, although it can also be used in that context. The use of WebSockets is dependent on the support by the underlying operating system (among Microsoft server technologies, only Windows Server 2012 supports them, although it is also available on Windows 8 and 8.1) and the whole network infrastructure between the client and server. Of course, browser support is also needed to take advantage of WebSockets, and recent versions of most common web browsers support it.

Instructor Notes:

Communication Models

- The SignalR API contains two models for communicating between clients and servers
  - Persistent Connections
  - Hubs

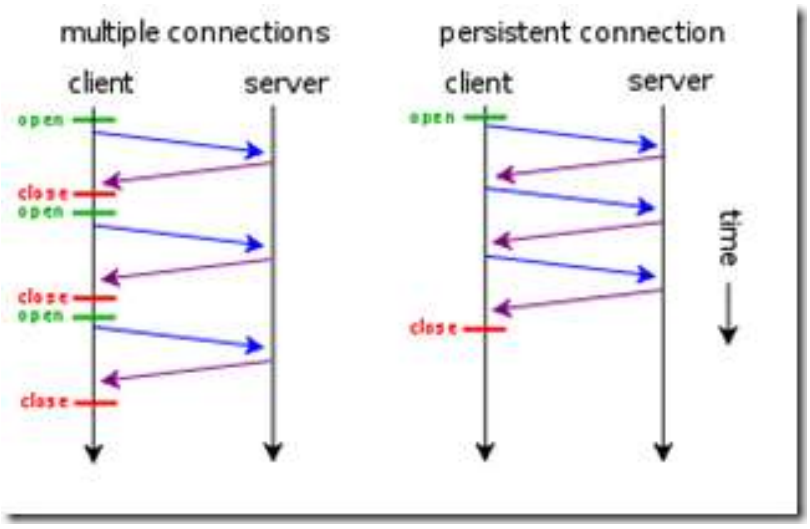
Instructor Notes:

Communication Models

- Persistent Connections:-
  - A Connection represents a simple endpoint for sending single-recipient, grouped, or broadcast messages.
  - The Persistent Connection API (represented in .NET code by the PersistentConnection class) gives the developer direct access to the low-level communication protocol that SignalR exposes.
  - Using the Connections communication model will be familiar to developers who have used connection-based APIs such as Windows Communication Foundation.

Instructor Notes:

Communication Model



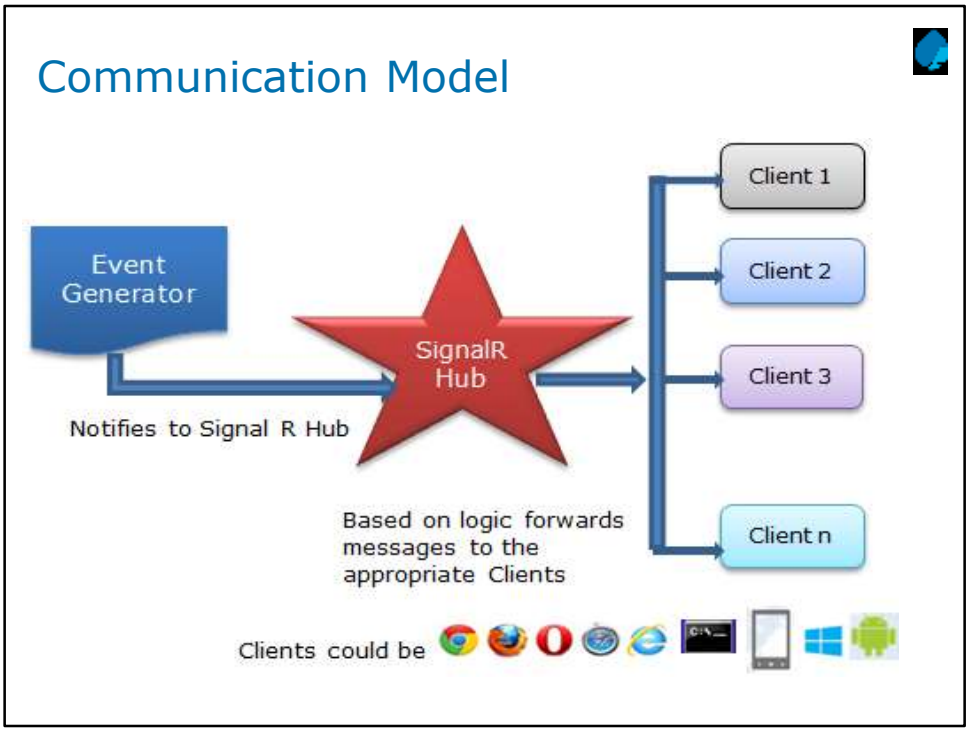
HTTP persistent connection, also called HTTP keep-alive, or HTTP connection reuse, is the idea of using a single TCP connection to send and receive multiple HTTP requests/responses, as opposed to opening a new connection for every single request/response pair. The newer HTTP/2 protocol uses the same idea and takes it further to allow multiple concurrent requests/responses to be multiplexed over a single connection

Instructor Notes:

Communication Model

- Hub
  - A Hub is a more high-level pipeline built upon the Connection API that allows your client and server to call methods on each other directly.
  - SignalR handles the dispatching across machine boundaries as if by magic, allowing clients to call methods on the server as easily as local methods, and vice versa.
  - Using the Hubs communication model will be familiar to developers who have used remote invocation APIs such as .NET Remoting.
  - Using a Hub also allows you to pass strongly typed parameters to methods, enabling model binding.

Instructor Notes:



Instructor Notes:

How Hub Works

- When server-side code calls a method on the client, a packet is sent across the active transport that contains the name and parameters of the method to be called (when an object is sent as a method parameter, it is serialized using JSON).
- The client then matches the method name to methods defined in client-side code.
- If there is a match, the client method will be executed using the deserialized parameter data.

Instructor Notes:

How Hub Works(Contd..)

- The method call can be monitored using tools like Fiddler.
- The following image shows a method call sent from a SignalR server to a web browser client in the Logs pane of Fiddler.

```
10:38:03:1298 Session846.WebSocket'WebSocket #846' - Pushing 104 bytes from server WebSocket
81 66 7B 22 43 22 3A 22 42 2C 31 35 7C 43 2C 30     f{"C":"B,15|C,0
7C 44 2C 30 7C 45 2C 30 22 2C 22 4D 22 3A 5B 7B  |D,0|E,0","M":[{
22 48 22 3A 22 4D 6F 76 65 53 68 61 70 65 48 75  "H":"MoveShapeHu
62 22 2C 22 4D 22 3A 22 75 70 64 61 74 65 53 68  b","M":"updateSh
61 70 65 22 2C 22 41 22 3A 5B 7B 22 6C 65 66 74  ape","A":[{"left
22 3A 35 30 31 2E 30 2C 22 74 6F 70 22 3A 33 30  ":501.0,"top":30
32 2E 30 7D 5D 7D 5D 7D                          2.0}]}]}
```

Instructor Notes:

Choosing a Communication Model

- Most applications should use the Hubs API. The Connections API could be used in the following circumstances:
  - The format of the actual message sent needs to be specified.
  - The developer prefers to work with a messaging and dispatching model rather than a remote invocation model.
  - An existing application that uses a messaging model is being ported to use SignalR.

Instructor Notes:

Sample Hub Code

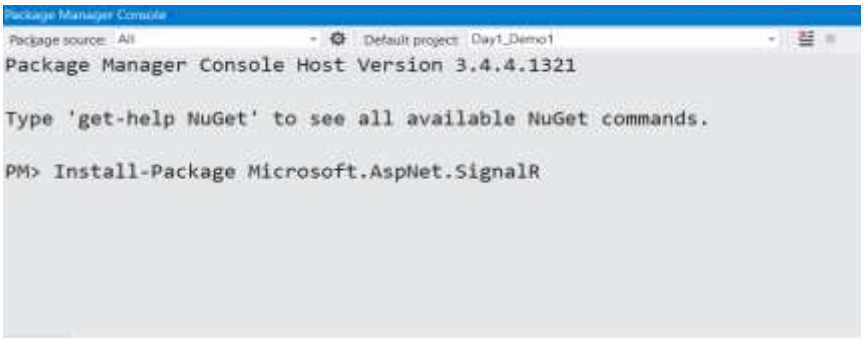
```
using System.Web;
using Microsoft.AspNet.SignalR;

namespace Day1_Demo1
{
    0 references
    public class ChatHub : Hub
    {
        0 references
        public void Hello()
        {
            Clients.All.hello();
        }
    }
}
```

Instructor Notes:

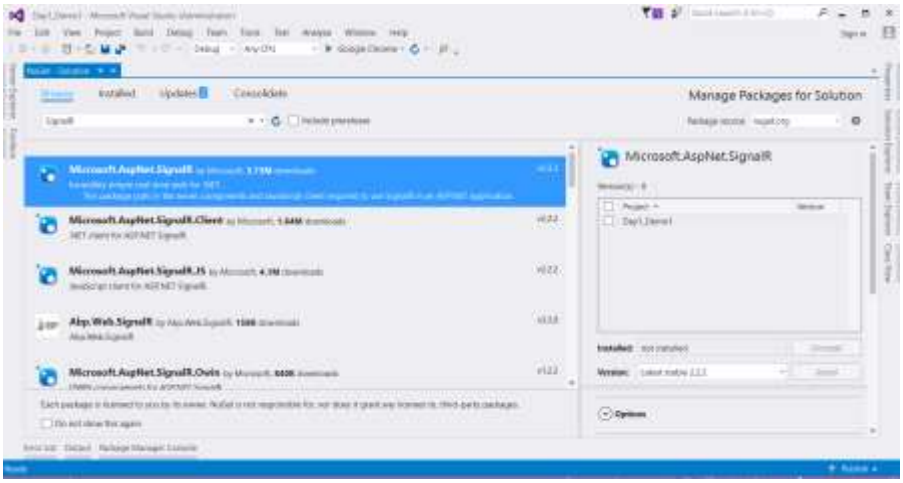
SignalR API

- SignalR API is distributed vai Nuget Package.
- We can use Nuget Package Manager or Nuget Package Manager Console to add the SingalR libraries to the project.



Instructor Notes:

Adding SignalR using Nuget Package Manager



Instructor Notes:

Demos

- Creating a Simple Chat SignalR Application
- Creating a SingalR application using MVC