**ASP.NET SignalR**

[ASP.NET Core SignalR](https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-2.2) 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.

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

**Transports**

SignalR supports several techniques for handling real-time communications:

* WebSockets
* Server-Sent Events
* Long Polling

SignalR automatically chooses the best transport method that is within the capabilities of the server and client.

**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.

**Implementing SignalR on .NET Core**

To get started we have to install Microsoft.AspNetCore.SignalR package from *Nuget package manager.*

Then add the SignalR in the startup class





Now we have to define the **Hub.**

A hub is a class that serves as a high-level pipeline that handles client-server communication.



Here MessageHub inherits from Microsoft.AspNetCore.SignalR.Hub. This Hub class manages connections, groups and messaging.

Here we don’t define any methods in the MessageHub class since we are just providing one-way communication (*Server sending the data to all clients*) and we can access all hub methods using [IHubContext<>](https://docs.microsoft.com/en-us/aspnet/core/signalr/hubcontext?view=aspnetcore-2.2) interface.

Now the server Hub is set up. To send all data to the clients, we use IHubContext<> interface.

First we have to initialize the IHubContext<> interface in the service class where we are sending the data to the socket (In our case, API service class)



Now we can use this IHubContext<> (*we need to initialize the instance using constructor Injection*) to send the data to the socket from the service class (*from the method for Mutation in our case*).



With this we are sending the *data* to all subscribed clients on the *BroadcastMessage* event.

This means that every client if it has a listener on the *BroadcastMessage* event, will receive the *data*.

**Using SignalR socket from client side**

We use [@aspnet/signalr](https://www.npmjs.com/package/@aspnet/signalr) package to use SignalR from angular application.



We use HubConnectionBuilder() to connect our application to the SignalR hub.



Now we use this hubConnection to listen to the event and thereby reading the data sent to the event.



