**SignalR in ASP .NET Core 3.1**

By Shahed C on May 11, 2020

[5 Replies](https://wakeupandcode.com/signalr-in-asp-net-core-3-1/#comments)



This is the nineteenth of a new [series of posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2020) on ASP .NET Core 3.1 for 2020. In this series, we’ll cover 26 topics over a span of 26 weeks from January through June 2020, titled **ASP .NET Core A-Z!** To differentiate from the [2019 series](https://wakeupandcode.com/aspnetcore/#aspnetcore2019), the 2020 series will mostly focus on a growing single codebase ([NetLearner!](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/)) instead of new unrelated code snippets week.

Previous post:

* [Razor Pages in ASP .NET Core 3.1](https://wakeupandcode.com/razor-pages-in-asp-net-core-3-1/)

**NetLearner on GitHub**:

* Repository: <https://github.com/shahedc/NetLearnerApp>
* v0.19-alpha release: <https://github.com/shahedc/NetLearnerApp/releases/tag/v0.19-alpha>

**In this Article:**

* S is for SignalR
* Dependencies
* Server-Side Hub
* Client-Side
* Configuration
* Running the App
* Azure SignalR Service
* Packaging Changes in 3.x
* References

**S is for SignalR**

SignalR has been around for 7+ years now, allowing ASP .NET developers to easily include real-time features in their web applications. SignalR Core has been available in ASP .NET Core since v2.1, as a cross-platform solution to add real-time features to web apps and more!

In this article, we’ll go over SignalR concepts, using a new sample I developed to allow web users to vote in a real-time online poll. Before you begin, take a look at the sample code project on GitHub:

* SignalR Poll project on GitHub: <https://github.com/shahedc/NetLearnerApp/tree/master/experimental/NetLearner.SignalRPoll>

Back in 2018, I ran a couple of polls on [Facebook](https://www.facebook.com/groups/637731783071479/permalink/1153241031520549/) and [Twitter](https://twitter.com/shahedC/status/1074862352787492864) to see what the dev community wanted to see. On Twitter, the #1 choice was “Polling/Voting app” followed by “Planning Poker App” and “Real-time game”. On Facebook, the #1 choice was “Real-time game” followed by “Polling/voting app”. As a result, I decided to complement this article with a polling sample app.

* Twitter poll results: <https://twitter.com/shahedC/status/1074862352787492864>

POLL: What kind of [#AspNetCore](https://twitter.com/hashtag/AspNetCore?src=hash&amp;ref_src=twsrc%5Etfw) [#SignalR](https://twitter.com/hashtag/SignalR?src=hash&amp;ref_src=twsrc%5Etfw) apps would you like to see with <https://t.co/lbmtwAyC6P> Core 2.x? [#webdev](https://twitter.com/hashtag/webdev?src=hash&amp;ref_src=twsrc%5Etfw) [#webapps](https://twitter.com/hashtag/webapps?src=hash&amp;ref_src=twsrc%5Etfw) [#realtime](https://twitter.com/hashtag/realtime?src=hash&amp;ref_src=twsrc%5Etfw)

— Shahed Chowdhuri @ Microsoft (@shahedC) [December 18, 2018](https://twitter.com/shahedC/status/1074862352787492864?ref_src=twsrc%5Etfw)

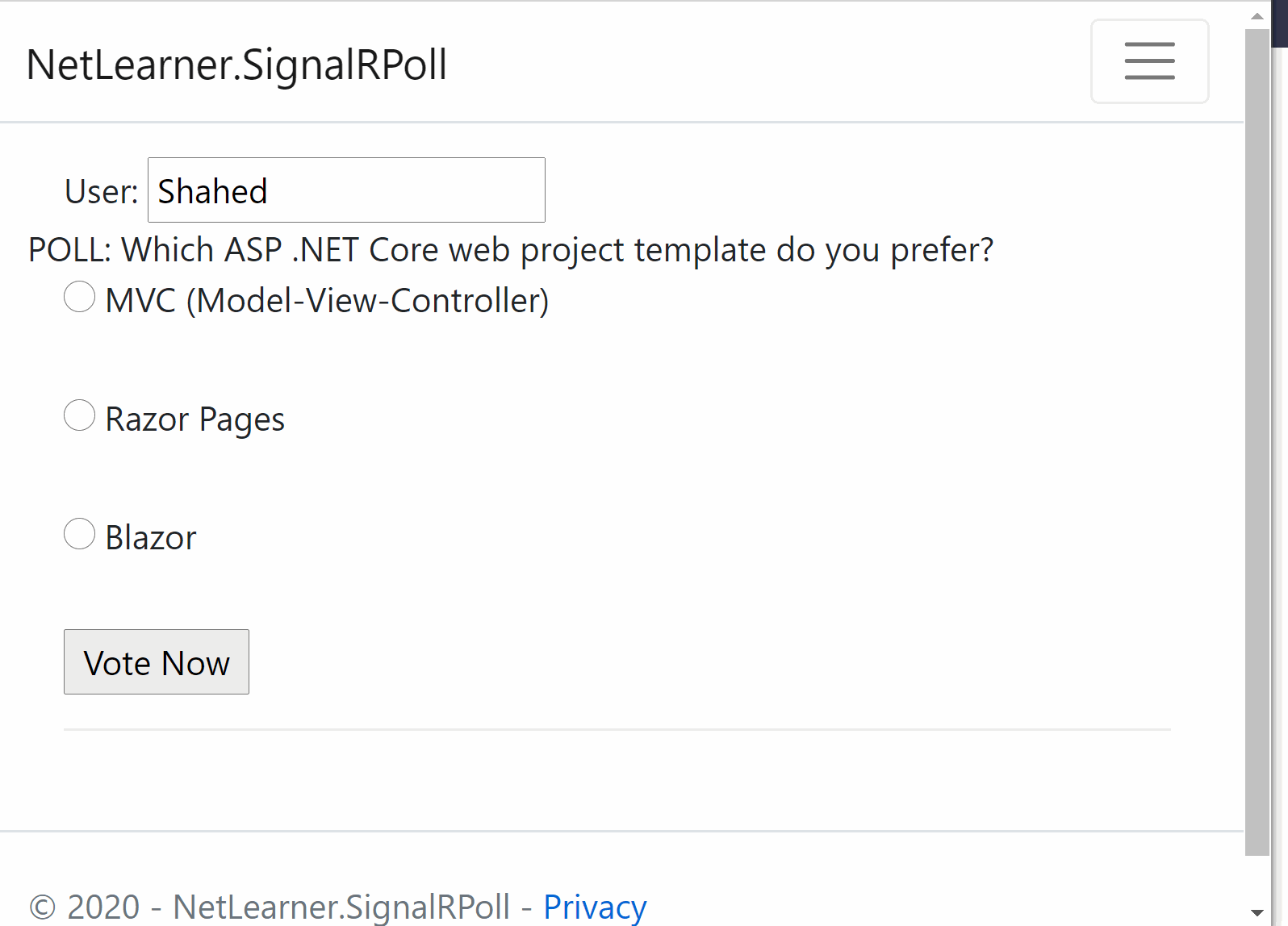
More importantly, [Brady Gaster](https://twitter.com/bradygaster/status/1075060352683933696) suggested that the sample app should definitely be “Not. Chat.” 🙂

* “Not. Chat.”: <https://twitter.com/bradygaster/status/1075060352683933696>

Not. Chat.

— Brady Gaster (@bradygaster) [December 18, 2018](https://twitter.com/bradygaster/status/1075060352683933696?ref_src=twsrc%5Etfw)

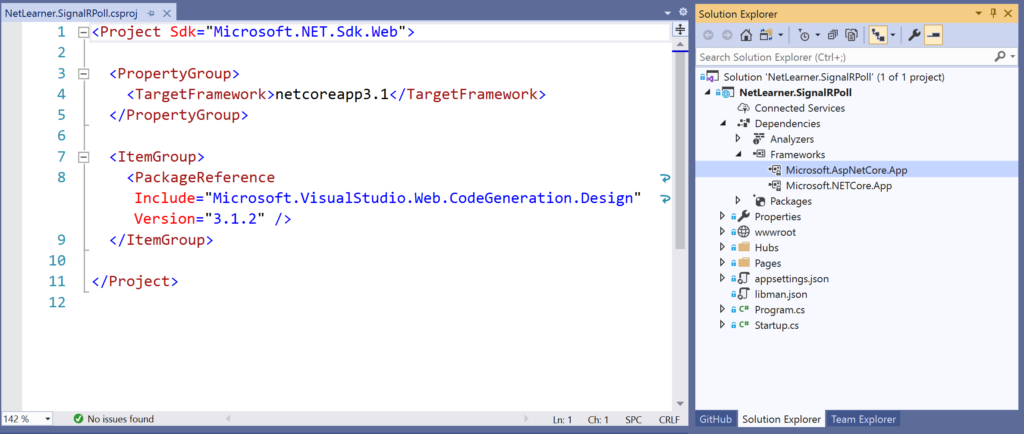
In the sample project, take a look at the **SignalRPoll** project to see how the polling feature has been implemented. In order to create a project from scratch, you’ll be using both server-side and client-side dependencies.

SignalR poll in action

If you need a starter tutorial, check out the official docs:

* Get started with ASP.NET Core SignalR: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/signalr>

**Dependencies**

Visual Studio showing csproj + dependencies

The **Server-Side dependencies** for SignalR Core are available via the Microsoft.AspNetCore.App package so this is a freebie when you create a new 3.1 web app project. In your server-side code, you can use the following namespace:

using Microsoft.AspNetCore.SignalR;

This will give you access to SignalR classes such as Hub and Hub<T> for your SignalR hub to derive from. In the sample project, the [PollHub class](https://github.com/shahedc/NetLearnerApp/blob/master/experimental/NetLearner.SignalRPoll/Hubs/PollHub.cs) inherits from the Hub class. Hub<T> can be used for strongly-typed SignalR hubs.

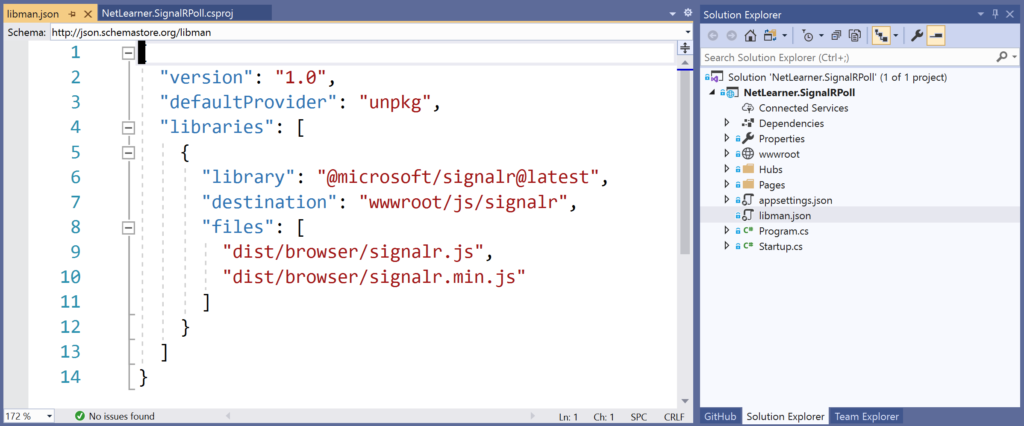
The **Client Side dependencies** for SignalR Core have to be added manually. Simply right-click on your web app project and select Add | Client-Side Library. In the popup that appears, select a provider (such as “unpkg”) and enter a partial search term for Library, so that you can ideally pick the latest stable version.

Steps to add client library via LibMan (aka Library Manager):

* Right-click project in Solution Explorer
* Select Add | Client-Side Library

In the popup that appears, select/enter the following:

* **Provider**: choose from cdnjs, filesystem, unpkg
* **Library** search term: @aspnet/signalr@1… pick latest stable if desired
* **Files**: At a minimum, choose specific files signalr.js and/or its minified equivalent

Visual Studio, showing libman.json with client-side references

If you need help with adding client-side references, check out this earlier blog post in this A-Z series:

* JavaScript, CSS, HTML & Other Static Files in ASP .NET Core 3.1: <https://wakeupandcode.com/javascript-css-html-other-static-files-in-asp-net-core-3-1/>

**Server-Side Hub**

In the sample app, the PollHub class has a simple **SendMessage()** method with a few parameters. Derived from the sample Chat application, it starts with the user’s desired “user” value and a custom “message” that can be passed to the SignalR Hub. For the the Captain Marvel/America poll, the method also passes an Id and Value for the selected radio button.

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

To ensure that the SendMessage method from the server has a trigger on the client-side, the client-side code must invoke the method via the SignalR connection created with **HubConnectionBuilder**() on the client side. Once called, the above code will send a call to ReceiveMessage on all the clients connected to the Hub.

**Client-Side**

On the client-side, the JavaScript file [poll.js](https://github.com/shahedc/NetLearnerApp/blob/master/experimental/NetLearner.SignalRPoll/wwwroot/js/poll.js) handles the call from the browser to the server, and receives a response back from the server as well. The following code snippets highlight some important areas:

var connection = new signalR.HubConnectionBuilder().withUrl("/pollHub").build();   
...   
connection.on("ReceiveMessage", function (user, message, myProjectId, myProjectVal) {   
 ...   
 document.getElementById(myProjectId + 'Block').innerHTML += chartBlock;  
});   
...  
  
document.getElementById("sendButton").addEventListener("click", function (event) {  
 ...  
 connection.invoke("SendMessage", user, message, myProjectId, myProjectVal)  
 ...   
});

The above snippets takes care of the following:

1. Creates a new connection objection using HubConnectionBuilder with a designated route
2. Uses connection.on to ensure that calls to ReceiveMessage come back from the server
3. Sets the innerHTML of a <span> block to simulate a growing bar chart built with small blocks
4. Listens for a click event from the sendButton element on the browser
5. When the sendButton is clicked, uses connection.invoke() to call SendMessage on the server

**Configuration**

The configuration for the SignalR application is set up in the [Startup.cs](https://github.com/shahedc/NetLearnerApp/blob/master/experimental/NetLearner.SignalRPoll/Startup.cs) methods ConfigureServices() and Configure(), as you may expect.

public void ConfigureServices(IServiceCollection services)   
{  
 ...  
 services.AddSignalR();  
}  
  
public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
{  
 ...  
 app.UseEndpoints(endpoints =>  
 {  
 ...   
 endpoints.MapHub<PollHub>("/pollHub");  
 });  
 ...  
}

The above code takes care of the following:

1. the **ConfigureServices**() method adds SignalR to the ASP.NET Core dependency injection system with a call to **AddSignalR**()
2. the **Configure**() method adds SignalR to the middleware pipeline, while setting up the necessary route(s), using a call to **UseSignalR**().

At the time of this writing, I have more than one route set up for multiple hubs. For the polling app, we only need the call to **MapHub**<**PollHub**>() that sets up the route “/**pollHub**“. You may recall this route from the client-side JavaScript code where the initial connection is set up.

For streaming fragments of data over time, you should also take a look at Streaming in SignalR Core:

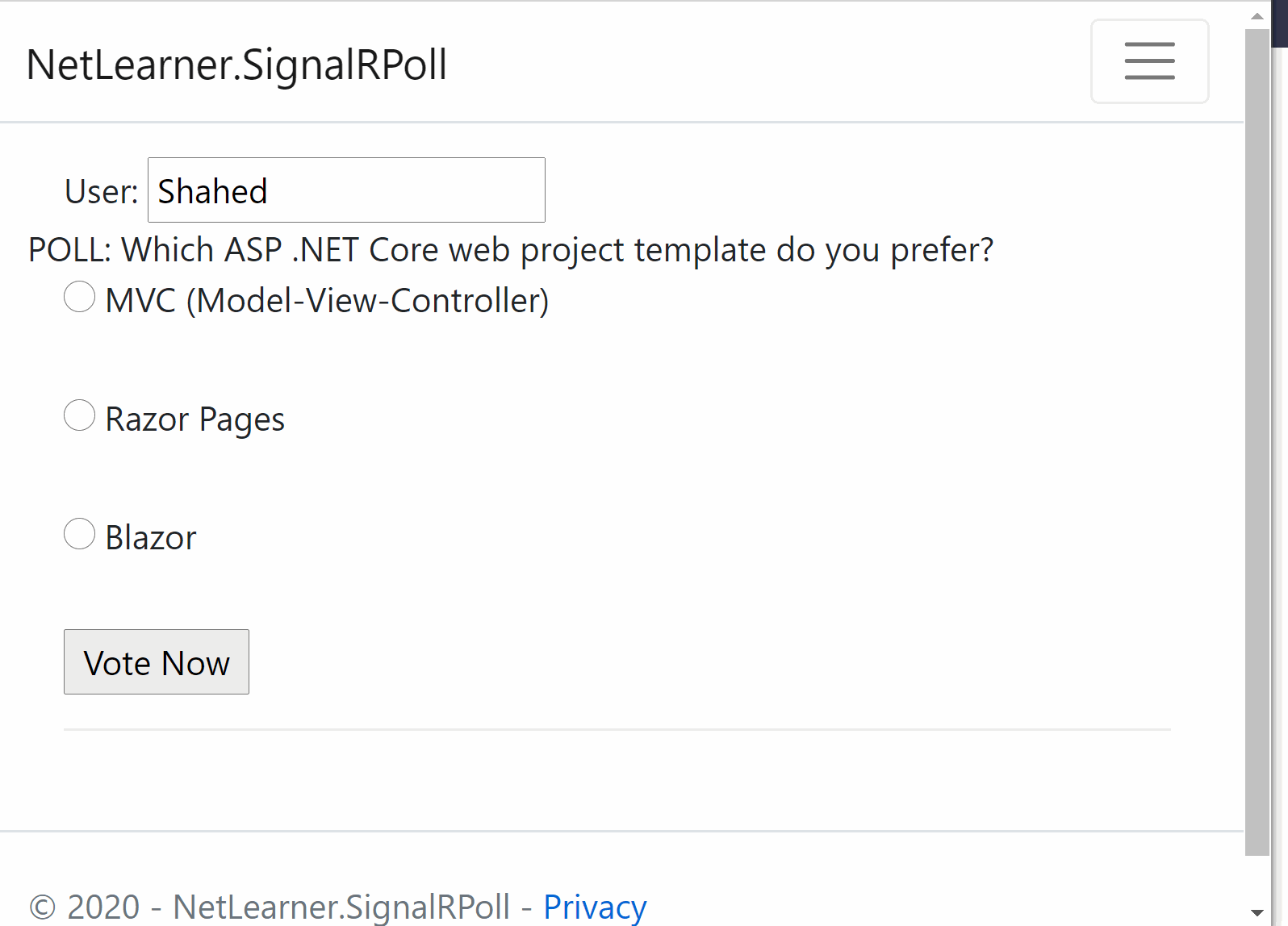
* Use streaming in ASP.NET Core SignalR: <https://docs.microsoft.com/en-us/aspnet/core/signalr/streaming>

**Running the App**

To run the app, simply run the SignalRPoll app Visual Studio or from the command line. Then, click the Poll item in the top menu to go to the [Poll page](https://github.com/shahedc/NetLearnerApp/blob/master/experimental/NetLearner.SignalRPoll/Pages/Poll/Index.cshtml). This page is a simple Razor page that contains all the HTML elements necessary to display the poll. It also includes <script> references to jQuery, SignalR and poll.js client-side references.

**NOTE**: Even though I am using jQuery for this sample, please note that jQuery is not required to use SignalR Core. On a related note, you can also [configure Webpack and TypeScript](https://docs.microsoft.com/en-us/aspnet/core/tutorials/signalr-typescript-webpack) for a TypeScript client if you want.

This GIF animation below illustrates the poll in action. To record this GIF of 1 browser window, I also launched additional browser windows (not shown) pointing to the same URL, so that I could vote several times.

SignalR poll in action

In a real world scenario, there are various ways to prevent a user from voting multiple times. Some suggestions include:

* Disable the voting button as soon as the user has submitted a vote.
* Use a cookie to prevent the user from voting after reloading the page.
* Use authentication to prevent a user from voting after clearing cookies or using a different browser.

For more information on authenticating and authorizing users, check out the official docs:

* Authentication and authorization in ASP.NET Core SignalR: <https://docs.microsoft.com/en-us/aspnet/core/signalr/authn-and-authz>

**Azure SignalR Service**

Azure SignalR Service is a fully-managed service available in Microsoft’s cloud-hosted Azure services, that allows you to add real-time functionality and easily scale your apps as needed. Using Azure SignalR Service is as easy as 1-2-3:

1. Add a reference to the Azure SignalR Service SDK
2. Configure a connection string
3. Call services.AddSignalR().AddAzureSignalR() and app.UseAzureSignalR in Startup.cs

For more information on Azure SignalR Service, check out the official docs and tutorials:

* What is Azure SignalR: <https://docs.microsoft.com/en-us/azure/azure-signalr/signalr-overview>
* C# Quickstart: <https://docs.microsoft.com/en-us/azure/azure-signalr/signalr-quickstart-dotnet-core>

**Packaging Changes in 3.x**

You may have heard that ASP .NET Core 3.0 changed the way packages are made available to developers. So how does this affect SignalR for 3.x projects? Here is a recap from the [official announcement](https://github.com/aspnet/Announcements/issues/325):

* Microsoft “will stop producing many of the NuGet packages that we have been shipping since ASP.NET Core 1.0. The API those packages provide are still available to apps by using a <FrameworkReference> to Microsoft.AspNetCore.App. This includes commonly referenced API, such as Kestrel, Mvc, Razor, and others.”
* “This will not apply to all binaries that are pulled in via Microsoft.AspNetCore.App in 2.x.”
* “***Notable exceptions include***: The SignalR .NET client will continue to support .NET Standard and ship as NuGet package because it is intended for use on many .NET runtimes, like Xamarin and UWP.”

**Source**: <https://github.com/aspnet/Announcements/issues/325>

**References:**

* Intro to ASP.NET Core SignalR: <https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction>
* Get started with ASP.NET Core SignalR: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/signalr>
* Create backend services for native mobile apps with ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/mobile/native-mobile-backend>
* Use ASP.NET Core SignalR with TypeScript and Webpack: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/signalr-typescript-webpack>
* SignalR Service C# Quickstart: <https://docs.microsoft.com/en-us/azure/azure-signalr/signalr-quickstart-dotnet-core>