

|  |
| --- |
| Werkstuk webapplicaties SignalR  Ouassim Abdellaoui &  Graduaat Informatica Programmeren  Webapplicaties |

Inhoudstafel

[Inhoudstafel 2](#_Toc133767090)

[1 Inleiding 3](#_Toc133767091)

[2 Aanpak 3](#_Toc133767092)

[2.1 Verwerking van behandelde research 3](#_Toc133767093)

[2.2 Manier van werken 3](#_Toc133767094)

[2.3 Bronnen 3](#_Toc133767095)

[3 Wat is SignalR? 4](#_Toc133767096)

[3.1 Intro 4](#_Toc133767097)

[3.2 Ontstaan 5](#_Toc133767098)

[3.3 Gebruik 6](#_Toc133767099)

[3.4 Waarom SignalR gebruiken? 7](#_Toc133767100)

[3.5 Belangrijkste features SignalR 7](#_Toc133767101)

[3.5.1 Verbindingsbeheer gebeurt automatisch 7](#_Toc133767102)

[3.5.2 Berichten naar alle, specifieke of groep actieve gebruikers 7](#_Toc133767103)

[3.5.3 Schaalt naar trafiek 8](#_Toc133767104)

[3.5.4 SignalR Hub Protocol 9](#_Toc133767105)

[3.6 WebSocket vs SignalR 9](#_Toc133767106)

[3.7 Voor- en nadelen 10](#_Toc133767107)

[3.7.1 Voordelen 10](#_Toc133767108)

[3.7.2 Nadelen 10](#_Toc133767109)

[3.8 Alternatieven SignalR 11](#_Toc133767110)

[3.8.1 Ably serverless WebSocket platform 11](#_Toc133767111)

[3.8.2 Azure Web PubSub 11](#_Toc133767112)

[3.8.3 Pusher 11](#_Toc133767113)

[3.8.4 Socket.IO 11](#_Toc133767114)

[3.9 Toekomst 11](#_Toc133767115)

[4 Reflectie 11](#_Toc133767116)

# Inleiding

In dit werkstuk voor het vak webapplicaties gaan wij het hebben over SignalR.

Beginnen doen we met onze aanpak. Hoe zijn we tot dit werkstuk gekomen? Welke research hebben we allemaal gedaan. Van waar hebben wij onze informatie gehaald en hoe hebben we deze informatie behandeld?

Daarna gaan we dieper in op wat SignalR juist is. Wat is SignalR juist? Waarvoor wordt het gebruikt en hoe wordt dit geïmplementeerd binnen ASP.NET? We proberen de materie zo goed mogelijk over te brengen zodat onze medestudenten de functies van SignalR begrijpen en dit ook kunnen gebruiken in hun toekomstige projecten. Met beeldmateriaal en voorbeelden hopen wij dat SignalR net iets duidelijker wordt.

Eindigen doen we met een reflectie. Hoe verliep de samenwerking in onze groep?

Wat we allemaal hebben bijgeleerd en of er eventueel ook dingen zijn die we beter anders hadden aangepakt.

# Aanpak

## Verwerking van behandelde research

## Manier van werken

## Bronnen

<https://learn.microsoft.com/nl-nl/aspnet/core/tutorials/signalr?WT.mc_id=dotnet-35129-website&view=aspnetcore-7.0&tabs=visual-studio>

<https://learn.microsoft.com/nl-nl/aspnet/core/signalr/introduction?WT.mc_id=dotnet-35129-website&view=aspnetcore-7.0>

<https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>

<https://learn.microsoft.com/en-us/aspnet/core/signalr/scale?view=aspnetcore-7.0>

<https://www.tutorialspoint.com/what-is-signalr-and-how-to-use-it>

<https://www.c-sharpcorner.com/UploadFile/abhijmk/what-why-and-how-about-signalr/>

<https://kb.2pintsoftware.com/help/signalr-and-connection-management>

<https://en.wikipedia.org/wiki/WebSocket#:~:text=WebSocket%20is%20a%20computer%20communications,as%20RFC%206455%20in%202011>.

<https://scientificprogrammer.net/2022/09/24/sending-messages-to-individual-signalr-clients-or-groups-of-clients/>

<https://scientificprogrammer.net/2022/05/07/signalr-on-net-6-the-complete-guide/>

<https://infinum.com/blog/scaling-out-your-own-signalr-chat-application/>

<https://medium.com/yemeksepeti-teknoloji/what-is-server-sent-events-sse-and-how-to-implement-it-904938bffd73>

<https://www.codemag.com/article/1807061/Build-Real-time-Applications-with-ASP.NET-Core-SignalR>

<https://ably.com/topic/signalr-vs-websocket>

<https://medium.com/geekculture/ajax-polling-vs-long-polling-vs-websockets-vs-server-sent-events-e0d65033c9ba#:~:text=Polling%20is%20a%20standard%20technique,an%20empty%20response%20is%20returned>.

<https://www.codemag.com/article/1807061/Build-Real-time-Applications-with-ASP.NET-Core-SignalR>

<https://scientificprogrammer.net/2019/08/18/why-you-absolutely-need-signalr-for-asp-net-development/>

https://ably.com/periodic-table-of-realtime/signalr

# Wat is SignalR?

## Intro

SignalR is een open-source library in ASP.NET die voor realtime webfunctionaliteit zorgt. Maar wat bedoelen we hier juist mee?

Wanneer je als client gebruikmaakt van een website of webapplicatie, dan wordt er data opgehaald van of verstuurd naar een server. Telkens wanneer je dus data opvraagt of verstuurt maak je gebruik van de server. De communicatie tussen client en server gebeurt met HTTP. HTTP kan je best zien als “verzoeken” tussen een gebruiker en server om communicatie te voorzien. Het HTTP verstuurt een request (vaak een POST of GET methode). De server leest je aanvraag, verwerkt dit en verstuurt, naargelang je verzoek, een antwoord terug. Dit heet het HTTP request-response model, via de request krijg je dus een antwoord van de server. Meestal gaat dit via een REST API.

Zoals in de onderstaande afbeelding is te zien gebruiken John en Mary een chatapplicatie. John stuurt een bericht via een POST methode naar de server. Omdat het een POST request is blijft het bericht alleen opgeslagen in de database, er gebeurt voor de rest niks.

Als Mary het bericht wil lezen zou zij eerst een GET-request moeten sturen. Mary weet niet wanneer John een bericht heeft verstuurd. Mocht zij het zo willen dat haar chatapplicatie elke seconde data opvraagt bij de server, dan zou zij elke seconde een GET-request moeten sturen. Dit maakt het realtime maar is alles behalve efficiënt. Dit legt namelijk teveel druk op de performance van zowel de server als de achterliggende database.

Diagram

Description automatically generated

Zoals je dus in het voorbeeld kan zien is in een typische web-applicaties de communicatie in één richting, namelijk van de client naar de server. De client stuurt een verzoek naar de server, de server doet wat taken en stuurt een antwoord terug naar de client. Hier is het gebruik van SignalR dus handig. Met SignalR is het mogelijk om server-code te schrijven die meteen met de clienten communiceert. SignalR maakt gebruik van WebSocket API om berichten naar een server te sturen en meteen antwoorden terug te krijgen zonder dat er een pull moet gebeuren naar de server voor een antwoord. Met het gebruik van SignalR is het dus mogelijk om een interactieve communicatiesessie in twee richtingen te openen tussen een gebruiker en een server.

Enkele goede voorbeelden hiervan zijn de gamingindustrie, sociale media, veilingsites en GPS. Een bekend voorbeeld waarbij SignalR wordt gebruikt zijn Google documenten. Als 2 mensen dezelfde spreadsheet openen en persoon A begint te typen, kan persoon B dit meteen zien zonder zijn webpagina te verversen.

## Ontstaan

SignalR is in 2011 in het leven geroepen door David Fowler en Damian Edwards. Het werd toegevoegd in het ASP.NET project en uitgebracht als deel van ASP.NET in 2013.

In 2013 was het WebSocket protocol juist gestandaardiseerd maar ondersteunde de meeste browsers dit nog niet. WebSocket is hetgeen dat SignalR gebruikt om het mogelijk te maken om data tegelijkertijd in twee richtingen te sturen. Over WebSocket gaan we zo meteen dieper in.

Voor het bestaan van SignalR was er niet echt een goede methode voor real-time communicatie. Twee manieren die wel gebruikt werden waren AJAX polling en server-sent events, dat laatste werd bovendien bij het begin van SignalR nog niet ondersteund door de meeste browsers. Beide methodes waren redelijk omslachtig en hiervoor is SignalR dan ook in het leven geroepen.

Om het verschil duidelijk te maken leggen we beide even uit:

AJAX (long) polling is eigenlijk wat er gebeurt in ons eerste voorbeeld van John en Mary. Er worden constant HTTP calls naar de server gestuurd en de client wacht af tot de server iets terugstuurt. Als er geen data beschikbaar is wordt er gewoon een leeg antwoord gestuurd.

Met long-polling wordt er eigenlijk bijna hetzelfde gedaan. Het verschil is hier dat als er geen data beschikbaar is de server de request vasthoudt tot er iets is om terug te sturen. Dit wordt ook wel een “hanging GET” genoemd. Als er data beschikbaar wordt stuurt de server deze terug naar de client. De client stuurt dan meteen een request terug naar de server, waardoor de server bijna altijd wel een wachtende request heeft.

Server-sent events zijn events die door een browser gebruikt kunnen worden. Een browser “abonneert” zogezegd op een aantal events. Wanneer er zo’n event voorvalt stuurt de browser een notificatie naar de server. Zo kan er bepaalde code verstuurd worden telkens een specifiek event gebeurt. Denk hier aan alarmen, meldingen, nieuws of aan bijvoorbeeld sportscores. Stel dat een goal een event is. Elke keer er dus een goal gemaakt wordt stuurt de server data door naar de webbrowser waarna deze wordt geüpdate.

Diagram

Description automatically generated

## Gebruik

Er zijn twee versies van SignalR: ASP.NET SignalR en ASP.NET Core SignalR.

Het verschil hier is dat gewone ASP.NET de oude versie is. Toen SignalR in het begin uitkwam maakte bijna elke webapplicatie gebruikt van jQuery. Het was dus logisch dat SignalR ook gebruik maakte van de functionaliteiten van jQuery.

In 2016 is ASP.NET helemaal opnieuw geschreven en heeft SignalR ook een update gekregen. De grootste lijnen zijn wel hetzelfde geblevenn met enkele kleine aanpassingen. Omdat jQuery minder en minder werd gebruikt hebben ze de nieuwe SignalR versie niet meer afhankelijk gemaakt van jQuery. Hubs bleven wel het voornaamste verbindingspunt tussen servers en clienten. Clienten roepen nog altijd methodes op op deze hubs en de hubs hebben controle over de verbindingen.

Doordat het gebruik van jQuery niet meer noodzakelijk was werd de mogelijkheid om andere technologieën zoals JavaScript/TypeScript te gebruiken veel makkelijker. Hierdoor konden ook modernere frameworks zoals Angular, React, Vue,… gebruikt worden.

Nog iets dat nieuw was tegenover de standaard ASP.NET versie is het gebruik van dependency injections. Met dependency injections kan er code “van buitenaf” gebruikt worden. Je geeft een object bepaalde objecten mee die deze kan gebruiken. Een goed voorbeeld van dependency objections bij SignalR is de NavigationManager. Deze wordt mee geïnstalleerd met SignalR en kan gebruikt worden na zo’n inject.

SignalR gebruikt bovendien ook encryptie en een digitale handtekening om de verbindingstoken te beschermen. Voor elk verzoek valideert de server de inhoud van het token om er zeker van te zijn dat het verzoek afkomstig is van de opgegeven gebruiker. De gebruikersnaam moet overeenkomen met de connection ID.

Als we dan terug gaan naar het voorbeeld van John en Mary kunnen we het verschil aantonen met het gebruik van SignalR. John en Mary maken beide een connectie met de server en database. Telkens wanneer een gebruiker een bericht naar de server stuurt herkent de server de gebruiker en stuurt dit door naar de andere gebruiker. Deze verbinding blijft open tot John of Mary de verbinding sluit.

Diagram

Description automatically generated

## Waarom SignalR gebruiken?

Zoals daarjuist in het voorbeeld te zien was, is de SignalR methode veel efficienter, legt het geen druk op de performance van de server en biedt het realtime functionaliteit aan. SignalR is dus ideaal voor applicaties die vaak en snel updates moeten krijgen van de server. Zoals gaming, sociale media, veiling en GPS. Met de ingebouwde API die instaat voor de calls tussen server en gebruiker is het makkelijk om functies op te roepen die op de server in .NET Core code staan. Door het gebruik van hubs kunnen er berichten naar één, een groep of alle gebruikers verstuurd worden.

## Belangrijkste features SignalR

We weten nu dat SignalR instaat voor een bidirectionele open connectie tussen een webgebruiker en een server. Maar welke voordelen zijn er nog aan het gebruik van SignalR? We hebben hier de belangrijkste features opgelijst:

### Verbindingsbeheer gebeurt automatisch

SignalR gebruikt een API die instaat voor het verbindingsbeheer. Hiervoor wordt WebSocket gebruikt. WebSocket is communicatieprotocol dat het mogelijk maakt om in realtime data te sturen van en naar de server. Communicatie is consistent, bidirectioneel en de data kan tegelijk in twee richtingen gestuurd worden.

Moest SignalR geen gebruik maken van WebSocket zou je bij elke verbinding nog veel functionaliteiten extra moeten installeren (denk hier bijvoorbeeld aan oudere systemen die niet compatibel zijn met nieuwere technologieën). Door het gebruik van WebSocket worden deze automatisch geïmplementeerd. Over WebSocket vertellen we later meer.

### Berichten naar alle, specifieke of groep actieve gebruikers

SignalR gebruikt hubs om te communiceren tussen gebruikers en servers. Een hub is een pijplijn die toelaat dat een gebruiker en server methodes kunnen oproepen bij elkaar. Dit wordt automatisch gedaan door SignalR. Het zorgt voor de verbinding tussen de machines en de onderlinge communicatie tussen server en gebruiker of vice versa. Dit gebeurt in JSON of MessagePack, wat een compactere vorm van JSON is.

Deze hubs worden gebruikt op de serverzijde voor het versturen van berichten naar gebruikers. Het volstaat om een klasse aan te maken die over erft van Hub. Hierin kunnen verschillende methodes, taken, interfaces,… aangemaakt worden.

Voor berichten te sturen zijn er 3 methoden: SendMessage, SendMessageToCaller en SendMessageToGroup. SendMessage gebruikt de property Clients.All om naar alle gebruikers een bericht te sturen. SendMessageToCaller krijgt de id binnen van diegene die een bericht heeft gestuurd door de Clients.Caller property. Hiermee wordt een antwoord teruggestuurd naar die gebruiker. Als laatste is er nog SendMessageToGroup die SignalR Users groups gebruikt. Denk bijvoorbeeld aan chatruimtes waarbij elke ruimte een bepaalde group is. Over het gebruik van Hubs komt er zo meteen nog een uitgebreidere uitleg.

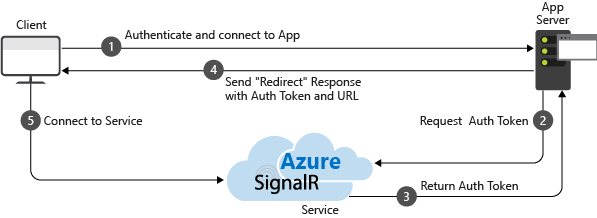
### Schaalt naar trafiek

Een applicatie dat gebruik maakt van SignalR moet steeds alle verbindingen bijhouden, wat problemen geeft bij meerdere servers. Stel dat één server een bericht wil sturen naar alle gebruikers. Dat bericht wordt alleen verzonden naar de gebruikers die met die ene server zijn verbonden. Gebruikers die met een andere server verbonden zijn krijgen dit bericht dus niet. Moest je het schalen van de gebruikers zelf willen implementeren zou dit veel tijd en vooral geld kosten. Om dat probleem op te lossen zijn er twee opties: Azure SignalR Service en Redis backplane.

Diagram

Description automatically generated

**Azure SignalR Service:** Azure verhelpt dit probleem door bij elke connectie van de gebruiker een omleiding te leggen naar de service. De gebruiker verbindt met de service zelf waardoor elke server maar een beperkt aantal connecties moet bijhouden. Azure SignalR Service biedt wereldwijd bereik aan, en is schaalbaar naar een miljoen verbindingen. Je moet natuurlijk wel een Azure abonnement hebben en je hubs worden gehost in de de cloud van Microsoft.



**Redis backplane:** Redis werkt met een publiceer/abonneer manier. Wanneer een gebruiker verbindt wordt deze informatie doorgegeven naar de “backplane” van Redis. Wanneer een server dan informatie wil sturen naar alle gebruikers wordt dit via het backplane gedaan.

Timeline

Description automatically generated with medium confidence

### SignalR Hub Protocol

SignalR maakt gebruik van hubs om de communicatie van de server naar cliënten en van cliënten naar server te doen. In de code definieer je methoden die via de server aangeroepen kunnen worden, SignalR handelt zelf de routing af.

Hieronder is een voorbeel van zo’n hub. Door middel van de methode “SendMessage” kan een gebruiker een bericht sturen. In deze methode staat ook al de methode waarop de andere gebruikers gaan luisteren naar nieuwe berichten, in dit geval is dit “ReceiveMessage”.

Text

Description automatically generated

In de code van de webpagina zelf is dus de methode “ReceiveMessage” toegevoegd. Deze moet dezelfde naam hebben als degene die in de hub is aangemaakt. Het luistert naar berichten waarin de ReceiveMessage wordt opgeroepen.

Text

Description automatically generated

## WebSocket vs SignalR

Daarjuist bij de features haalden we aan dat SignalR WebSocket gebruikt voor het verbindingsbeheer. Als we met WebSocket ook een bidirectionele verbinding kunnen bereiken, waarom dan SignalR gebruiken? Om het verschil uit te begrijpen moeten we eerst WebSocket wat uitleggen.

WebSocket maakt gebruik van HTTP (Hypertext Transfer Protocol). HTTP is het protocol dat zorgt voor de communicatie tussen webpagina en server. Deze communicatie bestaat uit requests en responses. WebSocket begint dus als HTTP-request. Het stuurt een GET-request en doet eerst een handshake met de server. Een handshake kan je zien als een signaal tussen de gebruiker en de server om een link op te zetten tussen beide. WebSocket maakt gebruik van de HTTP Upgrade Header in de HTTP-request om te upgraden naar WebSocket. Als de server daarna een “HTTP 101 Switching Protocols” terugstuurt is de bidirectionele WebSocket connectie succesvol.

SignalR maakt dus gebruik van WebSocket om de connectie open te houden.

Wat het gebruik van SignalR aantrekkelijker maakt is dat SignalR naast de bidirectionele verbinding ook automatische herverbinding heeft. Als een gebruiker zijn verbinding bij WebSocket verliest is dit moeilijker te achterhalen. Een methode die vaak gebruikt wordt heet de ‘ping pong’-methode. Er wordt ingesteld dat na een bepaalde tijd de gebruiker of de server een ping verstuurd naar de andere partij, deze antwoord dan met een pong. Als de pong niet binnen een vooraf bepaalde tijd ontvangen is, is de verbinding verbroken. Dit wil wel zeggen dat er op regelmatige tijdstippen een pong moet verstuurd worden, wat belasting geeft op de server.

SignalR doet dit op een andere en betere manier. Moest je je verbinding verliezen dan wordt deze niet direct gesloten. Omdat SignalR bepaalde informatie onthoudt, weet het genoeg om bij het verlies van connectie de verbinding opnieuw op te starten.

SignalR beslist bij het maken van de connectie ook zelf wat de beste methode voor de verbinding is. Moest WebSocket bijvoorbeeld niet beschikbaar zijn dan valt het automatisch terug op andere technologieën, zoals de server-sent event of longpulling die we eerder aanhaalden.

SignalR heeft dus de functionaliteiten van WebSocket maar biedt ook nog extra’s aan. Wat natuurlijk dan wel vanzelfsprekend is, is dat het minder tijd in beslag neemt om een WebSocket connectie te starten tegenover een SignalR connectie.

Of je nu beter SignalR of WebSocket gebruikt hangt van verschillende factoren af. WebSocket wordt meer gebruikt voor simpele scenarios, zoals bijvoorbeeld bij sportwebsite’s die op bepaalde momenten hun website moeten updaten. Wil je een chatapplicatie maken, waarbij automatische herverbinding een troef is, is het gebruik van SignalR aangeraden. Nog iets waar je rekening mee moet houden is de flexibiliteit van je applicatie. Met WebSocket is het makkelijker om je eigen protocollen te schrijven en te gebruiken met welke programmeertaal je maar wilt. Dit is natuurlijk weer moeilijker en tijd consumerend. SignalR heeft heel veel vooraf beschikbaar, wat natuurlijk makkelijker is maar hier heb je dan weer minder vrijheid om je eigen ding te doen.

## Voor- en nadelen

### Voordelen

* Meerdere opties om te schalen naargelang trafiek. Bijvoorbeeld Redis, Azure Service,…
* Maakt deel uit van het ASP.NET Framework. Hierdoor is het makkelijk te gebruiken in combinatie met andere ASP.NET functies zoals authenticatie, authorizatie en dependency injections.
* Zoals eerder aangegeven gebruikt SignalR WebSocket als dit beschikbaar is. Indien niet beschikbaar valt het terug op oudere technologieën zoals de Server-Sent Event en HTTP long polling.

### Nadelen

* Omdat er een open connectie is kunnen de events zeer snel achter elkaar opgeroepen worden. De kans bestaat dat de gebruiker niet altijd klaar is om berichten te ontvangen. Hierdoor bestaat de kans dat er berichten niet afgeleverd worden. SignalR kan +- 1000 berichten bufferen per gebruiker, worden dit er meer dan worden de berichten weggegooid. Eventuele ontvangstbevestigingen zullen dus zelf ingebouwd moeten worden.
* SignalR biedt maar een beperkt aantal software ontwikkelingspakketten aan: C#, Java, Python and Javascript. Modernere technologieën zoals iOS, Android, PHP worden niet ondersteund.
* SignalR is ontworpen voor single-region gebruik. Bij het uitbreiden van de regio kan dit ten koste gaan van de prestatie, betrouwbaarheid en beschikbaarheid.

## Alternatieven SignalR

### Ably serverless WebSocket platform

Zoals je van de naam al kan afleiden maakt Ably ook gebruik van WebSocket. Door middel van Amazon’s AWS kunnen er WebSocket API’s aangemaakt worden die instaan voor de verbinding. Voor er een verbinding kan gemaakt worden voert de Lambda API net zoals bij SignalR eerst authorizatie uit . Op hun website bieden ze servers aan die fouttolerant en vaak beschikbaar zijn. Ook is het mogelijk om je applicatie uit te breiden volledig naar je eigen wensen. Houd hier wel rekening mee dat er een prijskaartje aan vast hangt.

### Azure Web PubSub

Maakt ook gebruik van WebSocket. Extra is hier nog dat er Azure hosting, schaalbaarheid en load balancing wordt aangeboden. Gemakkelijk is hier natuurlijk wel dat andere Azure technologieën gebruikt kunnen worden. Ze geven ook weer aan van een hoge beschikbaarheid te hebben en dat ontwikkelaars volledig ondersteund worden in hun projecten. Met behulp van Azure services zijn dingen zoals spraakherkenning, machine learning en beveiliging snel ingebouwd.

### Pusher

Pusher biedt zowel voor mobiel als voor web bibliotheken aan die instaan voor de real-time verbinding. Flexibel, uitbreidbaar en gemakkelijk te onderhouden zijn Pusher’s verkooppunten. Pusher is onder andere beschikbaar in JavaScipt, PHP, .NET, Python en Java. Door events die Pusher zelf beschikbaar stelt is het aanmaken van applicaties met heel gemakkelijk. Een voorbeeld is het “new-price”-event. Telkens wanneer er op bijvoorbeeld een aandelen website dit event wordt aangeroepen verandert de prijs.

### Socket.IO

Maakt ook gebruik van WebSocket voor de connectie en bestaat uit twee delen: een server die gebruik maakt van NODE.JS en een gebruiker bibliotheek. Socket kan geïnstalleerd worden via de package manager en kan daarna in de code geïnitialiseerd worden. Het heeft net zoals SignalR een fallback systeem (polling en long-polling).

## Toekomst

Met de release van .NET 5 in 2020 heeft Microsoft gezegd dat er elk jaar een nieuwe .NET versie op de markt ging komen. Dat hebben we gezien met .NET 6 dat tijdens de coronacrisis (!) in november 2021 op de markt kwam en .NET 7 in november vorig jaar. In november 2023 komt .NET 8 dus uit. Microsoft is gefocust op .NET meer weerbaar en krachtig te maken door elk jaar relevante updates uit te voeren. In tussentijd krijgen de oudere systemen ook kleine updates wanneer deze het nodig hebben.

Wij vragen ons natuurlijk af of bij deze nieuwe versies er ook updates voor SignalR bij zitten. Iets het waard om met iets dat goed werkt te gaan spelen?

# Reflectie