**Introductie: Wat is SignalR en hoe werkt het?**

**SignalR** is een open-source library in ASP.NET die voor realtime webfunctionaliteit zorgt. Maar wat bedoelen wij hiermee precies?

Wanneer je als client gebruikmaakt van een website of webapplicatie, dan is het zo dat je bepaalde data wilt ophalen of bepaalde gegevens via de website wilt delen. Het bekijken van de meest recente content op social media en het sturen van berichten via een chatapplicatie zijn enkele voorbeelden van het ophalen en sturen van data.

Telkens wanneer je data stuurt of opvraagt bij een webapplicatie, dan communiceer je als het ware met de server van de webapplicatie. De communicatie tussen jou als webclient en de server gebeurt door het protocol HTTP. Met name, door middel van een HTTP request spreek je de server aan. Dit is dan vaak een POST of GET methode. Dit heet ook het **HTTP request-response model**; via een request krijg je een antwoord van de server. En over het algemeen gaat dit via een REST API.

**Voorbeeld HTTP request-response**

Zoals in de onderstaande afbeelding te zien is, maken John en Mary gebruik van een chatapplicatie. John stuurt een bericht via een POST methode naar de server. Echter, door de POST request blijft het bericht alleen opgeslagen in de database van de server. Er gebeurt dan niks anders.

Als Mary het bericht wil lezen, dan moet zij handmatig een GET request sturen. Mocht zij haar chatapplicatie zo willen hebben dat deze om elk seconde data opvraagt bij de server, dan zou zij een GET request moeten sturen die dan om elk seconde data ophaalt. Dit maakt het wel meer realtime, maar het is totaal niet efficient. Dit legt namelijk teveel druk op de performance van zowel de server als de achterliggende database.

**Afbeelding 1: Voorbeeld HTTP requests**

Diagram

Description automatically generated

Het voordeel van de HTTP request-response methode is dat deze eenvoudig aangemaakt kan worden. Alleen is dit niet efficient en ook niet realtime, want het liefst wil de gebruiker dat het bericht direct naar de ontvanger(s) gaat.

Dat is waar SignalR in beeld komt.

SignalR zorgt voor **realtime webfunctionaliteit**; de mogelijkheid voor de server om content naar verbonden clients direct te pushen, in plaats van dat de server wacht tot een client om nieuwe gegevens vraagt. Normaal moeten we telkens een nieuw HTTP request verzenden om met de server te communiceren, maar SignalR zorgt voor een permanente verbinding tussen de client en de server.

Als voorbeeld, stel er zijn 2 gebruikers die aan hetzelfde document werken in Google Docs. Wanneer een gebruiker wijzigingen aanbrengt, dan kan tegelijkertijd een andere gebruiker de wijzigingen zien zonder de pagina opnieuw te laden.

**Hoe werkt SignalR?**

SignalR gebruikt WebSocket-technologie om gegevens te verzenden. **WebSocket** is een HTML5 API die het mogelijk maakt om een interactieve communicatiesessie in twee richtingen te openen tussen de browser van de gebruiker en een server. Met andere woorden, met deze API kun je berichten naar een server sturen en antwoorden ontvangen zonder dat je de server bijvoorbeeld een GET request hoeft te sturen.

**Voorbeeld SignalR**

We gaan even terug naar het vorige voorbeeld, maar in dit geval maken we gebruik van SignalR in plaats van HTTP requests.

Beiden chatapplicaties van John en Mary maken verbinding met de server en database. Telkens wanneer een gebruiker een bericht stuurt naar een andere gebruiker, dan herkent de server de gebruikers en stuurt deze de berichten direct door terwijl de verbindingen open blijven, totdat een gebruiker de verbinding zelf afsluit.

**Afbeelding 2: Voorbeeld SignalR**

Diagram

Description automatically generated

**Waarom SignalR gebruiken?**

Zoals je in Afbeelding 2 zag, de SignalR methode is veel efficienter, legt geen druk op de performance van de server en biedt een realtime functie aan. Alleen is het wel moeilijker te bouwen vergeleken met een HTTP request response model.

**Bruikbaarheid**

Met SignalR kunnen we webapplicaties maken die hoogfrequente updates van de server vereisen. Grote voorbeelden van dergelijke realtime webapplicaties zijn als volgt:

* dashboards,
* games,
* platformen voor aandelen- en cryptohandel,
* en chatapplicaties.

SignalR kan in de volgende situaties gebruikt worden.

1. Meldingen: Als de webapplicatie alle klanten wilt informeren door middel van feedback, waarschuwingen of herinneringen.
2. Chat: Het is heel eenvoudig om een chattoepassing te implementeren met behulp van SignalR, of het nu een-op-een of een groepschat kan zijn.

Diagram

Description automatically generated

1. Gaming: SignalR helpt bij het maken van een gaming-applicatie die regelmatig van een server moet worden gepusht.

**Security**

SignalR gebruikt encryption en een digitale handtekening om het 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.

**Nadelen SignalR**

Alhoewel SignalR een realtime functie aanbiedt en de druk wegneemt bij de server, heeft deze toch een paar nadelen.

Allereerst is het zo dat alhoewel SignalR nog ondersteund wordt, worden er geen nieuwe functies meer toegevoegd. Alleen kritieke bugs worden opgelost. Ook bestaat SignalR voornamelijk uit jQuery code, wat namelijk verouderd is en in de toekomst steeds meer overbodig zal worden.

Verder heeft SignalR wat beperkingen. SignalR gebruikt per-connection buffers om inkomende en uitgaande berichten te beheren. SignalR beperkt deze buffers standaard tot 32 KB. Het grootste bericht dat een client of server kan verzenden is 32 kB. Dus het maximale geheugenverbruik van een verbinding voor berichten is 32 KB.

Tenslotte zijn er naast SignalR een aantal alternatieven die ook worden gebruikt door grote webapplicaties. Voorbeelden zijn Firebase, pusher, webRTC, RabbitMQ, gRPC, en MQTT. Een bekende alternatief is Socket IO.