

|  |
| --- |
| Oefening signalr  Ouassim Abdellaoui &  Graduaat Informatica Programmeren  Webapplicaties |

1. Oefening signalr

Opgave

In deze oefening wordt er een Blazor Server applicatie gemaakt waarin we met SignalR stap voor stap een chat(web)applicatie maken die communiceert met een console applicatie. Gewenste eindresultaat:

Graphical user interface, text, application, Teams

Description automatically generated

Vooraf

A screenshot of a computer

Description automatically generated with medium confidenceOm te beginnen kan je de startbestanden downloaden.

In dit bestand zijn al enkele dingen aangemaakt die we later nodig zullen hebben voor deze oefening.

Deze solution bestaat uit een Blazor Server- en WPF-applicatie. Deze gaan we met elkaar laten communiceren met behulp van SignalR. Ook is er in de Blazor Server-applicatie al een map “Hubs” met klasse “ChatHub.cs” aangemaakt, hier straks meer over.

Om SignalR te kunnen gebruiken is via Nuget Package Manager de package “Microsoft.AspNetCore.SignalR.Client” al in beide programma’s geïnstalleerd. Veel succes!

ChatHub

We beginnen met de ChatHub. In de Blazor Sever applicatie vind je deze in de map ChatHub. Merk op dat er bovenaan “using Microsoft.AspNetCore.SignalR” staat, we gebruiken hier dus de package die al vooraf geïnstalleerd is.

Je kan de ChatHub best vergelijken met een server. Deze gaat luisteren naar berichten met behulp van een methode en deze doorsturen naar alle clienten. Maak de methode SendMessage() aan zoals hieronder:

Text

Description automatically generated

Deze methode staat in voor het versturen van berichten. Als er iemand de SendMessage methode oproept gaan we een gebruiker en bericht meegeven. Wanneer dit gebeurt kijkt de methode welke gebruikers momenteel verbonden zijn en krijgen ze allemaal dit bericht met behulp van de methode ReceiveMessage (deze gaan we later toevoegen aan onze Index-pagina).

Program.cs

We blijven in onze Blazor Server applicatie en openen nu Program.cs.

Hier moeten we onderdaan de link meegeven waarop we onze ChatHub die we juist hebben aangemaakt willen laten werken. Dit kan met app.MapHub<ChatHub>(pattern: "/chathub"); zoals hieronder is ingegeven.

Text

Description automatically generated

Indexpagina

Nu gaan we onze indexpagina aanmaken. Open in de Pages map in je Blazor Server applicatie “Index.razor”, hier is al een simpel formulier klaargezet met twee invulvelden. Eén voor de gebruikersnaam en één voor het bericht. Daaronder een knop met een onclick- en disabled-event, deze gaan we direct nog aanmaken. Om alle berichten te tonen maken we gebruik van een unordered list.

Text

Description automatically generated

Onderaan de indexpagina gaan we in de code 3 methodes aanmaken. We beginnen met de clientside om de berichten te kunnen ontvangen. Typ volgende code over in de OnInitializedAsync-methode.

Text

Description automatically generated

We gebruiken hier de HubConnection om de url mee te geven die we eerder in Program .cs hadden voorgedefinieerd. Verder luistert de HubConnection naar nieuwe berichten door de ReceiveMessage-methode (dezelfde benaming als in de ChatHub). Omdat Blazor niet zelf weet dat er een nieuwe waarde is gebruiken we de InvokeAsync(StateHasChanged), daar wordt de connectie met de server geopend.

Om berichten te kunnen versturen hebben we een @onclick=”Send”-event op onze knop. Om deze te laten werken zet je volgende code in de Send-methode.

Text

Description automatically generated

Als we connectie hebben met de hub gaan we via de SendMessage-methode een bericht versturen. Merk hier op dat de methode dezelfde benaming heeft als de methode in de ChatHub.cs klasse die we als eerste hadden aangemaakt.

In de knop op het formulier hadden we ook disabled=@(!IsConnected) staan. Om dit te laten werken zet je volgende true/false onder de Send()-methode.



Als er geen HubConnection is, is de knop disabled. Is er een connectie dan kan je berichten versturen.

Als laatste willen we de connectie op een “propere” manier afsluiten. Dit gaan we doen met een ValueTask. Als de HubConnection nog niet gesloten is sluiten we deze.

Text

Description automatically generated

Onze Blazor Server zijde is nu af. Je kan dit al eens testen door de applicatie te starten en de link die opent te kopiëren en in een nieuw venster te openen. Als je deze naast elkaar houdt heb je eigenlijk al een werkende chatapplicatie.

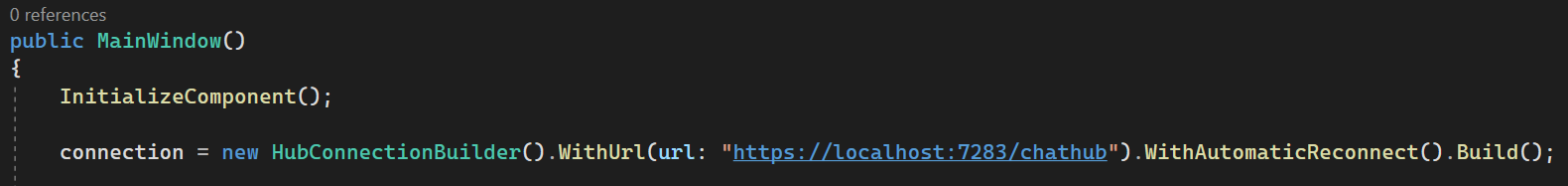
Graphical user interface, text, application

Description automatically generated

Consoleapplicatie

In de consoleapplicatie is er al een XAML-scherm aangemaakt. We gaan enkel de achterliggende code nog moeten maken.

In de code van het MainWindow gaan we eerst de connectie aanmaken. Dit doen we zoals in de Blazor Server applicatie met HubConnection en kan met volgende code:



We geven hier de url mee die opent bij het opstarten van het programma. We kunnen deze url gemakkelijk op gaan zoeken in de Blazor Server applicatie bij Program.cs. Hier zien we dat ons programma HttpsRedirection gebruikt.

Text

Description automatically generated

Nog steeds in de Blazor Server applicatie bij Properties -> launchSettings.json kunnen we terug vinden welke url er gebruikt wordt met HTTPS.

Text

Description automatically generated

Conclusie: onze Blazor Server applicatie gaat url <https://localhost:7283> openen, deze kunnen we dan ook invullen in onze consoleapplicatie.

De connection die we juist hebben aangemaakt heeft standaard 3 events: Reconnecting, Reconnected en Closed. Deze zijn in de constructor van MainWindow al aangemaakt. Zoals je uit de benaming wel kan afleiden zorgen zij ervoor dat er een bericht wordt getoond bij het verbinden, herverbinden en sluiten van de connectie.

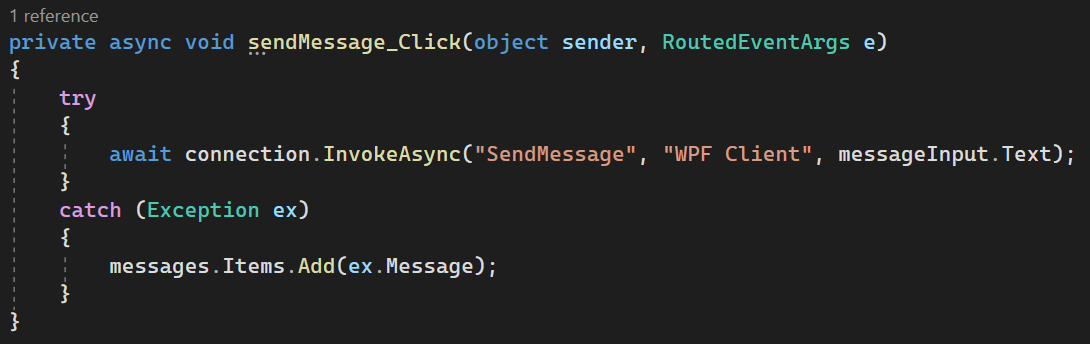
Om deze connectie te kunnen openen zet je volgende code in het click-event van de openConnection-knop.

Text

Description automatically generated

Net zoals in de Blazor Server applicatie wordt er hier geluisterd naar nieuwe berichten via de ReceiveMessage-methode, daarna wordt de verbinding gestart door middel van de connection.StartAsync().

Om berichten te kunnen versturen moeten we nog één laatste ding doen, namelijk het click event van de knop van code voorzien. Hiervoor heb je volgende code nodig:



We maken weer gebruik van de SendMessage-methode van de ChatHub. Als naam geven we WPF Client mee en het bericht is wat de gebruiker in de textbox heeft getypt.

Beide applicaties tegelijk starten

Voordat onze applicatie af is willen we nog instellen dat beide programma’s tegelijk opstarten. Dit kan door rechtermuisknop op je solution te doen en voor “Configure Startup Projects” te kiezen.

Graphical user interface, text, application

Description automatically generated

Hier kiezen we voor “Multiple startup projects” en zetten beide actions op “Start”.

Graphical user interface

Description automatically generated

Proficiat! Je Blazor Server SignalR applicatie is nu helemaal klaar! De Blazor server- en console-applicatie starten tegelijk en kunnen met elkaar communiceren.