FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Labo Webtechnologieën
Reeks 6: Node.js en websockets

4 December 2019

1 Inleiding

In deze opgave gebruiken we Node.js als backend om een eenvoudige REST backend te implementeren. We maken ook kennis met websockets. Websockets laten ons toe om data over en weer te sturen tussen de client en de server. In tegenstelling tot HTTP calls (zoals bij AJAX) kan de server ook een bericht sturen naar de client zonder dat die eerst hiervoor een verzoek heeft gestuurd. Websockets houden dus een bidirectioneel communicatiekanaal open tussen client en server.

De uiteindelijke applicatie zal volgende functionaliteit ondersteunen:

- De client toont een grafiek van historische data (temperatuur).
- De server biedt een REST api aan om alle data op te vragen en om nieuwe data toe te voegen.
- Er wordt een websocket open gehouden tussen client en server. Wanneer er een nieuw datapunt naar de server gestuurd wordt, wordt dit ook direct naar alle verbonden clients gestuurd. De client kan dan de grafiek direct updaten.

De startcode op Minerva bevat onderstaande bestanden:

- graph.html: alle code nodig om een grafiek te tonen.
- temperatures.js: wat code om dummy data te genereren.
- sensor.html: een html formulier dat je kan gebruiken om nieuwe data door te sturen naar de server.

2 Een nieuw Node.js express project

- Maak een nieuw project van het type “Node.js Express App” aan met Webstorm of Intellij (installeer indien nodig de plugin voor node.js).

- Kies voor “Pug” als template en “Plain CSS” als CSS.
- Verken het project. Welke bestanden zijn er ? Welke requests kunnen afgehandeld worden ? Waar wordt dit gedaan ? Wat is de rol van node.js, express en pug ? Wat is de rol van de routes ? Wat doen de Views ? Waar worden die opgeroepen ? Waar wordt het poortnummer geconfigureerd ?

3 index.html omvormen naar een Pug pagina

- Verwijder de twee subroutes in *app.js* (*app.use(...)*). Voor de eenvoud gaan we alles afhandelen in *app.js* zelf.
- Registreer een nieuwe route voor een GET-request en stuur de inhoud van *index.pug* terug.
- Pas *index.pug* aan zodat het dezelfde structuur bevat van de gegeven *graph.html*. Verplaats de javascript en css naar een apart bestand.

4 Nieuwe routes voor het opvragen en toevoegen van data.

- Kopieer de code vanuit *temperatures.js* naar je *app.js* file. Deze code genereert wat dummy data.
- Voorzie volgende routes:

URL	**Methode**	**functionaliteit**
/temperatures	GET	Vraagt alle beschikbare data van alle kamers op
/rooms	GET	Vraagt de namen van alle beschikbare kamers op
/temperatures/{kamer}	GET	Vraagt alle data op van deze kamer
/temperatures/{kamer}	POST	Voegt een nieuw datapunt toe voor deze kamer.

- Alle data wordt als JSON naar de client gestuurd.
- Met behulp van het formulier in *sensor.html* kan je POST requests doen naar de backend.

5 Websockets

- Gebruik ws https://www.npmjs.com/package/ws#sending-binary-data om een websocket aan te maken op de server.
- Installeer “ws”. Op de labo PC’s installeer je een oudere versie met “npm install ws@6.2.1”, op je laptop kan je gewoon “npm install ws” doen.
- De websocket route heeft als parameter de naam van de kamer waarop de client zich wil abonneren.
- Bij het verbinden wordt alle beschikbare data van de kamer naar de client gestuurd.

- Als er een nieuw datapunt beschikbaar is moet dit ook direct naar alle geregistreerde clients gestuurd worden.
- Zorg er nu voor dat de client een websocket verbinding maakt met de server bij het laden van de pagina en dat de data die afkomstig is van de server getoond wordt.
- De client moet de grafiek automatisch opnieuw tekenen als er nieuwe data is binnen gekomen.