Onderzoek WebSockets

Created	@September 30, 2024 10:27 AM
Tags	

Hoe kan ik een WebSocket integreren in een Next.js project?

Deelvragen

- ~~Wat is de definitie van een WebSocket?~~
- ~~Waar worden WebSockets voor gebruikt?~~
- ~~Hoe wordt de verbinding tussen client en server opgezet in een WebSocket?~~

Naast deze deelvragen ga ik ook 2 approaches proberen om het eindproduct te realiseren:

- ~~WebSocket die in een apart Node.js wordt gerund~~
- ~~WebSocket die in het hoofdproject wordt gerund~~

Inleiding

Voor mijn persoonlijke portfoliowebsite kwam ik op het idee om een feature te maken, waarmee gebruikers de cursors van andere gebruikers kunnen zien. Uit ervaring weet ik dat functies zoals deze vaak tot altijd met een WebSocket gemaakt worden. Echter heb ik nog nooit met een WebSocket gewerkt, dus dat leek mij een leuke uitdaging.

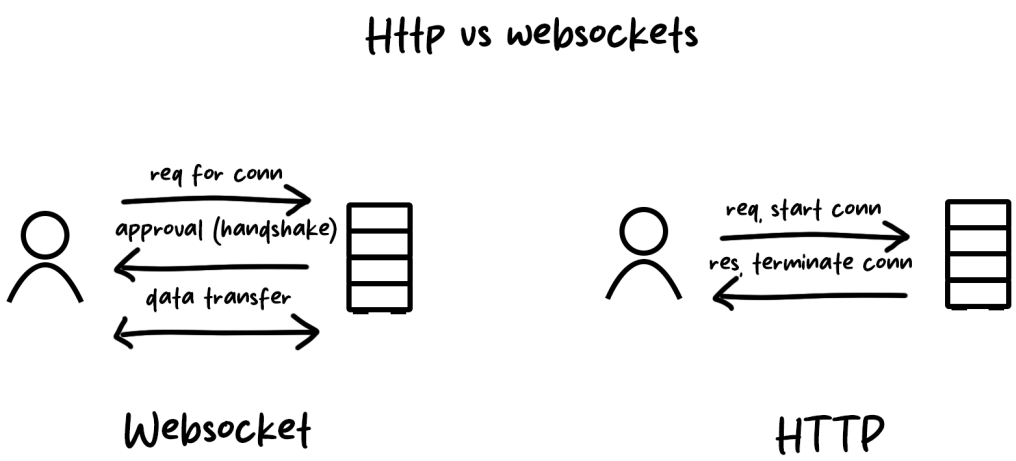
Wat is de definitie van een WebSocket?

Een WebSocket biedt een permanente verbinding tussen de client en de server, waardoor continue dataoverdracht mogelijk is in beide richtingen. In tegenstelling tot traditionele (HTTP) communicatie, die unidirectional is (de client stuurt een verzoek en de server geeft antwoord) zijn WebSockets bidirectional. Dit betekent dat zowel de client als de server data naar elkaar kunnen sturen zonder dat ze steeds een nieuwe request moeten maken.

Bij een HTTP request wordt er een verzoek gemaakt vanuit de client (zoals het aanvragen van een pagina of het submitten van een form). De server verwerkt vervolgens het verzoek en stuurt een response. Nadat de server de response heeft verstuurd, wordt de connectie automatisch gesloten, daarom moet er een hele nieuwe request aangemaakt worden wanneer er additionele data nodig is (denk aan een API call of een refresh van de pagina, wanneer een form is ingevuld).

Bij WebSockets wordt er net zoals bij een HTTP request als eerst een request gedaan. De server keurt deze request dan goed en stuurt een soort goedkeuring terug naar de client (ook wel een ‘handshake genoemd’). In tegenstelling tot HTTP wordt de connectie nu niet gesloten, maar ontstaat er een kanaal waarin de server en de client data aan elkaar uit kunnen wisselen. Als 1 van de 2 partijen de connectie sluit, wordt de connectie ook aan de andere kant gesloten.

(Bron: https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/)



Waar worden WebSockets voor gebruikt?

WebSockets worden erg veel gebruikt bij grote applicaties die veel interactie nodig hebben tussen verschillende gebruikers.

Een aantal voorbeelden zijn:

- Multiplayer games
- Websites met collaborate editing (denk aan Google Docs)
- Chatapplicaties
- Real-time track-and-trace voor locatie
- HMR van Webpack (bron: https://webpack.js.org/guides/hot-module-replacement/)

Voorbeelden die technisch gezien **geen** WebSockets zijn:

- **Pushmeldingen**: de server stuurt een melding naar de client, maar daarna wordt de connectie gelijk gestopt
- **WebRTC-systemen** (voor audio/video streaming): In WebRTC ligt de nadruk op peer-to-peer communicatie tussen clients, waarbij de server vooral wordt

gebruikt om de initiële verbinding te starten, niet voor de eigenlijke dataoverdracht.

(Bron: https://medium.com/the-developer-journal/10-most-amazing-use-cases-of-websockets-go-real-time-166b71e0e711)

Hoe wordt de verbinding tussen client en server opgezet in een web socket?

De client stuurt een `GET` request naar de server, waarin hij aangeeft een HTTP connectie wilt upgraden naar een WebSocket connectie. Deze request wordt de 'Opening Handshake' genoemd. Als de request geaccepteerd wordt stuurt de server een `HTTP 101 Switching Protocols` response terug, hierdoor kan de connectie opgezet worden. (Bron: https://ably.com/topic/websockets#:~:text=Establishing a connection at the,the client and the server)

Een voorbeeld van de request om de websocket op te zetten (JavaScript):

```
const webSocket = new WebSocket(url, protocols);
```

Vervolgens kan je een functie opzetten die wordt gecalled wanneer er een nieuwe message binnenkomt:

```
socket.onmessage = function(msg) {
    if(msg.data instanceof ArrayBuffer) {
        // If the data is binary (like an image)
        processArrayBuffer(msg.data);
    } else {
        // If the data is text (like JSON)
        processText(msg.data);
    }
}
```

Een voorbeeld om iets te sturen naar een WebSocket:

```
socket.send(JSON.stringify({'msg': 'Hello Websocket!'}));
```

Conclusie

In dit onderzoek heb ik onderzocht hoe een WebSocket data ontvangt en verstuurt. Ik heb geconcludeerd dat een WebSocket een protocol is dat een verbinding mogelijk maakt waarover data heen en weer verstuurd kan worden. Hierdoor kunnen gegevens efficiënt en in real-time worden uitgewisseld zonder de noodzaak voor constante API-calls of het herladen van de pagina. Deze functionaliteit maakt WebSockets super waardevol voor applicaties die een real-time connectie nodig hebben tussen gebruikers.