

Een Snellere Datastroom:

Websockets Onder de Loep

Orens Jasper
Indestege Michelle
Lenaerts Roald
Level27, Hasselt

Woord vooraf

Voor u ligt de graduaatsproef “*Een snellere datastroom: Websockets onder de loep*” Dit project markeert het hoogtepunt van mijn opleiding Programmeren aan de PXL in Hasselt. Sinds februari ben ik met veel enthousiasme aan dit onderzoek begonnen.

Tijdens mijn studie merkte ik dat sommige aspecten uitdagender waren dan verwacht. Zo had ik bijvoorbeeld het voornemen om het cachingmechanisme van WebSockets te onderzoeken, maar ontdekte later dat dit niet door WebSockets wordt ondersteund. Desondanks heb ik veel waardevolle ervaring opgedaan en ben ik gegroeid door de uitdagingen die ik tegenkwam. Het harde werk, vaak tot laat in de nacht, heeft geleid tot deze scriptie.

Ik wil graag mijn PXL-coach Michelle Indestege bedanken voor haar geweldige begeleiding. Ik kan eerlijk zeggen dat ik erg blij ben dat zij mij heeft bijgestaan. Mijn grootste dank gaat uit naar Roald Lenaerts, mijn WPL-coach, die tijdens het werkplekleren veel tijd heeft geïnvesteerd om mij alles zo goed mogelijk te leren. Zijn toewijding en bereidheid om zelf dingen te onderzoeken en uit te leggen, hebben een grote impact op mij gehad.

Ook wil ik Juan Jacobs en Joran Dirkzwager van het platformteam bij Level27 bedanken. Het was een genoegen om met zulke getalenteerde programmeurs samen te werken. Hun hulp en kennis hebben mijn traject bij Level27 verrijkt, en de aangename sfeer zorgde ervoor dat ik me altijd welkom voelde.

Tot slot wil ik dit project opdragen aan Erwin Peters, mijn persoonlijke held, stiefvader en beste vriend. Hij is helaas overleden voordat ik aan mijn opleiding begon, maar hij heeft mij de motivatie gegeven om altijd het beste uit mezelf te halen en veranderingen te omarmen als iets tegenzit. Zijn inspirerende woorden hebben me ertoe gebracht deze opleiding te starten bij PXL, en daarvoor ben ik hem eeuwig dankbaar.

Ik hoop dat deze graduaatsproef u bevalt en wens u veel leesplezier toe.

Jasper Orens,

13 mei 2024, 3500 te Hasselt.

Inhoudsopgave

Inhoud

Woord vooraf.....	2
Inhoudsopgave.....	3
1 Bedrijfsvoorstelling	4
1.1 Situering van het bedrijf	4
1.2 Werkomgeving.....	5
2 Projectvraag, onderzoeksacties en resultaten	5
2.1 Situering probleemstelling.....	6
2.2 Projectvraag en deelvragen	7
2.3 (Onderzoeks-)acties.....	7
2.3.1 Wat zijn de belangrijkste eigenschappen van websockets en welke meetbare voordelen kunnen ze bieden voor realtime communicatie in een webapplicatie, met een volledige implementatie tegen eind Mei?	7
2.3.2 Wat zijn de meest voorkomende websocketimplementaties in de wereld om data te tonen en welke implementatie presteert het meest efficiënt gebaseerd op de onderzochte eigenschappen?.....	8
2.3.3 Wat is de meest optimale datastructuur om een applicatie te laten connecteren met een database door gebruik van een websocket?	8
2.3.4 Is de nieuwe implementatie een verbetering op de huidige websocket die gebruikt wordt bij Level27?	8
2.4 Verzamelde resultaten	8
2.4.1 Wat is een websocket?	8
2.4.2 Socket.IO	10
2.4.3 RxJS websocket.....	11
2.4.4 SockJS.....	12
2.4.5 Alternatieven voor websockets.....	13
2.4.6 SockJS versus RxJS websocket versus Socket.IO, welke implementatie presteert het best?	17
3 Conclusies en aanbevelingen.....	16
3.1 Conclusies	17
3.2 Aanbevelingen	17
4 Persoonlijke reflecties en kritische kanttekeningen.....	17
5 Referentielijst	18
6 Bijlagen	18

1 Bedrijfsvoorstelling

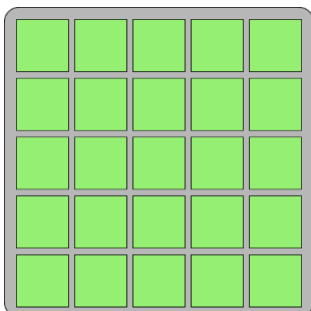
1.1 Situering van het bedrijf

Level27, gevestigd in Hasselt, België, is een prominente speler in de hostingindustrie en biedt een breed scala aan diensten op internationaal niveau. Opgericht in 2007, heeft Level27 zich ontwikkeld tot een toonaangevende aanbieder van webhosting, agency hosting, managed services en cloud-services. Naast domeinnaamregistratie en e-maildiensten, biedt Level27 ook gespecialiseerde dedicated en shared hosting aan. Dedicated hosting bij Level27 omvat servers uitgerust met virtual machines (VM's) op fysieke databases, met constante backups en beveiliging, terwijl shared hosting een kosteneffectieve oplossing biedt door servers te delen met andere klanten.

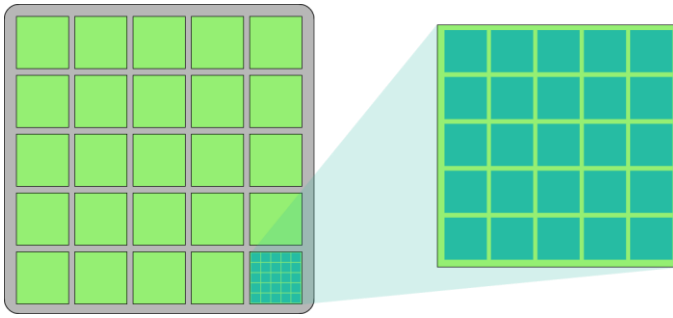
Level27 heeft een sterke reputatie opgebouwd dankzij hun betrouwbare en veilige hostingdiensten, met een nadruk op hoge uptime en uitstekende klantenondersteuning. Het bedrijf biedt 24/7 ondersteuning en garandeert een uptime van 99,9%, ondersteund door regelmatige backups die elk uur worden gemaakt en 30 dagen worden bewaard. Level27 werkt samen met cloudproviders zoals AWS en Azure om flexibele en krachtige oplossingen te bieden aan hun klanten. Deze samenwerking stelt hen in staat om hoogwaardige IT-infrastructuur en -diensten te leveren aan kleine en middelgrote ondernemingen die behoefte hebben aan robuuste en betrouwbare online oplossingen.

Bovendien beschikt Level27 over geavanceerde datacenters in België, uitgerust met de nieuwste technologieën om snelle connectiviteit, hoge uptime en robuuste beveiligingsmaatregelen te garanderen. Deze infrastructuur speelt een cruciale rol in het vermogen van het bedrijf om stabiele en efficiënte hostingdiensten te leveren aan klanten zowel lokaal als internationaal.

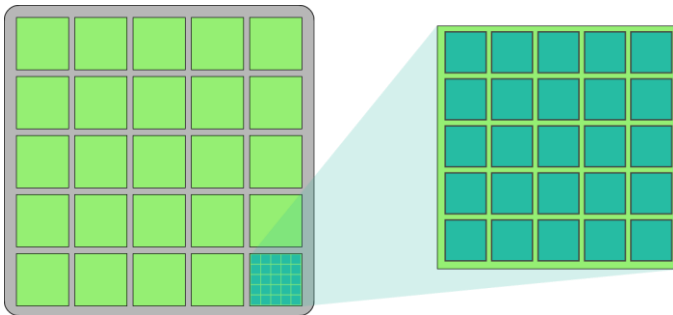
Dedicated Hosting: Dedicated hosting bij Level27 betekent dat een klant een volledige server tot zijn beschikking heeft. Dit biedt maximale controle en prestaties, aangezien de volledige server niet gedeeld wordt met anderen. Elke server is uitgerust met VM's op fysieke databases, wat zorgt voor hoge prestaties en beveiliging. Level27 zorgt ervoor dat de servers up-to-date blijven en maakt elke 15 minuten backups, die 30 dagen worden bewaard. Dit type hosting is ideaal voor grote bedrijven of websites met veel verkeer, omdat het optimale prestaties en veiligheid biedt zonder dat andere gebruikers de serverbronnen beïnvloeden. Onderstaande voorbeeld toont verschillende servers voor.



Shared Hosting: Bij shared hosting wordt één server gedeeld door meerdere klanten. Dit is een kosteneffectieve oplossing, maar het delen van serverbronnen kan leiden tot prestatieproblemen als een van de klanten veel CPU of geheugen gebruikt. Shared hosting bij Level27 is voorzien van regelmatige backups en een gebruiksvriendelijk controlepaneel. Dit type hosting is geschikt voor kleinere websites of bedrijven met een beperkt budget die geen uitgebreide serverbronnen nodig hebben. Op het voorbeeld hieronder is voorgesteld hoe één server wordt opgesplitst in verschillende partities. Iedere gebruiker van shared hosting neemt één of meerdere van deze partities in gebruik.



Agency Hosting: Agency hosting bij Level27 biedt een hybride oplossing tussen dedicated en shared hosting. Hierbij worden de resources van een VM gedeeld, maar elke partitie is geïsoleerd, zodat de acties van één klant geen invloed hebben op de anderen. Dit biedt meer stabiliteit en prestaties dan traditionele shared hosting. Agency hosting is ideaal voor digitale bureaus en ontwikkelteams die meerdere websites of applicaties beheren en een betrouwbare, geïsoleerde omgeving nodig hebben voor hun klanten. Onderstaande voorbeeld toont aan hoe elke partitie geïsoleerd zou zijn.



1.2 Werkomgeving

De werkomgeving van Level27, met name binnen de afdeling "Platform", kenmerkt zich door innovatie en samenwerking. Het team binnen deze afdeling is verantwoordelijk voor de ontwikkeling en het onderhoud van het controlepaneel voor serverbeheer. Roald houdt zich bezig met scripting in Chef op het Ruby-framework, Juan ontwikkelt de frontend in React, Ola werkt aan de backend in PHP, en Joran fungeert als functioneel analist. Taken worden verdeeld via ClickUp, een projectmanagementtool, waardoor het team efficiënt en overzichtelijk kan werken zonder traditionele methodologieën zoals Scrum of Kanban te gebruiken.

Elke week vullen medewerkers een zelfreflectietemplate in, die vervolgens wordt besproken met het afdelingshoofd. Op vrijdag presenteert een werknemer de voortgang van hun afdeling, wat bijdraagt aan een transparante en collaboratieve werkomgeving. De werkomgeving bij Level27 kan beschreven worden als rustgevend, met een sterke nadruk op het bevorderen van een goede sfeer en collegiale relaties. Dit draagt bij aan een positieve en productieve werkcultuur binnen het bedrijf, waar het welzijn van de werknemers centraal staat.

Level27 stimuleert innovatie en professionele groei, met veel aandacht voor persoonlijke ontwikkeling en teamwork. De combinatie van een ondersteunende werkomgeving en de focus op geavanceerde technologieën maakt Level27 een aantrekkelijke werkgever voor professionals in de IT- en hostingsector.

Door deze sterke focus op zowel technische excellentie als een positieve werkomgeving, blijft Level27 een voorloper in de hostingindustrie, klaar om tegemoet te komen aan de behoeften van moderne bedrijven die betrouwbare en efficiënte online oplossingen zoeken.

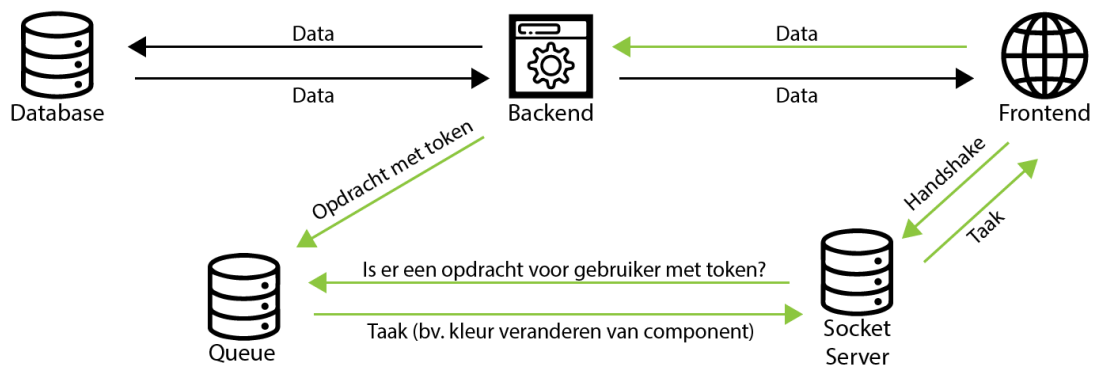
2 Projectvraag, onderzoeksacties en resultaten

2.1 Situering probleemstelling

Voor een bedrijf dat actief is in onlinehosting is het essentieel dat data efficiënt en correct wordt verstuurd van een applicatie naar een database. Eén van de mogelijke manieren om dit te realiseren is met behulp van een websocket. Websockets leggen een handshake (een aanvraag voor een connectie) tussen een applicatie en een database. Wanneer deze handshake is geaccepteerd, ontstaat er een continue verbinding tussen beide, waardoor data in realtime wordt geüpdate.

In deze scriptie wordt de huidige websocketimplementatie bij Level27 geëvalueerd. Het onderzoek bekijkt diverse implementaties van websockets om data efficiënter van server naar client te versturen. Er is geen noodzaak om de implementatie voor dataoverdracht van client naar server te verbeteren.

In onderstaand voorbeeld is de huidige implementatie getoond. In dit voorbeeld heeft de gebruiker op een knop geklikt om een server te activeren.



Om de gegevens in de database aan te passen, worden wijzigingen verstuurd van de frontend naar de backend via HTTP-verzoeken. De backend verwerkt deze verzoeken en stuurt de bijgewerkte gegevens door naar de database. Daarnaast geeft de backend een opdracht aan de queue om de kleur van een knop op de gebruikersinterface te veranderen. Een voorbeeld van de opdracht is hier te zien.

```
[
  "system",
  {
    "entity": "system",
    "entityId": 655,
    "type": "field_change",
    "changedFields": {
      "runningStatus": {
        "oldValue": "running",
        "newValue": "stopping",
        "meta": {
          "statusCategory": "blue"
        }
      }
    }
  }
]
```

Deze queue dient als vangnet voor het geval er een probleem optreedt met de socketserver. Opdrachten in de queue worden maximaal 15 minuten bewaard en worden op basis van first in, first out naar de socketserver gestuurd. De socketserver heeft een constante verbinding met de frontend, waardoor de gebruiker de wijzigingen direct kan zien zonder de pagina te hoeven verversen. In de studie wordt ook bekeken of dit geoptimaliseerd kan worden.

2.2 Projectvraag en deelvragen

- Wat zijn de belangrijkste eigenschappen van websockets en welke meetbare voordelen kunnen ze bieden voor realtime communicatie in een webapplicatie, met een volledige implementatie tegen eind mei?
- Wat zijn de meest voorkomende websocketimplementaties in de wereld om data van de server te tonen en welke implementatie presteert het meest efficiënt gebaseerd op de onderzochte eigenschappen?
- Wat is de meest optimale datastructuur om een applicatie te laten connecteren met een database door gebruik van een websocket?
- Is de nieuwe implementatie een verbetering van de huidige WebSocket die gebruikt wordt bij Level27?

2.3 (Onderzoeks-)acties

2.3.1 Wat zijn de belangrijkste eigenschappen van websockets en welke meetbare voordelen kunnen ze bieden voor realtime communicatie in een webapplicatie, met een volledige implementatie tegen eind Mei?

Het onderzoek is gestart met opzoeken naar lessen die betrekking hadden tot websockets, de resultaten waren relatief beperkt voor zulk specifiek onderwerp zoals websockets. Pluralsight heeft

een cursus[1] ter beschikking met betrekking tot HTTP-methodes en websockets. Deze cursus biedt basis kennis over websockets. Hierdoor zorgde dit voor een goede start om het concept te begrijpen.

Als volgt is het duidelijk geworden welke eigenschappen zorgen voor een goede websocket implementatie.

2.3.2 Wat zijn de meest voorkomende websocketimplementaties in de wereld om data te tonen en welke implementatie presteert het meest efficiënt gebaseerd op de onderzochte eigenschappen?

Om beter begrip te verkrijgen van de werking van websockets heeft het onderzoek bekeken welke websockets er momenteel gebruikt zijn. Er is een Node.js server/applicatie opgesteld om de noodzakelijke data mee te versturen voor alle websocket implementaties en libraries op te laten werken. [8]. In de eerste fase is de node.js server zelf opgesteld en als tweede stap is de database hierin opge maakt. Er wordt data verstuurd en ontvangen van zowel de client als de server. De server verstuurd bij iedere transmissie een string van een lijst met groenten, en de client stuurt een transmissie met fruit. In de frontend wordt de snelheid van het binnen komende en uitgaande data verkeer.

2.3.3 Wat is de meest optimale datastructuur om een applicatie te laten connecteren met een database door gebruik van een websocket?

2.3.4 Is de nieuwe implementatie een verbetering op de huidige websocket die gebruikt wordt bij Level27?

Om beter begrip

2.4 Verzamelde resultaten

2.4.1 Wat is een websocket?

Algemene werking van websockets

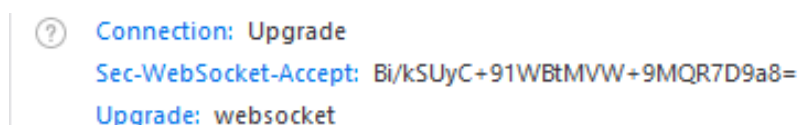
Een websocket legt een persistente, full-duplex verbinding tussen een client en een server, waardoor realtime communicatie mogelijk is zonder de overhead van HTTP-verzoeken. WS(Websocket) biedt deze verbinding onbeveiligd, terwijl WSS (Websocket Secure) dezelfde functionaliteit biedt maar met encryptie van SSL/TLS voor een veilige gegevensoverdracht.

Waarom WS boven WSS kiezen?

WSS is een meer beveiligde vorm, maar deze bevat ook meer overhead (tijd, geheugen en resources die nodig zijn). Het is niet altijd aan te raden om voor WSS te kiezen als de data die verstuurd wordt sneller en met minder bits moet toe komen bij de client of server. Wanneer men websockets wil gebruiken moet men dus de keuze maken of meer encryptie of snelheid prioriteit hebben.

Handshake

De client verstuurd een Request met 'upgrade' als onderdeel van de header. Deze heeft als waarde 'websocket' en een veld 'connection' met als waarde 'Upgrade'.



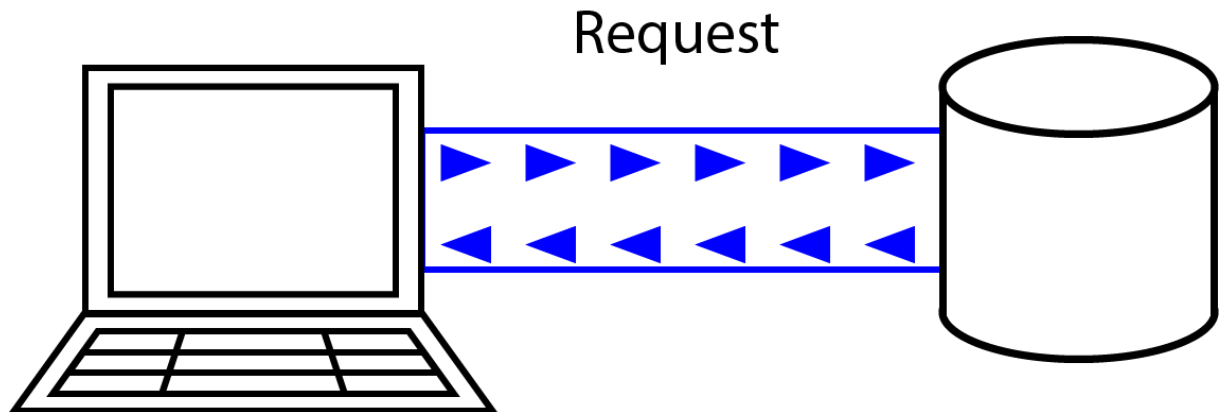
```
? Connection: Upgrade
Sec-WebSocket-Accept: Bi/kSUyC+91WBtMVW+9MQR7D9a8=
Upgrade: websocket
```

Screenshot van console in Firefox van een websocket implementatie

De client verstuurd dan een BASE64 encoded Key. Wanneer de Server dan een reactie

geeft, als deze bekwaam is voor websockets te aanvaarden, gebruikt de Server de HTTP 101 response. (Switching Protocols).

Daarna verstuurt de server een header met dezelfde waarden als die van de client initieel verstuurde, namelijk 'Upgrade: websocket' en 'Connection: Upgrade'. De server neemt vervolgens de Key van de client, voegt hier een GUID aan toe en berekent hier een SHA-hash voor. Op deze manier weet de client dat de server het verzoek heeft geaccepteerd. Deze authenticatie is essentieel om een 'Man-in-the-Middle'-aanval te voorkomen, waarbij een hacker zich voordoeft als de server. Zodra deze acties zijn uitgevoerd, is er bidirectionele communicatie tussen de server en de client.



Afbeelding door Jasper Orens

Voordelen van websockets

- Minder overhead: Door de full-duplex verbinding is er minder overhead in vergelijking met traditionele HTTP-verzoeken die telkens een nieuwe verbinding moeten opzetten.
- Realtime updates: Ideaal voor toepassingen die realtime-gegevensupdates vereisen, zoals chattoepassingen, live sportverslaggeving en financiële tickers.
- Lagere Latency: Websockets verminderen de latency aanzienlijk doordat ze een persistente verbinding onderhouden, waardoor de noodzaak van herhaaldelijke HTTP-requests vervalt.

Nadelen van websockets

- Complexiteit: De implementatie en het beheer van websockets kan complexer zijn dan traditionele HTTP-verzoeken.
- Firewall en Proxy issues: Sommige firewalls en proxies ondersteunen mogelijk geen websocket-verbindingen, wat kan leiden tot verbindingsproblemen.
- Scalability Challenges: Het onderhouden van een groot aantal gelijktijdige websocket-verbindingen kan schaalbaarheidsproblemen veroorzaken.

Use case

Chat applicatie:

Facebook Messenger maakt bijvoorbeeld gebruik van websockets om constant data te verzenden van client 1 naar server en naar client 2.

Social Media updates:

Push notificaties kunnen van social media kunnen gebruikt worden voor de gebruiker berichten te laten ontvangen wanneer er updates binnen komen (bv.: “Iemand heeft je getagged in een foto”, van Facebook).

Gok websites:

Om scores aan te tonen van Live wedstrijden moeten deze altijd gelijktijdig naar alle gebruikers worden ge-update. Websockets zijn hier zeer geschikt voor.

IoT-toepassingen:

Websockets kunnen worden gebruikt om communicatie met Internet of Things (IoT)-apparaten mogelijk te maken, waardoor apparaten zoals sensoren en slimme lampen realtime gegevens naar een server kunnen sturen.

2.4.2 Socket.IO

Wat is het?

Socket.IO is een library dat low-latency, bidirectioneel en event-based communicatie tussen client en server voorziet. De connectie kan opgesteld worden tussen verschillende low-level transports. Deze zijn:

- HTTP long-polling
- WebSocket
- WebTransport

Socket.IO kiest zelf het beste pad afhankelijk van

- De mogelijkheden van de browser
- Het netwerk (sommige netwerken blokeren websockets en/of WebTransport connecties)

Deze library is geen implementatie van een websocket. Deze gebruikt wel websockets voor transport wanneer mogelijk, maar het voegt extra metadata toe in elk packet. Hierdoor kan een traditionele websocket geen connectie maken met Socket.IO omdat de websocket geen weg weet met deze extra data van Socket.IO. [7].

Gebruiksgemak en documentatie

De initiële setup voor socket.IO is relatief gemakkelijk. De basis setup is getest dankzij de duidelijke documentatie [10] Waarin het ook mogelijk is om de library te benutten via een virtual machine in de browser. De documentatie is zeer gebruiksvriendelijk, en éénmaal als de initiële setup afgerond is. Is het algemene gebruik overzichtelijk.

Performance en schaalbaarheid

De performance kan niet enkel verschillen met hoe de data verstuurd wordt, maar ook de browser heeft invloed op de totale performance. Over het algemeen verbruikt Firefox minder CPU dan Chrome. Dus wij zullen de performance hiermee testen.

Ondersteuning en community

Er is buiten de zeer goede documentatie weinig ondersteuning te vinden op hun platform. Er is echter wel een doorverwijzing naar stackoverflow.

Licentie en kosten

socket.IO is gratis en wordt verdeeld onder het MIT-licensie. [11]

Compatibiliteit en integratiemogelijkheden

Socket.IO is zeer compatibel en kan geïntegreerd worden met veel programmeertalen en platforms, waaronder JavaScript, Java, Python, en C++. Het ondersteunt zowel webbrowsers als mobiele apps en biedt functies zoals automatische reconnection en event broadcasting. Dit maakt het geschikt voor real-time webapplicaties zoals chats en interactieve collaboratieve tools. Voor meer technische details. [7, 11]

Voordelen

- Automatische fallback naar HTTP long-polling: Waar gewone websocket-implementaties afhankelijk zijn van de permanente beschikbaarheid van een websocket-verbinding, kan Socket.IO automatisch terugvallen op HTTP long-polling als websockets niet beschikbaar zijn.

- Eenvoudige API voor complexe functionaliteiten: Socket.IO vereenvoudigt de implementaties van complexe functies zoals broadcasting naar meerdere sockets en het afhandelen van reconnecties na verbindingsverlies, wat meer codering en configuratie zou vereisen bij direct gebruik van websockets.

Nadelen

- Overhead: Socket.IO voegt extra bytes toe aan elk bericht voor het beheren van zijn functionaliteiten zoals namespaces en rooms, wat resulteert in grotere bericht groottes vergeleken met een zuivere websocket-oplossing.

- Complexiteit: Voor projecten waarbij eenvoudige berichtuitwisseling voldoende is, kan Socket.IO overkill zijn vanwege de extra functionaliteiten en de ingebouwde ondersteuningsmechanismen die misschien niet nodig zijn.

2.4.3 RxJS websocket

Wat is het?

De RxJS websocket is uniek door de integratie met het *Reactive Extensions framework*, waardoor het eenvoudig kan worden gecombineerd met andere RxJS operators voor geavanceerde datastroomcontrole. Het biedt een *WebSocketSubject* dat zowel een *Observer* als *Observable* is wat bidirectionele communicatie vereenvoudigt. Daarnaast beschikt het over ingebouwde *retry* logica en configuratieopties voor het beheren van verbindingen en het afhandelen van fouten, wat extra robuustheid toevoegt aan real-time-applicaties.

Gebruiksgemak en documentatie

Installatie gaat zeer vlot, de documentatie legt de installatie duidelijk uit zodat de gebruiker zonder problemen kan volgen.

Performance en schaalbaarheid

Het gebruik van complexe operator-ketens, ('map', 'filter', 'merge' bijvoorbeeld), kan de performance beïnvloeden. Het is cruciaal om resources zoals subscriptions op te nemen om geheugenlekken te voorkomen voor schaalbaarheid biedt RxJS krachtige tools zoals concurrency management en backpressure handling, waardoor het effectief grote hoeveelheden gelijktijdige gegevensstromen kan verwerken.

In RxJS kan het gebruik van complexe operator-ketens de performance beïnvloeden omdat elke operator een nieuwe stap in de verwerkingspijplijn toevoegt. Dit kan leiden tot verhoogde CPU- en geheugenbelasting, vooral als de ketens lang en complex zijn.

Ondersteuning en community

RxJS heeft een sterke ondersteuning met uitgebreide documentatie en talloze tutorials die ontwikkelaars helpen snel aan de slag te gaan. Er is een actieve community op platforms zoals GitHub, Stack Overflow, en Gitter, waar ontwikkelaars vragen kunnen stellen en problemen kunnen bespreken.

Licentie en kosten

RxJS is open source. Het wordt gedistribueerd onder de Apache License 2.0, wat betekent dat het vrij beschikbaar is voor gebruik, modificatie en distributie door iedereen.

Compatibiliteit en integratiemogelijkheden

RxJS is compatibel met moderne browsers en server-side omgevingen zoals Node.js. Het integreert naadloos met frameworks zoals Angular, React en Vue.js en kan worden gebruikt met state management libraries zoals Redux en NgRx. RxJS is ideaal voor het beheren van websockets en API-calls door zijn krachtige operators voor asynchrone gegevensstromen. Dankzij zijn brede compatibiliteit en integratiemogelijkheden is RxJS een veelzijdige tool voor reactief programmeren in web- en mobiele toepassingen.

Voordelen

- Integratie met RxJS: Indien er gebruikt wordt gemaakt bij een project waar men Reactive Extensions framework

gebruikt, is het eenvoudig te combineren met andere operators van RxJS. Dit zorgt voor een betere dataflowbeheer.

- Retry-logica: Ingebouwde mechanismen om automatisch opnieuw te proberen verbinding te maken bij storingen.

- Bidirectionele communicatie: WebSocketSubject fungeert als zowel een Observer als een Observable, wat bidirectionele communicatie vergemakkelijkt.

Nadelen

- Complexiteit: De uitgebreide mogelijkheden en integraties kunnen leiden tot een steilere leercurve voor nieuwkamers.

- Afhankelijkheid: Vereist kennis en gebruik van het RxJS-framework, wat een extra laag van afhankelijkheid toevoegt aan het project.

2.4.4 SockJS

Wat is het?

SockJS is een Javascriptbibliotheek die een WebSocket-achtige API biedt. Het is ontworpen om een betrouwbaar transport te bieden voor realtime webapplicaties, zelfs in omgevingen waar WebSockets niet beschikbaar zijn. SockJS valt terug op alternatieve transportmethoden zoals XHR-streaming, XHR-polling en iframe HTML file streaming wanneer WebSockets worden geblokkeerd door firewalls of niet worden ondersteund door de browser.

Gebruiksgemak en documentatie

SockJS is eenvoudig te gebruiken met een duidelijke en uitgebreide documentatie. Het onderzoek heeft ontdekt dat er conflicten kunnen ontstaan met React. Er is geen officiële website alle nodige documentatie is te vinden op hun *GitHub* [11, 12] De installatie en basisconfiguratie zijn goed gedocumenteerd, wat het makkelijk maakt voor ontwikkelaars om snel aan de slag te gaan. Bovendien biedt SockJS voorbeeldprojecten [13, 14] en gedetailleerde handleidingen voor verschillende use cases.

Ondersteunende functies en mogelijkheden

SockJS ondersteunt verschillende fallback-mogelijkheden die automatisch worden gekozen op basis van de mogelijkheden van de client en het netwerk. Dit maakt het betrouwbaar voor een breed scala aan netwerkomstandigheden. Het biedt ook ingebouwde ondersteuning voor cross-origin requests, wat belangrijk is voor het bouwen van moderne webapplicaties.

Performance en schaalbaarheid

SockJS is ontworpen om efficiënt en schaalbaar te zijn. Door fallback-mechanismen te gebruiken, kan het zelfs in beperkte netwerkomgevingen goede prestaties leveren. Echter, de prestaties kunnen variëren afhankelijk van het gebruikte fallback-transport. XHR-polling, bijvoorbeeld, kan meer overhead en latentie introduceren dan native WebSockets.

Ondersteuning en community

SockJS heeft een actieve gemeenschap en goede ondersteuning via forums zoals GitHub en Stack Overflow. Er is veel documentatie beschikbaar, evenals talrijke tutorials en voorbeeldprojecten die ontwikkelaars kunnen helpen bij het oplossen van problemen en het implementeren van SockJS in hun projecten.

Licentie en kosten

SockJS is een open-source project en wordt gedistribueerd onder de MIT-licentie, wat betekent dat het vrij beschikbaar is voor gebruik, modificatie en distributie door iedereen zonder kosten.

Compatibiliteit en integratiemogelijkheden

SockJS is compatibel met veel moderne browsers en kan worden geïntegreerd met verschillende server-side omgevingen zoals Node.js, Java, en Python. Het biedt eenvoudige integratie met andere realtime bibliotheken en frameworks, waardoor het een veelzijdige keuze is voor het bouwen van realtime webapplicaties.

Voordelen

- Fallback-mogelijkheden: SockJS kan automatisch overschakelen naar alternatieve transportmethoden wanneer WebSockets niet beschikbaar zijn, waardoor het zeer betrouwbaar is.

- Cross-origin support: Ondersteuning voor cross-origin requests maakt het geschikt voor moderne webapplicaties die data uit verschillende domeinen moeten halen.

- Gebruiksvriendelijk: Eenvoudige API en goede documentatie maken het makkelijk te implementeren en gebruiken.

Nadelen

- Overhead: Sommige fallback-transporten, zoals XHR-polling, kunnen meer overhead en latentie veroorzaken in vergelijking met native WebSockets.

- Complexiteit: De aanwezigheid van meerdere transportlagen kan de complexiteit van de debugging en het onderhoud van de applicatie vergroten.

2.4.5 Alternatieven voor websockets

2.4.5.1 Andere websocket implementaties

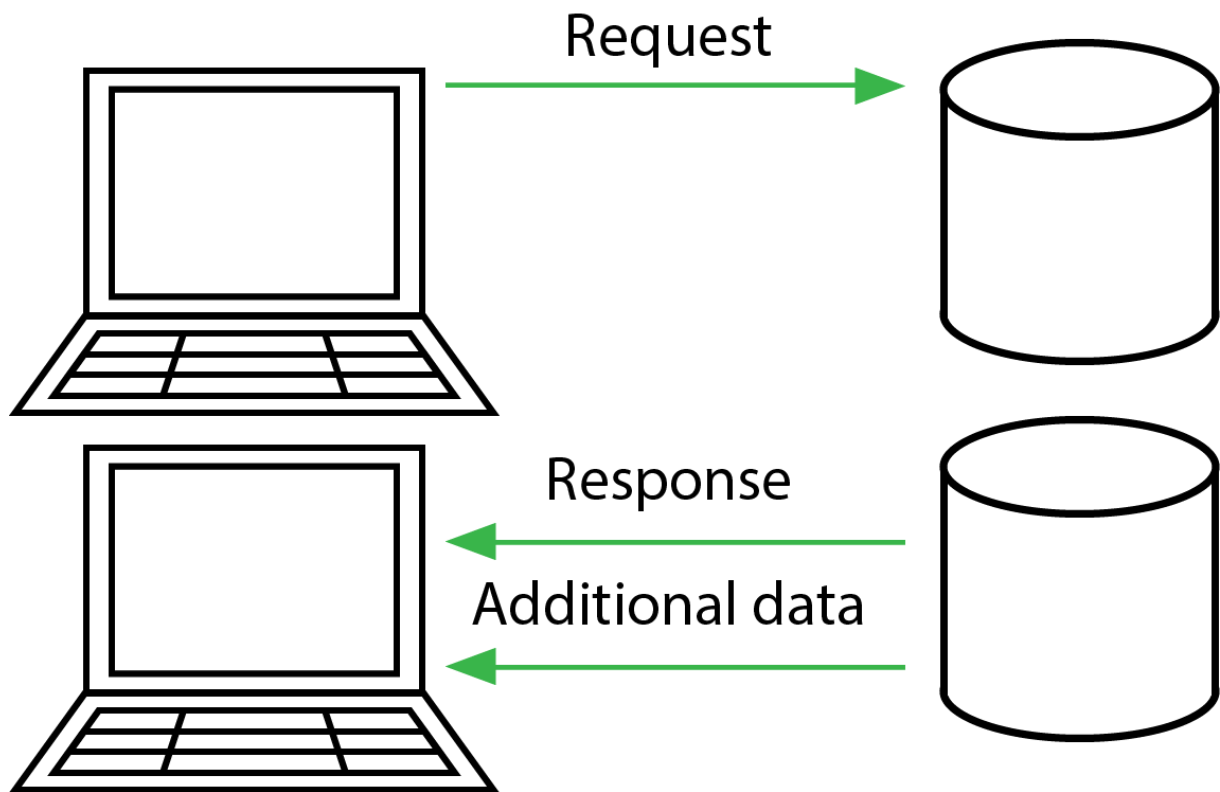
Het onderzoek gaat niet dieper in op Pusher en GraphQL Subscriptions omdat deze diensten vaak hogere overhead en complexiteit met zich meebrengen vanwege hun specifieke gebruiksscenario's en integraties. Pusher is een beheerde service die extra kosten en afhankelijkheden introduceert, wat minder relevant is voor een algemene evaluatie van WebSocket-prestaties. Het is ook noodzakelijk voor externe instellingen te doen via het dashboard van Pusher, dit zijn onnodige extra taken voor de simpliciteit die gezocht wordt voor de implementatie. GraphQL Subscriptions voegt een extra laag complexiteit toe door de noodzaak om GraphQL-context te behouden, wat niet noodzakelijk is voor de kernanalyse van WebSocket-technologieën. Daarom richt het onderzoek zich op meer directe en brede Websocketimplementaties zoals Sock.js, Socket.IO, en RxJS WebSocket, die relevanter zijn voor een bredere reeks toepassingen.

2.4.5.2 HTTP/2 Push

Er werd onderzocht naar HTTP/2 Push. Maar Google heeft besloten om deze functie te verwijderen. Verdere implementatie van deze functie is dus niet uitgewerkt binnen het project. Google heeft het aantal requests op Chrome Browsers bestudeerd dat push requests verzonden en ontvangen. Uit hun onderzoek bleek dat 1.25% van HTTP/2 websites gebruik maakten van deze functie. Bij verdere opvolging bleek dat dit aantal gezakt was naar 0.7%. Op basis van deze analyse besloot Google om deze functie niet verder uit te werken. [2]

2.4.5.2.1 Werking van HTTP2/Push

De client verstuurt een verzoek voor een bepaalde webpagina of bron, zoals HTML. De Server ontvangt het verzoek van de client en identificeert alle vereiste bronnen die nodig zijn om de gevraagde webpagina correct te renderen. In plaats van te wachten tot de client afzonderlijke verzoeken voor elke bron verzendt, kan de server proactief beslissen om bepaalde bronnen naar de client te duwen. Dit wordt gedaan door extra bronnen te 'pushen' naar de client over dezelfde verbinding zonder te wachten op een afzonderlijke verzoek van de client. Zodra de client de gepushte bronnen ontvangt, kan hij deze lokaal opslaan in de cache. Wanneer de client later die bron nodig heeft, hoeft hij geen nieuwe verzoek naar de server te sturen, omdat de bron al beschikbaar is in de cache van de client. [3]

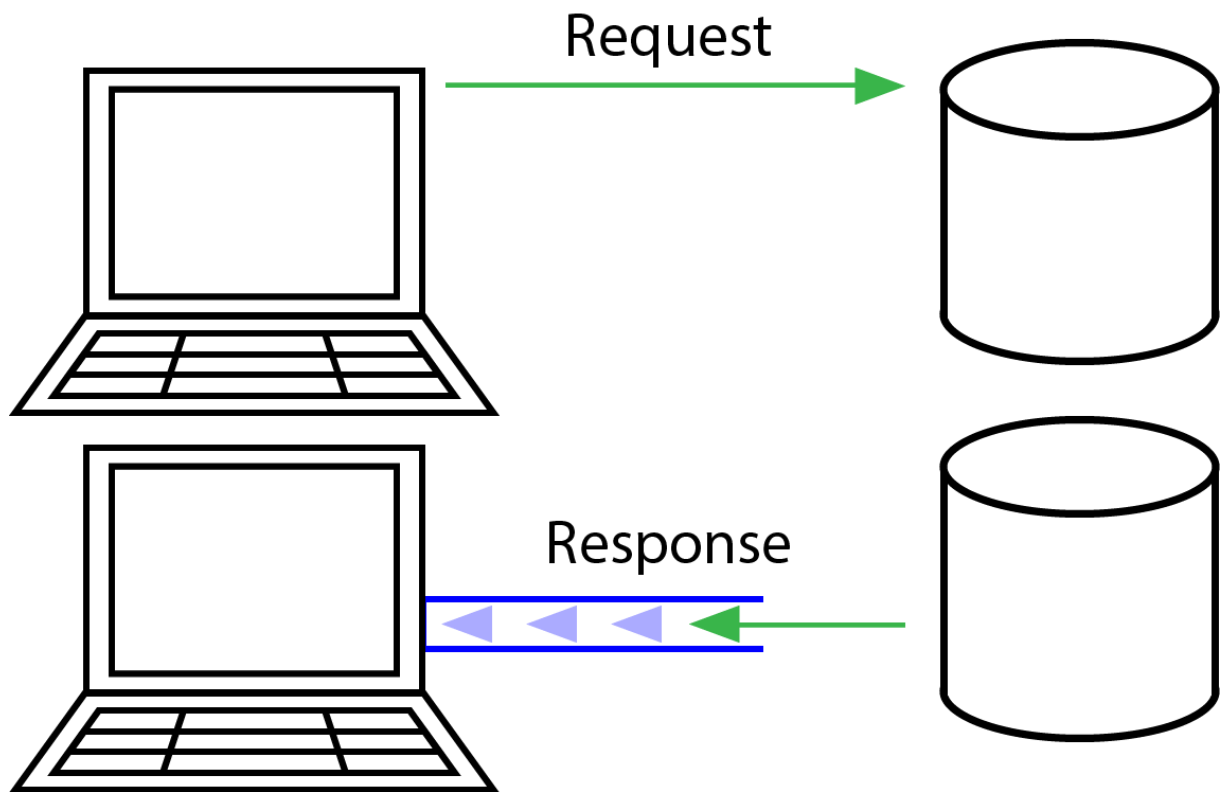


Afbeelding door Jasper Orens

2.4.5.3 Long Polling

2.4.5.3.1 Werking van Long Polling

Deze methode stuurt een verzoek naar de resource, als er geen antwoord komt op het verzoek van de client dan blijft de connectie met de resource open tot er wijzigingen gebeuren en data wordt overgemaakt van de resource naar de client. Via deze methode worden het aantal requests naar de server dus beperkt omdat de connectie langer openblijft en de client dus niet voortdurend requests moet sturen tot er een positief antwoord komt. Long Polling maakt gebruik van HTTP request in de formaten applicatoion/json, tekst/plain en de open connectie blijft 100 tot 300 seconden geldig.



Afbeelding door Jasper Orens

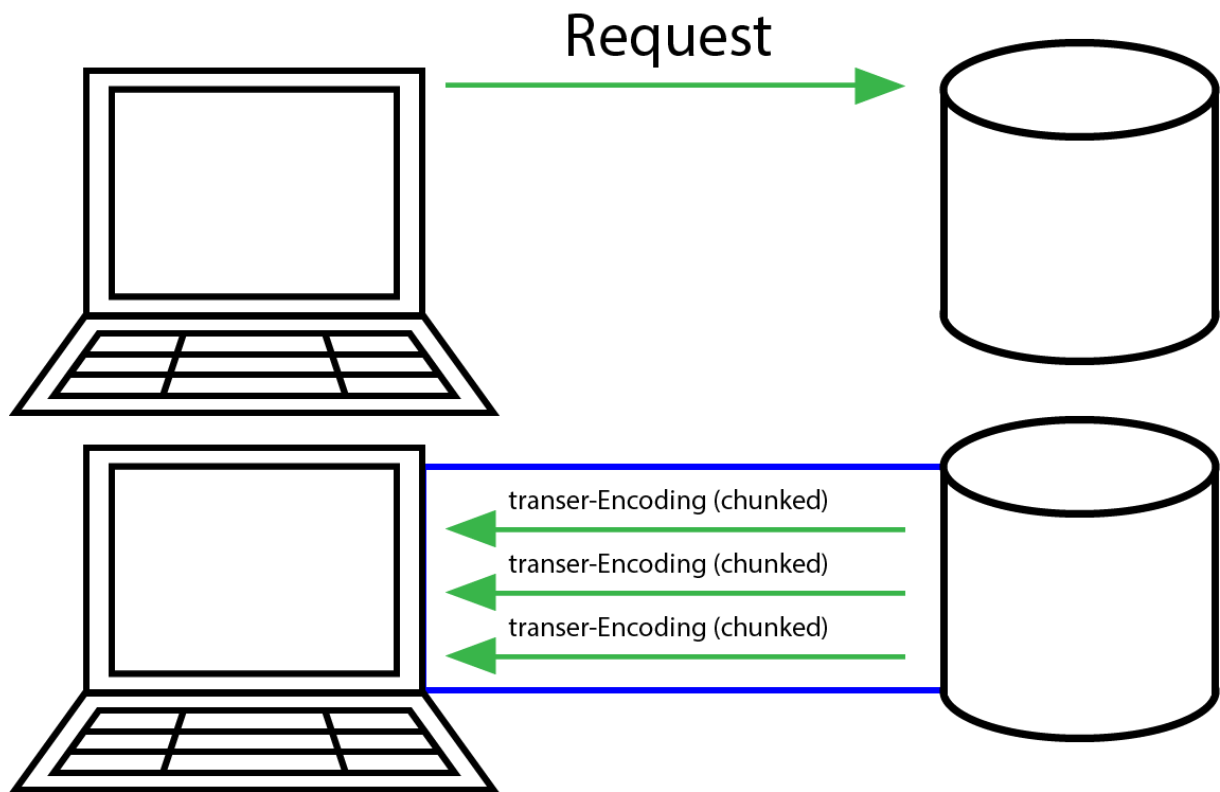
2.4.5.3.2 Use case

De beste benutting voor Long polling is voor chat applicaties waarin realtime updates noodzakelijk zijn. Specifiek chat communicatie tussen twee partijen. (Voor meerdere partijen zijn websockets de betere keuze.) Dit omdat het de connectie met de server openhoudt tot er nieuwe data (een bericht bv.) beschikbaar is. Zodra er nieuwe data aanwezig is op de server verstuurd deze de inhoud naar de client. De long polling open connectie wordt dan gesloten tot er van uit de client een nieuw polling request wordt verstuurd. [4]

2.4.5.4 Server-Sent Events (SSE)

2.4.5.4.1 Werking van SSE

Bij een SSE request wordt de aanvraag gedaan via een 'event-stream' in plaats van een .JSON-formaat of tekstformaat. Via deze weg weet de server dat er gewerkt wordt met een SSE-connectie. De server geeft als volgt meerdere antwoorden in de vorm van 'event-stream' in chunks. Dit zijn grootte blokken code die werden opgesplitst. Er wordt één request gestuurd voor een connectie te openen en de server beslist nadien welke data het wilt sturen naar de client. SSE maakt gebruik van HTTP requests enkel in het tekst/event-stream formaat. Data wordt verstuurd als Prefix. De connectie blijft open tot de client een disconnect request verstuurd.



Afbeelding door Jasper Orens

2.4.5.4.2 Use case

Server-Sent Events (SSE) worden gebruikt voor het live streamen van gegevens, vergelijkbaar met platforms zoals YouTube of Twitch voor video streams. Deze technologie maakt het mogelijk om realtime gegevens te verzenden van de server naar de client, waardoor een continue stroom van informatie wordt gecreëerd zonder dat de client herhaaldelijk om updates hoeft te vragen. SSE is vooral nuttig voor applicaties die afhankelijk zijn van live data, zoals sportupdates, financiële marktgegevens en meer. Het bijzondere van SSE is dat het een bidirectioneel communicatiekanaal biedt, hoewel het geen authentieke full-duplex communicatie is. In plaats daarvan stelt SSE servers in staat om gegevens in realtime naar clients te sturen, terwijl clients geen mogelijkheid hebben om direct naar de server te communiceren via hetzelfde SSE-kanaal. Deze eenrichtingsstroom van gegevens maakt het mogelijk voor clients om continu bijgewerkte informatie te ontvangen zonder de noodzaak van polling, waardoor de serverbelasting wordt verminderd en de efficiëntie van de communicatie wordt verbeterd.[5]

Het onderzoek gaat niet dieper in op Pusher en GraphQL Subscriptions omdat deze diensten vaak hogere overhead

2.4.6 Database structuur

3 Conclusies en aanbevelingen

In dit gedeelte geef je je conclusies die je vanuit je uitgevoerde onderzoek kan afleiden, weer. Je beantwoordt hier je onderzoeksvraag: je geeft op een synthetische en goed beargumenteerde manier een antwoord op je onderzoeksvraag en staat stil bij de belangrijkste vaststellingen van je onderzoek. Je geeft tips en aanbevelingen voor je werkplek naar voor schuiven die je vanuit je onderzoek aangereikt krijgt/gekregen hebt. Je kadert de meerwaarde hiervan.

3.1 Conclusies

- 3.1.1 Wat zijn de belangrijkste eigenschappen van websockets en welke meetbare voordelen kunnen ze bieden voor realtime communicatie in een webapplicatie, met een volledige implementatie tegen eind Mei?
- 3.1.2 Wat is de meest optimale datastructuur om een applicatie te laten connecteren met een database door gebruik van een websocket?
- 3.1.3 Is de nieuwe implementatie een verbetering op de huidige websocket die gebruikt wordt bij Level27?
- 3.1.4 SockJS versus RxJS websocket versus Socket.IO, welke implementatie presteert het best?

3.2 Aanbevelingen

Tekst tekst tekst

4 Persoonlijke reflecties en kritische kanttekeningen

*Je noteert hier hoe jij het werken aan je Graduaatsproef ervaren hebt a.d.h.v. het STARR reflectiemodel. Belangrijke persoonlijke leerinzichten en kritische kanttekeningen die je meeneemt vanuit het doorgemaakte proces kunnen hier hun plaats krijgen. Je beschrijft wat voor jou persoonlijk de meerwaarde van de uitwerking van je Graduaatsproef is; wat kan/wil je **voor jezelf** vanuit het doorlopen van je onderzoek als **leerinzichten** naar voor schuiven, koppel deze leerinzichten naar de **OLR** en **X-factor**. Daarnaast zijn ook jouw persoonlijke kritische kanttekeningen belangrijk om te vermelden: waar zit voor jou de **meerwaarde en eventuele beperkingen van je onderzoek zoals jij dit doorlopen hebt**? Belangrijk hierbij is dat je in deze reflecties de link legt met hetgeen bovenstaand werd neergeschreven en dat je in je reflecties dus verwijst naar concrete informatie/acties ... vanuit je Graduaatsproef.*

Tekst tekst tekst

5 Referentielijst

De referenties die je raadpleegde en die belangrijk zijn in de uitwerking van je Graduaatsproef dienen hier vermeld te worden. Je noteert dit op een consequente en correcte manier.

- [1] **Cursus network requests in JavaScript.** Network Requests in JavaScript by Christian Wenz (<https://app.pluralsight.com/library/courses/javascript-network-requests/table-of-contents>)
- [2] <https://developer.chrome.com/blog/removing-push>
- [3] <https://www.ioriver.io/terms/http-2-server-push>
- [4] <https://www.pubnub.com/guides/long-polling/>
- [5] <https://medium.com/deliveryherotechhub/what-is-server-sent-events-sse-and-how-to-implement-it-904938bffd73>
- [6] <https://webrtc.org/getting-started/firebase-rtc-codelab>
- [7] <https://socket.io/docs/v4/>
- [8] https://www.w3schools.com/nodejs/nodejs_mysql_create_db.asp
- [9] <https://www.npmjs.com/package/@faker-js/faker>
- [10] Officiële Socket.IO github pagina:
<https://github.com/socketio/socket.io/blob/main/LICENSE>
- [11] <https://github.com/sockjs/sockjs-client>
- [12] <https://github.com/sockjs/sockjs-node>
- [13] https://github.com/sockjs/sockjs-node/tree/main/tests/test_server
- [14] <https://github.com/sockjs/sockjs-node/tree/main/examples/echo>

6 Bijlagen

Je voegt hier je genummerde bijlagen toe (bijv de enquête die je hanteerde, de uitprint van bepaalde resultaten,) die belangrijk zijn om het verslag op een goede manier te kunnen 'lezen' en 'interpreteren'.

Tekst tekst tekst

