

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

[Woord vooraf 2](#_Toc168233468)

[Inhoudsopgave 3](#_Toc168233469)

[1 Bedrijfsvoorstelling 4](#_Toc168233470)

[1.1 Situering van het bedrijf 4](#_Toc168233471)

[1.2 Werkomgeving 5](#_Toc168233472)

[2 Projectvraag, onderzoeksacties en resultaten 7](#_Toc168233473)

[2.1 Situering probleemstelling 7](#_Toc168233474)

[2.2 Projectvraag en deelvragen 8](#_Toc168233475)

[2.2.1 Projectvraag 8](#_Toc168233476)

[2.2.2 Deelvragen 8](#_Toc168233477)

[2.3 (Onderzoeks-)acties 9](#_Toc168233478)

[2.3.1 Wat zijn de belangrijkste eigenschappen van de op dit moment bestaande websockets en welke meetbare voordelen kunnen ze bieden voor realtime communicatie in een webapplicatie, met een volledige implementatie? 9](#_Toc168233479)

[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? 9](#_Toc168233480)

[2.3.3 Wat is de meest optimale datastructuur om een applicatie te laten connecteren met een server door gebruik van een websocket? 9](#_Toc168233481)

[2.4 Verzamelde resultaten 9](#_Toc168233482)

[2.4.1 Wat is een websocket? 9](#_Toc168233483)

[2.4.2 Websocket implementaties: Socket.IO 11](#_Toc168233484)

[2.4.3 Websocket implementaties: RxJS websocket 13](#_Toc168233485)

[2.4.4 Websocket implementaties: SockJS 14](#_Toc168233486)

[2.4.5 Websocket implementaties: Alternatieven voor websockets 16](#_Toc168233487)

[2.4.6 Websocket vergelijkingsstudie: RxJS Websocket versus Socket.IO versus *native* websocket 19](#_Toc168233488)

[2.4.7 Applicatie architectuur met gebruik van websockets 22](#_Toc168233489)

[3 Conclusies en aanbevelingen 25](#_Toc168233490)

[3.1 Conclusies 25](#_Toc168233491)

[3.2 Aanbevelingen 26](#_Toc168233492)

[4 Persoonlijke reflecties en kritische kanttekeningen 26](#_Toc168233493)

[4.1 Situatie 26](#_Toc168233494)

[4.2 Taak 26](#_Toc168233495)

[4.3 Actie 26](#_Toc168233496)

[4.3.1 Onderzoek en Analyse 26](#_Toc168233497)

[4.3.2 Implementatie en Testing 27](#_Toc168233498)

[4.3.3 Samenwerking 27](#_Toc168233499)

[4.3.4 Documentatie en Presentatie 27](#_Toc168233500)

[4.4 Resultaat 27](#_Toc168233501)

[4.5 Reflectie en Leerinzichten 28](#_Toc168233502)

[4.5.1 Technische Vaardigheden 28](#_Toc168233503)

[4.5.2 Probleemoplossend Denken 28](#_Toc168233504)

[4.5.3 Samenwerking en communicatie 28](#_Toc168233505)

[4.5.4 Veiligheid en Privacy 28](#_Toc168233506)

[4.6 Krisitische Kanttekeningen 28](#_Toc168233507)

[4.7 Koppeling naar de X-factor van PXL 28](#_Toc168233508)

[4.7.1 (Em)passie 28](#_Toc168233509)

[4.7.2 Internationaal samen(net)werken 29](#_Toc168233510)

[4.7.3 Ondernemend & Innovatief 29](#_Toc168233511)

[4.7.4 Multi-& desciplinariteit 29](#_Toc168233512)

[5 Referentielijst 29](#_Toc168233513)

[6 Bijlagen 31](#_Toc168233514)

# Bedrijfsvoorstelling

## 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 of Virtual Machine. Dit is een *software*-emulatie van een fysieke computer die een besturingssysteem en applicaties kan draaien alsof het een echte, zelfstandige computer is.), met constante *back-ups* en beveiliging. Buiten *dedicated* hosting is er ook shared hosting, dit is een kosteneffectieve oplossing door servers te delen met andere klanten en tot slot is er ook nog *agency hosting* verdere uitleg in dit hoofdstuk.

Het bedrijf biedt 24/7 ondersteuning en ondersteund de servers door regelmatige *back-ups* die elk 15 minuten worden gemaakt. Het bedrijf stelt hen in staat om hoogwaardige IT-infrastructuur en -diensten te leveren aan kleine en middelgrote ondernemingen die behoefte hebben aan robuuste en betrouwbare onlineoplossingen.

Bovendien maakt Level27 gebruik van geavanceerde datacenters in België, uitgerust met de nieuwste technologieën om snelle connectiviteit, hoge *uptime* (*Uptime*: is de tijdsduur dat een systeem, server of netwerk zonder onderbreking operationeel is en beschikbaar blijft voor gebruik.) en robuuste beveiligingsmaatregelen te garanderen. Deze infrastructuur speelt een cruciale rol in het vermogen van het bedrijf om stabiele en efficiënte *hosting*-diensten te leveren aan klanten zowel lokaal als internationaal.

***Dedicated* Hosting**:

*Dedicated hosting* bij Level27 betekent dat een klant een virtuele server tot zijn beschikking heeft. Dit biedt maximale controle en prestaties, aangezien de VM niet gedeeld wordt met anderen. Elke hypervisor (Een hypervisor is *software* die virtuele machines aanmaakt en uitvoert door de hardwarebronnen van een *hostmachine* te verdelen. Er zijn twee types: type 1 draait direct op de hardware voor betere prestaties, terwijl type 2 op een bestaand besturingssysteem draait.) is uitgerust met VM’s, wat zorgt voor hoge prestaties en beveiliging. Level27 zorgt ervoor dat de servers *up-to-date* blijven en maakt elke 15 minuten *back-ups*, op het einde van de dag wordt er een *back-up* gemaakt die 30 dagen wordt 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 *virtual machines* voor.

A green square grid with black border

Description automatically generated with medium confidence

Figuur 1 Vituele machines (Jasper Orens)

**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 (CPU, Central Processing Unit, Dit is een hardware onderdeel van een computer wat bepaald hoe snel de computer taken kan verwerken.) of geheugen gebruikt. *Shared* en *dedicated hosting* bij Level27 zijn voorzien van regelmatige *back-ups* en een gebruiksvriendelijk controlepaneel. Dit type hosting is geschikt voor kleinere websites of bedrijven met een beperkt budget die geen uitgebreide serverbronnen nodig hebben. In het voorbeeld hieronder is voorgesteld hoe één Virtual Machine wordt opgesplitst in verschillende virtuele mappen. Iedere gebruiker van shared hosting neemt één of meerdere van deze partities in gebruik.

A screenshot of a computer game

Description automatically generated

Figuur 2 Virtuele machine met virtuele mappen (Jasper Orens)

**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.

A screenshot of a computer game

Description automatically generated

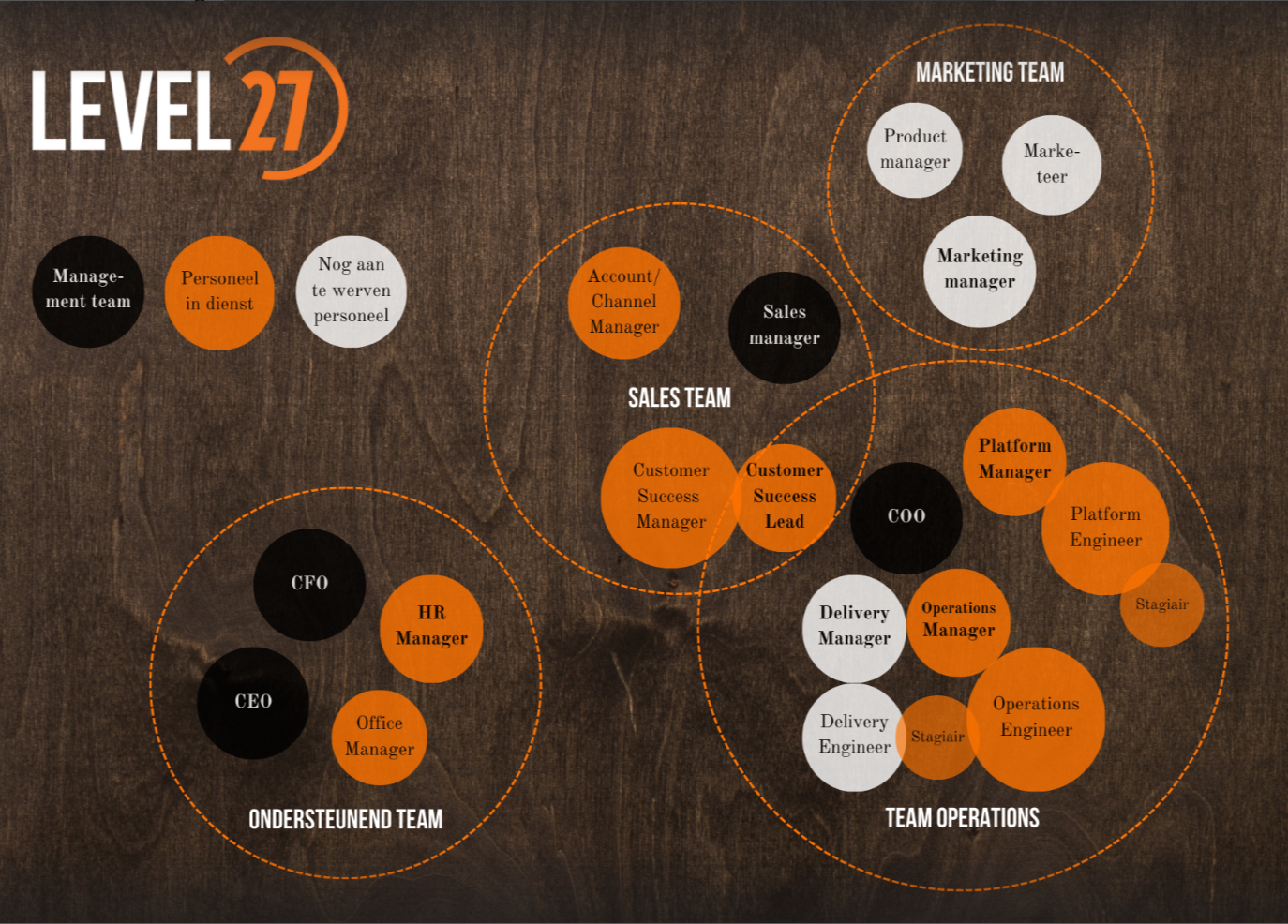
Figuur 3 Virtuele machine met containers rond virtuele mappen (Jasper Orens)

## 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 Ruby op het Chef-*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 projectmanagement*tool*, waardoor het team efficiënt en overzichtelijk kan werken in een aangepaste vorm van *Scrum*. Er vinden *daily standups* plaats, ze werken met 2-wekelijkse *releases*, voeren *retrospectives* uit en delen het werk op in blokken

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.

De structuur van Level27 ziet er als volgt uit.



Figuur 4 Organisatie Level27 (Roald Lenaerts)

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 onlineoplossingen zoeken.

# Projectvraag, onderzoeksacties en resultaten

## Situering probleemstelling

Voor een bedrijf dat actief is in onlinehosting is het essentieel dat data efficiënt en correct wordt verstuurd van server naar client (de gebruiker). In de hudige situering heeft dit specifiek betrekking tot een visueel onderdeel in het controle panneel voor Level27. De klant klikt op een knop dat gegevens verstuurt naar de servers en de servers moeten de aanpassingen direct vertonen op hun huidige pagina zonder dat de pagina moet verversen. Één van de mogelijke manieren om dit te realiseren is met behulp van een websocket. Websockets leggen een continue verbinding tussen een applicatie en een server hierdoor kan data in *realtime* worden geüpdatet.

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. Een websocket voorziet data-overdracht in beide richtingen (ookwel tweerichtingsverkeer communicatie genoemd) maar deze richting voor data te versturen wordt niet gebruikt omdat de data wat de gebruiker verstuurt eerst door de *backend* wordt behandel.

In onderstaand voorbeeld is de huidige implementatie getoond. In dit voorbeeld heeft de gebruiker op een knop geklikt om een server af te sluiten. De groene lijn toont de data die invloed heeft op de websocket.

A diagram of a data flow

Description automatically generated

Figuur 5 Huidige applicatie architectuur bij Level27 (Jasper Orens)

Om de gegevens in de database aan te passen, worden wijzigingen verstuurt 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 status van running naar shutting down in testellen, De kleur van een knop op de gebruikersinterface wordt dan gewijzigd zodra de data is geüpdatet Een voorbeeld van de opdracht is hier te zien.



Figuur 6 Verzonden taak naar de queue (Jasper Orens)

Deze queue dient als vangnet voor het geval er een probleem optreedt met de socketserver. En om ervoor te zorgen dat de server niet overspoeld geraakt. 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.

## Projectvraag en deelvragen

### Projectvraag

* Wat is op dit moment de meest geschikte manier om een websocket te implementeren voor een hosting omgeving?

### Deelvragen

* Wat zijn de belangrijkste eigenschappen van de op dit moment bestaande websockets en welke meetbare voordelen kunnen ze bieden voor realtime communicatie in een webapplicatie, met een volledige implementatie?
* 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 server door gebruik van een websocket?

## (Onderzoeks-)acties

### Wat zijn de belangrijkste eigenschappen van de op dit moment bestaande websockets en welke meetbare voordelen kunnen ze bieden voor realtime communicatie in een webapplicatie, met een volledige implementatie?

Het onderzoek zal starten met het opzoeken van lessen die betrekking hebben op websockets. De resultaten zullen relatief beperkt zijn voor zo'n specifiek onderwerp als websockets. Pluralsight heeft een cursus [1] ter beschikking met betrekking tot HTTP-methodes (dit is een andere manier om te communiceren met een database) en websockets. Deze cursus biedt basiskennis over websockets. Dit zal zorgen voor een goede start om het concept te begrijpen. Als volgt zal het duidelijk worden welke eigenschappen zorgen voor een goede websocket implementatie. Er zal ook online research gedaan worden.

### 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 de werking van websockets beter te begrijpen, zal er uit het onderzoek blijken welke websockets het best gebruikt worden. Er zal een server/applicatie in het Node.Js framework worden opgesteld om de noodzakelijke data mee te versturen zodat alle websocket implementaties en libraries kunnen werken. (een library is een verzameling van *tools* voor een programmeer taal). In de eerste fase zal de Node.js server zelf worden opgesteld en als tweede stap zal de database hierin worden opgemaakt. Er zal data worden verstuurd en ontvangen van zowel de client als de server. De server zal bij iedere transmissie een string (een woord) van een lijst met groenten versturen, en de (*frontend*) client verstuurt een string (een woord) van een lijst met fruit. De *frontend* zal de snelheid van het binnenkomende en uitgaande dataverkeer tonen van beide voorbeeld lijsten.

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

Er zal onderzoek worden gedaan naar een basis applicatiestructuur wanneer er gebruik wordt gemaakt van een database. Een applicatiestructuur of datastructuur is de organisatie van hoe onderdelen van een website met elkaar spreken. Hoe de *backend* met de *frontend* en de database communiceert bijvoorbeeld. Vervolgens zal er ook onderzoek gedaan worden naar het implementeren van een websocketserver in deze basis structuur. De huidige applicatie structuur bij Level27 zal vergeleken worden met het onderzoek.

## Verzamelde resultaten

### Wat is een websocket?

**Algemene werking van websockets**:

Een websocket legt een continue, tweerichtingsverkeer (een communicatiemethode die in 2 richtingen werkt) verbinding tussen een client en een server, waardoor realtime communicatie (communicatie zonder wachttijd of vertraging) mogelijk is. WS (websocket) biedt deze verbinding onbeveiligd, terwijl WSS (websocket secure) dezelfde functionaliteit biedt maar met beveiliging (encryptie) van SSL/TLS voor een veilige gegevensoverdracht. (SSL/TLS zijn protocollen die veilige en versleutelde communicatie tussen een *webbrowser* en een server mogelijk maken, waardoor gegevensbescherming en privacy tijdens internettransacties worden gegarandeerd.)

**Handshake**

Dit is de aanvraag om een connectie te maken tussen de gebruiker en de websocket. Dit is nodig om een initiële verbinding te leggen om op deze manier een ‘brug’ te openen tussen de client en *backend*, Zonder de *handshake* kan de brug niet geopend worden. De client verstuurt een *Request* met ‘*upgrade’* als onderdeel van de *header* (dit is een onderdeel van het verzoek). Deze heeft als waarde ‘websocket’ en een veld ‘*connection’* met als waarde ‘*Upgrade’*.



Figuur 7 Screenshot van console in Firefox van een websocket implementatie (Jasper Orens)

De client verstuurt dan een BASE64 encoded Key (Dit is een manier om gewone letters en cijfers om te zetten in een speciale code, zodat het makkelijk en veilig verstuurd kan worden. Het is een zoals een geheime taal die alleen de computer begrijpt). Wanneer de Server dan een reactie geeft, indien deze bekwaam is om 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 (dit is een unieke identificatie) aan toe en berekent hier een SHA-hash voor (een hash is het verbergen van woorden). 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 voordoet als de server. Zodra deze acties zijn uitgevoerd, is er tweerichtingsverkeer communicatie tussen de server en de client.

A blue and black rectangle with triangles

Description automatically generated

Figuur 8 Websocket connectie (Jasper Orens)

**Voordelen van websockets**

* Minder *overhead*: (*Overhead* zijn extra bytes die worden toegevoegd aan een pakket) Door de tweerichtingsverkeer verbinding is er minder *overhead* in vergelijking met traditionele HTTP-verzoeken die telkens een nieuwe verbindingen moeten opzetten.
* *Realtime* *updates*: Ideaal voor toepassingen die *realtime*-gegevensupdates vereisen, zoals *chat-apps*, live sportverslaggeving en financiële tickers.
* Lagere *Latency*: (Latency is de tijdsvertraging tussen het verzenden van een signaal en de ontvangst ervan.) Websockets verminderen de vertraging 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: (Scalability/Schaalbaarheid is het vermogen van een systeem om prestaties te behouden of verbeteren) Het onderhouden van een groot aantal gelijktijdige websocket-verbindingen kan schaalbaarheidsproblemen veroorzaken.

Use case

Chat-apps: 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 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: (smart technologie): 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.

### Websocket implementaties: Socket.IO

Wat is het?

Socket.IO is een implementatie die zorgt voor snelle, tweerichtingscommunicatie op basis van gebeurtenissen tussen client en server. De connectie kan opgesteld worden tussen verschillende *realtime* webtechnologiën. Deze zijn:

* HTTP long-polling (Dit is een HTTP-verzoek)
* Websocket
* Webtransport

Socket.IO kiest zelf het beste pad afhankelijk van

* De mogelijkheden van de browser
* Het netwerk (sommige netwerken blokkeren 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 pakket. 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 [2] 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 de manier waarop de data verstuurd wordt, maar ook de browser heeft invloed op de totale performantie. Over het algemeen verbruikt Firefox minder CPU dan Chrome [3]..

**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 [4]

**Licentie en kosten**

Socket.IO is gratis en wordt verdeeld onder het MIT-licentie. [5]

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 reconnecties en *event broadcasting*. Dit maakt het geschikt voor *realtime* webapplicaties zoals *chats* en interactieve collaboratieve *tools*. Voor meer technische details. [6, 7]

**Voordelen**

- Automatische fallback naar HTTP long-polling: Waar gewone websocketimplementaties afhankelijk zijn van de permanente beschikbaarheid van een websocket-verbinding, kan Socket.IO automatisch terugvallen op HTTP long-polling (meer informatie over long-polling wordt aangehaald in 2.4.5) 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 omvang van de berichten vergeleken met een native 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.

### Websocket implementaties: RxJS websocket

Wat is het?

De RxJS websocket is uniek doordat deze werkt met het Reactive Extensions framework, waardoor het eenvoudig kan worden gecombineerd met andere RxJS *operators* (bijvoorbeeld ‘map’, ‘filter’, ‘merge’ en ‘switchMap’’, deze helpen je om dataverkeer te transformeren, filteren, combineren en te schakelen tussen nieuw dataverkeer) Het biedt een WebsocketSubject dat zowel een *Observer* als *Observable* is wat bidirectioneele 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 *realtime*-applicaties.

Gebruiksgemak en documentatie

Installatie gaat zeer vlot, de documentatie legt de installatie helder en duidelijk uit zodat de gebruiker probleemloos kan volgen.

Performance en schaalbaarheid

Het gebruik van complexe *operator*-ketens, (‘map’, ‘filter’, ‘merge’ bijvoorbeeld), kan invloed hebben op de snelheid. Het is belangrijk om goed op je *resources* te letten, zoals het beheren van actieve verbindingen, om geheugenproblemen te voorkomen. RxJS biedt krachtige hulpmiddelen voor schaalbaarheid zoals het beheren van meerdere taken die gelijktijdig lopen en het omgaan met overbelasting van data, zodat er op een efficiënte manier grote hoeveelheden gegevensstromen verwerkt kunnen worden.

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 NjRx. 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 gebruik van gemaakt wordt bij een project waar men Reactive Extensions framework gebruikt, is het eenvoudig te combineren met andere *operators* van RxJS en zorgt voor een verbeterd dataflowbeheer.

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

- Bidirectioneele communicatie: WebsocketSubject fungeert als zowel een *Observer* als een *Observable*, wat bidirectioneele communicatie vergemakkelijkt. Onder Observer wordt verstaan een component die luistert naar en reageert op updates van een Observable, terwijl een Observable een component is die updates uitzendt naar zijn Observers, hetgeen bidirectioneele communicatie mogelijk maakt doordat het WebsocketSubject beide rollen vervult.

Nadelen

- Complexiteit: De uitgebreide mogelijkheden en integraties kunnen leiden tot een hogere leercurve voor nieuwkomers.

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

### Websocket implementaties: 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. Uit het onderzoek blijkt dat er conflicten kunnen ontstaan met React. Er is geen officiële website want alle nodige documentatie is te vinden op hun GitHub [7, 8] De installatie en basisconfiguratie zijn goed gedocumenteerd, wat het makkelijk maakt voor ontwikkelaars om snel aan de slag te gaan. Bovendien biedt SockJS voorbeeldprojecten [9, 10] 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. (Cross-origin requests zijn verzoeken die worden gedaan van één domein naar een ander domein, vaak beperkt door beveiligingsmaatregelen zoals CORS.)

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 vertraging introduceren in vergelijking met *native* websockets.

Ondersteuning en *community*

SockJS heeft een actieve gemeenschap en goede ondersteuning via forums zoals GitHub en Stack Overflow. Er is een uitgebreide documentatie beschikbaar, evenals talrijke *tutorials* en voorbeeldprojecten die ontwikkelaars kunnen ondersteunen 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 keuzemogelijkheden biedt 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 vertraging 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.

### Websocket implementaties: Alternatieven voor websockets

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 om 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 websockettechnologiën. Daarom richt het onderzoek zich op meer directe en brede websocketimplementaties zoals Sock.js, Socket.IO, en RxJS websocket, die relevanter zijn voor een breder scala aan toepassingen.

HTTP/2 Push

Er werd onderzoek gedaan 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 die *push* *requests* hebben verzonden en ontvangen. Uit hun onderzoek bleek dat 1.25% van HTTP/2 websites gebruik maakten van deze functie. In een latere fase van het onderzoek bleek dat dit aantal gezakt was naar 0.7%. Op basis van deze analyse besloot Google om deze functie niet verder uit te werken.[11]

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 optimaal te laten renderen. In plaats van te wachten tot de client afzonderlijke verzoeken voor elke bron verzendt, kan de server proactief beslissen om bepaalde bronnen al 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. [12]

A black background with green arrows

Description automatically generated

Figuur 9 HTTP Push connectie (Jasper Orens)

Long Polling

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 application/json, tekst/plain en de open connectie blijft 100 tot 300 seconden geldig.

A screenshot of a video game

Description automatically generated

Figuur 10 Long polling werking (Jasper Orens)

Use case

De beste benutting voor Long polling is voor *chat-apps* 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 verstuurt 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 stuurt. [13]

Server-Sent Events (SSE)

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 vervolgens meerdere antwoorden in de vorm van ‘*event-stream’* in *chunks*. Dit zijn grote blokken code die werden opgesplitst. Er wordt één *request* gestuurd om 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 als *Prefix* verstuurd, de connectie blijft open tot de client een disconnect request verstuurt.

A screenshot of a computer screen

Description automatically generated

Figuur 11 Server sent events werking (Jasper Orens)

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 tweerichtingsverkeer communicatie is. In plaats daarvan biedt SSE aan servers de mogelijkheid om gegevens in realtime naar clients te sturen, terwijl clients geen mogelijkheid hebben om direct met 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 gereduceerd en de efficiëntie van de communicatie wordt geoptimaliseerd. [14]. Er is geen verbetering nodig van client naar server bij het onderzoek, echter er wordt wel aangeraden om deze te behouden en deze optie niet volledig af te schrijven bij Level27.

Het onderzoek gaat niet dieper in op Pusher en GraphQL Subscriptions omdat deze diensten vaak hogere *overhead* met zich meebrengen.

### Websocket vergelijkingsstudie: RxJS Websocket versus Socket.IO versus *native* websocket

**Inleiding**

In dit onderzoek analyseren we drie verschillende websocket implementaties binnen één enkele applicatie: Socket.IO, RxJS websocket en native websocket. Deze implementaties worden vergeleken op basis van hun prestaties en efficiëntie in dataoverdracht tussen een server en een *frontend*.

**Applicatie Overzicht**

De applicatie is ontworpen om bidirectioneele communicatie te demonstreren tussen een Node.js server en een React *frontend*. Het hoofddoel is om een continu dataverkeer te genereren en meten tussen de server en de client, waarbij de server groenten stuurt en de client fruit terugstuurt.

**Componenten van de server**

* **Socket.IO Server:**
* Gebruikt het Socket.IO bibliotheek voor WebSocket communicatie.
* Stuurt groenten naar de client en ontvangt fruit terug.
* Meet de totale verzonden en ontvangen bytes, evenals de snelheden.
* **RxJS WebSocket Server:**
* Gebruikt de RxJS bibliotheek voor WebSocket communicatie.
* Functioneert op een vergelijkbare manier als de Socket.IO server met groenten en fruit uitwisseling.
* Houdt statistieken bij zoals verzonden en ontvangen bytes en snelheden.
* **Native WebSocket Server (ws)**
* Implementeert native WebSocket communicatie zonder extra bibliotheken.
* Stuurt groenten naar de client en ontvangt fruit terug.
* Houdt verzend- en ontvangstatistieken bij.

**Overzicht van de applicatie structuur**

**A diagram of a server

Description automatically generated**

Figuur 12 Websocket vergelijkingsproject (Jasper Orens)

**Metingen en vergelijking**

Elke server stuurt groenten op regelmatige intervallen naar de client. De client stuurt vervolgens fruit terug naar de server. Beide zijden meten en rapporteren: De applicatie beheert de totale bytes, zowel verzonden en ontvangen: Hoeveelheid data die over de websocket verbinding is gegaan en verzend- en ontvangsnelheden: De snelheid waarmee data wordt overgedragen. En tot slot houdt het ook een overzicht bij van de maximale snelheden: De hoogste gemeten verzend- en ontvangsnelheden tijdens de sessie.

A screenshot of a computer

Description automatically generated

Figuur 13 Vergelijkingsapplicatie (Jasper Orens)

**Resultaten van de vergelijkingsstudie**

RxJS Websocket presteert het beste op het gebied van snelheid. De native websocket presteert het beste op het vlak van efficiëntie.

* Socket.IO: 3.62 bytes per object
* RxJS websocket: 3.72 bytes per object
* Native websocket: 3.30 bytes per object

### Applicatie architectuur met gebruik van websockets

Wat is een applicatie architectuur?

Een applicatie-architectuur is een model dat de structuur en het gedrag van een applicatie beschrijft. Tevens geeft het een overzicht van de verschillende componenten van de applicatie en hoe deze met elkaar communiceren. Dit model helpt ontwikkelaars om de applicatie efficiënter te ontwerpen, te bouwen en te onderhouden. Het omvat aspecten zoals de keuze van technologieën, de organisatie van code, en de interactie tussen verschillende modules. Een goede applicatie-architectuur zorgt voor schaalbaarheid, flexibiliteit en herbruikbaarheid van de componenten.

De basisstructuur

Een website met een databank die connectie legt met een server voor data heeft altijd nood aan minstens drie applicaties.

* De *frontend*, dit is de website waar de gebruiker op terecht komt. Deze website kan geschreven zijn in PHP, React, Vue, basis HTML en JavaScript of één van de vele andere *frontend*talen.
* De *backend*: Dit kan een REST API zijn, een applicatie op het internet dat spreekt met de *frontend* en de database om data door te geven van de ene component naar de andere. Een applicatie structuur kan theoretisch zonder een *backend* werken en direct met de database spreken. Maar dit is niet aan te raden gezien dit de integriteit van de database in gevaar brengt. Een *backend* is altijd aanbevolen om een vorm van veiligheid te garanderen, zelfs als er maar beperkt gesproken wordt met de database.
* De database: De database is de locatie op een server waar alle gegevens die de website nodig heeft te vinden. Dit kan variëren van gebruikersgegevens zoals een gebruikersnaam, een wachtwoord, maar de database kan ook componenten van de *frontend* bewaren zoals afbeeldingen.

De structuur met een websocketserver

Zodra er een websocket wordt geïmplementeerd is het aangeraden om een websocketserver toe te voegen aan de applicatie structuur. Deze structuur ziet er als volgt uit. De websocketserver dient als connectie tussen de *backend* en de *frontend* om data veilig te versturen in beide richtingen.

A diagram of a websocket server

Description automatically generated

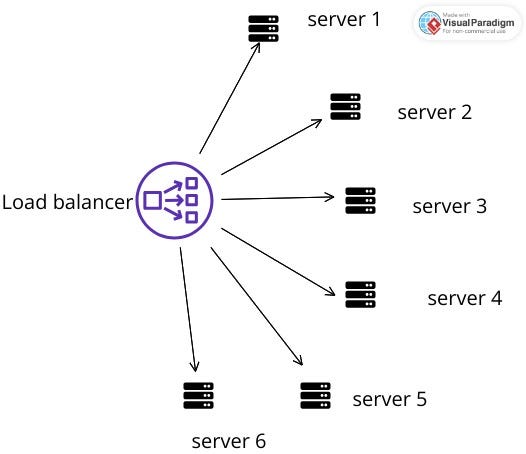
Figuur 14 Applicatie structuur met websocket (Better Programming)

De toevoegingswaarde van een queue.

Een queue is een datastructuur die werkt volgens *het First-In-First-Out* (FIFO) principe, waarbij het eerste item dat binnenkomt als eerste wordt verwerkt. Het wordt gebruikt voor asynchrone taakverwerking, waardoor systemen beter met piekbelastingen kunnen omgaan en betrouwbaarder worden. Queues helpen bij het verdelen van werk over meerdere servers, wat de schaalbaarheid van applicaties verbetert en de integratie tussen verschillende systemen vereenvoudigt. Ze verhogen de betrouwbaarheid door taken op te slaan totdat ze kunnen worden verwerkt, zelfs bij systeemstoringen. [15] Een queue voegt aanzienlijke waarde toe aan een applicatie-architectuur bestaande uit een *backend*, *frontend*, database en websocketserver. Hier zijn enkele specifieke voordelen en toepassingen:

****Betere schaalbaarheid en betrouwbaarheid****:

Load Balancing: Een queue kan helpen bij het gelijkmatig verdelen van de belasting over meerdere servers. Dit zorgt ervoor dat je *backend* niet overweldigd raakt door pieken in de verzoeken. Bijvoorbeeld, wanneer een websocketserver berichten ontvangt, kunnen deze via een queue worden verspreid naar meerdere *backend* servers voor verwerking​ [16].



Figuur 15 Load balancer (Medium.com) [17]

Failover Mechanismen: Door gebruik te maken van queues, kunnen berichten worden opgeslagen en opnieuw worden verzonden in het geval van een serverfout, wat bijdraagt aan de betrouwbaarheid van je systeem​. [18]

****Losse koppeling en modulariteit****:

Asynchrone Verwerking: Queues maken het mogelijk om taken asynchroon te verwerken, waardoor de verschillende componenten van je applicatie los van elkaar kunnen werken. Dit is handig voor taken zoals het verwerken van zware database-operaties of het verwerken van gebruikersactiviteit die niet direct een reactie vereist​. [19, 20]

Eenvoudiger Onderhoud en Uitbreiding: Omdat componenten minder strak gekoppeld zijn, kunnen nieuwe functionaliteiten gemakkelijker worden toegevoegd of bestaande componenten worden aangepast zonder het hele systeem te beïnvloeden​. [21]

*****Realtime Updates* en *Event-driven* Architectuur****:

Berichten Distributie: Met behulp van queues kunnen berichten efficiënt worden gedistribueerd naar verschillende diensten en gebruikers. Dit is vooral nuttig bij scenario's waar *realtime* *updates* nodig zijn, zoals *chat-apps*, live statistieken en collaboratieve documentbewerking​

Event Processing: Queues kunnen worden gebruikt om gebeurtenissen te beheren en te verwerken, wat essentieel is voor een *event-driven* architectuur. Dit maakt het mogelijk om snel te reageren op veranderingen en *updates* in de data​ [16, 20]

*****Performance* Verbeteringen****:

Optimalisatie van Resourcegebruik:Door gebruik te maken van queues kunnen servers efficiënter worden gebruikt. Taken kunnen in de wachtrij worden geplaatst en verwerkt wanneer er voldoende *resources* beschikbaar zijn, wat helpt om de algehele prestaties te verbeteren en *latency* te reduceren​ [20]

# Conclusies en aanbevelingen

## Conclusies

De belangrijkste eigenschappen van websockets zijn de integriteit, beschikbaarheid en toegankelijkheid van data, gebaseerd op toegangsrechten. Het onderzoek heeft waardevolle inzichten opgeleverd in andere implementaties, zoals de snelheid van RxJS websockets en het *fallback*-mechanisme van Socket.IO, dat overschakelt naar een ander protocol als websockets niet beschikbaar zijn. Echter, deze extra functies blijken uiteindelijk geen toegevoegde waarde te bieden voor de specifieke vereisten van het project.

Terugkijkend op het onderzoek kan worden geconcludeerd dat de huidige applicatie-architectuur van Level27 momenteel optimaal is. Het onderzoek heeft geen directe verbeteringen geïdentificeerd die noodzakelijk zijn voor de bestaande infrastructuur. De huidige structuur biedt effectieve *load balancing* voor het verdelen van inkomende data, wat een goede bescherming biedt tegen *Slowloris*-aanvallen. Deze aanvallen versturen namelijk een groot aantal verzoeken naar de *backend* om de servers te overspoelen en uit te putten. De architectuur zorgt momenteel voor een veilige en efficiënte dataoverdracht.

Voor de doeleinden van het project blijft de architectuur ongewijzigd. De enige wijziging die wordt voorgesteld, is de implementatie van native websockets in plaats van externe bibliotheken zoals Socket.IO of RxJS websocket.

Een *native* websocket-implementatie biedt verschillende voordelen:

Minder Overhead: Native websockets versturen minder data aangezien ze geen extra metadata toevoegen, zoals bij Socket.IO, dat functies zoals namespaces en *fallback*-mechanismen bevat. Deze extra functies zorgen voor meer gegevens bij elk pakket dat wordt verzonden. Deze *fallback*-mechanismen zijn momenteel niet meer noodzakelijk gezien de meeste browsers websockets ondersteunen.

Lagere Complexiteit: RxJS websocket vereist het Reactive Extensions framework, hetgeen resulteert in zwaardere en complexere berichtverwerking wat de prestaties negatief kan beïnvloeden.

Eenvoudige Installatie: *Native* websockets zijn eenvoudiger te installeren en te gebruiken, wat resulteert in een efficiëntere en snellere implementatie.

Door te kiezen voor *native* websockets, wordt de datatransmissie efficiënter en wordt de complexiteit verminderd, wat bijdraagt aan betere prestaties en een eenvoudigere architectuur. Deze benadering minimaliseert de *overhead* en zorgt voor een robuuste en schaalbare oplossing voor *realtime* communicatie.

## Aanbevelingen

**Native Websockets Implementatie:** Overweeg de overstap naar een native websockets implementatie in plaats van het gebruik van externe bibliotheken zoals Socket.IO of RxJS websocket. Dit kan resulteren in lagere *overhead* en minder complexiteit, wat bijdraagt aan betere prestaties en een eenvoudigere architectuur.

**Veiligheidsmaatregelen:** Versterk de beveiligingsmaatregelen door het implementeren van versleutelde verbindingen via WSS (WebSocket Secure) in plaats van WS. Hoewel WSS meer *overhead* met zich meebrengt, biedt het een hoger beveiligingsniveau hetgeen essentieel is voor gevoelige gegevensoverdracht.

**Applicatie structuur:** De huidige applicatiestructuur is de meest geschikte voor een hostingbedrijf. Het is dan ook niet nodig om hier nieuwe veranderingen te implementeren.

Deze resultaten samengebundeld zijn de meest geschikte manier om een websocket te implementeren voor een hosting omgeving.

# Persoonlijke reflecties en kritische kanttekeningen

*. 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.*

## Situatie

Tijdens mijn graduaatsproef, getiteld "Een snellere datastroom: Websockets onder de loep," werkte ik aan het evalueren en optimaliseren van de huidige websocket-implementatie bij Level27. Dit project was het sluitstuk van mijn opleiding Programmeren aan de PXL in Hasselt en vereiste een diepgaand onderzoek naar verschillende websockettechnologiën en hun toepassingen.

## Taak

Mijn taak was om de efficiëntie en prestaties van de bestaande websocket-implementatie te verbeteren. Dit omvatte het analyseren van diverse websocket-bibliotheken zoals Socket.IO, RxJS websocket, en SockJS, en het bepalen welke technologie het meest geschikt was voor de behoeften van Level27. Dit sluit aan bij OLR1. Naast een evaluatie van de huidige infrastructuur werd gevraagd om aanbevelingen te doen voor verbeteringen.

## Actie

### Onderzoek en Analyse

Naast het bestuderen van de eigenschappen en voordelen van websockets, werden verschillende implementaties vergeleken op basis van prestaties, betrouwbaarheid, en schaalbaarheid. Hierbij maakte ik gebruik van diverse bronnen zoals de officiële documentatie en community forums​. Ik heb ervoor gezorgd dat alles binnen de deontologische richtlijnen van het project bleef en opdat er geen privacy grenzen overschreden konden worden heb ik enkel met mockdata gewerkt binnen het project. Dit sluit aan bij OLR11 en OLR13.

### Implementatie en Testing

Ik implementeerde proefopstellingen van de verschillende websockettechnologiën in een Node.js-omgeving en testte deze op hun snelheid en efficiëntie in een live-omgeving​. Dit sluit aan bij OLR4, OLR5, OLR3 en OLR7.

### Samenwerking

De nauwe samenwerking met mijn begeleiders bij PXL en Level27, evenals met het platformteam, hielpen me enorm om ervoor te zorgen dat mijn bevindingen praktisch en toepasbaar waren binnen de bestaande infrastructuur van Level27​. Dit sluit aan bij OLR2, OLR6, OLR8

### Documentatie en Presentatie

Mijn bevindingen en aanbevelingen werden uitvoerig gedocumenteerd, ik presenteerde deze aan mijn team en begeleiders om zoveel mogelijk *feedback* te krijgen en verdere verbeteringen door te voeren​​. Dit sluit aan bij OLR9 en OLR10

## Resultaat

Het resultaat is een grondig uitgewerkt document met alle bevindingen. Het onderzoek heeft aangetoond dat de huidige applicatie architectuur de meest geschikte architectuur is, maar dat er wel een verbetering mogelijk is met welke websocket de connectie gelegd wordt.

## Reflectie en Leerinzichten

Onderstaande data voldoen aan de vereisten voor OLR12

### Technische Vaardigheden

Mijn kennis van websockettechnologiën en hun implementaties is aanzienlijk verbetert na me te verdiepen in deze boeiende materie. Ik leerde de nuances van verschillende bibliotheken en hoe deze kunnen worden ingezet voor specifieke use cases. Zo kan weet ik momenteel wanneer ik best gebruik maak van socket.IO en wanneer van RxJS websockets bijvoorbeeld.

### Probleemoplossend Denken

Door de complexiteit van het project moest ik creatieve oplossingen bedenken wanneer er onverwachte problemen opdoken, zoals het zoeken/testen van mogelijke alternatieven en uitzoeken of er eventueel nog andere implementaties zijn. Zo was het zoeken naar specifieke lessen naar websockets een uitdaging op leerplatformen zoals Plurarsight. Ik heb mij hierna meer zitten verdiepen in websockets via individuele onderdelen van websockets op te zoeken van websockets om zo de werking nog beter te begrijpen.

### Samenwerking en communicatie

Het werken in een multidisciplinair team heeft me geleerd hoe belangrijk effectieve communicatie is en hoe boeiend en leerrijk een goede samenwerking kan zijn voor het succes van een project. Zo heb ik samengewerkt met een zeer diverse team van mannen en vrouwen op Level27. Hun backend programmeur bevindt zich momenteel in Tenerife, dus de communicatie was al direct op internationaal niveau. Dit sluit aan bij OLR8 en OLR9 die de nadruk leggen op constructieve samenwerking en efficiënte communicatie.

### Veiligheid en Privacy

Ik heb geleerd hoe essentieel het is om veiligheids- en privacyrichtlijnen strikt te volgen bij de ontwikkeling van *software*applicaties zoals benadrukt werd in OLR13. Er is altijd mockdata gebruikt om te verhinderen dat er gewerkt werd met bestaande data die inbreuken op privacy kunnen bevatten.

## Krisitische Kanttekeningen

Naar mijn mening biedt de studie een aanzienlijke meerwaarde aan Level27, specifiek op het gebied van gebruiksgemak en efficiëntie binnen de bestaande structuur. Ik ben van mening dat het gewenste resultaat is bereikt en dat het onderzoek succesvol is afgerond. Aangezien de conclusie is getrokken om een native websocket te implementeren, zie ik persoonlijk geen toegevoegde waarde in het verder voortzetten van het onderzoek. De keuze voor een native oplossing is gemaakt vanwege de lagere complexiteit en de beperkte hoeveelheid extra data die ermee gepaard gaat. Andere niet-onderzochte websockets (genoteerd in 2.4.5 onder andere) zouden deze extra data en complexiteit met zich meebrengen.

## Koppeling naar de X-factor van PXL

### (Em)passie

Ik heb dit project met onuitputtelijke toewijding onderzocht en daarbij een diepgaande interesse ontwikkeld voor nieuwe platformen zoals Socket.IO. Deze technologieën hebben mijn verwachtingen ruimschoots overtroffen en mijn nieuwsgierigheid voortdurend aangewakkerd. Vaak werkte ik tot diep in de nacht, gedreven door mijn passie en vastberadenheid. Daarnaast was ik me voortdurend bewust van mijn omgeving en streefde ik ernaar empathisch om te gaan met mijn werkomgeving. Ik hield rekening met de tijdschema's van alle betrokkenen, om zo hun tijd en inzet te respecteren. Mijn doel was steeds om een harmonieuze en productieve samenwerking te bevorderen.

### Internationaal samen(net)werken

Gezien het onderzoek zich uitstrekte over verschillende applicatieniveaus zoals frontend, backend en serverbeheer, was het essentieel om samen te werken met experts op deze gebieden. In dit kader heb ik internationaal samengewerkt met Ola, een backend ontwikkelaar bij Level27, die vanuit Tenerife werkt voor het Belgische bedrijf. We hebben een online vergadering georganiseerd om diepgaande inzichten te verwerven in de backend aspecten van Level27. Deze internationale samenwerking was interessant voor het project en onderstreepte het belang van effectieve communicatie en samenwerking over landsgrenzen heen.

### Ondernemend & Innovatief

Tijdens mijn onderzoek heb ik proactief een project ontwikkeld met als doel de verschillende implementaties van WebSockets te vergelijken. Dit initiatief stelde mij in staat om op innovatieve wijze de variaties en voordelen van diverse WebSocket-oplossingen te evalueren en de studie van de meest nauwkeurige en bruikbare data te voorzien. Door eigenhandig een project op te zetten, kon ik niet alleen nieuwe technologieën verkennen maar ook de impact van deze technologieën op real-time datatransmissie demonstreren. Deze benadering heeft me geholpen om potentiële verbeteringen te identificeren en mijn aanbevelingen stevig te onderbouwen met gedegen argumentatie en empirisch bewijs. Mijn ondernemende geest en innovatieve benadering kwamen duidelijk naar voren in mijn vermogen om initiatief te nemen, uitdagingen aan te gaan en oplossingen te vinden die zowel creatief als doeltreffend zijn.

### Multi-& desciplinariteit

Voor het onderzoek heb ik een gedetailleerde agenda opgesteld om alle taken efficiënt te plannen en te verdelen, zodat ik de projectdeadline zou halen. Het was essentieel om me strikt te houden aan zowel mijn eigen agenda als die van anderen, vooral bij het plannen van vergaderingen. Aangezien planning niet mijn sterkste vaardigheid was, heb ik bewust gewerkt aan het verbeteren van mijn zelfdiscipline. Deze inspanningen hebben geleid tot een succesvolle verbetering van mijn planningsvaardigheden en hebben bijgedragen aan de tijdige en georganiseerde afronding van het project.

# Referentielijst

|  |  |
| --- | --- |
| [1] | C. Wenz, „Network Requests in JavaScript,” [Online]. Available: https://app.pluralsight.com/library/courses/javascript-network-requests/table-of-contents. [Geopend 2024]. |
| [2] | Miguel Piedrafita en Damien Arrachequesne, „Officiele Socket.IO github pagina,” Github, 2018. [Online]. Available: https://github.com/socketio/socket.io/blob/main/LICENSE. [Geopend 05 2024]. |
| [3] | Shan Abdul, „Which Browser Uses the Least RAM and CPU on Windows, macOS, and ChromeOS?,” Make Use Of, 20 06 2023. [Online]. Available: https://www.makeuseof.com/browser-windows-macos-chromeos-uses-least-ram-cpu/. [Geopend 05 2024]. |
| [4] | „socket.io community,” stackOverflow, [Online]. Available: https://stackoverflow.com/questions/tagged/socket.io. [Geopend 05 2024]. |
| [5] | „Socket.IO MIT License,” Github, 2018. [Online]. Available: https://github.com/socketio/socket.io/blob/main/LICENSE. [Geopend 05 2024]. |
| [6] | „Introduction,” Socket.io docs, 10 04 2024. [Online]. Available: https://socket.io/docs/v4/. [Geopend 05 2024]. |
| [7] | „SockJS-client,” Github, 09 2023. [Online]. Available: https://github.com/sockjs/sockjs-client. [Geopend 05 2024]. |
| [8] | „Sockjs-node,” Github, 12 2023. [Online]. Available: https://github.com/sockjs/sockjs-node. [Geopend 05 2024]. |
| [9] | Bryce Kahle, „SockJS-Node test,” Github, 2020. [Online]. Available: https://github.com/sockjs/sockjs-node/tree/main/tests/test\_server. [Geopend 05 2024]. |
| [10] | Bryce Kahle, „SockJs-Node Test server,” Github, 2020. [Online]. Available: https://github.com/sockjs/sockjs-node/tree/main/examples/echo. [Geopend 05 2024]. |
| [11] | Barry Pollard, „HTTP/2 Server Push verwijderen uit Chrome,” Google, 18 08 2022. [Online]. Available: https://developer.chrome.com/blog/removing-push. [Geopend 05 2024]. |
| [12] | „IO River,” [Online]. Available: https://www.ioriver.io/terms/http-2-server-push. [Geopend 05 2024]. |
| [13] | „What is Long Polling?,” Pubnub.com, [Online]. Available: https://www.pubnub.com/guides/long-polling/. [Geopend 05 2024]. |
| [14] | G\*okhan Ayrancroğlu, „What is Server-Sent Events,” Medium.com, 07 02 2022. [Online]. Available: https://medium.com/deliveryherotechhub/what-is-server-sent-events-sse-and-how-to-implement-it-904938bffd73. [Geopend 05 2024]. |
| [15] | „GeeksForGeeks,” GeeksForGeeks, 11 05 2024. [Online]. Available: https://www.geeksforgeeks.org/queue-data-structure/. [Geopend 05 2024]. |
| [16] | Harsh Pandey, „Scalable WebSocket Architecture,” Hathora , 28 09 2022. [Online]. [Geopend 05 2024]. |
| [17] | Yuvarajan, „Day 4: Load Balancers & Message Queue,” Medium.com, 31 06 2023. [Online]. Available: https://medium.com/@UVvirus/day-4-load-balancers-message-queue-6303d02630fa. [Geopend 01 06 2024]. |
| [18] | „How to architecture a realtime-heavy websockets-based web application?,” StackExchange, 2017. [Online]. Available: https://softwareengineering.stackexchange.com/questions/327644/how-to-architecture-a-realtime-heavy-websockets-based-web-application. [Geopend 05 2024]. |
| [19] | Avanthika Meenakshi, „WebSockets tutorial: How to go real-time with Node and React,” LogRocket, 29 01 2024. [Online]. Available: https://blog.logrocket.com/websocket-tutorial-real-time-node-react/. [Geopend 05 2024]. |
| [20] | Yysun, „Database-Driven Applications Using WebSockets,” Dev.to, [Online]. Available: https://dev.to/yysun/database-driven-applications-using-websockets-2b9o. [Geopend 05 2024]. |
| [21] | Tomasz Dąbrowski, „Using Spring Boot for WebSocket Implementation with STOMP,” Toptal Developers, [Online]. Available: https://www.toptal.com/java/stomp-spring-boot-websocket. [Geopend 05 2024]. |
| [22] | „Node.js MySQL Create Database,” W3 Schools, [Online]. Available: https://www.w3schools.com/nodejs/nodejs\_mysql\_create\_db.asp. [Geopend 05 2024]. |
| [23] | „Officiele Socket.IO github pagina:,” Github, 2018. [Online]. Available: https://github.com/socketio/socket.io/blob/main/LICENSE. [Geopend 05 2024]. |

# Bijlagen

[Figuur 1 Vituele machines (Jasper Orens) 4](#_Toc168233515)

[Figuur 2 Virtuele machine met virtuele mappen (Jasper Orens) 5](#_Toc168233516)

[Figuur 3 Virtuele machine met containers rond virtuele mappen (Jasper Orens) 5](#_Toc168233517)

[Figuur 4 Organisatie Level27 (Roald Lenaerts) 6](#_Toc168233518)

[Figuur 5 Huidige applicatie architectuur bij Level27 (Jasper Orens) 7](#_Toc168233519)

[Figuur 6 Verzonden taak naar de queue (Jasper Orens) 8](#_Toc168233520)

[Figuur 7 Screenshot van console in Firefox van een websocket implementatie (Jasper Orens) 10](#_Toc168233521)

[Figuur 8 Websocket connectie (Jasper Orens) 10](#_Toc168233522)

[Figuur 9 HTTP Push connectie (Jasper Orens) 17](#_Toc168233523)

[Figuur 10 Long polling werking (Jasper Orens) 18](#_Toc168233524)

[Figuur 11 Server sent events werking (Jasper Orens) 19](#_Toc168233525)

[Figuur 12 Websocket vergelijkingsproject (Jasper Orens) 21](#_Toc168233526)

[Figuur 13 Vergelijkingsapplicatie (Jasper Orens) 21](#_Toc168233527)

[Figuur 14 Applicatie structuur met websocket (Better Programming) 23](#_Toc168233528)

[Figuur 15 Load balancer (Medium.com) [17] 24](#_Toc168233529)