HO GENT

Departement IT en Digitale Innovatie

Wat is de meest performante manier om data te ontvangen en te verzenden op het web: Een vergelijkende studie tussen gRPC met Protocol Buffers en REST met JSON en proof-of-concept

Sven Pepermans

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Antonia Pierreux
Co-promotor:
Kristof van Moorter

Instelling: The Fashion Society

Academiejaar: 2020-2021

Tweede examenperiode

Departement IT en Digitale Innovatie

Wat is de meest performante manier om data te ontvangen en te verzenden op het web: Een vergelijkende studie tussen gRPC met Protocol Buffers en REST met JSON en proof-of-concept

Sven Pepermans

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Antonia Pierreux
Co-promotor:
Kristof van Moorter

Instelling: The Fashion Society

Academiejaar: 2020-2021

Tweede examenperiode

Woord vooraf

Ik heb deze bachelorproef geschreven voor het voltooien van mijn opleiding Toegepaste Informatica met als afstudeerrichting Mobile Apps. Ik heb dit onderzoek rond performantie tussen gRPC met Protocol Buffers en REST met JSON voor de implementatie van The Fashion Society uitgevoerd omdat ik grote interesse heb in het ontwikkelen back-end applicaties en services. Daarnaast vind ik de back-end structuur van The Fashion Society waarmee ik leren werken heb gedurende mijn stage zeer intrigerend en vond ik het passend om te onderzoeken of het systeem performanter te maken is door enkel het gebruikte dataformaat, en de bijhorende structuur, aan te passen. Dit onderzoek heeft mij doen inzien dat er meer dan enkel REST en JSON is voor back-end applicaties en services. Waar ik voordien standaard JSON gebruikte zal ik nu eerst grondig nadenken of het niet beter is om een alternatief dataformaat te gebruiken.

Deze bacherlorproef zou echter niet tot stand zijn gekomen zijn zonder de hulp en bijstand van enkele mensen. Wat hierop volgt is een bedanking aan alle mensen die mij gesteund en geholpen hebben bij het ontwikkelen van deze bachelorproef.

Eerst en vooral zou ik The Fashion Society en specifiek Yoerick Lemmelijn willen bedanken voor het vertrouwen dat zij in mij hebben gestoken met het toegang verlenen tot de interne test server waarop een kopie van hun systeem draait en dewelke ook gebruik maakt van gevoelige data zoals klantgegevens. Als volgt wil ik zeer graag mijn co-promotor Kristof van Moorter bedanken. Dankzij zijn uitgebreide kennis van zowel het interne systeem van The Fashion Society als back-end applicaties, microservices en dataformaten is toch wel een groot deel van mijn bachelorproef mogelijk gemaakt.

Alsook wil ik graag mijn promotor Antonia Pierreux bedanken voor de bijstand en feedback op de inhoud van deze bachelorproef en voor het altijd klaar staan voor te antwoorden op

soms, wat ik zelf kan beschrijven als domme, vragen.

Tot slot zou ik ook mijn ouders en vriendin willen bedanken voor mij te pushen op de momenten dat het nodig was zodanig dat deze bachelorproef voor de deadline zou afgeraakt zijn.

Bij deze wens ik u een aangename leeservaring toe.

Samenvatting

Op basis van dit onderzoek kan The Fashion Society kiezen om de huidige structuur van de Discovery API, zijnde REST met JSON, te behouden of over te schakelen naar gRPC met Protocol Buffers. Dit onderzoek is interessant doordat The Fashion Society als maar blijft uitbreiden en er dagelijks honderdduizenden requests passeren door zowel de Orchestrator als de Discovery API en opmerkelijke verbeteringen in performantie door de eindgebruiker, zijnde het personeel van The Fashion Society en onderliggende bedrijven, duidelijk gevoeld kunnen worden. In dit onderzoek worden REST met JSON en gRPC met Protocol Buffers onderzocht en met elkaar vergeleken. Voor het uitvoeren van dit onderzoek wordt gebruik gemaakt van de testserver van The Fashion Society waar de huidige structuur reeds op geïmplementeerd is. De twee structuren worden vergeleken op basis van performantie in twee groeperingen, de kleine payload bestaande uit minder dan honderdduizend requests en de grote payload bestaande uit vijfhonderdduizend requests. Verder in dit document zult u een inleiding tot het onderwerp vinden waarin onder andere de huidige stand van zaken, de onderzoeksvraag en het verdere verloop van deze bachelorproef beschreven staan. Alsook vindt u voor elk van beide structuren de resultaten van beide payloads.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1. Inleiding

De inleiding moet de lezer net genoeg informatie verschaffen om het onderwerp te begrijpen en in te zien waarom de onderzoeksvraag de moeite waard is om te onderzoeken. In de inleiding ga je literatuurverwijzingen beperken, zodat de tekst vlot leesbaar blijft. Je kan de inleiding verder onderverdelen in secties als dit de tekst verduidelijkt. Zaken die aan bod kunnen komen in de inleiding (**Pollefliet2011**):

- context, achtergrond
- afbakenen van het onderwerp
- verantwoording van het onderwerp, methodologie
- probleemstelling
- onderzoeksdoelstelling
- onderzoeksvraag
- ...

1.1 Probleemstelling

Zoals eerder vermeld in deze bachelorproef is The Fashion Society constant aan het uitbreiden, dit onder meer door overnames en het openen van eigen nieuwe ZEB winkels. Elk van deze nieuwe winkels werkt via het centraal systeem en maakt dus onwetend gebruik van de Discovery API bij ondermeer het informeren van de stock, verkopen van een artikel, etc. Naarmate er meer winkels zijn zullen er ook meer gelijktijdige requests afgehandeld worden door de Discovery API. Door deze toename aan requests kan het al eens zijn dat de performantie daalt. Met dit onderzoek wordt hierop ingespeeld en wordt gekeken of gRPC met Protocol Buffers een performanter alternatief kan zijn op de huidige

implementatie.

1.2 Onderzoeksvraag

De onderzoeksvragen die gevormt worden bij de vergelijking tussen gRPC met Protocol Buffers en REST met JSON zullen dan ook zijn als volgt.

- Welk dataformaat is het meest performant voor de Discovery API van The Fashion Society?
- Is gRPC met ProtocolBuffers sneller dan REST API met JSON voor de implementatie van The Fashion Society?
- Is gRPC met ProtocolBuffers efficiënter dan REST API met JSON voor de implementatie van The Fashion Society?

1.3 Onderzoeksdoelstelling

Wat is het beoogde resultaat van je bachelorproef? Wat zijn de criteria voor succes? Beschrijf die zo concreet mogelijk. Gaat het bv. om een proof-of-concept, een prototype, een verslag met aanbevelingen, een vergelijkende studie, enz.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

In de inleiding is duidelijk geworden dat het onderzoek gericht zal zijn op twee mogelijke dataformaten aan de hand van 2 verschillende structuren, namelijk JSON aan de hand van REST en gRPC aan de hand van Protocol Buffers. Om dit onderzoek volledig te kunnen begrijpen is het belangrijk om de werking en de basisprincipes van deze dataformaten en de daarbij behorende structuren te begrijpen. Om deze reden zullen eerst de werking en basisprincipes van JSON en REST worden uitgelegd, en als volgt die van gRPC en Protocol Buffers.

2.1 JSON

2.1.1 Algemeen

JavaScript Object Notation of in het kort, JSON, is zoals beschreven in Standard ECMA-404 (ECMA, 2017) een tekstsyntaxis of dataformaat dat geïnspireerd is door de object literalen van JavaScript dat ook gekend staat als ECMAScript. Het maakt een gegevensuitwisseling tussen alle programmeertalen op een gestructureerde manier mogelijk. Hiervoor maakt JSON gebruik van een structuur van accolades, haakjes, dubbele punten en komma's dewelke zeer nuttig kan zijn in verschillende contexten, profielen en applicaties. Ondanks dat JSON geïnspireerd is door ECMAScript probeert het de interne data representatie ervan niet op te leggen aan andere programmeertalen, in plaats daarvan deelt JSON een onderdeel van ECMAScript's syntax met alle andere programmeertalen.

JSON mag niet gezien worden als een specificatie van een volledige gegevensuitwisseling want dat is het ook niet. Bij een zinvolle gegevensuitwisseling wordt er een overeen-

stemming over de semantiek die gekoppeld is aan een gebruik van de JSON-syntaxis tussen producent en consument vereist. JSON kan wel gezien worden als een syntactisch raamwerk waaraan een specifieke semantiek kan worden gekoppeld.

Doordat er veel verschillende types getallen zijn zoals, maar niet beperkt tot, decimale en binaire getallen, kiest JSON voor enkel een weergave van getallen die mensen gebruiken, namelijk een reeks cijfers. Ook al zijn alle programmeertalen het niet altijd eens over de interne representaties van getallen, ze weten wel hoe ze cijferreeksen moeten begrijpen.

Zoals nu wel duidelijk is kunnen programmeertalen sterk verschillen in mate van wat ondersteund wordt en hoe iets moet gerepresenteerd worden. Dit geldt ook voor objecten, niet alle programmeertalen ondersteunen objecten, en indien ze dit wel doen is er nog steeds een groot verschil in de kenmerken en beperkingen de objecten bieden. Met andere woorden kunnen de modellen van objectsystemen heel erg uiteenlopen. Om dit probleem aan te pakken biedt JSON een eenvoudige notatie aan voor het uitdrukken van verzamelingen met naam / waarde-paren. Om zulke verzamelingen weer te geven hebben de meeste programmeertalen reeds een functie zoals maar niet beperkt tot, struct, map, hash, object. Daarnaast biedt JSON ook ondersteuning voor geordende zoeklijsten, alsook hiervoor hebben alle programmeertalen een functie om deze weer te geven zoals array, vector en list. Aangezien objecten en arrays zich kunnen nesten kunnen aan de hand van JSON complexe structuren zoals boomstructuren worden gerepresenteerd.

Hieruit kan dus geconcludeerd worden dat door het aanvaarden van JSON's simpele conventies, complexe datastructuren uitgewisseld kunnen worden tussen wat anders incompatibele programmeertalen zijn.

2.1.2 Achterliggend

Een JSON-tekst bestaat uit een reeks tokens die gevormd zijn uit Unicode-codepunten en die in overeenstemming zijn met de JSON-waardegrammatica. Deze reeks tokens bevat tekenreeksen, cijfers, zes structurele token en drie letterlijke naamtokens. Hieronder volgt een opsomming van de zes structurele tokens en de drie literal name tokens samen met de bijhorende Unicode.

De zes structurele tokens:

- [Linker vierkante haak, U+005B
-] Rechter vierkante haak, U+005D
- { Linker accolade, U+007B
- } Rechter accolade, U+007D
- : Dubbele punt, U+003A
- , Komma, U+002C

De drie literal name tokens:

- true U+0074 U+0072 U+0075 U+0065
- false U+0066 U+0061 U+006C U+0073 U+0065

- null U+006E U+0075 U+006C U+006C

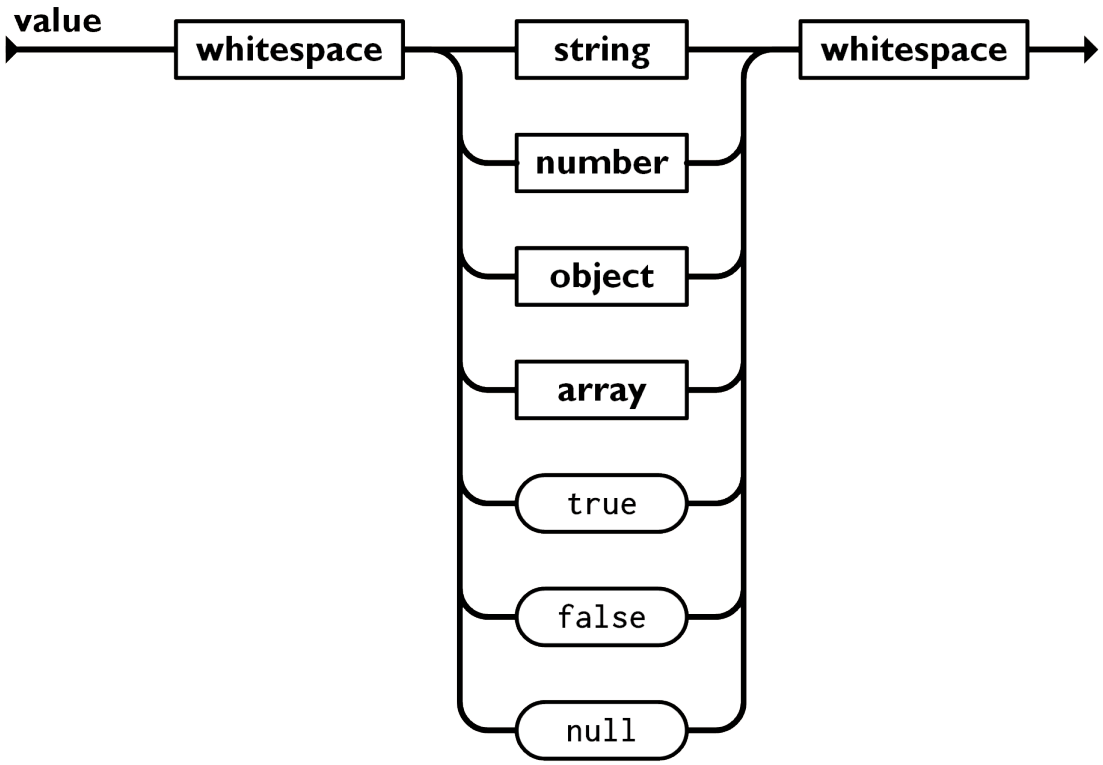
Voor of na een token is het toegestaan om onbelangrijke witruimte te gebruiken. Witruimte kan beschreven worden als een willekeurige reeks van één of meerdere van onderstaande codepunten.

- Tekentabel U+0009
- Regelinvoer U+000A
- Regelterugloop U+000D
- Spatie U+0020

Witruimte is niet in elke token toegestaan, een spatie is hierop een uitzondering en is wel toegestaan in strings.

2.1.3 Waarden

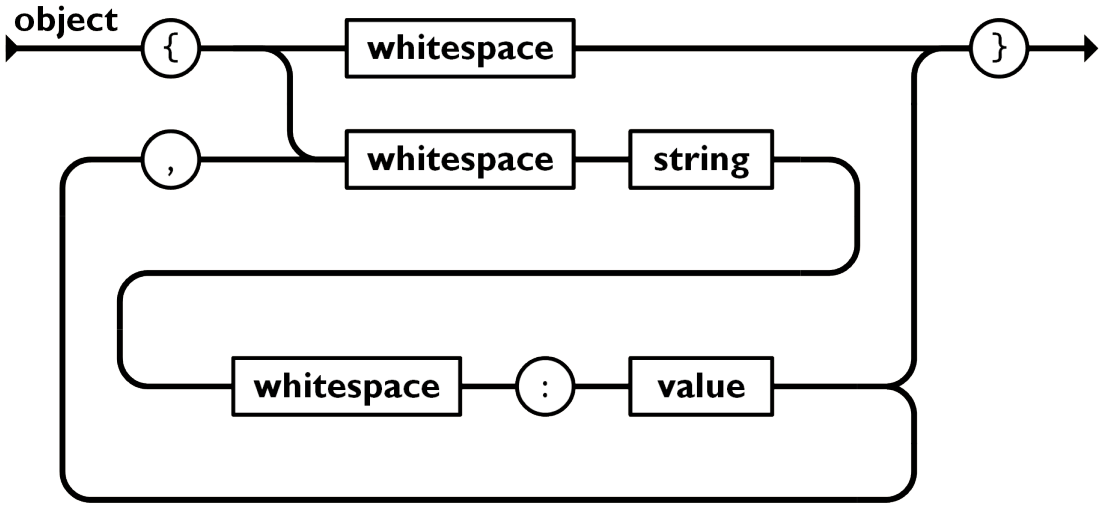
De bovengestelde tokens vormen de ruggengraad van een JSON-bestand, echter is het de bedoeling dat een JSON-bestaand iets van data overdraagd. Deze data kan aan de hand verschillende waarden worden voorgesteld, namelijk als objecten, arrays, nummers, strings, true, false, null.



Figuur 2.1: De structuur van een JSON-waarde met alle mogelijke waarden die deze kan bevatten

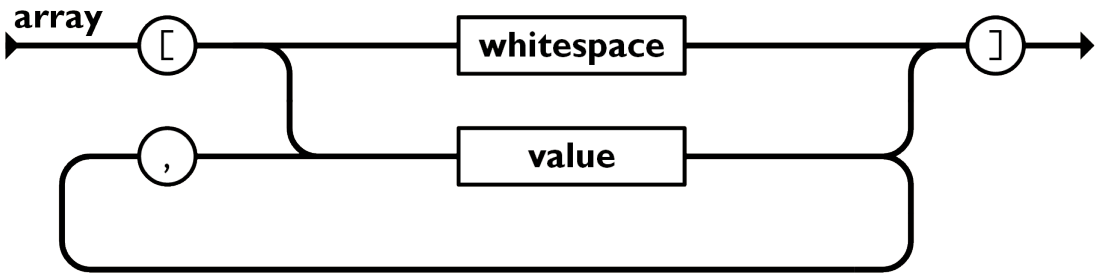
De eerste waarden die besproken worden zijn de objecten, deze worden voorgesteld over een paar accolades die geen of meerdere name/value paren kunnen incapsuleren zoals te

zien is in figuur 2.2. In zo een name/value paar is de naam een string en wordt gevolgd door een dubbele punt die de naam en waarde van elkaar onderscheidt. Optioneel is een komma na de waarde, die de waarde en de eventuele volgende naam van elkaar onderscheiden. De JSON syntax legt geen beperkingen op op vlak van de strings gebruikt als namen. Deze moeten met andere woorden niet uniek zijn en ze moeten geen bepaalde ordening volgen.



Figuur 2.2: De structuur van een JSON object.

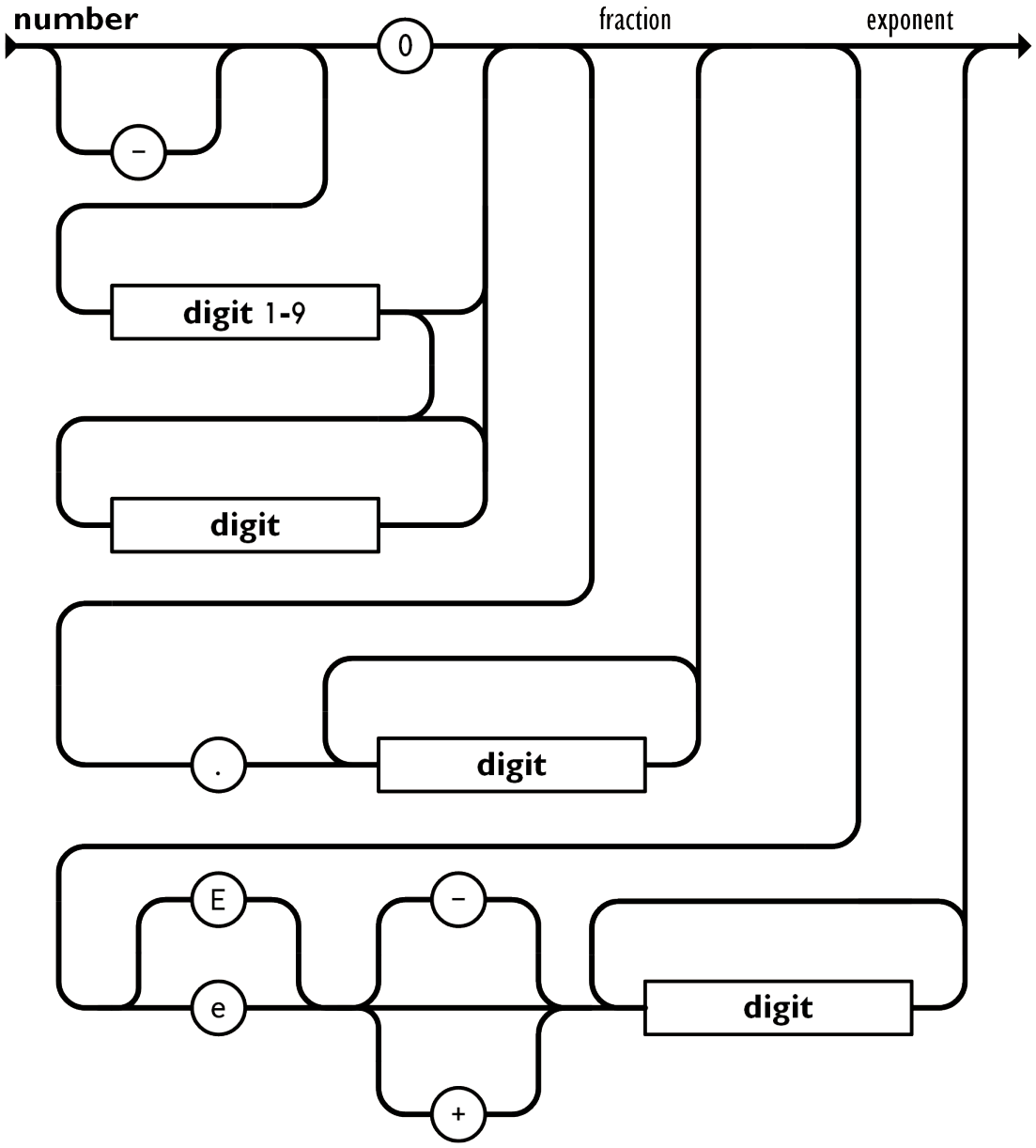
Een tweede waarde zijn de arrays, dit zijn vierkante haken die geen of meerdere waarden incapsuleren. Het bijzondere aan arrays is dat deze niet beperkt zijn tot een name/value paar maar dat deze zelf dus meerdere waarden en dus ook andere arrays kunnen bevatten en dus genest kunnen worden. Dit kan afgeleid worden uit figuur 2.3. De array structuur wordt net zoals bij objecten geen beperkingen opgelegd op vlak van de ordening van de waarden, echter wordt de array structuur vooral gebruikt in situaties waar de ordening wel enig belang heeft.



Figuur 2.3: De structuur van een JSON array.

Als volgt zijn er de nummers, een nummer kan gedifinieerd worden als een sequentie van decimale cijfers. Bij deze sequentie cijfers is er geen overbodige voorloopnul, een nummer kan wel voorafgegaan worden door een min-teken met U+002D als Unicode code. Om te werken met kommagetallen wordt een decimaal punt gebruikt. Als ook kan een nummer voorafgegaan worden door een e met U+0065 als Unicode code of een E met U+0045 als Unicode code om een exponent aan te duiden, deze kunnen eventueel ook bijgestaan

worden door + (U+002B) of - . Deze structuur is te zien in figuur 2.4. In Unicode kunnen de cijfers gevonden worden van U+0030 tot en met U+0039.



Figuur 2.4: De structuur van een JSON nummer.

Tot slot zijn er de strings, deze representeren een tekst-waarde bestaande uit een sequentie van Unicode codepunten dewelke geïncapsuleerd zijn door aanhalingstekens die als Unicode code U+0022 hebben. Binnen deze aanhalingstekens zijn er echter enkele codepunten die niet gebruikt mogen worden, namelijk de karakters die moeten worden geëscaped. Dit zijn dan aanhalingstekens, backslash (U+005C), en de control karakters zijnde U+0000 tot en met U+OO1F Voor sommige escape-reeksweergaven bestaande uit twee tekens bestaat er wel een representatie binnen de string. Hieronder een representatie van zulke karakters.

- \“ aanhalingstekens

- \\ backslash
- \/ slash
- \b backspace
- \f form feed
- \n nieuwe lijn
- \r regelterugloopkarakter
- \t karakter tabulatie karakter

Daarnaast kan ook elk codepunt weergegeven worden als een hexadecimale sequentie, de betekenis hiervan is vastgelegd in ISO/IEC 10646. Hexadecimale getallen kunnen zowel cijfers als kleine letters en hoofdletters van A tot F zijn, de cijfers zijn te vinden in Unicode van U+0030 tot en met U+0039. De hoofdletters zijn van U+0041 tot en met U+0046 terwijl de kleine letters te vinden zijn van U+0041 tot en met U+0046. De codepunten die zich bevinden in het Basic Multilingual Plane, zijnde U+0000 tot en met U+FFFF, kunnen gerepresenteerd worden als een sequentie van zes karakters, namelijk een backslash gevolgd door een kleine letter u, en tot slot gevolgd door vier hexadecimale getallen die een codepunt encoderen.

2.2 REST

Representational State Transfer (REST) is zoals beschreven door Fielding (2000) een architecturale stijl voor het ontwerpen van gedistribueerde hypermediasystemen. Om een interface als RESTful te kunnen beschrijven moet deze voldoen aan zes principes dewelke hieronder zullen worden uitgelegd.

2.2.1 Client-Server

Het eerste principe is Client-Server, hiermee wordt separation of concerns bedoeld. Door de concerns omtrent de user interface op te splitsen van de data storage concerns wordt de portabiliteit van de user interface verbeterd samen met de schaalbaarheid door het versimpelen van de server componenten. Als ook laat de opsplitsing toe dat de componenten zelfstandig kunnen groeien.

2.2.2 Stateless

2.2.3 Cacheable

2.2.4 Uniform interface

2.2.5 Layered system

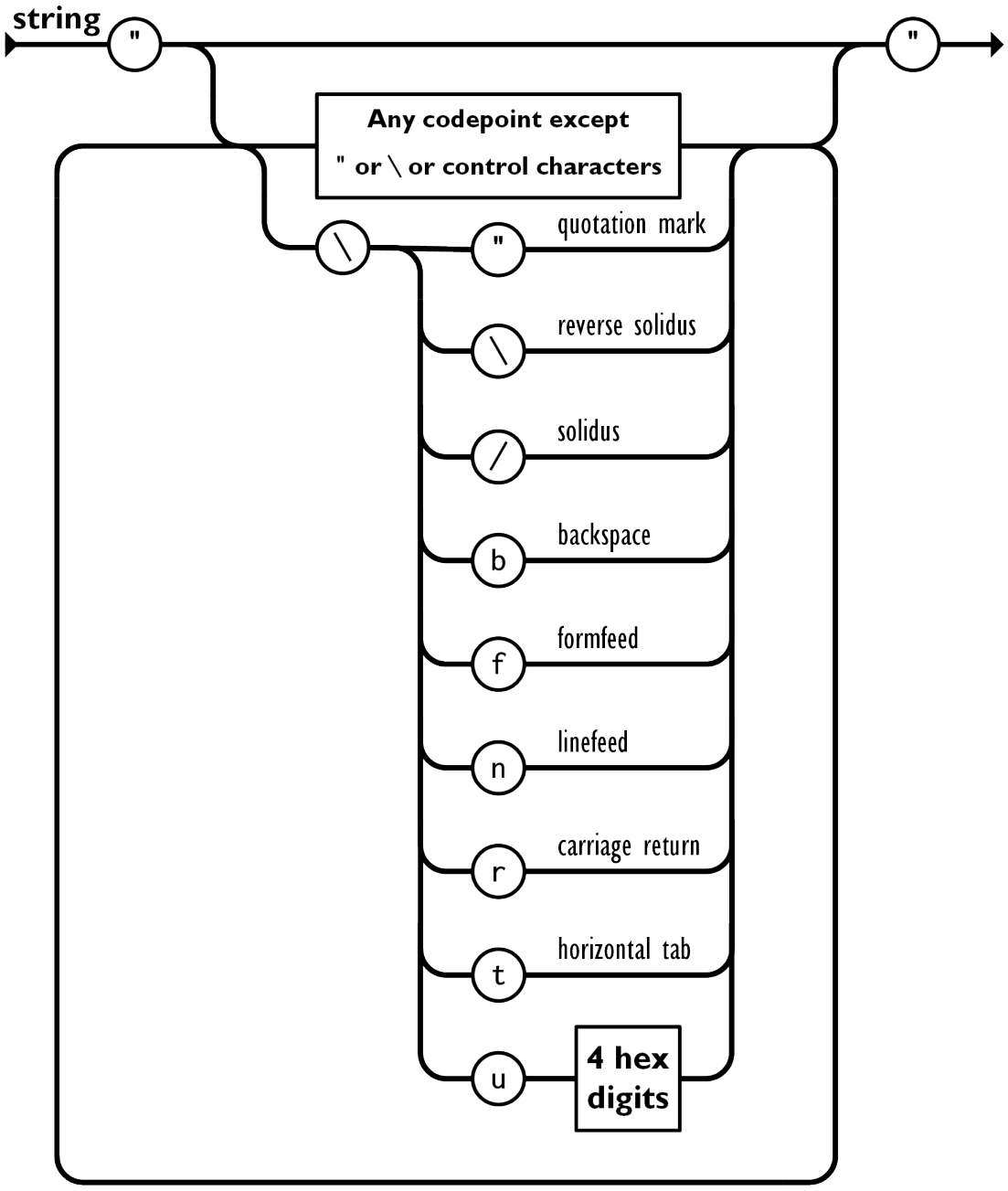
2.2.6 Code on demand

2.2.7 Resource

Daarnaast kan bij REST elk stuk informatie een resource zijn, dit kan van alles zijn zoals documenten, services, collecties van andere resources, afbeeldingen, en dergelijke. In sommige gevallen wordt “Everything as a resource” een zevende principe van REST genoemd.

Er valt af te leiden dat de definitie van een resource zeer abstract is, dit zorgt er voor dat cruciale features van de Web architectuur gebruikt kunnen worden. Eerst en vooral zorgt het voor generaliteit door meerdere bronnen en informatie te omvatten zonder deze te onderscheiden op basis van type of implementatie. Als volgt laat de abstracte definitie een late binding van de referentie aan een representatie toe, hierdoor kan content negotation plaatsvinden op basis van de kenmerken van de request. Tot slot maakt de abstracte definitie het mogelijk voor de auteur om een concept te refereren in plaats van een specifieke representatie van dat concept, hierdoor moeten links naar deze resource niet continue aangepast worden wanneer de representatie veranderd. Deze links, ook wel Resource Identifiers genoemd, verwijzen naar een resource die gebruikt wordt in een interactie tussen componenten.

Om acties uit te voeren op een resource maakt REST gebruik van een representatie om de huidige of bedoelde staat van deze resource vast te leggen en deze uit te wisselen tussen componenten. Een representatie kan beschreven worden als een sequentie van bytes en bevat ook representatie metadata om deze bytes te beschrijven. Daarnaast bestaat een representatie uit verschillende onderdelen, namelijk de data, de metadata die de data beschrijft, en soms ook metadata die de metadata beschrijft, dit meestal met het doel om de integriteit van het bericht te verifiëren. Het dataformaat van een representatie is een media type (N. Freed, 1996).



Figuur 2.5: De structuur van een JSON string.

3. Methodologie

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

Maecenas non massa. Vestibulum pharetra nulla at lorem. Duis quis quam id lacus dapibus interdum. Nulla lorem. Donec ut ante quis dolor bibendum condimentum. Etiam egestas

tortor vitae lacus. Praesent cursus. Mauris bibendum pede at elit. Morbi et felis a lectus interdum facilisis. Sed suscipit gravida turpis. Nulla at lectus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent nonummy luctus nibh. Proin turpis nunc, congue eu, egestas ut, fringilla at, tellus. In hac habitasse platea dictumst.

Vivamus eu tellus sed tellus consequat suscipit. Nam orci orci, malesuada id, gravida nec, ultricies vitae, erat. Donec risus turpis, luctus sit amet, interdum quis, porta sed, ipsum. Suspendisse condimentum, tortor at egestas posuere, neque metus tempor orci, et tincidunt urna nunc a purus. Sed facilisis blandit tellus. Nunc risus sem, suscipit nec, eleifend quis, cursus quis, libero. Curabitur et dolor. Sed vitae sem. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas ante. Duis ullamcorper enim. Donec tristique enim eu leo. Nullam molestie elit eu dolor. Nullam bibendum, turpis vitae tristique gravida, quam sapien tempor lectus, quis pretium tellus purus ac quam. Nulla facilisi.

4. Conclusie

Curabitur nunc magna, posuere eget, venenatis eu, vehicula ac, velit. Aenean ornare, massa a accumsan pulvinar, quam lorem laoreet purus, eu sodales magna risus molestie lorem. Nunc erat velit, hendrerit quis, malesuada ut, aliquam vitae, wisi. Sed posuere. Suspendisse ipsum arcu, scelerisque nec, aliquam eu, molestie tincidunt, justo. Phasellus iaculis. Sed posuere lorem non ipsum. Pellentesque dapibus. Suspendisse quam libero, laoreet a, tincidunt eget, consequat at, est. Nullam ut lectus non enim consequat facilisis. Mauris leo. Quisque pede ligula, auctor vel, pellentesque vel, posuere id, turpis. Cras ipsum sem, cursus et, facilisis ut, tempus euismod, quam. Suspendisse tristique dolor eu orci. Mauris mattis. Aenean semper. Vivamus tortor magna, facilisis id, varius mattis, hendrerit in, justo. Integer purus.

Vivamus adipiscing. Curabitur imperdiet tempus turpis. Vivamus sapien dolor, congue venenatis, euismod eget, porta rhoncus, magna. Proin condimentum pretium enim. Fusce fringilla, libero et venenatis facilisis, eros enim cursus arcu, vitae facilisis odio augue vitae orci. Aliquam varius nibh ut odio. Sed condimentum condimentum nunc. Pellentesque eget massa. Pellentesque quis mauris. Donec ut ligula ac pede pulvinar lobortis. Pellentesque euismod. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent elit. Ut laoreet ornare est. Phasellus gravida vulputate nulla. Donec sit amet arcu ut sem tempor malesuada. Praesent hendrerit augue in urna. Proin enim ante, ornare vel, consequat ut, blandit in, justo. Donec felis elit, dignissim sed, sagittis ut, ullamcorper a, nulla. Aenean pharetra vulputate odio.

Quisque enim. Proin velit neque, tristique eu, eleifend eget, vestibulum nec, lacus. Vivamus odio. Duis odio urna, vehicula in, elementum aliquam, aliquet laoreet, tellus. Sed velit. Sed vel mi ac elit aliquet interdum. Etiam sapien neque, convallis et, aliquet vel, auctor non, arcu. Aliquam suscipit aliquam lectus. Proin tincidunt magna sed wisi. Integer blandit

lacus ut lorem. Sed luctus justo sed enim.

Morbi malesuada hendrerit dui. Nunc mauris leo, dapibus sit amet, vestibulum et, commodo id, est. Pellentesque purus. Pellentesque tristique, nunc ac pulvinar adipiscing, justo eros consequat lectus, sit amet posuere lectus neque vel augue. Cras consectetuer libero ac eros. Ut eget massa. Fusce sit amet enim eleifend sem dictum auctor. In eget risus luctus wisi convallis pulvinar. Vivamus sapien risus, tempor in, viverra in, aliquet pellentesque, eros. Aliquam euismod libero a sem.

Nunc velit augue, scelerisque dignissim, lobortis et, aliquam in, risus. In eu eros. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Curabitur vulputate elit viverra augue. Mauris fringilla, tortor sit amet malesuada mollis, sapien mi dapibus odio, ac imperdiet ligula enim eget nisl. Quisque vitae pede a pede aliquet suscipit. Phasellus tellus pede, viverra vestibulum, gravida id, laoreet in, justo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer commodo luctus lectus. Mauris justo. Duis varius eros. Sed quam. Cras lacus eros, rutrum eget, varius quis, convallis iaculis, velit. Mauris imperdiet, metus at tristique venenatis, purus neque pellentesque mauris, a ultrices elit lacus nec tortor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent malesuada. Nam lacus lectus, auctor sit amet, malesuada vel, elementum eget, metus. Duis neque pede, facilisis eget, egestas elementum, nonummy id, neque.

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introductie

Zonder dat het te weten of te beseffen maakt de internetgebruiker continue gebruik van dataformaten en protocollen. Deze komen in verschillende soorten, XML, JSON, Protocol Buffers, GraphQL, ... Deze data formaten zorgen aan de hand van een achterliggende implementatie in een Application Programming Interface (API) voor dat mensen hun vrienden hun nieuwe Facebook of Instagrampost kunnen zien in een internetbrowser of op een app. APIs maken het meest gebruik van XML en JSON, indien toch een dominerend formaat gekozen moet worden zal dit JSON zijn. JSON is veel sneller om te lezen en te schrijven dan XML.

Echter wil dit niet zeggen dat JSON het snelste data formaat is. De Fashion Society is een moederbedrijf met verschillende kledingsketens onder zich. Om alles vlot te laten verlopen wordt gebruik gemaakt van een centraal beheersysteem voor zowat alles dat nodig is, dit houdt in maar is niet beperkt tot pickorders, webshops, ... Dit interne systeem maakt gebruik van het Façade patroon, dit is een structuur waar er maar één toegangspunt is tot een collectie ojecten met diensten en services. In het Façade patroon van de Fashion Society is de Orchestrator het toegangspunt tot een collectie van meerdere API's, deze krijgt een request binnen voor een bepaalde service of dienst maar weet niet naar welke hij deze request moet doorsturen. Om dit op te lossen wordt gebruik gemaakt van de Discovery API, de Orchestrator zal deze aanroepen en de Discovery zal vervolgens de

juiste Service ophalen en terugsturen naar de Orchestrator zodanig dat deze de originele request kan doorsturen naar de correcte API.

De Fashion Society wordt alsmaar groter en is op zoek naar een manier om de Discovery API performanter te laten wezen om zo meer realtime requests te kunnen afhandelen. De huidige Discovery API is een REST API dat gebruik maakt van JSON. In deze Bachelorproef zal onderzocht worden of Protocol Buffers ge¨ımplementeerd aan de hand van gRPC eventueel een even performant of zelfs performanter alternatief kan zijn voor de huidige implementatie van de Fashion Society dat gebruik maakt van een REST API met JSON. De bijhorende onderzoeksvragen zullen dan ook als volgt zijn:

- Is gRPC met ProtocolBuffers sneller dan een REST API met JSON?
- Is gRPC met ProtocolBuffers efficiënter dan een REST API met JSON?

A.2 State-of-the-art

gRPC is open source Remote Procedure Call (RPC) framework en is in 2015 ontstaan uit zijn voorganger Stubby, deze was een single general-purpose RPC infrastructuur (gRPC Authors, g.d.). Deze technologie laat het toe voor een programma om procedures te starten op andere computers in, eventueel, verschillende adresruimten aan de hand van een smal comminucatiekanaal (Nelson, 1981). gRPC zelf is geen dataformaat zoals JSON maar kan gebruik maken van verschillende data formaten, standaard is dit Protocol buffers, ook wel Protobufs genoemd.

JSON en Protocol Buffers hebben zowel gelijkenissen als grote verschillen, beide zijn een language-neutral dataformaat zo blijkt uit de documentatie van Protocol Buffers (Google, g.d.) en JSON (ECMA, 2017). Het grootste, visuele verschil voor de gebruiker is te vinden in de structuur. JSON is een collectie van naam/value paren of een geordende lijst van waarden, daarintegen maken Protobufs gebruik van een soort model, er moet maar eenmaal gedefinieerd worden hoe de data gestructureerd zal zijn, daarna kan gebruik gemaakt worden van gegenereerde code om gemakkelijk data van en naar een variëteit van data streams te lezen en schrijven. Dit kan allemaal geprogrammeerd worden in heel wat verschillende programmeertalen.

gRPC wordt reeds gebruikt door verschillende grote bedrijven zoals Cisco en Netflix. Cisco gebruikt gRPC als het ideale, uniforme transportprotocol voor modelgestuurde configuratie en telemetrie. Dit komt dankzij de ondersteuning voor hoogwaardige bidirectionele streaming, op TLS gebaseerde beveiliging en het brede scala aan programmeertalen. Netflix koos dan weer voor gRPC omdat ze belang hechtte aan de architectonische kennis in de IDL (proto) dat een onderdeel is van gRPC en aan de, van deze proto-afgeleide, codegeneratie. Daarnaast speelde ook de cross-language compatibility en codegeneratie in gRPC een belangrijke rol bij de keuze voor gRPC bij Netflix (Foundation, 2018).

Eerder werd er nog geen officieel onderzoek uitgevoerd naar dit onderwerp, echter zijn wel online artikels te vinden die gRPC met REST gaan vergelijken aan de hand van een benchmark. Eén zo een onderzoek is uitgevoerd door Fernando (2019). Dit onderzoek was

een benchmark tussen gRPC met Protocol Buffers en REST met JSON concludeerd dat gRPC sneller was dan REST behalve bij het streamen van data, hier is gRPC iets trager als REST.

Het onderzoek dat in deze Bachelorproef zal uitgevoerd worden is gebaseerd op dat van Ruwan Fernando, hiermee wordt bedoeld dat deze proof-of-concept geprogrammeerd zal worden in C# en dat de verschillende impelementaties met elkaar vergeleken zullen worden aan de hand van een benchmark. Het verschil met Fernando R. zijn onderzoek kan men vinden in de verwerking van de data, in dit onderzoek zal de data uit een aanhangende databank gehaald worden. Alsook zal in dit onderzoek een tienvoud van het aantal iteraties doen.

A.3 Methodologie

Het onderzoek zal gevoerd worden aan de hand van simulaties en experimenten in een Proof-of-Content (PoC). Er zullen 2 identieke APIs opgesteld worden in C#, een .NET Core REST API voor JSON, en .NET Core API voor gRPC. Aan de hand van een benchmark tool, dewelke nog niet specifiek gekozen is, zullen er 2 verschillende categorieën aan GET en POST calls uitgevoerd worden. Big payload calls, hier zal substantief meer data opgehaald en verzonden worden dan bij de Small payload calls. Daarnaast zal het experiment meerdere malen uitgevoerd worden met 1000 en 2000 iteraties, dit om inaccurate metingen van kleine uitvoertijden te verminderen en een duidelijke conclusie te kunnen vormen. Eens de resultaten van de experimenten verzameld zijn zullen we deze vergelijken en conclusies trekken.

A.4 Verwachte resultaten

Bij de grote payloads wordt een duidelijk verschil in voordeel van gRPC en Protocol Buffers verwacht, echter wordt er wel ook verwacht dat REST API met JSON in de kleine payloads nog degelijk zal presteren en misschien zelf nog licht overwegend beter zal zijn dan gRPC en Protocol Buffers.

A.5 Verwachte conclusies

Dit onderzoek moet doen blijken of gRPC en Protocol Buffers sneller en efficiënter zijn dan een REST API met JSON. In de PoC zal duidelijk worden dat gRPC en Protocol Buffers een sneller en efficiënter alternatief zijn voor JSON. Dit resultaat zou eventueel verklaard kunnen worden doordat JSON een ouder dataformaat is en dus niet geoptimaliseerd is voor de huidige technologieën terwijl gRPC zeer recent ontworpen is om optimaal met de huidige technologieën om te gaan.

B. Copyright Notice Ecma International

The limited permissions granted above are perpetual and will not be revoked by Ecma International or its successors or assigns.

This document and the information contained herein is provided on an "AS IS"basis and ECMA INTERNATIONAL DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

Bibliografie

ECMA. (2017). *The JSON Data Interchange Syntax*. ECMA International. Verkregen 16 december 2020, van https://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-404%201st%20edition%20October%202013.pdf

Fernando, R. (2019, april 3). *Evaluating Performance of REST vs. gRPC*. https://medium.com/@EmperorRXF/evaluating-performance-of-rest-vs-grpc-1b8bdf0b22da

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (proefschrift). University of California, Irvine.

Foundation, C. N. C. (2018, december 4). *Netflix: Increasing developer productivity and defeating the thundering herd with gRPC*. https://www.cncf.io/case-studies/netflix/

Google. (g.d.). *Protocol Buffers*. Google. https://developers.google.com/protocol-buffers

gRPC Authors. (g.d.). *gRPC, A high-performance, open source universal RPC framework*. https://grpc.io/

N. Freed, N. B. (1996). *Multipurpose Internet Mail Extensenions (MIME) Part One: Format of Internet Message Bodies*. Verkregen 14 maart 2021, van https://tools.ietf.org/html/rfc2045

Nelson, B. J. (1981). *Remote Procedure Call* (proefschrift). Carnegie-Mellon University. https://ia801900.us.archive.org/22/items/bitsavers_xeroxparctoteProcedureCall_14151614/CSL-81-9_Remote_Procedure_Call.pdf