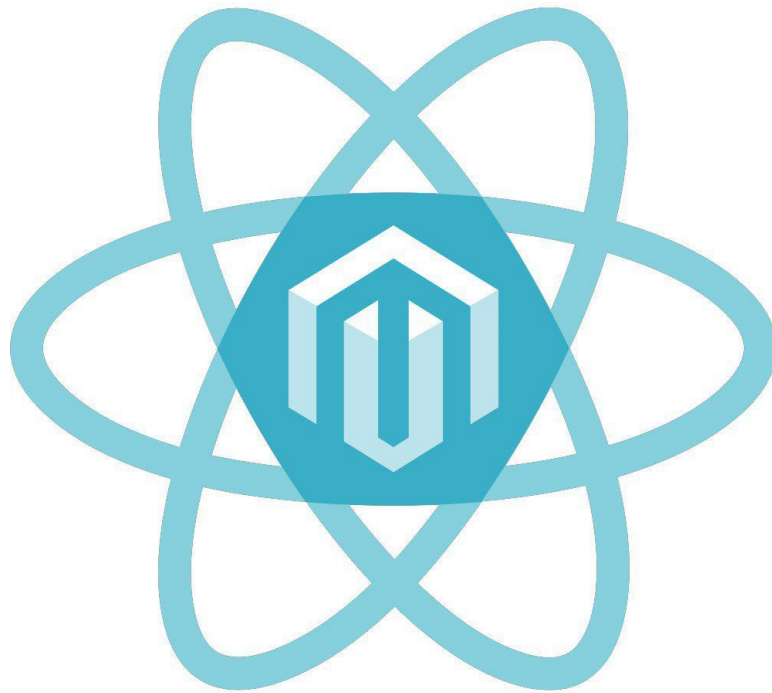




Headless Magento 2

React.JS als frontend

Rick Kuilman

29-10-2017

Headless Magento 2

ReactJS als frontend

Auteur

Rick Kuilman

Studentnummer: 291407

Datum afdruk

30-10-2017

Opleiding

HBO-ICT, major webdevelopment

Instituut voor Communicatie, Media & IT

Hanzehogeschool Groningen

Bedrijfsbegeleider

Jeroen Boersma

Srcode

Meester P.J. Troelstraweg 149

8919 AA Leeuwarden

Docentbegeleider & eerste beoordelaar

Tjerk Harkema

Tweede beoordelaar

Jos Bos

Bron van afbeelding op de omslag

Deze afbeelding is zelfgemaakt en is opgebouwd uit het logo van ReactJS

(<https://facebook.github.io/react/img/logo.svg>) en het logo van Magento

(https://static.magento.com/sites/all/themes/mag_redesign/images/magento-logo.svg).

Dit afstudeerrapport is geschreven onder verantwoordelijkheid van de Hanzehogeschool Groningen. Het copyright berust bij de auteur. Zowel de Hanzehogeschool Groningen als de auteur verklaren, dat zij eventuele gegevens van derden die voor dit afstudeeronderzoek zijn gebruikt en die door deze derden als vertrouwelijk zijn aangemerkt, als zodanig zullen behandelen.

Voorwoord

Dit document leeft de richtlijnen na zoals deze beschreven staan in '*Beoordelingsformulier PID maart 2017*', '*Stappenplan voor afstudeerproject*' (versie 8 februari 2015), '*Schrijfwijzer HBO-ICT*', '*Handleiding Afstudeerproject 2016-2017, semester 2*' die goedgekeurd is door de examencommissie in februari 2017.

Deze documenten zijn terug te vinden op Blackboard, onder de *course* "Afstuderen vol- en deeltijd".

Rick Kuilman

Groningen, 29-10-2017

Samenvatting

Srcode is een klein Fries bedrijf dat zich grotendeel bezighoudt met de technische implementatie van uiteenlopende webshops. Het softwarepakket dat Srcode gebruikt voor het maken van deze webshops is Magento, een bekend e-commerce platform gemaakt in PHP. Magento is zeer populair, zowel bij ontwikkelaars als webshopbeheerders. Dit komt mede door de brede beschikbaarheid aan *modules* en de abstracte manier waarop deze nauw binnen het systeem zijn verweven.

Deze abstracte opzet en het feit dat alles intern nauw verweven is heeft echter een nadeel. Magento is hierdoor een relatief zwaar softwarepakket om te draaien op een server. De hoeveelheid aan functies van het systeem zijn immers heel groot en moeten allemaal onderling werken met de additionele software die geïnstalleerd staat op het systeem.

Srcode ondervindt momenteel twee problemen. Allereerst worden webshops traag wanneer er veel extensies geïnstalleerd staan; Magento moet deze immers allemaal inladen voordat de pagina geladen kan worden. Ten tweede is Srcode van mening dat de frontend en de backend van elkaar gescheiden zouden moeten zijn, zodat de keuze vrij is aan de ontwikkelaar om een bepaald systeem te gebruiken.

Vanuit deze problemen zijn er twee vragen opgeworpen: **(I) is het mogelijk om een andere frontend te gebruiken voor Magento2?** **(II) En in hoeverre heeft het vervangen van de frontend impact op de laadtijd van de webshop?** Dit proces heet het “*headless*” maken van software, oftewel: *headless Magento 2*.

Er is een proof of concept gemaakt om te bepalen of het mogelijk is om een andere frontend te gebruiken dan die van Magento. Het resultaat is dat dit inderdaad mogelijk is, in dit geval met behulp van de programmeertaal ReactJS (I).

Om de impact op de laadtijden te bepalen is een experiment opgesteld waarbij virtuele gebruikers de laadtijden hebben gemeten van beide systemen, waarna de resultaten met elkaar zijn vergeleken. Uit dat experiment is naar voren gekomen dat de Magento 2 frontend sneller is met het laden van pagina's dan de ReactJS frontend (II).

Uitgezocht moet worden of dit aan de Magento 2 API ligt, of dat de bottleneck ergens anders is. Een van de aanbevelingen luidt dan ook om op zoek te gaan naar deze bottleneck om de ReactJS frontend wellicht alsnog sneller te kunnen maken.

Inhoudsopgave

Voorwoord	5
Samenvatting.....	7
1 Inleiding.....	10
2 Over Srcode	13
3 Probleem	14
3.1 Probleemdefinitie	14
3.2 Probleempositionering.....	16
4 Aanpak	18
4.1 Beroepsproduct	18
4.2 Kwaliteitseisen & requirements	18
4.3 Methode, technieken en tools.....	20
4.3.1 Agile Scrum ontwikkelmethodiek.....	21
4.3.2 Loadimpact	22
4.3.3 Siege	23
4.3.4 RESTful Web services.....	24
4.3.5 Postman.....	24
5 Theorie	25
5.1 Agile Scrum.....	25
5.1.1 Aanpassingen Agile Scrum ten behoeve van dit project.....	26
5.1.2 Ontwikkelproces project binnen Srcode	27
5.2 Middleware.....	28
5.3 Frontend framework.....	29
5.3.1 AngularJS.....	30
5.3.2 ReactJS.....	30
5.3.3 Vergelijking.....	31
6 Beroepsproduct	34
6.1 Componenten binnen het product.....	34
6.2 Component “Magento 2 Webshop”	36
6.3 Component “Lumen middleware”	37
6.3.1 De communicatie tussen Lumen en ReactJS.....	37
6.3.2 De communicatie tussen Lumen en Magento 2.....	39
6.4 Component “ReactJS Applicatie”	40

6.4.1	Webpack en Babel.....	41
6.4.2	Lodash.....	42
6.4.3	Axios.....	44
7	Praktijkgericht onderzoek.....	45
7.1	Inleiding.....	45
7.2	Methoden.....	47
7.3	Resultaten	48
7.4	Discussie en analyse	52
7.5	Conclusies en aanbevelingen	52
8	Conclusie	54
9	Advies & aanbevelingen	56
9.1	Advies uitrollen van het beroepsproduct.....	56
9.2	Aanbeveling voor afronden van het beroepsproduct.....	57
9.3	Aanbeveling voortzetting van het beroepsproduct.....	58
9.4	Advies gebruik andere frontend frameworks	59
10	Project	60
10.1	Project risico's	60
10.2	Betrokken partijen.....	60
10.3	Competenties	61
10.4	Planning en contactmomenten.....	61
11	Bibliografie	63
12	Bijlagen	65

1 Inleiding

In dit hoofdstuk zal worden ingegaan op de organisatie, de context waarin het afstudeerproject plaatsvindt, het probleem dat er bij het bedrijf is, het belang van een oplossing en de beschrijving van de afstudeeropdracht. Tevens zal de structuur van dit afstudeerrapport worden beschreven. Het doel hiervan is om geïntroduceerd te raken tot het onderwerp en vooruit te blikken op de verschillende onderdelen van het rapport.

Srcode is een bedrijf dat zich grotendeels bezighoudt met het maken van webshops. Het bedrijf heeft meerdere klanten van verschillende grootte, maar richt zich met name op het klein tot middelgroot bedrijf. Er wordt verder ingegaan op het bedrijf in hoofdstuk 2 (“Over Srcode”) om een beter inzicht te geven in de context.

De webshops die Srcode in haar beheer heeft worden gebouwd met behulp van het softwarepakket *Magento*. Magento is een softwarepakket dat speciaal gemaakt is voor het maken van webshops en is een grote partij in de wereld van het digitale winkelen. Dit is mede te danken aan de mate van modulariteit waarmee het systeem gebouwd is. Het installeren en ontwikkelen van additionele software voor Magento is hierdoor toegankelijk gemaakt.

Deze flexibiliteit heeft echter een keerzijde, namelijk dat Magento de server zwaar belast. De hoeveelheid aan functies van het systeem zijn immers heel groot en moeten allemaal onderling werken met de additionele software die geïnstalleerd staat op het systeem.

Srcode ondervindt momenteel het probleem dat de voorkant van de website (datgene wat de bezoekers van de webshop te zien krijgen) trager wordt. Dit is een groot probleem voor de klanten van Srcode, omdat bezoekers van de trage webshop de site eerder zullen verlaten. Hierdoor lopen klanten van Srcode potentiële inkomsten mis. Het is dus van belang om dit probleem op te lossen.

Bij een poging tot het oplossen van dit probleem is echter een ander (samenhangend) probleem ontdekt. Dit heeft er mee te maken dat Magento intern een stuk minder modulair is dan de rest van het pakket. Daardoor zijn aanpassingen aan deze interne onderdelen moeilijk door te voeren. Een van de onderdelen is de gehele frontend van de website. Het is dus erg lastig om veranderingen aan het beheerpaneel te maken zonder dat dit invloed heeft op de frontend en andersom. Een aanpassing in de code van het laden van een product in

het beheerpaneel zou dus potentieel problemen kunnen veroorzaken aan de frontend van de website. Een uitgebreidere uitleg van het probleem komt in hoofdstuk 3 (“Probleem”) terug.

De combinatie van deze twee problemen heeft bij Srcode een interessante vraag opgewekt: **is het mogelijk om een andere frontend te maken, zodat deze volledig los staat van het beheerpaneel van Magento?**

Dit zou potentieel beide problemen kunnen oplossen. Allereerst zou door het gebruik van een andere voorkant de webshop sneller moeten gaan werken, immers hoeft de losse voorkant niet rechtstreeks te werken met het trage Magento systeem. Daarnaast zouden wijzigingen in het beheerderspaneel (door bijvoorbeeld een software-update) minder of geen effect moeten hebben op de voorkant van de webshop.

Uitgezocht gaat worden in hoeverre het maken van een losstaande voorkant mogelijk is en wat het verschil in laadsnelheid van de pagina's is. Omdat Srcode uiteraard wil dat de voorkant sneller gaat worden luidt de doelstelling van het project concreet: **“In de gewenste situatie zullen de laadsnelheden van de geselecteerde pagina's ten minste sneller zijn dan de laadtijden van de huidige situatie.”** Deze requirement is ook terug te vinden in hoofdstuk 4.2 (“Kwaliteitseisen & requirements”).

Doormiddel van het maken van een werkend prototype van de nieuwe voorkant kunnen er metingen worden verricht naar de laadsnelheid van het systeem, waarna deze vergeleken kunnen worden met de huidige laadsnelheden van Magento 2. Een experiment is een onderzoeksmethode die aansluit bij het onderzoeken van oorzaak-gevolgrelaties met kwantitatieve metingen. De beweegredenen voor het doen van een experiment wordt toegelicht in hoofdstuk 7 (“Praktijkgericht onderzoek”). De metingen zullen worden verricht met behulp van software die bedoeld is voor het meten van de laadsnelheid doormiddel van het simuleren van bezoekers.

De **geselecteerde pagina's** die zullen worden meegenomen in het experiment zijn de categoriepagina's en de productpagina's. De rest van de pagina's staan buiten de scope van de opdracht. De **laadsnelheden** zijn gemiddelden, waarbij de ruwe data eerst verwerkt wordt om onjuiste metingen te filteren. Specifiekere informatie over de requirements en afbakening van de opdracht is te vinden in hoofdstuk 4 (“Aanpak”).

Doel van het rapport is verslag doen van het gehele ontwerpproces. Allereerst zal de het probleem uiteengezet worden. Daarna zal de aanpak uitgelegd worden, waarna de theorie en het onderzoek aan bod komen. Ook de manier van (projectmatig) werken zal worden toegelicht, alsmede de inventarisatie van risico's, randvoorwaarden en andere projectmatige kwesties. Zie daarvoor hoofdstuk 10 ("Project").

2 Over Srcode

Dit hoofdstuk is bedoeld om het probleem dat beschreven gaat worden in het volgende hoofdstuk beter te kunnen plaatsen in de organisatorische context van het bedrijf.

Srcode is een bedrijf, gevestigd in Leeuwarden, dat zich grotendeels bezighoudt met het ontwikkelen en onderhouden van webshops (en gebruikte software). Klanten van het bedrijf zijn met name midden- en kleinbedrijven met uiteenlopende producten of diensten. Voorbeelden hiervan zijn *Kuipers Nautic*, *Outlaw Racing* en *slimtelefoonboek.nl*. Srcode werkt veel samen met andere organisaties en heeft daarom, naast haar eigen klanten, ook veel klanten van andere organisaties.

Een groot deel van de klanten van Srcode heeft betrekking tot de tak van de *e-commerce*. Dit brede begrip omvat alle manieren waarop handel gedreven kan worden via een computernetwerk, waaronder webshops. Srcode is specialist op het gebied van het bouwen, onderhouden en uitbreiden van webshops.

De eigenaar van het bedrijf is Jeroen Boersma. Hij heeft veel ervaring met Magento, een *open-source* softwarepakket voor het maken van webshops. *Open-source* houdt in dat de broncode van het programma voor iedereen in te zien is, iedereen vrij is om aanpassingen te maken in deze software en deze opnieuw uit te geven (Laurent, 2004). Jeroen is groot voorstander van deze vorm van bijdragen aan computersoftware. De naam van zijn bedrijf is hiervan dan ook afgeleid; “Srcode” is een woordspeling op het woord “Source code”.

Het is belangrijk om te weten dat de klanten van Srcode de webshopbeheerders zijn; Srcode is dus zelf geen webshopeigenaar. Wanneer er in dit document gesproken wordt over de “klant”, wordt hier bedoeld de klanten van Srcode (dus de webshopeigenaren). Wanneer er gesproken wordt over de “eindklant”, wordt hier bedoeld de klanten van de webshopeigenaren. Dit is een belangrijk verschil, omdat die verschillende klanten verschillende belangen hebben.

3 Probleem

In dit hoofdstuk zal het probleem dat Srcode ondervindt worden uitgelegd en toegelicht. Dit zal worden gedaan aan de hand van het definiëren van het probleem (binnen de context waarin het probleem plaatsvindt), wat de probleem positionering is en geeft in introductie in de oorzaak van het probleem.

3.1 Probleemdefinitie

Magento, het softwarepakket waarmee Srcode dagelijks werkt, is populair bij zowel ontwikkelaars als webshopbeheerders. Dit is mede te danken aan de brede beschikbaarheid aan *extensies* en de abstracte manier waarop deze nauw binnen het systeem worden verweven. Extensies zijn stukjes software die gebouwd worden bovenop het platform, zonder het platform zelf aan te passen. Dit is mogelijk omdat het systeem heel modulair is opgebouwd. Hierdoor kunnen onderdelen los van elkaar werken, zonder dat het een onderling probleem oplevert.

Dit geldt voor de extensies, maar minder voor Magento intern. Zo zijn er veel onderdelen van het softwarepakket die leunen op andere interne onderdelen. Wanneer er aanpassingen gedaan moeten worden binnen een intern onderdeel kan dit problemen opleveren bij andere interne onderdelen. De interne onderdelen zijn op dat moment *tightly coupled* te noemen, wat inhoudt dat ze afhankelijk van elkaar zijn om correct te werken. Zo zou bijvoorbeeld de zoekfunctionaliteit heel afhankelijk kunnen zijn van de manier waarop Magento haar productstructuur heeft opgebouwd. Deze manier van bouwen is niet wenselijk. Bijvoorbeeld: wanneer de productstructuur aangepast zou worden, kan het zomaar zijn dat de zoekfunctie op de website plotseling niet meer werkt.

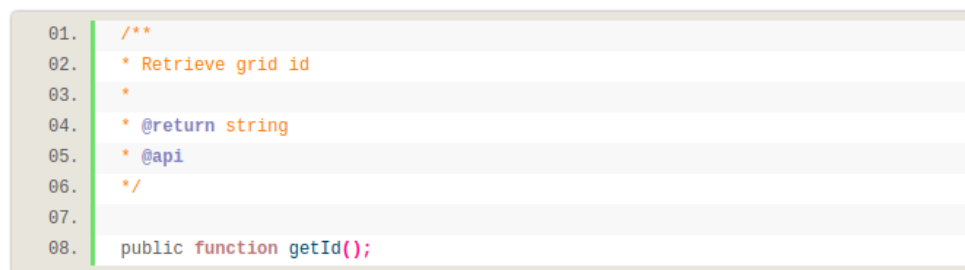
Een ander onderdeel dat *tightly coupled* is, is de *frontend* van de webshop. De frontend is datgene wat de eindklant ziet, denk hierbij aan zaken als de productpagina of de afrekenpagina. Het is bijna onmogelijk om code in de *backend* te verwijderen, zonder dat de frontend hier last van heeft. De backend is het gedeelte van de website dat de eindklant niet te zien krijgt, zoals het administratiepaneel of het voorraadsysteem. Tot voor kort was dit geen probleem voor Srcode, omdat ze vooral gebruik maken van de interne onderdelen van Magento.

Het probleem voor Srcode is ontstaan toen de frontend van de webshop steeds trager werd door het toepassen van externe modules. Dusdanig traag dat er veel

extra rekenkracht toegevoegd moest worden aan de servers om de laadsnelheid van de pagina's voor de eindklant enigszins op peil te houden.

Vanuit dit probleem is er een interessante vraag voortgekomen: is het mogelijk om de frontend van de webshop volledig los te koppelen van de backend? Magento 2 heeft namelijk functionaliteiten aan boord om data uit de backend te kunnen halen, zonder daarvoor de frontend aan te spreken. Daarmee lijkt het mogelijk om de frontend direct los te koppelen van de backend. Echter zit Magento 2 nog volop in ontwikkeling, waardoor bepaalde functies nog niet geïmplementeerd zijn.

Omdat Magento 2 een grootschalig project is en ontwikkelaars aan de slag willen met deze datauitwisseling, heeft Magento orde in de chaos aangebracht door functionaliteiten die niet meer worden gewijzigd te markeren. Dit wordt gedaan door de tekst "@api" te zetten in de docblocks van een functie. (Magento, 2016) De docblock is een stuk commentaar dat geschreven kan worden bij een functie. Zoals te zien is in Codevoorbeeld 1, wordt er middels het toevoegen van "@api" aangegeven dat deze functionaliteit publiekelijk beschikbaar is en stabiel is voor ontwikkeling.



```
01.  /**
02.   * Retrieve grid id
03.   *
04.   * @return string
05.   * @api
06.   */
07.
08.  public function getId();
```

Codevoorbeeld 1 Het markeren van een publieke API functie die stabiel is bevonden door Magento aan de hand van de tekst "@api"

De tegenhanger van “@api” is “@internal”. Deze geeft aan dat de functionaliteit nog kan veranderen en dat er wordt afgeraden om deze functies te gebruiken. In Codevoorbeeld 2 is te zien hoe dit wordt weergegeven.

```
01.  /**
02.   * Unescapes a character that was found in a double-quoted string.
03.   *
04.   * @param string $value An escaped character
05.   *
06.   * @return string The unescaped character
07.   *
08.   * @internal This method is public to be usable as callback. It should not
09.   *           be used in user code. Should be changed in 3.0.
10.   */
11.  public function unescapeCharacter($value)
12.  {
13.      switch ($value[1]) {
14.          ...
```

Codevoorbeeld 2 Het markeren van publieke API functie waarvoor wordt afgeraden om deze te gebruiken in productie aan de hand van de tekst “@internal”

Door deze onzekerheid kan niet direct bepaald worden of alle benodigde functionaliteiten aanwezig zijn om de frontend te scheiden van de backend. De vraag of het mogelijk is om de webshop volledig los te koppelen van de backend blijft dus bestaan: er kan niet direct worden aangenomen dat het mogelijk is.

Er zal dus gekeken moeten worden of het mogelijk is om *Magento* zogenaamd *Headless* te maken. Srcode wil niet alleen uitgezocht hebben of dit mogelijk is, maar ook in hoeverre het gebruik van een ander frontendsysteem de laadtijden van pagina’s beïnvloed bij een bepaalde hoeveelheid bezoekers.

3.2 Probleempositionering

Een frontend van een webshop die gebruik maakt van het softwarepakket *Magento* gebruikt is zeer afhankelijk van de backend, omdat deze nauw met elkaar verweven zijn. De frontend leunt als het ware op de backend en de backend gaat er vanuit dat de frontend gebruikt wordt van hetzelfde pakket.

Hoewel Srcode jaren heeft gewerkt met dit systeem, ondervindt de organisatie een probleem met het schalen van de webshop. Zowel op het gebied van snelheid van de webshop, als de snelheid van het ontwikkelproces waarmee de webshop gemaakt wordt. Wanneer er nu iets aangepast moet worden in de code van de

backen van de webshop, zal er altijd moeten worden nagedacht over de impact die deze wijzigingen met zich meebrengen op de frontend. Het idee dat wijzigingen die aan de backend gedaan worden directe invloed kunnen uitoefenen op de gebruikerservaring van de eindklant is onwenselijk.

Er is nog een factor die voor Srcode meespeelt. In de huidige situatie is het namelijk zo dat wanneer een webshop Magento gebruikt als beheersysteem, de beheerder verplicht wordt om de frontend te gebruiken van Magento. Er is geen keuzevrijheid in het type frontend dat er gebruikt kan worden, waardoor Magento met haar monopolypositie alle vrijheid heeft bepaalde functionaliteiten te wijzigen of te verwijderen. Deze situatie is eveneens onwenselijk in het belang van Srcode en haar klanten.

Samenvattend wil Srcode twee zaken uitgezocht hebben door de opdrachtnemer:

- 1. Is het mogelijk om een andere frontend te gebruiken voor Magento?**
- 2. In hoeverre heeft het vervangen van een frontend impact op de laadsnelheid van bepaalde pagina's van de webshop?**

In het hier opvolgende hoofdstuk zal worden uitgelegd wat de aanpak is om deze vraag te beantwoorden en hoe de opdracht uitgevoerd zal worden. Ook zullen daar de wensen van Srcode worden toegelicht en zullen de requirements worden opgesteld.

4 Aanpak

In dit hoofdstuk zal ingaan op de aanpak die moet leiden tot het oplossen van het probleem, zoals deze beschreven is in hoofdstuk 3 (“Probleem”). Er zal tevens uitgelegd worden hoe dit bijdraagt aan het halen van de doelstelling, zoals beschreven in hoofdstuk 1 (“Inleiding”).

4.1 Beroepsproduct

Om te bepalen of het mogelijk is om de frontend van de backend te scheiden is er een *proof of concept* gemaakt. Een proof of concept is een methode om aan te tonen of een bepaalde oplossing voor een probleem in de praktijk te realiseren is. (InvestorWords, 2017) Als het ware is het een prototype van het uiteindelijke potentiële product.

Een dergelijk beroepsproduct draagt bij aan het halen van de doelstelling, omdat dit product de mogelijkheid biedt om te bepalen of het maken van een eigen frontend voor Magento mogelijk is. Dit beantwoordt de eerste van de twee vragen, zoals te vinden in hoofdstuk 3.2 (“Probleempositionering”).

Het hoofdzakelijke doel van deze proof of concept is om te bepalen of het mogelijk is om een eigen frontend te implementeren voor Magento. Een ander belangrijk aspect dat hier uit voortvloeit, is dat het proof of concept vervolgens ingezet kan worden voor het experiment naar de laadsnelheden van bepaalde pagina’s. Hierop word dieper ingegaan in hoofdstuk 7 (“Praktijkgericht onderzoek”). Om aan de doelstelling van de opdracht te voldoen, zal immers een snelheidsmeting gedaan moeten kunnen worden. Dit product is dus zeker nodig om de doelstelling van de opdracht te halen en het oplossen van het probleem te realiseren.

In overleg met de opdrachtgever zijn kwaliteitseisen en belangrijkste requirements opgesteld voor de proof of concept.

4.2 Kwaliteitseisen & requirements

Voor de proof of concept zijn samen met de opdrachtgever kwaliteitseisen en requirements opgesteld. Deze zijn hieronder puntsgewijs opgesomd en toegelicht.

De requirements zijn opgesteld aan de hand van de *MoSCoW-methode*. Deze techniek wordt gebruikt om de eisen aan het resultaat van een project in te delen en te prioriteren. De letters M, S, C en W in het woord “MoSCoW” zijn afkorting

voor de verschillende categorieën met ieder een eigen betekenis. De termen zijn Engelstalig, maar zijn vrij vertaald voor de uitleg. De letters staan voor (Vliet, 2008):

- M – Must Have (*moet hebben*). Dit requirement moeten in het eindresultaat terugkomen en is kritiek voor de werking van het project. Zonder dit requirement is het product onbruikbaar.
- S – Should Have (*zou moeten hebben*). Dit requirement wordt als zeer gewenst beschouwd, maar is niet kritiek voor de werking van het product. Het product is zonder deze eisen bruikbaar.
- C – Could Have (*zou kunnen hebben*). Dit requirement komt alleen aan bod als er genoeg tijd is om het te realiseren.
- W – Won't Have (*zal niet hebben*) of Woud Have (*zou hebben*). Dit requirement is niet zo zeer gericht op het huidige project, maar is bedoeld als requirement voor een toekomstig (vervolg-)project.

Met deze methode in het achterhoofd heeft er een brainstormsessie plaatsgevonden tussen de belanghebbende (de stakeholders) van dit project. In dit geval is dat een sessie geweest tussen de opdrachtnemer en de heer Boersma. Hij is gevraagd voor deze sessie omdat hij op technisch en organisatorisch vlak het meest betrokken is bij het project. Daarbij opereert hij zowel op tactisch als op strategisch niveau, wat hem de aangewezen persoon maakt om een goede invulling aan het project te kunnen geven.

Zonder aan de heer Boersma de techniek van MoSCoW uitgelegd te hebben, zijn alle ideeën die beide partijen hebben bij dit product vastgelegd. Vervolgens is hier een selectie in aangebracht waar het product minimaal aan moet voldoen, en welke wensen minder belangrijk zijn.

De onderdelen die belangrijk zijn bevonden, zijn vervolgens gebundeld tot een eisenpakket. Aan de hand van de MoSCoW methode zijn deze eisen omgezet naar de *requirements* van dit project.

Geordend van “Must have” tot “Won’t have”, zijn daar de volgende requirements uit voort gekomen:

1. Must have – De proof of concept moet een (versimpelde) weergave van de categoriepagina’s en de productpagina’s te bevatten.
 - a. De **categoriepagina’s** zijn de pagina’s waarop alle producten van een bepaalde categorie te zien zijn. Bij elk product zal de naam van het product, de afbeelding van het product en de prijs van het product weergegeven moeten worden.
 - b. De **productpagina’s** zijn de pagina’s waarop een geselecteerd product weergegeven moet worden. Op deze pagina’s moeten de naam, afbeelding, beschrijving en de prijs van het product weergegeven moeten worden.
2. Must have – Deze proof of concept moet niet langer meer gegenereerde (frontend) elementen van Magento te gebruiken. Immers is het doel van de opdracht om niet langer afhankelijk te zijn van Magento.
3. Must have – De webshop MOET 16.000 (zestienduizend) artikelen bevatten, verveeld over 300 categorieën.
4. Must have – Er moeten minimaal 60 bezoekers tegelijkertijd op de website terecht kunnen.
5. Must have – De proof of concept moet uitsluitend met versie 2.x van Magento moeten werken. Hierbij mag de opdrachtnemer de specifieke versie kiezen.
6. Must have – De te maken proof of concept moet kunnen worden ingezet voor iedere Magento 2.x webshop.
7. Should have – De laadsnelheid van de pagina’s van de proof of concept zou gemiddeld sneller moeten zijn dan de laadsnelheid van de pagina’s in de huidige situatie.
8. Could have – De mogelijkheid om producten in de winkelwagen toe te voegen
9. Could have – De mogelijkheid om producten en/of categorieën te zoeken via een zoekbalk
10. Won’t have – De proof of concept zal niet gebruikt worden op een live server van een klant en zal ook geen data gebruiken van deze live server.

4.3 Methode, technieken en tools

In deze paragraaf wordt uiteengezet welke methodes, technieken en *tools* worden gebruikt om het beroepsproduct te vervaardigen. Een aantal van deze onderdelen

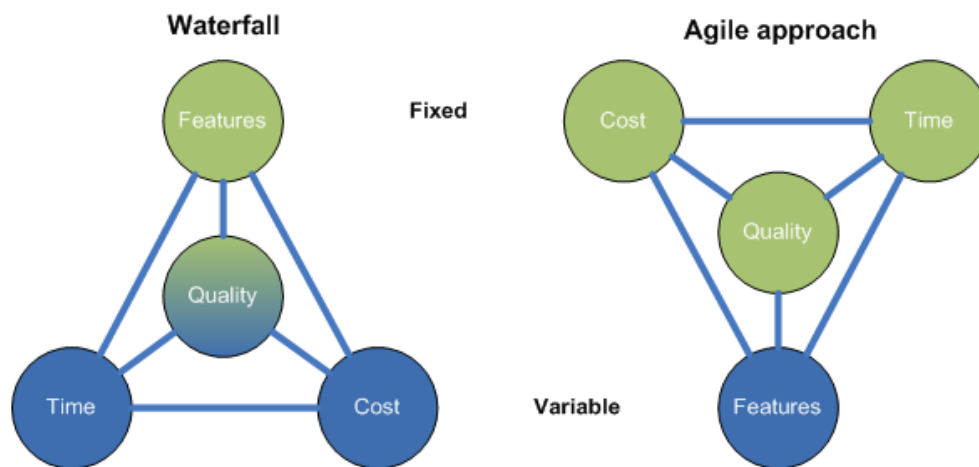
komen voort uit logica, anderen zijn voortgeschreven door de organisatie of zijn gebaseerd op een methode uit literatuur. Bij ieder onderdeel zal worden toegelicht waarom deze nodig is om het beroepsproduct te maken.

4.3.1 Agile Scrum ontwikkelmethodiek

Binnen Srcode wordt de ontwikkelmethodiek Agile aangehouden, in combinatie met Scrum. Vanuit Srcode is de vraag opgeworpen of dezelfde ontwikkelmethodiek ook voor dit project inzetbaar is.

Scrum is uitermate geschikt als er nog geen strakke planning ligt, de features van een project nog niet volledig (of duidelijk) zijn uitgewerkt en er binnen korte tijd een (gedeelte van het project) af moet zijn. Deze situatie is wenselijk binnen deze opdracht, omdat er snel duidelijkheid moet zijn of het mogelijk is om de frontend los te koppelen van de backend. Er kan daarna gewisseld worden van traject. Het grote nadeel van Scrum, het feit dat er van tevoren niet duidelijk is hoeveel geld en tijd iets kost, doet niets af aan het grote voordeel dat er nog gestuurd kan worden tijdens het verloop van het project.

Een andere methode, zoals *Waterfall*, werkt alleen als er van tevoren vastgelegd is wat er gedaan moet worden. Deze methode is dus niet flexibel met features die geïmplementeerd moeten worden. (Kucheriavy, 2017) Zoals aangegeven in de vorige alinea is dit een groot gemis, omdat het project nog een andere koers op kan gaan. In Figuur 1 is het verschil tussen Waterfall en Agile weergegeven, met de termen “Fixed” (vast) en “Variable” (variabel). Hierin is af te lezen dat de “Time” (tijd) en “Cost” (kosten) vast staan bij Waterfall, daar waar dat bij Agile niet zo is. De flexibiliteit van de features van Agile is hier prominent in af te lezen.



Figuur 1 Weergave verschil tussen "Waterfall" en "Agile"

Aan het begin van deze paragraaf is de vraag opgeworpen of het mogelijk is om deze manier van werken aan te houden. Door de manier van werken is scrum (in vergelijking met *Waterfall*) een goede methode om aan te houden binnen het project.

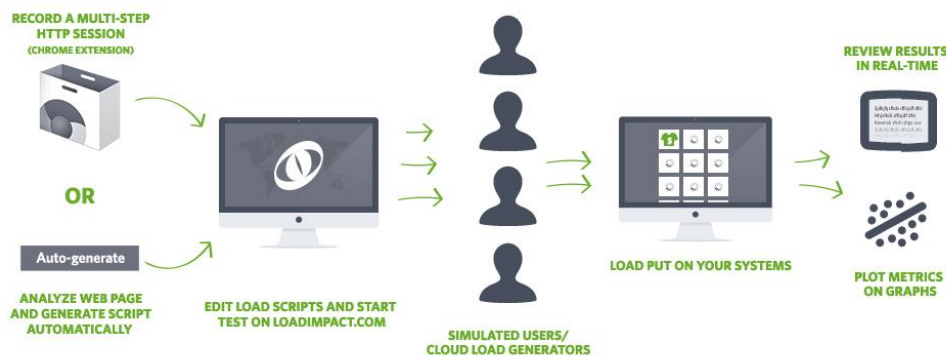
Meer informatie over Agile Scrum als theorie wordt toegelicht in hoofdstuk 5.1 ("Agile Scrum"). In hoofdstuk 5.1.1 ("Aanpassingen ten behoeve van dit project") is te lezen hoe de Scrum methode binnen dit project vormgegeven is. Daarbij kan in hoofdstuk 5.1.2 ("Ontwikkelproces project binnen Srcode") gelezen worden hoe dit uiteindelijk uitgevoerd is.

4.3.2 Loadimpact

Loadimpact is een tool waarmee laadtijden en prestaties van een dienst getest kan worden. Dit kan een webapplicatie, mobiele applicatie of een API zijn. Deze tool is nodig voor het meten van de laadsnelheden van de pagina's, zoals aangegeven in de doelstelling in 3.1 ("Probleemdefinitie"). Tijdens het maken van de proof of concept hoeft geen extra rekening gehouden te worden met LoadImpact, omdat het een dienst is die in de Cloud draait en geen verdere installatie vereist. Wel moet de proof of concept gaan werken op het internet. Zie Tabel 5 Projectrisico's hoe dit risico wordt afgevangen. Immers moet LoadImpact bij het programma kunnen om metingen te verrichten. De tool geeft tevens een goed inzicht in de resultaten, omdat deze zelf een uitgebreide functie heeft om de resultaten te filteren en te analyseren. (About LoadImpact, 2017)

LoadImpact is ook een geschikte tool om de performance te testen van de applicatie. Dit is nodig omdat het onderdeel is van de doelstelling van de opdracht.

De tool doet dit door virtuele gebruikers na te bootsen en ze bepaalde acties op de pagina's uit te laten voeren. Dit is weergegeven in Figuur 2 Werking prestatietest LoadImpact. Allereerst wordt er een reeks van *acties* opgenomen met behulp van een browser extensie. Bij een actie kan gedacht worden aan het klikken van een plaatje of het navigeren naar een pagina. Deze acties worden opgeslagen als een bestand, die geüpload kan worden naar het programma. Dit programma kan deze acties vervolgens simuleren en de laadtijden bijhouden. Daarbij kunnen er een bepaald aantal virtuele gebruikers aangemaakt worden die precies dezelfde acties uitvoeren.



Figuur 2 Werking prestatietest LoadImpact

Door dezelfde acties uit te voeren op zowel de frontend van Magento als de frontend van de proof of concept kan hierdoor bruikbare data worden gegenereerd voor het experiment. Meer informatie over dit onderzoek kan worden gevonden in hoofdstuk 7 ("Praktijkgericht onderzoek").

4.3.3 Siege

Siege is een *command line tool* die ongeveer dezelfde functies heeft als LoadImpact, maar dan op de lokale machine. Mocht het zo zijn dat de proof of concept niet online kan draaien en LoadImpact dus niet gebruikt kan worden (zie 10.1 "Project risico's"), zal er gebruik worden gemaakt van Siege. Deze tool kan pagina weergaven nabootsen en de testresultaten teruggeven. Deze testresultaten lopen uiteen van reactietijd, hoeveelheid data er verstuurd wordt, het aantal succesvolle transacties en meer. (siege(1): HTTP/HTTPS stress tester - Linux man page, 2017)

4.3.4 RESTful Web services

Er zal gewerkt worden met *RESTful Web services*, hierna te noemen *REST*. REST is een methode van dataoverdracht doormiddel van verzoeken aan een webserver. (Relationship to the World Wide Web and REST Architectures, 2004)

De reden hiervoor is dat er een vorm van communicatie door middel van verzoeken van en naar een webserver gedaan moet bestaan. Omdat Magento 2.x deze techniek ondersteunt, is REST de perfecte kandidaat om deze communicatie te faciliteren. (List of service names per module | Magento 2 Developer Documentation, 2016) De kennis over REST is al bekend verondersteld bij de opdrachtnemer, omdat dit tijdens de opleiding (in de specialisatie *webdevelopment*) behandeld is.

4.3.5 Postman

Er wordt een tool gebruikt om de API uit te proberen, deze heet Postman. (Postman | Supercharge your API workflow, 2017) De applicatie stelt de gebruiker in staat om de REST API endpoints aan te spreken en de resultaten in te zien. Dit is met name handig om zonder al te veel code de API uit te pluizen en om risico #1 van Tabel 5 Projectrisico's zo spoedig mogelijk uit de weg te gaan.

5 Theorie

De methodieken die gebruikt worden bij het uitvoeren van de opdracht berusten op een bepaalde theorie. In dit hoofdstuk zullen deze worden uitgelegd.

5.1 Agile Scrum

Scrum is een ontwikkelmethode die gebruik maakt van zogenaamde *sprints*. Een sprint kan gezien worden als een vaste periode waarin vooraf geselecteerde problemen opgelost worden, aan de hand van prioriteit. Voordat deze problemen (ook wel *stories* genoemd) in een sprint terecht komen worden ze beoordeeld op moeilijkheid en hoeveelheid tijd dat een programmeur hier mee bezig zal zijn. Doordat de sprints een vaste periode lang zijn, kan het zo zijn dat niet alle problemen in één sprint op te lossen zijn. Er zal dus gekozen moeten worden welke stories er mee doen met een sprint. De rest van de stories belanden op de *backlog*: een verzameling aan stories die niet meedoen met de huidige sprint. Elke dag vindt er een stand-up (een korte vergadering aan het begin van de dag) plaats waarin bepaald wordt hoe het er voor staat met de huidige sprint.

Aan het einde van een sprint wordt er een demo gegeven van de gerealiseerde oplossingen. Er worden vervolgens nieuwe stories bedacht en beoordeeld om een nieuwe sprint op te zetten (Rubin, 2012).

Binnen Scrum zijn er een drietal rollen te verdelen, namelijk die van de *Scrum Master*, de *Product Owner* en het *Team*.

- De **Scrum Master** is de persoon die het ontwikkelproces ondersteunt en het team aanstuurt tot het verrichten van werk voor het oplossen van de stories. Verder zorgt hij er voor dat het team ongestoord kan werken, zonder tussenkomst van ander werk. Ook zorgt hij ervoor dat de projectmanager niet direct contact heeft met het team om bijvoorbeeld de eisen van de stories aan te passen.
- De rol van de **Product Owner** is om de belangen van de klant te vertegenwoordigen, dit kan de klant zelf zijn of de opdrachtgever. Daarbij betaalt deze partij de rekeningen; hij heeft daarom ook het meeste belang bij kwalitatief goede resultaten. Een andere rol die de Product Owner speelt is het bepalen welke stories van de backlog er mee gaan in de aankomende sprint. Hierdoor heeft de klant zelf de touwtjes in handen door te kiezen wat er opgepakt gaat worden.

-
- **Het Team** is verantwoordelijk voor het afleveren van het product aan het einde van iedere sprint. De leden van het team zorgen er voor dat het product voldoet aan de wensen van de klant, zoals deze beschreven staan in de stories. Het team werkt onder andere aan de analyse, het ontwerp, de ontwikkeling en het testen van het product.

De rol van deze theorie binnen de afstudeeropdracht is het feit dat deze ontwikkelmethodiek toegepast wordt voor het realiseren van de te maken producten. Scrum maakt het mogelijk om veel eenvoudiger tot de realisatie van een (tussen)product te komen binnen een vooraf gedefinieerde termijn. Door het doen van daily standups en het evalueren van sprints kan er tijdens het project nog bijgestuurd worden, daar waar dat nodig lijkt te zijn. (Rubin, 2012)

Dit is een vorm van voortschrijdende kennis, omdat er methodes voorgeschreven worden waarmee problemen kunnen worden opgelost.

5.1.1 Aanpassingen Agile Scrum ten behoeve van dit project

Scrum telt normaal gesproken drie rollen, namelijk die van de *Scrum Master*, de *Product Owner* en het *Team*. Omdat het Team bestaat uit een enkel individu en er geen echte Product Owner is, zijn er bepaalde onderdelen aangepast van deze ontwikkelmethode.

Srcode heeft de rol van zowel de Scrum Master als de Product Owner ingenomen. Mede doordat de stories van de sprints vastgelegd worden door de Product Owner en deze tijdens de sprint niks mag wijzigen kan dit uitgevoerd worden. Tijdens de stand-up neemt Srcode de rol in van de Scrum Master, met alle taken van dien.

Er zijn in totaal 3 sprints met de duur van 2 week gedaan. De stand-ups zijn iedere dag gedaan voor een *Scrum Board*. Op dit fysieke bord zijn vier kolommen aangebracht met de koppen “Backlog”, “Sprint”, “Mee bezig” en “Afgerond”. Onder de verschillende koppen hingen plakbriefjes met daarop de beschrijving van de stories. Dagelijks zijn de stories (van de sprint) bekeken en is de voortgang besproken.

Gedurende de dag verschuift het Team de stories tussen de verschillende kolommen, om aan te geven wat de status is van een bepaalde story. Deze voortgang is door de Scrum Master in de gaten gehouden gedurende het ontwikkelproces.

5.1.2 Ontwikkelpoces project binnen Srcode

Binnen dit project zijn er stories aangemaakt in een programma genaamd *GitLab*. Dit is een platform waar onder andere code opgeslagen staat en waar een speciaal systeem in zit om Scrum uit te kunnen voeren. Hier is binnen het project gebruik van gemaakt. Stories heten in GitLab *issues*.

Er zijn issues aangemaakt per onderdeel dat gemaakt moest worden. Daarna ze zijn verdeeld over verschillende categorieën, om het geheel overzichtelijker te maken en de issues te verdelen in specifieke onderdelen. Zo is er bijvoorbeeld een categorie "*Magento 2*" gemaakt waar alleen issues in gezet zijn die te maken hebben met de API of met de implementatie van Magento 2.

Deze issues zo behapbaar mogelijk gemaakt door iedere issue op te delen in zogenaamde *tasks*. Deze tasks zijn een soort lijsten waaraan voldaan moet zijn voordat een issue afgerond is, en gesloten kan worden. Een voorbeeld hiervan kan worden gevonden in Figuur 3.

The screenshot shows a GitLab issue interface. At the top, it says 'Closed' in a red box, followed by 'Issue #5 opened a month ago by Rick Kuilman' with a profile picture, and '5 of 5 tasks completed'. The main title is 'Show Product of Category'. Below the title are three sections: 'How', 'Done when', and 'Test'. Each section contains a list of tasks with checkboxes, all of which are checked. The 'How' section has two tasks: 'Figure out what is needed to show Product of (let's say the home-) category' and 'Implement this feature'. The 'Done when' section has one task: 'The products of a certain category show up'. The 'Test' section has two tasks: 'Check if the products correspond with the category' and 'Add new product to category'. At the bottom, it says 'Edited a month ago'.

Closed Issue #5 opened a month ago by Rick Kuilman 5 of 5 tasks completed

Show Product of Category

How

- ☒ Figure out what is needed to show Product of (let's say the home-) category
- ☒ Implement this feature

Done when

- ☒ The products of a certain category show up

Test

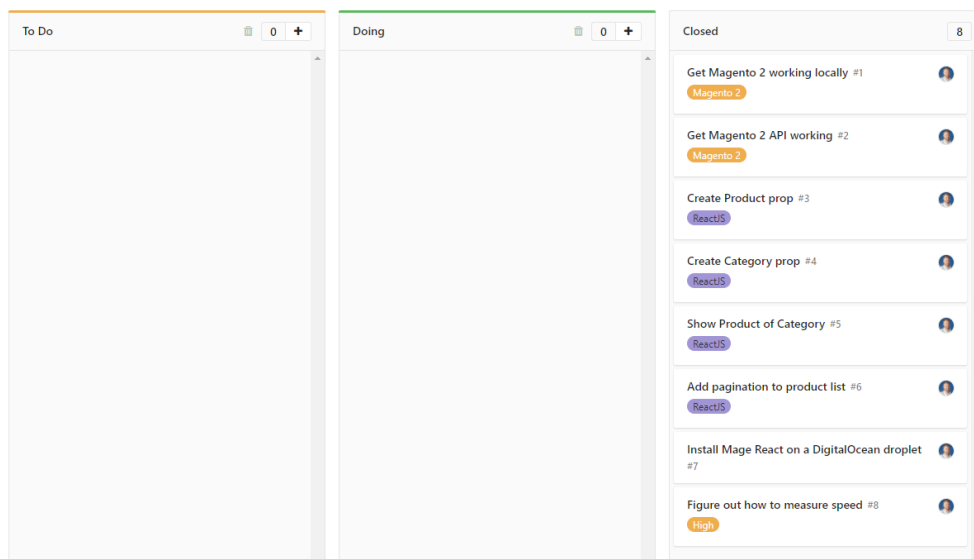
- ☒ Check if the products correspond with the category
- ☒ Add new product to category

Edited a month ago

Figuur 3 Voorbeeld van een issue opgedeeld in tasks. "How" (hoe de issue opgelost moet worden), "Done when" (om te bepalen of de issue gesloten kan worden) en "Test" (om de functionaliteit te testen op haar werking). In dit figuur is af te zien dat alle tasks voltooid zijn.

De issues zijn opgedeeld in zogenaamde sprints. De Product Owner geeft voor het begin van de sprint aan welke issues meegenomen moeten worden in de nieuwe sprint. De Scrum Master heeft deze issues doorgegeven aan Het Team en wijst de

taken toe aan de juiste persoon. Een voorbeeld van een voltooide (in dit geval de eerste) sprint kan gevonden worden in Figuur 4. De kolommen zijn “To Do”, “Doing” en “Closed”. Wanneer Het Team bezig was met een story, werd deze van “To Do” afgehaald en werd deze geplaatst bij “Doing”. Wanneer alle tasks voltooid waren, kon deze verplaatst worden naar “Closed”.



Figuur 4 Het digitale scrumbord waar de stories van de huidige sprint (in dit geval de eerste sprint) op te zien zijn. In dit geval is er af te lezen dat alle stories afgerond zijn.

5.2 Middleware

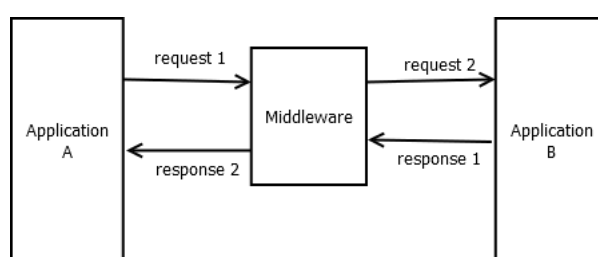
Voordat er een proof of concept gemaakt kan worden zal er eerst uitgezocht moeten worden op welke manier dit gerealiseerd kan worden.

Het boek *“Essential software architecture”* geeft duidelijkheid over de verschillende manieren waarop software gemaakt kan worden. Er wordt een architectuur beschreven die naadloos aansluit bij datgene wat er gemaakt moet worden om de doelstelling van het project te ondersteunen, namelijk een *middleware*.

Wanneer er gesproken wordt over middleware kan er veel mee bedoeld worden. Binnen de informatica is een middleware een bewezen methode om informatie uit te wisselen tussen verschillende (complexe) softwareapplicaties. In deze architectuur is middleware als het ware de pijpleiding tussen een of meerdere softwarecomponenten. Middleware kan ook gezien worden als de lijm tussen twee

softwarecomponenten, die de softwarebronnen aan elkaar koppelt. Dit kan zowel een een-op-een, een een-op-meer of een meer-op-meer verbinding zijn. Vanuit het perspectief van de klant en de eindklant hoeft er niets veranderd te worden. Een middleware is een verborgen onderdeel; de gebruikers hebben niet door dat er intern iets gewijzigd is. Behalve wanneer het uiteraard niet meer werkt, dan heeft de gebruiker er direct last van. (Gorton, 2011)

Aan de hand van Figuur 5 wordt de flow van requests en responses uitgelegd. (Defining Technology, Inc., 2012)



Figuur 5 Applicatie A kan data opvragen via de Middleware, middels het doen van een request (1). De middleware zal deze request (1) opvangen, bekijken wat de aangevraagde data is, om vervolgens zelf een request (2) te doen naar Applicatie B. Deze zal een directe response (3) teruggeven aan de Middleware. De Middleware vangt de response (3) op en kan eventueel nog iets doen met deze data. Zo zou een Middleware bijvoorbeeld ingezet kunnen worden om bepaalde data weg te laten of toe te voegen. De Middleware stuurt de data vervolgens, al dan niet bewerkt, terug naar Applicatie A in de vorm van een response (4). Applicatie A heeft niks door van Applicatie B en andersom.

Er zijn heel veel verschillende systemen en programmeertalen die het bouwen van een middleware mogelijk maken. Voor dit project is gekozen voor het PHP framework Lumen. Deze is al eerder gebruikt door de opdrachtnemer, en is simpel in opzet en gebruik.

5.3 Frontend framework

Om de doelstelling van de opdracht te ondersteunen zal er een frontend gebouwd moeten worden. Om niet alles zelf vanaf nul te hoeven bouwen, zal er een framework gebruikt worden. Er is enorm veel keuze in frameworks voor frontend systemen.

Het valt op dat veel van deze frameworks gebaseerd zijn op *Javascript*. Dit is een scripttaal die wordt meegeleverd in een webbrowser om webpagina's interactief te maken en applicaties te ontwikkelen. Zonder Javascript wordt er slechts een statische html pagina gereserveerd aan de bezoeker, waar geen wijzigingen meer in kunnen plaatsvinden. Javascript is tegenwoordig zo ingenesteld dat vrijwel iedere website een vorm van Javascript heeft draaien. Daarbij is het ook een gemakkelijke taal om aan te leren; het enige dat een ontwikkelaar nodig heeft is een browser en een tekstverwerker (Wiley, 2016)

Omdat de meest populaire frontend frameworks in Javascript zijn geschreven en het meest actief worden onderhouden (Korotya, 2017), is er voor dit project ook gekeken naar een dergelijk Javascript framework.

Er zijn twee grote partijen die zich kandidaat maken voor het te maken beroepsproduct: AngularJS en ReactJS. Deze zijn uiteengezet om te bepalen welke het meeste overeenkomt met de requirements van de proof of concept aan de hand van een artikel. (Kumar & Kumar, 2016) Omdat de keuze vrij is aan de opdrachtgever is er goed uitgezocht welke van deze twee het beste is in gebruik.

5.3.1 AngularJS

AngularJS is een open-source Javascript framework dat opgezet is door Google en nog steeds wordt onderhouden. Het framework zorgt er voor dat ontwikkelaars zogenaamde *single page applications* kunnen maken. Dit houdt in dat de html pagina soms in zijn geheel opnieuw wordt gegenereerd, zonder dat hierbij daadwerkelijk een nieuwe request naar de server wordt gedaan. Zoals al eerder vermeld: Javascript draait in de browser. AngularJS maakt gebruik van het MVC model, een model waar veel ontwikkelaars reeds bekend mee zijn. Ook maakt het framework gebruik van *Dependency injection*. Dit zorgt ervoor dat wanneer er extra modules worden gebruikt in de tekstverwerker, deze automatisch worden toegevoegd. Een ander uniek onderdeel van AngularJS is de manier waarop het Model en de View met elkaar in verbinding staan. Middels een technologie dat *Two-way Binding* heet, worden deze twee componenten met elkaar gesynchroniseerd. Wijzigingen in het model worden doorgezet naar de view en vice versa.

5.3.2 ReactJS

ReactJS is ook een open-source Javascript framework, is ontwikkeld door Facebook en wordt ook daadwerkelijk gebruikt in productie door Facebook. Dit framework onderscheidt zich van anderen door gebruik te maken van een zogenaamde *Virtual*

DOM, die zowel client-sided als server-sided kan worden gegenereerd. Een Virtual DOM is vergelijkbaar met de normale DOM, beide hebben ze een lijst met alle onderdelen van de webpagina met informatie over hoe ze zich onderling verhouden. ReactJS maakt dankbaar gebruik van dit gegeven door bij iedere wijziging in data slechts het gedeelte van de pagina opnieuw te renderen die daar belang bij heeft. Hierdoor is ReactJS heel efficiënt, ook wanneer de applicatie relatief zware berekeningen moet uitvoeren.

5.3.3 Vergelijking

Om methodisch tot een conclusie te kunnen komen welk framework beter is om te gebruiken, wordt er gebruik gemaakt van een *Likertschaal*. De Likertschaal is een schriftelijke methode om een *ordinale schaal*, die lastig te kwantificeren is, toch te kunnen behandelen op een *interval meetniveau*. (Sekaran, 2013) Een **ordinale schaal** is een schaal waarbij er natuurlijke ordening plaatsvindt. Dit houdt in dat het duidelijk is dat er een volgorde is, maar deze is niet te meten. Een **interval meetniveau** houdt in dat een meting uitgedrukt kan worden in getallen, en dus kwantificeerbaar is. Dit wordt gedaan door de natuurlijke (ordinale) schaal om te zetten naar punten.

In het geval van het vergelijken van de frontend frameworks, wordt er gekeken naar hoe zwaar bepaalde onderdelen wegen voor het de opdracht. Door deze verschillen vervolgens uit te zoeken, de Likertschaal toe te passen en uitkomsten van de weging bij elkaar op te tellen kan er gekwantificeerd worden welk framework het beste is.

De volgende punten zijn belangrijk bevonden:

1. Is het **open-source**? Srcode wil graag met open-source software werken, waardoor dit punt zwaar meeweegt.
2. Hoe is de **leercurve**? Als de leercurve erg steil is, kan het tijdig afronden van het beroepsproduct in het geding raken. Deze moet dus zo laag mogelijk zijn en weegt dus zwaar mee.
3. Kan er **server sided** worden **gerenderd**? Voor de toekomst is het wellicht handig om de mogelijkheid te hebben om ook de server te laten rekenen, maar voor nu weegt het niet zwaar.
4. Maakt het framework gebruik van het **MVC-model**? Zowel de opdrachtgever als de opdrachtnemer is bekend met dit model. Dit kan schelen in tijd en ontwikkelkosten, maar weegt niet heel zwaar.

-
5. Maakt het framework gebruik van een **DOM** waarvan de **performance** hoog is? Het weegt heel zwaar om zo veel mogelijk performance uit het framework te halen. (Zie 4.2 “Kwaliteitseisen & requirements”)
 6. Is er een **template engine** aanwezig? Om tijd te besparen is het handig om een template engine te hebben. Echter weegt dit niet heel zwaar.

De Likertschaal loopt van weging “Totaal niet zwaar”, “Niet zwaar”, “Niet heel zwaar”, “zwaar” en “Zeer zwaar”. Deze worden omgezet naar de interval meetniveau 's **1, 2, 3, 4 en 5**.

Wanneer een punt duidelijk de voorkeur heeft (zoals bijvoorbeeld dat de leercurve zo **laag** mogelijk moet zijn), krijgt dat framework de punten en het andere framework (dat duidelijk niet de voorkeur heeft) geen punten in de interval meetmethode. Dit komt neer op 0 punten.

De Likertschaal en de belangrijke onderdelen zijn terug te vinden in Tabel 1. Tevens is hier de vergelijking gemaakt tussen de twee systemen.

Tabel 1 Vergelijkingstabel met Likertschaal tussen AngularJS en ReactJS, op basis van verschillende onderdelen

	WEGING	ANGULARJS	REACTJS
1. OPEN-SOURCE	Zwaar	Ja	Ja
2. LEERCURVE	Zwaar	Hoog	Laag
3. SERVER SIDED RENDERING	Niet zwaar	Niet mogelijk	Mogelijk
4. AANWEZIGHEID MVC-MODEL	Niet heel zwaar	Ja	Alleen "View" laag
5. TYPE DOM HEEFT HOGE PERFORMANCE	Zeer zwaar	Reguliere DOM	Virtuele DOM (hoge performance)
6. AANWEZIGHEID TEMPLATE ENGINE	Niet heel zwaar	Ja (in HTML)	Ja (in JSX bestanden)

Door de nominale schaal "weging" om te zetten naar de interval schaal, en de uitkomsten bij elkaar op te tellen geeft dit het resultaat dat terug te vinden is in uit Tabel 2.

Tabel 2 Vergelijkingstabel met interval schaal tussen AngularJS en ReactJS, op basis van verschillende onderdelen, inclusief totalen

	WEGING	ANGULARJS	REACTJS
1. OPEN-SOURCE	4	+ 4	+4
2. LEERCURVE	4	+0	+4
3. SERVER SIDED RENDERING	2	+0	+2
4. AANWEZIGHEID MVC-MODEL	3	+3	+0
5. TYPE DOM HEEFT HOGE PERFORMANCE	5	+0	+5
6. AANWEZIGHEID TEMPLATE ENGINE	3	+3	+3
TOTAAL		10	18

Na beide frameworks op hun verschillen te hebben vergeleken is uit Tabel 2 op te maken dat ReactJS de hoogste score heeft gehaald op basis van de belangrijkste punten. Er zal daarom ook gewerkt gaan worden met ReactJS.

6 Beroepsproduct

Zoals beschreven in inleiding is een onderdeel dit rapport om het ontwikkelproces te beschrijven. In dit hoofdstuk zal beschreven worden hoe het ontwikkelde proof of concept tot stand is gekomen.

Het beroepsproduct bestaat uit meerdere onderdelen die in dit hoofdstuk los benoemd zullen worden. Allereerst zal er worden beschreven uit welke losse componenten het product bestaat. Daarna zal per component uiteen worden gezet hoe dat component bijdraagt aan het geheel en zal er met een technische blik worden gekeken naar de implementatie.

6.1 Componenten binnen het product

In dit hoofdstuk zullen de verschillende componenten waarmee gewerkt is om het beroepsproduct te maken worden benoemd. Daarbij zullen de onderlinge relaties vastgelegd worden.

Om een duidelijk en systematisch beeld te scheppen zal er gebruik worden gemaakt van Unified Modeling Language (afgekort UML). UML is een modelmatige taal om ontwerpen voor een informatiesysteem te kunnen maken en deze grafisch weer te geven. (Ambler, 2004)

Allereerst is het goed om te begrijpen wat de losse elementen zijn en hoe deze fysiek aan elkaar verbonden zijn. Dit schept een globaal beeld van het beroepsproduct, zodat het verder op in het verslag duidelijk is hoe de samenhang is.

In Diagram 1 is de fysieke laag van het beroepsproduct te zien met de verschillende componenten en hun onderlinge verbinding. Er valt op te zien dat er drie componenten zijn, verdeeld over twee nodes. Deze nodes zijn fysieke servers waar de componenten op geïnstalleerd staan.

De server nodes (zie legenda) staan gehost bij DigitalOcean en bevatten 2 GB geheugen, 40 GB schijfruimte en draaien Ubuntu versie 14.04.

De ReactJS applicatie (als component op server 1) maakt met behulp van https een verbinding met de Lumen Middleware (als component op server 2). Op deze zelfde

server staat ook de Magento 2 webshop (verbonden met een https-verbinding) en de bijbehorende database (verbonden met behulp van mysql).

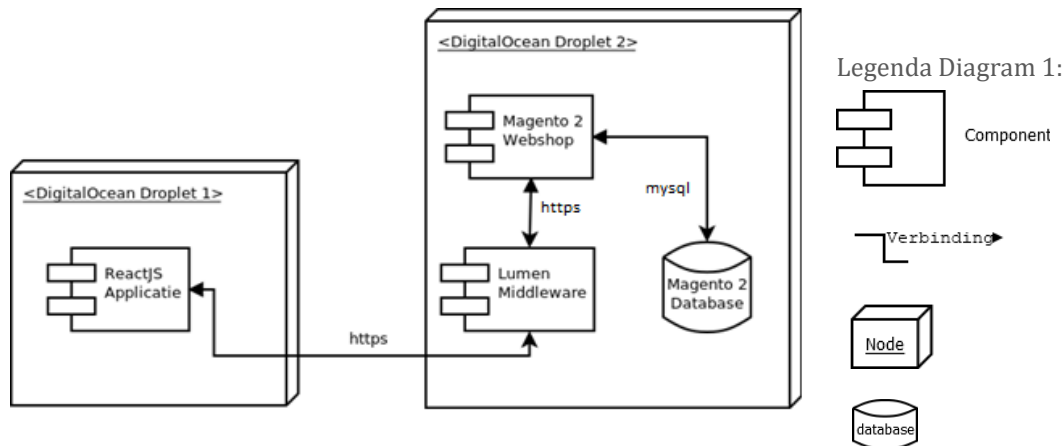


Diagram 1 UML diagram voor het grafisch weergeven van de fysieke laag van het beroepsproduct, bestaande uit twee Nodes (servers), drie Componenten en de onderlinge Verbinding. Rechts staat een legenda voor Diagram 1.

De onderlinge werking tussen de verschillende componenten is te zien in Diagram 2. Deze sequentiediagram geeft weer hoe de verschillende componenten aan elkaar verbonden zijn en met welke methode ze met elkaar communiceren. Een voorbeeld van een volledige werking van het product is als volgt:

1. De gebruiker (de Visitor) doet een aanvraag naar de ReactJS applicatie.
2. De ReactJS applicatie zal een http-request doen om de benodigde data te verkrijgen naar de Lumen middleware.
3. Lumen zal controleren of deze aanvraag valide is en zal daartoe handelen. Als de aanvraag valide is, zal hij op zijn beurt een http-request doen naar de Magento 2 applicatie met een authenticatie token. Hoe dit werkt, zal in hoofdstuk 5.2 worden toegelicht.
4. De Magento 2 applicatie zal deze request opvangen, waarna deze een SQL query stuurt naar de MySQL database.
5. De MySQL database haalt het data object op en stuurt deze terug naar Magento 2.
6. Magento 2 zal dit object omzetten naar JSON formaat, waarna deze teruggestuurd wordt naar Lumen.

-
7. Lumen herkent het antwoord van Magento 2, en stuurt hetzelfde JSON object terug naar de ReactJS applicatie
 8. De ReactJS applicatie bouwt het HTML document op om terug te kunnen sturen naar de bezoeker (de Visitor)

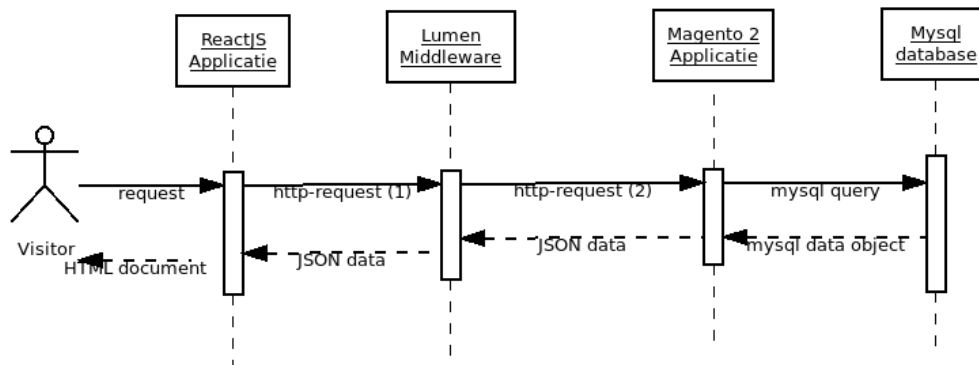


Diagram 2 - Sequentiedigram die de onderlinge relatie tussen de verschillende componenten weergeeft.

In de opvolgende hoofdstukken zullen de verschillende componenten bij langs gegaan worden om beter toe te lichten hoe de onderdelen werken en hoe ze in verhouding staan tot de andere componenten.

Eerst wordt de *Magento 2 webshop* uitgelegd, daarna de *Lumen middleware* en als laatste de *ReactJS Applicatie*.

6.2 Component “Magento 2 Webshop”

Zoals al eerder aangegeven biedt Magento 2 een rijke REST API aan om data vanuit de database te versturen via het HTTP protocol. Zoals te zien is in Diagram 1 communiceert Magento 2 met Lumen dit protocol.

Om deze verbinding tot stand te brengen is er een zogenaamde *token* nodig om bij afgeschermden delen van de backend te kunnen. Bepaalde onderdelen van de backend zijn namelijk niet publiek toegankelijk. In de backend van Magento 2 kan een dergelijke token aangemaakt worden. Deze token wordt opgeslagen in de Magento 2 backend, waarna deze ten alle tijde gevalideerd kan worden.

Wanneer er een HTTP-request wordt gedaan moet deze token worden meegestuurd. Als dit niet wordt gedaan, zal Magento 2 aangeven dat er onvoldoende rechten zijn.

In de documentatie van Magento 2 is beschreven hoe een token aangemaakt kan worden. Deze token is aangemaakt voor het beroepsproduct, zodat de Lumen middleware requests kan doen op het Magento 2 component.

6.3 Component “Lumen middleware”

Lumen is een lichtgewicht PHP framework die uitblinkt in het ontvangen van HTTP requests en het serveren van een response. Dit component staat geïnstalleerd op dezelfde server als Magento, maar staat wel los van de Magento 2 installatie.

De middleware heeft als functie het verkeer tussen de ReactJS applicatie en de Magento 2 applicatie te regelen. Dit moet gebeuren omdat Magento 2 een token nodig heeft om de communicatie af te kunnen handelen. Als deze token direct in de ReactJS applicatie wordt gebruikt, valt dit uit te lezen door de client en kunnen kwaadwilligen gemakkelijk misbruik maken door deze token te gebruiken voor andere doeleinden. Daarbij geeft de Lumen middleware een boel flexibiliteit met betrekking tot het manipuleren van data.

Lumen gebruikt een package voor het doen van een http request, namelijk *Guzzle*. Dit pakket is nodig omdat de manier van communiceren tussen de Magento 2 server en Lumen via HTTP gaat.

In de volgende paragrafen worden de twee onderdelen van de Lumen middleware beschreven, namelijk:

- De communicatie tussen Lumen en ReactJS
- De communicatie tussen Lumen en Magento 2

6.3.1 De communicatie tussen Lumen en ReactJS

Wanneer de ReactJS applicatie een request doet naar de middleware, zal deze als eerst bekijken hoe de request opgebouwd is en of deze valide is. Het opvragen van de categorielijst is bijvoorbeeld een GET request op de endpoint van “/categories”.

In Codevoorbeeld 3 is te zien hoe deze validatie wordt gedaan. Op regel 1 valt af te lezen dat een request met het type GET op de endpoint “/categories” afgehandeld wordt door de CategoriesController. Daarbij is in dit codevoorbeeld te zien welke routes er precies beschikbaar zijn.

```
01. $router->get('/categories', 'CategoriesController@index');
02. $router->get('/categories/{category_id}', 'CategoriesController@show');
03. $router->get('/products/{product_id}', 'ProductsController@show');
```

Codevoorbeeld 3 Alle routes die de middleware ondersteund, waarbij de verschillen in endpoint bepalen welke controller deze request afhandelt.

Met behulp van deze drie routes is alle informatie op te halen die nodig is voor het beroepsproduct. Hieronder is te zien welke routes er zijn, een korte beschrijving van de route, welke data deze route nodig heeft en welke data terugkomt.

- **/categories**

Haalt data van alle categorieën op. De data bestaat uit:

name (naam van de categorie)

id (de identifier voor de categorie)

Deze functie wordt gebruikt voor het opbouwen van het menu aan de bovenkant van de applicatie. De naam van de categorie wordt gebruikt om het knopje een beschrijvende naam te geven. De identifier wordt gebruikt om de knop te koppelen aan een bepaalde categorie, die gebruikt kan worden in de rest van de applicatie

- **/categories/{category_id}**

Haalt data op van een specifieke categorie, meegegeven via de GET request. De data bestaat uit:

name

products

Deze functie wordt gebruikt voor het opbouwen van de categoriepagina. Op deze pagina staat de naam van de categorie en een lijst van producten die onder de categorie vallen. De product data bestaat van ook uit een naam, een afbeelding, een identifier en een prijs.

-
- `/products/{product_id}`

Haalt data op van een specifiek product, meegegeven via de GET request.
De data bestaat uit:

id
name
image
description
price

Deze functie wordt gebruikt voor het opbouwen van de productpagina. Op deze pagina staat de naam, afbeelding, omschrijving en prijs van het product.

Nadat de request herkend is, wordt de data doorgestuurd naar de desbetreffende controller, zoals te zien in Codevoorbeeld 3. De controller zal de communicatie leggen met Magento 2, waarna de data teruggestuurd wordt naar ReactJS.

Hoe de communicatie tussen Lumen en Magento 2 verloopt, staat beschreven in de volgende paragraaf.

6.3.2 De communicatie tussen Lumen en Magento 2

Nadat de data van de GET request is doorgestuurd naar de bijbehorende controller, zal deze een eigen request doen naar Magento 2. Dit gebeurt aan de hand van de package Guzzle.

In Codevoorbeeld 4 is te zien hoe de correcte data wordt opgehaald door een GET request te doen naar de Magento 2 server. Op regel 11 wordt de “base_url” opgehaald van de Magento 2 installatie. Doordat dit pad kan veranderen is dit er niet *hardcoded* ingezet, maar is hier een variabele voor gemaakt.

De token die Magento 2 nodig heeft voor het autoriseren van een request (zoals uitgelegd in 6.2 Component “Magento 2 Webshop”) wordt meegestuurd via een header op regelnummer 14.

De daadwerkelijke request wordt uitgevoerd op regel 20. Er is te zien dat de request voor het ophalen van de data voor een specifieke categorie er vrij complex

uit ziet. Deze manier van ophalen staat beschreven in de documentatie van Magento 2. Tevens is te zien dat de betreffende “categorie_id” wordt ingevoerd op deze regel.

De response van Magento 2 wordt vervolgens terug gestuurd naar de router die het weer terugstuurt naar de ReactJS frontend, zoals beschreven in de vorige paragraaf.

```
01. namespace App\Http\Controllers;
02.
03. use Laravel\Lumen\Routing\Controller as BaseController;
04. use GuzzleHttpClient;
05.
06. class CategoriesController extends BaseController
07. {
08.     public function show($category_id) {
09.
10.         $client = new Client([
11.             'base_uri' => env("MAGENTO_BASE_URL"),
12.             'headers' =>
13.             [
14.                 'Authorization' => env("MAGENTO_TOKEN")
15.             ]
16.         ]);
17.
18.         return $client->get('products?searchCriteria[filter_groups][0][filters]
19.         [0][field]=category_id&searchCriteria[filter_groups][0][filters][0][value]=' .
20.         $category_id . '&searchCriteria[filter_groups][0][filters][0]
21.         [condition_type]=eq&searchCriteria[filter_groups][1][filters][0]
22.         [field]=type_id&searchCriteria[filter_groups][1][filters][0]
23.         [value]=configurable&searchCriteria[filter_groups][1][filters][0]
24.         [condition_type]=eq',
25.         )->getBody()->getContents();
26.     }
27. }
```

Codevoorbeeld 4 de CategoriesController die aan de hand van input de correcte data ophaalt van de Magento 2 API met behulp van een token

6.4 Component “ReactJS Applicatie”

Binnen Srcode wordt er met name ontwikkelt in PHP. Voor deze frontend applicatie is er gekozen om gebruik te maken van ReactJS, zoals toegelicht in hoofdstuk 5.3 (“Frontend framework”).

ReactJS op zichzelf is een framework waar applicaties mee gemaakt kunnen worden en is opgebouwd uit allerlei kleinere stukjes code. Wanneer deze worden samengebracht kan er worden gesproken over een framework. Er kunnen functionaliteiten worden toegevoegd door zogenaamde packages toe te voegen aan het framework. Een package is een verzameling aan code dat ingericht is om functies toe te voegen aan het framework naar de wensen van de ontwikkelaar.

Node Package Manager (kortweg npm) is een programma die deze packages download van een centrale online bron en deze koppelt aan het framework (What is npm? | npm Documentation, 2017). Wanneer een ontwikkelaar functionaliteit mist in het framework kan deze zelf gemaakt worden, maar het kan zijn dat andere ontwikkelaars dezelfde functionaliteit reeds heeft gemaakt en beschikbaar heeft gesteld (in de vorm van een package) voor andere ontwikkelaars. Op de website van npm kan worden gezocht naar packages.

In de codebase is een speciaal bestand aanwezig die de gebruikte packages bevat, namelijk *package.json*. Wanneer npm wordt aangeroepen in de codebase, zal deze de packages met de juiste versienummer downloaden en wegschrijven op het systeem. De packages die zijn gekozen voor dit project zullen hieronder worden beschreven, alsmede de reden waarom deze zijn gekozen.

6.4.1 Webpack en Babel

ReactJS ondersteunt zogenaamde JSX en ES6 Javascript syntax. JSX is een modernere vorm van Javascript dat nog niet door alle browsers wordt ondersteund. Om wel gebruik te kunnen maken van deze moderne Javascript, moet deze code omgezet worden naar gewone Javascript zodat alle browsers dit kunnen uitvoeren.

Webpack en Babel zijn twee packages die JSX en ES6 code omzetten naar Javascript dat elke browser herkent (ES5). Dit wordt gedaan door alle projectbestanden uit te lezen en deze samen te voegen (met behulp van een compiler) tot één bestand (*application.js*), zie Diagram 3 (Grider, 2017). Op deze manier kan er gebruik worden gemaakt van een moderne taal, zonder dat er nagedacht hoeft te worden over de compatibiliteit met de verschillende browsers.

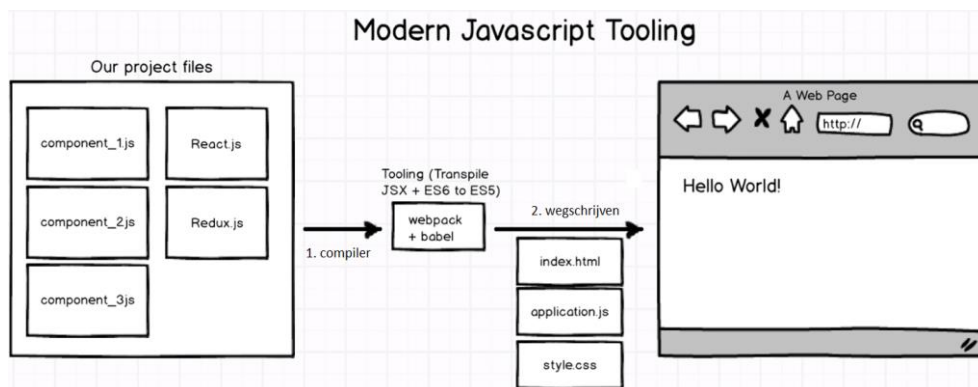


Diagram 3 Getekende diagram van het compilen (1.) van JSX javascript bestanden van het project naar CS5 Javascript bestanden. Deze worden weggeschreven (2.) op het systeem (Grider, 2017)

6.4.2 Lodash

Er ontbreken veel functionaliteiten in Javascript die andere programmeertalen wel bieden om bijvoorbeeld te itereren over een lijst of het correct manipuleren van verschillende datatypes. Lodash is een verzameling aan handige functies die deze functies wel beschikbaar maakt voor Javascript. Dit neemt veel werk uit handen van de ontwikkelaar en bevordert de leesbaarheid van de code. (Lodash, 2017)

In Codevoorbeeld 5 is te zien hoe Lodash is gebruikt in het project. Op regel 1 is te zien dat lodash wordt geïmporteerd onder de alias “_”. De reden waarom een underscore als alias wordt gebruikt, is om de leesbaarheid van de code te bevorderen. Het kan gebeuren dat Lodash meerdere malen op een regel wordt gebruikt. Door een simpele alias te gebruiken (in dit geval “_”), blijft de code netjes leesbaar.

De functionaliteit die in deze code wordt toegevoegd is het mappen van een lijstje aan categorieën, zie regel 6. Een dergelijke functionaliteit bestaat in Javascript, maar Lodash geeft een extra optie mee om tegelijk de losse elementen te bewerken. In dit geval wordt er een lijstje opgesteld met een link naar de categorie.

```

01. import _ from 'lodash';
02.
03. class CategoryIndex extends Component {
04.   renderCategories() {
05.
06.     return _.map(this.props.categories, category => {
07.       return (
08.         <li key={category.id}>
09.           <Link to={'/c/${category.id}'} >
10.             {category.name}
11.           </Link>
12.         </li>
13.       );
14.     });
15.   }
16. }

```

Codevoorbeeld 5 Gebruik van Lodash om ontbrekende Javascript functionaliteit toe te voegen en leesbaarheid van de code te verbeteren.

Een ander goed voorbeeld is te zien in Codevoorbeeld 6. Daar wordt de functie *mapKeys()* op regel 7 gebruikt om de *keys* van een *array* te nemen, en deze gelijk individueel te manipuleren.

Vanuit Javascript is dit te doen door eerst de keys van de array op te halen, waarna er een *foreach*-loop geschreven moet worden om ze individueel te manipuleren. Wederom draagt dit bij aan leesbare code .

```

01. import { FETCH_CATEGORIES } from '../actions';
02. import _ from 'lodash';
03.
04. export default function(state = {}, action) {
05.   switch (action.type) {
06.     case FETCH_CATEGORIES:
07.       return _.mapKeys(action.payload.data.children_data, 'id')
08.     default:
09.       return state
10.   }
11. }

```

Codevoorbeeld 6 Het gebruik van *mapKeys()* om direct door de losse keys heen te lopen in plaats van het schrijven van een losse *foreach*-loop.

6.4.3 Axios

Axios is een package die het mogelijk maakt om allerlei vormen van HTTP requests te sturen naar een server. De proof of concept zal er van een type gebruik maken, namelijk de GET request. Hoewel het vanuit Javascript mogelijk is om HTTP requests te maken, is het beter om gebruik te maken van Axios. Dit komt met name doordat de ontwikkelaar niet zelf het complexe systeem van het maken van HTTP requests hoeft te maken, wat tijd en mankracht bespaard.

Zoals te zien is in Codevoorbeeld 7 wordt het de axios-package ingeladen op regel 1. Op regel 6 is te zien dat de basis url van de middleware-laag wordt geïnitieerd.

In dit voorbeeld worden alle categorieën opgehaald door een GET request te doen naar de middleware op de “categories” endpoint. Op regel 10 is te zien dat deze request wordt opgebouwd aan de hand van de eerder gedefinieerde basis url. De request en de resultaten (de payload) van deze request worden vervolgens opgevangen en teruggestuurd naar het object die deze functie aangeroepen heeft.

```
01. import axios from 'axios';
02.
03. export const FETCH_CATEGORIES = 'fetch_categories';
04. export const FETCH_PRODUCTS = 'fetch_products';
05.
06. const BASE_URL = 'https://188.226.137.116/rest/V1';
07.
08. export function fetchCategories() {
09.
10.   const request = axios.get(`${BASE_URL}/categories`);
11.
12.   return {
13.     type: FETCH_CATEGORIES,
14.     payload: request
15.   }
16. }
```

Codevoorbeeld 7 De implementatie van axios voor het maken van HTTP requests naar de middleware.

7 Praktijkgericht onderzoek

In dit hoofdstuk zal er beschreven worden hoe het praktijkgerichte onderzoek de 2^e hoofdvraag van de opdracht (“**In hoeverre heeft het vervangen van een frontend impact op de laadsnelheid van bepaalde pagina’s van de webshop?**”) ondersteund. In de inleiding van dit hoofdstuk zal het onderzoek en de onderzoeksstrategie geïntroduceerd worden en zal er een hypothese worden opgesteld. Het daarop volgende hoofdstuk zal worden toegelicht op welke manier er bestaande kennis is gebruikt binnen het onderzoek en hoe deze bijdraagt aan het onderzoek.

Ook worden de methoden die gebruikt zijn voor het uitvoeren van het onderzoek uitgelegd en worden de resultaten van het onderzoek beschreven en weergegeven. Er wordt een analyse en discussie uitgevoerd, alvorens het concluderen en aanbevelen.

7.1 Inleiding

Om het verschil in snelheid van beide frontend systemen te onderzoeken zal er als onderzoeksstrategie *een experiment* worden gedaan worden. Een experiment heeft als doel om oorzaak-gevolgrelaties te onderzoeken met kwantitatieve resultaten. Het type is een single-case-mechanisme-experiment (Denscombe, 2004). Deze kijkt niet alleen naar oorzaak-gevolg, maar er wordt ook gekeken hoe de oorzaak (input) en gevolg (output) met elkaar samenhangen door de omgeving waarin het ICT-artefact functioneert te manipuleren. Door wijzigingen te maken in de architectuur, kan gekeken worden welke aanpassing invloed heeft op het resultaat.

Een experiment sluit goed aan bij de wens van de opdrachtgever. Deze is immers benieuwd naar de prestatieverschillen, zoals terug te vinden is in hoofdstuk 3.1 (“Probleemdefinitie”). Dit is ook opgenomen in de lijst van requirements, zoals terug te vinden in hoofdstuk 4.2 (“Kwaliteitseisen & requirements”). Door het wijzigen van de frontend (**input**) kan gekeken worden in hoeverre dit gevolgen heeft voor de snelheid van het systeem (**output**).

Dit experiment zal bestaan uit het meten van de laadsnelheid van bepaalde pagina’s. De laadsnelheid is de snelheid (uitgedrukt in seconden) van het moment van het aanspreken van de pagina tot het volledig geladen zijn van de pagina.

Voor het experiment zal er met speciaal voor deze *proof of concept* gegenereerde productenset gewerkt worden. Er zullen 16.000 (zestienduizend) artikelen in de webshop komen te staan, verdeeld over een 300 categorieën. Deze producten worden gegenereerd door een optie die standaard in Magento zit. De aantallen hiervan zijn terug te vinden in de requirements in hoofdstuk 4.2 ("Kwaliteitseisen & requirements").

Een andere requirement die is opgesteld is dat er zestig bezoekers tegelijkertijd op de pagina terecht moeten kunnen komen. Een onderdeel van het experiment zal ingaan op de vraag of de snelheid van de frontend afhankelijk is van het aantal bezoekers dat tegelijk op de webshop aanwezig is. Voor dit experiment zal er worden gekeken naar de laadsnelheid bij oplopende virtuele bezoekers. Dit zal in de eerste instantie gaan van 1 tot 60, over een tijd van zes minuten. Wanneer blijkt dat er geen duidelijke conclusies getrokken kunnen worden uit de resultaten ten gevolge van het aantal metingen, zal deze opgehoogd worden per 5 over dezelfde tijdsspanne.

De laadsnelheid (t) van de pagina's van de proof of concept (nieuw) moet sneller zijn dan de laadsnelheid (t) van de pagina's in de huidige situatie (huidig).

In de gewenste situatie zullen de laadsnelheden van de geselecteerde pagina's ten minste sneller zijn dan de laadtijden van de huidige situatie.

De hypothese van het experiment luidt al volgt:

Door het gebruik van de ReactJS frontend (*nieuw*) zullen de gemiddelde laadtijden ($\bar{t}$), genomen over alle metingen, van de geselecteerde pagina's ten minste lager zijn dan de gemiddelde laadtijden ($\bar{t}$), genomen over alle metingen, van de geselecteerde pagina's in de huidige situatie (*huidig*).

$$H = \bar{t}_{\text{nieuw}} < \bar{t}_{\text{huidig}}$$

Deze hypothese is te bevestigen (en te ontkrachten) wanneer uit het experiment blijkt dat de Magento frontend lagere laadtijden heeft.

7.2 Methoden

Conform de beheersmaatregel van risico #3 in Tabel 5 ("Projectrisico's") van hoofdstuk 10.1, zal de data verkregen worden met het programma LoadImpact en niet met Siege. Dit komt omdat de proof of concept online werkt en er dus niet lokaal gemeten hoeft te worden.

Deze Clouddienst is speciaal ingericht voor het testen van performance en het meten van laadtijden van een specifieke set acties. LoadImpact biedt de mogelijkheid aan om een server te kiezen vanaf waar de virtuele bezoekers komen. Deze is ingesteld op Frankfurt in Duitsland.

Beide frontend systemen staan gehost op een online server bij DigitalOcean. Deze partij maakt het mogelijk om je eigen hosting op te zetten en deze op afstand in te richten. Ze staan beide in Amsterdam in Nederland. Beide systemen beschikken over 2 GB geheugen, 40 GB schijfruimte en draaien Ubuntu versie 14.04

Op de ene server is een installatie van Magento 2 gezet, waarop vervolgens de producten en de categorieën zijn gegenereerd volgens de requirements. De REST API is open gezet en de cache van Magento is aangezet om een eerlijke test te kunnen doen. Daarna is de server op correcte werking handmatig getest.

Op de andere server is een installatie van de frontend van ReactJS gezet en ook deze is op de correcte werking getest.

Hierna zijn er twee verzamelingen gemaakt met alle acties die plaatsvinden met het navigeren door de webshop. De volgende acties worden gedaan op beide webshops:

1. Er wordt genavigeerd naar de hoofdpagina
2. Er wordt genavigeerd naar de categoriepagina "Categorie 4"
3. Er wordt genavigeerd naar de productpagina "Product 17220"
4. Er wordt genavigeerd naar de categoriepagina "Categorie 65"
5. Er wordt genavigeerd naar de productpagina "Product 18400"
6. Er wordt genavigeerd naar de categoriepagina "Categorie 251"
7. Er wordt genavigeerd naar de productpagina "Product 19600"

Er wordt niet gepauzeerd tussen de requests door. Wanneer de ene request klaar is, zal de volgende automatisch worden gestart. De acties bevatten behalve de

pagina ook alle andere resources die de pagina nodig heeft om te laden. Hierbij kan gedacht worden aan afbeeldingen, extra scripts en *stylesheets*.

De tests zijn gevalideerd in LoadImpact, waarna de tests is uitgevoerd. Door oplopend (over een tijdsspan van 6 minuten) van 0 naar 60 simultane virtuele gebruikers te gaan, kan er bekeken worden wat het aantal bezoekers over de verloop van tijd doet met de laadsnelheid van de pagina's. Tussen de tests van beide systemen zijn de systemen volledig uit-en-aangezet om er voor te zorgen dat de ene test de andere niet beïnvloedt.

De test is meerdere malen uitgevoerd, op verschillende momenten van de dag om vertraging tussen de servers uit te sluiten.

LoadImpact geeft ontzettend veel data terug. De nodige data voor het bijdragen aan de doelstelling is vervolgens gefilterd, namelijk (1.) de laadtijd van de pagina's en (2.) het aantal actieve virtuele bezoekers zoals beschreven in hoofdstuk 4.2 ("Kwaliteitseisen & requirements"). Er is meer interessante data te vinden, maar deze dragen niet direct bij aan de doelstelling van de opdracht.

7.3 Resultaten

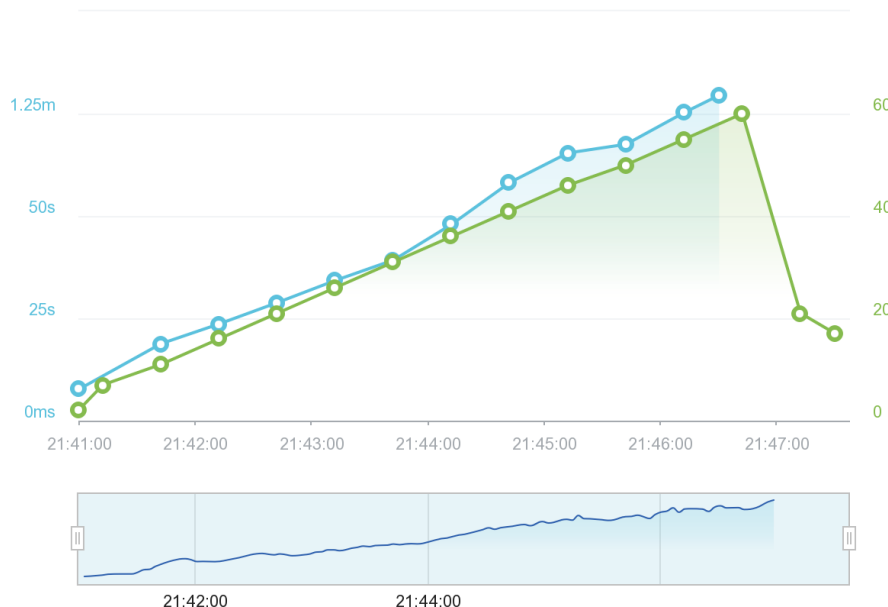
De resultaten van de twee tests zijn hieronder weergegeven in de vorm van een lijndiagram. In de lijndiagrammen van figuren 6 en 7 zijn drie assen te vinden. De linker verticale as geeft de laadtijd van de pagina's uitgedrukt in seconden weer. De horizontale as geeft de tijd weer van het moment van uitvoeren van de test. De rechter verticale as geeft het aantal actieve virtuele bezoekers weer.

Er is gekozen om deze twee vormen van data in één grafiek te plaatsen, zodat er afgelezen kan worden wat het aantal bezoekers doet met de laadtijd van de pagina, met als gemeenschappelijke factor de tijd van de test. De losse grafieken zijn terug te vinden in figuren 8 en 9 (voor Magento) en 10 en 11 (voor ReactJS)

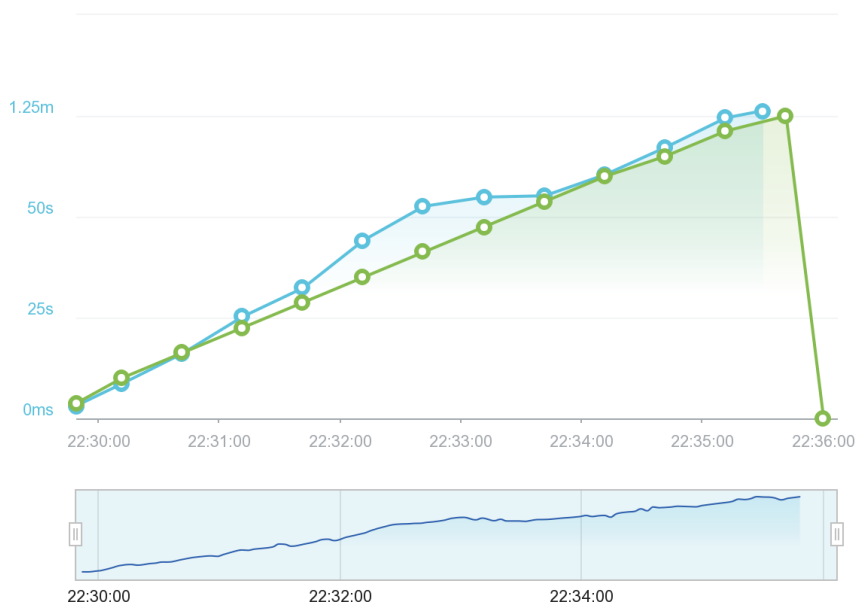
In Figuur x staan de resultaten van de Magento 2 frontend. In Figuur y staan de resultaten van de ReactJS frontend. De frontend van Magento 2 heeft in de 6 minuten 3,29 GB in 204.313 requests verstuurd. De frontend van ReactJS heeft in de 6 minuten 74,62 MB in 3334 requests verstuurd.

In de bijlagen op het eind van dit verslag zijn ook de tabellen te vinden met de resultaten van de tests van LoadImpact. Voor de Magento 2 frontend is dit Tabel 6

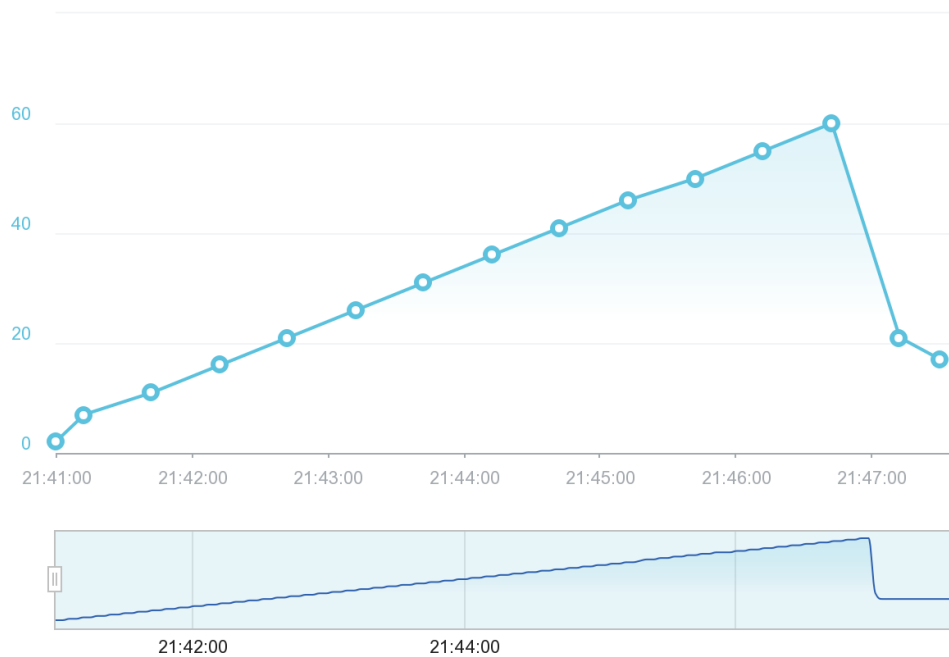
en voor de ReactJS frontend is dit Tabel 7. Hierin is onder andere de gemeten pagina, de locatie van de virtuele gebruiker, de snelst gemeten laadtijd, de gemiddelde laadtijd en de langzaamste laadtijd af te lezen.



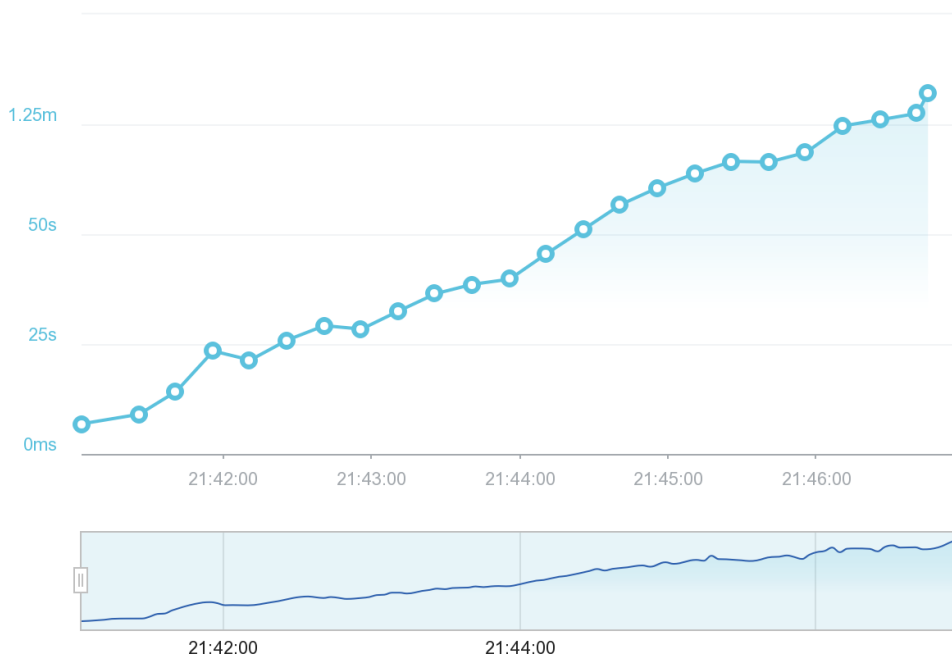
Figuur 6 Laadsnelheden (in het blauw), uitgedrukt in seconden op de linker verticale as en aantal actieve virtuele bezoekers (in het groen), uitgedrukt in aantallen op de rechter verticale as afgezet tegen de tijd voor de Magento 2 Frontend



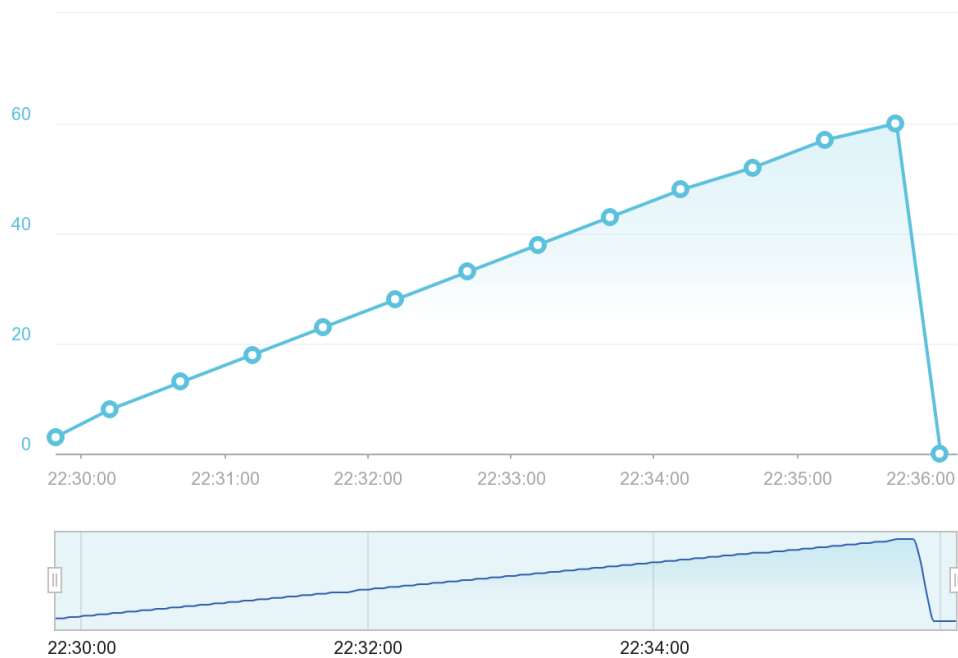
Figuur 7 Laadsnelheden (in het blauw), uitgedrukt in seconden op de linker verticale as en aantal actieve virtuele bezoekers (in het groen), uitgedrukt in aantallen op de rechter verticale as afgezet tegen de tijd voor de React JS Frontend



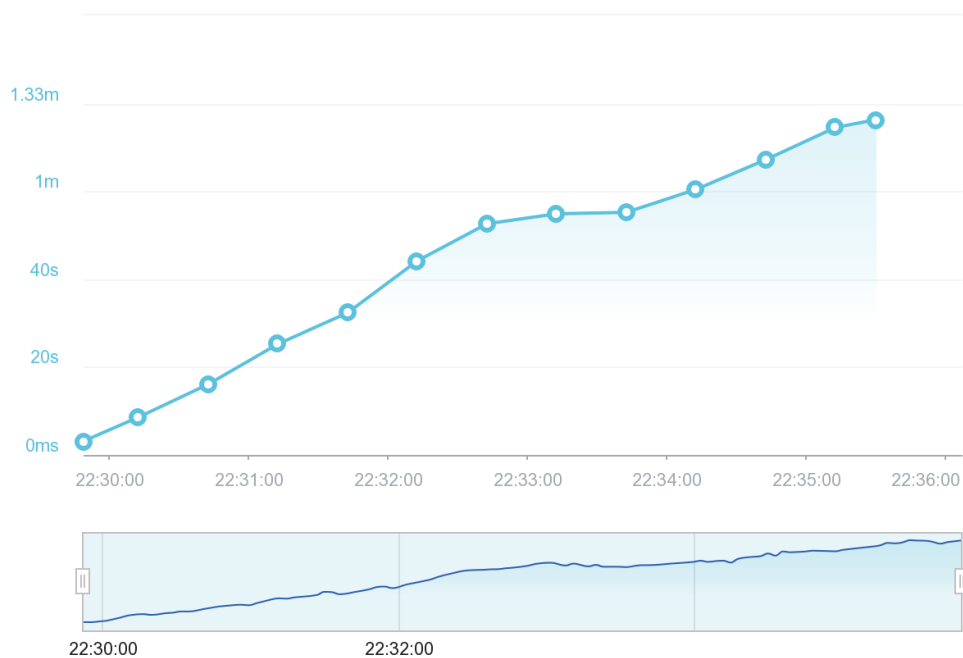
Figuur 8 Aantal actieve virtuele bezoekers uitgedrukt in aantallen op de verticale as afgezet tegen de tijd van het uitvoeren van het experiment van de Magento 2 Frontend



Figuur 9 Laadtijd van de pagina's uitgedrukt in seconden op de verticale as afgezet tegen de tijd van het uitvoeren van het experiment van de Magento 2 Frontend.



Figuur 10 Aantal actieve virtuele bezoekers uitgedrukt in aantallen op de verticale as afgezet tegen de tijd van het uitvoeren van het experiment van de ReactJS Frontend.



Figuur 11 Laadtijd van de pagina's uitgedrukt in seconden op de verticale as afgezet tegen de tijd van het uitvoeren van het experiment van de ReactJS Frontend

7.4 Discussie en analyse

In deze paragraaf wordt geanalyseerd of de het experiment correct verlopen is, en daarmee valide is bevonden.

Door figuur 8 en figuur 10 te bekijken (waarin het aantal virtuele gebruikers af te lezen valt) is te zien dat beide tests juist (valide) verlopen zijn. Beide tests hebben over een tijdsspan van zes minuten zestig virtuele gebruikers de tests succesvol laten doen.

In de tabellen voor de Magento 2 frontend (Tabel 6) en voor de ReactJS frontend (Tabel 7) is te zien dat er meerdere metingen zijn gedaan. Daarbij kan van de tabellen afgelezen worden dat de tests juist uitgevoerd volgens het schema, zoals deze beschreven staan in hoofdstuk 7.2 ("Methoden"). Hiermee is het experiment gevalideerd op juiste werking.

7.5 Conclusies en aanbevelingen

In deze paragraaf zal er een conclusie worden getrokken op basis van het onderdeel conclusie en analyse, waarna deze wordt aangevuld met aanbevelingen.

Om de hypothese ($t_{\text{nieuw}} < t_{\text{huidig}}$) te kunnen bevestigen of te kunnen ontkrachten zal er gekeken worden naar gemiddelde laadtijd van de geselecteerde pagina's. Omdat deze niet goed af te lezen zijn in de grafieken, zal hiervoor gekeken worden naar Tabel 6 en naar Tabel 7 (zie Bijlagen). Deze bevatten (onder andere) de gemiddelde laadtijden van de geselecteerde pagina's, uitgedrukt in seconden.

Door de gemiddelde laadsnelheid (*Avg.*) van de geselecteerde pagina's (*Pages*) te nemen, kan er een overzicht gemaakt worden zoals te zien in Tabel 3. ·

Tabel 3 Overzicht gemiddelde laadtijden in seconden van specifieke pagina's van de Magento 2 frontend, de ReactJS frontend en het verschil in laadtijden.

	Magento 2	ReactJS	Vershil
Homepage	6,57 s	8,71 s	-2,14 s
Category Page: id 5	5,06 s	7,56 s	-2,5 s
Category Page: id 65	5,46 s	7,54 s	-2,08 s
Category Page: id: 251	5,48 s	7,71 s	-2,23 s
Product Page: id 2	4,9 s	6,62 s	-1,72 s
Product Page: id 32	4,96 s	6,49 s	-1,53 s
Product Page: id 125	5,2 s	6,51 s	-1,31 s

De gemiddelde laadtijd (t), genomen over alle metingen, van de Magento 2 frontend (*oud*) is voor alle gespecificeerde pagina's sneller (de verschillen zijn negatief) dan de gemiddelde laadtijd ($\hat{t}$), genomen over alle metingen, van de ReactJS frontend (*nieuw*). De hypothese ($t_{\text{nieuw}} < t_{\text{huidig}}$) wordt hiermee niet waargemaakt en is daarmee ontkracht. Hiermee is aangetoond dat de frontend van ReactJS niet sneller is dan de Magento 2 frontend.

In theorie zou de frontend van ReactJS sneller moeten zijn, omdat deze minder data gebruikt (zie 7.3 "Resultaten") en niet hoeft te wachten tot de server een html pagina serveert. Uit het experiment blijkt dat er bijna 43 keer minder data wordt verstuurd in de nieuwe situatie.

Bij de start van de tests zijn de pagina's relatief snel geladen, maar dit lijkt lineair en parallel te stijgen naarmate er meer virtuele bezoekers worden toegevoegd. Het heeft er alle schijn van dat er ergens in het systeem een hardnekkige bottleneck is die er voor zorgt dat beide systemen niet sneller gemaakt kunnen worden op basis van het gebruikte framework.

Een aanbeveling voor het verbeteren van het experiment (en wellicht het oplossen van de problemen) is dan ook het toevoegen van een *PHP Performance Testing Suite*. Dit is een programma die in de gaten houdt hoe lang functies binnen PHP er over doen om te reageren op een bepaalde input. Vanuit hier kan vervolgens worden gekeken of (en indien dit mogelijk is: waar) de bottleneck is. Dit zal echter uitgezocht moeten worden in een andere opzet van het experiment.

8 Conclusie

Er is een losse ReactJS frontend geschreven in de vorm van een proof of concept die de requirements zoals ze zijn opgesteld in hoofdstuk 4.2 ("Kwaliteitseisen & requirements") niet geheel heeft kunnen nakomen. Requirement nummer 7 (*"De laadsnelheid van de pagina's van de proof of concept zou gemiddeld sneller moeten zijn dan de laadsnelheid van de pagina's in de huidige situatie."*) is niet kritiek voor de werking van het product, maar werd wel als zeer gewenst beschouwd. Het is gebleken dat deze requirement niet gehaald is na het uitvoeren van een experiment, waarbij gekeken is of de laadtijden van de frontend van ReactJS lager zijn dan die van Magento 2.

Door gebruik te maken van het programma LoadImpact zijn de twee frontendsystemen los van elkaar getest op performance. Er is een experiment opgesteld waarbij in zes minuten tijd zestig virtuele bezoekers vooraf gespecificeerde pagina's werden bezocht en de laadtijden hiervan werden opgeslagen.

De resultaten van het experiment zijn verrassend te noemen. De gemiddelde laadtijden van pagina's in de huidige situatie (met de frontend van Magento 2) zijn lager dan de gemiddelde laadtijden van pagina's in de nieuwe situatie (met de frontend van ReactJS). Het vervangen van de Magento 2 frontend levert vooralsnog dus geen snelheidswinst op.

Theoretisch gezien zou de frontend van ReactJS sneller moeten zijn, mede doordat ReactJS de pagina's zelf opbouwt in plaats van de server. Uit het experiment blijkt wel dat de hoeveelheid verstuurde data velen malen minder is met de ReactJS frontend ten opzichte van de Magento 2 frontend; maar liefst 43 keer minder. Dit is uiteraard een groot voordeel, maar dit maakt het des te vreemder dat de pagina's niet sneller laden.

Om de twee hoofdvragen van de opdrachtgever te beantwoorden:

1. **"Is het mogelijk om een andere frontend te gebruiken voor Magento?"** – Ja. Het is mogelijk om een andere frontend voor de Magento 2 backend te zetten. Dit is bewezen doormiddel van het maken van een werkende proof of concept. Deze doelstelling is gehaald en de deelvraag is beantwoord.

-
2. **“In hoeverre heeft het vervangen van een frontend impact op de laadsnelheid van bepaalde pagina’s van de webshop?”** – Het vervangen van de Magento 2 frontend heeft een negatieve impact op de laadtijden van de pagina’s. De laadtijden van de pagina’s zijn er niet op vooruit, maar op achteruit gegaan. Ook deze doelstelling is gehaald en de deelvraag is beantwoord.

De opdrachtgever is wel tevreden met het behaalde resultaat, mede door de aanbevelingen die de opdrachtnemer heeft gedaan (zie hoofdstuk 9 “Advies en Aanbevelingen”) om het beroepsproduct verder uit te kunnen werken tot een product dat wellicht wel aan de doelstelling voldoet.

Het probleem is met deze oplossing niet volledig opgelost voor Srcode, maar is wel een stap dichterbij het loskoppelen van de backend en de frontend. Hier is Srcode zeer tevreden mee, en wil hiermee dan ook graag verder werken. In de aanbevelingen is te lezen op welke manier dat gedaan zou kunnen worden volgens de opdrachtnemer.

9 Advies & aanbevelingen

In dit hoofdstuk levert de opdrachtnemer advies aan de opdrachtgever met betrekking tot de opdracht die de opdrachtgever heeft gegeven. De opdrachtnemer heeft hierin een adviserende rol. Elke aanbeveling wordt kort onderbouwd door de opdrachtnemer. Bij een dergelijke onderbouwing staan twee vragen centraal:

1. Wat gaat de maatregel opleveren, of wat levert de maatregel aan, voor de oplossing van het probleem?
2. Waarom vind de opdrachtnemer dat?

Omdat ‘*Waarom vind de opdrachtnemer dat?*’ een vage vraag is, zijn er een aantal randvoorwaarden gesteld aan het beantwoorden van deze vraag:

1. De aanbevelingen moeten duidelijk aangeven **wat** er door **wie** gedaan moet worden.
2. Het moet duidelijk worden wat de maatregel op gaat leveren.
3. Het moet duidelijk worden waarom de maatregel nodig is.
4. De oplossingen moeten op korte termijn kunnen worden opgepakt.

Er zullen een aantal punten langs komen op basis van onderwerpen die eerder aan bod zijn gekomen in dit verslag.

Door de niet-afgeronde eisen van de MoSCoW-methodiek langs te lopen, zoals ze gedefinieerd staan in hoofdstuk 4.2 (“Kwaliteitseisen & requirements”), kan er een advies worden gedaan waar Srcode iets aan heeft. Immers zijn de eisen in overleg met Srcode opgesteld. Hiermee is het advies dat afgegeven wordt nuttig voor de organisatie, wat direct de “waarom” vraag beantwoordt.

9.1 Advies uitrollen van het beroepsproduct

Het beroepsproduct dat gemaakt is tijdens deze opdracht is niet bruikbaar voor gebruik in *productie*. Dit wil zeggen dat het product in de huidige staat niet gebruikt kan worden voor de klanten van Srcode. De reden hiervoor is dat er nog veel functies ontbreken die minimaal nodig zijn voor het functioneren van een webshop. Denk hierbij bijvoorbeeld aan het winkelwagentje vullen en afrekenen. Om uit te zoeken of dit product uiteindelijk wel op productie gedraaid kan worden, moeten (de werknemers van) de opdrachtgever een aantal zaken uitzoeken. Deze zaken zijn toegelicht en uitgelegd in paragraaf 9.2 en 9.3. Door dit uit te zoeken levert het

de opdrachtgever uitsluitend of het product in te zetten is in de huidige bedrijfsvoering.

9.2 Aanbeveling voor afronden van het beroepsproduct

Zoals het beroepsproduct (de *proof of concept*) er nu bij ligt is het een werkend prototype. Echter is er uit de requirements op te maken dat een aantal punten nog open staan. Omdat dit prototype nog niet aan alle wensen (requirements) van de opdrachtgever voldoet, heeft de opdrachtnemer een aantal aanbevelingen voor de opdrachtgever om deze requirements wellicht alsnog te kunnen behalen.

7. Should have – De laadsnelheid van de pagina's van de proof of concept zou gemiddeld sneller moeten zijn dan de laadsnelheid van de pagina's in de huidige situatie.

Deze “should have” requirement (nummer 7) is niet gerealiseerd. Volgens de MoSCoW methode wordt deze eis als zeer wenselijk beschouwd, dit is de reden waarom deze aanbeveling nodig is.

Het advies luidt dan ook om uit te zoeken waarom de laadtijden van de ReactJS frontend niet lager of even snel zijn als de Magento 2 frontend. Zeker omdat uit het onderzoek naar voren is gekomen dat de hoeveelheid dataverkeer van ReactJS velen malen lager is. Het volgende moet gedaan worden door (de werknemers van) de opdrachtgever:

- Het kan zijn dat er een bottleneck is in de Magento API. Een manier om uit te zoeken of het aan de kant van Magento ligt is door een *Performance Test Suite* voor PHP te installeren. Een PHP Performance Test Suite is een speciaal programma die regel voor regel bijhoudt hoe lang de code bezig is met het uitvoeren van die code. Het programma kan vervolgens exact aangeven waar hij lang mee bezig is, en waar het snel gaat. Met die informatie kan eventueel bekeken worden of daar iets aan gedaan kan worden.
- Een andere optie is wellicht om de data die de middleware serveert te cachen, zodat er niet constant een aanvraag gedaan hoeft te worden naar de Magento 2 webshop. Ook is er een mogelijkheid om slechts een fractie van de data door te sturen naar de ReactJS frontend. Deze kan op dat moment al beginnen met het genereren van de pagina, en ontvang op een later moment (een paar seconden later) de rest van de data. Op die manier

is het voor de eindgebruiker een betere ervaring; de pagina lijkt immers in het begin sneller te laden in de ReactJS frontend.

Beide opties zijn op korte termijn uit te voeren en kunnen onafhankelijk van elkaar uitgevoerd worden.

9.3 Aanbeveling voortzetting van het beroepsproduct

Als er wel besloten wordt om het beroepsproduct voort te zetten, missen er nog een tweetal eisen in de “could have” categorie van MoSCoW. Deze twee “could have”-eisen (nummer 8 en 9) zijn niet gerealiseerd, maar zijn volgens de MoSCoW-methode ook minder belangrijk. Dit komt omdat de requirements alleen maar gedaan hoeven te worden als er tijd genoeg voor was.

8. Could have – De mogelijkheid om producten in de winkelwagen toe te voegen

9. Could have – De mogelijkheid om producten en/of categorieën te zoeken via een zoekbalk

Omdat de twee veel op elkaar lijken, zijn deze twee samen genomen. Het advies omtrent deze twee eisen is om (de werknemers van) de opdrachtgever gebruik te laten maken van de Magento 2 Docs. Deze zijn te vinden op “http://devdocs.magento.com/swagger/index_20.html”.

Voor het vullen van het winkelwagentje (requirement 8) moet er gebruik worden gemaakt van de POST request van de “*quoteCartItemRepositoryV1*” implementatie. Door de documentatie uit te klappen staat er beschreven hoe het toevoegen van een specifieke item aan de winkelwagen werkt.

Om een zoekopdracht te doen (requirement 9) moet er gebruik worden gemaakt van de GET request van de “*searchV1*” implementatie. Door de documentatie uit te klappen staat er beschreven hoe een zoekopdracht gedaan kan worden voor zowel de producten als de categorieën.

Deze functionaliteit moet worden toegevoegd aan zowel de Lumen middleware als aan de ReactJS applicatie.

Door dit advies op te volgen levert het een completere webshop op, waaraan eventueel doorgewerkt kan worden. De reden waarom dit advies moet worden

doorgevoerd is om te kunnen bepalen of het gebruik van het beroepsproduct (op korte termijn) ingezet kan worden op een productieserver.

9.4 Advies gebruik andere frontend frameworks

In de huidige vorm gebruikt het beroepsproduct ReactJS. Dit is gedaan aan de hand van een vergelijking tussen AngularJS en ReactJS. Er zijn echter veel meer frontend frameworks.

Op korte termijn zal uitgezocht moeten worden door (de werknemers van) de opdrachtgever of er andere frontend frameworks zijn die betere performance kunnen leveren dan de reeds vergeleken frameworks. Dit moet uitgezocht worden omdat de vergelijking die gedaan is door de opdrachtnemer voor het maken van dit beroepsproduct slechts tussen twee frameworks is gedaan. Dit levert als resultaat op dat Srcode weet of er een beter frontend framework bestaat voor het implementeren van een Headless oplossing voor Magento 2. Dit framework kan op dat moment worden vervangen.

10 Project

10.1 Project risico's

De projectrisico's zijn opgenomen in tabel 5 ("Projectrisico's"). De tabel kan worden afgelezen van links naar rechts. Het beschrijft een aantal risico's, de kans dat het gebeurt, de impact, het directe gevolg, de prioriteit dat er aan gegeven moet worden, de preventieve maatregel en de maatregel die getroffen moet worden als het verkeerd gaat.

10.2 Betrokken partijen

De betrokken partijen voor het project zijn:

- **De stagiair**

Rick Kuilman
rickkuilman@gmail.com
rick@srcode.nl

Zijn taak is het om de opdracht uit te voeren en de beroepsproducten af te leveren. Daarbij heeft hij de rol aangenomen van het Team in Scrum.

- **De bedrijfsbegeleider**

Jeroen Boersma
jeroen@srcode.nl

Zijn taak binnen het project is om de stagiair van het project bij te staan vanuit het bedrijf en sturing geven waar het nodig is. Daarbij heeft hij de rol aangenomen als Scrum Master en Project Owner voor de Agile Scrum ontwikkelmethodiek.

-
- **De stagecoördinator vanuit de Hanzehogeschool Groningen**

Dick Beks
d.b.beks@pl.hanze.nl

Het coördineren van de afstudeerstages.

- **Eerste examiner (docentbegeleider)**

Tjerk Harkema
t.j.harkema@pl.hanze.nl

De eerste examiner en begeleider vanuit de Hanzehogeschool Groningen. Tevens de contactpersoon voor de stagiair. Heeft een ondersteunende rol in het proces.

- **Tweede examiner**

Jos Bos
j.h.bos@pl.hanze.nl

De tweede examiner vanuit de Hanzehogeschool Groningen.

10.3 Competenties

Er is in dit document rekening gehouden met de competenties, zoals deze gesteld worden in bijlage "VII Leeruitkomsten per major" van de *handleiding afstudeerproject HBO-ICT 2016-2017*.

10.4 Planning en contactmomenten

In Tabel 4 is de planning te vinden voor het project. De rode lijn rechts naast "kalenderweek 16" is de Go-NoGo deadline, die in week 10 plaatsvindt. Dit is het moment om contact te hebben met de docentbegeleider. Daarna heeft de docentbegeleider een ondersteunende rol. Indien de stagiair het wenst kan contact worden opgenomen met de docentbegeleider. Dit zal de stagiair ook doen wanneer hij niet verder komt met het uitwerken van het afstudeerverslag, of als hij andere vragen heeft met betrekking tot het afstuderen.

Tabel 4 Project planning

Kalenderweek	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Werkweek	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Opstarten																			
Toepasbare Theorie opzoeken																			
Schrijven PID																			
Herschrijven PID																			
Uitvoeren opdracht op basis van theorie																			
Uitwerken beroeps- product																			
Afstudeer- rapport schrijven																			

11 Bibliografie

- About LoadImpact*. (2017). Opgehaald van Load Impact AB:
<https://loadimpact.com/about>
- Ambler, S. W. (2004). *The Object Primer: Agile Model Driven Development with UML 2*. Cambridge: Cambridge University Press.
- Defining Technology, Inc. (2012, 05 29). *What is Middleware?* Opgehaald van Middleware.org:
<https://web.archive.org/web/20120629211518/http://www.middleware.org/whatis.html>
- Denscombe, M. (2004). *The Good Research Guide: For small-scale social research projects*.
- Gorton, I. (2011). *Essential software architecture* (tweede ed.). Springer.
- Grider, S. (2017, 02). *The Purpose of Boilerplate Projects*. Opgehaald van Udemy:
<https://www.udemy.com/react-redux/learn/v4/t/lecture/4224608>
- InvestorWords. (2017, 05 01). *What is Proof Of Concept? definition and meaning*. Opgehaald van InvestorWords:
http://www.investorwords.com/3899/proof_of_concept.html
- Korotya, E. (2017, 01 19). *5 Best JavaScript Frameworks in 2017*. Opgehaald van Hackernoon: <https://hackernoon.com/5-best-javascript-frameworks-in-2017-7a63b3870282>
- Kucheriavy, A. (2017). *Agile vs. Waterfall Website Project Management Methodologies*. Opgehaald van Intechnic:
<https://www.intechinc.com/blog/agile-vs-waterfall-website-project-management-methodologies/>
- Kumar, A., & Kumar, R. S. (2016). COMPARATIVE ANALYSIS OF ANGULARJS AND REACTJS. *INTERNATIONAL JOURNAL OF LATEST TRENDS IN ENGINEERING AND TECHNOLOGY*, 225-227.
- Laurent, A. M. (2004). *Understanding Open Source & Free Software Licensing*. O'Reilly Media.
- List of service names per module | Magento 2 Developer Documentation*. (2016). Opgehaald van Magento Devdocs:
http://devdocs.magento.com/guides/v2.0/rest/rest_endpoints.html
- Lodash*. (2017, 05). Opgehaald van Lodash: <https://lodash.com>
- Magento. (2016). *magento-cloud/CHANGELOG.md at master*. Opgehaald van GitHub: <https://github.com/magento/magento-cloud/blob/master/CHANGELOG.md#0740-beta10>
- Postman | Supercharge your API workflow*. (2017). Opgehaald van GetPostman:
<https://www.getpostman.com/>

Relationship to the World Wide Web and REST Architectures. (2004, 02 11).

Opgehaald van W3.org: <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest>

Rubin, K. S. (2012). *Essential Scrum: A practical guide to the most popular Agile process.* Addison-Wesley.

Sekaran, U. (2013). *Research Methods For Business, fourth edition.* John Wiley & Sons Inc.

siege(1): HTTP/HTTPS stress tester - Linux man page. (2017). Opgehaald van die.net: <https://linux.die.net/man/1/siege>

Vliet, H. v. (2008). *Software Engineering: Principles and Practice, third edition.* Chichester: John Wiley And Sons Ltd.

What is npm? / npm Documentation. (2017, 03 09). Opgehaald van npmjs: <https://docs.npmjs.com/getting-started/what-is-npm>

Wiley. (2016, 04 07). *What Is JavaScript?*Wiley. Opgehaald van Dictionary: <http://www.dictionary.com/browse/javascript>

12 Bijlagen

Tabel 5 Porjectrisico's

#	Risico/ gebeurtenis	Kans	Impact	Prioriteit	Preventief	Beheersmaatregel
1	Het is niet mogelijk om een frontend te maken op de backend van Magento 2.	Klein. De grote community die achter Magento zit heeft beschreven dat dit goed mogelijk is.	Groot. De doelstelling van de opdracht is weliswaar snel afgerond, maar voor de opdrachtnemer is het snel gedaan met het onderzoek.	Hoog. Wanneer het niet mogelijk is, zal snel omgeschakeld moeten worden naar een andere oplossing.	Er zal in het beginstadium zo snel mogelijk gekeken worden naar de mogelijkheden op dit gebied. Door er op tijd bij te zijn, kan er nog gestuurd worden in de afloop van de opdracht.	Om de andere opdracht, waarbij uitgezocht moet worden of het mogelijk is om een eigen frontend te plaatsen voor Magento's backend, niet in het nauw te drijven kan er een (met behoud van dezelfde theorie voor de proof of concept) gekozen worden om een stukje software <i>mockingdata</i> te laten genereren. Mockingdata is data dat niet echt uit een systeem komt, maar doet alsof het uit een vergelijkbaar systeem gehaald word. Mocht later toch blijken dat Magento de optie aanbiedt om de backend te scheiden van de frontend, kan deze mockingdata alsnog gemakkelijk vervangen worden door het daadwerkelijke systeem.

#	Risico/ gebeurtenis	Kans	Impact	Prioriteit	Preventief	Beheersmaatregel
2	Een experiment geeft verschillende resultaten bij dezelfde configuratie	Aanwezig. Een computer kan altijd afwijken van een nulstand	Middelmatig. De resultaten van het experiment zijn minder betrouwbaar	Hoog. Wanneer blijkt dat een experiment grote verschillen weergeeft, zal direct overgegaan moeten worden naar het de beheersmaatregel	Er wordt zo spoedig mogelijk bekeken of de metingen redelijk overeen komen onderling, in dezelfde nulstand.	Er kan besloten worden zogenaamde virtuele machines op te zetten die vanaf een nulstand altijd gelijk zijn aan elkaar. Deze worden na een iteratie gereset, waarna ze de volgende iteratie doen.
3	De proof of concept werkt niet online, maar slechts lokaal. Hierdoor kan LoadImpact niet gebruikt worden.	Klein. Het gebruikte framework is specifiek gemaakt voor het web.	Middelmatig. Er is een andere tool bekend (Siege, zoals gesteld in hoofdstuk 4.3.3 Siege) die ongeveer hetzelfde kan	Middelmatig. Er moet vanuit worden gegaan dat het niet online werkt, hoewel de kans groot is dat dit het wel doet.	Er zal, zodra er iets klaar is, zo spoedig mogelijk iets online gezet worden om snel te kunnen schakelen in het meetproces	Er wordt overgestapt naar een andere tool, genaamd Siege. Zie hoofdstuk 4.3.3 ("Siege")

#	Risico/ gebeurtenis	Kans	Impact	Prioriteit	Preventief	Beheersmaatregel
4	De tools die gebruikt worden zijn niet geschikt om juiste metingen te verrichten	Klein. Deze tools is specifiek gemaakt voor het stresstesten van webpagina's	Er kunnen geen metingen verricht worden met deze tool	Laag. Er zijn andere tools beschikbaar die ongeveer hetzelfde doen.	Wanneer er metingen verricht kunnen worden, moet dit zo snel mogelijk gedaan worden.	Er zal gekeken moeten worden naar een andere tool om de experimenten mee te doen.
5	De frontend kan niet worden gemaakt binnen de tijd	Klein. Er lijkt voldoende tijd vrijgemaakt voor de daadwerkelijke implementatie van de proof of concept	Er kan niet goed vergeleken worden in hoeverre de frontend sneller is dan de Magento frontend	Middelmatig. Hoewel het voor de opdrachtgever wellicht een minder valide antwoord geeft	Er zal ten minste een pagina eerst volledig werkend worden gemaakt. Mocht de andere pagina dan niet werken, is er ten minste iets werkend waarmee metingen gedaan kunnen worden.	Er kan teruggegrepen worden op een framework dat bekend is bij de opdrachtnemer. Op deze manier kan er alsnog gemeten worden. Een andere maatregel is om een bepaald deel niet functioneel te maken, en deze dan buiten de metingen te laten.

Tabel 6 Resultaten van de test van LoadImpact van de Magento 2 frontend (Engelstalig)

Page	Load zone	User scenario	Min	Avg	Max
Ajax Request #1	Frankfurt, DE (Amazon)	Magento 2	449.58m s	4.75s	12.83s
Ajax Request #2	Frankfurt, DE (Amazon)	Magento 2	414.92m s	4.45s	12.6s
Ajax Request #3	Frankfurt, DE (Amazon)	Magento 2	326.52m s	4.25s	14.33s
Block Loader #1	Frankfurt, DE (Amazon)	Magento 2	137.44m s	588.27m s	5.37s

Page	Load zone	User scenario	Min	Avg	Max
Block Loader #2	Frankfurt, DE (Amazon)	Magento 2	133.28m s	561.88m s	5.85s
Block Loader #3	Frankfurt, DE (Amazon)	Magento 2	142.92m s	573.55m s	6.25s
Block Loader #4	Frankfurt, DE (Amazon)	Magento 2	140.86m s	903.73m s	6.89s
Category Page: id 5	Frankfurt, DE (Amazon)	Magento 2	460.37m s	5.06s	14.92s
Category Page: id 65	Frankfurt, DE (Amazon)	Magento 2	496.72m s	5.46s	16.25s
Category Page: id: 251	Frankfurt, DE (Amazon)	Magento 2	507.25m s	507.25m s	12.04s
Homepage	Frankfurt, DE (Amazon)	Magento 2	437.58m s	6.57s	17.47s

Page	Load zone	User scenario	Min	Avg	Max
Product Page: id 125	Frankfurt, DE (Amazon)	Magento 2	429.03m s	5.21s	13.09s
Product Page: id 2	Frankfurt, DE (Amazon)	Magento 2	531.82m s	4.9s	12.94s
Product Page: id 32	Frankfurt, DE (Amazon)	Magento 2	473.54m s	4.96s	13.63s

Tabel 7 Resultaten van de test van LoadImpact van de ReactJS frontend (Engelstalig)

Page	Load zone	User scenario	Min	Avg	Max
Categories	Frankfurt, DE (Amazon)	ReactJS	541.73ms	8.71s	16.64s
Category Page: id 5	Frankfurt, DE (Amazon)	ReactJS	432.3ms	7.56s	13.63s
Category Page: id 65	Frankfurt, DE (Amazon)	ReactJS	362.55ms	7.54s	14.89s
Category Page: id: 251	Frankfurt, DE (Amazon)	ReactJS	333.06ms	7.71s	13.82s
Product Page: id 2	Frankfurt, DE (Amazon)	ReactJS	285.75ms	6.62s	12.41s
Product Page: id 32	Frankfurt, DE (Amazon)	ReactJS	302.42ms	6.49s	13.1s
Product Page: id 125	Frankfurt, DE (Amazon)	ReactJS	315.58ms	6.51s	12.98s