# Afstudeerscriptie

Diract-IT

hoofdletter

, en mag niet misschien als nieuwe zin

Slimmer applicatie contextualiseren van informatie, minder tonen

Algemeen onderzoeken verdedigen.

Proof-of-concept eigen kopjes

Sieger Falkena

500602792

Inhoud

[Afstudeerscriptie 1](#_Toc450681687)

[Inleiding 5](#_Toc450681688)

[Diract-it 5](#_Toc450681689)

[Concentrator 5](#_Toc450681690)

[WMS 5](#_Toc450681691)

[Probleemstelling 6](#_Toc450681692)

[Service Oriented Architecture 6](#_Toc450681693)

[Toegankelijk voor mobiele applicaties 6](#_Toc450681694)

[Bruikbaar voor meerdere teams 6](#_Toc450681695)

[Onderzoeksmethoden 6](#_Toc450681696)

[Doelstelling 6](#_Toc450681697)

[Wat is een goede front-end opzet voor diract-it? 7](#_Toc450681698)

[Wie zijn de gebruikers en waar hechten zij waarde aan? 7](#_Toc450681699)

[PACMAD 7](#_Toc450681700)

[Deelconclusie 8](#_Toc450681701)

[Wat zijn mobiele limitatie en hoe kunnen deze worden gecompenseerd? 9](#_Toc450681702)

[Limitaties 9](#_Toc450681703)

[Visueel 9](#_Toc450681704)

[Invoer 9](#_Toc450681705)

[Netwerk 10](#_Toc450681706)

[Hardware 11](#_Toc450681707)

[Voordelen 11](#_Toc450681708)

[Touchscreen 11](#_Toc450681709)

[Microfoon 11](#_Toc450681710)

[Camera 11](#_Toc450681711)

[Tactiel / locatie gebaseerde informatie 12](#_Toc450681712)

[Deelconclusie 12](#_Toc450681713)

[Hoe kunnen we de webapplicatie structureren om de mobiele toegankelijkheid te vergroten? 12](#_Toc450681714)

[Business rules 13](#_Toc450681715)

[Export- en importregels 13](#_Toc450681716)

[Producten en attributen 13](#_Toc450681717)

[Kanalen 13](#_Toc450681718)

[Details abstraheren 13](#_Toc450681719)

[Wat is nodeJs en waarom gebruikt iedereen het? 13](#_Toc450681720)

[Node.JS 13](#_Toc450681721)

[Dependency management 13](#_Toc450681722)

[Build-management 14](#_Toc450681723)

[Testen en ontwikkelondersteuning 14](#_Toc450681724)

[Integratie met visual studio 2015 14](#_Toc450681725)

[Conclusie 14](#_Toc450681726)

[Welke javascript frameworks sluiten het beste aan bij de vereisten? 15](#_Toc450681727)

[AngularJS 16](#_Toc450681728)

[Voordelen 16](#_Toc450681729)

[Nadelen 16](#_Toc450681730)

[Sencha ExtJS 17](#_Toc450681731)

[Voordelen 17](#_Toc450681732)

[Nadelen 17](#_Toc450681733)

[ReactJS 18](#_Toc450681734)

[Voordelen 18](#_Toc450681735)

[Neutraal 18](#_Toc450681736)

[Nadelen 18](#_Toc450681737)

[Deelconclusie 18](#_Toc450681738)

[Welke CSS framework past het beste bij concentrator? 19](#_Toc450681739)

[Waarom gebruiken ontwikkelaars een front-end framework? 19](#_Toc450681740)

[Selectie van frameworks 19](#_Toc450681741)

[Material Design 19](#_Toc450681742)

[Voordelen 19](#_Toc450681743)

[Speciaal voor mobiele webapplicaties 19](#_Toc450681744)

[Nadelen 19](#_Toc450681745)

[Bootstrap 19](#_Toc450681746)

[Voordelen 20](#_Toc450681747)

[Nadelen 20](#_Toc450681748)

[Metro-ui 20](#_Toc450681749)

[Zurb foundation 20](#_Toc450681750)

[Deelconclusie 20](#_Toc450681751)

[Hoe werkt een service oriented architecture? 21](#_Toc450681752)

[REST 21](#_Toc450681753)

[SOAP 22](#_Toc450681754)

[Backend webserver 22](#_Toc450681755)

[Deelconclusie 23](#_Toc450681756)

[Hoe werkt de authenticatie van het systeem? 24](#_Toc450681757)

[Conclusie 24](#_Toc450681758)

[Verklarende woordenlijst 24](#_Toc450681759)

[Bibliografie 24](#_Toc450681760)

# Inleiding

Dit onderzoeksrapport is geschreven naar aanleiding van mijn afstudeeronderzoek bij Diract-IT, verder genoemd als Diract.

## Diract

Diract-it is een software fabrikant die detailhandel ondersteund met software. Processen zoals distributie en productinformatie kunnen met de software van Diract versimpeld en geautomatiseerd worden, wat een voordeel voor de klanten van Diract is.

Diract was dicht gekoppeld aan de BAS-groep, een verkoopketen gericht op consumentenelektronica. De BAS-groep was de primaire afnemer van de systemen van Diract, tot en met het punt dat beide bedrijven in hetzelfde gebouw gehuisvest waren. Diract heeft meerdere innovatieve systemen aan BAS-groep geleverd, o.a. een mobiele kassa en omni-channel verkoop.

Binnen de organisatie van de BAS-groep is er nu een ruimte om zich op een bredere klantenbasis te oriënteren. Meerdere producten zijn toe aan een nieuwe versie, er is een druk om met opgedane ervaring een nieuw systeem te bouwen.

Alle Diract systemen zijn onderdeel van de Ceyenne suite en daarom makkelijk met elkaar te integreren.

## Concentrator

Concentrator is een systeem dat verantwoordelijk is voor het managen van productinformatie. Het Concentrator systeem verwerkt de binnenkomende productinformatie van fabrikanten of importeurs, en exporteert deze productinformatie naar webshops of fysieke winkels, maar kan ook exporteren naar andere systemen zoals warehouse management systemen.

De Concentrator is de centrale spil binnen de Ceyenne suite. Andere systemen van Diract zijn zeer afhankelijk van de informatie uit de Concentrator.

Producten bevatten veel informatie zoals maat, afbeeldingen, naam en/of kleur. Deze informatie is beschikbaar in verschillende contexten zoals taal, import- of exportlocatie.

## Warehouse Management System

WMS is een ander Diract product. Het is een product dat gericht is op het managen van een distributiecentrum (warehouse) en bijbehorende processen zoals ordermanagement en orderpicking met barcodescanners. Het WMS maakt veel gebruik van de productinformatie uit de concentrator.

## Andere producten van Diract

Diract heeft ook nog andere producten zoals een kassasysteem en een business-intelligence afdeling. Ook deze zijn afhankelijk van de Concentrator.

# Probleemstelling

Voor de volgende versie van concentrator willen de ontwikkelaars de functionaliteit van de webapplicatie ook op mobiele appraten goed aanbieden. In de huidige versie is de webapplicatie maar matig toegankelijk voor mobiele gebruiker.

De huidige versie (concentrator 2) is al een single-page applicatie, voor de volgende versie van concentrator willen ze dat zo houden. Diract wil in de nieuwe concentrator ook gebruik kunnen maken van nieuwe HTML5 capaciteiten.

Binnen Diract is er een wens alle producten makkelijker aan elkaar te koppelen. Bedrijfsbreed is er besloten alle nieuwe producten en productversies een service laag te geven die beide door eindgebruikers als andere systemen als centraal aanspreekpunt voor de applicatie kan worden gebruikt.

De wens vanuit Diract is dat functionaliteit kan worden ingezet voor meerdere applicaties. Niet product specifieke onderdelen moeten modulair worden geprogrammeerd met de bedoeling deze te kunnen gebruiken in meerdere producten.

# Doelstelling

Doel is het onderzoeken van een front-end opzet voor de Concentrator en de WMS. Deze moet aansluiten op de wensen en noodzaak van Diract.

De werking wordt aangetoond door middel van een proof-of-concept. Back-end code moet in C# worden geprogrammeerd worden en gebruik maken van de Microsoft producten die Diract gebruikt. Bijbehorend is documentatie voor het bouwen en draaien van de applicatie.

Er wordt specifiek niet gezocht naar een nieuw platform voor de business logica of database.

# Onderzoeksmethoden

Er is gekozen voor een combinatie van onderzoekmethoden. Er wordt gebruik gemaakt van (informele) interviews binnen Diract om te onderzoeken waar de waarde voor het bedrijf ligt. Daarna wordt er gebruik gemaakt van deskresearch om te onderzoeken hoe deze waarde het beste gerealiseerd kan worden.

Het geheel wordt praktisch aangetoond met een proof-of-concept. Het proof-of-concept is een minimale systeemopzet die gemaakt is op basis van de deskresearch, daarmee de realiseerbaarheid aantoont.

# Wat is een goede front-end opzet voor Diract?

# Wie zijn de gebruikers en waar hechten zij waarde aan?

subtitel

## PACMAD

People At the Center of Mobile Application Development

Mobiele gebruiksvriendelijkheid is anders dan gebruiksvriendelijkheid voor desktopsystemen, en veel problemen worden vaak over het hoofd gezien. Harrison et. al. hebben een metaonderzoek gedaan naar andere onderzoeken die eigenschappen voor mobiele gebruiksvriendelijkheid classificeerden. (Harrison, et al., 2013)

Aantonen waarom geldig onderzoek.

Volgens Harrison zijn de volgende factoren van invloed op mobiele gebruiksvriendelijkheid.

* De gebruiker
* De taken die de gebruiker wil ondernemen
* De context in welke de gebruiker deze taken wil uitvoeren

In het geval van Diract is de gebruiker een *product information manager* of *order manager*. Zij zijn respectievelijk verantwoordelijk voor het bijhouden van productinformatie en order afhandeling. Van beide kan een basiskunde van computers worden verwacht. Tussen *product information managers* en *order managers* is een overlap in het takenpakket, en de functies worden in sommige organisaties door dezelfde werknemer gedaan.

De taken voor beide gebruikers ligt in het onderhouden en beheren van de data in het informatiesysteem, ondersteund door professionals van Diract. Voorbeelden zijn het inzien en aanpassen van productinformatie, het activeren en deactiveren van producten en het oplossen van problemen bij orders met missende informatie.

De nieuwe concentrator kent verschillende contexten; de applicatie kan door de huidige opzet momenteel alleen gemakkelijk met een desktop worden gebruikt. De organisatie wil dat de concentrator ook dynamisch op de werkvloer kan worden gebruikt met een tablet of smartphone.

Een ander context is dat Diract als bedrijf internationaal wil kunnen opereren en services aan wil bieden. De huidige webapplicatie is Engelstalig, maar het is niet vanzelfsprekend dat buitenlandse gebruikers de Engelse taal machtig zijn.

In het PACMAD model hebben deze contexten invloed op de volgende attributen;

* Effectiviteit
  + Het kunnen uitvoeren van taken. Dit heeft veel te maken met het accuraat implementeren van user-stories. uitdiepen
* Efficiëntie
  + De snelheid en accuraatheid van taken
* Bevrediging
  + De subjectieve gebruikerservaring
* Leerbaarheid
  + De snelheid waarin de gebruiker leert zijn taken in het systeem te vervullen.
* Onthoudbaarheid
  + De snelheid waarin een gebruiker vergeet hoe hij zijn taken binnen het systeem kan doen
* Fouten
  + De fouten die de gebruiker veroorzaakt, en de bugs die de ontwikkelaar introduceert. Fouten zijn onvermijdelijk, maar de mate waarin feedback wordt gegeven en de snelheid waarmee bugs worden verholpen zijn cruciaal voor een goed gebruikersgemak
* Cognitieve Ballast
  + De concentratie die de applicatie vereist.

De eerste drie attributen (effectiviteit, efficiëntie en bevrediging ) zijn het meest onderzocht en het meest van invloed op gebruikerservaring. Het gaat hier over of de gebruiker zijn taken kan uitvoeren en daarnaast hoe snel hij zijn taak kan uitvoeren.

Het vinden van bepaalde producten naar criteria is een vast onderdeel in het takenpakket van *product information managers*. Dit is in de huidige concentrator niet helemaal uitgewerkt, maar komt de efficiëntie van de werknemer zeer ten goede.

Prefetching is een van de technieken die de efficiëntie ten goede komt. Door veelgebruikte functionaliteit automatisch op de achtergrond te laden, is de laadtijd nagenoeg nul. Eindgebruikers hoeven dan bijna nooit te wachten op het laden van data.

De andere attributen; (leerbaarheid, onthoudbaarheid, fouten en cognitieve ballast) zijn in de meeste studies alleen subjectief gemeten. Ondanks het gebrek aan informatie over deze attributen is het wel mogelijk om hier rekening mee te houden.

De leerbaarheid en onthoudbaarheid van een applicatie heeft primair te maken met het intuïtief opdelen van alle gebruikerscontexten (product, trainingen, etc. ).

Het is ook van cruciaal belang om foutafhandeling op een intuïtieve manier aan te kunnen bieden. Een voorbeeld is de validatie van velden voor het verzenden naar de server, en aangeven of het een gebruikers of serverfout is. Dit is ook in het belang van support omdat zij verantwoordelijk is voor serverfouten en ondersteuning van gebruikers.

De Cognitieve Belading is ook primair een geval van intuïtief ontwerp. Omdat de nieuwe software dynamisch op de werkvloer moet worden gebruikt is het belangrijk de cognitieve belading laag te houden. De primaire wijze om de cognitieve belading laag te houden is het maken van een navigatie boom die niet al te diep

## Deelconclusie

In de nieuwe Concentrator zijn er nieuwe gebruikscontexten aanwezig. De applicatie moet nu ook beschikt. Meertaligheid is in het proof-of-concept uitgewerkt. Deze context van mobiel gebruik wordt in het volgende hoofdstuk beter behandeld.

Multilanguage applicatie zijn die zowel op desktop als mobiele platformen goed toegankelijk is.

De primaire gebruikers van Concentrator zijn product information- en ordermanagers die vooral om de effectiviteit en efficiëntie van hun eigen taken geven. De effectiviteit van de manager is in grote mate afhankelijk van de implementatie van de user stories.

De efficiëntie van de manager is in grote mate afhankelijk van de onderliggende techniek. De gebruiker wil liever niet wachten op het laden van de webapplicatie. Voor de efficiëntie van de gebruiker worden gedacht aan het implementeren van zoekfunctionaliteit over alle velden. Uitdiepen met meer informatie.

Foutafhandeling kan worden bewerkstelligd met het valideren van velden en pop-ups die fouten aangeven. Ongeacht gebruiker- of systeemfout moet het systeem duidelijk aangeven wat er verkeerd is gegaan en liefst ook waarom. Voor gebruikers is feedback cruciaal bij het corrigeren van een fout, en voor ontwikkelaars en support maakt goede foutafhandeling het mogelijk sneller te kunnen reageren. Daarnaast zijn een paar velden van validatie voorzien.

Voor de leerbaarheid en onthoudbaarheid van de applicatie is het nodig een aan te bieden, aan de hand van de taken van de gebruiker. Deze kan digitaal worden verstrekt in de webapplicatie zelf en/of via documentatie en trainingen.

Het intuïtief houden van de applicatie is een van de primaire doelen.

## Proof-of-concept

In het proof-of-concept is er een veilige aanname gedaan dat er in ieder geval voor producten CRUD *(Create, Read, Update, Delete)* functionaliteit gaat worden gebruikt. Dit zijn simpele taken zoals het lezen, toevoegen, aanpassen en verwijderen van producten en gerelateerde resources. Dit is gedaan naar de effectiviteit van de applicatie. Er is ook een lijst aan producten aangemaakt waarover gezocht kan worden.

Nieuwe contexten zoals het aanbieden van de applicatie in meerdere talen (Engels, Nederlands) is verwerkt door het maken van een module die een key-value store leegtrekt. Dit is in de conclusie van het volgende hoofdstuk beter beschreven.

Producten hebben gerelateerde data zoals attributen en taalcontexten die geselecteerd moeten worden. Op de invoervelden van deze gerelateerde dat word er automatisch een aantal opties getoond, die met (tijdelijke) invoer gefilterd kan worden. Dit onderdeel staat over in het volgende hoofdstuk beschreven.

Op het gebied van foutafhandeling is het beter om te voorkomen dan te genezen; invoervelden zijn voorzien van eisen en geven aan dat de invoer incorrect is. Voor daadwerkelijke fouten en uitzonderingen wordt er een pop-up getoond die aangeeft welk request is foutgegaan, de reden waarom de request is mislukt en enige statuscode.

# Wat zijn mobiele limitatie en hoe kunnen deze worden gecompenseerd?

De limitaties van mobiele apparaten tegenover het traditionele desktop met toetsenbord en muis zijn significant, en moeten worden onderzocht.

## Limitaties

### Visueel

Visueel hebben mobiele apparaten een over het algemeen kleiner beeldscherm en meer pixels/cm2. Dit maakt dat elementen (knoppen en tekst) die als goed bruikbaar op desktop worden ervaren, als te klein worden ervaren op mobile. Vice-versa kunnen tekst en knoppen voor mobile als te groot worden ervaren in een desktop context.

Een van de oplossingen is het gebruik van Responsive Web Design. RWD geeft de ontwikkelaar de mogelijkheid elementen in verschillende maten aan te bieden, afhankelijk van de afmetingen van het beeldscherm. Het beeldscherm wordt opgedeeld in 12 kolommen, en de grootte van de elementen kan per combinatie beeldscherm en afmeting worden aangegeven.

Er zijn meerdere frameworks waarmee RWD gemakkelijk te implementeren is. Voornaamste spelers zijn Twitter bootstrap, Foundation, Metro-UI en Material Design. Qua RWD is er weinig verschil tussen de frameworks te bekennen. Alle frameworks gebruiken een grid van 12 kolommen en de volgende beeldschermafmetingen;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Voorbeeld apparaat | Afmetingscode | Minimum wijdte | Maximum wijdte | Container |
| Kleine telefoon | XS | -∞ pixels | 768 pixels | Auto |
| Tablet | SM | 768 pixels | 992 pixels | 750 pixels |
| Laptop, tablet in landschap mode | MD | 992 pixels | 1200 pixels | 970 pixels |
| Desktop | LG | 1200 pixels | ∞ pixels | 1170 pixels |

Bij elke beeldschermafmeting is het responsive grid 12 kolommen breed. Als een combinatie van elementen breder is dan 12 kolommen, worden de elementen die er buiten vallen op een volgende rij gezet.

### Invoer

Mobiele apparaten beschikken over een touchscreen als primair invoerapparaat. Het touchscreen functioneert beide als muis en toetsenbord. Tussen 2012 en nu (2016) is er niet veel veranderd aan de invoercapaciteiten van mobiele apparaten.

|  |  |  |
| --- | --- | --- |
|  | **Mouse** | **Fingers** |
| **Precision** | High | Low |
| **Number of points specified** | 1 | usually 1 2–3 with multi-touch |
| **Number of controls** | 3: left/right button, scroll wheel | 1 |
| **Homing time?** | Yes | No |
| **Signal states** | Hover, mouse-down, mouse-up | Finger-down, finger-up |
| **Accelerated movements** | Yes | No |
| **Suitable for use with big desktop monitors (30-inch or more)** | Yes, because of acceleration | No: arm fatigue |
| **Visible pointer/cursor** | Yes | No |
| **Obscures view of screen** | No, thus allowing for continuous visual feedback | Yes |
| **Suitable for mobile** | No | Yes: nothing extra to carry around |
| **Ease of learning** | Fairly easy | Virtually no learning time |
| **Direct engagement with screen and "fun" to use** | No: an indirect pointing device | Yes |
| **Accessibility support** | Yes | No |

Bron: <https://www.nngroup.com/articles/mouse-vs-fingers-input-device/>, 10-4-2012

**Touchscreen als muis**

Het touchscreen doet niet onder de muis, maar is wel sterk anders in gebruik. Vingers zijn minder precies dan een muiscursor, en nemen daarnaast het scherm in beslag. Daartegen is de touchscreen wel intuïtief in gebruik, omdat ze de meest natuurlijke impuls van de vinger weet te gebruiken.

**Touchscreen als toetsenbord**

Een mobiele telefoon is minder geschikt voor de invoer van lange stukken tekst, voornamelijk omdat de aanraaksensatie van een toetsendruk ontbreekt en omdat de knoppen over het algemeen kleiner zijn dan die van een fysiek toetsenbord. Beginnende touchscreen typisten halen een gemiddelde van 10 wpm op een (klein) touchscreen, vergeleken met 41,2 wpm voor een normaal toetsenbord. **Studie terugvinden**

Typefouten zijn makkelijker te maken op een touchscreen dan een , voornamelijk vanwege de kleine afmetingen van het touchscreen toetsenbordje.

De oplossing voor slechte tekstinvoer is het automatisch aanvullen van gedeeltelijke informatie. Door de gebruiker automatisch aanvullingsopties te geven hoeven er minder toetsen te worden aangeslagen.

### Netwerk

Een snelle internetconnectie is voor mobiele apparaten minder vanzelfsprekend dan een desktop. Telefoons zijn afhankelijk van snelle draadloze netwerken die niet altijd en overal beschikbaar zijn. Zelfs wifi kan overbezet worden door de bandbreedte van de radiofrequenties.

Een lage bandbreedte in combinatie met een grote applicatie kan tot een trage gebruikerservaring (Effectiviteit) leiden. Wachten op het laden van de applicatie kan voor frustraties zorgen. Dit is een grote frustraties voor *productmanagers* die efficiënt en effectief een aantal operaties willen uitvoeren.

Het dynamisch inladen van dependencies kan laadtijden verkorten. Voornamelijk dependencies met een significante grootte en situationeel gebruik hoeven niet vanaf het begin ingeladen te worden.

Door dependencies en data vooraf te laden kunnen wachttijden worden verminderd naar nagenoeg nul. Door de dependencies van modules net-op-tijd in te laden is het mogelijk de initiële laadtijd naar nagenoeg nul te brengen, en wint de gebruiker aan Efficiëntie. In bepaalde omstandigheden kan ook alvast data worden ingeladen; bijvoorbeeld het inladen van een twee pagina in een overzicht van zoekresultaten.

### Hardware

Mobiele telefoons beschikken over het algemeen over minder geheugen dan een desktop. Als er te veel geheugen wordt gebruikt, wordt het apparaat traag en onbruikbaar, wat ten koste gaat van de efficiency. Javascript data-objecten worden automatisch opgeschoond, maar het is niet het geval dat een javascript ontwikkelaar luklaak objecten in het geheugen kan blijven laden.

Garbage-collectors werken door elke object en gerelateerde objecten in een boom te stoppen. Op het moment dat een object vanuit de root (document) niet meer bereikbaar is, wordt het object uit het geheugen verwijderd. Het is in barebone javascript behoorlijk makkelijk om objecten aan te maken en nooit te verwijderen.

Een goed framework zoals kan een applicatie architectuur te forceren. Door een architectuur te forceren (MVVC) kan het automatisch relevante data-objecten inladen en is het voor de garbage-collector duidelijk welke objecten verwijderd kunnen worden.

## Voordelen

De meeste smartphones en tablets komen met capaciteiten die niet standaard beschikbaar zijn op een normale desktop. Het meest overduidelijke voordeel is de transporteerbaarheid van mobiele apparaten.

### Touchscreen

Touchscreens zijn intuïtief in gebruik. Bijna alle touchscreens komen met de capaciteit voor Multi Touch. Touchscreens zijn zeer goed in herkennen van gestures (gebaren). Dit zijn relatief grove bewegingen die dingen bewerkstelligen. Denk bijvoorbeeld aan het openen van een menu door

### Microfoon

De meeste tablets en smartphones zijn uitgerust met een microfoon. Deze kan gebruik maken van nieuwe ontwikkelingen in het gebied van spraakherkenning. De toepassing van spraakherkenning is niet triviaal, de toepasbaarheid op de werkvloer is niet ideaal, maar het is in de praktijk mogelijk.

### Camera

Mobiele camera’s zijn steeds beter geïntegreerd met mobiele browsers.

Eindgebruikers van Concentrator en WMS werken veel met barcodes, en deze kunnen door bepaalde libraries live of vanuit een afbeelding worden gelezen.

Een camera met goede resolutie kan worden gebruikt om teksten te herkennen. De informatie in een stuk tekst kan met een computer worden ingelezen en als invoer worden gebruikt. Dit zou de moeizame invoer van tekst op een touchscreen gedeeltelijk kunnen vervangen.

### Tactiel / locatie gebaseerde informatie

Ik heb geen toepassing voor de tactiele en locatie-gebaseerde informatie kunnen bedenken in enige context van de opdracht. **Veel te onnauwkeurig om te gebruik RFID tags, andere technieken.**

### Uitdiepen over handzaamheid van mobiele applicatie

## Uitgewerkt in proof of concept

## Deelconclusie

Een mobile platform heeft een flink aantal nadelen tegenover een desktop, voornamelijk qua overzichtelijkheid en invoer van informatie. Het is minder effectief en efficiënt om met een telefoon of tablet de applicatie te beheren. Dit leidt tot frustraties bij de eindgebruikers (zie hoofdstuk 1).

Een Responsive Web Design framework kan de applicatie overzichtelijk maken door de inhoud van de applicatie te schalen naar het formaat van het platform. Er zijn meerdere platformen getest, en zijn qua ondersteuning van RWD allemaal ongeveer het zelfde. Er is geen sluitende beslissing gemaakt, omdat dit meer in de handen van een eventuele webdesigner ligt. Voor de praktische, korte term is er gekozen voor bootstrap vanwege de hoge native community ondersteuning van Bootstrap.

De invoer van data is een van de meest frustrerende aspecten van mobiele applicaties. Bepaalde functionaliteit, zoals zoeken en automatisch aanvullen van velden, kan er met minder invoer meer worden gedaan. Dit is uitgewerkt in het proof-of-concept. Meer esoterische functionaliteit zoals het gebruik van Optical Character Recognition heeft voor Diract weinig prioriteit op de korte term.

Stukje barcode

Om de telefoon of tablet wél doeltreffend platform te gebruiken, moeten de krachten van mobiele platformen worden uitgebuit. Deze krachten liggen voornamelijk bij intuïtief gebruik van het touchscreen, en de camera.

**Mobiele applicatie voor bepaalde interacties.**

# Hoe kunnen we de webapplicatie structureren om de mobiele toegankelijkheid te vergroten?

De context waarin de gebruiker zich bevind moet grafisch duidelijk worden weergegeven. Dit kan op veel verschillende manieren; het tonen van een klein icoontje is er een van.

Momenteel is de meeste informatie getabt aanwezig, maar dit is niet per sé de meest gebruiksvriendelijke manier (onderzoek terugvinden). Onderzoek toont aan dat er een subjectieve voorkeur is voor een ‘plat’ geheel, terwijl gebruikers 0,3 seconden effectiever zijn in een gelaagd ontwerp. Gebruikers krijgen een subjectief gevoel van overzicht in een platte applicatiestructuur, maar kunnen hun taken net iets sneller uitvoeren in een gelaagd ontwerp. Ik ga er van uit dat de subjectieve

## Details abstraheren

# Wat is NodeJs en waarom gebruikt iedereen het?

Elke technologie komt met een bepaalde ontwikkelomgeving, javascript komt met Node.

Javascript blijft een steeds grotere rol spelen in de interactie tussen het document en browser. Waar voor 2008 javascript voornamelijk een tool was voor het valideren van forms, wordt nu de hele form met javascript verstuurd.

De voornaamste reden voor de explosie aan javascript is de open-source V8 engine die voor alle platformen beschikbaar is. De V8 engine compileert javascript naar native-code.

Javascript is een laagdrempelige taal, maar mist veel functionaliteit die voor programmeurs als onmisbaar worden ervaren.

* Debuggen van javascript gebeurt in de browser.
* Javascript is geen object-georiënteerde taal
  + Kent geen klassen
  + Werkt voornamelijk met een Prototype klassen die gekopieerd en aangepast kunnen worden.
* De functies en eigenschappen van een javascript object kunnen dynamisch worden toegevoegd en verwijderd, wat de betrouwbaarheid van javascript niet ten goed komt.
* Javascript kent geen encapsulatie en kan moeilijk abstractielagen verbergen.
* Javascript heeft een zwak en dynamisch typesysteem.
* Javascript kent maar een enkele scope; de functiescope.
* Javascript kent geen multithreading of multiprocessing in de browser\*

De kernset aan functionaliteit binnen javascript is ook zeer beperkt vergeleken met programmeertalen als C# of java. Javascript is gebaseerd op functioneel programmeren.

Dit maakt het moeilijk goed onderhoudbare javascript te maken en over langere tijd te onderhouden. Javascript ontwikkelingen hebben een noodzaak gevonden voor het gebruik van verschillende buildtools. Ook deze buildtools zijn in javascript uitgevoerd zodat ontwikkelaars die enkel javascript-ontwikkelaars hun eigen builds kunnen beheren.

## Node.JS

NodeJs is een javascript engine gebaseerd op de V8 javascript engine die systeemresources beschikbaar maakt.

NodeJs heeft nog steeds geen multithreading capaciteit, maar maakt in plaats daarvan gebruik van multiprocessing en asynchrone callbacks. Webservers gebaseerd op NodeJS maken gebruik van een enkele thread voor het inlezen en monitoren van requests, en sub-processen die systeemresources aanspreken.

Deze architectuur is langzaam overgenomen door de nieuwe generaties webservers, en is niet meer uniek aan NodeJs webservers. Nieuwe versies van bijvoorbeeld ASP.Net (Core 1) gebruikt dan ook een kleine hoeveelheid request-handlers(1-2) en een pool worker threads.

NodeJS is de defacto omgeving voor ontwikkeling in Javascript. Het draait snel en komt met een impressief arsenaal aan functionaliteit en buildmanagement in een commandline.

## Dependency management

Dependency management versimpelt het proces van deployment naar enige servers. Dependencies zijn over het algemeen groter dan de applicatie zelf, wat deploy proces vertraagd. Door een dependency manager toe te passen kan een webserver zelf de dependencies binnenhalen en op de juiste plaats neer zetten.

Node Package Manager (NPM) is een van de grotere NodeJs repositories. Het bevat voornamelijk libraries voor NodeJs zelf, en in mindere mate dependencies voor de client.

Bower is de andere grotere NodeJS repository. De Bower repository bevat voornamelijk font-end libraries.

Deze repositories zijn onbeheerd, en er is nogal wat sprake. Er zijn nogal wat instabiele en onbeheerde packages te vinden, en projecten in de repositories hebben zelf dependencies op andere projecten.

Persoonlijk denk ik dat bower en NPM niet de uiteindelijke oplossingen zijn voor dependencymanagement, en dat er vast en zeker een nieuwe generatie zal opkomen met betere oplossingen.

## Karma frameworks

Het Karma testframework is bedoelt om verschillende geautomatiseerde testparadigma’s zoals unit-testing en … testing te implementeren. De karma stack bestaat uit drie onderdelen; karma, een aanname framework en een browser. Het is hiermee mogelijk de functionaliteit op meerdere browsers te testen.

Ik ben vast en zeker dat javascript testen waarde oplevert, maar veel javascript code is simpelweg ontestbaar. Pas bij een grotere hoeveelheid gebruikers zou het waardevol zijn unittests door het systeem te voeren.

## Transpilers

Compilers compileren programmacode naar machinetaal. Transpilers transpileren een superset van javascript naar javascript. Transpilers lossen problemen op waarbij libraries eigenlijk te kort schieten.

Javascript wordt het meeste zonder tranpiler gebruikt, en goede javascript ontwikkelaars gebruiken bijna exclusief javascript. Coffeescript en Dart vallen voornamelijk uit de gratie omdat ze niet of weinig overeenkomsten hebben met javascrip.

Typescript bied een hoop nieuwe functionaliteit; maar de voornaamste toevoegingen zijn interfaces, een harder typesystem en generics. Deze functionaliteit is zeer handig bij het ontwikkelen van een complexere javascript applicatie die meerdere jaren aan uitbreidingen tegemoet moet komen.

## Build-management

Alhoewel er tijdens het bouwen code wordt gecompileerd, moeten dependencies wel op een gedefinieerde plaats. Handmatig kopiëren is tijdrovend en foutgevoelig.

Gulp is een buildmanager voor nodejs. Het bestaat uit taken die automatisch kunnen worden uitgevoerd. Buildstappen zoals “clean”, ”build” of ”install” kunnen met gulp worden geautomatiseerd. Ook kan gulp de ontwikkelaar ondersteunen door bijvoorbeeld gedeeltelijke “builds” te doen als de ontwikkelaar een bestand opslaat.

## Overige tooltjes

Underscore.js is een setje breed-toepasbare functies. Het implementeert functies als “foreach” en bepaalde datastructuren als een iterator en memoization en het casten van objecten naar arrays.

Require.js maakt het mogelijk dependencies dynamisch te laden. Dit kan handig zijn in het geval dat de totale grootte van de applicatie significant wordt.

Quagga.js maakt het mogelijk de camera van de applicatie als barcode scanner te gebruiken.

## Integratie met Visual Studio 2015

Visual Studio komt met integraties voor NodeJS en voornamelijk Typescript. Events uit het VS15 bouwsysteem kunnen automatisch gulp-tasks laten uitvoeren.

Daarnaast heeft VS15 Typescript integraties waarbij onder andere de linting en code-completions voor typescript kunnen worden aangeboden.

## Conclusie

Javascript is een volwaardige programmeertaal, en om als ontwikkelaar efficiënt en leuk te ontwikkelen is een ontwikkelomgeving onmisbaar. NodeJS bied een relatief compleet (maar veranderlijke omgeving) om javascript in te ontwikkelen. NodeJs is daarnaast goed ondersteund door VS15.

Gulp wordt als buildmanager door VS15 gebruikt. Andere keuzes (grunt, etc.) niet. De keuze is daarmee snel gemaakt.

Hoewel Bower en NPM repositories waarschijnlijk ooit op de schop gaan, is het toch belangrijk de hoeveelheid javascript dependencies met een tool te beheren. Repositorymanagers komen met versiebeheer en kunnen upgrades van frameworks versimpelen. De kwaliteit van bepaalde libraries moet per library worden bepaald, en de noodzaak moet ook per library worden bepaald. Bepaalde dependencies zijn zeer integer en voegen veel toe aan het ontwikkelproces. Require

Karma, en soortgelijke test-frameworks zijn moeilijk te integreren met frameworks, en vereisen een specifieke vorm van programmeren.

De keuze om bepaalde libraries te gebruiken komt neer op twee vragen;

Moet hier een library voor worden gebruikt?

Kan een andere library dit al?

Door de complexiteit van de applicatie is het aan te raden Typescript te gebruiken. Het gebruik van Intellisense voorkomt kleine foutjes die programmeurs maken, en maakt het makkelijk componenten te onderzoeken. Daarnaast helpt het gebruik van types het correct programmeren van de applicatie met gedefinieerde methodes.

## In het proof-of-concept

In het Proof-of-concept zijn er meerdere NodeJs functionaliteiten gebruikt.

Er is in het proof-of-concept geen gebruik van typescript gemaakt, maar dit zou eigenlijk wel moeten. De reden voor het niet implementeren was omdat er intern al een ander proof-of-concept is gemaakt dat Typescript goed in het licht zette.

De keuze voor buildsysteem viel op wat Visual Studio uit de doos al goed ondersteunde. Gulp is een flexibele buildmanager die met een script kan worden aangestuurd. De volgende taken zijn geautomatiseerd;

* (gedeeltelijk) Bouwen van javascript applicatie
* Injectie van javascript code in index.html
* Opschonen van de webroot
* Kopiëren van assets en applicatie
* Detecteren van het opslaan van codeveranderingen
  + Bouwen na detecteren van veranderingen
  + Automatisch herladen van pagina na codewijzigingen

# Welke javascript frameworks sluiten het beste aan bij de vereisten?

Er valt niet aan een framework te ontkomen. Om goed om te gaan met verschillende functionaliteit moet er een keuze worden gemaakt naar de verschillende frameworks. Het gaat niet zozeer om de basisfunctionaliteit.

Enkel CSS is niet genoeg om de website responsive te maken, een javascript library is verantwoordelijk voor het mogelijk maken van de interactiviteit en staat veranderingen. Deze library moet binnen een bastaand ecosysteem passen, welke soms behoorlijk expansief kan zijn.

Bovenop het basisframework draait dan een library en een CSS die verantwoordelijk voor het uiterlijk van de webapplicatie. Er zijn verschillende opties, maar er is een voorselectie gemaakt.

Vanuit de business-case is er een sterke neiging naar een bedrijfbreed gedragen oplossing voor de frontend, en er is al een PoC voor angular opgezet.

Er zijn vanuit het team en eigen inzicht verschillende criteria aan het framework gesteld;

* Het moet een GRID (uitgebreide tabel) ondersteunen
* De ontwikkeling van het framework moet stabiel zijn
* Het moet responsive libraries ondersteunen
* Definieert een architectuur
  + Bied een alternatief aan DOM manipulaties.
* Komt met ingebouwde functionaliteit
* Is goed te gebruiken voor back-end developers

De wens is naar een MVC of MVVC architectuur te gaan.

## AngularJS

Groot ecosysteem van google. Angular1 was een van de eerste gratis javascript frameworks. De ontwikkelaars van Angular zijn in 2010 opgekocht door Google.

Angular1 komt met een groot scala aan functionaliteit; onder andere het beheren van cookies, filters, en object-bindingen.

Angular2 is een herschreven versie van Angular1, en is momenteel in beta.

### Voordelen

Angular is gratis

On-officieel het SPA-framework van google, en word gefinancierd uit de budgetten van google.

Angular is het meest toegepaste framework, en heeft de meeste libraries van derde partijen. Functionaliteit zoals GRIDS, touch en integraties.

Angular is binnen Diract al eerder successvol gebruikt voor een proof of concept.

Angular is een echt framework, met veel beschikbare functionaliteit.

### Nadelen

Angular 2 is een herschreven versie van angular 1, en bijna alle functionaliteit van Angular 1 gaat veranderen. Angular2 is op het moment van schijven net in beta, en de meeste Angular1 community packages zijn nog niet overgeport naar Angular2.

Angular1 is zeer breed opgezet, en bevat overal functionaliteit voor. Sommige modules zijn echter sterk ondermaats, zoals de routing module. Er zijn wel alternatieven voor deze modules.

## Sencha ExtJS

De eerste versie van Ext.js kwam in 2007 uit en was daarmee de eerste. Het is een zeer sterk platform voor het ontwikkelen van desktop-achtige webapplicaties. Slepen van elementen is makkelijk definieerbaar. De voorbeelden met wat

### Voordelen

Meest uitgebreide GRID, met grafiekjes in de vakken.

Professionele ondersteuning bij aankoop.

Heel goed in het maken van webapplicaties die op een desktop-applicatie lijken.

### Nadelen

Van nature niet responsive. Er is een splitsing in Ext.JS 6 waarbij alle geavanceerde elementen (GRID, layout manager, mooie elementen) op een niet-responsive manier zijn gemaakt.

De voorbeelden zijn onleesbaar zonder extensieve kennis van Ext.JS .

Kost 9000 tot 14000 euro.

Heeft bijna geen community met libraries en toevoegingen.

Voorbeelden zijn niet bepaald verhelderend.

## ReactJS

Onderdeel van het facebook FLUX ecosysteem. Flux kwam in 2014 op de markt, maar is pas recentelijk opgepikt door nieuwe projecten.

### Voordelen

Kleine losstaande onderdelen maken het mogelijk een kleine, snel laadbare applicatie op te leveren.

Veel functionaliteit van derde partijen.

Ook gratis.

Onofficieel het framework van en voor facebook.

### Neutraal

Onderhouden door facebook.

### Nadelen

Tijdens het onderzoek brak ReactJs door een van de dependencies. Left-pad was een stukje javascript dat een string naar een bepaalde grootte bracht door teken aan de linkerhand in te voeren. Het was een indicatie van de kwaliteit van facebook ontwikkelaars.

ReactJs betreft enkel views, andere functionaliteit (controllers, models) moeten via andere FLUX onderdelen worden geregeld.

De voorbeelden zijn simpel en helder, maar niet optimaal. Templates staan in de javascript controllers, en er weinig sprake van scheiding tussen views en controllers.

## Deelconclusie

Javascript is een zeer dynamische taal, en voor niet front-enders niet direct inzichtelijk. De keuze is gevallen voor Angular1 omdat deze een duidelijke architectuur oplegt. De ontwikkelaars zijn geen javascript specialisten, en zijn primair geïnteresseerd in het ontwikkelen van backend functionaliteit.

Daarnaast is er out-of-the-box veel functionaliteit in Angular beschikbaar, in tegenstelling tot andere frameworks die meer op integraties van andere producten berusten (react). Dit is niet makkelijk inzichtelijk te maken, en een back-end ontwikkelaar wil niet zoeken welke dependency welke functionaliteit bevat.

Ext.JS is een zeer mooi framework, maar de kracht van Ext ligt overduidelijjk niet bij responsive design. Het gebruikt Bootstrap voor de responsive gedeeltes, en de paradepaardjes uit de voorbeelden zijn niet responsive. Ext is wel prachtig mooi, maar de meerwaarde voor Ext is niet toepasbaar op mobiele platformen.

Daarnaast ligt het al bijna vast dat andere teams binnen diract Angular gaan gebruiken. De kennisdrempel zal hiermee verlaagd worden, waardoor bijvoorbeeld inhouse ontwerpers ook makkelijk aan de front-end van de concentrator kunnen sleutelen.

# Welke CSS framework past het beste bij concentrator?

CSS frameworks zijn verantwoordelijk voor de uitstraling en overzichtelijkheid van de front-end

## Waarom gebruiken ontwikkelaars een front-end framework?

Er wordt een stuk meer verwacht van een front-end. De uitstraling van een webapplicatie is voor sales net zo belangrijk als de technologie die erachter draait. Een framework is meestal ontworpen en ontwikkeld door mensen die algemene oplossingen zagen voor veelvoorkomende problemen, en komt daarmee met de kennis en kunde van een professional.

De ontwikkelaars van browsers zijn in een constante strijd over marktaandeel verwikkeld. Hoewel er standaarden aanwezig zijn, nemen browserontwikkelaars het niet al te nauw met deze standaarden om concurrentie tegen te werken of zelf betere functionaliteit aan te kunnen bieden. De interpretatie van scripts en stylesheets kan tussen browsers sterk verschillen. Ook soortgelijke functionaliteit kan tussen browsers verschillen. Een framework knoopt al deze losse eindjes aan elkaar en zorgt ervoor dat elementen op alle browsers opgemaakt zijn.

Een front-end framework bevat meestal javascript om de limitaties van CSS te overbruggen.

## Selectie van frameworks

Het front-end framework moet zo begrijpelijk mogelijk blijven. De meeste Concentrator en WMS ontwikkelaars werken incidenteel met de front-end, en het is niet hun favoriete bezigheid.

## Material Design

Iedereen met de Google Gmail app kent dit ontwerp. MD is ontwikkeld door google voor in-house projecten. Het is gratis beschikbaar, maar komt niet met harde eisen.

### Voordelen

### Goed voor mobiele applicaties

Veel voorbeelden

### Nadelen

Zwaardere hardware-eisen

Uitgesproken google.

## Bootstrap

Bootstrap is ontwikkeld bij twitter. Het was een van de eerste frameworks die alle uiterlijke functionaliteit probeerde te standaardiseren. Daarnaast was bootstrap het eerste framework dat op beide desktop en mobile goed toonbaar was.

### Voordelen

In actieve ontwikkeling met ondersteuning van twitter

Goede integratie met AngularJS.

Grootste community

Veel functionaliteit is op bootstrap gebaseerd

Uitgebreide documentatie met voorbeelden.

Breed aanbod aan alternatieve scripts

### Nadelen

Het natuurlijke uiterlijk van bootstrap is zeer herkenbaar.

## 

## Metro-ui

Metro-ui is ontwikkeld door een team uit wit-rusland. Het is een interpretatie van de Microsoft metro huisstijl.

### Voordelen

Het mooiste framework

### Nadelen

Voorbeeldjes zijn maar matig responsive.

Kleine community

## MetroCSS

## Zurb foundation

Voor ontwerpers

### Voordelen

Veel functionaliteit die belangrijk is voor ontwerpers.

Grote community.

### Nadelen

Veel functionaliteit die niet relevant is aan de opdracht.

## Deelconclusie

Er zijn relatief weinig verschillen tussen de frameworks te bekennen. Er zijn weinig relevante (RWD) verschillen tussen de frameworks te bekennen. Er is daarom gekozen voor bootstrap, omdat deze out-of-the-box het meeste functionaliteit bevat en door de community het best ondersteund wordt.

# Hoe werkt een service oriented architecture?

Concentrator 2 was ontwikkeld naar een Model View Controller architectuur. Dit is goed voor gebruikers, maar minder goed voor communicatie tussen systemen.

Het voornaamste idee is dat de interacties tussen server en gebruiker, en server-server dezelfde interface gebruiken. Dezelfde functionaliteit die voor menselijke gebruikers beschikbaar is, moet ook beschikbaar zijn voor andere systemen.

Dit in tegenstelling tot bijvoorbeeld MVC, dat specifiek bedoelt is voor menselijke eindgebruikers, en meestal moeilijk voor andere systemen bereikbaar is. De volgende versie van de concentrator wil de software als een service aanbieden, waardoor communicatie tussen systemen een grotere prioriteit krijgt.

Omdat de webapplicatie gewoon op de browser beschikbaar moet zijn, is het verstandig een specifiekere architectuur te kiezen die over http beschikbaar is.

## REST

REST staat voor representational state transfer en is bedacht door Roy Fielding, een software doctorandus. Roy Fielding ontwierp REST naar de verschillende problemen die organisaties ondervonden bij hun IT activiteiten.

Er zijn keuzes te maken

De doelen die Fielding voor ogen had was een architectuur die de volgende eigenschappen had;

* Performance
* Schaalbaarheid
* Simpliciteit van interfaces, om de
* Aanpasbaarheid van onderdelen terwijl de applicatie draait.
* Zichtbaar voor agents (caching, load-balancing)
* Overdraagbare functionaliteit
* Betrouwbaarheid, zelfs bij falen van de back-end systemen

Aan de hand deze doelen heeft Fielding de volgende beperkingen opgelegd:

* **Client-Server** Communicatie verloopt tussen client-server
* **Stateless –** De server houdt geen sessies bij, en elke request bevat alle informatie die nodig is om de request succesvol af te ronden.
* **Cache –** De server kan responses cachen, en is verantwoordelijk voor de invalidatie.
* **Interface / uniform contract –** De communicatie tussen systemen is expliciet en ge
* **Layered system –** De API is enkel een laag in het system, en is niet sterk gekoppeld met andere lagen.
* **(optioneel) code on demand -**  De API kan de client van functionaliteit voorzien; bijvoorbeeld in het interpreteren van rare datatypes. De server kan functies en modules aan de client verstrekken.

#### HATEOAS

HATEOAS staat voor Hypermedia as the Engine of Application State. Het voornaamste idee is dat alle functionaliteit van de server bereikbaar is vanaf een centrale root “/api/”. De RESTful service geeft dan de verschillende functies in een context weer. Het belangrijke van HATEOAS is dat het de service op het gebruikersniveau inzichtelijk maakt. Het is daarmee makkelijk voor de programmeur om de iets tegen de API aan te bouwen. Dit is een vorm van zelf-documenterende code.

De server kan ook nieuwe, of niet gestandaardiseerde mediatypen aangeven. Ook functionaliteit om nieuwe media kan door middel van code-on-demand aan de client worden verstrekt.

## Lagen Structuur

De front-end is een abstractie van de processen die op de achtergrond draaien, en is relatief ontkoppeld van de business logica en heeft zeker niets te maken met de database van de applicatie.

Verschillende back-end applicaties kunnen dezelfde front-end delen.

## SOAP

Soap staat voor Simple Object Access Protocol, dat minder simpel is dan REST. Een SOAP bericht is een XML document dat aan bepaalde standaarden voldoet. De specificatie vereist een speciale XML opbouw waarin in ieder geval een envelop met een body aanwezig moet zijn.

De interpretatie en afhandeling van SOAP berichten zijn gestandaardiseerd. SOAP is daarmee een protocol, en REST is geen protocol. Het protocolzijnde van SOAP maakt het mogelijk om heel duidelijke afspraken te maken tussen de client en server.

SOAP ondersteund transacties en messaging. SOAP daardoor meer geschikt voor betrouwbare informatie-uitwisseling tussen systemen.

## Deelconclusie

Eigenlijk is er geen overlap tussen de gebieden waar SOAP voor bedoelt is, en REST voor bedoelt is. SOAP is een protocol dat voornamelijk bedoeld is om ACID transacties tussen systemen uit te voeren, terwijl REST erop gericht is services aan te kunnen bieden.

ACID kritieke functionaliteit kan met SOAP worden gedaan, op zich met het verstrekken van een javascript SOAP client met Code-on-demand. Dit is echter niet praktisch uitgewerkt vanwege een gebrek aan tijd.

REST is een architecturale stijl gericht op ontkoppelen van webservice, applicatie en client, terwijl SOAP een echt protocol behelst.

Er is gekozen om in het PoC alleen de REST laag te implementeren, voornamelijk omdat het implementeren van een SOAP laag niet in de tijd zou kunnen.

Vanwege de eerdere ervaringen en kennis binnen Diract is er een keuze gemaakt voor het ontwikkelen van een webservice gebaseerd op IIS server met ASP Core. Hoewel ASP Core momenteel nog in alpha ontwikkeling staat, zijn er werkende release-candidates met veel functionaliteit.

Om de werking van de applicatie aan te tonen is er wel een back-end aanwezig, maar deze is enkel illustratief. Deze is ontwikkeld met EntityFramework 7, dat ook in staat van alfa is.

Het Poc beschikt qua functionaliteit over het draaien van queries, updaten en toevoegen en verwijderen van producten en gerelateerde entiteiten.

# Hoe implementeer ik de authenticatie van het systeem?

Authenticatie en autorisatie zijn integraal voor het werken van het systeem.

Omdat een REST server geen sessies bijhoud is authenticatie niet zo vanzelfsprekend als bij een MVC applicatie.

Om integratie de software voor derde partijen te versimpelen is er een keuze voor single-sign-on gemaakt door de lead-developer van concentrator. Dit houdt in dat de identiteit van een gebruiker door verschillende (externe) partijen kan worden beoordeeld.

Binnen AspCore is er een module beschikbaar (Identity) die mogelijk maakt.

## Deelconclusie

Authenticatie verloopt nu via een basic request, waarbij de server een cryptografische authenticatie cookie meestuurt waarin de rollen van de gebruiker staan. Deze cookie word bij elk request meestuurd, en kan worden ge invalideert door Angular als . Het legen van alle cookies volstaat als logout aangezien de gebruiker.

Omdat authenticatie niet mag worden gecached en het significant is moet het met een POST worden gedaan.

Er is een korte impressie van login gemaakt, maar voorlopig nog geen daadwerkelijke implementatie van enige identiteit binnen het systeem.

# Conclusie

Er is een proof-of-concept ontwikkeld waarin de meeste aanbevelingen zijn uitgewerkt. De webapplicatie draait tegen een RESTful webservice. De applicatie is goed bruikbaar op alle platformen en beeldafmetingen.

Verschillende aanbevelingen qua mobile development zijn niet uitwerkt in het proof-of-concept. Het niet uitwerken is bewust gedaan om andere waarde te realiseren (REST).

Enige dingen die niet daadwerkelijk zijn geïmplementeerd zijn caching en prefetching. De keuze om deze niet uit te werken heeft te maken met een infrastructuur die nog niet bestaat, moeilijkheiden met cachen en invalidatie van de cache. Angular en AspNet doen uit de doos al aan veilig cachen van templates en andere statische informatie. Daarnaast heeft een cache ook te maken met authenticatie en authorizatie.

Er is voor een RESTful webservice gekozen omdat Diract als organisatie haar software als een service wil kunnen aanbieden. RESTful schaalt beter en heeft betere performance dan traditionele MVC webapplicatie. Daarnaast verlaagt het de onderhoudskosten van de interfaces naar gebruikers en systemen. Meerdere backend systemen kunnen daarnaast naast elkaar draaien en functionaliteit aanbieden via een enkele webservice.

Het implementeren van authenticatie en authorizatie was ten tijden van het schrijven van deze conclusie nog in ontwikkeling. De keuze is gevallen voor het implementeren van een Identity service, waarbij de authenticatie en de rechten van gebruikers door derde partijen kan worden bepaald (en dus ook dus de klant zelf). Het toekennen en overzicht houden van gebruikersrechten is wel nog steeds een kerntaak van de concentrator.

# Verklarende woordenlijst

**API**  *application programming interface*

Een interface voor programmeurs die systemen willen integreren.

**Bower**

Een op NodeJs gebaseerde dependency manager voor javascript, gericht op front-end dependencies.

**Caching**

Tijdelijke opslag van data. Deze data is opgeslagen omdat het opnieuw verkrijgen van deze data tijd of bandbreedte zou kosten.

**RWD** *Responsive Web Design*

Een manier van pagina-opbouw waarbij het webdocument haar elementen van grootte en positie kunnen veranderen. De keuze om

**Mobile** *colloquiaal*

Mobiele apparaten zoals telefoons en tablets. Met *Mobile* word er meestal op een platform zoals *Windows-phone, Android* of *iOS* gedoeld.

**IDE** *integrated development environment*

Grote tekstverwerker die programmeurs ondersteund met functionaliteit zoals *debuggers*, *code-completion*, *buildsystems*, en *fout-herkenning*.

**NodeJs**

Een javascript omgeving die met het OS in plaats van browser id geïntegreerd. NodeJs bied front-end developers een scala aan functionaliteit die normaal in een *IDE* te vinden is.

**NPM** *Node Package Manager*

Een dependency manager voor javascript.

**REST** *Representational State Transfer*

Een architecturele stijl van *API* ontwerp met de focus op schaalbaarheid en performance.

**Scalable** s*chaalbaarheid*

Of de webapplicatie ook functioneel blijft staan na een sterke toename in gebruikersaantallen. Meer gebruikersaantallen vragen om meer hardware capaciteit, en het is niet vanzelfsprekend dat een webapplicatie deze capaciteit daadwerkelijk kan gebruiken.

**Front-end**

De lagen van het system waar de eindegebruiker direct mee in contact staat. Onder gebruikers vallen ook externe systemen.

**Garbage-collection**

Techniek waarbij er automatisch wordt bijgehouden welke data-objecten in het geheugen zijn geladen en daarvan in actief gebruik zijn. Elementen die niet meer actief zijn, worden uit het geheugen geladen. Dit is een volautomatisch proces dat geheugengebruik en de programmeur bespaart op complexe problemen.

**Thread**

*Draad van opeenvolgende instructies aan de processor*

**Back-end**

De achterliggende logische lagen van een webapplicatie; databases, businesslagen.

**HATEOAS** *Hypermedia as the Engine of Application State*

Zie hoofdstuk

**Service oriented architecture**

**Zie hoofdstuk**

# Bibliografie

**Harrison, Rachel, Flood, Derek en Duce, David. 2013.** *Usability of mobile applications.* sl : SpringerOpen, 2013.

**http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\_arch\_style.htm**