7-6-2016

Sieger Falkena

Hogeschool van amsterdam – Diract-it

Responsive front-end

voor Diract-IT

# Abstract

This research report details the findings into a business-case of a for Diract-IT (further named Diract). Diract is a software development firm, specializing in products supporting the retail industry in their processes with tailor-made extensions and integrations.

For a new generation of their products, oriented at both service-oriented and on-premise installation, Diract wants a modern approach to User Interface architecture. This research was executed to this purpose.

New contexts of product usage and users were explored and leveraged to give users of Diract products a better user eXperience. Main points of research were the new capabilities of mobile platforms, and how to avoid it’s weaker capabilities while exploiting mobile strengths. Another new context was the multilingual use of the application by a international userbase and clients with an international character.

To make these new contexts of usage possible, several SPA (single page application) frameworks were selected and evaluated for their usability and stability in a dynamic and fast-paced software development environment.

To enable Diract to expand to a service-based approach, new developments in web-based API’s (*Application Programming Interfacse*) were investigated. Advantages and disadvantages of REST and SOAP based WebAPI techniques were analyzed and selected by business needs, recent innovations and trends in API design.

All research was consolidated into a working Proof-of-Concept, and delivered with appropriate documentation.

Inhoudsopgave

[Abstract 1](#_Toc453251983)

[Inleiding 5](#_Toc453251984)

[Diract 5](#_Toc453251985)

[Concentrator 5](#_Toc453251986)

[Warehouse Management System 5](#_Toc453251987)

[Andere producten van Diract 5](#_Toc453251988)

[Probleemstelling 5](#_Toc453251989)

[Doelstelling 6](#_Toc453251990)

[Onderzoeksmethoden 6](#_Toc453251991)

[Wat is een goede front-end opzet voor Diract? 7](#_Toc453251992)

[Wie zijn de gebruikers en waar hechten zij waarde aan? 7](#_Toc453251993)

[PACMAD 7](#_Toc453251994)

[Deelconclusie 9](#_Toc453251995)

[Proof-of-concept 9](#_Toc453251996)

[Wat zijn mobiele limitatie en hoe kunnen deze worden gecompenseerd? 11](#_Toc453251997)

[Limitaties 11](#_Toc453251998)

[Voordelen 14](#_Toc453251999)

[Deelconclusie 15](#_Toc453252000)

[Uitgewerkt in proof of concept 15](#_Toc453252001)

[Wat is NodeJs en waarom gebruikt iedereen het? 16](#_Toc453252002)

[Node.JS 16](#_Toc453252003)

[JavaScript webservers 17](#_Toc453252004)

[Dependency management 17](#_Toc453252005)

[Karma test framework 18](#_Toc453252006)

[Transpilers 18](#_Toc453252007)

[CSS preprocessors 18](#_Toc453252008)

[Build-management 18](#_Toc453252009)

[Overige tools 19](#_Toc453252010)

[Integratie met Visual Studio 2015 19](#_Toc453252011)

[Conclusie 20](#_Toc453252012)

[In het proof-of-concept 20](#_Toc453252013)

[Welke SPA frameworks sluiten het beste aan bij de vereisten? 22](#_Toc453252014)

[AngularJS 23](#_Toc453252015)

[Sencha ExtJS 24](#_Toc453252016)

[ReactJS 26](#_Toc453252017)

[Deelconclusie 27](#_Toc453252018)

[Proof-of-concept 27](#_Toc453252019)

[Welke CSS framework past het beste bij concentrator? 28](#_Toc453252020)

[Waarom gebruiken ontwikkelaars een front-end framework? 28](#_Toc453252021)

[Selectiecriteria van frameworks 28](#_Toc453252022)

[Material Design 29](#_Toc453252023)

[Twitter Bootstrap 29](#_Toc453252024)

[Metro-ui 30](#_Toc453252025)

[Foundation 6 30](#_Toc453252026)

[Conclusie 31](#_Toc453252027)

[Proof-of-concept 31](#_Toc453252028)

[Waar word de pagina gerenderd? 32](#_Toc453252029)

[Conclusie 33](#_Toc453252030)

[Proof-of-concept 33](#_Toc453252031)

[Hoe gebruik ik de Javascript client tegen de service laag? 34](#_Toc453252032)

[REST 34](#_Toc453252033)

[REST properties 35](#_Toc453252034)

[REST constraints 36](#_Toc453252035)

[HATEOAS en HAL 38](#_Toc453252036)

[SOAP 39](#_Toc453252037)

[Deelconclusie 40](#_Toc453252038)

[In het proof-of-concept 40](#_Toc453252039)

[Conclusie 42](#_Toc453252040)

[Proof-of-concept 44](#_Toc453252041)

[Client 44](#_Toc453252042)

[Webapplicatie 44](#_Toc453252043)

[Verdere noten 45](#_Toc453252044)

[Verklarende woordenlijst 46](#_Toc453252045)

[Bibliografie 50](#_Toc453252046)

# Inleiding

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

## Diract

Diract 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 Diract 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 product- en order informatie. Het Concentrator systeem verwerkt de binnenkomende informatie van fabrikanten of importeurs. Daarna exporteert de Concentrator 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 product- en orderinformatie 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 apparaten 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. Bedrijfs-breed is er besloten alle nieuwe producten en productversies een service laag te geven die door eindgebruikers evenals andere systemen als aanspreekpunt van 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.

Ook wil Diract op termijn hun software als een service (SaaS) kunnen aanbieden, om nog meer flexibiliteit aan de klanten te kunnen bieden. Meerdere klanten zullen dan dezelfde Concentrator instantie gebruiken.

# 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.

Het proof-of-concept richt zich in stadium alleen op productinformatie. Andere cases worden binnen dit proof-of-concept niet aan de orde gebracht.

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?

People At the Center of Mobile Application Development

## PACMAD

Mobiele gebruiksvriendelijkheid is anders dan gebruiksvriendelijkheid voor desktopsystemen, daardoor worden er 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). Tussen 2013 en nu 2016 is er weinig veranderd. De voornaamste

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.

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 interpreteren en implementeren van gebruikerswensen.
* 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
  + Dit zijn de fouten en excepties die optreden. De maar de mate waarin feedback wordt gegeven en de snelheid waarmee bugs worden verholpen zijn cruciaal voor een goed gebruikersgemak en efficiënte support.
* 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. Manieren om de cognitieve lading laag te houden zijn overzichtelijke en herkenbare schermen.

## Deelconclusie

In de nieuwe Concentrator zijn er nieuwe gebruikscontexten aanwezig. De applicatie moet nu ook beschikken over lokalisatie en internationalisatie en moet mobiel beschikbaar zijn.

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.

Het intuïtief houden van de applicatie is een van de primaire doelen, omdat dit van invloed is op de meer subjectieve kwaliteiten van de webapplicatie zoals de leerbaarheid, onthoudbaarheid en de cognitieve ballast. Terugkerende objecten (zoals bijvoorbeeld een productformulier) kunnen voorzien worden van unieke kenmerken, of belangrijke attributen. Het selecteren van kleur kan makkelijk met een kleurmixer worden gedaan.

## 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 uit een JSON bestand 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.

Voor de herkenbaarheid en de intuïtieve ervaring verschillend objectmodellen van uiterlijk.

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

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

## 2.1. Limitaties

**Visueel**

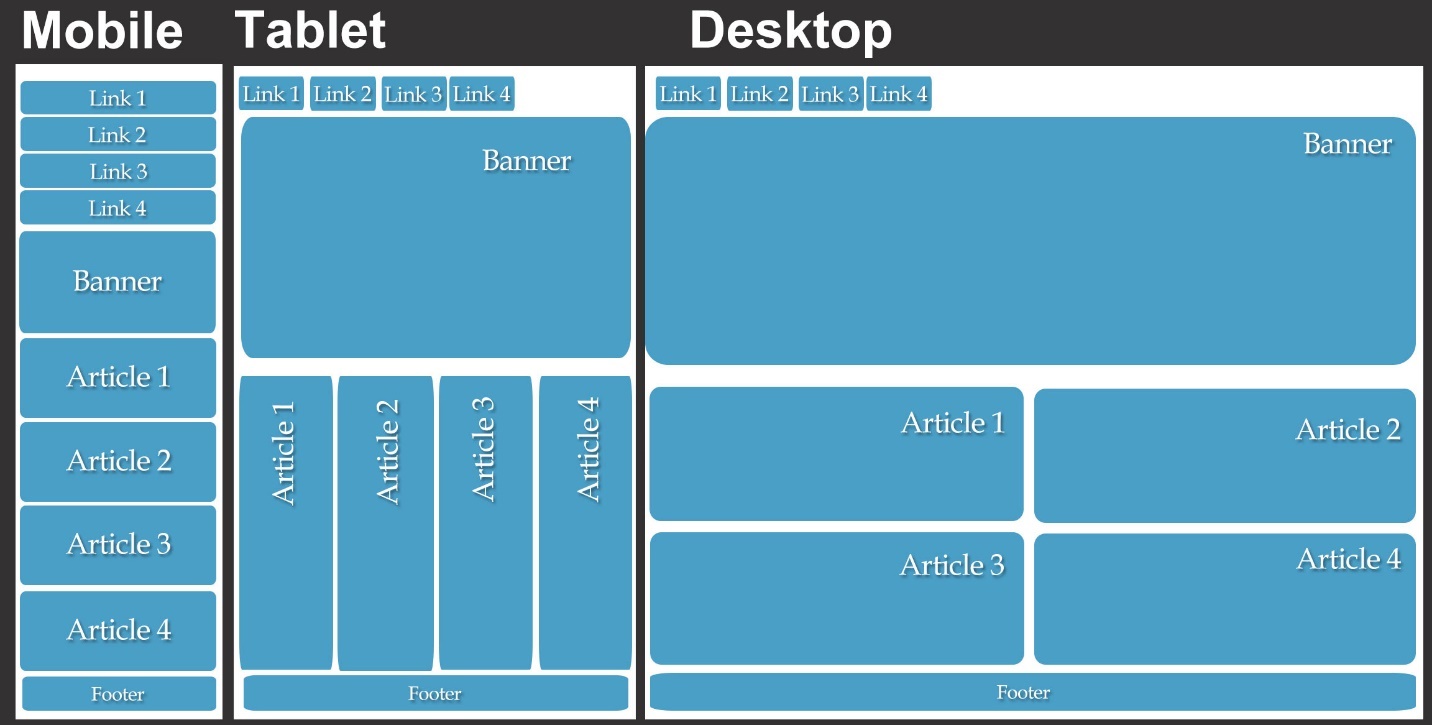
Visueel hebben mobiele apparaten een over het algemeen kleiner beeldscherm en meer pixels/cm2. Dit maakt dat elementen (knoppen en tekst) die als goed bruikbaar op desktop worden ervaren, als te klein worden ervaren op mobile. Vice-versa kunnen tekst en knoppen voor mobile als te groot worden ervaren 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 breedte van de elementen kan per combinatie beeldscherm en afmeting worden aangegeven. De hoogte van elementen word bepaald door de inhoud.

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.



Responsive voorbeeld . Het toont een abstracte applicatie met elementen die afhankelijk van de beeldscherm beedte worden voorzien van andere afmetingen. Bron: http://www.digitalfamily.com/tutorials/how-to-create-responsive-adaptive-web-sites/

|  |  |  |
| --- | --- | --- |
|  | **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

**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, behalve dat ze beter in natte omstandigheden fungeren.

**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 toetsen een stuk kleiner zijn dan die van een fysiek toetsenbord. Beginnende touchscreen typisten halen een gemiddelde van 21,23 wpm op een klein smartphone toetsenbordje, vergeleken met 55,59 wpm voor een normaal toetsenbord. <http://people.ucsc.edu/~swhittak/papers/MobileHCI2011_FINAL.pdf>

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 en kunnen er ook minder fouten gemaakt worden.

**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.

**Locatie gebaseerde informatie**

Bij het WMS kan er gebruik worden gemaakt van locatiedata om dichtbijzijnde producten te tonen. De GPS informatie van een mobiele telefoon is met een fout van 8 meter echter onnauwkeurig. Enige locatiedata word pas relevant als gps technologie nauwkeuriger word, of de foutmarge niet significant is.

## 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.

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.

## Uitgewerkt in proof of concept

Het proof-of-concept is responsive uitwerkt met Bootstrap als framework. Elementen kunnen van formaat verschillen. De keuze om Bootstrap te gebruiken kwam neer op de hoeveelheid functionaliteit en de ondersteuning door community.

Daarnaast is typeahead in het PoC uitgewerkt. Typeahead is een dropdownselectie die gefilterd kan worden. Het voordeel van typeahead is dat het weinig schermruimte in beslag neemt en met weinig toetsaanslagen veel selecties kan maken.

# Wat is NodeJs en waarom gebruikt iedereen het?

Elke technologie komt met een ontwikkelomgeving, javascript komt met NodeJS.

Javascript blijft een steeds grotere rol spelen in de interactie tussen het document en browser, vanuit een wens webpagina’s met meer functionaliteit en interactiviteit te ontwikkelen. Naar dit doel is de V8 javascript engine ontwikkeld die javascript naar native code compileert zodat deze sneller uitgevoerd kan worden en niet hoeft te worden geïnterpreteerd zoals een script.

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

* Debuggen van javascript gebeurt in de browser.
* Javascript is geen object-georiënteerde taal
  + 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 objecten 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
* Javascript mist een goede SDK en is zeer afhankelijk van externe libraries.

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. NodeJS geeft ontwikkelaars de mogelijkheid om hun buildtools in javascript te schrijven.

## 3.1. Node.JS

NodeJS is een javascript engine gebaseerd op de V8 javascript engine die systeemresources beschikbaar maakt en javascript de capaciteit voor multiprocessing verschaft. Daarmee maakt NodeJS snelle efficiënte build-tools en webservers mogelijk.

## JavaScript webservers

NodeJS heeft 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. De requests worden daarna door sub-processen afgehandeld die systeemresources en IO aanspreken.

Javascript webservers kunnen sneller zijn traditionele applicatieservers zoals Tomcat of IIS omdat ze efficiënter met hun threads omgaan. Daarentegen word deze applicatiearchitectuur langzaam door grotere applicatieservers overgenomen. Asynchroon afhandelen van requests is een onderdeel van IIS webservers.

## Dependency management

Een project bevat bijna altijd dependencies. Dependencies zijn externe onderdelen die door de webapplicatie worden gebruikt, maar niet door de ontwikkelaar is geschreven of in de SDK aanwezig is. Dependencies kunnen belangrijke updates of upgrades krijgen, wat vereist dat een ontwikkelaar zijn verwijzing moet veranderderen.

Dependencies zijn over het algemeen groter dan de applicatie zelf, wat het deploy proces vertraagd. De ontwikkelaar kan een lijst van gewenste dependencies met versienummers opgeven om deze in de applicatie gebruiken. De dependencymanager haalt deze dan uit het web op, zodat niet alle dependencies naar de webserver hoeven worden verstuur. Dit versneld het deploy-proces enorm.

NodeJS kent twee dependency managers; NPM en Bower. Beide beschikken over een repository en kunnen ook github repositories als bron gebruiken.

Node Package Manager (NPM) is een van de grotere NodeJs repositories en bevat dependencies voor de backend.

Bower is de andere grotere NodeJS repository en bevat voornamelijk dependencies voor de front-end.

Deze repositories zijn onbeheerd en er is nogal wat sprake van instabiele packages.

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

## Karma test framework

Het Karma test-framework is bedoelt om verschillende testparadigma’s zoals unit-testing en usability testing te automatiseren. Een karma stack bestaat uit drie onderdelen;

* Karma als testrunner
* Een aanname framework (Jasmine) dat de tests op correctheid controleert
* Een browser waarop de tests worden uitgevoerd

Het is hiermee mogelijk de functionaliteit en correctheid van de client op meerdere browsers automatisch te testen. Met kwalitatief goede tests is het hiermee mogelijk de stabiliteit en functionaliteit van de client te garanderen.

Karma is moeilijk in een testopzet te integreren, voornamelijk vanwege het onderzoekend ontwikkelen waarbij er veel veranderd. De kracht van een testframework ligt bij het stabiel houden van een applicatie over een langere tijd.

## Transpilers

Compilers compileren programmacode naar machinetaal. Transpilers transpileren een superset van javascript naar javascript. Transpilers lossen problemen inherent aan javascript op waar libraries tekort schieten.

Javascript wordt het meeste zonder transpiler gebruikt. Transpilers als Coffeescript en Dart vallen voornamelijk uit de gratie omdat ze niet of weinig overeenkomsten hebben met javascript.

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.

## CSS preprocessors

Cascading stylesheets (CSS) scripts zijn puur verantwoordelijk voor het uiterlijk van representatie. Net zoals javascript transpilers zijn er processors die een dialect van CSS uitschrijven naar bruikbare CSS.

De voornaamste keuzes voor CSS processors zijn SASS en LESS. SASS ziet zichzelf als een verbetering op CSSen LESS ziet zich op haar beurt weer als verbetering op SASS. Beide zijn niet echt onderzocht omdat een eerder proof-of-concept zich hier al over heeft gebogenen omdat het al zeker stond dat er een CSS framework voor gaat worden gebruikt.

## 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 tools

Underscore.js is een setje breed-toepasbare functies. Het implementeert functionaliteit 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

VS15 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. Deze integraties helpen de programmeur enorm bij het productief maken van code.

## Conclusie

Javascript is een volwaardige programmeertaal en om als ontwikkelaar efficiënt en effectief te ontwikkelen is een ontwikkelomgeving onmisbaar. NodeJS is de enige bruikbare ontwikkelomgeving voor javascript in combinatie met VS15.

Gulp wordt als buildmanager door VS15 gebruikt. Andere keuzes (grunt, etc.) ondersteund VS15 niet. De keuze is daarmee snel gemaakt om gulp als dependency manager te gebruiken.

Hoewel Bower en NPM repositories waarschijnlijk ooit op de schop gaan, is het toch belangrijk de hoeveelheid javascript dependencies met een tool te beheren. Dependencymanagers 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.

Karma en soortgelijke test-frameworks zijn moeilijk te integreren met SPA frameworks en vereisen een specifieke vorm van programmeren. De keuze om deze test-frameworks te gebruiken

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.

Dit onderzoek heeft zich verder niet over CSS preprocessors gebogen, omdat het meeste styling werk al met een framework gaat worden bewerkstelligd. De laatste functionaliteit die in CSS moet worden gedefinieerd is niet noemenswaardig genoeg een preprocessor aan te besteden.

## In het proof-of-concept

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

De keuze voor buildsysteem viel op Gulp 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

Er is in het proof-of-concept geen gebruik van Typescript gemaakt. De reden voor het niet implementeren was omdat er intern al een ander proof-of-concept was gemaakt dat Typescript goed in het licht zette en daarom niet onderzocht hoefde te worden.

Ook van CSS preprocessors als LESS is geen gebruik gemaakt. Eveneens is deze in het eerdere proof-of-concept al uitgedacht en in het licht gezet. Het uiteindelijke CSS script is daarnaast niet erg lang en hoeft, met het doel op de opdracht, geen efficiëntie te kennen.

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

Om goed om te gaan met verschillende functionaliteit moet er een keuze worden gemaakt naar de verschillende single-page-application frameworks.

Clients bestaan tegenwoordig voornamelijk uit frameworks gebaseerd op javascript. Het framework is verantwoordelijk voor het verwisselen van pagina, het implementeren van controllers en het bijhouden object levenscyclussen. Er kan zonder een framework worden ontwikkeld, maar het zou veel meer tijd kosten voor een kwalitatief minderwaardig product.

Voor de selectie van een framework zijn er een paar kwalitatieve eisen neergezet.

* Het moet een GRID (uitgebreide tabel) ondersteunen
* De ontwikkeling van het framework moet stabiel zijn
* Het framework ondersteund garbage-collection
* Definieert een architectuur en bied daarmee een alternatief voor DOM manipulaties
* (pre) kan het javascript van het CSS framework ondersteunen
* (pre) kan door een back-end ontwikkelaar worden begrepen.

jQuery is niet meegenomen omdat jQuery of varianten op de achtergrond van de meeste frameworks aanwezig is. JQuery valt meer als een tool dan een framework te beschouwen, omdat het geen kernfunctionaliteit zoals MVC aandraagt en selectors maar matig te onderhouden zijn. jQuery en afgeleiden kunnen naast het SPA framework worden gebruikt.

Alle SPA frameworks hebben een behoorlijke leercurve en komen met een flinke hoeveelheid documentatie. Er is geen sprake van een framework dat zonder voorkennis effectief gebruikt kan worden en sommige functionaliteit (zoals dependency injection) vergroten de leercurve, maar versimpelen het gehele ontwikkelproces.

## https://www.pentapie.com/wp-content/uploads/2015/12/angularjs-stacked1.png4.1. AngularJS

Angular was een van de eerste gratis javascript SPA frameworks. De ontwikkelaars van Angular zijn in 2010 opgekocht door Google, die Angular voor hun eigen producten gebruiken.

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

Angular2 is een herschreven versie van Angular en is momenteel in beta.

**Voordelen**

Angular is gratis

Onofficieel het SPA-framework van Google en wordt gefinancierd uit de budgetten van Google.

Angular is verreweg het meest toegepaste framework en heeft de meeste community libraries van derde partijen. Functionaliteit zoals GRIDS, touch en integraties met andere systemen zijn beschikbaar op repositories en *GitHub*. De kwaliteit van deze externe libraries is zeer goeden voorzien genoeg functionaliteit voor de meeste use-cases.

Angular is binnen Diract al eerder voor een proof of concept gebruikt. Dat is zeer goed bevallen bij de medewerkers van Diract.

Angular komt met een gedefinieerde architectuur, Model View ViewController. MVVC is conceptueel vergelijkbaar met Model View Controller opzet. Het splitsen van datatypes, logica en uiterlijk maakt het ontwikkelproces overzichtelijker.

Angular komt met functionaliteit zoals Dependency Injection, dat een ontwikkelaar helpt bij het importeren van dependencies.

**Nadelen**

Angular is zeer breed opgezet en bevat overal functionaliteit voor. Sommige modules zijn echter sterk ondermaats, zoals de routing module. Er zijn wel alternatieve modules voor deze functionaliteit, zoals de ui-routing module van het Angular-ui team.

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

## Sencha ExtJS

De eerste versie van Ext.js kwam in 2007 uit en was daarmee het eerste SPA framework. Het is een zeer sterk platform voor het ontwikkelen van desktop-achtige webapplicaties. Het ExtJS is recentelijk door bedrijf Sencha gekocht, welke nu eigendom is van ExtJS.

ExtJS komt in drie varianten; Standard, Pro en Premium, waarvan de licenties per developer zijn berekend worden met een minimum afname van 5 licenties. De prijzen voor het concentrator team (8 ontwikkelaars) liggen tussen de $7000 (€6000) en $14000 (€12300).

De professional versie van ExtJS komt met VS15 integraties en een visuele editor voor de opbouw van views.

De premium versie van Ext.js komt met een pivot data-grid voor data analyse en een adapter voor de D3JS library voor data-visualisatie. Deze D3 library is zelf overigens open-source en gratis.

**Voordelen**

ExtJS heeft veruit het meest uitgebreide grid. Het grid ondersteund functionaliteiten als grafiekjes en afbeeldingen. Voor managers zijn dit zeer welkome toevoegingen.

ExtJS zou met 320 uur support komen in het geval van een team van 8 man. Dit is genoeg om een sterke user interface neer te zetten.

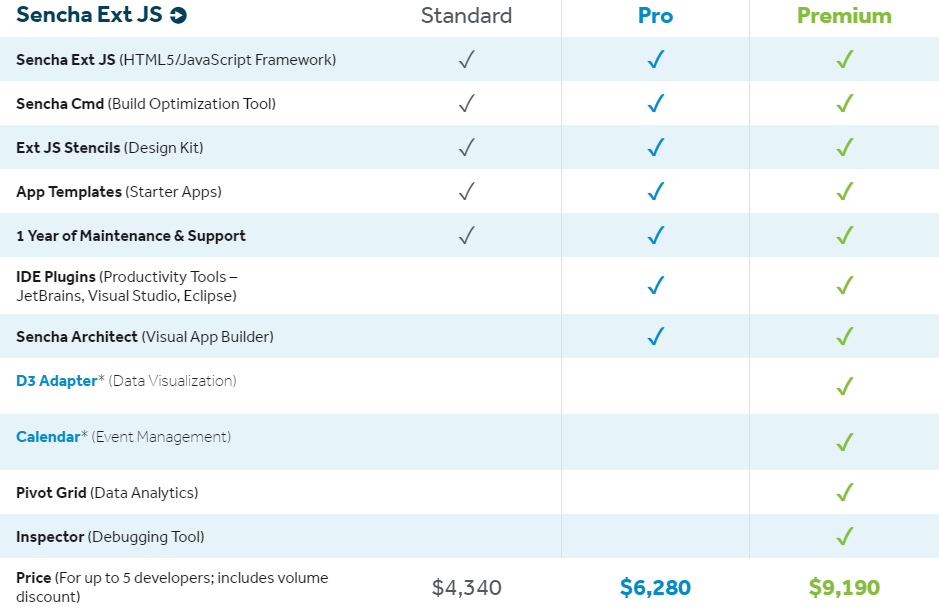
ExtJS bied binnen javascript verschillende opties om javascript een stabielere ontwikkelomgeving te bieden. Het framework bied onder andere een alteratief voor javascript prototype classes.

**Nadelen**

ExtJS is maar gedeeltelijk responsive. De lay-out manager is niet responsive te krijgenen is dus niet te gebruiken in een responsive applicatie. Dit houdt in dat de goede kanten van ExtJS niet gebruikt kunnen worden in de responsive applicatie.

ExtJS doet niet aan garbage collection. Ontwikkelaars moeten zelf hun objecten vernietigen. Dit kan zeer moeilijk oplosbare bugs veroorzaken. Gelukkig komt de premium versie met een profiler waarmee je kan zien waar het fout gaat.

ExtJS heeft geen community en kan dus moeilijk inspelen op nieuwe innovaties. Ik heb grondig moeten zoeken voor enig enthousiasme bij niet-Sencha ontwikkelaars.



## http://red-badger.com/blog/wp-content/uploads/2015/04/react-logo-1000-transparent.pngReactJS

ReactJS is een SPA framework van Facebook. Het is in 2014 uitgekomen, maar ondervind nu pas een grote belangstelling. ReactJS is onderdeel van de Flux suite van Facebooken vervult daarin de rol van View in een losstaand MVC framework.

**Voordelen**

ReactJS is gratis.

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

Veel functionaliteit van derde partijen.

Onofficieel het framework van en voor facebook.

**Nadelen**

Tijdens het onderzoek brak ReactJD door een van de dependencies. Left-pad was een kleine dependency die strings naar een bepaalde grootte opvulde, iets wat de meeste programmeurs in drie lijntjes code zouden moeten kunnen doen. Left-pad brak uiteindelijk door een niet-werkende update van de maker. Het was een indicatie van de kwaliteit van Facebook ontwikkelaars, die te veel functionaliteit van derden proberen te gebruiken of te incompetent zijn deze te maken.

ReactJs betreft enkel views. Andere functionaliteit (controllers, models) kunnen o.a. met andere onderdelen uit de REDUX / FLUX suite worden gemaakt. Dit maakt het echter niet makkelijker om een werkend geheel neer te zetten en primair back-enders de front-end te laten aanpassen.

De voorbeelden zijn simpel en helder, maar maken gebruik van veel bad-practises. Er is geen scheiding tussen representatie en logica in de voorbeelden, wat een slecht voorbeeld is voor nieuwe ontwikkelaars.

React heeft veel verschillende data-grids, maar geen van de data-grid’s bevat een complete set aan functionaliteit zoals het tonen van complexes of pagination van data verkregen van een WebAPI.

## Deelconclusie

ExtJS is een zeer mooi framework, maar de kracht van ExtJS ligt overduidelijk niet bij responsive design. Het gebruikt Bootstrap voor de responsive gedeeltes en de paradepaardjes uit de voorbeelden zijn niet responsive. Daarnaast ontbreekt het ExtJS aan garbage-collection, wat zwaar weegt op het comfortabel en foutloos ontwikkelen. Het bijhouden van geheugenobjecten is geen makkelijke taaken is moeilijk te onderhouden.

De waarde van ExtJS ligt bij het aanbieden van een desktop ervaring in een webapplicatie, maar dit is geen meerwaarde als het niet ook in een responsive omgeving kan worden gebruikt. De keuze om ExtJS niet te gebruiken ligt dan ook voor de hand. Ook de voordelen van enterprise ontwikkeling worden grotendeels door functionaliteit van Typescript overbrugden kunnen niet met javascript datastructuren zelf worden overbrugd.

ReactJS is een mooi framework, maar bied zonder architectuur geen houvast voor ontwikkelaars. Het betreft bij ReactJS alleen maar het View onderdeel van een ModelViewController omgevingen het toevoegen van libraries zorgt voor meer uitzoekwerk bij het analyseren van de applicatie. ReactJS bied met haar modulariteit wel de mogelijkheid om op bandbreedte te besparen, maar dit is waarde die niet gerealiseerd gaat worden met een lage hoeveelheid gebruikers die vaak terug komen.

Daarnaast vereist ook ReactJS dat geheugen-management een ding is. De doodsteek voor ReactJS was het left-pad fiasco. Left-pad was een stukje code dat door een capable programmeur goed te programmeren moet kunnen zijn, maar werd als dependency geïmporteerd. Dit geeft een indicatie van de stabiliteit van ReactJS.

Angular heeft problemen met de overgang naar Angular2en gaat net als ReactJS naar een microframework van losstaande componenten. Veel functionaliteit kan daarbij niet direct van Angular naar Angular2 worden ge-port. Sommige onderdelen van Angular zijn niet naar standaarden kennen een duidelijke verbetering.

Desondanks is Angular een van de meest functionele en stabiele frameworks, welke door hun zeer significante marktaandeel word onderschreven. Angular bied front-end ontwikkelaars een duidelijke en onderhoudbare architectuur met moderne functionaliteit zoals Dependency Injection en breekt de garbage-collector van javascript niet.

## Proof-of-concept

De client is met Angular 1.5 uitgewerkten maakt met de volgende uitzondering gebruik van Angular standaard elementen.;

Routing is met de Angular-UI-Routing module geïmplementeerd welke een substantiële verbetering op de originele routing module vormt omdat het een daadwerkelijke statemachine inbouwd en parameters uit de URL kan halen. Het data-grid is met het Angular-UI-Grid onderdeel uitgewerkt, ook afkomstig van het Angular-UI team. Met het data-grid kunnen producten in een lijst worden getoond en gefilterd op eigenschappen. De lijst maakt gebruik van pagination en filterd resultaten op de server.

Data word met functionaliteit uit de ngResource module opgehaald en afgehandeld.

Functionaliteit die geen toepasselijke officiële module kent, zoals lokalisatie (l10n) of alerts, zijn uitgewerkt in een Angular service.

# Welke CSS framework past het beste bij concentrator?

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

## 5.1 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 minstens 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.

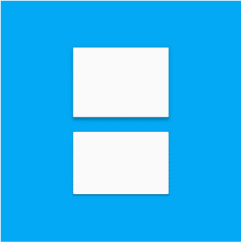
De ontwikkelaars van browsers zijn in een constante strijd over marktaandeel verwikkeld. Hoewel er standaarden aanwezig zijn, nemen browserontwikkelaars het niet al te nauw met deze standaarden om concurrentie tegen te werken of zelf betere functionaliteit aan te kunnen bieden. De interpretatie van stylesheets 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. Deze javascript onderdelen werken soms wat stroef met het javascript framework, maar het zeker niet onmogelijk om deze samen te gebruiken zonder expliciete integratie.

## Selectiecriteria van frameworks

* In actieve ontwikkeling
* Heeft een community
* Ondersteund gebruik met AngularJS
* Is aanpasbaar

## Material Design

MD is ontwikkeld door Google voor in-house projecten. Het is ontworpen met nieuwe wetenschappelijke inzichten gebruik van software. Kenmerkend is het platte uiterlijk en veel gebruik van schaduw om de illusie van papieren onderdelen te geven.

**Voordelen**

Goed voor mobiele applicaties

De beste documentatie

**Nadelen**

Zwaardere hardware-eisen

Lijkt uitgesproken op de huisstijl van Googleen kent geen alternatieve CSS scripts.

## Twitter Bootstrap

Bootstrap is ontwikkeld bij Twitter en is momenteel een dochterbedrijf van Twitter. Het was een van de eerste frameworks die alle uiterlijke functionaliteit probeerde te standaardiseren en soortgelijk weer te geven, ongeacht het platform van de client.

**Voordelen**

Bootstrap kent meerdere Javascript integraties met SPA frameworks. Dit maakt DOM manipulaties overbodig, wat het onderhoud versimpeld.

Bootstrap heeft een zeer grote community die gericht is op het verkopen van alternatieve scripts. Deze werken functioneel hetzelfde als Bootstrap, maar hebben een ander uiterlijk. Deze scripts zijn spotgoedkoop en komen meestal met kleine extra’s zoals animaties.

Veel functionaliteit uit front-end dependencies is gemaakt met Bootstrap. Dependencies nemen daardoor meteen het uiterlijk van de applicatie aan.

Uitgebreide documentatie met voorbeelden.

**Nadelen**

Bootstrap heeft een zeer kenmerkend uiterlijk, maar valt goed aan te passen. Bootstrap is een van de populairdere frameworks, voornamelijk omdat Bootstrap een van de eerste CSS frameworks was. Er zijn online zeer veel alternatieve CSS scripts te koop tegen een lage prijs.

Bootstrap mist een *DATETIME* input. Dit is vervelend, maar valt goed op te lossen met externe functionaliteit of het combineren van losse *DATE* en *TIME* velden.

## Metro-ui

Metro-ui is ontwikkeld door een team uit Wit-Rusland. Het is een interpretatie van de Microsoft metro huisstijl die ook via andere kanalen beschikbaar is.

**Voordelen**

Het mooiste framework.

**Nadelen**

Voorbeeldjes zijn maar matig responsive.

Kleine community

Geen ondersteuning

## Foundation 6

Foundation is een UI framework ontwikkeld door Zurb, een UI firma gericht op consultancy en software. Zurb ondersteund bedrijven die naar een nieuw uiterlijk op zoek zijn, maar levert ook ondersteuning bij het implementeren van hun software.

**Voordelen**

Foundation is door ontwerpers voor ontwerpers gemaakt. Foundation 6 is ontwikkeld vanuit expertise en ervaring in grafisch ontwerp. Zurb gebruikt haar eigen software ook in-houseen Foundation is ondertussen aan een zesde versie toe.

Zurb en het forum van Foundation bieden veel ondersteuning. Dit kan gratis en voor een prijs in het geval van consultancy.

**Nadelen**

Veel functionaliteit die niet relevant is aan de opdracht. De front-end moet mooi zijn, maar hoeft geen uniek karakter te hebben. Het is voornamelijk een functionele gebruikersinterface voor werk, geen ervaring om sales mee te drijven.

## Conclusie

Alle frameworks beschikken over responsive elementenen deze werken allemaal nagenoeg hetzelfde. Bootstrap voldoet aan alle verdere eisen;

* Het is makkelijk om het uiterlijk aan te passen door middel van een alternatief script
* Kent Angular integraties die DOM manipulaties overbodig maken
* Komt met de meeste functionaliteit, inclusief functionaliteit die door derden is gemaakt

Een dergelijke combinatie van eisen is niet bij de andere frameworks te vinden. Bootstrap is daarom geselecteerd als CSS framework.

## Proof-of-concept

Het proof-of-concept maakt exclusief gebruik van Bootstrap. Er waren wat problemen met de integratie tussen Angular en Bootstrap. Specifiek is het Bootstrap responsive-grid moeilijk met het *ng-repeat* *directive* van Angular te combineren. Andere CSS frameworks bieden geen uitkomst, wat deze werken op nagenoeg dezelfde manier.

Het ontbreekt Bootstrap aan functionaliteit om het *DATETIME* format goed te gebruiken. Dit valt op te lossen met dependencies of door een losse *date* en *time* te gebruiken en deze te concateneren.

# Waar word de pagina gerenderd?

Het samenstellen van een webpagina kan op de server en/of de client gebeuren.

Webpagina’s bestaan uit een document, typisch html. Deze html pagina word aan de client verzonden, die deze in de browser toont. Oudere webservers werken op deze manier en bouwen de hele pagina eerst op, waarna deze verzonden wordt en door de browser van de client. Bij enige navigatieveranderingen werd de gehele pagina opnieuw opgebouwd en naar de gebruiker verstuurd. Het resultaat is een wel-bekend ‘wit scherm’, waarbij de browser wacht op de pagina die door de server word opgebouwd.

Voornamelijk jQuery heeft hier verandering in aangebracht. jQuery kwam met de functionaliteit om AJAX *(asynchronous javascript and xml)* calls te doen, die specifieke elementen van een webpagina aan de server opvroeg en deze bij ontvangst in de webpagina ‘plakt’. Dit hield in dat een webpagina gedeeltelijk kon worden opgebouwd. Het voordeel was dat de pagina minder vaak ververst hoefde te wordenen dus een betere interactie met de gebruiker had.

Javascript frameworks zoals Angular kunnen het gehele document zelf renderen. De client krijgt dan enkel de indexpagina waaruit de client word opgebouwd. Enige data wordt dan vanaf een WebAPI verkregen.

Frameworks zoals Angular kunnen last hebben van het feit dat SEO (*search engine optimalization*) en Analytics lang niet goed samenwerkte met SPA frameworks. Hierdoor was het moeilijk de bezoeksdata en verkoopdata bij te houden bij te houden. Het is echter tegenwoordig wel mogelijk, maar vereist een meer hands-on aanpak met de Analytics software en dependencies.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Client-side rendering | | Server-side rendering | |
| Initiële laadtijd | Hoog | Client | Laag | Document |
| Laadtijd tussendoor | Laag | Data | Middel | Document + data |
| Complexiteit | Laag | Splitsing document +data | Middel | Opbouwen van document |
| Resourcegebruik | Voornamelijk op client | | Voornamelijk op server | |
| Goed voor tijdgebruik | Langere sessies met bekende gebruikers | | Korte sessies met veel unieke gebruikers | |
| Capaciteiten van client | Clients met moderne hardware | | Clients met sterk verouderde hardware | |

## 6.1. Conclusie

Gebruikers van concentrator gebruiken de webapplicatie voor langere tijd en herladen de pagina niet vaak. Dit maakt de initiële laadtijd bijna niet relevant vergeleken met de gehele levensduur van de sessie. Daarnaast word de statische informatie van de applicatie gecached opgeslagen in de browser, waardoor enige tweede laadtijd niet relevant is.

Daarnaast wil Diract ook een WebAPI hebben. In het geval van client-side rendering word WebAPI meteen gemaakt en is het primaire communicatiekanaal voor de client. Bij serverside of een mengvorm zou in ieder geval deze API ernaast moeten gemaakt en onderhouden.

Er kan nog worden gekozen om enkel de initiële indexpagina te renderen. Dit zou de initiële laadtijd naar beneden brengenen geen webpagina in de weg zitten. Dit komt, net als andere mengvormen, met kosten in de vorm van complexiteit.

## Proof-of-concept

In het proof-of-concept staat de client van de applicatie relatief ontkoppeld van de webserver. De client word statisch gehost op het adres “/app/” met een indexpagina die alle scripts laad. De WebAPI wordt op een andere root “/api/” gehost.

# Hoe gebruik ik de Javascript client tegen de service laag?

De servicelaag is verantwoordelijk voor de invoer en de uitvoer van data.

De nieuwe Concentrator applicatie komt met een servicelaag. Deze servicelaag is exclusief verantwoordelijk voor de communicatie met buiten liggende systemen. Dit is besloten vanuit een oogpunt van *seperation of concerns*, waarbij onderdelen van applicaties discrete functionele eisen hebben en zo de complexiteit van de gehele applicatie wordt verlaagd.

In voorgaande hoofdstukken is er besloten de webpagina’s bij de client te renderen en data via een aparte interface (WebAPI) met de client uit te wisselen. Deze WebAPI kan met verschillende protocollen worden gemaakt.

De keuze valt tussen REST en SOAP, omdat beide over het HTTP protocol kunnen opereren. Esoterische oplossingen, zoals het gebruik van webSockets (directe TCP/IP), zijn buiten beschouwing gelaten omdat deze nooit praktisch worden toegepast in een enterprise omgevingen geen brede ondersteuning kennen.

## 7.2 REST

REST staat voor “*REpresentational State Transfer*” en is bedacht en beschreven door Roy Fielding, een software doctorandus, in zijn proefschrift *“Architectural Styles and the Design of Network-based Software Architectures”*. Fielding ontwierp REST naar de verschillende problemen die organisaties ondervonden bij hun netwerk architectuur. Fielding was eerder verantwoordelijk voor het ontwerpen van het HTTP protocolen heeft hiervoor honderden experts geïnterviewd.

De doelen die Fielding voor ogen had was een netwerk-architectuur die de zeven volgende kwaliteiten had;

## 7.3. REST properties

**Performance**

Prestaties van het systeem, voor de gebruikers en het gebruik van bandbreedte.

**Schaalbaarheid**

Schaalbaarheid is een moeilijk onderwerp. Het gaat er om dat de netwerkarchitectuur functioneel blijft bij een grote toename in gebruik, met een evenredige toename aan capaciteit. Het is niet vanzelfsprekend dat een systeem deze capaciteiten effectief kan gebruiken.

**Simpliciteit**

Het simpel houden van onderdelen die een specifieke taak uitvoeren. Het is een generalisatie van “Seperation of Concerns”. Software onderdelen generiek en breed toepasbaar houden maakt het makkelijker om het systeem te analyseren

**Aanpasbaarheid**

Hoezeer de netwerkarchitectuur kan groeien, hergebruikt, geconfigureerd en aangepast kan worden, het liefst zonder downtime.

**Zichtbaarheid**

De netwerkarchitectuur moet open inzichtelijk zijn voor andere systemen. Veel netwerkinfrastructuur is afhankelijk van deze informatieen als de architectuur geen informatie inzichtelijk maakt, kunnen zij hun werk niet doen of werkt de architectuur niet.

**Overdraagbaarheid**

Logica moet zich over het netwerk kunnen verplaatsen naar andere systemen.

**Betrouwbaarheid**

De netwerkarchitectuur moet betrouwbaarheid ondersteunen. Dit door onder ander systeem redundantie en monitoring te ondersteunen.

## 7.4. REST constraints

Aan de hand de properties heeft Fielding de volgende Constraints opgelegd:

**Client-Server**

Communicatie verloopt tussen client en server, vanuit het oogpunt van simpliciteit en functionele verantwoordelijkeheid. De verantwoordelijkheid van de server is dat zij de data vasthoud en luistert naar vragen van de clients die deze data nodig heeft. Fielding maakt hier een contrast tussen Client-Server en complexe gedistribueerde of samengestelde systemen.

**Stateless**

De server houd geen staat of sessies bij en clients moeten zelf hun logica bepalen. Dit maakt communicatie tussen client en server zichtbaar, betrouwbaar en schaalbaarder dan applicaties waar de server de staat van de sessie bijhoud.

Het nadeel is dat sommige requests meerdere interacties vereisen en dus meerdere keren verzonden moet worden.

**Cache**

Uitwisseling van informatie kan worden gecached door de client of netwerkserver. Dit bespaart het netwerk op bandbreedte en calls naar de backenden bespaart de client dus wachttijd. De cachebaarheid van de response is sterk afhankelijk van de request. De response moet daarom zelf aangeven ‘of en hoe lang’ haar informatie cachebaar is.

**Layered system**

Door de netwerkarchitectuur gelaagd op te zetten is het mogelijk de achterliggende systemen afzonderlijk te ontwikkelen en te onderhouden. Als voorbeeld kan er achter de REST interface een servicebus liggen die oudere systemen met nieuwere systemen koppelt. Dit maakt dat de achterliggende implementatie van services aangepast kan worden zonder de interface aan te hoeven passen.

**(optioneel) Code-On-Demand**

Clients kunnen versimpeld worden door functionaliteit via de REST interface aan te bieden. Dit houdt in dat bijvoorbeeld de interpretatielogica op de REST interface beschikbaar is en dus niet door elke client afzonderlijk hoeft worden geïmplementeerd en onderhouden hoeft te worden.

De front-end client is eigenlijk een vorm van Code-On-Demand. Alle functionaliteit (code) om met de server interacties uit te voeren wordt on-demand door de browser binnengehaald.

Code on demand is optioneel in de architectuur van Fielding, omdat het voor agents (firewalls, caches) niet makkelijk te interpreteren is.

**Uniform interface**

Belangrijk aan REST is dat REST vier contraints aan de interface stelt.

De eerste Uniform Interface constraint is dat REST aangeeft waar resources zich bevinden via een URI (U*niform Resource Identifier*) en dat deze identificatie consistent blijven door de levenscyclus.

De tweede is dat resources kunnen worden aangepast op de relevante URI. Dit door gebruik van http verbs zoals o.a. ”GET”, “PUT” en “DELETE”.

De derde constraint is dat resources verschillen van hun representatieen representaties in verschillende mediatypes kunnen worden aangeboden zoals (*JSON, SOAP en/of XML*). Mediatypen kunnen ook samengevoegd worden in de volgende vorm;

top-level type name / subtype name [ ; parameters ]

top-level type name / [ tree. ] subtype name [ +suffix ] [ ; parameters ]

Bron: <https://en.wikipedia.org/wiki/Media_type>

Resultaat is een breed scala aan mediatypen die gedeeltelijk kunnen worden geïnterpreteerd. In het geval van een Product in de Diract Concentrator kan deze als HAL, JSN en een Diract.Concentrator.v1.Product worden geïnterpreteerd;

application/vnd.diract.concentrator.v1.product.HAL+json

Elke volgende bericht bevat genoeg informatie om deze op de client te kunnen behandelen. Berichten geven duidelijk aan welk mediatype het betreft zodat een parser ze goed kan behandelen. Dit maakt het mogelijk om samengestelde mediatypes te creëren en aan te bieden. De representatie is verantwoordelijk voor het aangeven van het resourcetype.

De laatste constraint over de interface van de webapplicatie is HATEOAS en wordt in het volgende deelhoofdstuk beschreven.

## 7.5. HATEOAS en HAL

De laatste constraint heet HATEOAS (*Hypermedia As Extension Of Application State*) en is vergelijkbaar met hoe een browser pagina’s interpreteert. De browser of client doet geen aannames over hoe het over de WebAPI navigeerten is afhankelijk van de links die de server verstuurd.

Deze constraint is nogal extreem. Een echte implementatie van HATEOAS zou resulteren in het bouwen van een browser binnen de browser zelf. De kern en waarde van HATEOAS zijn echter zeer helder; de WebAPI beschrijft met HATEOAS zijn eigen functionaliteiten zodat er geen extra documentatie geschreven hoeft te worden en programmeurs vloeiend de WebAPI kunnen implementeren en in hun applicatie kunnen gebruiken.

Daarnaast kan een client tegen de metadata van de applicatie worden geprogrammeerd. Dit houd in dat, hoewel de eigenschappen van een object kunnen veranderen of optioneel zijn, bepaalde onderdelen in stand kunnen blijven.

Om HATEOAS te implementeren zijn er een paar mediatypen waaruit gekozen kan worden, waarvan er geen enkele als officiële standaard door werkgroepen is aangenomen. De meest vooraanstaande is HAL (*hypermedia application language*).

HAL is een internet mediatype dat twee representaties kent, XML en JSON. Binnen een HAL representatie worden in ieder geval de volgende data meegegeven.

* **\_link** Links naar gerelateerde resources en methodes
* De data zelf
* **\_embedded** Gerelateerde data
  + **\_link** links naar gerelateerde data van de gerelateerde data

In zekere zin is dit niet anders dan hoe een menselijke gebruik over het web navigeert. Echter in plaats dat er uitwisseling van hele webdocumenten plaatsvind, vind er alleen uitwisseling van data en navigatiei-nformatie plaats. Een client kan bij wijze van spreken over de data heen surfen.



Voorbeeld van het HAL mediatype.

Natuurlijk kan er tegen een HATEOAS met remote procedure calls worden geprogrammeerd, of HATEOAS functionaliteit met een header worden uitgezet zodat overhead wordt bespaard. Maar door functionaliteit als symbolische links te geven, kan de client in ieder geval gedeeltelijk met de API meegroeien.

## 7.6. SOAP

SOAP (*simple object access protocol*) is een messaging protocol dat gebruikt word voor het uitwisselen van berichten. Het is een simpel protocol, maar uiteindelijk complexer in als WebAPI dan REST.

SOAP berichten bestaan in de kern uit een envelop die aangeeft dat het om een SOAP message gaat, een aantal headers met meta-informatie en een body met data. Door de jaren heen zijn er verschillende interpretatielagen aan SOAP toegevoegd, die het mogelijk maken onder andere *transacties* uit te voeren en transportzekerheid te bieden . Daarmee werd SOAP te complex om de naam Simple Object Access Protocol nog waar te maken.

Het voordeel van SOAP boven REST is dat SOAP veel werd toegepast voor API. Er is daarmee veel legacy vraag bij betrokken partijen. SOAP bied daarnaast een meer robuuste infrastructuur en security in de vorm van transacties en encryptie. Dit is voor sommige applicaties een absolute must, maar voor anderen helemaal niet. Voornamelijk grotere bedrijven zijn van SOAP overgestapt op een primair REST gebaseerde architectuur.

## 7.7. Deelconclusie

RESTful WebAPI’s zijn voor programmeurs makkelijk om tegen te implementeren en te integrerenen vragen om minder onderhoud in documentatie. Daarom zijn de meeste grote WebAPI’s RESTful uitgevoerd. REST is simpelweg de meest pure vorm van het HTTP protocol gebruiken, wat niet verwonderlijk is aangezien de ontwerper van REST ook het HTTP protocol heeft ontworpen.

SOAP bied functionaliteiten boven REST, in de vorm van encryptie en het ondersteunen van transacties. De vraag is echter of deze per se nodig zijn in de concentrator applicatie. Naar mijn zeggen is dit niet het geval. In het geval dat dit wel nodig blijkt te zijn, kan er altijd nog een extra API naast worden gezet.

REST bied een flexibelere API aan dan SOAP. Het vereist geen additionele documentatieen kan daarmee makkelijker geïmplementeerd worden. Voor applicaties die geen transacties vereisen en object atomisch tussen systemen moeten worden verplaatst komt SOAP met veel overhead en complexiteit. Diract wil als bedrijf flexibeler functionaliteit kunnen opleverenen REST kan hierbij ondersteuning bieden.

Er valt een rare case te verzinnen waar er een frankenstein SOAP over REST kan worden gemaakt. REST is zo een abstract en theoretisch geheel dat het niet ondenkbaar is dit te doen, maar in een normale omgeving kan er altijd nog een andere SOAP WebAPI naast worden gezet.

REST is niet RESTful zonder HATEOAS. Het voordeel van REST boven SOAP is weg zonder het implementeren van een HATEOAS type. HATEOAS kent verschillende Media Types, waarvan geen enkele daadwerkelijk officieel geregistreerd is. HAL maakt hier wel de meeste kans open word door grote corporaties als Microsoft gebruikt. Daarnaast bestaan er HAL client-libraries voor javascript en andere talen.

De kracht van HATEOAS ligt ook in het programmeren tegen symbolische urls die nieuwe functionaliteit opleveren. In het geval van het HAL mediatype, moet er zo veel mogelijk tegen de symbolische links worden geprogrammeerd.

## 7.8. In het proof-of-concept

In het proof of concept is er een basis backend gemaakt. Deze slaat informatie op met behulp van *Entityframework 7* naar een database instantie van *SQL Server 2014*, maakt gebruik van data repositories.

De webapplicatie is verder opgebouwd op basis van *.NET Core*, *ASP.NET 5* en een *Kestrel* webserver. Op het moment van schrijven zijn deze drie producten nog in Beta en niet in gebruik bij Diract, maar dit was nodig om een goede impressie van modern tools te krijgen. Er is een kleine fout begaan door gebruik te maken van de *Kestrel w*ebserver en .NET Core, maar op zich zijn dit subsets van IIS en .NET 4.6 respectievelijk.

De webservice maakt gebruik van Resources en HTTP verbs. Een HATEOAS mediatype als HAL is niet geïmplementeerd, door het ontbreken van HATEOAS libraries voor *.NET Core* en een gebrek aan tijd deze zelf te maken.

Alle services zijn rond de Product Resource gemaakt, die zelf beschikbaar is op de URL “api/product”. Een GET request naar “api/product” resulteert in een lijst van producten, standaard de eerste 25 producten. Andere query parameters zijn onder andere

* min\_lastModified
* max\_lastModified
* min\_created
* max\_created
* SKU
* Name
* ProductTypeID
* BrandID
* Page
* AttributeID

Soortgelijke functionaliteit bestaat voor aan product gerelateerde resources *api/brand?Name={parameter}* en in het geval van koppeltabellen *api*/*product*/{*productID*}/*productAttribute/{productAttributeID}*

# Conclusie

Diract is een dynamisch bedrijf dat zich richt op het ondersteunen van de detailhandel en heeft speciale aandacht voor het proces van deze bedrijven. De software van Diract beslaat meerdere processen binnen de bedrijfsvoeringen ondersteund onder andere de distributie, verkoop, bevoorrading en ordermanagement met software. Deze software kan met elkaar geïntegreerd worden, maar ook met software van andere fabrikanten en serviceaanbieders.

De kracht van Diract ligt dan ook op het op maat maken van oplossingen voor de bedrijfsprocessen van klanten. Diract wil zich nog sterker maken in het maken van flexibele en aanpasbare software door hun software ook als een service (SaaS) te bieden. Bijhorend is een nieuwe, gelaagde architectuur met een servicelaag voor binnenkomende en uitgaande informatie. Dit brengt nieuwe problemen en uitdagingen met zich mee, zoals een hogere werklading voor de front-end servers en een bredere gebruikersbasis die zich ook in het buitenland kan bevinden.

Daarnaast wil Diract haar haar software ook goed op mobiele platformen kunnen aanbieden. De huidige front-end opzet is enigszins gedateerden kan niet goed op mobiele platformen worden gebruikt. Dit is niet langer acceptabel voor de gebruikers van de software, die zich dynamisch op de werkvloer willen begeven en niet langer aan een desktop vast willen zitten.

Specifiek is er gekozen om voornamelijk voor het product Concentrator een nieuwe front-end opzet te onderzoeken. Dit is gedaan vanuit een oogpunt een concrete usecase te maken, maar de meeste informatie uit dit document is ook relevant voor de andere nieuw ontwikkelde producten van Diract met dezelfde functionele en niet-functionele eisen.

Er is een keuze gemaakt de front-end op de machine van de client te draaien. Dit omdat de hardware van zelfs budget mobiele apparaten krachtig genoeg is om wat zwaardere websites te tonen. Daarnaast word javascript efficiënter afgehandeld door middel van de nieuwe cross-platform V8 engine. De gemiddelde telefoonen zeker een de gemiddelde tabel, is meer dan capabel een complexe webclient te draaien. Dit verschuift de rekencapaciteit van de server naar de clienten ontkoppeld de data en representatie.

Daarnaast zijn er tegenwoordig gratis frameworks beschikbaar die het mogelijk maken om een webclient op een stabiele, efficiënte, uitbreidbare en onderhoudbare manier te ontwikkelen. Uit een voorselectie single-page-application (micro)frameworks is Angular gekozen. Angular is geselecteerd omdat deze het meeste comfort en functionaliteit voor ontwikkelaars bood en daarnaast stabiliteit en interactie aan de gebruikers bood. Andere frameworks die niet uit de geselecteerd zijn waren ExtJS, dat voornamelijk op desktopbrowsers was gerichten ReactJS dat geen architectuur bood.

Naar het doel van gebruikersvriendelijkheid op mobiele apparaten is er ook een CSS framework geselecteerd. Deze is verantwoordelijk voor het correct tonen van het uiterlijk op verschillende beelddimensies en verschillende browsers. Bootstrap is hieruit geselecteerd als beste keus, vanwege de grote functionaliteit, integratie met Angular en het makkelijk aanpasbare uiterlijk. Dit is echter geen harde keuze, omdat andere frameworks (voornamelijk Foundation 6) dezelfde mobiele functionaliteiten boden.

Als keuze voor WebAPI is REST boven SOAP gekozen. SOAP bood een stabiele, betrouwbare communicatie en legacy support ten kosten van complexe integratie en performance. REST bood een theoretische, maar zeer goed doordachte manier van informatie-uitwisseling waar de representatie los van de data staat. Daarnaast is een goede REST implementatie zelf-documenterend en kan makkelijk door een externe ontwikkelaar worden geïntegreerd. REST heeft hiermee SOAP overwonnen, wat ook te zien is aan de trend in API ontwikkeling. De theoretische achtergrond van REST sluit zelfs niet uit dat SOAP in REST kan worden gebruikt.

Belangrijk is dat REST een HATEOAS mediatype vereist. De voordelen van REST boven SOAP kunnen zonder een HATEOAS mediatype niet worden gerealiseerd. Hiervoor is HAL (*hypertext application language*) de enige realistische optie, omdat deze ondanks het ontbreken van een standaard het beste door het bedrijfsleven word ondersteund.

Met uitzondering van HATEOAS is alles in een proof-of-concept uitgewerkt. Deze word in het volgende hoofdstuk behandeld.

# Proof-of-concept

Naar aanleiding van het onderzoek is het proof of concept gemaakt. Het proof-of-concept bestaat uit een C# Solution file met een project “NEXT”.

Het NEXT project bestaat uit drie onderdelen;

* Een client
* Een webserver
  + Statische hosting van de client
  + Hosting van de REST WebAPI
* Build configuratie voor de client en webserver

## Client

De client is gemaakt met een combinatie van Angular voor de logicaen Bootstrap voor het uiterlijk.

De client is meertaligen nieuwe talen kunnen makkelijk worden toegevoegd. De vertalingsbestanden bestaan uit een JSON file die dynamisch kunnen worden opgehaald en ingelezen door de client. Momenteel worden deze nog statisch gehost, maar het is een koud kunstje de Angular vertalingsservice aan te passen naar een beter onderhoudbaar systeem.

De client is responsive en kan dus op meerdere platformen comfortabel worden gebruikt. Pagina elementen passen hun dimensies aan naar de dimensies van het beeldscherm. In de productDetail pagina is dit gemaakt met behulp van de dependency “Angular-Dynamic-Layout”. Deze dependency bevat tevens de functionaliteit om elementen lokaal op attribuut te filteren, alhoewel dit niet is geïmplementeerd.

Het uiterlijk van de client is door middel van een alternatief Bootstrap script verwezenlijkt. Deze was gratis en kwam met extra icoontjes. Het lijkt dus niet op Bootstrap, maar het is het daadwerkelijk wel.

## Webapplicatie

De webapplicatie is met *.NET Core* als SDK, *Kestrel* als webserver en *ASP Core* als webframework gemaakt. Deze componenten zijn met een strakkere architectuur geoptimaliseerd voor cloud-omgevingen. Vergeleken met ASP 4 applicaties zijn ze modulairder opgezeten hoeven minder te laden. *.NET Core* kan in combinatie met *.NET 4.6 worden gebruikt.*

De webserver host de client vanaf een statische locatie. Dit is een relatief simplistische opzet, maar werkt goed en is niet onveilig.

Daarnaast host de webserver een WebAPI. Deze is REST maar niet RESTful; het mist het kritieke HATEOAS dataformaat om de data inzichtelijk te maken en daadwerkelijk de applicatiestaat op de client te verwezenlijken. Het ontbreken van HATEOAS is voornamelijk te wijten aan het gebruikt van *.NET Core*, dat nog in beta is en nog geen libraries van externe ontwikkelaars heeft.

De WebAPI heeft verschillende resources, allemaal gericht op product en gerelateerde datamodellen. De WebAPI kan lijstjes selecteren op alle velden van producten daarnaast op de velden van gekoppelde objecten. In ieder geval kunnen er producten worden gelezen, toegevoegd, verwijderd en aangepast worden. Andere modellen kunnen alleen uitgelezen worden omdat deze verder niet relevant aan de opdracht waren.

## Verdere noten

Er nog tangentieel onderzoek naar andere aspecten van de client gedaan, zoals naar security en authenticatie. Dit is echter niet meegenomen in het onderzoeksrapport omdat deze niet grondig genoeg was onderzocht of geïmplementeerd.

Authenticatie gebeurt momenteel met een luttele cookies met een waarde die aangeeft of de gebruiker ingelogd is.

Cross-site-scripting aanvallen zijn af te wenden met het gebruik van de Angular Sanitize module. Deze weert het interpreten van scripts en HTML tags in output.

Denial-of-service aanvallen valt goed tegen te verzekeren met authenticatie van gebruikersen resourceintensieve functies van anonieme gebruikers te weren. Distributed-denial-of-service aanvallen kunnen met netwerkinfrastructuur en het inkopen van servertijd worden ontkracht.

# 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.

**CSS** *Cascading Style Sheet*

Scripttaal dat verantwoordelijk is voor het uiterlijk van een webpagina. Bevat functionaliteiten voor het bepalen van kleuren van elementen, animaties, grootte en positie.

**Dependency**

Onderdeel software dat door derden is geschreven. Het betreft meestal breed toepasbare functionaliteit die door ervaren en betrouwbare ontwikkelaars is gemaakt.

**Dependency Injection**

Maakt het de ontwikkelaar makkelijker dependencies in *Runtime* te onderhouden.

**Dependency Manager**

Houd bij welke dependencies, welke versie van deze dependency er verwacht wordten kan deze automatisch van een repository op het internet ophalen. Versimpeld en automatiseerd het bijhouden en toevoegen van dependencies.

**HTML** *HyperText Markup Language*

Op XML gebaseerde taal die bedoelt is om de inhoud van een webpagina te definiëren. Het houdt zich, idealiter, niet bezig met de uiterlijke kenmerken van een webpagina, dat is toebestemd aan een CSS script.

**RWD** *Responsive Web Design*

Een manier van pagina-opbouw waarbij het webdocument haar elementen van grootte en positie kunnen veranderen. Hierbij kan hetzelfde webdocument effectief op alle beeldschermafmetingen worden gebruikt.

**Mobile** *informele term*

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*.

**Jasmine**

Aanname framework voor javascript, gericht op unit-testen. Kan worden gebruikt voor de *Karma* testrunner.

**Karma**

Testrunner voor javascript die onder andere unit-tests kan draaien.

**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.

**Omni-channel** *alle(omni) kanalen(channel)*

Een klant kan meerdere verkoopkanalen (winkels) bezitten en beheren. Deze winkelformules kunnen online en offline opereren. Het idee achter omni-channel is dat meerdere winkelformules van dezelfde holding dezelfde distributiekanalen kunnen delen. Diract is hier specialist in.

**REST** *Representational State Transfer*

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

**SaaS** *Software as a Service*

Software as a Service houdt in dat klanten de capaciteiten van de software als een service willen gebruiken. Dit houd in dat ze alle infrastructuur en hosting uitbesteden aan de software partij, die de service online beschikbaar stelt.

**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.

**Transactions** *transacties*

Atomic *(ondeelbare)* transacties maken het mogelijk dezelfde verandering betrouwbaar tussen systemen uit te voeren. Dit is nodig bij processen waar de data ACID (Atomair, Consistent, Isolated, Durable) dient te zijn.

**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** *technisch*

Draad van opeenvolgende instructies aan de processor

**Back-end**

De achterliggende logische lagen van een applicatie bijv. databases of businesslagen die niet door een gebruiker te ‘zien’ zijn.

**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**](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

<https://github.com/mikekelly/hal_specification/blob/master/hal_specification.md>