7-6-2016

Sieger Falkena

Hogeschool van amsterdam – Diract-it

Responsive front-end

Responsive front-end voor Diract-IT

# 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

Inhoudsopgave

[Afstudeerscriptie 1](#_Toc453084199)

[Inleiding 6](#_Toc453084200)

[Diract 6](#_Toc453084201)

[Concentrator 6](#_Toc453084202)

[Warehouse Management System 6](#_Toc453084203)

[Andere producten van Diract 6](#_Toc453084204)

[Probleemstelling 7](#_Toc453084205)

[Doelstelling 7](#_Toc453084206)

[Onderzoeksmethoden 7](#_Toc453084207)

[Wat is een goede front-end opzet voor Diract? 7](#_Toc453084208)

[Wie zijn de gebruikers en waar hechten zij waarde aan? 8](#_Toc453084209)

[PACMAD 8](#_Toc453084210)

[Deelconclusie 9](#_Toc453084211)

[Proof-of-concept 10](#_Toc453084212)

[Wat zijn mobiele limitatie en hoe kunnen deze worden gecompenseerd? 11](#_Toc453084213)

[Limitaties 11](#_Toc453084214)

[Visueel 11](#_Toc453084215)

[Invoer 12](#_Toc453084216)

[Netwerk 13](#_Toc453084217)

[Hardware 13](#_Toc453084218)

[Voordelen 14](#_Toc453084219)

[Touchscreen 14](#_Toc453084220)

[Microfoon 14](#_Toc453084221)

[Camera 14](#_Toc453084222)

[Tactiel / locatie gebaseerde informatie 14](#_Toc453084223)

[Deelconclusie 14](#_Toc453084224)

[Uitgewerkt in proof of concept 15](#_Toc453084225)

[Hoe kunnen we de webapplicatie structureren om de mobiele toegankelijkheid te vergroten? 15](#_Toc453084226)

[Details abstraheren 16](#_Toc453084227)

[Wat is NodeJs en waarom gebruikt iedereen het? 17](#_Toc453084228)

[Node.JS 17](#_Toc453084229)

[JavaScript webservers 18](#_Toc453084230)

[Dependency management 18](#_Toc453084231)

[Karma test framework 18](#_Toc453084232)

[Transpilers 19](#_Toc453084233)

[Build-management 19](#_Toc453084234)

[CSS preprocessors 19](#_Toc453084235)

[Overige tools 19](#_Toc453084236)

[Integratie met Visual Studio 2015 19](#_Toc453084237)

[Conclusie 20](#_Toc453084238)

[In het proof-of-concept 20](#_Toc453084239)

[Waar word de pagina gerenderd? 21](#_Toc453084240)

[Conclusie 21](#_Toc453084241)

[Proof-of-concept 22](#_Toc453084242)

[Welke SPA frameworks sluiten het beste aan bij de vereisten? 23](#_Toc453084243)

[AngularJS 24](#_Toc453084244)

[Voordelen 24](#_Toc453084245)

[Nadelen 24](#_Toc453084246)

[Sencha ExtJS 25](#_Toc453084247)

[Voordelen 25](#_Toc453084248)

[Nadelen 25](#_Toc453084249)

[ReactJS 27](#_Toc453084250)

[Voordelen 27](#_Toc453084251)

[Neutraal 27](#_Toc453084252)

[Nadelen 27](#_Toc453084253)

[Deelconclusie 28](#_Toc453084254)

[Proof-of-concept 28](#_Toc453084255)

[Welke CSS framework past het beste bij concentrator? 29](#_Toc453084256)

[Waarom gebruiken ontwikkelaars een front-end framework? 29](#_Toc453084257)

[Selectie van frameworks 29](#_Toc453084258)

[Material Design 30](#_Toc453084259)

[Voordelen 30](#_Toc453084260)

[Goed voor mobiele applicaties 30](#_Toc453084261)

[Nadelen 30](#_Toc453084262)

[Twitter Bootstrap 30](#_Toc453084263)

[Voordelen 30](#_Toc453084264)

[Nadelen 30](#_Toc453084265)

[Metro-ui 31](#_Toc453084266)

[Voordelen 31](#_Toc453084267)

[Nadelen 31](#_Toc453084268)

[Foundation 6 31](#_Toc453084269)

[Voordelen 31](#_Toc453084270)

[Nadelen 31](#_Toc453084271)

[Conclusie 31](#_Toc453084272)

[Proof-of-concept 31](#_Toc453084273)

[Hoe gebruik ik de javascript client tegen de service laag? 32](#_Toc453084274)

[REST 32](#_Toc453084275)

[Performance 32](#_Toc453084276)

[Schaalbaarheid 32](#_Toc453084277)

[Simpliciteit 32](#_Toc453084278)

[Aanpasbaarheid 32](#_Toc453084279)

[Zichtbaarheid 33](#_Toc453084280)

[Overdraagbaarheid 33](#_Toc453084281)

[Betrouwbaarheid 33](#_Toc453084282)

[Contraints 33](#_Toc453084283)

[Client-Server 33](#_Toc453084284)

[Stateless 33](#_Toc453084285)

[Cache 33](#_Toc453084286)

[Uniform interface 33](#_Toc453084287)

[Layered system 34](#_Toc453084288)

[(optioneel) code on demand 34](#_Toc453084289)

[SOAP 34](#_Toc453084290)

[Deelconclusie 35](#_Toc453084291)

[Hoe implementeer ik de authenticatie van het systeem? 36](#_Toc453084292)

[Deelconclusie 36](#_Toc453084293)

[Hoe werkt security en authentication? 36](#_Toc453084294)

[Conclusie 36](#_Toc453084295)

[Verklarende woordenlijst 37](#_Toc453084296)

[Bibliografie 39](#_Toc453084297)

# 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. 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 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, 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 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. 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 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 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

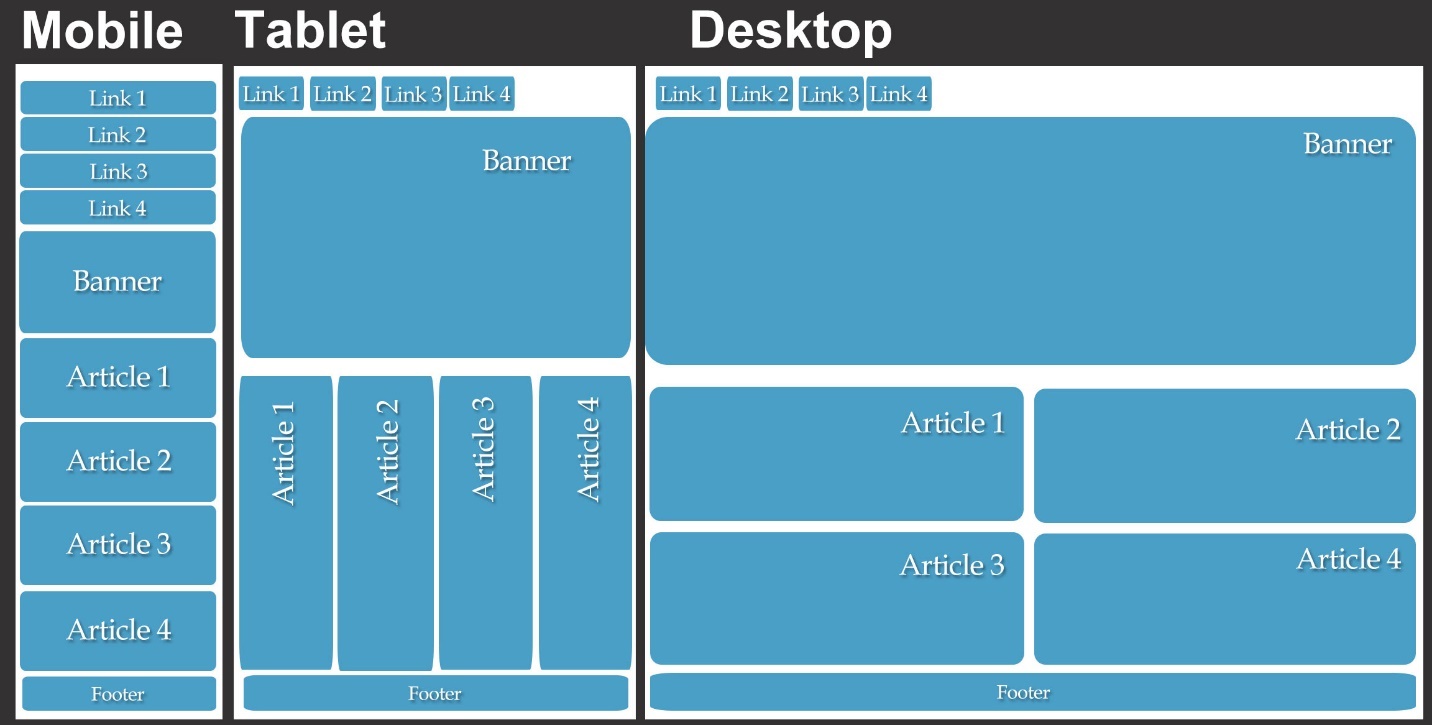
Visueel hebben mobiele apparaten een over het algemeen kleiner beeldscherm en meer pixels/cm2. Dit maakt dat elementen (knoppen en tekst) die als goed bruikbaar op desktop worden ervaren, als te klein worden ervaren op mobile. Vice-versa kunnen tekst en knoppen voor mobile als te groot worden ervaren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

### 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.

|  |  |  |
| --- | --- | --- |
|  | **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 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.

### Tactiel / 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.

# 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 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.

## 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 de buildtools en webservers beschikbaar.

## 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 en sub-processen die systeemresources aanspreken en IO afhandelen.

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 veranderd.

Dependency management versimpelt het proces van deployment naar enige servers. 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.

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 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 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.

## 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.

## CSS preprocessors

Cascading stylesheets (CSS) scripts zijn puur verantwoordelijk voor het uiterlijk van representatie. CSS regelt de grootte van lettertypes en

## 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 en word goed door VS15 ondersteund.

Gulp wordt als buildmanager door VS15 gebruikt. Andere keuzes (grunt, etc.) ondertsteund 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 frameworks en vereisen een specifieke vorm van programmeren.

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.

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 is gemaakt dat Typescript goed in het licht zette en ik zelf geen al geen ervaring met javascript ontwikkeling had.

# 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 op, waarna deze verzonden wordt.

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 ‘plakte’. Dit hield in dat een webpagina gedeeltelijk in kon worden opgebouwd. Het voordeel was een betere structuur voor alle

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 lang niet goed samenwerkte met SPA frameworks. Hierdoor is het moeilijk de bezoekersaantallen en interesses vbij te gouden. De Dot os moet relevant aam de businesscase van Diract.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 | |
|  | Clients met moderne hardware | | Clients met sterk verouderde hardware | |

## 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 deze meteen gemaakt en is het primaire communicatiekanaal voor de client. Bij serverside of een mengvorm zou in ieder geval de data via een ander kanaal dan de data moeten verlopen.

Er kan nog worden gekozen om enkel de initiële indexpagina te renderen. Dit zou de initiële laadtijd naar beneden brengen, en 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.

Vanwege een keuze in gebruik van .NET Core, zijn er nog geen REST frameworks die HATEOAS gebruiken. Er is daarom een klein stukje HATEOAS geïmplementeerd, dat enkel als illustratie voor de applicatie dient.

# 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.

Er zijn daarom 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.pngAngularJS

Angular1 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.

Angular1 komt met een 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

Onofficieel het SPA-framework van google en word 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. De kwaliteit van deze externe libraries is zeer goed, en voorzien genoeg functionaliteit voor de meeste usecases.

Angular is binnen Diract al eerder successvol 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.

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.

Angular 2 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 overgeport 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.

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.

### Voordelen

Ext.JS 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.

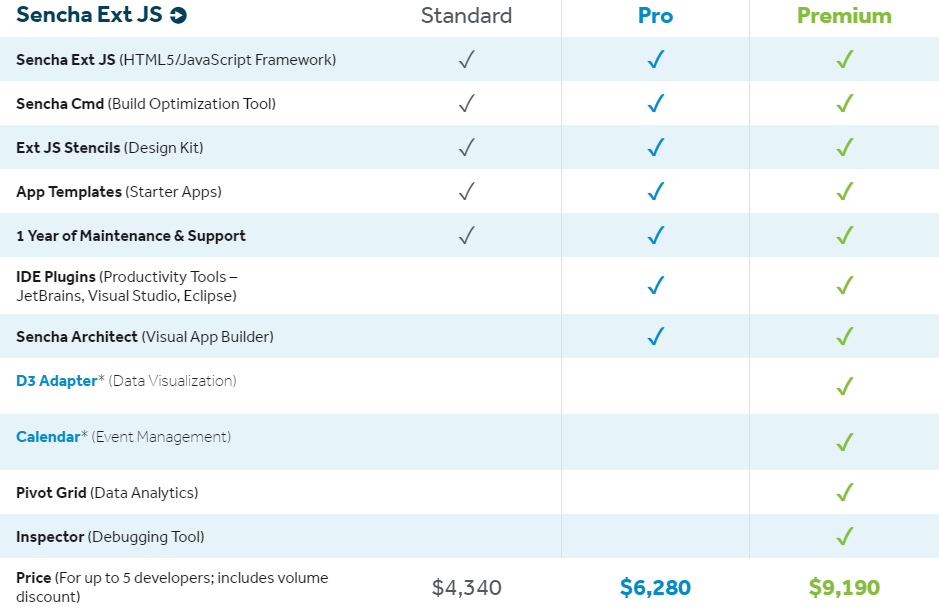
ExtJS bied oplossingen voor de nadelige eigenschappen van javascript voor enterprise ontwikkeling. Ext.Js bied een klasssestructuur aan, en

### Nadelen

ExtJS is maar gedeeltelijk responsive. De lay-out manager is niet responsive te krijgen, en 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 aanmaken en 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 goed 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 facebook. ReactJS beslaat een groot aan

### Voordelen

ReactJS is gratis.

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) 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 niet optimaal. JSX is een gedrogt dat representatie en logica samen combineert. Templates staan in de javascript controllers en er weinig sprake van scheiding tussen uiterlijk en logica.

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 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 taak, en 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 overbrugd, en 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 omgeving, en 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 laag en terugkomende bezoekers.

Daarnaast vereist ook ReactJS dat geheugen-maangement 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 Angular2, en 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 standaard, en kennen een duidelijke verbetering.

Desondanks is Angular een van de meest functionele en stabiele frameworks, dat 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 uitgewerkt, en maakt met uitzondering gebruik van Angular standaard elementen.

Routing is met de Angular-UI-Routing module, die 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 ook met een Angular-UI onderdeel uitgewerkt. Met het grid kunnen producten in een lijst worden getoond en gefilterd op eigenschappen. De lijst maakt gebruik van pagination.

Data word met de $resource 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

## 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.

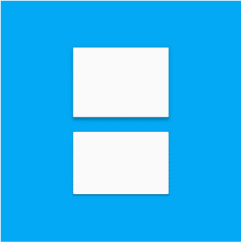
De ontwikkelaars van browsers zijn in een constante strijd over marktaandeel verwikkeld. Hoewel er standaarden aanwezig zijn, nemen browserontwikkelaars het niet al te nauw met deze standaarden om concurrentie tegen te werken of zelf betere functionaliteit aan te kunnen bieden. De interpretatie van scripts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

## Selectie 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 inzicht naar nieuwe user interface designs. Kenmerkend is het platte uiterlijk en veel gebruik van schaduw om de illusie van papieren onderdelen te geven. Material design komt met heel veel design invloed

### Voordelen

### Goed voor mobiele applicaties

De best documentatie

### Nadelen

Zwaardere hardware-eisen

Uitgesproken google, zonder alternatieven

De stijl van MD is niet makkelijk aan te passen.

## 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. Het neemt 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 een 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 leveren ook ondersteuning bij het implementeren van hun software.

### Voordelen

Voor ontwerpers ontworpen door ontwerpers. Zurb is ontwikkeld vanuit expertise enervaring in grafisch ontwerp. Zurb gebruikt haar eigen software ook in-house, en is ondertussen toe aan versie 6.

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 verkopen.

## Conclusie

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.

## 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 gebruiken. 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.

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

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

De service laag is de applicatie laag die exclusief verantwoordelijk is voor de invoer en uitvoer van data. Deze houdt

In voorgaande hoofdstukken is er besloten een client tegen een webAPI te implementeren. Deze keuze kwam voort uit een combinatie van het gebruikerspatroon van klanten en kostenbesparing.

## REST

REST staat voor representational state transfer en is bedacht 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 IT activiteiten. Fielding ontwierp verder het HTTP protocol aan de hand van zijn proefschrift.

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

### 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 architectuur 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 terwijl het systeem blijft draaien.

### Zichtbaarheid

De netwerkarchitectuur moet open inzichtelijk zijn voor andere systemen. Veel netwerkinfrastructuur is afhankelijk van deze informatie, en 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.

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

## Contraints

### Client-Server

Communicatie verloopt tussen client en server, vanuit het oogpunt van simpliciteit. De server slaat praktisch gezien de data op.

### 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 staat in bijhoud.

Het nadeel is dat sommige requests meerdere interacties vereisen, en dus meerdere keren verzonden moet worden. Ook aanpassingen op de interface van de server hebben gevolgen voor het correct uitvoeren van taken op de clients.

### Cache

Uitwisseling van informatie kan worden gecached door de client of netwerkservers. Dit bespaard het netwerk op bandbreedte, en bespaard de client op wachttijd. Dit is zeer afhankelijk van het request dat er gedaan wordt en moet dus expliciet in de response worden aangegeven of het cachebaar is een hoe lang het cachebaar is.

### Uniform interface

Belangrijk aan REST is dat rest drie contraints aan de interface stelt.

De eerste Uniform Interface constraint is dat REST aangeeft waar resources zich bevinden via een URI, wat voor data het is (http content-type) en hoe deze gebruikt kan worden. Deze locaties zijn permanent, maar resources kunnen verwijderd worden.

Het tweede constraint is dat de client en server weten over wat voor een resources gaat en hoe deze geïnterpreteerd moeten worden. Dit gebeurt door middel van een mediatype.

Het derde constraint is waar de data geprocessed word. Dit is de enige vrije keuze die een REST architect heeft.

* De data kan door de server worden geformat en daarna naar de client worden verstuurd.
* Verstuur de client samen met de capaciteiten om deze te interpreteren en weer te geven
* Verstuur rauwe data samen met relevante metadata zodat de client deze zelf kan interpreteren en bewerken.

### 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 houd 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 om met de server interacties uit te voeren word on-demand door de browser binnengehaald.

Code on demand is optioneel, omdat de code niet makkelijk door agents (firewalls, caches) te interpreteren valt.

## 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.

# Hoe werkt security en authentication?

De voornaamste risico’s voor een webapplicatie zijn distributed-denial-of-service (DDOS) en cross-site-scripting (XSS) aanvallen.

# Conclusie

Diract-IT is een dynamisch bedrijf dat zich richt op het ondersteunen van de detailhandel en heeft speciale aandacht voor het proces van deze bedrijven. De grootste kracht van Diract ligt in het aanpassen van een standaardproduct naar de processen van de klant.

Diract wil zich nog sterker maken in het aanbieden van flexibele en aanpasbare software voor hun klanten door hun software naast on-premise installatie ook als een service te bieden. Daarnaast moet de user interface ook op mobiele apparaten te gebruiken zijn. Naar dit doel is dit onderzoek verricht.

Er is kort onderzoek gedaan naar gebruikers en hun omgevingen, waaruit bleek dat gebruikers voornamelijk geïnteresseerd zijn in effectief en efficiënt uitvoeren van hun taken. Deze taken kunnen ver van een desktop verwijderd zijn en bijvoorbeeld op een tablet of telefoon en het zou handig zijn als de gebruiker deze taken dynamisch op de werkvloer zou kunnen gebruiken.

Daarnaast kan het zo zijn dat gebruikers zich in andere taalzones begeven. De front-end is dan ook meertalig opgezet.

Voor het aanbieden van de front-end aan de klanten is er gekozen voor Angular2 als Single Page Application framework. De redenen voor het kiezen van Angular2 waren breed, maar Angular2 bied een combinatie van comfort voor ontwikkelaars en stabiliteit voor de gebruikers. ondanks dat het proof-of-concept met de vorige versie *(Angular1)* werkt het nieuwe Angular framework grotendeels hetzelfde met duidelijke verbeteringen in snelheid en nieuwe inzichten die door alom vertegenwoordigd gebruik van Angular1 zijn blootgelegd.

Voor het aanbieden van de applicatie op mobiele platformen is Bootstrap geselecteerd uit een selectie frameworks. De redenen om Bootstrap te gebruiken zijn de uitbreidingen die gratis door derde partijen word aangeboden, het bieden van goede voorbeelden, en integratie met Angular.

Het opstellen van pagina’s zal volledig op de client moeten plaatsvinden, omdat de kosten naar de client kunnen worden ge-externaliseerd kunnen worden en de nadelen zoals een lange initiële laadtijd niet veel invloed hebben op de het verwachte gebruik. Dit bespaart de organisatie geld, en daarnaast word er impliciet een REST API opgeleverd.

Er is een aanbeveling gedaan om de server van een REST API te voorzien om de client van data te voorzien, en daarnaast als communicatie voor externe systemen kan worden gebruikt. Specifiek aan REST is het implementeren van HATEOAS, dat het voor programmeurs makkelijker maakt om integraties met de Concentrator te maken en aan documentatie bespaard. HATEOAS kost overhead, maar kan met wat infrastructuur en instelling worden uitgezet om data sneller uit te kunnen wisselen.

# 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.

**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 wordt, en kan deze automatisch van een repository op het internet ophalen. Versimpeld en automatiseerd het bijhouden en toevoegen van dependencies.

**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**