# Afstudeerscriptie

Diract-it

Sieger Falkena

500602792

Inhoud

[Afstudeerscriptie 1](#_Toc449907792)

[Inleiding 4](#_Toc449907793)

[Probleemstelling 4](#_Toc449907794)

[Concentrator 4](#_Toc449907795)

[Single page application 4](#_Toc449907796)

[Service Oriented Architecture 4](#_Toc449907797)

[Responsive 4](#_Toc449907798)

[Wat is een goede single page application opzet voor diract-it? 5](#_Toc449907799)

[Wat zijn de eigenschappen voor mobiele gebruiksvriendelijkheid? 5](#_Toc449907800)

[PACMAD 5](#_Toc449907801)

[Wat zijn mobiele limitatie en hoe kunnen deze worden gecompenseerd? 5](#_Toc449907802)

[Visueel 6](#_Toc449907803)

[Invoer 7](#_Toc449907804)

[Netwerk 8](#_Toc449907805)

[Hardware 8](#_Toc449907806)

[Wat zijn de voordelen van een mobiel apparaat ten opzichten van een desktop? 9](#_Toc449907807)

[Touchscreen 9](#_Toc449907808)

[Microfoon 9](#_Toc449907809)

[Camera 9](#_Toc449907810)

[Tactile 9](#_Toc449907811)

[Hoe kunnen we deze voordelen gebruiken om data inzichtelijk te maken? 9](#_Toc449907812)

[Gestures / swipe 9](#_Toc449907813)

[Barcode readers 9](#_Toc449907814)

[text readers 9](#_Toc449907815)

[Hoe kunnen we de webapplicatie structureren om de mobiele toegankelijkheid te vergroten? 9](#_Toc449907816)

[Business rules 10](#_Toc449907817)

[Export- en importregels 10](#_Toc449907818)

[Producten en attributen 10](#_Toc449907819)

[Kanalen 10](#_Toc449907820)

[Details abstraheren 10](#_Toc449907821)

[Hoe werkt een service oriented architecture? 11](#_Toc449907822)

[REST-over-SOA 11](#_Toc449907823)

[SOAP 12](#_Toc449907824)

[Backend 12](#_Toc449907825)

[Deelconclusie 12](#_Toc449907826)

[Welke javascript frameworks sluiten het beste aan bij de vereisten? 12](#_Toc449907827)

[AngularJS 14](#_Toc449907828)

[Voordelen 14](#_Toc449907829)

[Nadelen 14](#_Toc449907830)

[Sencha ExtJS 15](#_Toc449907831)

[Voordelen 15](#_Toc449907832)

[Nadelen 15](#_Toc449907833)

[ReactJS 15](#_Toc449907834)

[Voordelen 15](#_Toc449907835)

[Neutraal 15](#_Toc449907836)

[Nadelen 16](#_Toc449907837)

[Deelconclusie 16](#_Toc449907838)

[Welke CSS framework past het beste bij concentrator? 17](#_Toc449907839)

[Material Design 17](#_Toc449907840)

[Bootstrap 17](#_Toc449907841)

[Foundation 17](#_Toc449907842)

[Deelconclusie 17](#_Toc449907843)

[Conclusie 17](#_Toc449907844)

[Bibliografie 17](#_Toc449907845)

# Inleiding

Dit onderzoeksrapport is geschreven naar aanleiding van mijn afstudeeronderzoek bij Diract-it.

## Concentrator

De concentrator is een product dat gebruikt word voor het importeren en exporteren van productinformatie naar fysieke winkels en webshops.

Dit importeren en exporteren gaat aan de hand van business-rules die voor de klant worden geprogrammeerd.

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

## WMS

WMS is een ander product van diract-it. Het Warehouse Management Systeem houdt inventaris van een warehouse bij, en komt met verschillende.

# 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) was al een single-page applicatie, en dat wil de opdrachtgever graag zo houden.

## Service Oriented Architecture

Binnen Diract is er een wens alle producten makkelijker aan elkaar te koppelen. Dit verloopt tot nu toe een beetje moeilijk. Bedrijfsbreed is er besloten alle nieuwe producten en productversies een servicelaag te geven die beide door eindgebruikers als andere systemen als aanspreekpunt kan worden gebruikt.

## Toegankelijk voor mobiele applicaties

De webapplicatie moet bruikbaar zijn op kleinere apparaten en displays. De concentrator is daarnaast een relatief complexe applicatie die veel functionaliteit bevat en veel interactie van de gebruiker vereist. Relatief gezien is het een complexere applicatie dan bijvoorbeeld een webshop.

## Herbruikbaar voor meerdere team

De wens vanuit Diract is dat de oplossing kan worden gebruikt voor in ieder geval twee teams.

# Wat is een goede front-end opzet voor diract-it?

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

## PACMAD

People at the centre of mobile application development

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

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-IT is de gebruiker een *product 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 managers* en *order managers* is een overlap in het takenpakket, en de functies worden in sommige organisaties door dezelfde werknemer gedaan.

De taken voor beide gebruikers ligt in het onderhouden en beheren van het informatiesysteem, ondersteund door professionals van Diract-IT. Voorbeelden zijn het inzien en aanpassen van productinformatie, het activeren en deactiveren van producten en het te hulp schieten van mislukte orders.

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

In het PACMAD model hebben deze factoren invloed op de volgende attributen; Effectiviteit, Efficientie, Bevrediging, Leerbaarheid, Onthoudbaarheid, Fouten en Cognitieve Ballast. De eerste drie (Effectiviteit, Efficientie en bevrediging ) zijn het best te objectief te beoordelen en het meest van invloed op gebruikerservaring.

De andere attributen; (leerbaarheid, onderhoudbaarheid 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.

# 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.

## Visueel

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

De oplossing komt uit het gebruik van Responsive Web Design. Door in plaats van pixels (resolutie) of meeteenheden (afmeting) een percentage te gebruiken is het mogelijk dezelfde content op meerdere platformen goed aan te bieden.

Er zijn meerdere frameworks waarmee RWD makkelijk te implementeren is. Voornaamste spelers zijn (Twitter bootstrap, Foundation en Material Design). Bootstrap is veruit het populairs en makkelijkst om toe te passen, terwijl foundation meer configuraties en keuzevrijheid bied. Aangezien de requirements vroegen om een functioneel ontwerp, en ik geen ontwerper ben, is er voor het proof-of-concept gekozen bootstrap te gebruiken.

Het is op een kleiner beeldscherm moeilijker een overzicht te tonen. Het opdelen van functionaliteit is moeilijker te realiseren dan een langere

## Invoer

Mobiele apparaten beschikken over een touchscreen als primair invoerapparaat. Het touchscreen functioneert beide als muis en toetsenbord. Dit hoofdstuk behandeld alleen de zwaktes van het touchscreen t.o.v. muis en toetsenbord; de sterktes worden in een volgend hoofdstuk behandeld.

|  |  |  |
| --- | --- | --- |
|  | **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 weet te gebruiken.

**Touchscreen als toetsenbord**

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

Typefouten zijn daarnaast makkelijk te maken op een touchscreen, voornamelijk vanwege de imprecisie van de vinger.

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.

De tweede oplossing is het gebruik van de sterke eigenschappen van de telefoon, en wordt in volgende hoofdstukken behandeld.

## 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 grote applicatiegrootte kan tot een trage gebruikerservaring 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 (Quagga.js later meer) hoeven niet vanaf het begin ingeladen te worden.

Door dependencies vooraf te laden kunnen wachttijden worden verminderd naar 0. Door vanuit de navigatieboom alle dependencies van connected edges te laden hoeven te laden hoeft er tijdens een gebruikersactie niets te worden geladen omdat alles al geladen is. Ook bij zoekresultaten met verschillende pagina’s kan alvast de volgende pagina worden geladen.

Bepaalde onderdelen kunnen ook worden gecached in de browser, server en localstorage. De browser doet cached bepaalde objecten automatisch, maar bepaalde onderdelen kunnen niet makkelijk gecached worden. Zoekqueries met pagination kunnen alvast de tweede pagina laden.

## Hardware

Mobiele telefoons beschikken over minder geheugen dan een computer voor dezelfde prijs. Onzorgvuldig gebruik van dit geheugen kan leiden tot frustratie van de gebruiker. Javascript is garbage-collected, maar een combinatie van lange sessies en slecht objectbeheer kunnen op langere duur het geheugen van elk apparaat volstouwen. Het is belangrijk een goed objectmanagement uit te voeren, en niet-relevante objecten uit het geheugen te laden.

Een goed framework zoals AngularJS of React kunnen een applicatie architectuur te forceren, en door een architectuur te forceren (MVVC) is het duidelijk welke dependencies (views, controllers en bijbehorende objecten) wel-of niet relevant zijn.

# Wat zijn de voordelen van een mobiel apparaat ten opzichten van een desktop?

## Touchscreen

Touchscreen zijn intuitiever in gebruik dan een muis en toetsenbord.

## Microfoon

## Camera

## Tactile

# Hoe kunnen we deze voordelen gebruiken om data inzichtelijk te maken?

## Gestures / swipe

Hammer.js – Angular-Gestures

## Barcode readers

QuaggaJS

## text readers

OCRAD

# 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 .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 complexiteit van de applicatie toeneemt met een hogere gelaagdheid.

*Bronnen van onoverzichtelijkheid*

*Tonen van relaties*

*Platte structuur*

*Hoge doorklikbaarheid*

Typische CRUD functionaliteit kan dezelfde views delen, en dit bespaard aan complexiteit voor de eindgebruiker.

Relaties tussen verschillende objecten

Binnen de concentrator zijn de volgende contexten aanwezig. De context van

### Business rules

### Export- en importregels

### Producten en attributen

### Kanalen

## Details abstraheren

# Wat is nodeJs en waarom gebruikt iedereen het?

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

De complexiteit van front-end applicaties is sterk toegenomen. Een explosie aan functionele eisen aan front-end applicaties heeft voor een oerwoud aan front-end libraries gezorgd.

## Node.JS

Node.js is een javascript engine die niet op de browser draait. Het bevat native OS integraties en kan zo systeemresources directer benaderen. Er zijn bijvoorbeeld webservers die op Node kunnen draaien, met hun eigen voordelen. NodeJs is niet meer weg te denken uit het front-end ecosysteem.

Node.js is ook snel.

## Dependency management

NPM is een repository die veel functionaliteit voor front-end ontwikkeling bevat. Het zijn over het algemeen kleinere packages die kleine functionaliteit bevatten.

Bower bevat naast javascript ook veel front-end functionaliteit.

Het ophalen van dependencies gebeurt via de commandline.

## Build-management

Alhoewel er geen code wordt gecompileerd, moeten dependencies wel op een gedefinieerde plaats. Handmatig kopieren 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.

## Testen en ontwikkelondersteuning

NPM en Bower repositories bevatten meerdere testservers en verschillende testformaten. Karma is een van de test-engines waarbij er een sterke code.

## Integratie met visual studio 2015

Visual studio komt met integraties voor NodeJS. Gulp taken kunnen automatisch worden gedraaid tijdens het builden van de backend.

## Conclusie

Er is gekozen om de ontwikkeling van de front-end met Node te ondersteunen. Voornamelijk omdat de hoeveelheid dependencies enorm is toegenomen sinds de laatste concentrator front-end. Het PoC maakt gebruik van beide NPM en Bower.

Node maakt het ook mogelijk de pakketten de functionaliteit beter op te splitsen door.

Er zijn meerdere gulp taken en geintegreerd in visual studio. Het gulp-script heeft de volgende capaciteiten;

* dependencies ophalen
* dependencies en code kopieren naar distributiefolder
* Distributiefolder opschonen
* Andere assets zoals plaatjes beheren
* Automatisch kopieren van asset als deze wordt gewijzigd
* Geintegreerd in VS15 solution

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

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

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

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

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

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

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

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

## AngularJS

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

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

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

### Voordelen

Angular is gratis

Onofficieel het framework van en voor google, en word gefinancierd uit de budgetten van google.

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

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

Angular is een echt framework, met veel beschikbare functionaliteit.

### Nadelen

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

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

## Sencha ExtJS

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

### Voordelen

Meest uitgebreide GRID, met grafiekjes in de vakken.

Professionele ondersteuning bij aankoop.

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

### Nadelen

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

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

Kost 9000 tot 14000 euro.

Heeft bijna geen community met libraries en toevoegingen.

Voorbeelden zijn niet bepaald verhelderend.

## ReactJS

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

### Voordelen

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

Veel functionaliteit van derde partijen.

Ook gratis.

Onofficieel het framework van en voor facebook.

### Neutraal

Onderhouden door facebook.

### Nadelen

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

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

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

## Deelconclusie

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

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

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

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

# Welke CSS framework past het beste bij concentrator?

CSS frameworks zijn verantwoordelijk voor de uitstraling van de applicatie.

Front-end frameworks bestaan uit een grote collectie herbruikbare elementen zoals knopjes, titels en etc. Deze worden door middel van klassen geïmplementeerd.

## 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. Door een framework te gebruiken is het mogenlijk de kunde van een front-end professional te implementeren in plaats van zelf iets matigs te gebruiken.

De aanbieders van browsers nemen het niet zo nauw met de standaarden. Er zijn nogal veel verschillen in welke browsers wat voor een functionaliteit bieden, en op welke manier. Sommige functionaliteit moet daarom meerdere keren worden geïmplementeerd zodat deze op alle grote browsers zou werken. Front-0end frameworks bestaan uit een collectie van veelgevraagde

Sommige browsers bevatten proprietary videocodexen voor speciale media.

## Selectie van frameworks

De frameworks zijn gekozen naar hun relatieve bekendheid en toepasbaarheid.

## Material Design

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

### Voordelen

### Speciaal voor mobiele webapplicaties

### Nadelen

Minder goed voor mobiele

Zwaardere hardware-eisen

## Bootstrap

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

### Voordelen

In actieve ontwikkeling.

Goede integraties met AngularJS

Veel funcitonaliteit uit de doos.

### Nadelen

Het uiterlijk van bootstrap is zeer herkenbaar. Er zijn mensen die daar een hekel aan hebben.

## Metro-ui

Metro-ui is ontwikkeld door een team uit wit-rusland. Het is een itnerpretatie van de windows metro huisstijl. De voorbeelden waren niet echt –up to spec.

## Zurb foundation

## Deelconclusie

Er zijn relatief weinig verschillen tussen de frameworks te bekennen. Bootstrap is wel het meest onderhouden framework met de beste documentatie en de meeste functionaliteit out-of-the-box.

# Hoe werkt een service oriented architecture?

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

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

Dit in tegenstelling tot bijvoorbeeld MVC, dat specifiek bedoelt is voor menselijke eindgebruikers, en meestal moeilijk voor andere systemen bereikbaar is.

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

## REST

REST staat voor representational state transfer en is bedacht door Roy Fielding, een software PhD. REST is onafhankelijk van het communicatieprotocol, maar in de praktijk wordt er veel gebruik gemaakt van HTTP en JSON.

REST bestaat uit de volgende constraints;

**Client-Server**

De communicatie verloopt tussen een client en een server.

**Stateless**

De server houdt geen staat bij. Elke serviceaanvraag komt met alle relevante informatie om de aanvraag te doen. Dit is een vorm van ontkoppelen en simplificatie.

**Cache**

Cachinginformatie wordt door de applicatie beheert.

**Interface / uniform contract**

De communicatie tussen client-server is expliciet vastgelegd via protocollen. In de praktijk komt dit neer op HTTP methodes, heldere locaties (URL) en gedefinieerde mediatypen.

**Layered system**

De applicatie is gelaagd opgezet. Dit maakt het mogelijk om in extreme gevallen en gehele nieuwe versie tegen de oude te ontwikkelen.

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

Als de aanvrager in staat is functies uit te voeren, dan kan dat mogelijk zijn. De server kan bijvoorbeeld javascript functies aangeven.

Veel services die restful zijn opgezet zijn niet daadwerkelijk geschreven naar de specificaties van Roy Fielding. Een van de meest ontbrekende contraints is HATEOAS. HATEOAS is moelijk gedefinieerd, maar houdt in praktische termen in dat de gehele service vanuit een enkele root te navigeren moet zijn zonder vooraf gedefinieerde informatie.

### HATEOAS

HATEOAS staat voor hypermedia access transfer extension over access service. Het voornaamste idee is dat alle functionaliteit van de server bereikbaar is vanaf een centrale root “/api/”. De RESTful service geeft dan de verschillende functies in een context weer. De server kan ook nieuwe, of niet gestandaardiseerde mediatypen aangeven. Ook functionaliteit (v.b. tree traversal) die aan de kant van de client mist kan door middel van code-on-demand aan de client worden verstrekt.

Het belangrijke van HATEOAS is dat het de service op het gebruikersniveau inzichtelijk maakt. Dit is een vorm van zelfdocumenterende code, en maakt het makkelijk op een later tijdstip zelfs de complete front-end te verwisselen.

## SOAP

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

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

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

## Deelconclusie

Eigenlijk is er geen overlap tussen de gebieden waar SOAP voor bedoelt is, en REST voor bedoelt is. SOAP is een protocol dat voornamelijk bedoeld is om ACID transacties tussen systemen uit te voeren, terwijl REST bedoelt is om data op een laagdrempelige manier aan te bieden.

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

Er is gekozen voor (ROA)REST, voornamelijk omdat dit het makkelijkst toe te passen was voor een SPA applicatie. Er zou onnodig veel tijd steken in het maken of implementeren van een javascript SOAP interface, en de meerwaarde zou miniem zijn.

# Conclusie

# 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**