ELEKTRONICA-ICT

Academiejaar 2016–2017

DYNAMISCH ANNOTEREN VAN AFBEELDINGEN IN CX SOCIAL

Gheerwijn Clicque

Promotor: Jurriaan Persyn
Mentor: Davy De Winne

Samenvatting

Woord vooraf

Inhoudsopgave

Samenvatting i

Woord vooraf ii

Lijst met gebruikte afkortingen v

1 Voorstelling van het stagebedrijf 1

1.1 Ontstaan en geschiedenis . . . 1

1.2 CX Social . . . 2

2 Probleemstelling 6

2.1 Probleemstelling vanuit de stage . . . 6

2.2 Probleemstelling bachelorproef . . . 7

3 Voorstudie 8

3.1 Vereisten . . . 8

3.2 Frontend afbeelding manipulatie . . . 8

3.2.1 Fabric.js . . . 8

3.2.2 Konva.js . . . 12

3.2.3 EaselJS . . . 15

3.2.4 VanillaJS & JQuery . . . 17

3.2.5 Vergelijking van beschreven bibliotheken . . . 20

3.2.6 Performantie van beschreven bibliotheken . . . 21

3.3 Backend afbeelding manipulatie . . . 22

3.3.1 PHP . . . 22

3.3.2 Node.js . . . 23

3.3.3 Draaien van de service . . . 24

3.4 Base64 . . . 26

3.5 Functioneel ontwerp . . . 27

4 Praktische uitwerking **28**

4.1 Frontend implementatie 28

4.1.1 Aanmaken van een thema 28

4.1.2 Annoteren van de afbeelding 30

4.1.3 Editeren van tekst 33

4.1.4 Transformaties van tekst 34

4.1.5 Verwijderen van objecten 35

4.1.6 Importeren van een canvas 36

4.1.7 Exporteren van een canvas 37

4.2 Backend implementatie 38

4.2.1 Opzetten van de Node.js service 38

4.2.2 Opbouwen van het canvas 39

4.2.3 Exporteren van het canvas 41

4.2.4 Testen van de Node.js service 41

4.2.5 Scheduling / Plannen van thema's 42

4.2.6 Doorsturen naar Facebook/Twitter - Uploaden naar sociale media 44

Bibliografie **1**

A Bijlage **4**

Lijst met gebruikte afkortingen

CEM Customer Experience Management

CX Consumer Experience

KPI Key Performance Indicator

PHP Hypertext Preprocessor

MySQL My Structured Query Language

HTML HyperText Markup Language

SVG Scalable Vector Graphics

URL Uniform Resource Locator

JPEG Joint Photographic Experts Group

PNG Portable Network Graphics

RGBA Red, Green, Blue, Alpha

HSV/HSL Hue, Saturation & Value/Hue, Saturation & Luminosity

JSON JavaScript Object Notation

DOM Document Object Model

VM Virtuele Machine

ASCII American Standard Code for Information Interchange

URI Uniform Resource Identifier

AJAX Asynchronous JavaScript and XML

ID Identity Document

CSRF Cross-site request forgery

HTTP Hypertext Transfer Protocol

NPM Node Package Manager

CORS Cross-origin resource sharing

API Application Programming Interface

1 Voorstelling van het stagebedrijf

1.1 Ontstaan en geschiedenis

Engagor werd opgericht in 2011 door Folke Lemaitre (ex-werknemer van Netlog) met de bedoeling interactie met klanten, via sociale media, te vereenvoudigen. Sinds 2015 maakt Engagor deel uit van Clarabridge. Dit Amerikaanse bedrijf ontwikkelt software om automatisch online data te verzamelen, te categoriseren en te rapporteren. Gebruikelijk is dit data verkregen van sociale media zoals Facebook en Twitter. Maar andere platformen zoals YouTube, Instagram, Vimeo en Tumbler kunnen ook beheerd worden. Kort samengevat zijn ze dus gespecialiseerd in software voor Customer Experience Management (CEM). Het gaat hier dus vooral om feedback over interacties die plaatsvinden tussen de klanten en het bedrijf. Zo kunnen bedrijven hun strategie aanpassen om hun klanten zo goed mogelijk van dienst te zijn [1]. Tegenwoordig is sociale media niet meer weg te denken uit het dagelijkse leven, noch bij particulieren noch bij bedrijven. Dit resulteert in een zeer groot klantenbestand met grote klanten zoals Telenet, NMBS, AirBnb, Ikea, Turkish Airlines, ...

Na de overname werd Engagor onderdeel van CX Social (CX staat hier voor Consumer Experience) 1.1, een van de vier producten van Clarabridge. De andere drie zijn CX Survey, CX Studio en CX Designer. Met CX Survey kunnen surveys aangemaakt en verwerkt worden. De verwerking van surveys gebeurt onder andere met behulp van tekst en sentiment analyse. CX Studio is een *dashboarding tool* waarmee gebruikers hun CEM proces kunnen stroomlijnen. Als laatste is er ook nog CX Designer dat voor data modelering gebruikt wordt [2].

Clarabridge heeft momenteel ongeveer 250 werknemers, verspreid over de hele wereld. Het hoofdkantoor bevindt zich in Reston, Virginia, Amerika. Maar er zijn kantoren in Gent, Larkspur (California), Londen, Barcelona en Singapore [3]. Op elk van deze locaties en in India zijn *support teams* aanwezig. Zo kunnen ze 24/5 en in sommige gevallen zelfs 24/7, hun klanten bijstaan.

Voor mijn stage werk ik aan CX Social. Dit is dus de *customer care tool* voor sociale media. Gebruikers kunnen via de web applicatie snel en gemakkelijk antwoorden op vragen en/of klachten op hun sociale profielen zoals Facebook, Twitter en Instagram. Dit is voor grote bedrijven natuurlijk zeer handig aangezien zij dagelijks gigantisch veel berichten (zowel privéberichten als *posts*) op hun profielen moeten verwerken. Één gemeenschappelijke plaats waaruit alle profielen kunnen onderhouden worden, is dus onontbeerlijk.

Figuur 1.1: CX Social Logo [4]

Het CXSocial team bestaat uit ongeveer 35 werknemers. Zij werken constant aan het verbeteren en onderhouden van deze *tool*. Binnen het team werken de meesten als *Full stack software engineer*. Zij werken dus zowel aan de backend van de web applicatie als aan de API en frontend. Het team wordt vervolledigd door mensen die zich bezighouden met *data science* en design. *Data scientists* werken onder andere met *machine learning* en statistieken. Het belang hiervan valt in

een sociale media *monitoring tool* niet te onderschatten. Ook het design is zeer belangrijk tijdens het ontwikkelen van een web applicatie. Telkens een nieuw *feature* toegevoegd wordt, wordt hiervan een ontwerp gemaakt om dit nieuwe stukje zo goed mogelijk in het geheel te implementeren. De designers moeten er natuurlijk ook voor zorgen dat hun ontwerpen overeenkomen met alle producten die Clarabridge aanbiedt. Ook moet alles wat ze ontwerpen zo gebruiksvriendelijk mogelijk gemaakt worden [5].

Wanneer nieuwe *features* geïmplementeerd moeten worden, wordt dit ofwel door één iemand uitgewerkt of door enkele personen uit het team. Tijdens het ontwikkelen kan feedback en hulp van collega's ingeroepen worden waardoor iedereen altijd betrokken is bij verschillende projecten. Dit zorgt er voor dat niet enkel degene die het project toegewezen krijgt, weet hoe alles in elkaar zit. Zo is iedereen op elk moment op de hoogte van veranderingen aan de *code base*.

Op Figuur 1.2 is de organisatie binnen het CX Social team te zien. Dit organigram is niet volledig accuraat want de positie van Solutions Engineer moet hier nog aan toegevoegd worden maar het schept een zeer goed beeld van de verschillende functies binnen CX Social.

Figuur 1.2: Organigram CX Social

1.2 CX Social

Elk product, aangeboden door Clarabridge, is toegespitst op de analyse van customer feedback. CX Social is de sociale media *monitoring*, management en analyse tool die gebruikt wordt door verschillende grote bedrijven om hun klanten beter te ondersteunen. Het principe van de *tool* is dat op één gemeenschappelijke plaats verschillende profielen beheerd kunnen worden. Zo kunnen onder meer berichten van klanten beantwoord worden, statistieken bekeken worden of kan de inhoud van de profielen aangepast worden.

Engagor maakt in hun webapplicatie gebruik van PHP, JavaScript, HTML en CSS. Het grootste deel is geschreven in PHP met het DooPHP *framework*. Dit *framework* is momenteel niet meer

in ontwikkeling maar werd aangepast voor gebruik binnen de Engagor webapplicatie waardoor verdere ontwikkeling van DooPHP door de ontwikkelaars niet onmiddellijk nodig is [6].

In de frontend wordt vooral gebruik gemaakt van React (ook gekend als React.js/ReactJS). Dit is een open-source JavaScript bibliotheek, ontwikkeld door Facebook, waarmee zeer gemakkelijk gebruikers interfaces aangemaakt kunnen worden. Met React kunnen componenten gecreëerd worden, waarin wordt vastgelegd wat gerenderd moet worden voor de gebruiker. React zal ervoor zorgen dat de componenten efficiënt en correct geüpdatet worden wanneer data verandert. Aan deze componenten kunnen ook parameters meegegeven worden en op basis hiervan kunnen views aan de gebruiker getoond worden [7].

CX Social is dus, zoals eerder vermeld, een sociale media *monitoring*, management en analyse tool. Zo kunnen klanten op de webapplicatie een centrale inbox raadplegen (zie Figuur 1.3). Hier worden berichten per profiel verzameld en kunnen medewerkers reageren op de vragen/opmerkingen van klanten.

Figuur 1.3: Inbox CX Social [8]

Via de 'Insights' kunnen bedrijven ook de statistieken van hun pagina's bijhouden. Zo zijn de totale mentions van een *topic* te zien alsook de *mentions* per pagina. Dit alles wordt zeer intuïtief weergegeven in de vorm van cirkeldiagrammen en tijdlijnen. Hier wordt ook het sentiment berekend. Dit geeft een aanduiding van het percentage van de mentions die een positieve, negatieve of neutrale connotatie hebben. Ook trends (veelgebruikte woorden, hashtags etc.) en foto's die binnen een topic voorkomen zijn hier te vinden (zie Figuur 1.4).

Figuur 1.4: Insights CX Social [8]

Nog meer statistieken zijn terug te vinden op de ‘Performance’ pagina. Hier is bijvoorbeeld te zien hoe lang het duurde voor actie werd ondernomen, hoeveel unieke gebruikers geholpen werden of hoeveel antwoorden gemiddeld gegeven werden op een mention (zie Figuur 1.5).

Figuur 1.5: Performance CX Social [8]

Niet enkel berichten kunnen beheerd worden via de webapplicatie maar er kan ook gepost worden op profielen zelf. Dit kan per topic en per profiel gebeuren. Posts kunnen ook gepland worden om ze op een later tijdstip op het profiel weer te geven of opgeslagen worden als *canned response*. Dit zijn standaard antwoorden die gebruikt kunnen worden tijdens het communiceren met klanten [9]. Zo kunnen klanten sneller geholpen worden met hun problemen.

2 Probleemstelling

2.1 Probleemstelling vanuit de stage

Als stageopdracht dient een nieuwe feature toegevoegd te worden aan de CSXSocial webapplicatie: een *profile manager*. Deze *profile manager* moet gebruikers in staat stellen de profielfoto en/of omslagfoto van een profiel in te stellen. Dit gebeurt aan de hand van thema's. Een thema kan aangemaakt worden voor een specifieke service, bijvoorbeeld Facebook of Twitter. Voor Facebook kan een thema een omslagfoto en een profielfoto bevatten terwijl een thema voor Twitter naast de foto's ook de link kleur, naam en bio kan bevatten (zie Figuur 2.1). Het moet ook mogelijk zijn voor de klanten om deze afbeeldingen te passen. Zo kunnen ze tekst toevoegen aan de afbeelding of hun eigen *Key Performance Indicator* (KPI)[1] er op plaatsen. Aangezien de omslagfoto's van Facebook en Twitter niet dezelfde dimensies bezitten en de foto ook niet op volledige grootte aan de gebruikers kan getoond worden, zullen de afbeeldingen ook herschaald moeten worden naar de correcte dimensies.

Figuur 2.1: Mockup van de editeerpagina [10]

Verschillende thema's moeten aan een planning toegevoegd kunnen worden. Een klant kan dan instellen dat een bepaald thema ofwel binnen de *business hours* gebruikt wordt ofwel tijdens een bepaalde periode ofwel tijdens beide. Een speciaal thema voor bijvoorbeeld Kerstmis kan dan ingesteld worden om tijdens de kerstperiode ingesteld te zijn op het profiel. Wanneer meerdere thema's op een profiel zijn ingesteld, moet dus berekend worden welk van deze thema's op een gegeven moment toegepast moet worden.

[1] Een *Key Performance Indicator* is een meetbare waarde waarmee de prestaties van bedrijven of ondernemingen uitgedrukt worden. Binnen CXSocial zijn dit bijvoorbeeld de geholpen klanten per dag of hoe lang het gemiddeld duurt voor klanten geholpen worden

De webapplicatie bezit ook *Crisis Plans*. Dit zijn instellingen die bij een probleem binnen het bedrijf actief gesteld kunnen worden. Eens actief wordt een waarschuwing getoond, rechten van gebruikers verandert of een to-do lijstje opgesteld met stappen die ondernomen moeten worden om het probleem op te lossen. Via een *Crisis Plan* moet ook een thema ingesteld kunnen worden. Een gebruiksscenario hiervoor zou bijvoorbeeld kunnen zijn: het instellen van een specifieke omslagfoto en avatar wanneer een bedrijf kampt met een technische storing.

2.2 Probleemstelling bachelorproef

Deel van de stageopdracht is het verwerken van afbeeldingen om als omslagfoto of avatar te gebruiken. Hierbij is het de bedoeling dat klanten afbeeldingen kunnen uploaden die dan als omslagfoto en/of avatar zullen dienen. Een thema, bestaande uit deze afbeeldingen en in sommige gevallen nog enkele andere variabelen zoals link kleur voor een Twitter profiel, kunnen ingesteld worden om op bepaalde tijdstippen actief te zijn. Klanten moeten in staat zijn tekst en KPI's op deze afbeeldingen te plaatsen vanuit de webapplicatie. De afbeeldingen moeten samengesteld kunnen worden door de gebruiker maar moeten nadien ook opnieuw gegenereerd kunnen worden op de server met aangepaste KPI's.

Om afbeeldingen te annoteren is redelijk wat functionaliteit nodig. Ingegeven tekst moet eender waar op een afbeelding geplaatst kunnen worden maar moet ook gestyled kunnen worden. Lettertype, kleur, lettergrootte, gewicht en uitlijning moeten allemaal ingesteld kunnen worden door gebruikers. Dit moet zowel clientside als serverside mogelijk zijn aangezien de KPI's telkens zullen veranderen waardoor nieuwe afbeeldingen gegeneerd moeten worden.

Naast het genereren van de afbeeldingen moeten deze ook geüpload worden naar Facebook of Twitter op de ingestelde tijdstippen. Er moet dus geregeld berekend worden welke afbeelding met welke KPI's op een bepaald tijdstip ingesteld moet worden als omslagfoto. De gebruiker verwacht natuurlijk dat telkens de correcte foto op het profiel ingesteld is waardoor de betrouwbaarheid ook een zeer belangrijk punt is. In functie van de bachelorproef worden bibliotheken en technologieën bestudeert en vergeleken.

3 Voorstudie

3.1 Vereisten

Alle bibliotheken en technologieën die besproken zullen worden, moeten aan enkele voorwaarden voldoen. Zo moeten afbeeldingen en tekst: verplaatst, geschaald, geroteerd en over elkaar geplaatst kunnen worden (wat dus inhoudt dat de gebruikte bibliotheek/technologie 'lagen' moet ondersteunen). Idealiter kan tekst, eens op het canvas geplaatst, nog aangepast worden. Dit vooral om interactie met tekst voor de gebruikers eenvoudiger te maken. Nadien moeten de afbeeldingen geëxporteerd en hun eigenschappen (tekst, KPI's, kleur, lettertype etc.) bewaard kunnen worden. Aangezien KPI's aanwezig zijn op de afbeelding, moeten deze later ook aangepast kunnen worden. De afbeeldingen zullen dus om de zoveel tijd opnieuw gegenereerd worden met nieuwe, aangepaste KPI's.

Veel sociale media verwachten dat omslag-en profielfoto's bepaalde dimensies bezitten. De vereiste dimensies van omslagfoto's kunnen redelijk groot uitvallen om problemen met schaling te voorkomen. Twitter verwacht bijvoorbeeld een afbeelding van 1500 op 500 pixels. Dit is natuurlijk niet gebruiksvriendelijk weer te geven in een browservenster. Daarom moet herschalen van het volledige canvas met elk van zijn objecten mogelijk zijn. Hierbij is het belangrijk dat alle verhoudingen behouden blijven om uiteindelijk dezelfde afbeelding te bekomen zoals die door de gebruiker gemaakt werd. Naast de frontend manipulatie van afbeeldingen zal dus ook manipulatie in de backend onderzocht moeten worden.

3.2 Frontend afbeelding manipulatie

Redelijk wat bibliotheken om afbeeldingen te manipuleren zijn reeds te vinden. De meeste hiervan zijn gericht op het gebruiken van filters op afbeeldingen of het maken van animaties in een canvas. Hoewel deze bibliotheken veel te bieden hebben, zijn ze minder geschikt voor deze toepassing. Het is hier namelijk de bedoeling om elementen op een canvas te positioneren, zowel tekst als de afbeeldingen zelf. Het volledige canvas en alle inhoud moet op gepaste wijze opgeslagen kunnen worden zodat het canvas op een later tijdstip opnieuw opgebouwd kan worden.

3.2.1 Fabric.js

Een bibliotheek die onmiddellijk veelbelovend lijkt te zijn is Fabric.js. Het is een JavaScript bibliotheek die HTML5 canvas manipulatie gemakkelijker maakt. Het maakt het mogelijk om object-georiënteerd te werken binnen een canvas in plaats van simpelweg vormen te 'tekenen' op het canvas. Dit maakt het veel gemakkelijker om complexe interacties met het canvas uit te voeren. Selecteren, verplaatsen, herschalen, roteren en dergelijke transformaties kunnen ondernomen worden zonder veel problemen.

Functionaliteit
In plaats van te functioneren op de context van het canvas zal Fabric gebruik maken van objecten die aan het canvas toegevoegd worden [11]. Fabric bezit zeven verschillende standaard vormen: cirkels, ellips, lijn, polygon, polylijn, rechthoek en driehoek.

Alle objecten erven over van het *root object*. Dit stelt een tweedimensionale vorm voor in het tweedimensionaal canvas en bezit een positie in het canvas (x-en y-coördinaat), een hoogte en een breedte. Aangezien elk objct dus overerft van deze klasse, ontstaat een soort van hiërarchie en kunnen methodes geschreven worden die bruikbaar zijn voor elke instantie van een object [12].

Interessanter voor deze implementatie zijn de tekst en afbeelding objecten. Naast het gewone `fabric.Text` object, wat enkel statische tekst voorstelt, is ook de `fabric.IText` beschikbaar. Met dit object kan interactieve tekst aan het canvas toegevoegd worden. Deze tekst kan dus in het canvas zelf aangepast worden. Het Text object bezit properties om: grootte, stijl (vet/schuin/normaal), gewicht en lettertype aan te passen. Het standaard HTML canvas bezit slechts twee methodes om tekst te renderen: `strokeText(text, x, y [, maxWidth])` en `fillText(text, x, y [, maxWidth])` [13]. Naast deze methodes zijn ook heel wat eigenschappen beschikbaar waaronder lettertype, uitlijning, textBaseline (e.g. alphabetic) en richting (e.g. ltr). Functionaliteit hiervan is eerder beperkt en zeker niet uitgebreid genoeg om aan de gestelde eisen (zie sectie 3.1) te kunnen voldoen.

Het tekst object dat Fabric voorziet voegt heel wat nieuwe functionaliteit toe zoals: multiline ondersteuning, tekst decoratie, uitlijning, lijnhoogte en meer.

Vooral ondersteuning van multiline tekst en uitlijning hiervan zijn zeer handige *features*. Alle objecten op het canvas kunnen *by default* geselecteerd en verplaatst worden. Natuurlijk is dit met het standaard canvas element ook mogelijk, maar code hiervoor zou steeds complexer worden naarmate meer functionaliteit toegevoegd wordt.

Het *draggable* of verplaatsbaar maken van tekst in een standaard HTML canvas, vereist heel wat code. Alle tekst, die op het canvas geplaatst wordt, moet bijgehouden worden in een lijst of array. Wanneer een *mouse-down* event plaatsvindt, moet berekend worden of de klik daadwerkelijk op een item in het canvas gebeurde. Is dit het geval dan moet de index van het geselecteerde item opgeslagen worden. Bij het triggeren van een *mouse-move* event moeten de coördinaten van dit item aangepast worden naar deze van de cursor, waarna ieder item terug op het canvas getekend moet worden.

```
1  document.getElementById('addTextButton').click(function() {
2   var text = document.getElementById('text').value;
3   items.push(text);
4  });
5
6  document.getElementById('canvas').onmousedown = function() (
7   // Kijk wanneer positie van de cursor overeen komt met een tekst object
8  )
```

Ook afbeeldingen zijn gemakkelijk te manipuleren in een Fabric canvas. Afbeeldingen kunnen geladen worden vanuit een Scalable Vector Graphics (SVG) element, een object of een URL. Met het standaard canvas element kunnen enkel afbeeldingen via URL ingeladen worden. Hoewel dit bij deze toepassing op zich niet onmiddellijk een bottleneck vormt aangezien de afbeelding ingeladen zal worden vanuit een 'file server', kan het toch handig zijn om de extra functionaliteit ter beschikking te hebben. In de toekomst kan het bijvoorbeeld nodig zijn om een SVG afbeelding te creëren en in te laden in het canvas. Dan moet, bij het gebruik van Fabric, geen ander script/bibliotheek geïnstalleerd worden om deze functionaliteit te verkrijgen. Het fabric.Image object bezit heel wat methodes die het gebruik van afbeeldingen net dat beetje eenvoudiger maken. De `scaleToWidth()` functie kan gebruikt worden om de afbeelding de volledige breedte van het

canvas te laten innemen. Manipulatie van een object kan ook afgesloten worden. Naast de mogelijkheid tot inladen van afbeeldingen, kunnen hierop ook filters toegepast worden. Hoewel dit momenteel geen vereiste is voor het project, zou dit in de toekomst toegevoegd kunnen worden. Het Image object beschikt over de meest courante filters zoals helderheid, contrast, grijstinten, sepia, ruis en inverteren.

Een belangrijke vereiste is de ondersteuning van lagen aangezien tekst bovenop de afbeelding geplaatst moet kunnen worden. Hiervoor bezit Fabric verschillende methodes. Objecten kunnen een positie hoger of lager in de stack van objecten, die op dat moment op het canvas aanwezig zijn, geplaatst worden met behulp van de `sendBackwards(object)` of `bringForward(object)` methodes. Deze methodes bezitten een optionele *intersecting* parameter `(boolean)` die het object net voor of na het volgende snijdende object in de stack zal plaatsen (zie Figuur 3.1).

Figuur 3.1: Lagen in Fabric.js

Het canvas bezit ook twee methodes om objecten volledig onderaan of bovenaan de stack te plaatsen: `sendToBack(object)` en `bringToFront(object)`. Deze methodes zijn perfect om de afbeelding in te stellen als achtergrond van het canvas. Naast deze methodes bezit het canvas ook een belangrijke property, `preserveObjectStacking`. Wanneer `true` zal de positie van alle objecten op de stack bewaard blijven. Dit zorgt er voor dat objecten niet plots bovenaan de stack komen te staan als ze geselecteerd worden. Voor uitwerking van het project is de mogelijkheid tot het gebruiken van lagen onontbeerlijk. Het standaard canvas element bezit deze functionaliteit niet, althans niet zo naadloos geïmplementeerd.

Misschien een van de grootste voordelen van Fabric is dat het zelf zorgt voor het renderen van het canvas wanneer een object gemanipuleerd wordt en dat de state van het canvas bijgehouden wordt. Kortweg betekend dit dat de `canvas.draw()` functie niet telkens moet aangeroepen worden wanneer een manipulatie plaatsvindt. Om de performantie te verhogen wordt het canvas standaard gecached. De invloed hiervan is vooral te merken bij het herschalen en verplaatsen van complexe objecten. Fabric houdt een tweede verborgen canvas bij waar de objecten reeds op aanwezig zijn. Wanneer een object gemanipuleerd wordt, wordt de gecachte versie getoond op het canvas. Dit zorgt voor een vloeiendere ervaring voor gebruikers [14]. Naast de mogelijkheid om de cache voor objecten aan of uit te schakelen, kan caching tijdens schalen van objecten uitgeschakeld worden met de `noScaleCache` eigenschap. Meer opties voor caching zijn beschikbaar maar deze zullen de performantie niet drastisch veranderen in dit project. Er zal namelijk altijd maar één afbeelding ingeladen worden en enkele tekst objecten.

Gebruiksgemak
Fabric blijkt een erg veelzijdige bibliotheek te zijn die over zeer veel features beschikt die allen kunnen bijdragen tot een gebruiksvriendelijke implementatie. Gebruik van deze bibliotheek is eenvoudig (straightforward) en intuïtief. Na het aanmaken van een canvas object, kan een toegevoegd object onmiddellijk verplaatst, geschaald en geroteerd worden. Dit is een grote hulp tijdens het ontwikkelen, vergeleken met het standaard canvas element waarvoor meerdere event listeners aangemaakt moeten worden om gelijkaardige functionaliteit te verkrijgen.

Belangrijk bij het gebruik van bibliotheken zijn de ondersteunde browsers [15]. Fabric wordt ondersteund door volgende browsers:

- Firefox 2+
- Safari 3+
- Opera 9.64+
- Chrome (alle versies)
- IE10, IE11 en Edge

Serialisatie Geannoteerde afbeeldingen moeten ook opgeslagen kunnen worden. Een Fabric canvas kan op verschillende manieren geëxporteerd worden [16]. Zo bezit Fabric twee basismethodes voor de serialisatie van een canvas: `toObject()` en `toJSON()`. De `toJSON()` functie geeft een JavaScript Object Notation (JSON) object terug die er als volgt uitziet:

```
1 {"objects": [], "background": "rgba(0,0,0,0)"}
```

De *objects* eigenschap bevat alle informatie over de objecten die aanwezig zijn op het canvas zoals tekst, afbeeldingen en vormen. *background* geeft de kleur van het canvas weer.

De `toObject()` methode geeft hetzelfde terug als `toJSON()` maar dan als een object. Het verkregen object is eigenlijk het resultaat van het aanspreken van de `toObject()` methode van elk element op het canvas. Bij het omzetten van een element naar een object, kan meegegeven worden welke eigenschappen het object moet bevatten via de optionele `propertiesToInclude` parameter. Wanneer een nieuwe klasse wordt toegevoegd, kan dus simpelweg de `toObject()` methode aangepast of uitgebreid worden. In de scope van dit project kan deze functionaliteit gebruikt worden om een extra eigenschap aan een object te geven die aangeeft of een object een KPI voorstelt of gewoon tekst.

```
1 text.toObject = (function(toObject) {
2 return function() {
3   return fabric.util.object.extend(toObject.call(this), {
4       kpiType: this.kpiType
5     });
6   };
7 })(text.toObject);
```

Naast het omzetten van een canvas naar JSON of een object, kan een canvas ook als SVG geëxporteerd worden met behulp van de `toSVG()` methode. Deze methode kan net zoals de `toObject()` methode uitgebreid worden. Het voordeel van het exporteren als SVG-bestand is dat het zonder aanpassing gebruikt kan worden in een browser of applicatie die SVG ondersteunt. Zowel JSON als het object moeten eerst in een canvas geladen worden om ze weer te geven.

Natuurlijk is het ook mogelijk om een canvas als afbeelding te exporteren. Via de `toDataUrl()` methode kan het canvas zowel als JPEG als PNG opgeslagen worden. Via verschillende parameters van deze functie kunnen onder andere het formaat (PNG of JPEG), de kwaliteit, de schalingsfactor, de breedte en hoogte ingesteld worden.

Omdat afbeeldingen ook KPI's kunnen bevatten, kunnen ze niet onmiddellijk geëxporteerd worden als afbeelding. Ook moet rekening gehouden worden met de grootte van de bestanden wanneer deze terug naar de server verzonden worden. In een tekst bestand kan nu eenmaal veel meer informatie gestoken worden dan in een afbeelding van dezelfde grootte.

Vanuit een JSON string of een SVG element kan een canvas opnieuw opgesteld worden [17]. Voor beide representaties zijn twee methodes beschikbaar. JSON kan terug ingeladen worden met behulp van de `loadFromJSON()` of `loadFromDatalessJSON()` methodes, die beiden op het canvas zelf aangeroepen worden. `loadFromJson()` zal zoals verwacht gewoonweg een JSON string in het canvas laden. `loadFromDatalessJSON()` kan gebruikt worden bij het inladen van complexe Path objecten. Een pad zal heel wat informatie bevatten, die een JSON string al snel onleesbaar en gigantisch lang kan maken. Bij het inladen van een dataless JSON string zullen de complexe Path objecten via een SVG bestand opgehaald worden. Door het reduceren van lange paden tot een eenvoudig pad naar het SVG bestand is de representatie van het canvas veel compacter geworden. SVG elementen worden ingeladen via de `loadSVGFromURL()` methode, die als parameter een link naar de SVG inhoud verwacht.

3.2.2 Konva.js

Konva is een HTML5 Canvas JavaScript *framework* dat voor zowel desktop als mobiele applicaties extra functionaliteit aan het canvas toevoegt. Vormen kunnen aan een canvas toegevoegd worden waarna deze verplaatst, herschaald en geroteerd kunnen worden. Konva startte als een GitHub fork van KineticJS, een andere HTML5 Canvas bibliotheek. Helaas wordt KineticJS momenteel niet meer onderhouden, alhoewel de laatste release wel nog gedownload kan worden. Stabiele releases zijn nog steeds beschikbaar alsook een lichtere versie van KineticJS, genaamd Concrete.js. Het bespreken van KineticJS op zich wordt hier achterwege gelaten aangezien het niet langer onderhouden wordt en vooral gericht is op het tekenen en animeren in een canvas.

Functionaliteit

Het *root* element in Konva is de stage, die gebruikt wordt als basis van een project. Deze stage kan meerdere lagen bevatten, die op hun beurt andere objecten of groepen van objecten bevatten. Elke laag in de stage bezit twee canvas *renderers*. Een *renderer* zal instaan voor het weergeven van alle objecten, dit is dus een canvas die instaat voor de visualisatie. De andere is een zogenaamde *hit graph renderer*, dit is in principe een verborgen canvas dat instaat voor event detectie. Wanneer een bestaand object op het canvas wordt aangeklikt zal dus een event getriggerd worden op het verborgen canvas waarna deze opgevangen kan worden. Naast standaard objecten zoals rechthoeken, tekst, polygonen en afbeeldingen bezit Konva ook de Shape klasse waarmee aangepaste vormen aangemaakt kunnen worden. Elk van deze objecten zijn uitbreidingen op het Konva.Node object. In de context van Konva is een node een object dat in een laag geplaatst en aangepast kan worden.

Objecten kunnen versleepbaar gemaakt worden door de *draggable* attribuut op `true` te zetten. Zoals verwacht kunnen afbeeldingen, lijnen, objecten en groepen van objecten versleept worden op een laag maar ook de stage bezit de *draggable* eigenschap. De volledige stage, inclusief alle

lagen en hun objecten kunnen zo versleept worden. In principe is dit een soort pan over het canvas. Aan een object kunnen ook enkele *event handlers* gebonden worden die drag & drop events detecteren. Beschikbaar zijn een `dragstart`, `dragmove` en `dragend` event.

Tekst kan op twee manieren weergegeven worden op een laag. Met het Konva.Text object kan een string aangemaakt worden met allerlei verschillende eigenschappen. Enkele van deze eigenschappen zijn: tekstgrootte, lettertype, uitlijning, lijnhoogte en kleur.

Het Konva.TextPath object bezit grotendeels dezelfde eigenschappen als het gewone Text object maar plaatst de tekst op een meegegeven pad. Dit pad wordt meegegeven als een eigenschap in de vorm van een SVG string. Op deze manier kan tekst bijvoorbeeld op de rand van een cirkel geplaatst worden.

Afbeeldingen kunnen op de stage getoond worden via het Konva.Image object. Deze verwacht een HTML5 Image element als parameter. Optioneel kunnen een x-en y-positie, hoogte, breedte, id, naam en een object om de afbeelding bij te snijden, meegegeven worden. Ook vanuit een URL kan een afbeelding in een Konva.Image object geladen worden. Filters zoals *blur*, helderheid, RGBA, HSV/HSL, *emboss*, *enhance*, *gradient*, inverteren, grijstinten en patronen kunnen allen op een afbeelding toegepast worden. Verder kunnen afbeeldingen, net zoals alle andere objecten versleepbaar gemaakt worden in de laag waar ze getekend zijn.

Zoals eerder vermeld maakt Konva gebruik van lagen waarin objecten geplaatst kunnen worden. Dit zorgt er niet enkel voor dat objecten over elkaar getekend kunnen worden maar het is een zeer goede manier om het tekenen van objecten performant te houden. Lagen zullen pas opnieuw getekend/gerendered worden wanneer een object in die laag aangepast wordt. Zo kan een laag enkel statische elementen bevatten, terwijl een andere laag dynamische elementen bevat. Het is namelijk zinloos en helemaal niet performant om elke laag, dus ook deze waar niets in veranderde, opnieuw te tekenen wanneer een element geüpdatet wordt. Het werken met lagen wordt in Konva mogelijk gemaakt door voor elke laag een nieuw canvas element in het Document Object Model (DOM) aan te maken. Binnen een laag kunnen objecten elkaar ook overlappen.

Één van de vereisten voor het project was de mogelijkheid tot (intuïtief) herschalen van objecten op het canvas. Aan een afbeelding kunnen in elk van de hoekpunten ankers toegevoegd worden. Deze dienen als referentiepunten om de eigenschappen van de afbeelding te manipuleren. Deze ankers krijgen events toegewezen die, eens getriggerd, het object tijdelijk niet langer versleepbaar maken en de transformatie zullen afhandelen. X-en y-posities zullen geüpdatet worden waarna de laag opnieuw getekend wordt. Mits wat programmeerwerk is het herschalen van afbeeldingen in het canvas dus haalbaar. Een ander verhaal is het transformeren van tekst. Dit kan enkel via de fontSize eigenschap van het Konva.Text object. Natuurlijk zou het mogelijk zijn hier ook ankers aan te koppelen die dan invloed hebben op de grootte van de tekst. Dit zal logischerwijze niet gunstig zijn voor de performantie van de applicatie (extra event *listeners*, berekenen van nieuwe posities en bijhorende tekst grootte, opnieuw tekenen van tekst met nieuwe eigenschappen etc.).

Om performantie van het canvas te verbeteren, bezit Konva caching. Iedere node kan gecached worden om vlugger veranderingen aan het canvas af te kunnen handelen. Gecachete nodes worden omgezet in afbeeldingen die in een buffer canvas opgeslagen worden. Bij transformaties in het canvas kunnen deze buffer afbeeldingen gebruikt worden. Vooral bij complexe elementen zoals tekst of vormen met schaduwen kan caching een groot verschil maken.

Wat Konva mist qua functionaliteit is interactieve tekst. Hiermee wordt tekst bedoeld die rechtstreeks in het canvas aangepast kan worden. Dit heeft vooral invloed voor het gebruiksgemak van de applicatie. Het is namelijk intuïtiever om tekst in het canvas aan te passen door het te

selecteren dan het via een input kader te doen. Gebruikers zullen sneller de tekst zelf proberen selecteren om deze aan te passen dan het object te selecteren en in een input veld de aanpassingen te maken om uiteindelijk nog de veranderingen door te moeten voeren door middel van een submit knop.

Gebruiksgemak

Konva is een veelzijdige bibliotheek die manipulatie van het tweedimensionale HTML canvas mogelijk maakt met JavaScript. In de standaard bibliotheek is genoeg functionaliteit beschikbaar om complexe canvas interacties mogelijk te maken. Naast verschillende standaard objecten zoals rechthoeken, cirkels, lijnen, afbeeldingen en tekst kunnen nieuwe vormen eenvoudig aangemaakt worden (en dit zowel op desktop als op mobiel.)

Serialisatie

Om de stage op te kunnen slaan, kan deze geëxporteerd worden als JSON met behulp van de `toJSON()` functie. Een JSON object ziet er als volgt uit:

```
1  {
2    "attrs": {
3      "width": 578,
4      "height": 200
5    },
6    "className": "Stage",
7    "children": [
8     {
9       "attrs": {},
10      "className": "Layer",
11      "children": [{
12        "attrs": {
13        "x": 0,
14        "y": 50,
15        "width": 500,
16        "height": 100,
17        "fill": "black",
18        "rotation": 0.7,
19        "id": "rectangle"
20         },
21         "className": "Rect"
22       },
23       {
24       "attrs": {
25       "width": "auto",
26       "height": "auto",
27       "text": "Hello World!",
28       "fontFamily": "Arial",
29       "fontSize": 70,
30       "x": 80,
31       "y": 60,
32       "fill": "white",
33       "stroke": "white",
34       "strokeWidth": 2,
35         },
36         "className": "Text"
37     }]
38   }]
```

```
39 }
```

Elk object bezit dus attributen (attrs), de naam van de klasse (className) en zijn child objecten (children). De attributen bevatten de eigenschappen van het object zoals hoogte, breedte, id, x-en y-coördinaten, stijlinformatie en rotatiehoek. Alle objecten die naar dit ene object refereren en dus zijn children zijn, zitten in het children attribuut. Wordt bijvoorbeeld gekeken naar de stage, het eigenlijke canvas die de andere objecten bevat, dan is het duidelijk dat alle andere objecten tot de children van dit object behoren [18]. In bovenstaand voorbeeld bevat de stage één laag die op zijn beurt een tekst object en een lichtjes geroteerde rechthoek bevat:

Figuur 3.2: Konva.js - importeren van een canvas

Om vanuit een JSON object een stage te creëneren, moet eerst en vooral een nieuwe node aangemaakt worden. Met behulp van de `Konva.Node.create()` methode kan vanuit het JSON object een stage aangemaakt worden.

```
1 <div id="container"></div>
2 <script>
3   var stage = Konva.Node.create("JSON", "container");
4 </script>
```

Bevat een stage afbeeldingen of events, dan moeten deze opnieuw aangemaakt worden en met behulp van de `get()` methode en selectors ingesteld worden. Dit kan als volgt uitgevoerd worden [19]:

```
1 var image = new Image();
2   image.onload = function() {
3       stage.get('#image')[0].image(image);
4       stage.draw();
5   };
6   image.src = '/image.jpg';
```

Om een stage te exporteren als afbeelding kan gebruik gemaakt worden van de `toDataURL()` methode. Dit zal een URL teruggeven die voorafgegaan wordt door de data van de afbeelding. Standaard is dit een PNG afbeelding die base64 gecodeerd wordt. Via parameters kunnen het type van de afbeelding (JPEG/PNG), x-en y-coördinaten, hoogte, breedte en kwaliteit (enkel van toepassing bij JPEG) ingesteld worden. Belangrijk hierbij is dat afbeeldingen zich op een webserver moeten bevinden op hetzelfde domein als waar de uitgevoerde code staat [20].

3.2.3 EaselJS

EaselJS is één van de bibliotheken behorende tot de CreateJS suite. CreateJS is een verzameling van *open source* bibliotheken en tools om interactieve web applicaties te maken. Zo bezit CreateJS ondermeer bibliotheken voor het creëren van animaties (TweenJS), om te werken met audio

(SoundJS) en om het laden van data in een applicatie te regelen (PreloadJS). EaselJS maakt interactie met het HTML5 canvas element in JavaScript zeer eenvoudig. Het is een zeer krachtige bibliotheek die ondersteuning biedt voor zowel statische afbeeldingen als animaties. Ook bezit het enkele experimentele features om onder andere HTML elementen te beheren alsof ze deel uit maken van het canvas zelf. Dit kan handig zijn om een elementen bovenop het canvas te plaatsen, zonder dat deze tot het canvas zelf behoren, zoals een toolbar die een tekst editor bevat.

Functionaliteit

De basis van de Easel workflow is het Stage object. Dit object wordt opgebouwd op basis van een HTML canvas element en zal alle objecten bevatten die op het canvas getoond moeten worden. De stage bevat een `tick()` methode die, wanneer opgeroepen, alle objecten op het canvas zal renderen. Een Ticker klasse zorgt voor een gecentraliseerde tick, waarop *listeners* kunnen subscriben. In de *listeners* worden objecten in de stage verandert en wordt de stage geüpdatet via de `update()` methode. Omdat Easel zelf geen specifieke functionaliteit bezit om objecten te kunnen verslepen, moet dit gebeuren aan de hand van events. Zo zijn er de `mousedown`, `mouseup`, `pressmove` en `pressup` events die het mogelijk maken om objecten te verslepen over het canvas. Na het aanmaken van een event *listeners* op het `pressmove` event, kan een stuk code geschreven worden die de positie van het object aanpast naar de huidige positie van de cursor. Door telkens het canvas up te daten (met de `update()` methode) wanneer de positie verandert, kan de *drag & drop* functionaliteit bekomen worden [21].

Met de Text klasse kan tekst op het canvas geplaatst worden. Aan de constructor kunnen verschillende parameters meegegeven worden waaronder tekstkleur, lettertype, lijnhoogte, lijnbreedte, omlijning, rotatie en type uitlijning. Daarnaast kunnen ook de positie, schaal, zichtbaarheid en dergelijke meegegeven worden. Met behulp van eerder besproken event *listeners*, kunnen tekst objecten versleepbaar gemaakt worden. Wat deze klasse niet bezit is mogelijkheid tot het aanpassen van tekst in het canvas. In principe kan dit geïmplementeerd worden met behulp van de DOMElement klasse. Dit is de experimentele klasse waarmee HTML elementen aangepast kunnen worden alsof ze zich in het canvas zelf bevinden. Een tekst input kan dan over het canvas geplaatst worden en met behulp van extra event *listeners* en styling (om het input veld onzichtbaar te maken), kan deze input aan een Text object gebonden worden.

Afbeeldingen kunnen worden weergegeven op het canvas met de Bitmap klasse. Deze klasse kan naast afbeeldingen ook een canvas of een video weergeven. Als parameter kan een HTML *image*, canvas of video element meegegeven worden of simpelweg de Uniform Resource Identifier (URI) naar de afbeelding [22]. Wanneer een afbeelding/video/canvas geladen wordt via de URI, dan zal een nieuw Image object aangemaakt worden en zal dit meegegeven worden aan de *image* eigenschap van het Bitmap Object. Via de `setTransform()` methode kunnen x-en y-positie, schaal, rotatie en helling van het object aangepast worden [23].

Easel ondersteunt het direct schalen van objecten in het canvas niet. Om dit toch te verwezenlijken, moet een input gedefinieerd worden die de schaal van het object in kwestie zal aanpassen. Deze schaal kan aangepast worden met behulp van de `scaleX` en `scaleY` eigenschappen van het object. Na iedere verandering zal de stage geüpdatet moeten worden. Natuurlijk is dit ook mogelijk door gebruik te maken van de JQuery `resizable()` methode. Hier moet dan opnieuw gebruik gemaakt worden van de experimentele DOMElement klasse om een element te beheren dat zich buiten het Easel canvas bevindt.

Gebruiksgemak

Complexe objecten zoals afbeeldingen en tekst vergen veel om te renderen. Daarom is

het best om deze objecten te cachen. Wanneer de `cache()` methode aangeroepen wordt op een object wordt het object in een nieuw canvas geplaatst. Dit nieuwe canvas zal dan telkens gebruikt worden wanneer een aanpassing in het originele canvas gebeurt. Het spreekt voor zich dat gecachete objecten enkel gebruikt kunnen worden bij basistransformaties zoals verplaatsingen en rotaties. Bij complexere transformaties zoals herschalen of het toepassen van filters dient de cache opnieuw geüpdatet te worden. Aan de `cache()` methode kunnen x-en y-coördinaten, hoogte, breedte en schaal meegegeven worden. Deze parameters definiëren het gebied dat gerendered en gecachet zal worden.

EaselJS wordt ondersteund door elke browser die het HTML Canvas element ondersteunen. De meest recente browsers ondersteunen het canvas element en zijn dus perfect compatibel met EaselJS. In Internet Explorer 11 kan het canvas niet geëxporteerd worden met de `toDataURL()` methode wanneer afbeeldingen aanwezig zijn op het canvas. Aangezien het canvas niet opgeslagen dient te worden als afbeelding maar als JSON vormt dit niet onmiddellijk een probleem.

Serialisatie
Easel bezit geen functionaliteit om een stage te exporteren. Voor dit project in het bijzonder is dit natuurlijk een zeer groot nadeel. Als een stage niet geëxporteerd of geserialiseerd kan worden, kan deze niet opgeslagen worden. Natuurlijk is het nog steeds mogelijk om een stage te exporteren. Alle *children* van de stage kunnen overlopen worden om zo elk object om te zetten naar een JSON object. Om terug een stage op te stellen, moet vanuit het JSON object terug een stage aangemaakt worden waarna alle child objecten eerst gecreëerd en daarna toegevoegd aan de stage moeten worden. Een stage kan wel geëxporteerd worden als SVG afbeelding met behulp van de SVGExporter tool. Deze tool is nog experimenteel dus is het zeker niet aan te raden om dit in productie te gebruiken [24].

3.2.4 VanillaJS & JQuery

Een van de bekendste JavaScript bibliotheken is JQuery. Het is een kleine, snelle bibliotheek die het programmeren veel eenvoudiger maakt. Het ondersteund ondermeer *event handling*, animaties en *Asynchronous JavaScript and XML* (AJAX). Ook bezit JQuery alle functionaliteit die beschikbaar is bij het gebruik van *VanillaJS*. Voor JQuery zijn enkele plugins, zoals jCanvas, beschikbaar die manipulatie van het HTML5 canvas eenvoudiger maken. Aangezien jCanvas ongeveer dezelfde functionaliteit bezit als de eerder besproken bibliotheken, wordt de 'standaard' JQuery onder de loep genomen. Zo wordt duidelijk wat mogelijk is met enkel de JQuery bibliotheek.

JQuery UI is een uitbreiding op de JQuery bibliotheek en voorziet interface interacties, effecten, widgets en thema's. JQuery UI stelt bijvoorbeeld een kalender, autocomplete functie, tooltips en dialogen beschikbaar alsook interacties zoals `draggable`, `droppable` en `resizable`.

Functionaliteit
Veel van de opgelegde vereisten worden niet standaard door JQuery aangeboden. Daarom moet gezocht worden naar alternatieve manieren om bepaalde zaken op te lossen. Een van de vereisten is dat elementen op het canvas verplaatst kunnen worden. JQuery bezit `draggable()` en `droppable()` methodes waarmee eender welk DOM element verplaatsbaar wordt. Helaas is deze functionaliteit niet beschikbaar in een HTML5 canvas element, wat niet nuttig is voor de huidige *use case*. Merk op dat hier gebruik gemaakt wordt van het canvas element om andere functionaliteit te behouden. Zo is het perfect mogelijk om alle manipulaties (o.a. tekst over een

afbeelding plaatsen) in een element uit te voeren met behulp van de `draggable()` en `droppable()` methodes. Dit maakt het manipuleren een stuk eenvoudiger maar zal uiteindelijk problemen geven bij het exporteren van de data. Elk DOM element moet dan met alle eigenschappen zoals grootte, kleur, inhoud en positie, omgezet worden naar een object dat eenvoudig weggeschreven kan worden.

Om tekst in een canvas verplaatsbaar te maken moeten alle eigenschappen van het tekst bijgehouden worden in een object. Dit object bevat dan de tekst zelf, x-en y-positie, hoogte en breedte. Elk van deze eigenschappen is belangrijk aangezien er *hit-testing* zal plaatsvinden. Wanneer een *mouse-down* event plaatsvindt, worden alle objecten in het canvas overlopen en wordt berekend of het event al dan niet op een bestaand element plaatsvindt. Is dit het geval, dan wordt tijdens het *mouse-down* event de positie van het geselecteerde object aangepast naar de positie van de cursor van de gebruiker. Door dan telkens opnieuw het canvas opnieuw te renderen met de nieuwe posities, wordt de tekst versleepbaar.

Hetzelfde kan toegepast worden bij het manipuleren van een afbeelding. Na het aanmaken van een afbeelding in het canvas, worden ankers geplaatst op de hoekpunten. Via deze ankers kan een gebruiker dan gemakkelijk de afbeelding herschalen. Ook hier vindt *hit-testing* plaats. Klikt de gebruiker één van de hoekpunten aan en wordt dit verplaatst, dan worden de hoogte en breedte van de afbeelding aangepast. Het verplaatsen van de afbeelding verloopt op gelijkaardige manier. Wordt een *mouse-down* event getriggered op de positie van de afbeelding, dan wordt de positie van de afbeelding in het canvas aangepast.

Het standaard canvas element ondersteund geen lagen, wat een van de vereisten is voor dit project. Om dit probleem op te lossen wordt gebruik gemaakt van meerdere canvas elementen die bovenop elkaar liggen. Zo kan tekst telkens toegevoegd worden op een nieuw canvas, waardoor het boven reeds bestaande element, zoals de achtergrondafbeelding, wordt getekend. Een ander voordeel van het gebruik van meerdere canvassen is het feit dat elk tekst objec apart opgemaakt kan worden. Om tekst op te maken binnen een canvas wordt gebruik gemaakt van de `font` eigenschap van de context. De stijl, het gewicht, familie en grootte kunnen aangepast worden. Omdat de `font` eigenschap op de context wordt aangepast, bezit elk tekst object binnen het canvas dezelfde eigenschappen. Dit is natuurlijk erg limiterend voor de gebruikers. Het gebruik van meerdere canvassen lost dit probleem op aangezien iedere context andere instellingen kan bevatten.

Gebruiksgemak

Omdat hier geen gebruik gemaakt wordt van bibliotheken die gericht zijn op het manipuleren van het canvas, kan geen gebruik gemaakt worden van methodes en/of objecten die interacties vereenvoudigen. Zo zal het canvas telkens expliciet opnieuw getekend moeten worden wanneer veranderingen plaatsvinden. Eerder besproken bibliotheken handelen dit volledige intern af. Alle functionaliteit moet door de programmeur zelf geïmplementeerd worden.

Een voordeel van het gebruiken van *VanillaJS* is dat de code niet afhankelijk is van externe bibliotheken. Ook wordt het canvas element door alle moderne browsers standaard ondersteund. Enkel Internet Explorer 11 bezit een bug waardoor de `canvas.toDataUrl()` methode niet werkt wanneer afbeeldingen in het canvas worden ingeladen.

Serialisatie

Exporteren van een canvas is niet zo vanzelfsprekend. Uiteraard bezit het canvas enkele methodes, waaronder de `toDataUrl()` methode, die instaan voor serialisatie. Deze methodes kunnen echter het canvas enkel als afbeelding exporteren. Dit is natuurlijk niet erg handig

aangezien de KPI's op de afbeelding variabel zijn. Om een canvas in de huidige staat te exporteren moet over elk object in het canvas gegaan worden en alle eigenschappen weggeschreven worden. Op deze manier kunnen dan de objecten er uit gehaald worden die een andere waarde moeten krijgen en kan uiteindelijk de afbeelding gegenereerd worden.

3.2.5 Vergelijking van beschreven bibliotheken

	+	-
Fabric.js	Object georiënteerd Ruim assortiment aan objecten (inclusief interactieve tekst) Ondersteuning van lagen Standaard manipulatie (schalen, verplaatsen...) van elk object Node.js ondersteuning Caching	(geen ondersteuning voor IE9)
Konva.js	Object georiënteerd Voldoende ruim assortiment aan objecten Mogelijkheid tot splitsen van lagen met statische en dynamische objecten Aparte canvas voor hit-detectie in elke laag Node.js ondersteuning Caching	Geen interactieve tekst
Easel.js	Object georiënteerd Voldoende groot assortiment aan objecten Aanpassen van HTML DOM elementen (experimenteel) Caching	Geen Node.js ondersteuning Geen interactieve tekst Geen functionaliteit om objecten te schalen en verslepen Geen methodes om canvas te exporteren
VanillaJS	Geen dependencies Lightweight Volledige controle over functionaliteit	Niet object georiënteerd Geen Node.js ondersteuning Geen functionaliteit om objecten te schalen en verslepen Geen methodes om canvas te exporteren als JSON Geen caching

Tabel 3.1: Vergelijking van Fabric.js, Konva.js, Easel.js en VanillaJS

Bij het vergelijken van de bibliotheken is het duidelijk dat elke bibliotheek ongeveer dezelfde functionaliteiten aanbiedt. Wordt gekeken naar de vereisten van het project blijkt Fabric.js toch als winnaar uit de bus te komen. De ondersteuning van interactieve tekst is een grote meerwaarde van deze bibliotheek.

3.2.6 Performantie van beschreven bibliotheken

Aangezien het project enkel basis manipulatie van objecten in het canvas vereist, speelt performantie niet zo'n grote rol. Op het canvas zullen op een gegeven tijdstip enkel een afbeelding en wat tekst objecten aanwezig zijn. Dit zal in geen enkel geval invloed hebben op de performantie.

3.3 Backend afbeelding manipulatie

Eens de gebruiker zijn/haar afbeelding heeft samengesteld, moet deze opgeslagen worden in de database. Voor de foto geüpload wordt naar Facebook of Twitter, moeten de KPI's namelijk aangepast worden naar de huidige waarden. Dit gebeurt in een microservice die bijvoorbeeld om de 5 minuten de nodige KPI's berekend en met deze waarden een foto genereerd. Zoals eerder besproken zal het aangemaakte canvas geserialiseerd worden zodat het later opnieuw aangemaakt kan worden met de correcte waarden. Wat hier vooral een probleem vormt, is de nauwkeurigheid van de omzetting. De afbeelding die gegenereerd wordt moet namelijk identiek zijn aan de afbeelding die de gebruiker samenstelt.

Het is vanzelfsprekend dat de backend in staat is om het object, dat gegenereerd werd in de frontend, te interpreteren en de nodige bewerkingen (zoals het veranderen van KPI's) kan uitvoeren. Er kan gebruik gemaakt worden van verschillende programmeertalen en omgevingen om de afbeeldingen te genereren.

3.3.1 PHP

Het omzetten van een JSON object naar een afbeelding vormt een hele uitdaging in PHP aangezien het niet meer op een canvas geplaatst kan worden. In PHP kunnen afbeeldingen aangemaakt worden met behulp van de GD bibliotheek[1] [25]. Zo wordt een afbeelding ingeladen via de `imagecreatefromjpeg()` methode, Waarna deze geschaald kan worden naar de correcte dimensies met de `imagecopyresampled()` methode. Deze zal een kopie van de afbeelding maken en hierbij de pixel waarden interpoleren om de kwaliteit van de afbeelding zo goed mogelijkt te behouden. Eens de afbeelding de geschaald is, kan de tekst er op geplaatst worden met behulp van de `imagefttext()` methode. Aan deze methode kunnen de afbeelding, lettergrootte, rotatiehoek, coördinaten, kleur, lettertype en tekst meegegeven worden.

Volgend script laadt een afbeelding in, herschaald deze naar de correcte dimensies voor Twitter en plaatst de tekst "Hello World" op de afbeelding.

```php
<?php
  $fileName = "image.jpeg";
  $image = imagecreatefromjpeg($fileName);
  $destination = imagecreatetruecolor(1500, 500);
  list($width, $height) = getimagesize($fileName);
  imagecopyresampled($destination, $image, 0, 0, 0, 0, 1500, 500, $width,
      $height);
  $fontSize = 60;
  $angle = 0;
  $x = 600;
  $y = 250;
  $color = imagecolorallocate($image, 255, 255, 255); // white
  $fontFile = "helvetica.ttf";
  $text = "Hello World";
  imagefttext($destination, $fontSize, $angle, $x, $y, $color, $fontFile,
      $text);
```

[1] De GD bibliotheek is een grafische bibliotheek die heel wat functies bezit om afbeeldingen aan te maken en/of te manipuleren.

```
15   imagejpeg($destination, "test.jpeg", 99);
16 ?>
```

Figuur 3.3: PHP afbeelding []

Één van de nadelen bij het gebruik van PHP is dat het object, dat geëxporteerd wordt door de frontend, niet rechtstreeks omgezet kan worden. Ieder object in het geëxporteerde canvas moet overlopen worden en met de juiste eigenschappen op de afbeelding geplaatst worden. Zo moet voor elk tekst object de lettergrootte, kleur, lettertype, gewicht en tekst uit de JSON representatie van het canvas gehaald worden. In het geval van een KPI moet dan ook de tekst vervangen worden. Hoewel dit perfect mogelijk is, is het niet erg efficient en zelfs wat omslachtig. Ook kan het voorkomen dat bepaalde eigenschappen niet ondersteund of anders geïmplementeerd zijn.

3.3.2 Node.js

Veel gemakkelijker is het gebruiken van Node.js. Hiermee kan in de backend gebruik gemaakt worden van JavaScript code. Dit betekend dat dezelfde bibliotheken, die gebruikt worden om in de frontend de afbeeldingen te editeren, gebruikt kunnen worden in de backend. Het merendeel van de eerder besproken JavaScript bibliotheken bezit ondersteuning voor Node.js.

Omdat Fabric, Konva en Easel allen gebaseerd zijn op het HTML canvas element, moet ook een canvas implementatie aanwezig zijn voor Node.js. De `npm-canvas` *package* is gebaseerd op Cairo, een 2D *graphics* bibliotheek geschreven in C. Konva, Fabric en Easel werken in een Node omgeving hetzelfde als in een frontend omgeving. Een canvas wordt aangemaakt waaraan afbeeldingen, tekst, vormen en animaties toegevoegd kunnen worden. Zowel met Fabric als met Konva is het mogelijk om geserialiseerde data, in JSON formaat, terug om te zetten in een canvas. Bij Easel moeten alle objecten in het canvas opnieuw opgebouwd worden aangezien het met deze bibliotheek niet mogelijk is om JSON data om te zetten naar een canvas.

Het omzetten van een JSON data naar een afbeelding gebeurt als volgt: **Fabric.js**

```
1  var fabric = require('fabric').fabric;
2  var fs = require('fs');
3  var json = {...};
4
5  var canvas = fabric.createCanvasForNode(1500, 500);
6  canvas.loadFromJSON(json, function() {
7    canvas.renderAll();
```

```
8   });
9
10  var stream = canvas.createPNGStream();
11
12  var out = fs.createWriteStream(__dirname + '/test-' + service + '.png');
```

Zoals te zien in de voorbeeldcode bezit Fabric methodes specifiek voor gebruik met Node.js. De `createCanvasForNode()` methode maakt een canvas aan zonder dat hiervoor een HTML canvas element nodig is. De `createPNGStream()` functie geeft een Node Stream object terug, waarmee het canvas opgeslagen kan worden als een PNG bestand. Het Stream object kan ook de data teruggeven in een base64 formaat. Aangezien dit eenvoudigweg een tekstuele representatie van binaire data is, is dit uitermate geschikt voor dit project. Wordt een Node.js service opgezet, die als input JSON data verwacht. Dan kan de *response* de base64 geëncodeerde afbeelding bevatten. Zo wordt het mogelijk om de afbeelding bijvoorbeeld naar Facebook of Twitter te uploaden.

Konva.js

```
1  var konva = require('konva');
2  var json = {...};
3  var stage = Konva.Node.create(json);
4
5  stage.toDataUrl({
6    callback: function(data){
7      // Convert image data to png or base64
8    }
9  })
```

Konva bezit geen extra methodes voor gebruik met Node.js. Maar de container parameter kan weggelaten worden bij het creëren van de stage aangezien er geen DOM elementen aanwezig zijn. Met behulp van de `Konva.Node.create()` methode kan JSON data omgezet worden naar een stage/canvas. Daarna kan de stage geëxproteerd worden naar een PNG bestand of in base64 formaat met behulp van `toDataUrl()`.

3.3.3 Draaien van de service

Zoals eerder vermeld zal een microservice gebruikt worden om de afbeeldingen te genereren. Logischerwijze moet deze service op een server draaien zodat deze vanuit de backend, die hoofdzakelijk in PHP geschreven is, aanspreekbaar is. Om het deployen van de service gemakkelijker te maken, wordt gebruik gemaakt van Docker.

Docker is een *open-source* project gebaseerd op containers. Dit betekend dat software in geïsoleerde omgevingen zal draaien. In deze containers kunnen ontwikkelaars een applicatie opstellen met alle nodige bibliotheken en *dependencies*. Containers kunnen dan op eender welk systeem ingezet worden en zullen overal hetzelfde functioneren. In tegenstelling tot Virtuele Machines (VM) bevat een container geen volledig besturingssysteem maar enkel wat nodig is om de software te doen werken. De containers delen namelijk dezelfde kernel en zullen enkel de onderdelen installeren die nog niet aanwezig zijn. Deze containers kunnen bijgevolg zeer eenvoudig gedeeld worden tussen ontwikkelaars onderling [26].

Zoals te zien op Figuur 3.4 is het verschil tussen VM's en Docker duidelijk. Traditionele VM's vereisen elk een volleidg besturingssysteem wat er voor zorgt dat ze volledig afzonderlijk van

hun *host* functioneren. Een *hypervisor* staat in voor het beheer van de VM's. Wordt Docker gebruikt, dan zorgt de lightgewicht Docker Engine voor het beheer van alle containers [27]. Het is duidelijk dat dit systeem het zeer eenvoudig maakt op applicaties op te zetten en te testen. Een container kan opgezet worden in enkele seconden terwijl dit bij een VM meerdere minuten duurt. Ook kunnen meer containers op één server geplaatst worden aangezien niet telkens een besturingssysteem geïnstalleerd moet worden.

Figuur 3.4: Docker versus VM [28]

3.4 Base64

Dit project steunt op het bewerken van afbeeldingen. Zowel het genereren van de afbeelding in de backend als deze doorsturen naar sociale media vereist het 'transporteren' van deze afbeeldingen. Natuurlijk zou het mogelijk zijn om elke afbeelding op te slaan op de *fileserver*, maar al snel zou duidelijk zijn dat dit niet de beste oplossing is. Het is namelijk de bedoeling dat een omslagfoto gegenereerd wordt met aangepaste KPI's om de zoveel tijd. Dit betekend dat iedere dag tientallen foto's aangemaakt zullen worden voor één profiel. Hoewel afbeeldingen gecomprimeerd kunnen worden tot enkele kilobytes, is dit toch niet de beste oplossing. Ideaal is een manier om een afbeelding door te sturen als tekst. En dit is precies wat base64 encodering mogelijk maakt.

Base64 is een encoderingsproces die binaire data omvormt naar ASCII karakters. In principe worden simpelweg drie octetten omgezet in vier geëncodeerde karakters. Tijdens het encoderen wordt de binaire data gesplitst in groepjes drie bytes. Deze groepen bestaan dan uit vier groepen van zes bits (2^6 = 64 karakters). Deze zes bits worden met behulp van de base64 encoderingstabel (zie Figuur 3.5) omgezet in ASCII karakters **??**. Wanneer de laatste groep niet genoeg bytes bevat om een 6-bit karakter op te stellen, kan *padding* toegevoegd worden. In de binaire data vertaald dit zich in het toevoegen van nullen op het eind tot een 6-bit karakter gevormd wordt. In de ASCII tekst wordt deze padding voorgesteld door een '='. Een voorbeeld van het volledige proces is hier te zien:

Val	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Char	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Val	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
Char	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Val	52	53	54	55	56	57	58	59	60	61	62	63
Char	0	1	2	3	4	5	6	7	8	9	+	/

Figuur 3.5: Base64 encoding tabel [29]

Wordt de volledige stream van binaire data op deze manier geëncodeerd, wordt uiteindelijk één lange *string* bekomen die verstuurd kan worden in een *response* of *request*.

3.5 Functioneel ontwerp

// Insert mockups?

4 Praktische uitwerking

Het annoteren van afbeeldingen is één van de onderdelen binnen de *Profile Manager* van CX Social. De *Profile Manager* is een nieuwe feature die gebruikers moet helpen bij het beheren van hun sociale media profielen. Een thema kan aangemaakt worden voor Facebook of Twitter dat een omslag-en/of profielfoto bevat. Eens een omslagfoto geüpload is, is het mogelijk om annotaties toe te voegen. Dit kan simpelweg tekst zijn maar ook KPI's kunnen op de afbeelding geplaatst worden. Gebruikers kunnen voor geconnecteerde profielen instellen welk thema op welk tijdstip actief moet zijn. De aangemaakte afbeeldingen worden dan op de ingestelde tijdstippen geüpload naar het juiste profiel.

4.1 Frontend implementatie

Een groot deel van de frontend code van CX Social is geschreven met React. Met React kunnen makkelijk componenten aangemaakt worden om zo complexe interfaces aan te maken. Om de *Profile Manager* zo modulair mogelijk te maken worden de verschillende *features* opgesplitst in React componenten.

4.1.1 Aanmaken van een thema

Om een thema aan te maken zijn een naam, een *service* en twee velden nodig om afbeeldingen te uploaden (zie Figuur 4.1). Aangezien dezelfde functionaliteit vereist is wanneer een thema aangepast wordt, wordt gebruik gemaakt van een React component. Deze `ThemeComponent` stelt een volledig thema voor en zal alle veranderingen hieraan afhandelen. Dit betekend dus zowel het ingeven van de naam en service als het bijhouden van de geannoteerde afbeeldingen. Uiteindelijk is het deze component die instaat voor alle interacties met een thema. Bij het aanmaken van de `ThemeComponent` dienen enkele eigenschappen meegegeven te worden. Eerst en vooral wordt het, uit de databank opgehaalde, thema meegegeven indien dit beschikbaar is.

Figuur 4.1: ThemeComponent in de editeer pagina

Het account *Identity Document* (ID), *Cross-site requrest forgery* (CSRF) token, beschikbare services, en de annotatiepermissie zijn verplichte eigenschappen van de component. Deze eigenschappen zullen nooit veranderen tijdens de levensduur van het component, het zijn dus constanten. Het CSRF token wordt aan het formulier toegevoegd om ongeauthoriseerde acties in applicaties te voorkomen. Met het token wordt ervoor gezorgd dat geen ongeldige *requests* gemaakt kunnen worden. Niet elke gebruiker heeft dezelfde rechten in CX Social. Binnen een bepaald plan zijn verschillende gebruikersrollen beschikbaar. Zo zijn er gewone gebruikers, *admins*, managers, *editors*, *contributors* en *analytics users*, elk met hun eigen rechten. Permissies, die aan de gebruiker werden toegekend, worden meegegeven aan de Theme component om de annoteer *feature* al dan niet beschikbaar te stellen. Ieder inputveld bevat een `onChange` event waarmee veranderingen afgehandeld kunnen worden. Zoals te zien in codevoorbeeld 4.1 wordt de state van de component geüpdatet wanneer een verandering aan de inputvelden plaatsvindt. De state van de component kan, in tegenstelling tot de eigenschappen (props), wel veranderen binnen het component. De state kan aangepast worden met behulp van de `setState()` methode.

```
1 <select className={this.state.saving && this.state.service === '' ? 'error
    fieldset-input' : 'fieldset-input'} id="service" name="service" ref="
    service" value={this.getService()}
2   onChange={function (event) {
3     this.setState({service: event.target.value});
4   }.bind(this)}>
5   {this.props.services.map(function (service) {
6     return <option key={service} value={service}>{this.capitalize(service)}</
        option>
7   }.bind(this))}
8 </select>
```

Listing 4.1: Theme component - Dropdown

Naast het ingeven van een naam en *service* voor het thema, kunnen ook een omslag-en profielfoto toegevoegd worden. Het uploaden van de afbeeldingen wordt afgehandeld door een reeds bestaande component. Deze zal een, door de gebruiker geselecteerde, foto uploaden naar de *fileserver* en op de pagina weergeven. Eens een afbeelding aanwezig is, is het mogelijk om annotaties toe te voegen. Om een afbeelding te annoteren wordt een pop-upvenster geopend. Aangezien deze pop-up een volledig nieuwe *template* is, dienen enkele eigenschappen van het thema meegegeven te worden. Als parameters in de URL worden zowel de *service* als de bestandsnaam van de afbeelding op de fileserver meegegeven. Beide parameters zijn nodig om het canvas op te stellen. De *service* bepaald immers de dimensies van het canvas aangezien deze verschillen voor zowel Facebook als Twitter. Via de bestandsnaam kan de correcte afbeelding ingeladen worden in het pop-upvenster.

Zoals reeds besproken in sectie 3.2.5, lijkt Fabric.js de beste keuze te zijn om dit project uit te werken. Aangezien niet alle functionaliteit van deze bibliotheek van toepassing zijn op dit project, wordt een *custom build* aangemaakt. Deze bevat de tekst, interactieve tekst, animatie, serialisatie, interactie en node *features*. Bij het openen van het pop-upvenster wordt het `ThemeAnnotations` component geïnitialiseerd met de nodige eigenschappen. Deze zijn het account ID, de bestandsnaam van de afbeelding, de *service* en een kleurenpallet. Tussen deze kleuren en eender welke hexadecimale waarde kan gekozen worden tijdens het stylen van de annotaties.

```
1  $(document).ready(function() {
2    var annotationProps = {
3      accountId: '{{account.id}}',
4      image: '{{image}}',
5      service: '{{service}}',
6      colors:  [
7      "#FF691F",
8      "#FAB81E",
9      "#7FDBB6",
10     "#19CF86"
11     ]};
12
13   ReactRenderer.render(
14     React.createElement(ThemeAnnotations, annotationProps),
15     document.getElementById('annotations'),
16     $.noop()
17   );
18 });
```

Listing 4.2: ThemeAnnotations component - initialisatie

4.1.2 Annoteren van de afbeelding

Het annoteren van een geüploade afbeelding vindt plaats in een pop-upvenster. Zoals vermeld in sectie 4.1.1, wordt hiervoor ook een React component aangemaakt. Deze component handelt elke verandering aan het canvas en zijn objecten af en staat ook in voor het exporteren van het volledige canvas.

Wanneer het canvas element aanwezig is in het DOM, wordt het canvas geïnitialiseerd. Dit gebeurt in de `componentDidMount()` functie van de React component. Zowel Facebook als Twit-

ter verwachten omslagfoto's van een bepaalde resolutie. Wordt hier niet aan voldaan, zullen de foto's worden geschaald. Om kwaliteitsverlies te voorkomen, worden de foto's dus best in volle resolutie güpload. Helaas kan het canvas niet aangemaakt worden met de originele dimensies van de foto's. Bij Facebook zou dit een canvas van 828 op 315 pixels zijn terwijl dit voor Twitter een canvas van 1500 op 500 pixels zou zijn. Het is helemaal niet handig voor een gebruiker om op een canvas met een breedte van 1500 pixels te werken. Daarom worden de dimensies van het canvas dynamisch berekend. Dankzij de berekeningen kan ook de ra tio (hoogte ten opzichte van breedte) van de afbeeldingen behouden worden. Wanneer de afbeelding dan uiteindelijk gegenereerd wordt, moeten de dimensies simpelweg met dezelfde verkleiningsfactor vermenigvuldigd worden om zo een afbeelding met gewenste dimensies te bereiken. Het canvas wordt als volgt aangemaakt:

```
1  getInitialState: function () {
2    return {
3      facebookCover: {width: '828', height: '315'},
4      twitterCover: {width: '1500', height: '500'}
5    }
6  },
7  componentDidMount: function () {
8    var dimensions = {};
9    if (this.props.service === 'twitter') {
10     ratio = this.state.twitterCover.width / this.state.twitterCover.height;
11     dimensions = this.state.twitterCover;
12   }
13   else if (this.props.service === 'facebook') {
14     ratio = this.state.facebookCover.width / this.state.facebookCover.height;
15     dimensions = this.state.facebookCover;
16   }
17
18   this.canvas = new fabric.Canvas('annotationCanvas', {
19     width: this.refs.container.offsetWidth,
20     height: this.refs.container.offsetWidth / ratio,
21     preserveObjectStacking: true,
22     renderOnAddRemove: true,
23     multiply: ratio,
24     service: this.props.service,
25     dimensions: dimensions,
26   });
27 }
```

Eens het canvas bestaat, kunnen objecten toegevoegd worden. Met behulp van de `fromUrl(url, callback)` methode van het *Image* object kan een afbeelding aan het canvas toegevoegd worden. De URL wordt simpelweg opgesteld aan de hand van de meegegeven bestandsnaam. Eens de afbeelding ingeladen is worden enkele transformaties uitgevoerd om de afbeelding gepast weer te geven. Zo wordt de afbeelding gecentreerd, geschaald tot de volledige breedte van het canvas en naar de achtergrond van het canvas geplaatst. Deze transformaties zorgen ervoor dat het volledige canvas bedekt wordt door de afbeelding en dat alle andere objecten, zoals tekst en KPI's, bovenop de afbeelding terecht zullen komen.

Gewone tekst wordt als een `IText` object toegevoegd aan het canvas. Hierdoor is de gebruiker in staat om de tekst aan te passen in het canvas zelf en hoeft dus geen extra inputveld te worden voorzien. Net zoals de afbeelding naar de achtergrond wordt gestuurd, worden tekst objecten

naar de voorgrond gebracht met behulp van de `bringToFront()` functie van het canvas. Een tekst object wordt als volgt aan het canvas toegevoegd:

```
var text = new fabric.IText('click here to add your text', {
  fontFamily: 'Arial',
  left: this.canvas.width / 2,
  top: this.canvas.height / 2,
  fontSize: 30,
  transparentCorners: false,
  textAlign: 'left',
  lockUniScaling: true,
  borderColor: '#00b4d0',
  cornerColor: '#00b4d0',
  centeredScaling: true,
  textType: 'text',
});

this.canvas.add(text);
this.canvas.bringToFront(text);
```

Naast de standaard eigenschappen zoals tekstgrootte, lettertype en positie op het canvas bevat het object een extra `textType` eigenschap. Deze maakt duidelijk dat het om gewoon tekst object gaat en niet over een KPI. Na het toevoegen van het object aan het canvas wordt het naar de voorgrond gebracht om er zeker van te zijn dat de tekst bovenop de afbeelding terecht komt.

De gebruiker kan kiezen tussen vier verschillende soorten KPI's: 'respons tijdens het laatste uur', 'respons tijdens de laatste dag', 'geholpen gebruikers de laatste dag' en 'geholpen gebruikers de laatste zeven dagen'. Bij het toevoegen van een KPI aan het canvas moet het soort KPI dus ook opgeslagen worden.Wat een probleem vormt tijdens de uitwerking is de positionering van de KPI's. Aangezien dit variabele getallen zijn, is een getal van vier of vijf cijfers lang niet uit te sluiten. Hoewel de inhoud van een `IText` object uit te lijnen is (links, rechts of gecenteerd), is dit pas merkbaar bij tekst bestaande uit meerdere lijnen. Daarom wordt geopteerd voor een `Textbox` object om de KPI's weer te geven. Tijdens het initialiseren van dit object kan de breedte ingesteld worden waardoor ook een enkel woord uitgelijnd kan worden. Via een *switch* statement wordt de correcte type KPI aan het object toegekend (zie onderstaand codefragment).

```
var mockValue = '';
switch (value) {
  case 'last_hour_response_time':
    mockValue = '12';
    break;
  case 'last_day_response_time':
    mockValue = '30';
    break;
  case 'seven_days_serviced_users':
    mockValue = '218';
    break;
  case 'last_day_serviced_users':
    mockValue = '72';
    break;
  default:
    mockValue = 'NaN'
}
```

```

var KPI = new fabric.Textbox(mockValue, {
  fontFamily: 'Arial',
  left: this.canvas.width / 2,
  top: this.canvas.height / 2,
  fontSize: 70,
  textAlign: 'center',
  textType: 'kpi',
  kpiType: value,
  width: 400,
  editable: false
});

this.canvas.add(KPI);
this.canvas.bringToFront(KPI);
```

Gebruikers kunnen ook een *template* gebruiken. Dit zal enkele tekst objecten op het canvas plaatsen alsook de gekozen KPI. Zo kunnen gebruikers zeer eenvoudig hun afbeeldingen annoteren met bijvoorbeeld: 'We took care of 500 customers last week'. Alle elementen worden gecentreerd op het canvas en kunnen door de gebruiker zelf nog aangepast worden.

4.1.3 Editeren van tekst

Één van de *features* tijdens het annoteren van een afbeelding is het stylen van de tekst. Dit maakt het mogelijk eigenschappen zoals tekstgrootte, lettertype, kleur en gewicht (vet of schuin) aan te passen. Om dit op een modulaire manier te verwezenlijken, wordt gebruik gemaakt van een React component. Aan de `TextEditor` component worden enkele eigenschappen meegegeven om een bestaand tekst object aan te kunnen passen. Onder deze eigenschappen valt een `onChange()` functie, een `currentSettings` object en een `selectedColor` string. Wanneer een object in het canvas wordt geselecteerd, wordt het `object:selected` event getriggered. In de *handler* van dit event wordt, na confirmatie dat het om een tekst object gaat, de state van de `Theme` component aangepast naar de huidige instellingen van het object:

```
this.canvas.on({'object:selected': function () {
  var object = null;
  if (this.isTextSelected()) {
    object = this.canvas.getActiveObject();

    this.setState({
      currentSettings: {
        fontSize: object.getFontSize(),
        fontWeight: object.getFontWeight(),
        fontFamily: object.getFontFamily(),
        fontStyle: object.getFontStyle(),
        fontColor: object.getFill(),
        textAlign: object.getTextAlign(),
      }, textSelected: true, objectInfo: kpiType
    });
  }}.bind(this)
});
```

De update van de state zal ervoor zorgen dat de component opnieuw gerendered wordt. Met behulp van een check in de `render` functie van de `Theme` component kan de `TextEditor` component geïnitialiseerd worden met de nodige eigenschappen (codefragment 4.3). De belangrijkste eigenschap is de `onChange()` functie die vanuit de `TextEditor` component getriggered kan worden. Het triggeren gebeurt wanneer een inputveld van de editor verandert. De huidige instellingen worden opgehaald en de nodige aanpassingen worden gemaakt. Uiteindelijk wordt de state aangepast en wordt de `onChange()` functie aageroepen met de nieuwe instellingen als parameter (codefragment 4.4). Bij het triggeren van de `onChange()` zal het geselecteerde object in de `Theme` component aangepast worden met de nieuwe instellingen. Hierna wordt de `renderAll()` functie van het Fabric.js canvas aangeroepen om de aangepassingen op het canvas te tonen (codefragment 4.3).

```
1  <TextEditor onChange={function (value) {
2    if (this.isTextSelected()) {
3      var object = this.canvas.getActiveObject();
4      object.setFontWeight(value.fontWeight);
5      object.setFontSize(value.fontSize);
6      object.setFontStyle(value.fontStyle);
7      object.setFontFamily(value.fontFamily);
8      object.setTextAlign(value.textAlign);
9      this.canvas.renderAll();
10   }}.bind(this)}
11   currentSettings={this.getActiveObjectSettings()}
12   selectedColor={this.getActiveObjectSettings().fontColor}
13   colors={this.props.colors}>
14 </TextEditor>
```

Listing 4.3: Theme component - Text editor

```
1  <li>
2    <a className={this.getCurrentSettings().fontWeight === 'bold' ? 'button
       primary' : 'button'}
3      onClick={function () {
4      var settings = this.getCurrentSettings();
5      settings.fontWeight = (settings.fontWeight === 'bold') ? 'normal' : 'bold'
         ;
6      this.setState({currentSettings: settings});
7      this.props.onChange(settings);
8    }.bind(this)}>B</a>
9  </li>
```

Listing 4.4: TextEditor component - toggle bold

4.1.4 Transformaties van tekst

Objecten in het canvas kunnen onmiddellijk verplaatst, geschaald en geroteerd worden. Zo kan de gebruiker zeer gemakkelijk de achtergrondafbeelding herpositioneren en herschalen. Enkel het schalen van tekst en KPI's vormt een probleem in de Fabric bibliotheek. Wanneer een tekst object namelijk geschaald wordt, wordt de lettergrootte niet aangepast. Het schalen zorgt er enkel

voor dat er als het ware wordt ingezoomd op het object zelf. Tijdens het ontwerpen van een canvas (clientside) vormt dit geen probleem maar wanneer de afbeelding moet gegenereerd worden in de backend kan dit problematisch zijn. Dan moet immers een canvas aangemaakt worden met grotere afmetingen zodat de afbeelding in volledige resolutie bekomen wordt. Alle eigenschappen van ieder object (hoogte, breedte, positie) zullen dan ook met dezelfde schaalfactor (als het canvas) vermenigvuldigd worden om een correcte afbeelding te verkrijgen. Wanneer de positionering dan nog eens vermenigvuldigd wordt met het berekende ratio (zie 4.1.2), kan het voorkomen dat negatieve waarden bekomen worden. Dit vormt vooral een probleem wanneer tekst op de uitersten van het canvas (dus bijvoorbeeld in hoekpunten) geplaatst wordt.

Om dit te voorkomen wordt ervoor gezorgd dat tekst in het canvas op een andere manier schaalt. Door gebruik te maken van het `object:scaling` event wordt de tekstgrootte aangepast in plaats van de schaal van het object (zie codefragment 4.5). Bij het triggeren van het event wordt gekeken of het wel degelijk om een tekst object gaat (de afbeelding kan immers normaal geschaald worden). Daarna wordt de huidige tekstgrootte vermenigvuldigd met de horizontale schalingsfactor (merk op: de horizontale schalingsfactor wordt gebruikt omdat de `lockUniScaling` eigenschap van het object actief is waardoor zowel x-als y-as met dezelfde factor geschaald worden). Uiteindelijk wordt de tekstgrootte afgerond naar een geheel getal omdat pixels als eenheid gebruikt worden. Zowel de horizontale als verticale schaal wordt opnieuw op 1 geplaatst om eerder beschreven probleem met schaling te voorkomen.

```
1  this.canvas.on('object:scaling', function (e) {
2    if (e.target && this.isTextSelected()) {
3      e.target.fontSize *= e.target.scaleX;
4      e.target.fontSize = e.target.fontSize.toFixed(0);
5      e.target.scaleX = 1;
6      e.target.scaleY = 1;
7
8      this.setState({
9        currentSettings: {
10       fontSize: e.target.getFontSize(),
11       fontWeight: e.target.getFontWeight(),
12       fontFamily: e.target.getFontFamily(),
13       fontStyle: e.target.getFontStyle(),
14       fontColor: e.target.getFill(),
15       textAlign: e.target.getTextAlign()}
16     });
17   }
18 }.bind(this));
```

Listing 4.5: ThemeAnnotations component - text scaling

4.1.5 Verwijderen van objecten

Naast het toevoegen van tekst moet de gebruiker in staat zijn om tekst te verwijderen. De Fabric.js bibliotheek bezit hiervoor reeds de benodigde functionaliteit. Via de `remove()` functie van het canvas, kan een object van het canvas verwijdert worden. Gebruikers mogen niet in staat zijn om de afbeelding te verwijderen dus moet eerst bepaald worden of de geselecteerde objecten wel degelijk tekst objecten zijn. Wanneer slechts één object geselecteerd wordt, is dit eenvoudig te

verwezenlijken. Moeilijker wordt het wanneer meerdere objecten geselecteerd worden. Hiervoor moeten alle objecten in de actieve groep (de geselecteerde objecten) overlopen worden. Wanneer het type van de objecten overeen komt met deze van tekst, kan het object verwijdert worden.

```
if (this.canvas.getActiveGroup()) {
  var objectsInGroup = this.canvas.getActiveGroup().getObjects();
  objectsInGroup.forEach(function (object) {
    if (object.type === 'i-text' || object.type === 'textbox') {
      this.canvas.remove(object);
    }
  }.bind(this));
}
```

Listing 4.6: ThemeAnnotations component - delete group

4.1.6 Importeren van een canvas

Logischerwijze kan een thema later aangepast worden. Vanuit de databank wordt alle informatie van het thema aan de `Theme` component meegegeven. Om de annotaties aan te passen, moeten deze dus vanuit de `Theme` component doorgestuurd worden naar het pop-upvenster, de `ThemeAnnotations` component. In eerste instantie lijkt de eenvoudigste oplossing om de vereiste informatie van het thema mee te geven als URL parameters aan de pop-up. Dit vormt geen probleem voor de bestandsnaam van de afbeelding en de service maar het doorsturen van een lange JSON string in de URL is niet bepaald handig. Volgens het RFC7230 document moeten alle Hypertext Transfer Protocol (HTTP) zenders en ontvangers *request lines* van minimaal 8000 octetten (8 bits) lang [30]. Of dit daadwerkelijk het geval is hangt af van gebruikte software. Hoewel dus geen limiet staat op de lengte van een URL is het aangeraden om niet meer dan 2000 karakters te gebruiken. Dit verzekerd dat elke client-en serverside combinatie de URL kan verwerken.

Om de annotaties over te brengen naar de `ThemeAnnotations` component wordt gebruik gemaakt van events. Een globaal `ThemeDispatcher` object wordt gedefinieerd waar verschillende events aan toegevoegd worden. Wanneer de `Theme` component gemount is, worden de nodige events aangemaakt in de `ThemeDispatcher` (zie codefragment 4.7). Ook worden de nodige *handlers* voor deze events uitgewerkt.

```
componentDidMount: function () {
  $(ThemeDispatcher).on('annotations', this.annotationHandler);
  $(ThemeDispatcher).on('request-annotations', this.requestHandler);
},
requestsHandler: function () {
  $(ThemeDispatcher).trigger('response-annotations', this.getAnnotations());
},
annotationHandler: function (event, annotations) {
  this.setState({annotations: annotations});
},
getAnnotations: function () {
  return (
    this.state.annotations || this.props.annotations
  );
},
```

Listing 4.7: Theme component - events

Voor het ophalen van annotaties zorgen de `'request-annotations'` en `'response-annotations'` events. Bij het afhandelen van het *request* event wordt onmiddellijk een *response* gestuurd naar de `ThemeAnnotations` component met ofwel de annotaties uit de database of de reeds aangepaste annotaties uit de state.

```
1 componentDidMount: function () {
2     $(ThemeDispatcher).on('response-annotations', this.handleAnnotations);
3   $(ThemeDispatcher).trigger('request-annotations');
4 },
5 onSubmit: function (e) {
6   e.preventDefault();
7   $(ThemeDispatcher).trigger('annotations', this.exportCanvas());
8 },
```

Listing 4.8: ThemeAnnotations component - events

Bij het ontvangen van de annotaties worden deze ingeladen in het canvas met behulp van de `loadFromJSON()` functie. Na het inladen van alle objecten in het canvas, moeten nog enkele objecten aangepast worden. Tijdens het opslaan van het canvas worden alle objecten met hun eigenschappen geserialiseerd, dus ook de link naar de achtergrondafbeeling. Dit is nodig om vanuit de JSON data opnieuw een exacte kopie van het canvas te genereren. Maar dit kan voor problemen zorgen wanneer een gebruiker er voor kiest om zijn/haar afbeelding te veranderen. Dan zal de geüploade afbeelding wel aanwezig zijn in de state maar de annotaties zullen nog niet aangepast zijn waardoor, bij het openen van het annotatie venster, de oude afbeelding ingeladen wordt. Om dit te voorkomen wordt de afbeelding verwijdert uit het canvas en wordt een nieuw `fabric.Image` object aangemaakt met de nieuwe afbeelding. Op die manier kan de gebruiker simpelweg de afbeelding veranderen zonder reeds aangemaakte annotaties te verliezen.

4.1.7 Exporteren van een canvas

Wanneer de gebruiker zijn/haar annotaties opslaat (zie codefragment 4.9, `onSubmit()`), wordt het `'annotations'` event getriggered. Aan dit event wordt het volledige canvas meegegeven als een JSON string. De *handler* in de `Theme` component zorgt er voor dat deze annotaties opgeslagen worden in de state van de component. Zo worden de annotaties telkens up-to-date gehouden.

Het omzetten van het canvas naar JSON data is zeer eenvoudig te verwezenlijken dankzij de `toJSON()` methode die het canvas bezit. Als argument van deze functie kan een array met extra eigenschappen, die geëxporteerd moeten worden, meegegeven worden. Dit is zeer handig wanneer extra informatie noodzakelijk is tijdens het terug opbouwen van het canvas. Zo worden de breedte, hoogte, ratio (breedt/hoogte), service, tekst type, KPI type en de dimensies van het canvas ook opgeslagen. Elk van deze eigenschappen zijn onmisbaar tijdens het heropbouwen van het canvas in de backend.Meer hierover in sectie 4.2.

```
1 exportCanvas: function () {
2   if (this.canvas) {
3     var data = JSON.stringify(this.canvas.toJSON(['width', 'height', 'multiply
        ', 'service', 'textType', 'kpiType', 'dimensions']));
4     return (data);
5   }
```

```
6 }
```

Listing 4.9: ThemeAnnotations component - exporteren van het canvas

Normaal gezien is het de bedoeling dat ieder object op het canvas geëxporteerd wordt om later exact hetzelfde canvas op te kunnen bouwen. Een uitzondering hierop is het annoteren van een omslagfoto voor Twitter. De foto wordt namelijk niet volledig weergegeven op een profiel. Langs de boven-en onderkant wordt de omslagfoto bedekt door een stuk van de Twitter header. Om duidelijk te maken aan de gebruiker dat deze zones niet zichtbaar zullen zijn, worden twee afbeeldingen bovenop het canvas geplaatst.

Natuurlijk mogen deze afbeeldingen niet behouden blijven wanneer de afbeelding aangemaakt moet worden. Daarom wordt er voor gezorgd dat deze afbeeldingen niet opgenomen worden in de JSON string. Dit wordt verwezenlijkt door de `excludeFromExport` eigenschap van deze afbeeldingen.

4.2 Backend implementatie

Eenmaal de gebruiker een thema met omslagfoto heeft aangemaakt, kan deze toegekend worden aan een profiel. Op het moment dat het thema actief moet worden, moet vanuit de annotaties de afbeelding aangemaakt worden. Deze kan dan via de Facebook of Twitter API

4.2.1 Opzetten van de Node.js service

Uit de, door de gebruiker opgeslagen, annotaties moet op een gegeven tijdstip een afbeelding gegenereerd worden. Om deze annotaties in te lezen en om het canvas terug op te bouwen moet er logischerwijze opnieuw gebruik gemaakt worden van de Fabric.js bibliotheek. Er is dus nood aan een manier om de JavaScript bibliotheek buiten de browser te kunnen gebruiken. Dit kan verwezenlijkt worden met Node.js. Deze JavaScript *runtime* is gebouwd op de V8 JavaScript *engine* [31]. De Fabric.js bibliotheek voorziet zelfs enkele methodes om speficiek in een Node omgeving te gebruiken.

Via de Node Package Manager (NPM) worden de vereiste paketten gedownload. Hiertoe behoren zowel de Fabric module als Express, `body-parser` en `cors`. Express is een van de meest gebruikte Node.js *frameworks* en maakt het opzetten van een server zeer eenvoudig. Met een weinig aan code kan met Express een server opgezet worden met de vereiste functionaliteit. Na het opzetten van de Express app kunnen de nodige routes toegevoegd worden. Het belangrijkst is een route die meegekregen annotaties en KPI's omzet naar een afbeelding voor de correcte service. Dit moet dus een route zijn die een POST *request* afhandelt 4.10. Zowel de annotaties als de KPI's worden doorgestuurd in de *request body*. Deze worden als een JSON string doorgestuurd waardoor deze nog geparst moeten worden om ze te gebruiken in de code. De x-www-form-urlencoded *body* van de request kan met behulp van de `body-parser` module correct ingelezen worden. De doorgestuurde annotaties kunnen zeer veel objecten bevatten waardoor de JSON strings zeer lang worden. Door gebruik te maken van de *extended* optie van de `body-parser`, kunnen lange objecten en arrays geëncodeerd worden.

```
1 app.use(bodyParser.urlencoded({
```

```
2    extended: true
3  }));
4
5  app.post('/render', function (request, response) {
6    var data = JSON.parse(request.body.data);
7    var KPIs = JSON.parse(request.body.kpis);
8
9    renderer.render(data, KPIs).then((data) => {
10     return response.status(200).send({status: 'Image successfully generated',
         image:  data.toString('base64')});
11   }).catch((error) => {
12     return response.status(400).send(error);
13   });
14 });
15 }
```

Listing 4.10: index.js - Node.js server

De `cors` module is nodig om *Cross-origin resource sharing* (CORS) mogelijk te maken in de server. Dit maakt het mogelijk om *resources* te laden vanuit een ander domein dan hetgene waarvan het originele bestand komt. In de service komt dit voor wanneer de afbeeldingen ingeladen worden tijdens het opstellen van het canvas. De *source* van de afbeeldingen zal namelijk hetzelfde domein bevatten als waar de afbeelding geüpload wordt. Wordt bijvoorbeeld gewerkt op het *beta.engagor.com* domein, dan zal de afbeelding ingeladen worden vanuit *http://beta.engagor.com/images/*. Als geweten is dat de service op het *server603:3000* loopt, dan is duidelijk dat hier een *resource* vanop een ander domein ingeladen wordt. Dit kan dus een probleem vormen wanneer de `cors` module niet geïnstalleerd is. Een andere manier om CORS mogelijk te maken is het gebruiken van specifieke *response headers*. Door de `Access-Control-Allow-Origin` zo te configureren dat elke oorsprong wordt toegelaten (door het gebruik van een wildcard: *), wordt CORS mogelijk. Naast deze *header* moet ook vastgelegd worden welke *headers* toegestaan worden: `Access-Control-Allow-Headers: Origin`.

4.2.2 Opbouwen van het canvas

Éénmaal de routes opgesteld zijn, kan de logica geïmplementeerd worden. Eerst worden de `fabric` en `canvas-prebuilt` modules geïmporteerd [32]. `canvas-prebuilt` zorgt er voor dat het canvas gebruikt kan worden buiten de *browser*. Eens het Fabric canvas geïnstantieerd is, kunnen de annotaties en KPI's omgezet worden in een afbeelding. Eerder werd het probleem met verschillen in dimensies en ratios besproken (zie sectie 4.1.2). Om het canvas met al zijn objecten de correcte dimensies te geven, moeten eerst enkele berekeningen uitgevoerd worden. In de *request body* zijn naast de annotaties ook de *service*, originele dimensies van het canvas (zoals deze in het dialoog getoond werd) en de vereiste dimensies van de *service* terug te vinden. Eerst en vooral worden de correcte dimensies aan het canvas gegeven. Wordt bijvoorbeeld een omslagfoto voor Twitter gegenereerd, krijgt het canvas een breedte van 1500 pixels en een hoogte van 500 pixels. Wanneer de dimensies van het canvas wijzigen, moet tekst die op het canvas staat ook aangepast worden. Door de hoogte van het nieuwe canvas te delen door de hoogte van het canvas waarop de gebruiker annoteerde, wordt de schalingsfactor bekomen (zie codefragment 4.11):

$$schalingsfactor = \frac{\text{hoogte serverside canvas}}{\text{hoogte clientside canvas}}$$

Deze schalingsfactor zal gebruikt worden om elk object in het canvas correct te schalen zodat exact dezelfde afbeelding bekomen wordt zoals deze door de gebruikt werd aangemaakt.

```
1  return new Promise(function(resolve, reject) {
2  var originalHeight = data["height"];
3  var originalWidth = data["width"];
4  var service = data['service'];
5  var dimensions = data['dimensions'];
6
7  var fabricCanvas = fabric.createCanvasForNode(800, 200);
8
9  if (service === 'twitter') {
10 fabricCanvas = fabric.createCanvasForNode(parseInt(dimensions.width), parseInt
       (dimensions.height));
11 }
12 else if (service === 'facebook') {
13 fabricCanvas = fabric.createCanvasForNode(parseInt(dimensions.width),
       parseInt(dimensions.height));
14 }
15
16 var height = fabricCanvas.getHeight();
17 var scale = height / originalHeight; // multiplier for resizing canvas +
       objects
18
19 //Inlezen van de JSON data
20 }
```

Listing 4.11: renderer.js - Image renderer

Tijdens het verwerken van de annotaties zullen enkele functies asynchroon uitgevoerd worden. Om uiteindelijk een *response* terug te kunnen sturen met de aangemaakte afbeelding, moet dus rekening gehouden worden met dit asynchrone gedrag. Daarom wordt gebruik gemaakt van een `Promise`. Wanneer het opstellen van het canvas succesvol verloopt, wordt deze *promise* opgelost door de `resolve` functie. Wordt in de code gemerkt dat één of meerdere KPI's ontbreken, wordt de `reject` functie opgeroepen (zie codefragment **??**).

Zoals vermeld in sectie 3.3.2 stelt Fabric.js enkele functies beschikbaar voor gebruik in een Node.js omgeving. Zo kan met behulp van de `loadFromJSON()` methode een canvas opgesteld worden vanuit JSON data. Wanneer het canvas opnieuw bestaat, worden alle objecten aangepast aan de nieuwe dimensies. De eerder berekende schalingsfactor (zie codefragment 4.11) wordt gebruikt om de positie, hoogte en breedte van ieder object op gepaste wijze te vergroten. Bij tekst objecten wordt de tekstgrootte hier ook mee vermenigvuldigd. Zo wordt exact dezelfde afbeelding bekomen zoals deze door de gebruiker aangemaakt werd maar met de correcte dimensies om te gebruiken op Facebook of Twitter.

Één van de belangrijkste *features* van het annoteren is de mogelijkheid om KPI's te plaatsen op een afbeelding. Aangezien dit variabele waarden zijn, moeten ze ook om de zoveel tijd aangepast worden. Bij het aanspreken van de Node.js service worden naast de annotaties ook de pas berekende KPI's doorgegeven. Tijdens het schalen van de objecten wordt gekeken of het een

object van het `textType` 'kpi' is. Is dit het geval en is de corresponderende KPI aanwezig in de ontvangen data, kan de huidige tekst vervangen worden door de waarde van de KPI.

4.2.3 Exporteren van het canvas

Nu het canvas volledig is opgebouwd met de correcte afmetingen en KPI's, moet hieruit een afbeelding gegenereerd worden. Deze wordt dan uiteindelijk ingesteld als omslagfoto op Facebook of Twitter. Om de afbeelding te genereren wordt gebruik gemaakt van de `createPNGStream()` methode van Fabric. Bij het aanroepen van deze functie op het canvas wordt een *stream* opgezet van omgezette data. Op het moment dat deze stream data voortbrengt, wordt het `data` event uitgezonden. In de *handler* van dit event kan dan verder gewerkt worden met de stukken data. Zo kan een nieuwe stream omgezet worden om een PNG bestand op te bouwen. Hoewel dit ook zijn toepassingen heeft, is dit in niet gewenst. De afbeeldingsdata moet namelijk teruggestuurd worden als *response* zodat er verder mee gewerkt kan worden. Om te vermijden dat de foto eerst nog op de fileserver geplaatst moet worden, wordt er voor gekozen om onmiddellijk de base64 geëncodeerde afbeelding terug te sturen. Zoals in codefragment 4.12 te zien is, worden de stukjes data aan een *array* toegevoegd. Wanneer er geen data meer beschikbaar is wordt het `'end'` event uitgezonden. Bij het afhandelen van dit event zullen alle elementen van de *array* geconcateneerd worden via de `Buffer.concat()` functie. Dit resulteert in een base64 geëncodeerde afbeelding die simpelweg als *response* teruggestuurd kan worden. Er wordt gebruik gemaakt van een buffer omdat de `Buffer` klasse speciaal ontworpen is om rauwe binaire data te verwerken en dat is exact wat de PNG stream voorstelt.

```
1  var stream = fabricCanvas.createPNGStream();
2  var base64data = [];
3
4  stream.on('data', function (chunk) {
5    base64data.push(chunk);
6  });
7
8  stream.on('end', function () {
9    var result = Buffer.concat(base64data);
10   console.log('Picture for ' + service + ' was successfully generated');
11   resolve(result);
12 });
```

Listing 4.12: renderer.js - aanmaken van base64

4.2.4 Testen van de Node.js service

Om correcte werking van de service te garanderen is het logisch dat hiervoor een test geschreven wordt. Hier moet dus de functionaliteit van een Node.js service getest worden. Om dit te verwezenlijken wordt gebruik gemaakt van Jest **??**. Dit *framework* van Facebook maakt het zeer eenvoudig om JavaScript te testen. Jest wordt geïnstalleerd via NPM zoals elke andere module. Voor de test uit te werken, worden enkele instellingen aangepast. Aangezien Jest zal gebruikt worden in een Node omgeving, wordt *node* ingesteld als `testEnvironment` in het `package.json` bestand.

Wat hier getest moet worden is het genereren van een afbeelding. Alle logica, die de annotaties en KPI's omzet in een afbeelding, zit reeds in een aparte module verwerkt (zie sectie 4.2.2). In een nieuw test bestand, met een `.test.js` extensie, wordt de test opgezet. Deze test leest een reeds bestaande afbeelding in met behulp van de `fs` module waarna deze omgezet wordt naar base64. Daarna wordt een nieuwe afbeelding gegenereerd uit JSON data en KPI's. Uiteindelijk worden beide base64 *strings* met elkaar vergeleken via de `expect()` methode en `toBe()` *matcher*. Zijn beide geëncodeerde afbeeldingen aan elkaar gelijk, is de test geslaagd en werkt de code zoals het hoort.

```
1  test('test image rendering', () => {
2    var image = fs.readFileSync('./test-twitter.png');
3    var buffer = new Buffer(image).toString('base64');
4
5    renderer.render(data, KPIs).then((data) => {
6      expect(data.toString('base64')).toBe(buffer);
7    }).catch((error) => {
8      console.log(error);
9    });
10 });
```

Listing 4.13: render.test.js - Testen van de service

4.2.5 Scheduling / Plannen van thema's

Zoals vermeld in de beschrijving van het project (zie sectie 2.1), kunnen gebruikers voor een bepaald profiel verschillende thema's inplannen. Om een thema te plannen kan gekozen worden tussen de reeds bestaande *business hour schedules* of een aangepaste timing.

Figuur 4.2: Plannen van een thema

Na het opstellen van een planning kan het profiel opgeslagen worden. Is dit een nieuw profiel, waar nog nooit iets voor werd ingesteld, dan wordt hiervoor een nieuwe *task* aangemaakt. Een *task* wordt gekenmerkt door een ID, een groep, een account, data, prioriteit, `nextScheduleDate` en `lastSuccessfulRun`. Aangezien één taak per profiel wordt aangemaakt, volstaat het om het profiel ID te gebruiken om de taken van elkaar te onderscheiden. De groep bepaald waartoe de taak behoort, hier dus de `profile-manager`. Als data wordt simpelweg het ID van het profiel gebruikt. De data bevat alles nodige informatie om de taak correct uit te voeren. Er wordt verwacht dat de *task* onmiddellijk uitgevoerd wordt dus wordt als `nextScheduleDate` het huidige tijdstip genomen.

Eens de taak aangemaakt is, wordt deze in de database opgeslagen en via de *scheduler*, indien nodig, op de *queue* geplaatst. Eens een taak in de *queue* staat, wordt deze op zijn beurt uitgevoerd. Het uitvoeren van de taak gebeurt met behulp van de *dispatcher*. Deze linkt de correcte *handler* aan de taak zelf. In de *handler* wordt de taak dan afgehandeld en wordt een volgende taak gepland. Het schema op Figuur 4.3 toont het volledige proces.

Figuur 4.3: UniCrawler

Aangezien het de bedoeling is dat het thema op een profiel up-to-date gehouden wordt, wordt een taak om de vijf minuten herhaald. Bij het afhandelen van de taak worden verschillende checks uitgevoerd om te bepalen of het thema daadwerkelijk toegepast moet worden. Wanneer hetzelfde thema nog steeds actief is of de KPI's niet verandert zijn, is het nutteloos om de foto's opnieuw te genereren en te uploaden. Voor er overgegaan kan worden naar het toepassen van het thema, moet natuurlijk eerst bepaald worden welk thema actief moet zijn.

Om dit te weten te komen, wordt gebruik gemaakt van een algoritme. Dit algoritme haalt voor een profiel alle ingeplande thema's op. Voor ieder thema wordt gekeken of deze een aangepaste *timing* bevat of *business hours* of beide. Valt het huidige tijdstip (waarop de taak dus wordt afgehandeld) binnen de ingestelde *timing* dan wordt alle informatie over dat bepaalde thema bijgehouden. Hier moet vermeld worden dat tijdstippen uitgedrukt worden in Unix tijd om het vergelijken te vereenvoudigen. Eens de *custom timing* bekeken is, worden alle ingestelde *business hours schedules* overlopen. Gebruikers kunnen namelijk meerdere *schedules* selecteren om zo de planning gedetailleerder te maken. De tijdstippen van deze *schedules* worden vergeleken met het reeds berekende tijdstip en het meest recente wordt toegevoegd aan een lijst. Uit deze lijst wordt dan het meest recentste tijdstip genomen. Zo wordt het thema gevonden dat op dat moment actief moet zijn op basis van de *business hours schedules*. Uiteindelijk wordt de berekende *custom timing* vergeleken met de *business hours* om te bepalen wat de bovenhand neemt.

Wordt dit algoritme op iedere set (zie Figuur 4.2) toegepast, wordt uiteindelijk een lijst bekomen

met het tijdstip van iedere set die op dat moment actief kan zijn. Met behulp van een sorteermethode worden alle sets aflopend geordend. De eerste in de lijst (met de hoogste waarde) is dan het actieve thema en moet dus ingesteld worden op het profiel. Wordt noch een *custom timing* noch een *business hours schedule* gevonden, dan wordt deze set eenvoudigweg aan dezelfde lijst toegevoegd met een timestamp die gelijk is aan 0. Het thema in deze set zal dus enkel actief zijn wanneer geen andere sets die aan de eisen voldoen, aanwezig zijn. Sets zonder enige specificaties kunnen dus als standaard thema voor een profiel ingesteld worden.

Verschillende sets kunnen dus zo geconfigureerd worden dat een complexe planning van thema's ontstaat. Zo kunnen gebruikers bijvoorbeeld een standaard thema ingesteld hebben voor hun profiel maar wordt tijdens de openingsuren op de omslagfoto aangegeven dat het bedrijf open is. Op bepaalde feestdagen zoals Kerstmis of Pasen kan dan weer een aangepaste profielfoto getoond worden.

4.2.6 Doorsturen naar Facebook/Twitter - Uploaden naar sociale media

Elke vijf minuten bestaat de kans dat een thema toegepast wordt en dus ook foto's geüpload moeten worden naar Facebook of Twitter. Binnen CX Social wordt reeds gebruik gemaakt van de Application Programming Interface (API) van deze platformen waardoor er reeds code aanwezig is voor het maken van een *request* naar elk van deze services. Deze code zal ook een correct authenticatie token, voor de ingelogde gebruiker, met de *request* meegeven. Er moet dus enkel een *request* gestuurd worden naar de specifieke *endpoints* waarmee profielfoto, omslagfoto en profielinformatie aangepast worden.

Twitter

Om de omslagfoto van een Twitter-profiel aan te passen wordt gebruik gemaakt van de `/account/updateo_profileo_banner` *endpoint* [33]. De afbeelding zelf kan base64 geëncodeerd of als *raw image data* meegegeven worden. Aangezien de omslagfoto's telkens door de Node.js service worden gegenereerd en dus in het base64 formaat ontvangen worden (zie sectie 4.2.3), kand de afbeelding rechtstreeks meegegeven worden aan het *endpoint*. Zoals te zien in codefragment 4.14 worden de URL naar de *endpoint*, de afbeelding, *request* methode en enkele opties verwacht om de *call* te maken. Een van de opties is het verwachte antwoord op de *request*. Dit kunnen zowel een HTTP status codes 200 (OK), 201 (Created) of 202 (Accepted) zijn.

```
1  try {
2    $updateBanner = $this->call(
3      self::API_VERSION . '/account/update_profile_banner',
4      array(
5        'banner' => $image
6      ),
7      'POST',
8        /* $multipart = */ false,
9      /* $options = */ array(
10     'expectedResponseCodes' => array(200, 201, 202)
11   )
12 );
13
14 } catch (Exception $ex) {
15   throw $ex;
```

```
16 }
```

Listing 4.14: Uploaden van omslagfoto naar Twitter

Het uploaden van de profielfoto verloopt op gelijkaardige manier. Via de `/account/updateo_profileo_image` *endpoint* wordt de base64 geëncodeerde afbeelding naar Twitter gestuurd [33]. In tegenstelling tot de omslagfoto moet een profielfoto niet geannoteerd worden. Hierdoor kan de afbeelding simpelweg vanop de fileserver gehaald worden. Deze foto wordt dan base64 geëncodeerd voor gebruik op Twitter.

Facebook

De Graph API van Facebook biedt de mogelijkheid om omslag-en profielfoto aan te passen. Dit is echter een heel stuk complexer bij Facebook dan bij Twitter. Facebook verwacht namelijk dat gebruikers een foto, die reeds op hun tijdlijn staat, zullen instellen als omslagfoto. Dit betekent dat een foto eerst geüpload wordt naar Facebook waarna deze als omslagfoto of profielfoto wordt ingesteld. Dit zorgt ervoor dat de foto niet enkel in het album met tijdlijnfoto's maar ook in het album met omslagfoto's of profielfoto's terecht komt. Wordt aangenomen dat een foto om de vijf minuten geüpload wordt, is al snel duidelijk dat de verschillende albums rond de 300 foto's zullen bevatten na één dag.

Dit is natuurlijk niet de bedoeling dus wordt dit probleem op volgende manier opgelost.

Met behulp van de `/page-id/photos` *edge* wordt een foto naar Facebook geüpload. Een foto naar een pagina uploaden kan op twee manieren: via een verwijzing naar een foto die reeds op het internet staat (dus via een URL) of de afbeelding toevoegen als `multipart/form-data`. Mits de gegenereerde foto's nog niet beschikbaar zijn op het internet, wordt voor de laatste methode geopteerd. Naast een *access token* en wat opties, wordt het pad naar het bestand als *source* parameter meegegeven. Aangezien het niet gewenst is om een notificatie te krijgen wanneer een omslag-of profielfoto wordt gewijzigd, wordt de *no_story* parameter ook actief gezet (zie codefragment 4.15) [34].

```
1  $uploadedPhotoId = $this->call(
2    '/' . Services_Facebook_Migrations::VERSION_2_8 . '/' . $this->getServiceId
       () . '/photos',
3    $this->getAccessToken(),
4    Services_Facebook_Manager::MODE_JSON,
5    'POST',
6    array(
7      'source' => '@' . realpath($tempFile),
8      'no_story' => true,
9    ),
10   /* options */ array(
11     'postFieldsAsString' => false,
12     'timeout' => static::UPLOAD_TIMEOUT,
13   )
14 );
```

Listing 4.15: Uploaden van een foto naar Facebook

Na het uploaden van een foto naar de tijdlijn wordt als antwoord het ID van deze foto ontvangen. Met een call naar de pagina zelf kan deze foto ingesteld worden als omslagfoto. Parameters hierbij zijn *cover* (het ID van de foto) en *offset_x* en *offset_y* van de omslagfoto (zie codefragment

4.16). Ook hier wordt de *no_feed_story* parameter toegevoegd om te vermijden dat volgers van de pagina, om de x aantal minuten, een notificatie ontvangen.

Uiteindelijk worden enkele maatregelen getroffen om het aantal foto's in de albums te beperken. Zo wordt vóór het uploaden van een nieuwe foto, het ID van de huidige afbeelding opgehaald. Na het instellen van de nieuwe afbeelding, kan de andere foto via het ID uit het album verwijdert worden. Uiteindelijk wordt de foto ook uit het tijdlijn album verwijdert. Op deze manier wordt het aantal foto's in de albums tot een minimum beperkt.

```
1  $coverPhoto = $this->call(
2    '/' . Services_Facebook_Migrations::VERSION_2_8 . '/' . $this->getServiceId
       (),
3    $this->getAccessToken(),
4    Services_Facebook_Manager::MODE_JSON,
5    'POST',
6    array(
7      'cover' => $uploadedPhotoId['id'],
8      'offset_x' => 0,
9      'offset_y' => 0,
10     'no_feed_story' => true
11   ),
12   array(
13     'timeout' => static::UPLOAD_TIMEOUT,
14   )
15 );
```

Listing 4.16: Instellen van omslagfoto op Facebook

Het uploaden van een profielfoto verloopt op gelijkaardige manier. Het instellen als avatar gebeurt hier via de `/page-id/picture` edge van de Graph API [35]. Als parameters worden *photo* (het ID van de foto in het tijdlijn album), *width*, *height* en *no_feed_story* meegegeven. De hoogte en breedte van de foto worden ingesteld op 200 pixesl omdat Facebook een foto van minimum 180 op 180 pixels verwacht [36].

Net zoals bij het instellen van een omslagfoto moeten de vorige profielfoto en de geüploade foto verwijdert worden om een overvloed aan foto's te vermijden. In tegenstelling tot de omslagfoto, kan het ID van de huidige profielfoto niet opgehaald worden. Wanneer het ID niet beschikbaar is, kan ook de correcte foto niet verwijdert worden. Om dit probleem op te lossen wordt het volledige album van profielfoto's opgehaald. Aangezien de eerste foto in dit album de huidige profielfoto is, kan het ID hier uit gehaald worden om zo de foto te verwijderen. Hoewel deze methode het ene probleem oplost, veroorzaakt het een ander. Het verwijderen van de eerste foto in het album zorgt er voor dat gebruikers zelf geen profielfoto kunnen instellen via Facebook zelf. Deze zal namelijk verwijdert worden wanneer een thema wordt toegepast. De kans dat dit voorkomt is echter miniem omdat verwacht wordt van gebruikers dat ze de foto aanpassen vanuit de Profile Manager en niet op het profiel zelf.

Bibliografie

[1] F. Lemaitre, "Engagor, now a part of clarabridge." http://cxsocial.clarabridge.com/engagor-now-a-part-of-clarabridge/, 2015. Geraadpleegd op 26/02/17.

[2] Clarabridge, "Customer experience management software - clarabridge." https://clarabridge.com, 2017. Geraadpleegd op 09/03/17.

[3] Clarabridge, "Contact clarabridge." http://www.clarabridge.com/contact/, 2015. Geraadpleegd op 09/03/17.

[4] Clarabridge, "Cxsocial - a clarabridge product." http://3l3uol24eg0i3qephf1i107x.wpengine.netdna-cdn.com/wp-content/uploads/2017/03/cx-social_clarabridge-2017_logo-small.png, 2017. Geraadpleegd op 06/05/17.

[5] nn, "Cx social - developer documentation." https://developers.engagor.com/team, 2017. Geraadpleegd op 09/03/17.

[6] darkredz, "Doophp." https://github.com/darkredz/DooPHP, 2011. Geraadpleegd op 05/03/17.

[7] Facebook, "A javascript library for building user interfaces - react." https://facebook.github.io/react/, 2017. Geraadpleegd op 09/03/17.

[8] Engagor, "Engagor webapplicatie." https://app.engagor.com/, 2017. Geraadpleegd op 09/03/17.

[9] Engagor, "Clarabrige cx social dev stage." https://app.engagor.com/, 2017. Geraadpleegd op 10/03/17.

[10] J. Dierickx, "Cx social preview mode - invision." https://projects.invisionapp.com/d/main#/console/5000066/213989606/preview, 2017. Geraadpleegd op 09/03/17.

[11] J. Zaytsev, "Jsdoc: Home." http://fabricjs.com/docs/, 2015. Geraadpleegd op 13/03/17.

[12] J. Zaytsev, "Intro to fabric.js. part 1." http://fabricjs.com/fabric-intro-part-1, 2008. Geraadpleegd op 13/03/17.

[13] RegalVapor, nmve, teoli, fscholz, Jeremie, ethertank, Sheppy, jviereck, markg, lmorchard, and ernestd, "Drawing text - web apis." https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_text, 2017. Geraadpleegd op 15/03/17.

[14] J. Zaytsev, "Fabric.js object caching." http://fabricjs.com/fabric-object-caching/, 2015. Geraadpleegd op 13/03/17.

[15] J. Zaytsev, "Fabricjs - javascript canvas library." https://github.com/kangax/fabric.js/, 2017. Geraadpleegd op 18/03/17.

[16] J. Zaytsev, "Intro to fabric.js. part 3." http://fabricjs.com/fabric-intro-part-3#serialization, 2008. Geraadpleegd op 15/03/17.

[17] abbycar, Sebastianz, Sheppy, tifosi, jpmedley, fscholz, kscarfone, and Jeremie, "Svg element - web apis." https://developer.mozilla.org/en-US/docs/Web/API/SVGElement, 2016. Geraadpleegd op 18/03/17.

[18] A. Lavrenov, "Html5 canvas stage data url tutorial." https://konvajs.github.io/docs/data_and_serialization/Serialize_a_Stage.html, 2017. Geraadpleegd op 20/03/17.

[19] A. Lavrenov, "Load html5 canvas stage from json tutorial." https://konvajs.github.io/docs/data_and_serialization/Complex_Load.html, 2017. Geraadpleegd op 20/03/17.

[20] A. Lavrenov, "Save html5 canvas stage as json string." https://konvajs.github.io/docs/data_and_serialization/Stage_Data_URL.html, 2017. Geraadpleegd op 20/03/17.

[21] L. McNie, "Easeljs mouse interaction." http://www.createjs.com/tutorials/Mouse%20Interaction/, 2017. Geraadpleegd op 23/03/17.

[22] HTMLValidator, Peppesterest, fscholz, teoli, Jonathan_Watt, oldhill, groovecoder, ModMaker, and jsx, "Data uris - http." https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Data_URIs, 2016. Geraadpleegd op 20/03/17.

[23] L. McNie, "Easeljs v0.8.2 api documentation : Easeljs." http://www.createjs.com/docs/easeljs/modules/EaselJS.html, 2017. Geraadpleegd op 23/03/17.

[24] lannymcnie, "Svgexporter." https://github.com/CreateJS/EaselJS/tree/master/extras/SVGExporter, 2015. Geraadpleegd op 20/03/17.

[25] T. P. Group, "Image processing and gd." http://php.net/manual/en/book.image.php, 2017. Geraadpleegd op 09/05/17.

[26] D. Inc., "What is docker." https://www.docker.com/what-docker, 2017. Geraadpleegd op 16/05/17.

[27] J. Baker, "What is docker?." https://opensource.com/resources/what-docker, 2014. Geraadpleegd op 16/05/17.

[28] D. Inc., "What is a container." https://www.docker.com/what-container, 2017. Geraadpleegd op 16/05/17.

[29] rajesh, "Base64 basics - what? why? how?." http://myitlearnings.com/wp-content/uploads/2015/11/base64.jpg, 2015. Geraadpleegd op 12/05/17.

[30] R. T. Fielding and J. F. Reschke, "Hypertext transfer protocol (http/1.1): Message syntax and routing." https://tools.ietf.org/html/rfc7230#section-3.1.1, 2014. Geraadpleegd op 06/05/17.

[31] N. Foundation, "Node.js." https://nodejs.org/en/, 2017. Geraadpleegd op 06/05/17.

[32] C. Hearon, "node-canvas-prebuilt." https://github.com/chearon/node-canvas-prebuilt, 2017. Geraadpleegd op 12/05/17.

[33] Twitter, "Twitter developer documentation." https://dev.twitter.com/rest/reference, 2017. Geraadpleegd op 18/05/17.

[34] F. Inc., "Open graph page photos." https://developers.facebook.com/docs/graph-api/reference/page/photos/, 2017. Geraadpleegd op 18/05/17.

[35] F. Inc., "Page picture." https://developers.facebook.com/docs/graph-api/reference/page/picture/, 2017. Geraadpleegd op 18/05/17.

[36] J. Spencer, "2017 social media image sizes cheat sheet." https://makeawebsitehub.com/social-media-image-sizes-cheat-sheet/, 2017. Geraadpleegd op 18/05/17.

A Bijlage