2017•2018
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Ontwikkeling van een desktopapplicatie met webtechnologieën voor algemene JSON visualisatie

PROMOTOR :
dr. Kris AERTS

PROMOTOR :
ir. Kobe LEYSEN

Niels De Stickere
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE-3590 Diepenbeek
Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE-3500 Hasselt

2017 • 2018

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Ontwikkeling van een desktopapplicatie met webtechnologieën voor algemene JSON visualisatie

PROMOTOR :

dr. Kris AERTS

PROMOTOR :

ir. Kobe LEYSEN

Niels De Stickere

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

UHASSELT

KU LEUVEN

Woord vooraf

Ter afsluiting van mijn universitaire studies wens ik de mensen te bedanken die mij hebben bijgestaan in doorheen mijn studiecarrière, maar zeker ook de mensen die geholpen hebben om de masterproef tot een succesvol einde te brengen.

In de eerste plaats wens ik Ing. Kobe Leysen te bedanken voor de kans om de masterproef bij zijn bedrijf te ontwikkelen. Uit deze proef heb ik veel bijgeleerd en wil hem bedanken voor de hulp die hij mij geboden heeft doorheen de ontwikkeling van de applicatie. Daarnaast wens ik ook Prof. Dr. Kris Aerts te bedanken voor zijn hulp bij het uitwijzen van de mogelijke ontwikkelingsmogelijkheden en de hulp bij het schrijven van de thesis, abstract en onderzoeksopzet. Ik wil ook mijn dank betuigen aan de medewerkers bij Qompium die altijd klaar stonden om mij te helpen wanneer ik een vraag had en technologieën uit te wijzen die een bijdrage konden leveren aan het project.

Uiteindelijk zou ik ook mijn ouders willen bedanken die altijd achter mij stonden om mijn studies tot een succesvol einde te brengen.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Abstract

Qompium te Hasselt heeft een gespecialiseerde smartphone-applicatie ontwikkeld voor het meten en detecteren van hartritmestoornissen. Om het niet-technische personeel met de ruwe data om te laten gaan, worden tot nu toe gespecialiseerde visualisaties gemaakt. Het doel van deze masterproef is om een business intelligence tool te ontwikkelen dat eenvoudig te gebruiken is om ruwe data om te zetten in visualisaties, ongeacht technische vaardigheid. De ruwe data worden aangeleverd in de vorm van JSON-bestanden.

De applicatie moet voldoen aan vier criteria.

- In de eerste plaats moet de gebruiker delen kunnen selecteren uit de data in de JSON-bestanden die dan gevisualiseerd moet worden;
- De tool moet hiervoor een reeks toepasselijke visualisaties voorstellen. Het programma moet uitbreidbaar ontworpen zijn: Qompium wil nadien gemakkelijk extra visualisaties kunnen toevoegen;
- De userinterface moet eenvoudig bruikbaar zijn door niet-technisch onderlegde personen;
- Uiteindelijk moet er een open bestandsformaat vastgelegd worden waarin beschreven staat welke visualisaties van welke gegevens uit het JSON-bestand waar op het dashboard staan. Hiermee is het gemakkelijk om de gemaakte visualisaties tussen personen te delen.

Om de data te kunnen koppelen aan visualisaties werd vertrokken vanuit een specificatiemodel voor elke visualisatie. Dit model beschrijft welke datavormen en -combinaties geldig zijn voor een gegeven visualisatie, alsook de beperkingen die hieraan verbonden zijn. Om de userinterface eenvoudig te houden, werden de verschillende functionele onderdelen van de userinterface gegroepeerd op basis van welk onderdeel van de applicatie zij hun werkzaamheden uitvoeren. Zo bevinden alle functionaliteiten omtrent de visualisaties zich binnen dezelfde regio op het scherm. Hetzelfde geldt ook voor alles wat betreft de JSON-bestanden. Wat betreft de userinterface zijn er nog verbeteringen aan te brengen. De regio's die handelen rond JSON-bestanden, zijn permanent in beeld, waardoor er minder schermruimte overblijft om de visualisaties in te ordenen. Dit zou beter kunnen door het mogelijk te maken deze regio's te verbergen wanneer nodig. Het eindproduct werd ontwikkeld als een desktopapplicatie met webtechnologieën. Dit was belangrijk voor Qompium omdat de applicatie dan geen serverruimte in beslag neemt, terwijl de applicatie nog steeds cross-platform verspreid kan worden.

Abstract in English

Qompium, located in Hasselt, has developed a smartphone application specialized in the measuring, and detecting of heart rhythm disorders. To allow easy interpretation of raw data by non-technical personnel, the company must develop specialized visualizations for each use case. The aim of this master's thesis is to develop a business intelligence tool that is easy to use to make visualizations from raw data, regardless of technical proficiency.

The application must satisfy four criteria:

- Firstly, a user must be able to select various part from the data in a JSON file that must be visualised afterwards;
- The tool has to suggest a set of viable visualizations based on the selected data from the JSON file. The program must be designed extensibly. Qompium wants to be able to easily add new visualizations without much trouble;
- The user interface should be user friendly;
- Finally, a file format must be decided upon that describes which visualizations using which data from the JSON file should be placed where on the screen. With this, it should be simple to share the visualizations among peers.

To couple data to certain visualizations, a specification model was created for each supported visualization. This model describes which data forms and data combinations are valid for a given visualization, whilst also providing restrictions that may apply. In order to keep the user interface simple the various functional segments of the user interface were grouped based on which parts of the application they perform their duties on. For instance, every component that performs a certain action that relates to the visualisations is located in a confined region of the application screen. Similarly, every component related to JSON files is also kept in close proximity to other components that perform actions with regard to JSON files. There are still some improvements to be made with regard to the user interface. The regions on the screen that perform certain tasks concerning JSON files are permanently visible on screen. As a consequence, less room is remains for the actual visualizations. This could be improved by making it possible to hide these parts. The final product was developed as a desktop application using web technologies. This was important for Qompium as this would eliminate the need to host the application on their servers while still keeping the application cross-platform.

1 Inleiding

1.1 Situering

Qompium is de ontwikkelaar van de medische smartphone-applicatie FibriCheck dat gespecialiseerd is in het meten, detecteren en opsporen van hartritmestoornissen, meer specifiek voorkamerfibrillatie. Met behulp van deze smartphone-applicatie worden mensen gemonitord op het normaal verlopen van de hartslag en worden gebruikers gewaarschuwd wanneer voorkamerfibrillatie wordt ontdekt in de gemeten hartslag. De hartslagmetingen worden opgeslagen in de databanken van het bedrijf, waar het deze omvangrijke hoeveelheid data gebruikt om hun algoritmes te verfijnen. Het werkt hiervoor nauw samen met artsen die feedback kunnen geven over de werking van de applicatie en suggesties kunnen maken voor verbeteringen.

FibriCheck®

1. Bedrijfslogo

Qompium kiest ervoor om de verschillende datastromen te visualiseren in allerlei visualisaties, waaronder grafieken. Op die manier kunnen ook personen met weinig technisch-wetenschappelijke kennis gemakkelijk de gegevens interpreteren. Niet alleen hartslagmetingen worden zo voorgesteld, maar ook andere soorten data zoals; patiëntengegevens en bedrijfscijfers.

1.2 Probleemstelling

De dataservices van Qompium maken gebruik van een volledige JSON REST API. Het JSON-dataformaat is al een pak makkelijker dan XML, maar blijft relatief moeilijk te interpreteren door het niet-technische personeel. Daarom heeft Fibricheck visualisaties gebouwd. Deze visualisaties zijn specifiek per datagroep. Omdat de bestaande implementaties maar in zeer beperkte mate herbruikbaar zijn, moet men voor elke nieuwe activiteit of gegevensvorm nieuwe gespecialiseerde visualisaties maken.

Dit proces leidt tot productiviteitsverliezen bij zowel het niet-technische als het technische personeel. Het technische personeel investeert tijd en middelen in het ontwikkelen van de visualisaties terwijl het niet-technische personeel op verminderde capaciteit draait zolang de nieuwe visualisatie niet beschikbaar is.

1.3 Doelstellingen

Het doel van dit onderzoek is om een flexibel systeem te ontwikkelen dat toelaat om op een eenvoudige manier visualisaties aan te maken en deze visualisaties te kunnen verdelen onder het personeel. Op deze manier wil men de productiviteitsverliezen met het ontwikkelen van gespecialiseerde, maar inflexibele, visualisaties minimaliseren. Het is hierbij de bedoeling dat het eindproduct eender welke JSON kan aanvaarden en dat de gebruiker kan kiezen welke delen van de JSON-data gevisualiseerd moeten worden.

Gedurende het onderzoek is het belangrijk om een systeem te ontwikkelen dat verschillende datavormen kan koppelen aan de gepaste visualisaties, waaruit de gebruiker van het product kan kiezen. Hiervoor is het belangrijk te onderzoeken welk type systeem hiervoor toepasselijk is. Daarnaast moet dit systeem flexibel genoeg zijn om naderhand addities aan te maken.

Hierbij komt ook een userinterface kijken die op een intuïtieve manier gebruikers met uiteenlopende technische achtergrond visualisaties laat maken. Deze visualisaties moeten gemanipuleerd kunnen worden om een handig dashboard te maken.

Fibricheck kiest voor al zijn producten consistent voor webtechnologieën. Voor dit project ging de voorkeur uit naar React en Electron. Om toch ook offline te kunnen werken, is er voor deze applicatie gekozen om met behulp van de hierboven vermeldde technologieën een desktopapplicatie te bouwen die in een Chromium browser draait.

1.4 Methode

In eerste instantie werd het JSON-dataformaat onderzocht. Hierbij werd uitgekeken naar de verschillende karakteristieken die deze datadrager heeft. Een goede kennis hiervan was nodig omdat voor het eindproduct JSON's gevisualiseerd moeten worden. Deze visualisering van een JSON moet daarnaast ook interactief zijn.

Hierna konden de verschillende technologieën bestudeerd worden. Naast JavaScript, HTML en CSS die de bouwstenen vormen voor webapplicaties waren ook React, Electron en TypeScript noodzakelijkheden die vanaf het begin van het project werden vastgelegd. Toen een zekere basiskennis van de eerder vermeldde technologieën was verworven, kon begonnen worden aan de ontwikkeling van de applicatie. Op dit punt werd voorgesteld om met Visual Studio Code als editor te werken. Dit is ook de editor waar het technisch personeel van Qompium gebruik van maakt om websites en webapplicaties te ontwikkelen.

Bij het starten van de ontwikkeling van de applicatie werd eerst geopteerd om React en Electron te combineren in een enkel project. Dit verliep niet zo vlot, maar werd toch succesvol in een project samengevoegd. Doordat de kennis van React, Electron en Visual Studio Code op dit punt nog oppervlakkig was, was de ontwikkeling van het project redelijk moeizaam. De beslissing werd dan gemaakt om het project terug op te splitsen in een Electron-project en een React-project. Deze beslissing werd genomen om de focus te kunnen leggen op de applicatieontwikkeling in React en later terug te komen op het samensmelten van het React-project en Electron. Van hieruit werden verdere ontwikkelingen in de Chromebrowser getest omdat deze, net als Electron, gebouwd is op de Chromium browser.

Gelijklopend hieraan, werd ook de loginpagina ontwikkeld. Op dit punt was er nog geen sprake van JSON-bestanden inlezen van op de computer. De applicatie zou enkel JSON-bestanden inlezen van de services van Qompium. Na het splitsen van React en Electron in onafhankelijke projecten kwam een nieuw probleem naar boven. Het React-gedeelte van de applicatie kon geen berichten sturen naar de services van Qompium om JSON's op te vragen. Een van de externe werknemers van Qompium heeft toen geholpen door een Chromeversie op de computer te plaatsen die berichten van en naar de services van Qompium niet blokkeert.

Hierna kon de ontwikkeling van de applicatie wat sneller verlopen. Nadat de loginpagina volledig was, werd een inputveld aangebracht dat JSON-gegevens kon opvragen van de Qompiumservices. Dit was eenvoudig doordat de loginpagina het meeste van de nodige code al bezat. Pas daarna werd een component ontwikkeld die ontvangen JSON's kon omzetten in een visuele weergave van de JSON. Deze weergave zou de structuur van de ontvangen JSON nabootsen. Nadat de JSON-visualisatie interactief gemaakt was, werd werk gemaakt van het drag- en dropsysteem. Qompium stond op het gebruik van drag- en drop als intuïtieve manier om JSON-gegevens over te brengen naar het veld waar de effectieve visualisaties gemaakt worden. De component voor JSON-visualisatie werd doorheen het verloop van de ontwikkeling van de applicatie nog een aantal keren vernieuwd om sneller en/of gebruiksvriendelijker te zijn.

Toen deze systemen functioneel waren, kon begonnen worden aan de visualisaties. Gelijklopend aan elkaar waren er enkele ontwikkelingen die plaatsvonden:

- Aan de verschillende datavormen die versleept kunnen worden, werd bestudeerd welke visualisaties daarvoor toepasselijk zouden zijn;
- Er werd ook uitgekeken naar een grafiek-bibliotheek die eenvoudig in gebruik is, maar toch een voldoende aantal grafieken aanbiedt. Hierbij werd ChartJS gekozen. Deze bibliotheek biedt acht voorgemaakte grafieken. Daarnaast is het ook mogelijk om deze grafieken te personaliseren en te muteren – binnen zekere grenzen – naar complexere grafieken;
- Een tabelcomponent werd ontwikkeld die zou gebruikt zou kunnen worden voor alle data die versleept kan worden. Dit werd besloten omdat een tabel zowel enkelvoudige data, zoals een string, kan voorstellen, maar ook complexere data.

Daarna kon aan de hand van de studie van de datavormen die voor verschillende visualisaties geldig zijn een systeem gemaakt worden dat de versleepte data inspecteert. Uit deze inspectie komt een lijst van visualisaties terug die de versleepte data kunnen aanvaarden. De eerste vorm van dit systeem was rigide. Voor verschillende scenario's kwamen dan verschillende voorgedefinieerde lijsten van visualisaties terug. Hierbij werd dan ook een ID toegevoegd om later in het proces te kunnen achterhalen aan welk scenario de data gekoppeld was. Deze scenario's waren afhankelijk van bepaalde karakteristieken van de versleepte data. Uit de lijst van visualisaties kon de gebruiker dan een visualisatie kiezen. Aan de hand van het scenario ID werd bepaald op welke manier de data moest worden verwerkt opdat het bruikbaar was voor de gekozen weergave.

De volgende stap zou zijn om na het aanmaken van een visualisatie deze te laten updaten. Dit updaten kon betekenen dat het type van visualisatie moest veranderen of dat de data die de visualisatie moet voorstellen moest veranderen. Op dit punt werd het duidelijk dat dit niet zo eenvoudig zou zijn in React alleen zonder een onoverzichtelijke keten van callback-functies te creëren. Om nog niet te spreken over het opslaan en inladen van verschillende visualisaties. Om dit probleem op te lossen, suggereerde een van de werknemers bij Qompium om gebruik te maken van Redux. Met behulp van deze bibliotheek zou het mogelijk zijn om buiten de React-applicatie gegevens bij te houden. Dit was de oplossing die callback-keten probleem kon vermijden. De applicatie werd herwerkt om gebruik te maken van Redux, waarna de eerder vermeldde features werden ingewerkt. Kort hierna werd de keuze gemaakt om een React-bibliotheek toe te voegen die de esthetische opmaak van de applicatie uit handen kon nemen. Tot dan toe werd dit grotendeels zelf gemaakt met wisselend resultaat. Er werd voor Material UI gekozen te ondersteuning in het maken van de opmaak van de applicatie. Hierna werd de opmaak van de applicatie waar mogelijk herwerkt om Material-UI-componenten te gebruiken.

Op dit punt in het ontwikkelingsproces waren de kernfunctionaliteiten van het React-gedeelte van de applicatie aanwezig. Nu was er nog de kwestie van de I/O tussen de applicatie en de computer. Hiervoor was Electron nodig. De React- en Electron-projecten bleven gescheiden tijdens het ontwikkelen van de I/O-functionaliteiten. Testen van de React-applicatie in Electron werd uitgevoerd door de applicatie te 'compileren' en deze gecompileerde bestanden over te zetten naar het Electron-project via een commandolijn script. Pas wanneer het omzetten van het Electron-project naar een uitvoerbaar bestand aan bod kwam, werd Electron terug toegevoegd aan het React-project. Op dit punt was de kennis van Visual Studio Code, Electron en React voldoende groot om dit met kennis van zake te doen. Het Electron-project dat overbleef werd gebruikt om het omzetten naar een uitvoerbaar bestand te testen. Hiervoor werd de Electron-build-bibliotheek gebruikt. Toen dit goed geconfigureerd was, werd dit overgebracht naar het React-Electron-project. Tijdens dit proces werd ook vastgesteld dat Acer computers het correct opstarten en afsluiten van Electron-instanties verhindert. Hierdoor blijven 'zombieprocessen' bestaan op de achtergrond van de computer. Hetgeen veroorzaakt wordt door het ePowerEvent-programma van Acer. Door dit programma af te sluiten, werd het maken van zombieprocessen opgelost. Of andere computers gelijkaardige belemmeringen hebben kon niet worden vastgesteld.

De werknemers van Qompium hadden nog enkele toevoegingen die ze graag hadden gezien in de toenmalige toestand van de applicatie. Voor hen was het wenselijk om lokale JSON-bestanden te kunnen inlezen in de applicatie, instellen van de x- en y-assen en het in- en uitzoomen van lijngrafieken. Het inlezen van bestanden was een functionaliteit die al bestond in de applicatie en was snel toegevoegd. Het zoomen van lijngrafieken was daarentegen een groter obstakel dan aanvankelijk gedacht. ChartJS had blijkbaar geen opties voor zoomen. Er werd een bibliotheek gevonden die deze functionaliteit toevoegt aan ChartJS-componenten, maar bleek te traag te zoomen over grotere datasets. Deze datasets kunnen tot 15000 datapunten groot zijn. Een werknemer van Qompium gaf enkele suggesties die het exact type zoomfunctionaliteit had die gewenst was. Uit deze suggesties werd na testen van elk van deze suggesties was maar één snel genoeg. Dit was de DYGraph-bibliotheek. Andere bibliotheken die werden gesuggereerd, naast eigen opzoekingen, konden voldoende snel overweg met de grote datasets. Decimatie van de dataset was geen optie voor Qompium. Het instellen van de x- en y-assen van visualisaties is niet meer ingewerkt geraakt.

De visualisaties die werden gecreëerd in de applicatie werden als een verticale lijst gegenereerd in de applicatie. Elke visualisatie had daarbij nog eens een vaste omvang. Deze manier van visualisaties ordenen in de applicatie was aangebracht als een tijdelijke oplossing. In dit stadium van de ontwikkeling werd dit systeem vervangen. Het nieuwe systeem zou ervoor moeten zorgen dat gebruikers zelf hun visualisaties kunnen verdelen over het scherm. Daarnaast zou het voor hen ook mogelijk moeten zijn om zelf te bepalen hoe groot deze visualisaties moeten zijn. Hiervoor werden de React-Reresizable- en React-Draggable-bibliotheken toegevoegd en geïmplementeerd in het project.

Als laatste werd nog een kritische blik geworpen naar de manier waarop de verschillende visualisaties, benodigdheden en beperkingen in de applicatie gestructureerd waren. Het bestaande systeem was te

gecentraliseerd en inflexibel om eenvoudig nieuwe toevoegingen te maken. Er werd een nieuw systeem bedacht waarbij alle karakteristieken en benodigdheden van visualisaties apart voor elke visualisatie kan vastgelegd worden. Voor elke visualisatie kon zo een specificatiemodel worden vastgelegd. Hierbij moest ook een datatype systeem bedacht worden dat versleepte JSON-gegevens kan catalogeren. Deze datatypes worden gebruikt in het specificatiemodel om aan te geven welke datavormen acceptabel zijn voor elke visualisatie. Alle gegevens omtrent de verschillende visualisaties die gebruikt kunnen worden in de applicatie, konden dan samengevoegd worden. Met behulp van dit nieuwe systeem was het flexibiliteitscriterium beter beantwoord dan het voormalige systeem.

2 De applicatie

In dit onderdeel worden de verschillende aspecten die kwamen kijken bij het maken van de applicatie aan bod. Hier zal niet alleen de verzamelde kennis worden besproken die nodig was om de applicatie te kunnen ontwikkelen, maar ook de designkeuzes en code die werden gebruikt in de applicatie. De volledige code is eigendom van Qompium, maar kan indien gewenst bij hen worden opgevraagd.

2.1 Vooronderzoek

In dit hoofdstuk worden de verschillende onderdelen die werden gebruikt om de applicatie te maken besproken. Deze zijn opgedeeld in volgende thema's:

- JSON;
- Soortgelijke projecten;
- Software editor;
- Electron;
- Programmeertalen;
- Softwarebibliotheken;
- Grafiekbibliotheken;
- Andere bibliotheken.

2.1.1 JSON

In deze paragraaf wordt de JSON-standaard (*JavaScript Object Notation*) besproken. Deze standaard speelt een centrale rol in de applicatie die ontwikkeld werd in functie van deze thesis. Het is namelijk deze standaard die de houder is van alle data die aan de applicatie wordt aangeboden. Hieruit worden verdere visualisaties afgeleid. Er wordt besproken hoe de JSON-standaard opgebouwd is, hoe de JSON-standaard vergelijkt met XML[1] en het welk belang de JSON-standaard heeft in de IT-sector.

2.1.1.1 Opbouw

De manier waarop JSON wordt opgebouwd vloeit voort uit de JavaScript programmeertaal. Deze taal voorziet een datatype dat 'object' genoemd wordt. Deze objecten zijn gebaseerd op sleutel-waarde-paren. JavaScript heeft twee typen objecten; gewone objecten en lijsten. Zonder in te veel detail te treden, verschillende objecten enkel met wat de sleutels zijn en welke symbolen als afbakening worden gebruikt. Bij gewone objecten mogen de sleutels *strings* of getallen zijn zonder dat een volgorde wordt afgedwongen, en bestaat de afbakening uit accolades. Bij lijsten zijn de sleutels oplopende getallen, beginnende met het getal nul, en wordt er afgebakend met rechte haken.

Het startpunt van elke JSON is steeds een object, zijnde een gewoon object of een lijst. De inhoud van een object is niet gebonden aan één bepaald datatype. Een gevolg hiervan is dat een JSON-object meerdere datatypen kan bevatten, zonder dat dit voor conflicten zorgt. De datatypes die door de JSON-standaard worden ondersteund zijn de volgende: strings, getallen, objecten, lijsten, *null*, *undefined*. Hieruit is ook af te leiden dat een object zelf weer een object kan bezitten, wat *nesting* wordt genoemd. Dankzij nesting van objecten is het mogelijk om meerdimensionale datastructuren te bouwen. Een voorbeeld van nesting wordt geïllustreerd in figuur 2. [1]

[1] Extensible Markup Language. XML is een opmaaktaal dat het mogelijk maakt om op een gestructureerde manier gegevens op te slaan.

```
[
    {
        "SalesOrderID":43663,
        "Status":5,
        "PurchaseOrderNumber":"PO18009186470",
        "ShipDate":"2011-06-07T00:00:00",
        "P":[
            {"ProductID":760}
            ]
    },
    {
        "SalesOrderID":43687,
        "Status":5,
        "PurchaseOrderNumber":"PO4959110829",
        "ShipDate":"2011-06-07T00:00:00",
        "P":[
            {"ProductID":768},
            {"ProductID":765}
            ]
    }
]
```

2. Een geneste JSON [2]

De JSON-standaard werd oorspronkelijk ontwikkeld om het probleem van de *cross-platform*[2] communicatie, communicatie tussen verschillende programma's of systemen zonder onafhankelijk van de gebruikte programmeertalen, aan te pakken. De standaard is dus niet gebonden aan JavaScript, waar deze zijn oorsprong kent, maar is bruikbaar vanuit andere programmeertalen. De manier waarop de standaard wordt geïmplementeerd in een bepaalde programmeertaal is niet belangrijk, zolang de eigenschappen van de standaard bewaard blijven.

Deze programmeertaal onafhankelijkheid is een van de redenen dat JSON een belangrijke standaard is geworden. Een andere eigenschap die net zo zeer heeft bijgedragen, is dat de verhouding van opvullende symbolen ten opzichte van effectieve data relatief klein is. De enige symbolen die worden gebruikt zijn: accolades, rechte haken, afkappingstekens en komma's. De beperkte opvulling maakt deze standaard goed leesbaar voor de mens.

Met deze compacte manier om data te structureren, maakt JSON het onderscheid met andere bekende soortgelijke projecten zoals XML (*Extensible Markup Language*). XML leunt sterk aan bij de syntax van HMTL. Net als het geval is bij HTML, wordt er bij XML ook gewerkt met *tags* die data omsluiten. Aan deze tags kunnen ook attributen worden gekoppeld. Net als bij JSON, wordt er gewerkt met het nesten van elementen om hiërarchie te creëren.

De verschillen en gelijkenissen tussen XML en JSON zijn beter zichtbaar in figuur 3. In deze figuur is dezelfde informatie als uit figuur 2 is te zien, maar dan in XML-formaat. De JSON-data werd omgezet door een online vertaler van JSON naar XML. De XML-weergave is niet noodzakelijk de meest optimale manier waarop de data kan weergegeven worden in XML, maar voldoet als illustratie van de verschillen en gelijkenissen. In de XML-variant wordt direct informatie meegegeven over welke taal wordt gebruikt, welke versie van deze taal en de codering werd toegepast op het document. Bij JSON-documenten wordt deze informatie niet expliciet meegegeven. Bij XML is wel sprake van meer overbodige informatie doordat elke open-tag gepaard moet gaan met een sluit-tag. Hierbij moet de naam van het element herhaald worden. Zoals al eens werd aangehaald, gebruiken XML en JSON dezelfde manier om data te structureren. Beide moeten één hoofdelement hebben waaruit data genest kan worden.

[2] Overheen verschillende platformen, zoals Windows of MacOS, of programmeertalen.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <element>
      <P>
         <element>
            <ProductID>760</ProductID>
         </element>
      </P>
      <PurchaseOrderNumber>PO18009186470</PurchaseOrderNumber>
      <SalesOrderID>43663</SalesOrderID>
      <ShipDate>2011-06-07T00:00:00</ShipDate>
      <Status>5</Status>
   </element>
   <element>
      <P>
         <element>
            <ProductID>768</ProductID>
         </element>
         <element>
            <ProductID>765</ProductID>
         </element>
      </P>
      <PurchaseOrderNumber>PO4959110829</PurchaseOrderNumber>
      <SalesOrderID>43687</SalesOrderID>
      <ShipDate>2011-06-07T00:00:00</ShipDate>
      <Status>5</Status>
   </element>
</root>
```

3. XML-weergave

Waar XML het voordeel haalt ten opzichte van JSON is dat het ondersteuning voor validatie van documenten ingebouwd is in de taal zelf. Hiervoor kunnen DTD's (*Document Type Definition*) worden aangemaakt. Deze DTD's bevatten de regels die bepalen waar en wanneer een bepaald XML-element van toepassing is in een XML-document. Om dezelfde functionaliteit te voorzien bij JSON moet gebruik gemaakt worden van derde-partij software zoals JSON Schema die het mogelijk maken om gelijkvormigheid te testen. [1, 3, 4, 5, 6]

2.1.2 Soortgelijke projecten

Het hoofddoel van deze thesis is het maken van een *business intelligence*[3] (BI) desktopapplicatie dat zich richt op het gebruiksvriendelijk en intuïtief omzetten van JSON-data in grafische weergaven die als een flexibel *dashboard*[4] kunnen dienen. BI duidt op het verzamelen, visualiseren, rapporteren, enz. van data met betrekking tot de bedrijfsactiviteit. Deze paragraaf richt zich op het toelichten van alternatieve applicaties, die onder meer JSON ondersteunen, en het onderscheid maken tussen het thesisproject en deze andere applicaties. Hierbij wordt enkel gekeken naar de omgang met JSON-data. [7]

[3] "Verzameling van strategische bedrijfsinformatie met behulp van rapportagetools en datamining." [86]
[4] Een collectie van grafische weergaven gekoppeld aan databronnen. Deze weergaven worden actief bijgewerkt en kunnen worden inbegrepen in managementrapportages en webpagina's. Ze dienen als ondersteuning om data van allerlei bronnen eenvoudig en flexibel weer te geven. [87]

2.1.2.1 Tableau

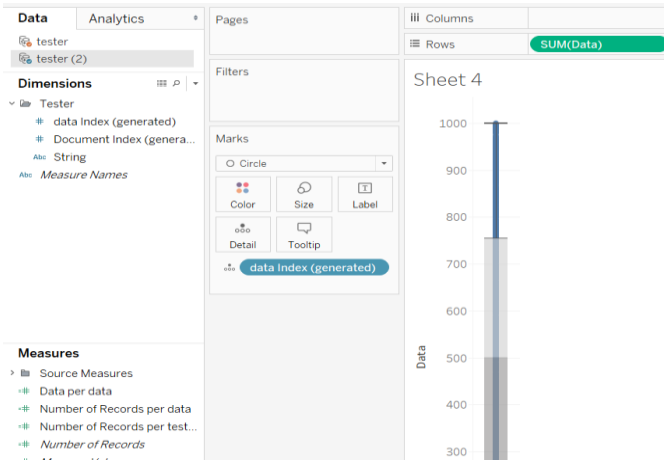
Tableau is een gekend en uitgebreide applicatie die, aan de hand van een eenvoudige *drag-and-drop*[5] *interface*[6], de gebruiker in staat stelt om dashboards op te stellen die data van verschillende bronnen samenvoegt tot een geheel. Deze applicatie ondersteunt ook een groot aantal databronnen, zowel lokaal op de computer als op afzonderlijke databanken of servers, waaronder ook JSON. De applicatie biedt ook nog een uitgebreid aantal visualisaties en personalisatie opties voor deze visualisaties. Deze eigenschappen zijn grotendeels kenmerkend aan dit type applicaties.

Bij het inladen van data wordt deze behandeld alsof het een MS Access tabel is. Hierbij is het dus mogelijk om *queries*[7] op stellen die data uit de bron halen en doorgeven naar het dashboard, maar ook handmatig zijn rijen of kolommen uit te halen. Dit scherm wordt weergegeven in figuur 4. Een nadeel is wel dat enkel één tabel per keer kan worden doorgegeven aan de dashboard bouwer. Om meerdere tabellen in te laden, moeten dezelfde stappen enkele keren worden herhaald. Ook vergelijkbaar met MS Access is het feit dat er relaties kunnen worden opgesteld tussen ingeladen tabellen, zoals geïllustreerd in figuur 5. Deze relaties kunnen gebruikt worden om filtering van één tabel of dataset ook door te voeren op de data uit tabellen die daaraan gekoppeld zijn.

Sort fields | Data source order

tester.json Document Ind...	tester.json data Index (ge...	tester.json Data	tester.json String	tester.js Data p
1	1	725.576	Op een dag in het...	*Undef*
1	2	549.827	Op een dag in het...	*Undef*
1	3	981.750	Op een dag in het...	*Undef*
1	4	962.879	Op een dag in het...	*Undef*
1	5	28.077	Op een dag in het...	*Undef*
1	6	853.267	Op een dag in het...	*Undef*
1	7	882.216	Op een dag in het...	*Undef*
1	8	841.062	Op een dag in het...	*Undef*
1	9	551.229	Op een dag in het...	*Undef*

4. Ingeladen data

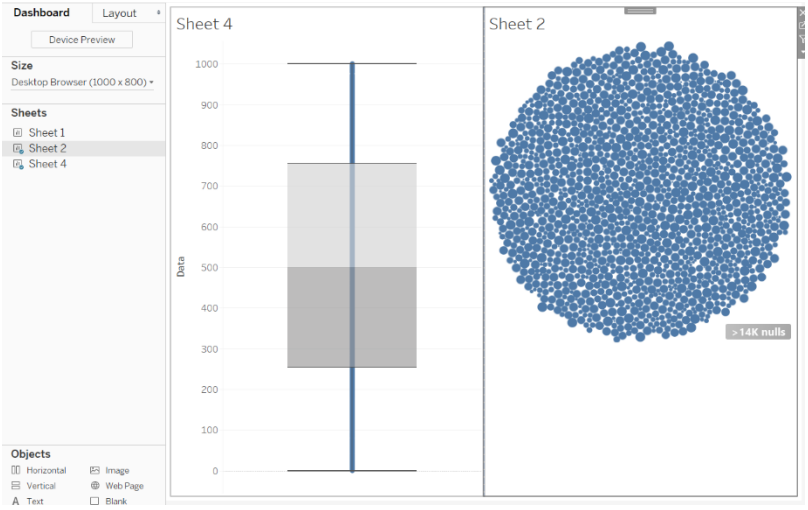


5. Aanmaak van een visualisatie

Alle functies en data zijn verzameld in hetzelfde venster waar de visualisaties kunnen worden opgemaakt. Dit kan door drag-and-drop of aanvinken van de data die in een weergave moet worden voorgesteld. Verdere

[5] Een functionaliteit die toestaat elementen uit een interface te verslepen en op een andere locatie in dezelfde of een andere interface te laten neervallen. Hierdoor wordt het element of de data uit dat element overgedragen tussen locaties in dezelfde interface of over interfaces heen.
[6] Een laag die helpt bij de communicatie tussen twee entiteiten. Dit kan zijn tussen mens en machine, maar ook tussen machines onderling.
[7] Zoekopdrachten die onderdelen van grotere gehelen terug te geven. In verband met software tabellen kunnen deze zoekopdrachten gebruikt worden om onderdelen aan elkaar te koppelen en bewerkingen uit te voeren op de data die er in aanwezig is.

personalisatie van visualisaties kan dan in een apart onderdeel van het venster waar filteren, kleuren, definiëren van x- en y-as, enz. mogelijk is. Voor elke mogelijke combinatie van data is elke visualisatie beschikbaar, maar als de data niet kan worden weergeven door de opgegeven visualisatie wordt ook geen visualisatie teruggeven, weergegeven in figuur 6. [8]

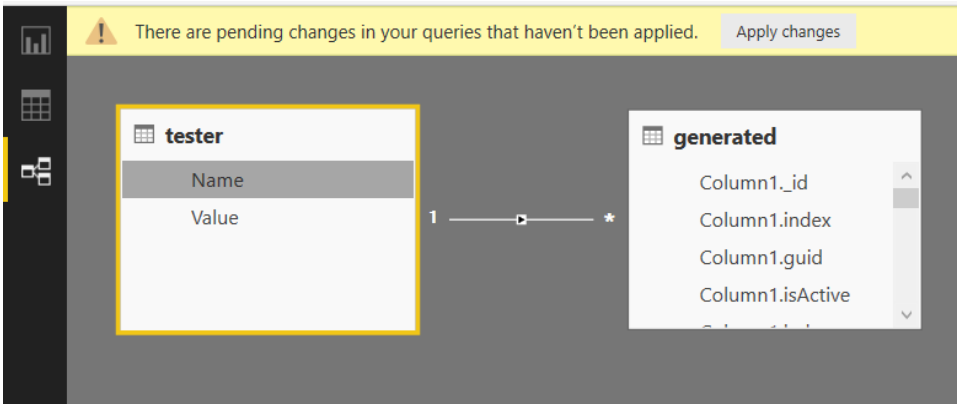


6. Visualisaties in het dashboard

2.1.2.2 Microsoft Power BI

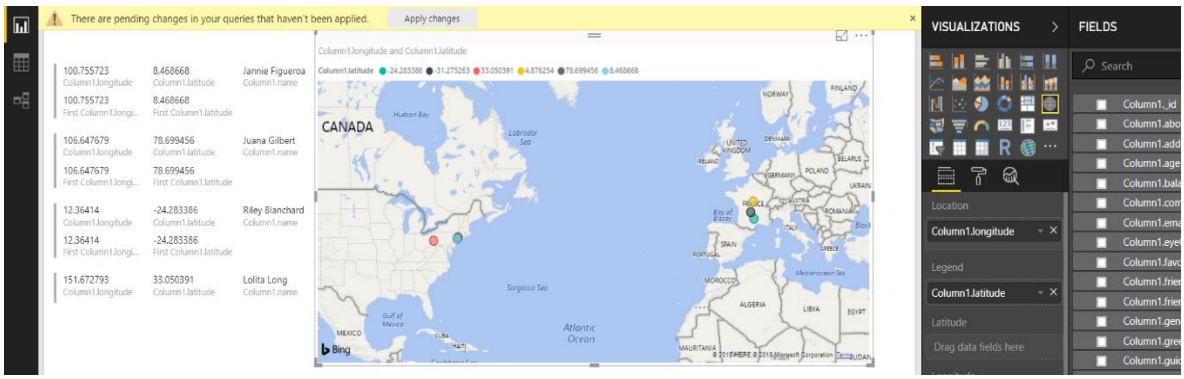
Buiten standaard eigenschappen zoals uitgebreide ondersteuning van databronnen en visualisaties, hanteert de Microsoft Power BI applicatie een andere aanpak voor het ophalen van informatie uit een bron en de manier waarop het dashboard kan worden opgebouwd.

Bij het inladen van data wordt deze behandeld alsof het een MS Access tabel is. Hierbij is het dus mogelijk om *queries* op stellen die data uit de bron halen en doorgeven naar het dashboard, maar ook handmatig rijen of kolommen op te halen. Het is wel zo dat enkel één tabel per keer kan worden doorgegeven aan de dashboard bouwer. Om meerdere tabellen in te laden, moeten dezelfde stappen enkele keren worden herhaald. Ook vergelijkbaar met MS Access is het feit dat er relaties kunnen worden opgesteld tussen ingeladen tabellen, zoals geïllustreerd in figuur 7. Deze relaties kunnen gebruikt worden om filtering van één tabel of dataset ook toe te passen op de data uit tabellen die daaraan gekoppeld zijn.



7. Relaties tussen tabellen

Alle functies en data zijn verzameld in hetzelfde venster, waar de visualisaties kunnen worden opgemaakt. Dit kan door drag-and-drop of aanvinken van de data die in een weergave moet worden voorgesteld. Verdere personalisatie van visualisaties kan dan in een apart onderdeel van het venster waar filteren, kleuren, definiëren van x- en y-as, enz. mogelijk is. Voor elke mogelijke combinatie van data is elke visualisatie beschikbaar, maar als de data niet kan worden weergeven door de opgegeven visualisatie wordt ook geen visualisatie teruggeven, weergegeven in figuur 8. [9]

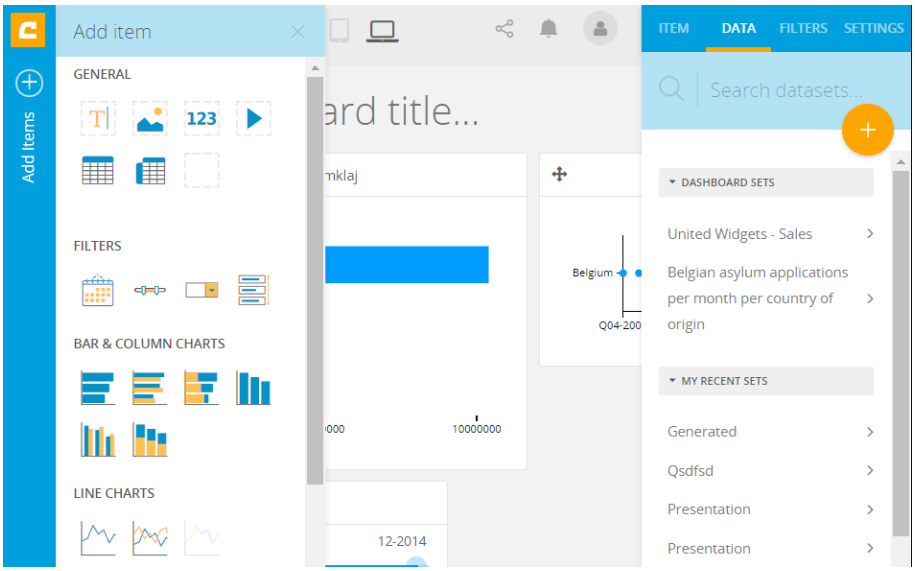


8. Power BI-dashboard

2.1.2.3 Cumul.io

Cumul is ook een BI-applicatie die niet wordt aangeboden als desktopapplicatie, maar volledig op het web te gebruiken valt. Het inladen en visualiseren van data in het dashboard is volledig beschikbaar vanuit hetzelfde venster. Om data in de applicatie te laden kan gebruik gemaakt worden van externe bronnen zoals databases of van lokale documenten.

Om het dashboard op te bouwen moet eerst een weergave worden gekozen, waarna de gewenste data versleept kan worden naar de weergave. Het is evenzeer mogelijk om elementen, zoals bijvoorbeeld een schuifbalk, aan visualisaties te koppelen om interactieve filtering toe te staan. De ondersteuning van Cumul voor het JSON-dataformaat beperkt zich wel tot het GeoJSON-dataformaat[8] en TopoJSON-dataformaat[9]. Op andere JSONs mogen enkel lijsten van lijsten of lijsten van objecten zijn. De opbouw van de site wordt weergegeven in figuur 9. [10, 11]



9. Cumul.io dashboard

2.1.2.4 Onderscheid

Ondanks dat de applicatie die werd ontwikkeld in functie van de thesis een vergelijkbare functie heeft ten opzichte van de drie besproken BI-applicaties, zijn er toch drie punten waarop het thesisproject zich onderscheidt van de andere applicaties:

Om te beginnen beperkt het thesisproject zich enkel en alleen tot de ondersteuning en visualisatie van JSON-data. Een gevolg hiervan is dat alle data die in de applicatie wordt ingeladen wordt getoond aan de gebruiker als een JSON. De andere applicaties daarentegen zijn veel breder toepasbaar. Zij ondersteunen een breed gamma aan informatiebronnen, zowel lokaal als extern. Maar JSON-data wordt niet in de vorm van een JSON aan de

[8] Een JSON-dataformaat dat met als specifiek doel om geologische data op een gestandaardiseerde manier bij te houden.
[9] Een JSON-dataformaat om topologische data op een gestandaardiseerde manier bij te houden.

gebruiker gepresenteerd. In het geval van Tableau en Microsoft Power BI wordt deze omgevormd in een tabelweergave. Cumul daarentegen legt beperkingen op de vorm die een JSON mag aannemen. Dit is waarschijnlijk omdat de applicatie van Cumul intern ook uitgaat van tabellen.

Het tweede onderscheid is dat een van de doelstellingen van het thesisproject is om de gebruiker bij te staan door suggesties te geven. Deze suggesties zijn een lijst van soortgelijke datapunten die werden teruggevonden uit de JSON-data die de gebruiker mogelijks ook zou willen visualiseren. Dit soort van functionaliteit werd niet teruggevonden bij de andere applicaties. Het gebrek van dit soort functionaliteit bij de andere applicaties kan te wijten zijn aan het feit dat alle data binnen deze applicaties tweedimensionaal zijn of worden gemaakt, waardoor een dergelijke functionaliteit niet nodig is.

Het laatste verschil tussen het thesisproject en de andere applicaties bevindt zich bij de manier waarop het dashboard kan gebruikt worden. Het is de doelstelling van het thesisproject om een gecreëerd dashboard te kunnen gebruiken op elke JSON die in de applicatie wordt ingeladen. Op deze manier is een dashboard bruikbaar voor meerdere bronnen, maar kan enkel gebruikt worden voor één databron tegelijk. Dit is anders dan de dashboards van de applicaties. Daar kunnen dashboards worden opgesteld aan de hand van meerdere bronnen, maar zijn enkel geldig voor die specifieke databronnen. Voor andere databronnen moeten ofwel nieuwe dashboards worden aangemaakt of manueel de databronnen van de visualisaties aanpassen.

2.1.3 Software editor

Voor de ontwikkeling van dit project werd gebruik gemaakt van Visual Studie Code (VS Code) als *editor*[10]. In deze paragraaf wordt kort toegelicht welke eigenschappen VS Code aantrekkelijk maken voor dit project, hoe deze vergelijkt met andere editors zoals: Atom, Sublime Text en Brackets, die allemaal redelijk hoog aangeschreven staan. Ook wordt de vergelijking gemaakt met Visual Studio. Uiteindelijk wordt dan de conclusie getrokken uit deze vergelijkingen. [12, 13, 14]

2.1.3.1 Visual Studio Code

VS Code is een *source-code editor* die werd ontwikkeld en wordt uitgegeven door Microsoft. Deze editor is eveneens *open-source*[11]. Deze editor werd ontworpen met aandacht voor de webontwikkeling wereld en biedt standaard ondersteuning aan voor onder andere: Node.js, JavaScript, TypeScript, React, HTML, CSS en JSON. Om in andere programmeertalen in deze editor te kunnen werken moeten *plugins*[12] worden gedownload. VS Code werd ontwikkeld met behulp van het Electron *framework*[13] en andere webtechnologieën zoals: JavaScript en Node.js. Hierdoor is deze editor cross-platform.

Buiten code schrijven, aanpassen, verwijderen en suggesties te krijgen tijdens het typen, zijn er ook andere functionaliteiten die standaard worden aangeboden door deze editor. Zo worden standaard de typen van variabelen en functies nagekeken om te kijken of deze overal wel correct zijn, zolang dit van toepassing is, maar is het ook mogelijk om applicaties direct vanuit de editor te bouwen en te debuggen. Los van de geschreven code biedt VS Code ook Git[14] ondersteuning en een ingebouwde terminal aan. [15]

2.1.3.2 VS Code vergeleken met Visual Studio

Om te beginnen spelen VS Code en Visual Studio niet echt in op dezelfde markt. Visual Studio is een IDE (*Integrated Development Environment*) en bevat alle functionaliteiten die een softwareontwikkelaar kan wensen om een project te ontwikkelen. Standaard ondersteund deze IDE volgende technologieën: C++, Node.js, Python, R, .NET, JavaScript en TypeScript. Ook is er ondersteuning voor het maken van mobiele applicaties, databaseconnecties en meer. VS Code richt zich meer op snelle ontwikkeling van applicaties en laat complexere functionaliteiten achterwege. Deze editor werd in eerste instantie ook ontworpen voor webontwikkeling. Hierdoor komt VS Code standaard uitgerust met ondersteuning voor een waaier aan webtechnologieën. Dat de

[10] Een applicatie dat toestaat om code te schrijven, wijzigen en opslaan.
[11] Software dat open staat voor elke persoon of organisatie om de code te bekijken, aanpassingen te maken of uitbreidingen toe te voegen.
[12] Een software module dat kan toegevoegd worden om de functionaliteit van applicaties uit te breiden.
[13] Een vast of conceptueel platform dat gemeenschappelijke code en algemene functionaliteit aanbiedt. Deze kunnen worden gespecialiseerd of overschreven worden door ontwikkelaars. Een framework is een collectie van bibliotheken die toegankelijk worden gemaakt aan de ontwikkelaar via een interface.
[14] Een versie controlesysteem. Hiermee kunnen ontwikkelaars de verschillende ontwikkelingsiteraties en aftakkingen van softwareprojecten bij houden.

beide programmeeromgevingen zich toespitsen op verschillende markten laat zich ook merken in de *hardware* vereisten die gesteld worden.

Als enkel naar de technologieën die beide ontwikkelomgevingen aanbieden wordt gekeken, dan is het niet mogelijk om een duidelijke lijn te trekken. Beide bieden namelijk ondersteuning aan voor technologieën die gebruikt worden in het thesisproject, namelijk: JavaScript, TypeScript en Node.js. Het enige verschil op dit vlak is terug te vinden bij de ondersteuning voor React en JSON. React, een van de vereiste technologieën voor dit project, is een JavaScript bibliotheek en kan dus ook gebruikt worden in Visual Studio. Hetzelfde kan worden gezegd over JSON, welke zijn naam dankt aan de JavaScript-programmeertaal.

Tot nu toe zijn beide ontwikkelomgevingen evenwaardig aan elkaar. Daarom wordt een blik geworpen op de verschillen tussen de minimale *hardware* vereisten die aan beide applicaties worden gesteld. Deze vergelijking is terug te vinden in tabel 1 en worden vergeleken op ROM-geheugen, RAM-geheugen en processorsnelheid vereisten. De punten waar het onderscheid meer dan duidelijk is, zijn de benodigdheden voor ROM- en RAM-geheugen. VS Code heeft een tiende tot een twintigste minder ROM-geheugen nodig dan Visual Studio. Voor RAM-geheugen is dit een bescheidenere 50% minder. Op het vlak van de minimaal benodigde processorsnelheid valt er bijna geen verschil op te maken.

	Visual Studio Code	Visual Studio
ROM	200 MB	20 – 50 GB
RAM	1 GB	2 GB
Processor	1,6 GHz	1,8 GHz

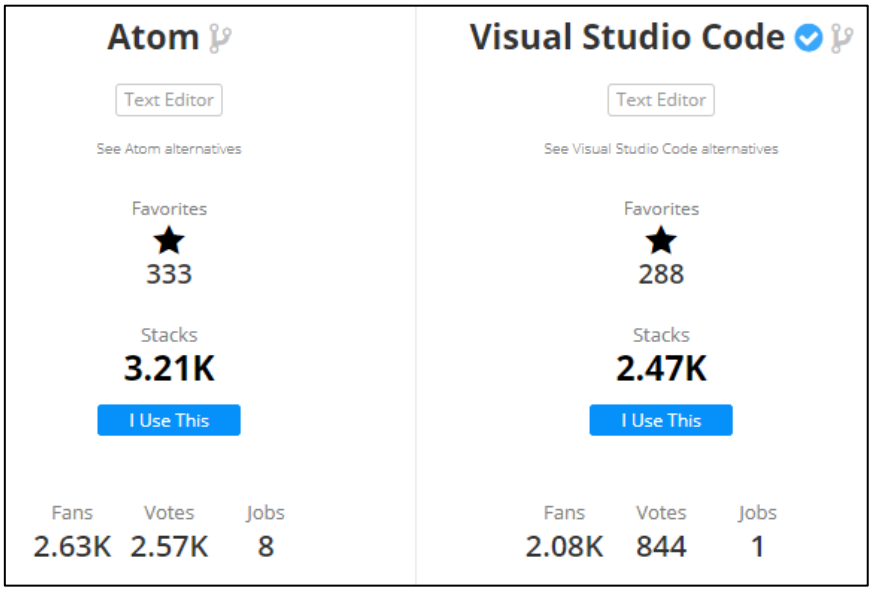
1.Minimale vereisten [16] [17]

Voor de doeleinden van het thesisproject is geen zware ontwikkelomgeving nodig. Doordat het project zich focust op het gebruik van webtechnologieën om een desktopapplicatie te maken, is ook geen nood aan extra ondersteuning voor databaseconnecties aan te maken, mobiele applicaties te ontwikkelen of andere programmeertalen die niet gericht zijn op webontwikkeling. Een ander voordeel dat VS Code heeft over Visual Studio is dat er minder strenge vereisten aan het gebruik van de ontwikkelomgeving hangen. [15, 16, 17, 18]

2.1.3.3 VS Code vergeleken met Atom

Atom is, net als VS Code, een editor die werd ontworpen met oog op webontwikkeling. Deze editor is zelf ontwikkeld met behulp van webtechnologieën zoals: Node.js, Electron, CoffeeScript en Less. CoffeeScript compileert naar JavaScript code en maakt deel uit van de getranscompileerde programmeertalen[15]. Less is opmaaktaal die functionaliteiten toevoegt bovenop de bestaande functionaliteiten van CSS. [19, 20, 21, 22]

Beide ontwikkelomgevingen hebben een soortgelijke achtergrond. Ze zijn allebei ontwikkeld met behulp van Electron, Node.js en JavaScript. De voornaamste kwalitatieve verschillen tussen de twee editors uiten zich in de performantie en de ondersteuning voor Git. In het geval van de performantie laat Atom te wensen over. Het openen van en schakelen tussen bestanden is relatief traag. Deze traagheid wordt nog eens verhoogd bij het openen van grote bestanden. Anderzijds kant heeft Atom een sterke ondersteuning voor Git die verder gaat dan de ondersteuning vanuit VS Code. Kwalitatieve verschillen die minder doorwegen is dat Atom autovervollediging van tekst en het paren van open en gesloten haken niet standaard ondersteund, maar deze functionaliteiten zijn wel beschikbaar in de vorm van plug-ins. In termen van populariteit zijn de twee editors aan elkaar gewaagd zoals te zien in figuur 10. [23, 24, 25]



10. Atom en Visual Studio Code popularitei [26]

2.1.3.4 VS Code vergeleken met Sublime Text

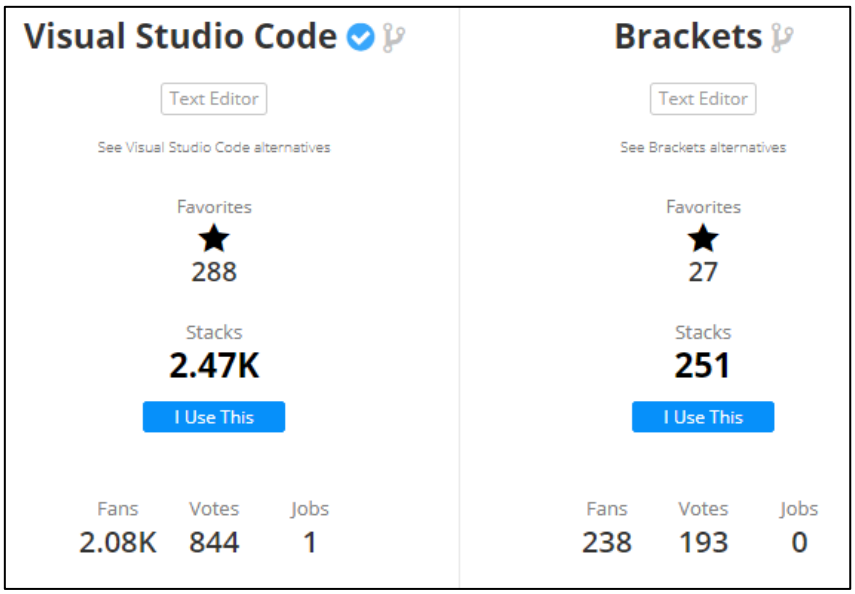
De verschillen tussen Sublime Text en VS Code zijn niet zo groot. Het grootste verschil tussen beide is dat Sublime Text een gepatenteerde editor is waarvoor $70 betaald moet worden om er blijvend gebruik van te maken. Dit is in tegenstelling tot VS Code, dat vrij uitgegeven wordt. Een ander punt waarin Sublime verschilt met VS Code is dat deze opmerkelijk weinig opslaggeheugen voor zich neemt, maar liefst 22MB. VS Code

[15] Programmeertalen die vertaald worden naar andere programmeertalen. Dit type programmeertaal heeft als doel om ontwikkelaars functionaliteiten aan te bieden die moeilijk of omslachtig te implementeren zijn in de doelprogrammeertaal.

daarentegen neemt met zijn 200MB toch wat meer ruimte in beslag. Veel van de andere functionaliteiten van Sublime zijn ook toegankelijk in VS Code. [27, 28, 29, 30]

2.1.3.5 VS Code vergeleken met Brackets

Brackets maakt een positief onderscheid met VS Code door zijn ingebouwde *live-browser preview*[16] en snelle aanpassing[17]. Deze functionaliteiten biedt VS Code niet aan. Met een grootte van 40MB is Brackets groter dan Sublime Text, maar nog steeds een stuk kleiner dan VS Code. Mindere aspecten aan Brackets zijn de kleinere gemeenschap die actief met de editor werken, dit is weergegeven in figuur 11, een gebrek aan indentatiegidsen[18] en ontbreekt automatische *bracket matching*[19]. [26, 31, 28, 32]



11. Populariteit van VS Code en Brackets [26]

2.1.3.6 Conclusie

Uit de vergelijking van alle editors, en IDE, is duidelijk dat ieder van de besproken programma's zijn eigen sterkte en zwaktepunten heeft. Zo is Visual Studio een zeer krachtige IDE die breed inzetbaar is en voor uiteenlopende doeleinden kan ingezet worden, maar deze kracht en veelzijdigheid draagt een kost met zich mee. De applicatie neemt namelijk veel opslagruimte op en heeft een redelijke hoeveelheid aan RAM-geheugen nodig om zijn werk te kunnen doen. Net dit laatste maakte Visual Studio een minder aantrekkelijke kandidaat voor de ontwikkeling van het thesisproject.

De editors die werden besproken hebben onderling veel gemeen. Ze bieden allemaal nagenoeg dezelfde functionaliteiten. De editors verschillen hoofdzakelijk enkel in een of twee functionaliteiten die al dan niet standaard worden ondersteund, de hoeveelheid opslaggeheugen ze nodig hebben, hoe snel de editor met bestanden kan werken en welke kostprijs eraan gepaard gaat. De editors die werden besproken zijn in grote mate gelijkwaardig aan elkaar en konden worden gebruikt om het project in te ontwikkelen. VS Code werd uiteindelijk gekozen omwille van zijn eenvoud. Daarbij levert het een hoop functionaliteiten die de ontwikkeling van dit project vereenvoudigen.

[16] Een functionaliteit die toelaat om HTML-elementen te laten oplichten in de browser door met de muis over het betreffende element in de editor te zweven.
[17] Een functionaliteit om de CSS-code van een HTML-element aan te passen zonder dat hiervoor moet gezocht worden in de CSS-documenten.
[18] Horizontale lijnen die verschillende indentatieniveau's weergeven.
[19] Het koppelen van open en gesloten haken, rechte haken of accolades. Wanneer een van de haken dan wordt aangeduid, wordt de daaraan gekoppelde haak opgelicht.

2.1.4 Node.js

Node.js is een *JavaScript runtime environment*[20] geschreven in C++ en maakt gebruik van de Google V8 JavaScript Engine. Node.js kan gebruikt worden als een webserveromgeving die kleine vertragingen en hoge datadebieten kan halen door gebruik te maken van *non-blocking* I/O[21]. Node.js kan ook gebruikt worden om JavaScript-code uit te voeren buiten de context van de browser.

Node.js werd ontwikkeld toen JavaScript het domein van de webbrowsers verliet en het mogelijk werd om met deze programmeertaal ook volwaardige applicaties te ontwikkelen. In deze paragraaf wordt kort aangehaald hoe gewoonlijk tewerk werd gegaan in webserveromgevingen en in meer detail wat Node.js aanbiedt. Er wordt in grote lijnen besproken wat Node.js doet, wat de *package manager* van Node.js te bieden heeft en kort besproken wat de V8 JavaScript Engine is. [33, 34]

2.1.4.1 Traditionele webserveraanpak

Voor de opkomst van Node.js, maakte meeste webservers voor elke inkomende aanvraag een aparte *thread*[22] of proces[23] aan om de aanvraag af te handelen en een antwoord te geven. Wat dit met zich mee draagt, is dat voor threads en processen overkoepelende organen nodig zijn die alles in goede banen leiden. Dit maakt dat voor zwaarbeladen systemen waardevolle tijd moet gespendeerd worden om deze processen en threads te coördineren. Dit zorgt voor vertragingen en leidt tot beperkingen in termen van schaalbaarheid en datadebiet. [33]

2.1.4.2 Node.js webserveraanpak

Voor het behandelen van aanvragen of connecties neemt Node.js een heel andere aanpak. Node.js werkt vanuit een enkele thread die alle aanvragen afhandelt. Voor elke aanvraag wordt dan een JavaScript-*callbackfunction*[24] uitgevoerd. Een callbackfunctie is wordt altijd asynchroon uitgevoerd. Het gebruik van dit type functies maakt het mogelijk om non-blocking I/O-afhandeling te voorzien.

Doordat Node.js in de regel op een enkele thread werkt is zijn er wel beperkingen voor verticale schaalbaarheid. De enkele thread die Node.js beheert, werkt vanuit een enkele processor. Tegenwoordig hebben veel thuiscomputers meerdere processorkernen en servers hebben er nog veel meer. Met zijn enkele thread om uit te werken, maakt Node.js geen ideaal gebruik van de bronnen die aangesproken kunnen worden.

Het probleem van de enkele thread kan wel verholpen worden. Het is mogelijk om in Node.js meerdere processen op te starten die op de andere processors hun werk verderzetten. Om controle over de processen te behouden kunnen ook *pipes* gemaakt worden tussen kinderprocessen en hun ouders.

Gebruik van Node.js levert over het algemeen performantiewinsten op ten opzichte van andere webserverapplicaties zoals: Apache HTTP Server, ASP.NET, enz. Buiten het domein van de webservers kan Node.js ook gebruikt worden om desktopapplicaties te ontwikkelen die gebruik maken van JavaScript. Doordat Node.js servers geprogrammeerd worden met de JavaScript-programmeertaal, een geïnterpreteerde taal die niet gecompileerd wordt naar binaire- of bytecode, is Node.js beperkter dan andere platformen die wel gebruik maken van gecompileerde talen. [33]

2.1.4.3 NPM

NPM (*node package manager*) is een ecosysteem aan gratis, herbruikbare Node.js-code. De package manager is de grootste in zijn soort. Elke package bestaat uit folders of registeritems die beschreven worden door een daarbij horende *package.json*. Elke package kan dan nog eens opgedeeld zijn in meerdere modules die in een JavaScript-project kunnen worden ingeladen in die bestanden waar ze nodig zijn.

[20] Een omgeving dat een applicatie toelaat om instructies en commando's te versturen naar de processor en toegang levert aan andere systeemfunctionaliteiten. [106]
[21] Non-blocking I/O voert een I/O initieert een I/O-instructie en gaat verder met berekeningen zonder te wachten op een antwoord. Het verwerken van het antwoord begint pas wanneer een antwoord wordt gegeven.
[22] De kleinste eenheid van verwerking die kan worden uitgevoerd door een *operating system*. [107]
[23] Een set van instructies die worden verwerkt door de processor. [108]
[24] Een functie dat wordt meegegeven aan een functie als parameter. De parameterfunctie wordt dan uitgevoerd onder bepaalde voorwaarden. [109]

Met NPM is het mogelijk om packages lokaal in een project te installeren als een afhankelijkheid[25] of globaal buiten de context van één enkel project. Met de manager is het ook mogelijk om alle afhankelijkheden van een project, die vermeld staan in de package.json, in één keer te installeren. Hierdoor is het niet nodig om elke bibliotheek of framework apart te installeren.

Het is niet noodzakelijk om NPM te gebruiken wanneer gewerkt wordt met Node.js. Er kan ook gebruik gemaakt worden van andere managers zoals bijvoorbeeld yarn van Facebook. Het is natuurlijk eenvoudig om NPM te gebruiken omdat deze standaard aanwezig is bij het installeren van Node.js. Met de package manager is niet alleen de grootste collectie aan gratis en herbruikbare code beschikbaar om uit te kiezen, maar kan ook vermeden worden dat elke gewenste bibliotheek of framework apart moet worden gekloond van een *repository*[26] en gebouwd moet worden. [33]

2.1.4.4 V8 Engine

De V8 JavaScript Engine is een *software engine*[27] die snel en efficiënt JavaScript-code uitvoert. De software engine werd geschreven in C++ en wordt gebruik door onder andere Google Chrome en Node.js. De V8 JavaScript Engine implementeert de specificaties van ECMAScript[28] die uitgeschreven staan in ECMA-262[29]. De software engine kan op zichzelf gebruikt worden of geïmplementeerd worden in C++-applicaties. Hierdoor is het mogelijk om C++-functies bloot te stellen aan JavaScript, waar deze functies dan gebruikt kunnen worden. [35, 34]

2.1.5 Electron

Er zijn twee belangrijke punten die in het opzet van dit project omschreven staan, namelijk dat de uiteindelijke applicatie ontwikkeld moet worden met webtechnologieën en als een desktopapplicatie te verspreiden moet zijn. Webtechnologieën, zoals de naam aangeeft, richten zich voornamelijk op toepassingen op het internet. Doorgaans zijn webapplicaties beschikbaar in de browser. Buiten de browser en het internet kunnen webtechnologieën niet zomaar gebruikt worden om desktopapplicaties te ontwikkelen. Net om dit mogelijk te maken werd het Electron framework in het leven geroepen en is eveneens de grootste speler op de markt voor de ontwikkeling van cross-platform desktopapplicaties.

Het gebruik van Electron is één van de vereisten verbonden aan de ontwikkeling van de applicatie. Dit framework staat toe om met behulp van webtechnologieën desktopapplicaties te maken die cross-platform zijn. Het framework werd ontwikkeld door de GitHub-organisatie[30] en wordt nog steeds actief door hen onderhouden. De oorspronkelijke doelstelling was om dit als framework te gebruiken voor de Atom-editor, maar is intussen uitgegroeid tot een zelfstandig framework dat open-source toegankelijk is. Electron werd zelf ontwikkeld met Chromium[31] en Node.js[32] als basis. Het framework werd onder andere gebruikt voor applicaties zoals: Slack, WhatsApp, Skype en VS Code.

Electron is niet het enige framework dat zijn zinnen heeft gezet op het toelaten van webtechnologieën voor desktopapplicaties. NW.js en Xojo bieden ook dergelijke functionaliteiten aan, maar zijn verre van zo populair als Electron. Hoewel elk van hen kan gebruikt worden om cross-platform desktopapplicaties te ontwikkelen met webtechnologieën, zijn er toch verschillen in hoe deze frameworks deze functionaliteit aanbieden. [36, 18, 37]

2.1.5.1 NW.js vergeleken met Electron

NW.js is aan de basis gelijkaardig aan Electron. Beide zijn gebouwd met Chromium en Node.js. Buiten deze gelijkenis, hebben beide frameworks andere keuzes gemaakt met betrekking tot de integratie van deze technologieën. NW.js heeft meer richting de browser georiënteerd. Een NW.js applicatie wordt gestart vanuit een HTML-document of JavaScript-document. Dit startpunt van de applicatie wordt gespecifieerd in de

[25] Een package waarvan de applicatie afhankelijk is voor de goede werking ervan.
[26] Een centrale plaats om software te verzamelen en af te halen wanneer nodig. [111]
[27] Een type software dat achter de schermen helpt bij het automatiseren van processen of verschillende software-elementen interactief met elkaar laten werken zonder menselijke tussenkomst. [110]
[28] *European Computer Manufacturers Association Script*. [112]
[29] De officiële standaard waaraan ECMAScript zich houdt. [112]
[30] Een organisatie die ontwikkelaars toelaat om onlineprojecten en de versie controle, aan de hand van het versie controlesysteem Git, van die projecten bij te houden. De projecten kunnen publiek of privé toegankelijk gemaakt worden. [88]
[31] Een open-source browser project ontwikkeld door Google dat de basis is voor de Chrome webbrowser [89].
[32] Een open-source serveromgeving dat ontwikkeld werd met de JavaScript programmeertaal. [90]

package.json[33] van het project. Dit startdocument wordt dan toegang gegeven tot Node.js. Als meerdere schermen worden gemaakt, dan wordt de toegang tot Node.js gedeeld. Electron daarentegen heeft zich meer gericht op Node.js. Bij het starten van de applicatie wordt een Node.js-instantie opgestart. Deze instantie opent dan een scherm waaraan een HTML-document kan gekoppeld worden.

De samenwerking tussen Chromium en Node.js in NW.js heeft ook een andere dynamiek dan die in Electron. In NW.js zijn aanpassingen in het Chromiumproject om een interactieve samenwerking te creëren tussen Node.js en Chromium. Bij Electron wordt gewerkt met Node.js processen. Er worden twee processen, een *mainprocess* en een *rendererprocess*[34]. Het mainprocess is het hoofdproces waaruit verschillende schermen kunnen worden geopend. Aan elk scherm wordt een rendererproces gekoppeld. Om communicatie mogelijk te maken tussen de processen worden communicatiekanalen geopend.

Om een duidelijker onderscheid te maken tussen de frameworks kan ook gekeken worden naar de populariteit op GitHub en StackOverflow[35]. Deze vergelijking kan als een reflectie dienen voor hoeveel en hoe snel bronnen kunnen gevonden worden om problemen op te lossen. De vergelijking is terug te vinden in tabel 2. Er is een duidelijk verschil in populariteit waar te nemen tussen de twee frameworks. Uit deze cijfers blijkt dat Electron zo goed als over de hele lijn tweemaal zo populair is dan NW.js. Op het punt van StackOverflow is het verschil opmerkelijk groot zelfs. [38, 39]

	GitHub Views	GitHub Stars[36]	GitHub Forks[37]	StackOverflow
NW.js	1815	33573	3735	316
Electron	2613	59149	7736	4799

2.Populariteit NW.js en Electron [36, 40]

2.1.5.2 Xojo vergeleken met Electron

Xojo is een bedrijf dat software tools aanbieden om ontwikkeling eenvoudig en snel te laten verlopen. Hun producten komen wel niet gratis. De prijsbepaling start bij $99 per jaar voor de Lite-versie en lopen op tot $1999 per jaar voor de Enterprise-versie voor de Xojo IDE. Xojo is het meest populair bij grote bedrijven (1000+). Electron is populairder bij kleine tot middelgrote bedrijven.

Xojo heeft ook een andere invalshoek bij de ontwikkeling van applicaties. Dit framework is nauw verbonden aan de ontwikkelomgeving van Xojo. UI's[38] kunnen met deze ontwikkelomgeving opgemaakt worden via een drag-and-drop interface. Extra functionaliteit kan dan ingebouwd worden met de eigen programmeertaal die door Xojo wordt aangereikt.

2.1.5.3 Conclusie

NW.js en Electron delen redelijk wat eigenschappen met elkaar, maar doordat NW.js minder populair blijkt te zijn in de programmeerwereld vergeleken met Electron is dit framework minder aantrekkelijk om te gebruiken. Deze lage populariteit en duidelijk gebrek aan StackOverflow vragen heeft tot gevolg dat minder bronnen beschikbaar zijn om een applicatie in dit framework te ontwikkelen. Xojo is beter geschikt voor grote bedrijven en leent zich dus niet goed voor een start-up zoals Qompium, waarvoor de applicatie wordt ontwikkeld. Hieruit blijkt dat, voor de doeleinden van het project, Electron een goed framework is om de applicatie in te ontwikkelen.

[33] Een JSON-bestand waar configuratie data van het project in verzameld staat.
[34] Een proces dat zich bezighoudt met de visualisering van de applicatie.
[35] Een website waar vragen kunnen gesteld worden en antwoorden gegeven worden in verband met softwareontwikkeling.
[36] Het aantal GitHub stars dat een project heeft reflecteert hoeveel personen dit project opvolgen. [99]
[37] Het aantal GitHub forks reflecteert hoeveel personen een kopie van het project bezitten waarop zij aanpassingen kunnen doen. Deze aanpassingen kunnen worden voorgesteld aan de eigenaar van het oorspronkelijke project of gebruikt worden om een nieuw project mee te starten. [100]
[38] Een gebruikersinterface bestaande uit visuele elementen.

2.1.6 Programmeertalen

Een onmiskenbaar onderdeel van dit project zijn de programmeertalen en opmaaktalen die gebruikt werden om het project tot een succesvol einde te brengen. In deze paragraaf wordt kort JavaScript, de programmeertaal van het web, besproken. Daarnaast worden ook vier andere programmeertalen besproken die compileren naar JavaScript met de bedoeling om betere code te genereren dan doorgaans mogelijk is door pure JavaScript te schrijven. De keuze voor deze vier programmeertalen werden uit [40, 41, 42] geïnspireerd.

2.1.6.1 JavaScript

Deze programmeertaal is een geïnterpreteerde of *just-in-time* (JIT) gecompileerde taal die voornamelijk gekend is al de programmeertaal van het web, maar kent ook buiten de webomgeving toepassingen. Deze taal is prototype-gebaseerd[39] en dynamisch getypeerd[40], en ondersteunt object-georiënteerde[41], imperatieve[42] en functionele[43] programmeerstijlen.

In de wereld van de webontwikkeling wordt JavaScript toegepast om website en webapplicaties dynamisch te laten reageren op de handelingen van de gebruiker of met de website of applicatie zelf. Deze taal wordt dan ook ondersteund door alle moderne browsers waar deze toegang heeft tot het *Document Object Model*[44] (DOM). Hierdoor is het mogelijk om dynamisch HTML-elementen toe te voegen, te verwijderen of aan te passen naargelang het gewenste effect. [43, 44]

2.1.6.2 TypeScript

TypeScript is in het leven geroepen door Microsoft om enkele van de pijnpunten van JavaScript te verzachten. In hoofdzaak betrekt dit zich tot de dynamische typering en de prototype-gerichte natuur van JavaScript. Deze programmeertaal is in feite JavaScript, maar dan met een statisch getypeerde en object-georiënteerde natuur. Hiervoor ontleent het veel van de paradigma's[45] van C++ en Java. Het blijft nog steeds mogelijk om typische JavaScript-code te schrijven in TypeScript-documenten zonder gebruik te maken van TypeScript-syntax.

Deze programmeertaal kan niet worden geïnterpreteerd door de browser. Hiervoor wordt TypeScript getranscompileerd naar JavaScript. Doordat TypeScript compileert naar JavaScript is, vertaalt de statische typering van de geschreven code niet naar de gebruikte code. Met andere woorden is de gecompileerde code nog steeds dynamisch en algemeen toepasbaar. TypeScript dient niet om statische typering af te dwingen op JavaScript, maar om deze stijl van code schrijven op te dringen aan de programmeur. De statische typering helpt dan ook om vroeg in het productieproces fouten op te sporen die anders moeilijker te achterhalen zijn met dynamische typering. Een voorbeeld met TypeScript en zijn getranscompileerde JavaScript is te zien in Figuur 12. [45, 46]

```typescript
function inspectAll(data:Array<object>):Array<object>{
    let newarr = []
    data.forEach(val => {
        let newobj = {}
        Object.keys(val).forEach(key => {
            newobj[key] = inspect(val[key]);
        });
        newarr.push(newobj);
    })
    return newarr;
}
```

```javascript
function inspectAll(data) {
    var newarr = [];
    data.forEach(function (val) {
        var newobj = {};
        Object.keys(val).forEach(function (key) {
            newobj[key] = inspect(val[key]);
        });
        newarr.push(newobj);
    });
    return newarr;
}
```

12.TypeScript (links) gecompileerd naar JavaScript (rechts)

[39] Een programmeerstijl waarbij reeds bestaande objecten worden gekopieerd en hergebruikt om een programma te maken. Net als object-georiënteerde programmeertalen wordt er gebruik gemaakt van overerving, maar dan zonder klassen. [92]
[40] Variabelen en functies accepteren elk datatype, zelfs nadat ze voor het eerst gebruikt zijn geweest.
[41] Een programmeerstijl dat volledig werkt met klassen.
[42] Een programmeerstijl waarbij het programma beschreven wordt als een collectie van toestanden. Deze toestand beschrijft het gedrag van het programma.
[43] Een programmeerstijl waarbij het volledige programma wordt beschreven als een collectie van functies die enkel variabelen ontvangen en teruggeven. Deze functies mogen geen externe variabelen aanpassen.
[44] Een API voor HTML- en XML-documenten. Deze API stelt een structuur op van de documenten en definieert de manier waarop documenten mogen geraadpleegd en aangepast worden. [93]
[45]

2.1.6.3 CoffeeScript

CoffeeScript is een programmeertaal met het expliciete doel om enkel de goede delen van JavaScript te benadrukken terwijl het er een kortere syntax op nahoudt. Deze taal behoudt wel de dynamische typering en het prototype-model van JavaScript. CoffeeScript is wel direct te gebruiken in de browser met behulp van een bibliotheek en is het mogelijk om CoffeeScript te gebruiken in combinatie met de React-bibliotheek.

Een kortere en meer modulaire opbouw van code in CoffeeScript maakt het een aantrekkelijke taal om te gebruiken. Doordat het de mindere aspecten van de JavaScript-programmeertaal verdoezelt, staat het ook toe om betere code te schrijven dan algemeen mogelijk is in JavaScript. Al deze functionaliteiten van deze taal komen wel met een kost. De opbouw en syntax van CoffeeScript-code is verschillend van die van JavaScript waardoor extra tijd moet worden geïnvesteerd in het leren schrijven van CoffeeScript-code. Een codevoorbeeld geschreven in CoffeeScript wordt weergegeven in figuur 13. [47, 42]

1313

```coffeescript
# Functions:
square = (x) -> x * x

# Arrays:
list = [1, 2, 3, 4, 5]

# Objects:
math =
  root:   Math.sqrt
  square: square
  cube:   (x) -> x * square x

# Array comprehensions:
cubes = (math.cube num for num in list)
```

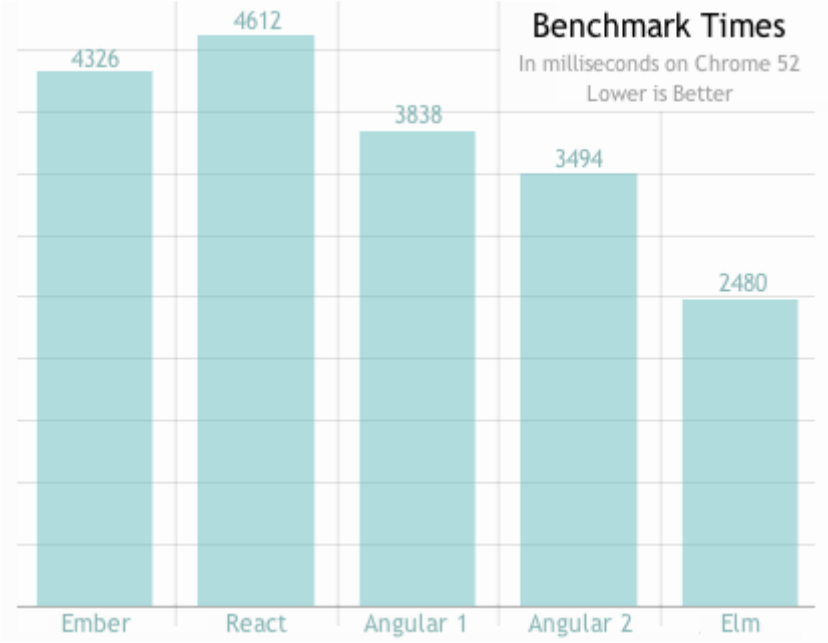
```javascript
// Functions:
var cubes, list, math, num, square;

square = function(x) {
  return x * x;
};

// Arrays:
list = [1, 2, 3, 4, 5];

// Objects:
math = {
  root: Math.sqrt,
  square: square,
  cube: function(x) {
    return x * square(x);
  }
};

// Array comprehensions:
cubes = (function() {
  var i, len, results;
  results = [];
  for (i = 0, len = list.length; i < len; i++) {
    num = list[i];
    results.push(math.cube(num));
  }
  return results;
})();
```

13. CoffeeScript (links) ten opzichte van JavaScript (rechts)

2.1.6.4 Elm

Elm maakt volledig gebruik van het functionele aspect van JavaScript en leent daarbij syntax en paradigma's die ook bij Haskell kunnen gevonden worden. Deze taal bestaat dus volledig uit functies en *immutable*[46] variabelen. Het type van variabelen wordt vastgelegd door middel van type-inferentie[47]. Elm beweert zelf geen *runtime exceptions*[48] te produceren als de code compileert. Als er toch fouten in de code aanwezig zouden zijn, wordt dit tijdens het compilatiestadium vastgesteld. Elm maakt gebruik van zijn eigen virtuele DOM[49] die helpt om snellere JavaScript code te genereren dan andere omgevingen die virtuele DOM's aanbieden. Dit is weergegeven in figuur 14.

Doordat Elm zich volledig richt op de functionele programmeerstijl verschilt het sterk van de manier waarop JavaScript-code wordt geschreven. Dit dat de overgang van het programmeren in JavaScript naar Elm een grote aanpassing is. Een groot voordeel aan Elm is dat programma's kunnen geschreven worden waarbij het zeker is dat er geen runtime exceptions zich zullen voordoen bij het uitvoeren van het programma. Een extra voordeel is de snelheidswinst die kan gemaakt worden dankzij het gebruik van de virtuele DOM van Elm ten opzichte van code die direct in JavaScript is geschreven of virtuele DOM's die werden gebouwd met JavaScript. Een codevoorbeeld in lijn met dat van Elm wordt weergegeven in figuur 15. [48, 49, 50, 51]



14. Snelheidsvergelijking van Elm ten opzichte van andere omgevingen met viruele DOM's (lager is beter) [49]

[46] Onveranderlijk na de eerste declaratie
[47] Het automatisch achterhalen van het datatype van expressies. [94]
[48] Fouten die tijdens het uitvoeren van een programma worden gegenereerd.
[49] Een abstractie van de DOM van de browser waarop sneller bewerkingen op uitgevoerd kunnen worden dan op de DOM van de browser. [95]

```
import Html exposing (li, text, ul)
import Html.Attributes exposing (class)

square x = x * x

cube x = x * square x

list = [1,2,3,4,5]

applySquare list =
  case list of
    [] ->
      []

    first :: rest ->
      cube first :: applySquare rest

main = text( toString( applySquare list))

```

15. Elm codevoorbeeld

2.1.6.5 ClojureScript

ClojureScript is net als Elm een functionele programmeertaal dat gebruik maakt van *immutable* datastructuren. Deze programmeertaal baseert zich op de gelijknamige Clojure programmeertaal, maar omgewend om naar JavaScript omgezet te kunnen worden. Met ClojureScript kunnen ook bestaande JavaScript-bibliotheken worden gebruikt en maakt zelf gebruik van de Google Clojure Compiler om geoptimaliseerde code te genereren.

Doordat ClojureScript een functionele programmeertaal is en zich syntactisch verwijdert van JavaScript, zorgt dit voor een extra leercurve om goed met de programmeertaal overweg te kunnen. Net als Elm, levert ClojureScript compactere code op dan JavaScript onder andere dankzij de invloeden van functioneel programmeren. ClojureScript voert aan de hand van de Google Clojure Compiler strenge optimalisaties door die onder andere ongebruikte code verwijderen uit de applicatie. [52, 53, 54]

2.1.6.6 Conclusie

Alle opties die werden vermeld zijn volwaardige talen die gebruikt hadden kunnen worden voor dit project. Allemaal proberen ze de tekortkomingen van JavaScript weg te steken of de goede punten van JavaScript naar de voorgrond te halen. Elm en ClojureScript proberen dit te doen met een puur functionele aanpak, waarbij TypeScript en CoffeeScript meer leunen naar de object-georiënteerde programmeerstijl. Exclusief TypeScript, hebben alle besproken programmeertalen hiervoor een nieuwe programmeertaal in het leven geroepen of grote porties geleend van bestaande programmeertalen. Doordat TypeScript JavaScript is met wat extra's, maakt dit het een makkelijke taal om aan te leren eens JavaScript gekend is. Dit is bij de andere besproken talen een barrière omdat de syntax van nul af aan geleerd moet worden. Net dit laatste maakte TypeScript interessant voor dit project omdat het minder tijdsintensief is om aan te leren dan de andere alternatieven.

2.1.7 Userinterface bibliotheken en frameworks

In het projectopzet werd het gebruik van React als userinterface bibliotheek vastgelegd. Er wordt in deze paragraaf in hoofdzaak de Reactbibliotheek meer toegelicht. Daarnaast worden kort twee alternatieve bibliotheken besproken die ook voor het project gebruik hadden kunnen worden.

2.1.7.1 React

React is een JavaScript-bibliotheek dat helpt om makkelijker en modulair userinterfaces te ontwikkelen. Hiervoor introduceert React het concept van de component. Een component wordt opgeroepen als een JSX-element[50]. Een component is een JavaScript-klasse of een JavaScript functie. Hierdoor moet niet noodzakelijk vastgehouden worden aan de JSX-notatie en kan JavaScript-notatie ook gebruikt worden om componenten te declareren. Een React-component is ofwel een presentatie-component ofwel een container-component. Het

[50] JSX is een voorbewerkingsstap die XML-syntax toelaat voor JavaScript. [97]

verschil tussen de twee typen componenten is dat een presentatie-component geen eigen toestand bijhoudt, waartegen een container-component dat wel doet. Meer specifiek is een presentatie-component een functie dat React-componenten en/of HTML-elementen teruggeeft. Een container-component is een JavaScript-klasse. Een container-component kan zijn eigenschappen veranderen aan de hand van zijn eigen toestand. Een statische component kan enkel van eigenschappen veranderen aan de hand van externe toestanden. Elke container-component bestaat uit vier onderdelen:

- Externe toestandsvariabelen ofwel *props*
- Interne toestandsvariabelen ofwel *state*
- Een *renderfunctie*
- Functies die handelen op de eigen toestand van de component of de toestand van de ouder

Elke React-applicatie start vanaf een hoofdcomponent dat zich koppelt aan een HTML-document. Vanuit deze hoofdcomponent wordt de rest van de applicatie hiërarchisch opgebouwd. In deze hiërarchie kan elke component andere componenten oproepen en variabelen aan doorgeven zonder enig probleem. Hierbij is de component die andere componenten oproept de oudercomponent en de componenten die werden opgeroepen de kindercomponenten. Het is dus eenvoudig om variabelen en functies door te geven aan kindercomponenten vanuit de oudercomponent, maar de omgekeerde bewerking is niet zo eenvoudig. Om vanuit de kindercomponent variabelen of functies door te geven aan de oudercomponent moet de oudercomponent eerst een *callback*[51] functie meegegeven hebben aan dat kindercomponent. Zo niet, dan kan de kindercomponent niet communiceren met de ouder.

Wanneer een component van toestand verandert door een verandering van de props of de state van die component dan wordt deze component geüpdatet. Om niet bij elke update heel het scherm opnieuw te moeten laten genereren maakt React gebruik van een virtuele DOM om te bepalen wanneer een component moet vernieuwd worden in de DOM van de browser of niet. [55]

2.1.7.2 Angular

Angular is een framework dat het mogelijk maakt om dynamische webpagina's, mobiele applicaties en desktopapplicaties mee te ontwikkelen. Dit framework werd door Google in het leven geroepen om snelle ontwikkeling van één-pagina applicaties mogelijk te maken. Doordat het een framework is, geeft Angular iets minder vrijheid over de manier waarop een webpagina kan worden opgebouwd dan een bibliotheek dit toelaat. Zelf werd het framework ontwikkeld in JavaScript, maar code in het framework wordt geschreven met TypeScript vanaf Angular2 en hoger.

Voor het opbouwen van webpagina's maakt Angular gebruik van het *Model-View-Controller*[52] principe. Hierdoor is het mogelijk om de logica van de applicatie los van de DOM manipulatie te maken, waardoor dynamisch updaten van de applicatie mogelijk is. Een van de concepten in Angular dat helpt bij het dynamisch opbouwen van webpagina's zijn directieven. Deze directieven zijn zelfgemaakte HTML-elementen die gekoppeld zijn aan JavaScript-code. Dit maakt het mogelijk om modulair tewerk te gaan bij het maken van een applicatie. Daarnaast maakt Angular ook gebruik van *data-binding*[53]. Data-binding zorgt ervoor dat wanneer het Model van toestand verandert de View mee wordt geüpdatet en omgekeerd. [56, 57, 58]

2.1.7.3 Vue

Vue is net als Angular een framework dat de dynamische opbouw van webapplicaties wilt bevorderen, maar is opmerkelijk jonger. Het framework kwam tot leven in het jaar 2016. Hiervoor maakt het net als de andere besproken technologieën in deze paragraaf gebruik van componenten en combineert enkele van de eigenschappen van Angular en React. Net als het geval is bij React, worden bij Vue de logica en opbouw van componenten gecombineerd in een enkel bestand. Vue gaat hier zelf iets verder in en voegt in het componentenbestand ook de stijlopmaak van die component. Om communicatie tussen componenten mogelijk te maken, maken Vue-componenten ook gebruik states en props. Net als React, komt Vue met een virtual DOM.

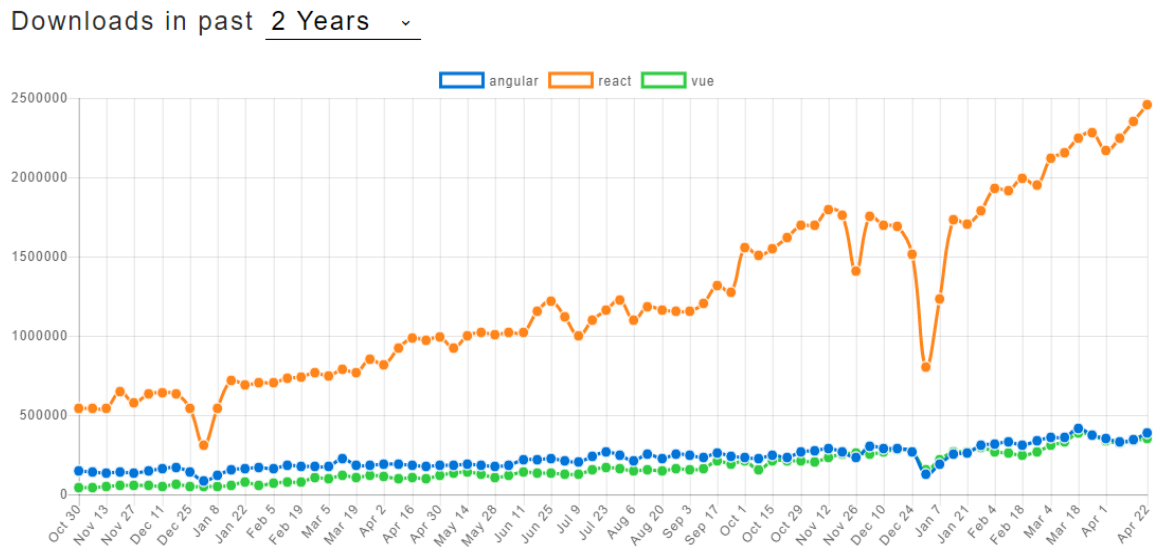
[51] Functies die zijn gekoppeld aan een bepaalde locatie of omgeving in de code van een project. Wanneer deze functies worden opgeroepen, dan handelen ze acties uit op die locatie of omgeving.
[52] Een principe waarin het model, het visuele aspect en de controle logica die wordt blootgesteld aan de gebruiker lost van elkaar staan.
[53] Synchronisatie tussen het model en het visuele aspect van een applicatie. [96]

Gelijkaardig aan Angular worden HTML-elementen samengevoegd met JavaScript-code en maakt het hiervoor ook gebruik van directieven. [59, 57]

2.1.7.4 Vergelijking

Elk van de besproken technologieën maakt gebruik van componenten om een applicatie op te kunnen delen in afzonderlijke functionele delen. Toch nemen de technologieën een andere houding aan om deze componenten van functionaliteit te voorzien. Angular splitst alles op aan de hand van het Model-View-Controller principe, waarbij React en Vue functionaliteit direct koppelen aan de View. React maakt gebruik van JSX om HTML en JavaScript te koppelen in tegenstelling tot Angular en View die HTML en JavaScript koppelen met directieven. Een groot verschil is ook dat React een JavaScript-bibliotheek is, waardoor er minder beperkingen zijn op hoe de applicatie wordt opgebouwd. Angular en Vue zijn frameworks die hier meer controle opeisen.

Om een duidelijker beeld te schetsen tussen de verschillende technologieën wordt er ook gekeken naar de populariteitscijfers die worden vrijgegeven door npm[54]. Uit figuur 16 is op te maken dat React veruit het meeste gebruik heeft gekend op npm. Daaruit is ook op te maken dat Vue en Angular nagenoeg dezelfde populariteit kennen. Figuur 17 tekent een milder beeld. Uit de GitHub-cijfers blijken React en Vue nagenoeg even veel stars en forks te hebben. Angular heeft minder stars dan Vue of React, maar wel meer forks. Dit geeft aan dat hoewel Angular minder wordt opgevolgd dan de andere technologieën, er toch nog veel projecten bestaan die proberen Angular te verbeteren of uit te breiden.



16. npm downloadcijfers [61]

Github Stats

	stars	forks	issues	updated	created
vue	91907	13500	164	Apr 22, 2018	Jul 29, 2013
angular.js	58355	28903	585	Apr 19, 2018	Jan 6, 2010
react	94221	17752	434	Apr 24, 2018	May 24, 2013

17. GitHub populariteitscijfers [61]

2.1.7.5 Conclusie

Er is niet een van de technologieën die als absoluut beter of als beste kan beschouwd worden. Elk van de besproken technologieën is een volwaardige technologie die gebruikt kan worden om allerlei applicaties mee te ontwikkelen. Belangrijke eigenschappen die React interessant maken voor dit project zijn de relatieve vrijheid die het bied om een applicatie op te bouwen en een groot aantal npm-pakketten. Dit laatste betekent dat een

[54] Node Package Manager. Bibliotheken die met behulp van Node.js kunnen worden gedownload.

redelijk aantal componenten niet zelf ontwikkeld moeten worden, maar dat deze van andere bronnen kunnen worden hergebruikt.

2.1.8 Opmaakbibliotheek

Voor het project werd gekozen om een opmaakbibliotheek in het project te voegen. Een opmaakbibliotheek is handig om makkelijk een stijl te geven aan een webapplicatie zonder dat hiervoor zelf veel CSS geschreven moet worden. Uit [60, 61, 62] kwamen Material UI, React Bootstrap en React Toolbox het meest belovend uit de bus.

Eerst is Material UI dat de stijlvoorwaarden van Google's Material Design implementeert. Deze opmaakbibliotheek levert meer dan 90 componenten om een UI mee op te maken. Gekoppeld aan deze bibliotheek bestaat er ook Material UI Icons die enkel en alleen bestaat uit iconen. Deze bibliotheek telt meer dan 900 iconen om uit te kiezen. Deze iconen zijn gemaakt om in Materiul UI gebruikt te worden. Voor elk van de componenten is het mogelijk om een eigen stijl te definiëren. [63, 64]

React Bootstrap werd opgebouwd met de stijldefinities van Bootstrap dat door Twitter wordt onderhouden. Deze opmaakbibliotheek telt 26 componenten die gebruikt kunnen worden om een UI mee op te maken. Deze componenten kunnen ook een eigen stijl worden gegeven. [60, 65]

Als laatste komt React Toolbox. Deze opmaakbibliotheek is ook gebaseerd op de Google Material Design specificaties. React Toolbox levert ongeveer evenveel componenten als React Bootstrap en net als de andere opmaakbibliotheken laat het toe om een eigen stijl te geven aan de componenten. [66]

Doordat Material UI een groot aantal componenten blootstelt om te gebruiken, meer nog dan de andere bibliotheken, samen met een heel groot aandeel aan iconen die kunnen worden gebruikt in samenspraak met de componenten van Material UI maakt deze bibliotheek interessant voor dit project. Dit laat redelijk wat keuze toe om eigen componenten flexibel op te bouwen. Dit wil niet zeggen dat Material UI alles aanbiedt wat gewenst kan worden. De verschillende bibliotheken bieden sommige andere componenten aan, maar voor de doeleinden van die project lever Material UI het nodige.

2.1.9 Grafiekbibliotheken

Een belangrijk onderdeel van het project zijn de grafische weergaven. In heel wat gevallen is data duidelijker weer te geven aan de hand van grafieken of andere vormen van compacte data weergave. Omdat het thesisproject zich richt op het ontwikkelen van een Business Intelligence applicatie voor JSON is het geen slechte zaak om gebruik te maken van een bibliotheek die enkele of alle van de gewenste grafische weergaven levert. Er worden in deze paragraaf enkele zulke bibliotheken besproken, met elkaar vergeleken en de uiteindelijke beslissing beredeneerd.

2.1.9.1 Chart.js

Chart.js is een open-source project en levert 8 verschillende basisgrafieken: een lijngrafiek, een staafdiagram, een radardiagram, een taartdiagram, een doughnutdiagram, een polairdiagram, een bubblediagram, een spreidingsdiagram, een oppervlaktediagram en een gemengd diagram. Voor elk van deze diagrammen bestaan personalisatiemogelijkheden. Zo is het mogelijk om de assen van de grafieken lineair, logaritmisch of in categorieën op te delen, maar kunnen ook *tooltips*[55], kleuren, interacties, ratio's, animaties, enz. bepaald worden. Er bestaat ook een variant van deze bibliotheek voor React onder de naam "react-chartjs-2". Wat niet inbegrepen zit in de bibliotheek is het zoomen en het verschuiven van het beeld van de grafiek. [67, 68]

2.1.9.2 D3

D3 is niet echt een grafiekbibliotheek, maar is een JavaScript-bibliotheek die documenten manipuleert aan de hand van data. Hiervoor voorziet het heel wat functies om zelf visuele weergaven te ontwikkelen om data weer te geven met behulp van HTML, SVG en CSS. Deze bibliotheek staat toe om willekeurige data te koppelen aan de DOM van browsers en hierop transformaties uit te voeren. Met D3 is het mogelijk om alle gewenste functionaliteiten, zoals interactie met de gebruiker of animaties, te koppelen aan visuele weergaven. Hierdoor bestaat er een grote flexibiliteit in wat met D3 kan worden gevisualiseerd. Een voorbeeld van de flexibiliteit van D3 wordt weergegeven in figuur 18. [69]

[55] Een venster met de waarde van het element waarover de cursor zich bevindt.

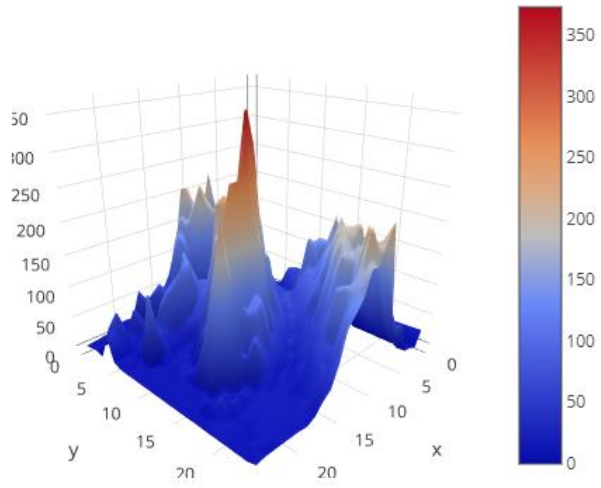
18. Een radiale boomdiagram [72]

2.1.9.3 DYGraphs

DYGraphs is een grafiekbibliotheek dat eigenlijk maar één grafiektype bevat, namelijk de lijngrafiek. Het is ook mogelijk om er een spreidingsdiagram mee te maken door de connecties tussen de verschillende datapunten uit te zetten, waardoor een grafiek van punten overblijft. Ondanks dat DYGraphs buiten de toon valt, wordt het toch besproken omdat geoptimaliseerd is om met grote datasets, in de grootteorde van miljoenen datapunten, makkelijk overweg te kunnen. Zoomen en verschuiven van het beeld zitten standaard inbegrepen in deze bibliotheek. De grafiek werkt wel slechts voor numerieke- of datumformats waardoor het enkel specialistisch gebruikt kan worden. [70]

2.1.9.4 Plotly

Plotly is een open-source en heel uitgebreide bibliotheek dat niet alleen 2D-grafieken, maar ook 3D-grafieken toegankelijk maakt. Om al dit mogelijk te maken werd Plotly ontwikkeld met D3 en stack.gl. Plotly biedt alle mogelijkheden om wetenschappelijke data om te zetten naar de gepaste visuele weergaven. Elke weergave kan dan daarenboven nog eens gepersonaliseerd worden. Deze bibliotheek is beschikbaar voor JavaScript en React, maar ook voor andere programmeertalen is deze bibliotheek toegankelijk. De bibliotheek werkt wel niet goed samen met WebPack. Daarvoor moeten enkele omwegen gemaakt worden om dit te doen werken. Een voorbeeld van wat mogelijk is met Plotly is weergegeven in figuur 19. [71, 72]



19. Plotly 3D diagram [76]

2.1.9.5 Highcharts

Highcharts is een commerciële bibliotheek, waarbij enkel niet betaald moet worden voor niet-commerciële instanties of voor persoonlijk gebruik. Highcharts heeft drie onderdelen: Highcharts, Highstock en Highmaps. Het onderdeel Highcharts is een collectie van visuele weergaven die algemeen toepasbaar zijn. Deze bibliotheek biedt buiten variaties op de acht grafieken die Chart.js aanbiedt, ook meterdiagrammen, hittediagrammen, boomdiagrammen en andere specialistische grafieken. Het biedt ook 3D-varianten aan van staafdiagrammen, taartdiagrammen en spreidingsdiagrammen. Highstock en Highmaps specialiseren zich respectievelijk voor het weergeven van beursgangdata en van geologische data. [73]

2.1.9.6 Conclusie

Highcharts en Plotly zijn de sterkste en uitgebreidste grafiekbibliotheken die werden besproken. Allebei leveren zij een breed gamma aan visualisaties die gebruikt kunnen worden om data weer te geven. Highcharts werd achterwege gelaten om eventuele wettelijke beperkingen door het commerciële karakter van de bibliotheek te vermeiden. Plotly, uit de overgebleven bibliotheken zou dan overblijven als makkelijkste om te integreren in het project. Integratie van de React versie van Plotly wou echter niet lukken omwille van een minder dan ideale samenwerking tussen WebPack en Plotly.

Uiteindelijk werden Chart.js, D3 en DYGraphs gekozen om de verschillende grafieken in het project te bouwen. Chart.js biedt de meest courante grafieken in een mooi en eenvoudig pakket. Voor grotere datasets wordt DYGraphs gebruikt. DYGraphs is geoptimaliseerd om vlot met deze grote hoeveelheden data overweg te kunnen zonder veel vertragingen. D3 kan dan gebruikt worden voor minder courante grafische weergaven van data die eventueel interessant zouden zijn.

Er bestaan veel grafiek bibliotheken en de selectie die hier werd gemaakt is niet noodzakelijk de beste in alle opzichten, maar deze selectie zorgt voor een basis aan grafieken die ondersteund worden in de applicatie. Met D3 is het ook mogelijk om deze flexibel uit te breiden. De applicatie hoeft ook niet gebonden te zijn aan de grafische bibliotheek. Deze kan op een later stadium nog steeds vervangen worden, zolang de transformatie van data naar de grafieken mee verandert.

2.1.10 Electron executable bibliotheek

Electron is handige tool om het bereik webapplicaties uit te breiden. Het framework kan wel niet direct omgezet worden naar een uitvoerbaar bestand dat gebruikt kan worden op eender welke computer. Hiervoor moet een tussenstap worden gemaakt. Er bestaan vier erkende manier die door Electron zelf worden voorgeschreven:

- Het gehele project omzetten naar een asar-bestand[56];
- Het project bundelen met behulp van de electron-builder bibliotheek;

[56] Bundelt alle bestanden samen in één groot bestand dat behandeld wordt als een archief. [75]

- Het project bundelen met behulp van de electron-packager bibliotheek;
- Het project bundelen met behulp van de electron-forge bibliotheek.

2.1.10.1 ASAR

Asar is een archief-type bestand dat willekeurige toegang ondersteunt. Het is mogelijk om asar in de console te gebruiken, maar ook programmatorisch. Asar slaat bestanden op in een JSON-formaat. Het laat ook toe om transformaties op het archief toe te passen.

Om met asar-bestanden te kunnen werken moeten eerst de *asar-utility* worden geïnstalleerd met npm. Daarna kunnen de bestanden van de applicatie worden verpakt met de *pack*-functionaliteit van asar. asar-bestand kan worden ingeladen in Electron via Node API's of Web API's.

Toch zijn er enkele zaken die niet mogelijk zijn met Node.js in verband met asar-bestanden. Zo zijn asar-bestanden *read-only* en kunnen dus niet gewijzigd worden door Node.js en kan het bestandstype ook niet gebruikt worden als een *working directy*. Een programma kan dus niet gestart worden vanuit het asar-bestand. [74, 75]

2.1.10.2 Electron-Builder

Electron-builder biedt zich aan als een volledige oplossing voor het verpakken en bouwen van Electron applicaties, als Proton Native en Muon applicaties die klaar zijn om verspreid te worden. De bibliotheek maakt een installeerbaar bestand aan. Ingebouwd in deze bibliotheek is het automatisch updaten van de afhankelijkheden van de applicatie. Hierdoor is de applicatie altijd *up-to-date*. Andere belangrijke ondersteuningen die de bibliotheek aanbiedt zijn:

- Verpakken naar specifieke of alle platformen (MacOS, Linux en Windows);
- Tot twee package.json's mogelijk;
- *Versioning* van de *build*.

Naast het downloaden via npm en het bijvoegen van iconen voor het uitvoerbare bestand, zijn er enkele noodzakelijke toevoegingen die moeten worden gemaakt aan de package.json:

- Toevoegen van een naam, een beschrijving, een versie en een auteur;
- Build specificaties toevoegen;
- De electron-builder scripts toevoegen

Standaard maakt deze bibliotheek een asar-bestand aan, maar specificaties kunnen worden meegegeven zodat het project wordt geëxporteerd onder een ander bestandstype. [76]

2.1.10.3 Electron-Packager

Electron-packager is een open-source Node.js bibliotheek dat Electron-applicaties bundelt naar een uitvoerbaar bestand. Deze bibliotheek wordt gebruikt in de console en kan niet gebruikt worden om installeerbare bestanden te maken. Net als het geval is voor Electron-builder, kan Electron-packager gebruikt worden om Electron-applicaties te verpakken voor Windows, MacOS en Linux.

Om gebruik te kunnen maken van deze bibliotheek moet deze eerst via npm worden geïnstalleerd ofwel als *development dependency* of als *CLI*[57]. Wat in de commandolijn moet worden meegeven is te zien in figuur 20. De verplichte velden betekenen het volgende:

- De folder waar het uitvoerbaar bestand naar geëxporteerd moet worden na verpakken;
- De naam van de applicatie;
- Het platform of de platformen waar de Electron-applicatie voor verpakt moet worden;
- De architectuur van het platform waarop de applicatie moet werken.

Er kunnen ook extra opties worden meegegeven om het verpakken specifieker toe te spitsen op het gewenste resultaat. Dit proces zal de correcte versie van Electron downloaden voor het platform of platformen die werden gekozen.

[57] Command Line Interface. Functies van de bibliotheek zijn dan beschikbaar in de console.

```
electron-packager <sourcedir> <appname> --platform=<platform> --arch=<arch> [optional
flags...]
```

20. Electron-packager commandolijn velden

Het is niet verplicht om het verpakken van de applicatie via de commandolijn uit te voeren. De verplichte parameters die Electron-packager nodig heeft om de applicatie te verpakken, kunnen ook worden meegegeven in de package.json. In dit geval moet slechts “electron-packager.” in de commandolijn worden ingegeven en zal de bibliotheek zelf uit de package.json de nodige informatie halen. [77]

2.1.10.4 Electron-Forge

Electron-forge neemt het proces van uitvoerbare bestanden te maken van webtechnologieën samen met Electron een stap verder. Deze bibliotheek stelt zich op als het startpunt waaruit een applicatie kan ontwikkeld worden. Met de bibliotheek kan een project opgestart worden. Hiervoor biedt het *templates* aan die het configureren van de applicatie optioneel maken. Het biedt templates aan voor projecten die gebruik maken van:

- Enkel JavaScript;
- React;
- React en TypeScript;
- Angular 2;
- Vue.js 2.0;
- Jade Templating.

Met deze bibliotheek kan de applicatie verpakt worden voor MacOS, Windows en Linux en elke architectuur die door Electron-forge ondersteund wordt met een enkel commando. De gewenste platformen en architecturen moeten worden in de package.json worden vermeld. De bibliotheek ondersteund ook integratie met GitHub, Travis Cl en AppVeyor. Hierdoor kan het bouwproces voor alle platformen en architecturen, los van de machine waarop de applicatie werd ontwikkeld, automatisch doorgevoerd worden. [78]

2.1.10.5 Conclusie

Uiteindelijk werd gekozen om de applicatie te verpakken met Electron-builder. Met deze bibliotheek is het redelijk eenvoudig om de applicatie te verpakken naar de gewenste platformen en architecturen samen met automatische update-functionaliteit. Ook bundelt het heel de applicatie in een installer, waardoor dit niet zelf moet ondernomen worden. Dankzij deze functionaliteiten kunnen meeste handelingen die komen kijken bij het maken van een distributie-klare applicatie uit handen worden gegeven, waardoor vergissingen en problemen tot een minimum gehouden kunnen worden.

Electron-forge was een betere oplossing geweest omdat heel wat configuratie in de beginstadia van het aanmaken van een project kunnen worden overgelaten aan deze bibliotheek. Niet in het minst omdat het een template aanbiedt dat perfect is voor de doeleinden van dit project. Het project bevond zich al in een relatief gevorderd stadium toen dit bekend werd. Een nieuw project aanmaken en alle bestanden migreren was op dit punt geen interessante oplossing.

Asar en Electron-packager zijn volwaardige alternatieven op de gekozen bibliotheek. Asar heeft echter enkele nadelen in verband met het gebruik in samenspraak met Node.js en laat meer verantwoordelijkheden over aan de programmeur. Electron-packager kan net als Electron-builder in de latere fasen van de ontwikkeling worden bijgevoegd in Electron-projecten, maar biedt minder functionaliteiten die direct in het project mee worden verpakt dan Electron-builder.

2.1.11 Andere bibliotheken

In deze paragraaf worden de overige bibliotheken die in het project werden gebruikt besproken. Deze bibliotheken worden gebruikt als ondersteuning om bepaalde functionaliteiten makkelijker in de applicatie in te werken. Zij worden in minder detail besproken dan voorgaande bibliotheken.

2.1.11.1 Redux

Redux is een bibliotheek die ontwikkeld werd om gebruikt te worden als een *single source of truth*[58]. Het doel van Redux is om een globale toestand, zelf een JSON-object, te voorzien dat de volledige applicatie beschrijft. Hierdoor kunnen applicaties gemaakt worden die zich consistent gedragen.

Drie onderdelen liggen aan de basis van Redux: het model (globale toestand), acties en *reducers*. Het model beschrijft alle toestanden van alle onderdelen van de applicatie. Acties beschrijven welke veranderingen er mogen gemaakt worden aan de globale toestand van de applicatie. De reducers zijn functies die een bepaalde actie aanvaarden en uitvoeren op een onderdeel van de globale toestand. Reducers moeten pure functies zijn die enkel een actie en de huidige toestand aanvaarden en de nieuwe toestand teruggeven. Er mogen dus geen neveneffecten[59] worden geproduceerd.

Redux maakt het eenvoudiger om eenduidig de toestand, en de interacties met deze toestand, te beschrijven. Dit wil daarom niet zeggen dat Redux altijd handig is of dat de toestand van de applicatie volledig in Redux beschreven moet worden. In React heeft elke component een eigen toestand en een toestand die van een oudercomponent wordt meegegeven. Als deze toestanden enkel nuttig zijn voor die component, dan is het niet zinvol om deze op te slaan in de Redux-toestand. Waar Redux dan wel een voordeel biedt, is dat het stromen van data niet langer strikt in één richting verloopt (van ouder- naar kindercomponent), maar dat de data los van de interacties tussen componenten komt te staan. Dit is voordelig wanneer data of toestanden beschikbaar moeten zijn voor niet-naburige componenten.

Om het mogelijk te maken voor componenten in de applicatie om te communiceren met de Redux-databank, moet in de hoofdcomponent van de React-applicatie de databank worden meegegeven aan de applicatie door middel van Provider-component[60], weergegeven in figuur 21. Elke component die in contact moet staan met de databank moet de connectfunctie van de bibliotheek importeren in het bestand waar de React-component zich bevindt. Er moeten ook mapperfuncties worden aangemaakt die de onderdelen en acties van de databank in de props – variabelen die van de implementerende component afkomstig zijn – van de React-component plaatsen. Met de connectfunctie worden deze mapperfuncties geïnjecteerd in de React-component. Een voorbeeld hiervan wordt weergegeven in figuur 22. [79, 80]

```
<Provider store={store}/>
      <Applicatie/>
</Provider>
```

21. Provider-component voor Redux-databank

```
function mapStateToProps(state){
    return{
        json:state.json.json,
        fetching:state.json.fetching,
        error:state.json.error
    }
}

function mapDispatchToProps(dispatch){
    return{
        startJSON:() => dispatch(jsonStart()),
        loadJSON:(json) => dispatch(jsonSuccess(json)),
        errorJSON:() => dispatch(jsonError()),
        fetchJSON:(from) => dispatch(fetchJSON(from))
    }
}

export default connect(mapStateToProps, mapDispatchToProps)(Workspace);
```

22. Voorbeeldgebruik van mapperfuncties en connectfunctie

[58] Een concept waarbij alle data slechts eenmaal voorkomt. [115]
[59] Een verandering aanbrengen aan een variabele zonder dat hiervoor een functie werd gebruikt of een gevolg is van de teruggegeven waarde van de functie zelf.
[60] Een component dat over het geheel of een onderdeel van een React-applicatie functionaliteiten aanbiedt.

2.1.11.2 Redux-Thunk

Redux Thunk is een bibliotheek die een tussenstap introduceert tussen de acties en reducers van Redux. Deze tussenstap kan worden gebruikt om een vertraging door te voeren voordat een actie wordt doorgegeven aan een reducer of wachten met het doorgeven van een actie totdat voldaan is aan een bepaalde voorwaarde. Dit is een meerwaarde wanneer Https-requests worden gestuurd naar servers en er gewacht moet worden op een antwoord. Wanneer een antwoord is verkregen, kan deze data dan doorgegeven worden naar Redux-databank.

Een thunk is een functie die zelf een interne functie bevat. In een thunk wordt de interne functie asynchroon uitgevoerd. Het gebruik van thunk komt van pas bij het opvragen van data uit een externe bron, zoals een server, of wanneer er langdurige berekeningen moeten plaatsvinden.

Naast Redux-Thunk bestaat er ook Redux-Saga. Het verschil tussen beide is dat Redux-Thunk toelaat om functies mee te geven die worden uitgevoerd voordat een actie wordt doorgevoerd op de globale toestand van Redux en dat Redux-Saga ES6[61]-generators gebruikt om asynchroon acties uit te voeren. Er zijn geen belangrijke redenen waarvoor Redux-Thunk boven Redux-Saga werd gebruikt. [81, 82]

2.1.11.3 Request

De request-bibliotheek dient om asynchroon Https-bevragingen te doen in JavaScipt. De bibliotheek ondersteunt het gebruik van streamen, Https-authenticatie, OAuth en nog veel meer. Deze bibliotheek is een totaaloplossing voor het opvragen van bij de services van het Qompium. Een belangrijke eigenschap hierbij is dat OAuth automatisch wordt bijgevoegd bij de bevraging. Dit is belangrijk omdat Qompium OAuth gebruikt voor goedkeuring van Https-bevragingen. Request wordt eveneens gebruikt om Https-bevragingen in JavaScript naar de services te sturen. Het gebruik van dezelfde bibliotheek is dan ook handig om code te kunnen hergebruiken en in lijn te zijn met andere applicaties. [87]

2.1.11.4 React-re-resizable

React-re-resizable maakt het mogelijk om React-componenten van grootte te doen veranderen. Deze bibliotheek is ook eenvoudig bij te voegen in een applicatie. De component die gebruikt kan worden door de bibliotheek hoeft enkel de gewenste component te omhullen. Aan de component zijn ook een aantal functionaliteiten verbonden die gebruikt kunnen worden om beperkingen op te stellen of acties waar te nemen. Enkele van de belangrijkste functionaliteiten van deze component zijn: [83]

- Een breedte en een hoogte vastleggen;
- Het vastleggen van de verhouding tussen breedte en hoogte;
- Welke assen verschaalt mogen veranderen;
- Een minimum- en maximumgrootte opleggen;
- Callbackfuncties voor het starten, stoppen en terwijl de grootte verandert.

2.1.11.5 React-Draggable

Voor het thesisproject is het interessant dat weergaven kunnen worden verschoven om zo het dashboard interactiever te maken voor de gebruiker. Hiervoor werd de React-draggable bibliotheek in het project toegevoegd. Met deze bibliotheek kunnen componenten verschuifbaar worden gemaakt enkel en alleen door de component te omhullen met de component van de bibliotheek. Aan deze omhullende component kunnen ook parameters worden meegegeven die onder andere: [84]

- De richting waarin de component mag verschoven worden bepaald;
- Een herkenningspunt waaraan de component zich kan hechten;
- De standaardpositie relatief ten aanzien van waar de component zich normaal gezien zou bevinden;
- De huidige positie;
- Een *grid* dat de grootte van de stappen in elke richting (x-richting of y-richting) vastlegt.
- Callback-functies voor het starten, stoppen en terwijl een versleping plaatsvindt.

[61] EcmaScript 6. De specificatie waar JavaScript zich aan houdt. [104]

2.2 JSON-bestanden visualiseren

De eerste doelstelling van dit project is JSON-bestanden omzetten naar een visuele weergave van dit bestand. Om dit te verwezenlijken werden drie React-componenten gemaakt die deze omzetting op zich nemen. Deze componenten werden zelf ontwikkeld. Er bestaan React-bibliotheken die dit ook doen. Er werd toch verkozen om deze zelf te ontwikkelen omdat de React-bibliotheken ofwel te beperkt waren in hun functionaliteit ofwel te complex om op korte tijd te begrijpen en de gewenste functionaliteiten toe te voegen.

In deze paragraaf worden de belangrijkste onderdelen van de applicatie die betrekking hebben tot het visualiseren van JSON-bestanden behandeld. Hierbij zal worden gekeken naar;

- Hoe JSON-bestanden in de applicatie worden ingeladen;
- Hoe onderdelen van JSON-bestanden hun weg vinden naar andere onderdelen van de applicatie;
- Wat het principes zijn die komen kijken bij het maken van een JSON-visualisatie;
- Hoe deze principes in de praktijk zijn geïmplementeerd.

2.2.1 Redux

Redux speelt een belangrijke rol in het afhandelen en opslaan van gegevens in de applicatie. Een van de grootgebruikers hiervan is de JSON-visualisatie. Er zal hier voornamelijk gekeken worden naar de structuur in de reducers. De acties die aan deze reducers verbonden zijn, zijn doorgaans dragers van gegevens. Ze voeren meestal zelf geen verwerkingen uit op of met de gegevens. De uitzonderingen op deze regel worden toegelicht in paragrafen 2.2.1.1 en 2.2.1.2. De JSON-visualisatie neemt drie van de vier secties van de Redux-databank voor zijn rekening. Deze drie onderdelen zijn:

- Een tokensonderdeel;
- Een JSON-onderdeel;
- Een versleeponderdeel.

2.2.1.1 Tokens

Het tokensonderdeel spreekt weinig tot de verbeelding, maar is vitaal voor het kunnen opvragen van JSON-bestanden van de services van Qompium. Zij werken namelijk OAuth om connectie te kunnen maken met hun services. Om deze connectie te kunnen maken, zijn een emailadres, passwoord en consumersleutel nodig. De consumersleutel is om de applicatie te kunnen authentiseren. Voor elke vraag naar gegevens van de services wordt een asynchrone request gemaakt. Deze request wordt gemaakt als gevolg van het uitvoeren van een Redux-actie. Omdat Redux uit zichzelf niet overweg kan met asynchrone acties, komt hier de Redux-Thunk *middleware*[62] tussenbeide. Deze zorgt ervoor dat deze request correct worden afgehandeld.

Heel het opslag- en interactiegebeuren met de tokenstoestand in de Redux-databank vindt plaats in de tokensreducer. Het opslaan van gegevens in dit onderdeel van de Redux-databank is direct. Alle gegevens die een actie doorgeeft, worden zonder aanpassing in de databank opgeslaan.

In deze reducer worden volgende parameters bijgehouden:

- Fetching;
- Fetched;
- Token;
- Token_secret;
- Error.

Fetching, fetched en error zijn parameters die gebruikt worden door de UI om feedback te geven aan gebruikers. De parameters Fetched en Fetching worden gebruikt om op te kunnen weten of de UI moet aangepast worden om de gebruiker te laten weten of de opvraag van gegevens nog steeds bezig is of niet. Voor de JSON-visualisatie zijn echter de token en token_secret belangrijk. Deze twee tokens zorgen ervoor dat verdere requests van JSON-bestanden van de services van Qompium mogelijk zijn.

[62] Middleware een softwarelaag dat zich tussen de applicatie en de *operating system* bevindt. [116]

2.2.1.2 JSON

Het JSON-onderdeel van de Redux-databank is gelijkaardig in gedrag aan het tokensonderdeel. Dit is omdat het JSON-onderdeel ook asynchroon verkregen data bijhoudt. De afhandeling hiervan kan dus op een gelijkaardige manier aan dat van de tokens gedaan worden. Het enige bestandsdeel van deze reducer dat inhoudelijk van belang is, is de parameter die de ontvangen JSON opslaat. Dit is de parameter die gebruikt wordt door de componenten die JSON-visualisaties maken. Dit onderdeel telt wel minder parameters dan het tokensonderdeel:

- Fetching;
- JSON;
- Error.

Er wordt hier geen gebruik gemaakt van een fetched-status. Een JSON wordt ontvangen of niet. Dit wordt duidelijk gemaakt in de applicatie door de inhoud van de JSON- en de error-parameter.

De JSON-parameter kan op twee manieren van gegevens voorzien worden. De eerste manier is door een URL[63] op te geven. Hierbij wordt dan naar deze URL een bericht gestuurd samen met de verkregen tokens. Als deze URL deze tokens accepteert, dan komt een positief antwoord terug en wordt de verkregen JSON opgeslagen in de Redux-databank. Dit proces maakt gebruik van de alle parameters van de JSON-reducer. De andere manier is om een bestand van op de computer in te lezen. Deze interactie tussen computer en applicatie wordt ondersteund door Node.js. Met uitlezen van bestanden op de computer worden de gegevens direct in de JSON-parameter geplaatst.

2.2.1.3 Verslepen

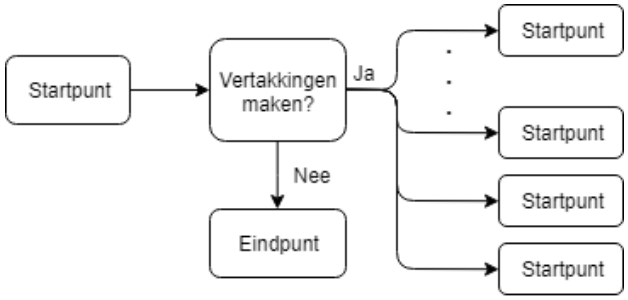
De derde en laatste reducer is die voor het bijhouden van de toestanden van versleepacties. De parameters van dit onderdeel van de Redux-databank zorgen ervoor dat versleepacties met de JSON-visualisatie correct verlopen. Hiervoor bestaan er drie parameters:

- Dragging;
- Was_dragging;
- Data.

De eerste twee parameters houden de huidige toestand en verleden toestand van de versleepactie bij. De data-parameter is de buffer die de ontvangen gegevens tijdelijk zal opslaan. Het gebruik van deze parameters in de applicatie wordt verder toegelicht in paragraaf 2.2.3.1.

2.2.2 Het principe

JSON, zoals uitgelegd in paragraaf 2.1.1, is een datastructuur waarin het mogelijk is om gegevens genest op te slaan. Dit is analoog aan het wortelstelsel van een boom. Het JSON-dataformaat gebruikt dan ook een boomstructuur om gegevens op te slaan. Beginnende vanaf de stam zijn er enkele vertakkingen. Sommige van deze vertakkingen stoppen zonder zijtakken te maken. Andere vertakkingen hebben zelf ook vertakkingen. Dit vertakkingsproces wordt uitgebeeld in figuur 23.



23. Blokschema JSON

Zoals in bovenstaande figuur te zien is, kan een JSON opgedeeld worden in drie onderdelen:

- Een startpunt;

[63] Uniform Resource Locator. Een adres dat naar een service leidt. [117]

- Een transitiepunt;
- Een eindpunt.

Deze onderdelen worden ook weerspiegeld in de componenten die werden gemaakt om de JSON om te zetten naar een interactieve visuele weergave van zichzelf. In tegenstelling tot het blokschema in figuur 23 zullen de componenten een bestaande JSON afgelopen, niet aangemaakt. Hierbij wordt begonnen vanuit een startpunt dat nagaat hoeveel vertakkingen en eindpunten deze heeft. Een eindpunt is alles dat geen JavaScript-object is. Als een vertakking wordt gevonden, dan wordt een component aangemaakt dat het vertakte gedeelte meekrijgt. Deze component kan dan weer verdere vertakkingen zoeken en cyclus verderzetten. Dit proces kan blijven doorgaan totdat alle eindpunten zijn gevonden.

Het schema in figuur 23 laat ook blijken dat deze makkelijk functioneel te doorlopen is. Het begint namelijk steeds vanuit een startpunt waarvan enkel geweten is dat het ofwel een JavaScript-object is ofwel een eindpunt. Elk JavaScript-object kan dan terug als een startpunt genomen worden. Als alleen dit principe zou worden toegepast dan wordt elke JSON in een keer volledig doorlopen. Het gevolg hiervan zou zijn dat in de applicatie elke JSON dat wordt opgehaald in volledig wordt omgezet in een visualisatie van de eindpunten. Dit is niet wenselijk omdat een gebruiker niet per se nood heeft om alles uit een JSON in eenmaal te kunnen zien. Een ander gevolg zou zijn dat deze visualisatie enorm veel plaats zou innemen. Elke vertakking wordt namelijk omsloten door rechte haken of accolades. Wenselijker zou zijn dat bij elke vertakking, buiten het effectieve startpunt van de JSON, de visualisering van de JSON wordt stopgezet. Hierdoor kan de gebruiker zelf iteratief door de JSON zoeken.

De mogelijkheid als gebruiker iteratief door een JSON te zoeken, wordt mogelijk gemaakt door de drie componenten. Het startpunt van de JSON wordt afgehandeld door een startcomponent die over deze bovenste laag zal itereren. Afhankelijk of hier JavaScript-objecten worden gevonden of niet worden respectievelijk transitiecomponenten of eindcomponenten aangemaakt. Een transitiecomponent geeft dan enkel weer welk type JavaScript-object het is. Deze component dient als een tijdelijke plaatshouder totdat de gebruiker aangeeft de onderliggende gegevens te willen weten. Een eindcomponent geeft de effectieve data weer.

In hoofdstuk 1 werd reeds aangegeven dat gegevens uit een JSON-visualisatie via drag en drop het aanmaken van visualisaties moet starten. Dit kan op twee manieren:

- De data uit de JSON-visualisatie wordt doorgegeven;
- De plaats in de JSON waar de data gevonden kan worden, wordt doorgegeven.

Voor de doeleinden van dit project werd voor de tweede optie gekozen. Door de plaats in de JSON waar de gegevens zich bevinden door te geven, kunnen deze gegevens altijd worden opgehaald. Dit is belangrijk voor de visualisaties omdat zij telkens wanneer de JSON die wordt ingeladen in de applicatie verandert, zij nieuwe gegevens moeten voorstellen. Deze nieuwe gegevens zijn afhankelijk van het de locatie in de JSON.

Gegevenslocaties in een JSON worden opgeslagen in een ‘sleutelpad’. Een sleutelpad is een reeks sleutels[64] dat moet worden afgegaan om op een bepaalde locatie in een JSON te geraken. Een sleutelpad wordt in de applicatie bijgehouden als een string waarbij tussen twee opeenvolgende sleutels een dubbelepunt is geplaatst. Dit principe wordt uitgelegd aan de hand van het JSON-voorbeeld in figuur 24.

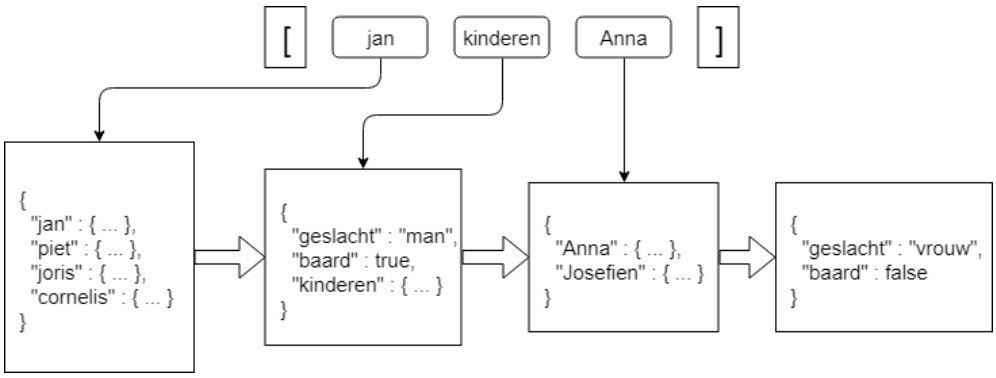
[64] Een sleutel is een index in een JavaScript-object. Voor gewone objecten – aangegeven door een omsluiting met accolades – hebben als index doorgaans strings. Lijsten hebben een getal als index.

```
{
	"jan" : {
		"geslacht" : "man",
		"baard" : true,
		"kinderen" : {
			"Anna" : {
				"geslacht" : "vrouw",
				"baard" : false
			},
			"Josefien" : { ... }
		}
	},
	"piet" : [ ... ],
	"joris" : [ ... ],
	"cornelis" : [ ... ]
}
```

24. Voorbeeld JSON

Stel dat een gebruiker geïnteresseerd is om alle informatie van over Anna te visualiseren, dan moet dit op een of andere manier duidelijk worden gemaakt in de rest van de applicatie dat het gaat om het kind van Jan dat Anna heet. Het is geweten dat binnen elk JavaScript-object enkel unieke sleutels mag bevatten. Als dit uitgebreid wordt naar geneste JSON, dan kan gesteld worden dat elke combinatie van sleutels die naar een bepaald punt in die JSON wijzen zelf ook uniek is. In dit geval zou het sleutelpad dat naar Anna verwijst “jan:kinderen:Anna” zijn. Het omvormen van de sleutelpaden naar strings is niet noodzakelijk, maar maakt het eenvoudiger om te interpreteren en op te slaan in de applicatie. Daarnaast is het eenvoudig om deze string via een regular expression terug om te zetten naar een lijst van strings.

Het proces om Anna terug te vinden uit de JSON is redelijk eenvoudig. Eerst wordt het sleutelpad met een regular expression opgebroken op in verschillende strings. Deze splitsingen worden gemaakt op elk dubbelepunt. Vanaf dit punt is het slechts een kwestie om over de lijst van sleutels te itereren. Bij de eerste iteratiestap wordt met de overeenstemmende sleutel een deel van de data uit de JSON gehaald. Dit deel wordt in een tijdelijke variabele opgeslagen. Vanaf dan wordt hetzelfde proces doorlopen met de tijdelijke variabele in plaats van de JSON totdat de laatste sleutel is bereikt. De gewenste data is dan gevonden. Figuur 25 geeft het weer hoe dit proces in de tijd verloopt.



25. Iteratie over een JSON met een sleutelpad

2.2.3 Praktisch

De implementatie van de principes die werden besproken, is toch net wat anders in de praktijk dan de principes laten uitschijnen. Eerst worden de belangrijkste werkingsprincipes van de JSON-visualisatie toegelicht. Daarna wordt de implementatie om JSON’s om te zetten in een serie van componenten behandeld. De individuele componenten worden niet apart besproken.

2.2.3.1 Algemene werking

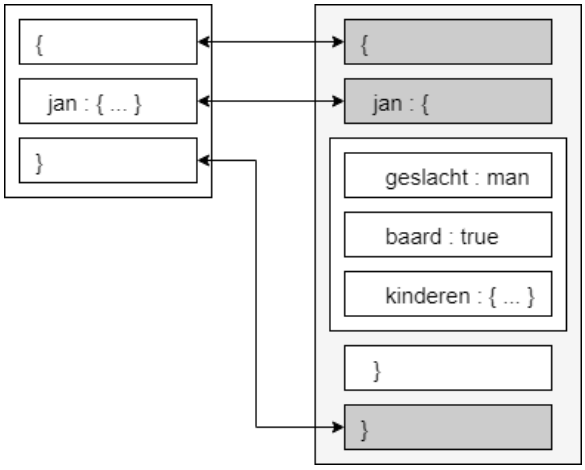
Tot nu toe is bekeken hoe een JSON kan worden doorlopen. Hierbij werd ook het principe van de sleutelpaden uitgelegd. Hier wordt uitgelegd hoe deze twee principes werden gebruik om een JSON om te zetten in een visuele representatie. Het JSON-voorbeeld uit figuur 24 wordt hiervoor terug gebruikt. In de figuren in deze

paragraaf wordt vaak verwezen naar de UI die voortvloeit uit de JSON-componenten. Dit is omdat in een redelijk aantal componenten in de applicatie de UI en verwerking voor de UI samen aanwezig zijn. In het geval van de JSON-visualisatie is dit het geval.

Er zijn drie eigenschappen die een visualisatie van een JSON heeft:

- Een onderdeel van de JSON moet open- en sluitbaar zijn;
- Onderdelen van de JSON moeten selecteerbaar zijn;
- Onderdelen van de JSON moeten versleepbaar zijn.

Ten eerste is er het open- en sluitbaar zijn van onderdelen van een JSON. In figuur 26 wordt weergegeven hoe de componenten worden toegevoegd wanneer de data in ‘jan’ wordt geopend. Om plaats te besparen worden andere sleutels buiten beschouwing gelaten. Tussen elke laag van componenten wordt ruimte gelaten. Dit is om duidelijker weer te kunnen geven hoe de verschillende component zich tot elkaar verhouden. In werkelijkheid worden hier geen spaties gelaten. Elke component is ook even breed dan alle andere componenten.

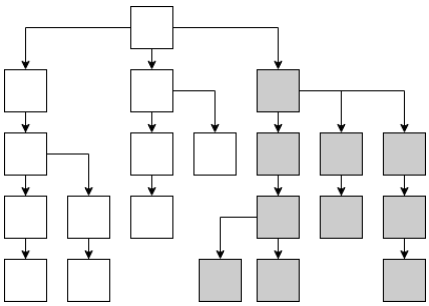


26. Open- en sluitmechanisme

In de figuur is het te zien dat bij het opvragen van de gegevens uit ‘jan’ de bestaande onderdelen worden behouden. Tussen de component dat ‘jan’ voorstelt en de component dat het einde van het object voorstelt, komt een nieuwe laag te staan. In deze laag worden de sleutels van ‘jan’ omgezet in een reeks van componenten. Na de nieuwe laag wordt ook een sluittag bijgevoegd. Zoals aangegeven door de dubbele pijlen werkt dit proces in twee richtingen.

Dan wordt gekomen aan het tweede item: onderdelen moeten selecteerbaar zijn. Wanneer een gebruiker ‘jan’ wil selecteren, heeft deze enkel op de component die ‘jan’ voorstelt te klikken. Maar wanneer ‘jan’ aangeklikt is, is het niet de bedoeling dat de kinderen van jan ook geselecteerd worden. Deze maken al deel uit van ‘jan’. Hier stelt zich een probleem voor.

De oplossing voor dit probleem is door een elke kindercomponent bewust te maken van de selecteerstatus van de oudercomponent. Dit over de hele keten die een eindcomponent verbindt met de allereerste startcomponent. Dit levert dan een watervalsysteem op. Elke component die een of meer kindercomponenten bezit, geeft zijn selecteerstatus door aan die kindercomponenten. De kindercomponenten ontvangen deze status, maar passen hun eigen selecteerstatus niet aan. Het tonen van de selecteerstatus wordt bepaald door een logische AND tussen de selecteerstatus van de oudercomponent en die van de huidige component. Wanneer de selecteerstatus van een oudercomponent de waarde true heeft, dan mogen de kindercomponenten hun eigen selecteerstatus niet meer aanpassen. Tegelijkertijd wordt hun eigen selecteerstatus ook op true gezet. Deze nieuwe selecteerstatus van de kindercomponent wordt op zijn beurt weer doorgegeven aan zijn kindercomponenten. Dit proces loopt door totdat alle eindcomponenten die in directe lijn verbonden zijn met de oudercomponent die werd geselecteerd hun selectiestatus hebben bijgewerkt. Figuur 27 geeft hier een weergave van. In deze figuur zijn de donker getinte blokken geselecteerd en de licht getinte blokken niet geselecteerd.



27. Watervalsysteem voor selecteerstatussen

Uiteindelijk wordt dan gekomen aan de laatste eigenschap: versleepbaar zijn. Deze eigenschap duidt niet op het kunnen verplaatsen van de onderdelen van de JSON en deze verspreiden over het beeldscherm. Het duidt op het overbrengen van gegevens uit de onderdelen van de JSON naar andere componenten in de applicatie die deze versleepte gegevens verder zullen gebruiken om te transformeren in visualisaties.

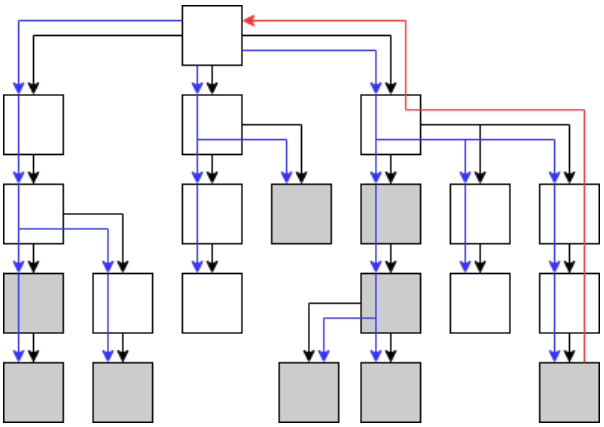
In de eerste plaats moet de levenscyclus van een versleepactie kunnen opgevolgd worden. Gelukkig hebben HTML-elementen hier ingebouwde functies voor: onDragStart, onDrag, onDrop, onDragEnd. Van deze eventlisteners worden enkel onDragStart en onDragEnd geïmplementeerd in de JSON-visualisatie. Waarom de onDrag-eventlistener niet wordt gebruikt, wordt later meegedeeld. Hier bevinden zich namelijk alle componenten die versleept kunnen worden. De onDrop-evenlistener wordt geïmplementeerd daar waar de data ontvangen hoort te worden.

Er zijn hier echter wel een aantal problemen die eerst opgelost moeten worden, zoals:

1. Hoe weten componenten dat ze hun sleutelpaden hebben doorgegeven?
2. Wat als er meerdere componenten geselecteerd zijn?
3. Wat als een versleepactie wordt uitgevoerd met een niet-geselecteerde component?

Het eerste en tweede probleem delen dezelfde oplossing. In figuur 27 is te zien dat alle componenten in rechte lijn verbonden zijn met de eerste startcomponent, maar onderliggende componenten staan niet in directe verbinding met elkaar. In zo een boomstructuur is communicatie tussen de stam en de eindpunten eenvoudig, maar communicatie tussen eindpunten onderling niet. Omdat de JSON-visualisatie zo goed mogelijk probeert de JSON-structuur op te volgen, is dit ook de structuur die de componenten zullen aannemen.

De React-manier om dit aan te pakken, zou zijn om een callback-keten te maken tussen alle eindcomponenten en de bovenste startcomponent. Bij het starten van versleepactie zou dan de versleepte component dan een signaal door de callback-keten naar de bovenste startcomponent sturen. Wanneer deze dit ontvangt, wordt dan een impulssignaal naar alle componenten gestuurd om aan te geven dat ze hun gegevens moeten doorgeven. Het gebruik van een impulssignaal is een vereiste omdat anders bij elke component voortdurend zijn gegevens zou doorgeven in plaats van maar één keer. Dit systeem wordt uitgebeeld in figuur 28, waarbij de grijze blokken geselecteerd zijn.



28. Callback-keten (rood); Gegevensopvraging (blauw)

In de applicatie wordt gebruik gemaakt van Redux. Hierdoor werd het mogelijk om callback-ketens achterwege te laten en elke component die een sleutelpad heeft door te verbinden met de Redux-databank. In deze databank

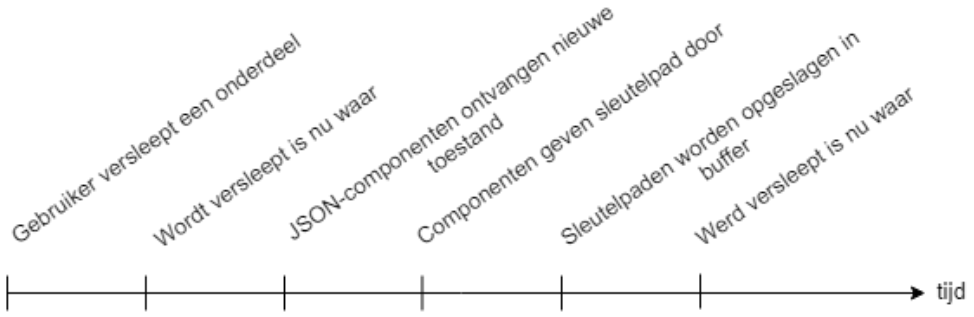
kan dan worden bijgehouden of een versleepactie aan de gang is. Dit alleen vervult de nood niet om een impuls te geven aan de componenten dat ervoor zorgt dat elke component slechts eenmaal zijn gegevens doorgeeft. Toch als ze geselecteerd zijn.

Om dit te kunnen bepalen, is een extra parameter nodig: of een versleepactie werd uitgevoerd. Door de huidige en voormalige toestand van een versleepactie bij te houden, kan een ogenblik bepaald worden waarop geselecteerde componenten van een JSON-visualisatie hun gegevens moeten doorgeven. Wanneer het doorgeven van gegevens moet plaatsvinden, wordt weergeven in tabel 3.

Wordt versleept	Werd versleept	Doorgeven van gegevens
False	False	False
False	True	False
True	False	True
True	True	False

3. Waarheidstabel voor doorgeven van gegevens

De applicatie kan dus bepalen om zijn gegevens door te geven wanneer er de versleepactie net werd gestart. Eerder werd al eens vermeld dat er geen gebruik wordt gemaakt van de onDrag-eventlistener. Deze wordt niet gebruikt omdat het updaten van de Redux-databank en van de componenten wat tijd nodig heeft. Uit ervaring is ook gebleken dat deze niet synchroon aan elkaar updaten. In de applicatie wordt niet enkel gerekend op Redux om de toestand van de applicatie te onderhouden, maar worden ook de interne toestanden van React gebruikt. Hierdoor wordt er pas een update naar de Redux-databank gestuurd wanneer de componenten zichzelf hebben kunnen vernieuwen. Een gevolg hiervan is ook dat alle componenten die met dit deel van de Redux-databank in contact staan de nieuwe toestand kunnen raadplegen. Wanneer de componenten zijn geüpdatet, dan wordt de status van de versleeptoestand aangepast. Tegen de tijd dat dit verlopen is hebben alle componenten hun sleutelpaden kunnen doorgeven. Het verloop wordt vereenvoudigd weergegeven in figuur 29.



29. Volgorde van de gebeurtenissen bij het starten van een versleepactie

Uiteindelijk blijft dan nog het laatste probleem over: “Wat als een niet-geselecteerde component wordt versleept?”. Dan zou eenvoudigweg de sleepactie genegeerd kunnen worden. Maar het is interessanter om ook in dit geval een versleepactie mogelijk te maken. Het namelijk omslachtig om iedere keer een JSON-component aan te duiden wanneer maar één datapunt gewenst is. Hiervoor wordt er een onderscheid gemaakt tussen het afhandelen van meerdere geselecteerde componenten en het verslepen van een enkele component. In tegenstelling tot het verslepen van geselecteerde componenten wordt de huidige en verleden toestand van het verslepen niet bijgehouden. In plaats daarvan wordt het sleutelpad in een keer doorgegeven naar de Redux-databank.

2.2.3.2 JSON naar componenten

Voor het omzetten van JSON’s naar componenten bestaat een enkele functie. Deze functie bevindt zich in de startcomponent. Hier kan deze functie altijd aan de JSON zoals deze werd doorgegeven bij de startcomponent. Daarnaast kan deze dan ook aan alle onderliggende componenten worden doorgegeven. Dit eindigt in callback-keten van elke component naar deze functie. Naast directe toegang tot de JSON ontvangt deze functie ook nog enkele parameters van de componenten die ze oproepen:

- Een ID;

- Een laagnummer;
- Een selectie-status.

In de vorige paragraaf werd de nood voor het doorgeven van de selectie-status reeds toegelicht. Het ID dat deze functie ontvangt is het sleutelpad dat moet doorlopen worden door de JSON om aan de nieuwe gegevens te komen. Deze nieuwe gegevens worden omgezet in componenten die dan worden teruggegeven aan de component die deze functie opriep. Het laagnummer is van belang om in elke component te kunnen bepalen hoeveel deze moet inspringen.

Bij het aanmaken van nieuwe componenten wordt in sommige gevallen niet de effectieve gegevens doorgegeven. In het geval dat een JavaScript-object wordt gevonden, dan wordt een substituut meegegeven. Dit substituut dient enkel om aan de gebruiker duidelijk te maken dat een JavaScript-object werd gevonden. Het substituut is "[…]" of "{…}". In alle andere gevallen worden de gegevens behouden. Een laatste bewerkingsstap die de gevonden gegevens doorgaat, is wanneer een lijst wordt teruggekeerd met meer dan 100 gegevens. Deze lijst wordt dan ingeperkt tot 100 datapunten met als laatste "…" toegevoegd om aan te geven dat er meer gegevens zijn. Uiteindelijk wordt het bekomen resultaat omgezet in een reeks van componenten. De code van deze functie wordt weergegeven in figuur 30.

```
const preMap = ({id, layer, selected}) => {
        layer = layer >= 2 ? layer : 2;
        let list = [];
        let data = identifier(json, id);
        let keys = Object.keys(data);
        let max = keys.length;

        if(max > 100 && data instanceof Array){
            max = 100;
            data.splice(0,98);
            data.push('...')
        }

        for(let i = 0; i < max; i++){
            let e = keys[i];
            if(typeof data[e] === 'object' && data[e] !== null){
               list = list.concat(<JList key={i} id={e} parent_id={id}
selected={selected} data={data[e]} preMap={preMap} layer={layer}/>)
            }
            else{
                list = list.concat(<JListItem key={i} id={e} parent_id={id}
selected={selected} data={data[e]} layer={layer}/>)
            }
        }
        return list;
    }
```

30. Functie die JSON omzet in componenten

2.3 Uitbreidbaarheid

Een van de belangrijkste doelstellingen is het uitbreidbaar maken van de applicatie. Deze uitbreidbaarheid is voornamelijk gericht op het flexibel kunnen toevoegen van visualisaties aan de applicatie. Dit moet op zo een manier ingevuld zijn dat met een minimum aan aanpassingen in de applicatiecode nieuwe visualisaties direct beschikbaar worden. Uitbreidbaarheid voor visualisaties die zeer uiteenlopende karakteristieken kunnen hebben, is een complex probleem dat op verschillende vlakken moet aangepakt worden.

2.3.1 Datatypes

JavaScript is een programmeertaal die dynamische typering gebruikt voor de gegevens die erin gebruikt worden. Dit maakt dat van tevoren niet kan worden uitgemaakt welk datatype een bepaalde parameter zal hebben. Dit maakt dat gegevens altijd moeten nagekeken worden als het type van de gegevens van belang is. In het eindproduct dat werd ontwikkeld in functie van de masterproef is dit regelmatig nodig. Dit komt doordat er nooit op voorhand kan geweten zijn welke datatypes de gebruiker zal kiezen om te laten visualiseren. De datatypes die

JavaScript kent, komen echter tekort om een flexibel en uitbreidbaar systeem voor visualisaties te maken. Daarom werd een systeem ontwikkeld dat enkele algemene vormen van data kan achterhalen.

2.3.1.1 Algemeen

Vooraleer een reeks van datatypes kon worden bepaald, moest eerst gekeken worden welke datatypes interessant zouden zijn voor visualisaties. Een visualisatie kan namelijk eender wat zijn. Het kan een grafiek zijn, maar ook een tabel of een wegenkaart. Elk van deze visualisaties verwacht een andere vorm van data. Gegevens die bruikbaar zijn voor een grafiek zijn daarom niet bruikbaar voor een wegenkaart of andersom.

Omdat de hoeveelheid verschijningsvormen van gegevens die gebruikt kunnen worden voor visualisaties enorm breed is, werd verkozen om het datatypesysteem algemeen te houden. Met het systeem zou het dan de bedoeling zijn om algemene structuren op te geven waaraan gegevens kunnen voldoen. Hiermee kan een groot aandeel aan visualisaties worden ondersteund. Voor een groot aantal visualisaties is het onbelangrijk of een bepaalde sleutel in een bepaald object aanwezig is of niet.

Het ontwikkelde systeem werd beïnvloed door het typesysteem van TypeScript. In het gemaakte systeem worden volgende types ondersteund:

- Getallen;
- Strings;
- Booleans;
- Any;
- Lijsten;
- Objecten.

Indien dit alles was, dan zou het hier niet besproken worden. Het systeem maakt een onderscheid met TypeScript wat betreft objecten. In TypeScript kunnen structuren met objecten worden vastgelegd aan de hand van interfaces. In deze interfaces kan worden vastgelegd welke sleutels er in een object mogen voorkomen. Daarnaast kan voor elke sleutel ook nog een type worden vastgelegd. In het systeem dat werd ontwikkeld werd gestreefd naar algemeenheid. Hierdoor werd afgeweken van de interface-manier van TypeScript. In tegendeel, objecten worden tot dezelfde sfeer als lijsten gebracht. Objecten worden in het systeem aanzien als itereerbare datatypes waarvan de sleutels niet uitmaken.

Objecten losgekoppeld houden van hun sleutels was belangrijk omdat visualisaties niet noodzakelijk geïnteresseerd zijn in welke sleutels er in een object voorkomt, maar eerder welke datatypes erin schuilgaan. Stel bijvoorbeeld een object dat bestaat uit een aantal lijsten van getallen. In dit geval kan het interessant zijn om de lijsten uit dit object te gebruiken om een visualisatie te maken dan aandacht te besteden aan welke sleutels dit object heeft. Dit maakt het eveneens eenvoudiger voor de gebruiker om data door te geven om te visualiseren aangezien in soortgelijke gevallen dan niet meer nodig is om elke lijst apart aan te duiden.

Door objecten te gebruiken op een gelijkaardige manier als lijsten wordt het mogelijk om een oneindig aantal combinaties van de bovenstaande ondersteunde types te creëren. In de applicatie wordt heel de gevonden datastructuur doorlopen om het datatype te kunnen achterhalen. Hierbij wordt het type van de gegevens ofwel bepaald voor heel de datastructuur ofwel beperkt tot op het laatste dieptepunt waar de gegevens nog van dezelfde aard waren. Een subset van de mogelijke datatypes wordt getoond in figuur 31. Om de term dieptepunt beter te illustreren wordt terugverwezen naar figuur 27. Elke rij van blokken in deze figuur stelt een dieptepunt voor. Het dieptepunt is dus niet meer dan het minimaal aantal stappen dat moet gezet worden om van een bepaald gegeven van een JSON tot het startpunt te komen.

```
String, number, boolean, any, string[], number[], boolean[], any[], string{}, number{},
boolean{}, any{}, string[][], number[][], boolean[][], any[][], string[]{}, number[]{},
boolean[]{}, any[]{}, string{}[], number{}[], boolean{}[], any{}[], string{}{},
number{}{}, boolean{}{}, any{}{}
```

31. Een aandeel van de datatypes

Het datatype dat zich voor de lijst of object indicator bevindt, duidt aan welk datatype uitsluitend aanwezig is in het laagst gelegen JavaScript-object dat gevonden werd. In het voorbeeld van het object dat bestaat uit lijsten van getallen zou dit datatype bepaald worden als: “number{}[]”.

2.3.1.2 Implementatie

In de applicatie zijn de gegevens die omgezet moeten worden in datatypes afkomstig van elders in de applicatie, maar de vorm waarin ze wordt gegeven is steeds dezelfde. Deze vorm is een object met als sleutels de sleutelpaden die verwijzen naar de locatie in de JSON waar de gegevens vandaan zijn gehaald. Het principe hierachter werd uitgelegd in paragraaf 2.2.2. De functies die gegevens omzetten in datatypes aanvaarden echter dit object niet, maar wel de gegevens die erin zijn opgeslagen. Het zijn dan ook deze gegevens waarvan het gewenst is het datatype te weten. Het proces om een gegeven om te vormen in een typebepaling bestaat uit drie functies:

1. Een functie dat voor elk gegeven op elk dieptepunt een object opslaat met de diepte waar dit gegeven werd gevonden en het proto-datatype[65] van dit gegeven;
2. Een functie die deze objecten reduceert tot een geldig datatype voor elke diepte;
3. Een functie die de gereduceerde gegevens omzet naar een gestandaardiseerd datatype.

De eerste functie is een functie dat recursief over de gegevens zal itereren van de gegevens die het werd gegeven. Voor het gegeven dat aan deze functie werd meegegeven wordt eerst bepaald welk proto-datatype kan bepaald worden aan de hand van het typebepalingssysteem van JavaScript zelf. Deze bepaling wordt door een hulpfunctie gedaan. Na het bepalen van het JavaScript-type wordt het volgende proces doorlopen:

- Als het JavaScript-type geen object en geen undefined is, dan wordt het JavaScript-type teruggeven;
- Als het gegeven een lijst is – dat een andere methode vraagt dan om het JavaScript-type te achterhalen – dan wordt '[]' teruggegeven;
- Als het object kan geïnterpreteerd worden als een true, dan is wordt '{}' teruggeven door de functie. Dit is omdat elke andere waarde dat als een true kan geïnterpreteerd worden op dit punt al een return zou hebben gehad;
- Uiteindelijk wordt een 'any' teruggegeven. Een any markeert hier dat het gegeven een null of een undefined is.

Nadat dit proto-datatype is ontvangen wordt het in objectvorm opgeslagen in een lijst die zich buiten de functie bevindt. Daarna wordt nog gecontroleerd of het datatype een object of een lijst was. Als dit het geval is, dan wordt over de data die aan de functie werd gegevens geïtereerd en roept voor elke index in het JavaScript-object de functie zichzelf op. Hierbij wordt de data op de index van het JavaScript-object, de diepte geïncrementeerd en de referentie naar de lijst die als accumulator dient meegegeven. Dit proces herhaalt zich totdat het datatype van elk gegeven in de data is achterhaald en opgeslagen samen met de overeenstemmende diepte in de accumulator. De functie en hulpfunctie zijn weergegeven in figuur 32.

[65] Een primitief datatype dat op zichzelf of in combinatie met andere een datatype opmaakt.

```
const type = ( unknown : object ) : string => {
       const tp = typeof unknown;
       if ( tp !== "object" && tp !== 'undefined' ) {
              return tp;
       } else if ( unknown instanceof Array ) {
         return '[]';
    } else if ( unknown ) {
         return '{}';
    } else {
         return 'any';
    }
}

const types = ( unknown : object, num : number, store : Array<Store> ) : void => {
       const tp = type( unknown );

    store.push({
              "layer" : num,
              "type" : tp
       });

    if ( tp === '{}' || tp === '[]' ) {
              const keys = Object.keys(unknown);
              const len = keys.length;

              for ( let i = 0; i < len; i++ ) {
                     const key = keys[i];
                     types( unknown[key], num + 1, store );
              }
       }
}
```

32. Datatypebepaling op elk dieptepunt

Op dit punt in het proces is met de data die werd ontvangen een lijst van objecten gemaakt. Deze objecten houden elk het proto-datatype en de diepte waarop het zich bevond bij. De volgende stap in het proces is om deze lijst van objecten om te zetten in een lijst met proto-datatypes.

Elke index van deze lijst verwijst naar de diepte van de gegevens in de ontvangen data. Het proto-datatype kan op dit punt gemuteerd worden. Wanneer elk proto-datatype van een bepaalde diepte hetzelfde is, dan wordt het opgeslagen in de lijst. Als dit niet het geval is, dan wordt het proto-datatype niet behouden. Het datatype any wordt in de plaats opgeslagen. De functie die dit uitvoert, is weergegeven in figuur 33.

```
const condense = ( str : Array<Store> ) :
       const len = str.length;
       let dense : Array<string> = [];

       for (let i = 0; i < len; i++ ) {
              const object = str[i];
              const lay = object['layer'];
              const tp = object['type'];
              if ( !dense[lay] ) {
                     dense[lay] = tp;
              } else if ( dense[lay] !== tp
                     dense[lay] = "any";
              }
    }
       return dense;
}
```

33. Reductie naar een lijst van proto-datatypes per diepte

De laatste stap in het proces is om de bekomen lijst van proto-datatypes om te zetten naar een datatype. Hierbij wordt over de lijst geïtereerd. Als het proto-datatype noch ‘{}’ is, noch ‘[]’ dan wordt het proto-datatype vooraan een accumulator geplaatst en het doorlopen van de lijst gestopt. De accumulator is zelf een string. Het doorlopen van de lijst wordt gestopt omdat als het proto-datatype geen JavaScript-object is, dan kan geen uitspraak meer gemaakt worden over de proto-datatypes die daarop volgen. Dit is voornamelijk bedoeld om het vormen van het

datatype te stoppen op punt dat een any wordt gevonden. Als het datatype wel '{}' of '[]' is, dan wordt dit toegevoegd aan de accumulator. Het datatype dat gevormd wordt dankzij deze functie is het finale datatype van de data. De functie implementatie van deze functie is te zien in figuur 34.

```
const makeType = ( condensed : Array<string> ) => {
	const len = condensed.length;
	let acc = '';

	for ( let i = 0; i < len; i++ ) {
		const con = condensed[i];
		if ( con !== '[]' && con !== '{}' ) {
			acc = con.concat( acc );
			break;
		} else {
			acc = acc.concat( con );
		}
	}
	return acc;
}
```

34. Datatypefunctie

2.3.2 Specificatiemodel

Het specificatiemodel is een model dat voor elke weergave een beschrijving geeft van deze weergave. Dit model wordt opgemaakt aan de hand van een JSON. Deze beschrijving kent drie onderdelen:

- Eigenschappen;
- Beperkingen;
- Functionaliteiten.

De eigenschappen en beperkingen van de weergave kunnen in een JSON-bestand worden opgeslagen. De eigenschappen zijn voornamelijk beschrijvingen van hoe de weergave zich moet gedragen in de applicatie. De beperkingen zijn beschrijvingen van het kader waarin data als geldig wordt aanschouwd voor een gegeven weergave. Hierdoor is het mogelijk om op voorhand te bepalen of een bepaalde dataset gebruikt kan worden voor een gegeven weergave. Het laatste onderdeel zijn de functionaliteiten. Deze functionaliteiten worden niet in de JSON toegevoegd omdat dit componenten zijn die in de applicatie zullen gebruikt worden. Een voorbeeld van zo een specificatiemodel wordt gegeven in figuur 35.

```
{
    "name" : "Radar",
    "settings" : {
        "show" : true,
        "disable" : false
    },
    "defaultSize" : {
        "width" : 500,
        "height" : 250
    },
    "acceptedDataTypes" : [
        "string[]",
        "number[]"
    ],
    "resizeEnabled" : true,
    "constraints" : {
        "groups" : [
            ["string[]", "number[]"],
            ["string[]"]
        ],
        "occurrences" : {
            "range" : {
                "max" : {
                    "string[]" : 1,
                    "number[]" : 4
                }
            }
        },
        "dataLength" : {
            "sameLength" : [true],
            "range" : {
                "max" : {
                    "string[]" : [15, 30],
                    "number[]" : [15]
                },
                "min" : {
                    "string[]" : [3, 1],
                    "number[]" : [3]
                }
            }
        }
    }
}
```

35. Specificatiemodel van een Radar-grafiek

2.3.2.1 Eigenschappen

Het onderdeel van de eigenschappen van het specificatiemodel bestaat uit een aantal bestanddelen:

- Een naam;
- De *settings*;
- De standaargrootte;
- De geaccepteerde datatypes;
- Het herschaalbaar zijn.

De naam in de eigenschappen is een string dat in de applicatie zal worden gebruikt om te achterhalen welk specificatiemodel gekozen werd door de gebruiker. De keuze om de naam van de weergave in het specificatiemodel op te slaan is zodat meerdere weergaven dezelfde naam kunnen delen. Dit kan handig zijn wanneer er meerdere versies van eenzelfde weergave zijn die elk gespecialiseerd is in een bepaalde functionaliteit, maar het niet zinnig zou zijn dit samen te voegen tot eenzelfde weergave. De voorwaarde voor de correcte werking is wel dat twee specificatiemodellen met dezelfde naam nooit gezamenlijk geldig mogen zijn voor een bepaalde dataset. Als dit wel het geval zou zijn, dan zou de applicatie niet kunnen achterhalen over welke weergave het gaat.

Het settingsgedeelte van de eigenschappen wordt gebruikt om aan te geven of de weergave gebruikt mag worden in de applicatie. Dit kan gebruikt worden om het logo dat de weergave zal representeren in de applicatie actief te maken en om aan te geven of het logo mag getoond worden. Het gebruik van deze settings is hoofdzakelijk bedoeld om aan te geven aan een gebruiker of een bepaalde weergave in ontwikkeling is of zelfs volledig niet te tonen.

De standaardgrootte van de weergave verwijst, zoals de naam aangeeft, naar de omvang die een weergave moet hebben als ze voor het eerst wordt vertoond. Dit is alleen geldig wanneer er nog geen grootte voor de weergave in de applicatie gekend is. De grootte van de weergave kan gekend zijn als het gaat om een ingeladen *template*. In dit geval moet worden gehouden aan de beschrijvingen van de inhoud van het dashboard door dit template.

Het voorlaatste item is de geaccepteerde datatypes. Dit onderdeel had eventueel onder de beperkingen geplaatst kunnen worden. Dit werd toch behouden onder het onderdeel eigenschappen omdat het meer vertelt over de aard van de weergave, dan dat het een sterke beperking is. Uiteindelijk is het ook een beperking natuurlijk.

Het laatste onderdeel van de eigenschappen is het herschaalbaar zijn van de weergave. In veel gevallen is een weergave herschaalbaar. In sommige gevallen, zoals bij een tabel, is herschalen van de weergave ongewenst.

2.3.2.2 Beperkingen

De beperkingen van een visualisatie in het specificatiemodel worden onder de sleutel ‘constraints’ gecombineerd. Dit onderdeel van het specificatiemodel bestaat uit vier onderdelen:

- De datagroeperingen;
- Het aantal voorkomens van een gegeven datatype;
- Beperkingen op de lengte van de data;
- Beperkingen op de waarden van de data.

De datagroeperingen zijn een uitbreiding op het onderdeel: “geaccepteerde datatypes” dat in voorgaande paragraaf werd behandeld. In het onderdeel datagroeperingen worden alle unieke combinaties van datatypes geplaats. Dit geldt ook voor enkele datatypes. De combinaties en enkele types worden hier in lijsten geplaatst. De motivatie achter dit onderdeel is dat een bepaalde weergave meerdere datatypes kan gebruiken om iets mee te doen, maar niet zomaar elke combinatie van deze datatypes mag gebruikt worden. Een voorbeeld hiervan is een lijngrafiek. Een lijngrafiek kan bijvoorbeeld een lijst van getallen, een lijst van strings en een lijst van datums gebruiken. Dit wil daarom niet zeggen dat al deze lijsten gezamenlijk gebruikt kunnen worden door de weergave. De combinaties: getallen- en stringlijsten, getallen- en datumlijsten en enkele getallenlijsten kunnen daarentegen afzonderlijk wel gebruikt worden.

Het aantal voorkomens beperkt het aantal keer dat een bepaald datatype gebruikt kan worden door een weergave. Deze beperking kan op twee verschillende manieren worden opgelegd:

- Een beperking die het bereik oplegt waarbinnen het aantal voorkomens moet liggen;
- Een beperking die het exacte aantal voorkomens voorlegt.

Bij het bereik kan het maximaal aantal voorkomens, het minimaal aantal voorkomens, en het maximaal en minimaal aantal voorkomens opgegeven worden. Als een van de twee niet wordt opgegeven, dan wordt er van uitgegaan dat dit de facto geldig is. De beperking van het aantal voorkomens is steeds een object waarin alle geaccepteerde datatypes als sleutel worden geplaatst. Als een datatype niet voorkomt, dan wordt het aantal voorkomens automatisch als geldig beschouwd. Gekoppeld aan deze sleutel is een getal. In figuur 35 is dit weergegeven. Daarnaast kan ook worden opgegeven of het aantal voorkomens van een datatype een veelvoud is van een bepaalde waarde. Het kan bijvoorbeeld zijn dat een weergave steeds paren van datasets nodig heeft, maar niets kan aanvangen met oneven aantallen datasets.

De beperking op de lengte van de data maakt ook gebruik van een bereik of een exacte hoeveelheid. In dit geval zijn aan de sleutels geen getal gekoppeld, maar een lijst van getallen. De index van deze lijsten verwijzen naar de diepte in de data. Bijvoorbeeld “number[]” : [15] in figuur 35 verwijst het getal 15 naar de lengte van de lijst, maar bij “string[]” : [15,30] verwijst het getal 15 naar lengte van de lijst en het getal 30 naar lengte van de strings in de lijst.

Daarnaast heeft de lengtebeperking nog een ander onderdeel: een beperking op dezelfde lengte. Deze beperking kan enkel worden gebruikt als de data meerdere itereerbare elementen heeft, zoals lijsten, objecten en strings. Dit onderdeel is weeral een lijst, maar dan met booleaanse waarden. Elke index komt overeen met het dieptepunt in

de data. Als op deze index de waarde true staat, dan moet alle data op die diepte dezelfde lengte hebben. Als er een false op staat, dan wordt er niet naar die lengte gekeken. In het voorbeeld in figuur 35 is de eerste index van dit gedeelte een true. Dit wil dan zeggen dat als een combinatie van ‘string[]’ en ‘number[]’ wordt ontvangen dat de lijsten van elk van de gegevens even lang moeten zijn, maar er wordt niet gekeken dat de lengte van de strings en de getallen van dezelfde lengte zijn.

Het laatste onderdeel is de beperking op de waarde. Dit onderdeel kan enkel gelden voor variaties met getallen. Deze beperking is enkel geldig voor een bereik zoals die van het aantal voorkomens. Het enige verschil is dat er enkel met een bereik kan gewerkt worden. Binnen dit bereik kunnen dan beperkingen worden gelegd op de maximale en minimale waarden. Dit kan belangrijk zijn wanneer de weergave niet gemaakt is om meer of minder dan een bepaalde numeriekewaarde weer te geven. Een taartdiagramma is bijvoorbeeld niet voorzien om negatieve waarden te tonen.

2.3.2.3 Functionaliteiten

Buiten de beschrijvingen van een visualisatie in een JSON-bestand kunnen ook extra parameters worden toegevoegd nadat het bestand reeds is ingeladen. De functionaliteiten die hier dan worden bijgevoegd zijn attributen of functionaliteiten die elders in de applicatie gebruikt worden. In de huidige implementatie van het programma betrekt dit zich tot het bijvoegen van het gewenste logo voor de weergave en het bijvoegen van een interfacecomponent. Het logo wordt gebruikt in de applicatie om de gebruiker een visueel idee te geven van de aard van de weergave. De interfacecomponent is een tussenstuk die de communicatie tussen de applicatie en de weergave bemiddelt.

2.3.3 Interfacecomponenten

In de applicatie wordt gebruik gemaakt van interfacecomponenten die als tussenstap worden gebruikt voor de communicatie tussen de effectieve visualisatie en de rest van de applicatie. Het gebruik van deze interfacecomponenten werd toegevoegd omdat de visualisaties van verschillende bibliotheken afkomstig kunnen zijn of eigen creaties kunnen zijn. Deze verschillende componenten aanvaarden daarom niet altijd dezelfde parameters noch hetzelfde dataformaat. Omdat het wenselijk is om de applicatie onwetend te houden over de specifieke implementatie van de weergaven werd gekozen om gebruik te maken van interfacecomponenten die de vertaling maken tussen hoe de applicatie data afhandelt en hoe deze gepresenteerd wordt aan de visualisaties. Op deze manier kunnen in de applicatie steeds dezelfde parameters worden doorgegeven aan de interfacecomponenten. Deze kunnen dan zelf uitmaken welke van de parameters zij al dan niet zullen gebruiken. Elke interface wordt altijd voorzien van volgende data in een object:

- De dataset, zoals deze in de applicatie wordt voorgesteld, die de weergave moet weten voor te stellen;
- De dimensies die de grootte van de visualisatie bepalen;
- De opties die de visualisatie kunnen personaliseren;
- De kindercomponent die aan de interfacecomponent wordt doorgegeven;
- De datatype-groepering van de datatset.

De dataset die aan de interfacecomponent wordt meegegeven is een object waarvan de sleutels de versleepte sleutelpaden zijn. Aan elk sleutelpad is de corresponderende data uit de JSON gekoppeld. Hoe deze data verwerkt wordt, is volledig afhankelijk van hoe de visualisatie data aanvaart. Het doorgeven van de dimensies aan de visualisatie is optioneel. Sommige weergaven herschalen automatisch, waardoor het niet nodig is dat deze wordt doorgegeven aan de component, maar het is wel handig voor die componenten dat dit niet automatisch doen.

De opties van een visualisatie zijn volledig afhankelijk van de component zelf. Deze opties betrekken zich tot de personalisatie van de visualisatiecomponent. Hier wordt in de applicatie geen veralgemening gemaakt. Het wordt verwacht dat deze bijgevoegd wordt onder de functionaliteiten van het specificatiemodel. Deze functionaliteit is wel nog niet ingewerkt in de applicatie zelf.

Het meegeven van een kindercomponent is geen noodzaak zolang de interfacecomponent een weergave teruggeeft. Het is mogelijk dat de interface enkel en alleen geldig is voor één specifieke weergave en niet herbruikbaar is. Onder deze omstandigheden is het niet belangrijk om de visualisatie mee te geven aan de interfacecomponent. Dit kan wel een goede manier zijn om een interfacecomponent bruikbaar te maken voor meerdere visualisaties. In dit laatste geval wordt gebruik gemaakt van de React-functie ‘React.cloneElement’. Hiermee is het mogelijk om de parameters van een component te overschrijven.

Als laatste resteert er nog de datatype-groeperingsparameter van een interfacecomponent. De reden waarom een typegroepering wordt doorgegeven aan de interface is omdat niet elke groepering noodzakelijk dezelfde bewerkingen moet doorstaan. In plaats dat voor elke interface deze functie geïmporteerd moet worden, werd ervoor gekozen om dit standaard aan elke interface aan te bieden. Een voorbeeld van een interface implementatie dat gebruik maakt van deze informatie wordt uitgebeeld in figuur 36.

```
const chooser = {
    '["string[]"]' : ( e ) => strArr( e ),
    '["string[]","number[]"]' : ( e ) => strNumArr( e )
}

const Interface = ( props, {} : React.Component<Props, {}>) : JSX.Element =>  {
    if ( props.data ) {
        const group : Array<String> = findGroup(props.data, props.groups);
        if ( group ){
            const strType = JSON.stringify( group );
            const datasets : interfaces.ChartJSData = chooser[strType]( props.data );

            return (
                React.cloneElement(props.children, {data : datasets, options :
props.options})
            )
        } return null;

    } return null;
}
```

36. Gebruik van datatypegroepering in een interfacecomponent

2.3.4 Specificatiecontainer

Om alle visualisaties overzichtelijk te groeperen worden ze verzameld in een container. Deze container is een lijst. Bij het aanmaken van deze lijst worden met de *spread operator* de specificatiemodellen voorzien van hun logo's en interfaces. Deze container wordt wel niet direct aan de rest van de applicatie beschikbaar gesteld. Kansen op corruptie van deze data wordt liefst tot een minimum gehouden. De applicatie krijgt wel toegang tot een object waaraan functies zijn gebonden. Deze functies zorgen ervoor dat de gewenste gegevens uit de container worden gehaald. Deze functies zijn:

- getInterfaces;
- getIcons;
- getDataSpecs;
- getInterfaceSize;
- getSize;
- getResizeEnabled.
- getInterface

Met uitzondering van getResizeEnabled, getInterfaceSize en getInterface creëert elk van deze functies een lijst van objecten met op zijn minst de naam van de weergave. Deze functies moeten steeds de hele container doorlopen. Naast de naam van de weergave worden ook andere attributen meegegeven afhankelijk de functie. Voor elk van deze functies zijn dit:

- getInterfaces: de interface van het specificatiemodel;
- getIcons; het logo en de settings van het model;
- getDataSpecs: de geaccepteerde datatypes en alle berperkingen van het model;
- geSize: de standaardgrootte van het model.

De functies getInterfaceSize, getResizeEnabled en getInterface zijn uitzonderingen op de andere functies. Deze functies maken geen gereduceerde versie van de container aan, maar geven een specifieke waarde terug. Welke waarde dit is, wordt bepaald aan de hand van de naam van het specificatiemodel waar ze een gegeven vandaan moeten halen. In het geval van de getInterfaceSize-functie wordt de standaardgrootte teruggegeven. De functie getResizeEnabled geeft terug of de visualisatie kan verschaald worden in de applicatie. De laatste van deze uitzonderingen geeft een interface terug.

2.4 Toepasselijke visualisaties

Tot nu toe is geweten dat hoe gegevens uit een JSON-visualisatie kunnen gehaald worden, hoe datatypes worden bepaald en hoe visualisaties aan de hand van enkele karakteristieken beschreven worden in de applicatie. In deze paragraaf wordt verder gebouwd op deze onderdelen. Hier zal worden bekeken welk proces doorlopen wordt om van een bepaalde dataset dat door de gebruiker werd gekozen om tot een lijst van visualisaties te komen die gebruikt kunnen worden om deze gegevens voor te stellen.

2.4.1 Algemeen

Om een lijst van toegelaten visualisaties te genereren voor een bepaalde dataset krijgt de applicatie maar één functie aangeboden. Deze functie neemt het hele validatieproces op zich en levert een lijst van visualisatienamen terug. Dit proces telt zes stappen:

1. Bepalen van de eigenschappen van de dataset;
2. Afhalen van de specificaties waaraan de eigenschappen van de dataset zullen getoetst worden;
3. Het bepalen van de datagroep waaronder de dataset valt;
4. Het valideren van de dataset tegenover de specificaties;
5. Het opslaan van de naam van de visualisatie die de dataset kan gebruiken;
6. Het teruggeven van de lijst van visualisatienamen.

De eerste drie stappen worden afgehandeld door gespecialiseerde functies. Bij de tweede stap wordt gebruik gemaakt van de getDataSpecs-functie uit paragraaf 2.3.4. Stap drie wordt niet apart besproken. In deze stap worden de gevonden datatypes – deze komen voort uit de bepaling van de eigenschappen – in een lijst samengevoegd. Stappen vier en vijf worden uitgevoerd in een *for-loop*. Deze loop doorloopt de hele lijst van specificaties die in stap twee werden verkregen. Bij elke iteratie van de loop wordt in volgorde volgende eigenschappen getoetst:

- Of de datatypes van de dataset overeenkomen met die uit het specificatiemodel;
- Of de datagroep kan worden gelinkt aan een datagroep uit het specificatiemodel;
- Of het aantal voorkomens van de datatypes voldoen aan de beperking uit het specificatiemodel;
- Of de lengte-eigenschappen voldoen aan het specificatiemodel;
- Of de waarden binnen de grenzen liggen van het specificatiemodel.

Als op slechts één van deze criteria wordt gefaald dan wordt een negatief antwoord teruggegeven en wordt de naam van de visualisatie niet opgeslagen in een lijstaccumulator. Als het op alle criteria geslaagd is, dan wordt de naam van de visualisatie wel opgeslagen. Pas als alle specificatiemodellen zijn doorlopen wordt de lijst met geaccepteerde visualisaties teruggegeven. De functies verantwoordelijk voor het doorlopen van dit proces zijn gegeven in figuur 37.

```
const validate = ({ spec, typegroup, properties }) => {
    let compliant = true;
    if ( spec.acceptedDataTypes[0] !== 'any' ) {
        compliant = validation.validTypes( typegroup, spec.acceptedDataTypes );
        if ( spec.constraints ) {
            if ( compliant ) {
                compliant = validation.validGroup( typegroup,
spec.constraints.groups );
            }
            if ( compliant ) {
                compliant = validation.validOccurrences( properties,
spec.constraints.occurrences);
            }
            if ( compliant ) {
                compliant = validation.validLength( properties,
spec.constraints.dataLength);
            }
            if ( compliant ) {
                compliant = validation.validValues( properties,
spec.constraints.values);
            }
        }
    } return compliant;
}

const options = ( object : object ) : Array<string> => {

    const data = Object.keys( object ).map( e => object[e] );
    const properties = validation.dataProps( data );
    const specs : Array<ChartDataProperties> = specifications.getDataSpecs();
    const typegroup = Object.keys(properties);
    const len = specs.length;
    let acc = [];
    for ( let i = 0; i < len; i++ ) {
        const spec = specs[i];
        const valid = validate({
            spec,
            typegroup,
            properties
        });
        if ( valid ) {
            acc.push( spec.name );
        }
    }
    return acc;
}
```

37. Generatorfunctie van gepaste visualisaties voor een gegeven dataset

2.4.2 Eigenschappen van data

Het bepalen van de eigenschappen verloop via een hoofdfunctie die enkele deelfuncties aanspreek om verschillende. De hoofdfunctie ontvangt een object met sleutelpaden als sleutels en data uit de JSON. Het omzetten van een dataset naar een collectie eigenschappen van deze dataset neemt enkele stappen:

1. Het datatype wordt bepaald voor alle gegevens in het object dat werd gegeven aan de hoofdfunctie;
2. Uit dezelfde gegevens worden de eigenschappen bepaald;
3. Uit de gegevens die werden verzameld uit de voorgaande twee stappen wordt bepaald hoe vaak een bepaald datatype is voorgekomen;
4. De gegevens worden gereduceerd om voor elk datatype één set van eigenschappen over te houden;
5. De eigenschappen van de datatypes worden nu aangevuld met het aantal keer dat dit datatype is voorgekomen;
6. Het finale product wordt teruggegeven.

De werking van de eerste stap werd reeds behandeld in paragraaf 2.3.1.2 en wordt direct overgegaan op de tweede stap. In deze stap worden de eigenschappen die eigen zijn aan de data zelf bepaald. Deze eigenschappen zijn de lengte voor elke diepte van de data, de maximale waarde en de minimale waarde. De laatste twee betrekking zich enkel tot numerieke waarden.

Het bepalen van de lengte-attributen van de data wordt bepaald door twee functies: een recursieve functie en een reducerende functie. De eerste functie krijgt als parameters de data, een accumulator en een diepte mee. Deze functie zal dan nagaan of dat de data van een JavaScript-object is, maar niet de waarde null heeft of als de data van het type string is. Als dit niet het geval is, is het zinloos om verder te gaan. Als er wel is voldaan aan de voorwaarde, dan wordt de lengte van de data bepaald. Deze lengte wordt dan in de accumulator als een object opgeslagen. Dit object bevat de diepte en lengte van de data. De tweede functie converteert de data in de accumulator in een lijst waarbij elke index de diepte aangeeft en de bijhorende waarde de maximale lengte is dat werd gevonden voor die diepte. De functies zijn gegeven in figuur 38.

```
const lengthReducer = ( acc : Array<LenDig> ) : Array<number> => {
	const len = acc.length;
	let list = [];
	for ( let i = 0; i < len; i++ ) {
		const depth = acc[i]['layer'];
		const length = acc[i]['length'];
		if ( !list[depth] || list[depth] < length ) {
			list[depth] = length;
		}
	}
	return list;
}

const lengthProperty = ( data : object, acc : Array<LenDig>, depth : number = 0 ) :
void => {
	if ( (typeof data === 'object' && data !== null) || typeof data === 'string' ) {

		const keys = Object.keys(data);
		const len = keys.length;

		acc.push({
			"layer" : depth,
			"length" : len
		});
		if (typeof data !== 'string'){
	       for ( let i = 0; i < len; i++ ) {
	           const key = keys[i];
	           const item = data[key];
	           if ( (typeof item === 'object' && item !== null) || (typeof item ===
'string' && len > 1) ) {
	               lengthProperty( item, acc, depth + 1 );
	           }
	       }
	   }
	}
}
```

38. Bepalen van de lengte-attribuut

De maximale en minimale waarde van de data wordt bepaald door een recursieve functie, die net als bij het bepalen van de lengtes, een accumulator gebruikt om gezamenlijk de minimale en maximale waarde in op te slaan. Deze accumulator is wel een object met sleutels ‘maxValue’ en ‘minValue’. Als een getal wordt teruggevonden in de data dan wordt gekeken of deze groter is dan de ‘maxValue’ of kleiner is dan de ‘minValue’. Als dit waar blijkt te zijn voor een van de twee parameters, dan wordt de oude waarde vervangen door de gevonden waarde. Als voor het eerst een waarde wordt gevonden, dan worden zowel de ‘maxValue’ en de ‘minValue’ hieraan gelijkgesteld. Het gelijk zijn van de ‘maxValue’ en ‘minValue’ kan gezien worden als een exacte waarde. Uiteindelijk worden de lengte- en waarde-attributen in een gezamenlijk object opgeslagen en teruggegeven Deze functie wordt weergegeven in figuur 39.

```
const valueProperty = ( data : object, acc : Value ) : void => {
	if ( typeof data === 'object' && data !== null ) {
		const keys = Object.keys(data);
		const len = keys.length;

		for ( let i = 0; i < len; i++ ) {
			const key = keys[i];
			const item = data[key];

			if ( typeof item === 'number' ) {
				if ( typeof acc.maxValue === 'number' && acc.maxValue < item
) {
					acc.maxValue = item;
				} else if ( typeof acc.minValue === 'number' && acc.minValue
> item ) {
					acc.minValue = item;
				} else if ( !acc.minValue && !acc.maxValue ) {
					acc.minValue = item;
					acc.maxValue = item;
				}
			}
			valueProperty( item, acc );
		}
	}
}
```

39. Bepalen van de waarde-attributen

Op dit punt is een lijst van eigenschappen bekomen, maar ontbreekt nog het aantal keren dat een datatype is voorgekomen. Om dit te kunnen bepalen wordt uit de lijst van eigenschappen en nieuwe lijst gemaakt met enkel de datatypes. In een aparte functie wordt voor elk datatype in de lijst geteld hoeveel keren deze voorkomt. De accumulator voor deze telling is een object met als sleutels de datatypes uit de lijst. Deze functie is gegeven in figuur 40.

```
const countOccurrence = ( list : Array<string> ) : OccurrenceConstraint => {
    let acc = {};
    const len = list.length;

    for ( let i = 0; i < len; i++ ) {
        const item = list[i];
        if( item in acc ){
            acc[item] += 1;
        } else {
            acc[item] = 1;
        }
    }

    return acc;
}
```

40. Telling van de datatypes

Nu blijven een lijst van eigenschappen en een object met het aantal keren dat een datatype is voorgekomen. Dit moet de hoofdfunctie verlaten als een object met de gevonden datatypes als sleutels. Aan deze sleutels zijn dan de eigenschappen voor dit datatype gebonden. Er moet dus nog een tussenstap worden gemaakt vooraleer de eigenschappen te koppelen aan het aantal voorkomens. Opnieuw komt hier een functie aan te pas, weergegeven in figuur 41. Deze functie neemt de lijst van eigenschappen als argument aan. In deze functie zullen de eigenschappen worden gereduceerd naar een object met dezelfde sleutels als dat van het aantal voorkomens, maar dan met een veralgemening van de eigenschappen van de datatypes. Hierbij wordt weer gebruik gemaakt van een accumulator om de gegevens samen te vatten. Het verloop hiervan is als volgt:

- Als de accumulator de naam van het gevonden datatype al bezit, vervang dan die bestaande eigenschappen in de accumulator als de overeenkomstige nieuwe eigenschappen groter zijn;
- Als de accumulator het datatype nog niet heeft, voeg alle eigenschappen direct toe.

```
const reduceCollection = ( list : Array<Props> ) : Props => {
    let acc : Props = {};
    const len = list.length;

    for( let i = 0; i < len; i++ ){
        const item : Props = list[i];
        if( item.name in acc ){
            const temp = acc[item.name];
            if ( temp.length < item.properties.dataLength ) {
                temp.length = item.properties.dataLength;
            }
            if ( temp.maxValue < item.properties.maxValue ) {
                temp.length = item.properties.maxValue;
            }
            if ( temp.minValue > item.properties.minValue ) {
                temp.minValue = item.properties.minValue;
            }
        } else {
            acc[item.name] = {
                dataLength : item.properties.dataLength,
                maxValue : item.properties.maxValue,
                minValue : item.properties.minValue
            }
        }
    }

    return acc;
}
```

41. Reductie van de eigenschappen per datatype

Finaal blijven dan twee objecten over met dezelfde sleutels, maar een met de globale eigenschappen van dat datatype voor een bepaalde dataset en de ander met het aantal keren dat een bepaald datatype werd teruggevonden. Nu is het slechts een kwestie van de twee object samen te voegen. Dit wordt niet meer geïllustreerd omdat dit een geen complexiteit vergt.

2.4.3 Validatie van het datatype

Om te kunnen achterhalen of de datatypes die gevonden werden in de dataset en de datatypes die geaccepteerd worden door een visualisatie bestaat een functie die dit zal nagaan. Deze functie verwacht twee parameters te ontvangen: een lijst van datatypes uit de dataset en een lijst van datatypes uit het specificatiemodel.

In eerste instantie wordt nagegaan of het specificatiemodel wel een lijst van geaccepteerde datatypes heeft. Als dit niet zo is, dan geeft de functie automatisch een positief antwoord terug. Als er wel een is opgegeven, dan zal in een geneste for-loop elk ten opzichte van elkaar worden afgetoetst. Dit gebeurt in de binnenste for-loop. Wanneer een match wordt gevonden, dan wordt deze for-loop onderbroken en aangegeven dat het datatype in beide lijsten aanwezig is. In het andere geval wordt dit niet gedaan. Wanneer de binnenste for-loop gedaan heeft, dan zal de eerste for-loop nagaan of het datatype werd teruggevonden. Alleen wanneer het datatype niet werd teruggevonden wordt een negatief antwoord teruggeven door deze functie. De functie is weergeven in figuur 42.

```
const validTypes = ( list : Array<string>, controlList : Array<string> ) : boolean => {
    if ( controlList ) {
        let valid = true;
        const len0 = list.length;
        const len1 = controlList.length;
        for ( let i = 0; i < len0; i++ ) {
            const item0 = list[i];
            let found = false;

            for ( let j = 0; j < len1; j++ ) {
                const item1 = controlList[j];
                if ( item0 === item1 ) {
                    found = true;
                    break;
                }
            }

            if ( !found ) {
                valid = false;
                break;
            }
        }

        return valid;
    }
    return true;
}
```

42. Validatie van het datatype

2.4.4 Validatie van de datagroep

Deze functie zal nagaan of de datagroep – de lijst van alle datatypes uit de dataset – kan worden teruggevonden in de datagroeperingen die werden gedefinieerd in het specificatiemodel. Buiten het controleren of dit onderdeel überhaupt aanwezig is in het specificatiemodel zal eerst een reductie worden gemaakt van deze gegevens. Deze reductie is bedoeld om alle datagroepen die niet van dezelfde lengte zijn als de datagroep die getest moet worden te verwijderen uit de lijst. De nieuwe lijst bevat dan enkel datagroeperingen waarvan geweten is dat ze mogelijks gelijk zijn aan de datagroep van de dataset.

Nadien wordt, aan de hand van de lengte van de nieuwe lijst, met een for-loop geïtereerd. In het geval dat de nieuwe lijst leeg is – omdat er geen datagroepen bestaan die van dezelfde lengte zijn als die van de dataset – dan zal er negatief antwoord worden gegeven door deze functie. In alle andere gevallen zal de functie voor het valideren van het datatype worden opgeroepen. Hieraan worden dan de datagroep van de dataset en de datagroep op een index uit de nieuwe lijst meegegeven. Als deze test een positief antwoord teruggeeft, dan wordt de for-loop onderbroken en het positieve antwoord teruggeven. Deze functie is weergegeven in figuur 43.

```
const validGroup = ( list : Array<string>, controlList : Array<Array<string>> ) :
boolean => {
    if ( controlList ) {
        let valid = false;
        const len0 = list.length;
        const reducedControl : Array<Array<string>> = controlList.reduce( (acc :
Array<Array<string>>, e : Array<string> ) => {
            if ( e.length === len0 ) {
                acc[ acc.length ] = e;
            } return acc
        }, []);
        const len1 = reducedControl.length;

        for ( let i = 0; i < len1; i++) {
            const list1 = reducedControl[i];
            valid = validTypes( list, list1);

            if ( valid ) {
                break;
            }
        }

        return valid;
    }
    return true;
}
```

43. Validatie van de datagroep

2.4.5 Validatie van het aantal voorkomens

Het validatie gedeelte voor het aantal voorkomens bestaat uit een hoofdfunctie en drie hulpfuncties:

- occEquals;
- occRange;
- occMulti.

De hoofdfunctie overziet enkel of het de beperking is opgegeven in het specificatiemodel en het oproepen van de hulpfuncties. De hoofdfunctie is weergegeven in figuur 44. Deze hulpfuncties nemen elk een onderdeel van de validatie op zich. Elk van de hulpfuncties verwacht dezelfde gegevens van de hoofdfunctie:

- De lijst van sleutels van het object – het object dat de eigenschappen van de dataset bevat – dat aan de hoofdfunctie werd meegeven;
- De lengte van die lijst van sleutels;
- Het object dat de hoofdfunctie ontving;
- De beperking op het aantal voorkomens uit het specificatiemodel.

```
export const validOccurrences = ( object : Props, constraint : Occurrence ) : boolean
=> {
    if (constraint) {
        const keys = Object.keys(object);
        const len = keys.length;

        let params : OccParams = {
            keys,
            len,
            object,
            constraint
        };

        const equals = occEquals(params);
        const range = occRange(params);
        const multi = occMulti(params);

        return equals && range && multi;

    } return true;
}
```

44. Hoofdfunctie voor testen van het aantal voorkomens

De hoofdfunctie zal, net als elke hulpfunctie, eerst nagaan of het onderdeel van het specificatiemodel bestaat. Zou dit niet bestaan dan wordt een positief antwoord gegeven. Er wordt hier van uitgegaan dat de afwezigheid van de beperking betekent dat er niet op hoeft getest te worden. In het andere geval, dan worden alle hulpfuncties opgeroepen. Als de data slaagt op alle testen, dan wordt een positief antwoord teruggegeven. Zo niet, een negatief antwoord.

2.4.5.1 occEquals

Deze functie, in figuur 45, gaat na of er het aantal voorkomens in het object overeenstemt met de exacte hoeveelheid die verwacht wordt in het specificatiemodel. Bij het testen op de beperking zal een for-loop worden geïtereerd tot de lengte van de sleutellijst van het object. Bij elke iteratie wordt dan nagekeken of de sleutel – de sleutel is zelf een datatype – bestaat in het onderdeel van het specificatiemodel en, als dit waar is, of de hoeveelheden in het object en in het specificatiemodel overeenkomen. Zouden de twee niet overeenkomen dan wordt een negatief antwoord teruggegeven. Als de for-loop volledig alle sleutels in het object is kunnen afgaan, dan wordt een positief antwoord teruggeven. Dit betekent dat ofwel geen enkel van de sleutels in het object werd teruggevonden of dat voor alle sleutels die werden teruggevonden geen enkele niet voldeed aan de voorwaarde.

```
const occEquals = ({len, keys, object, constraint} : OccParams ) : boolean => {
    if ( constraint.equals ) {

        for (let i = 0; i < len; i++) {
            const index = keys[i];
            const item = object[index];

            if ( index in constraint.equals ) {
                if ( constraint.equals[index] !== item.occurrence ){
                    return false;
                }
            }
        }
        return true;
    } return true;
}
```

45. Test op het gelijkheid

2.4.5.2 occRange

Deze hulpfunctie, in figuur 46, bestaat uit drie onderdelen die elk onder verschillende voorwaarden actief zullen zijn. Deze voorwaarden zijn afhankelijk van de staat van het bereik-onderdeel van het specificatiemodel:

- Zowel een maximum als een minimum zijn opgegeven;
- Enkel een maximum is opgegeven;
- Enkel een minimum is opgegeven.

Indien zowel een minimum als een maximum zijn opgegeven, dan wordt een for-loop geïnitialiseerd waarin de sleutels van het object zullen worden overlopen. Hierbinnen wordt een tweede controleronde gestart. Deze controle zal nagaan of voor het datatype – de sleutel in het object dat als parameter aan de hulpfunctie werd gegeven – een minimum en/of een maximum zijn opgegeven. De verdeling is wederom dezelfde als in de opsomming hierboven. In elk van deze secties wordt dan getest of het aantal voorkomens van het datatype binnen het bereik liggen dat wordt opgegeven door het specificatiemodel. Als slechts een van deze tests faalt, dan wordt onmiddellijk een negatief antwoord teruggeven door de functie. Anders is de object geslaagd voor deze test.

In de andere twee gevallen wordt er weer over op dezelfde manier over het object geïtereerd. Verschillend ten opzichte van het vorige zal enkel worden getest of het object geslaagd is voor het geval waarop getest wordt. Verder is de werking dezelfde als die voor het geval waar een minimum en maximum zijn opgegeven in het specificatiemodel.

```
const occRange = ({len, keys, object, constraint} : OccParams ) : boolean => {
    if ( constraint.range ) {
        let valid = true;
        const max = constraint.range.max;
        const min = constraint.range.min;

        if ( max && min ){
            for (let i = 0; i < len; i++) {
                const index = keys[i];
                const item = object[index];
                const iMax = max[index];
                const iMin = min[index];

                if ( typeof iMin === 'number' && typeof iMax === 'number' ){
                    valid = (iMax >= item.occurrence) && (iMin <= item.occurrence) ;
                } else if ( typeof iMax === 'number' ) {
                    valid = (iMax >= item.occurrence);
                } else if ( typeof iMin === 'number' ) {
                    valid = (iMin <= item.occurrence);
                }

                if (!valid){return valid;}
            }
        } else if ( max ) {
            for (let i = 0; i < len; i++) {
                const index = keys[i];
                const item = object[index];
                const iMax = max[index];

                if ( typeof iMax === 'number' ) {
                    valid = iMax >= item.occurrence;
                }

                if (!valid){return valid;}
            }

        } else if ( min ) {
            for (let i = 0; i < len; i++) {
                const index = keys[i];
                const item = object[index];
                const iMin = min[index];

                if ( typeof iMin === 'number' ) {
                    valid = iMin <= item.occurrence;
                }

                if(!valid){return valid;}
            }
        }
        return valid;
    } return true;
}
```

46. Test op het bereik

2.4.5.3 occMulti

De occMulti-hulpfunctie, in figuur 47, is sterk gelijkend op de occEquals-hulpfunctie. Er wordt opnieuw gebruik gemaakt van een for-loop en voor dezelfde reden. Nu zal er in de for-loop steeds getest worden of het aantal keren dat een bepaald datatype is voorgekomen een veelvoud is van de waarde in het specificatiemodel. Deze test wordt enkel uitgevoerd wanneer het veelvoud-gedeelte bestaat in het specificatiemodel. Zou de test falen voor slechts een van de datatypes, dan wordt een negatief antwoord teruggegeven door de functie. In alle andere gevallen is de test geslaagd.

```
const occMulti = ({len, keys, object, constraint} : OccParams ) : boolean => {
    if ( constraint.multipleOf ) {
        let valid = true;

        for (let i = 0; i < len; i++) {
            const index = keys[i];
            const item = object[index];
            const multi = constraint.multipleOf;

            if ( multi ) {
                valid = valid && (item.occurrence % multi === 0);
            }

            if ( !valid ) {
                return false;
            }
        }
        return true;
    } return true;
}
```

47. Test op meervoud

2.4.6 Validatie van de datalengte

Net als het geval was voor de validatie van het aantal voorkomens, bestaat de validatie voor de datalengte ook uit een hoofdfunctie, in figuur 48, en drie hulpfuncties:

- lenSameLen;
- lenEquals;
- lenRange;

De hoofdfunctie en hulpfuncties zullen wederom nagaan of hun onderdeel van het specificatiemodel bestaan en, als dit zo is, de verdere validatiestappen doorlopen. In het andere geval wordt er opnieuw van uitgegaan dat het niet bedoeld is om op dit onderdeel te testen en wordt een positief antwoord gegeven zonder verdere inspectie.

```
export const validLength = (object : Props, constraint : LengthConstraint) : boolean
=> {
    if ( constraint ) {
        const keys = Object.keys(object);
        const len = keys.length;

        const proto_params : ProtoLenParams = {
            object,
            constraint
        };

        const params : LenParams = {
            ...proto_params,
            keys,
            len
        };

        const samelen = lenSameLen(proto_params);

        if ( samelen ) {
            const equals = lenEquals(params);
            const range = lenRange(params);

            return equals && range;
        }

        return false;

    } return true;
}
```

48. Hoofdfunctie voor het valideren van datalengten

In de veronderstelling dat er wel een datalengte onderdeel aanwezig is in het specificatiemodel, dan worden twee parameters aangemaakt. Een parameter dat de parameters van de hoofdfunctie in een object omsluit. De andere parameter bouwt verder op de voorgaande parameter, maar voegt er de lijst van sleutels van het object en de lengte van deze lijst bij.

Eerst zal er getest worden of alle parameters in het object met eigenschappen van dezelfde lengte moeten zijn. Als deze functie een positief antwoord geeft, dan kunnen de resterende twee hulpfuncties worden opgeroepen. Indien een van de twee hulpfuncties een positief antwoord geeft, dan zal de hoofdfunctie ook een positief antwoord geven. In alle andere gevallen, dan wordt een negatief antwoord gegeven.

2.4.6.1 lenSameLen

In deze hulpfunctie, in figuur 49, wordt getest of alle gegevens van dezelfde lengte zijn voor een gegeven index. Dit is op voorwaarde dat in de ‘sameLength’-parameter van het specificatiemodel op diezelfde index de waarde true kan worden teruggevonden. Om dit na te gaan wordt gebruik gemaakt van een geneste for-loop. De eerste for-loop itereert tussen nul en de lengte van ‘sameLength’-lijst. De tweede for-loop itereert tussen nul en de lengte van de lijst van sleutels van het object.

```
const lenSameLen = ({constraint, object} : ProtoLenParams ) : boolean => {
    if ( constraint.sameLength ) {
        const len = constraint.sameLength.length;
        const keys = Object.keys(object);
        const lenlen = keys.length;

        for ( let i = 0; i < len; i++ ) {
            if ( constraint.sameLength[i] ) {
                for (let j = 0; j < lenlen - 1; j++ ) {
                    const index0 = keys[j];
                    const index1 = keys[j + 1];
                    const item0 = JSON.stringify(object[index0].dataLength);
                    const item1 = JSON.stringify(object[index1].dataLength);

                    if ( item0[i] !== item1[i] ) {
                        return false
                    }
                }

            }
            return true;
        }
    } return true;
}
```

49. Testen van dezelfde lengte

Bij elke iteratie van de eerste for-loop wordt gekeken of voor een bepaalde index de waarde van de ‘sameLength’-lijst in het specificatiemodel de waarde true is. Als dit waar is wordt de tweede for-loop gestart, anders niet. In de tweede for-loop wordt getest of elk naburig gegeven in het object dezelfde datalengte kan worden gevonden. Omdat de datalengte-parameter uit het object een lijst van getallen is, moet deze lijst wel eerst omgezet worden naar een string met de JSON.stringify-functie om correct de gelijkheid van de twee te kunnen testen. Als op eender welk punt twee naburige gegevens niet dezelfde lengte hebben, dan geeft de hulpfunctie een negatief antwoord terug. In alle andere gevallen resulteert dit in een positief antwoord.

2.4.6.2 lenEquals

Deze hulpfunctie, in figuur 50, is grotendeels gelijk aan de hulpfunctie van het aantal voorkomens dat nagaat of voor een bepaald datatype een exact aantal keren is voorgekomen in de dataset. De uitzondering tussen de twee is dat, na het testen of een bepaald datatype zich in het onderdeel van het specificatiemodel bevindt waarop getest wordt, de string-getransformeerde van de datalengte in het object wordt getoetst aan de string-getransformeerde uit het specificatiemodel. Als op eender welk punt in het testen van elk van deze datalengten uit het object dit niet waar zou zijn, dan geeft de hulpfunctie een negatief antwoord terug. Buiten deze ene voorwaarde wordt altijd een positief antwoord teruggegeven door de functie.

```
const lenEquals = ({keys, len, object, constraint} : LenParams ) : boolean => {
    if ( constraint.equals ) {
        for (let i = 0; i < len; i++) {
            const index = keys[i];
            const item = object[index];

            if ( index in constraint.equals ) {
                if ( JSON.stringify(constraint.equals[index]) !==
JSON.stringify(item.dataLength) ){
                    return false;
                }
            }
        } return true;
    } return true;
}
```

50. Testen van de gelijkheid

2.4.6.3 lenRange

Ook deze functie, in figuur 51, leent een redelijk groot deel van zijn samenstelling aan de functie die het bereik van het aantal voorkomens test. Om niet de hele functie opnieuw te overlopen, worden enkel de verschillen tussen deze hulpfunctie en zijn gelijke bij het testen van het aantal voorkomens uitgewezen.

```
const lenRange = ({len, keys, object, constraint} : LenParams ) : boolean => {
    if ( constraint.range ) {
        let valid = true;
        const max = constraint.range.max;
        const min = constraint.range.min;

        if ( max && min ){
            for (let i = 0; i < len; i++) {
                const index = keys[i];
                const item = object[index];
                const iMax : Array<number> = max[index];
                const iMin : Array<number> = min[index];

                if ( iMin && iMax ){
                    valid = iMax.map( (e, i) => (item.dataLength[i] <= e) &&
(item.dataLength[i] >= iMin[i]) ).reduce( (acc, e) => acc && e, true);
                } else if ( iMax ) {
                    valid = iMax.map( (e, i) => item.dataLength[i] <= e ).reduce( (acc,
e) => acc && e, true);
                } else if ( iMin ) {
                    valid = iMin.map( (e, i) => item.dataLength[i] >= iMin ).reduce(
(acc, e) => acc && e, true);
                }
                if (!valid){return valid;}
            }
        } else if ( max ) {
            for (let i = 0; i < len; i++) {
                const index = keys[i];
                const item = object[index];
                const iMax = max[index];
                if ( iMax ) {
                    valid = iMax.map( (e, i) => item.dataLength[i] <= e ).reduce( (acc,
e) => acc && e, true);
                }

                if (!valid){return valid}
            }

        } else if ( min ) {
            for (let i = 0; i < len; i++) {
                const index = keys[i];
                const item = object[index];
                const iMin = min[index];
                if ( iMin ) {
                    valid = iMin.map( (e, i) => item.dataLength[i] >= iMin[i] ).reduce(
(acc, e) => acc && e, true);
                }

                if(!valid){return valid}
            }
        } return valid;
    } return true;
}
```

51. Testen van het bereik

Het eerste verschil is natuurlijk dat ze verschillende onderdelen van het specificatiemodel opvragen om hun testen mee uit te voeren. Het tweede verschil is dat bij elk testen de gegevens uit het object en het specificatiemodel met mappings worden doorlopen. Deze mappings zijn terug te vinden in bovenstaande figuur.

Verder zijn de twee functies nagenoeg identiek aan elkaar in elk opzicht. De gelijke onderdelen zouden afgesplitst kunnen worden in latere versies van de applicatie zodat ze algemener gebruikt kunnen worden.

2.4.7 Validatie van de datawaarden

Deze laatste validatiefunctie, in figuur 52, zal nazien of de waarden van data zich binnen een bepaald bereik bevinden. Deze functie heeft slechts een hulpfunctie: 'valRange'. In deze functie wordt de lijst van sleutels van het object met de eigenschappen, de lengte van deze lijst, het object met de eigenschappen van de dataset en het onderdeel van het specificatiemodel waarop deze functie werkzaam is, gebundeld. Daarnaast zal deze functie enkel nog het resultaat van de hulpfunctie teruggeven. Net als bij de vorige validatiefuncties wordt ook hier ook eerst gekeken of het onderdeel waarop deze functie moet testen aanwezig is in het specificatiemodel. Deze functie geeft bijna altijd een positief antwoord. Enkel wanneer de test van de hulpfunctie faalt geeft de hoofdfunctie een negatief antwoord.

```
export const validValues = (object : object, constraint : ValuesConstraint) : boolean
=> {
       if (constraint) {
              const keys = Object.keys(object);
        const len = keys.length;

        const params : ValParams = {
            keys,
            len,
            object,
            constraint
        }

        return valRange(params);

    } return true;
}
```

52. Hoofdfunctie voor het valideren van de datawaarden

De hulpfunctie, in figuur 53, test of de gegevens in het object met de eigenschappen van de dataset binnen het bereik liggen dat werd opgegeven in het specificatiemodel. Het is nagenoeg identiek aan de functie die hetzelfde test voor het aantal voorkomens. Het enige verschil is dat hier het bereik getest wordt op de minimale en maximale waarden die in de dataset werden teruggevonden. Er wordt weer benadrukt dat deze functies in een later stadium zullen aangepast worden om de gelijkenissen af te splitsen in een algemene functie die pas een specifieke implementatie krijgt bij het oproepen door een van de validatiefuncties die deze functie gebruikt.

```
const valRange = ({len, keys, object, constraint} : ValParams ) : boolean => {
    if ( constraint.range ) {
        const max = constraint.range.max;
        const min = constraint.range.min;
        let valid = true;

        if ( max && min ){
            for (let i = 0; i < len; i++) {
                const index = keys[i];
                const item = object[index];
                const iMax = max[index];
                const iMin = min[index];
                if ( typeof iMin === 'number' && typeof iMax === 'number' ){
                    valid = valid && (iMax >= item.maxValue) && (iMin <=
item.minValue) ;
                } else if ( typeof iMax === 'number' ) {
                    valid = valid && (iMax >= item.maxValue);
                } else if ( typeof iMin === 'number' ) {
                    valid = valid && (iMin <= item.minValue);
                }
                if (!valid){return valid;}
            }
        } else if ( max ) {
            for (let i = 0; i < len; i++) {
                const index = keys[i];
                const item = object[index];
                const iMax = max[index];
                if ( typeof iMax === 'number' ) {
                    valid = valid && iMax >= item.maxValue;
                }

                if (!valid){return valid}
            }

        } else if ( min ) {
            for (let i = 0; i < len; i++) {
                const index = keys[i];
                const item = object[index];
                const iMin = min[index];
                if ( typeof iMin === 'number' ) {
                    valid = valid && iMin <= item.minValue;
                }

                if(!valid){return valid}
            }
        } else {
            return valid;
        }
    } else {
        return true;
    }
}
```

53. Test op het bereik

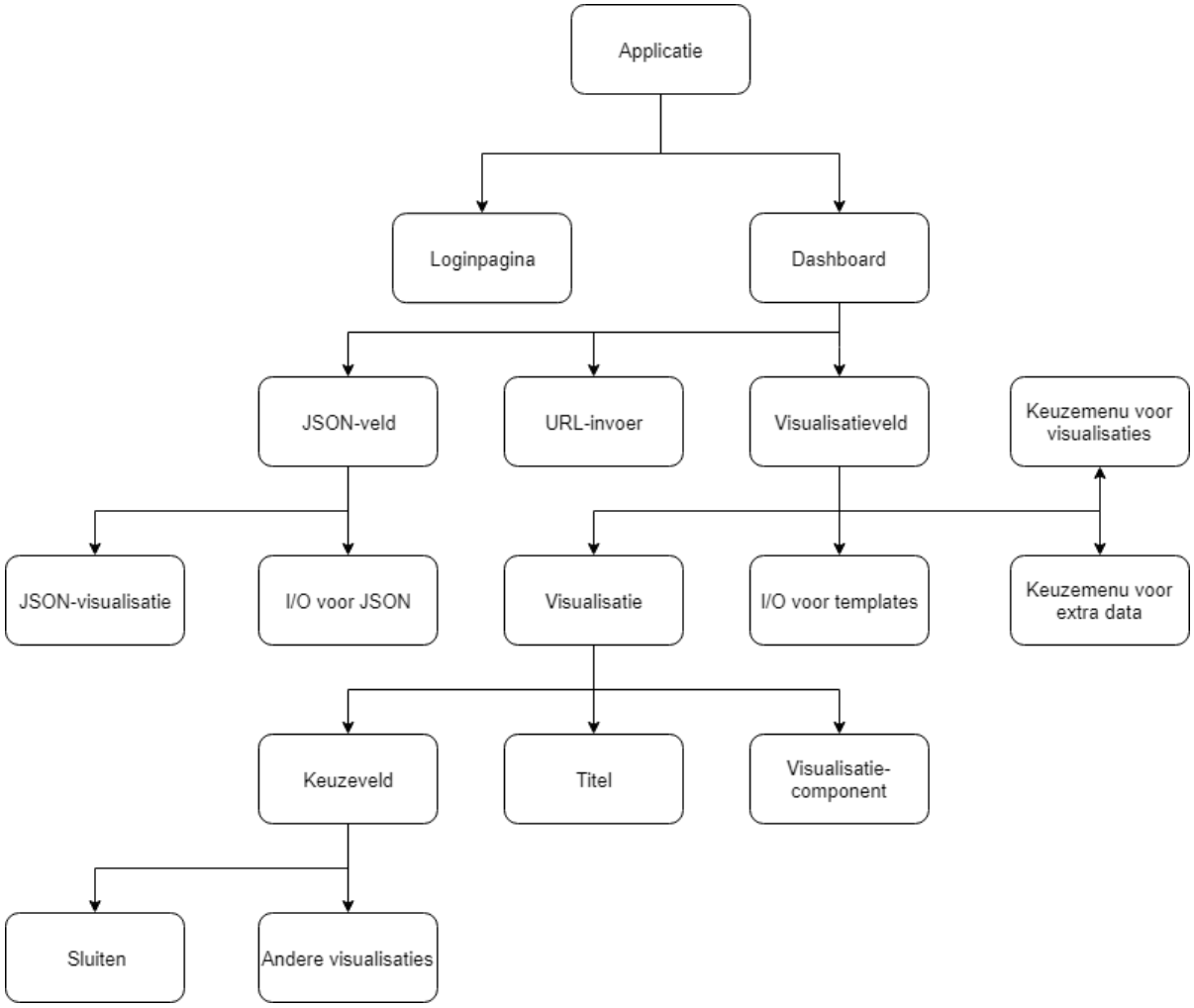
2.5 Userinterface

Het design van de userinterface is een belangrijk onderdeel voor applicaties waarbij de gebruiker afhankelijk is van de interface om op een eenvoudige en gestructureerde manier te kunnen navigeren doorheen de pagina. De userinterface van een business intelligence tool kan ruwweg opgedeeld worden in enkele thema's:

- Autorisatie;
- Presentatie van de data;
- Presentatie van de visualisaties;
- Keuze van het type visualisatie;

- I/O opties.

In de applicatie zijn deze thema's verdeeld over de loginpagina en de visualisatiepagina. De loginpagina is verantwoordelijk voor de autorisatie onderdeel van de applicatie. De visualisatiepagina is verantwoordelijk voor het afhandelen van de andere thema's. Hoe deze onderdelen zich structureren ten opzichte van elkaar wordt getoond in onderstaande figuur.

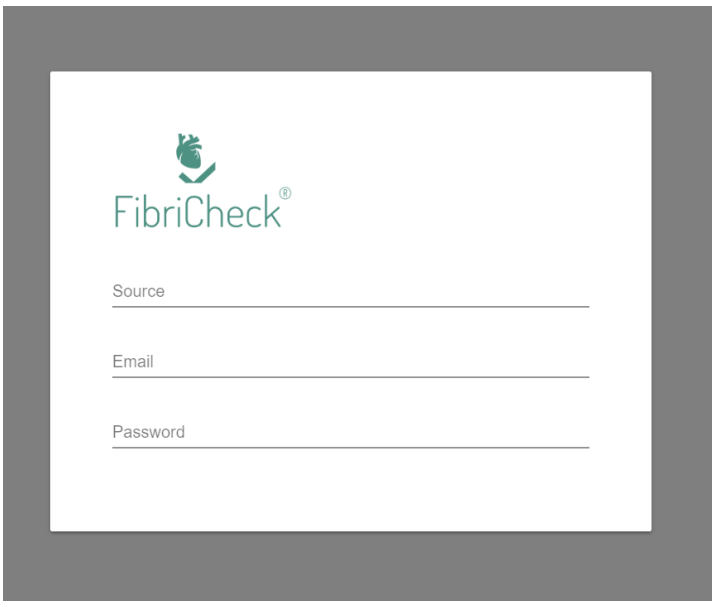


54. UI-structuur

In deze paragraaf zal eerst worden ingegaan op de userinterface structuur van de login- en visualisatiepagina. Dit zijn de hoogst liggende componenten die de opmaak van de applicatie beheren. Daarna wordt de opbouw van de onderdelen van de visualisatiepagina behandeld.

2.5.1 Loginpagina

Het autorisatie-gedeelte van de applicatie is een nodige stap om data te kunnen verkrijgen van de services van Qompium. Om een gebruiker geautoriseerd te krijgen, wordt gewerkt met een loginscherm. Dit is een eenvoudige en directe manier om autorisatiesleutels te ontvangen van de services van Qompium die gedurende de werking van de applicatie geldig blijven. In figuur 55 is de loginpagina weergegeven.



55. Loginpagina

Het voordeel van deze manier van werken, is dat het de gebruiker direct in staat stelt om te connecteren met de databron waarvan zij data wensen te halen en dat de autorisatie geldig blijft gedurende de hele sessie van de applicatie. Het nadeel hiervan is dat flexibel connecteren met verschillende informatiebronnen problematisch wordt. Hiervoor moet dan steeds via hetzelfde loginscherm worden gegaan. Op dit ogenblik vormt dit weinig hinder omdat Qompium een beperkt aantal bronnen heeft waarvoor autorisatie moet aangevraagd worden.

Een flexibelere manier om te connecteren met informatiebronnen zou zijn om de autorisatie in te werken in het visualisatie gedeelte van de applicatie. Hierdoor krijgt de gebruiker direct toegang tot de applicatie en kunnen de functionaliteiten direct worden gebruikt. Het wordt dan ook mogelijk om andere gebruikersnaam/email en passwoord combinaties te gebruiken om met bronnen te connecteren.

2.5.2 Visualisatiepagina

Zoals vermeld werd, neemt de visualisatiepagina het gros van de werking van de applicatie voor zijn rekening. Dit is het onderdeel van de applicatie waar de gebruiker de meeste tijd zal besteden. Hier worden alle functionaliteiten aan de gebruiker blootgesteld die nodig zijn om visualisaties te creëren, op te slaan en op te vragen.

De userinterface van de visualisatiepagina wordt grotendeels bepaald door de functionaliteiten die het moet bieden aan de gebruiker. Dit gedeelte van de applicatie bestaat uit 3 onafhankelijke onderdelen. Deze onderdelen nemen samen het hele schermoppervlak in beslag. Hun respectievelijke karakteristieken zullen bepalen hoe ze over het scherm verdeeld worden:

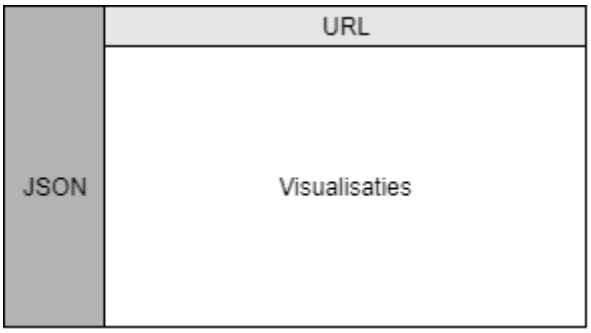
- Een JSON-veld dat de presentatie van JSON-data voor zich neemt;
- Een visualisatieveld dat de presentatie van visualisaties, opslaan en ophalen van visualisatietemplates afhandelt. Daarbij zal het de gebruiker ook alert maken van andere data die uit gelijkaardige delen van de JSON komen en de optie worden gegeven om deze bij te voegen in de visualisatie;
- Een URL-veld waar dat data opvraagt van de bron waar de autorisatiesleutels gekend zijn.

Het URL-veld, zoals de naam aangeeft, behandelt URL's. Net als dit het geval zou zijn in de browser, is het URL-veld niet meer dan een inputveld waar een lijst van karakters kunnen worden ingegeven. Dit maakt dat het URL-veld voornamelijk in de horizontale richting ruimte in beslag neemt en slechts een kleine hoeveelheid verticale ruimte.

De dimensies van het JSON-veld zijn minder duidelijk af te bakenen. Dit komt door de structuur van een JSON. Deze kan zowel horizontaal als verticaal uitbreiden, zoals in 2.1.1 werd besproken. Doordat een JSON vrij is in welke richting deze kan uitbreiden, wordt verder gewerkt onder de veronderstelling dat een JSON-visualisatie eender welke rechthoekige vorm kan aannemen.

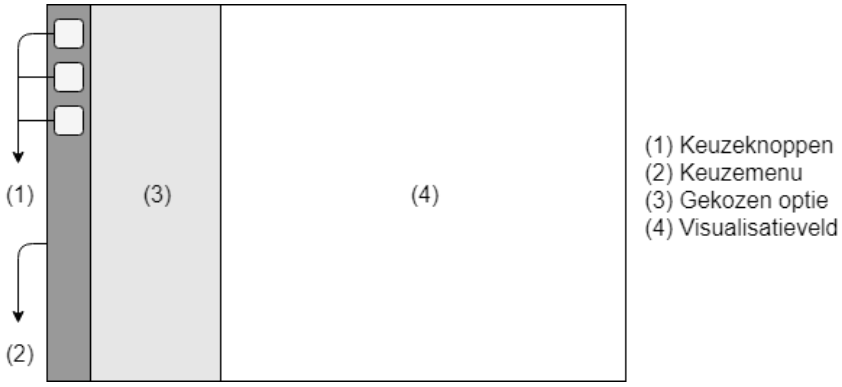
Het visualisatieveld toont de visualisaties aan de gebruiker. Dit is ook het centerstuk van de applicatie. Visualisaties kunnen allerlei ruimtelijke eigenschappen hebben en op verschillende manieren verdeeld worden over het dashboard. Deze eigenschappen van de visualisaties maken dat het visualisatieveld best een zo groot mogelijk oppervlak voor zich neemt.

Rekening houdend met de verschillende eigenschappen van de drie basisonderdelen van de visualisatiepagina, werd gekozen om een indeling van het scherm te organiseren zoals zichtbaar is in figuur 56. Hierbij wordt het URL-veld bovenaan de pagina geplaatst. Er wordt plaats gelaten voor het JSON-veld dat gebruik kan maken van de extra verticale ruimte. De meeste ruimte wordt voorzien voor het visualisatieveld.



56. Visualisatiepagina

In dit design is het JSON-veld permanent in beeld en ontneemt ruimte dat anders gebruikt zou kunnen worden voor het visualisatieveld. Een mogelijke verbetering hierop zou zijn om het JSON-veld te vervangen door een keuzemenu. In dit keuzemenu zou dan kunnen gekozen worden om het JSON-veld of het URL-veld te tonen. Hierdoor kan bijna het hele scherm worden gebruikt om visualisaties te maken en ordenen. De kanttekening hieraan is dan weer dat de functionaliteiten minder direct te bereiken zijn. In essentie zouden de I/O functionaliteiten die nu verspreid zijn tussen het JSON-veld en het visualisatieveld ook in het keuzemenu worden geplaatst. Dit creëert meer samenhang en bundelt de functionaliteiten van de applicatie op een geordende manier samen. Een representatie van dit alternatief is uitgebeeld in figuur 57.



57. Alternatief design

2.5.3 Visualisatieveld

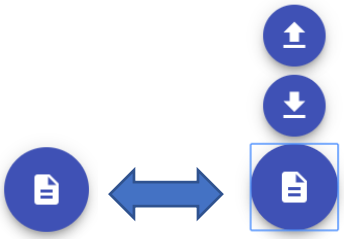
Het visualisatieveld is het centrum van het visuele gedeelte van de applicatie. Dit gedeelte van het design neemt enkele aspecten van de applicatie voor zijn rekening:

- Het tonen van visualisaties;
- Vragen aan de gebruiker of gelijkaardige bronnen in de JSON moeten worden toegevoegd aan de grafiek;
- Het inladen van templates in de globale toestand;
- Het opslaan van visualisatietemplates op de computer.

2.5.3.1 Opslaan en ophalen van data

Om visualisaties op te slaan en in te laden in de applicatie is het logisch dat hiervoor knoppen worden voorzien. Omwille van beperkingen in de applicatie op het ogenblik dat deze werd ontworpen, moesten deze zich dicht tegen de bron bevinden waar alle ontvangen data wordt verwerkt. Op termijn zou deze locatie binnen de applicatie er niet toe mogen doen.

Omdat deze component zich in het visualisatieveld moest bevinden werd gekozen om een UI-element te creëren dat vergelijkbaar is met de toevoegknop van Android toestellen in die zin dat het een ronde knop is dat zich rechts onderaan het scherm bevindt. Met de knop moet het mogelijk zijn om zowel templates in te laden als op te slaan op de computer. Daarom werd besloten om het proces in twee stappen te laten verlopen. Een representatie van de knop is weergegeven in figuur 58.



58. Knop om op te slaan en af te halen

In eerste instantie moet de activatieknop worden ingedrukt om de knoppen voor het opslaan en afhalen te laten verschijnen. De bovenste knop is die voor het opslaan, die daaronder voor het afhalen. Dit werd zo ontworpen om zo min mogelijk plaats in te nemen die anders vrij beschikbaar zou moeten zijn voor de visualisaties. Wanneer op de knop om op te slaan wordt geduwd, moet de gebruiker wel nog een bestandsnaam meegeven. Hiervoor wordt een eenvoudige pop-up gebruikt. De UI voor het beslissen waar een bestand weg te schrijven of in te laden is die van de *filesystem*[66] API van Node.js.

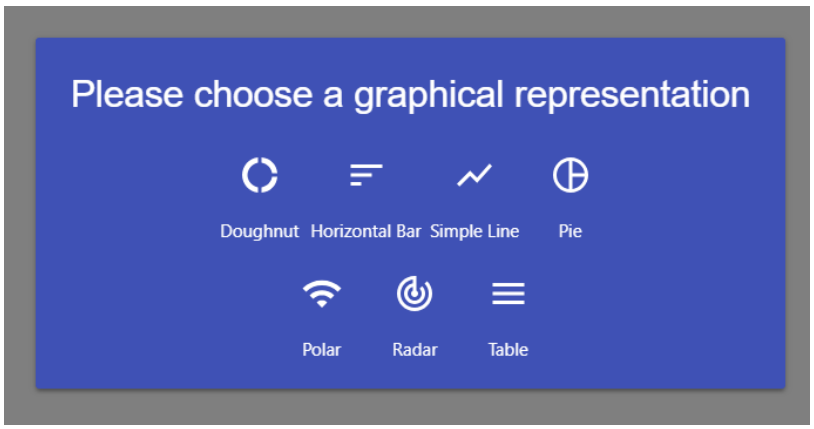
2.5.3.2 Kiezen van een visualisatie

Nadat een gebruiker gegevens heeft versleept uit de JSON-visualisatie en heeft losgelaten in visualisatieveld krijgt deze voor het eerst de kans om een weergave voor de gegevens te kiezen. Deze manier van werken werd verkozen omwille van pragmatische redenen. De applicatie weet uiteindelijk pas welke gegevens een gebruiker wenst te visualiseren nadat deze een versleepactie heeft gestart. Of deze gegevens effectief gevisualiseerd moeten worden, is pas duidelijk nadat de gebruiker deze definitief heeft losgelaten in het dashboard. Het is pas op dit ogenblik dat een inschatting van de gegevens kan gemaakt worden.

Wanneer de gegevens definitief gekend zijn, kan de applicatie verder met het bepalen welke visualisatie gepast zouden zijn voor de gegevens. Een bijkomstig voordeel hieraan is dat de gebruiker zich nooit moet afvragen welke visualisaties bruikbaar zijn voor de data die hij of zij wenst gevisualiseerd te krijgen. De keerzijde is wel dat de gebruiker niet op voorhand weet welke visualisaties hij of zij ter beschikking zal krijgen.

De visualisaties die in de applicatie als toepasselijk zijn gevonden voor een gegeven dataset worden vertoond aan de gebruiker aan de hand van een *modal*[67]. In deze modal worden dan een reeks iconen vertoond met daaronder tekst. Het icoon is bedoeld om op een snelle manier een bepaalde visualisatie te kunnen herkennen. De tekst dient ter ondersteuning in die gevallen dat het icoon niet voldoende duidelijk is. Dit onderdeel is weergegeven in onderstaande figuur.

[66] Een API van node die allerlei functies voorziet voor het schrijven, lezen, manipuleren van bestanden op de computer.
[67] Een modal is gelijkaardig aan een pop-up. Een modal neemt de voorgrond van het scherm in beslag en maakt tijdelijk de rest van de applicatie onbeschikbaar totdat aan een bepaalde voorwaarde is voldaan.

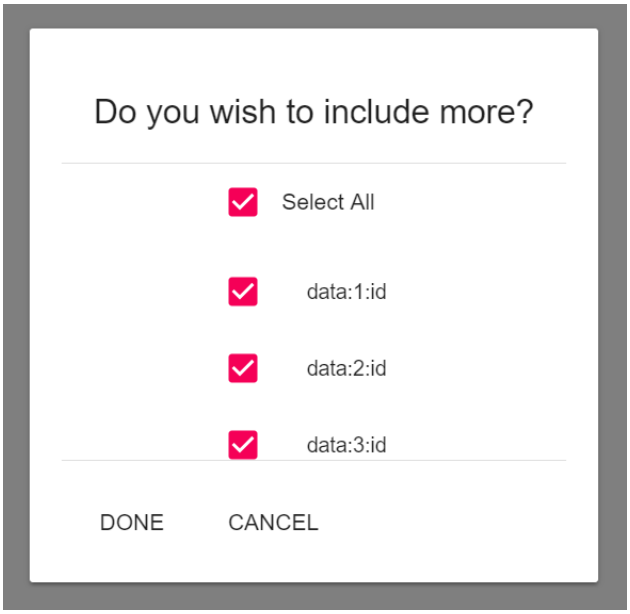


59. Keuze van de visualisatie

2.5.3.3 Andere bronnen toevoegen

Een van de functionaliteiten die gewenst was, was het kunnen toevoegen van gelijkaardige bronnen van data die zich in de JSON bevinden. Wanneer gelijkaardige bronnen worden gedetecteerd, dan wordt een modal gegeneerd dat de gebruiker hiervan op de hoogte brengt. Hierbij moet de gebruiker kunnen aanduiden of ze extra bronnen wilt toevoegen, en zo ja, welke.

Het design van de component is relatief eenvoudig en bestaat in hoofdzaak uit een lijst van bronnen waaruit meerdere kunnen worden aangeduid, alsook een knop om alle bronnen toe te voegen. Als de gebruiker gelukkig is met de keuze dan hoeft deze enkel op een confirmatieknop te duwen. Als geen extra bronnen moeten worden toegevoegd, dan moet enkel op een annuleerknop worden gedrukt. Het design van de component wordt verduidelijkt in onderstaande figuur.

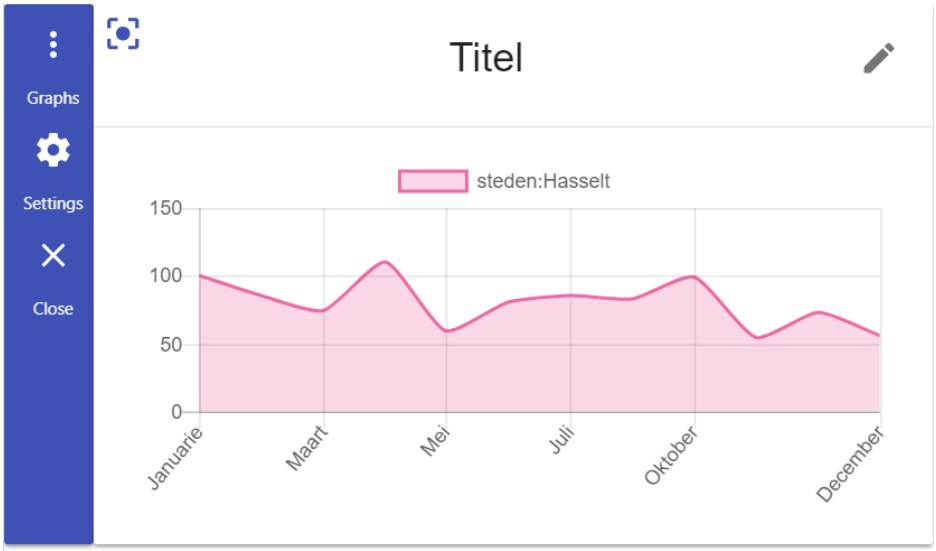


60. Toevoegen van bronnen

2.5.4 Visualisatieblok

Een visualisatieblok, weergegeven in figuur 61, is een component dat als een container werkt voor een visualisatie. Een visualisatieblok is vergelijkbaar in opbouw aan dat van het dashboard. Hoewel de werking van beide zeer verschillend is. Net als het dashboard, is de visualisatieblok opgedeeld in drie segmenten:

- Een keuzemenu-gedeelte dat uiterst links in het blok aanwezig is, waarmee andere visualisaties voor het blok kunnen gekozen worden of toestaat het blok te verwijderen;
- Een inputveld dat zich bovenaan het blok bevindt, waarmee het een titel kan gegeven worden;
- Een visualisatiegedeelte dat de resterende ruimte inpalmt en waar de datavisualisaties worden getoond.



61. Visualisatieblok design

Het keuzemenu werd in eerste instantie toegevoegd aan de applicatie om het mogelijk te maken om van visualisatie te kunnen wisselen zonder hiervoor telkens een nieuwe visualisatie te hoeven aanmaken. Daarnaast zou dit keuzemenu ook gebruikt kunnen worden om personalisatie van de visualisatie toe te laten. Dit laatste is nog niet aanwezig in de applicatie. Ten laatste kan met behulp van dit keuzemenu de visualisatie ook vernietigd worden.

De titel en visualisatie zijn, ten opzichte van het keuzemenu, relatief eenvoudig. Het doel van de titel is enkel om een naam te kunnen geven aan de gecreëerde visualisatie. De visualisatie zelf is een React-component dat al dan niet reageert op gebruikersinput. Deze componenten kunnen van externe bronnen afkomstig zijn of eigen creaties.

In bovenstaande figuur zijn ook twee iconen te zien. Het icoon aan de linkerzijde is een ankerpunt voor de visualisatie. Deze dient om de gebruiker in staat te stellen de visualisatie over het dashboard te kunnen verplaatsen. Het icoon aan de rechterzijde zorgt ervoor dat een gebruiker, nadat deze al een titel heeft opgegeven, een nieuwe titel kan geven aan de visualisatie.

Ten slotte is de visualisatie ook herschaalbaar. Hiervoor is geen icoon aanwezig en is niet zichtbaar op bovenstaande figuur. Echter de randen aan de onderkant en de rechterkant van de visualisatie dienen als ankerpunten. Als de cursor over een van deze beweegt, dan zal de gebruiker de cursor van vorm zien veranderen. Dit is om aan te geven dat het anker voor het verschalen van de visualisatie zich onder de cursor bevindt. Het is dan slechts een kwestie van de linkermuisknop ingedrukt te houden en de cursor te verplaatsen over het scherm.

2.6 Visualisatie

In paragraaf 2.3.2 werd gesproken over het specificatiemodel en hoe deze ervoor zorgen dat er in de applicatie flexibel gewerkt kan worden. Bij de uitleg over de structurering werd ook al het een en ander verteld over hoe deze visualisaties in de applicatie zullen vertoond worden. In deze paragraaf zal de connectie tussen de twee worden toegelicht en de belangrijkste functies. De opmaak die gerealiseerd wordt door gebruik van React-componenten en hun parameters worden hierbij buiten beschouwing gelaten. Er zal wel naar gerefereerd worden.

2.6.1 Aanmaak en vernietiging van de visualisatie

Wanneer visualisatiecomponent voor het eerst wordt aangemaakt zijn er twee functies die telkens worden opgeroepen: ‘componentWillMount’ en ‘componentDidMount’. Bij het vernietigen zal de functie ‘componentWillUnmount’ worden opgeroepen. Deze functies zijn levenscyclusfuncties die standaard bruikbaar zijn voor elke React-component dat een klasse is. Deze en andere levenscyclusfuncties zijn niet verplicht om te gebruiken. Het functies die worden opgeroepen bij het aanmaken en vernietigen van de visualisatie worden samen besproken omdat dit korte en eenvoudige functies zijn. De drie functies zijn weergegeven in figuur 62.

```
componentWillMount() {
    const {data, graph, options} = this.getStuff();
    const state = { resize : true };
    this.changeChild(data, graph, options, state);
}

componentWillUnmount(){
    this.card = null;
}

componentDidMount() {
    let offset = this.card.clientHeight;
    this.setState({offset});
}
```

62. Aanmaak- en vernietigingslevenscyclusfuncties

De componentWillMount-functie wordt opgeroepen nog voordat de visualisatiecomponent is verschenen op het scherm. De eerste actie dat hier wordt ondernomen is het verzamelen van gegevens die het zal moeten gebruiken. Deze gegevens zijn de data, het type weergave voor die data en de andere weergaveopties. Deze gegevens worden opgehaald aan de hand van een hulpfunctie. Daarna wordt een object aangemaakt dat aangeeft dat de component later nog eens zijn dimensies moet controleren. De betekenis hierachter wordt in paragraaf 2.6.2 duidelijk. Als laatste zal deze component een hulpfunctie oproepen dat een kindercomponent zal aanmaken. Deze kindercomponent is de interface van de visualisatie dat vertoond wordt door de React-component. Hieraan worden alle verzamelde gegevens doorgegeven. Deze hulpfuncties zijn: ‘getStuff’ en ‘changeChild’.

Wanneer voor het eerst de component op het scherm is verschenen, dan zal de componentDidMount worden opgeroepen. Het enige dat deze functie doet, is achterhalen welke hoogte het titel segment in de visualisatie heeft. Dit wordt bepaald aan de hand van een referentie naar een dat onderdeel in het render-gedeelte van de klasse. Deze wordt later gebruikt om de effectieve omvang van heel de visualisatie te weten te komen. Ten laatste is er nog de componentWillUnmount-functie. Deze functie wordt opgeroepen net voordat de React-component wordt vernietigd. In deze functie wordt de referentie, zou ze nog ergens naartoe wijzen, gelijkgesteld aan null.

2.6.1.1 getStuff

Deze hulpfunctie verzamelt alle gegevens die de visualisatiecomponent nodig heeft om correct op te starten. De gegevens die het moet ophalen zijn:

- De sleutelpaden die gebruikt moeten worden om gegevens uit de JSON te gaan zoeken;
- Het uithalen van de gegevens uit de JSON;
- Het afhalen van het type weergave dat werd gekozen;
- Het berekenen van de weergaveopties.

De sleutelpaden worden uit de Redux-databank van de component gehaald. Hiervoor gebruikt de component zijn eigen unieke ID. De sleutelpaden worden ontvangen als een lijst van sleutelpaden. Daarna wordt een andere functie opgeroepen die aan de hand van een tweedimensionale lijst van sleutelpaden en een JSON als argument de gegevens uit de JSON zal proberen te halen. Er wordt gewerkt met tweedimensionale lijsten van sleutelpaden omdat in voorgaande versies een andere manier van werken werd gehanteerd. Daarna wordt het type weergave, dat zich ook in de Redux-databank bevindt, opgehaald. Hiervoor wordt een hulpfunctie opgeroepen. Het opvragen van het huidige type weergave is ook elders nodig in deze component. Vandaar een hulpfunctie. Ten laatste wordt de validatiefunctie gebruikt om een lijst van geldige weergaven te laten genereren. Hiervoor wordt de gevonden data uit de JSON gebruikt. Van alle berekende gegevens worden de data, type weergave en de weergaveopties teruggegeven door deze functie. De volledige functie is weergegeven in figuur 63.

```
getStuff() {
    const sources  = this.props.state[this.props.id].data;
    const data = loader(this.props.json, [sources])[0];
    const graph = this.getGraph();
    const options = optionsGenerator(data);
    return {
       data,
       graph,
       options
    }
}
```

63. getStuff-functie

2.6.1.2 changeChild

Bij de werking van de changeChild-functie, in figuur 64, kunnen twee werkingsonderdelen geïdentificeerd worden:

- Het bepalen van de dimensies (hoogte en breedte) die de interface zal moeten aanvaarden;
- De correcte interface uitzoeken.

Voor het bepalen van de dimensies voor de interface wordt de standaard dimensie van de weergave opgevraagd uit de specificatiecontainer. Daarna wordt ook de huidige dimensie opgevraagd uit de Redux-databank. Indien een huidige dimensie is opgeslagen in de Redux-databank, dan wordt deze behouden en de standaard dimensie verworpen. Zo niet, dan wordt enkel de standaard dimensie overgehouden.

```
changeChild(data, graph, options, state = {}) {
    const stdDim = visualSpecs.getInterfaceSize( graph );
    const curDim = this.props.state[this.props.id].dimensions;
    let dimensions;

    if ( curDim ) {
        dimensions = curDim;
    } else {
        dimensions = stdDim;
    }

    const child = React.cloneElement( visualSpecs.getInterface( graph ) ,{
       data,
       dimensions
    });

    this.setState({
        ...state,
        child,
        data,
        graph,
        options,
    });
}
```

64. changeChild-functie

In het tweede deel wordt de interface die overeenstemt met het type weergave van de correcte gegevens voorzien. Hiervoor wordt de React.cloneElement-functie gebruikt. Deze functie ontvangt een component en voegt daaraan een object met de data en de dimensie toe. De data zijn de gegevens die uit de JSON werden gehaald.

Uiteindelijk wordt dan de lokale toestand van de visualisatiecomponent bijgewerkt. In de lokale toestand worden dan de data, het weergavetype, de kindercomponent en de weergaveopties toegevoegd in een accumulatorobject dat meegeven werd als parameter van de changeChild-functie. Deze accumulator bevat op zich al enkele parameters die in de lokale toestand opgeslagen moeten worden. Een accumulatorobject wordt gebruikt om het

aantal updates naar de lokale toestand tot een minimum te houden. Zou dit niet worden gebruikt, dan zouden op sommige plaatsen meerdere keren achtereen de lokale toestand worden bijgewerkt.

2.6.2 Hernieuwen van de visualisatie

Bij elk updaten van de visualisatiecomponent wordt de 'componentDidUpdate'-functie, weergegeven in figuur 65, gebruikt om de nodige controles en aanpassingen te maken binnen de component. Deze functie wordt pas opgeroepen wanneer de update van de component volledig is doorgevoerd. Deze functie wordt gebruikt omdat in een eerder stadium van de update – hiervoor kan de 'componentWillUpdate' gebruikt worden – sommige parameters zelf nog geen update hebben doorgaan. Er kan dan ook geen controle op worden uitgevoerd.

In deze functie worden eerst enkele waarden voorbereid:

- De data van de weergave;
- Het weergavetype;
- De string-getransformeerde van de huidige data;
- De string-getransformeerde van de verleden data.

Nadat deze gegevens zijn verzameld worden drie controles sequentieel uitgevoerd:

- Of de 'resize'-parameter in de lokale toestand van de component de waarde true heeft;
- Of het weergavetype in de lokale toestand niet langer gelijk is aan het weergavetype in de Redux-databank
- Of de stringwaarde van de huidige data niet langer gelijk is aan de stringwaarde van de verleden data.

In het eerste geval wordt de nieuwe dimensie van de weergave opgevraagd aan de hand van een hulpfunctie. Deze hulpfunctie gaat aan de hand van enkele if-else-statements na welke dimensie moet worden teruggegeven. Het opvragen van de dimensie is nodig om de component correct van dimensie te doen veranderen in de UI. De verkregen waarde wordt dan opgeslagen in de lokale toestand van de visualisatiecomponent en de resize-parameter wordt op false gezet.

In het tweede geval wordt een object met een resize-parameter aangemaakt dat de waarde true heeft. Er wordt dan gecontroleerd of het nieuwe weergavetype in de huidige lijst van weergaveopties voorkomt. De weergaveopties hoeven hiervoor niet opnieuw bepaald te worden. Dit is omdat de gegevens van de weergave, ongeacht het weergavetype, onveranderd zijn gebleven ten opzichte van de vorige toestand van de component. Deze weergaveopties hoeven enkel aangepast te worden wanneer de JSON waar de data vandaan wordt gehaald, is veranderd. Zou het weergavetype zich niet langer in de weergaveopties bevinden, dan wordt de interface dat onder de child-parameter in de lokale toestand is opgeslagen, vervangen door een errorbericht. Als het weergavetype zich wel in de lijst van weergaveopties bevindt, dan wordt een nieuwe interfacecomponent opgehaald en opgeslagen aan de hand van de changeChild-functie.

In het derde geval wordt de omgekeerde bewerking gemaakt wat betreft het nakijken of het weergavetype zich in de weergaveopties bevindt. In dit geval is het weergavetype gebruikt dat reeds bestaat en wordt deze getoetst met de nieuwe weergaveopties. Zou het weergavetype niet langer deel uitmaken van de weergaveopties, dan wordt opnieuw een errorbericht in de child-parameter van de lokale toestand geplaatst. Als weergavetype nog steeds geldig is, dan wordt de component in de child-parameter vernieuwd door deze van nieuwe parameters te voorzien.

```
componentDidUpdate(prevProps, prevState) {
    const data = this.getData();
    const graph = this.getGraph();

    const strPrevData = JSON.stringify(this.state.data);
    const strCurrData = JSON.stringify(data);

    if (this.state.resize) {
        const resize = this.getSize();
        this.setState({shownHeight : resize, resize : false});
    } else if ( this.state.graph !== graph ) {
        const state = { resize : true };
        if ( contains(graph, this.state.options) ) {
            this.changeChild(data, graph, this.state.options, state);
        } else {
            this.setState({...state, child : <ErrorMessage />});
        }
    } else if ( strPrevData !== strCurrData ) {
        const options = this.getOptions();
        if ( contains(graph, options) ) {
            const child = React.cloneElement(this.state.child,{
                data,
                dimensions : this.props.state[this.props.id].dimensions
            });
            this.setState({
                child,
                data,
                options
            });
         } else {
            this.setState({
                child : <ErrorMessage />,
                data,
                options
            });
        }
    }
}
```

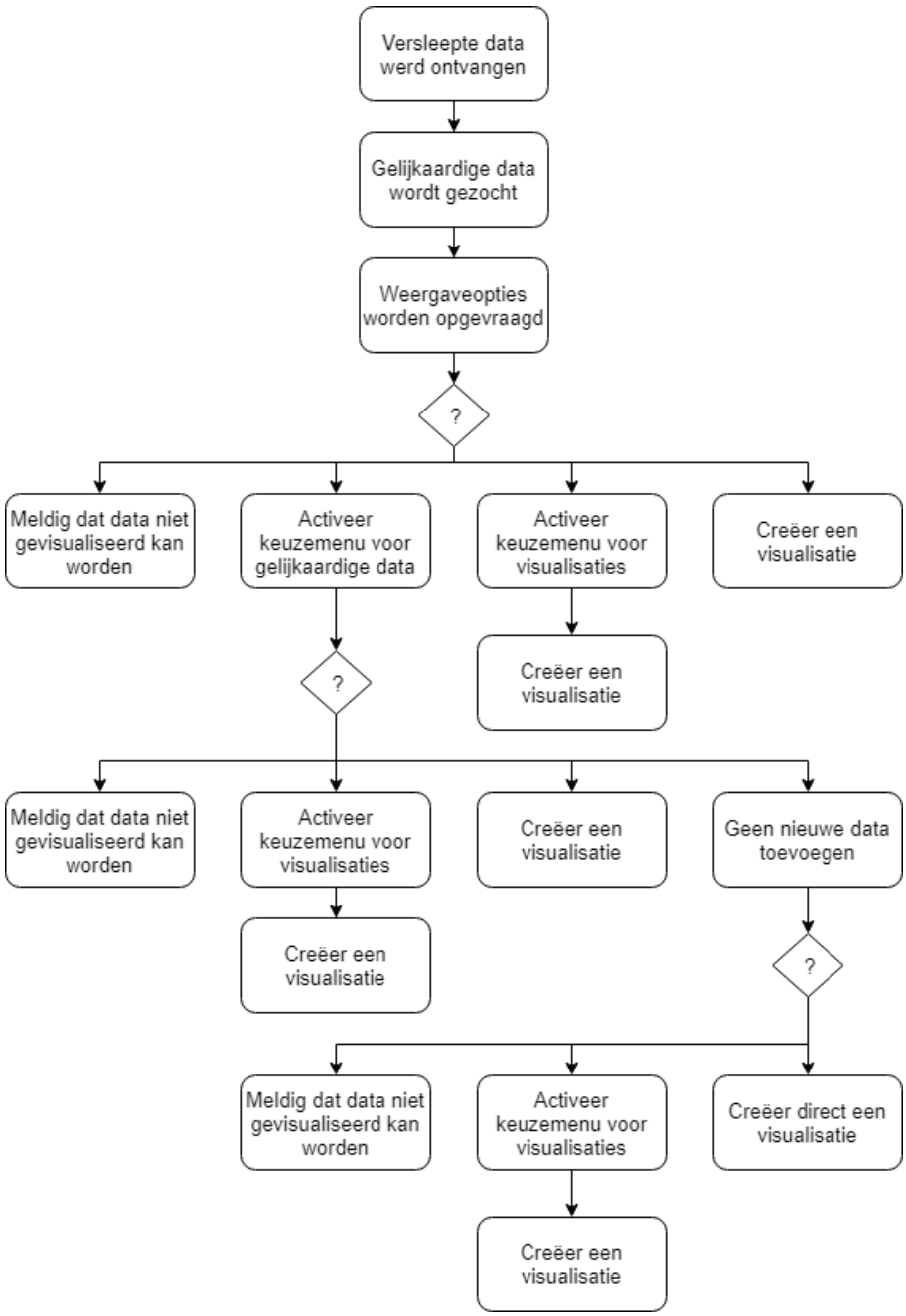
65. componentDidUpdate-functie

2.7 Dashboard

Het dashboard is waar alle werkzaamheden omtrent visualisaties zich bevinden. Dit onderdeel van de applicatie zorgt er niet alleen voor de een gebruiker een visualisatie kan kiezen, maar kan hier ook extra, gelijkaardige gegevens uit de JSON bij de versleepte data voegen. Daarnaast is het hier dat de mogelijkheid bestaat om een dashboard op te slaan of in te laden in de applicatie. Om een idee te geven over hoe versleepte gegevens worden afgehandeld in de applicatie wordt doorverwezen naar figuur 66. Deze figuur is een vereenvoudigde voorstelling van het afhandelingsproces.

In deze paragraaf zullen de belangrijkste onderdelen worden toegelicht. De opmaak en I/O worden hier niet behandeld. De onderdelen die besproken worden, zijn:

- De afhandeling van data dat naar het dashboard werd gesleept;
- Het vinden van extra, gelijkaardige data;
- Het afhandelen van gegevens na extra, gelijkaardige data aan de gegevens te hebben toegevoegd;
- Het afhandelen van de gegevens nadat een visualisatie werd gekozen.;
- Het updaten van het dashboard.



66. Afhandeling van data na ontvangen van versleepte data

2.7.1 Afhandelingsprocedure na verslepen van data

Wanneer een gebruiker gegevens versleept heeft naar het dashboard wordt de functie ‘onDrop’ geactiveerd. Deze functie wordt getoond in figuur 67. Het eerste wat deze functie zal doen is een kopie maken van de sleutelpaden in het versleepgedeelte van de Redux-databank. Een kopie wordt gemaakt omdat kort na het stoppen van de versleepactie de gegevens in de versleeptoestand worden gewist.

Vooraleer verdere bewerkingen plaatsnemen, wordt eerst nagekeken of de kopie een sleutelpad bevat. De onDrop-functie wordt namelijk geactiveerd telkens een versleepactie werd uitgevoerd, maar de herkomst van de versleepactie maakt hiervoor geen verschil. Als deze test faalt, dan wordt er niets gedaan door de functie. Als ze wel slaagt, dan worden verdere verwerkingsstappen ondernomen.

Bij de verwerking worden eerste enkele gegevens verzameld. Eerst zal gezocht worden of er gelijkaardige gegevens in de JSON te vinden zijn. Hiervoor wordt de functie ‘findAllSimilars’ gebruikt. Het antwoord van deze functie is dan een lijst met sleutelpaden die werden gevonden. In dit antwoord bevindt zich wel hetzelfde sleutelpad of dezelfde sleutelpaden die uit de versleeptoestand werden gekopieerd. Dit wordt uit de lijst van de gevonden sleutelpaden verwijderd.

Hierna wordt ook de data waar het sleutelpad of sleutelpaden in de kopie naar leidt uit de JSON gehaald. Met deze data worden dan alle visualisaties opgevraagd die de gegevens kunnen accepteren. Twee extra parameters worden dan aangemaakt. De eerste slaat de booleaanse waarde op uit de vergelijking of de lengte van de lijst van weergaveopties meer dan één is. De tweede slaat de vergelijking of de lengte van de lijst van sleutelpaden die werden gevonden meer dan nul is op. De laatste parameter die wordt gemaakt, is een index. Deze index wordt bepaald door na te gaan welke de hoogste index is dat gekoppeld is aan een visualisatie in het dashboard. De hoogste index wordt dan verhoogd met één.

```
onDrop(){
        const key = this.props.dndData;
        if ( key.length > 0 ) {

            let keys = findAllSimilars(this.props.json, key);
            keys = removeDuplicate(key, keys);
            const options = this.dataOptions( key );
            const choicemodal = options.length > 1;
            const addmodal = keys.length > 0;
            const index = Object.keys(this.state.blocks).reduce((acc, e) => Number(e)
> acc ? Number(e) : acc, 0) + 1;

            if ( addmodal ) {
                this.setState({addmodal, keys, key, options});
            } else if ( choicemodal ) {
                this.setState({choicemodal, keys, key, options});
            } else if (options.length === 1) {
                const then = ( e ) => {
                    this.props.addData( key, index, options[0] );
                    this.setBlock( e, index );
                }
                this.asyncDB({ then, index });
            } else {
                window.alert("Ow! \nWe're sorry but there are currently no
visualisations for the type of data you provided.");
            }
        }
    }
```

67. onDrop-functie

Uiteindelijk wordt dan gekomen aan een if-else-statement. Deze benaderende werking hiervan is weergegeven in figuur 66. Het verloop dat in deze functie wordt uitgevoerd, bevindt zich tussen de eerste en de tweede ruit met een vraagteken in. Er zijn vier delen in deze if-else-statement dat kunnen uitgevoerd worden:

- Als de lengte van de gevonden sleutelpaden groter is dan nul;
- Als de lengte van de weergaveopties groter is dan één;
- Als de lengte van de weergaveopties exact gelijk is aan één;
- Alle andere gevallen.

In het eerste geval is het gewenst om de gebruiker de keuze te geven om de extra gegevens die werden gevonden bij te voegen bij de versleepte sleutelpaden. Hiervoor moet het dashboard wel een update ondergaan. Daarom worden de gevonden sleutelpaden, het versleepte sleutelpad, de weergaveopties en een signaalparameter in de lokale toestand van het dashboard opgeslagen. De signaalparameter is dezelfde waarde die de if-statement heeft geactiveerd. De parameter geeft bij het vernieuwen van het dashboard aan een modal aan dat deze zich moet tonen. Het tweede geval is identiek aan het eerste, maar de waarde die zijn if-statement heeft geactiveerd wordt ditmaal in de lokale toestand bijgehouden om een andere modal te activeren.

In het voorlaatste geval wordt onmiddellijk een weergave aangemaakt. Wanneer er geen kans is om extra gegevens toe te voegen en er maar één weergaveoptie ter beschikking is, is het niet zinvol om de gebruiker een van de twee stappen te laten doorlopen aangezien er geen keuze is om te maken. Het aanmaken van de weergave kan opgedeeld worden in drie onderdelen:

- Een functie dat het aanmaken van visualisatiecomponenten asynchroon zal laten verlopen;

- De functie die de weergave aanmaakt;
- Afhandeling dat wordt uitgevoerd nadat de weergave correct is aangemaakt.

Het asynchroon aanmaken van de visualisaties is een voordeel omdat, afhankelijk van de weergave en de data, een visualisatie wat tijd kan nodig hebben vooraleer deze volledig klaar is om getoond te worden. De functie zelf is weergegeven in onderstaande figuur.

```
asyncDB({ then, index, onChangeGraph = this.props.changeGraph, onRemoveGraph =
this.onRemoveGraph }) {
        asyncwrapper( createSingleDB({
            index,
            onChangeGraph : e => onChangeGraph(e),
            onRemoveGraph : e => onRemoveGraph(e)
        }) )
        .then( e => then(e) );
    }
```

68. Asynchrone functie voor aanmaken van visualisatiecomponenten

De functie zelf is geen asynchrone functie. Daarvoor wordt een tweede functie gebruikt dat normale, sequentiële functies omsluit. Deze functie zorgt ervoor dat elke functie dat een waarde teruggeeft asynchroon zal worden uitgevoerd. Deze functie is weergegeven in figuur 69.

```
const fun = async ( f : Function ): Promise<any> => await f();
```

69. Functie voor het asynchroon maken van andere functies

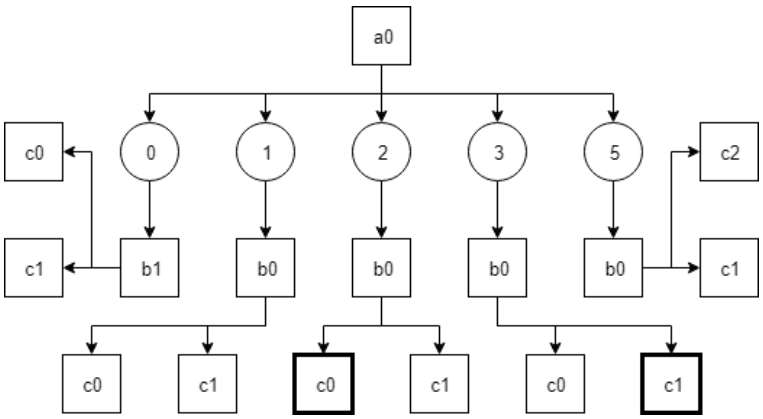
De functie die de weergave zal aanmaken, neemt een index en twee callback-functies aan. De index wordt gebruikt door de visualisatiecomponent om eigen gegevens te kunnen opvragen. De callback-functies zorgen er respectievelijk voor dat de visualisatiecomponent zijn weergavetype kan aanpassen in de Redux-databank en dat hij zichzelf kan verwijderen uit de Redux-databank.

In het voorlaatste onderdeel wordt eerst een actie verstuurd naar de Redux-databank zodat deze de gegevens initieert voor deze weergave. Daarna wordt de bekomen visualisatiecomponent opgeslagen in de lokale toestand van het dashboard op de index van deze component. Als laatste wordt de lokale toestand teruggebracht naar de standaard toestand met uitzondering van de visualisatiecomponenten die erin zijn opgeslagen.

Als uiteindelijk geen van de bovenstaande werd gebruikt, dan wordt een *window alert* geactiveerd. Deze dient om de gebruiker op de hoogte te brengen dat er geen visualisaties konden aangemaakt worden met de gegevens die werden versleept.

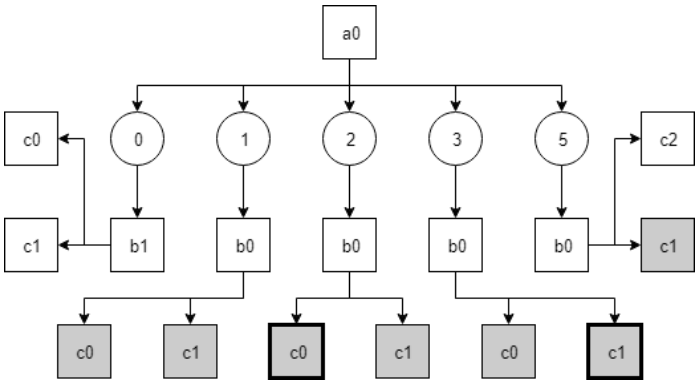
2.7.2 Zoeken van gelijkaardige data

Omdat er redelijk wat code komt kijken bij het achterhalen of er gelijkaardige data is in de JSON, zal deze paragraaf zich beperken tot het principe achter de werking. Het probleem achter vinden van gelijkaardige data in een JSON is eigenlijk een probleem van het vinden van gelijkaardige sleutelpaden die geldig zijn. Een gelijkaardig sleutelpad is geldig wanneer dit wijst naar gegevens in de JSON. Om het principe duidelijker over te brengen, wordt uitgegaan van een hypothetische JSON in figuur 70. In deze figuur stellen de cirkels met getallen indexen in een lijst voor. De vierkanten zijn objecten met een bepaalde sleutel. De vierkanten met een dikkere rand dan de andere stellen de versleepte data voor.



70. Voorbeeld JSON

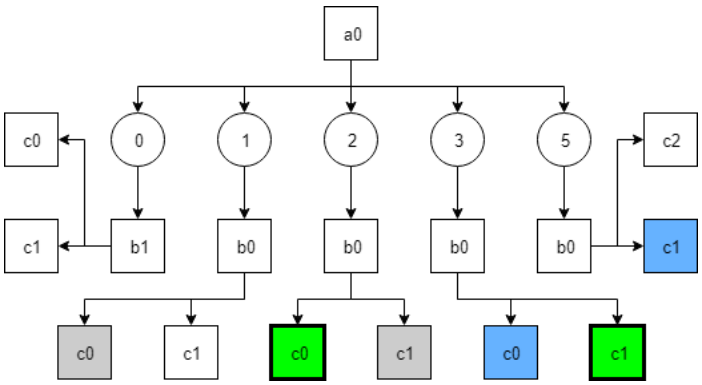
Om te beginnen kan niet elk sleutelpad leiden tot gelijkaardige sleutelpaden. Dit kan alleen maar wanneer in dit sleutelpad tekenen worden gevonden dat er over een lijst is gegaan. Een sleutelpad dat enkel objecten doorloop is uniek in een JSON. Wanneer een lijst moet doorlopen worden is duidelijk wanneer zich één of meerdere cijfers bevinden in het sleutelpad. Er is echter nog een beperking die wordt geplaatst op het vinden van gelijkaardige sleutelpaden. De gegevens die in de lijst of lijsten zitten, moeten JavaScript-objecten zijn. Deze beperking wordt gesteld omdat dan kan worden uitgegaan van uniforme data. Anders zouden elders extra controles moeten worden toegevoegd die elk gegeven uit de lijst moet nagaan of dit een object is. Vanuit deze definitie van gelijkaardige sleutelpaden zouden de grijze blokken in figuur 71 allemaal geldig zijn.



71. Geldige sleutelpaden

Waarom zijn deze waarden in de JSON geldig, maar niet andere waarden die ‘c0’ of ‘c1’ zijn? Dit is omdat, met uitzondering van lijsten, het sleutelpad in zijn exacte vorm moet doorlopen worden. Enkel op die punten in de JSON dat een lijst wordt gevonden, mag deze diepte in de JSON als een variabele worden beschouwd. En wordt dus over elke index van de lijst geïtereerd. Op dit punt werd er nog een beperking gesteld. Om context te kunnen bewaren, werd er gekozen om enkel de laatste lijst in beschouwing te laten. Elke andere lijst wordt behandeld alsof dit een object is.

Op dit punt is een lijst van sleutelpaden bekomen. Maar er is ook nog een verhouding tussen de twee versleepte sleutelpaden waar rekening mee moet worden gehouden. Ze komen namelijk uit twee verschillende indexen in de lijst. Hier moet ook rekening mee gehouden worden. Praktisch komt dit erop neer dat als er meer dan één sleutelpad aanwezig is, zal er gekeken worden of er combinaties van data vallen te maken met de correcte offset ten opzichte van elkaar. In de voorbeeld JSON zouden deze combinaties er uit zien zoals aangegeven in onderstaande figuur.



72. Geldige combinaties in de JSON

In de figuur valt ook op dat een van de datapunten in de JSON dat tot voor kort nog als geldig werd aanvaard, terug is gekeerd naar een niet-aanvaarde status. Dit komt omdat voor dit datapunt geen tegenhanger kan gevonden worden met een geldige offset.

2.7.3 Afhandelingsprocedure na kiezen van extra gegevens

Er zijn twee opties waaruit de gebruiker kan kiezen wat betreft het toevoegen van extra gegevens: accepteren om gegevens bij te voegen of de stap gewoonweg overslaan. Bij het overslaan worden enkel de versleepte gegevens omgevormd in visualisatiecomponenten.

2.7.3.1 Toevoegen van gegevens

Indien de gebruiker ervoor kiest om extra gegevens bij te voegen dan wordt de ‘onDone’-functie opgeroepen. Deze functie is te zien in figuur 73. Deze functie zal eerst nagaan of de gebruiker wel een keuze heeft gemaakt. Zou de gebruiker geen extra keuzes hebben gemaakt, dan wordt dezelfde functie opgeroepen als voor het overslaan. Als er wel keuzes zijn gemaakt, dan moeten eerst de lijsten van sleutelpaden die werden gekozen, geïsoleerd worden uit de datadrager. Deze datadrager is een object met als sleutels een lijst van sleutelpaden. De data in dit object is een booleaanse waarde die aangeeft of lijst werd gekozen of niet. Daarnaast moet ook worden gekeken of elke index lijst meer dan één sleutelpad bevat. Als elke index van de lijst meer dan één sleutelpad bezit, dan moet een andere reeks van bewerkingen worden gestart dan als maar één sleutelpad aanwezig is. Als laatste voorbereidende stap wordt de hoogste index in de lokale toestand bepaald en vermeerderd met één.

In het geval dat er meerdere sleutelpaden aanwezig zijn per index van de lijst, dan worden de versleepte sleutelpaden vooraan de lijst bijgevoegd. Een interne functie wordt dan aangemaakt waarin een for-loop is gedefinieerd. Deze for-loop zal bij elke iteratie een nieuwe visualisatiecomponent aanmaken waarbij het ID van deze component de optelling is van de index die reeds werd bepaald plus de huidige waarde van de iteratiestap. Deze interne functie wordt dan overgedragen aan een externe functie. Hiernaast ontvangt deze externe functie nog de weergaveopties en de totale tweedimensionale lijst van sleutelpaden.

In het andere geval wordt de tweedimensionale lijst van sleutelpaden gereduceerd tot een eendimensionale lijst van sleutelpaden. In dit geval is het wel nodig om opnieuw na te gaan of er visualisaties zijn die de nieuwe combinatie van gegevens kunnen aanvaarden. Net als in de vorige stap, wordt ook hier een interne functie gemaakt, maar dan zonder for-loop. Wederom wordt dezelfde externe functie opgeroepen.

```
onDone(){
    if(this.state.chosen){
        const chosen0 = Object.keys(this.state.chosen).filter(e =>
this.state.chosen[e] === true).map(JSON.parse);
        const numChosen = chosen0[0].length;
        const index = Object.keys(this.state.blocks).reduce((acc, e) => Number(e) >
acc ? Number(e) : acc, 0) + 1;
        if ( numChosen > 1 ) {
            const options = this.state.options;
            const chosen = [ this.state.key ].concat( chosen0 );
            const len = chosen.length;
            const fun = () => {
               for ( let i = 0; i < len; i++ ) {
                    const iindex = index + i;
                    const then = ( e ) => {
                        this.props.addData( chosen[i] , iindex, options[0] );
                        this.setBlock( e, iindex );
                    };
                    this.asyncDB({ then, index : iindex });
                }
            }
            this.generalDone( fun, options, chosen );
        } else {
            const chosen1 = chosen0.reduce((acc, e) => acc.concat(e),[]);
            const chosen = this.state.key.concat(chosen1);
            const options = this.dataOptions( chosen );
            const fun = () => {
                const then = ( e ) => {
                    this.props.addData( chosen, index, options[0] );
                };
               this.asyncDB({ then, index });
            }
            this.generalDone( fun, options, chosen );
        }
    } else {
        this.onCancel();
    }
}
```

73. onDone-functie

De externe functie, ‘generalDone’, neemt de volgende stap van de procedure op zich. Deze functie, weergegeven in onderstaande figuur, zal nagaan welke handelingen moeten worden genomen afhankelijk de lengte van de weergaveopties. Indien deze lengte groter is dan één, dan zal deze de lokale toestand bijwerken om aan te geven dat de gebruiker een keuze kan maken over welke visualisatie deze wil gebruiken. Als de lijst van weergaveopties maar één optie bevat, dan wordt deze stap overgeslagen en wordt de interne functie van voorheen geactiveerd. Als uiteindelijk geen enkele weergaveoptie bestaat voor de sleutelpaden, dan krijgt de gebruiker hier een melding over via de window.alert-functie.

```
generalDone( fun, options = this.state.options, key = this.state.key ){
    if ( options.length > 1 ) {
        this.setState({
            ...defaultState,
            key,
            blocks : this.state.blocks,
            choicemodal : true,
            options
        });
    } else if ( options.length === 1 ) {
        fun();

    } else {
        window.alert("Ow! \nWe're sorry but there are currently no visualisations for
the data you provided.");
        this.setState({...defaultState, blocks : this.state.blocks});
    }
}
```

74. Externe functie

2.7.3.2 Toevoegen overslaan

Bij het overslaan van het toevoegen van gegevens wordt de ‘onCancel’-functie opgeroepen, in figuur 75. Deze functie zal nagaan welke index (ID) moet worden aangemaakt voor de visualisatiecomponent. Net als bij de onDone-functie maakt deze een interne functie aan die het asynchroon aanmaken van een visualisatiecomponent afhandelt. Als laatste start deze de externe ‘generalDone’-functie, waaraan het de interne functie meegeeft.

```
onCancel(){
    const index = Object.keys(this.state.blocks).reduce((acc, e) => Number(e) > acc ?
Number(e) : acc, 0) + 1;
    const fun = () => {
        const then = ( e ) => {
            this.props.addData( this.state.key, index, this.state.options[0] );
        };
        this.asyncDB({ then, index });
    }
    this.generalDone( fun );
}
```

75. onCancel-functie

2.7.4 Afhandelingsprocedure na kiezen van een visualisatie

De ‘choiceMade’-functie wordt opgeroepen telkens wanneer een gebruiker een visualisatie heeft gekozen. Deze is weergegeven in figuur 76. Hierbij moet deze eerst nagaan of de sleutels een eendimensionale dan wel tweedimensionale lijst van sleutelpaden is. Dit komt omdat de keuze om extra gegevens toe te voegen voor deze stap wordt ondernomen. Het kan dus voorkomen dat de choiceMade-functie te maken zal krijgen met de tweedimensionale variant.

Indien een tweedimensionale lijst van sleutelpaden wordt gevonden, dan zal de functie voor elke index in de lijst een nieuwe visualisatiecomponent asynchroon aanmaken. In het andere geval kan de eendimensionale lijst van sleutelpaden direct worden gebruikt om een visualisatiecomponent aan te maken.

```
choiceMade({type, id}){
    const keys = this.state.key;
    const isarray = keys[0] instanceof Array;
    if ( isarray ) {
        const len = this.state.key.length;
        for ( let i = 0; i < len; i++ ) {
            const index = id + i;
            const then = ( e ) => {
                this.props.addData( keys[i] , index, type );
                this.setBlock( e, index );
            };
            this.asyncDB({ then, index });
        }
    } else {
        const then = ( e ) => {
            this.props.addData( keys, id, type );
        };
        this.asyncDB({ then, index : id });
    }
}
```

76. ChoiceMade-functie

2.7.5 Updaten van het dashboard

Bij elke verandering van de Redux-database of van de lokale toestand van de dashboardcomponent wordt de componentDidUpdate-functie, in onderstaande figuur, van React geactiveerd. Deze functie zal nagaan of er minder visualisatiecomponenten zijn in de Redux-databank dan in de lokale toestand van de dashboard component. Als dit zo is, worden de overtollige visualisatiecomponenten uitgezocht en verwijderd uit de lokale toestand. In alle andere gevallen zal er worden nagegaan of alle sleutels in de Redux-toestand dezelfde zijn als in de lokale toestand. Zou dit niet zo zijn, dan worden de ontbrekende componenten aangemaakt en in de lokale toestand geplaatst. Bij de volgende iteratie worden eventueel overtollige visualisatiecomponenten verwijderd.

```
componentDidUpdate(){
    let blocks = {...this.state.blocks};
    const statekeys = Object.keys(this.state.blocks);
    const statelen = statekeys.length;
    const propkeys = Object.keys(this.props.data);
    const proplen = propkeys.length;
    const strstate = JSON.stringify( statekeys );
    const strprops = JSON.stringify( propkeys );

    if(proplen < statelen){
        for(let i = 0; i < statelen; i++){
            if( !(statekeys[i] in this.props.data) ){
                const k = statekeys[i];
                blocks[k] = null;
                delete blocks[k];
            }
        }
        this.setState({...defaultState, blocks})
    } else {
       for(let i = 0; i < proplen; i++){
           const index = propkeys[i];
           if( !(index in this.state.blocks) ){
                const then = ( e ) => this.setBlock( e, index
               this.asyncDB({ then, index });
            }
         }
     }
}
```

77. Updatefunctie van het dashboard

2.8 Bestandsformaat

Het bestandformaat dat wordt gebruikt om visualisaties op te slaan in JSON-bestanden en terug uit te lezen, is een exacte kopie van het onderdeel van de Redux-databank dat de gegevens van individuele visualisatiecomponenten overziet. In de Redux-databank bestaat elke visualisatie uit volgende elementen:

- Een titel;
- De sleutelpaden die de visualisatie nodig heeft om zijn data af te halen;
- Het weergavetype;
- De positie waar het zich bevindt op het scherm met een x- en y-coördinaat;
- De grootte van de visualisatie met een hoogte en breedte parameter.

Al deze gegevens zijn gebundeld in een object voor elke visualisatie. De gegevens voor de titel, de positie en de grootte van de visualisatie zijn hierbij optioneel. Alle visualisaties zijn op hun beurt opgeslagen in een object. De sleutel waarop deze gegevens zich in dit object bevinden, is de ID van de visualisatie. Op deze manier kan elke visualisatie eenvoudig aan zijn eigen gegevens geraken.

2.8.1 Testen van het template

Als een bestand wordt afgehaald om visualisaties mee te maken, dan is het belangrijk om te weten of dit bestand wel voldoet aan de opmaak van een templatebestand. Er moet dus een controle op worden uitgevoerd. Deze controle is opgesplitst in enkele stappen:

- Controleren of het uitgelezen bestand wel een object is;
- Controleren of elk gegeven in dit object een object is;
- Controleren of de alle onderdelen van een visualisatieobject wel de juiste parameters heeft, die op hun beurt het juiste type gegevens bevatten;

In de eerste stap wordt de functie in figuur 78 gebruikt. Hier wordt er gecontroleerd of het uitgelezen bestand wel een object is, en als dat zo is, dan wordt over elk onderdeel van dit object geïtereerd. Elke waarde in dit object wordt dan doorgegeven aan een tweede functie, in figuur 79, dat zal nakijken of de verplichte en optionele velden van dit object de correcte datatypes hebben. Deze controle functies zijn afgebeeld in figuur 80.

```
const f = ( t : any ) : boolean => {
       if ( typeof t === 'object' && t !== null ) {
         const ks = Object.keys( t );
               const len = ks.length;
               for ( let i = 0; i < len; i++ ) {
                      if ( !ob( t[ks[i]] ) ){
                             return false;
                      }
               } return true;
       } return false;
}
```

78. Eerste controle

```
const ob = ( o : Vis ) : boolean => {
       if ( typeof o === 'object' && o !== null ) {
              const d = data( o.data );
              const g = graph( o.graph );
              const di = dimensions( o.dimensions );
              const t = title( o.title );
              const p = position( o.position );
              return d && g && di && t && p;
       } return false;
}
```

79. Tweede controle

```
const data = ( d : Array<string> ) : boolean => {
	if ( d instanceof Array ) {
		const len = d.length;
		for ( let i = 0; i < len; i++ ) {
			const j = d[i];
			if ( typeof j !== 'string' ) {
				return false;
			}
		} return true;
	} return false;
}

const graph = ( g : string ) : boolean => {
	if ( typeof g === 'string' ) {
		return true;
	} return false;
}

const __obj = ( f : Function, d : Dim | Pos ) : boolean => {
	if ( typeof d === 'object' && d !== null ) {
		return f();
	} else if ( d === undefined ) {
		return true;
	} return false;
}

const dimensions = ( d : Dim ) : boolean =>  {
	return __obj( () => d.width && d.height, d );
}

const title = ( t : string ) : boolean => {
	if ( typeof t === 'string' || t === undefined ){
		return true;
	} return false;
}

const position = ( p : Pos ) : boolean => {
	return __obj( () => p.x && p.y, p );
}

const ob = ( o : Vis ) : boolean => {
	if ( typeof o === 'object' && o !== null ) {
		const d = data( o.data );
		const g = graph( o.graph );
		const di = dimensions( o.dimensions );
		const t = title( o.title );
		const p = position( o.position );
		return d && g && di && t && p;
	} return false;
}
```

80. Controle op het type van de gegevens

2.9 I/O

Communicatie tussen het programma en de applicatie bestaat uit drie onderdelen:

- Een functie voor het opslaan van nieuwe bestanden;
- Een functie voor het overschrijven van bestaande bestanden;
- Een functie voor het uitlezen van een bestand.

2.9.1 Opslaan van gegevens

Voor het opslaan van een nieuw bestand maak de 'saveTemplateFile'-functie gebruik van drie parameters dat het ontvangt: de naam dat aan het bestand moet worden gegeven, de gegevens die moeten opgeslagen worden en een callback-functie om het success van de operatie te kunnen doorgeven. Deze functie is weergegeven in figuur 81.

```
const saveTemplateFile = (name, data, callback) => {
    let path = window.dialog.showOpenDialog({
        properties:['openDirectory']
    });

    if(path instanceof Array){
        path = path[0] + '\\' + name + extension;
        let template = JSON.stringify(data);

        if(window.fs.existsSync(path)){
            confirmer(path, template, callback);
        }
        else{
            window.fs.appendFile(path, template, 'utf8', (err) => {
                if(err){
                    callback('failed');
                }
                else{
                    callback('success');
                }
            })
        }
    }
    else{
        callback('failed');
    }
}
```

81. saveTemplateFile-functie

Het eerste wat deze functie zal doen, is een venster openen waarmee de gebruiker de locatie van het bestand zelf kan uitzoeken. Deze operatie geeft onder normale omstandigheden een lijst terug met daarin het pad naar de gekozen bestandslocatie. Zou dit falen wordt de callback-functie gebruikt en een gefaalde status doorgegeven. Aan het gekozen pad worden de bestandsnaam, die de gebruiker elders in de applicatie heeft moeten opgegeven, en de bestandsextensie toegevoegd. De data dat aan de functie werd gegeven, wordt hier ook omgezet in een string.

Er wordt dan gekeken of het bestand dat de gebruiker wil aanmaken al bestaat. Als dit zo is, dan wordt een andere functie opgeroepen. Zo niet, dan wordt het nieuwe bestand aangemaakt met behulp van een functie uit Node.js. Deze functie aanvaart een callback-functie. In deze callback-functie wordt de callback-functie van saveTemplateFile opgeroepen. Hier zal het succes of falen van het opslaan aan worden meegegeven.

2.9.2 Overschrijven van gegevens

Het overschrijven van gegevens verloopt in twee stappen:

- De gebruiker melden dat het bestand al bestaat en of deze het wil overschrijven;
- Het overschrijven van het bestand;

De eerste wordt gedaan in een aparte functie. In deze functie zal een window.confirm worden gelanceerd dat de gebruiker de keuze geeft om het bestand te overschrijven of niet. Als de gebruiker aangeeft het bestand te willen overschrijven, dan wordt een andere functie opgeroepen. Zo niet, dan worden geen verdere stappen ondernomen.

De functie die effectief het bestand zal overschrijven is de 'overrideTemplateFile'-functie. Deze functie ontvangt:

- Het pad waar het bestand zich bevindt;

- De data in de vorm van een string;
- Een callback-functie.

Het enige dat deze functie zal doen, is de gepaste Node.js functie activeren om bestanden te kunnen overschrijven. Deze Node-functie aanvaard dezelfde parameters als die voor het maken van een nieuw bestand. De callback-functie van de Node-functie wordt op dezelfde manier afgehandeld als bij het opslaan van gegevens. De functies voor het vragen van gebruikersinput en het overschrijven van gegevens, zijn weergegeven in onderstaande figuur.

```
const overrideTemplateFile = (path, template, callback) => {
    window.fs.writeFile(path, template, 'utf8', (err) => {
        if(err){
            callback('failed');
        }
        else{
            callback('success')
        }
    })
}

const confirmer = (path, data, callback) => {
    let confirm = window.confirm('File already exists. Do you wish to override the
file?');
    if(confirm){
        overrideTemplateFile(path, data, callback);
    }
}
```

82. Proces voor het overschrijven van bestanden

2.9.3 Afhalen van gegevens

Het afhalen van gegevens uit een bestand verloopt onafhankelijk van de voorgaande functies. De ‘loadTemplateFile’-functie ontvangt hiervoor ook enkel een callback-functie. Deze functie is weergegeven in onderstaande figuur. Bij het starten van deze functie zal er eerst weer een dialoogvenster worden geopend. Ditmaal worden er wel filters aan toegevoegd om ervoor te zorgen dat enkel bestanden met een JSON-extensie mogen gekozen worden.

```
const loadTemplateFile = (callback) => {

    let path = window.dialog.showOpenDialog({
        filters:[{
            name:'json',
            extensions:['json']
        }],
        properties:['openFile']
    });

    if(path instanceof Array){
        path = path[0];

        window.fs.readFile(path, 'utf8', (err, data) => {
            if(err){
                callback('failed')
            }
            else{
                try{
                    let json = JSON.parse(data);
                    callback(json);
                }
                catch(error){
                    callback('failed');
                }
            }
        })
    }
}
```

83. loadTemplateFile-functie

Het resultaat dat verkregen wordt, is wederom een lijst als er succesvol een bestand werd aangeduid. Als een bestand succesvol werd gekozen, dan wordt het bekomen pad gebruikt om via een Node-functie dit bestand uit te lezen. Ook deze functie maakt gebruik van een callback-functie om gegevens terug te geven. Als het bestand kon worden uitgelezen, dan worden de gegevens omgezet naar een JavaScript-object notatie en met de callback-functie van loadTemplateFile teruggegeven. Zo niet, wordt enkel teruggeven dat de bewerking gefaald is.

3 Resultaten

In deze paragraaf zullen de verschillende pragmatische onderdelen van de applicatie besproken worden. Het onderdeel van de UI wordt hier buiten beschouwing gelaten omdat er nog geen *field tests* hebben plaatsgevonden om te achterhalen of de applicatie ook voldoet aan de verwachtingen van het personeel bij Qompium.

3.1 JSON-visualisatie

3.1.1 Inladen van gegevens

3.1.1.1 Inladen van computerbestanden

Om te kunnen bepalen of de JSON-visualisatiecomponent geslaagd is in zijn opzet, wordt deze getest aan de hand van twee verschillende JSON's: een JSON dat begint als een lijst en een JSON dat begint als een object. Om compleet te zijn worden beiden van ruwweg dezelfde data voorzien. Elke JSON krijgt daarbij ook nog eens alle mogelijke datavormen die in een JSON kunnen worden opgeslagen. Deze JSON's zijn weergegeven in figuren 84 en 85. De resultaten van de JSON-visualisatie voor elk van deze JSON's zijn respectievelijk weergegeven in figuren 86 en 87. Voor de volledigheid worden ook de toestand van de Redux-databank zowel voor als na het inladen van één van de gegevens. Dit wordt getoond in figuur 88.

In de resultaten is te zien dat op plaatsen waar voorheen een 'undefined' was ofwel verdwenen zijn ofwel een 'null' zijn geworden. Dit is geen fout van de JSON-visualisatie, maar wel een gevolg van de JSON.parse-functie. Buiten deze eigenaardigheid kan in de resultaten gezien worden dat de bekomen JSON-visualisatie volledig

functioneel is. Het is niet alleen mogelijk om de bovenste laag van een JSON te visualiseren, maar ook elke laag van de JSON.

Bij het veranderen van de toestand van de Redux-databank is ook te zien dat er een overgang wordt gemaakt. Eerst had het JSON-gedeelte van de Redux-databank een andere JSON met andere parameters dan hetzelfde onderdeel dat zich daaronder bevindt. Deze twee gedeelten zijn aangeduid door een rood kader in figuur 88.

```
{
       "string" : "string",
       "number" : 10,
       "null" : null,
       "undefined" : undefined,
       "object" : {
              "string" : "string",
              "number" : 10,
              "null" : null,
              "undefined" : undefined,
              "array" : ["string",10,null,undefined]
       },
       "array" : ["string",10,null,undefined,{
                     "string" : "string",
                     "number" : 10,
                     "null" : null,
                     "undefined" : undefined
              }]
}
```

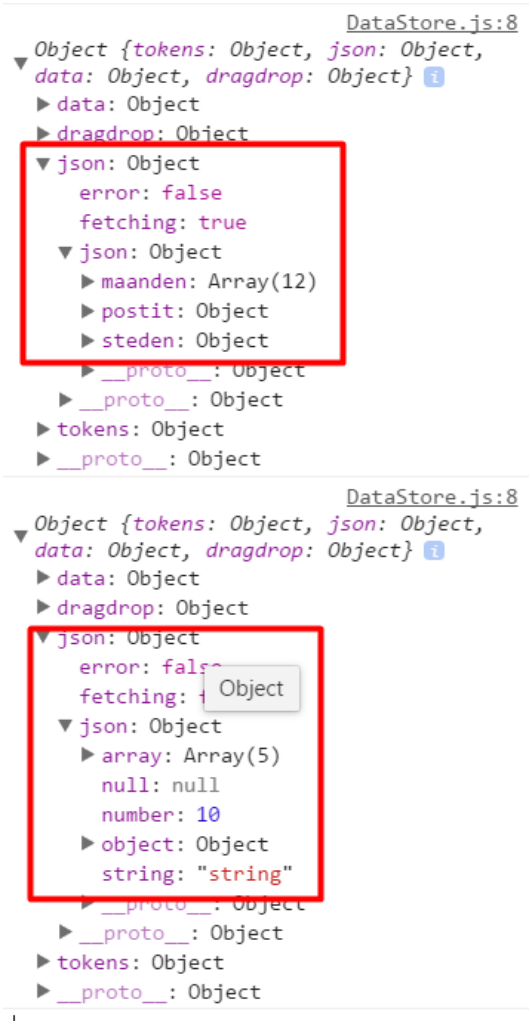
84. JSON dat begint als een object

```
[
    {"string" : "string"},
    {"number" : 10},
    {"null" : null},
    {"undefined" : undefined},
    {"object" : {
              "string" : "string",
              "number" : 10,
              "null" : null,
              "undefined" : undefined,
              "array" : [  "string", 10, null, undefined,]
       }},
    {"array" : [
              "string",
              10,
              null,
              undefined,
              {
                     "string" : "string",
                     "number" : 10,
                     "null" : null,
                     "undefined" : undefined
              }
       ]}
]
```

85. JSON dat begint als een lijst

86. JSON-visualisatie van het JSON-object

87. JSON-visualisatie van de JSON-lijst

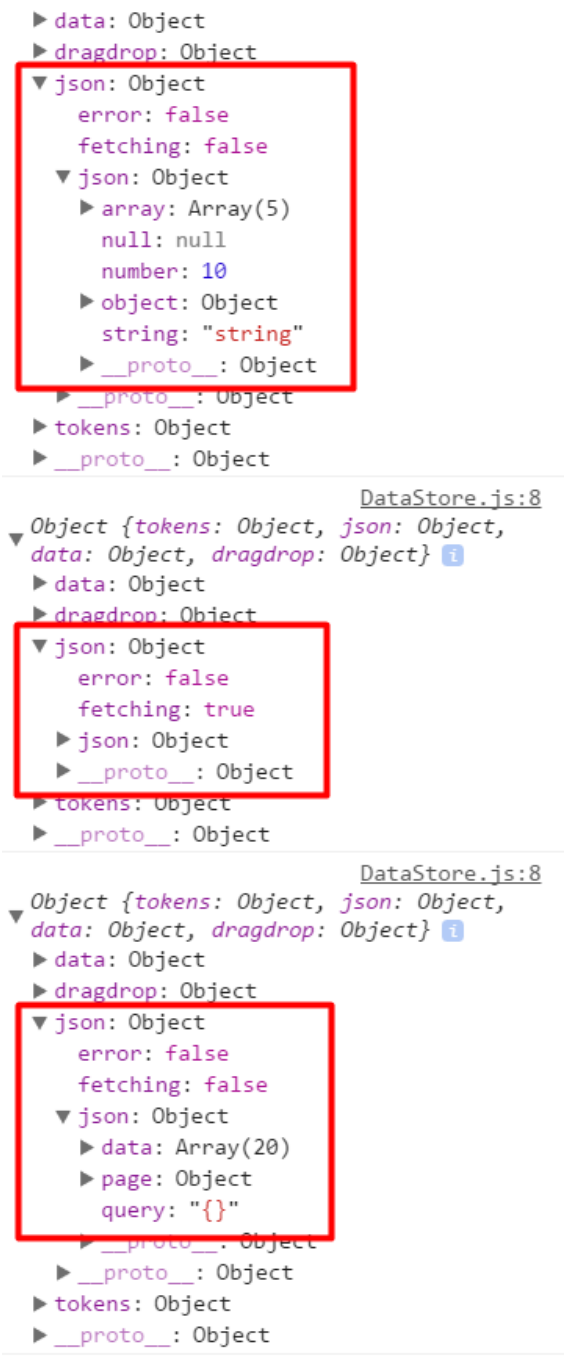


88. 'voor' en 'na' het inladen van gegevens

3.1.1.2 Inladen van externe gegevens

Om aan te geven dat ook externe bestanden kunnen ingeladen worden in de applicatie, zullen in deze paragraaf enkel de veranderingen in de Redux-databank worden getoond. De correcte werking van de JSON-visualisatie werd reeds getoond.

In figuur 89 is de verandering van de waarden in de Redux-databank duidelijk te zien. De voor, tijdens en na van de Redux-databankveranderingen, aangeduid door de rode kaders, worden in deze figuur getoond. Er wordt hier gestart met de eindwaarde uit figuur 88. Bij de volgende stap in het proces is duidelijk te zien dat de 'fetching'-parameter in de databank van een 'false' naar een 'true' is veranderd tussen de twee eerste kaders. Tussen de twee laatste kaders veranderd deze waarde terug naar 'false'. Het laatste kader in de figuur heeft nu een ander object opgeslagen in de databank in het JSON-onderdeel dan was opgeslagen in het eerste kader.



89. Verandering van de Redux-databankgegevens bij het opvragen van een JSON van een externe service

3.1.2 Verslepen van gegevens

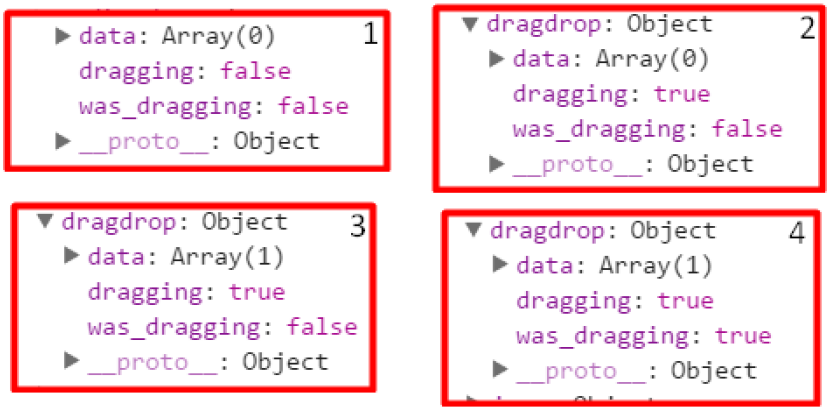
Bij het verslepen van gegevens zijn er twee gevallen te schetsen wat betreft de JSON-visualisatie: wanneer een versleepactie wordt gestart en wanneer een versleepactie geëindigd is.

3.1.2.1 Starten van een versleepactie

In deze paragraaf zullen de resultaten opnieuw worden vertoond aan de hand van de veranderingen van de gegevens in de Redux-databank. Bij het starten van een versleepactie zijn er twee gevallen die beschouwd kunnen worden:

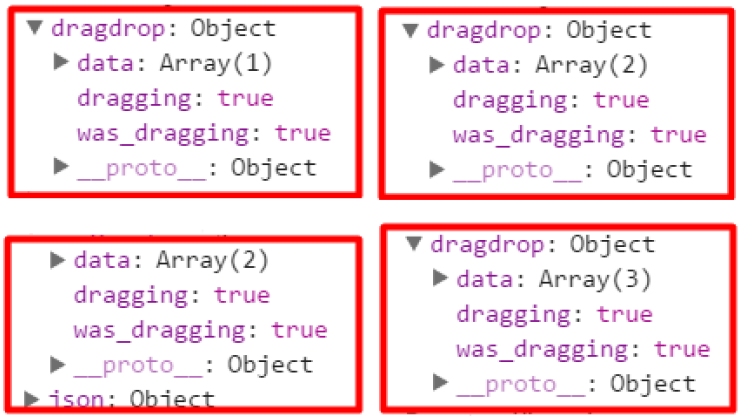
- Wanneer er meerdere gegevens uit de JSON worden versleept;
- Wanneer een enkel gegeven uit de JSON wordt versleept.

Eerst wordt het geval beschouwd waarbij meerdere gegevens worden versleept. Om te illustreren dat de werking verloopt zoals verwacht, wordt uitgegaan van een voorbeeld uit de applicatie waarbij drie gegevens werden versleept. In figuur 90 is te zien dat in de Redux-databank de status van ‘dragging’ en was ‘was_dragging’ veranderen door de tijd. In deze figuur is de volgorde van de verandering in de tijd aangegeven door een cijfer in de rechter bovenhoek van elk kader. In de overgang tussen kader 1 en kader 2 is het duidelijk dat bij het starten van het verslepen van gegevens eerste de ‘dragging’-parameter van ‘false’ naar ‘true’ wordt gezet. Hierbij is het duidelijk voor de andere componenten van de JSON-visualisatie dat ook zij hun gegevens moeten doorgeven als ze door de gebruiker werden aangeduid. Van kader 2 naar kader 3 is te zien dat de eerste component hier naar geluisterd heeft en zijn gegevens heeft doorgegeven. Dit is te zien doordat het onderdeel ‘data’ van een lege lijst naar een lijst met één gegeven erin. In de laatste overgang is te zien dat de ‘was-dragging’-parameter op ‘true’ wordt gezet.



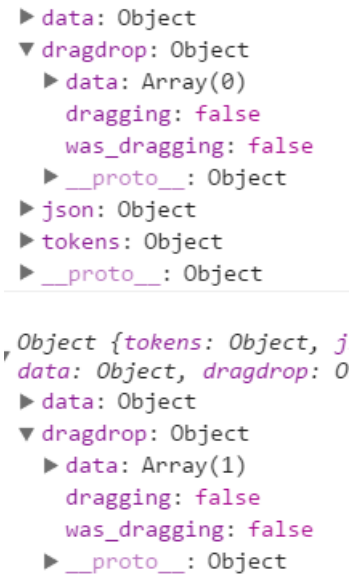
90. Veranderingen in de Redux-databank

Hiermee resteren nog twee gegevens uit de JSON die nog niet in de Redux-databank zijn opgeslagen. Deze zijn niet vergeten, maar worden chronologisch in de Redux-toestand geplaatst. De eerste component die zijn gegevens doorstuurt naar de Redux-databank is ook de eerste die de ‘was_dragging’-parameter zal aanpassen. De andere gegevens komen pas hierna toe. Dit wordt getoond in figuur 91. In de figuur is te zien dat de nakomelingen nog worden toegevoegd aan de databank. Wat hierbij ook opvalt is dat ze de databank doen updaten zonder dat er relevante informatie wordt bijgestoken. Dit komt omdat elke component dezelfde procedure doorloopt om zijn gegevens mee te geven. Dit systeem is voldoende voor de doeleinden van dit project, maar zal verbeterd moeten worden om zo min mogelijk de databank te laten updaten. Elke update zorgt er namelijk ook voor dat andere delen van de applicatie updaten en bijgevolg rekentijd in beslag neemt.



91. Nakomende gegevens

In het andere geval, bij het verslepen van één gegeven, zal dit proces er anders uitzien. Zoals in figuur 92 is aangegeven, zal hier geen gebruik worden gemaakt van de ‘dragging’- en de ‘was_dragging’-parameters in de databank. In plaats daarvan worden de gegevens onmiddellijk opgeslagen.



92. Verslepen van slechts één gegeven

3.1.2.2 Stoppen van een versleepactie

Bij het stoppen van een versleepactie is het nodig om de 'data' buffer te legen, zodat bij de volgende versleepactie geen gegevens aanwezig zijn die kunnen worden samengevoegd met de versleepte gegevens. Dat deze buffer wordt geleegd zoals dit verwacht wordt, is afgebeeld in onderstaande figuur.

93. Legen van de buffer

3.2 Toepasselijke visualisaties

Om na te gaan of in de applicatie de correcte visualisaties worden bepaald voor een gegeven versleept dataset is het niet mogelijk om de resultaten voor elk specificatiemodel te tonen. Daarom worden de resultaten besproken met de bevindingen van een subset van alle specificatiemodellen. De beperkingen en geaccepteerde datatypes van deze modellen zijn weergegeven in figuren 94 en 95. Deze twee modellen zullen nooit gezamenlijk als een

optie verschijnen. De gegevens die worden gebruikt om te testen dat elke visualisatie enkel als keuze worden gegeven als aan alle voorwaarden in het specificatiemodel is voldaan, zijn afgebeeld in figuur 96.

```
"acceptedDataTypes" : [
        "string[]",
        "number[]"
    ],
    "resizeEnabled" : true,
    "constraints" : {
        "groups" : [
            ["string[]", "number[]"],
            ["string[]"]
        ],
        "occurrences" : {
            "range" : {
                "max" : {
                    "string[]" : 1,
                    "number[]" : 4
                }
            }
        },
        "dataLength" : {
            "sameConstraints" : false,
            "sameLength" : [true],
            "range" : {
                "max" : {
                    "string[]" : [15, 10],
                    "number[]" : [15]
                },
                "min" : {
                    "string[]" : [3, 1],
                    "number[]" : [3]
                }
            }
        }
```

94. Deel van het specificatiemodel van het taartdiagramma

```
"acceptedDataTypes" : [
        "number[]"
    ],
    "resizeEnabled" : true,
    "constraints" : {
        "occurrences" : {
            "range" : {
                "max" : {
                    "number[]" : 7
                }
            }
        },
        "dataLength" : {
            "sameConstraints" : false,
            "sameLength" : [true],
            "range" : {
                "max" : {
                    "number[]" : [100000]
                },
                "min" : {
                    "number[]" : [20]
                }
            }
        }
```

95. Deel van het specificatiemodel voor de lijngrafiekcomponent 'functional line'

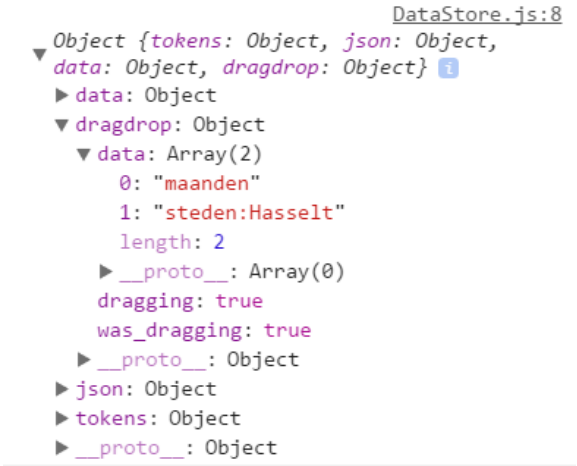
```
{
  "maanden" :
["Januarie","Februarie","Maart","April",Mei",Juni","Juli","Augustus","Oktober","Septem
ber","November","December"],
  "steden" : {
    "Hasselt" : [100, 86, 75, 110, 60, 80, 85, 83, 99, 55, 73, 56],
    "Brussel" : [100, 60, 80, 85, 83, 99, 86, 75, 110, 55, 73, 56],
    "Antwerpen" : [100, 86, 75, 110, 99, 55, 73, 56, 60, 80, 85, 83, 75, 110, 99, 55,
73, 56, 60, 80, 85]
  },
  "postit" : {
    "string{}" : {
      'item' : "Kawasaki",
      "price" : '500',
      "currency" : 'US Dollar'
    },
    "number{}" : {
      'number of legos' : 550,
      'stock' : 10000,
      'grade' : 8
    }
  }
}
```

96. JSON-testdata

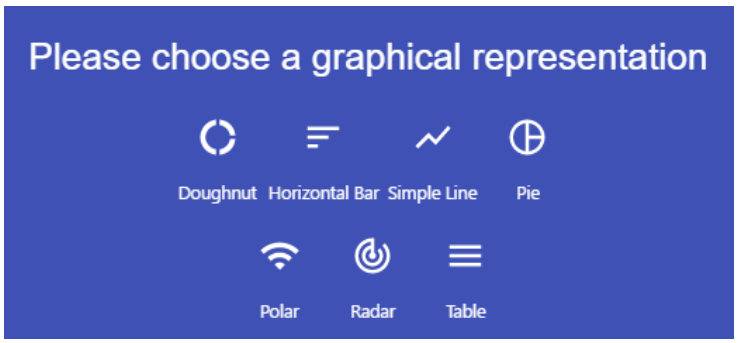
Om aan te tonen dat de correcte visualisaties als optie worden gekozen, worden drie gevallen beschouwd:

- Een geval waar het taartdiagram geldig is, maar niet de lijngrafiek;
- Een geval waar de lijngrafiek geldig is, maar niet het taartdiagram;
- Een geval waar geen van beide geldig is.

Voor het eerste geval worden de parameters ‘maanden’ en ‘Hasselt’ versleept naar het dashboard. Zoals aangetoond in figuur 97 bevinden de sleutelpaden van beide gegevens in de databuffer van de Redux-databank. Dat het taartdiagram in de opties aanwezig is, maar niet de lijngrafiek ‘functional line’ wordt afgebeeld in figuur 98. In deze figuur is ook te zien dat er weldegelijk lijngrafiek als optie bestaat voor deze dataset, maar dat de naam ‘simple line’ is en niet ‘functional line’. Dit stemt overeen met de verwachtte werking.

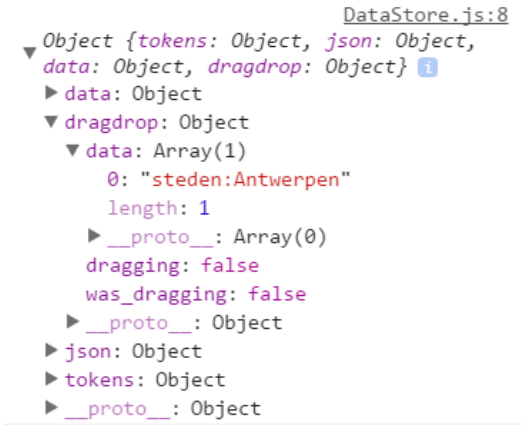


97. Versleepte gegevens

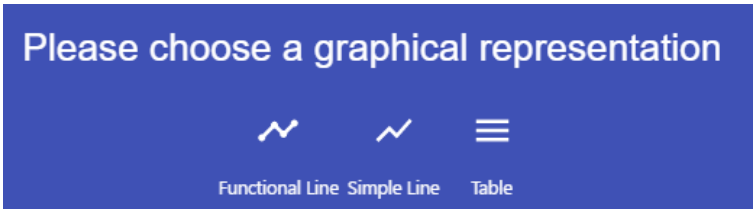


98. Opties voor de data

Voor het tweede geval wordt enkel de parameter ‘Antwerpen’ uit de testdata versleept. Deze lijst van gegevens bevat meer dan twintig getallen. Het taartdiagram zou hier niet geldig mogen zijn omdat deze enkel lijsten van strings of een lijst van strings en een of meer lijsten van getallen. Dat het sleutelpad van de gegevens van ‘Antwerpen’ uit de JSON in de databuffer aanwezig is, wordt getoond in figuur 99. De opties die de data van ‘Antwerpen’ kunnen aanvaarden, zijn weergegeven in figuur 100. Hierbij is te zien dat ‘functional line’ aanwezig is, maar dat het taartdiagram niet langer tot de opties behoort.

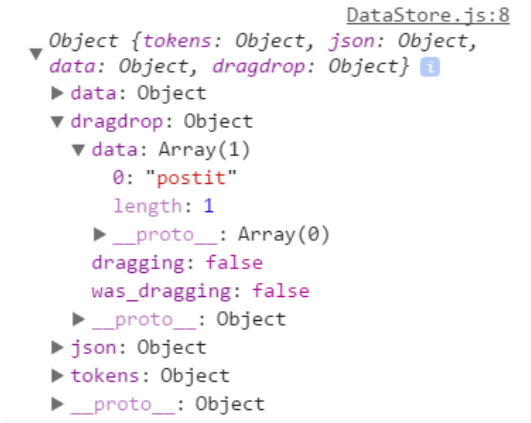


99. Sleutelpad 'Antwerpen' in de databuffer

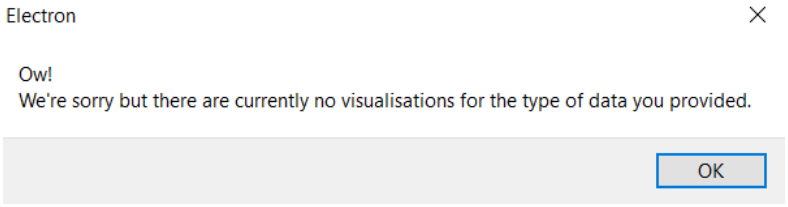


100. Gegenereerde opties voor gegevens in 'Antwerpen'

In de laatste test wordt het onderdeel ‘postit’ uit de testdata in het dashboard versleept. Deze datavorm kan niet worden gebruik door het taartdiagram of de lijngrafiek zoals aangegeven in hun respectievelijke specificatiemodellen. Dat de versleepte gegevens zich in de databuffer bevinden, wordt afgebeeld in figuur X en de gegenereerde opties in figuur 102. Alleen bestaan er nog geen visualisaties in de applicatie die de gegevens van ‘postit’ uit de testdata kan gebruiken om een visualisatie mee te maken.



101. Testdata 'postit' in de databuffer



102. Melding dat er geen visualisatieopties bestaan voor de gegeven data

3.3 Visualisaties

Voor de visualisaties zijn er enkele eigenschappen waarop getest kan worden:

- De visualisaties kunnen aangemaakt worden en de correcte gegevens tonen;
- De visualisaties kunnen een titel, een locatie en een omvang hebben en dit ook correct tonen in het dashboard;
- De visualisaties kunnen updaten als nieuwe informatie wordt ingeladen;
- Visualisaties kunnen ingeladen worden uit een template en direct vertoond worden in het dashboard.

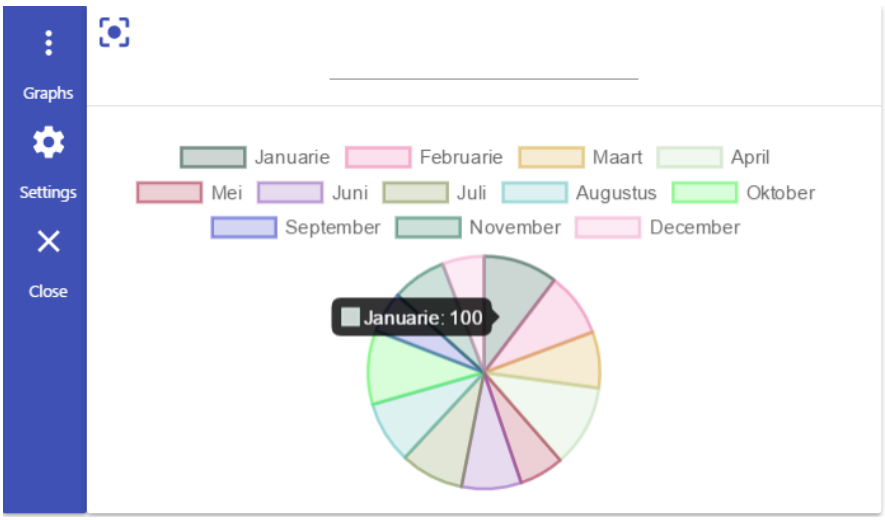
3.3.1 Aanmaken van visualisaties

Voor het aanmaken van visualisaties zijn er twee gevallen te onderscheiden:

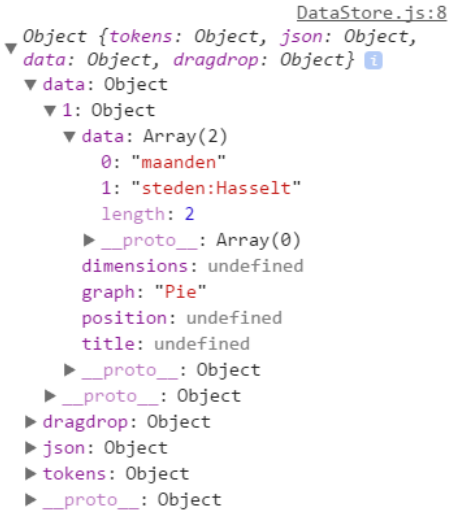
- Wanneer geen gelijkaardige data in de JSON te vinden valt;
- Wanneer wel gelijkaardige data in de JSON te vinden valt.

3.3.1.1 Geen gelijkaardige data

Wanneer geen gelijkaardige data in de JSON te vinden valt, is het proces om een visualisatie aan te maken eenvoudig. Data verslepen en een visualisatie kiezen. Om dit aan te tonen wordt gewerkt met dezelfde gegevens als uit paragraaf 3.2. Hierbij wordt het taartdiagram gecreëerd met de gegevens: ‘Hasselt’ en ‘maanden’ uit de data in figuur 96. Het resultaat hieruit is weergegeven in figuur 103. Uit het diagram in de figuur is moeilijk af te leiden of de gegevens wel correct worden vertoond, maar dankzij de *tooltip*-functionaliteit van de weergave is te zien dat de correcte gegevens vertoond worden voor de overeenstemmende maanden. Dat de gegevens ook in de Redux-databank zijn opgeslagen, is te zien in figuur X.



103. Taartdiagram met de gegevens uit 'Hasselt' en 'maanden'



104. Gegevens van het taartdiagram in de Redux-databank

3.3.1.2 Wel gelijkaardige data

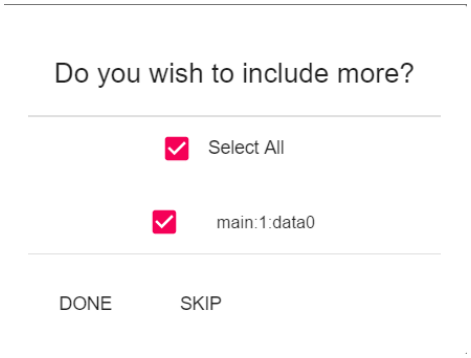
Voor dit gedeelte van de test wordt gewerkt met een andere dataset als JSON. Deze nieuwe JSON wordt getoond in figuur 105. Er zijn twee gevallen waarbij opties worden gegeven:

- Wanneer maar één gegeven versleept werd;
- Wanneer meerdere gegevens werden versleept zonder offset;
- Wanneer meerdere gegevens worden versleept met offset.

Wanneer maar één gegeven wordt versleept uit de JSON, ‘data0’ in het voorbeeld, dan zou de applicatie de optie moeten geven om de andere ‘data0’ ook toe te voegen. In figuur 106 is te zien dat dit ook effectief het geval is. Wanneer er dan voor gekozen wordt om deze toe te voegen, resulteert dit in de visualisatieopties in figuur 107. Hier wordt dan bijvoorbeeld gekozen voor de ‘simple line’ te gebruiken. Dat deze lijngrafiek beide gegevens heeft ontvangen is te zien in figuur 108.

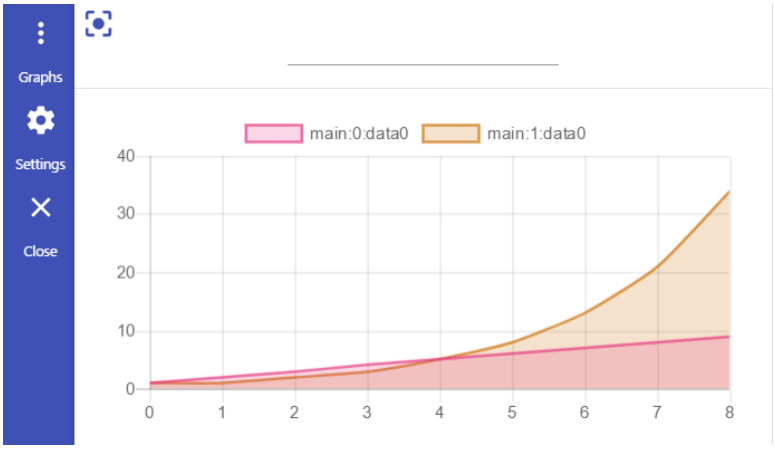
```
{
  "main": [
    {
      "data0": [1,2,3,4,5,6,7,8,9],
      "data1": [1,4,9,16,25,36,49,64,81]
    },
    {
      "data0": [1,1,2,3,5,8,13,21,34],
      "data1": [1,1,1,1,1,1,1,1,10]
    }
  ]
}
```

105. Testdata als JSON



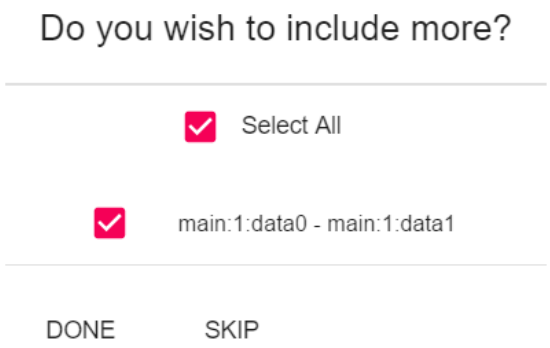
106. Gelijkaardige dataopties

107. Visualisatieopties



108. Beide gegevens in een lijngrafiek

Dit moet ook kunnen werken voor als meerdere gegevens worden versleept zonder een offset. Hiervoor worden zowel ‘data0’ als ‘data1’ uit hetzelfde object versleept. Dit resulteert in de opties voor de gelijkaardige gegevens zoals uitgebeeld in figuur 109. Als ervoor gekozen wordt om de gelijkaardige gegevens bij te voegen, dan resulteert dit in het aanmaken van twee visualisaties. Eén voor elke combinatie. Dit is te zien in figuur 110.



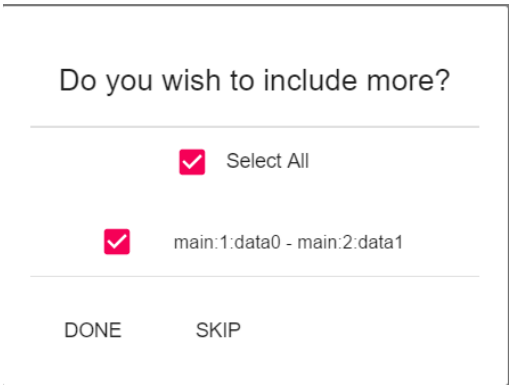
109. Gelijkaardige gegevens voor meer dan één versleept gegeven

110. Aanmaak van meerdere visualisaties

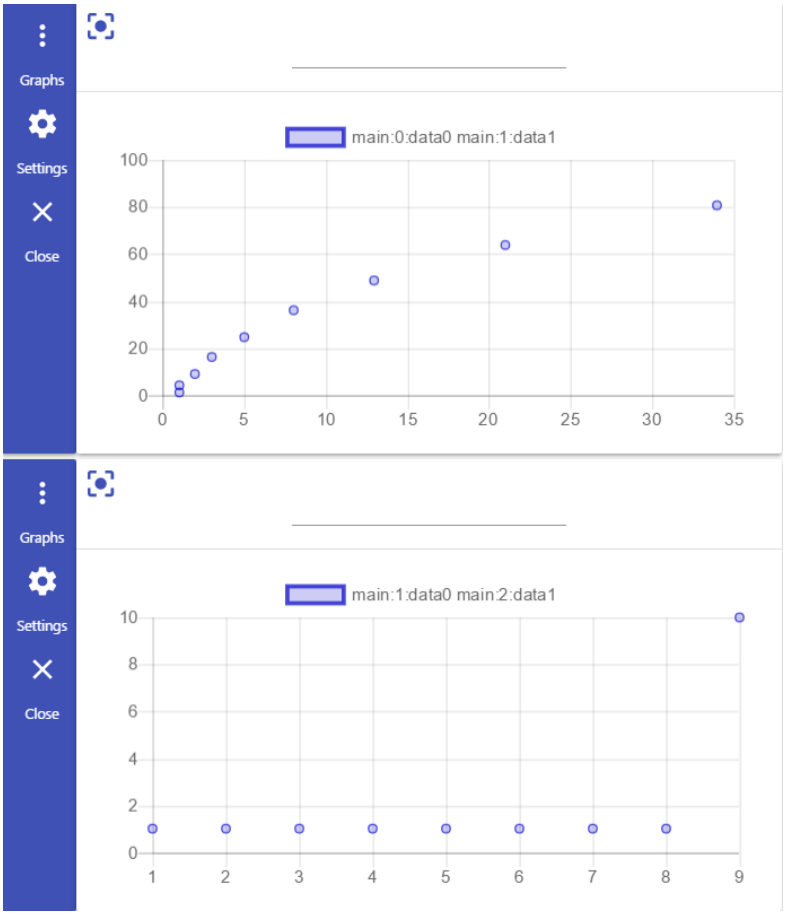
Voor het aanmaken van visualisaties met gelijkaardige bronnen die een offset hebben ten opzichte van elkaar wordt gebruik gemaakt van een lichte variatie toegepast op de JSON die tot nu toe werd gehanteerd. In deze JSON zijn de gegevens van de eerste index in de lijst gedupliceerd en achteraan de lijst toegevoegd. Deze JSON wordt getoond in figuur 111. De gegevens die in dit geval worden versleept zijn ‘data0’ uit de eerste index van de lijst en ‘data1’ uit de tweede index van de lijst. De opties voor het toevoegen van gelijkaardige gegevens en de bekomen visualisaties worden respectievelijk getoond in figuren 112 en 113. De gekozen visualisaties zijn ditmaal scatterplots.

```
{
  "main": [
    {
      "data0" :[1,1,2,3,5,8,13,21,34],
      "data1": [1,1,1,1,1,1,1,1,10]},
    {
      "data0": [1,2,3,4,5,6,7,8,9],
      "data1": [1,4,9,16,25,36,49,64,81]
    },
    {
      "data0": [1,1,2,3,5,8,13,21,34],
      "data1": [1,1,1,1,1,1,1,1,10]
    }
  ]
}
```

111. Aangepaste JSON



112. Opties voor gelijkaardige gegevens

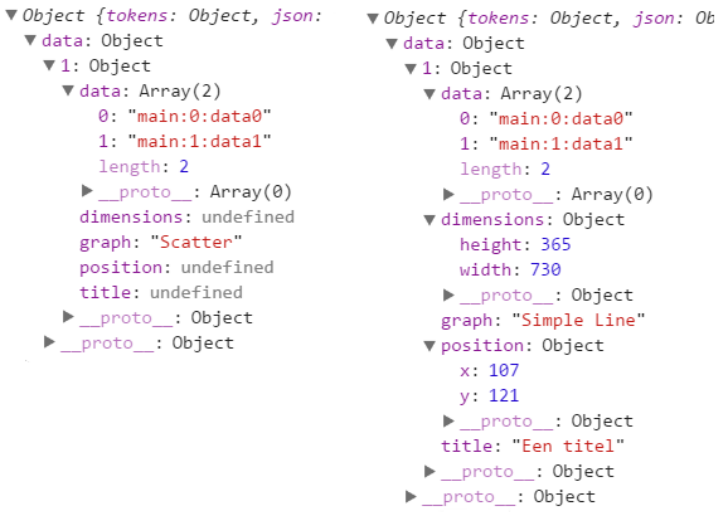


113. Gemaakte visualisaties

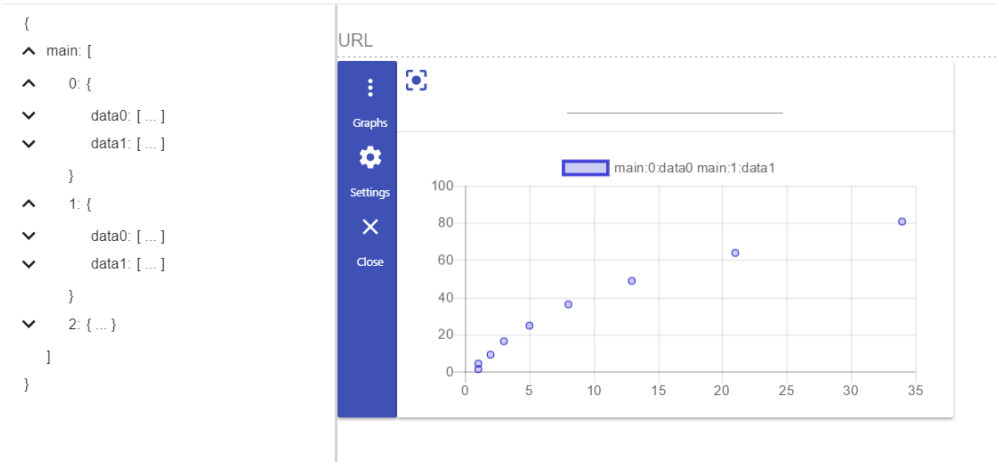
Dit wil daarom niet zeggen dat deze functionaliteit perfect is. Om te beginnen werkt ze nog niet onder alle gevallen waar meerdere gelijkaardige gegevens kunnen teruggevonden worden. Een geval waaronder het bijvoorbeeld nog niet werkt, is wanneer de JSON start als een lijst in plaats van een object. Daarbij komt ook nog eens dat bij het toevoegen van combinaties van gegevens het niet altijd wenselijk is om meerdere visualisaties aan te maken. In kort zal dit onderdeel van de applicatie nog verbeterd moeten worden opdat ze flexibeler te gebruiken is in de applicatie en de gebruiker meer keuzevrijheid geeft om te kiezen hoe of waar de gelijkaardige gegevens worden geplaatst.

3.3.2 Interacties met de visualisaties

Om alle interacties met de visualisatie te kunnen illustreren worden ze niet een per een getoond, maar wel aan de hand van een ‘voor’ en ‘na’ van de visualisatie. Dit wordt vertoond aan de hand van de verandering in de Redux-databank en in de UI van de applicatie. De verandering van de databank wordt getoond in figuur 114. Dat van de UI in figuren 115 en 116. Uit de figuren is op te maken dat de veranderingen van de gegevens door de interactie met de visualisaties correct worden afgehandeld.



114. De initiële toestand van de visualisatie (links) en de aangepaste toestand (rechts)



115. De visualisatie voordat er veranderingen aan werden gebracht

116. De visualisatie nadat er veranderingen werden aangebracht

3.3.3 Updaten van de visualisaties

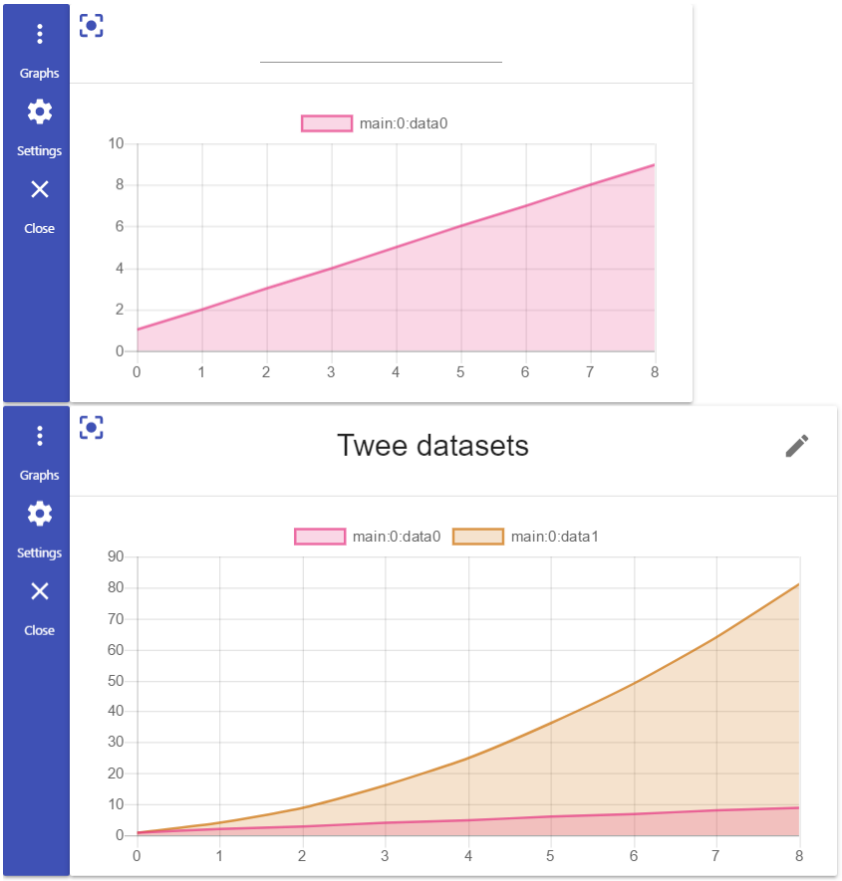
Om het correct updaten van de visualisaties na te kunnen gaan, wordt gebruik gemaakt van twee JSON's. De eerste JSON is die uit figuur 105. De tweede JSON is een variatie hierop, waarbij de gegevens uit beide werden omgedraaid en een gegeven uit verwijderd is. Deze data wordt getoond in figuur 117. Hierdoor kan aangetoond worden dan dat de visualisaties niet alleen hun uiterlijk aanpassen naargelang de data zich in de JSON's bevindt, maar ook dat ze niets zullen tonen indien hun data niet langer aanwezig is.

Er worden twee visualisaties gemaakt. Eén met het sleutelpad voor 'data0' uit de eerste index van de lijst en waaraan verder geen veranderingen zijn aangebracht. De andere visualisatie krijgt de gegevens voor zowel 'data0' als 'data1' uit de eerste index van de lijst. Het 'voor' en 'na' van de visualisaties wordt getoond in figuren 118 en 119.

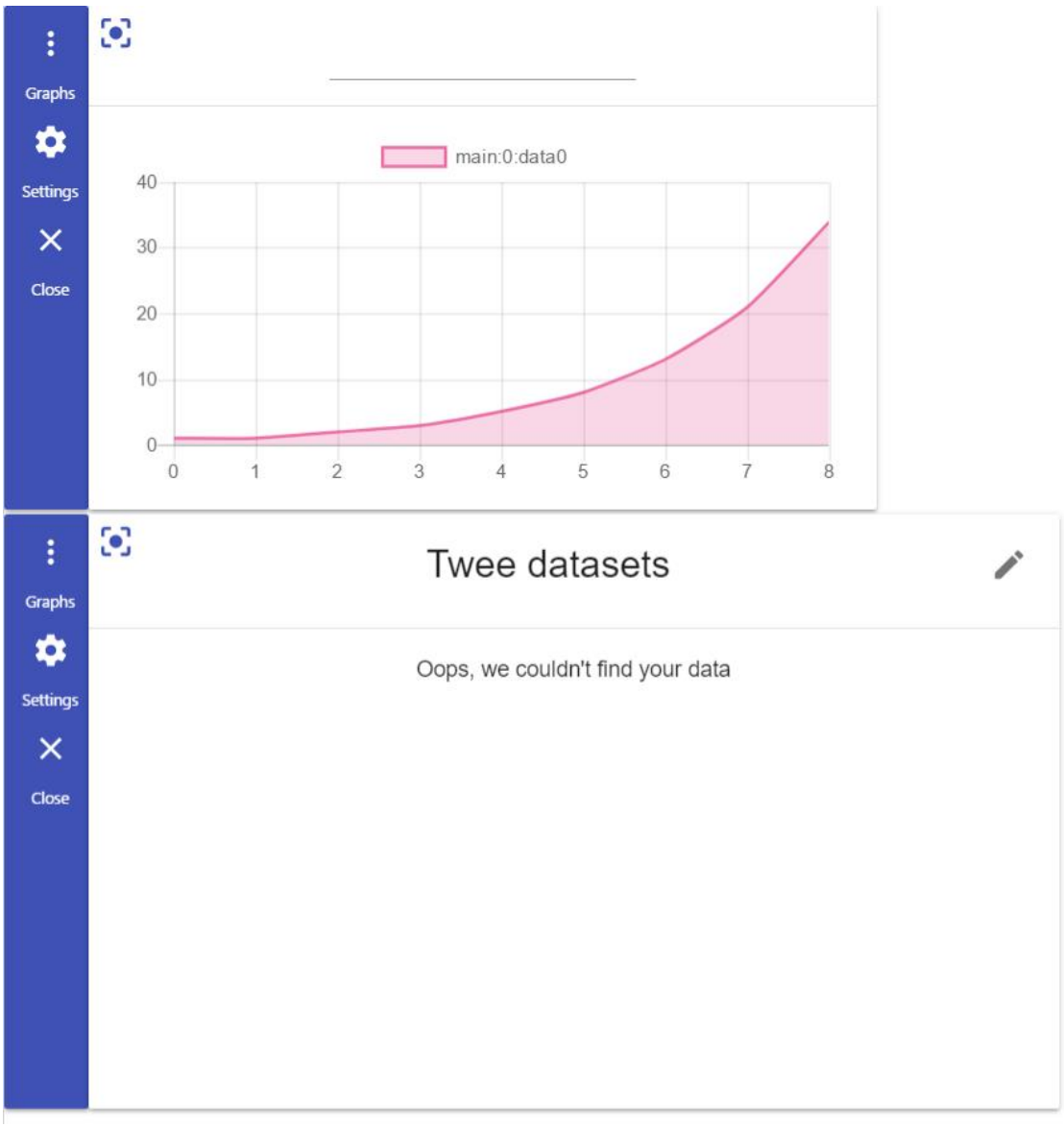
Dat de visualisaties correct hebben gereageerd op de verandering van hun gegevens is duidelijk op te maken in figuur 119. Hier is te zien dat de eerste visualisatie correct de nieuwe gegevens toont. De tweede visualisatie toont zoals verwacht dat de correcte gegevens niet aanwezig zijn in de JSON, maar heeft ook zijn locatie, titel en dimensies behouden.

```
{
  "main": [
    {
      "data0": [1,1,2,3,5,8,13,21,34]
    },
    {
      "data0": [1,2,3,4,5,6,7,8,9],
      "data1": [1,4,9,16,25,36,49,64,81]
    }
  ]
}
```

117. Aangepaste JSON



118. Initiële toestand van de visualisaties

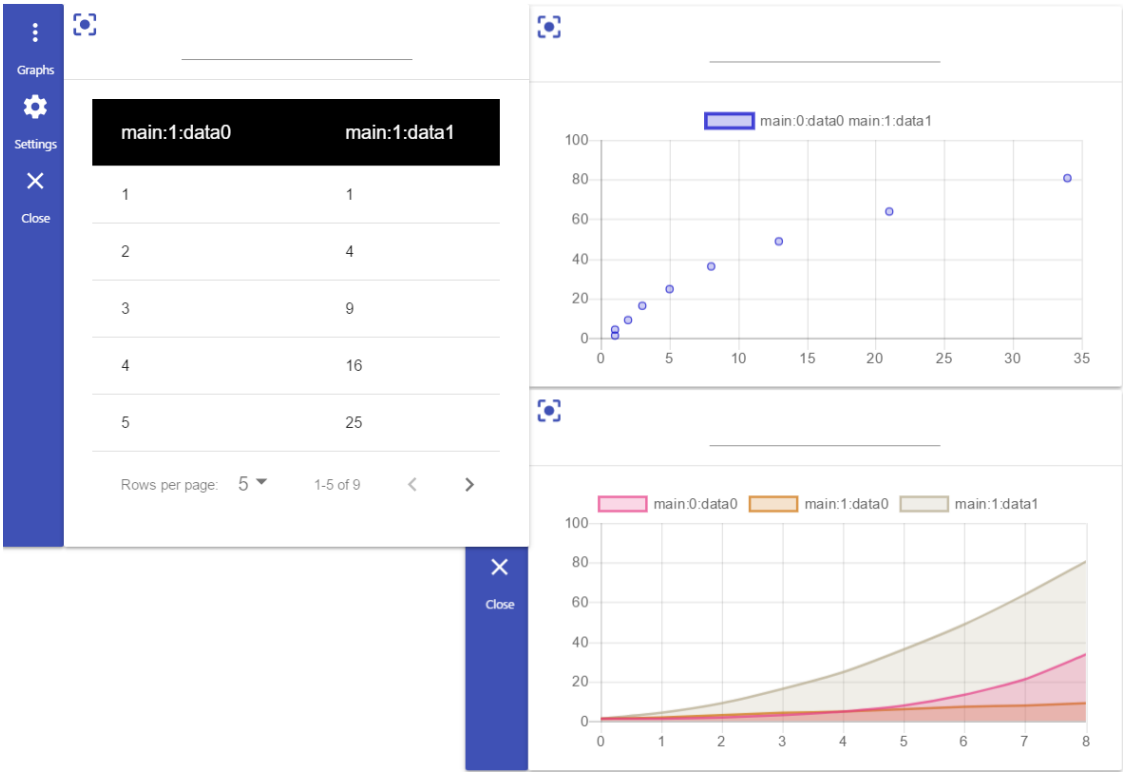


119. Nieuwe toestand van de visualisaties na inladen van de nieuwe JSON

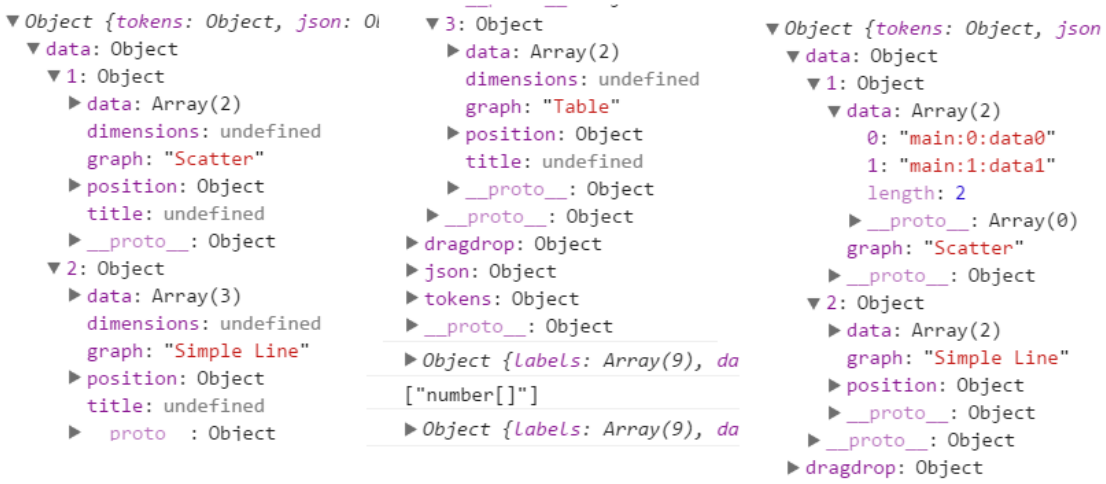
3.3.4 Visualisaties aanmaken uit een template

Om de werking hiervan aan te kunnen tonen, wordt gestart vanuit een bestaande set van visualisaties. Deze visualisaties zijn aangemaakt met de gegevens uit figuur 111. De gemaakte visualisaties worden getoond in figuur 120. Vanuit deze initiële fase, wordt een reeds bestaande template die dezelfde gegevens gebruikt ingeladen in de applicatie. Het bestand, de nieuwe toestand in de Redux-databank en de UI die erdoor werd gemaakt, worden respectievelijk in figuren 122, 121 en 123 getoond. Indien een bestand worden ingeladen dat niet voldoet aan de opmaak van een template wordt een waarschuwing getoond zoals in figuur 124.

Uit de figuren is op te maken dat niet alleen de gegevens in de databank zijn veranderd, maar ook dat de nieuwe visualisaties zijn aangemaakt en de correcte gegevens hebben kunnen halen uit de JSON. Bij het inladen worden de initiële visualisaties verwijderd. Hierbij is het wel belangrijk dat het bestand een effectieve template is. Elke template wordt gecontroleerd op het volgen van de opmaak van een templatebestand. Deze opmaak werd beschreven in paragraaf 2.8.



120. Initiële visualisaties

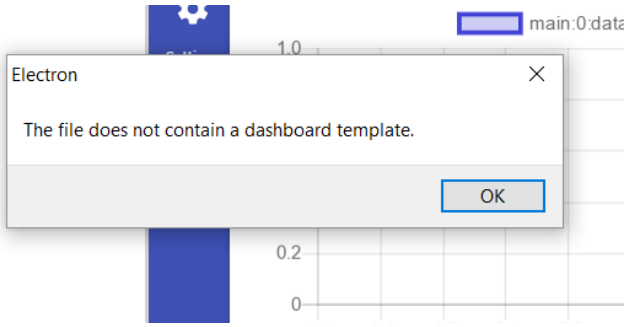


121. Verandering van de Redux-databankgegevens

```
{
  "1": {
    "data": ["main:0:data0", "main:1:data1"],
    "graph": "Scatter"
  },
  "2": {
    "data": ["main:0:data0", "main:1:data1"],
    "graph": "Simple Line",
    "position": {
      "x": 266,
      "y": 323
    }
  }
}
```

122. Templatebestand

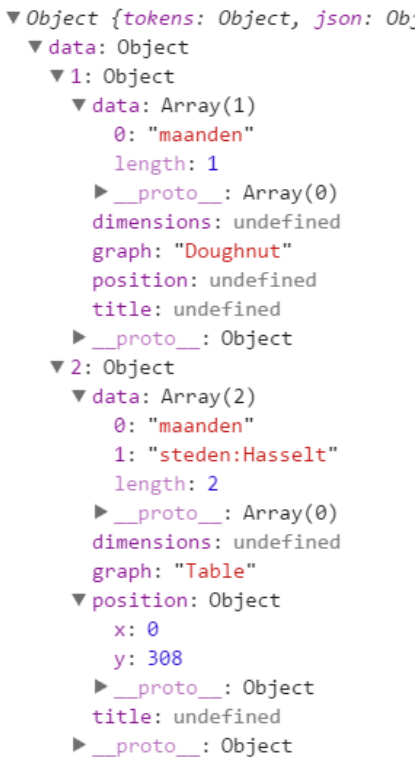
123. Geüpdatete dashboard



124. Waarschuwing

3.4 Bestanden en I/O

Dat bestanden zoals een JSON of een template van visualisaties kunnen ingeladen worden is ondertussen duidelijk. Deze paragraaf zal zich richten op de creatie van templates. Hierom wordt een nieuw dashboard gecreëerd. De waarden in de Redux-database zijn weergegeven in figuur 125. Het opgeslagen bestand wordt dan een naam gegeven, bijvoorbeeld ‘test’. Dat het bestand opgeslagen is op de computer is te zien in figuur 126. De inhoud van dit bestand wordt weergegeven in figuur 127.



125. Waarden in de databank

126. Bestand op de computer

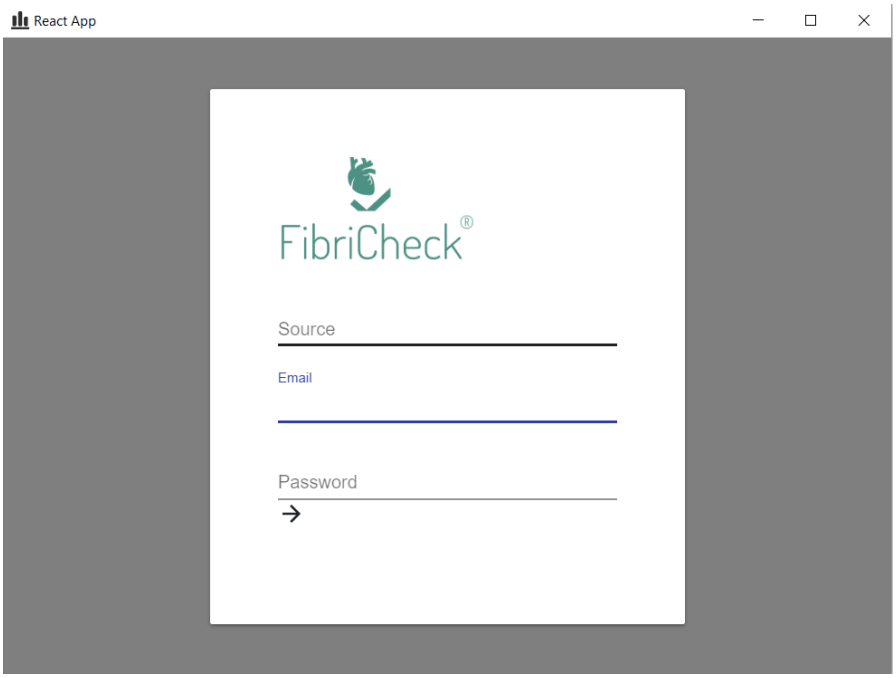
```
{"1":{"data":["maanden"],"graph":"Doughnut"},"2":{"data":["maanden","steden:Hasselt"],
"graph":"Table","position":{"x":0,"y":308}}}
```

127. Gegevens in het bestand

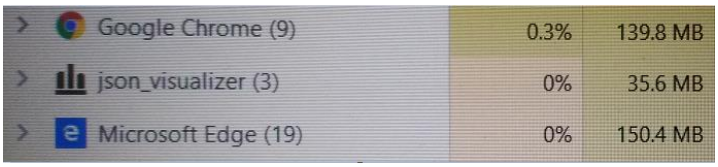
3.5 Executable

Uiteindelijk is het doel van het gebruik van Electron om een programma te maken dat kan worden gebruikt op de computer zonder dat het programma moet worden gehost op een server. Om aan te tonen dat dit onderdeel werkt, worden de gestarte applicatie op de computer en het proces in de *Task Manager* getoond. Deze worden afgebeeld in figuren 128 en 129.

Wel werd vastgesteld dat Acer-computers Electron-applicaties af en toe verhinderen om correct op te starten. Hierbij blijven dan processen draaien op de achtergrond terwijl het programma zelf nooit is opgestart. Het is gebleken dat het EPowerEvent [85], die op Acer-computers draait, de oorzaak is van deze storing.



128. Het gestarte programma



129. Werkend proces in de Task Manager

4 Conclusie en toekomstperspectief

Bij het maken van de masterproef kwamen verschillende uitdagingen kijken. Het gebruik van Electron, React, Redux, JavaScript en TypeScript moesten samen komen om een applicatie te vormen. Hierbij was het de bedoeling dat deze applicatie flexibel genoeg is om eenvoudig visualisaties te kunnen maken in de applicatie en visualisaties toe te voegen in de code van deze applicatie. Daarnaast was het de bedoeling dat deze visualisaties opgeslagen en ingeladen kunnen worden. De JSON-gegevens moesten zowel vanop de computer kunnen ingeladen worden in de applicatie als van services kunnen gehaald worden.

4.1 Conclusie van het programma

Tijdens het maken van de UI werd de focus meer gelegd op het pragmatische. Tijdens het verloop van de ontwikkeling moesten de verschillende webtechnologieën aangeleerd worden, wat er toe heeft geleid dat de opmaak van de applicatie organisch is gegroeid naarmate kennis en kunde toenamen. De UI is volledig functioneel en voldoet aan de vooropgestelde doelstelling hierrond. In de UI zijn alle functionaliteiten zo goed als direct beschikbaar voor de gebruiker. Doordat er nog geen testen door derden op het vlak van gebruikerservaring van de applicatie hebben kunnen plaatsvinden, kan er geen uitspraak gedaan worden omtrent de eenvoud van de applicatie.

Op het vlak van de uitbreidbaarheid van de visualisaties is aangetoond dat met het specificatiemodel eenvoudig nieuwe visualisaties kunnen toegevoegd worden aan de applicatie vanuit één enkele locatie in het programma. Welke visualisaties er kunnen toegevoegd worden is enkel beperkt tot de mate waarin het datatypemodel een gegeven kan beschrijven en door de beperkingen die op dit ogenblik in het specificatiemodel worden gehanteerd.

De resultaten geven ook weer dat alle onderdelen van de applicatie volledig werkzaam zijn, maar dat er toch nog ruimte voor verbetering van bepaalde punten is. Zo zou het voordeliger zijn dat bij verslepen van meerdere gegevens uit een JSON de Redux-databank niet meerdere keren geüpdatet moet worden om alle gegevens te ontvangen. Daarnaast zouden de functies die gelijkaardige gegevens uitzoeken uit een JSON algemener moeten gemaakt worden opdat ze voor elk gegeven waar gelijkaardige gegevens bestaan deze gelijkaardige gegevens ook steeds gevonden worden, ongeacht de JSON. Het niet gelukt binnen de termijn van de masterproef het aanpassen van de opmaak en de gegevens van visualisaties, na het aanmaken ervan, in te werken in de applicatie. Ten laatste zou er ook een oplossing gezocht moeten worden voor het EPowerEvent-probleem.

4.2 Toekomstperspectief

Om te beginnen zouden de elementen waar het programma nog tekort komt moeten aangepakt worden:

- Het overmatig updaten van de Redux-databank bij het verslepen van meerdere gegevens uit een JSON;
- Het vinden van gelijkaardige gegevens moet aangepast worden opdat dit consistent voor elke JSON werkt en voor elke combinatie van gegevens;
- Het toevoegen van een element dat ervoor zorgt dat een gebruiker een visualisatie kan aanpassen of bijwerken;
- Het EPowerEvent-probleem oplossen.

Hiernaast kan de userinterface nog bijgewerkt worden zodat het dashboard meer ruimte krijgt. Hierdoor zouden meer visualisaties per keer kunnen getoond worden in de applicatie dan op dit ogenblik mogelijk is. Uiteindelijk kan het specificatiemodel ook nog worden uitgebreid zodat ook specifieke datastructuren gedefinieerd kunnen worden en gebruikt als datatypes. Op dit ogenblik is het enkel mogelijk om algemene gevallen te gebruiken om te filteren op het datatype.

5 Literatuurlijst

[1] „http://www.json.org/," [Online]. [Geopend 11 2017].

[2] A. Ali, „www.databasejournal.com," databasejournal, 19 01 2016. [Online]. Available: https://www.databasejournal.com/features/mssql/getting-started-with-json-support-in-sql-server-2016-part-1.html. [Geopend 03 03 2018].

[3] „https://www.sitepoint.com/," Site Point, 24 Augustus 2005. [Online]. Available: https://www.sitepoint.com/really-good-introduction-xml/. [Geopend 26 April 2018].

[4] „https://www.freeformatter.com/," Free Formatter, [Online]. Available: https://www.freeformatter.com/json-to-xml-converter.html#ad-output. [Geopend 26 April 2018].

[5] „http://illegalargumentexception.blogspot.be/," illegalargumentexception, 6 Juni 2013. [Online]. Available: http://illegalargumentexception.blogspot.be/2013/06/java-detecting-json-character-encoding.html. [Geopend 25 April 2018].

[6] „http://json-schema.org/," JSON Schema, [Online]. Available: http://json-schema.org/. [Geopend 15 12 2017].

[7] E. B. Craig Stedman, „https://searchbusinessanalytics.techtarget.com/," TechTarget, Augustus 2017. [Online]. Available: https://searchbusinessanalytics.techtarget.com/definition/business-intelligence-BI. [Geopend 18 April 2018].

[8] Tableau, „https://www.tableau.com/," [Online]. Available: https://www.tableau.com/products. [Geopend 20 02 2018].

[9] Microsoft, „https://powerbi.microsoft.com/en-us/," Microsoft, [Online]. Available: https://powerbi.microsoft.com/en-us/. [Geopend 18 04 2018].

[10] Mihandra, Interviewee, *TechSupport Cumul.io.* [Interview]. 17 04 2018.

[11] „https://cumul.io," Cumul.io, [Online]. Available: https://cumul.io/features. [Geopend 20 02 2018].

[12] R. Arif, „https://themehunt.com/," Themehunt, 5 04 2017. [Online]. Available: https://themehunt.com/blog/19-web-tips-and-tricks/112-best-text-editors-for-web-development. [Geopend 18 04 2018].

[13] M. Heller, „https://www.infoworld.com," Infoworld, 17 Mei 2017. [Online]. Available: https://www.infoworld.com/article/3195951/application-development/review-the-10-best-javascript-editors.html. [Geopend 18 April 2018].

[14] B. Nice, „https://medium.com," Medium, 4 September 2017. [Online]. Available: https://medium.com/level-up-web/best-free-code-editors-for-web-developers-2837ccfd1d. [Geopend 18 April 2018].

[15] Microsoft, „https://code.visualstudio.com/," [Online]. Available: https://code.visualstudio.com/docs/. [Geopend 13 12 2017].

[16] „https://www.visualstudio.com/," Microsoft, [Online]. Available: https://www.visualstudio.com/vs/features/. [Geopend 19 April 2018].

[17] „https://docs.microsoft.com/," Microsoft, 30 01 2018. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/productinfo/vs2017-system-requirements-vs. [Geopend 03 03 2018].

[18] „https://code.visualstudio.com/,” Microsoft, 07 02 2018. [Online]. Available: https://code.visualstudio.com/docs/supporting/requirements. [Geopend 03 03 2018].

[19] „https://flight-manual.atom.io/,” GitHub, [Online]. Available: https://flight-manual.atom.io/getting-started/sections/why-atom/. [Geopend 19 April 2018].

[20] A. Sellier, „http://lesscss.org/#,” Alexis Sellier, [Online]. Available: http://lesscss.org/#. [Geopend 19 April 2018].

[21] „https://en.wikipedia.org/,” Wikipedia, 30 Maart 2018. [Online]. Available: https://en.wikipedia.org/wiki/Less_(stylesheet_language). [Geopend 19 April 2018].

[22] „https://en.wikipedia.org/,” Wikipedia, 16 Maart 2018. [Online]. Available: https://en.wikipedia.org/wiki/Atom_(text_editor). [Geopend 19 April 2018].

[23] „https://stackshare.io/,” StackShare, [Online]. Available: https://stackshare.io/stackups/atom-vs-visual-studio-code. [Geopend 19 April 2018].

[24] „https://www.slant.co/,” Slant, [Online]. Available: https://www.slant.co/versus/48/5982/~atom_vs_visual-studio-code. [Geopend 19 April 2018].

[25] „https://wp.nyu.edu/,” [Online]. Available: https://wp.nyu.edu/lucylu/softwares/post_comparison_between_text_editors/. [Geopend 19 April 2018].

[26] „https://stackshare.io/,” Stackshare, [Online]. Available: https://stackshare.io/stackups/atom-vs-visual-studio-code-vs-brackets. [Geopend 19 April 2018].

[27] „https://www.elegantthemes.com/,” Elegant Themes, 22 December 2016. [Online]. Available: https://www.elegantthemes.com/blog/resources/the-sublime-text-code-editor-an-in-depth-review. [Geopend 19 April 2018].

[28] „https://tomthedev.com/,” 02 Augustus 2017. [Online]. Available: https://tomthedev.com/blog-page/atom-vs-sublime-vs-brackets-vs-vs-code. [Geopend 19 April 2018].

[29] „https://www.slant.co/,” Slant, [Online]. Available: https://www.slant.co/versus/40/5982/~sublime-text_vs_visual-studio-code. [Geopend 19 April 2018].

[30] „https://reviews.financesonline.com/,” Finances Online, [Online]. Available: https://reviews.financesonline.com/p/sublime-text/. [Geopend 19 April 2018].

[31] „https://www.slant.com/,” Slant, [Online]. Available: https://www.slant.co/versus/52/5982/~brackets_vs_visual-studio-code. [Geopend 19 April 2018].

[32] „https://en.wikipedia.org/,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Brackets_(text_editor). [Geopend 19 April 2018].

[33] M. Heller, „https://www.infoworld.com/,” Infoworld, 24 Juli 2017. [Online]. Available: https://www.infoworld.com/article/3210589/node-js/what-is-nodejs-javascript-runtime-explained.html. [Geopend 26 April 2018].

[34] P. Patel, „https://medium.freecodecamp.org/,” Medium, 18 April 2018. [Online]. Available: https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5. [Geopend 26 April 2018].

[35] „https://github.com/,” Google, [Online]. Available: https://github.com/v8/v8/wiki. [Geopend 26 April 2018].

[36] GitHub, „https://electronjs.org/,” GitHub, [Online]. Available: https://electronjs.org/docs/. [Geopend 13 12 2017].

[37] Google, „https://www.chromium.org/," Google, [Online]. Available: https://www.chromium.org/Home. [Geopend 13 12 2017].

[38] „https://electronjs.org/," GitHub, [Online]. Available: https://electronjs.org/docs/development/atom-shell-vs-node-webkit. [Geopend 20 April 2018].

[39] „http://tangiblejs.com/," TangibleJS, 26 Maart 2016. [Online]. Available: http://tangiblejs.com/posts/nw-js-and-electron-compared-2016-edition. [Geopend 20 April 2018].

[40] „https://codeburst.io/," CodeBurst, 17 Januarie 2018. [Online]. Available: https://codeburst.io/8-javascript-alternatives-for-web-developers-to-consider-22f8d38bdfa9. [Geopend 23 April 2018].

[41] „https://www.slant.co/," Slant, 23 April 2018. [Online]. Available: https://www.slant.co/topics/101/~best-languages-that-compile-to-javascript. [Geopend 23 April 2018].

[42] J. Kolce, „https://www.sitepoint.com/," Site Point, 21 Augustus 2017. [Online]. Available: https://www.sitepoint.com/10-languages-compile-javascript/. [Geopend 23 April 2018].

[43] „https://en.wikipedia.org/," [Online]. Available: https://en.wikipedia.org/wiki/JavaScript. [Geopend 15 12 2017].

[44] „https://developer.mozilla.org/," Mozilla, [Online]. Available: https://developer.mozilla.org/. [Geopend 04 03 2018].

[45] Wikipedia, „https://en.wikipedia.org/," Wikipedia, 29 11 2017. [Online]. Available: https://en.wikipedia.org/wiki/TypeScript. [Geopend 15 12 2017].

[46] R. Lauer, „https://developer.telerik.com/," Telerik, 22 02 2017. [Online]. Available: https://developer.telerik.com/topics/web-development/what-is-typescript/. [Geopend 15 12 2017].

[47] „http://coffeescript.org/#top," CoffeeScript, [Online]. Available: http://coffeescript.org/#top. [Geopend 23 April 2018].

[48] „http://elm-lang.org/," Elm, [Online]. Available: http://elm-lang.org/. [Geopend 23 April 2018].

[49] A. Bard, „https://adambard.com/," AdamBard, 3 September 2017. [Online]. Available: https://adambard.com/blog/reason-vs-elm-vs-typescript/. [Geopend 23 April 2018].

[50] „https://medium.com/," Medium, 2 Januarie 2017. [Online]. Available: https://medium.com/front-end-hacking/es2015-vs-elm-vs-typescript-a88dbc5d14d9. [Geopend 23 April 2018].

[51] „https://stackshare.io/," StackShare, [Online]. Available: https://stackshare.io/stackups/elm-vs-typescript. [Geopend 23 April 2018].

[52] „https://clojurescript.org/index," ClojureScript, [Online]. Available: https://clojurescript.org/index. [Geopend 23 April 2018].

[53] „https://stackshare.io/," StackShare, [Online]. Available: https://stackshare.io/stackups/clojure-vs-typescript. [Geopend 23 April 2018].

[54] „http://vschart.com/," VSChart, 16 April 2018. [Online]. Available: http://vschart.com/compare/typescript/vs/clojure. [Geopend 23 April 2018].

[55] „https://reactjs.org/," [Online]. Available: https://reactjs.org/. [Geopend 20 02 2018].

[56] I. Bodrov-Krukowski, „https://www.sitepoint.com/," SitePoint, 22 Maart 2018. [Online]. Available: https://www.sitepoint.com/angular-introduction/. [Geopend 24 April 2018].

[57] J. Neuhaus, „https://medium.com/,” Medium, 28 Augustus 2017. [Online]. Available: https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176. [Geopend 24 April 2018].

[58] „https://angular.io/,” Google, [Online]. Available: https://angular.io/. [Geopend 24 April 2018].

[59] A. Sviatoslav, „https://rubygarage.org/,” RubyGarage, [Online]. Available: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end. [Geopend 24 April 2018].

[60] „https://blog.bitsrc.io/,” Bitsrc, 23 November 2017. [Online]. Available: https://blog.bitsrc.io/11-react-component-libraries-you-should-know-178eb1dd6aa4. [Geopend 24 April 2018].

[61] „https://hackernoon.com/,” Hackernoon, 4 April 2018. [Online]. Available: https://hackernoon.com/23-best-react-ui-component-libraries-and-frameworks-250a81b2ac42. [Geopend 24 April 2018].

[62] „https://ourcodeworld.com/,” Our Code World, 10 July 2017. [Online]. Available: https://ourcodeworld.com/articles/read/497/top-10-best-ui-frameworks-for-reactjs. [Geopend 24 April 2018].

[63] „https://material-ui-next.com/,” Material UI, [Online]. Available: https://material-ui-next.com/style/icons/. [Geopend 23 April 2018].

[64] „https://material.io/icons/,” Material, [Online]. Available: https://material.io/icons/. [Geopend 24 April 2018].

[65] „https://react-bootstrap.github.io/,” React Bootstrap, [Online]. Available: https://react-bootstrap.github.io/layout/media/. [Geopend 24 April 2018].

[66] „http://react-toolbox.io/,” React Toolbox, [Online]. Available: http://react-toolbox.io/#/install. [Geopend 24 April 2018].

[67] „http://www.chartjs.org/,” [Online]. Available: http://www.chartjs.org/. [Geopend 21 02 2018].

[68] „https://github.com/,” [Online]. Available: https://github.com/jerairrest/react-chartjs-2. [Geopend 02 Maart 2018].

[69] „https://d3js.org/,” [Online]. Available: https://d3js.org/. [Geopend 20 02 2018].

[70] „http://dygraphs.com/,” DYGraphs, [Online]. Available: http://dygraphs.com/. [Geopend 24 April 2018].

[71] „https://plot.ly,” [Online]. Available: https://plot.ly/products/react/. [Geopend 21 02 2018].

[72] Plotly, „https://plot.ly/,” Plotly, [Online]. Available: https://plot.ly/javascript/. [Geopend 15 12 2017].

[73] HighSoft, „https://www.highcharts.com/,” HighSoft, [Online]. Available: https://www.highcharts.com/products/highcharts/. [Geopend 15 12 2017].

[74] „https://github.com/electron/asar,” [Online]. Available: https://github.com/electron/asar. [Geopend 25 April 2018].

[75] „https://electronjs.org/,” GitHub, [Online]. Available: https://electronjs.org/docs/tutorial/application-packaging. [Geopend 25 April 2018].

[76] „https://github.com/,” [Online]. Available: https://github.com/electron-userland/electron-builder. [Geopend 25 April 2018].

[77] „https://github.com/,” [Online]. Available: https://github.com/electron-userland/electron-packager. [Geopend 25 April 2018].

[78] „https://electronforge.io/," Electron forge, [Online]. Available: https://electronforge.io/. [Geopend 25 April 2018].

[79] „https://redux.js.org/," Redux, [Online]. Available: https://redux.js.org/. [Geopend 25 April 2018].

[80] D. Abramov, „https://medium.com/," Medium, 19 September 2016. [Online]. Available: https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367. [Geopend 25 April 2018].

[81] „https://github.com/gaearon/redux-thunk," [Online]. Available: https://github.com/gaearon/redux-thunk. [Geopend 25 April 2018].

[82] „https://redux.js.org/," Redux, 02 Januarie 2017. [Online]. Available: https://redux.js.org/basics/usage-with-react. [Geopend 25 April 2018].

[83] „https://github.com/bokuweb/re-resizable," [Online]. Available: https://github.com/bokuweb/re-resizable. [Geopend 25 April 2018].

[84] „https://github.com/," [Online]. Available: https://github.com/mzabriskie/react-draggable. [Geopend 25 April 2018].

[85] „https://github.com/electron/," 5 01 2018. [Online]. Available: https://github.com/electron/electron-quick-start/issues/181. [Geopend 1 06 2018].

[86] „http://www.woorden.org/," Woorden.org, [Online]. Available: http://www.woorden.org/woord/Business%20Intelligence. [Geopend 20 April 2018].

[87] „http://www.encyclo.nl/begrip/," Encyclo, [Online]. Available: http://www.encyclo.nl/begrip/dashboard. [Geopend 20 April 2018].

[88] „https://www.thebalancecareers.com/," The Balance Careers, [Online]. Available: https://www.thebalancecareers.com/what-is-github-and-why-should-i-use-it-2071946. [Geopend 20 April 2018].

[89] C. Hoffman, „https://www.howtogeek.com/," How To Geek, 24 Februarie 2017. [Online]. Available: https://www.howtogeek.com/202825/what%E2%80%99s-the-difference-between-chromium-and-chrome/. [Geopend 20 April 2018].

[90] „https://nodejs.org/," Node.js, [Online]. Available: https://nodejs.org/en/. [Geopend 20 April 2018].

[91] „https://stackoverflow.com/," [Online]. Available: https://stackoverflow.com/. [Geopend 10 01 2018].

[92] „https://www.techopedia.com/," Techopedia, [Online]. Available: https://www.techopedia.com/definition/30040/prototype-based-programming. [Geopend 20 April 2018].

[93] „https://www.w3.org/," W3, [Online]. Available: https://www.w3.org/TR/WD-DOM/introduction.html. [Geopend 20 April 2018].

[94] „https://www.techopedia.com/," Techopedia, [Online]. Available: https://www.techopedia.com/definition/22695/type-inference. [Geopend 23 April 2018].

[95] B. Krajka, „http://reactkungfu.com/," 12 October 2015. [Online]. Available: http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/. [Geopend 23 April 2018].

[96] „https://www.w3schools.com/," W3Schools, [Online]. Available: https://www.w3schools.com/angular/angular_databinding.asp. [Geopend 24 April 2018].

[97] „http://buildwithreact.com/," Build with React, [Online]. Available: http://buildwithreact.com/tutorial/jsx. [Geopend 24 April 2018].

[98] „http://www.npmtrends.com/,” Node, [Online]. Available: http://www.npmtrends.com/angular-vs-react-vs-vue. [Geopend 24 April 2018].

[99] „https://help.github.com/,” Github, [Online]. Available: https://help.github.com/articles/about-stars/. [Geopend 24 April 2018].

[100] „https://help.github.com/,” Github, [Online]. Available: https://help.github.com/articles/fork-a-repo/. [Geopend 24 April 2018].

[101] „https://bl.ocks.org/mbostock/4063550,” 16 April 2018. [Online]. Available: https://bl.ocks.org/mbostock/4063550. [Geopend 24 April 2018].

[102] „https://plot.ly/,” Plotly, [Online]. Available: https://plot.ly/javascript/3d-surface-plots/. [Geopend 24 April 2018].

[103] „https://github.com/request/request#forms,” [Online]. Available: https://github.com/request/request#forms. [Geopend 25 April 2018].

[104] M. Aranda, „https://medium.freecodecamp.org/,” Medium, 28 October 2017. [Online]. Available: https://medium.freecodecamp.org/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5. [Geopend 25 April 2018].

[105] „https://www.instantssl.com/ssl-certificate-products/https.html,” Instant SSL, [Online]. Available: https://www.instantssl.com/ssl-certificate-products/https.html. [Geopend 25 April 2018].

[106] „https://www.techopedia.com/,” Techopedia, [Online]. Available: https://www.techopedia.com/definition/5466/runtime-environment-rte. [Geopend 26 April 2018].

[107] „https://www.techopedia.com/,” Techopedia, [Online]. Available: https://www.techopedia.com/definition/27857/thread-operating-systems. [Geopend 26 April 2018].

[108] „https://www.computerhope.com/,” Computer Hope, 26 April 2017. [Online]. Available: https://www.computerhope.com/jargon/p/process.htm. [Geopend 26 April 2018].

[109] „https://www.computerhope.com/,” Computer Hope, 26 April 2017. [Online]. Available: https://www.computerhope.com/jargon/c/callback.htm. [Geopend 26 April 2018].

[110] „https://www.techopedia.com/,” TechoPedia, [Online]. Available: https://www.techopedia.com/definition/24155/engine. [Geopend 26 April 2018].

[111] „https://www.techopedia.com/,” TechoPedia, [Online]. Available: https://www.techopedia.com/definition/32890/software-repository. [Geopend 26 April 2018].

[112] „https://www.techopedia.com/,” Techpedia, [Online]. Available: https://www.techopedia.com/definition/14291/ecmascript. [Geopend 26 April 2018].

[113] „http://form.guide/best-practices/validate-email-address-using-javascript.html,” [Online]. Available: http://form.guide/best-practices/validate-email-address-using-javascript.html. [Geopend 2018 Maart 10].

[114] „https://en.wikipedia.org/,” [Online]. Available: https://en.wikipedia.org/wiki/Password_policy. [Geopend 2018 Maart 10].

[115] M. Rouse, „https://whatis.techtarget.com/,” Whatis, April 2017. [Online]. Available: https://whatis.techtarget.com/definition/single-source-of-truth-SSOT. [Geopend 23 Mei 2018].

[116] „https://www.techopedia.com/,” Techopedia, [Online]. Available: https://www.techopedia.com/definition/450/middleware. [Geopend 24 Mei 2018].

[117] „https://www.techopedia.com/,” [Online]. Available: https://www.techopedia.com/definition/1352/uniform-resource-locator-url. [Geopend 25 Mei 2018].

[118] „https://en.wikipedia.org/,” [Online]. Available: https://en.wikipedia.org/wiki/Wrapper_function. [Geopend 30 05 2018].

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Ontwikkeling van een desktopapplicatie met webtechnologieën voor algemene JSON visualisatie

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2018**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

De Stickere, Niels

Datum: **3/06/2018**