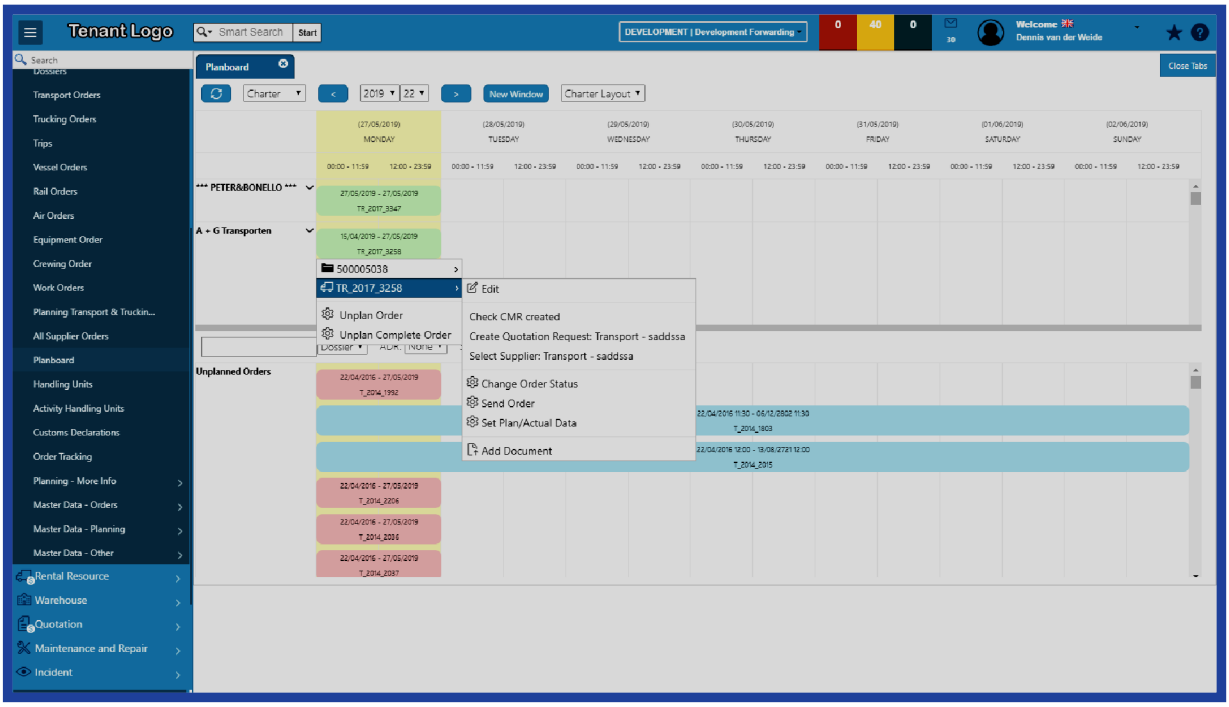
Afstudeerverslag

Implementatie Grafisch Planbord



Auteur	:	Dennis van der Weide
Studentnummer	:	15122999
Studie	:	HBO-ICT: Software Engineering
School	:	Haagse Hogeschool te Zoetermeer
Eerste Examinator	:	Tim Cocx
Tweede Examinator	:	Ahmad Kamal
Opdrachtgever	:	Toon Schilder, Ronald Korporaal
Plaats	:	Sliedrecht
Datum	:	04-02-2019
Bedrijf	:	Adaption

Voorwoord

Dit afstudeerverslag is ter beschikking gesteld door Dennis van der Weide. Sinds 04/02/2019 werk ik aan het afstudeertraject bij het bedrijf Adaption in Sliedrecht.

Ik wil van de mogelijkheid gebruik maken om enkele mensen te bedanken die mij hebben geholpen om zowel het project als het huidige afstudeerverslag tot stand te brengen.

Ten eerste wil ik graag Toon Schilder en Ronald Korporaal bedanken; zij hebben mij de kans gegeven om mijn afstudeertraject te doorlopen bij Adaption. Ik wil graag Verdaasdonk bedanken voor het begeleiden van mijn afstudeertraject. en dat voor zover er problemen waren met de planning dit altijd kon worden besproken.

Daarnaast wil ik natuurlijk alle medewerkers van Adaption bedanken.Echter in het bijzonder wil ik Sander Leget bedanken voor het helpen met het opzetten van een lokale server hetgeen wat het lokaal ontwikkelen mogelijk maakte.

Ook wil ik mijn begeleider Tim Cocx vanuit de Haagse Hogeschool vestiging Zoetermeer bedanken voor het begeleiden en verduidelijken van bepaalde criteria.

Als laatste wil ik de lezer van dit verslag bedanken voor het nemen van de tijd om dit door te nemen. Ik wens u veel leesplezier.

Dennis van der Weide

27-04-2019

Inhoudsopgave

1. Inleiding

Dit afstudeerverslag is ter beschikking gesteld door Dennis van der Weide, in 04/02/2019 ben ik begonnen met het afstudeertraject van de Haagse Hogeschool voor de studie Software Engineering. Dit traject wordt voltooid met dank aan het bedrijf Ada[tion gevestigd in Sliedrecht.

Adaption heeft mij een opdracht gegeven die ik kon gebruiken voor mijn afstudeerdossier. De desbetreffende opdracht houdt in dat er een digitaal planbord wordt ontwikkeld, hier is een prototype van beschikbaar en het is aan mij als student om deze uit te breiden met verschillende functionaliteiten.

Er zijn tijdens het ontwikkelproces verschillende stappen gezet die in dit afstudeerverslag nader worden behandeld. Een van de belangrijkste stappen was het onderzoeken of het huidige prototype nog levensvatbaar is met betrekking tot de technieken die zijn gebruikt. Ik heb dit onderzoek uitgevoerd en mijn bevindingen zijn te lezen bij **6.1 Vooronderzoek** en **6.3 Uitgebreid vooronderzoek**.

Gedurende dit onderzoek zijn de verschillende gewenste functionaliteiten verzameld en waar nodig, zijn deze verduidelijkt. Deze zogenaamde *requirements* zijn later gebruikt tijdens het realiseren van het product en mijn bevindingen zijn te lezen in **5.1 Requirements** later zijn deze nog gewijzigd en deze zijn te vinden in **6.2 Gevolgen van uitgebreid onderzoek**.

Na het verzamelen van de meeste wensen en afronden van het onderzoek, is er gestart met het realiseren van de applicatie. De resultaten van het onderzoek hebben invloed gehad op het ontwikkelproces en de gevolgen hiervan zijn te vinden in **6.2 Gevolgen van uitgebreid onderzoek**. Meer informatie over het realisatieproces is te vinden in **6.4 Applicatie**

Als laatste is het noodzakelijk om de functionaliteiten die zijn gerealiseerd te testen. Voordat deze echter kunnen worden uitgevoerd moet dit correct worden beschreven. Meer informatie over zowel de testcases en manier van testen zijn te vinden in **6.5 Testcases**

2. Verklarende Woordenlijst

Verlader
Een bedrijf die goederen vervoerd. Dit kan de producent van een product zijn maar dit hoeft niet. In het engels wordt dit ook wel een shipper genoemd.

Expediteur
Expediteur regelt naast het verzenden van goederen ook het administratieve gedeelte. Denk hierbij aan: Vergunningen, Prijsaanvragen en onderhandelingen.

Scrum
Flexibele manier van *project/team management* Er wordt in een korte tijd een werkend product gemaakt. Er wordt gewerkt in kleinere teams het liefst in dezelfde ruimte.

Oracle Apex
Apex is afgekort van Application Express. Dit is een ontwikkelomgeving die is gebaseerd om samen te werken met een Oracle Database. Er wordt hierin gebruik gemaakt van HTML, CSS en Javascript.

Selenium
Framework om testen mee te schrijven die het toelaat om deze automatisch uit te voeren. Vaak wordt dit geïntegreerd in de ontwikkelstraat van een bedrijf.

Framework
Een bepaalde standaard om een applicatie te ontwikkelen. Dit is een uitbreiding op een bepaalde programmeertaal waardoor er vaak meer hulp/tools aanwezig zijn.

Websocket
Bidirectioneel communicatie kanaal die wordt opgebouwd tussen twee verschillende onderdelen. In ons geval een gebruiker met de server

User Story
Eenvoudige beschrijving van wat er moet worden gemaakt. Vaak wordt dit beschreven met als uitgangspunt de eindgebruiker. Dit maakt het duidelijk wat er moet worden gemaakt.

Github
Verzamelplaats voor software die wordt uitgebreid. Op GitHub is de code van deze projecten in te zien en gebruikers kunnen ook meerdere versies beheren op de site.

Klassendiagram
Diagram die de structuur van het programma beschrijft. Het is opgedeeld uit objecten en relaties tussen deze objecten. Ook attributen van een object worden beschreven.

Xpath
Querytaal om html elementen te achterhalen. Wordt gebruikt in combinatie met Selenium om bepaalde elementen te vinden.

3. Organisatie

Het bedrijf Adaption is gevestigd in Sliedrecht, Het is in 2010 opgericht door Toon Schilder en Ronald Korporaal. Adaption is gericht op het ontwerpen, ontwikkelen en implementeren van bedrijfssoftware voor de sector Transport & Logistiek. Logistiek wordt door de Nederlandse regering gezien als een topsector.

Het bedrijf is nog groeiende; Medewerkers hebben veel kennis en ervaring in zowel het maken en implementeren van software als de logistieke markt. Functionaliteit en techniek gaan daarbij hand in hand. *("Over Adaption - Adaption", 2019)*

3.1. Werkzaamheden

Adaption heeft een product ontworpen wat zij op de markt hebben gebracht. Hun belangrijkste werkzaamheid is om nieuwe functionaliteiten toe te voegen aan dit product. Het gaat hier over de Logistics Cloud Suite die het mogelijk maakt om logistieke processen te automatiseren of te versimpelen. Hierdoor worden fouten die mogelijk worden gemaakt tijdens het plannen/uitvoeren van een logistiek proces voorkomen.

Adaption heeft zich gericht op een gespecialiseerde markt waar zij hun product onder de aandacht willen brengen. Het gaat hier over de logistieke sector met betrekking tot verladers, expediteurs, vervoeren van gevaarlijke stoffen en tankcontainers. Zij werken nauw samen met hun klanten als het gaat om het implementeren van nieuwe functionaliteiten in hun applicatie. *("Logistics Cloud Suite - Schaalbare Logistieke Cloud Software", 2019)*

3.2. Ontwikkelstraat

Adaption werkt met de software ontwikkelmethode van Scrum, zij hebben ervoor gekozen om iteraties/sprints van vier weken te hanteren. Elke ochtend komen de medewerkers kort bij elkaar om te bespreken welke werkzaamheden zij die dag zullen uitvoeren.

Elke medewerker heeft een lijst van taken die tijdens de huidige sprint moet worden uitgevoerd. Het is van belang voor de medewerkers om duidelijk zijn oplossing op papier te zetten en dit te bespreken met een leidinggevende. Als dit correct wordt bevonden zal de functionaliteit worden geïmplementeerd en hierna zal dit kort door de andere werknemers worden getest. Gedurende dit proces schrijft de verantwoordelijke de testcases die de werking van de desbetreffende functionaliteit waarborgt.

Deze testen worden elke avond automatisch uitgevoerd en fouten die tijdens het testen voorkomen worden geregistreerd en doorgegeven aan de medewerkers van Adaption. Overdag worden deze nogmaals uitgevoerd voor als er veranderingen zijn geweest. Voor elke taak die een medewerker heeft, worden de bovengenoemde acties herhaald.

Er worden in totaal vier sprints uitgevoerd waarbij nieuwe functionaliteiten worden ontwikkeld. Na deze sprints wordt er een week besteed aan het verifiëren van de testen die zijn geschreven en het corrigeren van kleine fouten in de applicatie. Als dit achter de rug is wordt er een nieuwe versie gedistribueerd naar de huidige klanten van Adaption.

3.3. Tools & Talen

Adaption gebruikt binnen het bedrijf verschillende programmeertalen en *tools* om hun ontwikkelproces uit te voeren. De belangrijkste taal die wordt gebruikt is Oracle Apex. Daarnaast wordt er ook gebruik gemaakt van Javascript, SQL en Java.

Elke taal heeft een eigen plek binnen de applicatie. De *frontend* wordt grotendeels opgebouwd door middel van Apex en Javascript. Er wordt soms uitgeweken naar Javascript, omdat bepaalde functionaliteiten niet te verwezenlijken zijn in Apex. *("Oracle Apex", z.d.)*

Voor de database wordt er gebruik gemaakt van Oracle DB en de standaard querytaal die hierin wordt gebruikt is SQl. Adaption gebruikt een *multi tenant* structuur om al hun klanten in een database te verzamelen en ervoor te zorgen dat de desbetreffende klant alleen zijn eigen gegevens te zien krijgt.

Apex kan grotendeels online worden opgebouwd, waar dit niet mogelijk is wordt Netbeans gebruikt om dit te ontwikkelen. Er wordt daarnaast ook gebruik gemaakt van TortoiseSVN als versiebeheer, zodat er geen bestanden verloren kunnen gaan en er een historie bestaat van de bestanden.

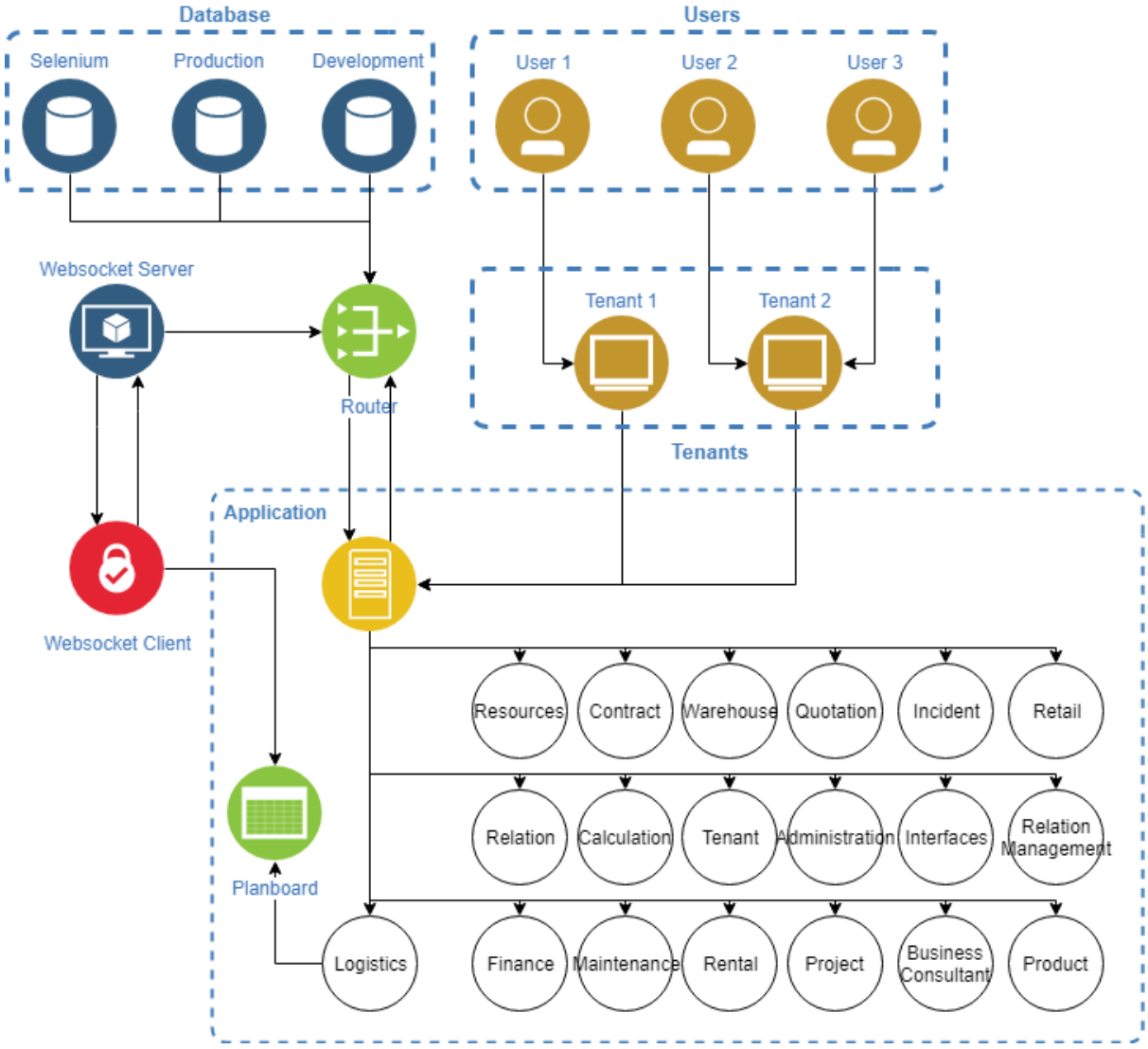
Als laatste wordt er gebruik gemaakt van de testtool Selenium. Dit is een framework die het mogelijk maakt om webapplicaties te testen. De testcases kunnen in verschillende programmeertalen worden geschreven en daarna worden uitgevoerd in een browser. Deze testen worden elke avond uitgevoerd bij Adaption om de kwaliteit van hun product te waarborgen.

3.4. Architectuur

In **figuur 1** wordt een voorbeeld neergezet van de architectuur die de applicatie van Adaption heeft. Ten eerste is te zien dat er drie verschillende *databases* aanwezig zijn voor productie, *development* en testen. Daarnaast horen users bij verschillende *tenants*. Hierdoor kunnen zij de applicatie gebruiken die aan de hand verschillende zaken de correcte data ophaalt.

De *user* in combinatie met *tenant* kan er voor zorgen dat de data verschilt en uit een ander dataset komen. Hierna is het voor de gebruiker mogelijk om de verschillende pagina's in de applicatie te gebruiken. Per *tenant* is het mogelijk dat bepaalde pagina's niet te zien zijn. De volledige selectie is niet in kaart gebracht omdat dit niet relevant is voor het project.

Het planbord valt onder *logistics* en maakt gebruik van een *websocket cliënt* die een connectie maakt met een server. Deze gebruikt dezelfde informatie over de *user* en *tenant* om de correcte data op te halen.



Figuur 1: *Voorbeeld van de architectuur binnen de Oracle Apex applicatie*

4. Opdrachtomschrijving

In dit hoofdstuk zal ten eerste de aanleiding van het project worden verduidelijkt, ten tweede zal de huidige situatie worden beschreven. Als afsluiting zal de gewenste situatie worden geschetst.

4.1. Aanleiding

Adaption biedt een pakket van webmodules aan die ervoor zorgen dat veel administratieve en financiële processen uit handen worden genomen van hun klanten. Er kan een licentie op deze software worden aangeschaft die daarna door middel van een webbrowser te gebruiken is.

Het pakket wordt steeds uitgebreid met verschillende functionaliteiten naar aanleiding van een overleg met de klant. Een dergelijke wens was de mogelijkheid om een grafisch planbord te gebruiken voor het transporteren van orders.

4.2. Huidige Situatie

Enkele jaren geleden is er onderzoek gedaan hoe het planbord het beste kon worden gerealiseerd. Uiteindelijk is hier toen een prototype voor ontwikkeld en beschikbaar gesteld voor de klanten van Adaption.

Het prototype is nu enkele jaren oud en het toevoegen van bepaalde functionaliteiten is nog steeds wenselijk. Vanuit het bedrijf zelf is er ook vraag naar een onderzoek om te garanderen dat het framework wat toen der tijd is gekozen nog steeds correct is.

Er is toentertijd gekozen om het planbord te realiseren in Javascript, omdat Oracle Apex bepaalde functionaliteiten niet ondersteunde, zoals:

- *Drag & Drop;*
- *Cachen van Data;*
- *Maken van grids zoals in het design gewenst is;*
- *Toevoegen van Dynamische filters.*

Het prototype is ontwikkeld door middel van het *framework* MarionetteJS en krijgt de benodigde data binnen via een *websocket* die verbonden is met de server van Adaption.

4.3. Doel

Adaption wil dat het huidige prototype verder wordt ontwikkeld tenzij het onderzoek anders uitwijst. Daarna dienen de gewenste functionaliteiten in kaart te worden gebracht, waarna er een planning kan worden gemaakt.

Er zijn enkele wensen bekend, maar deze moeten nog worden verduidelijkt en geprioriteerd. De wensen zijn als volgt:

- *filteren van orders, zodat een specifieke order sneller kan worden gevonden;*
- *de verschillende zogenaamde resource types moeten worden uitgebreid;*
- *er moeten bepaalde checks worden uitgevoerd als een order wordt ingepland;*
- *het moet mogelijk zijn om verschillende weeknoaties te gebruiken.*

Over bovenstaande wensen worden vragen gesteld die nog niet zijn afgebakend. De wensen zijn op dit moment erg uiteenlopend en dit heeft gevolgen voor de mogelijke *scope* van het project.

4.4. Afbakening

Het project moet worden afgebakend. Er is een bepaalde tijd om de applicatie te realiseren in het kader van mijn afstudeerproject. De afbakening is als volgt.

Er zijn verschillende programmeertalen. Het is noodzakelijk om Javascript te gebruiken of een andere programmeertaal die uiteindelijk Javascript code genereert. Dit komt omdat het planbord moet worden geïntegreerd binnen de huidige applicatie die is gerealiseerd in Oracle Apex. Apex gebruikt zelf HTML, CSS en Javascript. Met andere talen is succes niet te garanderen.

Er moet verbinding worden gemaakt met een websocket om de benodigde data van de server te kunnen lezen en wijzigen. Als er geen verbinding is met de *websocket*, zal de applicatie zijn functie niet kunnen uitvoeren. Omdat de data niet kan worden geupdate.

Alhoewel het een webapplicatie betreft en dit de mogelijkheid geeft om het op een mobiel *device* te openen, is dit niet een doelstelling van het project. Hierdoor kan de gewenste gebruikerservaring niet gegarandeerd worden op een van deze *devices*. Deze keuze is gemaakt, omdat het ontwikkelen van een planbord voor mobiele weergave een grotere wijziging is in het ontwerp. Deze tijd kan beter worden besteed aan de gewenste functionaliteiten.

Het realiseren van de applicatie zal worden uitgevoerd op een laptop die Adaption aan mij in bruikleen heeft gegeven met de benodigde licenties.

5. Plan van aanpak

Hierna volgt welke strategie ik heb gebruikt en waarom en welke activiteiten zijn uitgevoerd om dit project laten slagen.

5.1. Strategie

Voor het realiseren van een applicatie zijn er twee verschillende manieren van softwareontwikkeling. Dit is via een waterval methode of door iteratief te werken. Iedere manier is een correcte manier om software te realiseren.

Adaption maakt zelf gebruik van Scrum. Dit is een iteratieve manier van werken. Er zijn wel enkele aanpassingen nodig. Normaliter wordt er in teams gewerkt met Scrum maar dit is in het kader van dit project niet mogelijk. Vanwege de beperkte tijd voor dit project is een sprint eerder gestopt, zodat alle onderdelen van software development aan bod komen. Dit heeft met name te maken met de wijze waarop Adaption hun sprints indeelt en de wens dat ik meedoe in het proces. *(“Scrum - Wikipedia”, 2019)*

Scrum lijkt mij een goede keuze voor dit project omdat de gewenste functionaliteiten breed geïnterpreteerd kunnen worden. De kans is vrij groot dat tijdens het realiseren van de applicatie nieuwe functionaliteiten aan het licht komen. Door Scrum te gebruiken is het mogelijk om de volgorde van het realiseren te wijzigen. Dit betekent wel dat vanwege de tijdsdruk een andere requirement dan niet op tijd wordt gerealiseerd.

Door Scrum te gebruiken is het mogelijk om met de andere medewerkers mee te draaien en gebruik te maken van de *Daily Standup* om vragen te stellen of snel een afspraak met medewerkers te maken. Ook kan ik de *sprint retrospectives* gebruiken voor feedback.

Er zijn wel enkele activiteiten die volgens de watervalmethode worden uitgevoerd. Het onderzoek is hier een goed voorbeeld van. Ook het verzamelen van de *requirements* en het schrijven van het plan van aanpak zijn voordat er is gerealiseerd voltooid.
(“Watervalmethode - Wikipedia”, 2019)

Aan de hand van de resultaten van het vooronderzoek kan het te gebruiken framework en programmeertaal worden gewijzigd. Dit heeft invloed op de realisatie, omdat het planbord moet worden geschreven in de nieuwe programmeertaal.

5.2. Activiteiten

Om het project succesvol af te ronden zijn er bepaalde activiteiten die moeten worden uitgevoerd. Hierna beschrijf ik de activiteiten en waarom de activiteit is uitgevoerd.

De taken die moeten worden uitgevoerd worden hierin beschreven. Ook wordt er gekeken naar de mogelijke risico's en hoe dit kan worden verminderd. Er is een initiële planning gemaakt, dit is een aanname van hoelang ik denk bezig te zijn met de onderdelen.

Vooronderzoek

Voor dit project is het noodzakelijk om opnieuw te doen naar het probleemdomein. Er moet specifiek worden gekeken worden gekeken naar het geselecteerde *framework*. Het bestaande onderzoek is al wat ouder en er moet worden onderzocht of het *framework* wat toen is gekozen nog steeds de beste kandidaat is.

Requirements

Het was belangrijk dat de *requirements* verzameld en geprioriteerd werden. Hierdoor was het voor mij duidelijk welke taken ik moest gaan uitvoeren tijdens de realisatie. De *requirements* zijn verzameld door middel van gesprekken met de vertegenwoordigers van Adaption. Zij hadden een beter beeld wat de klant wou voor het planbord. In **5.3 Requirements** staan deze uitgewerkt.

Realisatie

Nadat het onderzoek was uitgevoerd was het noodzakelijk om de realisatie te starten. De *requirements* die gedefinieerd waren zijn opgepakt. Per sprint zijn er *requirements* uitgekozen die worden gemaakt.

Testcases

Terwijl de realisatie zijn er verschillende testcases beschreven die de werking van een specifiek onderdeel van de applicatie waarborgen. De testcases zijn opgebouwd door middel van de testtool Selenium. De testcases die ik heb geschreven kunnen later door de medewerkers van Adaption worden gebruikt om nieuw cases te schrijven.

Overdracht

Als laatste is het van belang dat de applicatie wordt toegevoegd aan het pakket van webmodules die Adaption aanbiedt aan zijn klanten. Daarnaast moet zowel de applicatie als documentatie officieel worden overgedragen aan Adaption.

5.3. Requirements

Tijdens het opbouwen van het plan van aanpak zijn de *requirements* verzameld, gedocumenteerd en geprioriteerd. Er is gebruik gemaakt van de MoSCoW methode. *("MoSCoW-methode," 2019)*

Must Have

- Verschillende filters moeten worden toegevoegd;
- Een order moet een andere kleur krijgen als de status veranderd;
- Informatie moet worden getoond als een order wordt geselecteerd;
- Order die is ingepland moet van datum kunnen wijzigen;
- Er moet een verbinding worden gemaakt met de bestaande Apex applicatie.

Should Have

- Ongeplande orders moeten kunnen worden gefilterd;
- Planbord moet in een apart venster kunnen worden geladen;
- Er moeten *post-condition* checks worden uitgevoerd;
- Verschillende notaties voor planning zoals: Dagdeel, Dag, Week en Maand;
- *Progressive Loading / Recycler View* moet worden toegepast.

Could Have

- Systeem moet vragen of een andere *resource* moet worden ingepland; *(Bijvoorbeeld: Chauffeur = Truck / Trailer = Truck / Container + Schip)*
- Elementen van het planbord moeten kunnen worden vergroot en verkleind;
- *Checkbox* zodat de optie om de datum te wijzigen kan worden aangezet;
- Door de gebruiker gedefinieerde suggesties op regio en *resources* toevoegen.

Would Have

- Bepaalde checks moeten worden uitgevoerd voordat een taak wordt gesleept;
- Elementen moeten kunnen worden verplaatst binnen het venster;
- Route die wordt afgelegd laten zien op een kaart;
- Automatisch een andere resource inplannen;
- Klant kan zelf kleuren ingeven die moeten worden getoond.

Won't Have

- Klant heeft de mogelijkheid om zelf filters aan te maken;
- Klant kan zelf instellen wat er tijdens een selectie moet worden getoond bij order;
- Er moet een *gridlayout* beschikbaar zijn voor de klant;
- Informatie moet kunnen worden toegevoegd via een *wizard.*

Dit zijn de *requirements* zoals ze waren verzameld tijdens het opbouwen van het plan van aanpak en onderzoek. De resultaten van het onderzoek zijn gedeeltelijk hierop gebaseerd. In **6.2 Gevolgen van uitgebreid onderzoek** zijn de *requirements* nog gewijzigd voordat de realisatie werd uitgevoerd.

5.4. Gebruikte methoden en Technieken

Tijdens het project zijn er verschillende methoden en technieken die ik heb gebruikt. Per tool zal er worden beschreven waarvoor het was bedoeld en hoe het gebruikt is in het project.

Scrumboard

De ontwikkelmethode Scrum is niet compleet zonder een *scrumboard*. Hier zijn echter verschillende keuzes in. Een *scrumboard* wordt gebruikt om de zogenaamde *requirements* onder te verdelen in behapbare stukken. Deze worden bijgehouden in het *scrumboard*en zo is het voor alle werknemers gemakkelijk om te zien wie waarmee bezig is.

Adaption maat zelf gebruik van Jira. Ikzelf heb dit ook gebruikt, zodat het voor iedereen gemakkelijk is om te zien waar ik mee bezig was. Daarnaast zijn andere systemen die ik heb gebruikt bij Adaption zoals Urenregistratie en Documentatie gekoppeld aan dit *scrumboard*.

Selenium

Het is van belang dat er testen worden geschreven om de werking van het op te leveren product te waarborgen. Een van de eisen van Adaption is dat er testen worden geschreven voor de *tool* die zij gebruiken, zodat deze in de toekomst gedeeltelijk kunnen worden overgenomen als er nieuwe functionaliteiten worden gemaakt. De *tool* heet Selenium. gebruikerstesten kunnen automatisch worden uitgevoerd en kan in verschillende *browsers* worden getest.

IDE's

Een IDE *(Integrated Development Environment)* is software die de programmeur ondersteunt bij het schrijven van een programma. *("IDE - Wikipedia", 2019)*

Bij Adaption wordt er gebruik gemaakt van de tool Netbeans om programmacode te schrijven. Ik heb dit echter alleen gebruikt om de testen te realiseren omdat de Netbeans versie die is geïnstalleerd alleen is ingericht op Java *development*. Ik heb Visual Studio Code gebruikt voor het onderdeel realisatie. Dit is een gratis *tool* die een meervoud van programmeertalen ondersteund. En kan van belang zijn aan de hand van het resultaat van het vooronderzoek.

5.5. Initiële planning

De initiële planning van dit project was als volgt.

Subject / Week		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Eenmalige Activiteiten																		
	Plan van Aanpak																	
	Requirements																	
	Vooronderzoek																	
Iteratief Proces																		
	Ontwerpen																	
	Realiseren																	
	Testen																	
	Demo																	
	Afstudeerverslag																	

Figuur 2: *Illustratie van de initiële planning.*

Bij **5.1. Strategie** was al te lezen hoe het ontwikkelproces zal worden opgelost. De eenmalige activiteiten zullen worden opgelost door middel van de waterval methode.

Er is een gat ontstaan tijdens het verzamelen van de requirements omdat Adaption een nieuwe release van hun software uitbracht. Hierdoor was het voor mij moeilijk om de vertegenwoordigers te spreken omdat zij deze week hun klantenbezoek hadden ingepland.

De laatste twee weken zullen grotendeels worden gebruikt als extra tijd als dit nodig blijkt te zijn. Daarnaast moeten de onderdelen voor mijn verslag al grotendeels zijn afgerond op dit punt zodat er genoeg tijd is voor mij om dit te verwerken in het verslag.

6. Werkzaamheden

De werkzaamheden die zijn uitgevoerd tijdens het project zullen in dit onderdeel naar voren komen. Ieder onderdeel zal worden toegelicht. Mogelijke keuzen worden uiteengezet en de resultaten worden samengevat.

6.1. Vooronderzoek

Vooronderzoek was noodzakelijk vanwege de vraag welk *framework* het beste zou zijn voor een digitaal planbord. Er is enkele jaren geleden al een onderzoek uitgevoerd om deze vraag te beantwoorden. Naar aanleiding van het vorige onderzoek werden er vier koplopers gebruikt: AngularJS, Dojo Toolkit, MarionetteJS en JavascriptMVC.

Als startpunt zijn de koplopers onderzocht ik ben er vanuit gegaan dat de *frameworks* die toen zijn gekozen geschikt waren. De tijd heeft niet stilgestaan en mogelijke ontwikkelingen zal ik kort beschrijven. Zowel DojoToolkit als AngularJS zijn doorontwikkeld en hieruit zijn twee nieuwe *frameworks* ontstaan, genaamd Dojo.IO en Angular. Beide *frameworks* zijn de programmeertaal Typescript gaan gebruiken in plaats van Javascript. Vanwege deze ontwikkeling zullen zowel de oude als nieuwe versies van DojoToolkit en Angular worden vergeleken. *(“Dojo Toolkit”, z.d.) (“AngularJS — MVW Framework”, z.d.)*

Daarnaast is de naam van JavascriptMVC gewijzigd in DoneJS, Hierna zal ik verwijzen naar DoneJS. Door zowel Dojo.IO en Angular mee te nemen in de vergelijking zijn er zes verschillende *frameworks* vergeleken op verschillende criteria en meegenomen in het experiment. *(“donejs - About”, z.d.)*

Er is getest op vijf verschillende onderdelen: Functionaliteiten, Snelheid, Bestandsgrootte, *Plugins* en *Community.* Deze vijf verschillende onderdelen geven een redelijk compleet beeld van de voor- en nadelen van de verschillende *frameworks.* Door slechts deze vijf te selecteren scoort een klein *framework* beter op snelheid en een groot *framework* beter op functionaliteiten.

Vanwege de beperkte tijd voor het vooronderzoek ben ik voorzichtig geweest om een extra *framework* toe te voegen. Ik heb in overleg met mijn bedrijfsbegeleider besloten om een van de populaire *frameworks* React toe te voegen. React stond tijdens het vorige onderzoek nog in de kinderschoenen, maar heeft sindsdien zes jaar de tijd gehad om door te ontwikkelen. Er zijn in totaal dus zeven *frameworks* die zijn onderzocht. *(“React - Wikipedia”, 2019)*

6.1.1. Functies

Voor de eerste vergelijking is er besloten om te kijken naar de verschillende functionaliteiten die de *frameworks* bezitten. Daaruit zijn vijf verschillende onderdelen gekomen die kort zullen worden toegelicht.

De onderdelen die hierboven zijn genoemd kunnen erg bruikbaar zijn tijdens het ontwikkelen van een planbord. Daarom is ervoor gekozen om een puntentelling te maken als een framework deze mogelijkheid bezit. De onderdelen die zijn benoemd zijn niet noodzakelijk om een planbord te ontwikkelen, maar kunnen een goede oplossing bieden.

Componenten
Door middel van componenten kan het planboard modulair worden opgebouwd. Dit zorgt ervoor dat bepaalde stukken code gemakkelijk kunnen worden hergebruikt of uitgebreid.

Observables
Als er een *request* binnenkomt worden de onderdelen die zich hier op hebben aangemeld geupdate.

Testing
Adaption prijst zichzelf vanwege het feit dat zij veel testen uitvoeren, voordat een module wordt opgeleverd aan de klanten. Als een *framework* dit bezit is er rekening gehouden met het testen binnen het *framework*.

2 Way Binding
Een bidirectionele communicatie die de data synchroon houdt voor als een gebruiker wijzigingen uitvoert op het planbord dit wordt gecommuniceerd naar andere gebruikers.

Routing
De applicatie wordt geïntegreerd in de huidige Apex omgeving. Routing kan het mogelijk gemakkelijker maken om een referentie naar het planbord toe te voegen.

Componenten zijn naar mijn mening het belangrijkste omdat dit tijdens het ontwikkelen en onderhouden van de applicatie veel problemen kan voorkomen. Er is geen onderscheid gemaakt met betrekking tot de puntentelling alle onderdelen zijn evenveel waard.

Mocht een framework de mogelijkheid van componenten echter niet bevatten, moet er worden overwogen of dit framework wel geschikt is voor het doel. De kans is vrij klein omdat de meeste frameworks deze mogelijkheid wel bezitten.

De meeste *frameworks* kunnen deze functionaliteiten leveren met behulp van een plug-in, maar er is voor gekozen om dit niet mee te nemen in deze vergelijking.

Dit neemt te veel tijd in beslag en , het is mogelijk dat een plugin die voorheen correct werkte nu niet meer wordt onderhouden. Dit zou kunnen betekenen dat er over een jaar of twee een nieuwe plugin moet worden gezocht die dezelfde functionaliteiten ondersteund.

Frameworks	Components	2W Binding	Observables	Routing	Testing	Totaal
AngularJS	✓	✓	X	✓	✓	4
Angular	✓	✓	✓	✓	✓	5
Dojo Toolkit	✓	X	✓	✓	✓	4
Dojo.IO	✓	X	✓	✓	✓	4
DoneJS	✓	✓	X	✓	✓	4
MarionetteJS	✓	X	X	✓	X	2
React	✓	X	X	✓	✓	3

Figuur 3: *Functionaliteiten van de verschillende frameworks.*

Van alle *frameworks* scoort Angular hier het beste. Het bezit alle onderdelen die van toepassing zijn voor het project. Angular is een zwaar *framework* om te gebruiken en het is dus ook logisch dat het de meeste functies bevat,

MarionetteJS en React beiden de minste functionaliteiten standaard aan. De *frameworks* zouden lichter moeten zijn dan de overige, ook werken ze vaker met plugins dan de overige *frameworks*.

Vervolgens krijgen de frameworks punten toegekend voor hun ranking waarbij het *framework* met de meeste onderdelen zes punten waard is de *frameworks* met daarna meeste onderdelen, vijf punten, ect. Als zij op het aantal onderdelen hetzelfde scoren krijgen ze dezelfde ranking.

RANKING

Angular	**6**
AngularJS	**5**
DojoToolkit	**5**
Dojo.IO	**5**
DoneJS	**5**
React	**4**
MarionetteJS	**3**

6.1.2. Snelheid

Hieronder wordt snelheid toegelicht. Dit betekend de tijd die het *framework* in beslag neemt om elementen te genereren. Het planbord moet namelijk veel orders *(elementen)* maken aan de hand van de geselecteerde waardes die de gebruiker invoert.

Omdat sommige *frameworks* zwaarder zijn om in te laden, is er gekozen om drie verschillende scenario's te testen. Er is gekozen voor de waardes van 100, 1000 en 10.000. In de praktijk zal het planbord niet snel meer dan 1000 elementen genereren, maar voor de volledigheid is 10.000 ook meegenomen.

Er is een experiment uitgevoerd waarbij een stuk code is gebruikt die elementen genereert.

```
for(let j = 0; j < (tempRange + 1); j++){
   let number = Math.floor(Math.random() * 100);
   return (<div>{number}</div>);
}
```

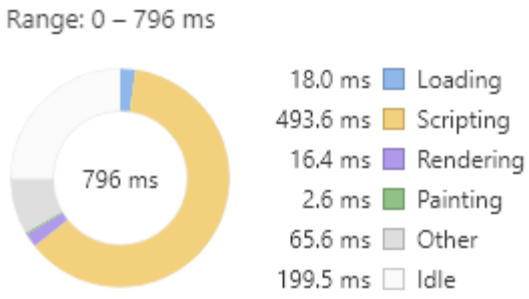
Figuur 4: *Code snippet om elementen te genereren in React*

Deze code zal ervoor zorgen dat er een aantal elementen worden aangemaakt en gekoppeld aan de webpagina. Deze elementen worden gegenereerd tijdens het laden van de pagina. Tijdens het genereren zijn de *frameworks* geïnspecteerd door middel van een *Tool* in Google Chrome. Google Chrome wordt gebruikt, omdat dit een standaard browser is waarin Adaption *development* uitvoert en Adaption raadt hun klant ook aan om deze browser te gebruiken.

Figuur 5 geeft een voorbeeld van de wijze waarop de data wordt getoond met alle onderdelen.

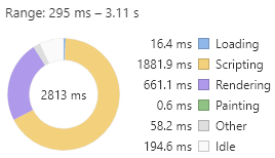
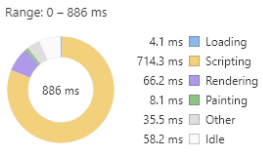
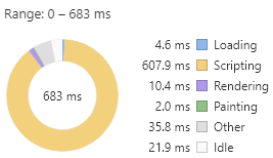
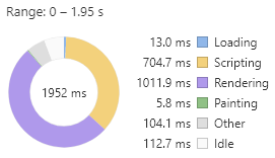
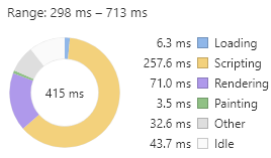
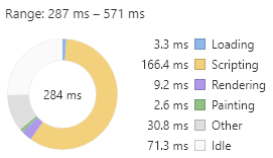
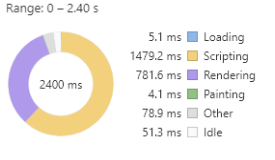
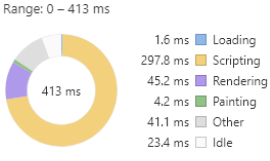
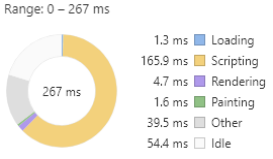
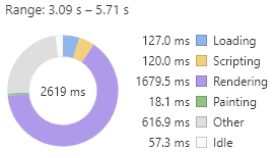
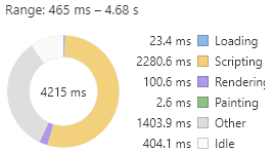
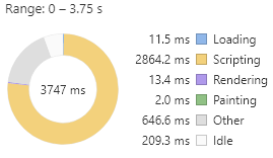
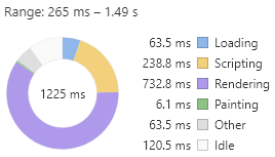
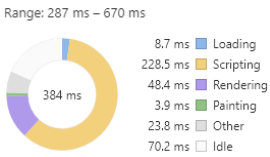
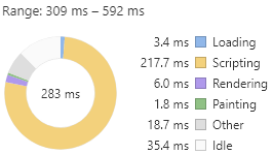
In het cirkeldiagram staan verschillende onderdelen die moeten worden uitgevoerd om de pagina te laden.

Bovenaan het figuur is een *range* voor de tijdlijn. In de meeste gevallen zal de range starten bij 0, tenzij er op dat moment geen acties zijn waargenomen bij de opname.



Figuur 5: *Resultaat van www.google.com*

Tijdens het experiment zijn er problemen aan het licht gekomen die ik hier kort zal verduidelijken samen met de impact die zij hebben op het experiment. Voor zowel Dojo Toolkit als Dojo.IO is het niet gelukt om resultaten te genereren. Bij Dojo.IO is het niet gelukt om de voor het experiment benodigde code te implementeren. Het is niet gelukt om dit binnen de beperkte tijd werkend te krijgen. Voor DojoToolkit had ik code geschreven, maar het werkte niet. Zelfs na het raadplegen van de documentatie en code van een bestaand project te hebben geprobeerd, werkte het niet. *("Hello Dojo," n.d.)*

	10.000 Records	1.000 Records	100 Records
Angular			
React			
AngularJS			
DoneJS			
MarionetteJS			

Figuur 6: *Resultaten per framework*

De resultaten van DoneJS zijn niet in lijn met de rest van de *frameworks.* Terwijl de rest grotendeels dezelfde groei heeft. is het bij DoneJS totaal het tegenovergestelde. Vanwege deze opmerkelijke resultaten is het experiment voor dit *framework* nogmaals uitgevoerd, maar dit gaf bijna geen verschil. Ik concludeer dat DoneJS geoptimaliseerd is voor grotere applicaties die veel data bevatten.

Per *framework* is er een score gegeven beginnend met vijf t/m nul *(voor de laagste score.)*

Per onderdeel is er een winnaar en alle *frameworks* tellen in gelijke mate mee voor het resultaat.

Alle punten van de verschillende *frameworks* zijn bij elkaar opgeteld en wordt nogmaals het aantal punten gegeven voor de eindscore.

RANKING

	10.000	1.000	100	Totaal	
MarionetteJS	5	5	4	14	5
AngularJS	3	4	5	12	4
React	4	3	3	10	3
Angular	2	2	2	6	2
DoneJS	1	1	1	3	1
Dojo.IO	0	0	0	0	0
DojoToolkit	0	0	0	0	0

6.1.3. Bestandsgrootte

Tijdens de installatie van de verschillende *frameworks*, is genoteerd hoe groot de desbetreffende *frameworks* waren met alle benodigde onderdelen in een *development* omgeving. Deze omgeving is belangrijk om te noteren, omdat er veel *tools* zijn die de bestandsgrootte kunnen vergroten.

Verder is er gekeken naar de verschillende grottes nadat het in een productieomgeving wordt geïnstalleerd. Veel onderdelen die nodig zijn om een applicatie te ontwikkelen worden niet meegenomen en, dit resulteert vaak in een kleiner formaat. Ook is er gekeken naar het bundel formaat van de *package* zelf. Dit is de grootte nadat er verschillende *tools* zijn gebruikt om de omvang te verkleinen.

FORMAAT

	DEV	PRO	BUNDLE
Dojo.IO:	488 mb	7,93 mb	X
Angular:	267 mb	20,8 mb	76,4 kb
React:	210 mb	422 kb	2.72 kb
DoneJS:	167 mb	375 kb	16,2 kb
MarionetteJS:	52,2 mb	842 kb	10,3 kb
AngularJS:	19,6 mb	2,08 mb	63,4 kb
DojoToolkit:	0 mb	X	X

Dojo Toolkit gebruikt *0 mbit,* omdat het via een *<script>* tag is geïnstalleerd. Voor een productieomgeving zal er wel een bestandsgrootte moeten zijn. Deze informatie was niet te vinden voor DojoToolkit. Voor Dojo.IO was het niet mogelijk om de informatie over de bundel te vinden, maar er kan vanuit worden gegaan dat dit kleiner is dan een productieomgeving.

RANKING

	DEV	DEV /2	PRO	BUNDLE	TOTAAL	
Dojo.IO:	0	0	2	1	3	1
Angular:	1	0.5	1	2	3.5	2
React:	2	1	5	6	12	6
DoneJS:	3	1.5	6	4	11.5	5
MarionetteJS:	4	2	4	5	11	4
AngularJS:	5	2.5	3	3	8.5	3
DojoToolkit:	6	3	0	0	3	1

De puntentelling is grotendeels hetzelfde als bij het vorige onderdeel. Elke omgeving heeft een winnaar en het totaal wordt bij elkaar opgeteld en daar komt ook weer een score uit.

Een uniek aspect bij dit onderdeel is dat er bij *development* is besloten om de scores te halveren. De grootte van het framework is in de *development* omgeving namelijk erg verschillend aan de hand van de *plugins* die zijn geïnstalleerd en in *development* is de grootte minder belangrijk in tegenstelling tot een productieomgeving. In een productieomgeving is er vaak een bepaalde grootte beschikbaar voor de applicatie.

6.1.4. Plugins

Er zijn verschillende functionaliteiten die nodig zijn om een werkend planbord te kunnen realiseren. Veel van deze functionaliteiten zijn vrij standaard in de meeste programmeertalen. In dit onderdeel wordt gekeken naar bepaalde oplossingen die het planbord meer mogelijkheden biedt en de snelheid zal verbeteren. Per framework wordt onderzocht of er een mogelijkheid bestaat om een oplossing toe te passen. Ook zal elk onderdeel toegelicht worden en de meerwaarde ervan verduidelijken.

	Drag & Drop		Websockets		Virtual Scrolling	
		Notitie		**Notitie**		**Notitie**
Angular	✓	Angular Material	✓	Socket.IO	✓	Angular Material
AngularJS	✓	jQueryUI	✓	Socket.IO	✓	AngularJS Material
React	✓	jQueryUI	✓	Socket.IO	✓	jQuery
DoneJS	✓	jQueryUI	✓	Native functionaliteit	✓	jQuery
MarionetteJS	✓	jQueryUI	✓	Socket.IO	✓	jQuery
DojoToolkit	✓	jQueryUI	✓	dojox/socket	✓	jQuery
Dojo.IO	✓	Widget #Drag	✓	Socket.IO	✓	Native functionaliteit

Figuur 7: *Mogelijkheden per framework*

Drag&Drop

Drag & Drop is een gewenste functionaliteit van het planbord. Dit maakt het mogelijk om verschillende elementen te slepen met de computermuis. Dit kan worden gebruikt om de taken van een planbord gemakkelijk in te plannen.

Websockets

Door middel van *websockets* kan informatie worden gestuurd naar andere gebruikers die op dat moment de applicatie aan het gebruiken zijn. Het kan worden toegepast om andere gebruikers te informeren over een taak die wordt ingepland. Hierdoor moet het niet langer mogelijk zijn om een afspraak dubbel in te plannen.

Virtual Scrolling *(Recyclerview)*

Virtual Scrolling is eigenlijk een andere naam voor een *recyclerview*. *recyclerview* komt uit *android development*.

Als een lijst 10.000 velden bevat, zou dit normaliter allemaal tegelijkertijd worden ingeladen en ook ingeladen blijven. Als deze velden veel data bevatten kan dit voor een trage applicatie zorgen.

Met *virtual scrolling* worden er zoveel velden geladen die op het scherm passen. Daarnaast worden er nog enkele velde ningeladen buiten het scherm. Zodra de gebruiker over de pagina scrollt worden de oudere velden verwijderd en de nieuwe ingeladen. Dit zorgt ervoor dat de applicatie sneller reageert.

Zoals is te zien in **Figuur 7** zijn alle gewenste functionaliteiten mogelijk in elk framework. Sommige scoren echter wel beter dan andere. Bijvoorbeeld bij Angular is het goed dat er maar twee verschillende onderdelen bij hoeven te komen, omdat meer onderdelen de grootte van de applicatie zou kunnen beïnvloeden.

DoneJS heeft een *native* implementatie van *websockets*. Dit betekent dat er niets geïnstalleerd hoeft te worden. Verder zal er jQuery en jQueryUi worden geïnstalleerd.

AngularJS, React en MarionetteJS hebben wel de mogelijkheid om dit werkend te krijgen, maar moeten wel meerdere onderdelen gebruiken. De mogelijkheid om bij alle implementaties hetzelfde te gebruiken is echter wel een pré, omdat de implementatie van deze onderdelen grotendeels hetzelfde zal blijven.

Voor de puntentelling zal er worden gekeken naar de hoeveelheid *plugins* die moeten worden geïnstalleerd om de functionaliteiten werkend te krijgen. Het *framework* met de minste plugins zal hoger scoren.

Omdat de puntentelling van de *frameworks* te dicht bij elkaar lagen is er dieper ingegaan op de materie. Angular is de winnaar, omdat de enige *plugin* die naast socket.io moet worden geinstaleerd, is gebouwd door hetzelfde team die Angular hebben ontwikkeld.

Dit geldt ook voor AngularJS, want AngularJS Material is van dezelfde maker. Deze scoort echter lager, omdat er twee verschillende *plugins* nodig zijn.

DoneJS heeft een *native* functionaliteit en er moeten dus twee verschillende *plugins* worden geïnstalleerd.

React heeft drie verschillende *plugins* nodig, maar in tegenstelling tot andere *frameworks*, kan dezelfde *plugin* worden gebruikt.

DojoToolkit scoort het laagst, omdat er ook een implementatie wordt geadviseerd die anders is dan de meeste *frameworks*.

RANKING

Angular	**6**
DoneJS	**5**
AngularJS	**5**
Dojo.IO	**5**
React	**4**
MarionetteJS	**3**
DojoToolkit	**2**

6.1.5. Community

Als laatste worden de *communities* van de verschillende *frameworks* onderzocht. Een *framework* met een grote *community* is in het algemeen gemakkelijker te gebruiken. Dit komt doordat er meer vragen, handleidingen en ontwikkelingen voor dat *framework* zijn.

Bij de vergelijking is er gekeken naar drie verschillende onderdelen: *Contributors, Commits* en *Stars.* Deze informatie is verkregen via de *repositories* die worden *gehost* op Github.

Naam	Community		
	Contributors	Commits	Stars
Angular	845	13.065	45.352
AngularJS	1.598	8.954	59.384
Dojo Toolkit	...	...	...
Dojo.IO	40	2.647	114
DoneJS	38	1.798	1.299
React	1.281	10.665	122.529
MarionetteJS + Backbone*	618	6.732	34.531
MarionetteJS	325	3.367	7.146
Backbone	293	3.365	27.385

Figuur 8: *Community per framework*

Contributors

Contributors zijn mensen die toegang hebben om het bestaande *framework* te wijzigen. Vaak wordt deze groep uitgebreid met mensen die veel voor het framework doen *(bijv. plugins of bugfixes voorstellen).*

Commits

Commits is het aantal wijzigingen dat het *framework* heeft gehad op GitHub. Dit geeft een goed beeld van de hoeveelheid veranderingen bij een *framework* sinds de eerste iteratie.

Stars

Stars wordt ook *favorites* genoemd. Iedereen met een GitHub account heeft de mogelijkheid om bepaalde *repositories* als favoriet toe te voegen. Als een gebruiker dit doet krijgt hij/zij informatie over die *repository* binnen. Dit is een indicatie van het aantal mensen die geïnteresseerd zijn in het *framework*.

MarionetteJS is berekend in combinatie met Backbone, omdat MarionetteJS niet zonder Backbone kan werken. Om de resultaten eerlijk te houden wordt Backbone ook meegenomen onder MarionetteJS. Voor de duidelijkheid staan ze ieder nog apart genoemd.

RANKING

	Contributions	Commits	Stars	Totaal	
React	5	5	6	16	6
Angular	4	6	5	15	5
AngularJS	6	4	4	14	4
MarionetteJS	3	3	3	9	3
Dojo.IO	2	2	1	5	2
DoneJS	1	1	2	4	1
DojoToolkit	0	0	0	0	0

Vanwege de verschillende onderdelen in deze test zal er per onderdeel worden gekeken. Per onderdeel zal er een cijfer worden gegeven tussen de zes en de nul. De zes wordt uitgereikt aan degene die het beste is en de nul aan de slechtste. Dit wordt per onderdeel gedaan. Daarna worden de resultaten opgeteld en de hoogste uiteindelijke score zal de meeste punten krijgen. Deze gaan ook weer van zes tot en met nul.

React scoort in het algemeen het beste. Dit is geen gigantische verrassing. Er zit een goede marketingmachine achter React die ervoor zorgt dat het framework bekend is.

Angular komt op de tweede plaats. Het *framework* heeft veel bekendheid vanwege de mogelijkheid om zowel een mobiele als desktop applicatie te maken.

AngularJS heeft ook nog een grote *community.* Dit is door de jaren heen opgebouwd en een gedeelte van de bestaande *community* zal meegegaan zijn naar Angular. Het andere gedeelte zal bij AngularJS zijn gebleven.

De rest mist zeker wat onderdelen in de *community*. Dit hoeft echter geen spelbreker te zijn. Er moet wel nagedacht worden over de keuze, want het lijkt niet handig te zijn om een *framework* te gebruiken die niet meer in trek is.

6.1.6. Conclusie

In dit onderdeel zal er een conclusie worden getrokken uit alle testen die zijn uitgevoerd. De methode hierachter zal nog verduidelijkt worden. Als laatste wordt er nog een advies geschreven.

In totaal zijn er vijf verschillende testen uitgevoerd:

- Functionaliteiten per *framework;*
- Grootte van *framework;*
- Laadsnelheid;
- *Plugins* voor extra functionaliteit
- Bekendheid

Aan het einde van elke test is er een tussenstand gegeven over welk *framework* het beste scoorde in de vergelijking. Het totaal van deze punten wordt opgeteld en het framework met de hoogste score heeft de meeste potentie voor het project.

	Functies	Snelheid	Formaat	Plugins	Community	Totaal	Eindscore
Angular	6	2	3	6	5	22	2 de #
DoneJS	5	1	6	5	1	18	3 de #
AngularJS	5	4	5	5	4	23	1 ste #
Dojo.IO	5	0	2	5	2	14	5 the
React	4	3	6	4	6	23	1 ste #
MarionetteJS	3	5	6	3	3	20	4 the
DojoToolkit	5	0	4	2	0	11	6 the

Advies

Na alle onderdelen te hebben onderzocht , is er een score uitgekomen welk *framework* het beste is om te gebruiken. Daarnaast is er ook een advies geschreven voor de opdrachtgever om te bepalen wat er moet gebeuren.

AngularJS en React hebben een gedeelde eerste plaats en Angular staat op de tweede plek. Daarna komt DoneJS als derde.

Het advies is om React te gaan gebruiken voor het planbord. Dit heeft grotendeels te maken met de kennis dat AngularJS alleen nog maar wordt geüpdatet als er *security* problemen zijn. Het *development* team is van plan dit voor drie jaar lang zo te houden. Dit betekent dat er geen nieuwe ontwikkelingen meer zullen zijn bij AngularJS. *(“AngularJS”, z.d.)*

Angular zou een andere optie kunnen zijn als Adaption problemen krijgt met React. Beiden hebben volgens dit onderzoek goede en slechte punten. React is licht en snel, terwijl Angular meer functionaliteiten bezit.

Het bedrijf verandert echter liever niet van de huidige implementatie, gebaseerd op MarionetteJS + Backbone. Het advies is om dit los te laten. Er missen functionaliteiten en het *framework* lijkt tegenwoordig weinig ondersteuning te krijgen.

6.1.7. Resultaat

React is sneller en kleiner dan Angular, maar Angular heeft meer functionaliteiten tot zijn beschikking waardoor minder *plugins* nodig zijn. Daarnaast wenst Adaption liever door te gaan met het huidige prototype.

Adaption is realistisch en wil niet met een framework werken dat niet goed meer ondersteund wordt. Dit kwam echter niet voldoende bij het onderzoek naar voren.

Er is gevraagd het onderzoek uit te breiden om React, Angular en MarionetteJS nog beter te onderzoeken. Ten eerste een onderzoek naar de levensvatbaarheid. Dit zal grotendeels worden gedaan door middel van de GitHub pagina te inspecteren en de *tools* te gebruiken.

Ook moet er gekeken worden naar de *requirements* en welk framework sommige onderdelen mogelijk niet kan bieden. Als laatste wordt er gekeken naar een vergelijking die ieder jaar wordt gedaan over Javascript en de verschillende *frameworks*: StateOfJS, Mogelijk kan hier nog data uit worden gehaald om een beter beeld te krijgen.

De planning zal gedeeltelijk worden aangepast om deze verandering te ondersteunen. In het volgende hoofdstuk zal hier dieper op in worden gegaan.

6.2. Gevolgen van uitgebreid onderzoek

De resultaten van het vooronderzoek zijn besproken met mijn bedrijfsbegeleider. Vanuit hem kwam de vraag om nog een extra onderzoek te doen waarbij ik dieper in moest gaan op de *community*. Tijdens de bespreking kwamen wij tot de conclusie dat dat een belangrijk onderdeel kan zijn met betrekking tot het kiezen van een *framework*. Dit kwam eigenlijk door het resultaat van AngularJS, die goed gescoord heeft in het vorige onderzoek maar waarvan de levensvatbaarheid een onzekere toekomst heeft.

Het is dus van belang dat er wordt gekeken naar hoeveel er wordt gedaan met het *framework*. Door de *community* beter te onderzoeken kan uitsluitsel geven. Om het onderzoek te versnellen is gekozen om drie *frameworks* te vergelijken. Ook de planning zal hierop moeten worden aangepast.

Subject / Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Eenmalige Activiteiten																	
Plan van Aanpak																	
Requirements																	
Vooronderzoek																	
Uitgebreid onderzoek																	
Iteratief Proces																	
Ontwerpen																	
Realiseren																	
Testen																	
Demo																	

Figuur 9: *Planning na het vooronderzoek*

Er worden in totaal twee weken extra gebruikt om dit onderzoek uit te voeren. Dit betekend echter wel dat er minder tijd zal zijn voor het realiseren van het planbord. Aangezien de demo-momenten vast staan zal ik tijdens de eerste vergadering niets concreets hebben om te tonen. Ik heb vooralsnog de laatste twee weken vrij gehouden in verband met uitloop en het verslag.

De *scope* van het project is ook veranderd. De intentie was om het *framework* uit te breiden met nieuwe functionaliteiten voor verschillende soorten klanten. De kans was aanwezig dat er op dit punt dan veel kleine wijzigingen zouden worden gedaan voor veel klanten en dat zij dit hopelijk konden gebruiken.

Om deze reden is er in overleg met Adaption gekozen om een soort gebruiker uit te kiezen en hierop te focussen. Er is gekozen om de *charters* zoveel mogelijk uit te werken. Dit vereist echter dat de *requirements* anders worden opgezet.

Nieuwe Requirements

De *requirements* die waren verzameld, zijn opnieuw samengevat en verwerkt in *user stories*. Ook zijn er nieuwe *requirements* toegevoegd, De kans bestond dat bepaalde *requirements* niet aan bod zouden komen. Een voorbeeld hiervan is het converteren naar een nieuw *framework* wat aan de hand van de resultaten van het onderzoek mogelijk niet werd uitgevoerd.

	Who?	What?	Why?
1	Developer	Convert the current planboard to the new framework	So i have the same functionality
2	Manager	Different time layouts for a planboard	Zoom in there right time slots
3	Planner	Filters for the orders on the planboard	Better overview of my work
4	Planner	Order gets a different color depending on its status.	Easily distinguish between orders
5	Planner	Detailed information when i hover over an order	See the information on this screen
6	Planner	I would like to use filters on unplanned orders	So i can find what i am looking for
7	Planner	Put the planboard in a separate frame	Quickly get started with my work
8	Planner	Post-condition(planbord) checks	So i can not plan wrong data
9	Planner	Planboard to support infinite loading	Quickly get started with my work
10	Planner	I would like to see a route that will be taken	See what location they are close to
11	Planner	I want to be able to put an agenda item on a resource	So i can see a note quickly
12	Planner	Filter on certain condition and draft the correct price	Good decision where to put an order
13	Planner	I want to be able to resize the windows	Zoom in the window that's important
14	Planner	Reschedule an order on a different day and charter	Quickly make last minute edits
15	Planner	I want a charter to be collapsible	Don't have to scroll through them
16	Planner	Edit an relation on a charter	So i can quickly make edits
17	Planner	Define time slots while using the planbord	Better overview of my work

Figuur 10: *Lijst van user stories die verzameld zijn.*

De verschillende *requirements* zijn per sprint gepland en zullen bij **6.4.1 Eerste Sprint** en **6.4.2 Tweede sprint** worden geïntroduceerd en besproken.

6.3. Uitgebreid vooronderzoek

Na aanleiding van het eerste onderzoek dat was uitgevoerd, is er samen met mijn opdrachtgever besloten om bepaalde *frameworks* nog dieper op in te gaan. Er moet worden onderzocht hoe levensvatbaar de verschillende *frameworks* nog zijn en of ze alle *requirements* die verzameld zijn, kunnen uitvoeren.

Er zijn in totaal drie *frameworks* die vergeleken worden. Als eerste wordt *React* bekeken, omdat deze in het vorige onderzoek erg goed gescoord heeft. AngularJS is niet meegenomen en de beweegreden kunnen worden gelezen in **6.1.6 conclusie**.

Om deze reden is Angular wel meegenomen. Deze stond in het vorige onderzoek op de tweede plaats en heeft meer potentie voor de toekomst. Als laatste is het huidige *framework* MarionetteJS nog meegenomen, zodat dit kan worden vergeleken met de andere *frameworks*.

6.3.1. GitHub Analyse

Er is een uitgebreide analyse uitgevoerd door middel van de statistieken die GitHub bijhoudt. Er is veel data te vinden waar informatie uit kan worden gehaald. Per onderdeel zullen de *frameworks* tegenover elkaar worden gezet en bevindingen worden gedocumenteerd. Elk onderdeel wordt kort besproken alsmede de werkwijze bij dat onderdeel.

Flat Analysis
De statistieken die zijn te lezen op de GitHub pagina zijn beoordeeld en vergeleken. Dit is om een eerste indruk te krijgen per *framework*.

Commits
Het aantal *commits* die binnen een jaar zijn uitgevoerd zijn vergeleken een eigenaardigheden zijn uitgesproken.

Release Windows
De *releases* van het *framework* zijn verzameld en er is een analyse uitgevoerd om te achterhalen hoe vaak een nieuwe *release* wordt uitgebracht.

Pulse
Pulse laat de activiteiten zien binnen het framework in een bepaalde periode. Dit in combinatie met de *flat analysis* geeft een goede *overview* van de *frameworks*.

Contributors
Contributors zijn de mensen die het meeste hebben bijgedragen aan de ontwikkeling van het *framework*. Voor de analyse zijn de top drie *contributors* gebruikt. Vanwege privacyredenen zijn zij geanonimiseerd.

Issues
De *issues* zijn problemen die gebruikers ervaren met het *framework*, hier is een analyse over gemaakt waarbij wordt gekeken hoelang het nog duurt voordat alle bestaande *issues* zijn opgelost.

6.3.1.1. Flat Analysis

Bij dit onderdeel zijn de statistieken bekeken die te zien zijn als de desbetreffende GitHub pagina wordt geopend per *framework*. Dit is gedaan om een beeld te krijgen van de verschillende *frameworks*.

Per *framework* is het nu gemakkelijk om zowel de verschillen en overeenkomsten te zien. Er moet worden genoteerd dat deze data zijn verzameld op 19/02/2019. Als er op een later tijdstip via de bijgeleverde *links* informatie wordt vergaard de data niet langer hetzelfde is.

MarionetteJS	React	Angular
3,369 Commits 6 Branches 149 Releases 325 Contributors 74 Issues 6 Pull Requests	10,665 Commits 31 Branches 110 Releases 1,281 Contributors 399 Issues 141 Pull Requests	13,065 Commits 58 Branches 297 Releases 845 Contributors 2321 Issues 392 Pull Requests
MIT License	MIT License	MIT License
Link	Link	Link

Figuur 11: *Simpele vergelijking van de beschikbare data.*

Het is hier niet de bedoeling om een ranking te maken van de verschillende *frameworks*, maar om snel een overzicht te krijgen per onderdeel waar in de volgende hoofdstukken dieper op wordt ingegaan. Wel zijn er al enige eigenaardigheden te ontdekken.

MarionetteJS heeft beduidend minder statistieken vergeleken met React en Angular. Het is wel belangrijk om rekening te houden met het feit dat MarionetteJS in combinatie moet worden gebruikt met Backbone.

React heeft veel *contributors*. Een verklaring hiervoor zou kunnen zijn dat het framework wordt onderhouden door Facebook. Mensen zijn waarschijnlijk geïnteresseerd wat er wordt gedaan met dit *framework*.

Bij Angular staan er in vergelijking met de rest veel *issues* open. Dit kan duiden op ontevredenheid onder de gebruikers maar mogelijk zijn er nieuwe functies toegevoegd die nog goed moeten worden getest. Verder onderzoek zou moeten uitwijzen of dit ook werkelijk het geval is.

6.3.1.2. Pulse

Via een tab genaamd *insights* op de GitHub pagina is het mogelijk om data van de *repository* in te zien. Eén van deze mogelijkheden is *pulse*. Dit kan worden ingesteld op verschillende periodes, er is in dit onderzoek gekozen om de vergelijking op maandbasis uit te voeren. Zodat incidentele afwezigheid minder impact heeft.

De selectie is van 19/01/2019 t/m 19/02/2019. Naast de getoonde onderdelen in **figuur 12** is er ook een lijst van *contributors*. Deze zijn geanonimiseerd.

Figuur 12: *De pulse van de drie verschillende frameworks*

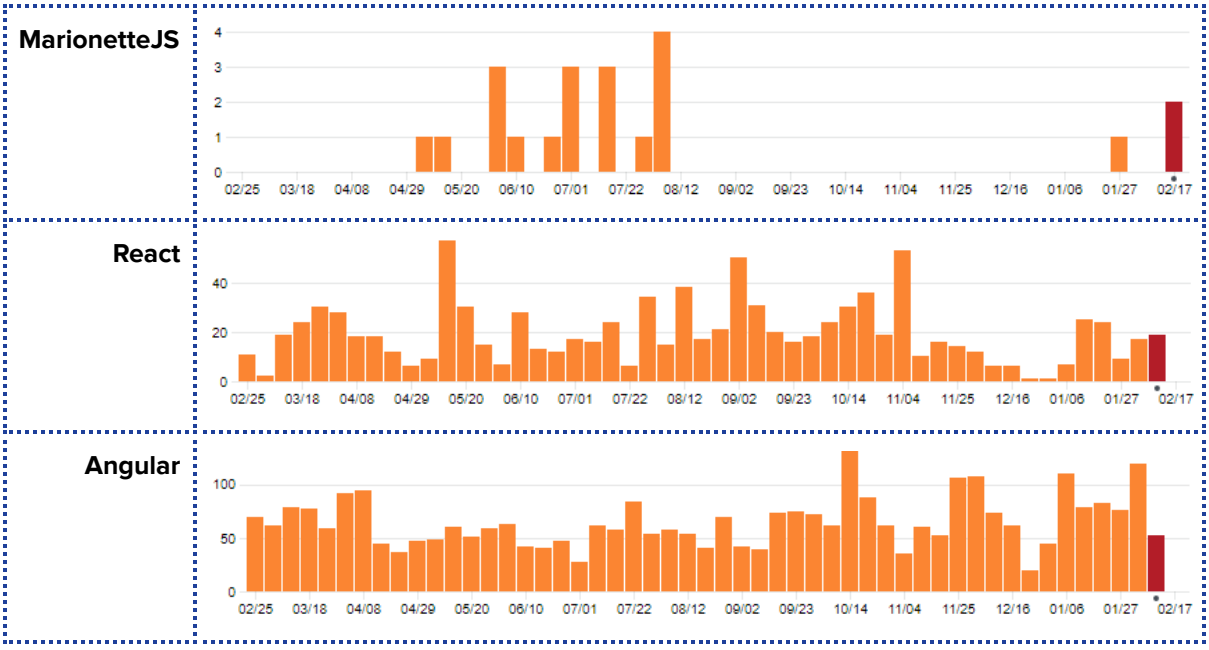
Nogmaals wordt benadrukt hoe groot React en Angular zijn in vergelijking met MarionetteJS. Wel moet er worden vermeld dat in een andere maand dit mogelijk niet het geval hoeft te zijn. Het was helaas niet mogelijk om dit te herleiden door middel van de *pulse* omdat de grootste selectie al was uitgevoerd.

Angular heeft veel nieuwe *issues* gekregen en weinig *pull requests* samengevoegd. Zelfs MarionetteJS heeft meer *pull requests*.

React ziet er goed uit al is er een persoon die veel doet aan het *framework*, de vraag is wat er zou gebeuren als deze wegvalt?

6.3.1.3. Commits

De *commits* die met elkaar worden vergeleken zijn van het gehele *framework*. Er is een jaaroverzicht gegenereerd met daarin de informatie over hoeveel *commits* er zijn uitgevoerd op weekbasis. Door dit te vergelijken is het mogelijk om trends te ontdekken en een vergelijking te maken tussen de verschillende *frameworks*



Figuur 13: *Week notaties van de commits per framework*

Zoals is te zien bij **figuur 13** zijn er bij MarionetteJS weinig *commits* uitgevoerd, De kans bestaat dat dit een beleid is binnen het team van MarionetteJS. Echter baart de periode tussen 08/12 en 01/27 mij wel zorgen. Dit is een lange tijd zonder een *update?*

Bij React verschilt de hoeveelheid *commits* per week. Dit kan mogelijk te maken hebben met het opbouwen naar een nieuwe *release* of functionaliteiten die worden toegevoegd.

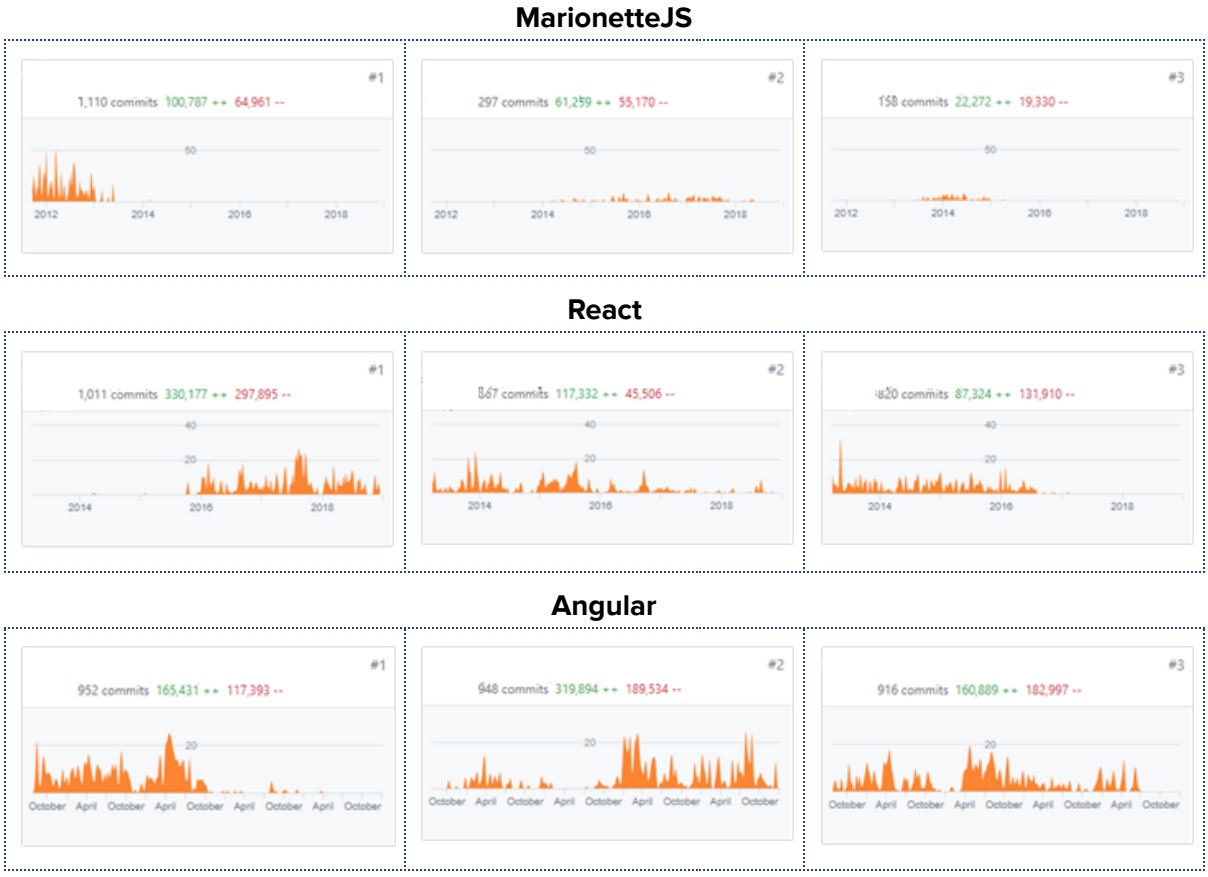
Angular heeft gemiddeld per week minimaal 50 *commits*. Dit is een opmerkelijk resultaat maar hoeft niet fout te zijn.

MarionetteJS staat er niet goed voor na deze vergelijkingen. Zelfs als we rekening houden met het feit dat het een klein *framework* is zijn de resultaten in vergelijking met Angular en React minimaal te noemen.

6.3.1.4. Contributors

Bij *contributors* is er gekeken naar de top drie. Deze keuze is gemaakt omdat het veel tijd in beslag zou nemen om alle *contributors* te vergelijken. De hoeveelheid *commits* is bij de *contributors* met een lagere rang niet echt te vergelijken.

In deze vergelijking zijn alle *commits* van de zogenaamd *master branch*. Er is ook te achterhalen hoeveel regels code zij hebben toegevoegd en verwijderd. De tijdlijn geeft aan wanneer ze dit hebben aangepast.



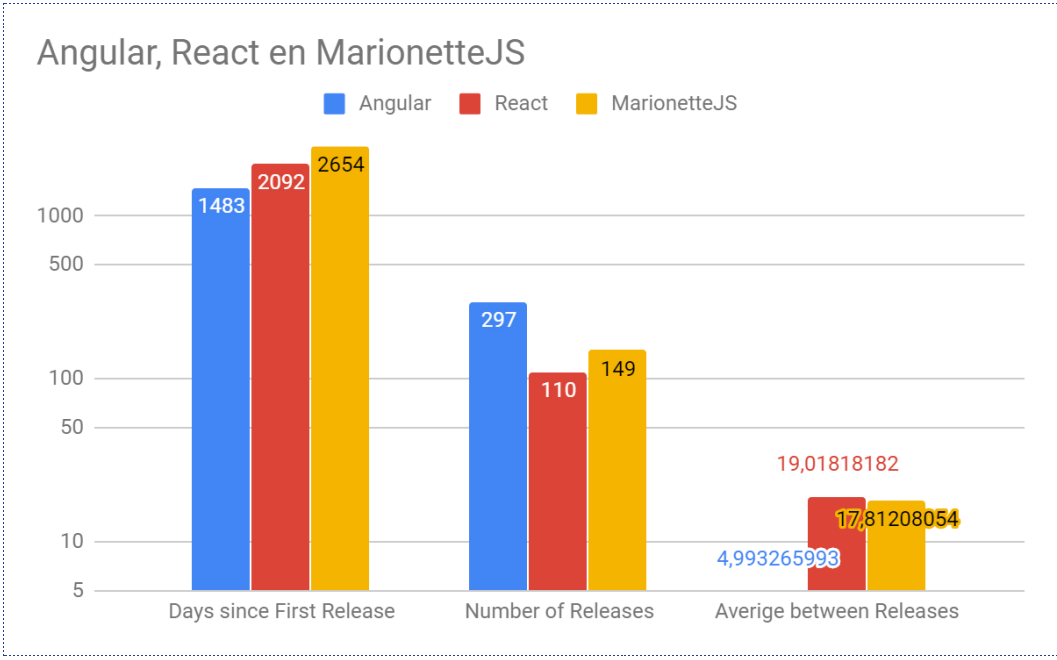
Figuur 14: *Verschillende top 3 contributors per framework*

Zoals in **figuur 14** is te zien had MarionetteJS in 2012 een grote *contributor*, tegenwoordig lijkt deze persoon zich niet meer bezig te houden met het *framework*. Bij React zijn twee van de drie *contributors* nog bezig met het *framework*.

Bij Angular was er geen informatie over het gehele jaar beschikbaar maar bepaalde data kan nog steeds worden verzameld. De eerste *contributor* is tegenwoordig niet meer met het *framework* bezig. Het is trouwens opmerkelijk dat de top 25 *contributors* van Angular allemaal minimaal 100 *commits* hebben uitgevoerd.

6.3.1.5. Release Window

De data die wordt getoond in **figuur 15** is verzameld op 19/02/2019. Vanwege de gevoeligheid voor verandering van de desbetreffende informatie moet dit worden gemeld.



Figuur 15: *Aantal releases per framework*

Er is gekeken wanneer de *frameworks* voor het eerst op GitHub zijn gezet. Hierna is er een verzameling van alle *releases* gemaakt. Er is geen onderscheid gemaakt wat voor soort *release* er is uitgevoerd omdat dit niet bij elk framework even duidelijk was.

Zoals is te zien in **figuur 15** bestaat MarionetteJS het langste volgens GitHub. Het framework bestaat mogelijk langer, maar deze data was niet te achterhalen.

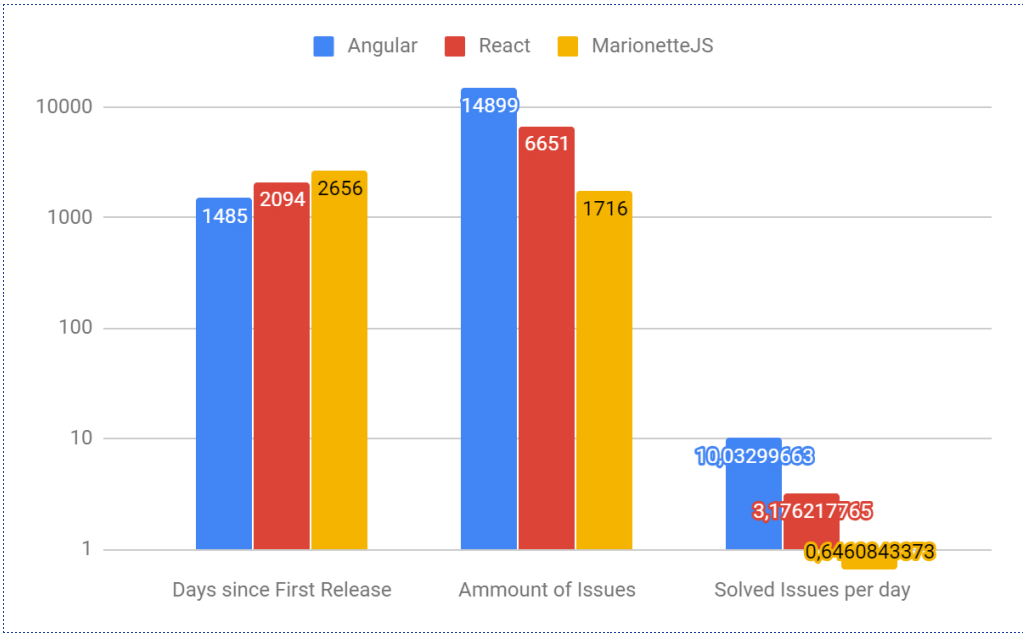
Het is opmerkelijk hoe snel Angular gemiddeld een nieuwe *release* heeft. Het gaat hier uiteraard om een gemiddelde maar vergeleken met de andere *frameworks* is dit erg snel.

Zowel MarionetteJS en React hebben een redelijk tijdframe tussen elke *release* zitten.

Angular heeft elke werkweek een nieuwe beschikbaar gesteld terwijl React een maand doet over een nieuwe versie. Marionette doet er ongeveer drie weken over om een nieuwe versie over te brengen.

6.3.1.6.Issues

Bij de *issues* is er een vergelijking gemaakt hoeveel *issues* er per dag gemiddeld worden opgelost. Dit wordt gedaan door het aantal gesloten *issues* te delen door het aantal dagen sinds de eerste *release*. Deze data zijn verzameld op 21/02/2019.



Figuur 16: *Verschillende weeknotaties van de commits per framework*

In **Figuur 17** staan de hoeveelheid *issues*, zowel gesloten als geopend.

Angular lost de meeste op per dag. Echter hebben zij ook de meeste om op te lossen. MarionetteJS lost het minste op qua *issues*. Wel hebben ze in totaal de minste *issues*.

Framework/Issues	Open	Closed	Total
Angular	2.314	14.899	17.223
React	406	6.651	7.057
MarionetteJS	48	1.716	1.764

Figuur 17: *Tabel van hoeveelheid issues*

Er is daarnaast nog berekend hoeveel dagen nodig zijn om alle *issues* die nu openstaan te kunnen sluiten. Het betreft een gemiddelde wat is berekend in de realiteit kan dit verschillen.

Angular heeft nog in totaal 231 dagen nodig om alle open *issues* te sluiten. React heeft er daarentegen gemiddeld 128 dagen voor nodig om de bestaande open *issues af te ronden.* MarionetteJS heeft nog 75 dagen nodig om de openstaande *issues* te verwerken.

6.3.1.7. Beoordeling

Na de verschillende *frameworks* te hebben onderzocht, kan worden geconcludeerd dat MarionetteJS een tijd niet is doorontwikkeld. Er is volgens GitHub bijna een jaar lang geen enkele *commit* uitgevoerd. Sinds kort is er weer activiteit gaande bij dit *framework*. De vraag is echter of dit *framework* dan moet worden gebruikt om door te ontwikkelen. Vergeleken met Angular en React is er teveel verschil. Het is bijna niet mogelijk dat het MarionetteJS *framework* in een jaar lang geen *issues* heeft gehad die moesten worden verholpen.

Angular heeft een goed aantal *releases* gehad en aan de hoeveelheid *commits* is te zien dat er duidelijk voortgang wordt gemaakt met het *framework*. De hoeveelheid openstaande *issues* en weinig *merges* zijn echter zorgelijk. Ik concludeer dat er veel experimentele *features* worden toegevoegd en de gebruiker deze kan gaan testen. Dit kan leiden tot een onstabiel *framework* waardoor het upgraden naar een nieuwe versie lastig kan zijn,

React heeft een mooi *releaseschema* en een goed aantal *merges*. De hoeveelheid *issues* zeker in vergelijking met Angular zijn klein. Over het algemeen is er weinig aan te merken op zowel de ontwikkeling en werkwijze bij dit *framework*.

Als alle vergelijkingen naast elkaar worden gelegd concludeer ik dat er gekozen moet worden voor het React *framework* in deze vergelijking. React lijkt een goed ontwikkeltraject te hebben en er is interesse in het *framework*.

Er moet nog worden gekeken naar de eisen en de mogelijkheden die elk *framework* te bieden heeft. Het advies om React te gebruiken is momenteel gebaseerd op de vergelijkingen die zijn gedaan. Bij **6.3.2 Analyse van Eisen** en **6.3.3 StateOfJS Analyse** wordt er nog gekeken naar de andere aspecten van de verschillende *frameworks*.

6.3.2. Analyse van Eisen

Het is van belang om de *requirements* die zijn verzameld nogmaals te verifiëren. Alle *requirement* zijn in **figuur 18** per *framework* onderzocht of de gewenste functionaliteit mogelijk was Het nummer correspondeert aan de hand van de nummers die bij **6.2 gevolgen van uitgebreid onderzoek** naar voren zijn gekomen.

6.3.2.1. Requirements

Per *requirement* is er onderzocht of de mogelijkheid bestond om deze te realiseren per framework. In totaal zijn er twee verschillende *requirements* die uitgebreid zullen worden beschreven. Zoals is te zien in **figuur 18** is het mogelijk om de gewenste functionaliteiten te realiseren in elk *framework*. De oplossing kan per *framework* uiteraard verschillen.

Story	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
MarionetteJS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Angular	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
React	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Figuur 18: *Mogelijkheden per framework*

De *requirements* die uitgebreid zullen worden beschreven zijn:

- **Nummer 7:** Planbord in een apart venster zetten
- **Nummer 13:** Vergroten en verkleinen van bepaalde elementen in het planbord

Daarnaast is het ook van belang om te achterhalen of het *framework* kan worden gebruikt binnen de huidige productieomgeving.

6.3.2.2. Beoordeling

De *requirements* die hierboven zijn beschreven zijn onderzocht maar er is geen doorslaggevend resultaat geboekt. Het was al duidelijk alle *requirements* konden worden uitgevoerd in elk framework.

Een nieuw venster openen moet geen enkel probleem zijn vanwege het *command window.open*. Hier kunnen verschillende waardes aan mee worden gegeven om data te versturen. *(“Window.open()”, 2019)*

De mogelijkheid om elementen te vergroten en verkleinen moet ook geen probleem opleveren. Tegenwoordig is dit een functionaliteit van HTML zelf. *(“CSS resize property”, z.d.)*

De beoordeling van dit onderdeel is daarom lastig te noemen. Er is daarom gekeken welke functionaliteiten momenteel al aanwezig zijn in het prototype. Voor dit onderdeel zal MarionetteJS worden gekozen zodat wat al aanwezig is kan worden hergebruikt en nieuwe functionaliteiten kunnen worden gerealiseerd.

6.3.3. StateOfJS Analyse

StateOfJS is een enquête die elk jaar sinds 2016 wordt verstuurd naar *developers* om de mening over de programmeertaal Javascript te peilen. De *frameworks* die geanalyseerd worden kunnen per jaar verschillen en er komen verschillende onderdelen in voor. *("State Of JavaScript: The State Of JavaScript", z.d.)*

Voor het onderdeel dat geanalyseerd is kunnen personen antwoord geven op de vraag of zij een *framework* kennen en wat hun ervaring is. De antwoorden bestaan uit:

- *Never heard of it;*
- *Heard of it, not interested;*
- *Heard of it, would like to learn;*
- *Used it before, won't use again;*
- *Used it before, would use again.*

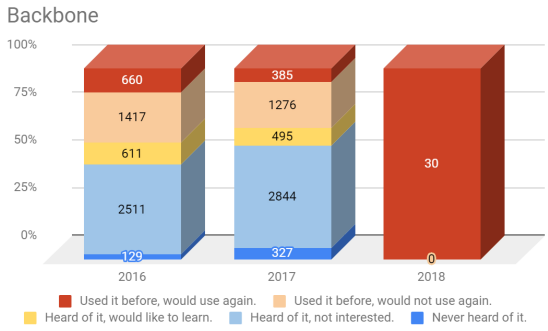
Er is data verzameld van de jaren 2016, 2017 en 2018. Deze zijn vergeleken en mogelijke verschillen of trends zijn ontdekt. Voor het jaartal 2018 zijn er enkele aspecten die moeten worden toegelicht. Bij zowel 2016 en 2017 was AngularJS en Angular gescheiden. Sinds 2018 is dit onder één naam ondergebracht. Mogelijk heeft dit de resultaten beïnvloed. *("The State Of JavaScript 2017", z.d.)*

Backbone wordt gebruikt in MarionetteJS. Vandaar dat dit is onderzoekt. Echter stond Backbone in 2018 niet langer in de top 10 *frameworks* en is er weinig informatie over verzameld. *("The State of JavaScript 2018", z.d.)*

Elk jaar zijn de hoeveelheid respondenten groter geworden. Hierdoor werd het steeds lastiger om trends te ontdekken. Er is toen gekozen om de resultaten van 2017 en 2018 terug te schalen naar de hoeveelheid respondenten van 2016.

Backbone

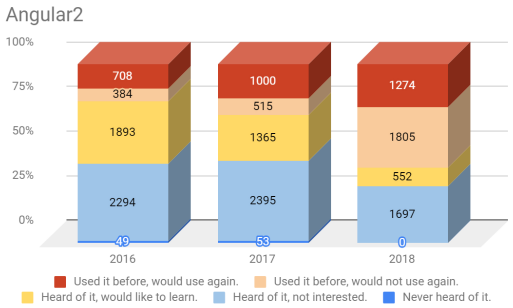
Bij backbone is er grotendeels gekeken naar de jaartallen 2016 en 2017. Te zien is dat de tevredenheid is afgenomen. De grootste verschuiving zit hem in de mensen die nog nooit van het *framework* hadden gehoord en de hoeveelheid die het nogmaals willen gebruiken.



Angular

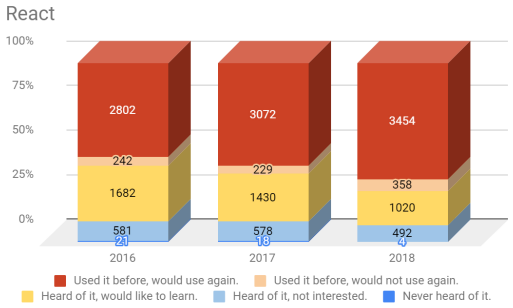
Bij Angular is grotendeels een groeiende lijn te zien. Elk jaar zijn meer mensen het gaan gebruiken. De ontevredenheid is in 2018 wel gestegen. Een mogelijke verklaring hiervoor is dat zowel Angular als AngularJS onder dezelfde kop zijn benoemd.

Het is mogelijk dat een gedeelte dat niet geinteresseerd was , het mogelijk toch heeft geprobeerd. En het uiteindelijk niets vond.



React

Bij React is grotendeels groei, en dan ook de goede kant op. In 2018 is het aantal mensen die het niet meer willen gebruiken gegroeid. Echter zijn er ook meer mensen die het *framework* hebben uitgeprobeerd. De kans dat er dan mensen zijn die het niets vinden, bestaat.



De verzamelde statistieken zijn te vinden in **figuur 19.** Er is gecalculeerd moet hoeveel procent een bepaalde keuze is gegroeid of gekrompen. Dit wordt aangegeven door middel van een rode of groene achtergrond. Rood is slecht en groen wordt gezien als goed. Geel betekent dat het niet perse goed of slecht is. De volledige calculatie is te vinden in **11.3 Bijlage C: StateOfJS onderzoek calculaties**

	MarionetteJS			Angular			React		
	2016	2017	2018	2016	2017	2018	2016	2017	2018
Never heard of it.	2,42%	6,14%	...	0,92%	1,00%	0,00%	0,39%	0,35%	0,07%
Heard of it, not interested.	47,13%	53,18%	...	43,06%	44,96%	31,85%	10,90%	10,85%	9,23%
Heard of it, like to learn.	11,47%	9,29%	...	35,53%	25,62%	10,37%	31,57%	26,84%	19,14%
Used it, not again.	26,60%	23,95%	...	7,21%	9,66%	33,88%	4,54%	4,30%	6,72%
Used it, would use again.	12,39%	7,24%	0,56%	13,29%	18,77%	23,91%	52,29%	57,67%	64,83%

Figuur 19: *Groei of afname per jaar per framework*

6.3.4. Advies

Er zal nu een voorstel worden gedaan aan de hand van de resultaten die zijn verzameld. Zoals al was gemeld bij **6.3.1 GitHub Analyse** was de keuze gevallen op het *framework* React. Dit vanwege het feit dat Angular veel *issues* heeft en MarionetteJS weinig *activiteit* leek te hebben.

Daarna is er gekeken naar de nieuwe *requirements* en of dit mogelijk was bij elk *framework*. Er was hier gekozen om MarionetteJS te gebruiken omdat alle *requirements* mogelijk zijn in elk *framework* alleen bij MarionetteJS is er al een basis aanwezig omdat het prototype hierin is gerealiseerd.

Bij de laatste vergelijking is de winnaar duidelijk React. Er is een duidelijke groei op de aspecten waar dat belangrijk is. Bij Angular lijkt ontevredenheid te heersen en MarionetteJS lijkt niet echt relevant meer te zijn.

Ik concludeer dat React het beste *framework* is om te gebruiken voor dit project. Alle functionaliteiten die gewenst worden zijn haalbaar. Het *framework* is het lichtste, en de tevredenheid en *community* blijft groeien.

Het enige nadeel dat het gebruik van React met zich meebrengt is dat het prototype niet kan worden doorontwikkeld. Gedeelten van de code zijn mogelijk nog te gebruiken.

Resultaat

Voor de begeleider was het schokkend hoe weinig er nog werd gewerkt aan MarionetteJS toen er werd gekeken naar de verschillende GitHub pagina's. Tijdens de resultaten van StateOfJS waren zij ook niet te spreken over MarionetteJS. Om deze reden hebben ze het advies aangenomen. Er is toen besloten om het planbord te realiseren in React. Dit houdt in dat de huidige implementatie van het planbord moet worden gebouwd in React voordat er aan nieuwe functionaliteiten kan worden gewerkt.

6.4. Applicatie

Zowel het vooronderzoek als uitgebreid vooronderzoek hebben ertoe geleid dat het planbord werd gerealiseerd door middel van het *framework* React. Er is gewerkt in twee verschillende sprints die ieder vier weken in beslag hebben genomen. Aan het begin van elke sprint zullen de *requirements* die aan bod komen worden verduidelijkt.

Bepaalde onderdelen van de sprint zullen worden toegelicht met de onderbouwing waarom deze mij helpen met het aantonen dat ik bepaalde beroepstaken voldoende beheers. Als laatste wordt er aan het einde nog gereflecteerd op de sprint.

6.4.1. Eerste Sprint

Er zijn tijdens de eerste sprint verschillende *user stories* aan bod gekomen:

- *ADAPTION-13139: Planbord ombouwen naar React Framework;*
- *ADAPTION-13146: Post-condition (planboard) checks;*
- *ADAPTION-13147: Recyclerview for orders;*
- *ADAPTION-13135: Filters for the planned orders;*
- *ADAPTION-13138: Allow Rescheduling of order on another day;*
- *ADAPTION-13152: Resize elements on planbord.*

ADAPTION-13139: React Framework

De belangrijkste *userstory* die moest worden uitgevoerd, is dat het planbord opnieuw moest worden gerealiseerd door middel van het *framework* React. Het was tijdens deze sprint niet duidelijk hoeveel tijd dit ongeveer zou gaan kosten. De functionaliteiten zijn:

- Wisselen van *resources*;
- Order kunnen inplannen;
- Connectie met de *websocket*;
- Data binnenkrijgen van de *websocket*;
- Moet in de apex omgeving werken;
- Wisselen van week moet mogelijk zijn;
- Popup na slepen;
- Contextmenu toevoegen

ADAPTION-13138: Allow Rescheduling of order on another day

De orders moet worden verzameld door middel van een *websocket.* Deze orders hebben al een vaste start- en einddatum. Het enige wat wordt gewijzigd in het prototype is bij welke *resource* het wordt ingepland. Voor deze functionaliteit is het dus van belang om met de lengte van een order de positie te bepalen.

ADAPTION-13152: Resize elements in the planboard

Het planbord heeft in het prototype maar beperkte ruimte om zowel alle geplande als ongeplande orders te tonen. Door de gebruiker de mogelijkheid te geven om de grootte van bepaalde elementen te wijzigen. Kunnen zij zelf beslissen hoe het moet worden ingedeeld. De ruimte die wordt vrijgemaakt door dit te doen moet echter wel optimaal worden benut door het desbetreffende element.

ADAPTION-13146: Post condition (Planboard) Checks

Als een order wordt ingepland zijn er verschillende condities waaraan moet voldoen. Aan de hand van deze condities is het mogelijk om de gebruiker een melding te sturen als zij een order proberen in te plannen. Mogelijkheden kunnen zijn:

- Vervoerder mag vracht niet vervoeren vanwege vergunning;
- Kan de order niet vervoeren vanwege de grootte;
- Vervoerder reist niet naar de eindlocatie of kan er niet naartoe vervoeren.

ADAPTION-13147: Recyclerview for orders

Het prototype werkt erg traag. Dit komt doordat alle orders moeten worden ingelezen voordat het planbord wordt getoond. Door een *recyclerview* te implementeren kan een gedeelte van de orders worden getoond en zodra de gebruiker gaat scrollen, zullen er via de *websocket* meer orders worden opgehaald.

ADAPTION-13135: Filters for planned orders

Het moest mogelijk worden om filters in te kunnen stellen zodat de gebruiker niet alle orders te zien krijgt die mogelijk niet van toepassing zijn voor hen. Mogelijke filters zijn:

- Orders met gevaarlijke stoffen;
- orders van en naar een bepaald land vervoeren;
- Alleen orders met vliegtuig of speciaal transport.

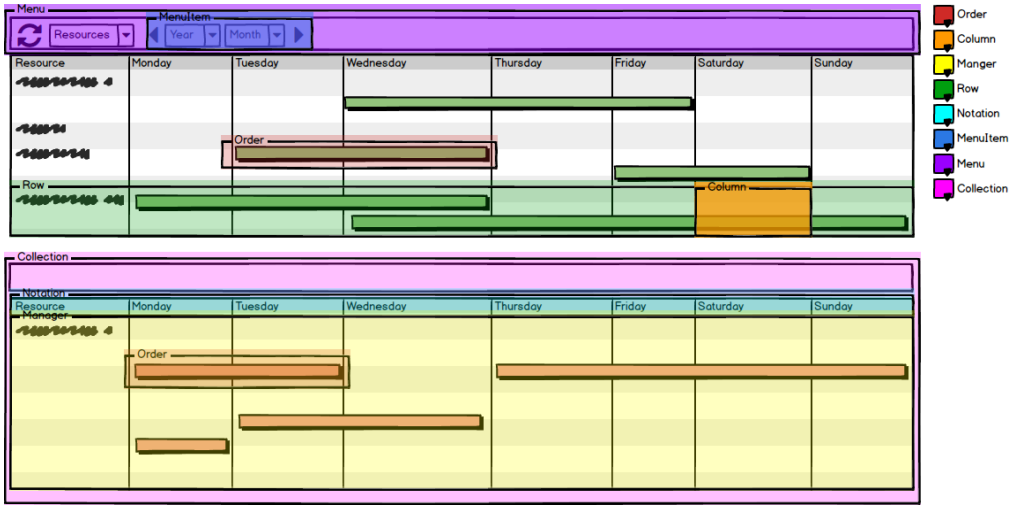
6.4.1.1. Ontwerpen van software

Per *user story* waar het ontwerpen en rol heeft gespeeld, zal worden beschreven wat er is uitgevoerd. Ook wordt er beargumenteerd waarom het op deze manier is gedaan.

ADAPTION-13139: React Framework

De eerste taak die werd opgepakt was het ontwikkelen van een planbord in het React *framework*. Het *framework* was mij onbekend en de documentatie heeft mij geholpen tijdens zowel de ontwerpfase als ontwikkelfase. Tijdens het lezen van deze documentatie was er een hoofdstuk die uitleg gaf hoe zij een programma ontwerpen *(“Thinking in React – React”, z.d.)*

Hierin wordt uitgelegd hoe zij een applicatie opbouwen met volgens hun de juiste manier van denken. De stappen die hierin worden beschreven zijn uitgevoerd op het prototype. Het resultaat daarvan is te zien in **figuur 20**.



Figuur 20: *Voorbeeld van het ontwerp van "Thinking in React"*

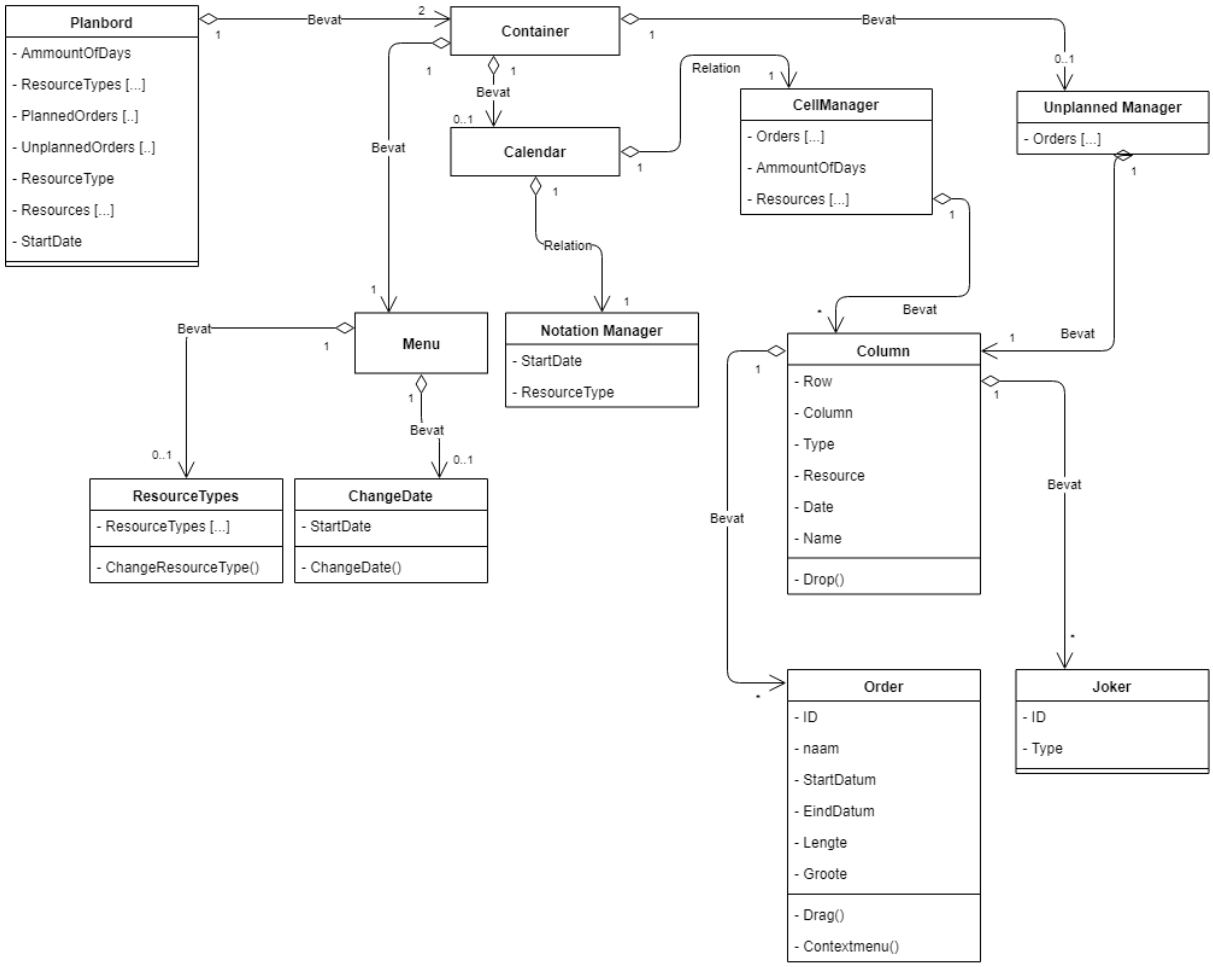
Door deze werkwijze te gebruiken was het al snel mogelijk om een lijst van zogenaamde componenten te beschrijven die nodig zijn voor het planbord. Deze zijn als volgt:

- **Orders:** houden de informatie vast en kunnen worden versleept;
- **Column:** houdt order vast en heeft informatie voor het correct plaatsen van orders;
- **Row:** houdt de kolommen vast zodat de applicatie weet welke orders bij welke resource horen;
- **Manager:** houdt alle rows vast in dat specifieke onderdeel van het planbord;
- **Notation**: laat de dagen zien in een rij, apart plaatsen zodat dit altijd zichtbaar is;
- **MenuItem:** bezit een bepaalde functionaliteit voor filteren of andere layout;
- **Menu:** houdt alle menu items vast;
- **Collection:** alle elementen voor of de geplande of ongeplande orders.

Nadat deze componenten waren waargenomen en kort beschreven kon er een klassendiagram worden gemaakt. Zoals is te zien in **figuur 21** zijn er bepaalde onderdelen in het klassendiagram die geen attributen en/of functies hebben. Deze zijn voor de volledigheid toegevoegd om een beeld te geven van hoe de software is opgebouwd. De relaties tussen deze *componenten* zijn gelimiteerd aan de hand van wat noodzakelijk is.

Er worden in totaal twee *containers* opgebouwd. Eén voor de geplande sectie en één voor de ongeplande. Deze hebben dezelfde waardes nodig. Het grootste verschil is het menu dat wordt getoond boven de kalender en natuurlijk welke orders er worden getoond.

Over het component Joker is nog niets verteld. Dit component is bedoeld voor het calculeren van de lengte voor een order. Het programma kan hierdoor zien dat de order eronder moet worden geplaatst.



Figuur 21: *Klassendiagram van planbord*

ADAPTION-13152: Resize elements in the planboard

Vanwege de componenten die zijn beschreven in **6.4.1.1 Ontwerpen van sofware** moest er een keuze worden gemaakt welke onderdelen er konden worden vergroot en verkleind. De meest voor de hand liggende keuze is, om de *containers* deze mogelijkheid te geven. Wel is het noodzakelijk dat de grootte van de scrollbar wordt aangepast, zodat alle onderdelen in het planbord nog te gebruiken zijn. Er zal een component worden toegevoegd dat het vergroten en verkleinen mogelijk maakt. De ruimte die een container verliest krijgt de ander toegevoegd.

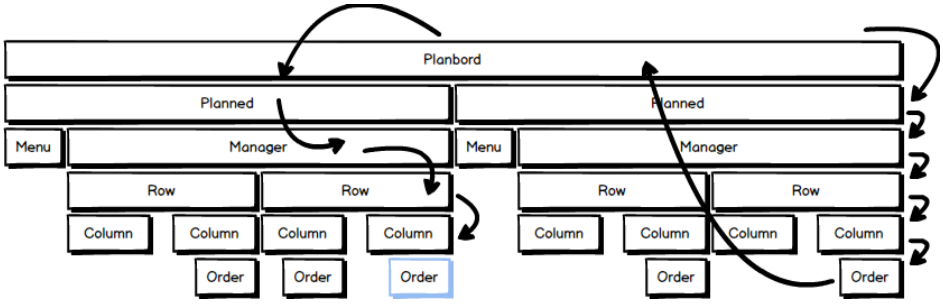
6.4.1.2. Realiseren van software

In dit hoofdstuk zal worden verduidelijkt wat de student tijdens de eerste sprint heeft uitgevoerd voor de desbetreffende beroepstaak. Per onderdeel zullen de beweegredenen worden uitgelegd en wat dit mogelijk voor gevolgen heeft gehad.

ADAPTION-13139: React Framework

Tijdens het realiseren van het planbord door middel van het React *framework* was er gekozen om de variabelen en functies te initialiseren in de componenten en waar nodig worden referenties doorgestuurd naar de zogenaamde kinderen van dat component. Dit zorgt ervoor dat elk component zijn eigen data vasthoudt en gemakkelijk weet wanneer het zijn eigen *content* moet aanpassen.

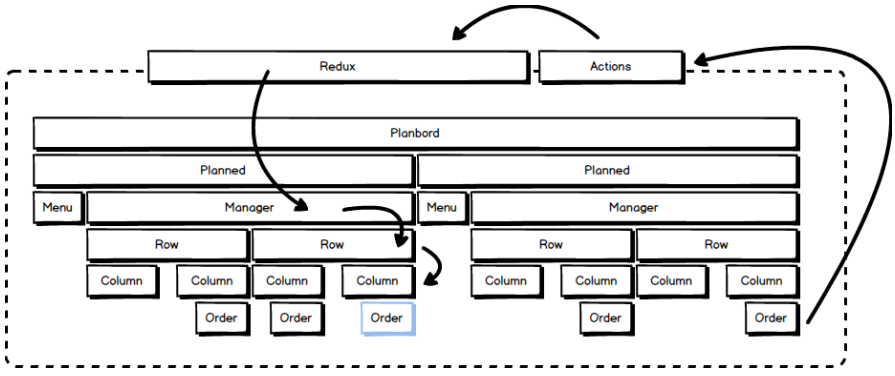
Deze beslissing zorgde later in het project voor problemen die opgelost moesten worden. Het werd steeds lastiger om de functies in de correcte componenten te plaatsen. Dit komt door de zogenaamde bomenstructuur die React aanhoudt. Dit kan problemen geven als veel data wordt verstuurd.



Figuur 22: *Visualisatie van de data overbrengen*

Telkens moet er data worden verstuurd naar het volgende component tot het wordt gebruikt. Als zowel de ongeplande manager en geplande manager iets moeten veranderen, moet dit bij de start worden geïnitialiseerd.

Op deze manier komen functies terecht in componenten waar ze eigenlijk niets mee te maken hebben. Ook de hoeveelheid onnodige data die wordt doorgegeven is een probleem. Redux biedt daar in tegen iets wat React in de basis mist. Een verzameling van data die door de gehele applicatie mogelijk is. Hierdoor kunnen functies worden aangeroepen in de componenten waar zij het meest logisch zijn.



Figuur 23: *Visualisatie van data overbrengen met Redux*

Zoals is te zien **figuur 23** vormt Redux zich als het ware over de gehele applicatie en hierdoor kan elk component een actie uitvoeren. Deze actie verandert de collectie die is opgeslagen in Redux. Redux stuurt deze verandering naar alle componenten die zich hiervoor hebben aangemeld. Hierdoor kunnen de functies worden geplaatst bij meest logische componenten.

Er zijn verschillende stappen die moeten worden genomen om dit werkend te krijgen. Het is noodzakelijk om een bestand te maken dat alle acties beschrijft. Een voorbeeld hiervan te zien in **figuur 24**.

```
export const ADD_ORDERS_TO_LIST = "ADD_ORDERS_TO_LIST"
export function addOrdersToList(ordersToAdd,isItPlanned){
   return {type: ADD_ORDERS_TO_LIST, ordersToAdd, isItPlanned}
}
```

Figuur 24: *Uitgeschreven action in Redux*

De *action* krijgt een naam die hetzelfde blijft in de gehele applicatie. Daarna wordt er een functie beschreven en variabelen worden teruggestuurd naar Redux. Hierna moet de functie worden uitgevoerd, dit is mogelijk door de code in **figuur 25**.

```
this.props.dispatch(addOrdersToList(plannedOrders,true));
```

Figuur 25: *Call to action.*

De *action* die wordt aangeduid, moet worden geïmporteerd naar het component waar die wordt aangeroepen. Dit verwijst naar het bestand waar het wordt uitgevoerd, ook wel de *reducer* genoemd. Als deze een actie binnen krijgt, wordt er door de mogelijke lijst van *actions* gezocht door middel van de naam van een *action*.

De componenten van het planbord kunnen de informatie die is opgeslagen in de *reducer* ophalen door middel van de code in **figuur 26**.

```
const mapStateToProps = state =>{ return { state } }
```

Figuur 26: *code om de state op te halen.*

De zogenaamde *state* van de *reducer* wordt opgehaald en teruggegeven aan het component dat deze code aanroept. In het huidige voorbeeld wordt de data echter niet gefilterd en de gehele *state* wordt teruggegeven. De data die is doorgegeven, kan worden gebruikt door middel van de *props* die elk component in React kan aanroepen.

ADAPTION-13138: Allow Rescheduling of order on another day

Bij het prototype was het niet mogelijk om een order op een andere datum te plaatsen. Dit is opgelost tijdens deze sprint. Om deze functionaliteit te bieden was het noodzakelijk om een order op een specifieke manier op te bouwen.

Als er orders binnenkomen, hebben deze altijd een start- en einddatum. Als een order wordt verplaatst of ingepland, wordt deze informatie gebruikt om een order correct weer te geven. Als een datum moet worden kunnen aangepast is het niet mogelijk om de bestaande start- en einddatum te gebruiken. Er moet juist rekening worden gehouden met hoe lang de order is.

Er zijn verschillende scenario's voor de orders die allemaal anders moeten worden opgelost. Ten eerste moet er worden gekeken welk scenario een order nodig heeft. De oplossing hiervoor zal ik kort beschrijven door middel van **figuur 27**.

```
let sRegion = moment(selectedDate,format).startOf('isoWeek').format(format);
let eRegion = moment(selectedDate,format).endOf('isoWeek').format(format);

let dStart = moment(oStartDate,format).diff(moment(sRegion,format),'days');
let dEnd = moment(eRegion,format).diff(moment(oEnd,format),'days');

if(dStart >= 0 && dStart < 7){startInside=true;}else{startOutside=true;}
if(dEnd >= 0 && dEnd < 7){endInside=true;}else{endOutside=true;}
if(dStart <= 0 && dEnd >=7){insideRange=false;}
```

Figuur 27: *code om te achterhalen wat voor soort order het is.*

In dit bovenstaande figuur worden verschillende variabelen geïnitialiseerd. Ten eerste wordt door middel van MomentJS de datums aangemaakt die evenredig zijn aan de start en eind van de huidige week die wordt getoond op het planbord.

Hierna wordt het verschil tussen de startdatum van de order en onze regio uitgerekend. Hetzelfde wordt uitgevoerd voor het einde. Deze data worden teruggestuurd als een aantal dagen.

Er wordt door verschillende *booleans*[1] vast te leggen uitgezocht of een order binnen de selectie valt. Door deze *booleans* te gebruiken zijn er vijf verschillende scenarios:

- De order start en eindigt binnen onze selectie;
- De order start wel maar eindigt niet binnen onze selectie;
- De order start niet binnen onze selectie maar eindigt er wel in;
- De order start en eindigt buiten de selectie maar komt wel door de selectie heen;
- De order start en eindigt buiten onze selectie.

Elke order krijgt een scenario toebedeeld en aan de hand daarvan worden de benodigde waardes doorgegeven zodat het correct kan worden opgebouwd.

ADAPTION-13139: React Framework (Contextmenu)

Het onderdeel contextmenu vereist enige uitleg. Normaliter, als er een rechtermuisklik wordt gedaan, zal er in de meeste gevallen een menu verschijnen met verschillende functionaliteiten. Adaption heeft binnen hun eigen applicatie menu's gemaakt met verschillende functionaliteiten.

Een groot gedeelte van het menu staat al binnen de Apex-omgeving en dit moet worden opgehaald en getoond worden. Dit wordt gedaan door middel van jQuery. Als er op een order wordt geklikt, worden de opties voor die order opgehaald. Als er op een optie wordt geklikt binnen het menu, wordt er een regel code meegegeven. Deze code wordt dan uitgevoerd binnen de apex omgeving.

[1] Een waarde die zowel **true** of **false** kan teruggeven.

6.4.1.3. Resultaat

In dit onderdeel wordt het resultaat van de sprint kort besproken. Enkele unieke aspecten komen nog kort aan bod, alsmede de gevolgen hiervan voor de rest van het project.

Userstories van de eerste sprint:

- ~~*ADAPTION-13139: Planbord ombouwen naar React Framework;*~~
- *ADAPTION-13146: Post-condition (planboard) checks;*
- *ADAPTION-13147: Recyclerview for orders;*
- *ADAPTION-13135: Filters for the planned orders;*
- ~~*ADAPTION-13138: Allow Rescheduling of order on another day;*~~
- ~~*ADAPTION-13152: Resize elements on planbord.*~~

In totaal zijn er drie verschillende *userstories* afgerond. Het planbord is opnieuw opgebouwd met React. Het planbord wordt in eerste instantie opgebouwd door middel van de huidige dag. Hierdoor is het mogelijk om de desbetreffende week te tonen en de orders die hierop van toepassing zijn, op te halen met de *websocket.*

Het is mogelijk om van week of jaar te wisselen. Ook het veranderen van *resourcetype* is mogelijk. De orders die worden aangemaakt hebben allemaal de mogelijkheid om gesleept te worden naar een andere *resource,* of mogelijk naar ongepland. Als dit wordt gedaan verschijnt er een *popup* met informatie over de order.

Het *contextmenu* kan worden getoond door middel van een rechtermuisklik. De benodigde data worden uit de Apex-omgeving gehaald en acties worden getoond. Als een optie wordt gekozen zal dit worden teruggestuurd naar de Apex-omgeving die de keuze weer gebruikt.

Het is mogelijk om de twee containers aan te passen zodat er meer ruimte overblijft voor de andere container.

Het is noodzakelijk om te melden dat ADAPTION-13147 niet kan worden uitgewerkt, omdat wordt gewerkt met een *websocket* die alle orders in een keer doorstuurt naar het planbord. Dit maakt het niet mogelijk om een gedeelte van de orders in te laden en later, als de gebruiker door de applicatie heen scrolt, meer binnen te halen. Wel is het goed om te melden dat de applicatie snel werkt zodra de data binnen is. Het initieel laden duurt het langste.

Er zijn helaas wel enkele gevolgen geweest in verband met de volgende sprint en het project in zijn compleetheid. De hoeveelheid gewenste functionaliteiten moest worden bijgesteld.

Dit is gebeurd tijdens de tweede sprint vanwege de werkwijze van Scrum. Ook is het niet gelukt om testen te schrijven omdat bepaalde *userstories* langer duurde dan verwacht. In overleg met de bedrijfsbegeleider is afgesproken dat dit aan het einde van de tweede sprint werd ingehaald.

6.4.1.4. Reflectie

Ik ben langer bezig geweest met het implementeren van het planbord met behulp van het *framework* React. Dit heeft echter verschillende redenen gehad. Ik heb het werk voor de implementatie onderschat. Daarnaast waren er enkele zaken waardoor ik verschillende onderdelen van de realisatie opnieuw moest uitvoeren of herschrijven.

Er is tijdens de realisatie voor gekozen om met Redux te gaan werken, wat zeker zijn vruchten heeft afgeworpen. Dit was echter niet de intentie toen er werd gestart met de realisatie. De combinatie van het toepassen van React + Redux heeft de student veel moeite gekost, omdat dit allemaal nieuwe *frameworks* waren die de gebruikte taal van Javascript toch op een andere manier worden gebruikt. Wel is het belangrijk dat ik eerder meldt als dingen niet goed gaan.

Ik ben lang bezig geweest om een solide basis op te zetten die goed uitbreidbaar moet zijn. Om deze reden verwacht ik een gedeelte van de verloren tijd in te kunnen halen. De doelen moeten wel opnieuw worden geëvalueerd.

De keuzes om bepaalde *libraries* toe te voegen aan het project, waren volgens mij goed onderbouwd. De enige die beter had gekund was jQuery. Dit kwam echter omdat Adaption deze implementatie al gebruikt in de huidige applicatie.

Kijkende naar het resultaat en de functionaliteiten die de student werkend heeft gekregen, vindt de student zelf dat er best wat werk is verricht. De problemen die zijn ondervonden en de keren dat onderdelen opnieuw moesten worden geschreven, zijn nog steeds voorgevallen en hebben gevolgen voor het project in zijn geheel. Maar het was grotendeels een miscalculatie in zowel het benodigde werk en de kennis die ik moest hebben om deze functionaliteiten werkelijkheid te laten worden.

Bij de tweede sprint was het van belang dat ik sneller om hulp/advies vraag als er problemen zijn. Dit is aan het einde van de eerste sprint al meerdere malen gebeurd. Een goed voorbeeld is informatie vergaren hoe een order moet worden ingepland, maar ook de problemen die ik heb ondervonden met de *websockets*.

6.4.2. Tweede Sprint

Er zijn verschillende *user stories* die aan bod zijn gekomen in de tweede sprint:

- *ADAPTION-13338: Different Notations;*
- *ADAPTION-13241: Collapsible resources;*
- *ADAPTION-13140: Unplanned Filters;*
- *ADAPTION-13414: Order Status Color;*
- *ADAPTION-13145: Open planbord in new window.*

ADAPTION-13338: Different Notations

Het moest mogelijk worden om verschillende notaties te ondersteunen. Dit wordt gedaan door middel van profielen die in een ander onderdeel van de Apex-omgeving kunnen worden aangemaakt. Deze profielen worden ingeladen in het planbord aan de hand van een dropdown lijst. Het was hier ook noodzakelijk dat tijdsloten konden worden toegevoegd en dagen moesten ook kunnen worden overgeslagen.

ADAPTION-13241: Collapsible resources

Een *resource* kan veel verschillende orders bevatten waardoor deze veel ruimte inneemt. Als het mogelijk is om een *resource* in te kunnen klappen, wordt er veel ruimte vrijgemaakt. Een *resource* moet een andere *layout* krijgen terwijl dezelfde functionaliteiten behouden blijven. Ook moet het nog mogelijk zijn om orders in te kunnen plannen.

ADAPTION-13140: Unplanned Filters

Er moeten verschillende filters worden toegevoegd, zodat het vinden van specifieke orders worden versimpeld. Degene die worden geïmplementeerd zijn:

- *Start en eindlocatie filter;*
- *Zoeken op dossiernummer of ordernummer;*
- *Orders moet of zonder gevaarlijke stoffen.*

ADAPTION-13414: Order Status Color

Orders moeten andere kleuren krijgen aan de hand van de status. Binnen de Apex-omgeving kan deze kleur worden gewijzigd per klant. De kleuren moeten worden opgehaald en aan de hand van de status correct worden doorgegeven aan de orders.

ADAPTION-13145: Open planboard in new window

Het moet mogelijk zijn om het planbord in een apart venster te plaatsen, zodat de gebruiker het planbord mogelijk op een tweede scherm kan zetten, terwijl de Apex-omgeving wordt gebruikt in de browser. De verschillende functionaliteiten van het planbord moeten blijven werken en als het *contextmenu* wordt gebruikt, moet de geselecteerde optie mogelijk worden geopend in de Apex-omgeving.

6.4.2.1. Ontwerpen van software

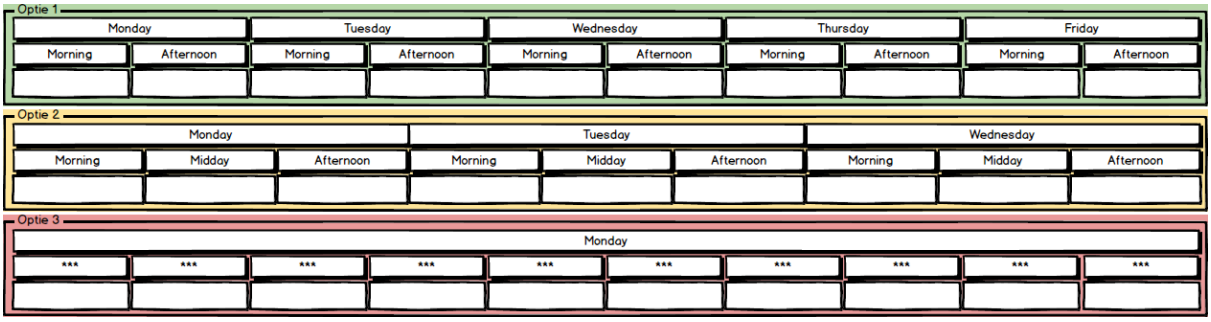
In dit onderdeel zijn de onderdelen die te maken hebben met het ontwerpen van de software aan bod komen. per *user story* zal er worden beargumenteerd wat er is gedaan en waarom.

ADAPTION-13338: Different Notations

De data die benodigd is wordt verzameld in zogenaamde profielen en deze hebben bepaalde waardes die worden gebruikt. Een profiel bestaat uit een notatie die is vastgelegd in de Apex-omgeving, de verschillende *timeslots* en de mogelijkheid om nog een aantal dagen in het verleden of toekomst te kijken.

Als eerste was het noodzakelijk dat het planbord *timeslots* zou gaan ondersteunen. Er moet een manier worden gevonden om deze weer te geven en het genereren van de lengte moet anders worden opgelost. De *timeslots* hoeven niet op elkaar aan te sluiten. Het kan dus zijn dat er een gat is van twee uur en de logica moet dit goed oplossen.

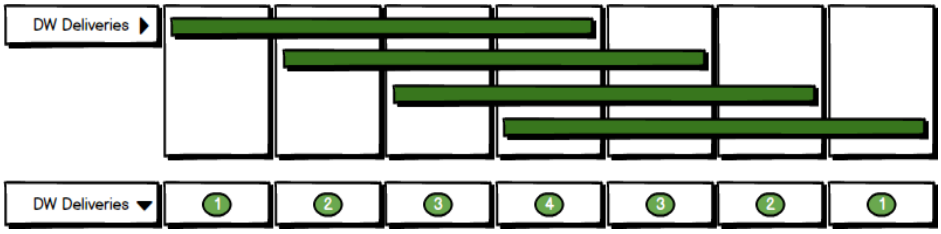
Samen met Adaption is bepaald, dat er geen harde eisen zijn in het planbord zelf. Dit wordt afgevangen tijdens het creëren van een profiel. Adaption is geadviseerd om niet méér kolommen te tonen dan 14. Het is mogelijk, maar de gebruiker moet een groot genoeg beeldscherm hebben. Ook twee identieke *time slots* aanmaken wordt niet afgevangen in het planbord. Maar er kan vanuit worden gegaan dat dit visuele problemen zal opleveren.



Figuur 28: *Verschillende mogelijkheden per timeslot.*

ADAPTION-13241: Collapsible resources

Voor het inklappen van een *resource* moesten er verschillende keuzes worden gemaakt. Hoe wordt er getoond hoeveel orders er in een *resource* zitten en waar wordt deze mogelijkheid aangeboden? Uiteindelijk is ervoor gekozen om de optie achter de naam van een *resource* te plaatsen. Deze wisselt dan tussen twee *layouts* waarbij één de orders laat zien in een tijdlijn en de andere laat het zien als een nummer binnen een specifieke dat. Er is gekozen om een lijst van de *resources* bij te houden die momenteel zijn geminimaliseerd, zodat als er van *layout* wordt gewisseld de *settings* bewaard blijven. Een voorbeeld is te zien in **figuur 29**.



Figuur 29: *Voorbeeld van ingeplakte resource.*

6.4.2.2. Realiseren van software

In dit hoofdstuk zal worden verduidelijkt wat de student tijdens de eerste sprint heeft uitgevoerd voor de desbetreffende beroepstaak. Per onderdeel worden de beweegredenen uitgelegd en wat dit mogelijk voor gevolgen heeft gehad.

ADAPTION-13338: Different Notations

Voor het realiseren van de verschillende notaties was het noodzakelijk om de verschillende *cases* uit te kunnen lezen. Er is voor gekozen om een stuk code te schrijven die deze data van een profiel binnen krijgt. Deze stuurt de *range* die moet worden getoond, terug zodat dit kan worden gebruikt tijdens het opbouwen van het planbord.

Alhoewel Adaption alleen zater en zondag wilde kunnen overslaan heeft de adapter de mogelijkheid opengelaten om ook andere dagen over te slaan. Een voorbeeld hiervan is **figuur 30**. Mocht er geen profiel worden meegegeven of niet kunnen worden opgehaald, zal er een *default* worden getoond van een week zonder *timeslots*.

```
switch(notation){
        case "1-day-week":
            startSel = sDate.clone().subtract(sMinus,'day').startOf('day')
            return calculator(startSel,sMinus,sPlus,1,Days);
        case "5-day-week":
            allowedDays[6].allowed = false;
            allowedDays[0].allowed = false;
            let day = sDate.clone().startOf('ISOweek').subtract(sMinus,'day')
            startSel = find(day,Days);
            return calculator(startSel,sMinus,sPlus,5,Days);
        case "6-day-week":
            ...
        case "7-day-week":
            ...
        default:
            let day = sDate.clone().startOf('ISOweek').subtract(sMinus,'day')
            startSel = find(day,Days);
            return calculator(startSel,sMinus,sPlus,7,allowedDays);
}
```

Figuur 30: *Selectie van profiel*

Er wordt ook rekening gehouden met de *time slots* doordat we de eerste dag van de week in de *range* laten starten om 00:00 precies en de *range* eindigt om 23:59. Hierna wordt er een calculatie gedaan met de data die nodig is. De code in **figuur 30** zal een lijst met dagen teruggeven die moeten worden getoond in het planbord.

Als een gebruiker wisselt van *resourcetype* waar het desbetreffende profiel niet aanwezig is, zal er gelijk worden gewisseld naar het eerste profiel dat wordt opgehaald. Als een gebruiker een profiel heeft dat alleen voor hun bedoeld is, zal deze als eerste worden ingeladen en anders wordt er gecheckt of het *resourcetype* dat is geselecteerd een standaard profiel heeft.

Dit is echter niet genoeg om de orders correct weer te geven in het planbord. De calculatie die moet worden uitgevoerd om de lengte terug te krijgen, moet worden uitgebreid.

```
if(item.format(format) === dateNow.clone().add(i,'day').format(format)){
   if(order.ammountOfDays === 1){
       test = findTimeOver1Day(startTime,endTime,timeslots,order.name)
   }else if(i === 0){
       test += findCorrectTime(startTime,timeslots,true,false,order.name);
   }else if(i+1 === ammount && tempEndDate === orderEnd.format(format)){
       test += findCorrectTime(endTime,timeslots,false,true,order.name);
   }else{
       test += timeslots.length;
   }
}
```

Figuur 31: *Correcte tijd vinden van een order en lengte genereren.*

Er zijn verschillende onderdelen toegevoegd om de lengte van een order te genereren. Dit is te zien in **figuur 31.** Het is nu mogelijk dat een dag is opgesplitst. Om deze reden is het noodzakelijk om te achterhalen welke situatie moet worden gebruikt. Deze code kijkt eerst naar het aantal gegenereerde dagen. De *time slots* zijn alleen relevant voor het begin van de order en het einde. Bij zowel het begin als einde moet er worden gecontroleerd of de order in een bepaalde *time slot* zit. Alle dagen tussen het begin en einde hoeft niet te worden berekend. Als het om een order gaat die maar een dag lang is moet er een aparte functie worden uitgevoerd.

```
if(
moment(startTime,"HH:mm").isSameOrAfter(slot.from) &&
moment(endTime,"HH:mm").isSameOrBefore(slot.to)) {
   OneTimeslot = true;
   sizeOfOrder = 1;
}else if(
!OneTimeslot && !startDefined &&
moment(startTime,"HH:mm").isSameOrAfter(slot.from)  &&
moment(startTime,"HH:mm").isSameOrBefore(slot.to)) {
   sizeOfOrder++;
   startDefined = true;
}
```

Figuur 32: *Calculeer of een order binnen een time slot valt.*

In **figuur 32** wordt de *timeslot* vergeleken met de kolom die wij op dat moment bezitten. Als de *timeslot* zich hiertussen bevindt, wordt de grootte van de order aangepast. Mocht dit niet het geval zijn dan wordt er gekeken of het huidige *timeslot* tussen het einde van de huidige kolom zit en het begin van de volgende. Als dit het geval is zal de lengte groter worden.

Dit is een van de *checks* die wordt uitgevoerd tijdens het genereren van de lengte voor een order.

ADAPTION-13145: Open planboard in new window

De functionaliteit om een apart venster te openen waar het planbord kan worden gebruikt was vrij eenvoudig te implementeren. Er was echter een onderdeel wat moest worden uitgebreid en dit was het *contextmenu*. Dit komt omdat dit normaliter gebruik maakte van de zogenaamde *parent* wat binnen de Apex-omgeving zit. In het nieuwe venster hebben wij deze echter niet tot onze beschikking. Er moest worden nagedacht hoe dit kon worden gecombineerd.

Data meesturen bleek lastig omdat dit normaliter alleen mogelijk is als zowel de *parent* en *child* van dezelfde locatie afkomstig zijn. Dit is helaas voor het planbord niet het geval. Er is uiteindelijk wel een manier gevonden om te achterhalen of het planbord zich in een venster bevindt. Dit is te zien in **figuur 33.**

```
if(window.top.location === window.parent.location){ … }
```

Figuur 33: *Check die wordt uitgevoerd om te achterhalen of planbord in venster staat.*

Door het *command "window.top"* en *"window.parent.location"* te gebruiken, kan worden achterhaald of het planbord zich in een nieuw venster bevindt. Als het planbord in een nieuw venster staat zijn deze identiek. Echter in de Apex-omgeving niet. Hierna wordt op verschillende onderdelen in de applicatie deze *check* uitgevoerd, zodat bepaalde onderdelen worden aangepast. Een voorbeeld hiervan is de mogelijkheid om een venster te openen wordt verborgen.

```
if(window.top.location === window.parent.location){
   option = option.replace("window.parent.getTopParent()",
"window.parent.opener.parent.getTopParent()");
}
```

Figuur 34: *Contextmenu werkend maken in een nieuw venster.*

In **figuur 34** wordt het *contextmenu* werkend gemaakt. Dit komt doordat als een venster wordt geopend, krijgt deze een referentie mee met zijn *parent:* Degene die het heeft aangemaakt. Door dit te gebruiken is het mogelijk om het *contextmenu* weer te laten werken. Het stuk code waar *"window.parent.opener"* staat zorgt ervoor dat we in het originele venster terecht komen. Hier wordt nogmaals gekeken naar de *parent* die daar een functie aanroept om de gekozen optie in het *contextmenu* uit te voeren.

6.4.2.3. Resultaat

In dit onderdeel wordt het resultaat van de sprint kort besproken. Enkele unieke aspecten komen nog kort aan bod, alsmede de gevolgen hiervan voor de rest van het project.

Userstories van de tweede sprint:

- ~~*ADAPTION-13338: Different Notations;*~~
- ~~*ADAPTION-13241: Collapsible resources;*~~
- ~~*ADAPTION-13140: Unplanned Filters;*~~
- ~~*ADAPTION-13414: Order Status Color;*~~
- ~~*ADAPTION-13145: Open planbord in new window.*~~

Alle *userstories* zijn deze sprint afgerond. Het is nu mogelijk om te wisselen tussen verschillende profielen die worden opgehaald door middel van de *websocket*. AAn de hand van het profiel wordt het planbord opnieuw opgebouwd en de orders krijgen hun correcte lengte aan de hand van de weergave van het planbord.

Time Slots kunnen nu worden toegevoegd aan het planbord en zelfs gaten tussen de *timeslots* worden ondersteund. Een order wordt niet getoond als deze in een gat zit.

Het planbord met ongeplande orders is opnieuw opgebouwd en heeft dezelfde weergave als de geplande orders. Het is nu duidelijker waar een ongeplande order wordt geplaatst. Daarnaast is het mogelijk om deze orders te filteren, zodat alleen relevante resultaten worden getoond. De orders hebben ook andere kleuren aan de hand van de status die binnenkomt van de *websocket*.

Als laatste is het mogelijk om het planbord in een nieuw venster te openen. De functionaliteiten van het *contextmenu* sturen de *requests* naar het goed venster. Als een *context menu* optie het toelaat, is het mogelijk om de optie te openen in het geselecteerde venster.

6.4.2.4. Reflectie

Deze sprint ging stukken beter dan de vorige. Ten eerste zijn alle functionaliteiten gereed en er waren geen verrassingen qua *development*. Het onderdeel van de *time slots* en verschillende notaties heeft meer tijd gekost dan in eerste instantie was ingecalculeerd. Dit had te maken met onderdelen die ik nodig had vanuit Adaption. Dit duurde iets langer dan eigenlijk de bedoeling was. Om deze reden ben ik naar mijn begeleider gegaan om hierover informatie te verzamelen en doorgegaan met een andere *user story*.

De orders een correcte lengte geven was lastiger dan in eerste instantie werd gedacht. De code hiervoor is meerdere keren veranderd omdat specifieke *usecases* nog fout gingen.

Zelf ben ik tevreden over hoe deze sprint is verlopen en de resultaten die zijn geboekt. Helaas is dit het einde van het realisatietraject binnen mijn afstudeeropdracht en moet er nu worden gefocust op het testen van de applicatie.

6.5. Testcases

In dit onderdeel komt het proces voor het maken en schrijven van de testcases aan bod. Ten eerste is er geïnventariseerd welke test cases uitgevoerd konden worden. Hierna zijn zij beschreven met de werkwijze die Adaption aanhoudt. Er zal een testcase worden genomen als voorbeeld, zodat kan worden uitgelegd wat bedoeld wordt bij de verschillende stappen. Alle testcases die zijn gerealiseerd zijn te vinden in **11.2. Bijlage B: Test Cases**. Hierna zijn de testcases geschreven in Java, zodat ze uitgevoerd konden worden in Selenium. De stappen voor het ontwikkelen van de testcases zullen aan bod komen, evenals de regels die Adaption hanteert.

Als laatste wordt het resultaat van de testen nog besproken.

6.5.1.Inventarisatie

Als eerste moesten de mogelijke testen worden geïnventariseerd. Ook moest de data die noodzakelijk was worden gegenereerd. In totaal zijn er 27 *cases* gevonden die moesten worden getest. Deze *cases* kunnen worden gecategoriseerd op verschillende punten, zodat deze kunnen worden gecombineerd in een testcase. De mogelijkheden zijn gerepresenteerd in **figuur 35**.

Layout	Filters	Orders	Contextmenu	Window
Switch resource	Dossier search	Planned/Unplanned	Contextmenu open	Window open
Switch layout	Order search	Unplanned/Planned	Edit order	Contextmenu
Switch Year	Adr search	Planned/Planned	Customs Declaration	Edit order
Switch Month	Start loc search	Cancel movement		Customs Declaration
Previous day	End loc search			
Next Day	Combination search			

Figuur 35: *cases gecategoriseerd per onderdeel*

Daarnaast is het van belang dat we beschikken over verschillende soorten data, zodat deze testen kunnen worden uitgevoerd. Na het inventariseren zijn dit de onderdelen die nodig zijn:

- *2 resource types;*
- *2 profiles;*
- *2 resources;*
- *2 geplande orders;*
- *5 ongeplande orders.*

Ook is het nodig om de diverse data die de onderdelen zo verschillend mogelijk te maken. Orders moeten starten op andere tijden en datums. De specificaties voor de verschillende onderdelen zijn te vinden in **11.1. Bijlage A: Test Data**

6.5.2. Beschrijving testcase

Een gedocumenteerde testcase heeft verschillende onderdelen. In dit hoofdstuk zal elk onderdeel kort worden beschreven en waarom dit wordt gedaan.

Bij **figuur 36** volgt een beschrijving van de volledige test case. Het moet voor de lezer duidelijk zijn wat er wordt getest door deze *summary*.

Summary	Test different settings to change the planbords view.

Figuur 36: *cases gecategoriseerd per onderdeel*

Bij **figuur 37** wordt de *user* neergezet. Dit is binnen Adaption in de meeste gevallen de SELENIUM TEST. Dit is om te specificeren wie en waarde testcase wordt uitgevoerd.

Users	SELENIUM TEST

Figuur 37: *cases gecategoriseerd per onderdeel*

Bij **figuur 38** worden de precondities beschreven deze moeten worden uitgevoerd voordat de test kan worden gestart. Adaption heeft ervoor gekozen om deze in de meeste gevallen leeg te laten, tenzij dit niet mogelijk is.

Preconditions	

Figuur 38: *cases gecategoriseerd per onderdeel*

Bij **figuur 39** worden de stappen neergezet die moeten worden uitgevoerd. Deze worden zo kort mogelijk beschreven en er wordt ook gespecificeerd hoe er kan worden achterhaald of een test geslaagd is.

Steps

#	Page Actions	Check
1	Select the value **Trucks**	Check of the resource that are added are **Grave Digger** and **Jurassic Attack**
2	Select the last dropdown in top menu	
3	Select the value **Default Layout**	Selection should now be 7 days starting with **Friday** and ending with **Monday**

Figuur 39: *cases gecategoriseerd per onderdeel*

Bij **figuur 40** wordt er verteld wat het volledige resultaat van de testcase is als alle stappen correct zijn uitgevoerd. Dit geeft de schrijver een beeld of zij het goed hebben gedaan.

Postconditions	Different fields have been tested to ensure that it changed the content of the planboard

Figuur 40: *cases gecategoriseerd per onderdeel*

Bij **figuur 41** wordt er beschreven wat er moet worden gedaan zodra de test is afgelopen. Als een test data wijzigt wordt dit achteraf teruggezet zodat andere testen die hierna worden uitgevoerd hier niet op kunnen falen.

Cleanup	Reload the planboard.

Figuur 41: cases gecategoriseerd per onderdeel

Alle stappen die moeten worden uitgevoerd bij een testcase zijn nu beschreven. De volledige test cases zijn te vinden in **11.2. Bijlage B: Test Cases** Toen alle testcases waren beschreven is er gestart met het ontwikkelen van deze *cases*.

6.5.3. Ontwikkeling testcase

Het ontwikkelen van de testcases is uitgevoerd door middel van Netbeans. Ten eerste was het van belang om een zogenaamde *page* aan te maken die de logica voor de testen die worden geschreven vasthoudt. Als een bepaalde functie wordt geschreven, zal dit in de meeste gevallen gebeuren in de *page*. Hierna is een testcase opgezet. Een klein voorbeeld hiervan is te zien in **figuur 42**.

```
planboardPage = new Planboard_150_1450(driver).openPage();
planboardPage.goToIframe("planboardFrame");

softAssert.assertTrue(
   planboardPage.findOrders("31624", "unplannedItems"),
   "Does the order with 31624 exist?"
);
```

Figuur 42: Opzet testcase met 1 stap.

Het is als eerste van belang om de desbetreffende *page* op te slaan in de test. Door dit te doen wordt ook de pagina binnen de Apex-omgeving geopend. Omdat het planbord binnen een *iframe* staat, is het noodzakelijk om de focus hierop te zetten zodat de verschillende *checks* kunnen worden uitgevoerd. Een voorbeeld hiervan is het zoeken van een order. Hiervoor is een functie geschreven, die de naam van de order nodig heeft en waar er naar de order moet worden gezocht. Deze functie is uitgeschreven in **figuur 43**.

```
public boolean findOrders(String inputValue, String search){
   WebDriverWait wait = new WebDriverWait(driver, 10);
   boolean value = wait.until(new ExpectedCondition<Boolean>(){
      @Override
      public Boolean apply(WebDriver driver) {
         WebElement order =
driver.findElement(By.xpath(".//div[contains(@class,'"+ search +
"')]//div[@class=\"order hasmenu\"]/div[contains(text(),"+ inputValue
+")]"));
         return order != null;
      }
   });
   return value;
}
```

Figuur 43: Functie om order te zoeken.

In **figuur 43** wordt er een *"wait"* toegevoegd. Dit is zodat de functie een tijd deze actie blijft proberen voordat het resultaat wordt teruggegeven. Mocht dit niet binnen de opgegeven tijd gebeuren, is het resultaat alsnog negatief. Binnen de functie wordt er gezocht naar een element door middel van Xpath. Hierin worden ook twee variabelen gebruikt die zijn meegegeven. *("XPath", 2019)*

Er wordt gezocht naar het element en het resultaat wordt teruggeven. Hierna is dus te zien of de test op dit punt al gefaald heeft.

Er zijn verschillende functies beschreven die deze mogelijkheid geven. Er is tijdens het realiseren grotendeels geprobeerd om dit met kleine functies op te lossen die kunnen worden hergebruikt. Een voorbeeld hiervan was de functie in **figuur 43** die zowel het zoeken mogelijk maakt bij het geplande en niet geplande onderdeel. Een andere functie die dit toelaat is te zien in **figuur 44.** Deze functie zoekt naar een *dropdown* menu binnen het planbord.

```
public void selectDropdown(String inputValue, String search) {
   WebElement select = driver.findElement(By.xpath(".//select[@id='"+ search
+ "']"));
   select.click();
   WebElement option = driver.findElement(By.xpath(".//select[@id='"+ search
+ "']/option[text()='"+ inputValue + "']"));
   option.click();
}
```

Figuur 44: *Functie om order te zoeken.*

Met de code in **figuur 44** is het gemakkelijk om alle *dropdowns* aan te roepen. Dit kan door middel van de variabele *"inputValue",* zelfs als er nieuwe worden toegevoegd.

Daarnaast zijn er functionaliteiten beschreven met betrekking tot het verplaatsen van orders. En het wisselen tussen de verschillende vensters die zijn geopend.

Als een testcase is afgerond, is het van belang om de waardes die zijn veranderd weer terug te zetten naar het origineel.

Als een test is afgerond wordt deze op toegevoegd aan de volgende test ronde. Dit gebeurd tweemaal per dag, Eenmaal in de middag, zodat mocht een test nog falen deze direct kan worden opgepakt en andermaal in de avond zodat er s ochtends kan worden onderzocht of deze gewerkt heeft.

6.5.4. Resultaat

Er zijn in totaal vijf verschillende testcases geschreven. In totaal zijn er 27 verschillende onderdelen getest.

Voordat de testen gerealiseerd werden zijn ze als eerste correct beschreven op de manier die Adaption aanhoudt en besproken voordat deze zijn gerealiseerd.

Tijdens het ontwikkelen van de testcases zijn er enkele uitdagingen naar voren gekomen. Als er werd gewisseld tussen de verschillende *tabs,* was het soms niet mogelijk om terug te gaan naar het planbord. Dit is opgelost door de pagina opnieuw op te roepen. Dit vernieuwt de pagina niet, dus de geselecteerde instellingen blijven bewaard.

Het verplaatsen van de orders had ook een uitdaging omdat de functionaliteit de Adaption gebruikt om elementen te verslepen niet werkte op de orders binnen het planbord. Uiteindelijk is het gelukt om deze functionaliteit werkend te krijgen.

Dit probleem zat in de manier van het verslepen. Er wordt binnen het planbord gebruik gemaakt van HTML5 Drag & Drop en Selenium ondersteund dit niet. Het was dus noodzakelijk om een andere manier te vinden. Er wordt nu tijdens de testcase via javascript een Drag & Drop uitgevoerd.

De testcases worden al automatisch uitgevoerd en er zijn al enkele wijzigingen geweest als een test niet goed uitgevoerd was. Grotendeels had dit te maken met een timer die langer moest worden gezet of op andere elementen *checken* omdat ze al werden gebruikt.

7. Procesevaluatie

Er zijn tijdens het project verschillende processen uitgevoerd. Deze processen zullen worden geëvalueerd. Per onderdeel zal er worden beschreven hoe ik vond dat ik heb het uitgevoerd en wat er mogelijk anders had gekund.

7.1. Plan van Aanpak

Het plan van aanpak was lastig te ontwikkelen omdat het voor mij niet gelijk duidelijk was welke stappen ik allemaal moest gaan uitvoeren. Daarnaast waren de requirements ook minimaal te noemen. Ik denk niet dat ik dit onderdeel veel beter had kunnen uitvoeren met de informatie die ik toen tot mijn beschikking had. Wel had ik meer kunnen vragen. Dit had het maken van het plan van aanpak mogelijk kunnen bespoedigen.

7.2. Planning

Het was mij in het begin niet duidelijk welke werkzaamheden allemaal nodig waren. Om deze reden is de planning meerdere keren veranderd. Er zijn zowel onderdelen weggehaald en toegevoegd. Vanwege de problemen bij de eerste sprint zijn doelen opgeschoven. Verder in het project werd het steeds duidelijker wat er moest gebeuren en heb ik mezelf herpakt. Het was helaas niet mogelijk om de verloren tijd volledig in te halen.

7.3. Ontwikkelmethode

Scrum was een goede keuze voor dit project. De sprints hadden achteraf gezien wel korter gekund maar omdat Adaption wou dat ik mee zou draaien in hun proces ben ik er toendertijd in meegegaan. Zeker in het begin waren de *daily stand ups* niet even bruikzaam omdat de werkzaamheden die ik moest uitvoeren weinig te doen had met de rest van de werknemers. Later zijn ze stukken nuttiger geworden zodat ik snel iets kon vragen of een afspraak in kon plannen.

7.4. Onderzoek

Zelf ben ik blij wat er is gebeurd tijdens het onderzoek. Achteraf waren er wel enkele onderdelen die ik anders had willen doen. De manier om scores te berekenen had beter gekund dan een simpele optelsom te maken. Ook had ik liever meer functionaliteiten willen testen. Dit is toendertijd niet gebeurd omdat de *requirements* op dat punt nog onduidelijk waren. Het uitgebreide onderzoek heb ik naar mijn eigen zeggen beter gedaan dan de eerste maar liever had ik gehad dat dit niet nodig was geweest. Liever had ik deze tijd gebruikt voor het realiseren van de applicatie maar het was een belangrijke stap binnen het project. Ik ben er niet zeker van of er anders voor React was gekozen.

7.5. Realisatie

Tijdens de realisatie heb ik veel stress ervaren. Dit kwam grotendeels door de eerste fase toen het planbord moest worden gerealiseerd in het *framework* React. De hoeveelheid nieuwe indrukken en de werkwijze bleek erg lastig om mijzelf aan te leren. Toen ik op een gegeven moment had besloten om ook Redux toe te voegen aan het project heb ik mezelf echt voor gek verklaard. Het werk dat het had gekost om tot dit punt te komen moest opnieuw worden gedaan.

Achteraf ben ik blij dat ik het heb gedaan. De manier van werken zal vast goed uitpakken voor kleinere applicaties waar minder *dependencies* onderling zijn. Voor het planbord had het waarschijnlijk uitgelopen tot een complete ramp. Aan het einde van de eerste sprint heb ik mijzelf ook zeker herpakt en zijn er nog enkele functies afgerond. Helaas niet genoeg om alle *user stories* te voltooien.

De tweede sprint ging al vele malen beter. Wel was het jammer dat een redelijke tijd gewerkt is aan de verschillende notities die mogelijk moeten zijn. Een gedeelte van de code die was geschreven tijdens de eerste sprint, moest hiervoor worden aangepast. Toen het mij later duidelijk was dat sommige dagen ook moesten worden overgeslagen, heeft dit nogmaals een verandering moeten ondergaan. Dit heeft echter wel geleid tot het belangrijkste stuk code in de applicatie.

7.6. Testcases

De testcases gingen beter dan ik in eerste instantie had gedacht. Er was een duidelijke *flow* aanwezig die moest worden doorgelopen. Wel zijn er enkele korte gesprekken geweest omdat Adaption vond dat de testcases die ik allemaal apart had verwerkt, te veel waren en wel konden worden gecombineerd. In essentie ben ik het daar ook mee eens. Alleen had ik de testen zo kleinschalig uitgevoerd omdat ik dit zo geleerd had tijdens mijn opleiding. Na dit gesprek heb ik de verschillende kleine testen in bepaalde categorieën ondergebracht zodat ze meer omvang hadden.

Ook was mij verteld om de *checks* die moesten worden uitgevoerd concreter te maken omdat bepaalde *checks* die ik had bedacht, niet gemakkelijk mogelijk waren in Selenium. Ik heb toen wat advies gevraagd en ben hiermee toen aan de slag gegaan. Dit was vrij vroeg in het gehele proces.

Tijdens het realiseren van de testcases werd mij wel duidelijk waarom dit werd gevraagd. Het is onnodig om in Selenium allemaal kleine testcases te maken die gemiddeld vijf regels code in beslag nemen. Dit betekende echter wel dat ik vrij duidelijk voor ogen had welke stappen ik moest doorlopen.

Ook heb ik tijdens het realiseren van code nog enkele ID's meegeven aan bepaalde elementen in het planbord die er in eerste instantie niet waren omdat dit het testen vele malen eenvoudiger maakte.

8. Productevaluatie

Tijdens mijn afstudeerperiode zijn er verschillende producten tot stand gekomen. Ten eerste is er een onderzoek uitgevoerd naar het beste *framework* voor een planbord met betrekking tot de wensen die Adaption daarvoor had. Dit onderzoek is later nog uitgebreid waarbij er nog drie *frameworks* zijn vergeleken. Ditmaal door middel van de *community* die er omheen zit. Daarna is er een applicatie gerealiseerd door middel van het *framework* React. Hier zijn tijdens de realisatie meerdere *plugins* gebruikt om de gewenste functionaliteiten werkelijkheid te maken zoals: Redux, Moment JS, React DND en jQuery. Als laatste zijn er nog testcases ontwikkeld door middel van de testtool Selenium die deze testen automatisch uitvoert om de werking van het planbord te waarborgen.

Ik ben zelf zeer tevreden over het eerste onderzoek wat ik heb aangeleverd. Dat was naar mijn idee goed weergegeven en de keuzes waren weloverwogen. Het tweede onderzoek was naar mijn idee iets minder bevredigend. Dit had grotendeels te maken met het feit dat ik deze niet had gepland. Ik wilde sneller klaar zijn zodat ik kon beginnen met het realiseren van de applicatie.

De realisatie was simpelweg lastig te noemen. Ik heb in het begin moeite gehad om de architectuur van React goed te gebruiken. Ook heb ik in het begin enkele keuzes gemaakt die ik later heb moeten terugdraaien. Een voorbeeld hiervan is het gebruik van Redux maar ook jQuery waren initieel niet door mij ingepland. Redux is achteraf wel mijn beste keuze geweest en heeft zeker tijdens de tweede sprint geholpen om gewenste functionaliteiten goed op te leveren.

Een probleem dat niet altijd mee heeft geholpen, vormde de user stories die vaak erg verschilden in het aantal werkzaamheden die nodig waren. Het realiseren van een planbord in React is velen malen meer werk dan het inklappen van een rij resources.

Echter ben ik wel tevreden met wat ik heb neergezet. Omdat ik zeker weet dat de onderdelen die er zijn correct werken en robuust genoeg zijn om nieuwe functionaliteiten toe te laten.

Enkele nieuwe functionaliteiten zijn gerealiseerd waardoor het voor de klanten van Adaption mogelijk is om het planbord efficiënter te gebruiken. Zoals het toevoegen van tijdnotaties of bepaalde dagen niet te tonen in de selectie. Ook het planbord volledig openen in een nieuw scherm terwijl het contextmenu blijft werken, zijn handige functionaliteiten.Net als de filters die zijn toegevoegd aan de ongeplande orders, die het gebruik van het planbord kunnen versnellen.

Het planbord is in vergelijking met het vorige prototype, vele malen sneller geworden en veel kleine verbeteringen zijn doorgevoerd om het gebruik te verbeteren.

9. Beroepstaken

9.1. A1: Analyseren probleemdomein & opstellen probleemdomein

Tijdens het onderzoek is het probleemdomein geanalyseerd en is er een probleemstelling opgesteld. Er zijn verschillende oplossingen onderzocht met betrekking tot welk *framework* zou kunnen worden gebruikt en deze zijn vergeleken met elkaar. Ook zijn de *requirements* verzamelt en waar nodig verduidelijkt. Per *framework* is ook gekeken of de *requirements* mogelijk waren en hoe dit zou kunnen werken. zie **5.3. Requirements**.

De *requirements* zijn later nog geschreven als *user stories* zodat het duidelijker werd wie de functionaliteit wil en waarom. zie **6.2. Gevolgen van uitgebreid onderzoek**.

Indien dit nodig was zijn de *requirements* en de planning aangepast om de activiteiten die tijdens het project aan bod zijn gekomen op te kunnen vangen.

9.2. C6: Ontwerpen van software

De manier waarop het planbord is opgezet, is in bepaalde stappen uitgevoerd. Er is als eerste gekeken naar het huidige prototype en de verschillende componenten die nodig zijn voor een planbord zijn toen gedefinieerd. Hierna is er een klassendiagram opgezet waarop de relaties tussen de verschillende componenten zijn beschreven. Een voorbeeld is te vinden bij **6.4.1.1. Ontwerpen van software** en **6.4.2.1. Ontwerpen van software**.

De componenten zijn ontwikkeld vanuit de gedachte dat zij hun eigen functies vasthouden zodat het duidelijk is waar deze zich bevinden. Ook zijn de verschillende onderdelen opgesplitst als ze te groot werden.

De *layout* van het planbord is grotendeels overgenomen. Al zijn er wel enkele wijzigingen gemaakt in verband met gebruiksvriendelijkheid. De datums en *timeslots* blijven altijd in beeld terwijl dit in het prototype kon verdwijnen. Het is mogelijk om een bepaald planbord groter of kleiner te maken. En door middel van het inklappen van een *resource* wordt ruimte bespaart. Ook het openen in een ander venster helpt de gebruiker met zijn taken.

9.3. D14 Realiseren van software

Het planbord is gerealiseerd door middel van het *framework* React in combinatie met Redux die verschillende functionaliteiten bevat. Naast de bekende basisfunctionaliteiten zoals het verslepen van orders en wisselen tussen de verschillende *resource types* en *layouts*, is het ook mogelijk om het planbord op een tweede scherm uit te voeren die werkt binnen de Apex-omgeving. Het realisatie proces is beschreven in **6.4.1.2. Realiseren van software** en **6.4.2.2. Realiseren van software**.

Bepaalde functionaliteiten, zoals het wisselen tussen verschillende *layouts*, zijn erg uitbreidbaar opgesteld. Hoewel een gebruiker momenteel maximaal vier verschillende *layouts* kan kiezen, is het vrij simpel om dit later uit te breiden.

9.4. D15 Testen

Vanuit Adaption was al bekend dat de testen moesten worden geschreven door middel van Selenium. Hier is ook gebruik van gemaakt, met betrekking tot het beschrijven van de testcases en het ontwikkelen hiervan met Netbeans.

Voordat de testen werden geschreven, was er al een duidelijk plan over wat er zou worden getest en hoe dit kon worden geverifieerd. De benodigde componenten zijn benoemd en uitgewerkt voordat de testen werden gerealiseerd. Tijdens het realiseren zijn de testen modulair opgebouwd, zodat deze stukken kunnen worden hergebruikt.

Hierna zijn de testen toegevoegd aan de Selenium build en worden tweemaal daags uitgevoerd om de werking van het planbord te waarborgen. Het gehele proces is te vinden in hoofdstuk **6.5. Testcases**.

9.5. Gc: Kritisch onderzoekend en methodisch werken

Tijdens het gehele project ben ik kritisch geweest op mijn eigen werk en of dit wel de manier is waarop ik een bepaald probleem wil/moet oplossen. Een voorbeeld hiervan is het gebruik van Redux. Het heeft mij redelijk wat tijd gekost om dit te implementeren maar deze keuze is gemaakt omdat naar mijn inzicht de applicatie hier later onder zou gaan leiden.

Methodisch werken is tijdens verschillende punten gebeurd in dit project: de user stories die zijn opgezet, sprints van een bepaalde lengte en de testcases die op een bepaalde manier moesten worden opgebouwd.

9.6. CF: Leren leren, Voorbereiding volgende studiefase en beroep

Er zijn voor mij tijdens dit project veel nieuwe dingen aan het licht gekomen. Het belangrijkste is waarschijnlijk de programmeertaal React en dan zeker in combinatie met Redux. De naam React was mij bekend en ik heb er tijdens mijn studie kort gebruik van gemaakt. Maar er is destijds niet zo diep op de stof ingegaan zoals tijdens dit project. Redux was mij volledig onbekend en kwam ik pas tegen tijdens het onderzoeken van de verschillende *frameworks* en toen in een oplossing nodig had. Ook de testtool Selenium kende ik nog niet. Maar ik heb mijzelf dit echter wel in een korte tijd aangeleerd.

Ook de werkwijze van Adaption waarbij afspraken op tijd maken erg belangrijk is, was voor mij vrij lastig tijdens de eerste paar weken. Ook het op tijd aangeven wanneer het werk wat bijna af was, zodat er nieuwe afspraken konden worden gemaakt, is een belangrijk leerproces gebleken.

10. Literatuurlijst

AngularJS. (z.d.). Geraadpleegd 28 mei 2019, van https://docs.angularjs.org/misc/version-support-status

AngularJS — MVW Framework. (z.d.). Geraadpleegd 30 mei 2019, van https://angularjs.org/

CSS resize property. (z.d.). Geraadpleegd 30 mei 2019, van https://www.w3schools.com/cssref/css3_pr_resize.asp

Dojo Toolkit. (z.d.). Geraadpleegd 30 mei 2019, van https://dojotoolkit.org/

donejs - About. (z.d.). Geraadpleegd 30 mei 2019, van https://donejs.com/About.html

Hello Dojo. (z.d.). Geraadpleegd 27 mei 2019, van https://dojotoolkit.org/documentation/tutorials/1.10/hello_dojo/index.html

IDE - Wikipedia. (2019, 16 april). Geraadpleegd 30 mei 2019, van https://nl.wikipedia.org/wiki/Integrated_development_environment

Logistics Cloud Suite - Schaalbare Logistieke Cloud Software. (2019, 20 mei). Geraadpleegd 30 mei 2019, van https://www.adaption-it.nl/producten/

MoSCoW-methode. (2019, 4 februari). Geraadpleegd 27 mei 2019, van https://nl.wikipedia.org/wiki/MoSCoW-methode

Oracle Apex. (z.d.). Geraadpleegd 30 mei 2019, van https://apex.oracle.com/en/

Over Adaption - Adaption. (2019, 20 mei). Geraadpleegd 30 mei 2019, van https://www.adaption-it.nl/over-adaption/

React - Wikipedia. (2019, 30 mei). Geraadpleegd 30 mei 2019, van https://en.wikipedia.org/wiki/React_(JavaScript_library)

Scrum - Wikipedia. (2019, 9 mei). Geraadpleegd 30 mei 2019, van https://nl.wikipedia.org/wiki/Scrum_(softwareontwikkelmethode)

State Of JavaScript: The State Of JavaScript. (z.d.). Geraadpleegd 30 mei 2019, van http://2016.stateofjs.com/

The State Of JavaScript 2017. (z.d.). Geraadpleegd 30 mei 2019, van https://2017.stateofjs.com/

The State of JavaScript 2018. (z.d.). Geraadpleegd 30 mei 2019, van https://stateofjs.com/

Thinking in React – React. (z.d.). Geraadpleegd 30 mei 2019, van https://reactjs.org/docs/thinking-in-react.html

Watervalmethode - Wikipedia. (2019, 9 mei). Geraadpleegd 30 mei 2019, van https://nl.wikipedia.org/wiki/Watervalmethode

Window.open(). (2019, 29 mei). Geraadpleegd 30 mei 2019, van https://developer.mozilla.org/en-US/docs/Web/API/Window/open

XPath. (2019, 25 april). Geraadpleegd 30 mei 2019, van https://developer.mozilla.org/en-US/docs/Web/XPath

11. Bijlage

11.1. Bijlage A: Test data

Charter

Resource #1	
name:	Thiefing Charter Service
id:	1

Resource #2	
name:	Priority Chartering Service
id:	2

Truck

Resource #1	
name:	Grave Digger
id:	3

Resource #2	
name:	Jurassic Attack
id:	4

Driver

Resource #1	
name:	Peter Pan
id:	5

Resource #2	
name:	Captain Hook
id:	6

Layouts

Layout #1	
name:	Default Layout
weekType:	5-day week
future:	1
past:	1

Layout #2	
name:	Single Day View
weekType:	1-day
future:	0
past:	0

Time Slots for Layout #1	
till:	00:00
from:	11:59
timeslot:	Morning
till:	12:00
from:	23:59
timeslot:	Evening

Time Slots for Layout #2	
till:	06:00
from:	07:59
timeslot:	Morning
till:	8:00
from:	15:59
timeslot:	Midday
till:	16:00
from:	20:59
timeslot:	Evening
till:	21:00
from:	23:00
timeslot:	Night

Ongeplande Orders

Order 1

dossier:	31623
order:	**31626**
adr_yn:	Y
start:	DE
eind:	BA
start_date:	04/01/2019
end_date:	10/01/2019

Order 2

dossier:	31623
order:	**31625**
adr_yn:	Y
start:	BA
eind:	LU
start_date:	04/01/2019
end_date:	15/01/2019

Order 3

dossier:	31623
order:	**31624**
adr_yn:	Y
start:	NL
eind:	NL
start_date:	07/01/2019 09:00
end_date:	07/01/2019 17:00

Order 4

dossier:	31627
order:	**31628**
adr_yn:	N
start:	DE
eind:	NL
start_date:	03/01/2019
end_date:	09/01/2019

Order 5

dossier:	31627
order:	**31629**
adr_yn:	N
start:	NL
eind:	DE
start_date:	10/01/2019
end_date:	20/01/2019

Geplande Orders

Order 1

dossier:	31636
order:	**31637**
adr_yn:	N
start:	LU
eind:	NL
start_date:	08/01/2019 06:00
end_date:	08/01/2019 23:00
resource:	3

Order 2

dossier:	31642
order:	**31643**
adr_yn:	N
start:	FR
eind:	NL
start_date:	04/01/2019 12:00
end_date:	10/01/2019
resource:	4

dossier:	31636
order:	**31638**
adr_yn:	N
start:	LU
eind:	NL
start_date:	10/01/2019 14:00
end_date:	15/01/2019
resource:	2

dossier:	31540
order:	**31641**
adr_yn:	Y
start:	CN
eind:	CN
start_date:	03/01/2019
end_date:	27/01/2019
resource:	1

11.2. Bijlage B: Test Cases

<table>
<tr><td>Summary</td><td>Test different settings to change the planbords view.</td></tr>
<tr><td>Users</td><td>SELENIUM TEST</td></tr>
<tr><td>Preconditions</td><td></td></tr>
<tr><td>Steps</td><td>
<table>
<tr><th>#</th><th>Page Actions</th><th>Check</th></tr>
<tr><td>1</td><td>Open Planboard (React) Page</td><td></td></tr>
<tr><td>2</td><td>Select the first dropdown menu afther the refresh button</td><td></td></tr>
<tr><td>3</td><td>Select the value Trucks</td><td>Check of the resource that are added are Grave Digger and Jurassic Attack</td></tr>
<tr><td>4</td><td>Select the last dropdown menu in the top menu</td><td></td></tr>
<tr><td>5</td><td>Select the value Default Layout</td><td>Selection should now be 7 days starting with Friday and ending with Monday</td></tr>
<tr><td>6</td><td>Select the dropdown menu with the year in it.</td><td></td></tr>
<tr><td>7</td><td>Select the value 2023</td><td>The date changed with the year including the year shown in the dropdown menu.</td></tr>
<tr><td>8</td><td>Select the dropdown menu next the the year</td><td></td></tr>
<tr><td>9</td><td>Select the value of 6</td><td>The daterange is now starting with 03/02/2023 and ending with 13/02/2023</td></tr>
<tr><td>10</td><td>Select the button with a caret in it signaling left.</td><td>The daterange is now starting with 27/01/2023 and ending with 06/02/2023</td></tr>
<tr><td>11</td><td>Select the button with a caret in it signaling right</td><td>The daterange is now starting with 03/02/2023 and ending with 13/02/2023</td></tr>
</table>
</td></tr>
<tr><td>Postconditions</td><td>Different fields have been tested to ensure that it changed the content of the planboard</td></tr>
<tr><td>Cleanup</td><td>Reload the planboard.</td></tr>
</table>

Summary	Use the different filters to find certain unplanned orders.
Users	SELENIUM TEST
Preconditions	

Steps

#	Page Actions	Check
1	Open Planboard (React) Page	
2	Select the first dropdown menu afther the refresh button	
3	Select the value **Trucks**	Check of the resource that are added are **Grave Digger** and **Jurassic Attack**
4	Select the dropdown menu with the year in it.	
5	Select the value **2019**	The date changed with the year including the year shown in the dropdown menu.
6	Select the dropdown menu next the the year	
7	Select the value of **2**	Order **31643** should now be visible on the planned section
8	Select the input field on the lower menu	
9	Fill the input field with data: **31627**	Order **31628** and **31629** are visible
10	Select the dropdown list next to the input field	
11	Select the option **Order**	Check if the selected value in the dropdown list has changed.
12	Will the input field with data: **31625**	Order **31625** are visible
13	Remove the data that was entered in the input field	Orders **31624**, **31625**, **31626**, **31628** and **31629**
14	Select the dropdown list next to the text of ADR	
15	Select the option of **Yes**	Orders **31626**, **31625** and **31624** should be visible
16	Select the option of **No**	Order **31628** and **31629** are visible.

Postconditions	Different orders are shown because of the selected value
Cleanup	Select the dropdown list again and select the option **None**

Summary	Filter on locations for unplanned orders.
Users	SELENIUM TEST
Preconditions	
Steps	

#	Page Actions	Check
1	Open Planboard (React) Page	
2	Select the first dropdown menu afther the refresh button	
3	Select the value **Trucks**	Check of the resource that are added are **Grave Digger** and **Jurassic Attack**
4	Select the dropdown menu with the year in it.	
5	Select the value **2019**	The date changed with the year including the year shown in the dropdown menu.
6	Select the dropdown menu next the the year	
7	Select the value of **2**	Order **31643** should now be visible on the planned section
8	Select the option **DE**	Order **31626** and **31628** are visible
9	Select the option **NL**	Order **31624** and **31629** are visible
10	Select the dropdown list next to the text of “startlocatie”	
11	Select the option **NL**	Order **31624** is visible

Postconditions	After all filters only one order is shown
Cleanup	Set both dropdown options to **None**

Summary	Use the filter in multiple combinations to find unplanned orders.
Users	SELENIUM TEST
Preconditions	
Steps	

#	**Page Actions**	**Check**
1	Open Planboard (React) Page	
2	Select the first dropdown menu afther the refresh button	
3	Select the value **Trucks**	Check of the resource that are added are **Grave Digger** and **Jurassic Attack**
4	Select the dropdown menu with the year in it.	
5	Select the value **2019**	The date changed with the year including the year shown in the dropdown menu.
6	Select the dropdown menu next the the year	
7	Select the value of **2**	Order **31643** should now be visible on the planned section
8	select the dropdown list next to ADR and select **NO**	Order **31628** and **31629** should be visible
9	select the dropdown list next to ‘start locatie’ and select **NL**	Order **31629** is visible
10	select the dropdown list next to ‘eind locatie’ and select **BA**	Orders **31626, 31625, 31624, 31628** and **31629** are not visible.

Postconditions	Multiple combinations of filters were tested and orders where shown if they where valid.
Cleanup	Empty the input field and Put all dropdownlist to the option of **None**

Summary	Move the orders to confirm it actually changes the data.
Users	SELENIUM TEST
Preconditions	

Steps

#	Page Actions	Check
1	Open Planboard (React) Page	
2	Select the value **Trucks, 2019** and **2** in the dropdown menus at the top menu.	
5	Move the mouse towards the order **31637**	
6	Hold and drag the order towards the unplanned section	
7	Let go of the order	A screen should be seen now with data regarding the specific order
8	Select **Save** in the screen that is shown	The order is removed from the planned section of the screen.
9	Move the mouse towards the order **31637**	
10	Hold and drag the order towards the planned section on the first resource called **Grave Digger**	
11	Let go of the order	A screen should be seen with data
12	Select **Save** in the screen that is shown	the resource **Grave Digger** now has the order of **31637**
13	Move the mouse towards the order **31643**	
14	Hold and drag the order towards the planned section on the resource called **Grave Digger**	
15	Let go of the order	A screen should be seen with data
16	Select **Save** in the screen that is shown	the resource **Grave Digger** now has the order of **31643**
17	Move the mouse towards the order **31643**	
18	Hold and drag the order towards the planned section on the resource called **Jurassic Attack**	
19	Let go of the order	A screen should be seen with data
20	Select **Cancel** in the screen that is shown	The order of **31643** stays on its current resource

Postconditions	Orders have been moved around to test the functionality

Cleanup	move **31643** to **Jurassic Attack** and click on **Save**

Summary	Tests regarding the contextmenu
Users	SELENIUM TEST
Preconditions	

Steps

#	Page Actions	Check
1	Open Planboard (React) Page	
2	Select the value **Trucks, 2019** and **2** in the dropdown menus at the top menu.	
3	Move the mouse towards the order **31643**	
4	Simulate a right mouse click on the order	A menu should now open with a multitude of functions, if only one option is available of **Error** then it failed
5	Hover over the option **31643**	More options should become available.
6	Select the option **Edit**	A new tab should be opened specific to that Order.
7	Go back to the planboard tab	
8	Move the mouse towards the order **31637**	
9	Simulate a right mouse click on the order	A menu should now open with a multitude of functions, if only one option is available of **Error** then it failed
10	Hover over the dossier that is **31636**	More options should become available
11	Select the option **Customs Declaration**	A modal should open in front of the user.

Postconditions	Tested the context menu with multiple options to guarantee functionality
Cleanup	Press the close button on the top right of the modal.

Summary	Test certain functionality while using a new window.
Users	SELENIUM TEST
Preconditions	

Steps

#	Page Actions	Check
1	Open Planboard(React) Page	
2	On the top menu click on the button **New Window**	A new window should now open with the planboard being loaded.
3	Select the value **Trucks, 2019** and **2** in the dropdown menus at the top menu.	
4	Move the mouse towards the order **31637**	
5	Simulate a right mouse click on the order	A menu should now open with a multitude of functions, if only one option is available of **Error** then it failed
6	Hover over the option **31637**	More options should be available.
7	Select the option **Edit**	A new tab should be opened in the old window specific to that Order.
8	Go back to the new window	
9	Move the mouse towards the order **31643**	
10	Hover over the option **31642**	More options should become available
11	Select the option **Send Email**	A modal should open in front of the user which allows them to send an email.

Postconditions	Tested certain functionality while using a new window.
Cleanup	Press the close button on the top right of the modal. Close the new window.

11.3. Bijlage C: StateOfJS onderzoek calculaties

2017 Corrected	Never heard of it.	Heard of it, not interested.	Heard of it, would like to learn.	Used it before, would not use again.	Used it before, would use again.	
Angular2	53	2395	1365	515	1000	5328
React	18	578	1430	229	3072	5328
Backbone	327	2844	495	1276	385	5328
2018 Corrected	Never heard of it.	Heard of it, not interested.	Heard of it, would like to learn.	Used it before, would not use again.	Used it before, would use again.	
Angular2	0	1697	552	1805	1274	5328
React	4	492	1020	358	3454	5328
Backbone	0	0	0	0	30	
2016 Percentage	Never heard of it.	Heard of it, not interested.	Heard of it, would like to learn.	Used it before, would not use again.	Used it before, would use again.	
Angular 2	0,92	43,06	35,53	7,21	13,29	100,00
React	0,39	10,90	31,57	4,54	52,59	100,00
Backbone	2,42	47,13	11,47	26,60	12,39	100,00
2017 Percentage	Never heard of it.	Heard of it, not interested.	Heard of it, would like to learn.	Used it before, would not use again.	Used it before, would use again.	
Angular 2	1,00	44,96	25,62	9,66	18,77	100,00
React	0,35	10,85	26,84	4,30	57,67	100,00
Backbone	6,14	53,38	9,29	23,95	7,24	100,00
2018 Percentage	Never heard of it.	Heard of it, not interested.	Heard of it, would like to learn.	Used it before, would not use again.	Used it before, would use again.	
Angular 2	0,00	31,85	10,37	33,88	23,91	100,00
React	0,07	9,23	19,14	6,72	64,83	100,00
Backbone	0,00	0,00	0,00	0,00	0,56	