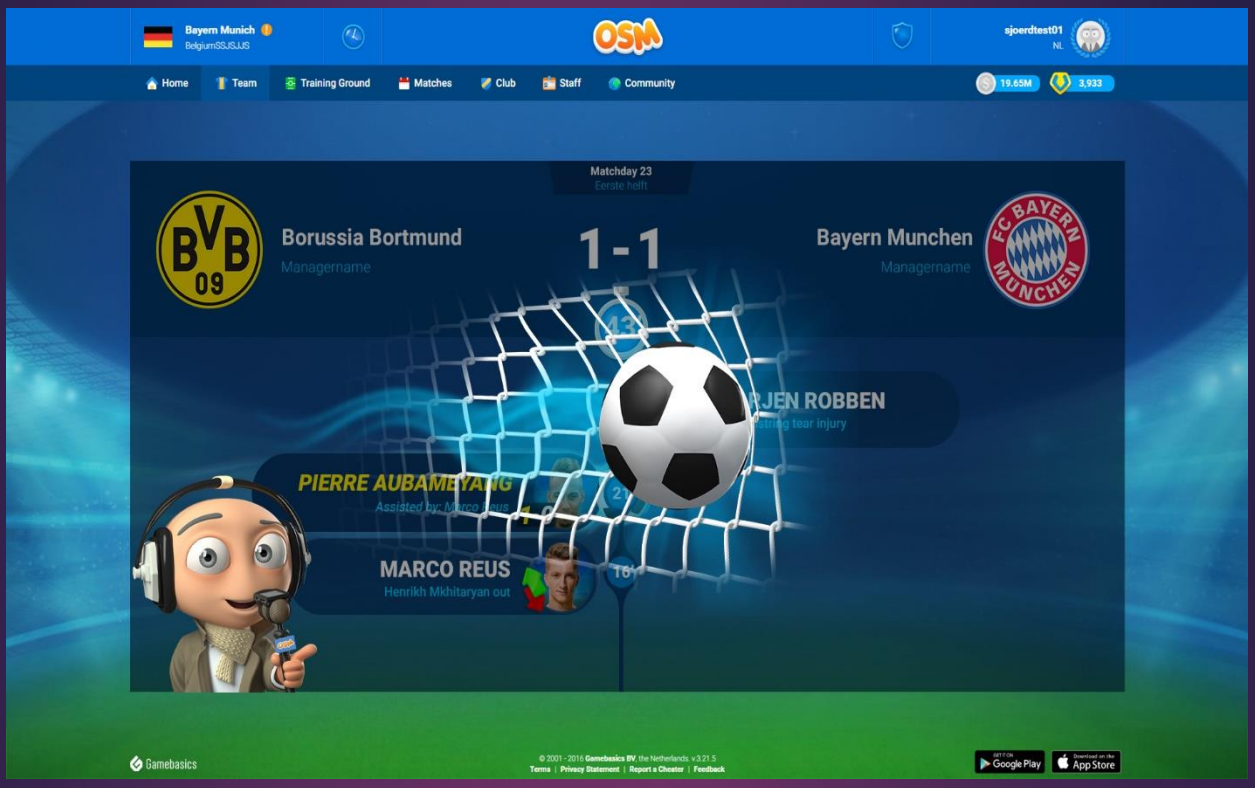
2017



Match Experience

AFSTUDEERVERSLAG

PATRICK AUBROECK

GAMEBASICS | Buitenom 263

REFERAAT

Titel: Match Experience

Naam: Patrick Aubroeck

Studentennummer: 12115282

E-mailadres: P.Aubroeck@hotmail.com/12115282@student.hhs.nl

Afstudeerperiode: 2017.1.1 (start 6 februari)

Onderwijsinstelling: De Haagse Hogeschool te Zoetermeer

Opleiding: Informatica

Begeleidend examinator: Ton Biegstraaten

Tweede examinator: Nanny Jacobs

Studioloopbaanbegeleider: Rob Loke

Organisatie: Gamebasics 'Team Best'

Opdrachtgever: Raymond Manche

Begeleiders: Jurgen Welschen, Jelle Braat

Kernwoorden: Match experience, Web, Javascript, Framework Onderzoek, Online Soccer Manager

VOORWOORD

Voor u ligt het afstudeerverslag van mijn werkzaamheden die zijn gedaan bij het bedrijf Gamebasics [1]. Deze werkzaamheden zijn gedaan in verband met mijn afstudeerperiode en het afronden van mijn opleiding informatica aan de Haagse Hogeschool te Zoetermeer.

Voordat ik begon met afstuderen had ik al reeds een jaar bij het bedrijf Gamebasics gewerkt. Aan deze werkperiode heb ik hele goede ervaringen overgehouden wat een van de redenen was om bij Gamebasics te gaan afstuderen.

De opdracht die ik heb gekregen van het bedrijf Gamebasics was voor mij een zeer uitdagende en leerzame opdracht. Gedurende mijn schoolperiode heb ikzelf de focus vooral gelegd op low-end talen, zoals C++ en wilde ik graag verder met 3D Graphics/Engines. De opdracht die ik bij Gamebasics heb gekregen is dat juist totaal niet, wat ervoor zorgde dat deze periode heel leerzaam voor mij was. In plaats van de focus op het backend te leggen lag de focus met deze opdracht juist op het front-end. Een van de punten waar ik persoonlijk blij mee was, dat ik vrij was in vrijwel alle keuzes qua methodieken, technieken, tools, etc. Hier ben ik Gamebasics en mijn begeleiders bij Gamebasics dan ook zeer dankbaar voor. Dat zij mij het vertrouwen hebben gegeven dat ik dit op deze manier heb mogen doen.

Als eerste wil ik Jasper Schwartz en Youri Trugg bedanken om mij uit te nodigen voor een gesprek. Daarnaast wil ik graag Raymond Manche bedanken voor de opdracht en de ondersteuning tijdens mijn afstudeerperiode. Als laatste wil ik graag Jurgen Welschen en Jelle Braat bedanken voor alle feedback en begeleiding die zij mij hebben gegeven tijdens mijn afstudeerperiode.

Tot slot wil ik graag mijn begeleidend examinators, Ton Biegstraaten en Nanny Jacobs, bedanken voor hun begeleiding tijdens mijn afstudeerperiode.

Gamebasics

TABLE OF CONTENTS

Gamebasics

BEGRIPPENLIJST

BEGRIP	DEFINITIE/VERKLARING
JSX	Een object-georiënteerde programming language dat wordt gebruikt in React. Is een combinatie van html met javascript.
ONE WAY DATA BINDING	Het model is de enige bron van waarheid. De view kan het model niet direct veranderen, wanneer de model data aangepast wordt zal hierdoor de view aangepast worden.
TWO WAY DATA BINDING	De model data is dynamisch verbonden aan de view. Het model veranderd wanneer de view wordt aangepast en visa-versa.
VIRTUAL DOM	Een kopie van de actuele DOM, op de virtuele DOM worden aanpassingen gedaan die na een rerender worden opgepakt door de actuele DOM.
CODE SPLITTING	Verdeel de code in verschillende en kleinere resources. Zorgt ervoor dat de resources op meerdere manieren beheert kan worden.
TYPESCRIPT	Een extensie van javascript dat compileert naar standaard javascript.
HOT-RELOADING	Zorgt ervoor dat alleen de files die zijn geüpdatet gerenderd worden
STUB	Simuleert het gedrag van een onderdeel waar de test van afhankelijk is.
ECMASCRIPT(GETAL)	Gestandaardiseerde javascript.
A-SYNC (ASYNCHRONOUS)	Events die gelijktijdig met andere events kunnen samenlopen.
MIXINS (LESS/SASS)	Bied de gebruiker de mogelijkheid om classes te definiëren die in dezelfde of in andere files hergebruikt kunnen worden.
ISOMORPHIC FETCH	Zorgt ervoor dat de fetch global wordt, waardoor de fetch consistent is tussen server en client.
STATIC TYPED LANGUAGE	Alle variable zijn op compile time al gedefinieerd en bekend.
UX-AFDELING	User-experience afdeling.
VIEWPORT	Het zichtbare gedeelte van een webpagina voor een gebruiker.
JSDOM	Jsdom is een javascript implementatie van vele web standaarden.
ACTION CREATOR (REDUX)	Een functie of object dat een action aanmaakt.
ACTION (REDUX)	Een object(payload) van informatie dat verstuurd wordt naar de store.
REDUCER (REDUX)	Een object dat de nieuwe state bewerkt aan de hand van de action object.

STORE (REDUX)	Een object dat de huidige state bijhoudt en het af en aanmelden van listeners bijhoudt.
PROVIDER (REDUX)	Een object dat de store aan de view koppelt. Ook wel view layer binder genoemd.
VIEW LAYER BINDER (REDUX)	Een object dat de store aan de view koppelt. Ook wel Provider genoemd.
REDUX-DESIGN PATTERN	Een design pattern dat de Ui-state probleem probeert op te lossen.
STATELESS COMPONENT	Een object, meestal een view component die alleen maar weet hoe de data weergeven moet worden.
DEPENDENCIES	Wanneer een gedeelte van de software afhankelijk is van een ander gedeelte.
RESPONSIVE	Een webdesign dat probeert websites optimaal weer te geven op alle mogelijke resoluties.
MOCHA	Een javascript test framework.
CHAI	Een assertion library.
SINON	Een standalone test library, dat gebruik maakt van spies, stubs en mocks.
MODULE BUNDLER	Koppelt meerdere dependencies aan elkaar en maakt hier een single file van.
PACKAGE MANAGER	Een systeem dat het installeren, upgraden, configureren en het verwijderen van softwareprogramma's op een consistente manier automatiseert.
ASSERTION	Een vergelijking.
LESS	Een extensie op de CSS language.
NPM	Een Package manager.
WEBPACK	Een module bundler.
BABEL	Een transpiler.
TRANSPILER	Een systeem/library dat een bepaalde source code language omzet naar een andere programming language.

Gamebasics

1. INLEIDING

1.1. DOELGROEP

Dit afstudeerverslag is bedoeld voor mijn begeleidend examinators, Ton Biegstraaten en Nanny Jacobs en zal als representatie worden gebruikt voor mijn afstudeerperiode. Daarnaast is dit afstudeerverslag bedoeld voor iedereen die meer wil weten over hoe ik mijn afstudeerperiode heb doorlopen.

1.2. SITUATIEBESCHRIJVING

Om een duidelijk beeld te schetsen over wat de opdracht inhoud zal er eerst nadere uitleg worden gegeven over het spel Online Soccer Manager [2]. De feature die gemaakt wordt, zal worden geïntegreerd in de huidige game. Online Soccer Manager, vanaf nu OSM genoemd is een spel waarin de gebruiker de manager wordt van een voetbalploeg. De gebruiker zal na het kiezen van een ploeg worden ingedeeld in een league, waarin ook andere gebruikers zitten. Het mooie hiervan is, dat de gebruiker niet alleen tegen computergestuurde ploegen speelt, maar ook daadwerkelijk het moet opnemen tegen andere gebruikers. De gebruiker heeft als doel om een zo goed mogelijke positie te behalen in de league en in de cup. Om dit te behalen zal er gebruik moeten worden gemaakt van verschillende tactieken, zullen er transfers moeten worden gedaan en zal er goed gebruik moeten worden gemaakt van de trainingskampen. Om in te zien hoe goed de manager het doet is er onder andere een wereldkaart, een historie en een ranking van alle managers in het spel OSM. Voor meer informatie over het spel OSM zie de website van OSM [2].

Op dit moment logt de gebruiker in op OSM, waarna deze wordt doorgestuurd naar zijn career Centre waar zijn clubslots kan vinden. Nadat de gebruiker zijn club heeft geselecteerd komt deze terecht op het hoofdmenu, waar direct de uitslag van de wedstrijd, die is gespeeld, wordt weergegeven. Hoewel dit praktisch is, is het niet leuk voor de gebruiker om gelijk zijn uitslag te zien. Dit terwijl er veel gebeurd kan zijn in de wedstrijd, bijvoorbeeld een cruciale penalty of rode kaart. De feature match experience die zal worden gemaakt tijdens deze afstudeerperiode moet ervoor zorgen dat de gebruiker meer wordt betrokken bij het spel. Het moet de gebruiker het gevoel geven dat aanpassingen aan de tactiek en het opstellen van bepaalde spelers ook daadwerkelijk invloed hebben gehad op de uitslag. Dit wordt gedaan door de match experience naar de gebruiker te brengen aan de hand van een samenvatting. De samenvatting zal door gebruik te maken van commentatoren en 2d Animaties de gebruiker het gevoel moeten geven dat ze de wedstrijd aan het spelen zijn. Dit moet ervoor zorgen dat de gebruiker zich meer verbonden voelt met zijn club en spelers en uiteindelijk ook met het spel OSM. Doordat de match experience al actief is op de huidige iOS en Androidversie van het spel, zijn de verwachtingen dat door de match experience meer spelers op het platform web het spel zullen blijven spelen voor een langere tijd.

Naast de match experience te ontwikkelen zal er ook onderzoek moeten worden gedaan naar welk javascript framework er zal worden gebruikt om de match experience te maken. Op het moment is het bedrijf Gamebasics en in het specifiek de afdeling web bezig met het herstructureren van de applicatie en zal dit onderzoek worden gebruikt om te bepalen of het uitgekozen framework zal worden gebruikt om de gehele applicatie mee te bouwen.

1.3. LEESWIJZER

In hoofdstuk 1 zal worden beschreven wat de opdracht is, hoe deze tot stand is gekomen. Daarnaast is er te zien hoe het verslag is ingedeeld.

In hoofdstuk 2 zal het bedrijf worden beschreven, waar het bedrijf voor staat en wat hun doelen zijn. Daarnaast zal er een organogram gebruikt worden om het bedrijfsstructuur weer te geven.

In hoofdstuk 3 zal de opdracht worden beschreven en de verwachtingen die het bedrijf heeft.

In hoofdstuk 4 zal de technieken, methodieken en tools die zijn gebruikt nader worden beschreven.

In hoofdstuk 5 en 6 zal de structuur van de applicatie worden beschreven en welke keuzes er zijn gemaakt om tot deze structuur te komen.

In hoofdstuk 7, 8, 9, 10 zal het gaan over de requirements, de opgestelde klassendiagrammen, de implementatie van de match experience en het testen van de functionaliteiten.

Als laatste zullen er in hoofdstuk 11 de evaluatiepunten worden beschreven over de producten, processen en beroepstaken die tijdens deze afstudeerperiode aan bod zijn gekomen.

2. HET BEDRIJF

2.1. GESCHIEDENIS

Gamebasics is het bedrijf achter het online voetbalspel Online Soccer Manager. Gamebasics is in 2004 opgericht, maar is begonnen in 2001 op de zolderkamer van Jeroen Derwort. Het bedrijf is inmiddels uitgegroeid tot een volwassen bedrijf van ongeveer 40 werknemers. Deze 40 werknemers werken elke dag hard aan het spel OSM dat op het moment meer dan 4 miljoen unieke spelers heeft. Elke avond worden er door de engines alle wedstrijden met de tactieken die de spelers hebben ingevoerd afgespeeld. Inmiddels is versie 3.0 van de game OSM uit en is het bedrijf elke dag volop bezig om nieuwe features toe te voegen aan het spel. Gamebasics werkt in een agile omgeving met multidisciplinaire teams die ieder aan een eigen platform werken.

2.2. MISSIE

De missie die voordat ik kwam afstuderen, was de onderstaande missie.

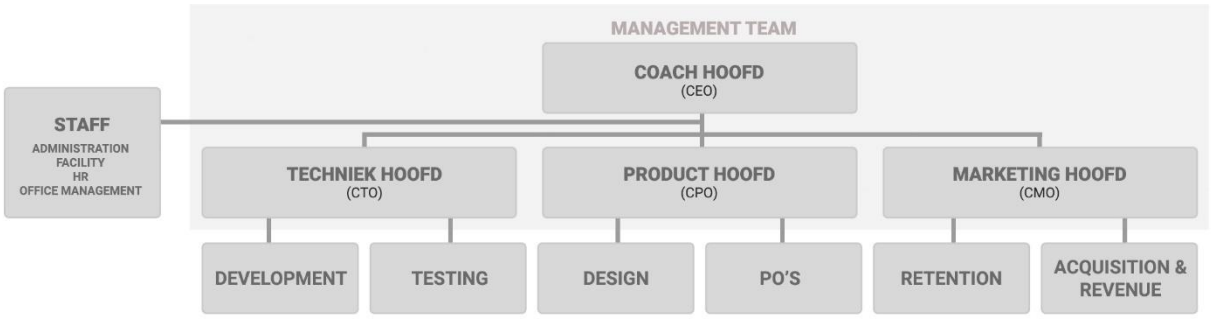
"Gamebasics streeft ernaar alle voetballiefhebbers wereldwijd het gevoel te geven dat ze manager zijn van hun favoriete voetbalclub. Elke dag en overal. Dit willen we bereiken door een game ervaring te bieden die authentiek, laagdrempelig en uitdagend is".
Jeroen Derwort en Frank Tijhuis Co-founders Gamebasics

Deze missie is veranderd naar het volgende.

“Wij geven alle voetballiefhebbers de mogelijkheid met elkaar de competitie aan te gaan door middel van een laagdrempelig en authentiek voetbal manager spel.”
Jeroen Derwort en Frank Tijhuis Co-founders Gamebasics

2.3. INTERNE ORGANISATIE

Zoals eerder beschreven bestaat het bedrijf uit ongeveer 40 medewerkers deze medewerkers zijn opgedeeld in meerdere groepen van rang en functie. In het organogram hieronder is te zien wat de indeling is bij het bedrijf Gamebasics.



Figuur 1 - Organogram Gamebasics

De directie is het overkoepelende orgaan wat toezicht houdt op alle afdelingen en kijkt of alle afdelingen naar behoren functioneren en geven richting aan de afdelingen.

De product owners bepalen wat voor elk team het belangrijkste is om aan te werken.

De afdeling staff houdt zich bezig met de financiën, uitjes, lunch en alles wat er rondom de werkzaamheden komt kijken.

De afdeling marketing is onderverdeeld in twee disciplines: retentie en acquisitie. Acquisitie houdt zich bezig met het binnenhalen van spelers. En retentie probeert de binnen gehaalde spelers langer aan het spel te binden.

De afdeling front-end development is onderverdeeld in drie platforms: Web, iOS en Android. Deze teams zijn op zichzelf-staande teams die allemaal een product owner, tester en een designer hebben.

Backend zorgt onder andere voor de applicatie en de API. De API wordt aangeroepen door alle front-end afdelingen. De backend moet er om deze reden voor zorgen dat de Api te allen tijde de juiste data teruggeeft aan de front-end teams.

User experience is sinds kort opgericht en staat nog in de kinderschoenen. Deze afdeling houdt zich bezig met het onderzoeken, ontwerpen en testen van nieuwe features. Dit wordt in samenspraak gedaan met de product-owners en het backend team.

Het organogram laat hiërarchische lagen zien in de organisatie maar in werkelijkheid is er een platte organisatie waarin iedereen openstaat voor feedback van elkaar. Alleen zijn er wel functies binnen de organisatie die de beslissingen nemen, vandaar heb ik het uitgebeeld met de verschillende lagen.

Tijdens de afstudeerperiode zal ik worden ingedeeld in het development team en in het specifiek het development team web.

3. OPDRACHT OMSCHRIJVING

De opdracht kan in twee delen worden opgesplitst, namelijk het eerste gedeelte een onderzoek naar javascript frameworks en het tweede gedeelte het maken van de match experience.

- Het onderzoek

Het eerste gedeelte van de opdracht zal zijn het onderzoeken van javascript frameworks en een adviesrapport geven over welk javascript framework het beste gebruikt kan worden voor het maken van de match experience. Het javascript framework zal aan de hand van requirements worden bepaald. Sommige van deze requirements zijn terug te vinden in hoofdstuk 6, voor alle requirements kan er worden gekeken naar de bijlage adviesrapport javascript frameworks.

De requirements zullen worden verkregen door het houden van interviews en zullen worden opgesteld aan de hand van het Handboek Requirements [68]. Hierbij zullen de product owner en de begeleiders van de afdeling web worden geïnterviewd om de requirements naar boven te halen.

- Match experience

Het tweede gedeelte en ook gelijk het belangrijkste onderdeel van de afstudeerperiode zal zijn het maken van een volledig werkend prototype van de feature match experience. De match experience zal moeten worden gemaakt in het framework dat als advies uit het onderzoek is gekomen in deel één. Ook voor de match experience zullen er requirements moeten worden opgesteld. Veel van de requirements zullen al bekend zijn, dit omdat er al een werkende match experience is op zowel iOS en Android. Om er zeker van te zijn dat voor de afdeling Web alle requirements naar boven zijn gekomen, zijn ook hier interviews gehouden.

De match experience zal naast het framework ook gebruik moeten maken van de al bestaande Web-Api en hier zal rekening mee moeten worden gehouden. Ook zal de match experience geïntegreerd moeten kunnen worden in de huidige OSM-game.

4. PLAN VAN AANPAK, METHODEN, TECHNIEKEN EN TOOLS

Een van de belangrijkste punten die uitgezocht moet worden is welke ontwikkelmethodieken, technieken en tools er gebruikt zullen worden tijdens de afstudeerperiode. Dit hoofdstuk zal in detail gaan over welke methodieken, technieken en tools er zijn gekozen en waarom deze methodieken, technieken en tools zijn gekozen.

4.1. PLAN VAN AANPAK

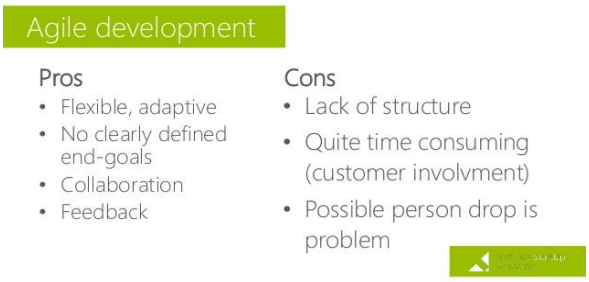
Om een plan van aanpak op te stellen is er gebruik gemaakt van Scribbr [3]. Scribbr is een website die allerlei functionaliteiten biedt om op Hbo-niveau documenten te schrijven. Waaronder een plan van aanpak. De indeling die is gebruikt voor het plan van aanpak is ook de indeling geweest die wordt beschreven op Scribbr. Het plan van aanpak zorgt ervoor dat de structuur van de afstudeerperiode goed zichtbaar is en dat er een duidelijke indeling is van welke taken er gedaan moeten worden tijdens een project/periode. In onderstaande tabel is de planning te zien van de eerste drie sprints. De overige sprints zullen in de loop van het project worden ingedeeld.

Sprint 1	Onderdeel
	Requirements opstellen match experience
	Klassendiagrammen maken structuur applicatie
	Project opzet
	Testen schrijven
Sprint 2	Onderdeel
	Data ophalen
	Studio (view) opzet (responsive)
	Data weergeven
	Testen schrijven
	Klassendiagrammen maken commentatorenteksten
Sprint 3	Onderdeel
	Studio (view) opzet (responsive)
	Data weergeven
	Testen schrijven

Voor meer informatie over het plan van aanpak kan er worden gekeken naar het Plan van aanpak dat is meegeleverd als bijlage.

4.2. ONTWIKKELMETHODIEKEN

Er zijn vele soorten methodieken beschikbaar, om de keuze te minimaliseren is er gekozen voor een agile ontwikkelmethodiek. Een van de redenen waarom er voor een agile ontwikkelmethodiek is gekozen is, omdat het project goed is op te splitsen in deelsystemen. Een ander voordeel is, dat er bij het bedrijf Gamebasics al gebruik gemaakt wordt van een agile ontwikkelmethodiek. Dit zorgt ervoor dat er vanuit het bedrijf kennis is over deze manier van ontwikkelen. In het diagram hieronder zijn de voor en nadelen te zien van een agile ontwikkelmethodiek.



Figuur 2 - Pro en Con's Agile Development [73]

4.2.1. DE OPTIES

Hieronder zijn een tweetal agile ontwikkelmethodieken te zien waar uit gekozen is om te gebruiken voor de afstudeerperiode.

Extreme programming (XP)

Extreme programming is voor het eerst gebruik in 1996[4]. Het voordeel dat extreme programming met zich meebrengt is, dat het mogelijk is om laat in de life cycle van een project nog requirements aan te passen. Dit kan worden gedaan omdat er volgens vijf regels wordt gewerkt. Namelijk Planning, Managing, Designing, Coding en Testing. Voor elk van deze regels staan een paar punten vast die moeten worden behaald [5] voordat deze naar de volgende fase mogen.

Scrum

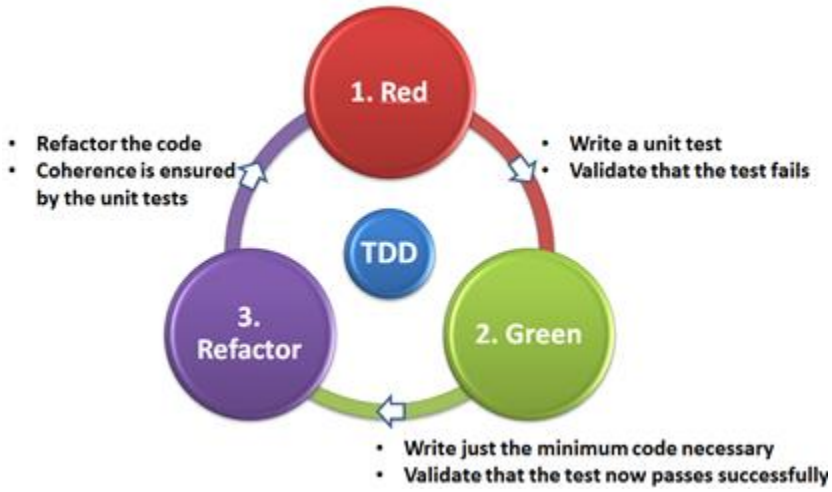
Scrum bestaat sinds 1990 [6] en is vooral gebruikt om complexe projecten te managen. De definitie die de makers van Scrum er zelf aan geven is het volgende.

“A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value”

Om dit te kunnen waarborgen wordt er gebruik gemaakt van verschillende scrum teams en zit er een scrum master in ieder team [7].

4.2.2. DE KEUZE

Er is gekozen om gebruik te maken van Scrum, naast gebruik te maken van Scrum zal er worden ontwikkeld door gebruik te maken van Test Driven Development [9-10]. Er zijn meerdere redenen waarom er gebruik gemaakt wordt van Scrum. Een van deze redenen is, dat het project zich goed weglegt om onderverdeeld te worden in delen, oftewel sprints. Een andere reden is dat er bij het bedrijf Gamebasics al gewerkt wordt met een Scrum ontwikkelmethodiek. Als laatste kan er op deze manier tijdens het traject de requirements worden aangepast om zo een betere versie te maken dan de al bestaande versies in iOS en Android. Daarnaast is er gekozen om gebruik te maken van een Test Driven Development. De reden waarom er gekozen is voor Test Driven Development is, omdat er alleen aan het project wordt gewerkt. Omdat er eerst testen moeten worden geschreven, voordat er begonnen wordt met de implementatie zal er goed moeten worden nagedacht over hoe de functionaliteiten zullen gaan werken. Bij gebrek aan onder andere code checks zal test first de kwaliteit moeten waarborgen van het project.



Figuur 3 - Test Driven Development life cycle

In bovenstaand diagram is te zien hoe de workflow werkt van een Test driven development cycle.

4.3. TECHNIEKEN

In dit hoofdstuk zullen de technieken worden besproken die voor de eerste sprint van de afstudeerperiode bekend waren. Alle andere technieken, tools of gekozen externe libraries zullen in hun respectievelijke sprint hoofdstuk worden beschreven.

4.3.1. FRAMEWORKS ONDERZOEK, OPTIES EN KEUZE

Er is begonnen met het opstellen van de requirements waaraan het javascript framework moest voldoen. In totaal zijn er acht requirements opgesteld aan de hand van interviews. De interviews zijn gehouden met de product owner, de begeleider en met de rest van het team van de afdeling web. Hieronder zijn de requirements te zien met de prioriteit die deze requirements hebben gekregen.

Om te voorkomen dat elke requirement een must have wordt. Is er gebruik gemaakt van de MoSCoW methode om requirements een belangrijkheid te geven [71]. De prioritering is opgesteld in overleg met de product owner en de begeleider vanuit Gamebasics. Aangezien alle acht de requirements belangrijk zijn om een keuze te maken voor het framework zal er geen Won't haves aanwezig zijn.

Nr.	Requirement	Prioriteit
1	Het javascript framework moet een single page kunnen ondersteunen.	M
2	Het javascript framework moet performance gericht zijn. (CRUD).	S
3	Het javascript framework moet een static typed language kunnen ondersteunen.	M
4	Het javascript framework moet externe libraries kunnen ondersteunen.	S
5	Het javascript framework moet goed testbaar zijn. (Dit kan door gebruik te maken van externe libraries of native test libraries).	S
6	Het javascript framework moet een active community hebben. (Dit zodat problemen sneller opgelost kunnen worden).	M
7	Het javascript framework moet het flux design pattern kunnen ondersteunen. (Dit kan door gebruik te maken van native of externe libraries).	S
8	Het javascript framework moet alle browsers ondersteunen vanaf versie IE – 11, Chrome – 56.0.2, Firefox – 51.0 , Safari – 10.0.2 Android – 4.1, iOS – 8.0	C

Nadat de requirements waren opgesteld voor het javascript framework en de prioritering aan deze requirements waren gegeven. Is er gekeken hoe het grote aantal javascript frameworks dat bestaat kan worden verkleind tot een acceptabel aantal dat in 2 tot 3 weken kan worden onderzocht. Om het aantal javascript frameworks te verkleinen is er in overleg met mijn begeleider, Jurgen Welschen, besloten om requirement 6 te pakken, namelijk;

- Het Javascript framework moet een actieve community hebben, zodat problemen sneller opgelost kunnen worden.

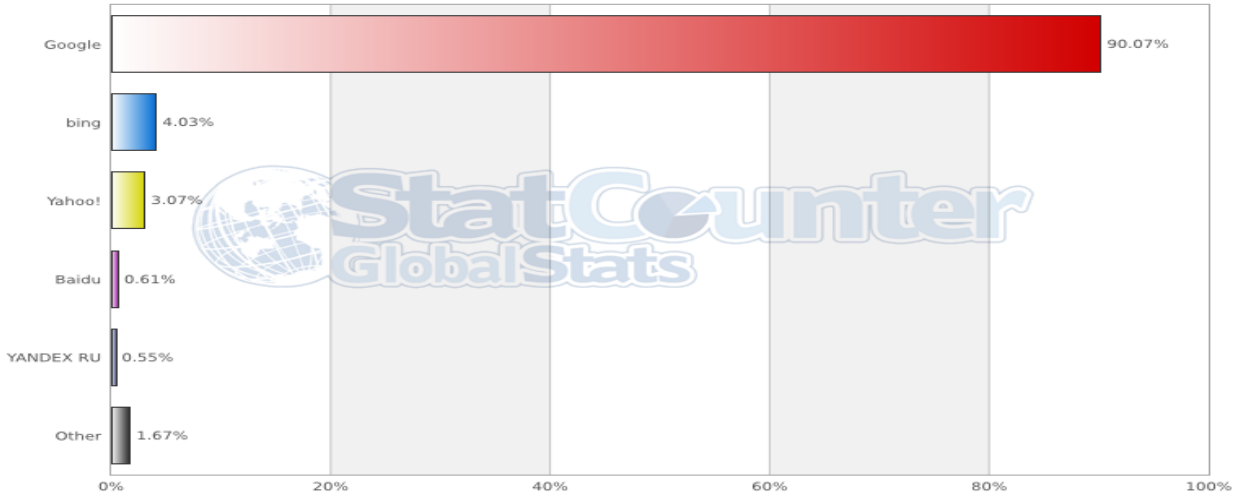
De reden dat er voor deze requirement is gekozen is, dat er op deze manier alle kleine javascript frameworks die weinig tot geen community hebben afvallen en zo alleen de populaire javascript frameworks overblijven. Als eerste is er op tien verschillende websites gekeken met de titel “Top javascript frameworks”. Van deze websites is een top 5 gemaakt, door te kijken hoe vaak ieder framework voorkwam. Zie de bijlage Adviesrapport javascript frameworks voor alle referenties naar deze websites. Hieronder is de top 5 te zien;

- Vue
- React
- AngularJs/Angular2
- Ember
- Backbone

Op deze 5 frameworks is de onderstaande criteria toegepast, om hieruit een top 3 te krijgen.

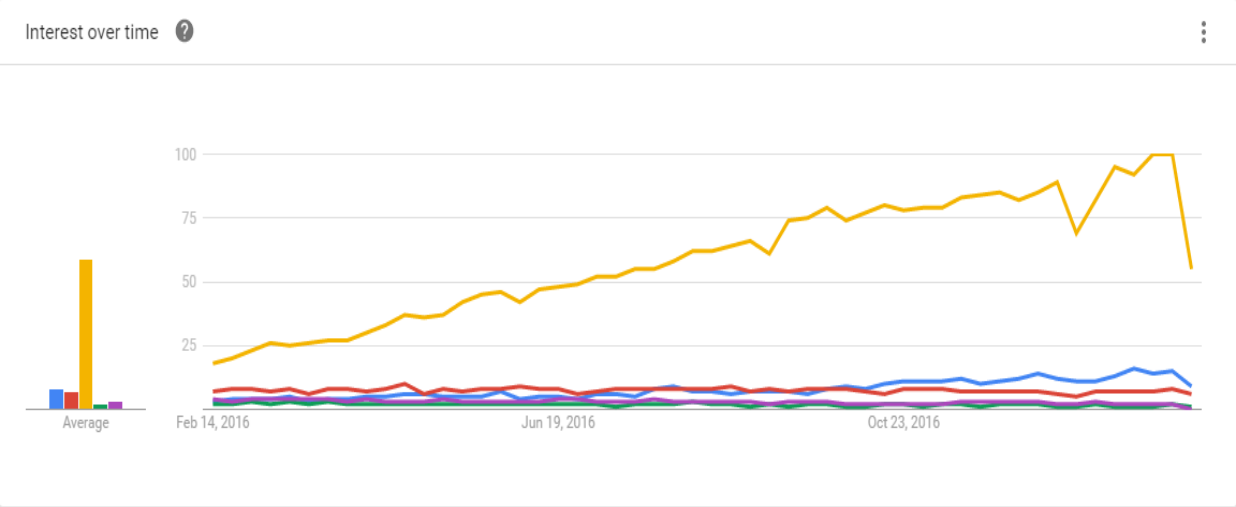
- Criteria; Top zoekopdrachten via de Google search engine.

De reden om alleen Google te gebruiken als search engine is gebaseerd op het feit dat google ongeveer 90% van de markt bezit [69-70].



Figuur 4 - marktaandeel bekende search engines

Hieronder is een chart te zien hoe vaak er wordt gezocht op een bepaald framework.



Figuur 5 - aantal searches per framework

Framework	Color
Vue	
React	
AngularJs/Angular2	
Ember	
Backbone	

Figuur 6 - legenda voor figuur 5.

De chart heeft data van de laatste twaalf maanden, hieruit is te halen dat Angular2 qua searches ver boven te rest uitstijgt met daarbij Vue en React op een tweede en derde plek.

Op deze top 3, namelijk Vue, React en Angular2 is de volgende criteria toegepast.

- Criteria; Activiteit op GitHub.

De reden voor GitHub is, al deze frameworks zijn open Source en zijn te vinden op GitHub. Hier is ook gelijk te zien hoe populair een bepaald framework is, aan de hand van het aantal backers/star ratings. Er is een minimumaantal star ratings opgesteld waaraan de frameworks moesten voldoen. Dit is in overleg met de begeleiders gezet op 10.000.

De Github star ratings zijn opgehaald op 07-02-2017.

Framework	Stars
React.js	59.154
Angular2	19.832
Vue.js	39.933

In de tabel hierboven is te zien dat alle drie de frameworks boven de 10.000 zijn geëindigd en zijn daarom alle drie meegenomen als optie.

4.3.1.1. DE OPTIES

De volgende javascript frameworks kwamen naar voren uit het onderzoek. Hieronder een korte uitleg over deze frameworks. Voor meer informatie over deze frameworks kan er worden gekeken naar de bijlage Adviesrapport javascript frameworks.

- React

React is een library dat is ontwikkeld door Facebook en onder andere door Netflix en Instagram wordt gebruikt. React is het V-gedeelte in een MVC-design pattern en om deze reden zal er gebruik gemaakt moeten worden van externe libraries om de overige onderdelen op te vangen. React maakt gebruik van one-way data binding, JSX als javascript syntax en een virtual DOM.

- Angular2

Angular2 is ontwikkeld door Google en is een framework, dit houdt in dat het de gehele MVC-design pattern gebruikt. Hierop kan een variatie worden toegepast namelijk een MVVM-design pattern. Angular2 maakt gebruik van two-way data binding en heeft de mogelijkheid om Code splitting toe te passen.

- Vue

Vue is ook een framework en maakt ook gebruikt van een MVC-design pattern. Vue wordt onder andere gebruikt door Alibaba en Baidu. Vue maakt gebruik van two-way binding en een virtual DOM.

4.3.1.2. DE KEUZE

Hieronder is het resultaat te zien van het onderzoek per prioritering en het uiteindelijke resultaat welk framework er is gekozen.

- Prioriteit 1 requirements (Must have)

	React	Angular2	Vue
Requirement 1: (1)	Single page/Large app	Large app/Single page	Large app/Single page
Requirement 3: (1)	Externe Library	Native	Externe Library
Requirement 6: (1)	Heel Goed	Heel Goed	Goed

Zowel React als Angular2 scoren op Prioriteit 1 requirements 2 uit 3. Vue scoort op geen 1 prioriteit 1 requirement het beste. Hierdoor is al te concluderen dat Vue niet gekozen zal worden als het javascript framework.

- Prioriteit 2 requirements (Should have)

	React	Angular2	Vue
Requirement 2: (2)	Goed	Goed	Heel Goed
Requirement 4: (2)	Goed	Goed	Goed
Requirement 5: (2)	Externe Library	Externe Library	Externe Library
Requirement 7: (2)	Native	Externe Library	Externe Library

Hier scoort React en Vue scoren op Prioriteit 2 requirements 3 uit 4, waarbij Angular2 uit 4 scoort. Hierdoor heeft React de voorkeur op zowel Angular2 en Vue.

- Prioriteit 3 requirements (Could have)

	React	Angular2	Vue
Requirement 8: (3)	Heel Goed	Goed	Goed

Hier scoort React het beste en omdat de voorkeur al was gezet op React aan de hand van de prioriteit 1 en 2 requirements is er gekozen om React te gaan gebruiken voor de match experience.

Voor meer informatie over het onderzoek of het advies dat hierover is gegeven kan worden gekeken naar de bijlage adviesrapport javascript frameworks.

4.3.2. MVC/FLUX/REDUX DESIGN PATTERN

Tijdens het onderzoek naar de javascript frameworks, was een van de requirements dat het framework een flux of Redux design pattern moest kunnen ondersteunen. Op het moment wordt er een MVC-design pattern toegepast, om deze reden is ook een MVC-design pattern meegenomen als optie. Het front-end moet steeds meer en meer gaan doen. Waaronder server response, cached data, local created data en UI-state. Het laatste is waar zowel Flux als Redux voor is ontwikkeld, het bijhouden van de state van de applicatie. Hieronder een korte uitleg van het MVC-design pattern en een meer detail tredende versie over zowel Flux als Redux design pattern.

4.3.1.1. DE OPTIES

- MVC-design pattern

Een MVC-design pattern is een architecturaal design pattern. Een MVC-design pattern maakt gebruik van bidirectional data flow. Wat betekent dat de data twee kanten op kan. Het kan van het model naar de view, maar ook van de view naar het model. In een MVC-design pattern zitten drie onderdelen, een model, een view en een controller [20-21]. Het model, de view en de controller zijn de basis van een MVC-design pattern. Er kunnen op dit pattern verschillende aanpassingen worden gedaan, waaronder het toevoegen van een Service en Repository laag. Ook kan er gekozen worden om het MVVM-design pattern te gebruiken in plaats van een standaard MVC. Indien er gebruik gemaakt wordt van de standaard drie componenten zal de data-flow gaan zoals hieronder beschreven. De user heeft interactie met de view, de view geeft dit door aan de controller dat er een action is aangeroepen. De controller voert uit wat er onder deze actie gedaan hoort te worden en geeft het model dat bij deze actie hoort. Het model update zich aan de hand van de nieuwe data en geeft dit weer door aan de controller. Waarna de controller, de view de nieuwe datamodel geeft [20-21].

- Flux design pattern

Het Flux design pattern is net zoals React ontwikkeld door Facebook [13]. Het maakt gebruik van React zijn uni directionele data flow om de state te allen tijde te beheren [14]. Dit is met de standaard MVC-design pattern wat veel wordt gebruikt een stuk lastiger om te beheren. De reden hiervoor is, dat een MVC-design pattern gebruik maakt van bidirectional data flow. Het flux design pattern bestaat uit drie onderdelen. Namelijk, de dispatcher, de store(s), actions (events) en de views (React Componenten).

- De view

  De view is een stateless component, het enige wat de views kan en zou moeten kunnen is het renderen van data en actions afvuren [15].

- De actions, Action creators

  De actions worden afgevuurd door de applicatie of door invloed van de gebruiker die een handeling verricht op de view. Een action is een event, een event waardoor de store weet wat het moet doen. Een action kan geen data manipuleren [15].

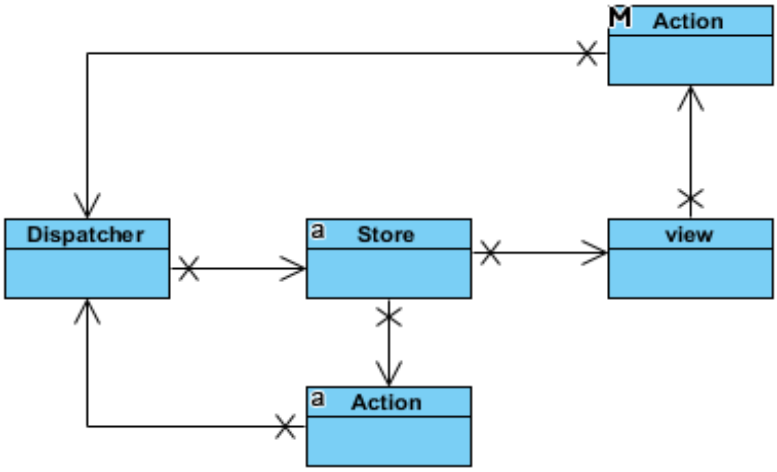
- De store(s)

  In de store(s) is alle business logica en application data te vinden. De store zorgt voor de data manipulatie aan de hand van de action soort die deze binnenkrijgt. Alleen de store(s) weet hoe deze data gemanipuleerd moet worden [15].

- De dispatcher

  De dispatcher is de lijm tussen de view en de store. De dispatcher zorgt ervoor dat de actions die door de gebruiker of door het systeem worden aangeroepen op de juiste manier worden doorgegeven aan de stores [15].

Hieronder is een diagram te zien met daarin de interacties die een flux design pattern heeft.



Figuur 7 - flux design pattern

Het flux design pattern werkt als volgt. De user heeft interactie met de view, de view vraagt aan de action creator een action object aan. De action creator maakt een action object en geeft deze action object door aan de dispatcher. De dispatcher zendt daarna de action object door aan de stores, ook aan de stores die niks met deze action object gaan doen. Wanneer de store klaar is met het aanpassen van de state aan de hand van het action object zal deze het terug geven aan de view. De view en de childviews re-renderen zichzelf aan de hand van de nieuwe state [15].

- Redux design pattern

Het Redux design pattern is ontwikkeld door Dan Abramov [12]. Redux is een verdere ontwikkeling van de eerdergenoemde Flux design pattern. Redux probeert net zoals Flux het state probleem op te lossen. De reden waarom dat Redux is ontwikkeld is, omdat er in Flux bepaalde design keuzes zijn dat ervoor zorgt dat het moeilijk is om uitbreidbaar te blijven [16]. Een voorbeeld dat wordt gegeven is dat flux inheritance over composition gebruikt. Om deze reden zit het Redux design pattern iets anders in elkaar ten opzichte van het Flux design pattern. In Redux wordt er gebruik gemaakt van, actions, een store, reducers, view layer binding en de views (React Componenten).

- Views

  De view is hetzelfde gebleven als bij het Flux design pattern. Ook hier is de view een stateless component, het enige wat de views kan en zou moeten kunnen is het renderen van data en actions afvuren [17].

- Actions, Action creators

  De actions, oftewel de events is iets anders als bij de Flux design pattern. Een action in een Redux design pattern verzendt niet de action naar de dispatcher, inplaats hiervan returned de action een formatted action(state) object [17].

- Store

  De store is ook anders dan in een Flux design pattern. In een Redux design pattern is er maar één store. De store in een Redux design pattern zorgt ervoor dat de taken worden doorgegeven aan de (root)reducer. In een flux design pattern bestaat er een dispatcher, de dispatcher is nu in de store samengevoegd en de store kan dus nu alles wat een dispatcher voorheen kon. De store kan door deze veranderingen geen data meer manipuleren, dit is overgezet naar de reducer(s) [17].

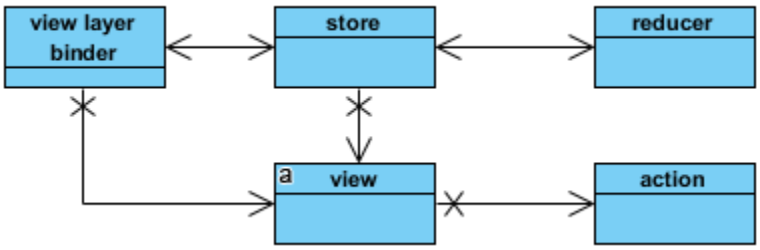
- Reducer(s)

  De reducer(s) krijgt vanuit de store een action object. Aan de hand van deze action object zal de root reducer deze action object doorgeven aan alle andere reducers, ook aan de reducers die niks met deze action object gaan doen. Een ander belangrijk punt is dat een reducer nooit het action object aanpast, maar altijd een copy maakt van het action object en deze aanpast. De reducers zijn als een boom met elkaar verbonden, waarbij de root reducer de wortel is van deze boom [17].

- View layer binding

  De view layer binding zorgt ervoor dat de view tegen de store kan praten aan de hand van action objects [17].

Redux zijn data flow is door al deze aanpassingen ook iets anders dan die van een Flux design pattern. Hieronder is een diagram te zien met daarin het Redux design pattern.



Figuur 8 - redux design pattern

Redux zijn data flow gaat als volgt, De gebruiker heeft interactie met de view. De view vraagt aan de action creator een action object aan. De action creator returned een action object terug naar de view. De view geeft deze action object door aan de store die deze doorgeeft aan de rootreducer. De rootreducer geeft deze dan weer door aan alle andere reducers. Wanneer alle reducers klaar zijn met de action object, wordt er een nieuw action object teruggegeven aan de root reducer. Als dit is gedaan geeft de root reducer deze action object weer terug aan de store. De store zegt tegen de View layer binder dat er een nieuwe action object is. Waarna de View layer binder vraagt aan de store om deze action object te geven. De View layer binder triggers een re-render van de view, waarna deze aan de hand van de nieuwe action object de view update [17].

4.3.1.2. DE KEUZE

Er is uiteindelijk gekozen voor het Redux design pattern. Dit had meerdere redenen. Als eerste is er een groot verschil tussen MVC en Flux/Redux. Het grootste verschil tussen deze twee keuzes is vooral het werk van uni of bi-directionele data-flow. Om deze reden moest er een keuze gemaakt worden tussen MVC of Flux/Redux. De keuze is gevallen op het gebruik van een Flux/Redux design pattern. Dit omdat bi-directionele data-flow voor vele problemen kan/zal zorgen wanneer de applicatie groter wordt en meerdere views hetzelfde model gaan gebruiken. Uiteindelijk is het mogelijk dat met een MVC pattern, en dus een bi-directionele data flow, er een mismatch bestaat tussen verschillende views met hetzelfde model. Nu MVC is weggevallen was de keuze Flux of Redux. Redux heeft een paar voordelen ten opzichte van een Flux design pattern. Een van deze voordelen is dat de locatie waar wat gedaan wordt in de applicatie overzichtelijker en duidelijker is. Ook is het implementeren van een Redux design pattern makkelijker toe te passen [16]. Dit omdat het hergebruiken van functionaliteiten binnen de applicatie, met een Redux design pattern, zonder problemen kan, terwijl hergebruiken binnen een Flux design pattern zorgt voor duplicate code of andere code smells [16]. Daarnaast kan Redux gebruik maken van Hot-reloading [18-19]. Hot-reloading zorgt ervoor dat alleen de files die veranderd zijn worden geüpdatet. Ook heeft het een grote community achter zich wat ervoor zorgt dat het makkelijk is om hierover informatie te vinden.

4.4. TOOLS

In dit hoofdstuk zullen de tools worden besproken die voor de eerste sprint van de afstudeerperiode bekend waren. Alle andere tools zullen in hun respectievelijke sprint hoofdstuk worden beschreven. Dit om de focus te leggen op de sprints.

4.4.1. IDE

Een belangrijk onderdeel van het ontwikkelen van een applicatie is de juiste keuze voor een IDE, oftewel Integrated development environment. In sectie 4.4.1.1 is te zien uit welke IDE's is gekozen en in sectie 4.4.1.2 is te zien welke IDE uiteindelijk is gekozen en waarom.

4.4.1.1. DE OPTIES

Ook voor IDE's zijn er tientallen mogelijkheden. De opties zijn verkleind tot vier IDE's die vaak naar voren kwamen als er werd gezocht op "Javascript IDE" in google.

- Atom

Atom is een free tekst-editor gemaakt door GitHub. Het is gemaakt in Node.Js en kan gebruikt worden op zowel Mac, Linux en Windows. Atom ondersteunt onder andere debugging, git, syntax highlighting en code completion. Naast Javascript ondersteunt Atom onder andere Less, SQL, CSS en C# [22-23].

- Sublime Text

Sublime is een free tekst-editor dat gebruikt kan worden als IDE. Sublime text is ontwikkeld door Jon Skinner [24-25]. Ook Sublime Text is cross platform te gebruiken en door verschillende plug-ins te downloaden zijn zowel alle talen te ondersteunen. Sublime Text ondersteunt syntax highlighting en code completion by default.

- Visual Studio Code

Visual Studio Code, niet te verwarren met Visual Studio is een source code editor. Visual Studio Code is ontwikkeld door Microsoft en is bedoeld voor het front-end development [26-27]. Visual Studio Code ondersteunt debugging, Git, syntax highlight en code completion.

- WebStorm

WebStorm is een IDE dat gemaakt is door JetBrains. Andere producten die JetBrains heeft gemaakt zijn onder andere, TeamCity, Resharper en IntelliJIDEA. WebStorm is een betaalde IDE, waarbij het mogelijk is om de eerste maand gratis het te proberen. WebStorm is net zoals alle vorige editors cross platform en ondersteunt het onder andere Javascript, Typescript, Less en React. WebStorm ondersteund daarnaast Git, SVN, debugging, syntax highlighting en code completion [28-29].

4.4.1.2. DE KEUZE

Al snel werd duidelijk dat Sublime Text, hoewel het gebruikt kan worden, te kort schoot. De overige 3 IDE's kunnen uiteindelijk veel meer dan Sublime Text zonder er add-ons voor te hoeven gebruiken. Uiteindelijk had WebStorm mijn voorkeur om de IDE te worden voor dit project. WebStorm ondersteunt namelijk als enige de JSX-syntax die React gebruikt. WebStorm had alleen een groot nadeel en dat is dat het een betaalde IDE is. Helaas is er om die reden dan ook voor gekozen om geen gebruik te maken van WebStorm. Dit hield in dat er nog keuze was uit twee IDE's, namelijk Visual Studio Code en Atom. De huidige situatie draait volledig op typescript en Visual Studio Code support typescript native. Dit was de reden waarom er is gekozen is om gebruik te maken van Visual Studio Code.

5. SPRINT 1 REQUIREMENTS, MODULES, DIAGRAMMEN EN BEGIN PROJECT OPZET.

In de tabel hieronder zijn de sprintdoelen te zien.

Planning huidige sprint
Requirements opstellen match experience. Klassendiagrammen structuur applicatie. Opzet project. (Begin applicatie). Testen.

5.1. REQUIREMENTS

Als eerste is er gewerkt aan het opstellen van de requirements. Dit is zoals eerder vermeld gedaan door interviews te houden met de product owner, de begeleiders en de software developers van team Web. De meeste requirements waren van tevoren al bekend, dit komt omdat de match experience al bij zowel iOS als bij Android geïmplementeerd is. Hieronder zijn de requirements beschreven die belangrijk waren voor deze sprint. Voor een volledige lijst van alle requirements die zijn opgesteld kan worden gekeken naar de bijlage Requirements Match experience.

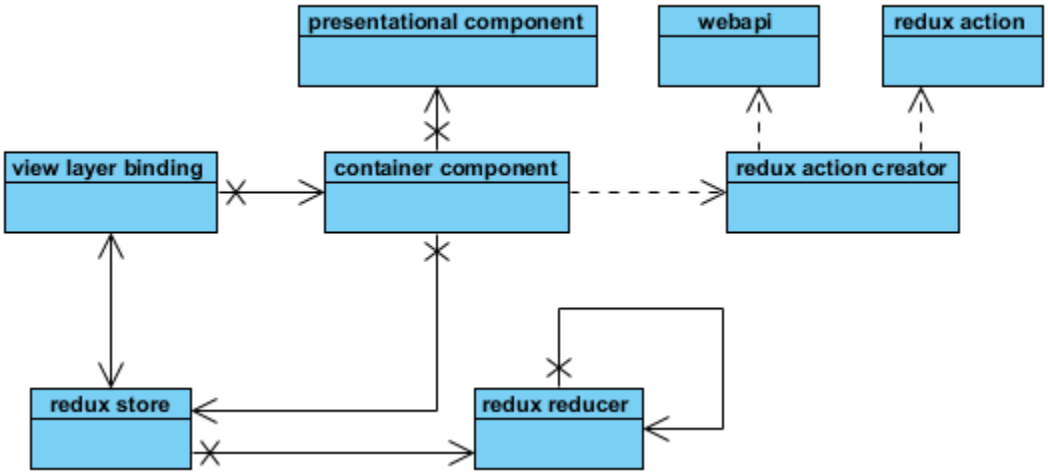
- Het systeem moet meerdere views ondersteunen/weergeven. (Functionele Requirement)
    - Timeline
    - Studio
    - Penalty's
- Het systeem moet responsive zijn. (Niet-functionele Requirement)
- Het systeem moet in een single page framework gebouwd worden. (Niet-functionele Requirement)
- Het systeem moet match experience data kunnen bijhouden. (Functionele Requirement)

5.2. DIAGRAMMEN

Om te bepalen hoe de structuur er uit gaat zien van het project is het van belang om een idee te krijgen hoe de applicatie eruit gaat zien en wat er allemaal in voor moet komen.

Als eerste is er een diagram gemaakt over hoe de applicatie zal moeten gaan samenwerken met de huidige API. Hieronder is het diagram te zien waarbij de applicatie op een abstracte manier is weergegeven.

5.2.1. ANALYSE KLASSENDIAGRAM



Figuur 9 - eerste opzet structuur applicatie

Voor meer informatie over de componenten redux store, redux reducer en redux action kan er worden gekeken naar hoofdstuk 4.3.2. De overige twee componenten staan hieronder nader uitgelegd.

De container component is waar de logica van de view zal komen, hier zal alle data manipulatie plaatsvinden.

De presentational component is zoals de naam het al zegt een presentational view, het enige wat deze component weet is wat het voor data moet weergeven en waar. Hoe het component aan zijn data komt weet het niet. Het is zoals deze regelmatig wordt genoemd, een dumb component.

Als laatste de web-Api, deze bestaat al en zal alleen maar worden aangeroepen om A sync calls op uit te voeren. De web-Api zal worden gebruikt om de match experience data op te halen of de teksten van de commentatoren tijdens de wedstrijd.

5.3. PROJECT OPZET

Om alle requirements te behalen is er meer nodig dan alleen React, dit omdat React alleen een view library is. Zoals eerder vermeld in het onderzoek naar javascript frameworks was een van de requirements dat het een Redux of Flux pattern moest kunnen ondersteunen. Om de business logica op de juiste plek te zetten zal er gebruik gemaakt worden van een Redux design pattern. Naast Redux zijn er nog andere externe libraries gebruikt om de applicatie zo goed mogelijk te laten werken. Tijdens dit hoofdstuk zullen deze externe libraries worden uitgelegd en uiteindelijk zal er een conclusie komen over deze libraries en waarom er is gekozen voor een bepaalde library. Voor de volgende onderdelen zijn externe libraries gebruikt.

- Test framework
- Package manager
- Module bundler
- Transpiler
- A sync calls

Voor elk van deze externe libraries zijn meerdere opties mogelijk, als eerste zal de package manager nader worden uitgelegd en de opties worden weergegeven.

5.3.1. PACKAGE MANAGER

Indien er gewerkt wordt aan een front-end project is het gebruikelijk dat er gebruik gemaakt wordt van een package manager. Een package manager zorgt ervoor dat het installeren en het updaten van project dependencies eenvoudig kan worden gedaan. Wat de package manager doet is, het download, pakt de package/modules uit en kopieert deze files in een (node modules) map waarin het project staat. Hierdoor kan er op een makkelijke manier worden samengewerkt met meerdere developers. Dit komt, omdat er maar één file is waarin alle dependencies staan van een geheel project. Op deze manier zorgt de package manager ervoor dat alle developers dezelfde versies van de packages/modules hebben. Hieronder is een voorbeeld te zien van een package.json file die gebruikt kan worden door beide package managers, die nader zijn onderzocht.

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "react": "^15.4.2",
    "react-dom": "^15.4.2",
    "react-redux": "^5.0.2",
    "redux": "^3.6.0"
  }
}
```

Figuur 10 - package.json file met react en redux dependencies

5.3.1.1. DE OPTIES

De volgende package managers zijn nader onderzocht en uiteindelijk is hier een keuze uit gemaakt welke het beste past bij dit project.

- NPM

NPM is ontwikkeld door Isaac Z. Schlueter. NPM is de default package manager voor de javascript runtime environment Node.JS. NPM is een command line client, dat connectie maakt met een remote registry. De remote registry is open source en er kan om die reden door iedereen, wanneer hier goedkeuring voor is gegeven, modules op worden geregistreerd. NPM is waarschijnlijk een van de bekendste als niet de bekendste package manager voor javascript front-end development [30].

- Yarn

Yarn is ontwikkeld door Facebook. Net zoals NPM zorgt Yarn ervoor dat de developer packages kan installeren door gebruik te maken van het package.json file. Een van de grote verschillen dat Yarn heeft ten opzichte van NPM is dat Yarn een cache gebruikt. Waardoor het mogelijk is om packages offline op te halen, indien deze eerder een keer zijn opgehaald [31-32].

Naast bovenstaande zijn er nog meer package managers, waaronder Bower een grote speler is. Deze is vanwege tijd niet meegenomen in de opties.

5.3.1.2. DE KEUZE

Er is gekozen voor NPM. NPM heeft zich al redelijk wat jaren bewezen als een van de belangrijkste package managers. Dit heeft als voordeel dat er veel informatie over te vinden is, wat de kans groter maakt dat, indien er een probleem is, dit probleem al door meerdere andere developers is ontdekt en opgelost. Een andere reden is, dat er eerder is gewerkt met NPM en nog niet met Yarn. Dit zorgt ervoor dat hier minder tijd wordt besteed aan het ontdekken hoe een package manager werkt en meer tijd kan worden besteed aan het opleveren van een werkend prototype.

Zoals beschreven bij hoofdstuk 6.1.1 de opties is er nog een andere package manager die net zoals NPM en Yarn zeer bekend is, namelijk Bower. Indien Gamebasics besluit om verder onderzoek te doen naar welke package manager er zou moeten worden gebruikt, raad ik aan om ook Bower in het onderzoek mee te nemen.

5.3.2. MODULE BUNDLER

Module bundlers zijn net zoals package managers een vereiste wanneer er gewerkt wordt aan het front-end. Een Module bundler zorgt ervoor dat alle script files die gebruikt worden door de webpagina in een file worden gezet. Het voordeel hiervan is, dat er maar een <script> </script> tag gebruikt hoeft te worden, waarin deze bundle staat. Dit zorgt ervoor dat de browser maar een script file hoeft in te laden, terwijl er zonder de module bundler meerdere script files ingeladen had moeten worden en de browser laad deze files een voor een in. De module bundler zorgt dus voor een enorme performance boost, wanneer er gebruik gemaakt wordt van meerdere script tags. Een ander voordeel dat een module bundler met zich mee brengt is, het maakt een minified version van de script file. Dit houdt in dat het alle comments, white spaces, new lines, etc. weghaalt, waardoor de file minder wordt qua grootte [33-34].

5.3.2.1. DE OPTIES

Voor de module bundlers waren er ook meerdere keuzes en ook hier is er weer een selectief aantal module bundlers onderzocht [36].

- Webpack

Webpack zijn eerste release kwam in 2012 en is om die reden, op het moment van schrijven, nog maar vijf jaar oud. Webpack kan cross platform gedraaid worden en is een open Source project. Webpack maakt, indien de gebruiker dit wil, gebruik van Code splitting. Code splitting zorgt ervoor dat alleen bepaalde delen van de applicatie wordt ingeladen, wat voor performance boosts kan zorgen [35].

- Browserify

Browserify heeft zijn eerste release gedaan in 2011, Browserify is net zoals Webpack een module bundler dat cross platform gedraaid kan worden. Ook Browserify is een open source module

bundler. Browserify is gebaseerd op het programmeren met gebruik van Node.js en werkt om die reden alleen op Node.js projecten [39-40].

- Gulp

Gulp is ontwikkeld door Fractal Innovations en ook Gulp kan cross platform gedraaid worden. Gulp is een task-runner dat gebaseerd is op Node.js en niet op NPM. Gulp maakt gebruik van een CLI, oftewel Command Line Interface en heeft een package.json en gulfile.js nodig [38].

Net zoals bij de package managers zijn ook hier nog een aantal module bundlers die nader onderzocht kunnen worden. Met name jspm en require.js [36].

5.3.2.2. DE KEUZE

De verschillen tussen die module bundlers waren klein, maar er moet een keuze gemaakt worden. Gulp wordt op het moment door het bedrijf Gamebasics gebruikt en heeft daar tot nu toe voor weinig tot geen problemen veroorzaakt. Daarnaast is Gulp qua static loading vele malen sneller dan de andere twee module bundlers. Browserify heeft als groot voordeel dat het makkelijk te begrijpen is, dit omdat de Api zeer goed gedocumenteerd is en de layout van een Browserify file voelt heel natuurlijk aan [37]. Webpack heeft als groot voordeel het mogelijk maken van code splitting, wat een grote performance boost kan geven aan de dynamische laadtijden van de applicatie. Ook heeft Webpack de optie voor hot-reloading, wat front-end development qua snelheid een enorme boost zal geven [37]. Hoewel Webpack ook een nadeel heeft en dat is namelijk, dat de learning curve van Webpack zeer hoog is [37].

De module bundler die gebruikt zal worden tijdens dit project is Webpack, dit komt vooral door de voordelen die hierboven zijn benoemd. Het grote nadeel is met de koop toe genomen.

5.3.3. TEST FRAMEWORKS

Aangezien React op zichzelf geen test framework is, of heeft, moet er gebruik worden gemaakt van een test framework.

5.3.3.1. DE OPTIES

- Mocha - Chai – Sinon

Mocha is een test framework dat vooral gebruikt wordt voor automated unit tests [41-42]. Zowel Mocha, Chai als Sinon zijn open source projecten. Chai is een assertion test framework dat de vergelijkingen doet [43]. Sinon maakt het mogelijk om spies toe te voegen en stubs te maken [44]. Deze drie frameworks bij elkaar zorgen voor een totaalpakket van een test framework.

- Jasmine

Jasmine is net zoals de combinatie hierboven een open source project. Jasmine is een test framework waar alles inzit. Het is dus niet nodig om meerdere libraries te gebruiken. Jasmine kan cross platform worden gedraaid en wordt vooral gebruikt voor unit testing [45-46].

5.3.3.2. DE KEUZE

De keuze voor een test framework bestond vooral uit het kiezen tussen of een framework dat alles kan, of gebruik maken van verschillende onderdelen. Het grote voordeel van Mocha is vooral dat je de keuze hebt in welke assertion en welke stub/spy framework je gaat gebruiken. Dit zorgt ervoor dat indien dit niet nodig is je deze libraries niet hoeft toe te voegen. Wat een voordeel is voor Mocha, kan ook gezien worden als een nadeel voor Mocha, namelijk dat alles gedaan moet worden door externe libraries binnen het test framework. Dit zorgt ervoor dat er meerdere libraries up-to-date moet worden gehouden. Uiteindelijk is er toch gekozen om Mocha-Chai-Sinon te gaan gebruiken en ook daadwerkelijk in deze vorm. De reden was dat Mocha-Chai-Sinon veel voorbeelden heeft in combinatie met React en dit voor een snellere workflow kan zorgen.

5.3.4. TRANSPILER + A SYNC + TEST ENVIRONMENT

Voor de volgende punten is geen onderzoek gedaan. Dit i.v.m. tijd, maar ook omdat er tijdens het onderzoeken van zowel React als externe libraries naar is gekeken. Een andere reden waarom er niet al te veel tijd is in gestoken is, omdat tijdens het onderzoek naar het javascript framework erin bijna alle gevallen gebruik werd gemaakt van de onderstaande externe libraries.

- Transpiler

Als transpiler is er gebruik gemaakt van Babel [48]. Een transpiler is nodig om een grotere range aan browsers te ondersteunen. React maakt bijvoorbeeld gebruik van JSX, daarnaast zal er gebruik gemaakt worden van Typescript of es6 wat beide terug gezet moet worden naar es5 om zo bijvoorbeeld IE8 te blijven ondersteunen.

- A sync

De data die opgehaald moet worden voor de match experience zal A sync moeten worden opgehaald. Standaard gaat React + Redux hier niet mee om dus moet er gewerkt worden met Await of Promises. Om dit om te vangen en dus Await en Promises te ondersteunen is er gebruik gemaakt van isomorphic-fetch [47]. Deze library zorgt ervoor dat de web-Api call dat in Json format wordt ontvangen wordt opgevangen in een Promise.

- Test Environment

Om de front-end onderdelen te testen, zoals een button of een ordered list moet de html worden gesimuleerd. Dit wordt gedaan door JSDOM. JSDOM zorgt ervoor dat er niet een gehele pagina hoeft te worden gebruikt, maar zorgt ervoor dat er een html pagina wordt gesimuleerd, waarin bijvoorbeeld een button aan kan worden toegevoegd die getest kan worden.

5.4. CONCLUSIE

Tijdens deze sprint is er gewerkt aan het houden van interviews en het opstellen van de requirements voor de match experience. Ook is er gewerkt aan het opstellen van klassendiagrammen om de structuur van de applicatie weer te geven. Als laatste is er gewerkt aan het opzetten van het project wat veel onderzoek naar externe libraries inhield.

Naast de activiteiten die hierboven staan beschreven is er een gesprek en planning geweest met de begeleiders en met de product owner. Uit dit gesprek zijn de volgende punten naar voren gekomen;

- CSS-structuur niet goed naar gekeken
- Te snel in de diepte willen gaan, vaker even een stap naar achteren zetten en opnieuw de situatie bekijken.
- Overige onderdelen van de structuur, naast CSS, goed naar gekeken, goede uitleg waarom er voor bepaalde libraries is gekozen.
- Onderzoek javascript frameworks was in orde.

Deze sprint is uiteindelijk goed verlopen naast de punten die uit het gesprek zijn gekomen, zoals de CSS-structuur zijn voor de rest alle punten behaald. De planning voor sprint 2 is hieronder te zien.

Planning volgende sprint
CSS-structuur bekijken
Structuur componenten nader bekijken, nadat de CSS is uitgedacht
Match Experience data ophalen, denk hierbij aan Generiek zijn. (JSON-format).
Testen functionaliteiten, Unit testen.
Verslag schrijven.

6. SPRINT 2 CSS STRUCTUUR, APPLICATIE STRUCTUUR

In de tweede sprint is er verder gegaan over wat er is besproken in Sprint 1, namelijk de CSS-structuur nader bekijken en hiervoor net zoals voor de rest van de applicatie een goede structuur opzetten. Ook zal er worden gekeken naar hoe de data zal worden opgehaald door gebruik te maken van het Redux-design pattern en A sync calls. Als laatste zullen er testen worden gemaakt over de functionaliteiten die worden toegevoegd.

6.1. REQUIREMENTS

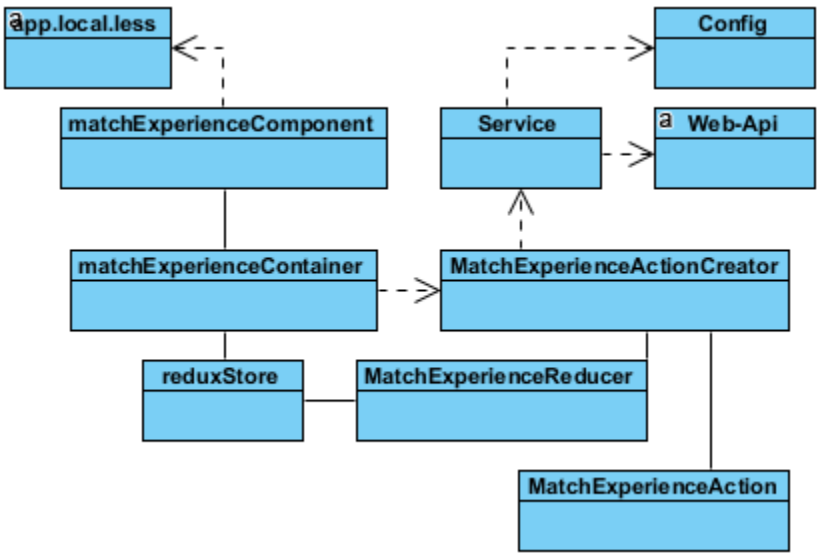
De volgende requirements zijn voor deze sprint van belang;

- Het systeem moet Json format data kunnen ophalen van de bestaande web-API. (Niet-functionele Requirement)
- Het systeem moet responsive zijn. (Niet-functionele Requirement)
- Het systeem moet match experience data kunnen bijhouden. (Functionele Requirement)

6.2. DIAGRAMMEN

Tijdens deze sprint is er onder andere het ophalen van de match experience data via een Redux design pattern toegepast en is de CSS-structuur opgezet. Hieronder zijn de diagrammen te zien die tijdens deze sprint zijn gemaakt.

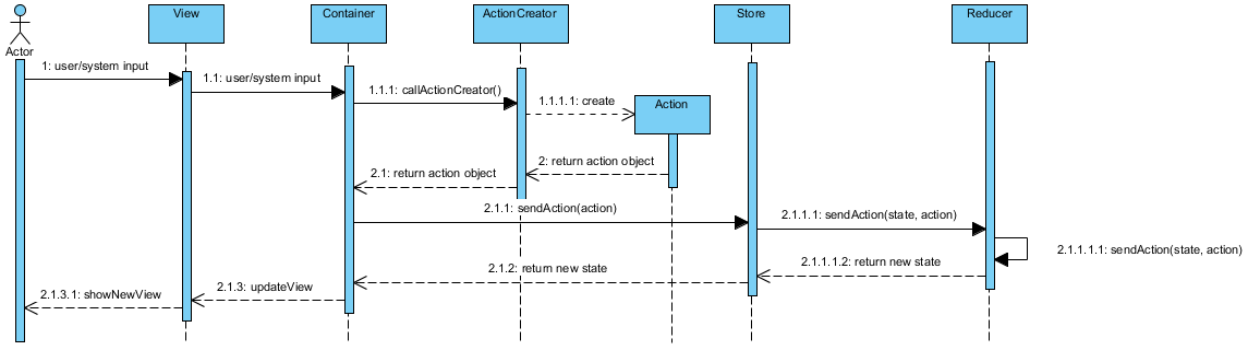
6.2.1. ANALYSE KLASSENDIAGRAM



Figuur 11 - analyse klassendiagram match experience

Tijdens het opzetten van de applicatie zijn er twee aanpassingen gedaan aan de hand van het klassendiagram die te zien is in hoofdstuk 5.2. Er zijn twee classes bij gekomen, namelijk de service en config class. Hierover meer in hoofdstuk 6.5.

6.2.2. SEQUENCE KLASSENDIAGRAM



Figuur 12 - data flow match experience

Naast een klassendiagram was het belangrijk om de data flow weer te geven. De flow in de applicatie is namelijk voor elke state chance hetzelfde, het enige grote verschil is de data die wordt aangepast en welke calls er worden gemaakt. Om de data flow goed weer te geven is er van de matchdata een sequence diagram gemaakt. Hierboven is het diagram te zien waarin de data flow voor de matchdata wordt weergegeven deze kan worden gebruikt voor elke state change binnen de applicatie.

6.3. CSS SYNTAX

Tijdens de vorige sprint is er naar voren gekomen dat er te weinig aandacht is besteed aan het onderzoeken van de CSS-structuur. Dit was dan ook het eerste punt waar er tijdens deze sprint de focus op is gelegd. Na een twee dagen onderzoek werd het duidelijk dat er een keuze gemaakt moest worden tussen welke CSS-syntax er zou worden gebruikt.

6.3.1. DE OPTIES

Er zijn tijdens het onderzoek drie opties naar voren gekomen die veel gebruikt worden in een front-end javascript website.

- CSS

CSS staat voor Cascading Style Sheets [49-50]. In een CSS wordt beschreven hoe de HTML-elementen moeten worden weergegeven op het scherm. Er wordt gebruik gemaakt van een markup language [51]. Net zoals er in een applicatie gebruik gemaakt wordt van abstractie is CSS dat voor de view. In de CSS-file wordt de styling weergegeven en in de view zelf wordt de document presentation weergegeven.

- LESS

LESS is een CSS pre-processor wat betekent dat het een extensie is van de standaard CSS [52-53]. LESS voegt de mogelijkheid toe om gebruik te maken van variable, mixins en functies. De bedoeling hiervan is, dat de style sheets beter kunnen worden onderhouden en kunnen worden aangepast [52-53]. LESS is ontwikkeld voor Node.Js maar kan ook buiten Node.Js gebruikt worden.

- SASS

SASS is net zoals LESS een pre-processor. Sass is ontwikkeld door Hampton Catlin en Natalie WeizenBaum [54-55]. Ook SASS voegt de mogelijkheid toe om gebruik te maken van variable, mixins en functies. SASS is ontwikkeld op de Ruby module.

6.3.2. DE KEUZE

Nadat de opties nader waren bekeken was er geen reden om niet LESS of SASS te gebruiken. Beide voegen zoveel extra mogelijkheden toe, zoals functies en variables, dat het schrijven van styling veel overzichtelijker en beter te onderhouden wordt. De keuze moest komen uit het gebruik van LESS of SASS. Er is uiteindelijk gekozen voor LESS, LESS had twee voordelen ten opzichte van SASS voor dit project. Het eerste voordeel is, dat in het Web-project, dat actief wordt gebruikt door Gamebasics, al gebruik gemaakt wordt van LESS. Het tweede voordeel is, wat ook kan worden gezien als een nadeel voor SASS, dat SASS gebruik maakt van Ruby, wat betekent dat de Ruby module moet worden toegevoegd. Om deze twee redenen, waarbij de eerste de grootste reden was, zal er gebruik gemaakt worden van LESS in de match experience.

6.4. CSS MODULES EN BOOTSTRAP

Een andere keuze die gemaakt moest worden was of er gebruik gemaakt zou worden van Bootstrap in combinatie met CSS-modules.

Een van de requirements die was opgesteld voor de match experience was dat de applicatie responsive moest zijn. Om dit toe te passen, zonder zelf heel veel werk te doen is er gebruik gemaakt van Bootstrap [56-57]. Bootstrap is een library waarin alle default styling elementen, zoals H1, H2, LI, etc. al gestyled zijn voor verschillende resoluties. Het gebruik van Bootstrap zorgt voor zoveel tijdwinst, dat het bijna niet te doen is om geen gebruik te maken van Bootstrap.

CSS-Modules zorgt ervoor dat alle CSS/LESS/SASS-files die worden gemaakt een local scope krijgen [58] in plaats van een global scope, zoals dat bij bootstrap het geval is. Dit heeft voordelen en nadelen. Het grote voordeel hiervan is dat er geen conflicten meer zijn tussen verschillende html elementen met dezelfde class name. Een ander voordeel is, dat er geen gebruik meer hoeft, en daarom geen gebruik meer mag, gemaakt worden van de “!important” tag. Wat het voordeel is van CSS-Modules is ook gelijk het nadeel. Het zorgt ervoor dat elke class een aparte class wordt, dit wordt gedaan door het toevoegen van een hash. Dit kan ervoor zorgen dat het testen van de stylingsheets een ingewikkelde taak kan worden, omdat er niet meer duidelijk is welke hash voor welk element staat.

Er is uiteindelijk gekozen om gebruik te maken van zowel Bootstrap als CSS-Modules. Dit omdat beide voordelen hebben die ervoor zorgen dat de applicatie leesbaarder en onder houdbaarder wordt. De volgende code conventie is toegepast. In de Bootstrap files staan de global variables die worden gebruikt om de applicatie responsive te maken en de local LESS-files worden gebruikt om de styling toe te voegen aan de html elementen. Aan de bootstrap files wordt in principe niks aangepast.

6.4.1. TOEPASSING

Om de eerder gekozen CSS-Modules en bootstrap te gebruiken zijn er externe libraries nodig. Deze libraries moeten ervoor zorgen dat de LESS-format wordt omgezet naar standaard CSS. In totaal zijn er hier zes loaders voor nodig.

- Less-Loader

De Less-Loader zorgt ervoor dat de LESS-format wordt omgezet naar CSS-format [61].

- PostCss-Loader

De PostCss-Loader zorgt ervoor dat bepaalde onderdelen die de CSS-Loader niet kan omzetten, worden genegeerd. De CSS-Loader kan bijvoorbeeld geen // comment tag lezen en zal hier een error op geven. De PostCss-Loader zorgt ervoor dat de // worden genegeerd [62].

- URL-Loader

De URL-Loader zorgt ervoor dat variable en files met de extensie svg, png, jpg en vele andere extensies worden omgezet naar een leesbaar format [63].

- File-Loader

De File-Loader zorgt ervoor dat files met de extensie ttf en eot kunnen worden omzet naar een leesbaar format [64].

- CSS-Loader

De CSS-Loader bekijkt en zorgt ervoor dat de CSS dat de vorige loaders hebben gemaakt leesbare CSS is dat voldoet aan alle CSS-standaarden [65].

- Style-Loader

De Style-Loader zorgt ervoor dat de file die de CSS-Loader heeft gemaakt kan worden toegepast op de html op de pagina [59].

Het is helaas niet voldoende om alleen deze libraries toe te voegen via NPM. Er moet namelijk worden geconfigureerd hoe deze loaders met elkaar omgaan. Het is namelijk van belang dat de volgorde van deze loaders kloppen. Aangezien er gewerkt wordt met LESS is het van belang dat als eerste de LESS-Loader de files omzet. De volgorde die van belang is in dit project is de volgende;

Less-Loader -> PostCss-Loader -> CSS-Loader -> Style-Loader. De overige twee loaders, namelijk de file en URL-Loader hebben geen invloed op de volgorde, zolang ze maar wel staan beschreven hoe deze moeten werken. Deze beschrijving zal gedaan worden in de webpack.config.

Hieronder is een voorbeeld te zien van hoe de URL-Loader is beschreven in de webpack.config.

```
{
    test: /\.((woff2?|svg)(\?v=[0-9]\.[0-9]\.[0-9]))|(woff2?|svg|jpe?g|png|gif|ico)$/,
    loader: 'url-loader?limit=10000'
},
```

Figuur 13 - URL-Loader webpack.config example

Een ander punt dat geconfigureerd moet worden is het verschil tussen bootstrap files en CSS-modules, oftewel local files. Bij default ziet de loaders geen verschil tussen de bootstrap Less files en de eigen gemaakte Less files. Dit zorgde ervoor dat ook alle bootstrap variables een hash kregen, zie hoofdstuk 7.2 voor meer informatie. Om deze hash te voorkomen in de bootstrap files is er een code conventie toegepast samen met een extra configuratie voor deze code conventie. De code conventie die van toepassing is op alle local files moeten vanaf nu de volgende syntax hebben; [name]_local_[extensie]. Doordat het woord local is toegevoegd weten de loaders nu dat hier een hash op gezet moeten worden en deze hash wordt nu niet gezet op de bootstrap files.

6.5. MATCH EXPERIENCE DATA

Het laatste wat gedaan moest worden qua implementatie is het ophalen van de match experience data. Dit was ook gelijk een goede test om te kijken hoe Redux werkt met betrekking tot de match experience.

6.5.1. TOEPASSING

Als eerste moest er worden gekeken hoe de match experience data kan worden opgehaald. Gamebasics heeft gelukkig een website [60] waarop de gehele web-Api staat uitgelegd en ook welke calls er gedaan kunnen worden. Nu duidelijk was welke calls er allemaal kon worden gedaan was het noodzaak om te bekijken welke calls er nodig waren voor de match experience, aangezien de match experience in zowel Android als voor iOS al was gemaakt was de makkelijke keuze om even langs deze teams te gaan om na te vragen welke calls hun hebben gemaakt. Daar kwamen in totaal twee calls uit waar de gehele match experience mee gemaakt kon worden. De eerste call is een call om de matchdata op te halen, de tweede call is om de commentatoren teksten op te halen. In deze sprint is aan de slag gegaan met de eerste call namelijk de matchdata.

Zoals eerder beschreven heeft het Redux-design pattern drie verschillende componenten, namelijk de action, de reducer en de store. Om het Redux-design pattern toe te passen is er begonnen met de action component. Tijdens het werken met het Redux design pattern en in het specifiek de A sync call kwam eruit dat het een groot voordeel zou leveren als de fetch geabstraheerd zou worden. Om deze reden zijn de volgende twee componenten toegevoegd, config.js en service.js.

- Config.js

De config component is toegevoegd om onderscheid te kunnen maken tussen productie en development omgeving. De productie en development verschillen in sommige opzichten namelijk qua data. De productieomgeving moet bijvoorbeeld een andere Web-Api URL aanroepen dan de development omgeving. Dit is allemaal vast gelegd in de dev.config.json en prod.config.json file. De config.js file leest aan de hand van de omgeving een van deze twee files uit.

- Service.js

De service.js laag is de wrapper component die de isomorphic-fetch [47] call abstraheert. Ook maakt deze wrapper component gebruik van de eerdergenoemde config.js component. Het voordeel van de abstractie is dat nu elke Action creator deze fetch kan aanroepen zonder dat er duplicate code ontstaat.

- Action Creator

Zoals eerder beschreven moet de Action Creator een action object maken aan de hand van de data die deze binnenkrijgt van de web-Api call. Om duplicate code te voorkomen is er gebruik gemaakt van een wrapper class. De action creator roept nu deze wrapper class Fetch functie aan. Aangezien het hier gaat om een A sync call is het een algemeen principe om twee verschillende action objecten te maken, zodat het duidelijk is in welke state de applicatie zich op dat moment bevindt. Het onderscheid wat hier wordt gemaakt is een Request en een Receive. De request zal aan de start van de action creator call worden aangemaakt en hier zal lege component/objecten/data staan. Daarna zal de fetch gedaan worden en deze zal worden omgezet naar een JSON-format. Daarna zal er een Receive action object worden aangemaakt, maar in dit geval zal hierin de data staan die net opgehaald is door de fetch.

- Reducer

Stap twee was het implementeren van de reducer. De reden om eerst de reducer te maken voordat de store wordt gemaakt is vrij simpel. De store verwacht namelijk een reducer als property. In de reducer wordt omgeschreven hoe de applicatie state is aangepast aan de hand van de action die is gedaan. In dit geval moet de matchexperiencereducer luisteren naar twee actions, namelijk de request en receive die eerder zijn geschreven in de action creator. Aan de hand van de data die deze binnen krijgt past de reducer de state aan. Een belangrijk punt is dat er te allen tijde niet wordt gewerkt aan de huidige state van de applicatie. Er moet

altijd een nieuw object worden aangemaakt, dit is cruciaal om ervoor te zorgen dat in de gehele applicatie dezelfde state wordt aangehouden. Om niet te werken aan de huidige state moet er een kopie worden gemaakt, dit wordt gedaan door gebruik te maken van een ecmascript 5 of ecmascript 6 functie [74]. In ecmascript 5 wordt er gebruik gemaakt van Object.Assign(). In ecmascript 6 wordt er gebruik gemaakt van de ...object syntax.

- De store

Nu de applicatie weet wat er veranderd is (action) en hoe deze veranderd is (reducer) is het tijd om de store toe te voegen. De store in redux hoeft qua programmeren niet veel aan gedaan te worden. De store is geheel abstract en het enige wat gedaan moet worden is deze store meegeven aan alle views. Ook dit gedeelte kan gedaan worden zonder veel te programmeren, dit omdat redux een component daarvoor heeft gemaakt die dit allemaal doet. Het enige wat er moest gebeuren was het importeren van een Provider en deze in de entry point van de applicatie meegeven. In onderstaande figuur is de entry point van de applicatie te zien, met daarin de Provider en hoe deze wordt meegegeven aan alle views in de gehele applicatie.

```
/**
 * Render function.
 * Provider tags to send the store down to all child components that need it.
 * See React-Redux Provider for more information.
 */
render(
  <Provider store={store}>
    <MatchExperience />
  </Provider>,
  document.getElementById('root')
)
```

Figuur 14 index.js entry point

Nu zowel de store, de reducer(s) en de action(s) zijn toegevoegd is de gehele redux cycle compleet. De data zal nu in de applicatie aanwezig zijn en het enige wat nu gedaan moet worden is deze data gebruiken en op de juiste manier weergeven aan de gebruiker.

6.6. TESTEN

Tijdens deze sprint wordt het ophalen van data getest. Het ophalen van data wordt gedaan in de action componenten en zullen om die reden dan ook getest worden.

Als eerste moet er bepaald worden welke testen hier gebruikt kunnen worden die van toegevoegde waarde zijn voor de applicatie. Testen is uiteraard belangrijk, maar moet wel ergens voor dienen. 100% test coverage is leuk om te hebben, maar zal vaak zo zijn dat er testen worden gemaakt om het testen. De testen die na overleg nuttig zijn gevonden is het testen of de functie request/receive-match experience wordt aangeroepen en of dat de data correct is die uit deze functies komen.

Zoals eerder vermeld zal er gebruik gemaakt worden van Mocha-Chai-Sinon en zal er JSDOM worden gebruikt indien er een html pagina gesimuleerd moet worden. Om JSDOM op te zetten is er een helper component gemaakt. Voor de html testen is er JSDOM nodig, om deze reden is het nodig geweest om JSDOM op te zetten. In onderstaand figuur is een testhelper.js file gemaakt die JSDOM nodig heeft om een html pagina te kunnen simuleren.

```
import jsdom from 'jsdom';
import chai from 'chai';
import chaiImmutable from 'chai-immutable';

// Setup Jsdom so it can work with react.
const doc = jsdom.jsdom('<!doctype html><html><body></body></html>');
const win = doc.defaultView;

// Set the doc and window on the global properties, this so React can find them when it accesses document or window.
global.document = doc;
global.window = win;

// Take all the properties the jsdom window object has, and hoist/copy them on the Node.js global object.
// This is done so the properties can be used without the window. prefix.
Object.keys(window).forEach((key) => {
    if (!(key in global)) {
        global[key] = window[key]
    }
});

// Use chaiImmutable, because immultable collections will be used.
chai.use(chaiImmutable);
```

Figuur 15 - test helper voor Jsdom (voorbereiding voor view/html testen)

- Action Creator Tests

Er zijn twee verschillende testen gemaakt. Er is een test die bekijkt of de request/recievematchexperience data wordt aangeroepen. De tweede test is dat er wordt

gekeken of de data die wordt geleverd de juiste data is. Hieronder is een test te zien die bekijkt of de data overeenkomt.

```
it("Should return action type Request_Match_Experience", () => {
    const store = mockStore({})

    const expected = [{
        type: 'REQUEST_MATCH_EXPERIENCE',
        data: {
            typeOfMatch: "league",
            leagueId: 7,
            matchId: 7,
            weekId: 7,
            matchData: [],
        }
    }]

    store.dispatch(MatchExperience.requestMatchExperience("league", 7, 7, 7))

    expect(store.getActions()).to.deep.equal(expected);
})
```

Figuur 16 - action creator test

6.7. CONCLUSIE

Tijdens deze sprint is er gewerkt aan het uitdenken van de CSS-structuur, is er gekozen welke CSS-syntax er zal worden gebruikt en of CSS-Modules een toekomst heeft in dit project. Naast de CSS-structuur is de eerste web-Api call een feit en is het Redux-design pattern toegepast om deze match experience data op te halen gebruikt. Als laatste zijn er testen gemaakt voor de reducer en de action creator.

Ook is er weer een gesprek geweest met de begeleider van Gamebasics. Uit dit gesprek zijn de volgende punten gekomen.

- Begeleiding zal worden veranderd naar meerdere personen, om de overige personeelsleden meer te betrekken bij het project en wat tijdsdruk van de begeleider af te nemen.
- Blij met de keuze om LESS te gebruiken.
- Overzichtelijker iets uitleggen, niet van de hak op de tak springen.
- Waarom javascript en geen typescript?

Naast de punten die hierboven zijn beschreven is er ook verder gekeken naar de planning van de volgende sprint.

Hieronder staan die punten die tijdens de volgende sprint zullen worden gemaakt.

Planning volgende sprint
Studio LESS layout
Studio LESS responsive
Studio Commentatoren + tekst ophalen.
(Alles in LESS indien mogelijk)
Verslag schrijven
Typescript keuze

7. SPRINT 3 DE STUDIO MATCH EXPERIENCE

In deze sprint ligt de focus op het maken van de studio voor de match experience. De studio is het eerste onderdeel waar de gebruiker op binnenkomt wanneer de match experience wordt gestart. De studio zal worden weergegeven, net zoals bij een echte wedstrijd, bij de voorbeschouwing, rust, voorbeschouwing extra tijd, voorbeschouwing penalty's en nabeschouwing. Het is om deze reden dan ook een cruciaal onderdeel van de match experience. Aangezien de requirements voor de match experience al waren opgesteld hoefde dat nu niet in het specifiek gedaan te worden voor het onderdeel de studio. Wel is er gekeken naar welke requirements behaald moesten worden deze sprint.

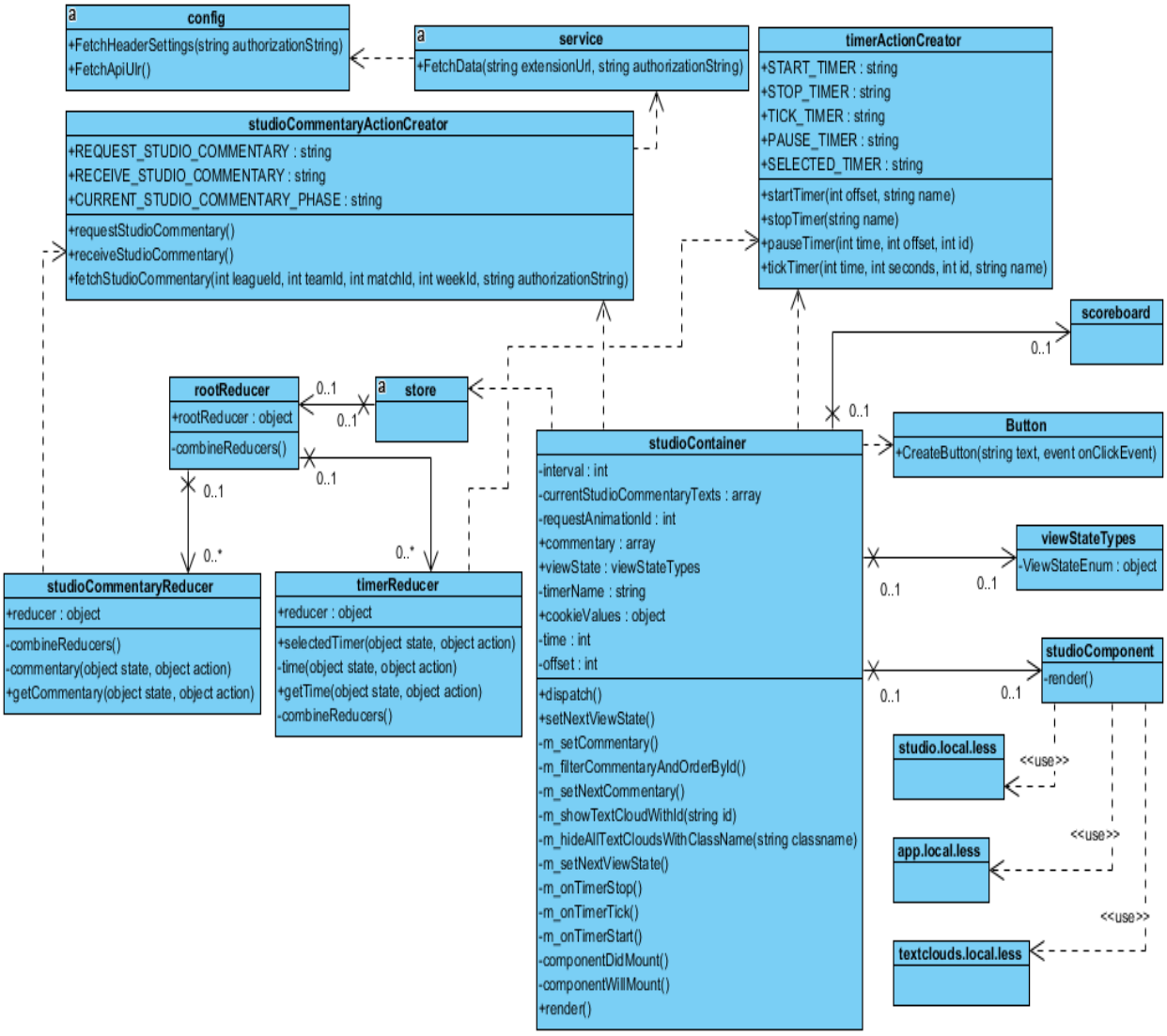
7.1. REQUIREMENTS

Hieronder zijn de requirements te zien waar de studio aan moest voldoen na afloop van deze sprint.

- Het systeem moet teksten kunnen ophalen (commentator teksten). (Functionele Requirement)
- Het systeem moet teksten kunnen weergeven (commentator teksten). (Functionele Requirement)
- Het systeem moet Json format data kunnen ophalen van de web-Api. (Niet-functionele Requirement)
- Het systeem moet images kunnen ophalen. (Functionele Requirement)
- Het systeem moet images kunnen weergeven. (Functionele Requirement)
- Het systeem moet verschillende views ondersteunen (Functionele Requirement)
    - Studio
- Het systeem moet verschillende views weergeven (Functionele Requirement)
    - Studio
- Het systeem moet responsive zijn. (Niet-functionele Requirement)
- Het systeem moet presentatoren kunnen weergeven. (Functionele Requirement)
- Het systeem moet thuis altijd links weergeven. (Functionele Requirement)
- Het systeem moet uit altijd rechts weergeven. (Functionele Requirement)

7.2. DIAGRAMMEN

7.2.1. DESIGN KLASSENDIAGRAM



Figuur 17 - design klassendiagram studio

7.3. TOEPASSING

- Responsive

Om responsive te blijven is er onderscheid gemaakt tussen resolutiegroottes. Er begint namelijk een degelijk verschil te komen op 768 pixels in de breedte, oftewel mobile (alles onder de 768 pixels) en tablet, laptop en desktop (alles boven de 768 pixels). Om de studio responsive te houden is er gebruik gemaakt van bootstrap en is er zoveel mogelijk gebruik gemaakt van viewport values, zodat deze altijd mooi mee schalen naar mate van de hoogte en breedte. In hoofdstuk 8 is een figuur te zien met hoe de studio eruitziet op 768+ pixels.

- Images

Twee andere requirements waren het ophalen en weergeven van images. Het meeste werk voor deze twee requirements waren al gedaan in de vorige sprint, namelijk het toevoegen van een Url en File-loader. Het enige wat moet worden gedaan om images te kunnen ophalen en weer te geven, is het toevoegen van een geldige URL in de background-image property.

- Web-Api call

Het ophalen van de commentatorenteksten is ook via een A-sync call gegaan naar de web-Api. In principe is er vrij weinig verschil tussen het ophalen van de matchdata, vergeleken met het ophalen van de commentatorenteksten. Het enige wat gedaan moest worden was het toevoegen van een Reducer, namelijk een StudioCommentatorenReducer en de action heeft andere data. De aanpassingen die op designniveau gedaan moesten worden zijn te zien in hoofdstuk 7.2.1.

- Studio View (meerdere views ondersteunen)

Naast de Studio zijn er straks ook nog andere views die moeten worden weergegeven. Omdat hier gebruik gemaakt wordt van een single page application, moet er onderscheid worden gemaakt, wanneer, welke view wordt weergegeven. Om deze reden is er rekening gehouden met een viewstate. In de viewstate staat gedefinieerd in welke fase de wedstrijd zit. Aan de hand van deze viewstate weet de applicatie of de Studio of een andere view moet worden weergegeven. De viewstate enum wordt net zoals de matchdata en de studio commentatoren teksten via het Redux pattern geüpdatet. Hieronder is een deel van de viewstate object enum te zien.

```
PRE_MATCH: {
    name: "PRE_MATCH",
    showName: "Pre Match",
    id: 0,
    matchPhaseId: 2,
    commentatorPhaseId: 0
},
FIRST_HALF: {
    name: "FIRST_HALF",
    showName: "First Half",
    id: 1,
    matchPhaseId: 4,
    commentatorPhaseId: -1
},
```

Figuur 18 – gedeelte viewstate enum

- Teksten weergeven

De teksten die weergeven moeten worden zijn al opgehaald en in de container van de studio gezet. De volgende stap is het weergeven van deze teksten. Het weergeven van deze teksten moeten aan de hand van de fase gebeuren waarin de applicatie zich verkeert. Indien de studio in Pre-Match is, moet de pre-match commentatorteksten worden weergegeven, voor halftime de half time teksten, etc. Om dit voor elkaar te krijgen is er zoals in het vorige punt gebruik gemaakt van een viewstate enum object. Wanneer de viewstate in de pre-match zit, zal alle commentatoren teksten worden gesorteerd en zal alleen de pre match teksten worden doorgegeven aan de studiocomponent. Deze zullen in deze sprint nog via een button click om en om worden weergegeven. In een later stadium zal dit allemaal timer based moeten worden gedaan, zodat het enige wat de gebruiker hoeft te doen is de match experience op te starten en daarna zal alles automatisch worden gespeeld.

- Presentatoren weergeven

Een belangrijk onderdeel om de gebruiker meer het gevoel te geven dat ze daadwerkelijk in een live wedstrijd zitten is door het gebruik van presentatoren. De teksten die worden weergegeven zal nu door een presentator worden "gesproken". Waardoor de gebruiker een betere speelervaring krijgt met de match experience.

- Thuis links, uit rechts.

Een andere requirement was dat de match experience duidelijk weergaf of de gebruiker thuis of uit speelt. Dit is net zoals in een real-life voetbalwedstrijd gedaan, door het thuisspelende team altijd aan de linkerkant van het scherm te tonen en het uitspelende team altijd aan de rechterkant te tonen. Dit was zeer eenvoudig toe te passen, aangezien er in de cookie waardes een variable wordt bijgehouden of de gebruiker uit of thuis speelt.

7.4. TYPESCRIPT

Tijdens sprint 1 en 2 was er naar voren gekomen om te kijken of het niet beter zou zijn om gebruik te maken van typescript [66]. De reden waarom dit naar voren kwam is om meerdere redenen.

- Static type checking

Typescript zorgt ervoor dat er static type checking gedaan kan worden. Dit houdt onder andere in dat er compile-time kan worden gekeken of de applicatie geen errors bevat. Daarnaast kan er typing worden toegepast aan variable. Dit betekent dat er bijvoorbeeld een string variable kan zijn, dit tegenover de standaard var, const en let die javascript ondersteunt.

- Huidige situatie

Een andere reden waarom er naar typescript gekeken moest worden is, omdat er in de huidige situatie al gebruik gemaakt wordt van typescript en dit zou de overgang naar de huidige applicatie moeten versimpelen.

7.4.1. DE KEUZE

Er is voor gekozen om geen gebruik te maken van typescript, dit terwijl React wel typescript zou kunnen ondersteunen wanneer hier een library voor wordt toegevoegd.

Er zijn meerdere redenen waarom er geen gebruik gemaakt zal worden van typescript. De eerste reden is, tijdens het maken van de applicatie wordt er al van vele nieuwe externe libraries gebruik gemaakt een library/nieuwe taal toevoegen zou ervoor kunnen zorgen dat er geen werkend prototype wordt opgeleverd. Een andere reden is dat React ook property checks heeft. Dit wordt alleen pas op run-time ondervonden.

Indien er meer tijd zou worden gegeven voor het afstuderen of wanneer er minder nieuwe libraries zou zijn gebruikt zou de voorkeur wel uitgaan naar het gebruik van typescript.

7.5. CONCLUSIE

Tijdens deze sprint kwam de studio aan bod. De planning was dat de studio helemaal klaar zou zijn tijdens deze sprint, dit is helaas niet gelukt. Hieronder zullen eerste de punten worden weergegeven die zijn behaald, daarna zal de punten worden beschreven die de volgende sprint periode meegenomen moet worden.

- Behaald
    - Responsive maken studio (match experience) (768 tot max pixels).
    - Web-Api call
    - Studio Meerdere views ondersteunen
- Niet behaald
    - Responsive maken studio (match experience) (min tot 768 pixels).
    - Teksten weergeven

Zoals elke sprint is er een gesprek geweest met de begeleiders van Gamebasics. Uit dit gesprek kwam de volgende punten.

- Sprint doel niet behaald
    - Om deze reden is ervoor gekozen om het responsive maken van de match experience op te splitsen in 0 tot 768 en 768 tot max.
- Code opschonen
- Studio Afmaken

Hieronder is de planning te zien van de volgende sprint.

Planning volgende sprint
Redux Design Pattern. (Reducers hiërarchie) (Code opschonen)
Studiocommentatorenteksten weergeven.
Code en comments toevoegen waar nodig. (Code opschonen)
Verslag schrijven

8. SPRINT 4 DE STUDIO MATCH EXPERIENCE PART 2

Tijdens deze sprint lag de focus nogmaals op de Studio. De Studio was zoals kan worden gelezen in de vorige sprint niet afgekomen. Om deze reden is er deze sprint nogmaals gewerkt aan de Studio. De volgende punten waren een vereiste voor deze sprint.

- Redux design pattern
- Studio teksten weergeven via een viewstate update
- Opschonen Applicatie (code en comments toevoegen waar nodig)
- Verslag schrijven

De requirements die in de vorige sprint zijn gebruikt zijn ook deze sprint weer gebruikt, waarbij er één requirement aangepast is.

- Het systeem moet responsive zijn.

Deze requirement is aangepast naar twee requirements. Namelijk;

- Het systeem moet responsive zijn voor tablets, laptops en desktops.
- Het systeem moet responsive zijn voor mobiles.

Door deze requirement op te splitsen kan er worden gefocust op de requirement responsive voor tablets, laptops en desktops.

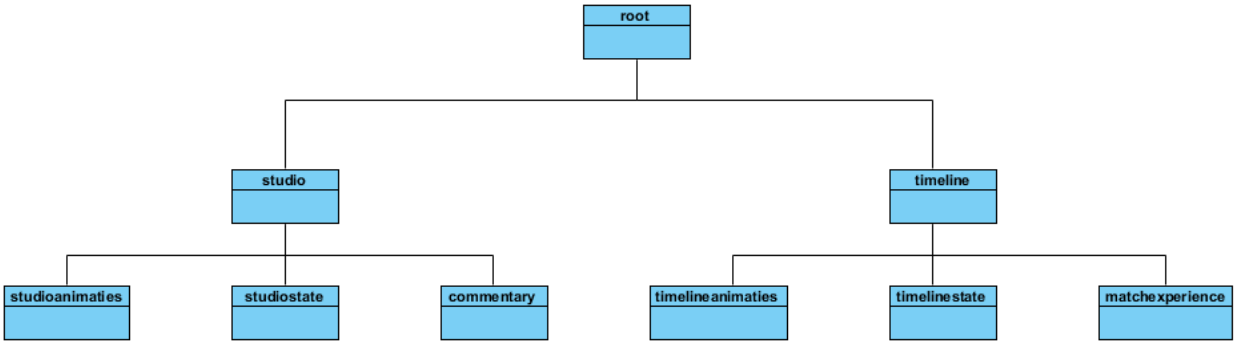
8.1. REDUX DESIGN PATTERN (REDUCERS)

Een punt dat naar voren kwam vorige sprint was hoe het Redux design pattern in elkaar zit en in het specifiek de reducers. Tijdens deze sprint is er tijd vrij gemaakt om hier dieper op in te gaan. Er is nogmaals gekeken naar Redux en hoe deze toegepast moeten worden in het huidige project. Hiervoor is gekeken naar voorbeelden die op de website van Redux staan, namelijk de A-sync en Real world example's [67]. Na deze voorbeelden door te hebben genomen zijn er twee opties over gebleven over hoe de reducer tree in elkaar wordt gezet. Op het moment wordt Optie 1 toegepast.

8.1.1. DE OPTIES

Als eerste de opties. Hieronder is voor elke optie een diagram te zien hoe deze in de huidige situatie zou gaan werken.

- Optie 1



Figuur 19 - reducers indeling optie 1

- Voordelen

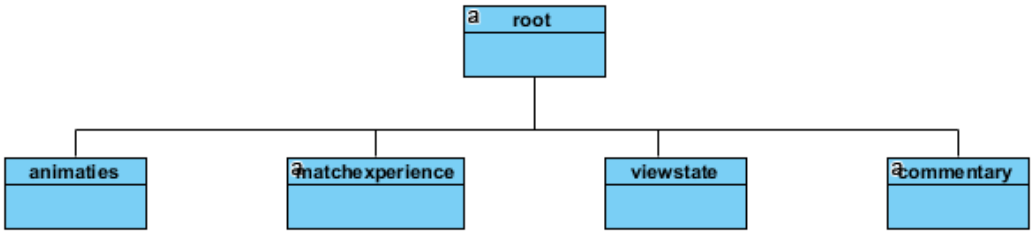
Leesbaarheid: elke reducer doet één taak.

- Nadelen

Duplicate code: studioAnimaties en timelineAnimaties zullen niet veel verschillen hebben, indien ze deze al hebben.

Uitbreidbaarheid: View Afhankelijk, elke keer als er een view bijkomt zal er een nieuwe tree gemaakt moeten worden voor die view.

- Optie 2



Figuur 20 - reducers indeling optie 2

- Voordelen

Uitbreidbaarheid: Niet view afhankelijk.

Geen duplicate code: Indien er meerdere views verschillende animaties moeten bijhouden kan er aan de hand van een ID of Name de animatie state uniek worden gemaakt.

- Nadelen

God-class: De root reducer krijgt nu in de huidige situatie van de applicatie alle reducers.

8.1.2. DE KEUZE

De huidige situatie maakt gebruik van Optie 1. Na zowel optie 1 en, nu ook, optie 2 te hebben heroverwogen is er gekozen voor Optie 2. Het nadeel dat Optie 2 op het moment heeft is te overzien, het zijn namelijk voor nu maar vier reducers. Het grote voordeel wat voor mij van doorslaggevend niveau was, is het feit dat er nu niet meer wordt gekeken naar views. De reducer hiërarchie is onafhankelijk van hoeveel verschillende views er zijn.

8.2. OPSCHONEN APPLICATIE EN AANPASSINGEN REDUCERS TOEPASSEN

Nadat de reducers opties waren uitgedacht en er was besloten welke optie er gebruikt zou worden. Was het noodzaak om Optie 2 toe te passen in de huidige situatie. Dit zorgde voor redelijk wat problemen. De data die voorheen goed in de container en component classes terecht kwamen, waren nu allemaal undefined. Na goed te hebben gekeken naar de A-sync en Real world example's [67] kwam ik erachter dat ik de reducers niet de juiste namen gaf tijdens het opzetten van de state op de properties van een container klasse. Ook is er gekeken naar het opschonen van de applicatie. Het opschonen van de applicatie houdt vooral in dat er geen extra variables staan die niet meer worden gebruikt, dat er waar nodig commentaar staat bij de functies. En dat er te allen tijde bij elke functie een descriptie is ingevuld.

8.2.1. TOEPASSING

De reducers zijn opgezet dat deze een id op de manier van een string verwacht. De properties van de componenten waren telkens undefined, omdat niet de juiste id werd meegegeven aan de reducer functie.

Om de data weer goed in de containers en in de componenten te krijgen, moest er worden gekeken naar de functie mapstatetoprops. In deze functie werd de verkeerde Id meegegeven aan de reducer functie.

De oplossing was uiteindelijk vrij simpel op te lossen, in de reducer is er een functie toegevoegd die de huidige State ophaalt, hierin staat ook de Id. Deze id wordt dan weer gebruikt om de data op te halen.

8.3. TEKSTEN WEERGEVEN EN CSS STYLING

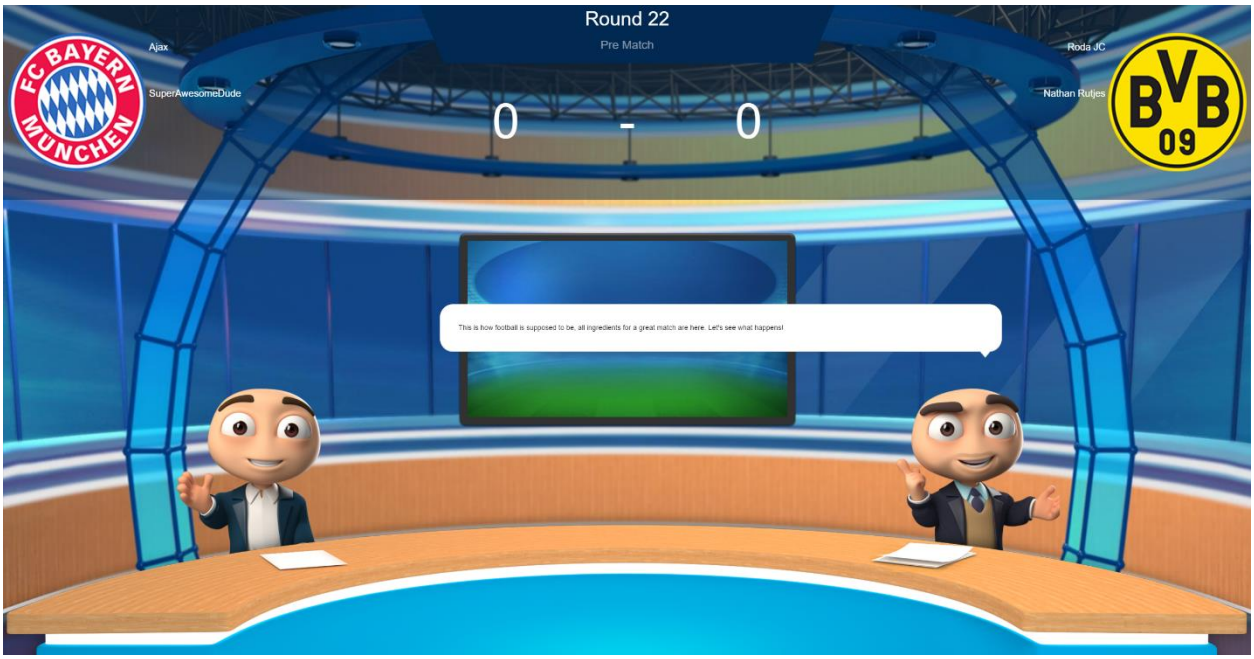
Het ophalen van de teksten was vorige sprint al toegepast, het weergeven was helaas niet behaald de vorige sprint. Om deze reden is er gekeken hoe de teksten het beste weergegeven kon worden. Hier is gekeken naar iOS en Android en hoe deze de commentatoren teksten weergeven.

Wanneer de teksten worden opgehaald haalt deze gelijk alle commentatoren teksten op, dit houdt in dat de applicatie de teksten heeft voor pre-match maar ook voor halftime en post-match. Deze commentatoren teksten moeten gesorteerd worden.

Het sorteren van de commentatoren teksten is gedaan aan de hand van de viewstate waarover in hoofdstuk 7.3 meer over is verteld.

Het weergeven van deze teksten was daarna vooral veel werk qua CSS-styling en html.

Hieronder is een image te zien van de Studio na het verwerken van de teksten en de laatste CSS-styling.



Figuur 21 - de studio completed

8.4. CONCLUSIE

Tijdens deze sprint kwam nogmaals de studio aan bod. Om de studio dit keer helemaal af te krijgen, moest de commentatoren teksten weergegeven worden en moest er een schoonmaak gedaan worden op de code. Alle punten die deze sprint op de agenda stonden zijn behaald en de Studio is dan ook voor nu 100% klaar.

Uit het gesprek met de begeleiders van Gamebasics zijn de volgende punten naar voren gekomen.

- Studio ziet er goed uit.
- Noodzaak om de timer te gaan toepassen, zodat alles automatisch gaat.
- Indien mogelijk het Redux pattern blijven toepassen.

Aan de hand van dit gesprek zijn de volgende punten opgesteld voor de volgende sprint.

Planning volgende sprint
Begin Timeline maken. De eerder opgehaalde matchdata weergeven in de timeline. Timeline responsive maken, net zoals de Studio alleen 768 tot max. De viewstate die nu gebruikt wordt, ook hergebruiken voor de timeline. Zowel de Studio als de Timeline Timer based maken. Verslag schrijven

9. SPRINT 5 DE TIMELINE MATCH EXPERIENCE

Sprint 5 is de een na laatste sprint, zoals beschreven in de vorige sprint staat de Timeline op de planning. De timeline is naast de studio het hart van de match experience. Op de timeline zal de matchdata worden weergeven die tijdens de eerste sprint is opgehaald.

9.1. REQUIREMENTS

Nu de studio is geïmplementeerd zal er gekeken worden naar de timeline, de volgende requirements zijn van toepassing op de timeline en zullen moeten worden behaald tijdens deze en de volgende sprint.

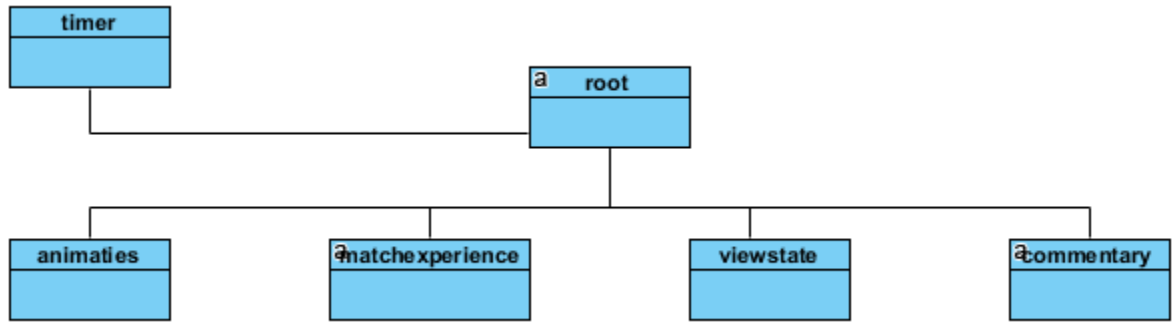
- Het systeem moet Json format data kunnen ophalen van de web-Api. (Niet-functionele Requirement)
- Het systeem moet images kunnen ophalen. (Functionele Requirement)
- Het systeem moet images kunnen weergeven. (Functionele Requirement)
- Het systeem moet verschillende views ondersteunen (Functionele Requirement)
    - Timeline
- Het systeem moet verschillende views weergeven (Functionele Requirement)
    - Timeline
- Het systeem moet responsive zijn. (Niet-functionele Requirement)
- Het systeem moet thuis altijd links weergeven. (Functionele Requirement)
- Het systeem moet uit altijd rechts weergeven. (Functionele Requirement)
- Het systeem moet de events time based kunnen afspelen. (Niet-functionele Requirement)
- Het systeem moet een gehele wedstrijd kunnen weergeven. (Functionele Requirement)
    - Studio (voorbeschouwing, rust, nabeschouwing verlenging, nabeschouwing penalty, nabeschouwing)
    - Timeline (eerste helft, tweede helft, verlenging, penalty's)
- Het systeem moet een timeline weergeven, waarop het wedstrijdverloop is te zien.

- Het systeem moet een timer kunnen weergeven die ten minsten van 0 tot en met 120 loopt.

9.2. KLASSENDIAGRAMMEN

Tijdens deze sprint is er een reducer bijgekomen, namelijk de timer. Hieronder is een weergave te zien van de nieuwer reducer hiërarchie.

9.2.1. ANALYSE KLASSENDIAGRAM



Figuur 22 - reducers indeling met timer

9.3. IMPLEMENTATIE

Qua implementatie moesten de volgende punten worden gedaan deze sprint.

- Timeline Responsive

Als eerste is er gewerkt aan het responsive maken van de timeline, wanneer namelijk de view is gemaakt kan er gewerkt worden aan het weergeven van de matchdata en waar deze matchdata weergegeven moet worden. Het responsive maken van de timeline kosten ten opzichte van de studio een stuk minder tijd, dit omdat het al eerder is gedaan bij de studio en alles nu een stuk duidelijker was hoe er gewerkt moest worden met de viewport values. Het responsive maken is dan ook net zoals de studio gedaan door gebruik te maken van bootstrap.

- Timeline matchdata weergeven

Nadat de timeline responsive was gemaakt, was het tijd om te werken aan het weergeven van de matchdata. De matchdata was in de eerste sprint al opgehaald. De data moest nu nog uitgelezen worden en op de juiste manier worden weergegeven. Dit heeft de meeste tijd gekost tijdens deze sprint. De moeilijkheid zat hem vooral in het zorgen dat de view componenten, pure bleven. Dit houdt in dat hier geen logica staat, maar alleen hoe het moet worden weergeven. Om de componenten pure te houden zal er gewerkt worden zoals in het analyse diagram van hoofdstuk 6.2.1. vermeld staat. Hieronder zijn twee figuren te zien, de eerste zal de component weergeven en de tweede zal de container weergeven.

```
import TimelineLESS from '../../content/components/timeline.local.less'
import ApplicationLocalLESS from '../../content/app.local.less'

export default class TimeLine extends React.Component {
    render() {
        return (
            <div className={["row", ApplicationLocalLESS.fullHeight].join(' ')}>
                <div className={["col-sm-12", TimelineLESS.timelineBackGroundContainer].join(' ')}>
                    <div className={["row", ApplicationLocalLESS.fullHeight].join(' ')}>
                        <div className={["col-sm-12", TimelineLESS.timelinebackground].join(' ')}>
                            {this.props.scoreBoard}
                            {this.props.timelineEvents}

                            <div className={["row"].join(' ')}>
                                <div className={["col-sm-6"].join(' ')}>
                                    {this.props.startPauseTimer}
                                    {}
                                </div>
                                <div className={["col-sm-6"].join(' ')}>
                                    {this.props.switchViewState}
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        )
    }
}
```

Figuur 23 - timeline container

```
class Timeline extends React.Component {
    componentWillMount() {
        this._seconds = 0;
        this._homeScore = this.props.homeScore;
        this._awayScore = this.props.awayScore;
    }
    componentDidMount() {
        this.m_onTimerStart();
    }
    m_onTimerStart = () => {
        const { dispatch } = this.props
        this.requestAnimationFrame = requestAnimationFrame(this.m_onTimerTick);
        dispatch(startTimer(Date.now(), this.props.timerName));
    }
    m_onTimerTick = () => {
        const { dispatch } = this.props
        this.requestAnimationFrame = requestAnimationFrame(this.m_onTimerTick);

        if (Date.now() - this.props.offset >= this.props.interval) {
            this._seconds = Math.round(((this.props.time + (Date.now() - this.props.offset)) / 1000));
            dispatch(tickTimer(Date.now(), this._seconds, this.requestAnimationId, this.props.timerName));
            //this.m_checkCurrentTimelineEvent();
        }
        if (this.props.viewState.endTime <= this._seconds || this._penaltiesDone) {
            this.m_onTimerPause();
            setTimeout(() => this.m_setNextViewState(), this.props.delay);
        }

    }
```

Figuur 24 - gedeelte timeline component

- Timeline en Studio event based(timer)

Een ander belangrijk onderdeel deze sprint was het toevoegen van de timer en op deze manier, zowel de timeline als de studio event based maken. Dit zorgt ervoor dat de gebruiker geen acties meer hoeft te doen met de applicatie, nadat deze is opgestart. Ook voor de timer is het Redux design pattern toegepast. Er is namelijk voor gekozen, wanneer er een state in meerdere klassen gebruikt wordt, deze via het Redux design pattern wordt toegepast. Als de

state alleen in één component van belang is, wordt hier geen Redux voor gebruikt, maar zal dit in de component zelf worden opgelost. Dit is de reden waarom er een Redux design pattern is gebruikt om de tijd state bij te houden. Dit omdat zowel de timeline als de studio gebruik maakt van deze timer.

9.4. CONCLUSIE

Tijdens deze sprint was het de taak om de timeline toe te voegen aan het project. Door de timeline in samenwerking met de studio wordt de match experience nog levendiger en krijgt de gebruiker nog meer het gevoel dat er ook daadwerkelijk iets gedaan wordt tijdens het spelen.

De punten die deze sprint op de planning stonden zijn dan ook bijna allemaal behaald. Het punt dat niet behaald is, het updaten van de viewstate aan de hand van de viewstate enum die al eerder was geïmplementeerd. Dit is gekomen, doordat er te weinig tijd over was na het toevoegen van de timer en het responsive maken van de timeline. Volgende sprint en tevens de laatste sprint zal dan ook onder andere dit punt hebben.

Net zoals elke sprint, is er ook deze sprint weer een gesprek geweest met de begeleiders van Gamebasics. Uit het gesprek zijn de volgende punten naar voren gekomen.

- Tevreden met de keuze om Redux te gebruiken wanneer deze op meerdere componenten zal worden gebruikt.
- Timeline ziet er goed uit, nog wel even nagaan bij de afdeling UX of dit ook volgens het design is dat het UX-team heeft opgesteld.
- Viewstate toepassen volgende sprint, zodat het overal gelijk is aan elkaar.

Naast bovenstaande punten zullen er nog meer punten op de planning staan voor volgende sprint. Hieronder zijn alle punten te zien die in de laatste sprint nog gedaan moeten worden.

Planning volgende sprint
De viewstate die nu gebruikt wordt, ook hergebruiken voor de timeline.
Het mogelijk maken dat er penalty shoot out gedaan kan worden.
Het mogelijk maken dat er verlenging gespeeld kan worden
Animaties toevoegen aan de events op de timeline.
Animaties toevoegen aan goal/no goal op de timeline.
Verslag schrijven.

10. SPRINT 6 DE TIMELINE MATCH EXPERIENCE PART 2

Tijdens deze sprint zal er gewerkt worden aan het updaten van de viewstate, zodat ook de timeline hiernaar zal gaan luisteren. Het toevoegen van de penalty shootout, het mogelijk maken dat er een verlenging gespeeld kan worden en animaties van zowel de events als de goal/no goal toevoegen aan de timeline.

10.1. REQUIREMENTS

De requirements van vorige sprint zijn ook deze sprint van toepassing. Daarnaast zijn er nog een paar requirements bijgekomen, namelijk de requirements die te maken hebben met de penalty view en animaties. Hieronder zijn de requirements te vinden die deze sprint erbij zijn gekomen.

- Het systeem moet animaties kunnen weergeven. (Functionele requirement)
- Het systeem moet animaties fullscreen weergeven. (Functionele requirement)
- Het systeem moet voor zowel goals als kansen animaties kunnen weergeven. (Functionele requirement)
- Het systeem moet verschillende views ondersteunen (Functionele requirement)
    - Penalty's
- Het systeem moet verschillende views kunnen weergeven (Functionele requirement)
    - Penalty's
- Het systeem moet penalty reeksen kunnen weergeven (1 tot 5 penalty's) (Niet-functionele requirement)
- Het systeem moet gescoorde penalty's groen inkleuren. (Niet-functionele requirement)
- Het systeem moet niet gescoorde penalty's rood inkleuren. (Niet-functionele requirement)
- Het systeem moet een donkere overlay tonen tijdens de animaties. (Niet-functionele requirement)

10.2. IMPLEMENTATIE

De volgende punten moesten geïmplementeerd worden;

- Viewstate aanpassen wanneer ervan de timeline wordt weg genavigeerd.

De viewstate enum en het redux pattern dat de viewstate moet aanpassen bestond al, dit omdat het al was gemaakt voor het switchen van de studio naar de timeline. Dit is nu ook toegepast wanneer de timer 45, 90 en 120 seconde aangeeft. Met deze aanpassing is er ook gelijk voor gezorgd dat er een verlenging gespeeld kan worden.

- Event animaties

Een andere vereiste was dat de matchdata events animaties zouden krijgen. Om dit voor elkaar te krijgen is er gebruik gemaakt van een externe library, namelijk; Animate.less. Animate.less heeft alle basic animaties al, waardoor dit niet meer handmatig geprogrammeerd heeft te

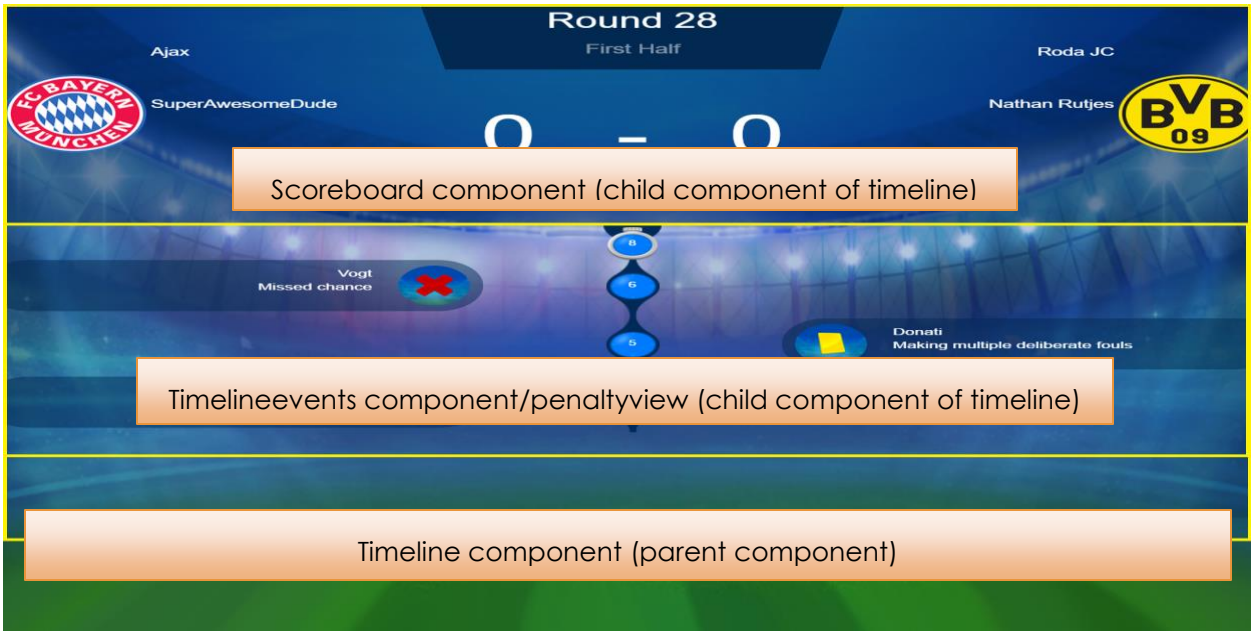
worden [72]. Een ander voordeel hiervan is, dat het allemaal in LESS is gemaakt. Net zoals bootstrap zal de Animate.Less niet meegenomen worden door de CSS-modules.

- Goal/ no goal animaties

Ook de goal en no goal animaties zijn gemaakt door gebruik te maken van het Redux design pattern. Er kan nu aan de hand van een event call in de timelineevents component een update worden gedaan, waardoor de animatie van een event wordt aangeroepen. Op het moment van schrijven is de achterliggende code al geschreven en het enige wat nog gedaan moet worden is de html en CSS-opmaak van de animatie zelf.

- Penalties view

Wanneer er een cupwedstrijd is en de stand is na 120 minuten nog steeds gelijk moet er een penalty view worden weergeven. Doordat de penalty view erbij is gekomen, is ervoor gekozen om de timeline en de timelineevents op te splitsen in twee componenten. Wanneer nu de penalty's weergegeven moet worden zal er op de timeline, inplaats van de timelineevents de penalty view worden weergegeven. Om dit te verduidelijken is hieronder een afbeelding te zien met hierin de timeline en timelineevents component.



Figuur 3 - timeline met timeline events

10.3. CONCLUSIE

Tijdens deze sprint was de tijdsdruk toch vrij goed te merken. Er moest namelijk nog redelijk wat aan het verslag worden gedaan, wat ervoor zorgde dat niet alle punten zijn behaald. Hieronder is samengevat welke punten zijn behaald en welke niet en waar om deze reden nog een keer naar gekeken moet worden.

- Behaald
    - Viewstate aanpassen wanneer er genavigeerd wordt van timeline naar studio
    - Animaties van events op de timeline
    - Achterliggende code van zowel de penalty view als goal/ no goal animatie
    - Het mogelijk maken van verlenging
    - Het mogelijk maken van penalty's
- Niet behaald
    - Het weergeven van de penalty view
    - De animaties van goal/ no goal.

Ook tijdens deze sprint is er een gesprek geweest met de begeleiders en product owner. Tijdens dit gesprek zijn de volgende punten besproken.

- Overgang applicatie naar huidige team.
- Focus op verslag
- Tevreden met het resultaat
- Presentatie geven Gamebasics

11. EVALUATIE AFSTUDEERTRAJECT

Over het algemeen ben ik tevreden met het verloop van het afstudeertraject. Gamebasics had bij de start een wens dat er onderzoek werd gedaan naar verschillende javascript framework en dat hiermee een prototype zal worden gemaakt voor de feature match experience. De wens zelf was vrij concreet, hoe het op te lossen of aan te pakken ben ik helemaal vrij in gelaten. Uiteindelijk ben ik van mening dat zowel het onderzoek als het prototype van toegevoegde waarde kan en zal zijn voor Gamebasics. In deze evaluatie blik ik terug op het afstudeertraject en probeer ik de lezer duidelijk te verschaffen in de, in mijn ogen, sterke en minder sterke kanten van mijn denk en handelswijze. De evaluatie is opgesplitst in een evaluatie van aanpak, oftewel proces en een evaluatie van beroepstaken. In de evaluatie van aanpak kijk ik terug op mijn eigen denkwijze en de gemaakte beslissingen. In de evaluatie van beroepstaken leg ik uit hoe ik bepaalde beroepstaken die van tevoren zijn gedefinieerd in het afstudeerplan.

11.1. AANPAK

Aan het begin van het afstudeertraject is het afstudeervoorstel ingediend, waarin de opdracht is omschreven en waarin in grote lijnen de te nemen stappen staan beschreven. Tevens is er een stappenplan gemaakt wanneer deze taken afgerond dienden te worden. Deze planning is tijdens het afstudeertraject gevolgd en is uiteindelijk vrij realistisch, maar niet compleet gebleken. De vrijheid die tijdens het afstudeertraject is gegeven was fijn, maar zorgde ervoor dat er redelijk wat dagen zijn besteed aan het uitzoeken van verschillende libraries. Uiteindelijk was het doel van het afstudeertraject om advies te geven over welk javascript framework het beste past bij de feature match experience en moest er een prototype gemaakt worden in dit gekozen framework. Aan beide punten zijn voldaan, hoewel voor het prototype nog requirements open staan die gedaan moeten worden, voordat deze daadwerkelijk gebruikt kan worden voor productie.

11.1.1. PROJECT STRUCTUUR

Het opzetten van het project en welke libraries er gebruikt zouden moesten worden heeft veel tijd in beslag genomen. Dit zorgde in het begin van het afstudeertraject voor een redelijk grote scope. Helaas heb ik dit niet meegenomen in het afstudeerplan en dit zorgde voor dat er minder tijd over is gebleven om daadwerkelijk het prototype te maken. Dit heeft ook voordelen met zich meegebracht, doordat er over de meeste externe libraries goed was nagedacht was de overdracht naar het huidige web-team eenvoudig te volbrengen.

11.1.2.REQUIREMENTS OPSTELLEN

Het in kaart brengen van de requirements is gedaan aan de hand van het handboek requirements. Dit boek is ook gebruikt tijdens de gehele opleiding en zorgde voor een goede leidraad tijdens het houden van interviews en het opstellen van deze requirements. Daarnaast ben ik ook zeer tevreden met hoe de requirements naar boven zijn gekomen, door de interviews te houden is gebleken dat hoewel de applicatie al in op de andere platforms is gebouwd, er nog nieuwe requirements naar boven zijn gekomen die voor de afdeling web gelde. Zoals bijvoorbeeld het zorgen dat deze responsive moet zijn. Ook de flexibiliteit van de product owner en de begeleiders waren hierbij van zeer toegevoegde waarde. Om deze reden zijn voor een grotendeels alle must have's en should have's toegepast in het huidige prototype.

11.1.3. ONDERZOEK

Het onderzoek naar de javascript frameworks is een groot deel van mijn afstudeerperiode geweest en ik kan hier met een goed gevoel op terug kijken. Tijdens het onderzoek is er gekeken naar de criteria 's van de product owner en begeleiders en is er aan de hand hiervan een keuze gemaakt voor een javascript framework. Het enige wat een smet is op het onderzoek is dat de keuze van het aantal javascript frameworks terug gebracht is naar drie. Dit zou in principe alle frameworks moeten zijn die tijdens het schrijven van dit verslag gereleaset waren. Dit zou voor een nog beter onderzoek zorgen naar welk javascript framework er gebruik moet worden.

11.1.4. DESIGN

Het design van de match experience was een zeer nuttige toevoeging, regelmatig heb ik het idee gehad tijdens mijn opleiding dat design geen extra waarde toevoegt aan het systeem. Tijdens deze afstudeerperiode heb ik ondervonden dat ik het zeer nuttig vond om een analyse klassendiagram te maken van hoe het Redux design pattern in elkaar stak. Daarnaast heb ik voor het eerst ook het gevoel gehad dat een sequence diagram ook daadwerkelijk toegevoegde waarde heeft op het product.

11.1.5. IMPLEMENTATIE

De implementatie van het prototype verliep redelijk. Dit komt vooral, omdat er gewerkt is met veel nieuwe programma's en libraries waar nog niet eerder mee is gewerkt. Daarnaast is er ook gebruik gemaakt van nieuwe design patterns dat toch langer duurde om te begrijpen dan verwacht. Hoewel dit allemaal veel tijd heeft gekost ben ik zeer tevreden met het uiteindelijke resultaat en met de keuzes die zijn gemaakt. Het prototype is een goed werkende prototype waarin bijna alle requirements zijn verwerkt die tijdens het begin van de afstudeerperiode zijn opgesteld. De keuze om meer tijd te stoppen in het onderzoeken naar deze libraries en design patterns hebben ervoor gezorgd dat het prototype qua code goed leesbaar en uitbreidbaar is en de overdracht naar het huidige team verliep dan ook zonder problemen.

11.1.6. TESTEN

Een aspect tijdens deze afstudeerperiode was het toepassen van TDD als testtechniek. Hier ben ik zelf wat minder tevreden over. De testen zijn goed en werken allemaal zoals het hoort, ook is voor alle niet UI-functionaliteiten TDD toegepast. De reden waarom dat ik hier wat minder tevreden over ben is, omdat TDD niet voor mij is weg gelegd. Ik ben tijdens het programmeren graag dingen aan het uitproberen om zo tot het beste resultaat te komen. TDD zorgt er voor mijn gevoel voor dat ik dat niet meer kan/mag, omdat ervan tevoren al de test geschreven moet zijn. Hierom zou ik in de toekomst niet zo snel meer voor TDD kiezen. Wel ben ik van mening dat TDD en in het generiek testen een zeer nuttige toevoeging is voor zowel back als front-end development en zeker de moeite waard is, wanneer er degelijke testen worden geschreven.

11.2. BEROEPSTAKEN

11.2.1. SELECTEREN VAN STANDAARDSOFTWARE.

Een groot onderdeel van deze afstudeerperiode was het onderzoeken van een standaardsoftware, oftewel in dit geval een javascript framework. Er is een javascript framework gekozen en er is hierbij gekeken naar de requirements die het bedrijf stelde aan het javascript framework. Daarnaast is er een advies rapport gemaakt voor het bedrijf Gamebasics om React te gaan gebruiken als javascript framework. Ook is er een advies gegeven, indien het onderzoek groter wordt uitgepakt, welke javascript frameworks een nader onderzoek horen te krijgen.

11.2.2. SELECTEREN VAN METHODEN, TECHNIEKEN EN TOOLS

Tijdens de afstudeerperiode ben ik geheel vrij geweest om zelf te kiezen welke methodieken, technieken of tools ik zou gaan gebruiken voor het maken van het prototype. De methodiek dat is gekozen is Scrum, ook is er gekozen om TDD als test techniek te gaan gebruiken. Naast het gebruik van deze methodiek en techniek zijn er meerdere tools en design patterns onderzocht en toegepast.

11.2.3. ONTWERPEN SOFTWAREARCHITECTUUR

De architectuur van de applicatie is weergeven in meerdere klassendiagrammen. Er is naast de standaard analyse en design diagrammen ook gebruik gemaakt van een sequence diagram om de data flow goed weer te geven. Voor het ontwerpen van deze klassendiagrammen is ook hier gebruik gemaakt van een tool, genaamd Visual Paradigm. In deze diagrammen was het nodig om de interactie met de huidige situatie weer te geven, maar was het ook nodig om het prototype qua architectuur goed weer te geven.

11.2.4. ONTWERPEN SYSTEEMDEEL

De applicatie die is gebouwd is een standalone applicatie en kan om deze reden dan ook werken zonder dat deze geïntegreerd wordt in de huidige applicatie. Er zijn meerdere

webpagina's gemaakt waarbij rekening gehouden moest worden met meerdere resolutie groottes. Ook is er gebruik gemaakt van 2D animaties om zo een betere look en feel te geven aan de pagina's. Om dit alles voor elkaar te krijgen is er gebruik gemaakt van meerdere libraries, zoals Bootstrap, Less, Animate, CSS Modules, etc.

11.2.5. BOUWEN APPLICATIE

Zoals ook in het afstudeerplan lag de focus op het bouwen van een prototype van de feature match experience. Om dit voor elkaar te krijgen is er gebruik gemaakt van React en twee belangrijke design patterns. Namelijk een strategie pattern en een Redux design pattern. Ook is er rekening gehouden met het mogelijk maken van integratie in de huidige applicatie. Daarnaast is de applicatie event based gestuurd wat voor veel UI-state changes zorgt. Dit alles is gemaakt door gebruik te maken van meerdere programmeertalen.

11.2.6. UITVOEREN VAN HET TESTPROCES

Er is gekozen om gebruik te maken van TDD als test techniek. Naast deze techniek is er gebruik gemaakt van unit testen om de functionaliteiten te testen. Om de functionaliteiten te testen moest er gekeken worden naar test frameworks. Ook hier is een keuze in gemaakt. Het framework dat gekozen is, een combinatie van meerdere libraries. Namelijk; Mocha als test framework, Chai als assertion test framework en Sinon om spies en stubs te maken.

12. REFERENCIES

1. Gamebasics – Gamebasics b.v. (n.d.). Opgehaald op Maart 24, 2017, van http://www.gamebasics.nl/
2. Online Soccer Manager – OSM the game. (n.d.). Opgehaald op Maart 24, 2017, van http://us.onlinesoccermanager.com/Login
3. Scribbr – Plan van aanpak opstellen (n.d.). Opgehaald op Maart 24, 2017, van https://www.scribbr.nl/
4. XP – Extreme Programming (n.d.). Opgehaald op Maart 24, 2017, van http://www.extremeprogramming.org/
5. XP – Extreme Programming Rules (n.d.). Opgehaald op Maart 24, 2017, van http://www.extremeprogramming.org/rules.html
6. Scrum – Scrum (n.d.). Opgehaald op Maart 24, 2017, van http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100
7. Scrum – Scrum (n.d.). Opgehaald op Maart 24, 2017, van https://www.scrum.org/
8. TDD – Test Driven Development (n.d.). Opgehaald op Maart 24, 2017, van https://www.agilealliance.org/glossary/tdd/
9. TDD – Test Driven Development unit-testing (n.d.). Opgehaald op Maart 24, 2017, van https://www.agilealliance.org/glossary/unit-test/
10. TDD – Test Driven Development refactoring (n.d.). Opgehaald op Maart 24, 2017, van https://www.agilealliance.org/glossary/refactoring/
11. DSDM – dynamic system development method (n.d.). Opgehaald op Maart 24, 2017, van https://www.opengroup.org/architecture/wp/dsdm/DSDM_Framework_and_TOGAF.pdf
12. Github – Dan abramov Redux init release (n.d.). Opgehaald op Maart 27, 2017, van https://github.com/reactjs/redux/releases/tag/v0.2.0
13. Flux – Flux design pattern (n.d.). Opgehaald op Maart 27, 2017, van https://facebook.github.io/flux/
14. Flux – Indept info flux design pattern (n.d.). Opgehaald op Maart 27, 2017, van https://facebook.github.io/flux/docs/in-depth-overview.html#content
15. Flux – Flux explained (n.d.). Opgehaald op Maart 27, 2017, van https://code-cartoons.com/a-cartoon-guide-to-flux-6157355ab207#.v7lkrqro8
16. Stackoverflow – Redux why? (n.d.). Opgehaald op Maart 27, 2017, van http://stackoverflow.com/questions/32461229/why-use-redux-over-facebook-flux
17. Redux – Redux explained (n.d.). Opgehaald op Maart 27, 2017, van https://code-cartoons.com/a-cartoon-intro-to-redux-3afb775501a6#.rq70kfe4q
18. Webpack – Hot Reloading (n.d.). Opgehaald op Maart 27, 2017, van https://webpack.github.io/docs/hot-module-replacement.html
19. React – Hot Reloading (n.d.). Opgehaald op Maart 27, 2017, van https://facebook.github.io/react-native/blog/2016/03/24/introducing-hot-reloading.html
20. Wiki – MVC-design pattern (n.d.). Opgehaald op Maart 27, 2017, van https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
21. Softwareengineering – MVC-design pattern (n.d.). Opgehaald op Maart 27, 2017, van http://softwareengineering.stackexchange.com/questions/127624/what-is-mvc-really
22. Atom – Atom (n.d.). Opgehaald op Maart 27, 2017, van https://atom.io/
23. Wiki – Atom (n.d.). Opgehaald op Maart, 27, 2017, van https://en.wikipedia.org/wiki/Atom_(text_editor)
24. Wiki – Sublime Text (n.d.). Opgehaald op Maart, 27, 2017, van https://en.wikipedia.org/wiki/Sublime_Text

25. Sublime Text – Sublime Text (n.d.). Opgehaald op Maart, 27, 2017, van http://www.sublimetext.com/
26. Visual Studio Code – Visual Studio Code (n.d.). Opgehaald op Maart, 27, 2017, van https://code.visualstudio.com/
27. Wiki – Visual Studio Code (n.d.). Opgehaald op Maart, 27, 2017, van https://en.wikipedia.org/wiki/Visual_Studio_Code
28. Webstorm – Webstorm (n.d.). Opgehaald op Maart, 27, 2017, van https://www.jetbrains.com/webstorm/
29. Wiki – JetBrains-Webstorm (n.d.). Opgehaald op Maart, 27, 2017, van https://en.wikipedia.org/wiki/JetBrains
30. NPM – NPM (n.d.). Opgehaald op Maart, 28, 2017, van https://www.npmjs.com/
31. Yarn – Yarn (n.d.). Opgehaald op Maart, 28, 2017, van https://yarnpkg.com/en/
32. Yarn – Yarn explained (n.d.). Opgehaald op Maart, 28, 2017, van https://code.facebook.com/posts/1840075619545360
33. Freecodecamp – Module bundlers explained part 1 (n.d.). Opgehaald op Maart, 28, 2017, van https://medium.freecodecamp.com/javascript-modules-a-beginner-s-guide-783f7d7a5fcc
34. Freecodecamp – Module bundlers explained part 2 (n.d.). Opgehaald op Maart, 28, 2017, van https://medium.freecodecamp.com/javascript-modules-part-2-module-bundling-5020383cf306
35. Webpack – Webpack (n.d.). Opgehaald op Maart, 28, 2017, van https://webpack.github.io/docs/what-is-webpack.html
36. Slant – Top javascript module bundlers (n.d.). Opgehaald op Maart, 28, 2017, van https://www.slant.co/topics/3900/~frontend-javascript-module-bundlers
37. Namangoel – Webpack & Browserify explained (n.d.). Opgehaald op Maart, 28, 2017, van http://blog.namangoel.com/browserify-vs-webpack-js-drama
38. Wiki – Gulp (n.d.). Opgehaald op Maart, 28, 2017, van https://en.wikipedia.org/wiki/Gulp.js
39. Wiki – Browserify (n.d.). Opgehaald op Maart, 28, 2017, van https://en.wikipedia.org/wiki/Browserify
40. Browserify – Browserify explained (n.d.). Opgehaald op Maart, 28, 2017, van http://browserify.org/
41. Wiki – Mocha (n.d.). Opgehaald op Maart, 28, 2017, van https://en.wikipedia.org/wiki/Mocha_(JavaScript_framework)
42. Mocha – Mocha explained (n.d.). Opgehaald op Maart, 28, 2017, van https://mochajs.org/
43. Chai – Chai (n.d.). Opgehaald op Maart, 28, 2017, van http://chaijs.com/
44. Sinon – Sinon (n.d.). Opgehaald op Maart, 28, 2017, van http://sinonjs.org/
45. Wiki – Jasmine (n.d.). Opgehaald op Maart, 28, 2017 van https://en.wikipedia.org/wiki/Jasmine_(JavaScript_testing_framework)
46. Jasmine – Jasmine explained (n.d.). Opgehaald op Maart, 28, 2017 van https://jasmine.github.io/
47. NPM – Isomorphic-fetch (n.d.). Opgehaald op Maart, 28, 2017, van https://www.npmjs.com/package/isomorphic-fetch
48. Babel – Babel explained (n.d.). Opgehaald op Maart, 28, 2017, van https://babeljs.io/
49. W3schools – Cascading style sheets (n.d.). Opgehaald op April, 10, 2017, van https://www.w3schools.com/css/css_intro.asp

50. Wiki – Cascading style sheets (n.d.). Opgehaald op April, 10, 2017, van https://en.wikipedia.org/wiki/Cascading_Style_Sheets
51. Wiki – Markup Language (n.d.). Opgehaald op April, 10, 2017, van https://en.wikipedia.org/wiki/Markup_language
52. LESS – LESS explained (n.d.). Opgehaald op April, 10, 2017, van http://lesscss.org/
53. Wiki – LESS (n.d.). Opgehaald op April, 10, 2017 van https://en.wikipedia.org/wiki/Less_(stylesheet_language)
54. SASS – SASS explained (n.d.). Opgehaald op April, 10, 2017, van http://sass-lang.com/
55. Wiki – SASS (n.d.). Opgehaald op April, 10, 2017, van https://en.wikipedia.org/wiki/Sass_(stylesheet_language)
56. NPM – Bootstrap-LESS (n.d.). Opgehaald op April, 10, 2017 van https://www.npmjs.com/package/bootstrap-less
57. Bootstrap – Bootstrap CSS (n.d.). Opgehaald op April, 10, 2017, van http://getbootstrap.com/
58. GitHub – CSS-Modules (n.d.). Opgehaald op April, 10, 2017, van https://github.com/css-modules/css-modules
59. StackOverflow – Style-loader (n.d.). Opgehaald op April 10, 2017, van http://stackoverflow.com/questions/34039826/webpack-style-loader-vs-css-loader
60. Swagger – Web-api explained (n.d.). Opgehaald op April 10, 2017, van https://staging-api.onlinesoccermanager.com/swagger/ui/index
61. GitHub – Less-Loader (n.d.). Opgehaald op April 20, 2017, van https://github.com/webpack-contrib/less-loader
62. GitHub – PostCss-Loader (n.d.). Opgehaald op April 20, 2017, van https://github.com/postcss/postcss-loader
63. GitHub – Url-Loader (n.d.). Opgehaald op April 20, 2017, van https://github.com/webpack-contrib/url-loader
64. GitHub – File-Loader (n.d.). Opgehaald op April 20, 2017, van https://github.com/webpack-contrib/file-loader
65. GitHub – Css-Loader (n.d.). Opgehaald op April 20, 2017, van https://github.com/webpack-contrib/css-loader
66. Typescript – Typescript (n.d.). Opgehaald op Mei 8, 2017 van https://www.typescriptlang.org/
67. Redux – Redux (n.d.). Opgehaald op Mei 8, 2017, van http://redux.js.org/docs/introduction/Examples.html
68. de Swart. Nicole (2010). Handboek Requirements. Delft: Eburon
69. Pure-Im – Google markaandeel (n.d.). Opgehaald op February 7, 2017, van http://www.pure-im.nl/blog/marktaandelen-zoekmachines-q4-2016/
70. Dewebanalist – Google markaandeel (n.d.). Opgehaald op February 7, 2017, van http://www.dewebanalist.nl/marktaandeel-google-zoekmachine-september-2016
71. Wiki – MoSCoW method (n.d.). Opgehaald op Mei 9, 2017, van https://en.wikipedia.org/wiki/MoSCoW_method
72. NPM – Animate.less (n.d.). Opgehaald op Mei 15, 2017, van https://github.com/machito/animate.less
73. SlideShare – Agile development pros – cons (n.d.). Opgehaald op Mei 30, 2017, van https://www.slideshare.net/SoftwareStartUpAcademyOsijek/pm-scrum-and-tfs-ivan-markovi
74. Wiki – Ecmascript explained (n.d.). Opgehaald op Mei 30, 2017, van https://en.wikipedia.org/wiki/ECMAScript