Afstudeerdossier

Ontwikkelen van een platform voor games van de Game Factory

Ruben Deurloo

25 Maart 2022

CGI

Referaat

CGI is een groot wereldwijd IT consultancy bedrijf. De Game Factory van CGI heeft in de afgelopen jaren meerdere games ontwikkeld, waarbij elke game een eigen website en database heeft. Echter zorgt dat voor complicaties voor zowel CGI als hun klanten, waardoor de Game Factory aan mij de opdracht heeft gegeven om één platform te ontwikkelen voor alle games met één database.

Dit platform moest worden ontwikkeld met een microservice architectuur op Azure met één database, waarbij alle code is getest. Het platform moest de volgende onderdelen bevatten: inloggen door gebruikers, een dashboard met alle games van een gebruiker, spelen van games door gebruikers en een systeem dat bepaald wat een gebruiker mag en kan doen op het platform.

De opdracht was onderverdeeld in een opzet-en oriëntatie fase en een ontwikkel fase. In de opzet-en oriëntatie fase is het hele platform opgezet en in de ontwikkel fase zijn de onderdelen ontworpen en ontwikkeld.

In de opzet fase zijn er onderzoeken uitgevoerd, de Azure omgeving opgezet, de applicaties voor het platform aangemaakt, de requirements verzameld, het database diagram ontworpen en het Scrum backlog aangemaakt.

In de ontwikkel fase is het platform ontworpen en ontwikkeld via de Scrum methode in vier sprints van drie weken. Tijdens de sprints zijn er meerdere onderzoeken gedaan naar oplossingen voor problemen en is er veel communicatie geweest tussen mijn begeleider (tevens opdrachtgever) en mijzelf. Na de ontwikkel fase is het platform aan de opdrachtgever gedemonstreerd en opgeleverd, waarbij de opdrachtgever zeer tevreden was over het ontwikkelde platform dat zijn wensen voldeed.

De aanbeveling voor de CGI Game Factory is om het platform verder te ontwikkelen.

Voorwoord

Dit is het afstudeerdossier over het ontwikkelen van een platform voor de games van de Game Factory van CGI. Dit afstudeerdossier is geschreven met oog op mijn afstuderen voor de opleiding HBO-ICT aan de Haagse Hogeschool in Zoetermeer en in opdracht van bedrijf CGI. In de periode van September 2021 tot en met Maart 2022 ben ik bezig geweest met het schrijven van het afstudeerdossier en ontwikkelen van het platform.

Ik wil graag van deze gelegenheid gebruik maken om een paar mensen te bedanken.

Hierbij wil ik mijn begeleider, Rens van der Meijs, bedanken voor zijn medewerking en ondersteuning bij het uitvoeren van de opdracht. Hij stond altijd voor mij klaar om zowel technische als administratieve vragen te beantwoorden.

Tevens wil ik Ton Biegstraaten van de Haagse Hogeschool, bedanken voor zijn ondersteuning en medewerking bij het maken van dit afstudeerdossier. Ik kon hem altijd bereiken voor vragen of tips over het maken van dit afstudeerdossier en voor emotionele support.

Ik wens u veel plezier met het lezen.

Ruben Deurloo

Zoetermeer, 25 Maart 2022

Inhoudsopgave

1 Inleiding

De CGI Game Factory heeft meerdere games gemaakt, waar bedrijven accounts voor kunnen kopen. Momenteel heeft elke game een eigen website, wat problemen veroorzaakt in het kader van onderhoud en administratie.

Een groot deel van de code van de websites komt overeen met elkaar, dus als ze één website updaten, is er een grote kans dat ze de andere websites ook moeten updaten. Voor de klanten van CGI veroorzaakt het administratief problemen, wanneer ze meerdere games aanschaffen voor een groep medewerkers. Voor elke game heeft een medewerker dan een apart account. Dus met twee games, heeft één medewerker, twee accounts. Het kost CGI veel geld om voor elke game een website en een database te onderhouden.

CGI had mij daarom de opdracht gegeven, om één platform te maken waar gebruikers al hun games op kunnen spelen. Dit platform moet ik dan maken met een microservices architectuur voor de backend, Azure voor de database en het hosten van de applicaties en het Angular framework voor de frontend.

Het doel van de opdracht, is dat het platform er voor moet zorgen dat er geen code duplicatie meer is door alle games op één website te zetten, administratief tijd en kosten te besparen en het aantrekkelijker maakt voor klanten, om meer dan één game aan te schaffen.

Dit project zal worden uitgevoerd door eerst een oriëntatie- en opzet fase te doen, waar ik alle wensen van de opdrachtgevers opstel, de Cloud omgeving opzet, de werkomgeving opzet, de database maak en de werkmethode opzet. Daarna zullen de wensen van de opdrachtgevers, in vier blokken van drie werkweken worden geïmplementeerd.

Eerst zal ik een korte geschiedenis van CGI beschrijven en mijn plaats in CGI tijdens dit afstudeertraject in hoofdstuk twee. In hoofdstuk drie zal ik de opdracht beschrijven. Daarna zal ik in hoofdstuk vier beschrijven, hoe de situatie er voor stond bij aanvang op het bedrijf. In hoofdstuk vijf zal ik alle middelen beschrijven waar ik gebruik van hebt gemaakt tijdens het uitvoeren van de opdracht. Daarna zal ik in hoofdstuk zes beschrijven hoe ik de opdracht hebt aangepakt.

In hoofdstuk zeven zal ik de afspraken beschrijven die ik met mijn opdrachtgevers gemaakt had in de oriëntatiefase. Dan zal ik in hoofdstuk acht de opzet fase beschrijven, met de voortgang van elke week van die fase. In hoofdstuk negen zal ik het eerste blok werk beschrijven, waar het eerste systeem en het inloggen door gebruikers is ontwikkeld. Het blok met het onderzoeken en ontwerpen van het tweede systeem, zal ik beschrijven in hoofdstuk tien. In hoofdstuk elf zal ik het implementeren van het tweede systeem beschrijven. Het ontwikkelen van de weergave van games die gebruikers kunnen spelen en de rechten daarvoor van het vierde blok werk, zal ik beschrijven in hoofdstuk twaalf.

Het reflecteren op het afstudeertraject zal ik beschrijven in hoofdstuk dertien. In hoofdstuk veertien zal ik evalueren op het eindproduct en de gekozen aanpak. De aanbeveling die ik had voor CGI over het platform zal ik beschrijven in hoofdstuk vijftien. In hoofdstuk zestien zal ik aantonen aan welke beroepstaken ik hebt voldaan en waarom. De lijst met figuren en de bronnenlijst zijn terug te vinden in hoofdstuk zeventien en achttien.

2 CGI

In dit hoofdstuk zal ik eerst de geschiedenis van CGI vertellen en wat CGI doet, daarna zal ik mijn plaats binnen het bedrijf beschrijven. Als laatste zal ik beschrijven met welke medewerkers ik het meest contact zal hebben en wat hun rollen zijn.

2.1 Geschiedenis van CGI

CGI is in 1976 opgericht door Serge Godin en André Imbeau in Quebec City in Canada. CGI groeide over de jaren en begon snel met het aanbieden van systeem integratie naast consultancy. Ze kregen ook een aantal overheidscontracten in handen. Tegen het eind van de jaren 80 was CGI verder aan het uitbreiden met de aankoop van meerdere kleine bedrijven en begonnen ze zich uit te breiden buiten Canada.

In 1994 kreeg CGI ISO 9001 certificaat waardoor ze het eerste IT consultancy bedrijf in Noord-Amerika waren die aan de ISO standaard voldeed. Bij het jaar 1997 had CGI na aankoop van CDSL Holdings Limited 2500 medewerkers.

Bij het jaar 2000 had CGI ook banken als cliënten en had CGI ondertussen 25.000 medewerkers. In de jaren die volgden kocht CGI meerdere bedrijven op waardoor ze wereldwijd een groter bereik en meer contracten kregen bij de overheid en in de commerciële sector.

In 2012 had CGI Logica gekocht waardoor ze van 31.000 medewerkers naar 68.000 medewerkers gingen. Dit zorgde er ook voor dat ze nummer 5 grootste processen en IT consultancy bedrijf in de wereld werden.

Momenteel heeft CGI meer dan 78.000 medewerkers verspreid over 400 locaties in meer dan 29 landen. (CGI, 2021)

2.2 Mijn positie in CGI

Ik ga mijn positie in CGI verduidelijken met behulp van).

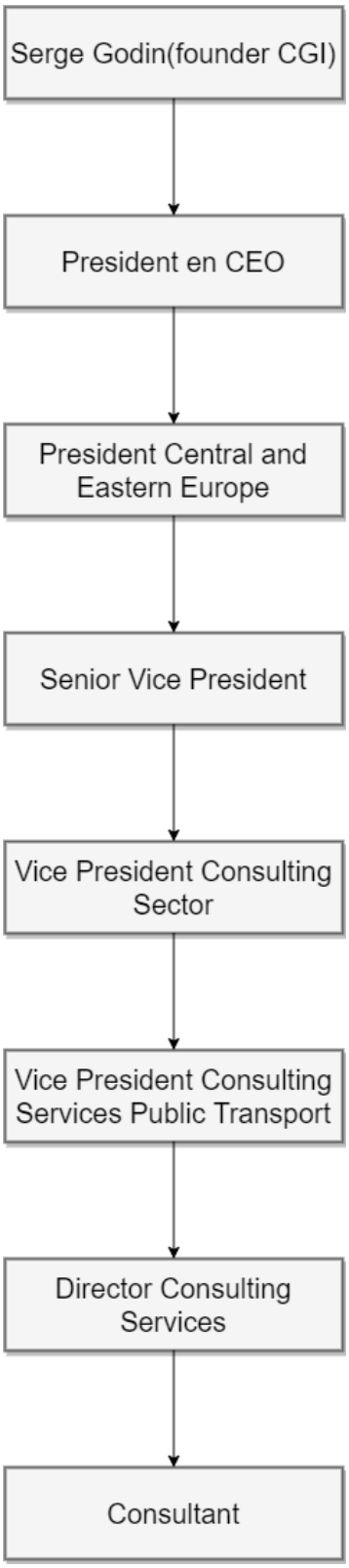
Als je een bundel wil downloaden van deze server dan moet je de folder waar dat bestand in zit meegeven in het configuratie bestand van Angular.

Wanneer je het project dan start via de Angular CLI, wordt dat bestand ook meegenomen wanneer de applicatie gebuild wordt.

Stel je hebt een bundel genaamd testbundel in het project zitten, op de locatie “src/assets/testbundel” en je wilt die downloaden. Je moet dan de testbundel toevoegen aan de assets folder van de applicatie en het pad “src/assets/testbundel” toevoegen in het configuratie bestand van de Angular applicatie.

Wanneer de applicatie gestart is kan je een GET aanvraag sturen naar http://localhost:4200/assets/testbundel. Hierbij zal ik dan beschrijven van de top van de hiërarchie naar mijn positie. Aan de top van de hiërarchie bevindt zich Serge Godin als executive chairman. Onder Serge Godin in de hiërarchie valt de President en CEO George D Schindler.

CGI is een wereldwijde organisatie en heeft de delen van de wereld verdeeld in delen die ieder een eigen President heeft. Die delen zijn dan onder andere Central and Eastern Europe, United States en Asia Pacific. De president waar ik ver onder zit is die van Central & Eastern Europe.



Figuur 1: Hiërarchie CGI

Elk van die delen zijn weer verdeeld in landen, delen van landen en collecties van landen. Elk deel heeft een Senior Vice President als hoofd. Onder de president van Central & Eastern Europe vallen dan onder andere

Nederland, West Duitsland, Noord Duitsland en de collectie van Tsjechië, Slowakije & Oost Europa. De Senior Vice President waar ik onder val is die van Nederland.

Het gebied van een Senior Vice President is dan verdeeld in sectoren. Elke sector heeft heeft als hoofd een Vice President Consulting Sector. De Vice Presidents Consulting Sectoren die onder de Senior Vice President vallen van Nederland zijn onder andere Randstad Government, Randstad Vital Civil Infrastructure, Randstad Transport & Logistics, Energy & Communications & North, Eindhoven – Maastricht. De Vice President Consulting Sector waar ik onder val is van de sector Randstad Vital Civil Infrastucture.

Elke sector is dan weer onderverdeeld in consulting services. Elke consulting service heeft een Vice President Consulting Services als hoofd. Onder de sector Randstad Vital Civil Infrastructure vallen onder andere de consulting services Ultilities & Water, Infrastructuur en Rail Infrastructuur. De Vice President Consulting Services waar ik onder val is van Infrastructuur.

Elke Vice President Consulting Services heeft managers onder hen zitten die Director Consulting Services heten. Onder elk van die managers vallen dan de Consultants. Consultant is de positie die ik heb binnen CGI.

Ondanks dat ikzelf, mijn begeleider en de product owner allemaal bij de Game Factory horen vallen we alle drie onder een andere Director Consulting Services. Dit komt, omdat de Game Factory een initiatief is van een aantal Directors Consulting Services om meer Gamificatie services aan te kunnen bieden aan cliënten. Met gamificatie pas je de technieken en ideeën van games toe om een educatieve applicatie te maken met spelelementen om mensen iets te leren. Voorbeelden van een paar games die de Game Factory maakt zijn een game om security te verbeteren en een game om benzinekosten te verlagen.

Hoe CGI normaal werkt is dat ze een opdracht krijgen van een cliënt die er dan een budget voor opstelt. Bij de Game Factory werkt het andersom. De Game Factory maakt dan games die ze aanbieden aan cliënten. Dit is omdat veel bedrijven geen of weinig verstand hebben van gamificatie en daardoor niet CGI benaderen met een opdracht (bron van deze alinea is CGI medewerker).

2.3 Medewerkers in mijn omgeving

Tijdens mijn afstudeertraject zal ik regelmatig contact hebben met een aantal medewerkers. Hier zal ik per medewerker zijn rol beschrijven.

Rens van der Meijs:
Rens van der Meijs is een developer bij de Game Factory en mijn begeleider. Hij is ook samen met Adolfo Sanchez de product owners van mijn opdracht.

Adolfo Sanchez:
Adolfo Sanchez is een developer bij de Game Factory en de product owner bij mijn opdracht.

Maurice Klein:
Maurice Klein (eerst was het Tiziano Quintarelli) is mijn DCS (Director Consulting Services) bij CGI. Bij hem kan ik terecht voor vragen over onder andere licenties en budget.

3 De opdracht

In dit hoofdstuk zal ik de opdracht beschrijven. Eerst zal ik de probleemstelling voorleggen. Vervolgens zal ik de opdracht beschrijven. Dan zal ik het doel van de opdracht beschrijven. Als laatste zal ik beschrijven wat er met het eindproduct zal gebeuren

3.1 Probleemstelling

In de afgelopen jaren heeft de Game Factory meerdere games ontwikkeld met Unity. Al deze games hebben hun eigen website, waarbij de website gemaakt is met het Angular framework. Op deze websites kunnen de games gespeeld worden door gebruikers, waarop een gebruiker bij de meeste games eerst ingelogd moet zijn, om de game te spelen.

Deze situatie zorgt voor veel problemen. Ten eerste kost het de Game Factory veel tijd om al die websites te onderhouden. Dit komt doordat de web code van de websites veel overeenkomsten met elkaar hebben, waardoor je bij een aanpassing bij één website hoogstwaarschijnlijk ook alle andere websites moet aanpassen.

Als tweede kost het ook veel geld om voor elke game een aparte website en database te hebben.

Het derde probleem is dat wanneer een bedrijf twee games beschikbaar wilt maken voor een groep medewerkers, het voor elk van die medewerkers een account aangemaakt moet worden op de website van beide games. Dit zorgt ervoor dat elk van die medewerkers dan voor beide games een apart account heeft. Het laatste probleem met een database per game hebben is dat het extra tijd kost om accounts te beheren op meerdere websites tegelijk.

3.2 Opdrachtbeschrijving

De opdracht die ik van CGI gekregen heb, is om een platform te maken waar alle games van de Game Factory op kunnen staan en gebruikers die games kunnen spelen. Dit platform moet dan gemaakt worden in de programmeertaal TypeScript met een NodeJS environment. De backend van het platform moet gemaakt worden met microservices. De opdracht gever wil dat de website zo dynamisch mogelijk is, waardoor het platform elke game die ze nu hebben of in de toekomst zullen maken aan zou kunnen. Voor het platform moeten ook testen geschreven worden.

3.3 Doelstelling

De doelstelling is dat het platform ervoor moet zorgen dat er voor alle games maar één database met één website nodig is. Daarnaast is het doel dat de microservices van het platform ervoor zorgen dat er minder code duplicatie is tussen de games om het beter te kunnen onderhouden. Het derde doel is om gebruikers toegang te geven tot al hun games met maar één account in plaats van dat ze voor elke game een ander account hebben.

3.4 Resultaat

De opdracht is om een concept platform op te leveren, dat de Game Factory (als ze dat willen) verder kan ontwikkelen. Indien het concept succesvol is en er vraag naar is, zal de Game Factory het platform verder ontwikkelen om vervolgens alle games en databases naar toe te migreren.

4 Situatie bij aanvang

Op de eerste werkdag was er een Joinersdag. Op de dag kreeg ik een paar presentaties over het ontstaan van CGI tot en met nu en hoe de organisatie eruit ziet van de top naar de bodem van de organisatie boom. Op deze dag kreeg ik ook mijn werk laptop die ik zal gebruiken om de opdracht op uit te voeren.

Kort daarna had ik een gesprek met mijn opdrachtgevers, waarin we meerdere onderwerpen besproken over de huidige stand van zaken. De belangrijkste onderwerpen waren middelen die ik bij dit project zou gaan gebruiken, licenties, een wijziging van het plan van aanpak (zie Bijlage 1: Afstudeerplan voor plan van aanpak), wensen voor een Definition of Done en een Definition of Ready en testen.

Ik zal in dit hoofdstuk deze onderwerpen beschrijven.

4.1 Middelen

Bij het uitvoeren van mijn opdracht zal ik gebruik maken van meerdere middelen. Tijdens het plannen van dit afstudeertraject, wist ik al meerdere middelen waar ik gebruik van moest maken bij het uitvoeren van mijn opdracht (een aantal van deze middelen zijn benoemd op Bijlage 1: Afstudeerplan).

In het gesprek kreeg ik te horen dat ik GitLab in plaats van BitBucket moest gaan gebruiken, om de code op te bewaren. Deze verandering kwam intern uit CGI, omdat ze over wilden stappen van BitBucket naar GitLab, maar het was onbekend voor mij waarom deze beslissing was gemaakt of door wie.

Ik wist niet wat voor consequenties deze verandering ging hebben op de opdracht, maar ik verwachtte dat er weinig of geen problemen zouden ontstaan, want de manier om code op te slaan was op beide platformen hetzelfde. Het zou kunnen dat er problemen ontstaan, bij het maken van een pipeline bij het opzetten van de ontwikkel omgeving, want hoe je een pipeline opzet kon bij beide platformen anders zijn.

De volledige lijst van voorgeschreven middelen was als volgt:

- Angular framework gebruiken voor de frontend.
- NodeJS gebruiken als environment.
- Azure gebruiken voor opslag, SQL database en hosten van de website en de backend.
- Scrum methodiek toepassen.
- Jira gebruiken om scrum in bij te houden.
- GitLab gebruiken voor opslag van de code.
- TypeScript gebruiken als scripttaal.
- Microsoft Teams voor communicatie.

4.2 Licenties

Ik kreeg te weten dat ik voor Jira, Azure en GitLab nog licenties nodig had. Deze licenties konden pas aangevraagd worden zodra ik een CGI account had. Dit account kreeg ik pas op de eerste werkdag, dus die licenties konden helaas niet van tevoren worden aangevraagd.

Het aanvragen van deze licenties was ook wat problematisch. Dit kwam omdat je normaal gesproken een budget hebt van een klant waaraan CGI vervolgens een product code aan kan toewijzen waar medewerkers vervolgens licenties voor aan kunnen vragen. Omdat deze opdracht vanuit CGI zelf kwam was er geen officieel budget aanwezig, waardoor er geen product code was, waardoor je geen licentie kon aanvragen. Ondanks dat mijn opdrachtgevers hier hard achteraan gingen was het nog onbekend wanneer ik die licenties zou ontvangen.

4.3 Wijziging van originele plan van aanpak

Bij het afstudeerplan stond er bij het plan van aanpak een leaderboard bij (voor het afstudeerplan zie Bijlage 1: Afstudeerplan). Dit hield in dat er voor elke game een leaderboard moest komen waarop de scores te zien zijn van elke gebruiker die de game behaald heeft.

Bij het gesprek kreeg ik te weten, dat het leaderboard niet meer de hoogste prioriteit had en een ander onderdeel in zijn plaats wilden hebben.

In plaats van een leaderboard wilden ze een activiteiten systeem. Zover ik begreep wilden ze dat het activiteiten systeem de activiteiten van een gebruiker kan dirigeren. Door bijvoorbeeld een nieuwe gebruiker een tutorial te geven hoe de website werkt of wanneer een gebruiker een spel voor het eerst start dat hij een tutorial van het spel krijgt of dat het kan zeggen tegen de gebruiker dat hij tot een bepaalde datum heeft om een bepaald level van een spel te doen.

4.4 Definition of Done en Definition of Ready

Bij het gesprek kreeg ik te weten dat de opdrachtgevers een Definition of Done en een Definition of Ready wilden hebben. De Definition of Done en Definition of Ready zijn afspraken die gaan over de user stories van Scrum (voor meer informatie over scrum zie hoofdstuk 5.1 Scrum).

De Definition of Ready zegt waar een user story aan moet voldoen voordat de user story opgepakt kan worden in een sprint. De Definition of Done zegt waar een opgeleverde user story aan moet voldoen voordat de user story de status Done mag krijgen.

4.5 Testen

Bij het gesprek hadden wij het gehad over wat voor testen ik moet maken voor dit project, waar uit kwam dat ik unit testen en end-to-end testen moet maken.

5 Voorgeschreven en gekozen middelen

In dit hoofdstuk ga ik de middelen beschrijven waar ik tijdens dit project gebruik van zal maken die mij voorgeschreven zijn. Ik zal ook een paar middelen beschrijven waar ik zelf voor gekozen heb en middelen die deel uitmaken van Angular. In dit hoofdstuk zal ik ook beschrijven voor elk middel waarom er voor dat middel gekozen is door ofwel CGI of mijzelf.

5.1 Scrum - voorgeschreven

Scrum is een werkmethode om Agile te werken.

Agile is een iteratieve methode om software te ontwikkelen dat software stapsgewijs oplevert in plaats van helemaal in één keer (What is Agile?, 2022). Met Agile staat klanttevredenheid voorop, waarbij organisaties in staat zijn snel in te spelen op veranderingen in wensen, eisen en prioriteiten (Scrum vs agile, 2022).

Met Scrum werk je in sprints met een backlog van user stories. Een sprint kan één week tot een maand duren. Elke sprint plan je dan user stories in die de product owner de hoogste prioriteit vindt. Een user story is een beschrijving met wat de product owner wilt en waarom. De hele lijst met user stories is dan de backlog en is geprioriteerd waarbij de belangrijkste user stories bovenaan staan.

Deze manier van werken zorgt ervoor dat je flexibel bent en altijd bezig bent met hetgeen dat de product owner het eerst wilt hebben. Scrum is dus ook handig bij projecten waarbij het niet in te schatten is hoelang het duurt om het project te voltooien.

Een alternatief van Scrum is de waterval methode. Bij de watervalmethode doorloop je achter elkaar verschillende fasen. De fasen zijn in volgorde van eerst tot laatste analyse, ontwerp, implementatie, testen en onderhoud. Eerst in de analyse fase wordt het probleem geanalyseerd om tot eisen te komen die de applicatie moet hebben. Dan wordt er in de ontwerp fase een ontwerp gemaakt van de hele applicatie. Dat ontwerp wordt vervolgens geïmplementeerd in de implementatie fase. De code van de applicatie wordt dan getest in de test fase. De laatste stap is dan onderhoud waarbij de applicatie onderhouden wordt zodat het blijft functioneren (Waterval-methode, 2021).

Het is mogelijk met de waterval methode dat een opdrachtgever een opdracht geeft en een jaar later pas wat terug hoort over die opdracht. Met de watervalmethode kan een opdrachtgever ook geen nieuwe wensen meer toevoegen aan het project wanneer het begonnen is.

De voorgeschreven methode Scrum was een goede keuze geweest van CGI vanwege de bovenstaande redenen. Mijn opdracht is een concept maken dat nog vele jaren in ontwikkeling kan zijn. Omdat je met Scrum het werk in sprints opdeelt kan je snel een andere werknemer op het project zetten dan wanneer je een lange fase hebt, waarin je alles in één keer implementeert. Zo begint de nieuwe werknemer met code die ontworpen, geïmplementeerd en getest is waarop verder gebouwd kan worden in plaats van code en/of ontwerpen die nog maar half klaar zijn.

De keuze om scrum te gebruiken, was voorgeschreven door CGI en had ik geen inspraak op. Ik kon zelf geen bezwaar bedenken om Scrum niet te gebruiken, dus had ik maar weinig tijd besteed aan een onderzoek naar verschillende werkmethoden.

5.2 MoSCoW - gekozen

De MoSCoW methode is een methode om een backlog te prioriteren. Met de MoSCoW methode priorteer je door de user stories in te delen in de groepen Must-Have, Should-Have, Could-Have en Won't-Have. Must-Have user stories moeten gedaan worden. Zonder de Must-Have stories zal het eindproduct van een project niet kunnen werken. De Should-Have user stories zijn belangrijk maar niet cruciaal. Het eindproduct zal wel werken zonder deze stories maar er zullen wel belangrijke onderdelen ontbreken. User stories die onder Could-Have vallen zijn wel gewenst maar niet zo belangrijk als Should-Have user stories. Zonder de

Should-Have stories zal het eindproduct wel functioneel zijn met de belangrijkste onderdelen, maar er zullen wel wat minder belangrijke onderdelen ontbreken.

MoSCoW is één van de meest populairste methoden voor het prioriteren van een backlog. Het is ook de simpelste methode voor het prioriteren van een backlog voor projecten op kleine schaal (The Most Popular Prioritization Techniques and Methods, 2022). Met het maken van het platform werk ik alleen met wat begeleiding. Met MoSCoW kan je eenvoudig heel snel een backlog zelfstandig prioriteren.

Er bestaan meerdere manieren om te een backlog te prioriteren (The Most Popular Prioritizatoin Techniques and Methods, 2022). Ik had zelf voor MoSCoW gekozen, omdat ik zelf kennis en ervaring had met deze methode en MoSCoW een eenvoudige en snelle methode is om een backlog te prioriteren voor projecten op kleine schaal.

5.3 Azure - voorgeschreven

Azure is een Cloud omgeving van Microsoft waar je applicaties op kan hosten. Het bied veel verschillende services aan waar je gebruik van kan maken bij het ontwikkelen van je applicaties. Het bied onder andere services aan om databases te hosten, bestanden op te slaan en te downloaden, security en nog veel meer. Ik ga onder andere de services storage, container register, SQL databases gebruiken. Op Azure kan ik met de services onder andere de SQL database maken en hosten, de API en microservice containers deployen, de games opslaan en de frontend applicatie op hosten.

Het gebruik van Azure was aan mij voorgeschreven door CGI. CGI had wel een goede keuze gemaakt om gebruik te maken van Azure voor dit project, want Azure bood meer dan genoeg services om dit project op volledig op te zetten. Op Azure kon de frontend van de applicatie een URL krijgen waardoor het gelijk online zou staan. Zowel de frontend applicatie en de backend applicaties kon ik op Azure hosten. Op Azure konden alle games opgeslagen worden en kon de database bevatten. Azure bood dus alle services, die bij dit project nodig waren, aan op één platform.

Ik besloot geen tijd te besteden aan onderzoek naar andere Cloud services, omdat ik zelf al positieve ervaring had met Azure, geen bezwaar kon bekenken om Azure niet te gebruiken en ik zelf de Cloud omgeving niet kon kiezen, omdat ik Azure moest gebruiken.

5.4 Docker - gekozen

Om container instanties te maken in Azure van images moest ik eerst een programma hebben dat images kan maken. Een Image bevat een package van een app die je dan in een container kan draaien. Een container in Docker zorgt er voor dat de app lokaal toegankelijk wordt alsof het op een server staat. De data van een container kan opgeslagen worden in een volume.

Images specificaties en de runtime specificaties voor het uitvoeren van die images als containers voor de Azure Container Registry moeten voldoen aan de Open Container Initiative(OCI) of van Docker vandaan komen om geüpload te mogen worden (Container registry, 2021). Het OCI is een structuur voor het maken van industrie standaards voor container en image formaten (Open Container Initiative, 2021).

Docker is een gratis programma dat in staat is Images te maken en containers te draaien met volumes. Gemaakte images kan je met de Docker command line pushen naar de Azure Container Registry. Die images in de Azure Container Registry ga ik dan gebruiken om container instanties van te maken in Azure (5.3 Azure).

Ondanks dat er meerdere alternatieven voor Docker beschikbaar zijn zoals bijvoorbeeld red hat buildah (buildah, 2021), ORAS (Oras introduction, 2021) en Podman (What is Podman?, 2021), maakte het bij dit project niet veel uit voor welke software ik ging, zolang het maar images kan maken die aan de OCI standaarden voldoen en containers van die images kan draaien (Azure container registry, 2021). Het grote verschil tussen de verschillende programma's die images kunnen maken die voldoen aan de OCI standaarden is de manier hoe en waarop ze de containers draaien. Dit maakt voor deze opdracht niet uit, want het plan is om alle containers in Azure te draaien.

Zelf had ik gekozen om Docker te gebruiken, omdat het de Docker command line had wat het makkelijk maakt containers naar Azure te pushen. Daarnaast was ik ook al bekend met Docker waardoor ik geen nieuw programma hoefde te leren gebruiken, wat dan onnodig tijd zou hebben gekost.

5.5 TypeScript - voorgeschreven

TypeScript is een superset van JavaScript dat ontwikkeld en onderhouden wordt door Microsoft (Typescript, 2021). Dit betekent dat TypeScript alle functionaliteit van JavaScript heeft en is uitgebreid en/of verbeterd met andere functionaliteit (superset, 2021).

TypeScript volgt de object georiënteerd programmeren(OOP) principes en is heeft een beter typesysteem dan JavaScript. Dit houdt in dat je ziet wanneer een type niet klopt of ontbreekt bij een variabele. Door dit soort fouten te detecteren wanneer ze gemaakt worden zijn ze sneller te repareren dan wanneer je ze ontdekt bij het runnen van de app. TypeScript maakt de code dus beter te onderhouden en kwalitatief beter.

Ondanks dat het meer tijd zou kosten tijdens het maken van de app, zal het gebruik van TypeScript uiteindelijk tijd besparen doordat de fouten al eerder gedetecteerd worden. De script taal die Angular gebruikt is ook TypeScript. Door de redenen die hierboven beschreven zijn, was er gekozen TypeScript te gebruiken.

TypeScript transpiled al zijn code naar JavaScript, maar je kan zelf kiezen naar welke versie van JavaScript. Wanneer een nieuwe versie van JavaScript uitkomt kan het jaren duren voordat de browsers die nieuwe versie accepteren. TypeScript kan dan al wel de nieuwste versie van JavaScript gebruiken, omdat het dan alle code kan transpilen naar een eerdere versie van JavaScript.

Dit is een zeer handig onderdeel van TypeScript, want dan kan de code altijd de nieuwste versie gebruiken en als de browser dan over zal gaan naar een nieuwe versie, hoef je alleen maar de JavaScript versie waar je naar transpiled aan te passen.

Dit zorgt ervoor dat de code eenvoudiger te onderhouden is, omdat je dan niet de code hoeft aan te passen als er een nieuwe versie van JavaScript uitkomt.

Het platform maken in TypeScript was mij voorgeschreven, maar zelf zou ik ook voor TypeScript gekozen hebben. Ik zou hier zelf ook voor gekozen hebben, omdat Angular TypeScript gebruikt en TypeScript code kwalitatief goed is door de hoge type veiligheid.

5.6 Angular - voorgeschreven

Angular is een TypeScript frontend framework voor het maken van web applicaties. Met Angular kan je losse componenten maken die HTML, CSS en TypeScript code hebben. Die componenten kan je dan gebruiken in je HTML bestand van je website die je wilt laten zien. Je kan dan bijvoorbeeld een button component maken en dan 10 van die buttons aanmaken op je website.

Hierdoor krijg je een zeer object georiënteerde applicatie want je laat die componenten alleen maar zien en elk component doet zijn eigen taak (What is angular?, 2021). Door de frontend te verdelen in componenten is de code makkelijker te unit testen want elk component doet zijn eigen taak. Dit zorgt er ook voor dat de code beter te onderhouden is want individuele componenten zijn beter te onderhouden dan grote classes die veel doen (The good and the bad of angular, 2021).

Angular heeft gedetailleerde documentatie en heeft ondersteuning van Google. Mijn opdrachtgevers hadden ook communicatie ontwikkeld tussen de game instantie in de browser en de applicatie (waar de game op draait) dat gemaakt is in Angular.

Deze communicatie hadden ze gemaakt omdat de unity game instantie die draait in de browser zelf niet bij de front-end kon komen. Hierdoor kon je dus nooit op een start knop drukken of een naam van de frontend naar de game instantie sturen. Wat unity wel toestaat is het oproepen van een aantal native-javascript

functies op de pagina. Met die functies hebben zij een messaging bus opgezet tussen de frontend en de unity game.

Hoewel je wel web calls vanuit de unity game kon doen gebruikten ze dit niet, omdat daar beperkingen op zaten. Toen ze dit gebruikte in het verleden, hadden ze problemen gehad met het versturen van tokens tussen de game instantie en de browser. Dit probleem kwamen sommige klanten tegen die volgens hun Security Policy die calls vanuit de unity game niet toestonden maar alleen calls toestonden vanaf de pagina zelf (de bron hiervan is een gesprek met mijn begeleider).

De reden dat ze voor Angular hadden gekozen, was dat bij het eerste project wat ze kregen, een frontend developer hadden die Angular ervoor gebruikte. Sinds dien hadden ze de rest van hun games ook gemaakt met het Angular framework.

Een alternatief van Angular is het React framework van Facebook. React is ook op componenten gebaseerd net als Angular (Angular vs react vs vue, 2021). React en Angular hebben alle twee ook een hoge performance en kunnen beiden TypeScript aan (Angular vs react, 2021).

Hoewel React een minder steile learning curve heeft dan Angular is de learning curve ongeveer even groot als Angular wanneer React Redux van toepassing is (React vs angular key difference, 2021). React Redux zou bij deze opdracht zeer handig zijn als ik React zou gebruiken, om de status van de app bij te houden voor bijvoorbeeld inlog sessies van gebruikers.

Ook een groot verschil is dat React een one-way data binding heeft en Angular two-way data binding (React vs angular key difference, 2021). Met one-way data binding veranderen UI elementen niet zonder eerst een update te geven aan de status van het model van die elementen. Met two-way data binding wordt het model geüpdatet wanneer de view verandert en visa versa (Difference between one-way and two-way databinding, 2021).

Neem bijvoorbeeld het bijhouden van elke ingevulde letter van een tekst box als voorbeeld. Met one-way data binding moet je dan met elke ingetypte letter een update sturen naar de model om die te updaten om vervolgens de view te kunnen updaten. Met two-way data binding kan je een letter intypen die de view dan ziet en de update stuurt naar de model. Two-way data binding is dus veel handiger bij websites waarbij veel interactie is zoals bijvoorbeeld bij een platform met games.

Een andere alternatief dat net als Angular two-way data binding heeft is Vue (Vue is good, but is it better then angular or react?, 2021). Vue en React hebben beide server-side rendering terwijl Angular cliënt-side rendering heeft. Met server-side rendering wordt de hele pagina in één keer geladen. Met cliënt-side rendering wordt eerst een lege pagina geladen waarna dan de rest van de website wordt opgehaald, waarna alles weergegeven wordt. Een voordeel van cliënt-side rendering is dat de hele UI niet opnieuw geladen hoeft te worden bij elke call dat naar de backend gedaan wordt. Wanneer je bijvoorbeeld privileges ophaalt van een game wanneer je op die game klikt verandert de UI in plaats van dat je de UI eerst moet herladen (Client vs server side rendering, 2021).

Ondanks dat Angular aan mij was voorgeschreven door de opdrachtgevers, zou ik zelf ook Angular gekozen hebben, vanwege de bovenstaande redenen, maar voornamelijk vanwege de communicatie bus die mijn begeleider gemaakt heeft tussen Angular en Unity.

Hoewel het Angular framework aan mij was voorgeschreven, had ik toch onderzoek gedaan naar frontend frameworks, omdat het mogelijk was dat een ander framework beter zou zijn in deze situatie. Dit komt, omdat dit platform bedoeld was voor meerdere games, terwijl elke game momenteel een eigen applicatie heeft. Het was dus mogelijk dat in de nieuwe situatie, waarbij alle games op één platform staan, een ander framework beter zou zijn geweest, dan het Angular framework.

5.7 NodeJS - voorgeschreven

NodeJS is een cross-platform, open source runtime environment (What is node.js, 2021). NodeJS wordt voornamelijk gebruikt voor servers en de backend van applicaties en API's. Het kan onder andere connecties

open zetten en requests versturen (About Node.js, 2021). Ik ga dit dan gebruiken om verbindingen te maken en requests te sturen in de applicatie, API en microservices. Het Angular framework draait ook op de NodeJS environment.

NodeJS is gekozen vanwege meerdere redenen. Als eerste zijn de applicaties van de Game Factory gemaakt met Express dat draait op NodeJS. Dit maakt het voor CGI overzichtelijker en beter te onderhouden doordat alle code op dezelfde environment draait. Ten tweede draait Angular op NodeJS waardoor je geen keuze hebt voor environment en NodeJS wel moet gebruiken. Zelf heb ik ook al eerder met NodeJS gewerkt en heb er goede ervaringen mee.

CGI had mij de NodeJS environment voorgeschreven, maar ik zou zelf waarschijnlijk ook voor NodeJS gekozen hebben als ik de keuze voor environment kon maken.

5.8 ESLint - gekozen

ESLint helpt met implementeren van code doordat je zelf regels kan toevoegen voor onder andere naamgeving(dat je bijvoorbeeld alle namen PascalCase hebt van public functies), quotes en witte regels. Wanneer je de regels overtreed zal ESLint dan een error melding geven alsof je syntax niet klopt met de regel die je daar overtreed. ESLint zit standaard bij Angular inbegrepen maar je kan ook de keuze maken om ESLint niet te gebruiken.

Ik had gekozen ESLint wel te gebruiken om mij te helpen het platform zo consistent mogelijk te ontwikkelen en hogere kwaliteit software op te leveren. ESLint kan ook fouten detecteren die je normaal alleen tijdens het draaien van de code tegenkomt. Dat zal tijd besparen bij het uitvoeren van dit project, omdat ik die fouten dan in een eerder stadium kan herstellen. Om deze redenen had ik gekozen ESLint te gebruiken.

5.9 Jasmine en Karma - gekozen

Angular maakt gebruik van Jasmine en Karma voor het testen van de code. Jasmine is het framework waar de testen van Angular in gemaakt zijn. Karma is een tool die de testen van (in dit geval) Jasmine draaien.

Jasmine maakt gebruik van Behaviour-Driven-Development (BDD). BDD houdt in dat de testen op een formaat geschreven worden dat zelfs als je niet technisch bent dat het een geeft van wat er getest wordt (Jasmine and Karma, 2022). Jasmine heeft meerdere functies die je kan gebruiken om testen in op te zetten en af te breken. Hierin zou je bijvoorbeeld componenten kunnen aanmaken en verwijderen die je gebruikt voor de testen. Jasmine bevat alles wat je nodig hebt om testen te maken zonder extra onderdelen nodig te hebben (Jasmine hoofdpagina, 2022).

De testen van Jasmine zou je handmatig in een browser kunnen draaien maar dan zou je elke keer de pagina moeten herladen wanneer je de testen opnieuw wilt draaien. Karma kan browsers openen om de testen van Jasmine in te draaien en herlaad automatisch wanneer er veranderingen zijn in de code van zowel de applicatie als de testen.

Wanneer je een Angular applicatie aanmaakt zit Jasmine en Karma er standaard bij. Zelf heb ik geen ervaring met Karma, maar ik heb wel ervaring met het Jest framework dat gemaakt is op Jasmine waarbij de code voor het testen vrijwel identiek is. Bij Jest open je alleen geen browser, terwijl je dat met Karma wel doet.

Jest is (op het moment van het maken van dit afstudeertraject) het populairste JavaScript framework voor het maken van testen (Best javascript testing framework, 2022). Bovenop de tools die Jasmine heeft kan je met Jest Snapshot Testing doen. Snapshot Testing houdt in dat de test een snapshot maakt van de webpagina en het snapshot dan vergelijkt met een vorig snapshot als die test er eerder een heeft aangemaakt. Jest heeft ook parallel testen, dit houdt in dat de testen naast elkaar worden gedraaid in plaats van na elkaar waardoor het draaien van de testen veel sneller is (Jest, 2021). Omdat Jest geen echte browser gebruikt zullen de testen in Jest waarschijnlijk sneller draaien dan met Jasmine en Karma (er zijn hiervoor vele bronnen hier zijn er een paar: (Which one is better for unit test? JEST or Jasmine, 2022) (Why use jest over karma for Angular testing?, 2022)).

Ik besloot toch Jasmine en Karma te gebruiken aangezien ik nog niet weet of een echte browser nodig is bij het testen van een game. Indien er geen conflicten zullen zijn, is het in de toekomst wel te overwegen om Jest te gebruiken.

5.10 Jira – voorgeschreven

Jira was aan mij voorgeschreven om te gebruik tijdens het uitvoeren van mijn opdracht. Jira is een software development tool voor projecten waar de Agile software ontwikkel methode wordt gebruikt (Jira software, 2022). Het helpt software ontwikkel teams om hun Agile ontwikkel methode te organiseren. Het bevat onder andere sprints, backlogs, user stories, sprint planning, tijdbesteding aan user stories, roadmaps, epics en reportages van sprints (Jira features, 2022).

CGI host Jira zelf op hun eigen servers. Dat zorgt ervoor dat CGI meer controle hebben over Jira en CGI heeft er voor gezorgd dat Jira alleen toegankelijk is binnen hun intranet. Er bestaan meerdere alternatieven om een Agile ontwikkel methode in te organiseren die CGI ook zelf zou kunnen hosten (Self-hosted Jira alternatives, 2022).

Waarom CGI voor Jira had gekozen over de andere opties, wist ik niet. Omdat deze methode aan mij was voorgeschreven en ik er geen inspraak op had, had ik geen tijd besteed om hier onderzoek naar te doen. Zelf heb ik geen ervaring met Jira, maar wel met een aantal andere tools om Agile op te organiseren. Voor het eindresultaat van de opdracht maakt de keuze voor Agile organisatie tool niet uit. Dit komt omdat deze keuze alleen maar bepaald waar het is georganiseerd en niet hoe het is georganiseerd.

Persoonlijk zou ik zelf voor een eenvoudigere tool gekozen hebben zoals Trello. Bij deze opdracht zijn veel van de features die Jira bied niet nodig en is het voor de developer (in dit geval mijzelf) eenvoudiger te gebruiken. Het zou kunnen dat CGI veel van deze features wel gebruikt bij hun projecten, maar hier heb ik geen kennis over.

6 Plan van aanpak

Om een gestructureerd plan te maken besloot ik alle werkzaamheden die ik moest verrichten op te delen. Eerst wilde ik een fase om mij te oriënteren op het bedrijf en het project om vervolgens het project op te zetten. Daarna wilde ik de onderdelen die ik zou gaan maken onderverdelen in epics (zie hoofdstuk 6.1 Oriëntatie- en opzet fase voor hoe ik dat doe).

Als derde zal ik een risico beschrijven, waar ik rekening mee moet houden bij het uitvoeren en inplannen van de Oriëntatie- en opzet fase.

Als vierde zal ik de architectuur beschrijven die bij dit project toegepast zal worden.

Dan als laatste zal ik het uitgewerkte plan van aanpak weergeven.

6.1 Oriëntatie- en opzet fase

In de oriëntatie- en opzet fase wilde ik eerst alle afspraken maken, de hardware regelen en benodigde software daarvoor installeren.

Vervolgens wilde ik alle requirements verzamelen, om de wensen van de opdrachtgevers zo veel mogelijk te verfijnen, zodat ik precies weet wat ze willen en daardoor een product kan opleveren dat zo veel mogelijk aan hun wensen voldoet.

Daarna wilde ik alle requirements omzetten naar user stories. Een user story is een losse requirement of een bundel van requirements. De user stories wil ik dan samen bundelen tot epics om een goed beeld te krijgen welke user story bij welk systeem hoort.

De eerste taken die ik wilde verrichten waren het database model maken, onderzoek doen naar een framework voor de API en de microservices en een onderzoek doen naar manieren om games op te slaan. Het database model maken wilde ik eerst doen, omdat het helpt een beter beeld te krijgen, hoe de applicatie er uit zal gaan zien. Tevens kon het helpen eventuele gemiste requirements te vinden (bijvoorbeeld voor de lengte van de gebruikersnaam of dat attributen wel of niet null mogen zijn).

6.2 Epics en sprints

De eerste epic waar ik aan wilde beginnen, is de user management epic. Ik wilde dit eerst doen, omdat je waarschijnlijk users moet hebben om de games te kunnen spelen. Daarna wilde ik het games systeem epic doen.

Als laatste wilde ik aan het activiteiten systeem beginnen, omdat sommige van de activiteiten aan een spel gerelateerd zijn en gerelateerd zijn aan een gebruiker. Ik wilde dus eerst die systemen geïmplementeerd hebben.

Met Scrum is het mogelijk dat de opdrachtgevers een andere volgorde willen voor het uitvoeren user stories of epics. Indien dat het geval zou zijn, zal ik hun gewenste volgorde van epics of user stories volgen.

Deze epics wilde ik implementeren in sprints van drie weken waarvoor er voor elk systeem een sprint is. Ik had er voor gekozen sprints van drie weken te nemen in plaats van twee weken, omdat mijn afstudeertraject anderhalf keer langer is dan een standaard afstudeertraject.

Mijn afstudeertraject was langer, omdat ik vier dagen in de week wilde werken in plaats van vijf en toch aan de minimale hoeveelheid dagen wilde voldoen, meer tijd wilde hebben om de opdracht uit te voeren en meer tijd kon besteden aan dit afstudeerdossier.

6.3 Licenties risico

Een risico waar ik rekening mee hield, is dat het mogelijk was, dat het maanden ging duren voordat ik mijn licenties had. Als dit risico werkelijkheid werd kon ik, om de impact van het risico zo veel mogelijk te mitigeren, de volgorde van de bovenstaande werkzaamheden veranderen. Wat de volgorde dan zal zijn is

moeilijk te zeggen, want dat zal liggen aan welke licentie(s) ik op dat moment nog wel en niet heb ontvangen (bijvoorbeeld als ik de licentie van Jira later krijg zou ik eerst de onderzoeken kunnen doen).

In het geval dat ik pas laat in het afstudeertraject de licentie voor Jira zou krijgen, wilde ik de user stories prioriteren volgens de MoSCoW methode. Dit besloot ik dan te doen, omdat het tijd zal besparen bij het prioriteren door de opdrachtgevers, omdat er dan al een prioritering is. Door hier tijd te besparen konden de sprints dan eerder beginnen, dan als er eerst nog een volledige prioritering zou moeten plaats vinden.

6.4 Microservice architectuur

Omdat ik bij dit project microservices moet gebruiken, zal ik bij het uitvoeren van dit project een microservice architectuur maken. Dit houdt in dat het project zal bestaan uit meerdere losse applicaties die allemaal hun eigen verantwoordelijkheid hebben en samen kunnen werken door berichten naar elkaar te versturen (5 design principles for microservices, 2022)..

Bij deze architectuur zal ik dan één frontend applicatie hebben die de gebruiker te zien krijgt in de browser. De backend van het platform zal bestaan uit één API applicatie en meerdere microservice applicaties.

De API applicatie is de applicatie waar de frontend al zijn aanvragen naar toe zal versturen om iets te krijgen van Azure (bijvoorbeeld de database). Bij alle aanvragen dat de API ontvangt zal het de ontvangen aanvraag doorsturen naar de microservice dat verantwoordelijk is voor het onderdeel waar die aanvraag naar vraagt.

Bij deze opdracht wilde ik voor elk systeem in ieder geval één microservice applicatie hebben, dat verantwoordelijk is over dat systeem. Ik wilde dus in ieder geval één microservice maken dat verantwoordelijk was over gebruikers, één microservice dat verantwoordelijk was over het activiteiten systeem en één microservice dat verantwoordelijk was over de games.

Hoewel het framework voor de frontend applicatie aan mij was voorgeschreven, kon ik wel zelf kiezen welk framework ik wilde gebruiken om de API applicatie en alle microservice applicaties mee te maken. Voor deze keuze wilde ik ook een onderzoek in plannen, om zo het framework te kiezen die het meest geschikt is voor de backend applicaties van dit project.

6.5 Uitwerking plan van aanpak

Hieronder heb ik het bovenstaande plan onderverdeeld en verder gespecificeerd naar een Oriëntatie en opzet fase, Epic User management, Epic Games systeem en Epic Activiteiten systeem.

Oriëntatie- en opzet fase – 2 weken

- Bedrijf verkennen/introduceren
- Werkwijze en afspraken bespreken (wekelijkse meetings of dat soort afspraken)
- Plan van aanpak aanscherpen
    - Door met Rens en Adolfo the overleggen
    - Door de huidige situatie te bekijken
- Opstellen Scrum met epics en user stories
- Definition of Done afspreken
- Definition of Ready afspreken
- Benodigde hardware ontvangen
- Benodigde software installeren
- Werk environment opzetten (onder andere de repository opstellen, Azure, ESLint, overige benodigdheden, etc.)

- Joinersdag bijwonen (de eerste van de maand waar alle nieuwe members komen van die maand en presentaties krijgen)
- Opzet maken voor het afstudeerdossier
- Leren over API's/microservices en het opslaan van bestanden op een server/database
    - Framework kiezen voor de API en de microservices
- Requirements opstellen
- Database ontwerpen

Epic User management: 6-9 weken:

- Users systeem ontwerpen
- User privileges systeem ontwerpen
- User privileges systeem opzetten
- Front-end users opzetten
- Front-end koppelen aan de backend

Epic Games systeem: 6-9 weken:

- Games systeem ontwerpen
- Games database/file system ontwerpen
- Games database/file system opzetten
- Games systeem ontwerpen (hoe van de database/file system wordt afgehaald, hoe het geladen wordt, etc.)
- Games systeem implementeren
- Front-end games opzetten
- Front-end koppelen aan de backend

Epic Activiteiten systeem: 6-9 weken:

- Activiteiten systeem ontwerpen
- Activiteiten systeem implementeren
- Activiteiten systeem koppelen aan front-end

7 Afspraken

In dit hoofdstuk zijn de Definition of Done en de Definition of Ready beschreven. De Definition of Done had ik opgesteld, door een gesprek met mijn begeleider te voeren over de Definition of Done. De Definition of Ready had ik opgesteld, door een gesprek met mijn opdrachtgever (Adolfo Sanchez) te voeren over de Definition of Ready.

In dit hoofdstuk beschrijf ik ook de afspraken die ik met Rens en Adolfo gemaakt had in de oriëntatiefase van het afstudeertraject.

7.1 Definition of Done

Voordat een user story naar de master branch mocht, moesten de volgende stappen verlopen zijn:

1. Er zijn unit testen gemaakt voor de code die is geschreven.
2. De development branch is gemerged in de master branch.
3. Alle unit testen van het project zijn succesvol.
4. De code moet gedocumenteerd zijn met opmerkingen.

Deze criteria waren er, zodat de code van een user story succesvol getest is, gedocumenteerd is en er ook geen errors voorkomen op de master branch met de nieuwe code.

7.2 Definition of Ready

Voordat een user story in een sprint mocht worden opgepakt, moest het voldoen aan de volgende eisen:

1 De user story moet duidelijk zijn (er moet precies staan wat er gedaan moet worden).

2 Er moet staan hoe veel effort het gaat kosten .

3 De user story kan gemaakt worden in 1 sprint.

4 Er zijn geen afhankelijkheden meer of de afhankelijkheden zijn al gemaakt (bijvoorbeeld story A moet eerst gedaan worden voordat er aan story B begonnen kan worden of er moet een bepaalde extensie geïnstalleerd zijn).

Deze criteria waren er, zodat er op een user story duidelijk te zien is, wat er moest worden gedaan en hoeveel tijd het zal kosten. De derde criteria was er, om ervoor te zorgen dat de user stories de regels van Scrum volgden. De vierde criteria was er, om te voorkomen dat er een user story ingepland wordt, waar eerst een andere user story voor gedaan moest worden.

7.3 Vrijheid

Ik mocht zelf bepalen hoe ik het project aanpakte en hoe ik mijn tijd besteedde. Ik rapporteerde dan regelmatig aan mijn begeleider, bij de stand-ups of via berichten in Teams, wat ik aan het doen was of van plan was te doen. Ik mocht ook zelf bepalen wanneer en hoe lang ik aan dit afstudeerdossier werkte. Wat ik ook zelf mocht kiezen is wat voor framework ik koos voor de backend applicaties. Deze afspraken had ik gemaakt tijdens een gesprek met mijn opdrachtgevers.

7.4 Stand-up

Ik had met mijn begeleider afgesproken om twee keer per week een stand-up te doen, waarbij ik vertelde hoe het was gegaan sinds de vorige stand-up en de voortgang van de opdracht. Met de stand-ups hield ik op die manier mijn product owner(s) op de hoogte van de voortgang van het platform. Bij deze stand-ups had ik ook de gelegenheid om code en problemen te bespreken die ik tegen kwam, tijdens het uitvoeren van de opdracht. Op de dag dat een nieuwe sprint begon, deden we ook een code review en gaf ik ook een demo, indien in die sprint ook functionaliteit is toegevoegd.

8 Opzet fase

De opzet fase van mijn afstudeertraject speelde zich af over een periode van zeven weken. In deze fase liepen meerdere processen tegelijk af. De reden dat er meerdere processen tegelijk afspeelden kwam onder andere door de lange duur van aanvragen van de licenties, deadlines van mijn begeleider, beveiliging op de laptop waardoor ik scripts niet kon afspelen, onverwachte requirements, integratie tussen GitLab en Azure en Azure budget.

Tijdens deze fase communiceerde ik elke dag via Teams berichten en/of gesprekken over onder andere problemen, wat ik aan het doen ben, wat ik van plan ben te doen, voortgang met het aanvragen van licenties en feedback. Dit deden we niet alleen om elkaar goed op te hoogte te houden maar ook als tijdelijke oplossing aangezien we nog geen sprints met stand-ups hadden waarbij je elkaar op de hoogte houdt.

In dit hoofdstuk zal ik het proces in chronologische volgorde beschrijven, om zo een zo goed mogelijk beeld te geven van hoe deze fase verlopen is.

Als laatste deed ik nog een terugblik op het plan van aanpak, om te zien of ik nog op schema liep of ik het plan aan moest passen, door alle verkregen kennis tijdens deze fase.

8.1 Week 1 – Eerste requirements en database diagram

Het eerste punt dat ik had aangepakt (zoals genoemd in hoofdstuk 6 Plan van aanpak), was het opstellen van de requirements. Ik had deze requirements opgesteld, door de wensen van de opdrachtgevers te verfijnen, om een zo goed mogelijk beeld te krijgen van alle wensen van de opdrachtgevers.

Nadat ik die requirements had opgesteld, was ik aan het eerste ontwerp van het database diagram begonnen. Dit deed ik, omdat ik bij het maken van dit ontwerp nog eventuele gemiste requirements kon ontdekken die ik dan nog kon bespreken met mijn begeleider.

8.1.1 Users en games requirements

Deze requirements had ik verzameld, door met mijn opdrachtgevers in gesprek te gaan. In dit gesprek vroeg ik naar de wensen voor het platform, waarbij ik dan vragen stelde op basis van informatie uit dat gesprek en mijn bestaande kennis over de wensen voor het platform.

Door dit gesprek had ik de eerst versie van de lijst van requirements opgesteld, met requirements over de gebruikers en de games. Door een drukke periode van mijn opdrachtgevers was het niet mogelijk, om nog in deze week een gesprek te voeren over de requirements van het activiteiten systeem.

De lijst van requirements had ik in die week nog verder uitgewerkt, door vragen te stellen via Microsoft Teams aan mijn begeleider.

De requirements die ik had opgesteld over de gebruikers en de games zijn te zien in Bijlage 2: Requirements users en games.

8.1.2 Eerste versie database diagram

Op Bijlage 5: Eerste versie database diagram is te zien hoe het database diagram er uit zag. Ik zal het proces beschrijven van hoe ik het diagram gemaakt had.

Eerst had ik alle tabellen aangemaakt die genoemd zijn in de requirements. Dit zijn: games, users, privileges, organisations, rollen en settings. Vervolgens had ik de attributen voor deze tabellen uit de requirements ingevuld in de tabellen. Daarna had ik de meer-naar-meer relatie tabellen aangemaakt en ingevuld, vanuit de requirements. Vervolgens had ik de overige relaties toegevoegd aan het model aan de hand van de requirements, om te zien wat wel of niet null mag zijn.

Tijdens dit proces koppelde ik meerdere malen terug met mijn begeleider voor feedback.

Als laatste had ik naar de constraints van elke tabel gekeken, om zeker te zijn dat er geen gedupliceerde tupels in kunnen komen die er niet in mogen zitten.

Tijdens het afstudeertraject zijn hier nog wel een paar kleine wijzigingen aan gedaan (zoals naam van een attribuut wijzigen, constraints toevoegen), maar dit deel van de database is vrijwel hetzelfde gebleven.

8.2 Week 2 – Onderzoeken

Het had mijn voorkeur om eerst alle requirements te verzamelen, voordat ik aan de onderzoeken begon. Dat ging helaas niet lukken, omdat mijn begeleider die week geen tijd had om in gesprek te gaan over de requirements. In deze week had ik nog niet één van mijn licenties ontvangen, waardoor de werk omgeving opzetten nog niet mogelijk was.

In deze week besloot ik daarom om de onderzoeken uit te voeren naar een framework voor de backend applicaties en het onderzoek naar methodieken om de games op te slaan. Dit besloot ik, omdat ik voor deze onderzoeken geen van de licenties en de overige requirements nodig had.

8.2.1 Onderzoek naar keuze framework voor de backend applicaties

Het doel van dit onderzoek, was om kennis te verzamelen over verschillende frameworks voor backend applicaties (de API en de microservices), zodat ik de meest geschikte keuze voor framework kon maken, om de backend applicaties van het project in te ontwikkelen.

Ik besloot dit onderzoek te doen omdat ik tot de realisatie kwam dat er veel frameworks beschikbaar zijn voor het maken van een API applicatie en microservice applicaties. Door een onderzoek te doen kon ik dan beter specificeren wat ik zocht in zo een framework om zo het framework te kiezen dat het meest overeenkomt met wat er gevraagd wordt van de API applicatie en de alle microservice applicaties.

8.2.1.1 Waarom een framework?

Simpel gezegd is een framework een collectie van al bestaande structuur, functies, code en procedures (API framework, 2021). Een framework zou je dus kunnen helpen bij het ontwikkelen van software.

Een framework helpt je alleen met ontwikkelen als je het juiste framework gekozen hebt. Wanneer je een framework kiest dat dat niet de optimale keuze is voor je doeleinden kan dit veel extra werk bezorgen en wellicht verlies van werk door een andere framework te kiezen. Het is dus zeer belangrijk dat hier de goede keuze voor wordt gemaakt.

8.2.1.2 Selectie van frameworks

Er zijn heel veel frameworks voor het maken van een API applicatie en microservice applicaties in JavaScript. Bij het selecteren van kandidaten heb ik gefocust op een lijst met de 25 meest gebruikte API frameworks (Lijst met 25 API frameworks, 2021). Met het onderzoeken van frameworks heb ik gefocust op frameworks die voldoen aan de volgende punten:

- Hoge aantal GitHub stars (meer dan tienduizend). Als een framework veel GitHub stars heeft betekend dit dat er veel mensen zijn die interesse hebben in het framework en het waarderen.
- Support. Het framework moet wel nog support hebben, niet dat de laatste update een jaar geleden is uitgevoerd bijvoorbeeld.
- Documentatie. Er moet goede documentatie beschikbaar zijn, anders wordt het ontwikkelen ermee lastig.
- Microservices. Het moet geschrikt zijn voor het ontwikkelen van losse microservice applicaties want ik wil het framework gebruiken om microservice applicaties mee te maken.
- Open source. Het framework moet open source zijn, want dan kan CGI het gebruiken zonder een licentie nodig te hebben of er kosten aan verbonden zijn.

- TypeScript. Het framework moet TypeScript ondersteunen. Voor uitleg voor de keuze voor TypeScript zie hoofdstuk 5.5 TypeScript.

8.2.1.3 De kandidaten

De kandidaten die over bleven waren Express en NestJS. SocketIO had de lijst met kandidaten bijna gehaald. SocketIO viel af, omdat hij een verbinding maakt tussen server en cliënt in plaats van dat hij de functionaliteit van een API heeft.

8.2.1.4 Loopback

Het Loopback framework viel al snel af vanwege verschillende problemen. Ten eerste heeft het minder dan 4000 GitHub stars. Er zijn dus niet veel mensen die vragen kunnen beantwoorden over problemen. Ten tweede heeft het hele slechte documentatie (naar mijn mening) voor het maken van microservices (vrijwel niet bestaand) (Loopback microservice documentatie, 2021). Voor een losse API is er wel genoeg documentatie, maar het is kwalitatief niet zo goed. Dit maakt loopback zeer ongeschikt voor het maken van microservices. Express en NestJS zijn in al deze categorieën beter en eenvoudiger om te leren waardoor Loopback afviel (loopback pros and cons, 2021).

8.2.1.5 Express

Express is een open source framework met een lage learning curve dat TypeScript ondersteunt. Express heeft meer dan 50000 GitHub stars en wordt nog up-to-date gehouden. Het is flexibel met goede documentatie (Express, 2021). Ondanks dat de hoeveelheid documentatie minder is dan met NestJS is het meer dan voldoende. Express wordt gebruikt bij meer dan 10 miljoen projecten (Express github, 2021). Een voordeel van Express is ook dat het al gebruikt wordt bij de huidige games van de Game Factory (het wordt gebruikt via Angular). Een ander voordeel is dat je met Express zelf de structuur kan kiezen. Het is mogelijk om met Express microservices te maken.

8.2.1.6 NestJS

NestJS is gebouwd op Express en heeft een lage learning curve (NodeJS + express vs NestJS, 2021). Het is gebaseerd op de architectuur van Angular (Getting started with NestJS, 2021) en heeft TypeScript als programmeer taal. Dit zal helpen bij het leren van NestJS en Angular omdat ik dan niet twee verschillende architecturen hoef te leren. NestJS houdt zich ook aan de principes van object georiënteerd programmeren(OOP), wat zeer handig is aangezien dit project volgens de OOP principes ontwikkeld zal worden.

NestJS applicaties volgen een architectuur met controllers en providers. De binnenkomende aanvragen worden door de controllers ontvangen, die de aanvragen dan laten afhandelen door providers.

NestJS heeft meer dan 40000 GitHub stars, wordt nog up-to-date gehouden en wordt gebruikt bij meer dan 92000 projecten (NestJS github, 2021). NestJS heeft veel documentatie met tutorials (NestJS, 2021). Er is onder andere documentatie voor het opzetten van microservices die handig kunnen zijn bij het maken van microservices. NestJS heeft ook tools die je kunnen helpen bij het maken van testen voor NestJS onderdelen. Het framework dat NestJS gebruikt voor zijn test is het Jest framework. Het testen met Jest in NestJS zou op vrijwel dezelfde manier verlopen als de testen met Jasmine in Angular .

8.2.1.7 Resultaat

Ik had uiteindelijk voor het NestJS framework gekozen. Dit was omdat de software de OOP principepen moest volgen en NestJS daar goed bij helpt, want NestJS is al OOP opgezet. Het platform moest uiteindelijk gaan werken met microservices en die waren ook zeer eenvoudig te implementeren met NestJS, doordat NestJS tools voor microservices heeft ingebouwd.

Voor het maken van unit testen was NestJS ook handig, omdat het niet alleen tools heeft om NestJS onderdelen te testen, maar ook een test framework gebruikt dat (qua code) vrijwel identiek is aan Jasmine. Bij microservices maakte het niet uit in welke taal of met welk framework het gemaakt is, om er mee te

kunnen communiceren. Dus ondanks dat de apps van de Game Factory gebruik maakten van het Express framework, zouden ze nog steeds met de microservices kunnen communiceren.

De resultaten van dit onderzoek had ik ook teruggekoppeld aan mijn begeleider. Nadat ik de resultaten van dit onderzoek aan hem had voorgelegd, was hij het er mee eens om NestJS te gebruiken als framework voor de API en de microservice applicaties.

8.2.1.8 Demo

Nadat ik tot dit resultaat was gekomen, werd aan mij gevraagd door mijn begeleider om een demo te maken om te laten zien hoe het werkt. Deze demo ging ik uitvoeren met Docker, omdat de backend van het platform ook uiteindelijk in containers zal gaan draaien. Dit zal een goed beeld geven hoe de backend van het platform zou werken, als de backend applicaties het NestJS zal gebruiken.

Het ontwikkelen van een demo was eenvoudig en liep volledig naar wens. Het deed precies wat ik van de API en de microservices verwachtte. Door deze demo gemaakt te hebben had ik het vertrouwen gekregen dat dit framework inderdaad precies is wat ik wil hebben voor de backend van het platform.

8.2.2 Onderzoek naar het opslaan van games

Momenteel had elke game een eigen website waar de game op app niveau wordt opgeslagen. Maar aangezien het platform later meerdere games, een database, een API en meerdere microservices heeft, is het dan nog wel zo handig om de games op te laten slaan op app niveau?

Ik mocht zelf kiezen hoe ik de games op Azure opsloeg. Ik besloot hier een onderzoek van te maken, omdat ik door middel van een onderzoek beter de voor- en nadelen tegen elkaar kan afwegen, om zo een betere keuze te kunnen maken voor het opslaan van de games. De uitkomst van dit onderzoek had ik teruggekoppeld naar mijn begeleider.

Bij dit onderzoek had ik vier oplossingen gevonden dat allemaal mogelijk zijn op Azure. Twee van deze oplossingen zijn oplossingen die op Azure werken, maar ook zonder Azure kunnen werken. De andere twee oplossingen zijn specifiek voor Azure, omdat die gebruik maken van verschillende services van Azure.

8.2.2.1 Blob opslaan in database

De eerste oplossing die ik vond is om de games op te slaan in de database als blob formaat (Blob data type, 2022). Om met deze oplossing een game op converteer je een game file naar blob formaat en dat sla je dan op in een database met optioneel de extensie van de file (je hebt niet altijd de extensie nodig). Je kan het dan weer converteren naar de game file wanneer je de game wilt gebruiken.

Het nadeel hieraan, is dat je mogelijk dataverlies hebt door het converteren van blob naar het formaat van het bestand en terug. Een ander nadeel, is dat het veel druk zet op de database doordat het dan de games naar de gebruiker moet sturen wanneer een gebruiker wil spelen (Should binary files be stored in the database?, 2021). Een ander nadeel, is dat wanneer een game geüpdatet wordt, diegene dat de game update dan de game naar de database moet sturen in plaats van stopt op een opslagplek voor bestanden dat extra stappen en tijd kost. Het voordeel hiervan is wel dat de games en de data (zoals users) op één plek staat.

8.2.2.2 File system

Als tweede oplossing kon ik gebruik maken van een file system. Met een file system sla je dan het pad op naar de locatie waar de game is opgeslagen in de database. De app kan een game dan ophalen door eerst het pad naar de game op te halen van de database om dan dat pad te gebruiken om de game op te halen.

Een nadeel hiervan is dat de paden naar de games worden verbroken, als de games op een andere plek worden opgeslagen of de naam van de game te wijzigen (Should binary files be stored in the database?, 2021).

Een voordeel hiervan, was dat de database aanvraag dan sneller was, om de games op deze manier op te halen, dan wanneer je ze als blob opslaat in een database (Saving Files In A Database Or In A File System,

2021). Dit komt, omdat je hiermee geen query hoeft te doen om een game bestand op te halen, maar in plaats daarvan een klein pad ophaal naar het bestand. Dit was ook een nettere manier van opslaan, want je hoeft niks te converteren door de games als bestand op te slaan, terwijl je bij het opslaan in de database als blob, het blob formaat wel moet converteren naar het bestand dat je wilt.

8.2.2.3 Azure Storage

De eerste Azure specifieke oplossing die ik vond is om de games op te slaan op Azure storage.

Azure storage is een opslagplaats voor data in de Cloud. Azure storage heeft dan verschillende opties voor het opslaan van data (Azure storage accounts, 2021).

- Azure Blob Storage is voor het opslaan van grote hoeveelheden ongestructureerde data zoals tekst of binair. Het is goed voor het opslaan van afbeeldingen voor een browser, streaming video en audio en het opslaan van bestanden met gedistribueerde toegang.
- Azure File Storage is voor het delen van bestanden over een netwerk. Bijvoorbeeld dat meerdere mensen aan hetzelfde Word document tegelijk kunnen werken.
- Azure Disk Storage is voor de opslag van disks voor gebruik voor virtual machines.

Voor het opslaan van de games is dan de Azure Blob Storage het beste want het is goed voor het opslaan van bestanden met gedistribueerde toegang.

De games laten weergeven op de app werkt met Azure Storage hetzelfde, als bij hoofdstuk 8.2.2.2 File system, waarbij je de games dan kan ophalen met een URL en je het pad er naar toe op kan slaan in de SQL database.

8.2.2.4 Azure Databricks File System

De laatste Azure specifieke oplossing die ik vond is de Azure Databricks File System. Azure Databricks bevat een Database File System dat samen kan werken met Azure Storage (Technical overview of azure databricks, 2021).

Een Database File System is een interface dat bovenop de opslag plek van bestanden die opgeslagen zijn in een database. De Database File System zorgt er dan voor dat de bestanden makkelijker toegankelijk zijn dan met alleen Azure Storage. De bestanden kan je dan vinden door middel van een folder structuur in plaats van een URL (bijvoorbeeld “/mnt/storage1” in plaats van “https://<storage-account>.blob.core.windows.net) (FileStore, 2021).

Het voordeel hiervan is dat het makkelijker is om bij de bestanden te komen dan wanneer je een URL moet gebruiken om bij de games te komen op de Azure Storage.

8.2.2.5 Conclusie

Ik heb gekozen voor een combinatie van File system met Azure Blob Storage en Azure Databricks. Ik heb hiervoor gekozen om meerdere redenen. Om mee te beginnen geeft een file system minder druk op de database waardoor de database minder belast wordt en beter presteert. Door een file system te gebruiken hoef je ook de games niet meer te converteren van en naar blobs. Hierdoor voorkom je dan potentieel dataverlies omdat je niet meer hoeft te converteren.

Azure Databricks heb ik gekozen omdat het ervoor zorgt dat het makkelijker is voor de microservices om de games op te halen en te uploaden.

8.3 Week 3 – Activiteiten systeem

In deze week kon ik het gesprek voeren met mijn begeleider om de wensen van het activiteiten systeem te verfijnen door elke wens van mijn begeleider te specificeren om zo een zo goed mogelijk beeld te krijgen van alle wensen over het activiteiten systeem. Dit gesprek had ik alleen gevoerd met mijn begeleider, omdat hij ging over alle technische aspecten van het project.

Dit langdurige gesprek veroorzaakte veel veranderingen voor zowel het plan van aanpak en het project zelf.

Eerst zal ik beschrijven hoe dat gesprek was verlopen. Vervolgens zal ik de resultaten van dat gesprek beschrijven. Dan zal ik de werking van het activiteiten systeem beschrijven en als laatste zal ik beschrijven hoe ik het activiteiten systeem in het database diagram had ontworpen.

8.3.1 Het gesprek over de requirements

Tijdens langdurige gesprek stelde ik vragen op basis van mijn kennis over het activiteiten systeem. Ik stelde ook vragen om te bevestigen of ik zijn wensen goed begrepen had (bijvoorbeeld “is x wat je wilt?”). Wanneer ik het niet goed begreep kon hij mijn begrip van zijn wensen corrigeren. Tijdens dit gesprek had ik meerdere diagrammen en/of code voorbeelden laten zien om mijn begrip beter uit te leggen.

Tijdens dit gesprek kwam ik er achter dat ik het niet helemaal begrepen had wat er precies bedoeld werd met het activiteiten systeem, maar bij het einde van het gesprek begreep ik precies wat zijn wensen hiervoor waren.

Het activiteiten systeem wat ik in gedachte had was een los systeem dat taken voor een user kan ophalen en aan de gebruiker presenteert wat hij moet doen. Als de gebruiker dat gedaan heeft wordt in de database te gezet dat die gebruiker die activiteit uitgevoerd heeft. Bijvoorbeeld dat de gebruiker een tutorial van een bepaald spel moet doen.

Mijn begrip van het activiteiten systeem was alleen maar een deel van wat er van het activiteiten systeem gevraagd werd.

Het activiteiten systeem wat gewenst werd was meer een fundering waar de hele frontend op gebouwd zal worden. Het idee was ook dat elke gebruiker een andere combinatie van taken kan krijgen dat de frontend uitvoert wanneer een gebruiker een bepaalde actie uitvoert. Bijvoorbeeld dat gebruiker A na het authentiseren bij het inloggen naar zijn dashboard gaat en dat gebruiker B na het authentiseren bij het inloggen naar een bepaalde game toe gaat.

8.3.2 De resultaten van het gesprek en de impact daarvan

De impact op het plan van aanpak en het project, van de verandering van een los onderdeel op een bestaand systeem, naar de fundering van het hele systeem, is groot.

Het plan voor implementatie van de epics werd hierdoor, dat eerst het activiteiten systeem moest worden gemaakt, dan het users systeem en als laatste het games systeem. Dit komt omdat er eerst acties moeten zijn, om gebruikers uit te laten voeren en ik daarna pas het games systeem kan implementeren, omdat er anders geen gebruikers zijn, om de actie dat een game start uit te voeren.

Deze nieuwe wensen over het activiteiten systeem, hadden dus ook een grote impact op het implementeren van alle andere systemen. Het games systeem zal niet heel veel veranderen door dit activiteiten systeem, maar het zal wel veel impact hebben op users management systeem.

Het games systeem moest zoals eerst nog steeds een game kunnen laden en een lijst van games tonen. Wat hierbij wel verandert, is hoe een gebruiker op die game pagina en het dashboard komt. Dit zal dan liggen aan de activiteiten en taken dat de gebruiker heeft.

Het users management systeem zal zeer afhankelijk zijn van het activiteiten systeem. Bijvoorbeeld voor wat het systeem moet doen als een gebruiker succesvol ingelogd is.

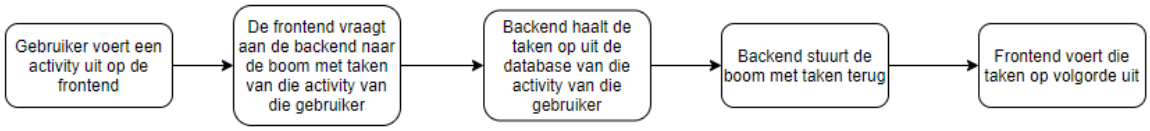
Waar dit nieuwe activiteiten systeem ook impact op had, is de hoeveelheid werk dat er gedaan moet worden. Het activiteiten systeem als een los onderdeel ontwikkelen, is veel minder werk, dan een hele fundering ontwikkelen waar de frontend applicatie op werkt.

Tijdens het gesprek kreeg ik ook een project dat mijn begeleider had gemaakt, waar het idee voor mijn opdracht vandaan kwam. In dit project zat een uitgewerkt idee voor een activiteiten systeem, waarop je twee verschillende games kon spelen.

De requirements over het activiteiten systeem die ik had opgesteld zijn te zien in Bijlage 3: Activiteiten systeem requirements.

8.3.3 Werking nieuw activiteiten systeem

Hoe het activiteiten systeem ongeveer zal gaan werken, zal ik aan de hand van het onderstaande diagram uitleggen.



Figuur 2: Diagram workflow activiteiten requirements gesprek

Het idee met het nieuwe activiteiten systeem dat ze wilden (zoals staat bij hoofdstuk 8.3.1 Het gesprek over de requirementsHet gesprek) is dat alles wat een gebruiker doet op de frontend (en sommige acties in een game) een activiteit start. Zo een activiteit bevat dan op de database taken bij die activiteit van die gebruiker. Die taken zeggen dan wat de frontend (of de game) moet doen en als het meerdere taken zijn ook op welke volgorde.

Een activiteit kan meerdere taken bevatten en het kan dat een activiteit een taak heeft, dat moet wachten totdat een andere taak van die activiteit is gedaan.

De taken van een activiteit van een gebruiker vormen een boom van taken, waarbij de taak waar niet op gewacht hoeft te worden als eerste wordt gedaan, dan de taak dat op die eerste taak wacht en zo voort. Deze boom van taken zal gemaakt worden op de activiteiten microservice applicatie van de resultaten dat het terugkrijgt van de aanvraag aan de database.

Deze boom van taken zal de activiteiten microservice dan terug sturen naar de API die het dan terug naar de frontend stuurt. De frontend kan dan de taken boom op de goede volgorde uitvoeren door middel van klassen waarbij elke klasse een taak kan uitvoeren.

8.3.4 Uiteindelijke database diagram

Bij het ontwerpen van het volledige database diagram maakte ik gebruik van de vele diagrammen en code voorbeelden die gemaakt zijn (deels door mij deels door mijn begeleider) tijdens het gesprek met mijn begeleider (zie hoofdstuk 8.3.1 Het gesprek over de requirements). Bij het ontwerpen van dit database diagram ontwierp ik verder op het diagram dat ik eerder gemaakt had (zie hoofdstuk 8.1.2 Eerste versie database diagram).

Eerst ging ik alle benodigde tabellen aanmaken. Vervolgens ging ik de attributen voor deze tabellen invullen in de tabellen. Toen voegde ik de overige relaties toe aan het model om te zien wat wel of niet null mag zijn.

Bij het ontwerpen koppelde ik meerdere keren terug met mijn begeleider voor feedback en vragen.

Nadat het diagram goedgekeurd was had ik nog wat wijzigingen toegepast om de database meer integer te maken (waaronder extra constraints met checks voor unieke waardes, on delete checks en on update checks).

Het uiteindelijke database diagram is te zien op Bijlage 6: Database diagram met activiteiten systeem.

8.4 Week 4 en eerste twee dagen week 5 – Azure

Bij het einde van week 3 van de opzet krijg ik eindelijk mijn eerste licentie en die was voor Azure. Deze licentie kwam net op tijd, want zonder een licentie kon ik alleen nog maar de werk environment opzetten en dat zal geen week nodig hebben.

Eerst wilde ik alle nodige services voor het platform aan te maken. Vervolgens kon ik dan de services configureren, om alle verbindingen tussen de applicaties, van de API naar de frontend en om bij de frontend applicatie te kunnen komen in de browser. Dan zal ik beveiliging toepassen op de Azure services, aangezien de data in de database gevoelige informatie bevat.

8.4.1 Services aanmaken en configureren

De meeste services die ik aan moest maken op Azure waren al bekend. Deze kennis kwam van zelf informatie opzoeken op internet, mijn begeleider en uit de onderzoeken die ik had uitgevoerd (voor de onderzoeken zie hoofdstukken 8.2.1 en 8.2.2). Deze services zijn:

1. Een resource group om alle services in aan te maken.
2. Een storage account om de games in op te slaan.
3. Een SQL database.
4. Een virtual network om de containers met elkaar te kunnen laten communiceren.
5. Een container registry om de image van elke applicatie (de frontend, API en elke microservice) in op te slaan.
6. (wanneer de images op de container registry staan) Voor elke image een container instantie, zodat de containers kunnen draaien.

Nadat ik al deze services had aangemaakt liep ik bij het maken van container instanties tegen een probleem aan waar een klein onderzoek voor nodig was om de juiste keuze te maken.

Het probleem was dat een container instantie in Azure maar één IP adres heeft (Container instances virtual network concepts, 2021). Dit IP adres is voor container instanties in een virtual network private. Dit betekent dat je binnen dat virtual netwerk wel met andere containers kan communiceren via dat IP adres maar van buitenaf niet bij die containers kan komen.

Ik moest dus een oplossing vinden om er voor te zorgen dat één applicatie (wat de frontend wordt) wel vanaf buiten toegankelijk is terwijl de rest van de applicaties (de API en alle microservices) alleen via de frontend te bereiken is.

8.4.2 Private IP adres probleem

Er waren drie oplossingen die ik had gevonden waarbij je de frontend kan bereiken vanuit buitenaf maar niet de backend kan bereiken.

De eerste oplossing was om de frontend als app service te deployen in plaats van een container instantie. Met een app service kan je een public IP adres als voordeur gebruiker dat naar de webpagina gaat. Dus wanneer je vanaf de browser naar die public IP adres gaat kom je op het website terecht. Een app service kan integreren met een virtual network om de website toegang te geven tot containers in dat virtual network.

De tweede oplossing was om een firewall en een public IP adres aan te maken die aan elkaar gekoppeld zijn (Container instances egress ip adresss, 2021). De firewall kan dan het verkeer van het public IP adres naar de container instanties sturen en visa versa. Dit kan wanneer de firewall verbonden is met het virtual network.

De derde oplossing is om een Application gateway te gebruiken. Hier heeft de Application gateway een Public IP adres als listener, een backend pool als doel om naar toe te sturen en een rule die het verkeer van de public IP adres doorstuurt naar de frontend. Het public IP adres zit buiten het virtual network, zodat je er

van buiten bijkomt. De backend pool bevat het IP adres waar je het verkeer heen wilt sturen. De Application gateway zelf zit in het public subnet van het virtual network. Vanuit de public subnet kan je communiceren met de containers in het private subnet van het virtual network (Azure container example, 2021).

Ik had de voorkeur om voor de eerste oplossing te kiezen, omdat het geen extra services vereist. Bij de tweede oplossing heb je namelijk een firewall extra nodig en bij de derde oplossing heb je een Application Gateway en een subnet extra nodig. Bij de tweede en derde mogelijk moet je ook meer configureren om het goed op te zetten dan bij de eerste mogelijkheid.

Helaas waren de firewall service en de app service te duur om binnen het budget te vallen. Ik heb dus voor de derde mogelijkheid moeten kiezen dat de Application gateway is. Het liefst wilde ik de eerste oplossing, aangezien die oplossing netter zou zijn in de architectuur en het de minste configuratie nodig had.

8.4.3 Demo

Om te zien of ik alle services goed had opgezet, had ik de API en de users microservice aangemaakt, op de container registry gestopt en een container instantie van gemaakt. De code van deze applicaties stonden alleen nog lokaal opgeslagen, aangezien ik de GitLab licenties nog niet had. Om de verbindingen tussen en naar de containers te testen gebruikte de API als gateway container en de users microservice waar de gateway naar toe gaat.

Ik kon bij deze demo een aanvraag sturen naar de API waarbij de microservice de aanvraag succesvol ontving en een reactie terug stuurde. Bij deze demo was het ook niet mogelijk om van buitenaf bij de microservice te komen dus deze demo was een volledig succes.

8.4.4 Azure security

Dit besloot ik, omdat het belangrijk is dat de Cloud omgeving goed beveiligd is. Je wilt namelijk onder andere voorkomen dat er onbevoegd bijvoorbeeld gebruikers worden aangemaakt, privileges worden toegekend, games worden verwijdert worden of activiteiten voor een gebruiker toegevoegd worden.

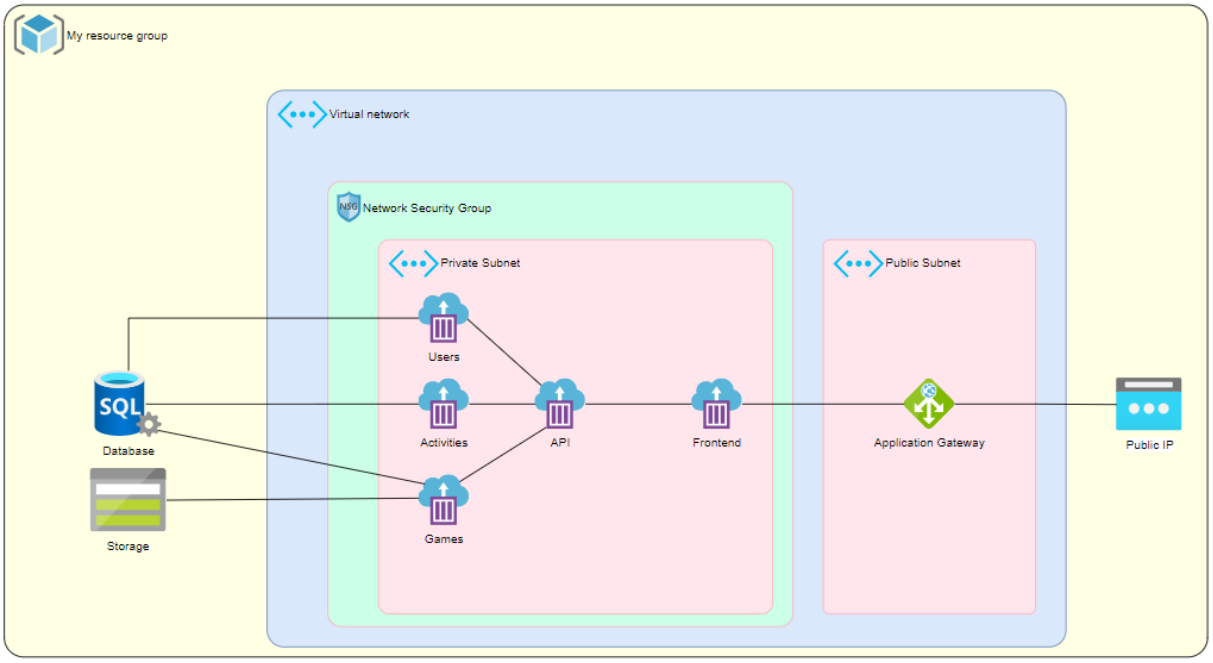
Voor beveiliging ging ik voornamelijk kijken naar drie aspecten namelijk de storage account, database en virtual network. Ik ging voornamelijk naar die drie kijken want de database is de plaats waar alle data is opgeslagen, de storage account is de plaats waar alle games zijn opgeslagen en het virtual network waar de microservices aanvragen kunnen doen naar de storage en/of database.

Om de database te beveiligen had ik publieke toegang vanuit buitenaf uitgezet waarbij je alleen bij de database kon komen vanuit het private subnet van het virtual network (waar de containers in draaien. Dit deed ik omdat alleen de containers toegang nodig hebben tot de database.

Bij de storage had ik het op zo'n manier ingesteld waarbij je alleen vanuit het private subnet van het virtual network toegang had tot de storage. Om ervoor te zorgen dat de developers die de games maken nog steeds bij de storage kunnen komen kan je individuele IP adressen aangeven bij de storage account die dan bij de storage account kunnen komen.

Om het virtual network beter te beveiligen had ik de Azure service Network Security Group aangemaakt. Een Network Security Group is een extra laag beveiliging die alleen verkeer van bepaalde sources en/of poorten door laat gaan naar een subnet van een virtual network. Dit heb ik dan zo ingesteld dat je alleen vanuit het public subnet van het virtual network (waar de gateway in zit) toegang hebt tot het private subnet. Dit is om er zeker van te zijn dat alleen de gateway bij de container instanties in het private virtual network kan komen.

In Figuur 3: Architectuur Azure services is een overzichtelijk diagram te zien, waar alle applicaties zich bevinden in de opgezette microservice architectuur op Azure. (Azure Databricks was duur dus die wilde ik pas toevoegen wanneer dat nodig was).



Figuur 3: Architectuur Azure services

8.5 Week 6 en laatste twee dagen week 5 – GitLab

Op de derde werkdag van week 5 (donderdag) kreeg ik eindelijk de licenties voor GitLab en Jira. Wat ik als volgende stap van plan was, indien ik geen nieuwe licenties kreeg, was om naast de API en de users microservice applicaties ook de frontend applicatie en activiteiten microservice applicatie op Azure te zetten.

Ondanks dat ik nu de licentie van Jira had werkte de website van Jira op dat moment zo slecht dat het niet mogelijk was om de user stories aan te maken op Jira. De reden dat Jira op dat moment zo slecht werkten ben ik nooit achter gekomen. Het enige dat ik wist was dat het probleem vanuit CGI afkwam. Wat ik wel kon doen is met GitLab aan de slag gaan om daar een pipeline in op te zetten en de repositories voor alle applicaties aan te maken.

Eerst wilde ik alle applicaties aanmaken, in hun aangewezen GitLab repository stoppen en de applicaties opzetten op Azure. Vervolgens kon ik dan de pipeline opzetten voor elke repository, zodat ik een aantal processen kon automatiseren.

8.5.1 Platform opzetten

Mijn begeleider had in GitLab de repositories aangemaakt voor alle applicaties die wilde maken (zie 6.4 Microservice architectuur).

Eerst had ik de API en users microservice die ik al aangemaakt had op de repository te zetten (zie hoofdstuk 8.4.3 Demo). Daarna ging ik de frontend applicatie aanmaken en de activiteiten microservice applicatie en die ook in hun repositories stoppen.

Ik had vervolgens de frontend en activiteiten microservice applicaties op de Azure container registry gestopt en er container instanties van gemaakt (net als de API en activiteiten microservice applicaties, zie hoofdstuk 8.4.3 Demo). Ik gaf dan alle applicaties de juiste verbindingen, waarbij de frontend applicatie van buitenaf bereikbaar was (zie hoofdstuk 8.4.2 Private IP adres probleem voor de werking van de gateway).

8.5.2 Pipeline opzetten

Er waren meerdere taken die ik de pipeline wilde laten uitvoeren, wanneer je code pushed naar de master branch op een repository. Als eerste wilde ik dat het eerst alle code test en de code niet naar de master

branch toestuurt als die testen falen. Ten tweede wilde ik dat wanneer er code naar de master branche gepusht is dat er een image aangemaakt wordt en dat het de image op Azure van die applicatie update met een nieuwe image met de nieuwe code. Als laatste wilde ik dat de container instanties herstart worden, nadat die images geüpdatet zijn. En bij deze laatste stap kwam ik een probleem tegen.

Om bij Azure in te loggen met mijn CGI account werd er two-factor authenticatie toegepast. Bij two-factor authenticatie genereer je een code die je dan bij het inloggen in moet vullen. Maar het was niet mogelijk dit te automatiseren, omdat die code op mijn mobiel gegenereerd werd. Het was dus niet mogelijk een pipeline aan te maken op GitLab die aan alle wensen voldoet.

De reden dat ik dit probleem niet tegen kwam bij het updaten van de image in de container registry, was omdat ik daarbij apart in kon loggen zonder two-factor authenticatie.

Een tweede poging om een pipeline te maken was met Azure Tasks. Met Azure tasks kan je acties laten uitvoeren op het moment dat er naar een branch op een repository nieuwe code gepushte is. Toen ik deze mogelijkheid probeerde liep ik tegen een probleem aan. Azure Tasks ondersteunt alleen repositories van GitHub, BitBucket en Azure DevOps, maar ondersteunt dus niet repositories van GitLab (Container registry faq, 2021). Deze mogelijkheid voor een pipeline viel dus ook af.

De enige optie die ik kon bedenken om enigszins de functionaliteit van een pipeline te hebben, was om command prompt scripts aan te maken. Ik kan een script maken voor elke applicatie, dat een image maakt, de image pushed naar de container registry en dan zijn container herstart. Het nadeel van deze manier was dat die scripts alleen werkten op mijn eigen werklaptop tenzij diegene de locatie van die scripts toevoegt aan de Path op een Windows computer (op andere systemen kan dit ook maar dan op een iets andere manier).

Hoe het hele proces dan verliep, was om eerst de testen te runnen om te zien of alles goed is om. Dan de code te pushen naar de repository, als de testen goed zijn. Als laatste kon ik dan het script gebruiken, om de image van die applicatie op Azure up te daten en zijn container te herstarten.

Maar, omdat de optie hierboven de enige optie was die ik en mijn begeleider konden vinden voor automatiseren, hadden we toch voor die optie gekozen.

Dit hield wel in dat het alleen op mijn computer deels geautomatiseerd was, tenzij toekomstige developers van dit project die scripts verkrijgen en hun systemen zo instellen dat de command prompt de scripts kan vinden.

Deze scripts gaf ik ook door aan mijn opdrachtgevers, zodat zij later die scripts door kunnen geven en/of gebruiken.

Daarnaast was het op dat moment het geval dat ik de enige ben die aan die code werkt dus het was op dat moment nog geen probleem.

Een nadeel hiervan is dat het niet echt een pipeline is maar dat ik zelf handmatig nog steeds de testen moet doorlopen, om daarna het script te starten. Bij een pipeline wilde ik eigenlijk dat alles automatisch word gedaan. Helaas was dit dus niet gelukt, doordat Azure en GitLab niet goed met elkaar samenwerken.

8.6 Week 7 – Jira en start activiteiten systeem

Het enige wat nog opgezet moest worden was Jira. De website van Jira werkte begin deze week nog steeds te slecht om user stories in aan te maken (zie hoofdstuk 8.5 Week 6 en laatste twee dagen week 5 – GitLab voor uitleg over het slecht werken van Jira). Ik besloot daarom alvast aan het activiteiten systeem te beginnen, omdat ik wist dat het activiteiten systeem de hoogste prioriteit zal krijgen omdat daar de hele applicatie op gebouwd zal worden.

Nadat ik twee dagen bezig was met het activiteiten systeem, werkte Jira eindelijk op de derde werkdag van deze week. Deze twee werkdagen heb ik inbegrepen in hoofdstuk 9 Sprint 1 – Activiteiten systeem en inloggen om een overzichtelijke structuur aan te houden in dit dossier.

Ik had al met MoSCoW de prioritering gedaan (zie hoofdstuk 6.3 Licenties risico), waardoor de opdrachtgever maar een paar user stories van prioriteit hoefde te veranderen, om de prioritering naar wens te hebben. De volledige originele geprioriteerde backlog is te zien in Bijlage 9: Geprioriteerde backlog.

Dit was een goede beslissing geweest om al de user stories te prioriteren. Jira werkte later op die dag niet meer, maar door de eerdere prioritering met MoSCoW kon ik wel de belangrijkste user stories erin zetten (ik kon er maar ongeveer 70% van de user stories in zetten). De user stories die ik er niet meer in kon zetten had ik later in het traject nog in Jira toegevoegd en door mijn begeleider geprioriteerd.

In Jira rekenden wij dan dat één effort point in Jira gelijk is aan één uur werk. De hoeveelheid effort die een user story kost bepaalde ik door rekening te houden met welke code al gemaakt is en wat er moet gebeuren om die user story te voltooien. Hierbij gebruikte ik mijn eigen ervaringen als basis voor het inschatten hoeveel tijd ik met een user story bezig zal zijn.

We besloten de sprint pas op de volgende maandag te beginnen want op die dag heb ik samen met de opdrachtgevers de stand-up en begint de week. Samen met mijn begeleider maak ik in de eerste stand-up van elke nieuwe sprint de sprintplanning met wat ik ga doen in die sprint.

8.7 Planning terugblik

Deze fase duurde veel langer dan ik had verwacht. Zowel de onderzoeken uitvoeren, het opzetten van het project (onder andere Jira en GitLab) en het project configureren in Azure kostten meer tijd dan ik dacht. Dit kwam deels door problemen die ik tegenkwam bij die onderdelen, en doordat ik onderschat had hoeveel tijd dat allemaal ging kostten.

Met sprints van drie weken (zoals benoemd bij hoofdstuk 6.2 Epics en sprints) was er nog tijd voor ongeveer 4 á 5 sprints. Wat ik in ieder geval bij het einde van het afstudeertraject af wilde hebben, was dat het activiteiten systeem erin zat, gebruikers konden inloggen, privileges systeem erin zat, gebruikers een dashboard hadden met de games erop waar ze privileges voor hebben en dat gebruikers de games kunnen spelen.

Op dit moment dacht ik wel, dat ik voldoende tijd had om op zijn minst die onderdelen te implementeren. Het administratie gedeelte (waar administrators onder andere gebruikers kunnen aanmaken, activiteiten geven aan gebruikers en privileges toekennen aan gebruikers) verwachtte ik geen tijd meer voor te hebben tijdens mijn afstudeertraject, maar die onderdelen hadden dan ook een lagere prioriteit dan de onderdelen die ik er in ieder geval in wilde hebben.

9 Sprint 1 – Activiteiten systeem en inloggen

De eerste sprint van het afstudeertraject richtte zich op het activiteiten systeem en het inloggen in de applicatie.

Aan het begin van de dag dat deze sprint begon kregen we te horen, dat CGI niet in staat was de credits te verhogen in Azure. Tijdens het bespreken van dit bericht met mijn begeleider probeerden we op allerlei services de kosten van Azure te reduceren.

Wij kwamen al snel tot de conclusie dat het niet mogelijk was de kosten voldoende te reduceren en de enige optie was, dat ik de API en de microservice applicaties lokaal ging draaien in Docker. Dit kwam, omdat de containers van de frontend, API en de microservices laten draaien in Azure het grootste deel van de kosten waren. Daarom hadden wij de User Story 'Als developer kan ik de API en microservices lokaal runnen als development omgeving.' aan de backlog toegevoegd en bovenaan gezet met hoogste prioriteit.

Doordat de frontend, API en microservice applicaties niet meer in Azure konden draaien, was de website niet meer online beschikbaar, maar dat maakte op dat moment nog niet uit aangezien het platform een concept was en nog niet naar productie zou gaan. Wanneer het platform naar productie zou gaan, moeten de credits wel worden aangepast, want anders is het platform na een maand offline, door het gebrek aan credits.

Op basis van onze schatting wat haalbaar is in een sprint hadden wij de volgende User Stories in de sprint gezet:

Epic	User Story	Schatting effort points
	Als developer kan ik de API en microservices lokaal runnen als development omgeving, zodat ik de Azure containers niet meer nodig heb voor development	8
Activiteiten Systeem	Als systeem wil ik op de frontend de bomen van alle taken kunnen verkrijgen van een activiteit van een gebruiker waarbij de activiteit gestart is, nog niet geëindigd is of geen datum heeft, zodat ik bomen van geldige taken heb om uit te kunnen voeren	24
Activiteiten Systeem	Als systeem wil ik de taken van een activiteit van een gebruiker kunnen uitvoeren op de goede volgorde van laagste kind taak tot hoogste parent taak om de taken uit te kunnen voeren op de goede volgorde	24
	Als product owner wil ik dat de hoofdpagina de inlogpagina is, zodat dat de eerste pagina is die gebruikers zien als ze de website openen	1
Users Systeem	Als gebruiker wil ik in kunnen loggen met mijn email en gebruikersnaam om toegang te krijgen tot mijn account	7

Figuur 4: Sprint 1 user stories

9.1 Aanpak van sprint

Als eerste stap wilde ik het concept project van mijn begeleider bestuderen, om te zien hoe het activiteiten systeem daar werkte (zie hoofdstuk 8.3.2 De resultaten van het gesprek en de impact daarvan voor het verkrijgen van het project). Met die kennis zou ik dan een inzicht hebben, op wat ik al dan niet in mijn ontwerp wilde hebben.

Als tweede stap wilde ik een ontwerp maken voor de frontend. Ik wilde een ontwerp hebben voor de frontend, zodat ik precies wist wat de frontend nodig had van de backend.

Als derde stap wilde ik ontwerpen wat er op de backend moest gebeuren, om alles dat de frontend nodig had van de backend te weten te komen.

Als vierde stap wilde ik het activiteiten systeem implementeren (op zowel de frontend applicatie, de API applicatie en de activiteiten microservice applicaties).

Als vijfde stap wilde ik het inloggen van een gebruiker implementeren (op zowel de frontend, API en de microservice applicaties). Ik wilde dit als volgende stap, zodat ik dan een taak heb (authentiseren van het inloggen) waarmee ik de werking van het activiteiten systeem kan testen.

Als zesde stap wilde ik de code volledig testen waarbij de code op de frontend, de API en alle microservice applicaties 100% succesvol getest is.

Als laatste stap wilde ik de code reviewen voor eventuele verbeteringen, opschoning van redundante code en om opmerkingen toe te voegen.

9.2 Bestuderen activiteiten systeem

Tijdens het bestuderen van het concept project van mijn begeleider vond ik meerdere problemen bij het activiteiten systeem, waarbij ik er een paar zal benoemen.

Dit concept project had een hardcoded serie van taken, dat het systeem uit moest voeren in een serie achter elkaar. Elke taak had als variabele de volgende taak die uitgevoerd moest worden, nadat het zijn eigen taak gedaan had. De taak uitvoerders hadden ook veel dependenties[1] met andere componenten om de taken uit te voeren. De taken waren dus zeer afhankelijk van elkaar en van andere componenten.

De taak uitvoerders werden aangemaakt bij het inladen van een component, dus elke keer dat component werd geladen, werd de lijst met taak uitvoerders opnieuw aangemaakt, terwijl dat niet nodig was.

Het bestuderen van het activiteiten systeem van zijn concept project gaf mij wel een beeld hoe een statisch activiteiten systeem eruit ziet.

9.3 Ontwerp frontend

Na het bestuderen ging ik mijn eigen ideeën combineren met wat inspiratie van het concept project van mijn begeleider, om zo tot een zo goed mogelijk ontwerp te komen voor een activiteiten systeem.

Voordat ik begon met het ontwerpen ging ik eerst opschrijven wat de frontend moet hebben, om een goed werkend activiteiten systeem te krijgen zonder code duplicatie. Hieruit kwam:

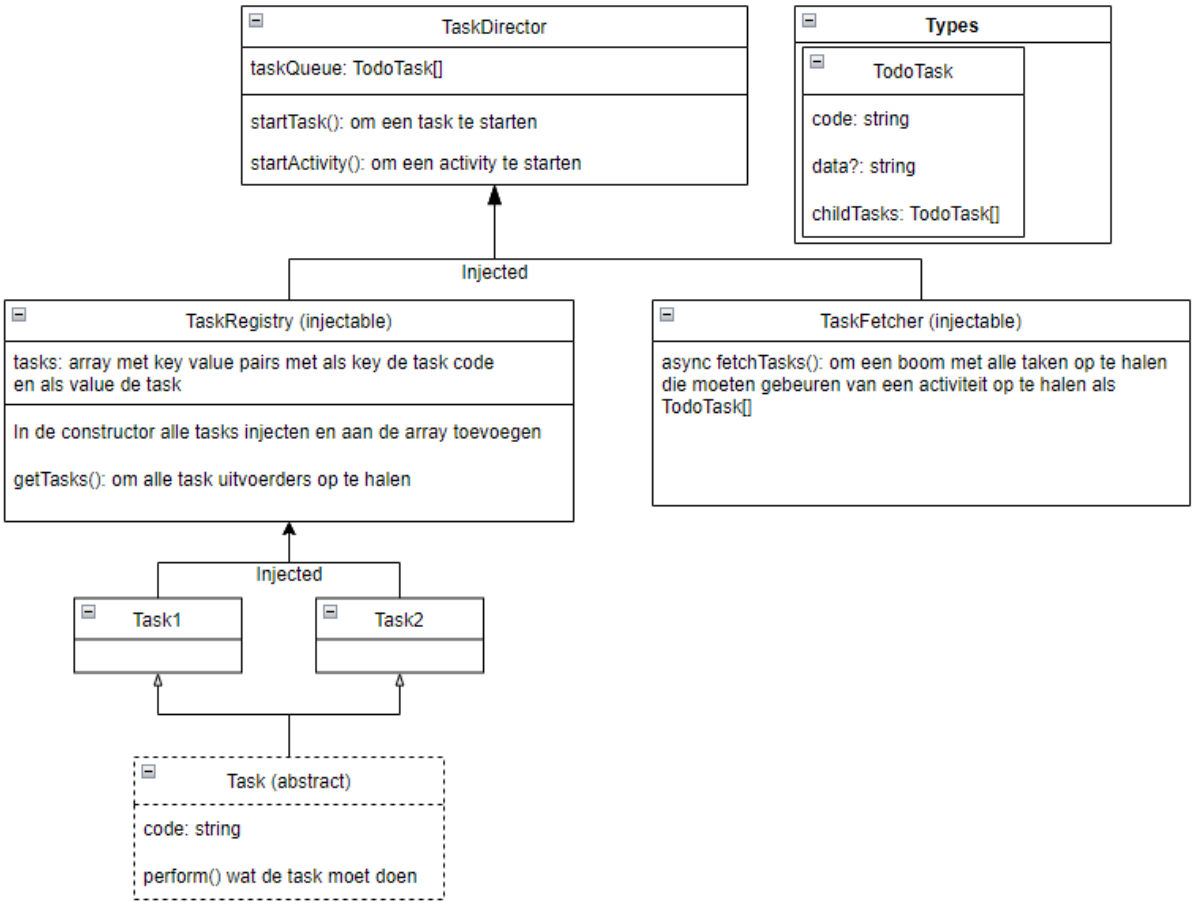
- Het kan de taken die gedaan moeten worden voor een activiteit ophalen van de backend.
- Het heeft de lijst met taken die gedaan moeten worden.
- Het kan taken uitvoeren op de goede volgorde.
- Het heeft klassen die taken kunnen uitvoeren.
- Het heeft een abstracte taak uitvoerder klasse om de taak uitvoerders klassen mee te maken.
- Het heeft een lijst met alle taak uitvoerders met taak code om voor elke taak die de frontend moet doen te weten wie die taak moet uitvoeren.

Nadat ik dit bepaald had begon ik met een ontwerp met wie wat doet, waarbij ik mij hield aan de principes voor Object Oriented Design (OOD), om een zo object georiënteerd mogelijk ontwerp te maken (SOLID, 2022).

Het gemaakte ontwerp is te zien is in Figuur 5: Activiteiten systeem ontwerp.

---

[1] Ik gebruik in dit dossier meerdere malen het Engelse woord dependenties in plaats van het Nederlandse woord afhankelijkheden, omdat het woord dependenties een betere context geeft dan het Nederlandse woord afhankelijkheden. De reden voor dependenties te gebruiken leg ik alleen hier uit en zal ik verder in het verslag niet meer beschrijven.



Figuur 5: Activiteiten systeem ontwerp

9.4 Ontwerp backend

Ik besloot de boom van taken aan te maken op de activiteiten microservice, in plaats van op de frontend (zie hoofdstuk 8.3.3 Werking nieuw activiteiten systeem voor wat de boom van taken is). Deze boom wilde ik op de activiteiten microservice maken, zodat de frontend precies terug krijgt wat het nodig heeft om de taken te starten en niet eerst nog van alle taken een boom zou moeten maken.

De aanvraag voor de taken van een activiteit van een gebruiker stuur ik naar de activiteiten microservice, omdat deze microservice verantwoordelijk was voor taken en activiteiten.

De users microservice wilde ik gebruiken voor het authentiseren van het inloggen van een gebruiker. Ik besloot dat op deze microservice te ontwikkelen, omdat deze microservice verantwoordelijk was voor de gebruikers.

Op de API besloot ik voor elke microservice een controller en provider aan te maken. Dit besloot ik, zodat er op de API voor de aanvragen voor elke microservice, door een eigen controller worden ontvangen, in plaats van dat er één controller is die alle aanvragen ontvangt. Elke controller wilde ik dan ook een eigen provider geven, zodat elke provider alleen maar de aanvragen afhandelt van één microservice. Elke microservice een eigen controller en provider geven, maakt de API zeer object georiënteerd. De controllers en providers volgen dan de single-responsibility principle van OOD (Single responsibility principle, 2022), omdat elke controller en provider dan maar één verantwoordelijkheid heeft .

9.5 Implementatie deel 1

Bij het implementeren van het activiteiten systeem bij de eerste stap liep ik nog niet tegen problemen aan, maar er waren dan ook nog geen taken, om op de frontend de werking van het systeem te testen.

Bij het implementeren van het inloggen bij stap twee, kwam ik wel een probleem tegen, wanneer het inloggen niet succesvol was.

Er was op dat moment nog geen melding dat een gebruiker laat weten als zijn inlogpoging niet gelukt is.

Om een taak uitvoerder bijvoorbeeld tekst weer te laten geven, in het geval een gebruiker faalt in te loggen, moest de taak uitvoerder wel eerst de mogelijkheid hebben om (tekst) elementen te beïnvloeden.

Maar wat was nou de beste manier, om de taak uitvoerders elementen te geven, om te beïnvloeden?

9.6 Het elementen probleem

Het systeem dat ik wilde hebben voor het bijhouden van elementen is dat elke taak uitvoerder zelf alles ophaalt wat hij nodig heeft, zodat elke taak uitvoerder onafhankelijk zou zijn. Op die manier is de code ook veel meer Object georiënteerd, omdat zowel de taak uitvoerders als de componenten met de elementen, dan meer voldoen aan het Single-Responsibility Principle van OOD (Single responsibility principle, 2022).

Er waren vier mogelijke oplossingen die ik kon bedenken om taak uitvoerders referenties naar elementen te geven. Van deze oplossingen moest ik een keuze maken in welke oplossing het beste het probleem oploste.

9.6.1 Elementen meegeven aan de taak uitvoerders

Eerst ging ik bekijken hoe mijn begeleider dat gedaan had in zijn concept project. Hier had hij aan elke taak uitvoerder de elementen referenties meegegeven die hij nodig had om zijn taken uit te voeren (zie hoofdstuk 8.3.3 Werking nieuw activiteiten systeem voor meer informatie over taken). Zijn applicatie had maar één pagina, waar het activiteiten systeem op werkte. Het maakte de taak uitvoerders aan op het component van die pagina.

Hier waren een aantal problemen mee als ik die manier wilde toepassen op mijn applicatie. Mijn frontend zal meerdere pagina's bevatten, waardoor er op elke pagina dan taak uitvoerders zouden worden aangemaakt, om ze elementen referenties mee te geven van die pagina. De taak uitvoerders op verschillende locaties aanmaken is wat ik wilde voorkomen bij het ontwerpen.

Deze manier van taken elementen geven levert ook inefficiëntie op. Het eerste nadeel hiervan is dat de applicatie elke keer dat het een nieuwe pagina laad, de taak uitvoerders opnieuw aanmaakt. Niet alleen zal dit totaal onoverzichtelijk worden als elk component tientallen taak uitvoerders zou hebben, maar het is ook niet object georiënteerd. Dit komt, omdat elke taak uitvoerder dan zwaar afhankelijk is van het component dat die taak uitvoerder aanmaakt. Een ander nadeel hiervan is, wanneer er een serie van taken is dat de applicatie moet uitvoeren, waarbij de taken over meerdere pagina's zijn verspreid, het systeem een error kan geven dat die taken niet bestaan op de huidige pagina.

Deze methode had ik om de bovenstaande redenen niet toegepast.

9.6.2 Eventemitter

Het tweede idee om dit te doen was om van elke taak uitvoerder een eventemitter te maken. De taak uitvoerder kan dan een event emitten met eventuele data, waarna een functie in een component dan een taak uitvoert, wanneer het bericht waar die functie op wacht wordt emit, door aan dat bericht te subscriben.

Het probleem met dit idee was dat elk component dan meerdere functies zou hebben, die dan subscribed zijn aan verschillende berichten van verschillende taken.

Deze manier was totaal niet object georiënteerd, want niet alleen heeft elke component code die een bepaalde taak uitvoerder dan zou moeten hebben, maar doet de applicatie eigenlijk ook elke taak twee keer, één keer in de taak uitvoerder en één keer in het component.

Deze manier heb ik om die reden dus niet toegepast. Ik had nog op andere manieren gekeken om een eventemitter te gebruiken, maar ik kwam elke keer op dezelfde conclusie, dat deze manier geen kwalitatieve code oplevert en de code ook niet object georiënteerd is.

9.6.3 Element id

De derde manier die ik vond was om DOM manipulatie te gebruiken, om de elementen uit de DOM te halen in de taak uitvoerders (DOM manipulation, 2022). Hoewel deze manier wel zou werken zonder extra code in het component (zoals elementen meegeven of events emitten), was dit geen nette manier van elementen ophalen, omdat de taak uitvoerder dan het element via de DOM vindt, in plaats van via de code.

Deze optie had om de bovenstaande redenen niet mijn voorkeur. Ik ging dus vervolgens verder zoeken naar oplossingen.

9.6.4 Observable pattern

De laatste manier die ik had gevonden, om dynamisch elementen te krijgen, was door gebruik te maken van het observable design pattern. Met het observable design pattern wordt gebruik gemaakt van variabelen die observable zijn en subscribers die functies zijn. Door een functie aan een subject variabele te subscriben, wordt die functie uitgevoerd, elke keer als de waarde van zijn subject verandert.

Met deze methode kan elk component alle elementen die de taak uitvoerders nodig hebben, in een service opslaan, bij het laden van dat component. Elke taak uitvoerder kan dan subscriben aan dat subject met een lijst van elementen in die service, zodat elke keer dat die lijst geüpdatet wordt, de taak uitvoerder de geüpdatet lijst van elementen krijgt.

Op deze manier zoekt elke taak uitvoerder zelf uit wat het nodig heeft, waardoor de applicatie alleen nog maar bij de componenten, de elementen voor de taak uitvoerders hoeft op te slaan in de service. De componenten weten zo niks van de taken en slaan ze alleen maar referenties naar elementen op in een service. Deze implementatie maakt de code redelijk Object georiënteerd, maar nog niet helemaal, omdat de elementen niet zichzelf opslaan, maar dat het component dat doet.

9.6.5 Conclusie

Ik besloot voor de Observable pattern manier te gaan. Ondanks dat de componenten bij deze methode dan nog steeds code bevatten over die elementen, geeft het betere kwaliteit code en meer object oriëntatie dan de andere methoden.

De taak uitvoerders en de service die de elementen referenties bijhoudt, volgen dan het single-responsibility principle van OOD, waarbij de klassen maar één verantwoordelijk hebben. De componenten met elementen referenties volgen dat principe dan wel minder strak, omdat die componenten dan nog wel de elementen referenties moeten opslaan in de service. Echter was deze oplossing de beste van oplossing van de vier oplossingen, waardoor het helaas onvermijdelijk was dat die componenten die principe minder strak volgen.

9.7 Implementatie deel 2

Het implementeren van de gekozen observable pattern methode voor het elementen probleem verliep naar wens.

De laatste stap van het inloggen, was dat er wat moest gebeuren als het inloggen wel succesvol was. Hiervoor maakte ik een taak uitvoerder, dat de gebruiker naar een andere pagina kon sturen en stuurde de gebruiker naar de dashboard pagina (dit deed ik door een leeg dashboard component te maken).

Bij het implementeren van deze taak uitvoerder liep ik niet tegen problemen aan, maar ik kwam er wel achter dat het wachten op een taak niet werkte. Dit probleem had ik ook opgelost, door gebruik te maken van het observable pattern, waarbij er gewacht wordt met het uitvoeren van een taak, totdat alle taken, waar die taak op moet wachten, gedaan zijn.

Nu dat alles naar wens werkte kon ik naar de volgende stap gaan.

9.8 Clean-up en review

Bij deze stap waren er een aantal punten waarbij ik de code refactored had.

9.8.1 Authentiseren taak uitvoerder

Als er op de inlog knop werd gedrukt haalde de frontend de taken op via de backend uit de database van de standaard gebruiker, met de login activiteit code. De enige taak die in de database stond bij de login activiteit van de standaard gebruiker in de database, was authentiseren. Deze authentiseer taak voerde de frontend dan uit en, indien het inloggen succesvol was, haalde het vervolgens de inlog taken op van de ingelogde gebruiker.

Wat ik besloot te doen was dat ik eerst authentiseer op de inlog pagina en het bij een succesvolle login de taken van die gebruiker ophaalt, om dan die taken uit te voeren (bijvoorbeeld ga naar game A of ga naar dashboard). Bij een niet succesvolle login werd in tekst weergeven aan de gebruiker dat het inloggen niet succesvol was.

Ik deed deze refactor vanwege meerdere redenen. Ten eerste is het inloggen dan sneller, want dan hoeft de applicatie niet eerst de inlog taak van de standaard gebruiker op te halen. Ten tweede, is het netter om eerst te authentiseren en dan pas de taken van de correcte gebruiker op te halen, omdat de standaard gebruiker dan niet meer meegestuurd moet worden, om eerst de authentiseer taak op te halen. Door de standaard gebruiker niet meer mee te sturen is de code schoner en meer object georiënteerd, want inloggen doet dan alleen nog maar authentiseren, in plaats van eerst een taak op laten halen om dan te authentiseren. Ten derde is het ook onnodig om de authenticatie taak op te halen, want bij elke inlog poging moet het altijd die inlog poging authentiseren, om te zien of de inlog gegevens correct zijn.

9.8.2 Angular Material en validatie

Op dit moment zag de inlogpagina er nog niet heel mooi uit. Om deze pagina er beter uit te laten zien besloot ik gebruik te maken van Angular Material.

Angular Material is een package dat visuele componenten bevat, die gemaakt zijn met Material Design (Hoofdpagina, 2022). Material Design is een Android georiënteerd design van Google, waarbij de bewegingen die de gebruiker uit zou voeren op papier, worden nagebootst (Material Design, 2022). Een groot aantal applicaties die Google heeft ontwikkeld zijn gemaakt op basis van Material Design of gaat nog migreren naar Material Design (Google's Material, 2022). Deze componenten zijn ook voor andere frameworks (dan Angular) beschikbaar, zoals bijvoorbeeld het React framework (MDC Web on other frameworks, 2022).

De componenten van Angular Material zien er grafisch niet alleen goed uit, maar heeft ook validatie inbegrepen. Deze validatie tools gebruikte ik om de inloggegevens eerst te valideren, voordat ik de inloggegevens authentiseer (bijvoorbeeld checken dat het email adres wel echt een email adres is en @ er in heeft staan). Door eerst te valideren, kan de gebruiker eerder zien of zijn inlog poging gelukt is of niet, doordat er niet eerst een aanvraag voor authentiseren naar de backend wordt verstuurd.

9.9 Testen

Het testen maken liep zeer soepel zonder enige grote problemen. De paar kleine problemen die ik tegenkwam hadden te maken met mijn onervarenheid over het schrijven van testen voor Angular. Ik heb een 100% code coverage bereikt voor zowel de frontend, API en elke microservice. Voor de API heb ik end-to-end testen gemaakt, om te testen of het bij een http aanvraag de goede functie start.

9.10 Retrospectief

Deze sprint had uiteindelijk 4 weken geduurd. Dat de sprint langer heeft geduurd dan gepland, kwam voornamelijk, omdat ik onderschat had hoelang ik met elke user story bezig zou zijn. Deze ervaring wilde ik meenemen bij het plannen van de volgende sprint, waarbij ik voor de volgende sprint minder zal inplannen.

Het implementeren op de API, de users microservice en de activiteiten microserves verliep redelijk soepel en naar wens zonder veel problemen. Bij geen van deze problemen hoefde ik een keuze of afweging te maken.

9.10.1 User stories verkorten

Voor de volgende sprint ga ik de user stories onderverdelen in taken. Dit ging ik doen, omdat sommige user stories nog onderzoek vereisen wat veel tijd kost. Het was dus mogelijk dat een user story waar onderzoek voor gedaan moest worden niet haalbaar was binnen één sprint.

Door user stories met onderzoeken erin onder te verdelen in taken, kan er bijvoorbeeld in een user story een ontwerp taak, onderzoek taak, implementeer taak en een test taak zitten. Dan zou ik in de eerste sprint het onderzoek doen en ontwerpen, om dan in de eerstvolgende sprint de user story te implementeren en te testen. Dat geeft dan een beter beeld voor de opdrachtgevers waar ik op dat moment precies mee bezig ben.

9.10.2 Element referenties opslaan toekomst

De huidige manier van element referenties opslaan vond ik nog niet ideaal, omdat de componenten die elementen bevatten die opgeslagen moesten worden, nog wel al die elementen referenties moest sturen naar de service dat de elementen referenties bijhoud.

Op dit moment had ik nog te weinig kennis van Angular om hier een betere oplossing voor te bedenken, maar als ik later een betere oplossing voor het opslaan van referenties vindt zal ik dat zeker toepassen.

9.10.3 Ontwerp wijzigingen

Het ontwerp klopte bijna helemaal met de gerealiseerde code. Het enige verschil was dat de taak uitvoerders niet meer inject worden maar dat de TaskRegistry klasse voor elke taak code een nieuwe taak uitvoerder aanmaakt.

Ik moest dit zo doen, omdat je bij het injecteren van een klasse geen variabele aan die klasse mee kan geven (behalve één specifieke Angular klasse). In de TaskRegistry kon ik dus bij het injecteren van een taak uitvoerder of bij het toewijzen van een taak uitvoerder aan taak code, geen variabele meegeven.

Om naar het dashboard te gaan had ik een taak uitvoerder gemaakt dat naar een meegegeven pagina kan gaan op de frontend (zie hoofdstuk 9.7 Implementatie deel 2). Deze taak zou dus niet werken als je injecteerde in plaats van nieuw aanmaakte in de TaskRegistry.

Aan de taak uitvoerders in de TaskRegistry gaf ik de injector van de Dependency Injection door, waar de taak uitvoerders dan hun overige dependenties uithaalde en die taak uitvoerders alleen maar op te halen waren vanaf de TaskRegistry. De taak uitvoerders volgden daardoor nog steeds de OOD principes, doordat het nog steeds al zijn dependenties kreeg via Dependency Injection en alleen maar een sterke koppeling kreeg met de taak uitvoerders.

10 Sprint 2 – Onderzoek en ontwerp games systeem

De tweede sprint ging over het ontwerpen van het games systeem, met behulp van onderzoek naar het concept project van mijn begeleider. Bij deze sprint hadden wij in plaats van hele user stories, een paar taken van één user story ingepland. Mijn begeleider en ik hadden dit samen besloten, omdat ik onderzoek moest gaan doen naar de code van het concept project van mijn begeleider.

Dat project had niet alleen code aan de kant van de web applicatie (Angular), maar ook aan de kant van de game. De leesbaarheid en de structuur van de Angular code, in combinatie met de game code, zorgde voor een onderzoek dat veel tijd zou gaan kosten. Ook een factor in dit besluit, was dat de sprint vroeg eindigde vanwege de kerstvakantie, waardoor deze sprint maar twee en een halve week duurde (10 werkdagen).

In Figuur 6: Sprint 2 user story taken is een overzicht te zien van de user story met taken en de schatting van effort points van deze sprint:

Epic	User story	Taak	Schatting effort points
Games systeem	Als gebruiker wil ik een game kunnen spelen op de frontend waar ik geen privileges voor nodig hebt zodat ik de games kan spelen die geen privileges nodig hebben	Onderzoek het project van Rens	48
		Ontwerp games systeem	16

Figuur 6: Sprint 2 user story taken

Bij de start van deze sprint had ik met mijn begeleider gekeken naar de kosten van Azure Databricks, aangezien er al niet genoeg credits waren, om de container instanties te draaien. We kwamen er al snel achter dat ook Azure Databricks buiten het budget viel. Dit zal geen problemen opleveren voor het platform, maar het heeft wel invloed op de onderhoud baarheid van het platform. Dit komt, omdat Azure Databricks de URL's naar de bestanden in Azure storage eenvoudiger maakt, wat beter is te onderhouden, dan de standaard URL's naar Azure Storage.

Als eerste besloot ik me te oriënteren op de code, om een beeld te krijgen hoe zijn programma de games laad en hoe de communicatie tussen de game en de webapplicatie werkt. Ik besloot dit te doen om een beter beeld te krijgen hoe ik dit onderzoek het beste aanpak.

10.1 Oriëntatie resultaten

Als eerste kwam ik erachter hoe de communicatie bus tussen de game instantie en de webapplicatie werkte. De webapplicatie kon berichten ontvangen, door aan een window element in de browser te subscriben, waar de game instantie zijn berichten naar toe stuurde. De webapplicatie verstuurde events naar de game instantie, door direct een bericht te sturen naar een functie van een object van de game instantie (zie (Unity web player and browser communication, 2022) hoe dat werkt).

Als tweede kwam ik er achter dat de webapplicatie de games inlaadt, door een serie van events af te lopen. De webapplicatie en de game instantie gebruiken de communicatie bus, om de events naar elkaar te communiceren. De webapplicatie start deze serie door de game instantie aan te maken, waarna de game het eerste event van de serie naar de webapplicatie stuurt. Dan stuurt de webapplicatie weer een event terug naar de game, wanneer het ontvangen event uitgevoerd is. Elke game gebruikt dezelfde game instantie, maar games zelf worden geladen via bundels.

Elke game heeft in ieder geval twee bundels nodig, één bundel voor alle assets van de game en de andere bundels voor de verschillende scenes (de scene is de omgeving waar de game zich afspeelt). Deze bundels haalde de webapplicatie op van zijn eigen server (zie Bijlage 10: Angular heeft een eigen server uitleg voor hoe dat werkt). Indien een game zijn bundels niet krijgt kan het spel niet geladen worden.

Het derde resultaat, was dat de webapplicatie op verschillende wijze events verstuurde. Het stuurde events bij een activiteit van een gebruiker, als reactie op het ontvangen van een event van de game, wanneer een ander event klaar was of terwijl een ander event bezig was.

Het vierde resultaat was dat de code van de webapplicatie chaotisch en onoverzichtelijk was, doordat vrijwel alle code (over het inladen van games), was verspreid over twee klassen. De code was ook chaotisch, omdat sommige dependenties van de klassen die events verstuurden op de webapplicatie, soms wel 6 dependenties diep kon gaan en de events die gestuurd werden, ook op meerdere plaatsen stonden.

10.2 Plan na oriëntatie

Ik besloot het aan te pakken door eerst informatie te verzamelen, dan de informatie te specificeren, om vervolgens tot een ontwerp te komen. Ik pakte dit aan door de volgende stappen te doorlopen.

Ik besloot eerst alle events, die nodig waren om een game te starten, te verzamelen op volgorde. Dit wilde ik om een beeld te krijgen, over wat er gebeurt tijdens dit proces.

Als tweede besloot ik alle code gerelateerd aan de events bij elkaar te verzamelen en alle gedupliceerde code eruit te halen, om alleen nog de code over te houden dat nodig is om een game te laden en te starten.

Door de vorige stappen zou ik te weten komen, wat er werd gedaan om een game te starten en welke code van het project voor dit proces werd gebruikt. Echter weet ik dan nog niet wat de webapplicatie voor elk event moet doen, omdat het proces om een game te laden en te starten achter elkaar door gaat.

Als derde stap besloot ik daarom alle events te specificeren, om van elk event precies te weten te komen wie het verzend (webapplicatie of de game), wanneer dat event wordt verzonden en wat de webapplicatie moet doen bij dat event. Door deze stap kom ik te weten welk event het proces start en welk event het proces eindigt.

Als laatste wilde ik een ontwerp maken. Ik kon deze stap dan nemen, doordat ik bij de vorige stappen alle benodigde kennis had verzameld, om een ontwerp te maken.

10.3 Verzamelen van de serie van events resultaten

De tabel met alle events om een game te starten, dat ik tijdens deze stap had gemaakt, staat in

Bijlage 4: Tabel serie van events.

Ik had de informatie van de tabel verzameld, door veel te loggen en alle game code over events te bestuderen.

Door de resultaten van deze stap, had ik een goed beeld gekregen over het proces om een game te laden en te starten. Tevens gaf mij dit een beeld over de rol van de webapplicatie bij elke stap van dit proces.

10.4 Verzamelen code resultaten

Bij deze stap had ik gebruik gemaakt van de tabel dat gemaakt was bij de vorige stap, om te helpen de code om een game te laden en te starten te identificeren.

Het resultaat van deze stap, was één klasse met ongeveer 160 regels code. Deze klasse gaf een veel duidelijker beeld, dan het volledige project, hoe de webapplicatie de ontvangen events afhandelde.

10.5 Lostrekken van events resultaten

Ik voerde deze stap uit door in de code van de webapplicatie veel te loggen, waardoor ik precies kon zien wie het event verstuurt, waar het event wordt verstuurd, wanneer het event werd verstuurd en wanneer het een event ontvangt van de game.

De tabel die uit dit specificeren is gekomen is te zien bij Bijlage 7: Tabel losgetrokken events.

10.6 Analyse losgetrokken events tabel

Wat uit de tabel is op te maken is dat er zes events zijn dat Angular verstuurt en vijf events dat de game verstuurt.

Van die zes events zijn er twee events die de webapplicatie verstuurt als response op de download van een bundel, één event als reactie op het ontvangen van de container-install-app event, twee events om een game te starten en één event voor het uitladen van een scene.

De twee events die de webapplicatie moet versturen om een game te starten zijn app-container-start-activity en app-container-load-content-bundle-scene.

Met app-container-load-content-bundle-scene laad de game instantie de scene in, om dan met app-container-start-activity de scene te starten.

Het event voor het uitladen van een scene is app-container-unload-content-bundle-scene. Dit moet verstuurd worden, wanneer een gebruiker klaar is met het spel.

De twee events die de webapplicatie verstuurt, wanneer het klaar is met het downloaden van een bundel, zijn container-async-operation-status en container-async-operation-chunk.

Wanneer de webapplicatie het event container-install-app ontvangt, moet het naar de game het event container-configuration sturen met de paden van de bundels die de game nodig heeft.

De webapplicatie verstuurd zelf 5 events naar de game. Bij alle vijf events moet de game wat doen, wanneer het één van die events ontvangt.

Bij het event container-start-async-operation wordt een event meegegeven dat async gestart wordt. Het downloaden van een bundel is een async operatie, dus bij dit event moet er gekeken worden of het meegegeven event aan dit event, het event container-install-bundle is.

Met dit resultaat wist ik precies van elk event dat de webapplicatie ontvangt, wat er dan moet gebeuren en van elk event dat het verstuurt, wanneer het verstuurd moet worden en wat er aan meegegeven moet worden.

10.7 Ontwerp maken frontend

Om het ontwerp te maken gebruikte ik de code van hoofdstuk 10.4, de tabel die ik had gemaakt in hoofdstuk 10.5, om de code uit elkaar te trekken en de informatie uit de analyse van hoofdstuk 10.6, om te weten wat de frontend moet doen, om alle events op de goede volgorde af te lopen. Net als bij hoofdstuk 9.3 Ontwerp frontend volgde ik voor dit ontwerp ook de principes van OOD.

Als eerste maakte ik een eigen klasse voor alle functies en variabelen, die gingen over de communicatie tussen de game en de webapplicatie. Vervolgens had ik alle events in een apart bestand gezet, zodat de events te importeren zijn door elke klasse dat events nodig heeft.

Vervolgens had ik voor elk event dat de webapplicatie ontvangt een eigen klasse gemaakt. Deze klassen erven van een abstracte klasse dat de berichten bus injecteert en een abstracte functie bevatte. Deze functie bevat voor elke event klasse, de code dat de webapplicatie moet uitvoeren, wanneer de webapplicatie het event voor die klasse ontvangt. Door de abstracte klasse, staat de berichten bus code maar bij één klasse, in plaats van bij elke event klasse.

Een aantal events moesten data meekrijgen om hun code uit te voeren. Bij het project van mijn begeleider kregen ze deze data hardcoded mee uit de serie met events. Hoewel er verschil zit tussen de data dat elk van die events meekrijgt, is al die data wel gerelateerd aan de naam van de game. Ik besloot daarom de naam van de game te gebruiken, om dynamisch de data aan te maken dat die events nodig hebben.

Het component waar de game op wordt geladen is de game pagina, maar ik moest nog wel aan dit component de naam van de game meegeven dat het moet laden. Nadat ik hier onderzoek naar had gedaan

besloot ik de naam van de game aan het component te geven via route parameters (parameterised routes, 2022).

Het Angular framework heeft standaard een router component, om naar verschillende pagina's op de website te gaan. Dit component heeft een lijst met routes, waarbij elke route een combinatie is van een URL en een component. Met route parameters kan je aan een route een parameter meegegeven, dat de parameter vervolgens doorgeeft aan het component.

De URL dat ik gebruik om naar het game component te gaan is "/game/:id". Als je bijvoorbeeld gaat naar "/game/some-game-name", dan kan het component de naam "some-game-name" uit de route halen.

Er waren drie events die Angular zelf moest starten en niet activeert als reactie op het ontvangen van een event. Deze drie events waren er om de game te starten en uit te laden (te lezen in 10.6 Analyse losgetrokken events tabel).

De events om een game te starten deed ik in een eigen klasse en liet ik de klasse versturen naar de game, wanneer de frontend het event container-ready ontvangt, waardoor de game dan start.

Het event om de game uit te laden had ik ook een eigen klasse gegeven en liet ik versturen naar de game, wanneer de frontend van de game het event app-container-complete-activity zou ontvangen. Dan wordt de game uit geladen wanneer de gebruiker klaar is met spelen.

Om naar het game component te gaan besloot ik dezelfde taak uitvoerder te gebruiken als bij hoofdstuk 9.7 Implementatie deel 2, om naar de game pagina te gaan.

Het ontwerp is te zien in Bijlage 8: Ontwerp games systeem.

10.8 Ontwerp maken backend

Voor dit systeem op de backend wilde ik de games microservice gebruiken, omdat die microservice verantwoordelijk is voor de tabellen gerelateerd aan de games tabel in de database. Deze microservice wilde ik als enige microservice verbinden met Azure storage, omdat alleen de games op Azure storage staan en de andere microservices Azure storage niet nodig hebben.

Net als voor de andere microservices, wilde ik op de API voor deze microservice een eigen controller en provider hebben (zie hoofdstuk 9.4 Ontwerp backend waarom ik dit wilde).

Op de games microservice zelf, besloot ik een tweede provider aan te maken. Eén provider wilde ik dan laten verbinden met de database en de database aanvragen afhandelen en de andere provider wilde ik laten verbinden met Azure Storage, om de bundels van te downloaden. Op die manier was de microservice meer object georiënteerd volgens het single-responsibility principle van OOD. Elke provider had dan zijn eigen verantwoordelijkheid, waarbij de ene provider verantwoordelijk was voor de database en de andere provider verantwoordelijk was voor Azure storage.

10.9 Retrospectief

Deze sprint was volledig volgens plan en naar wens verlopen. Ik had in de sprint het hele onderzoek naar het games systeem uit kunnen voeren. Bij deze sprint had ik ook zoals gepland voldoende tijd om de eerste sprint en deze sprint in dit dossier te verwerken.

Ik was zeer tevreden met het ontwerp en hoe de stappen waren verlopen.

11 Sprint 3 – Implementeren games systeem

In deze sprint vervolg ik de user story van de vorige sprint (zie 10 Sprint 2 – Onderzoek en ontwerp games systeem).

Epic	User story	Taak	Schatting effort points
Games systeem	Als gebruiker wil ik een game kunnen spelen op de frontend waar ik geen privileges voor nodig heb, zodat ik de games kan spelen die geen privileges nodig hebben	Implementeren backend	28
		Implementeren frontend	28
		Testen backend	12
		Testen frontend	12

Figuur 7: Sprint 3 user story taken

Ik besloot dit stap voor stap te implementeren, omdat ik op dat moment niet bekend was met een game in een browser draaien en niet bekend was met unity bundels. Bij het implementeren van het games systeem was er dus een grote kans dat ik daar wat extra tijd voor nodig zou hebben en ik veel problemen tegen kon komen.

11.1 Aanpak van implementatie

Als eerste stap wilde ik de bundels op dezelfde manier ophalen, als bij het concept project van mijn begeleider, waarbij de webapplicatie de bundels ophaalt van zijn eigen server (zie hoofdstuk 10.1 Oriëntatie resultaten).

Om deze stap te implementeren, maakte ik gebruik van de verzamelde code (zie hoofdstuk 10.4 Verzamelen code resultaten) en het ontwerp van het games systeem (zie hoofdstuk 10.7 Ontwerp maken).

Als tweede stap wilde ik de bundels downloaden van Azure storage, maar nog niet via de backend (zie hoofdstuk 8.2.2.5 Conclusie voor de keuze voor Azure storage). Dit wilde ik zo doen, omdat het goed mogelijk was dat er wat complicaties konden ontstaan bij het downloaden van Azure storage (bijvoorbeeld beveiliging, verkeerde URL's en de game bundels corrupt ontvangen). Door deze stap te nemen was de kans klein dat dezelfde fouten voor zouden komen bij de volgende stappen.

Als derde stap wilde ik de bundels downloaden van Azure storage via de backend volgens het ontwerp van hoofdstuk 10.8 Ontwerp maken backend.

Als vierde stap wilde ik de lijst van bundels dat een game nodig heeft om te werken, dynamisch verkrijgen op de frontend via die games microservice. Deze stap wilde ik als vierde, want dan weet ik al hoe ik data van Azure kan halen via de backend door de vorige stappen en die kennis zou mogelijk helpen bij het maken van een lijst van alle bundels van een game.

Als vijfde stap wilde ik voor alle code testen schrijven.

Als laatste stap wilde ik de code reviewen voor eventuele verbeteringen en opschoning van redundante code.

11.2 Bundels ophalen zonder Azure of backend

Het implementeren van deze stap ging goed en vrijwel helemaal volgens ontwerp. De paar veranderingen ten opzichte van het ontwerp, waren alleen maar zaken zoals bijvoorbeeld andere namen geven en meer of minder parameters geven aan functies.

Na wat debuggen werkte deze stap van implementeren zoals gewenst en was de game speelbaar in de browser.

11.3 Azure storage configureren

Voordat ik de bundels op kon halen van Azure storage moest ik eerst Azure storage configureren en de bundels in opslaan. Ik wilde voor elke game op de storage een eigen container aanmaken waar al zijn bestanden in zitten. In elke game container wilde ik dan een folder maken met zijn bundels om online de game te laden, een folder waar in de toekomst alle bestanden in kunnen staan die nodig zijn om de game te downloaden en een folder waar afbeeldingen in komen te staan, waaronder het logo van de game.

Deze laatste stap lukte alleen niet, want in een container van Azure storage is het wel mogelijk om folders te maken, maar dat is alleen voor de gebruiker te zien. Dit kwam, omdat een container op de achtergrond de folders niet ziet en zich gedraagt alsof alle bestanden van de container direct in de container zitten zonder folders (How to download blob container/Directory from azure storage to local folder?, 2022).

Om deze redenen had ik voor elke game één container aangemaakt met zijn bundels er in met als container naam, de naam van de game. Elke game heeft ook een core bundel nodig om te werken, dus had ik ook een container aangemaakt voor de core bundel.

Op dat moment had elke game een eigen container waar zijn bundels en logo in staan opgeslagen. Als in de toekomst het downloaden van games mogelijk wordt, is het mogelijk om dan voor die download versie van de game een eigen container te maken. Dan zou je ook een container kunnen maken voor overige bestanden voor de game die de download versie en de online versie nodig hebben (bijvoorbeeld de afbeeldingen van die game).

Een betere oplossing zou dan zijn om gebruik te maken van Azure Databricks, want dan kan je wel met een folder structuur te werk gaan. Zoals eerder vernoemd (zie hoofdstuk 9 Sprint 1 – Activiteiten systeem en inloggen) zijn er voor nu nog te weinig credits op Azure maar als dit project verder ontwikkeld zal worden naar productie krijgt het mogelijk meer credits voor Azure van CGI.

Momenteel had elke game dus maar één container met al zijn bundels.

11.4 Bundels ophalen via game naam

Tijdens het configureren kwam ik erachter dat ik de bundels van Azure storage kon ophalen, door middel van een URL naar de storage met de naam van de container en de naam van de bundel (bijvoorbeeld ‘https://platform.blob.core.windows.net/conveyor-sorter-bundels/conveyor-sorter-scene’). Ik dacht eerst dat de URL’s naar Azure storage ingewikkelde URL’s waren die niet dynamisch aan te maken zijn.

Om de game op te halen van Azure Storage, had ik dus de URL naar de bundel locatie van een game uit de database niet meer nodig, aangezien de applicatie het pad naar de bundels van een game af kan leiden van de naam van de game.

Deze manier van bundels ophalen was beter dan het via de paden van de database te doen, omdat CGI de paden in de database niet meer hoeft in te vullen en bij te houden, waardoor het platform dan beter te onderhouden is. Deze manier zorgt er voor dat er geen extra aanvraag nodig is, om de paden naar de games op te halen, waardoor dat tijd bespaart met het laden en de games dus sneller laden voor de gebruikers.

Om de bovenstaande redenen besloot ik de bundels op te halen van Azure storage, door het pad naar de bundels te maken met de game naam, in plaats van het pad naar de bundels op te halen uit de database.

11.5 Bundels ophalen van Azure zonder backend

Tijdens het implementeren van deze stap kwam ik er achter, dat er toch wel een verschil zit tussen de data dat de webapplicatie krijgt bij het ophalen van een bundel van zijn eigen server en van Azure storage.

Ik kreeg een error in de browser bij het laden van een game dat niet van de frontend of backend af kwam, waardoor ik vermoedde dat het van de game afkwam. Na gevraagd te hebben aan mijn begeleider wat die error inhield bleek het dus dat die error betekende dat de game data kreeg dat hij niet kan converteren

naar een bundel. Na veel debuggen kwam ik zelf niet achter de oorzaak van de error waarbij ik elke mogelijk oorzaak van de error gecheckt had, dat het daar niet aan lag.

Ik vroeg mijn begeleider voor hulp aangezien de enige plek waarbij het fout kon gaan is in de data dat het naar de game stuurt en mijn begeleider wist hoe die data eruit moest zien. Met zijn hulp kwamen we er al snel achter dat bij het ophalen van de data van Azure storage er een aantal karakters extra worden toegevoegd aan de data waardoor de game de data niet meer kon lezen.

De oorzaak van deze toevoeging van karakters was dat het de bundel met een iets andere encoding ontving van Azure dan wanneer het de bundel ophaalt van zijn eigen server. Bij het ontvangen van de bundel staat de encoding in de data vooraan voordat de daadwerkelijke data neergezet wordt.

Na die extra karakters weggehaald te hebben werkte de game waarbij de bundels van Azure storage kwamen.

11.6 Bundels ophalen van Azure via de backend

Bij deze stap kon het meeste mis gaan. Dit kwam doordat je nu een ketting van berichten en aanvragen hebt in plaats van een directe aanvraag naar Azure storage. De frontend applicatie moet dan een http aanvraag doen naar de API applicatie voor een bundel.

Dan moet de API applicatie een bericht sturen via TCP protocol (dat is waarmee de API en microservice applicaties met elkaar praten) naar de games microservice, om de bundel op te halen. De games microservice moet dan de bundel ophalen van Azure storage met een http aanvraag.

Tussen de API en de games microservice zal er waarschijnlijk geen problemen ontstaan, want die applicaties kunnen met elkaar communiceren zonder problemen met TCP. Waar er hoogstwaarschijnlijk wel wat problemen zullen ontstaan is bij de http requests.

Problemen konden ontstaan, omdat er twee http aanvragen plaatsvinden, waarbij het goed mogelijk is dat de data die de frontend uiteindelijk ontvangt, anders is dan de data die het direct zou ontvangen van Azure storage.

Tijdens het implementeren liep ik zoals ik al verwacht had tegen problemen aan met de http aanvragen.

11.7 Bundels ophalen van games microservice probleem

Als eerste ging het fout bij het ophalen van de bundel op de games microservice applicatie. In de bundel data dat de microservice ontving van Azure storage, zaten veel ontbrekende karakters. Na wat onderzoek op internet en debuggen kwam ik erachter dat zo een dergelijk probleem voor kan komen wanneer encoding van de aanvraag niet goed is.

De methode die ik gebruikte op de microservice om bundels op te halen was door gebruik te maken van een package genaamd Axios dat de NestJS website adviseerde te gebruiken voor aanvragen.

De bundels stonden opgeslagen op Azure storage met als formaat ‘application/octet-stream’ (volgens Azure). Ondanks dat ik bij de aanvraag voor een bundel de header ‘Content-Type’ naar ‘application/octet-stream’ had gezet, kreeg ik nog steeds de data binnen met missende karakters.

Na vele manieren geprobeerd te hebben om dit probleem op te lossen (onder andere headers veranderen, encoding aanpassen en in JSON formaat ophalen) lukte het niet om het probleem op te lossen.

Omdat het niet lukte om met het de Axios package de bundel te downloaden, ging ik zoeken naar een andere methode om bestanden te downloaden van Azure storage, waarbij ik al snel een package vond genaamd ‘@azure/storage-blob’ (NestJS API File Operations Using Azure Blob Storage, 2022).

Met het gebruik van de tools van die package lukte het om de bundel correct op te halen van Azure storage op de microservice. Bij de package had ik een selectie van variabelen die ik kon zetten, waar de package headers van maakt in plaats dat ik zelf de header moet schrijven (met Axios moest ik de headers zelf schrijven). Bij deze selectie begonnen alle variabelen met het woord blob waaronder

blobContentEncoding. Door deze variabele te gebruiken lukte het om de bundel succesvol te downloaden van Azure storage.

Bij het terugsturen van de bundel data naar de frontend kreeg ik een probleem op de frontend.

11.8 Ontvangen bundel format probleem

Bij het ontvangen van de data op de frontend van de backend kreeg ik hetzelfde probleem dat ik op de backend kreeg waar er veel karakters ontbreken in de ontvangen data. Hier was er alleen een andere oorzaak van het probleem.

Deze keer was de oorzaak van het probleem, dat de aanvraag die de frontend applicatie stuurde naar de API applicatie, nog gebaseerd was op het direct ophalen van Azure storage. Na wat aanpassingen van de aanvraag was het gelukt om de bundel data van de backend correct te ontvangen.

11.9 Lijst van bundels krijgen van Azure via backend

Op dit moment stuurde ik nog de lijst met bundels die verkregen moest worden hardcoded naar de game, wanneer de game via een event er om vroeg.

Deze lijst aanmaken op de backend en ophalen van de backend op frontend, kon ik zonder problemen implementeren op zowel de frontend als de backend. Op de games microservice gebruikte ik weer de @azure/storage-blob package, om de lijst van bundels van voor die game samen met de core bundels samen te stellen. Deze lijst stelde de backend dan samen door van alle core bundels en alle bundels een lijst te maken.

11.10 Clean-up en review

Bij de review van de code had ik een aantal stukken refactored en had ik op de database wat kolommen weggehaald.

11.10.1 Refactor

Bij de klasse dat de bundels download, had ik wat onnodige conversies uit de code kunnen halen, omdat ik nu van de microservice de bundel krijg in het exacte formaat dat het nodig heeft, in plaats het formaat dat het zou krijgen van Azure storage (toen kreeg het de data in blob formaat en nu in tekst formaat).

Bij de rest was er op de frontend geen refactor meer nodig. Bij de games controller op de API was er nog wel wat refactor nodig om de games controller, activiteiten controller en de users controller allemaal dezelfde structuur te geven (de API applicatie heeft voor elke microservice een controller en een service dat alle aanvragen voor die microservice, naar die microservice stuurt).

11.10.2 Database

Op de database had ik wel twee kolommen weggehaald bij de games tabel. Hoe ik nu de bundels ophaal is dynamisch op basis van de naam van de game en heeft de paden in de database niet meer nodig.

Deze manier van bundels ophalen is beter dan het via de paden van de database te doen, omdat je niet de paden in de database meer hoeft in te vullen en bij te houden. Deze manier zorgt er ook voor dat er geen extra aanvraag nodig is om de paden naar de games op te halen waardoor dat tijd scheelt met het laden en de games dus sneller laden. Om deze redenen had ik op de database bij de tabel games de kolommen browser_path en download_path weggehaald.

11.11 Testen

Ondanks dat ik verwacht had dat het testen moeilijk zou worden vanwege de game instantie, verliep het maken van de testen zeer goed. De game instantie bleek namelijk bij het testen geen problemen te veroorzaken.

Het testen van de games microservice applicatie ging verassend makkelijk. Om de testen te maken voor het ophalen van de bundel van Azure moest ik wel van drie klassen van de @azure/storage-blob package die ik gebruik mocken. Naast dat ik op de microservice die mocks moest maken, verliep het maken van de testen naar wens.

Bij alle microservices, de API en de frontend applicaties had ik allemaal 100% code coverage. Voor de API applicatie had ik ook end-to-end testen gemaakt.

11.12 Retrospectief

Het implementeren van het games systeem ging verassend goed. Ondanks een paar obstakels met de bundel van Azure storage halen en veel moeten debuggen ging het implementeren goed. De testen maken verliep ook zonder obstakels.

De gekozen aanpak om het games systeem te implementeren was een goede keuze geweest. Op deze wijze kon ik zoals verwacht snel problemen identificeren en oplossen.

11.12.1 Ontwerp kritiek

Ik had iets kritischer moeten zijn bij het maken van het ontwerp van het games systeem bij het geven van de namen van de klassen en de functies. Op het ontwerp zou het dan duidelijker geweest zijn wat elke klasse en elke functie precies doet, want tijdens het implementeren had ik meerdere klasse namen veranderd naar een naam die beter beschrijft wat die klasse inhoud.

11.12.2 Keuze package voor bundel ophalen

Het was mogelijk dat het ophalen van de bundel ook mogelijk was met het Axios package, als ik daar de header dat de @azure/storage-blob maakt van de variabele blobContentEncoding, had gebruikt in plaats van de header ‘Content-Type’. Echter was er een kans dat Axios deze header niet accepteert en de aanvraag weigert.

Ondanks dat de mogelijkheid bestond dat de bundel ophalen kon werken met Axios, besloot ik toch de @azure/storage-blob package te gebruiken. Bij de @azure/storage-blob package is de code leesbaarder, netter en meer type safe. De @azure/storage-blob package maakt eerst verbinding met de container waar de bestanden in staan, vervolgens maakt het verbinding met het bestand in die container, om dan het bestand te downloaden. Ook was het instellen van een header type safe, omdat het je een selectie aanbied van headers, in plaats van dat ik het zelf volledig moet typen, waar makkelijker fouten ontstaan, dan als je een header selecteert.

11.12.3 Karma en Jasmine of Jest

Nu dat het games systeem was geïmplementeerd, kon ik overwegen om van Karma en Jasmine over te stappen naar Jest. Na wat informatie opgezocht te hebben over het overstappen van Jasmine en Karma naar Jest, was het wel mogelijk om deze overstap te maken (Testing Angular faster with Jest, 2022). Bij deze overstap zouden er wel een paar nadelen zijn, met onder andere dat er weinig documentatie is voor het gebruik van Jest met Angular (Why we chose Jasmine over Jest and Mocha, 2022).

Ik besloot deze overstap niet te maken. Dit besloot ik omdat de testen in de huidige situatie met Jasmine en Karma de testen binnen een paar seconden allemaal gedaan zijn, terwijl de testen op alle backend applicaties (die Jest gebruiken) meer dan vijftien seconden lang duren. In de huidige situatie zijn ook de extra methodes van Jest, zoals snapshot testen, niet nodig.

Als er in de toekomst een goede reden zou zijn om over te stappen naar Jest is dat te overwegen, maar in de huidige situatie was er geen aanleiding om deze overstap te maken.

12 Sprint 4 – Dashboard en privileges

Bij deze sprint waren er 3 user stories ingepland. Ik had er voor gekozen om in deze sprint weer volledige user stories in te plannen, aangezien er voor deze user stories geen onderzoeken nodig waren. Tijdens deze sprint had ik een paar dagen vakantie genomen waardoor er iets minder ingepland was. Tevens wilde ik tijd inruimen om dit afstudeerdossier te maken.

Dit was mijn laatste sprint, waardoor mijn begeleider in ik een oplevering gesprek hadden ingepland. In dit gesprek liet ik de werking van het platform zien, legde ik alle code van alle applicaties uit en liet ik zien hoe Azure werkte. Tijdens dit gesprek gaf ik hem ook alle scripts die ik had gemaakt, waarbij ik uitlegde wat elk script deed (over verloop van tijd waren er meer scripts bijgekomen).

Epic	User Story	Schatting effort points
	Als gebruiker wil ik een game kunnen spelen waar ik de privileges voor heb om die te mogen spelen.	16
	Als gebruiker wil ik een dashboard hebben waar ik alle games op kan zien waar geen privileges voor nodig zijn en de games waar ik privileges voor heb, zodat ik een overzicht heb van alle games die ik kan spelen	34
	Als gebruiker wil ik een bericht zien wanneer een error voor komt op de website.	8

Figuur 82: User stories sprint 4

De user story “Als gebruiker wil ik een bericht zien wanneer een error voor komt op de website” was in verloop van het project toegevoegd aan de backlog. Deze user story had ik toegevoegd aan de backlog, nadat ik de suggestie voorgesteld had aan mijn opdrachtgevers.

Ik had dit voorgesteld, zodat de website duidelijk aan gebruikers kan laten zien als er iets fout gaat op de website of de backend. Als er bijvoorbeeld een error voor komt op de website hoeft een gebruiker dan niet nodeloos te wachten, totdat de website werkt.

Deze berichtgeving toevoegen zal de website een stuk gebruiksvriendelijker maken en kan helpen bij het ontwikkelen van de applicatie.

12.1 Aanpak user stories

De eerste user story die ik besloot aan te pakken, was de user story over de berichtgeving van errors. Ik wilde deze user story als eerste implementeren, zodat ik de berichtgeving dan gelijk kon meenemen bij het ontwikkelen van de andere twee user stories.

De privileges user story besloot ik als tweede aan te pakken. Je moet eerst privileges kunnen checken, voordat je op de dashboard de games kan laten zien waar de gebruiker privileges voor heeft.

De games weergeven op het dashboard van gebruikers besloot ik als laatste user story te doen.

Als laatste stap (net als bij de andere sprints) wilde ik een clean-up doen en de code reviewen.

De user stories van deze sprint zijn meer losse individuele onderdelen als toevoeging op een bestaand systeem, dan dat het de basis vormt van het platform, zoals bij de user stories van de vorige sprints. Bij deze user stories was het dus mogelijk om de user stories één voor één te implementeren, in plaats van tegelijk zoals bij sprint één. De user stories van deze sprint waren ook klein genoeg dat het niet nodig was er taken aan te geven.

Ik kon ook de user stories ook op een andere volgorde implementeren, maar de volgorde dat in de bovenstaande alinea’s is vernoemd, leek mij de meest efficiënte volgorde.

Ik kon bijvoorbeeld eerst de game items implementeren, dan het privilege systeem en dan de berichtgeving. Echter moet ik dan wel de privileges toevoegen op de games op het dashboard en de

berichtgeving toevoegen aan de aanvragen die het privilege systeem en de games op het dashboard sturen.

12.2 Error berichten systeem

Om de berichtgeving te implementeren besloot ik de snackbar component van Angular Material te gebruiken (zie 9.8.2 Angular Material en validatie voor meer informatie over Angular Material). Wat dit component kan doen, is visueel een box laten zien met een bericht erop.

Ik wilde de mogelijkheid inbouwen dat je ook een waarschuwing of een bevestiging kan sturen als bericht. Dit zou bijvoorbeeld handig zijn, om te waarschuwen als een gebruiker ergens geen privilege voor heeft of om te bevestigen dat een account succesvol is aangemaakt. Ook een reden om de snackbar van Angular Material te gebruiken, was dat ik al op andere plekken in het project componenten van Angular Material gebruikte, waardoor ik geen nieuwe packages hoeft te installeren.

Het implementeren van de snackbar bestond uit twee onderdelen, één component dat het bericht zijn inhoud en uiterlijk geeft en één injecteerbare service dat het component kan oproepen om een bericht weer te geven.

Als component besloot ik een eigen component aan te maken voor de uiterlijk van het bericht. Dit wilde ik doen, omdat je aan de basis snackbar geen iconen kan toevoegen. Ik wilde er wel iconen aan toevoegen, om beter aan te geven wat voor type bericht het is (bijvoorbeeld een error icoon als het bericht voor een error is). Deze iconen komen ook van Angular Material af (Icon, 2022).

Bij de service had ik drie functies gemaakt voor elk van de type berichten (error, waarschuwing en bevestiging). Ik deed dit zo, omdat je dan aan zo een functie alleen het bericht hoeft mee te geven en het de rest van de variabelen al heeft, om het goede type bericht weer te geven.

Om een beeld te geven hoe zulke berichten eruit zagen voor de gebruiker, zijn de verschillende berichten te zien in Figuur 9: Berichten (links error, midden waarschuwing en rechts bevestiging).

Figuur 9: Berichten

Dit berichten systeem had ik vervolgens toegepast bij elke aanvraag naar de backend. Als een aanvraag faalt, dan wordt een error bericht weergegeven met als tekst waar de fout is opgetreden. Op het moment dat er een aanvraag faalt, ziet de gebruiker dan een error bericht en kan de gebruiker dan gelijk actie ondernemen (bijvoorbeeld website herladen), in plaats van onnodig blijven te wachten.

12.3 Privilege systeem

Voordat ik kon beginnen met het privilege systeem implementeren, moest eerst weten hoe ik dit wilde doen op zowel de frontend als de backend.

12.3.1 Privilege systeem backend

Ik moest een afweging maken, waar de aanvragen naar de database voor privileges vandaan kwamen. Hierbij was de afweging of ze van de users microservice vandaan komen of van een nieuwe eigen privileges microservice.

Ik had besloten voor de privileges een privileges microservice aan te maken. Dit had ik besloten, omdat het idee is van microservices dat elke microservice zijn eigen verantwoordelijkheden heeft en de users microservice al verantwoordelijk is voor de gebruikers (5 design principles for microservices, 2022). Deze microservice zal dan verantwoordelijk zijn over alle privileges en rollen tabellen in de database.

Net zo als voor de andere microservices, maakte ik op de API voor de privilege microservice, een controller en een provider aan (zie hoofdstuk 9.4 Ontwerp backend waarom ik dit deed).

12.3.2 Frontend implementatie onderzoek

Bij het onderzoeken naar manieren om het privilege systeem te implementeren, hield ik een aantal punten in gedachten.

- Aan de frontend zou je bijvoorbeeld privileges nodig kunnen hebben om de knop voor je account instellingen te zien of om bepaalde games te zien op je dashboard. Een privilege kan dus gelden voor allerlei soorten html elementen (zoals onder andere buttons en containers).
- Elke pagina kan een andere hoeveelheid onderdelen hebben die privileges nodig hebben. Voor elk van die elementen moet de privilege gecheckt worden.
- Wat ik wilde voorkomen is dat je in een component dat elementen bevat met privileges je iets van een lijst hebt met alle elementen waarvoor de privileges gecheckt moeten worden. Ik wilde dus dat elke element zelf kan checken of de gebruiker zijn privilege heeft.

Na overwogen en onderzocht te hebben hoe ik dit het beste kon implementeren, besloot ik om een component te gebruiken. Een component kan je in andere componenten gebruiken zo vaak als je wilt. Ik zou het component de code kunnen geven, om de privilege aan te vragen aan de backend en de elementen die in dat component staan, wel of niet te laten zien, op basis van het resultaat van die aanvraag.

Een component volgens de bovenstaande manier implementeren, zou aan alle wensen voldoen waarbij het component om elk html element heen kan staan. Een component dat privileges heeft kan zo veel privilege componenten hebben als het nodig heeft en het component heeft dan geen extra code nodig voor de privileges af te handelen, want dat staat in het privilege component.

Op deze manier hoefde het component dat privileges checkt, niks te weten van de privileges. Elk privilege component checkt dan zelf of de gebruiker zijn meegegeven privilege heeft. Deze wijze is zeer object georiënteerd volgens het single-responsibility principe van OOD, omdat de componenten op deze wijze niet verantwoordelijk zijn voor de privileges, maar de privileges verantwoordelijk zijn voor zichzelf.

Om een beeld te geven hoe het privilege component zit in de html code van andere componenten, is een voorbeeld te zien op Figuur 10: Voorbeeld privilege component.

```
<privilege *ngIf="hasPrivilege===true" [code]="app-settings">
    <button class="settings">Settings</button>
</privilege>
```

Figuur 10: Voorbeeld privilege component

12.3.3 Implementatie manier vervolgonderzoek

Na het implementeren van het privilege component, was ik nog niet tevreden, ondanks dat het component werkte. Om met dit component een privilege toe te voegen, moet ik nog wel het privilege component om de elementen in de html zetten, waar een privilege moet worden gecheckt.

Om elke element dat een privilege nodig heeft, het privilege component zetten, gaf code duplicatie. Dit kwam, doordat elk van die privilege componenten dezelfde *ngIf attribuut nodig heeft, om de elementen onder het component te laten zien, wanneer de gebruiker die privilege heeft. Indien er veel privileges op een pagina moeten worden gecheckt, wordt de html code ook slechter, omdat er dan veel code duplicatie is, doordat er voor elke privilege een extra html element in staat, in de vorm van het privilege component.

Daarnaast bevatte het privilege component nu een eigen CSS en html bestand voor hoe het component eruit ziet (dit is om aan te geven dat het elementen kan bevatten). Dit wilde ik ook liever niet hebben,

omdat het geen template nodig zou moeten hebben, omdat het zelf geen elementen bevat en het alleen het gedrag van de elementen, die onder dat component zitten, wilt aanpassen.

Ik besloot dus te onderzoeken of het mogelijk was componenten te hebben zonder template en zonder code duplicatie.

Al snel vond ik een oplossing in de vorm van een directive (Architechture components, 2022). Met directives kan je het gedrag en uiterlijk van html elementen aanpassen.

Structural directives kunnen elementen toevoegen, verwijderen of vervangen (Built-in directives, 2022). De *ngIf van regel één van Figuur 10: Voorbeeld privilege component is een voorbeeld van een structural directive, dat Angular standaard bevat. De * voor de attribuut geeft aan dat het een structural directive is. Bij het geval van de *ngIf verbergt de structural directive het element, wanneer de variabele die is meegegeven aan de *ngIf false is.

Een attribuut directive kan het gedrag of uiterlijk van een bestaand element aanpassen (Built-in directives, 2022).

Componenten zijn technisch gezien directives, maar componenten zijn verschillend genoeg van directives dat de ontwikkelaars van Angular het definiëren als componenten (Architechture components directives, 2022).

Een attribuut directive gebruiken voor de privileges was de beste oplossing, omdat een privilege het gedrag van een bestaand element moet aanpassen op basis of het de privilege heeft of niet.

Ik besloot met deze nieuwe informatie het component te refactoren naar een attribuut directive, waar er maar een paar wijzigingen nodig waren, om het component te veranderen naar een attribuut directive.

Om een beeld te geven hoe het privilege systeem werkt met een attribuut directive is hieronder een voorbeeld (Figuur 11: Voorbeeld privilege directive) te zien, hoe dezelfde button als het vorige voorbeeld (Figuur 10: Voorbeeld privilege component) eruit ziet met de privilege als attribuut directive:

```
1  <button privilege="app-settings" class="settings">Settings</button>
```

Figuur 11: Voorbeeld privilege directive

Zoals te zien is in Figuur 11: Voorbeeld privilege directive, hoefde alleen het privilege attribuut met privilege aan een element worden meegegeven, om voor een element een privilege te checken. Door het privilege systeem als directive kreeg ik dus niet meer de code duplicatie die ik kreeg bij het privilege systeem als component.

De privilege als een attribuut directive, in plaats van een component werkte volledig naar wens en was ik zeer tevreden mee. Het component refactoren naar een attribuut directive was een goed besluit geweest.

12.4 Elementen referenties opslaan directive

In hoofdstuk 9.10 had ik benoemd, dat als ik later een betere oplossing vond voor het opslaan van elementen referenties, dat ik die manier toe zou passen. Om die elementen referenties op te slaan zou een attribuut directive een veel betere manier zijn, dan de oplossing dat ik had toegepast in sprint 1.

Met een attribuut directive zou ik alleen maar aan een element, een attribuut moeten meegeven, om de referenties naar zijn element op te slaan. Elk element met die directive slaat zichzelf dan op in de klasse dat de element referenties bijhoudt. De componenten die elementen referenties bevatten voor taak uitvoerders, zouden op die manier geen code meer nodig hebben om elementen referenties op te slaan. Die componenten zijn dan ook veel meer object georiënteerd volgens het single-responsibility principle van OOD, want die componenten hoeven dan niet meer de van elementen referenties op te slaan.

Aangezien het refactoren van het opslaan van elementen in componenten, naar het gebruiken van een attribuut directive weinig tijd zou gaan kosten, ging ik de element referentie attribuut directive gelijk implementeren.

Het implementeren hiervan ging zoals verwacht heel snel (binnen een half uur gedaan) en volledig naar wens. Ik was zeer tevreden over hoe de elementen nu werden verzameld. Een element referentie opslaan werkte dan op zo een zelfde manier als in Figuur 11: Voorbeeld privilege , met in plaats van de code van de privilege, krijgt de attribuut directive de naam van de referentie mee.

12.5 Dashboard games weergeven

Op het dashboard zou van elke game die de gebruiker mag spelen de naam, versie, beschrijving en logo te zien zijn. De code en de styling voor al die games kon ik zetten bij het dashboard component. Echter zou het dashboard component dan veel code bevatten dat alleen nodig is voor het weergeven van een game. Daarnaast moet er in de html van het dashboard component ook veel elementen worden toegevoegd, om alle games weer te geven met bijhorende data. Het CSS bestand van het component wordt dan een mix worden van zijn eigen styling en de styling voor de weergave van de games.

Om deze weergave van games te implementeren besloot ik dus een game list item component aan te maken, dat alles bevat dat een game op het dashboard nodig heeft. Op die manier hoefde het dashboard component niks te weten van de game list items en hoefde het alleen maar een lijst te maken van games, om game list items van te maken. Het dashboard component en het game list item component voldeden dan aan het single-responsibility principle van OOD, omdat beide componenten dan hun eigen verantwoordelijkheden hebben.

12.5.1 Plan voor implementatie

Het game list item component moest de volgende functionaliteit bevatten:

- Elk item heeft een logo dat opgehaald moet worden van de backend
- Elk item kan zijn eigen game starten door op dat item te klikken
- Elk item heeft een privilege waarbij gekeken moet worden of de gebruiker de privilege heeft om die game te zien.
- Elk item laat van zijn game de naam, versie en beschrijving zien.

Om dit allemaal te implementeren maakte ik eerst een plan hoe ik dit wil doen.

De eerste stap was het component aanmaken en vormgeven (met html en CSS), zodat het er goed uit ziet en het de naam, versie, beschrijving en logo kan weergeven.

Als tweede wilde ik de privilege en de functie om een game te starten implementeren. Dit wilde ik als tweede doen, want die functionaliteit was al van geïmplementeerd.

Als derde stap wilde ik het logo ophalen en weergeven. Dit wilde ik nu pas doen aangezien dit nieuwe functionaliteit zou toevoegen aan het component. Het logo wilde ik op laten halen van Azure storage via de games microservice, omdat die microservice verantwoordelijk was over de games.

12.5.2 Game list item component maken

Voordat ik aan het uiterlijk van het component begon, wilde ik eerst inspiratie op doen. Ik ging op internet zoeken naar platformen voor games, om te zien hoe die websites hun games grafisch weergeven.

Na inspiratie te hebben opgedaan, ging ik het component maken.

12.5.3 Privilege en game starten toevoegen

Om een game te starten gebruikte ik de taak uitvoerder dat naar andere pagina's kan gaan (zie hoofdstuk 9.7 Implementatie deel 2). Aan die taak uitvoerder werd de naam van de game meegegeven, zodat de game

pagina) de naam van de geselecteerde game kon krijgen (via route parameters, zie hoofdstuk 10.7 Ontwerp maken voor meer over route parameters).

Na het toevoegen van de privilege liep ik wel tegen een probleem aan waarvoor ik een keuze moest maken.

12.5.3.1 Component verbergen probleem

Hoe de privilege directive werkte, was dat wanneer een element zijn privilege niet had, het element werd verborgen.

Het dashboard component maakte voor elke game dat hij van de backend kreeg, een game list item component aan. Bij het renderen van een component rendeert Angular niet alleen de elementen in het component, maar rendeert het ook een element voor het component zelf.

Wat er dus gebeurde op het dashboard, wanneer de gebruiker de privilege voor een game niet had, was dat het element van het game list item component wel getoond werd, maar het liet alle elementen die in dat component zitten niet zien. Het resultaat was dat er een lege ruimte ontstond op het dashboard, waar het game list item component element dan nog staat, maar dan zonder de elementen van het component.

Tijdens het onderzoeken naar de beste oplossing om dit probleem op te lossen vielen een aantal oplossingen die ik vond af. Die vielen af, omdat die oplossingen meerdere extra regels code zouden toevoegen aan het game list item component en bij sommige van die oplossingen zouden er zelfs extra dependenties bij komen. Na het onderzoek had ik gekozen voor de oplossing die maar één regel code nodig had.

Bij deze oplossing hoefde ik maar één regel CSS code toe te voegen aan het game list item component, om het component element te verbergen. Wat die regel deed, was dat het alleen maar de elementen in het component weergeeft en niet meer het element van het component zelf.

Ik was zeer tevreden met deze oplossing, want hiervoor hoefde ik maar één regel CSS toe te voegen, om het component naar wens te laten werken.

12.5.4 Logo ophalen

Voordat ik kon beginnen met implementeren van het logo, moest ik eerst overwegen waar ik het logo van elke game op wilde slaan.

12.5.4.1 Keuze opslaan van het logo

Er waren drie opties om te overwegen:

1. Het logo opslaan van Azure storage net zo als de bundels, door het pad naar het logo op Azure storage dynamisch te maken door de naam van de game te gebruiken.
2. Het logo opslaan als blob op de database in de games tabel (zie hoofdstuk 8.2.2.1 Blob opslaan in database hoe dat werkt). Als het dan de game data (naam, versie, beschrijving, etc.) ophaalt van de database heeft het ook gelijk het logo.
3. Het logo opslaan op Azure storage en het pad daar naar toe opslaan in de database bij de tabel games (zie hoofdstuk 8.2.2.2 File system voor meer informatie over deze optie).

Na overweging had ik besloten voor optie 1 te gaan. Dit besloot ik, omdat de bundels al op Azure storage stonden, waardoor ik een al bestaande manier kon gebruiken om het logo van Azure af te halen. Een andere reden om voor deze optie te kiezen, is dat de bundels al op deze wijze worden opgehaald en het inconsistent zou zijn, om de bundels en het logo op verschillende manieren op te halen.

De tweede optie viel af, want dat zou te veel ruimte innemen op de database en al helemaal als er vele games op zou het platform komen te staan.

De derde optie viel af, omdat het logo dan bijna dubbel wordt opgeslagen. Het logo wordt dan opgeslagen op Azure storage en het pad naar dat logo toe in de database. Om het logo op te halen moet dan eerst het

pad worden opgehaald, om daarna met dat pad het logo ophalen. Bij deze optie zal het op de frontend ook langer duren voordat de gebruiker het logo te zien krijgt, omdat de frontend dan een extra aanvraag moet doen waar de frontend dan op moet wachten.

Om deze redenen heb ik dus voor optie 1 gekozen.

12.5.4.2 Implementeren

Het logo wilde ik ophalen van Azure storage op de games microservice. Dit wilde ik hier doen, omdat deze microservice verantwoordelijk was voor de games, als enige in verbinding stond met Azure storage en de @azure/storage-blob package had (zie hoofdstuk 11.7 Bundels ophalen van games microservice probleem over informatie over die package).

Het implementeren van het ophalen van het logo de backend verliep zonder problemen. Op de frontend verliep het implementeren wat moeizaam, om met de data van de backend een afbeelding te laten zien, maar dat was uiteindelijk gelukt.

12.6 Testen

Om de privilege directive en de element verzamel directive te testen, maakte ik voor elke directive in de test klasse van die directive test componenten aan, dat elementen bevatten met die attribuut directive.

Het testen van alle code op zowel de frontend, de API en de microservice verliep naar wens en zonder problemen. Ik had 100% code coverage behaald op alle applicaties, waarbij ik op de API ook end-to-end testen gemaakt.

12.7 Clean-up en review

Nadat ik de code had gereviewd, was er maar één stuk code dat ik aangepast had dat ik nog even wil benoemen.

Ik had de privilege directive veranderd naar een structural directive van een attribuut directive. De functionaliteit van de directive bleef hierdoor hetzelfde, maar de kwaliteit van de code van de privilege directive was met deze wijziging wel verbeterd. In plaats van het element verbergen, werd het element hierdoor aangemaakt of weggehaald, waardoor er mogelijk minder geladen moet worden, doordat het dan mogelijk minder elementen moet laden.

12.8 Retrospectief

Ik was zeer tevreden hoe deze sprint was verlopen. Het implementeren van elke user story verliep op zowel de frontend, API en games microservice applicatie naar wens. Er zijn wel een paar punten waar ik nog wel wat over wil zeggen.

12.8.1 Privilege systeem

Het was een goed besluit om vervolgonderzoek te doen op de werking van het privilege systeem. Ik was zeer tevreden hoe dat systeem door middel van een directive werkte. De werking van het privilege systeem door middel van een privilege directive, was precies hoe ik de werking van dit systeem wenste.

12.8.2 Element verzamelen directive

Ondanks dat ik zeer tevreden was hoe het elementen verzamel systeem werkte met een directive, had ik deze refactor beter kunnen implementeren in de clean-up en review fase. Dit zou beter zijn geweest, omdat er een kans bestond hoe klein dan ook dat er iets fout zou zijn gegaan, waardoor het meer tijd zou vereisen. Als ik te veel tijd kwijt zou zijn aan het element referentie directive, was er een kans ik daardoor tijd te kort zou hebben om de laatste user story te ontwikkelen.

Het zou dus beter geweest zijn als ik die directive in de clean-up en review fase meenam en als ik geen tijd meer had gehad, dat ik het dan toe zou voegen aan de backlog, zodat het dan in een volgende sprint van het project meegenomen kan worden.

13 Reflectie

In dit hoofdstuk ga ik reflecteren op mijn handelen tijdens dit afstudeertraject.

Ik heb het tijdens het uitvoeren van de opdracht zeer naar mijn zin gehad. Met deze opdracht moest ik werken met een frontend, API, microservices, database, opslag, Cloud omgeving en games. Door deze opdracht ben ik er achter gekomen dat ik het leuker vind om op verschillende software niveaus te werken, dan maar op één niveau (leuker dan bijvoorbeeld alleen een frontend of alleen een backend maken). Ik heb tijdens dit afstudeertraject ook veel geleerd over onder andere een Azure Cloud omgeving opzetten en configureren, een backend ontwikkelen en hoe dat moet met NestJS, frontend en backend frameworks, http aanvragen, microservices, het ontwikkelen van een frontend met het Angular framework en hoe unity games werken in een browser.

Bij mijn vorige stage had ik een werkweek van 36 uur, waarbij ik elke halve woensdag vrij nam. Die uren nam ik aan vanwege mijn autisme, omdat ik de hoeveelheid prikkels die ik binnen zou krijgen met een 40 uur werkweek te veel zou zijn. Ondanks dat ik 36 uur in de week werkte, was ik bij het einde van dat stage traject zowel fysiek als mentaal volledig uitgeput, waarbij ik zelfs een aantal keer niet lekker werd van de uitputting. Tijdens deze stage deed ik ook niks anders dan werken, uitrusten en slapen.

Bij dit afstudeertraject had ik daarom gekozen om 32 uur in de week te werken in plaats van 36 uur, waarbij ik elke woensdag vrij nam. Het was een zeer goede keuze geweest, om voor 32 uur per week te kiezen. Tegen het einde van het afstudeertraject voelde ik me nog net zo goed als bij het begin, waarbij ik regelmatig zelfs nog energie had, om aan hobby's te besteden.

Corona had ook een grote rol gespeeld tijdens dit afstudeertraject. Ik werkte alleen maar thuis en was totaal maar vier keer naar het kantoor gegaan. De mogelijkheid was er wel om vaker naar het kantoor te gaan, maar de keren dat ik op kantoor was, waren er zeer weinig mensen aanwezig. Op mijn werkplek thuis, had ik ook betere apparatuur, wat ook een rol had gespeeld bij de hoeveelheid kantoor werkdagen.

Het continu thuiswerken in plaats van elke dag op kantoor werken zal ook wel een rol gespeeld hebben bij mijn gezondheid, naast de 32 uur per week werken in plaats van 36 uur. Echter denk ik toch wel dat 32 uur per week werken de grootste invloed heeft gehad op mijn gezondheid. Voor de toekomst zal ik dus zeker meenemen, om 32 uur per week te werken.

Ondanks de corona pandemie was de communicatie met andere collega's van CGI en mijn opdrachtgevers goed, door veel gebruik te maken van Microsoft Teams en e-mails. Tijdens het afstudeertraject waren er geen obstakels door slechte communicatie.

Het uitvoeren van mijn opdracht had ik goed aangepakt, waarbij ik alles eerst uitzocht of ontwierp, om een zo goed mogelijk resultaat te krijgen. Echter waren er wel een paar punten waar ik anders had moeten handelen.

De refactor van het opslaan van elementen referenties had ik ook anders moeten handelen (zie hoofdstuk 12.8.2 Element verzamelen directive). Dit zal ik ook meenemen voor de toekomst, waar ik dan zo een refactor mee zal nemen in de review aan het einde van de sprint of aan de backlog zal toevoegen. Dan is er geen risico meer dat ik door zo een refactor tijd de kort komt, om de user stories van de sprint te ontwikkelen.

Ik had tevens vaker moeten checken voor updates van de packages van alle applicaties. Ik checkte dat ongeveer één keer per maand, terwijl dat eigenlijk op zijn minst één keer per week moet worden gecheckt. Dit komt omdat updates invloed kunnen hebben op onder andere de performance, veiligheid en integriteit van de code. Het proces om die packages up te daten is ook veel langer als er één maand van updates moet worden gedaan, in plaats van één week. Deze ervaring wil ik meenemen naar de toekomst, waarbij ik dan vaker zal checken voor updates voor packages van projecten.

Bij sprint één, drie en vier, waar ik code ontwikkeld had, deed ik bij het einde van elk van die sprints een clean-up en review van de code. Ik was zeer tevreden dat ik die stap deed elke sprint, want doordoor heb ik

de code meerdere malen kunnen verbeteren. Voor toekomstige projecten wil ik dit ook meenemen, dat ik na een onderdeel of ontwikkelperiode (bij dit project met scrum is dat een sprint) de gemaakte code review, want dat zal de kwaliteit van die projecten verbeteren.

Het gebruik van Scrum en Jira verliep redelijk goed. Hoewel Scrum goed verliep, waren er wel wat complicaties mee.

Sprint één duurde vier weken in plaats van de afgesproken drie weken. Het zou beter zijn geweest als ik de sprint na drie weken eindigde, wanneer het ook had moeten eindigen, dan een extra week door te gaan. Hoewel de opdrachtgevers dit geen probleem vonden, was het niet professioneel van mij om op die manier te handelen. Naar de toekomst wil ik dus ook meenemen dat ik me beter aan afspraken moet houden, om problemen met opdrachtgevers te voorkomen en professioneler over te komen.

Eén user story was opgedeeld in taken over twee sprints, in plaats van twee losse user stories. Met Scrum heb je normaal gesproken een vaste lengte per sprint, waarbij in elke sprint een backlog item wordt gedaan. Als een item dan te groot is voor één sprint wordt het opgedeeld in kleinere onderdelen. Indien een user story nog niet klaar is wanner een sprint eindigt, wordt die user story meegenomen naar een volgende sprint.

De beslissing om de user story over twee weken te doen hoort dus normaal niet. Echter konden we ook niet de user story opdelen, want elk onderdeel van het games systeem was cruciaal. Wat wel een mogelijkheid zou zijn geweest, was om de user story te verdelen in één user story met ontwerp en onderzoek en één user story voor implementeren. Hoewel de regels van scrum niet waren verbroken door de user story te splitsen, was wel één regel van de Definition of Ready verbroken, ondanks dat mijn begeleider het eens was met die oplossing. Hier had ik mij dus ook beter aan de afspraken moeten houden, door de user story te verdelen over twee user stories, naast dat het ook de bedoeling is met Scrum om meerdere kleine user stories te hebben, in plaats van grote user stories.

De sprintplanning begin elke sprint verliep goed. De stand-ups verliepen ook goed en ik veel baat bij. Door de stand-ups kon ik mijn begeleider goed up-to-date houden met mijn voortgang met de opdracht en hoe het ging met het dossier. De stand-ups hebben veel geholpen, omdat ik daardoor onder andere geïnformeerd bleef over de stand van zaken van zoals licenties en Azure credits en ik om zijn mening over gemaakte keuzes en onderzoek resultaten kon vragen. Op een sociaal niveau waren de stand-ups ook fijn, omdat de stand-ups vrijwel de enige momenten waren, waar ik sociaal contact had met collega's, naast een paar momenten op kantoor. De stand-ups hebben mij dus veel geholpen om geïnformeerd te blijven, bijsturing te krijgen door feedback en sociaal contact te houden.

Door de retrospectief van elke sprint kon ik een terugblik werpen op hoe de sprint was verlopen, wat er goed was gegaan, wat er fout was gegaan en hoe die fouten in de toekomst kunnen worden voorkomen, of het volgens ontwerp is geïmplementeerd en de wijzigingen daaraan. Bij de retrospectief van sprint drie had ik ook tijd besteed om keuze te maken tussen Karma en Jasmine of Jest, om voor toekomstige sprints te bepalen of die overstap beter zou zijn voor de frontend, aangezien ik die keuze door het resultaat van die sprint kon maken (voor informatie hierover zie hoofdstuk 11.12.3 Karma en Jasmine of Jest). De retrospectieven hadden mijn dus geholpen de kwaliteit van zowel de sprints zelf, als de inhoud van de sprints te verbeteren.

De stand-ups en retrospectieven van Scrum hebben mij dus geholpen om dit afstudeertraject tot een succesvol einde te brengen.

Het werken met Jira was zeer lastig, maar er ontstonden geen problemen mee. Dit kwam voornamelijk omdat ik op meerdere locaties ingevulde waarden niet kon wijzigen of verwijderen (ik kon bijvoorbeeld geen taken van user stories verwijderen of besteedde tijd aan een user story wijzigen). Door deze onaangename ervaring zal ik voor toekomstige projecten, indien mogelijk, geen Jira gebruiken maar een andere tool, maar als ik er later weer mee moet werken, weet ik wel hoe ik er mee om moet gaan.

14 Evaluatie

In dit hoofdstuk evalueer ik mijn aanpak tijdens dit afstudeertraject en het eindproduct.

14.1 Gekozen aanpak

Bij het maken van het plan van aanpak was ik veel te optimistisch, over hoeveel tijd ik nodig zou hebben voor de opzet fase. In plaats van de geschatte twee weken duurde het zeven weken. Ik had zwaar onderschat hoe lang ik bezig zou zijn met het opzetten van Azure, de onderzoeken en de requirements verzamelen.

Wat beter zou zijn geweest, was om meer te oriënteren op de opdracht, voordat ik het plan van aanpak ging maken. Dan zou ik een beter beeld gehad hebben over wat mij te doen stond en ik daardoor wellicht een betere schatting had kunnen maken. De ontdekking dat ik het activiteiten systeem verkeerd had begrepen, was mogelijk al eerder boven water gekomen, als ik mij meer had georiënteerd op de opdracht.

De opzet en oriëntatie fase had ik wel goed aangepakt. Door er rekening mee te houden, dat de mogelijkheid bestond dat ik de licenties pas laat kon krijgen, kon ik tijdens de opzet fase werkzaamheden genoeg verschuiven, waardoor ik altijd werk te doen had en ik geen vertraging opliep.

Zoals vermeld in hoofdstuk 13 Reflectie, had ik de eerste sprint niet goed aangepakt en had die users stories mee moeten nemen naar een volgende sprint, in plaats van die sprint verlengen.

Doordat het bij de eerste sprint fout ging, kon ik wel de nodige stappen ondernemen, om te voorkomen dat het bij de tweede sprint weer fout zou gaan, door de user story op te delen in taken. Dit had ik wel weer goed aangepakt, aangezien de tweede en derde sprint volledig volgens plan verliepen.

De vierde sprint ging ook volgens plan, maar zoals ik had beschreven in hoofdstuk 13 Reflectie, had ik de elementen referentie refactor niet moeten doen midden in de sprint. In plaats daarvan had ik het bij het einde van de sprint moeten doen, indien ik tijd over zou hebben, of anders dat ik de refactor niet zou doen in die sprint en op de backlog een issue aanmaak voor de refactor van element referenties, voor een volgende sprint.

14.2 Eindproduct

Ik ben zeer tevreden over het eindproduct. Het eindproduct bevat hoge kwaliteit code, is gebruiksvriendelijk, visueel aantrekkelijk en bevat alle geplande onderdelen (voor de planning zie hoofdstuk 8.7 Planning terugblik).

Het eindproduct voldoet vrijwel helemaal aan mijn verwachtingen. Wat ik niet had verwacht was het probleem met het opzetten van pipelines. Ik had verwacht dat ik een normale pipeline op kon zetten voor elke applicatie, maar in plaats moest ik een creatieve oplossing bedenken, om nog enigszins de functionaliteit van een pipeline te hebben.

Wat ik ook niet had verwacht, was dat het niet mogelijk was om alle applicaties op Azure te zetten, door gebrek aan credits. De nodige scripts om de applicaties op Azure te zetten had ik al wel gemaakt. Die scripts kan de Game Factory in de toekomst gebruiken om applicaties op Azure te zetten.

Mijn begeleider liet mij weten dat het opgeleverde eindproduct precies is wat hij in gedachten had voor de opdracht. De opdrachtgevers vonden dat het platform er visueel goed uit zag.

Dit product willen ze niet gelijk toepassen op alle applicaties, want er moet eerst vraag naar komen van klanten. Om dit product naar productie te laten gaan moet er nog wel veel werk worden gedaan en moeten er meer credits aangeschaft worden op Azure. Onder andere moet de pagina waar administrators gebruikers, privileges, activiteiten, taken en groepen kunnen aanpassen nog gemaakt worden en moeten de rest van de games die de Game Factory heeft aangepast worden om met het events te werken.

15 Aanbevelingen

Er zijn een aantal aanbevelingen die ik wil meegeven aan de opdrachtgevers voor de toekomst. Deze aanbevelingen zal ik in dit hoofdstuk beschrijven.

De opdrachtgevers hadden als wens, om rollen met templates van privileges te hebben, die administratoren op het platform aan gebruikers kunnen geven, om niet elke privilege handmatige aan gebruikers toe te kennen.

Dit zou ook mogelijk zijn om te doen voor de activiteiten en taken. Dan hoef je aan een gebruiker alleen maar een rol en een activiteiten template te geven, om alle privileges, taken en activiteiten toe te kennen. Dat zou het meest handig zijn wanneer een administrator meerdere gebruikers moet aanmaken, want dan zou diegene voor elke gebruiker alle activiteiten en taken voor die gebruiker handmatig moeten invullen.

Een andere mogelijkheid is om aan de rol ook de activiteiten template te geven. In plaats van een activiteiten template en een rol, bevat een rol dan zowel privileges als activiteiten.

De tweede aanbeveling die ik wil geven is om een vaste structuur op te stellen voor de codes van taken, activiteiten en privileges. Momenteel is hier een structuur in houden eenvoudig, maar om het op lange termijn gestructureerd te houden is een vaste opgestelde structuur nodig. Als het platform veel games heeft en veel gebruikers van verschillende organisaties heeft, is het mogelijk dat de hoeveelheid activiteiten, taken en privileges veel toe zou kunnen nemen. Zonder vaste opgestelde structuur is het dan mogelijk om snel het overzicht te verliezen, door de hoeveelheid codes voor activiteiten, taken en privileges.

16 Beroepstaken

In dit hoofdstuk zal ik beschrijven hoe ik voldaan hebt aan de beroepstaken, die ik had gekozen op het afstudeerplan (voor het afstudeerplan zie Bijlage 1: Afstudeerplan).

Op advies van de expert examinator en met toestemming van de begeleidende examinator en de expert examinator, heb ik de beroepstaak D-4 configureren toegevoegd. Deze beroepstaak besloot ik toe te voegen, omdat ik tijdens dit afstudeertraject een volledige Cloud omgeving had geconfigureerd met een frontend, een API, meerdere microservices, een database en opslag.

In onderstaande tabel is links de beroepstaak, in het midden het niveau van die beroepstaak en rechts waarom ik aan die beroepstaak hebt voldaan. De eerste drie beroepstaken zijn de verplichte beroepstaken en hebben geen niveau.

Beroepstaak	Niveau	Waarom ik die beroepstaak heb voldaan
A-1 Analyseren probleemdomein & opstellen probleemstelling	-	Ik heb tijdens dit afstudeertraject de probleemstelling en doelstelling in kaart gebracht. In dit dossier beschrijf ik ook wanneer, voor wie en waarom welke problemen zich voordoen, bij de huidige stand van zaken van de Game Factory. Ik beschrijf ook de uitgangssituatie, waarbij ik onder andere uitleg dat elke game een eigen database en website heeft en dat de website is gemaakt met het Angular framework.
G-C Kritisch, onderzoekend & methodisch werken	-	Tijdens het afstudeertraject heb voor alle gekozen middelen onderzoek gedaan, om de juiste keuze voor dat middel te maken. Voor de middelen die aan mij waren voorgeschreven, waar mijn mening nog wel invloed had, had ik ook onderzoek voor gedaan. Alleen voor de voorgeschreven middelen, waar ik absoluut geen keuze in had, had ik weinig of geen onderzoek naar gedaan. Bij elke situatie waar ik een keuze moest maken deed ik onderzoek, om de beste keuze te maken. Dit deed ik onder andere voor de keuze van backend framework, manier om een pipeline te maken, hoe en waar de game bundels worden opgeslagen, manieren om onderdelen te implementeren en test framework. Ik heb deze opdracht uitgevoerd met de Scrum werkmethode. Voor elke sprint en sommige losse onderdelen, maakte ik een plan om de sprint of het onderdeel aan te pakken. Ik had ook een aantal gestandaardiseerde technieken toegepast, met onder andere het observable design pattern en een microservice architectuur. Ik ben kritisch geweest bij het ontwikkelen, door bij het einde van elke sprint waar code in gemaakt is, de code te kritisch te reviewen en waar nodig de code te verbeteren. Tijdens de stand-ups had ik meerdere malen om feedback gevraagd, om zijn mening over de code te weten te komen. Ik had ook kritisch gekeken naar het project van mijn begeleider.
G-F Leren leren: voorbereiden op volgende studiefase & beroep	-	Door dit afstudeertraject ben ik mij bewuster geworden over wat voor werk ik wel en niet interessant vind om te doen. Ik ben er achter gekomen dat ik het veel interessanter vind, om op meerdere software niveaus te werken, dan op één niveau. Bij dit project werkte ik met frontend, backend, database, opslag en games. Ik weet hierdoor dus beter wat voor leer wensen ik heb voor de toekomst. Door dit afstudeertraject heb ik veel kennis opgedaan over het vinden en beoordelen van relevant leer materiaal, door alle onderzoeken dat ik heb verricht.

		Ik heb tijdens dit afstudeertraject ook veel kennis opgedaan, over het creëren en onderhouden van een netwerk met andere professionals, met onder andere LinkedIn.
D-1 Realiseren van software	3	Ik heb het hele platform zelfstandig Object Georiënteerd ontworpen en ontwikkeld, waarbij ik complexe problemen moest oplossen en creatieve oplossingen had toegepast. Ik had hiervoor zes applicaties ontwikkeld, namelijk één frontend, één API en vier microservices. Bij het ontwerpen en ontwikkelen van de applicaties had ik zowel de principes voor Object Oriented Design als de principes van Object Oriented Programming gevolgd. De Object Oriented Programming principes was ik niet op ingegaan in dit dossier, aangezien die zich plaats vonden op een zeer technisch niveau en buiten de scope van dit dossier vielen. De ontwikkelde frontend applicatie had ik gemaakt op basis van gemaakte ontwerpen en hield zich aan de architectuur van het Angular framework (voor architectuur van Angular zie (Architecture, 2022)). Vrijwel elke klasse op de frontend was een Angular onderdeel (module, component, service of directive), waarbij de enige uitzondering de klasse van de taak uitvoerders zijn. De backend applicaties volgen de NestJS architectuur met controllers en providers. De backend is ontworpen met een microservice architectuur, waarbij elke microservice zijn eigen verantwoordelijkheden heeft. Elke controller en provider op de API gaat maar over één microservice. Op de games microservice had ik ook twee providers, waar er één provider ging over de games in de database en de andere provider over aanvragen van Azure Storage ging. De code heeft een hoge kwaliteit door ESLint en TypeScript, maar ook door vele code reviews en onderzoeken.
D-2 Testen & evalueren	2	Met alle testen van alle applicaties bij elkaar opgeteld, heb ik totaal heb 155 test cases gemaakt, waarvan 145 unit testen en 10 end-to-end testen. Voor alle applicaties heb ik 100% code coverage bereikt, waar elke pad dat de code kan nemen getest is (met een if statement heb je bijvoorbeeld twee paden). Voor de API had ik de end-to-end test gemaakt, om te zien of de aanvragen correct aankomen op de API.
D-4 Configureren	2	Ik heb een volledige Cloud omgeving ontwikkeld in Azure. Ik had een virtueel network geconfigureerd waar de container instances van alle applicaties in zaten, waarbij alleen de frontend van buitenaf te bereiken is. De frontend kon http aanvragen sturen naar de API. De API gaf de aanvraag dan door aan de correcte microservice via een TCP verbinding en kreeg het ook via die verbinding weer terug, wanneer de microservice klaar was met de aanvraag. De providers van de microservices waren in verbinding met het deel van database, waar die microservice verantwoordelijk voor was. Eén provider van de games microservice was als enige verbonden niet verbonden met de database, maar met Azure storage. Ik had ook bij meerdere services op Azure beveiliging geconfigureerd.

17 Lijst van Figuren

18 Bronnenlijst

5 design principles for microservices. (2022, Februari 3). Opgehaald van redhat.com: https://developers.redhat.com/articles/2022/01/11/5-design-principles-microservices#five_design_principles_for_microservices

About Node.js. (2021, September 24). Opgehaald van Node.js: https://nodejs.org/en/about/

Angular vs react. (2021, Oktober 25). Opgehaald van Simform: https://www.simform.com/blog/angular-vs-react/

Angular vs react vs vue. (2021, Oktober 25). Opgehaald van Codeinwp: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

API framework. (2021, September 14). Opgehaald van RapidAPi: https://rapidapi.com/blog/api-glossary/api-framework/

Architechture components. (2022, Februari 7). Opgehaald van Angular.io: https://angular.io/guide/architecture-components

Architechture components directives. (2022, Februari 8). Opgehaald van Angular.io: https://angular.io/guide/architecture-components#directives

Architecture. (2022, Maart 16). Opgehaald van angular.io: https://angular.io/guide/architecture

Azure container example. (2021, Oktober 5). Opgehaald van GoDataDriven: https://godatadriven.com/blog/azure-container-instance-example/

Azure container registry. (2021, Oktober 25). Opgehaald van Azure.microsoft.com: https://azure.microsoft.com/en-us/services/container-registry/#overview

Azure storage accounts. (2021, September 27). Opgehaald van Microsoft: https://docs.microsoft.com/en-us/learn/modules/azure-storage-fundamentals/azure-storage-accounts

Best javascript testing framework. (2022, Januari 20). Opgehaald van openbase.com: https://openbase.com/categories/js/best-javascript-testing-framework-libraries

Blob data type. (2022, Maart 8). Opgehaald van docs.oracle: https://docs.oracle.com/javadb/10.8.3.0/ref/rrefblob.html

buildah. (2021, Oktober 25). Opgehaald van Red hat catalog: https://catalog.redhat.com/software/containers/detail/5dca3d76dd19c71643b226d5?container-tabs=overview

Built-in directives. (2022, Maart 15). Opgehaald van angular.io: https://angular.io/guide/built-in-directives

CGI. (2021, September 7). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/CGI_Inc.

Client vs server side rendering. (2021, Oktober 25). Opgehaald van toptal: https://www.toptal.com/front-end/client-side-vs-server-side-pre-rendering

Container instances egress ip adresss. (2021, Oktober 19). Opgehaald van Microsoft docs: https://docs.microsoft.com/nl-nl/azure/container-instances/container-instances-egress-ip-address

Container instances virtual network concepts. (2021, Oktober 19). Opgehaald van Microsoft docs: https://docs.microsoft.com/nl-nl/azure/container-instances/container-instances-virtual-network-concepts

Container registry. (2021, Oktober 22). Opgehaald van Azure: https://azure.microsoft.com/nl-nl/services/container-registry/

Container registry faq. (2021, Oktober 29). Opgehaald van Microsoft docs: https://docs.microsoft.com/nl-nl/azure/container-registry/container-registry-faq

Difference between one-way and two-way databinding. (2021, Oktober 25). Opgehaald van Stackoverflow: https://stackoverflow.com/questions/34519889/can-anyone-explain-the-difference-between-reacts-one-way-data-binding-and-angula

DOM manipulation. (2022, Februari 3). Opgehaald van Typescriptlang.org: https://www.typescriptlang.org/docs/handbook/dom-manipulation.html

Express. (2021, September 14). Opgehaald van Express: https://expressjs.com

Express github. (2021, September 14). Opgehaald van Github: https://github.com/expressjs/express

FileStore. (2021, September 27). Opgehaald van Microsoft docs: https://docs.microsoft.com/en-us/azure/databricks/data/filestore

Getting started with NestJS. (2021, September 14). Opgehaald van Better Programming: https://betterprogramming.pub/getting-started-with-nestjs-a4e8b0b09db4

Google's Material. (2022, Februari 22). Opgehaald van theverge.com: https://www.theverge.com/2021/9/10/22666355/google-material-you-design-gmail-calendar-docs-android

Hoofdpagina. (2022, Februari 22). Opgehaald van material.angular.io: https://material.angular.io

How to download blob container/Directory from azure storage to local folder? (2022, Januari 6). Opgehaald van Microsoft developer network: https://social.msdn.microsoft.com/Forums/azure/en-US/ea40563e-3295-4ae8-8ed4-94357fc5bd27/how-to-download-blob-containerdirectory-from-azure-storage-to-local-folder?forum=windowsazuredata

Icon. (2022, Maart 21). Opgehaald van material.angular.io: https://material.angular.io/components/icon/overview

Jasmine and Karma. (2022, Januari 20). Opgehaald van codecraft.tv: https://codecraft.tv/courses/angular/unit-testing/jasmine-and-karma/

Jasmine hoofdpagina. (2022, Januari 20). Opgehaald van Jasmine: https://jasmine.github.io

Jest. (2021, September 27). Opgehaald van Jest.io: https://jestjs.io

Jira features. (2022, Maart 21). Opgehaald van atlassian.com: https://www.atlassian.com/software/jira/features

Jira software. (2022, Maart 21). Opgehaald van atlassian.com: https://www.atlassian.com/software/jira

Lijst met 25 API frameworks. (2021, September 14). Opgehaald van Esparkinfo: https://www.esparkinfo.com/node-js-frameworks.html

Loopback microservice documentatie. (2021, December 20). Opgehaald van Loopback.io: https://loopback.io/search/?q=microservice&sidebar=lb4_sidebar

loopback pros and cons. (2021, December 20). Opgehaald van voidcanvas: https://www.voidcanvas.com/loopback-pros-and-cons

Material Design. (2022, Februari 22). Opgehaald van Interaction-design.org: https://www.interaction-design.org/literature/topics/material-design

MDC Web on other frameworks. (2022, Februari 22). Opgehaald van material.io: https://material.io/develop/web/guides/framework-wrappers

NestJS. (2021, September 14). Opgehaald van NestJS: https://docs.nestjs.com

NestJS API File Operations Using Azure Blob Storage. (2022, Januari 11). Opgehaald van learmoreseekmore.com: https://www.learmoreseekmore.com/2021/03/nestjs-api-file-operations-using-azure-blob-storage.html

NestJS github. (2021, September 14). Opgehaald van Github: https://github.com/nestjs/nest

NodeJS + express vs NestJS. (2021, September 14). Opgehaald van dev.to: https://dev.to/giovannikleincampigoto/nodejs-express-vs-nestjs-a-vision-about-architecture-and-good-pratices-571i

Open Container Initiative. (2021, Oktober 22). Opgehaald van Opencontainer: https://opencontainers.org

Oras introduction. (2021, Oktober 25). Opgehaald van Oras: https://oras.land

parameterised routes. (2022, Maart 14). Opgehaald van codecraft.tv: https://codecraft.tv/courses/angular/routing/parameterised-routes/

React vs angular key difference. (2021, Oktober 25). Opgehaald van Guru99: https://www.guru99.com/react-vs-angular-key-difference.html

Saving Files In A Database Or In A File System. (2021, September 27). Opgehaald van habiletechnologies: https://habiletechnologies.com/blog/better-saving-files-database-file-system/

Scrum vs agile. (2022, Maart 21). Opgehaald van agilescrumgroup.nl: https://agilescrumgroup.nl/scrum-vs-agile/

Self-hosted Jira alternatives. (2022, Maart 21). Opgehaald van alternativeto.net: https://alternativeto.net/software/jira/?platform=self-hosted

Should binary files be stored in the database? (2021, September 27). Opgehaald van Stackexchange: https://dba.stackexchange.com/questions/2445/should-binary-files-be-stored-in-the-database

Single responsibility principle. (2022, Maart 22). Opgehaald van Stackify: https://stackify.com/solid-design-principles/

SOLID. (2022, Maart 22). Opgehaald van digitalocean.com: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

superset. (2021, December 20). Opgehaald van thefreedictionary: https://encyclopedia2.thefreedictionary.com/superset

Technical overview of azure databricks. (2021, September 27). Opgehaald van Azure: https://azure.microsoft.com/es-es/blog/a-technical-overview-of-azure-databricks/

Testing Angular faster with Jest. (2022, Januari 8). Opgehaald van xfive.co: https://www.xfive.co/blog/testing-angular-faster-jest/

The good and the bad of angular. (2021, Oktober 25). Opgehaald van Altexsoft: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/

The Most Popular Prioritization Techniques and Methods. (2022, Januari 27). Opgehaald van Altexsoft: https://www.altexsoft.com/blog/business/most-popular-prioritization-techniques-and-methods-moscow-rice-kano-model-walking-skeleton-and-others/

The Most Popular Prioritizatoin Techniques and Methods. (2022, Januari 28). Opgehaald van Altexsoft: https://www.altexsoft.com/blog/business/most-popular-prioritization-techniques-and-methods-moscow-rice-kano-model-walking-skeleton-and-others/

Typescript. (2021, September 24). Opgehaald van Microsoft: https://devblogs.microsoft.com/typescript/

Unity web player and browser communication. (2022, Maart 14). Opgehaald van docs.unity3d.com: https://docs.unity3d.com/530/Documentation/Manual/UnityWebPlayerandbrowsercommunication.html

Vue is good, but is it better then angular or react? (2021, Oktober 25). Opgehaald van valuecoders: https://www.valuecoders.com/blog/technology-and-apps/vue-js-comparison-angular-react/

Waterval-methode. (2021, Oktober 22). Opgehaald van Centillion: https://www.centillion.nl/project-management/hoe-houd-ik-it-project-kosten-in-de-hand/attachment/67106520-waterfall-development-process/

What is Agile? (2022, Maart 21). Opgehaald van agilenutshell.com: http://www.agilenutshell.com

What is angular? (2021, September 24). Opgehaald van angular.io: https://angular.io/guide/what-is-angular

What is node.js. (2021, September 24). Opgehaald van Netguru: https://www.netguru.com/glossary/node-js

What is Podman? (2021, Oktober 25). Opgehaald van Podman: https://docs.podman.io/en/latest/

Which one is better for unit test? JEST or Jasmine. (2022, Januari 24). Opgehaald van Reddit: https://www.reddit.com/r/Angular2/comments/oa800b/which_one_is_better_for_unit_test_jest_or_jasmine/

Why use jest over karma for Angular testing? (2022, Januari 24). Opgehaald van medium.com: https://charith-hettiarachchi.medium.com/why-use-jest-over-karma-for-angular-testing-b56ffa82f8

Why we chose Jasmine over Jest and Mocha. (2022, Februari 8). Opgehaald van bitsrc.io: https://blog.bitsrc.io/angular-9-testing-a-comparison-between-jasmine-jest-and-mocha-acc57bcab836

Bijlage 1: Afstudeerplan

Afstudeerblok	Blok 1 2021 – blok 3 2022
Startdatum uitvoering afstudeeropdracht	30 Augustus 2021
Inleverdatum afstudeerdossier volgens jaarrooster	22 Maart 2022

Studentnummer	16036247
Achternaam	Deurloo
Voorletters	R.A.
Roepnaam	Ruben
Adres	Essehout 49
Postcode	2719 MD
Woonplaats	Zoetermeer
Telefoonnummer	0793618509
Mobiel nummer	0616221818
Privé emailadres	ruben.deurloo@gmail.com

Differentiatie	SE
Afstudeerprogramma	GDS
Domein*	SE
Locatie	Zoetermeer
Variant	Voltijd

**geef hier aan binnen welk domein jouw opdracht valt, m.a.w. wat voor type opdracht het is*

Naam studieloopbaanbegeleider	Machteld Schölvinck
Naam begeleidend examinator	Ton Biegstraaten
Naam expert examinator	Sebastiaan Rieter

Bedrijf

Naam	CGI
Afdeling	Game Factory
Bezoekadres	George Hintzenweg 89
Postcode	3068 AX
Plaats	Rotterdam
Telefoonnummer	0885640000
URL	https://www.cginederland.nl/nl

Opdrachtgever

Achternaam	Roman
Voorletters / Voornaam	Adolfo
Titel / Opleiding	Dhr
Functie	Serious Games & Gameification designer
Telefoonnummer (werk)	0617117069
Emailadres (werk)	Adolfo.romansanchez@cgi.com
is de opdrachtgever ook de bedrijfsmentor?	Nee

Bedrijfsmentor

Achternaam	Van der Meijs
Voorletters / Voornaam	Rens
Titel / Opleiding	Dhr

Functie	Game Developer
Telefoonnummer (werk)	0641647198
Emailadres (werk)	rens.vander.meijs@cgi.com

Titel afstudeeropdracht

Ontwikkelen van een platform voor de web games van de Game Factory van CGI.

Opdrachtomschrijving

1. Bedrijf & Probleemdomein

CGI is een Canadees multinationaal commercieel IT bedrijf voor consulting en systeem integraties. Diensten die CGI levert zijn onder andere applicaties, bedrijfsprocesdiensten, IT-infrastructuurdiensten, bedrijfsadvies, IT-outsourcingsdiensten en systeemintegratiediensten. CGI heeft 77500 mensen in dienst over meer dan 400 kantoren over 40 landen verspreid. (https://en.wikipedia.org/wiki/CGI_Inc.#cite_note-investwek-11)

CGI heeft een grote hoeveelheid klanten in verschillende markten waaronder: communicatie, gezondheid, productie, olie en gas, berichten en logistiek en transport. Klanten zijn onder andere overheden, staten, gemeentes, ministeries van defensie, inlichtingen, ruimte, gezondheid, financiën en transport. (https://en.wikipedia.org/wiki/CGI_Inc.#cite_note-investwek-11)

Een van de groepen binnen CGI is de Game Factory. De game factory maakt Gamification oplossingen voor bedrijven om bijvoorbeeld brandstofgebruik te verlagen bij DHL of beveiliging te verbeteren. Hierdoor heeft de Game Factory een grote hoeveelheid web applicaties die allemaal een eigen domein hebben.

2. Aanleiding & probleemstelling

De Game Factory van CGI heeft inmiddels een grote hoeveelheid games onder zich. Dit zijn allemaal web games die ook gedownload kunnen worden. Al deze games hebben deel van hun code vrijwel overeen komen met elkaar, omdat ze allemaal web games zijn. Het probleem hierbij is wanneer je een applicatie moet updaten/upgraden, je de code van de andere applicaties ook moet updaten. Ze hebben mij gevraagd een platform te maken met Typescript en NodeJS waar die games dan op kunnen draaien.

Hierbij willen ze dan dat er microservices gebruikt worden die ze dan kunnen gebruiken voor alle games. Dan hebben ze veel minder code duplicatie in hun games zitten.

3. Doelstelling

Het doel is dat na de afstudeeropdracht de Game Factory een platform heeft waar alle webgames van de Game Factory op staan, dat gemaakt is met microservices. Hierdoor hebben ze veel minder gedupliceerde code hebben en ze alle games op een plek hebben staan.

4. Concrete werkzaamheden en producten.

De werkzaamheden die ik ga verrichten heb ik onderverdeeld in Features. Voor dat ik aan alle features van de opdracht begin heb ik eerst een oriëntatie waaronder een starterdag van CGI. De werkzaamheden ga ik uitvoeren met scrum. Hierbij is elke feature dan onderverdeel in kleinere taken

die allemaal getest zullen worden. Het plan is dan om alle features af te hebben bij het einde van het afstuderen, maar er is zeker een mogelijkheid dat dat niet mogelijk is.

Oriëntatie - 1-2 weken

- Bedrijf verkennen/ introduceren
- Werkwijze en afspraken maken(wekelijkse meetings of dat soort afspraken die ze hebben)
- Plan van aanpak maken
- Scrum opzetten
    - Features omzetten naar epics en user stories
    - Definition of done maken
    - Acceptatie criteria maken
- Lijst met features/backlog aanscherpen door probleem te onderzoeken
- Opzetten environment
- Hardware en software regelen
- Starterdag bij CGI
- Onderzoeken wat het beste framework is voor een platform voor webgames
- Onderzoeken wat het beste test teamwork is voor het testen van een platform voor webgames en een database

De features:

Feature 1: Het inloggen - 2-4 weken

- Een frontend maken voor het inloggen en aanmaken van gebruikers
- Een database ontwerpen en maken voor users
- De backend maken voor het inloggen van users
- De backend maken voor het aanmaken van users
- De website laten draaien met docker of op een server als ik die krijg
    - Uitzoeken hoe ik dit doe
- Alles testen

Feature 2: De games - 8-10 weken

- Een frontend maken voor de pagina waar de games staan en gespeeld kunnen worden
- De games ergens opslaan
    - Uitzoeken waar en hoe

- De lijst met games tonen op de frontend
- De games kunnen spelen
- Een database aanmaken waarin onder andere wordt opgeslagen:
    - Wie welke game geüpload heeft
    - De versie van de game
- Alles testen

Feature 3: Het leaderboard - 4-6 weken

- Uitzoeken hoe de leaderboard werkt bij de games
- Een ontwerp maken hoe de leaderboard database eruitziet
    - Die database vervolgens maken
- Met een microservice de score van een game naar de juiste database sturen
- (Een aantal) games aanpassen zodat de score te zien is in de juiste leaderboard
- Alles testen

Feature 4: Het uploaden - 2-3 weken

- Een frontend maken voor het uploaden games
    - Hierbij ook opslaan wie het geüpload heeft
    - Een game moet een unieke game naam hebben
- De backend maken voor het uploaden van unity games waarbij de game op dezelfde manier wordt opgeslagen als de ander games
- Alles testen

Feature 5: Het updaten - 2-3 weken

- De frontend voor het uploaden van unity games aanpassen zodat er ook een nieuwe versie van een bestaande game geüpload kan worden
    - Alleen door degene die de vorige versie van die game geüpload heeft
- Alles testen

Om dit uit te voeren ga ik onder andere gebruik maken van HTML, CSS, Typescript, Azure, Unity, NodeJS, ESLint en scrum.

5. Beroepstaken

Verplichte 3:

- A1 analyseren probleemdomein
    - Bij dit project ga ik analyseren wat het probleem precies is om zo het plan van aanpak/concrete werkzaamheden aan te scherpen.

- GC kritisch en methodisch werken
    - Bij dit project gebruik ik ESLint als kwaliteitscontrole. Daarnaast ga ik met scrum werken, wat een veel gebruikte methode is. Ik ga ook alle code die ik schrijf bij een user story testen om er zo veel mogelijk fouten uit te halen.
- Gf Leren leren
    - Hierbij leer ik hoe ik moet zoeken en beoordelen van relevant studiemateriaal, want hiermee zal ik op onbekend terrein werken.

Van afstudeerprogramma 1:

- D1 Realiseren van software
    - Bij mijn afstudeeropdracht ga ik software ontwikkelen.

Gekozen 2:

- D3 Realiseren & gebruiken database
    - Bij mijn afstudeeropdracht ga ik een database aanmaken, gebruiken en testen.
- D2 Testen & Evalueren
    - Bij de afstudeeropdracht ga ik alle code die ik maak testen.

6. Concept Bibliografie

Stackoverflow: *Stackoverflow*. (2021). Stackoverflow. https://stackoverflow.com

Bijlage 2: Requirements users en games

- De gebruikers hebben de mogelijkheid om in te loggen met een email adres en een wachtwoord.
- Inloggen is niet verplicht.
- Elke gebruiker krijgt een id om als referentie te gebruiken in de database vanwege privacy redenen.
- Een admin kan accounts aanmaken voor gebruikers met een naam, email adres, wachtwoord en een rol.
- Het wachtwoord en de naam van een gebruiker mogen niet langer zijn dan 32 karakters.
- Het systeem controleert bij het aanmaken van accounts voor gebruikers of de gegevens valide zijn ingevuld.
- Elke organisatie heeft op zijn minst één groep waar gebruikers bij kunnen horen.
- Elke groep hoort bij een organisatie.
- Elke organisatie heeft een naam en een id.
- Elke groep heeft een id, naam en beschrijving.
- Bij een groep kunnen geen of meerdere gebruikers horen.
- Een gebruiker kan bij één of meerdere groepen horen.
- Een organisatie kan meerdere groepen hebben.
- Een gebruiker kan bepaalde privileges hebben om acties uit te mogen voeren (bijvoorbeeld acties zoals bepaalde games mogen spelen, wachtwoord wijzigen).
- Privileges gelden voor het platform zelf of voor verschillende games.
- Een admin kan privileges toekennen aan een gebruiker.
- Elke gebruiker kan andere privileges hebben.
- Een privilege van een gebruiker kan een eind datum hebben die aangeeft tot wanneer die privilege geldig is.
- Een privilege van een gebruiker kan een begin datum hebben die aangeeft tot wanneer die privilege geldig is.
- Een rol bevat één of meerdere privileges.
- Bij het aanmaken van een gebruiker krijgt de aangemaakte gebruiker de privileges die horen bij de rol die geselecteerd is door het systeem.
- Een admin kan een rol aanmaken
- Een admin kan privileges toekennen aan een rol.
- Een privilege heeft een code dat aangeeft waar die privilege voor is.
- Individuele games kunnen bepaalde privileges nodig hebben om gespeeld te worden door gebruikers.
- Elke game is te spelen op het platform in de browser.
- Een game heeft ook de mogelijkheid om gedownload te worden om lokaal te spelen.
- Van elke game wordt bijgehouden welke versie van die game gespeeld kan worden.
- Elke game heeft een naam, logo en een beschrijving.

- Op het platform is te zien elke game met logo en beschrijving indien de gebruiker daar de benodigde privileges voor heeft.
- Elke instelling die een gebruiker heeft gedaan voor zowel het platform of een game wordt opgeslagen door het systeem.
- Het systeem hoeft geen pagina te hebben voor het uploaden of updaten van games.
- Het systeem heeft als hoofdpagina de loginpagina.
- Een gebruiker kan een dashboard zien na het inloggen met alle games waar die gebruiker privileges voor heeft.
- Elke admin krijgt ook een dashboard zien na het inloggen. (wat hier op komt hoor ik tegen die tijd

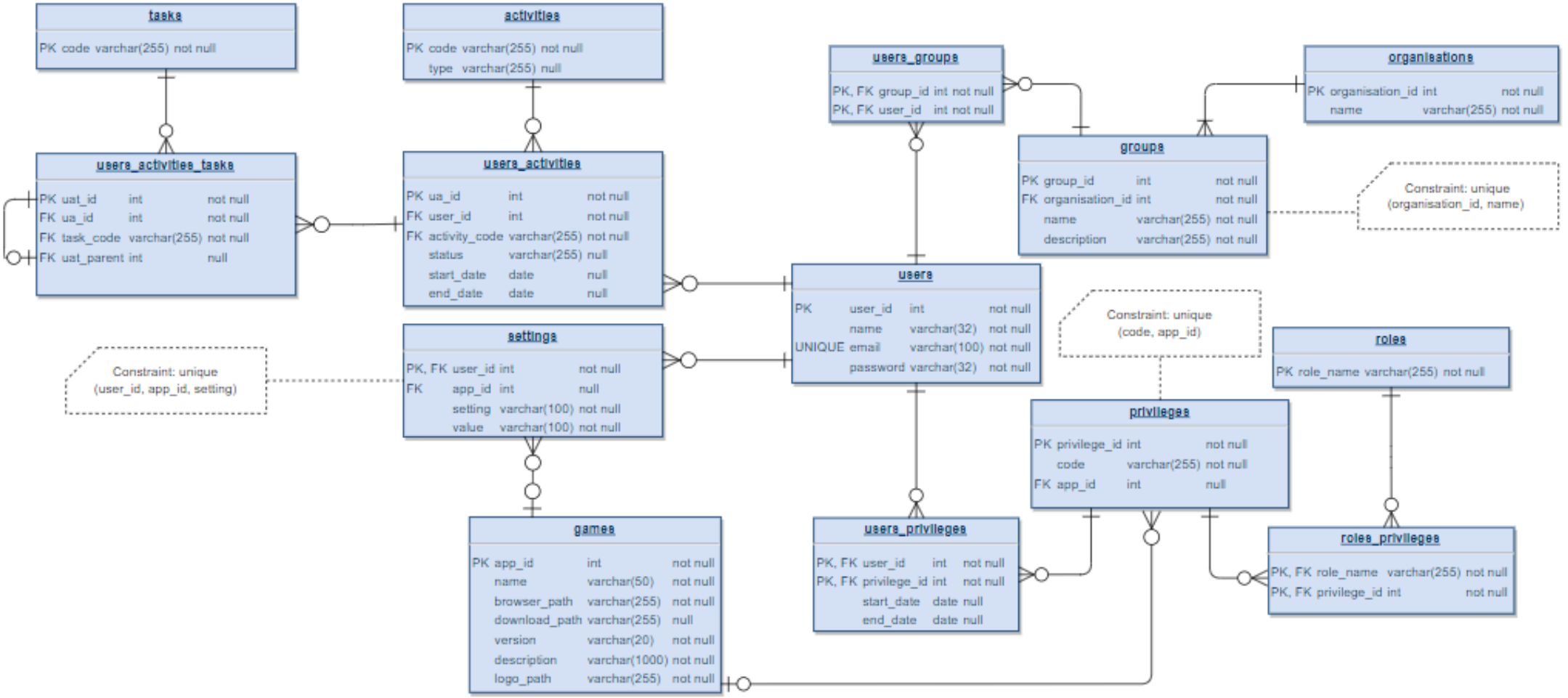
Bijlage 3: Activiteiten systeem requirements

- Wanneer een gebruiker een activiteit activeert wordt de lijst met acties opgehaald die dan uitgevoerd moeten worden.
- De acties die het systeem uitvoert bij elke activiteit van een gebruiker kunnen per gebruiker anders zijn.
- Een activiteit bevat een code die aangeeft wat die activiteit is (bijvoorbeeld confirm-login, goto-dashboard, etc.)
- Een actie van een activiteit van een gebruiker kan een andere actie hebben als parent die ook bij diezelfde activiteit van die gebruiker hoort.
- Een activiteit van een gebruiker kan meerdere acties bevatten.
- De parent van een actie van een gebruiker moet door het systeem eerst uitgevoerd worden voordat de actie zelf uitgevoerd mag worden.
- Een actie kan maar één ding doen(bijvoorbeeld laad component of verstuur data).
- Het systeem slaat per activiteit per user of die user die activiteit gedaan heeft.
- Er zijn verschillende typen activiteiten.
- Eén van de typen die een activiteit kan zijn is herhaalbaar.
- Andere typen activiteiten zijn nog niet bekend maar de opdrachtgever is er wel zeker van dat er nog andere typen kunnen zijn later.
- Een gebruiker kan een activiteit die herhaalbaar is herhalen.
- Een activiteit kan per gebruiker een eind datum hebben dat aangeeft tot wanneer die gebruiker die activiteit kan uitvoeren.
- Een activiteit kan per gebruiker een begin datum hebben dat aangeeft vanaf wanneer de gebruiker die activiteit kan uitvoeren.
- Bij het aanmaken van een gebruiker is het mogelijk om activiteiten en acties toe te kennen aan die gebruiker.

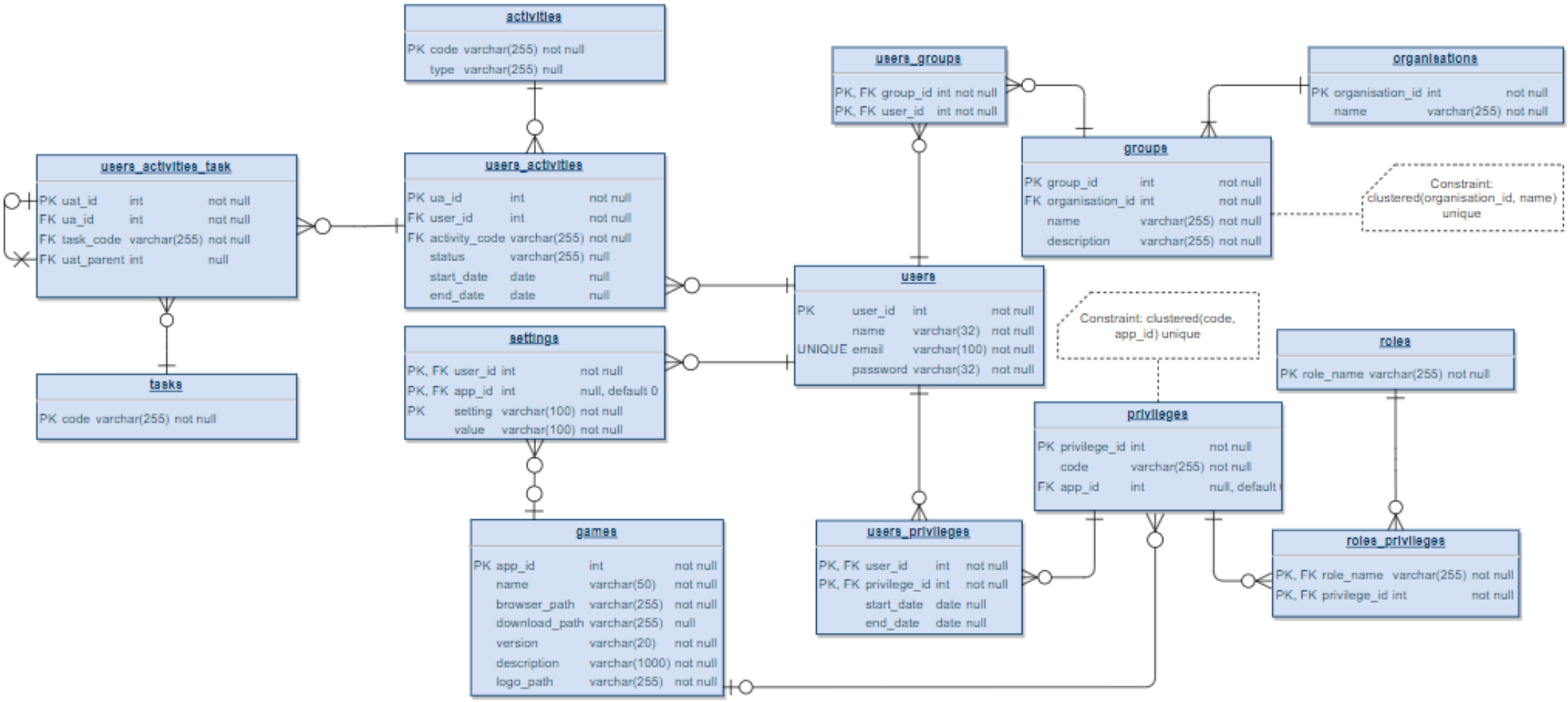
Bijlage 4: Tabel serie van events

Stap	Web	Unity
1	Voeg de unityloader.js toe	
2		Installeert App. Stuurt dan container-install-app.
3	Ontvang container-install-app -> verstuur dan container-configuration	
4		Ontvang container-configuration
5		verstuur container-start-async-operation met id: 0, alias: container-install-bundle en payload: bundles/core-bundle
6	Ontvang container-start-async-operation met payload -> triggers de awaitEvent in applyContainer in app-container-service -> dat start startBundleOperation met de payload van de async event -> in startBundleOperation verstuur container-async-operation-status met als payload de id, status en percentage	
7		Ontvang container-async-operation-status
8	Verstuur container-async-operation-status Als de operatie op 100% staat verstuur in chunks de bundle door de bundle op te delen in chunks en voor elk chunk te versturen naar unity container-async-operation-chunk met als payload de chunk van de bundle	
9		Ontvang de container-async-operation-chunk events
	Herhaal stap 5 tot en met 9 totdat alle bundles die meegegeven zijn in de configuratie bij stap 3 gedownload zijn. Dan verstuurd Unity container-ready	
	Ontvang container-ready -> Zet installed boolean op true	
	Klik op een ui element om verder te gaan (start game)	
	Verstuur app-container-load-content-bundle-scene met payload: “conveyor-sorter-scene:Conveyor Sorter Scene” Dan verstuur app-container-starts-activity	
		Ontvang app-container-load-content-bundle-scene met payload: “conveyor-sorter-scene:Conveyor Sorter Scene”
		Ontvang app-container-start-activity met payload play-conveyor-sorter-game -> Installeerd de bundle van conveyor sorter game dan start die game

Bijlage 5: Eerste versie database diagram



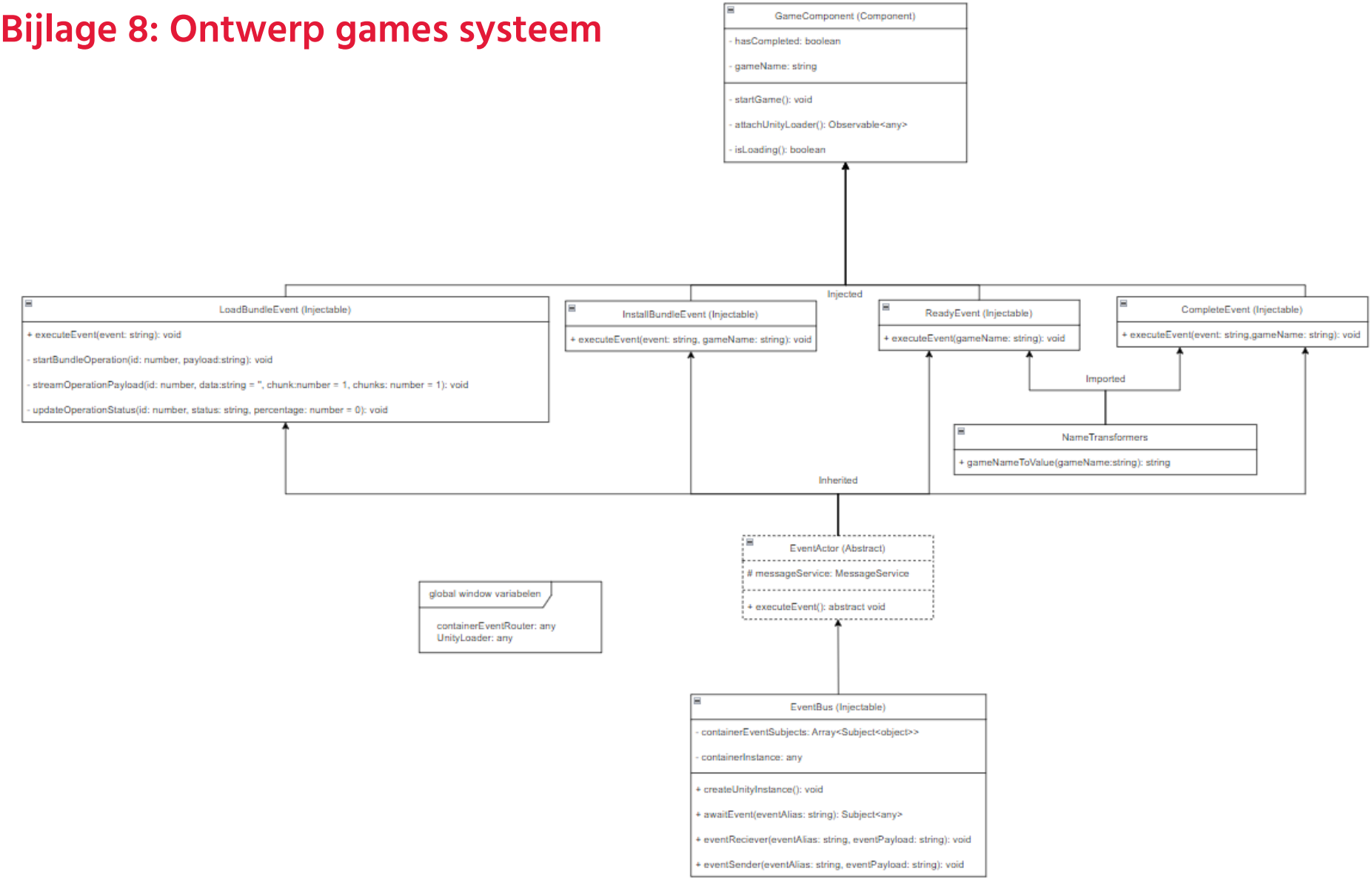
Bijlage 6: Database diagram met activiteiten systeem



Bijlage 7: Tabel losgetrokken events

Event name web	Event name unity	Code	Zender	Wat het betekend
CONTAINER_INSTALL_APP	CONTAINER_INSTALL_APP_ALIAS	container-install-app	Unity	Unity verstuurd dit wanneer de unity instantie is opgestart om web te laten weten dat het klaar is om events te ontvangen.
CONTAINER_CONFIGURATION	CONTAINER_CONFIGURATION_ALIAS	container-configuration	Web	Web verstuurd een serie van strings die laten weten de bundles die gedownload moeten worden. Dit is als reactie op het krijgen van event container-install-app.
CONTAINER_START_ASYNC_OPERATION	CONTAINER_START_ASYNC_OPERATION_ALIAS	container-start-async-operation	Unity	Unity verstuurd dit naar het web om een async operatie te starten op het web.
CONTAINER_ASYNC_OPERATION_STATUS	CONTAINER_ASYNC_OPERATION_STATUS_ALIAS	container-async-operation-status	Web	Web verstuurd dit naar unity om unity up-to-date te houden met hoe een async operatie verloopt met als payload een bericht dat zegt scheduled, success, inProgress of error.
CONTAINER_ASYNC_OPERATION_CHUNK	CONTAINER_OPERATION_CHUNK_ALIAS	container-async-operation-chunk	Web	Web verstuurd dit naar unity met een stuk een gedownloade bundel, wanneer het downloaden van een bundle gelukt is.
CONTAINER_READY	CONTAINER_READY_ALIAS	container-ready	Unity	Dit verstuurd unity wanneer het alle bundles (die meegestuurd zijn met de container-configuration event) ontvangen heeft.
CONTAINER_INSTALL_BUNDLE	CONTAINER_LOAD_BUNDLE_ALIAS	container-install-bundle	Unity	Unity stuurt dit als reactie op het ontvangen van container-configuration event met de naam van de bundle die unity wilt hebben (voor elke bundle in de configuratie). Web start dan de async operatie om de bundle van de payload te downloaden.
CONTAINER_LOAD_CONTENT_BUNDLE_SCENE	APP_CONTAINER_LOAD_CONTENT_BUNDLE_SCENE	app-container-load-content-bundle-scene	Web	Web stuurt dit naar unity wanneer het een bepaalde scene bundle geladen wilt hebben. Unity laad dan de scene.
CONTAINER_UNLOAD_CONTENT_BUNDLE_SCENE	APP_CONTAINER_UNLOAD_CONTENT_BUNDLE_SCENE	app-container-unload-content-bundle-scene	Web	Web verstuurd dit wanneer het een bundle niet meer nodig heeft waarbij unity het dus kan unloaden. Web geeft dan mee als payload de bundle string.
CONTAINER_START_ACTIVITY	APP_CONTAINER_START_ACTIVITY	app-container-start-activity	Web	Web verstuurd dit naar unity wanneer het spel wilt starten. (moet wel eerst geladen zijn)
CONTAINER_COMPLETE_ACTIVITY	APP_CONTAINER_COMPLETE_ACTIVITY	app-container-complete-activity	Unity	Unity stuurt dit naar web wanneer het spel uitgespeeld is en verder wilt gaan.

Bijlage 8: Ontwerp games systeem



Bijlage 9: Geprioriteerde backlog

User story:	Prioriteit:	Epic:
Als systeem wil ik op de frontend de bomen van alle taken kunnen verkrijgen van een activiteit van een gebruiker waarbij de activiteit gestart is, nog niet geëindigd is of geen datum heeft zodat ik bomen van geldige taken heb om uit te kunnen voeren	Must have	Activity systeem
Als systeem wil ik de taken van een activiteit van een gebruiker kunnen uitvoeren op de goede volgorde van laagste kind taak tot hoogste parent taak om de taken uit te kunnen voeren op de goede volgorde	Must have	Activity systeem
Als product owner wil ik dat de hoofdpagina de inlogpagina is zodat dat de eerste pagina is die gebruikers zien als ze de website openen	Should have	
Als gebruiker wil ik in kunnen loggen met mijn email en gebruikersnaam om toegang te krijgen tot mijn account	Should have	Users systeem
Als gebruiker wil ik een game kunnen spelen op de frontend waar ik geen privileges voor nodig heb, zodat ik de games kan spelen die geen privileges nodig hebben	Should have	Games systeem
Als gebruiker wil ik een game kunnen spelen waar ik de privileges voor heb om die te mogen spelen.	Should have	Games systeem en Users systeem
Als gebruiker wil ik een dashboard hebben waar ik alle games op kan zien waar geen privileges voor nodig zijn en de games waar ik privileges voor heb, zodat ik een overzicht heb van alle games die ik kan spelen	Should have	
Als admin wil ik een dashboard hebben om administratie op uit te kunnen voeren	Should have	Users systeem
Als admin wil ik een gebruiker kunnen aanmaken op mijn dashboard als ik de privileges hebt om dat te doen zodat ik gebruikers kan aanmaken	Should have	Users systeem
Als admin wil ik activiteiten kunnen toekennen aan gebruikers op mijn dashboard als ik daar privileges voor hebt zodat ze activiteiten kunnen uitvoeren	Could have	Users systeem
Als admin wil ik taken kunnen toekennen aan gebruikers op mijn dashboard als ik daar privileges voor hebt zodat de activiteiten ook wat kunnen doen wanneer een gebruiker een activiteit doet	Could have	Users systeem
Als admin wil ik privileges kunnen geven aan gebruikers als ik daar privileges voor hebt zodat ik gebruikers privileges kan toekennen	Could have	Users systeem
Als admin wil ik een rol kunnen aanmaken op mijn dashboard als ik daar privileges voor hebt zodat ik rollen kan aanmaken	Could have	Users systeem
Als admin wil ik privileges kunnen toekennen aan een rol op mijn dashboard als ik daar privileges voor hebt zodat ik een rollen kan hebben met privileges om met een template privileges aan gebruikers toe te kennen	Could have	Users systeem
Als admin wil ik privileges kunnen geven aan gebruikers met gebruik van rollen op mijn dashboard als ik daar privileges voor hebt zodat ik niet handmatig alle privileges hoeft toe te kennen aan de gebruiker	Could have	Users systeem
Als admin wil ik organisaties kunnen toevoegen op mijn dashboard als ik daar privileges voor hebt omdat ik organisaties wil om groepen voor aan te maken	Could have	Users systeem
Als admin wil ik groepen kunnen toevoegen op mijn dashboard als ik daar privileges voor hebt om groepen te hebben om gebruikers aan toe te kunnen voegen	Could have	Users systeem
Als admin wil ik gebruikers kunnen toevoegen aan groepen op mijn dashboard als ik daar privileges voor hebt voor administratie	Could have	Users systeem

Als gebruiker wil ik activiteiten kunnen hebben die mij een opdracht geven waarbij ik iets op de website moet doen (bijvoorbeeld open je profiel) om de opdracht te volbrengen.	Could have	Activity systeem
Als gebruiker wil ik dat mijn instellingen voor het platform (bijvoorbeeld taal) opgeslagen worden zodat ik diezelfde instellingen hebt de volgende keer dat ik inlog	Could have	Games systeem
Als gebruiker wil ik de game kunnen downloaden zodat ik het op mijn computer lokaal kan spelen	Won't have	Games systeem
Als gebruiker wil ik dat mijn instellingen voor games opgeslagen worden zodat ik diezelfde instellingen hebt de volgende keer dat ik het spel speel	Won't have	Games systeem
Als gebruiker wil ik activiteiten kunnen hebben die mij een opdracht geven waarbij ik iets in een game moet doen (bijvoorbeeld doe de tutorial van game A) om de opdracht te volbrengen.	Won't have	Activity systeem

Bijlage 10: Angular heeft een eigen server uitleg

Wanneer je een Angular project start via de Angular Command Line Interface(CLI) dan draait dat op zijn eigen lokale server (standaard http://localhost:4200).

Als je een bundel wil downloaden van deze server dan moet je de folder waar dat bestand in zit meegeven in het configuratie bestand van Angular.

Wanneer je het project dan start via de Angular CLI, wordt dat bestand ook meegenomen wanneer de applicatie gebuild wordt.

Stel je hebt een bundel genaamd testbundel in het project zitten, op de locatie “src/assets/testbundel” en je wilt die downloaden. Je moet dan de testbundel toevoegen aan de assets folder van de applicatie en het pad “src/assets/testbundel” toevoegen in het configuratie bestand van de Angular applicatie.

Wanneer de applicatie gestart is kan je een GET aanvraag sturen naar http://localhost:4200/assets/testbundel om die bundel te downloaden.

Het haalt de bundel dus op van zijn eigen server.

Bijlage 11: Evaluatie van bedrijfsmentor

Geachte bedrijfsmentor/opdrachtgever,

U heeft de afgelopen periode een student van onze opleiding bij uw bedrijf begeleid. Ik verneem graag wat uw indruk is van de student. Wilt u onderstaande vragen op dit formulier beantwoorden? Ik zal uw evaluatie ook delen met de examinatoren.

Student:	Ruben Deurloo
Periode:	1 September – 25 Maart
Bedrijf / organisatie:	CGI Nederland BV
Bedrijfsmentor / opdrachtgever:	Rens van der Meijs
Plaats:	Rotterdam
Datum:	18 maart 2021

1. **Heeft de student zich zelf snel en goed ingewerkt in het bedrijf en de uit te voeren afstudeeropdracht?**

Ruben was in staat zich snel in te werken ondanks de Pandemie. Het onboarding proces binnen CGI verliep soepel. Vervolgens kon Ruben snel aan de slag met zijn afstudeeropdracht die overigens snel duidelijk was. Al snel kwam hij met suggesties voor eventuele verbeteringen en handige toevoegingen.

2. **Hoe beoordeelt u de communicatieve vaardigheden van de student (in de samenwerking met collega's, in contacten met de opdrachtgever, bij mondelinge presentaties, schriftelijke rapportages)?**

Hoewel een groot deel van de werkzaamheden op afstand plaatsvond was de communicatie vrijwel altijd in orde. Ruben is altijd erg enthousiast om te vertellen waar hij mee bezig is en welke problemen hij heeft opgelost. Qua schriftelijke communicatie binnen het bedrijf moest ik hem in het begin vaak helpen. Hier merkte ik soms toch enige mate van onzekerheid.

3. Kunt u op basis van onderstaande items uw indruk geven van het functioneren van de student tijdens de uitvoering van de opdracht bij uw organistie?

- Verantwoordelijkheid goed / ~~voldoende / matig / onvoldoende~~
- Zelfstandigheid goed / ~~voldoende / matig / onvoldoende~~
- Planmatig werken ~~goed~~ / voldoende / ~~matig / onvoldoende~~
- Creativiteit ~~goed~~ / voldoende / ~~matig / onvoldoende~~
- Productiviteit goed / ~~voldoende / matig / onvoldoende~~
- Samenwerken met collega's goed / ~~voldoende / matig / onvoldoende~~
- Draagvlakontwikkeling goed / ~~voldoende / matig / onvoldoende~~
- Inspelen op bedrijfscultuur ~~goed~~ / voldoende / ~~matig / onvoldoende~~
- Initiatief nemen tot veranderen ~~goed~~ / voldoende / ~~matig / onvoldoende~~

4. In hoeverre matchen de kennis en vaardigheden van de student met uw verwachting van een bijna afgestudeerde HBO-ICT'er?

De match was goed. Gedurende zijn stage heeft Ruben erg zelfstandig gewerkt, mede doordat hij graag onderzoek doet en zelf met oplossingen komt wanneer hij vast zit. Ook kan hij zelf met creatieve oplossingen voor problemen komen en deze duidelijk overbrengen/bespreken tijdens onze wekelijkse meetings. Wanneer hij bezig is met zijn onderzoek merk je dat hij voldoende vakkennis heeft om ideeën en concepten aan elkaar te knopen. Daarnaast is hij productief genoeg en heeft hij tijdens iedere meeting voortgang in het project aan kunnen tonen.

Toch merkte ik dat in het begin het planmatig werken wat moeilijker ging. Waarschijnlijk omdat hij graag snel praktisch aan de slag wilde gaan. Over verloop van tijd ging Ruben naar mijn idee steeds planmatiger werken.

5. Kunt u de kwaliteit van de opgeleverde (tussen)producten aangeven aan de hand van onderstaande items:

a) In hoeverre heeft u gekregen wat is afgesproken (kwantiteit)?

Het verkregen product voldoet aan wat wij hebben afgesproken.

b) In hoeverre voldoet het (eind)product aan uw verwachtingen (functionaliteit)?

Qua functionaliteit voldoet het eindproduct ook aan onze verwachtingen.

c) Wat is de bruikbaarheid en onderhoudbaarheid hiervan (kwaliteit)?

Het eindproduct is inzetbaar voor het doel dat wij voor ogen hebben en zal waarschijnlijk niet in een productie context worden geplaatst. Op dit moment is de kwaliteit een iets minder belangrijkere factor dan de functionaliteit.

d) Wat gebeurt er met het opgeleverde (eind)product (toepasbaarheid)?

Het eindproduct dient als eerste aanzet en demonstratie van een service die CGI mogelijk wil gaan faciliteren aan klanten. Daarnaast geeft het eindproduct ons een extra mogelijkheid om nogmaals te brainstormen over het ontwerp nu we meer praktische kennis hebben opgebouwd.

e) Kunt u direct met het opgeleverde product aan de slag (implementeerbaarheid)?

Het opgeleverde product heeft op dit moment de benodigde flexibiliteit om verschillende soorten content (met name games) te tonen en te beheren. Dit is voor ons van belang om de eerste aanzet te implementeren/verwerken in het ontwerp proces.

6. Zijn er nog aspecten voor u van belang die nog niet aan de orde zijn geweest?

Nee.

Bedankt voor het invullen van dit formulier. U kunt het formulier rechtstreeks naar mij sturen per e-mail.

Met vriendelijke groet,

J.C. (Arjan) Mulder
Afstudeercoördinator opleiding HBO-ICT
j.c.mulder@hhs.nl