Koekoek

VPS Monitorings tool

Versie: 1.0
Auteur: Bastiaan Klein
Studentnummer: 10014845
Opleiding: Informatica
Bedrijf: 42
Afstudeerbegeleiders:
T.Cocx
E.P. Koenen
Bedrijfsbegeleider: R.A. Bor
Datum: 06-10-2014

Afstudeerverslag

VOORWOORD

Dit verslag is geschreven naar aanleiding van mijn afstudeerstage bij het bedrijf 42 voor mijn opleiding Informatica aan de Haagse Hogeschool in Zoetermeer. Bij het bedrijf 42 heb ik gewerkt aan een opdracht om een VPS monitorings tool te ontwikkelen. Ik hoop dat u door het lezen van dit verslag een goed beeld krijgt van mijn werkzaamheden die ik heb verricht voor deze afstudeeropdracht.

De tijd die ik heb gespendeerd bij het bedrijf 42 heb ik als erg prettig ervaren. Het moment dat ik daar binnenkwam voelde ik mij al thuis en werd ik heel hartelijk ontvangen. Dit zorgde ervoor dat ik met een positief gevoel begon aan mijn opdracht. Daarnaast werd ik goed begeleid door mijn bedrijfsbegeleider.

Hierbij wil ik iedereen bedanken die mij op welke manier dan ook, heeft geholpen bij deze afstudeerstage en het opstellen van dit verslag. Mijn speciale dank gaat uit naar Robert Bor (CFO en begeleider), Aibert Riksen (opdrachtgever), Tim Cocx (docentbegeleider) en natuurlijk alle andere collega’s en docenten die mij geholpen hebben.

INHOUDSOPGAVE

WOORDENLIJST

Sprint	Korte periode van één tot vier weken.
Confluence	Commerciële wiki software van de firma Atlassian
JSON	Json is een deelverzameling van de programmeertaal JavaScript. Het wordt gebruikt voor het uitwisselen van datastructuren.
DBMS	“Met een databasemanagementsysteem (vaak afgekort tot DBMS) wordt het systeem aangeduid dat als database opgeslagen gegevens ontsluit, bewaakt en beheert.” [1]
Bean	Een bean is een singleton instance van de klasse die de bean definieert. Verder zorgt het Spring framework ervoor dat de bean alle faciliteiten nodig heeft om te kunnen werken.
Dependency	Een dependency is een framework of een bepaalde plugin/library waar de applicatie gebruik van maakt voor bepaalde functionaliteiten.
Scope	“Scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events.”[2]
API	“Een verzameling programmeeropdrachten (vaak ook als interfaces aangeduid) die de functies van een programma aanroepen. Andere programma's kunnen de API van een programma gebruiken om diensten te vragen of om met dat programma te communiceren.”[3]

[1] Quotatie bron: http://nl.wikipedia.org/wiki/Databasemanagementsysteem
[2] Quotatie bron: https://docs.angularjs.org/guide/scope
[3] Quotatie bron: http://www.computerwoorden.nl/direct--18617--API.htm

INLEIDING

Gedurende het laatste jaar van de studie Informatica aan de Haagse Hogeschool is het noodzakelijk om een afstudeerstage te doen voor een periode van 18 weken. Tijdens het zoeken van een afstudeerstage ben ik bij het bedrijf 42 terecht gekomen, dit ook na aanbevelingen van andere medestudenten. Tijdens het afstuderen wordt er van je verwacht dat je aan het einde een eindverslag inlevert dat beschrijft welke werkzaamheden je verricht hebt. Hier komt onder andere in te staan hoe en waarom je deze werkzaamheden zo hebt gedaan, dit is ook de aanleiding tot het schrijven van dit verslag.

Bij het bedrijf 42 heb ik mij bezig gehouden met het verder ontwikkelen en het verbeteren van de kwaliteit van de VPS monitorings tool. Dit systeem moet de verschillende virtuele servers die het bedrijf heeft monitoren. Hierdoor kan het bedrijf sneller en makkelijker zien hoe de servers draaien en sneller beoordelen of er een bedreiging is waar het bedrijf voor moet ingrijpen.

Met dit verslag hoop ik de beoordelaars genoeg te kunnen informeren over mijn bezigheden zodat ze mij een goede evaluatie kunnen geven. Ik wil dit bereiken door per sprint aan te geven wat ik heb gedaan en hier een evaluatie van te geven.

In het eerste hoofdstuk beschrijf ik het bedrijf en mijn positie binnen het bedrijf. In het tweede hoofdstuk zal ik mijn opdracht beschrijven. Hierna zal ik per sprint vertellen wat ik heb gedaan en hoe ik dit heb gedaan. Hierin zal een evaluatie komen te staan van mijn werkzaamheden. Hierna zal mijn evaluatie geven over het product. Als laatste zal ik elke beroepstaak langs gaan en uitleggen op welke wijze ik aan mijn beroepstaken heb voldaan.

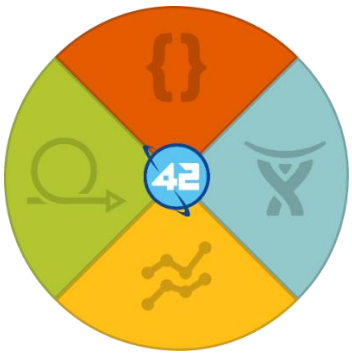
HET BEDRIJF 42

"42, opgericht in 2003 als een in JAVA gespecialiseerd software bedrijf en in 2009 samengegaan met Kensas, een expert in maatwerk oplossingen voor complexe vraagstukken. Hierdoor ontstond er een ICT organisatie met zo'n 35 eigen medewerkers met een hoofdkantoor in Zoetermeer." [4] Hiernaast ontwikkelt het bedrijf 42 sinds kort IOS apps.

Tijdens het ontwikkelen richten ze zich vooral op het ontwikkelen van hoogwaardige oplossingen en diensten. Zo leveren ze ook diensten die het totale ontwikkelingstraject ondersteunen, denk hierbij aan:

- analyse gebruikerswensen
- opstellen functioneel ontwerp
- technisch ontwerp en realisatie
- support en beheer.

De werkwijze van 42 kan het best uitgelegd worden door middel van de onderstaande cirkel. [5]



- Oranje: Technologie.
- Blauw: Tools
- Geel: Kwaliteit
- Groen: Proces

Hieronder zal worden uitgelegd wat de kleuren inhouden.

Technologie. 42 maakt gebruik van technologieën die hun waarde hebben bewezen en breed gedragen worden door de markt. Het voordeel hier van is dat de betrouwbaarheid van de applicaties verbeterd wordt. Als iets zijn waarde al bewezen heeft, betekent dat er dus minder kans is op bugs dan bij nieuwe technologieën.

Tools. 42 gebruikt tools die het ontwikkelproces ondersteunen en de kwaliteit van de software verbeteren. Zo wordt bijvoorbeeld heel de suite van het bedrijf Atlassian gebruikt. Deze suite bevat alles wat nodig is. Denk aan een ticket systeem om alle tickets op orde te houden. Een Git repository voor versie beheer en tools die kunnen meten hoeveel code je getest hebt van een applicatie. Hiernaast zijn er ook tools om te kijken of jouw code wel voldoet aan de standaarden die gesteld zijn binnen het bedrijf.

[4] Quotatie bron: http://www.42.nl/display/maatwerk/Over+42

[5] 42 kwadrant: http://42.nl/display/maatwerk/Werkwijze

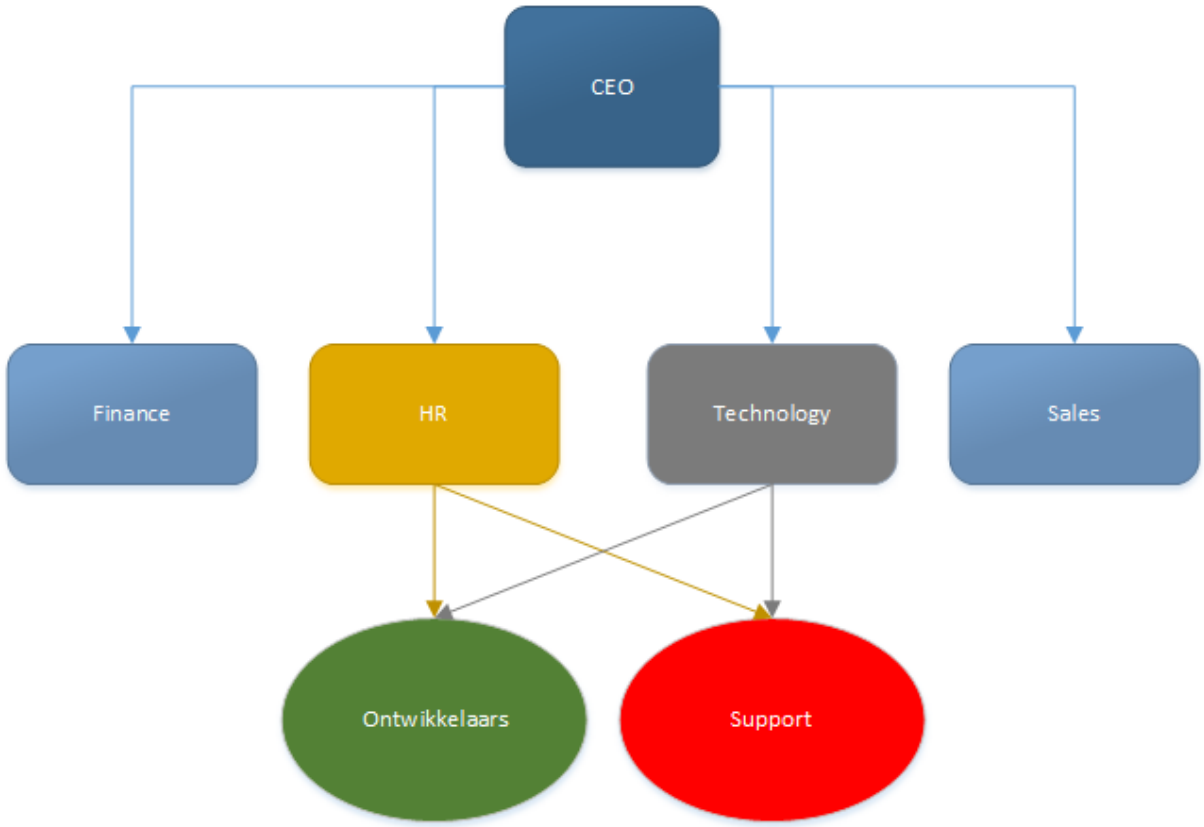
Kwaliteit. 42 vindt het belangrijk dat ze een project oplevert dat van voldoende kwaliteit is voor de klant, dit wordt mede teweeg gebracht door de eerder genoemde kleuren.

Proces. 42 zal aan het begin van een project met de klant bepalen welke iteratieve ontwikkelmethode het beste past bij het project. Dit betekent dat niet altijd hetzelfde ontwikkelproces wordt gebruikt. Hierdoor kan er een optimaal resultaat worden geleverd.

Hiernaast levert 42 ook nog diensten op het gebied van audits en consultancy. Met consultancy helpt 42 de klant om bedrijfsprocessen te analyseren en oplossingen vorm te geven. Maar ook biedt 42 ondersteuning voor het opstellen van functionele “requirements” bijvoorbeeld bij pakketselectie.

Mocht een project volledig ontsporen dan kan 42 ook iets betekenen in de vorm van audits. Hiermee word de kwaliteit en het ontwikkelingstraject geanalyseerd. Dit kan op verschillende niveaus: van een quick scan tot een diepgravende scan. Wanneer dit gedaan is zal een herziend projectvoorstel aangeleverd worden. 42 kan zelfs het project overnemen om alles toch tot een goed einde te brengen.

De structuur binnen 42 is te zien in figuur 1.



Figuur 1 Organogram 42

Mijn stageopdracht volbracht ik bij de ontwikkelaars.

DE OPDRACHT

Probleemstelling

Het bedrijf 42 heeft op dit moment bij verschillende providers remote servers draaien, de meeste hiervan staan bij CloudVPS. Deze servers zijn onder andere te monitoren via een monitoringsprogramma van de aanbieder zelf. Enkel voldoen deze niet aan de wensen van het bedrijf, mede door vele bugs, denk hierbij aan diverse bugs en user interface gebreken. Hierdoor is het lastig om de juiste data af te lezen, er kan dus ook niet goed gekeken worden of een server nog goed draait en wanneer problemen ontstaan. Hiernaast laten deze monitoren ook niet alle gegevens zien die door 42 gewenst zijn, bijvoorbeeld het versienummer van verschillende applicaties. Tenslotte wil 42 niet vast zitten aan één hostingprovider. In de huidige situatie betekent dit dat er automatisch gebruik moet worden gemaakt van meerdere monitoringtools, dit is onhandig en onoverzichtelijk.

Doelstelling

De doelstelling van de opdracht is om de monitor die al deels ontwikkeld was door twee mede studenten van de Haagse Hogeschool door te ontwikkelen naar een bruikbaar product. Ook zal de kwaliteit hiervan verbeterd worden waar dit moet. Ik zal dus niet alleen nieuwe functies toevoegen, maar ook bestaande functies analyseren en verbeteren.

TOOLS

Voor deze opdracht is de ontwikkelingsmethode SCRUM gekozen. Deze is gekozen omdat dit ten eerste de standaard methodiek is die gebruikt wordt voor interne projecten binnen 42. Daarnaast is SCRUM gekozen omdat de wensen nog niet duidelijk zijn en er nog veel kan veranderen binnen het project. Nu kan er dus met SCRUM per sprint gekeken worden wat er veranderd moet worden aan de applicatie en wat de nieuwe wensen zijn. Dit houdt het flexibel voor de opdrachtgever. Een sprint binnen dit project zal bestaan uit ongeveer tien werkdagen, dus twee weken. Twee weken is de standaard voor SCRUM bij interne projecten. Een week vindt het bedrijf te weinig, in een week kan je niet veel progressie maken. Als een sprint langer duurt dan twee weken kan de schade groot zijn als je verkeerd bezig bent, het kost dan veel tijd om dit terug te draaien. Aan het einde van elke sprint zal er een demo gegeven worden met een werkende versie van de applicatie. Tevens zal er een retrospectieve worden gehouden en wordt er een sprint planning opgeleverd voor de volgende sprint.

Hieronder volgt een lijst met welke technieken aan bod zullen komen om de front- en back-end van de applicatie te ontwikkelen:

Front-end	Back-end
AngularJS	Java
HTML5	Spring
CSS3	JPA
Bootstrap	Hibernate
	Maven
	Junit
	JMockit
	PostgreSQL

Front-end

Diverse tools en technieken worden gebruikt om de front-end te ontwikkelen. AngularJS is een javascript front-end framework. Met AngularJS kan je heel eenvoudig dynamische front-end applicaties maken. Eenvoudiger dan met standaard javascript en zelfs nog eenvoudiger dan met jQuery.

HTML zal gebruikt worden om de pagina's structuur te geven. CSS zal gebruikt worden om de pagina te stijlen. Daarnaast zal ook gebruik gemaakt worden van Bootstrap. Bootstrap is een front-end framework dat het mogelijk maakt om responsive web applicaties te maken. Dit betekent dus dat je een applicatie kan maken die er goed uitziet op zowel een PC monitor als op een mobiele telefoon. Tevens levert Bootstrap heel veel standaard componenten waardoor iedereen een web applicatie kan maken die er goed uit ziet. Denk hierbij aan formulieren, tabellen, headers enz. Natuurlijk kan HTML dit ook, maar bootstrap zorgt ervoor dat er gelijk een ingebouwde styling is zodat het er gelijk mooi uit ziet.

AngularJS werd gebruikt toen ik de applicatie opgeleverd kreeg. Dit omdat het en eis was van de opdrachtgever, hierom ben ik hier verder mee gaan werken. De keuze om gebruik te maken van Bootstrap is gemaakt in overleg met een front-end programmeur, hierover zult u later in dit verslag meer lezen.

Back-end

Ook voor alle back-end technieken geldt dat deze al gebruikt werden toen ik de applicatie aangeleverd kreeg. Dit zijn ook de standaard technieken bij het bedrijf en hier zal ik niet van af wijken. Hieronder zal ik uitleggen waarom deze technieken gebruikt worden.

Aangezien de standaard programmeertaal binnen 42 Java is, ligt hier ook hun expertise. Dit is dus ook de reden dat ik Java zal gaan gebruiken voor deze opdracht. Om een goede web service applicatie te maken zal ik gebruik maken van het Spring framework. De standaard binnen 42 voor web services. Het Spring framework is een framework om RESTful webservices te maken. Het is ook mogelijk om te kiezen voor een SOAP webservice. SOAP is echter zwaarder dan REST en heeft veel meer standaarden en plugins/addons. Al die extra functionaliteit is niet nodig en is er dus geen reden

om voor SOAP te kiezen. Binnen 42 wordt om dezelfde redenen nooit gebruik gemaakt van een SOAP webservice.

Communicatie met de database wordt gedaan door middel van JPA en de ORM mapper Hibernate. Deze constructie neemt veel werk van de developer uit handen, zoals standaard CRUD query's. Tevens maakt dit het mogelijk om te werken met objecten in plaats van resultsets, dit is beter omdat type safety dan gelijk afgevangen wordt. Ook deze twee componenten waren al in het project gebruikt en zal ik ook mee verder gaan.

Maven zal gebruikt gaan worden als dependency manager. Dit is een makkelijke tool om de dependencies bij te houden. Hierdoor hoeft een developer niet handmatig alle frameworks en plugins/addons in een project te zetten en worden dependencies automatisch geüpdate. Als laatste wordt als database PostgreSQL gebruikt. Dit is een relationeel DBMS. De reden waarom deze wordt gebruikt is wederom omdat dit de standaard is voor interne projecten binnen 42. Dit heeft 42 gedaan zodat de medewerkers van de support afdeling niet vier database management systemen uit het hoofd hoeven te leren. De focus ligt op PostgreSQL om de kennis te vergaten.

Voor het testen zal JUnit gebruikt worden. Dit is de standaard binnen Java voor unit testen. Dit wordt wel gecombineerd met JMockit. JMockit maakt het mogelijk om het gedrag van klasse na te bootsen. JMockit wordt op verzoek van de opdrachtgever gebruikt.

WAT LIGT ER NU

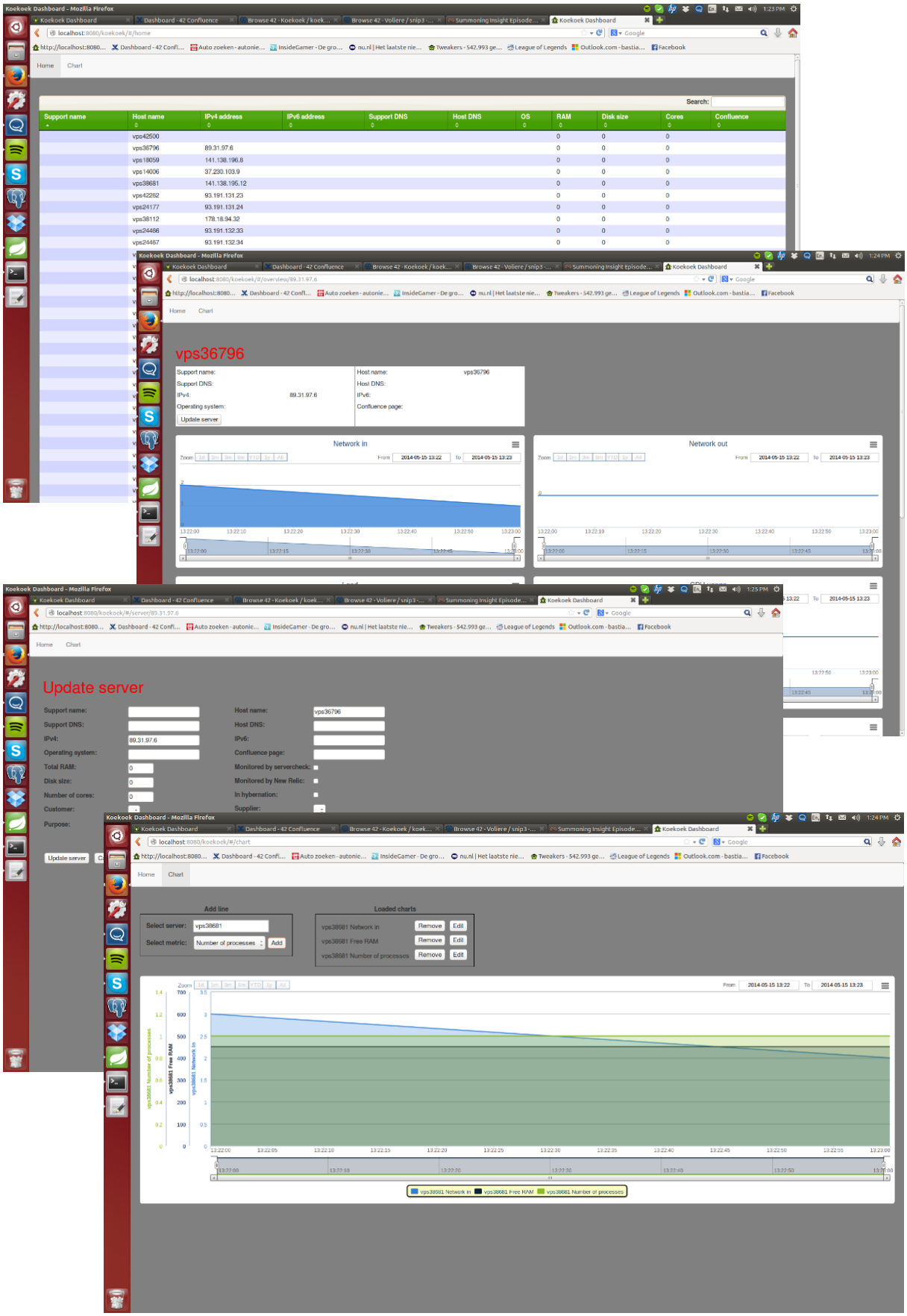
Zoals eerder gezegd is deze applicatie al voor een deel ontwikkeld door mede studenten. Dit betekent dat er al een stuk ligt van de applicatie. Op dit moment is er een front en back-end van de applicatie. Deze maken gebruik van de eerdergenoemde tools. Tevens heeft de applicatie de volgende functionaliteit:

- Overzicht bekijken met alle servers. In dit overzicht kan verschillende informatie gezien worden.
- Informatie bekijken van een server. Dit bestaat uit alle beschikbare server informatie en geeft meer informatie dan in de overzicht tabel.
- Historische meetgegevens bekijken van een server, zoals de CPU,load of aantal processen in een bepaalde periode.
- Informatie van een server updaten, bijvoorbeeld hoeveel cores de server heeft of hoe groot de harde schijf is.
- Verschillende meetgegevens van verschillende servers met elkaar vergelijken.

Tevens is er in Java een zogenoemde 'n2 API' gemaakt. Deze API haalt elke minuut gegevens op van de servers die gehost zijn het bedrijf CloudVPS. Deze api haalt slechts een maximaal aantal van 52 servers op, veroorzaakt door een fout aan de kant van CloudVPS.

In het bestaande deel van de applicatie was nog geen enkele functionaliteit geheel af, wel moest nog verbeterd of uitgebreid worden. Een voorbeeld hiervan is dat een server een klant heeft, echter was er geen mogelijkheid om een klant aan te maken of de koppeling tussen de twee te regelen. Het

enige onderdeel dat gereed was, was de ‘n2 API’. Op dit moment ziet de applicatie er voor de gebruiker als volgt uit:



Resultaat

Uiteindelijk zal er een werkende applicatie moeten komen die gebruikt wordt om diverse remote servers van het bedrijf 42 te monitoren. Deze monitor moet om kunnen gaan met de verschillende metingen van de servers die het ontvangt of ophaalt. Deze metingen moeten vervolgens per server genormaliseerd worden naar een standaard die voor alle servers geldt. Vervolgens kunnen deze metingen bekeken/vergeleken en geanalyseerd worden.

Tevens moet deze applicatie het huidige administratie systeem van het bedrijf vervangen. Het bedrijf wilt dat alle informatie van de servers op één plek zichtbaar wordt. Het systeem moet dus alles kunnen wat het huidige administratieve systeem ook kan. Verder moet de front-end zo gemaakt worden dat het in lijn kwam met de huidige interne applicaties.

INITIALISATIE

Met de initialisatie fase wordt de eerste week van het afstuderen bedoeld. Deze week heb ik gebruikt om overal goed rond te snuffelen en bekend te worden met het bedrijf en de gewoontes. Zo moest ik in het begin erg wennen aan het besturingssysteem Linux. Deze heb ik leren gebruiken, omdat dit dichter staat bij de omgevingen waar ik mee zou werken. Ook was deze week een goed moment om door te code heen te lopen en om te kijken of ik het allemaal begreep.

Zoals eerder beschreven was de applicatie geschreven in Java voor de back-end en AngularJS voor de front-end. Met Java had ik op school ervaring opgedaan, maar deze kennis was erg weggezakt. Met AngularJS had ik nog geen ervaring. Hier heb ik mijzelf in de eerste week in verdiept om mijzelf hier bekend mee te maken.

Deze eerste week was ook een goed moment om te beginnen met mijn plan van aanpak. Dit heb ik gedaan zodat ik mijn scope goed kon definiëren en duidelijk mijn taken kon afschermen. Zo heb ik de verwarring van wat van mij verwacht wordt geminimaliseerd en de verwachting van wat ik op moet leveren gemaximaliseerd. Ik heb wel rekening gehouden met de veranderingen van de wensen van de opdrachtgever in de loop van de tijd, dit kan immers resulteren in nieuwe functionaliteiten die erbij komen. Vandaar is, zoals eerder gezegd, voor SCRUM gekozen. Verder was het plan van aanpak maken goed te doen, omdat ik veel uit mijn opdrachtomschrijving kon halen. Het maken van de planning was wel een moeilijk punt. Het project is nog vol in leven en de wensen van de opdrachtgever kunnen veranderen. Vandaar de keuze om per sprint te bekijken wat de wensen zijn van de opdrachtgever. Hierna wordt per sprint een planning gemaakt.

Natuurlijk was er ook een gesprek gepland om te bespreken wat er van mij verwacht wordt en om nog een keer van de opdrachtgever en mijn begeleider te horen wat de applicatie uiteindelijk moet doen. Hierdoor was dit gesprek vooral oriënterend voor mij. Tevens zijn er doelen gesteld voor de eerste sprints.

- Sprint 1
    - Code review
    - Interface redesign
- Sprint 2
    - Test coverage verhogen
    - Database model beoordelen

Het belangrijkste was dat er zo snel mogelijk iets naar productie toe moest. Natuurlijk komt dit wel eerst op een test omgeving, omdat het project dan veel meer zou gaat leven voor iedereen binnen het bedrijf. Hierdoor krijg ik ook meer nuttige feedback van andere mensen die zullen gaan werken met het product.

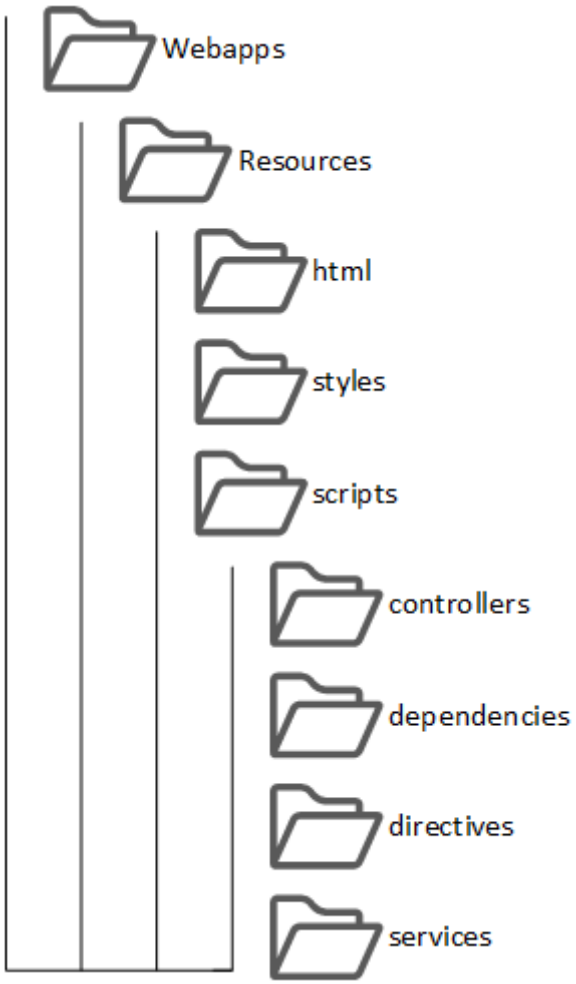
Deze doelen zijn opgesteld in overleg met mijn begeleider en specifiek opgesteld om mijzelf in de werking van de applicatie te krijgen. Ook kan ik hierdoor gelijk mijn kennis op dit project reflecteren. Hierdoor zal gelijk de kwaliteit verbeterd worden, één van mijn taken voor dit project.

Tevens was er een document voor mij achter gelaten waar alles in stond over de applicatie, zoals deze tot nu toe opgeleverd is. Denk hierbij aan documentatie over de applicatie zelf, de database, en de werking van bepaalde stukken code. Dit bleek erg nuttig te zijn en ik kon mijzelf nog beter inlezen in de applicatie en de werking ervan. Tevens stonden er mogelijke verbeterpunten in die de vorige developers al hadden bedacht. Deze verbeterpunten heb ik ook geanalyseerd om te kijken welke verbeterpunten goed waren en welke niet. Dit zodat ik later wist welke ik zou kunnen verwerken in mijn applicatie.

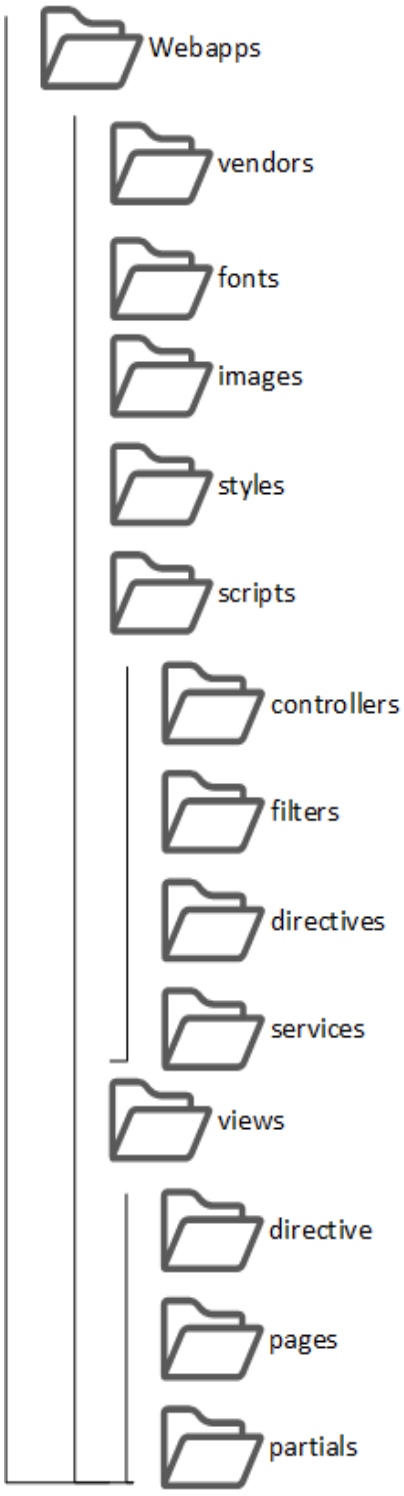
Zoals eerder beschreven was één van de doelen een interface her design. Ik heb het advies opgevolgd om hiervoor te praten met een front-end developer. Het eerste wat de front-end developer opviel was de mappen structuur van de front-end. Dit was niet de standaard, die gebruikt werd binnen interne projecten van 42. Hij adviseerde mij dan ook om deze gelijk te veranderen om zodoende alles in één lijn te trekken met de rest van de applicaties.

Als de applicatie in één lijn ligt met de rest van de interne applicaties van 42, is het makkelijker voor andere ontwikkelaars om te werken met deze applicatie. Ze weten gelijk waar ze moeten zoeken om iets aan te passen zonder eerst veel tijd te verspillen. Ik heb dit advies opgevolgd en de applicatiestructuur veranderd.

Voor de rest van de front-end hadden we afgesproken om dit aan het begin van sprint één te bespreken, zodat de front-end developer ook wat meer tijd had om het te bekijken.



Figuur 3 Oude map structuur



Figuur 2 Nieuwe map structuur

SPRINT 1, CODEREVIEW EN BOOTSTRAP

Het eerste dat op de agenda stond in sprint één was de code review van de huidige code. Dit was een goede kans om gelijk in de code te duiken en om te kijken hoe alles precies in elkaar zit. Hierdoor kon ik mijn kennis van de eerste week uitbreiden. En tevens kreeg ik de kans om mijn eigen kennis te reflecteren tegen die van andere medestudenten. Sommige code vond ik niet goed en zou ik graag gelijk veranderen. Hierna zou een redesign van de front-end op de planning staan. Ook stond in deze sprint stond de rules compliance en test coverage op de planning. Hierover volgt later meer.

CODE REVIEW

Front-end

Zoals eerder gezegd, wordt bootstrap gebruikt voor dit project. Dit was al geïmplementeerd in het project. Het viel mij echter op dat dit niet goed gebruikt werd. Bootstrap bestaat uit twee gedeeltes. Het eerste gedeelte is de CSS om de standaard opmaak mooier te maken, het tweede gedeelte is de javascript. Dit laatste deel van bootstrap wordt gebruikt om bijvoorbeeld eenvoudig mooie dropdown menus te maken. Het kan ook gebruikt worden voor bijvoorbeeld tooltips. In dit project was alleen het CSS gedeelte geïmplementeerd en niet het javascript gedeelte. Naar mijn mening werd de CSS ook niet goed gebruikt. Over het uiterlijk later meer.

Het tweede punt dat mij op viel is dat er veel data opgehaald werd in de controllers van AngularJS. Dit betekent dat veel http calls naar de back-end hierin werden uitgevoerd. Het is gebruikelijk volgens AngularJS om dit in services te doen in plaats van in controllers. Dit is ook het gene wat de front-end developer aan mij verteld heeft.

Tijdens mijn derde jaars stage heb ik bij een bedrijf gewerkt waar alles flexibel gemaakt moest worden, bijvoorbeeld een extra veld in de front-end moest erbij komen door alleen maar een rij aan de database toe te voegen. Hierdoor zat ik nog in die mind set, zonder goed te overleggen of dit wel nodig was. Dit heeft ook invloed gehad op mijn code-review. Om deze reden wilde ik bijvoorbeeld het formulier om server data up te daten flexibel maken. Daarnaast ook de tabel waar je een overzicht hebt van alle server informatie, omdat alles nog hard coded in html stond. Kortom, ik wilde een systeem waar je bijvoorbeeld alleen maar een rij aan de database toe hoefde te voegen.

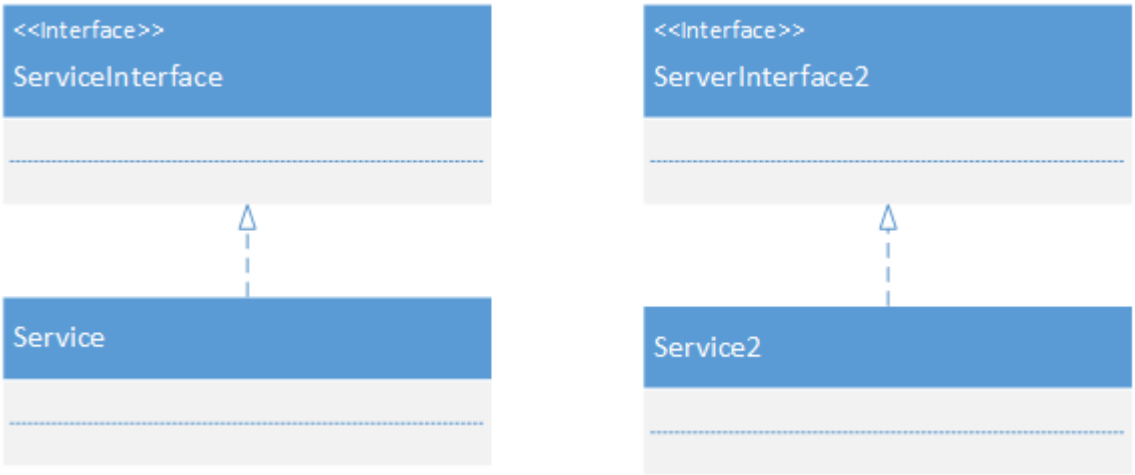
Het laatste wat ik opvallend vond in de front-end was een volledig zelf opgebouwde JSON string dat een object representeerde en dit in combinatie met vier if statements. Aan de hand van een bepaalde conditie werd er een verschillend server object gestuurt, terwijl het juist de bedoeling is dat je gewoon een object meestuurt. Hierna moet je in de back-end kijken wat je met het object doet. Op de volgende pagina een voorbeeld hoe een server object opgebouwd werd.

Opbouwen van server object in de oude situatie
data: JSON.stringify({"id": $scope.server.id,"supportName": $scope.server.supportName, "hostName": $scope.server.hostName, "ipv4": $scope.server.ipv4, "ipv6": $scope.server.ipv6, "supportDns": $scope.server.supportDns, "hostDns": $scope.server.hostDns, "operatingSystem": $scope.server.operatingSystem, "totalRam": $scope.server.totalRam, "totalDiskSize": $scope.server.totalDiskSize, "cores": $scope.server.cores, "confluencePageUrl": $scope.server.confluencePageUrl, "monitoredByServercheck": $scope.server.monitoredByServercheck, "inHybernation": $scope.server.inHybernation, "purpose": $scope.server.purpose, "monitoredByNewrelic": $scope.server.monitoredByNewrelic, "remarks": $scope.server.remarks, "supplier": {"id": $scope.server.supplier.id, "name": $scope.server.supplier.name, "code": $scope.server.supplier.code}})}

Back-end

Zoals eerder gezegd wordt er bij 42 gebruikt gemaakt van de Atlassian suite. Deze suite bevat een programma dat kan kijken hoeveel van jouw Java code getest is. Ook wordt bekeken hoe de code voldoet aan de standaarden die gesteld zijn door het bedrijf (Rules Compliance). De naam van dit programma is 'Sonar'. Hiernaast draait het programma 'Bamboo'. Deze build automatisch het project en voert de testen uit. Sonar haalt hier dan ook zijn gegevens uit. 42 heeft als wens om de test coverage boven de 80% te hebben en de rules compliance boven de 95%. Wat mij gelijk opviel toen ik in Sonar ging kijken was dat de test coverage 38% was, dit was een uitdaging om dit op het gewenste niveau te krijgen. De rules compliance was ongeveer 80%, deze uitdaging was dus minder groot.

Bij de beoordeling van de code van de back-end constateerde ik zaken die ik anders zou doen dan in de huidige situatie. Zo was bijvoorbeeld de naamgeving inconsistent. In de ene klasse heette een methode 'edit' en in een andere klasse 'editServer'. Het heeft de voorkeur om één van de twee manieren te kiezen. Ook waren er veel interfaces die naar mijn mening niet nodig waren, zoals bij de services. De structuur ziet er als volgt uit.



Elke service had zijn eigen interface en dit heeft volgens mij geen nut. Je forceert niks en geeft alleen maar meer werk en onderhoud. Als elke service dezelfde interface implementeert, heeft het toegevoegde waarde voor mijn gevoel. Zo waren er aan de andere kant plekken waar interfaces juist gebruikt konden worden terwijl dit niet gedaan werd.

FRONT-END

Tijdens deze sprint moest het uiterlijk van de front-end aangepast worden. Voor het veranderen van het uiterlijk van de front-end heb ik gesproken met een front-end ontwikkelaar bij 42. Tijdens dit gesprek werd gevraagd wat er op het moment gebruikt wordt voor de front-end en wat mijn ideeën waren. Helaas ben ik geen front-end expert, hierdoor kon ik niet veel input geven tijdens het gesprek. Ik vind het wel leuk om stukken te ontwerpen, hoewel ik op het moment van het gesprek nog geen idee had wat ik met de front-end aan moest. Het enige wat ik wist was dat het een redesign nodig had, omdat de opdrachtgever het niet mooi vond en er veel interface design niet correct waren. Denk hierbij aan zaken zoals de begin focus op een veld. Ook wist ik dat Bootstrap niet correct was toegepast. De front-end developer vond het goed dat er aan Bootstrap begonnen was. De front-end developer was ook bezig met een interne applicatie waarin Bootstrap geïmplementeerd zou gaan worden. Hij adviseerde mij dan ook om hiermee verder te gaan. Dit advies heb ik dan ook opgevolgd zonder tegenspraak. Dit heb ik gedaan omdat het uiteindelijke doel was om elke interne applicatie er, logischerwijs, gelijk uit te laten zien.

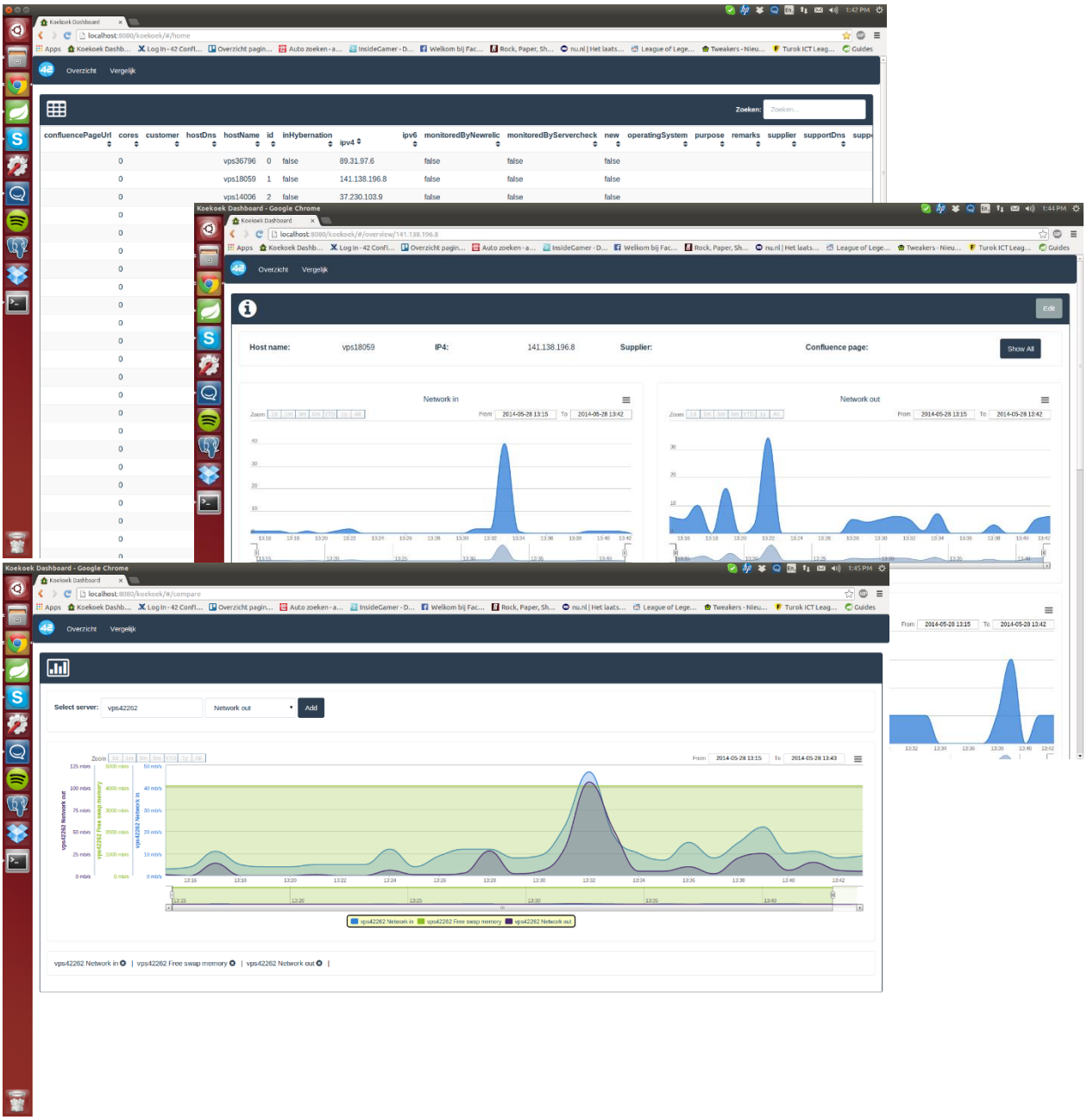
Op dit gebied heeft bootstrap ook een voordeel. Met bootstrap kan je namelijk een thema erover heen zetten. Zo kan je met het vervangen van één CSS bestand je applicatie er compleet anders uit laten zien. De front-end ontwikkelaar ook bezig met een thema voor 42 voor interne applicaties van 42. Ik ben na dit gesprek ook aan de slag gegaan om bootstrap goed toe te passen. Dit betekent het gebruik maken van zowel de javascript als de CSS. Maar ook met het toepassen van het grid systeem van bootstrap. Het grid systeem zorgt ervoor dat je applicatie responsive wordt. Het systeem werkt met rijen waar maximaal twaalf kolommen in verwerkt zijn. Zo kan je elk element een breedte geven van één t/m twaalf kolommen. De Bootstrap CSS bepaalt dan zelf de grootte en positie aan de hand van je scherm grootte.[6]

Als laatste werd mij gevraagd om de overzicht tabel te vervangen door een tabel die via AngularJS is opgebouwd, voorheen opgebouwd met een jQuery plugin. Maar het was niet de bedoeling dat ik angularJS en jQuery door elkaar ging gebruiken. Bij 42 hebben ze liever dat je één techniek gebruikt, omdat er anders de mogelijkheid bestaat dat ze elkaar gaan tegenwerken.

[6] Bootstrap grid: http://getbootstrap.com/css/#grid

De eerste wens tijdens het oriëntatie gesprek waren ‘floating table headers’. Dit betekent dat de table header mee scrollt met het scherm. Dus als een tabel groter is dan de pagina scrollt die netjes mee. Eerst had ik dit geprobeerd met standaard CSS. Dit is mogelijk, maar dit werkte niet goed en de tabel kwam er hierdoor heel vreemd uit te zien. Of hij was te klein of niet goed uitgelijnd. Vervolgens heb ik na advies van de front-end developer gezocht naar een AngularJS directive. Een directive is een stuk code dat je op een DOM element kan binden. Dit kan als een attribuut, element naam of als CSS klasse. Wat de compiler van AngularJS vervolgens doet is het gedrag dat jij hebt gespecificeerd in je code binden aan het DOM element of het kan zelfs heel het DOM element vervangen. Ik heb een directive kunnen vinden, alleen deze werkte niet 100% naar behoren. Dit omdat de alignement van de header en body niet goed overgenomen werd. Aangezien hier te veel tijd in ging zitten heb ik voor nu even laten zitten.

Hieronder kunt u zien hoe de front-end er uit zag na de redesign.



Tevens moest ik kijken wat er allemaal veranderd kon worden aan de front-end en de keuzes hiervan noteren. Deze zou ik vervolgens bespreken tijdens de sprint demo. Nu ben ik zoals eerder gezegd geen front-end design expert waardoor ik ook niet zoveel kon bedenken. Wat ik wel heb kunnen bedenken zijn verschillende opties voor de overzicht tabel. Want deze tabel laat nu te weinig informatie zien. De wens was om op deze pagina alle informatie te kunnen zien van servers. Op dit moment zie je alleen informatie over de server zelf, maar niets over de versies of de laatste meting die ze binnen hebben gekregen. Dus om al deze data te laten zien heb ik drie opties bedacht:

- **Alles in één tabel.** Dit zou mogelijk zijn door de tabel een verticale en horizontale scrollbar te geven. Dit lijkt mij persoonlijk de minst aantrekkelijke optie.
- **Verschillende tabellen voor verschillende groepen data.** Deze optie is zeker één die zou kunnen werken. Hier heb je verschillende tabellen. Deze tabellen tonen informatie dat bij elkaar hoort. Dus één met alle server informatie. De andere server versies, enz.
- **Een checkbox lijst maken.** Hiermee maak je een lijst maken waarin je kan aanvinken welke informatie je wel en niet in de tabel wilt zien op dat moment.

Mijn voorkeur gaat uit naar optie twee, de optie met verschillende tabellen voor verschillende groepen data. Bij deze optie heb je niet een te grote onoverzichtelijke tabel en ook niet een irritante checkbox lijst. Het grootste pluspunt aan deze optie is de gebruiksvriendelijkheid. Met één actie kan gelijk een andere set data gezien worden. Hierdoor lijkt mij optie twee het beste. De opdrachtgever heeft natuurlijk de doorslaggevende stem bij de keuze tussen de opties. Tenslotte bedacht ik om het updaten van de server informatie naar een popup te verplaatsen. Dit zou weer een pagina schelen om naar toe te navigeren.

Reflectie

Van mijzelf weet ik dat het mij moeite kost om in te komen in een nieuwe programmeertaal of een programmeertaal op te pakken waar ik al een tijd niet meer in geprogrammeerd heb. Dit was ook nu het geval waardoor het werk langzamer ging dan dat ik zelf had gehoopt.

Tevens ben ik te lang bezig geweest met bepaalde front-end stukken die niet goed lukte. Bijvoorbeeld de floating table headers. Hier heb ik ongeveer anderhalve dag aan verloren. Waar ik eigenlijk had moeten beslissen om eerder verder te gaan met testen wat ook moest gebeuren. Maar ook met andere interface componenten die dan weer net niet goed stonden.

Een ander doel van deze sprint dat ik de test coverage en rules compliance omhoog zou halen naar een meer acceptabeler niveau. Helaas is dit niet gelukt, omdat ik teveel met de front-end bezig was en is dit dus verschoven naar de volgende sprint.

Hieruit blijkt dat ik minder bezig moet zijn met de minder belangrijkere taken, zoals een knopje dat niet goed staat, en dus meer moet focussen op de andere taken die nog moeten gebeuren tijdens een sprint. Gelukkig was dit pas de eerste sprint en was er nog genoeg tijd en ruimte om te leren.

SPRINT 2, TESTEN EN DE NIEUWE FRONT-END

Na de sprint demo van sprint één zijn er een paar doelen opgesteld voor deze sprint. Deze doelen bestonden uit:

- Het verbeteren van de test coverage en rules compliance, omdat ik hier vorige sprint niet aan toe kwam.
- De front-end code verbeteren, zodat het overzichtelijker werd en in de nieuwe map structuur paste.
- het database model beoordelen om te kijken hoe dat misschien verbeterd zou kunnen worden.
- Nadenken over de applicatie architectuur.

TESTEN EN RULES COMPLIANCE

Ik ben deze sprint begonnen met het omhoog brengen van de test coverage en rules compliance. Zoals eerder gezegd kon ik dan wat meer in de back-end van de applicatie duiken en deze zo beter onder de knie krijgen. Eerder heb ik ook verteld dat de test coverage wordt gemeten door het programma Sonar. Daarvoor moet je elke keer je code committen naar je repository om vervolgens in Bamboo het project builden. Dit is tijd intensief. Dus heb ik na navraag bij een andere afstudeerder het programma ‘Clover’ geïnstalleerd. Dit programma doet hetzelfde als Sonar met als verschil dat niet eerst je code hoeft te committen naar de repository.

De manier waarop bij 42 getest wordt, is door middel van unit tests. Mijn kennis van unit testen was ver weg gezakt, omdat wij hier niet zoveel les in krijgen op school. De laatste keer dat ik dit gedaan had was ongeveer twee en een half jaar geleden. Om deze reden ben ik dus ook begonnen met de makkelijkere tests. Hiermee bedoel ik het testen van de setters en getters van de models. Het enige dat je bij deze testen doet is een object aanmaken en de setters aanroepen. Vervolgens kijk je of de waarde die geset is overeen komt met de waarde die je ophaalt door middel van de gets.

Later bedacht ik dat het makkelijker zou zijn om een hele cyclus te testen, hierdoor pak je ook gelijk meer mee van de code. Met een cyclus bedoel ik het starten met een http call en eindigen met een uitgevoerde opdracht op de repository. Alleen werd tijdens de sprint demo duidelijk dat dit niet de bedoeling was. Het is de bedoeling dat elke klasse individueel getest wordt, zodat de testen zo min mogelijk afhankelijk zijn van andere klassen. Hierdoor hebben wijzigingen in andere klassen geen invloed op de werking van je tests. Dit resulteerde in aparte test klasse voor elke normale klasse, en om alles overzichtelijk te houden heb ik voor elke methode een aparte tests gemaakt.

Vervolgens ben ik verder gegaan met de controllers testen. Hierbij wordt gekeken of elke http call goed wordt afgevangen naar de juiste methode en wordt er gekeken of er aangeroepen wordt wat ik verwacht, en krijg ik terug wat ik verwacht. Deze testen had ik meer moeite dan de eerdere testen omdat ik nu met het begrip ‘mocken’ te maken kreeg. Een object kan afhankelijk zijn van andere complexere klassen. Om zoals eerder gezegd alle klasse individueel te testen mock je deze klasse.

Een mock simuleert dan het gedrag van deze klasse. Op de volgende pagina staat een voorbeeld van hoe een test met mocking er uit ziet.

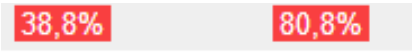
Een test voor een methode die alle customers ophaald

```
@Test
  public void getCustomerListTest() throws Exception {
    new NonStrictExpectations() {
      {
        customerServiceMock.getAllCustomers();
        result = customerBuilder.createCustomerList(2, "cus", "C", false);
      }
    };
    mockMvc.perform(get("/customer/customerList/"))
        .andExpect(status().isOk())
        .andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8))
        .andExpect(jsonPath("$[0].name", is("cus0")))
        .andExpect(jsonPath("$[0].code", is("C0")))
        .andExpect(jsonPath("$[1].name", is("cus1")))
        .andExpect(jsonPath("$[1].code", is("C1")))
        .andDo(MockMvcResultHandlers.print());
    new Verifications() {
      {
        customerServiceMock.getAllCustomers();
        times = 1;
      }
    };
  }
```

De manier waarop mocken hier wordt gebruikt is, door het resultaat van de methode ‘getAllCustomers’ van te voren aan te geven. Zodra de methode dan aangeroepen wordt, zal het altijd dit resultaat teruggeven. Zo kan je testen als je via een http call alle customers wilt ophalen of de methode wel goed aangeroepen wordt.

Verder heb ik ook de rules compliance omhoog proberen te brengen. In het programma Sonar kan je bekijken wat voor opmerkingen er zijn op je code. Door deze problemen op te lossen zal het percentage van de rules compliance omhoog gaan. Veel opmerkingen hadden te maken met gemiste javadoc. Ik heb javadoc toegevoegd waardoor het percentage omhoog ging. Een ander probleem was de code niet overal null safe was, waardoor mogelijke null pointers voor konden komen.

Hiermee ben ik ongeveer twee á drie dagen mee bezig geweest. Hieronder ziet u het resultaat van deze dagen. Het linkse getal representeert de test coverage en rechts de rules compliance.



Figuur 5 begin situatie

Figuur 4 eind situatie

Het is nog niet op het gewenste niveau, en vanwege de tijddruk heb ik besloten eerst verder te gaan met andere taken die nog gedaan moesten worden.

FRONT-END

In de vorige sprint heb ik het uiterlijk van de front-end opnieuw opgezet. Bij deze sprint was het tijd voor het aanpakken van de javascript. Na de code review wist ik dat er het nodige gedaan moest worden, maar dit bleek meer dan gedacht. Uiteindelijk heb ik ongeveer 90 á 95% van de front-end code herschreven.

Om te beginnen heb ik de foute if-else constructie weggehaald om een server object te versturen. Ik dacht dat dit niet veel werk zou zijn, het enige wat ik moest veranderen was een server object meesturen. Door deze wijziging ontstond er een fout bij de back-end waardoor ik geen servers meer op kon slaan. Op dit moment kwam ik in aanraking met 'Jackson'. Jackson is kort gezegd een JSON parser. Het parsed JSON die je binnen krijgt naar een Java model en hier zat het probleem ook. Er werd elke keer een attribuut 'new' meegestuurd. Dit attribuut stond niet in de model, hierdoor kreeg je elke keer terug dat de JSON syntax niet klopte. Voor dit soort problemen kan er met Jackson bij een model een 'JsonIgnore' annotatie worden gezet. De naam zegt het al, hierdoor zeg je dat Jackson dit attribuut moet negeren.

Verder heb ik de front-end structuur aangepast omdat bepaalde acties op de verkeerde plek stonden. Dat betekent dus dat het ophalen van data verplaatst is naar services in plaats van de controllers. Hierna zijn er voornamelijk veel bug fixes gedaan en is er veel code herschreven. De code is herschreven om het allemaal overzichtelijker te maken voor mij en toekomstige developers. Zo werden er bijvoorbeeld DOM elementen gecreëerd in de javascript. Dit maakt het geheel zeer onoverzichtelijk en lastig onderhoudbaar. Met AngularJS kan je HTML voor zichzelf laten spreken en dat is wenselijk. Het later aanpassen van een pagina wordt vrijwel onmogelijk als overal in de Javascript DOM elementen toegevoegd worden. Natuurlijk is het wel nodig om soms bepaalde onderdelen te verstoppen of juist te laten zien. Met AngularJS kan je dit bijvoorbeeld oplossen door een bepaald stuk van je DOM te verbergen bij onder bepaalde conditie. Met deze herschrijving is de code compacter geworden, wat resulteert in beter leesbare code. Natuurlijk werd met de herschrijving de huidige functionaliteit behouden.

DATABASE

Een ander doel van deze sprint was om naar het database model te gaan kijken en zo nodig deze opnieuw te ontwerpen. Met een eerste blik op het oude model bleek al snel dat een server centraal staat binnen deze applicatie.

Wat mij opviel toen ik dit diagram bekeek was de manier waarop gebruikers opgeslagen werden. Ik zal eerst een context geven zodat het duidelijker is waar dit over gaat. Een server heeft twee soorten metingen die ontvangen worden. Zo heb je aan de ene kant de prestatie metingen zoals het disk gebruik of CPU gebruik van een server. Dit zijn gegevens die je vaak wilt hebben en deze worden elke minuut gestuurd. Aan de andere kant heb je nog data die minder snel veranderen. Dit is bijvoorbeeld informatie over de status van een server, denk hierbij aan of er een script fout is voorgekomen, of dat de server een herstart nodig heeft of hoeveel updates er klaar staan. Ook versie en gebruikers data horen hierbij. Het vreemde was dus dat gebruikers als één kolom bij de status data stond. Dit

betekent dus dat alles als één grote string zou worden opgeslagen, waar dit mij niet de bedoeling leek. Verder zaten versies ook aan deze status tabel gekoppeld. Dit liet ik zo staan, omdat van je ook de geschiedenis van versies moest kunnen zien in de applicatie. Door ze aan status metingen tabel te koppelen wist je de tijd van wanneer die versie erop stond. Het nadeel is wel dat je voor elke status meting alle versies elke keer opslaat. Dit doe je voor alle servers, hierdoor word deze tabel snel groot. Maar met een index kan je dit alsnog snel queryen dus dit vormde geen probleem.

Toen ik de manier zag waarop users werden opgeslagen heb ik de JSON structuur bekeken dat naar de applicatie gestuurd wordt. Hierdoor kreeg ik ook een beter beeld wat waar nu bij hoort. Toen ik deze structuur zag waren de users een lijst met namen en niet één grote string. Hierdoor heb ik besloten om van de users een aparte tabel te maken, ook omdat er vaak dezelfde users op een server staan. Door dit aan een server te koppelen maak je het overzichtelijker en heb je niet onnodige data in een tabel en database.

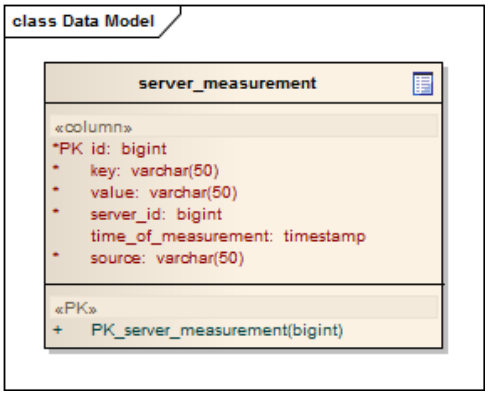
Hierna heb ik gekeken naar de tabellen die de gewone/prestatie metingen opslaan. Het is de bedoeling dat alle oorspronkelijke data die gestuurd worden bewaard blijven. Helaas is daar het feit dat je in de toekomst data van misschien wel vijf of meer bronnen kan krijgen. Deze bronnen kunnen ook weer allemaal verschillende data sturen. Hoe kan je dit het beste opslaan? Hiervoor heb ik de volgende opties bedacht:

- **Per bron een aparte tabel.** Per bron maak je een aparte tabel waar de data in opgeslagen wordt van de betreffende bron.
- **Alles in één tabel.** Alle data van alle bronnen in één tabel opslaan. Deze onderscheid je door een kolom bron te hebben.
- **Key, value.**

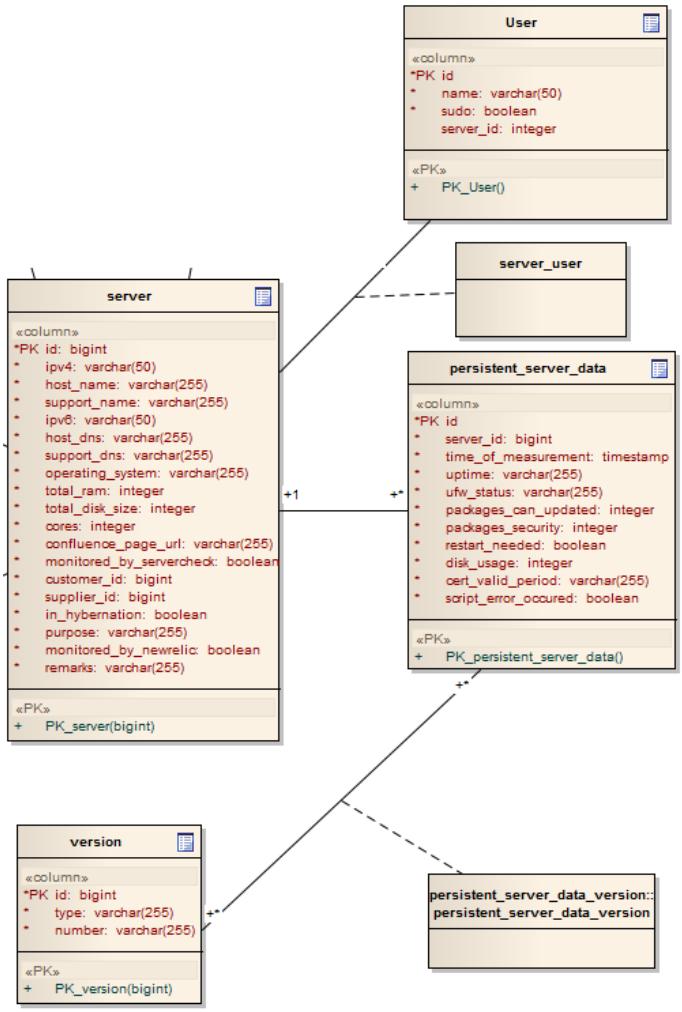
Eerst zal ik key, value wat beter uitleggen. Een key, value tabel zou hier uit de volgende kolommen bestaan.

- ID (optioneel)
- Tijd van meting
- Bron
- Server
- Key
- Value

Een meting hoort bij een server dus deze spreekt voor zich. Dan heb je een key wat de meting is. Denk hierbij aan CPU gebruik of disk gebruik. De value is dan natuurlijk de waarde van die meting. Met de tijd van de meting en de bron zou je dan deze waardes kunnen koppelen en combineren tot één meting. Een voordeel hiervan is dat je flexibel bent. Je zal nooit een aanpassing hoeven te maken in de database voor deze tabel. Het nadeel is dat het onoverzichtelijk is en je niet in een oogopslag goed alle data kan zien mocht dit nodig zijn. Tevens kan deze tabel binnen no-time te groot worden.



Figuur 6 Key, value meting tabel



Figuur 7 Nieuwe situatie status data

Welke optie het uiteindelijk wordt, wordt besproken op de sprint demo.

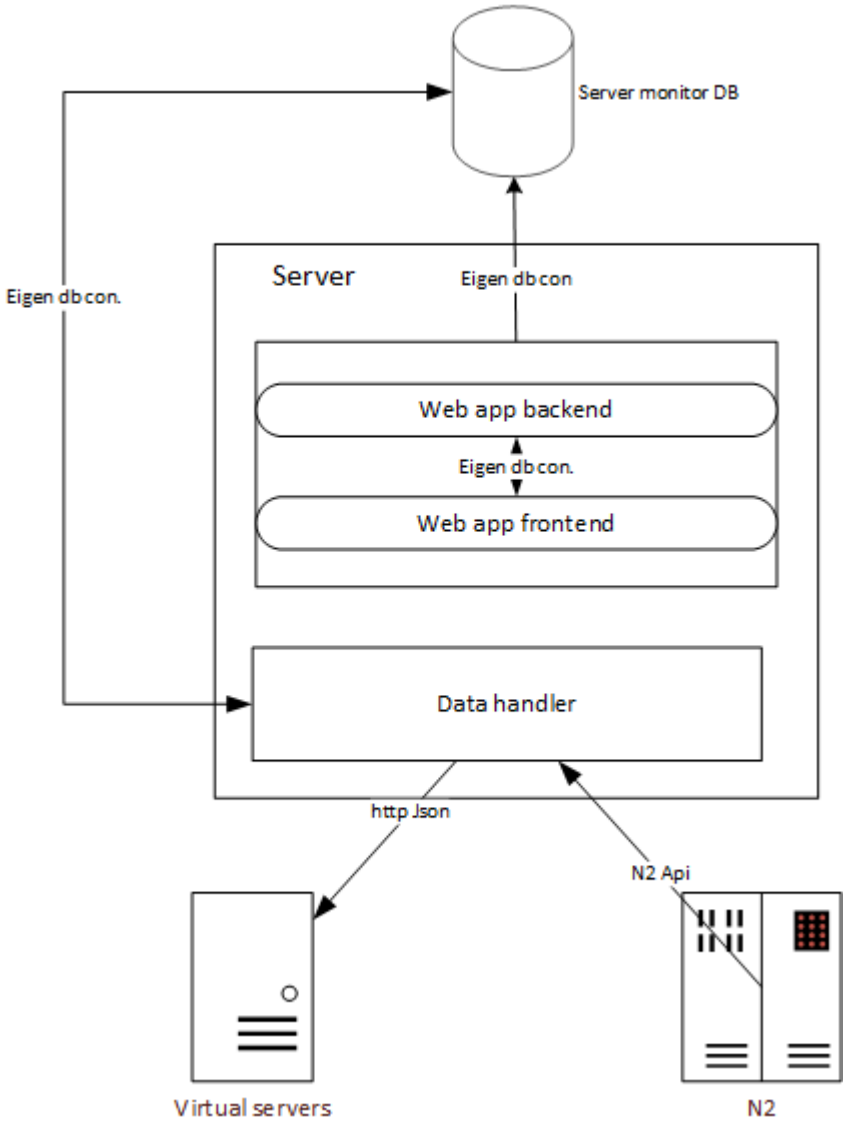
APPLICATIE ARCHITECTUUR

Tijdens de vorige sprint demo kwam ook even de applicatie architectuur ter spraken. In de toekomst is er een grote kans dat op een gegeven moment meer dan 500 servers gegevens pushen. De server waar de applicatie op draait kan dan veel te verduren krijgen en dat kan invloed hebben op de ervaring van gebruiker. Deze moet dan bijvoorbeeld lang wachten voordat hij in de front-end gegevens te zien krijgt, omdat die moet wachten op de verwerking van andere gegevens. Dus aan mij de taak om mogelijke andere architecturen te bedenken voor de applicatie. We hebben dus als eerste optie één waar alles in één applicatie zit.

Hierna heb ik nog een optie bedacht die gebruik maakt van een zogenoemde ‘data handler’. Dit wordt dan een aparte Java applicatie waarnaar alle metingen gestuurd zullen worden. Deze zal dan schrijven naar de database waar de rest van de applicatie ook zijn data uit haalt. Vervolgens kan je

nog kiezen of je deze ‘data handler’ op een hele aparte server gaat zetten. Je kunt er ook voor kiezen om hem gewoon als aparte service te draaien op de server waar de andere applicatie ook draait. Beide met als voordeel dat de applicaties niet in elkaars vaarwater gaan zitten. Ze hoeven niet op elkaar te wachten voor een opdracht. Het nadeel is dat je meer moet onderhouden en deployen enz. Weegt dit op tegen de winst die je er mee haalt? Ook dit zou besproken worden tijdens de volgende sprint demo.

Hier ziet u een voorbeeld van hoe de architectuur er uit zou zien met een aparte data handler.



Figuur 8 Mogelijke architectuur, aparte data handler

JSON

Om eerst nog wat context te geven. Zoals eerder verteld krijgt de applicatie uit meerdere bronnen data. Eén daarvan is een programma dat zelf is geschreven door een medewerker van de support afdeling. Dit heeft de naam servercheck. Deze pushed prestatie en status data naar koekoek. Dit doet hij door JSON naar Koekoek te sturen met de data van de server.

Nadat ik de JSON structuur had bekeken van de data die gestuurd word had ik een idee om dit overzichtelijker te maken, omdat het nu ongesorteerd binnen kwam. Ook maakte dit het makkelijker voor mij om het te verwerken in de toekomst.

Deel van de gestuurde JSON
iptablesRules":["tcp-213.126.14.2-22","tcp-90.145.68.26-22","tcp-84.53.109.99-22","tcp-93.191.131.139-22","tcp-37.34.54.120-22","tcp-178.18.83.54-22","tcp-79.170.89.58-22","tcp-79.170.89.59-22","tcp-37.230.96.193-80","tcp-0.0.0.0/0-80","tcp-0.0.0.0/0-443"],"puppetVersion":"2.7.11","javaVersion":"","postgresVersion":"","mysqlVersion":"","mongodbVersion":"","apache2Version":"","tomcatVersion":"","crowdVersion":"","confluenceVersion":"","jiraVersion":"","stashVersion":"","bambooVersion":"","fisheyeVersion":"","certValidPeriod":"",”sudoUsers":"richard edwin ailbert michiel danny"

Zoals u kunt zien en zoals gezegd staat alles door elkaar. Het ene staat wel in een array (blauw) en het andere niet (rood). Mijn tip was dan ook om alle data te groeperen in een eigen array of map , zodat als je de data uitleest je netjes kan definiëren dat je een lijst van users hebt of een map met server data. Dit bevordert de leesbaarheid en duidelijkheid voor mensen die in de toekomst met deze applicatie gaan werken. Dit advies werd gelukkig met open armen ontvangen en is ook doorgevoerd.

Reflectie

Deze sprint vond ik een stuk beter gaan dan de eerste. Ik heb meer werk kunnen verzetten dan in de eerste sprint. Dit is ook wel logisch omdat je steeds meer aan de materie gewend raakt. Maar ook omdat ik voor mijn gevoel minder tijd heb besteed aan onnodige of onbelangrijke zaken. Hierdoor had ik genoeg tijd om vrijwel heel de front-end te kunnen herschrijven.

Ik vond het prettig dat mijn suggesties serieus werden genomen. Bijvoorbeeld mijn suggestie betreffende de JSON die verstuurd werd. Alleen testen ging niet zo snel en goed als dat ik had gehoopt, ik had het testwerk erg onderschat en daarom is het nog niet op het gewenste niveau.

SPRINT 3, EERSTE DEPLOYMENT

Vanaf de eerste sprint is er gezegd dat het liefst zo snel mogelijk iets naar een testomgeving moest, omdat er dan meer leven en aandacht voor de applicatie komt. Ook levert dit meer feedback van de gebruikers op. Aangezien ik met de eerste twee sprints vooral ben bezig geweest met het herschrijven van bestaande code, werd het nu tijd voor iets anders. Mijn doel deze sprint was om de applicatie klaar te maken voor een eerste deployment.

BESLISSINGEN DATABASE EN ARCHITECTUUR

Vorige sprint is er verteld dat er tijdens de sprint demo van sprint twee beslissingen gemaakt zouden worden over de database structuur en de applicatie architectuur. Dit omdat mij tijdens de sprint demo van sprint één mij geadviseerd werd door een developer van 42 hier goed over na te denken. Dit omdat er ook andere opties waren die misschien geschikt waren voor dit project. De opties van deze twee onderwerpen heb ik vervolgens tijdens de vorige sprint demo neergelegd bij mijn begeleider en de opdrachtgever.

Zo zou er een beslissing worden gemaakt over welke optie er gebruikt zou worden voor de measurements. De tweede optie viel gelijk af en dat verwachtte ik ook wel. Niemand wilde een gigantische tabel hebben waar je alles in opslaat. De tabel is gigantisch, wat onoverzichtelijk wordt en er is een risico op veel null waardes. Dit omdat niet elke meting alles implementeert. Hierdoor bleven er nog twee opties over. Aparte tabellen per bron, of een key, value tabel. Dan was het nu de vraag wat de voordelen zouden zijn van de overgebleven opties. Ik zat nog in de mind set van flexibele applicaties. Dit was dus ook één van de redenen om een key, value tabel te gebruiken. Het is flexibel en als je een extra parameter mee krijgt hoef je niets aan je database aan te passen. Het tegenargument kwam van mijn begeleider. Hij vroeg zich af hoe vaak er een parameter bij kwam. De opdrachtgever dacht dat dit niet zo vaak zou gebeuren. Dit betekent dat je weinig de applicatie aan hoeft te passen en mocht dit wel nodig zijn dan zou dit alsnog niet al te veel moeite. Hiermee werd de keuze wel bepaald. Want wilde we nu echt zoveel tijd besteden aan alles flexibel maken terwijl er niet heel snel een parameter bij komt? En als het komt is het niet veel moeite. Het antwoord was nee van mijn begeleider was nee. Om deze reden is dus ook voor optie één gekozen.

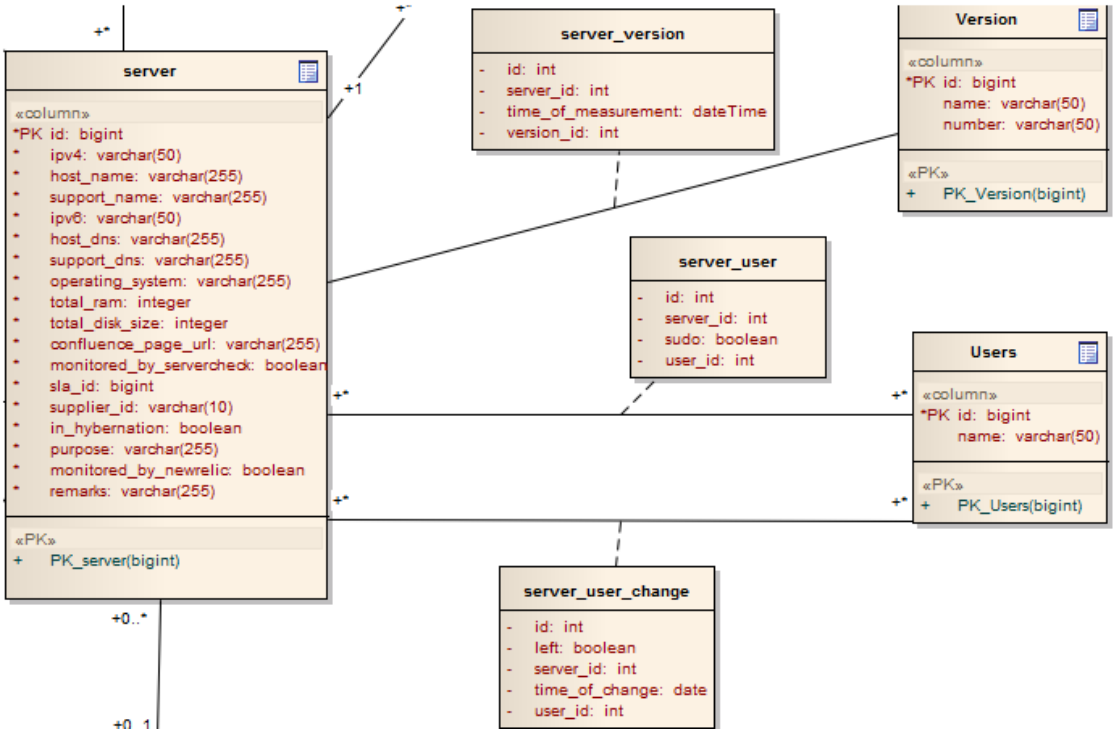
Dan was er nog de keuze van de applicatie structuur. De keuze was om het allemaal te houden zoals het was, dus alles in dezelfde applicatie of een aparte data handler. De reden om het wel te doen was zodat in de toekomst, als er bijvoorbeeld 500+ servers zijn, last weg te nemen bij de server waar de applicatie op draait. Het nadeel is wel dat je nu veel tijd kwijt bent om deze architectuur te veranderen. Waar mijn begeleider mee kwam was dat het ontvangen of ophalen van data weinig moeite kost. Dit omdat het alleen maar wegschrijven van data is, één van de snelste processen die je kan hebben. Dus de server zou dit gewoon aan moeten kunnen. Hier kon ik hem ook niks anders dan gelijk in geven. Een server moet dat gewoon aankunnen. Om deze reden vond hij het niet nodig om heel de architectuur van de applicatie te gaan veranderen. Ik zou tijd gaan verspillen voor iets wat nog helemaal niet relevant is.

VERWERKEN VAN DE JSON

Het originele plan was om zo snel mogelijk te deployen tijdens deze sprint. Dit werd al snel verschoven naar het einde van de sprint. Dit had verschillende redenen. Zo wilde ik ten eerste een stabiele applicatie die ook echt werkte en ten tweede dat de applicatie ook nuttige data liet zien aan de gebruiker. Op dit moment was er alleen maar een tabel met wat gegevens. Ook was er nog niets gedaan aan het verwerken van de data die ik van servercheck krijg. Dit betekende dus dat de applicatie niets meer weergaf dan een lijst met servers van CloudVPS. Deze lijst was niet compleet omdat de API niet goed werkte. Je zou hierdoor weinig aan de applicatie hebben.

De database

Als eerste moest gekeken worden naar de database. Aan de hand van de sprint demo werd duidelijk dat er een wens was om ook de gebruikers geschiedenis op te slaan. Dus wanneer zijn gebruikers verwijderd van een server, wanneer zijn ze erbij gekomen. En voor de versies is besloten alleen de veranderingen in versies op te slaan. Dit omdat het veel onnodige data scheelt en meer heb je ook eigenlijk niet nodig. Hierdoor kwam het database model er weer wat anders uit te zien.



Figuur 9 Nieuwe situatie database sprint 3

Zoals u kunt zien zijn er een paar dingen veranderd. Zo staan versies nu apart van ‘persistent_server_data’. Dit om het overzichtelijker te maken en alleen de veranderingen op te slaan. Verder zijn er nu dus twee relaties tussen de user en server. De ene is er om te kijken welke users er op dit moment op een server zijn. Hier heb ik tevens het ‘sudo’ attribuut verplaatst naar de koppeltabel. Dit omdat ik eerst niet goed had nagedacht, maar een user hoeft niet altijd op alle servers een sudo user te zijn. De andere tabel is om alle user veranderingen op te slaan. Dus wanneer is er een bijgekomen of weggegaan.

Tevens is er een extra tabel bijgekomen voor de metingen. [**Zie bijlage: Databasemodel, Sprint 3**] Zo zie je nu twee bron tabellen. Server_measurement en n2_measurement. Dit is waar de metingen van de individuele bronnen in op worden geslagen, de ruwe data. De functie van de derde tabel 'measurements' is om de metingen samen te voegen tot één meting per tijdseenheid. Op dit moment gebeurt dit nog niet en is het een aggregatie tabel die gebruikt wordt om data uit op te halen. Daarom voor nu ook het veld 'source', zodat je weet uit welke bron de data afkomstig is.

Zoals u misschien al is opgevallen heeft iedere tabel een 'id' attribuut. Dit was in het begin niet mijn bedoeling maar uiteindelijk was het niet anders. In het begin had ik veel samengestelde primary keys. Op school is mij namelijk geleerd om een samengestelde primary key te nemen als dit kan. Dit ging in de praktijk anders. Ik ben bijna anderhalve dag kwijt geweest met het proberen om veel op veel relaties te mappen in hibernate. Helaas is dit met hibernate en JPA heel lastig, omdat elk object/tabel een unieke kolom moet hebben om de objecten te kunnen onderscheiden. Ik moest zelf een generated id te maken voor de koppel tabellen en dit werkte niet goed. Ik kreeg elke keer een error als ik het programma wilde runnen of als ik objecten op wilde halen of op wilde slaan. Na anderhalve dag heb ik met mijn begeleider gesproken om samen na te denken over een betere oplossing. Het antwoord was simpel. Gewoon overal een uniek id meegeven in de tabel en dan wordt de mapping automatisch twee keer een 'één op veel'. Hij zei dat dit ook vaak de standaard is binnen bedrijven, mede door dit soort problemen.

JSON verwerking zelf

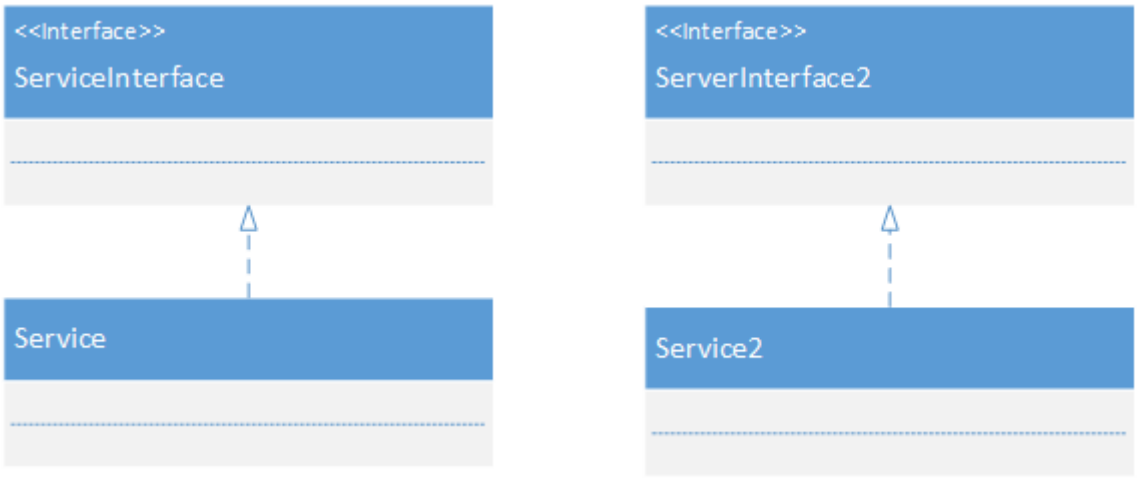
Om JSON te verwerken werd er al gebruik gemaakt van de zogenoemde 'form models'. Toen ik deze voor het eerst zag, wist ik hier het nut niet van en snapte ik niet waarvoor ze gebruikt werden. Maar na navraag bleken deze 'form models' de JSON te representeren die binnen kwam. Eerst vroeg ik me af waarom niet gewoon de models zelf gebruikt werden. Dit was omdat je met een form model kan forceren dat je maar een deel van de attributen van een model hoeft te sturen. Ook deze JSON werd geparsed door middel van Jackson. Een form model moest ook gebruik gaan worden voor het verwerken van de JSON die gestuurd wordt door servercheck. Omdat deze JSON niet overeen kwam met één van mijn models. Dus in de methode die de URL afvangt zeg je dat er een form model binnenkomt. Jackson parsed dit dan automatisch aan de hand van de attribuut namen. Hierna kon ik in dit form model alle andere objecten maken die in de JSON zaten en opslaan.

Een belangrijk deel van het opslaan van de JSON is dat de applicatie niet stuk mag gaan als de JSON waardes bevat of mist die wel verwacht worden. Dit kan als er een fout is opgetreden aan de server kant of de server draait een oudere of juist nieuwere versie van servercheck. Eerst had ik geen idee hoe ik dit op zou moeten lossen. Dit bleek makkelijker te zijn om op te lossen dan ik dacht. Jackson levert een annotatie die zegt dat het alles moet negeren dat niet bekend is bij het model.

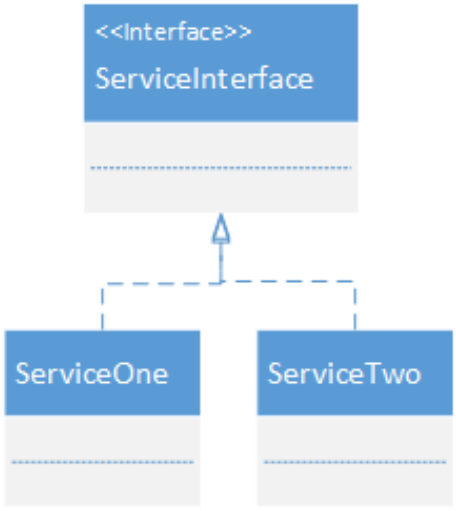
Toen ik eenmaal door had hoe alles werkte was het goed te doen. Alleen omdat alles zo robuust mogelijk gemaakt moest worden zijn bijna alle attributen strings en doubles. Omdat je zelf moet bepalen tijdens het parsen wat voor type het is. Mocht er een foute waarde inzitten of een waarde die je niet eerder kreeg of verwacht, wil je niet dat de applicatie stuk gaat.

FRONT EN BACK-END

Als de opdracht een applicatie ontwikkelen is kan er bijna geen sprint zijn zonder het schrijven van code. En dus ook deze sprint zijn er diverse dingen aan de front- en back-end veranderd. Tijdens deze sprint heb ik naast het toevoegen van code ook code opgeruimd. Zoals toegelicht in sprint één bestond voor elke service een interface. Hieronder nog een keer hoe dat eruit zag:



Figuur 10 Interface structuur



Figuur 11 Services met één interface

Dit is overbodig. Met deze structuur forceer je niks. Als je een methode niet meer nodig hebt kan je deze weghalen uit de service en interface. Dit geld ook voor het omgekeerde. Als je een methode nodig hebt zet je deze in de interface en service, zonder dat er ook maar enige complicaties zijn met andere services. Deze implementeren de interface tenslotte niet.

Als de structuur zou zijn zoals het plaatje hiernaast zou ik het begrijpen. Hier implementeert elke service de zelfde interface. Als men dan iets verandert aan de interface heeft dit gevolgen voor alle services. Zoals gezegd was dit nu niet het geval dus waren de interfaces in mijn ogen nutteloos. Om deze reden heb ik ze ook verwijderd.

Tevens heb ik gekeken naar de mapping van alle objecten. Hiermee bedoel ik of de fetch types wel goed stonden. Je hebt fetch type ‘LAZY’ en ‘EAGER’. Zo heeft een server bijvoorbeeld een klant. Bij lazy haalt die het object alleen op als dit gevraagd word. Bij eager haalt die het klant object ook op zodra een server object wordt opgevraagd. Hierdoor kan je als je niet uitkijkt met één query de

gehele database leeg trekken. Dit is natuurlijk niet de bedoeling. Tevens kan je geen eager hebben bij een veel op veel relatie, dit omdat je dan een recursive loop krijgt. Neem bijvoorbeeld de relatie tussen server en user. De server haalt eager de users op en bij user wordt eager de servers opgehaald en dit herhaalt zich constant totdat de server crasht. Hierom is het belangrijk dat je alleen alles mee fetched wat nodig is.

In sprint één moest ik verschillende design keuzes voor de front-end definiëren. Het enige dat ik toen kon bedenken waren keuzes voor de overzicht pagina. Ik vond zelf meerdere tabellen met gegroepeerde data het beste. De opdrachtgever vond dit ook en dit heb ik geïmplementeerd tijdens deze sprint. In overleg met de opdrachtgever is besloten om drie tabellen neer te zetten op de overzichtspagina. Ten eerste een tabel met server informatie. Ten tweede de server status, dit is een overzicht van elke laatste meting van een server. Als laatste is er een overzicht van alle huidige versies op een server. De query om alle laatste versies op te halen van alle servers was iets moeilijker dan ik dacht. Op school was mij geleerd om in deze situatie je hoofd te legen en de query van achter naar voor op te bouwen. Dit heb ik dus ook gedaan en hier kwam de volgende query uit.

Query om laatste versies van een server op te halen

```
SELECT v.name, v.number FROM version v INNER JOIN server_version sv
ON sv.version_id = v.id WHERE sv.version_id IN (SELECT MAX(svt.version_id)
FROM server_version svt INNER JOIN version vt ON svt.version_id = vt.id GROUP BY vt.name)
AND sv.server_id = :serverId
```

Ik heb hier nog redelijk lang op gezeten. Dit omdat ik al een tijdje niet meer met query's heb gewerkt. Helaas lukte het niet om dit voor elke server in één query te doen. Hierdoor haal ik eerst met een andere query alle servers op en voer dan per server deze query uit. Het zou natuurlijk beter zijn als ik dit later ook in de query kan verwerken. Dit scheelt weer performance.

Reflectie

Deze sprint vond ik zelf redelijk goed gaan op het gebied van progressie. Echter waren er wel weer taken waar ik langer mee bezig was dan dat ik in eerste instantie hoopte. Zo zijn er wel wat tegenslagen geweest tijdens deze sprint.

Zoals verteld, was het de bedoeling dat ik deze sprint zou gaan deployen naar test omgeving. Dit is helaas niet gelukt. Omdat de applicatie nog niet overweg kon met de JSON die het binnen kreeg moest dit eerst ingebouwd worden. Dit duurde langer dan verwacht. Voor mijn gevoel maak ik iets te rooskleurige planningen. Hier moet ik rekening mee houden in de volgende sprints. Hierdoor is wel besloten om gelijk aan het begin van de volgende sprint te deployen. Dan staat er eindelijk iets online om mee te werken en feedback op te geven.

Wat ik mij nu ook realiseer is dat ik het services verhaal beter aan had kunnen pakken dan dat ik nu heb gedaan. Zoals u heeft kunnen lezen heb ik besloten om de meeste interfaces van de services weg te gooien. Het zou beter geweest zijn als ik een interface had bewaard zoals in figuur 11. Dit omdat bijna elke interface wel een get, save/update en delete methode heeft. Het zou dus beter geweest zijn om dit in een interface te zetten om te forceren dat elke service deze methodes ook heeft.

SPRINT 4, REQUIREMENTS EN NOG EEN REDESIGN

Deze sprint had een paar doelen. Ten eerste moest er nu echt gedeployed worden op de test server. Dit was dus ook de eerste taak. Verder is er nog een redesign geweest van de applicatie. In deze sprint zijn ook de requirements pas voor het eerst aan bod gekomen. En zijn er nieuwe functionaliteiten toegevoegd aan de applicatie.

DEPLOYEN

Zoals beloofd is het eerste wat we deden deze sprint het deployen van de applicatie. Omdat dit mijn eerste keer was dat ik dit zou doen werd ik geholpen door een medewerker van support en tevens mijn opdrachtgever. Dit omdat ik nog geen rechten had om op de server te komen waar Koekoek op stond. De eerste dag ging het deployen al fout. We hadden alle stappen ondergaan **[Zie bijlagen: deployment plan]** maar er gebeurde heel weinig op de applicatie. In de logs van applicatie was niks te vinden. Na verder zoeken naar een aanwijzing bleek dat de fout in de localhost log van apache kwam. Er bleek een java error te komen die ik niet kreeg tijdens het testen van mijn applicatie. Na het probleem opgelost te hebben en nog een keer geprobeerd te hebben kregen we weer een error. Tijdens dit proces ben ik erachter gekomen dat een applicatie via apache draaien heel anders werkt dan met een tomcat plugin. Vervolgens heb ik lokaal apache geïnstalleerd en alle errors opgelost die naar voren kwamen. Uiteindelijk error vrij konden we op de tweede dag de applicatie deployen. Eindelijk stond er een werkende versie online. Helaas verloor ik hierdoor wel een complete dag van de sprint.

REQUIREMENTS

Ik heb ervoor gekozen om het samenstellen van de requirements tot deze sprint niet te doen. Het is gebruikelijk dat dit gebeurt aan het begin van een project. Nu had ik het geluk dat dit al een bestaand project was en dat de vorige ontwikkelaars ook al requirements hadden opgesteld. Tevens kon ik in dit project ook niet van te voren alle requirements samen stellen. Dit omdat het project nog vol in leven is. Dit betekend dat er meer wensen zouden komen als het project vorderde. Neemt niet weg dat ik in het begin wel requirements samen had kunnen stellen. Maar dit heb ik niet gedaan omdat ik tot nu toe bezig ben geweest met het verbeteren van de kwaliteit van de huidige applicatie. Hierdoor voldeden nog de requirements die al eerder opgesteld waren. Natuurlijk heb ik deze aan het begin van het project wel gecontroleerd of ze nog klopte met wat mij verteld was.

Aangezien alle huidige functionaliteit verbeterd was, werd het tijd om uit te breiden, zodat het een volwassen applicatie wordt die de gebruikers ook echt voor iets kunnen gebruiken. Voor het samenstellen van de requirements heb ik als eerste alle oude requirements opnieuw geanalyseerd. Hierbij heb ik gekeken welke nog voldeden, welke duidelijk waren opgesteld en welke vragen opriepen. Hier kwam uit dat ik veel oude requirements kon overnemen, dit omdat deze de huidige functionaliteit beschreven. Alleen wat ik al vanaf het begin aan al wist was dat sommige erg onduidelijk waren geformuleerd. Als je de context van het project kende zoals ik wist je wat ermee bedoeld werd. Maar als je aan een developer zou vragen van: maak deze requirement, kon ik garanderen dat je niet kreeg wat je zou willen.

Voorbeeld van oude requirements
Het systeem moet de tijdlijn van de gegevens kunnen veranderen.

Hierboven ziet u een voorbeeld van een oude requirement. Niet heel duidelijk. Welke tijdlijn? Welke gegevens? Een onbekende developer zou hier niks mee kunnen. Wat hier wel mee bedoeld wordt is: De user kan de history van de variabelen gegevens bekijken van een server. Hoe dit kan wordt beschreven in use-cases, maar dit is in ieder geval een stuk duidelijker.

Maar hoe zijn de nieuwe requirements tot nu toe dan samen gesteld. Hoe gaat dit in de toekomst? Tijdens elke sprint demo is er tot nu toe wel feedback gekomen van de opdrachtgever en mijn begeleider. Deze feedback beslaat nieuwe en oude functionaliteit. Wat wil ik erbij? Wat kan er anders? Door deze feedback kwam er altijd een goede discussie. Deze discussies gingen dan over hoe de opdrachtgever wilt dat deze functionaliteit ingebouwd werd. Van alle sprint demo's tot nu toe heb ik notities gemaakt. In deze notities staat wat voor nieuwe functionaliteit de opdrachtgever wilt of wat verandert en hoe. Tevens zijn er aparte gesprekken geweest met mijn opdrachtgever en begeleider. Dit ging meestal over een functionaliteit dat tijdens de sprint demo naar voren kwam en ik nog niet precies wist hoe het geïmplementeerd moest worden. Ook van deze gesprekken zijn notities gemaakt. Deze notities zijn vervolgens geanalyseerd en de wensen die hierin stonden zijn op een rijtje gezet. Zo kon ik eenvoudig kijken wat dubbel was of wat al in de oude requirements stond. Van al deze wensen heb ik vervolgens requirements gemaakt.

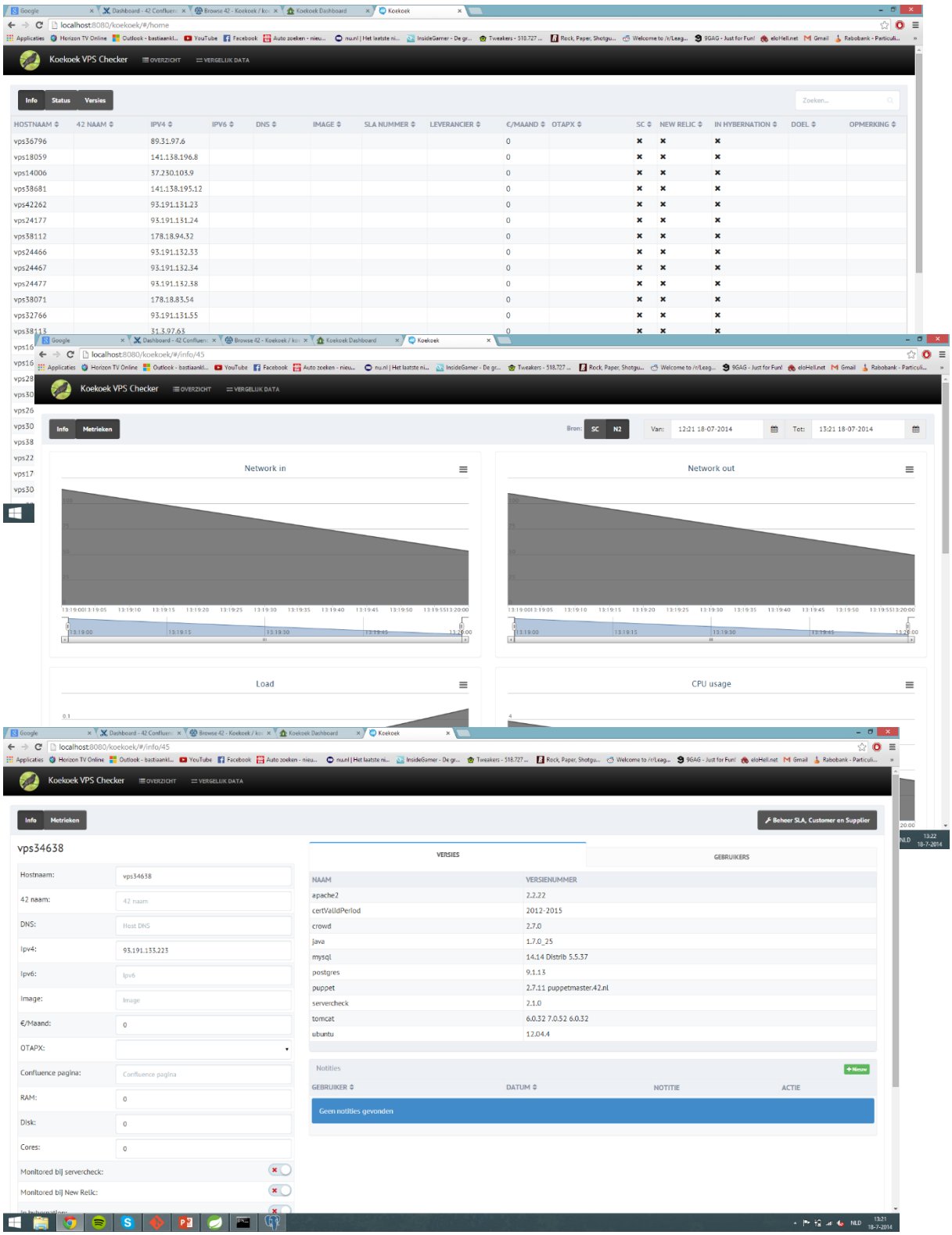
Nadat ik alle requirements had gemaakt, heb ik deze laten verifiëren door mijn opdrachtgever en begeleider. Na het verifiëren bleken de meeste requirements te kloppen. Sommige requirements hadden alleen wat herformulering nodig, omdat ze toch nog te onduidelijk waren. Ook gaf mijn begeleider aan om mijn requirements in JIRA op te nemen als een issue. Zo kan iedereen precies zien welke requirement al gedaan is en welke niet. Natuurlijk kan een requirement meerdere sub-issues bevatten.

Dit proces zal herhaald worden voor toekomstige requirements.

REDESIGN

Tijdens de eerste sprint is er al redesign van de applicatie geweest. Ik vond zelf dat deze er nog niet goed uit zag, de meeste dingen konden overzichtelijker en mooier. Maar het was functioneel en ik ben nu eenmaal geen front-end designer. Ook deze sprint kreeg ik een review van de front-end code van de front-end developer. Gelukkig was er weinig commentaar op mijn AngularJS code. Dit zat goed in elkaar, alleen hier en daar wat kleine puntjes. Dit waren ook puntjes die ik zelf nog wilde aanpassen, maar nog geen tijd voor had gehad. Dit was dan meestal oude code die gerefactored moest worden. Wel was er een handige tip. Dit was om commentaar toe te voegen bij sommige methodes. Omdat ik zelf met applicatie bezig bent vergeet ik zoiets snel omdat ik toch wel weet hoe het werkt. Maar voor andere ontwikkelaars is dit natuurlijk niet duidelijk.

Een ander punt, zoals eerder gezegd, was het uiterlijk van de front-end. Zoals ik al eerder heb gezegd ben ik hier niet goed in, maar Bootstrap neemt hier veel zorgen van weg. Wat ik ook eerder verteld had was dat de front-end ontwikkelaar bezig was met een Bootstrap thema voor interne applicaties van 42. En goed nieuws voor mij, hij had dit thema klaar. Ik kreeg dus ook het advies om dit thema in mijn project te zetten. En voor design van de front-end te kijken naar een applicatie die hij had gemaakt voor het bedrijf. Dit heb ik dus ook gedaan. Het resultaat was het volgende:



De front-end leek nu ineens een stuk meer op de applicatie die hij had geschreven en was in lijn met de interne applicaties van 42. Hierdoor zijn redesigns niet meer nodig. Het zal u ook niet verbazen dat ik veel stijl element heb gebruikt van zijn applicatie, zoals het laatste plaatje. Dit is de nieuwe pagina waar de gebruiker de informatie van een server kan zien en bewerken. Links staat een formulier waar alle server informatie in staat die de gebruiker zien en bewerken. Rechts staan diverse tabellen met informatie over de gebruikers en versies van de server.

APPLICATIE TOEVOEGINGEN

Behalve een redesign is er ook weer het nodige toegevoegd aan de applicatie. Zo moest er stamtabel beheer gekomen voor klanten, leveranciers en SLA's. Stamtabel beheer betekent eigenlijk dat er create, read, update en delete (CRUD) functionaliteiten voor moest komen. Deze functies lijken voor klant, leverancier en sla erg op elkaar. Hierdoor was het niet moeilijk om ze allemaal te maken toen er eenmaal één klaar was. Omdat een SLA bij een klant hoort heb ik deze wel ondergebracht in één scherm in plaats van aparte schermen. Zo heb je alles overzichtelijk bij elkaar.

Tevens moest het mogelijk zijn om een notitie aan te kunnen maken bij een server. Een notitie kan iemand aanmaken als diegene iets bij de server heeft gedaan wat van noembare waarde is. Bijvoorbeeld een server opnieuw opgestart omdat die gecrashed is. Een andere reden kan zijn het updaten van diverse applicaties. Ook deze moest CRUD functionaliteit hebben.

Tevens heb ik opnieuw de rules compliance bijgewerkt. Dit was weer vooral achterstallige javadoc. Dit was dus redelijk makkelijk op het gewenste niveau te brengen.

Liquibase

Ook wilde ik weer gaan testen. Er is tot nu toe al aardig wat veranderd en bijgekomen bij de applicatie, daarom was het weer eens hoog tijd om de testen bij te werken en deze naar een acceptabel niveau te krijgen. Dit ging helaas niet zo soepel als ik had gehoopt. Waarom? Hiervoor zal ik eerst wat meer context geven.

Om database aanpassingen te beheren maakt 42 gebruik van Liquibase. Liquibase zorgt ervoor dat je geen moeilijke handelingen hoeft te verrichten om een aanpassing in de database te maken bij een nieuwe release. In het begin maak je een zogeheten 'groovy' script. Hier zet je een script in die jouw basis/eerste release database maakt. Je kan er voor kiezen om dit met SQL te doen. Maar dit kan ook in de groovy script taal. Het voordeel van deze groovy script taal is dat die DBMS onafhankelijk is.

Als je bij een volgende release een database aanpassing moet maken, maak je een nieuw groovy script aan naast je oude script. Immers elk script is een 'change set'. In dit bestand zet je neer welke aanpassingen er moeten gebeuren in je database. Als je dan gaat deployen voer je de database migrator uit. Deze zorgt ervoor dat alles netjes wordt aangepast. Liquibase maakt ook extra tabellen aan in de database. Deze houden bij welk script al is uitgevoerd op die specifieke database en wat er in dat script stond. Dit systeem werkt perfect, mits je het goed doet.

Wat ik had gedaan in het begin toen ik mijn nieuwe database ging maken was het volgende. Om eenvoudig en snel klaar te zijn heb ik alle query's uit de PostgreSQL admin tool gehaald en deze in mijn groovy script file geplaatst. Hierdoor hoefde ik niet mijn database in groovy script taal te schrijven en kon ik snel verder met andere taken. Tot deze sprint had ik niet meer gekeken naar de testen en ik wilde dat ik dat wel gedaan had. Aangezien ik mijn query's uit de PostgreSQL admin heb gehaald waren deze dus in PostgreSQL dialect. Testen worden gedraaid op een in-memory database. Ik kreeg dus SQL syntax errors. Ik heb met mijn begeleider overlegd wat de beste oplossing was in dit scenario. Want als ik heel het script zou herschrijven naar groovy script moest de database gedropt worden op de test server, omdat het groovy script anders is dan geregistreerd. Helaas was de conclusie dat ik wel dit script moest gaan herschrijven. Hierdoor kon ik wel weer testen. Maar door eerdere gemakzucht was ik nu veel tijd verloren.

METINGEN BEWERKEN

Nu alle basis functionaliteit in de applicatie is verwerkt was het mijn bedoeling om de prestatie metingen die elke minuut komen goed te gaan verwerken. Hiervoor moest eerst nagedacht worden wat nou precies de bedoeling is. Aangezien mijn begeleider en ik op dit onderwerp niet op één lijn zaten heb ik een afspraak met hem gemaakt. Tijdens dit gesprek zou duidelijk moeten worden hoe de metingen verwerkt zouden gaan worden.

Eerder is verteld dat één server uit meerdere bronnen data kan ontvangen. Hierdoor kan je niet zomaar de ruwe data gebruiken en klaar. Dit is nu nog wel het geval omdat elke server toevallig om de minuut data krijgt en toevallig maar van één bron. Maar dit verandert in de toekomst. Deze data zal niet om de minuut komen, maar de ene elke minuut en de andere bijvoorbeeld elke halve minuut of drie minuten. En dan is er ook nog kans dat je de data in een ander formaat krijgt. Het ene bijvoorbeeld in procenten en het andere in absolute waardes. Veel verschillende variaties zijn er dus om rekening mee te houden.

Tijdens het gesprek is besloten dat de metingen binnen drie stappen verwerkt worden. Dit zijn dan de volgende stappen:

- Opslaan van brondata in eigen tabel
- Data verwerken en opslaan voor een nader te bepalen tijdseenheid
- Selectie op de data uitvoeren. Uit welke bron willen we wat.

Hieronder zal ik per stap uitleggen waarom deze gekozen is en wat er gebeurt in die stap.

De eerste stap spreekt bijna voor zichzelf. Het is de bedoeling dat alle originele bron data bewaard blijft. Zoals eerder besloten krijgt elke bron zijn eigen tabel waar hij de gegevens in op slaat. Ten alle tijden wil je kunnen bekijken wat de bron data is/was.

De tweede stap wordt iets ingewikkelder. Zoals bekend zal niet elke bron de data op gelijke tijdintervallen sturen. Hierdoor kan je de metingen niet één op één met elkaar vergelijken. Daarom word in stap twee alle metingen per tijdsinterval geaggregeerd. Deze tijdinterval is nader te bepalen. Maar dit kan per één of per 5 minuten zijn. Het liefst zo klein mogelijk. Ook is nog niet duidelijk hoe we deze willen opslaan. Krijgt elke bron dan weer zijn eigen tabel of word het één grote? Ook dit moet nog nader bepaald worden. Maar belangrijker voor nu, hoe gaat dit dan te werk? Als voorbeeld zijn er twee bronnen. Eentje die zijn gegevens elke minuut stuurt en de andere elke drie minuten.

Minuut	1	2	3	4	5	6
Bron 1	-	-	X1	-	-	X2
Bron 2	X	X	X	X	X	X

Hier is het probleem. We moeten per minuut een meting aggregeren alleen bron één geeft de data maar per drie minuten. Wat vullen we dan in op de lege plekken? Dan is het de bedoeling dat het systeem het gemiddelde neemt tussen de twee metingen. Maar de ene telt wel zwaarder dan de andere. Dit klinkt heel onduidelijk. Hieronder ziet u wat ik bedoel.

Minuut	1	2	3	4	5	6
Bron 1	-	-	X1	(X1 * 0.66) + (X2 * 0.33)	(X1 * 0.33) + (X2 * 0.66)	X2
Bron 2	X	X	X	X	X	X

Zoals u kunt zien word het dan op de volgende manier opgelost. Deze metingen worden ingevuld door bestaande metingen. Maar hoe dichter bij de ene meting qua tijd (bijvoorbeeld X1) hoe zwaarder deze meetelt voor het resultaat. Met wiskunde kan je hier een mooie formule voor maken en dan maakt het niet meer uit om de hoeveel tijd je de metingen krijgt.

Als laatste is er stap drie. Voor deze stap is het de bedoeling dat we na een tijd zeggen, load nemen we van bron 1 en CPU van bron 2. Dit wordt pas bepaald als we weten dat bepaalde data uit bron x betrouwbaarder is dan uit bron y. Dit zal natuurlijk niet door mij bepaald worden maar door de opdrachtgever en de medewerkers van de support afdeling. Wel moet er in de toekomst een fallback systeem ingebouwd worden. Dit houd in dat als er een tijd geen data uit bron x komt hij automatisch terug moet vallen op bron y. De data is dan misschien wat onbetrouwbaarder, maar zo kan je altijd je servers monitoren. Natuurlijk moet het systeem dan wel een melding geven dat er uit bron x al tijd geen data is gekomen. Dit betekent dus dat er waarschijnlijk iets mis is.

Heel dit proces moet ten alle tijden opnieuw uitgevoerd kunnen worden voor alle metingen. Er zal dus een soort aggregatie script moeten komen dat je met één druk op een knop uit kan voeren.

Reflectie

Deze sprint vond ik tot nu toe het beste gaan van allemaal. Ik zat goed in het werkproces en ik denk dat ik redelijk wat werk heb verzet. Misschien had ik op dit punt eindelijk geleerd om een minder ambitieuze planning te maken. Het enige punt waar ik vast heb gezeten was het deployen. Ik had zelf kunnen bedenken dat het handig was om eerst op apache te gaan testen. Verder ben ik blij dat de requirements goed ontvangen zijn. Zo kan ik gewoon door ontwikkelen. Jammer vond ik wel dat ik door die query's te kopiëren in eerdere sprints nu veel tijd nodig had om dit script te herschrijven. Al met al was het een positieve sprint.

SPRINT 5, ADMINISTRATIE PRIORITEIT

Na de sprint demo van sprint vier was naar voren gekomen dat het administratieve gedeelte op dit moment meer prioriteit heeft. Dit betekent dus dat de implementatie van de metingen zoals beschreven in sprint drie nog niet aan de orde komt. Maar waarom heeft het administratieve dan opeens voorrang? Het bedrijf wil dat Koekoek op ten duur Confluence gaat vervangen voor server administratie. Confluence is onderdeel van de Atlassian suite dat 42 gebruikt. Confluence kan je beschouwen als een soort van Wikipedia. Met Koekoek hebben ze meer vrijheid over hoe ze alle gegevens bij gaan houden. Tevens kan Koekoek zo gemaakt worden dat het aan alle wensen voldoet van het bedrijf en precies werkt zoals ze willen.

Tevens had ik tijdens de sprint demo te horen gekregen dat de applicatie gebruikt zal gaan worden voor het blok 'INF-E' op de Haagse hogeschool in Zoetermeer. En dat de applicatie ook langzaam klaargestoomd moest gaan worden hiervoor. De bedoeling was dat er een stabiele en werkende versie functie van de applicatie lag voor de studenten.

GEGEVENS VERBERGEN

Zoals hierboven beschreven moest de applicatie klaar gemaakt worden voor de Haagse Hogeschool. Dit houdt in dat de studenten geen gevoelige informatie mogen zien waar ze misbruik van kunnen maken. Dit stond op dit moment nog wel in de applicatie. Er stonden nog gegevens in de applicatie waar de n2 API verbinding mee kon maken naar buitenaf. Ik heb tot dit punt niet daar de n2 API gekeken en niet naar deze gegevens. Dit omdat ik dacht dat de applicatie alleen maar intern gebruikt zou worden en ik niet eens wist dat deze gegevens in de applicatie stonden.

De gegevens om connectie te maken stonden blijkbaar in de test applicatie context. Iedereen zou ze dus zo kunnen zien. Voor lokaal testen heb ik besloten dat de gegevens voortaan uit een bestand gelezen worden. Maar hoe ging ik zorgen dat de applicatie niet ging crashen als de gegevens er niet waren. In het begin had ik hier een vrij moeilijke oplossing voor bedacht. Ik wilde kijken of het bestand al bestond of dat er bepaalde properties geset waren. Dit kreeg ik niet voor elkaar. Na lang genoeg bezig te zijn heb ik besloten om hulp te vragen aan mijn begeleider. Hij zei dat ik een boolean toe kon voegen die aangaf of het bestand er wel was. Als deze false was zou die nooit proberen connectie te maken en klaar. Dit werkte voor lokaal testen.

Voor de testomgeving en voor de omgevingen van de studenten zou dit anders in elkaar zitten. Deze omgevingen draaien logischerwijs de applicatie onder apache tomcat. En daar gaat het iets anders dan met een bestandje. Daar is besloten om het te verbergen in de web context van apache. Dit is besloten door een medewerken van support, die over deze servers gaat. Om dan de gegevens uit de context te halen moest ik gaan werken met zogenoemde JDNI lookups.

JDNI staat voor 'JAVA Naming and Directory Interface'. Met JDNI kan je diverse externe data bronnen definiëren die hij zal proberen te zoeken door middel van een naam. Denk hierbij aan een web configuratie maar bijvoorbeeld ook LDAP servers. [7]

Om niet in de zelfde fout te gaan als eerst bij het deployen heb ik dezelfde setup neergezet als op de test server. Dus de applicatie via apache draaien met de zelfde web context als op de servers. Deze gegevens stonden in een 'resource tag'. [8] In een resource tag kan je kort gezegd definiëren welke waardes bepaalde attributen hebben van een Java klasse. Na de resource tag gemaakt te hebben en de applicatie config aangepast te hebben om hier mee te werken, werkte het niet. Ik kreeg continu een error dat het JDNI lookup niet kon doen omdat de namen incorrect waren. Maar ook een error dat hij de klasse waar de connectie gegevens in opgeslagen werden voor de applicatie niet kon vinden. Na bijna een dag hier aan te zitten heb ik mijn begeleider gevraagd of hij mij kon helpen. Na alles één voor één nagelopen te hebben en gekeken te hebben bij andere projecten, bleek dat ik één variabele miste in mijn resource tag. De zogeheten 'BeanFactory'. Hierin moet je zetten welke factory hij moet gebruiken om deze Spring bean aan te maken. Deze stond er dus niet en dit resulteerde in een error.

FRONT-END

Inloggen

Een puntje bij elke applicatie is beveiliging. Je wil niet dat iedereen in jouw applicatie kan en alles kan veranderen. Zelfs al wordt het alleen maar voor interne doeleinden gebruikt, alsnog wil je niet dat een kwaadwillige alle data veranderd. Daarom dacht ik dat het tijd werd voor een inlogsysteem. Aangezien dit ook een requirement is.

Gelukkig zou het inlogsysteem niet teveel moeite kosten. Dit omdat elke interne applicatie van 42 gebruikt maakt van een inlogsysteem. Waarom zou ik dan zelf een heel inlogsysteem maken als deze al beschikbaar is bij andere applicaties. Deze heb ik dus ook zo veel mogelijk proberen te kopiëren. Alleen zat ik wel met één probleem. 42 heeft een centraal systeem waar geregeld staat welke user in welke applicatie mag, genaamd Crowd. Maar de huidige inlogsystemen praten via een andere systeem genaamd 'Postduif'. Alleen heeft mijn systeem niks met Postduif te maken. Na navraag bleek dat Postduif ook communiceert met crowd. Hierdoor zag ik op dat moment geen problemen met dit inlogsysteem.

Ik dacht snel klaar te zijn en een werkend inlogsysteem te hebben. Helaas bleek Postduif op een andere manier te communiceren met crowd dan ik nodig had. Tevens moest mijn applicatie niet van Postduif afhankelijk zijn. Dus mocht ik dit niet systeem niet gebruiken in zijn huidige vorm. Ten tweede had ik niet goed nagedacht over de Haagse Hogeschool. Studenten hebben natuurlijk heel dit systeem niet. Logisch. Dus toen ik al wat tijd aan het inlogsysteem gespendeerd had bleek dat het of weg moest of ik een andere oplossing bedenken. Om niet alle progressie te verliezen heb ik maar

[7] JDNI: http://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface
[8] Apache resource definition :
http://tomcat.apache.org/tomcat-5.5-doc/config/context.html#Resource_Definitions

besloten dat een gebruiker standaard ingelogd is met ‘admin’ en als wachtwoord ‘admin’. Voor nu dan. Later is het wel de bedoeling dat de applicatie een volwaardig inlogsysteem heeft natuurlijk. Maar voor nu dus veel verspilde tijd.

Bekende en de geschiedenis

Verder waren er nog extra wensen die de opdrachtgever graag in de applicatie zou hebben. Zo moest je kunnen aangeven voor bepaalde objecten zoals gebruikers of ze bekend waren. Dit betekent dat deze user altijd op een server staat of niet belangrijk is voor een server. Dit is heel simpel gedaan met een tabel waar bijvoorbeeld alle users in staan. Om het niet ingewikkeld te maken staat bij elke user een checkbox. Hier kan je kiezen of die user bekend is of niet. Hier moet je natuurlijk wel op kunnen filteren op de informatie pagina en ook dit is verwerkt. De informatie pagina laat standaard alleen de niet bekende gebruikers zien, tenzij anders is aangegeven. Tevens is op de informatie pagina ook een filter onder gebracht waarmee je kan kiezen of je alle huidige users of versies wilt zien of ook de geschiedenis.

Refactoren

Zo nu en dan is het wel eens nodig om wat code te refactoren. En deze sprint heb ik in de front-end vooral code gerefactored wat beter een directive kon zijn. Zo moet je om de geschiedenis van diverse metingen te kijken een tijd kunnen selecteren. Hiervoor heb ik een date-time picker directive gebruikt. Nu gebruik ik op meerdere plekken deze date-time picker gebruiken, alleen lukte het mij eerst niet om hier een directive van te maken. Ik ging de directive die ik gedownload had in een nieuwe directive verwerken, dit omdat die nog niet helemaal was zoals ik wilde. Aangezien ik deze op steeds meer plekken ging gebruiken moest het nu wel. Dit omdat je anders in heel veel controllers de zelfde code krijgt. En code duplicatie willen we zo veel mogelijk voorkomen. En ik heb nu meer ervaring met AngularJS.

Gelukkig was het me deze sprint wel gelukt. Alleen zit ik tot de dag vandaag nog wel met één probleem met deze directive. De directive heb ik zo gemaakt dat je een methode kan meegeven die hij moet uitvoeren als er een nieuwe tijd geset wordt. Maar om één of andere reden voert hij de methode in de controller sneller uit dan dat de tijd van de variabelen wordt veranderd in de controller. Ik denk dat dit te maken heeft met de manier waarop AngularJS werkt. Je bindt een variabele zo aan de directive dat als die in de directive verandert hij deze ook in de controller veranderd en visa versa. Ik denk dat de tijd dat AngularJS nodig heeft om deze variabelen te veranderen in de controller langer is dan dat de rest van het script nodig heeft om uit te voeren. Dit heeft als gevolg dat hij al de methode uitvoert voordat de variabelen veranderd zijn. Hierdoor pakt het de oude waarde van de variabelen. Ik heb het probleem niet kunnen vinden op het internet. Om dit probleem toch op te lossen heb ik een knop gemaakt waar de gebruiker op moet drukken om nieuwe data op te halen.

Eerst had ik ook een directive die een boolean waarde verving met een vinkje of een kruisje. Helaas voldeed deze ook niet meer. Omdat ik te horen kreeg de meeste waardes waarvan je denkt dat het een boolean is ook een derde waarde kunnen hebben. Bijvoorbeeld null als onbekend. Hierdoor heb

ik een directive gemaakt die alle waardes kan vervangen in wat je maar wilt. Dit zodat het breed toepasbaar is in de applicatie.

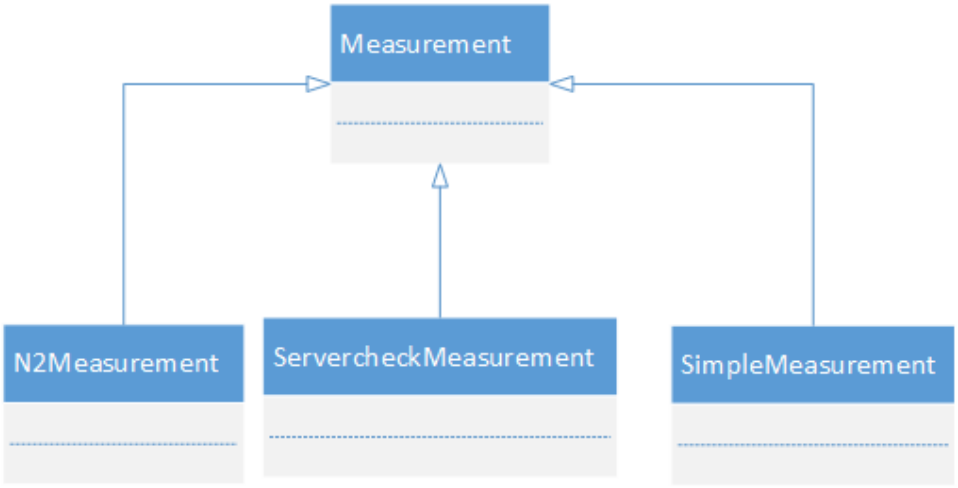
Als laatste was er een wens om een nieuwe hoofdpagina te creëren. Zoals eerder verteld kunnen gebruikers notities aanmaken. De nieuwe hoofdpagina werd een zogeheten 'activitystream'. Op deze pagina kan je alle notities van alle servers zien. Dit zodat de gebruiker in één oogopslag kan zien wat er allemaal gebeurt op alle servers.

BACK-END

Om te beginnen zat ik met een fout die ik tot de dag van vandaag niet op de correcte manier heb opgelost. Zoals verteld wordt er gebruik gemaakt van een PostgreSQL database. Wat kan voorkomen bij sommige velden in een database zijn standaard waardes. Deze had ik ook nodig, bijvoorbeeld de meeste booleans in de database staan standaard op false. Bijvoorbeeld of mensen sudo user zijn op een server of niet. Meer om één of andere reden pakt PostgreSQL niet de default gedefinieerde waardes. En ik weet nog steeds niet waarom. Daarom moest ik elke standaard waarde alsnog in de back-end definiëren. Terwijl je dit door de DBMS af wil laten handelen.

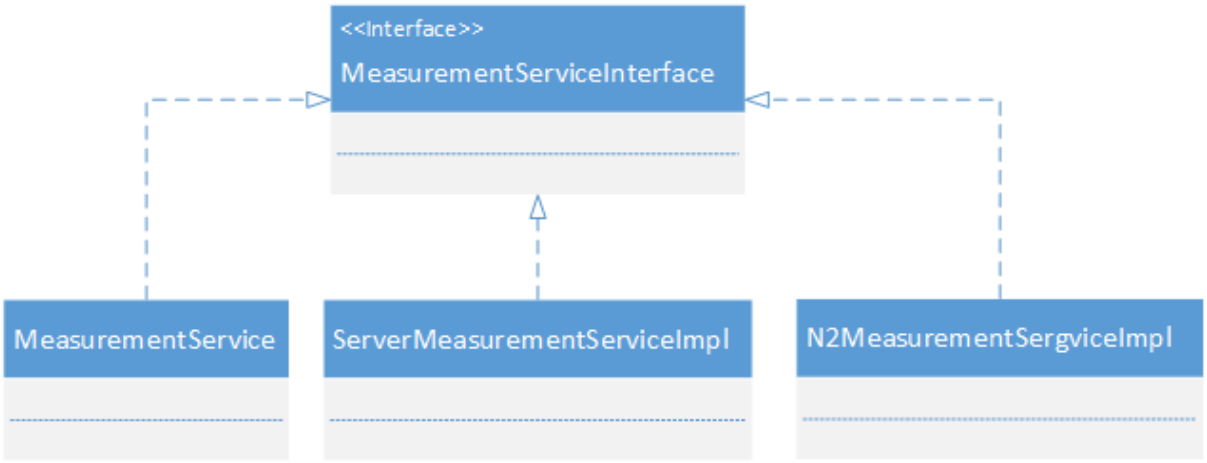
Metingen

Aangezien de applicatie klaar gemaakt moest worden voor de Haagse Hogeschool was ik niet meer de enige die aan Koekoek werkte. Andere medewerkers werkten er ook aan om lessen voor te bereiden. Hierdoor kreeg ik errors in alle measurement klassen. In eerste instantie zag ik niet waarom. Maar later bleek dat er een measurement toegevoegd was, om een onderdeel beter te kunnen uitleggen. Maar ik was op dit moment even heel het overzicht over alle metingen kwijt. De models zagen er als volgt uit.



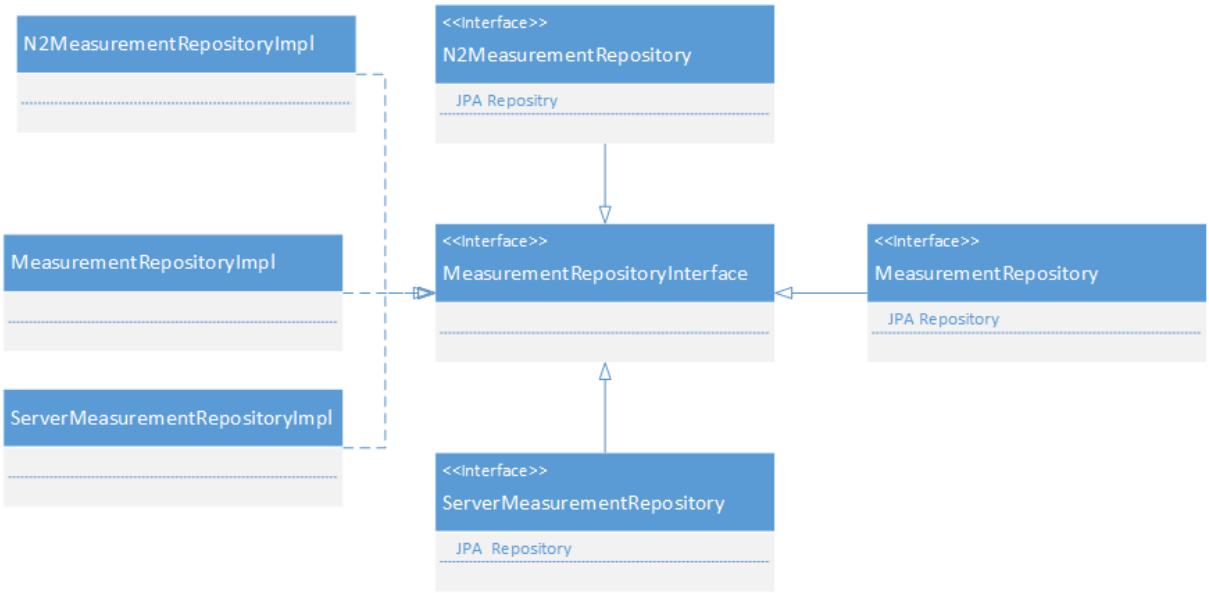
Figuur 12 Oude structuur measurements models

Zoals u kunt zien zijn er nu drie measurement models. Dit is vreemd want ik krijg uit twee bronnen data. Tevens had ik besloten om gelijk alle services en repositories van de measurements op te ruimen. Omdat deze naar mijn mening ook onoverzichtelijk werden.



Figuur 13 Oude structuur measurement services

Op het eerste gezicht ziet dit er logisch uit en ik kan begrijpen waarom je deze structuur zou nemen. Alleen het probleem was dat de MeasurementServerInterface methodes had die andere services niet nodig hadden. Zo had je in sommige services methodes die niet geïmplementeerd waren, omdat bijvoorbeeld een ServerMeasurement en een N2Measurement niet meer functionaliteit nodig hebben dan opslaan. Maar persoonlijk vond ik het pas bij de repositories onoverzichtelijk worden.



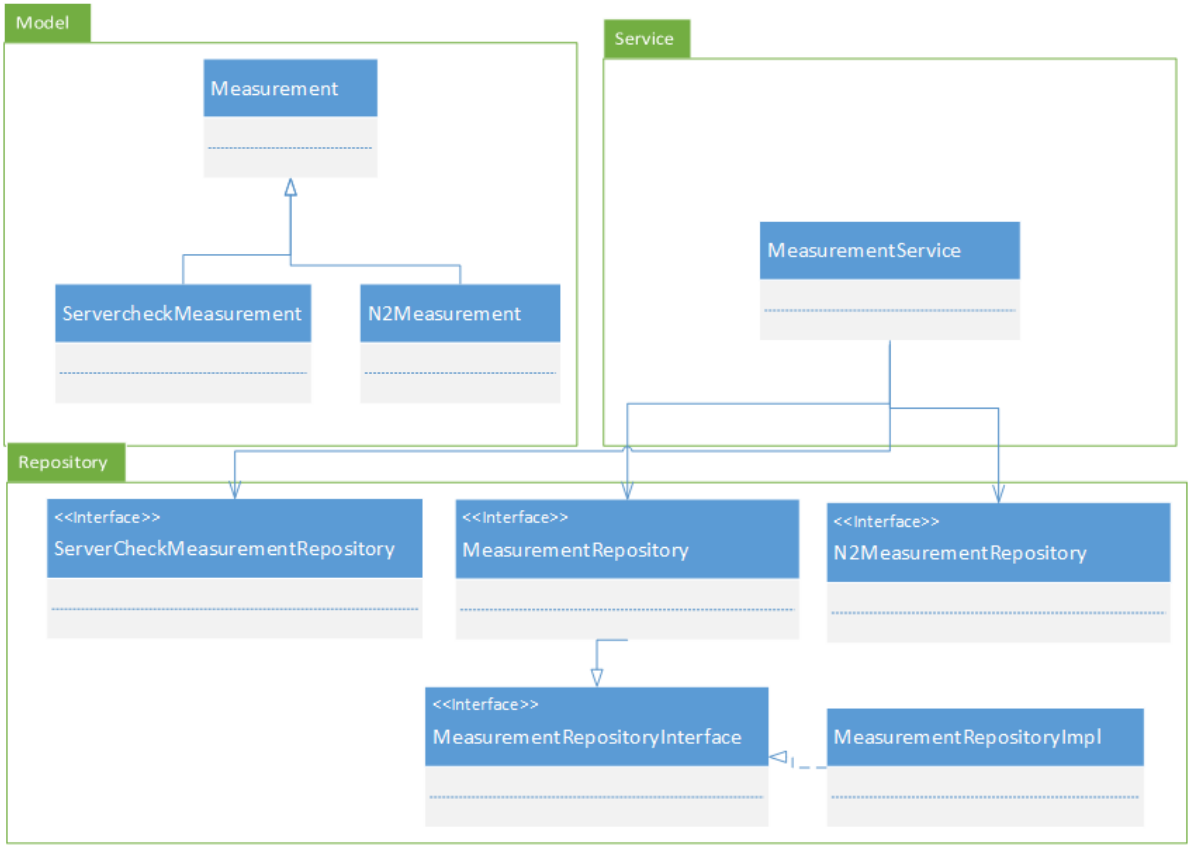
Figuur 14 Oude structuur measurement repositories

Ook hier was het probleem dat sommige methodes niet geïmplementeerd waren, omdat deze wel geforceerd werden door de interface maar een klasse ze niet nodig had. Ik zag door alle klassen de structuur niet meer. Ten eerste heb ik gevraagd waarom nu de SimpleMeasurement toegevoegd was. Dit was om inherintance in combinatie met Hibernate uit te leggen. Hiermee bedoel ik dat twee verschillende modellen gebruik maken van de zelfde database tabel. Maar dat ze onderscheid worden, in dit geval, door één kolom. Hier was dat de kolom source. Ik heb deze structuur voor hem gehouden, maar ik heb wel weer de SimpleMeasurement weggehaald, omdat deze geen functie had binnen de applicatie en alleen maar voor verwarring zou zorgen. Een nadeel is wel dat van de

oorspronkelijke twee bron tabellen nu weer één was gemaakt. Voor nu heb ik dit even laten staan, omdat ik voorlopig toch niets met de metingen zou doen. En zo zou nog de goede structuur behouden blijven voor zijn les. Als er later wat met de metingen gebeurt, moet dit natuurlijk wel weer veranderd worden.

Hierna heb ik alle services teruggebracht naar één service. Hiermee ga je de niet geïmplementeerde methodes tegen. Tevens hebben ServerMeasurement en N2Measurement tot nu toe alleen maar een opslaan functie nodig. Dit kon eenvoudig onder gebracht worden in de MeasurementService.

Voor de repositories heb ik gekozen om één custom repository over te houden. Hierin komen alle query's te staan die niet gegenereerd kunnen worden met een JPA repository. Dit maakt het overzichtelijker. Ten tweede hoef je niet meer net zoals bij de services, als je één methode wilt toevoegen drie klassen aanpassen. En je hebt geen klassen meer die methodes moeten implementeren die ze eigenlijk niet willen implementeren. Verder is het niet nodig om de JPA repositories de originele MeasurementRepository te laten extenden. Dit doen ze nu dus ook niet.



Figuur 15 Overzicht nieuwe measurement situatie

Reflectie

Deze sprint vond ik over het algemeen goed gaan. Ik heb het gevoel gehad dat ik veel werk heb verzet tijdens deze sprint en dat er genoeg voortgang zit in het project. Ook tijdens de sprint demo was de opdrachtgever en mijn begeleider tevreden met het resultaat. Dit is altijd fijn om te horen. Ik vond het jammer dat er veel tijd verloren is gegaan aan het verbergen van de n2 gegevens.

SPRINT 6, DE HAAGSE HOGESCHOOL

Elk jaar beginnen de hogescholen op één september. Zoals eerder te lezen is in dit verslag ging de applicatie gebruikt worden voor een blok op de Haagse Hogeschool. Dit zou gebeuren in het eerste blok van het jaar, het blok waar de applicatie gebruikt zou worden heet 'INF-E'. Dit blok gaat over beheer. Door samenwerking met de docenten is dit blok opnieuw opgezet door 42, werknemers van 42 zullen ook les gaan geven in dit blok. Voor de studenten moest er dus voor één september een goede versie liggen van Koekoek zonder half werkende functies. Dit is dus ook de focus deze sprint. Hiernaast zal er nog gewerkt worden aan extra functionaliteit voor de applicatie.

JAVA 8

Tijdens de ontwikkeling van dit project werd Java 8 steeds serieuzer. Maar tot nu toe was altijd ontwikkeld met Java 7. Ik kwam erachter dat de applicatie nog niet werkte met Java 8, sommige studenten zouden dus niet met de applicatie kunnen werken. Studenten met een nieuwe laptop draaien waarschijnlijk Java 8, omdat deze gewoon de nieuwste versie instaleren. Hierdoor moest de applicatie dus ook kunnen draaien op Java 8. Het is ook voor de toekomst handig dat het alvast Java 8 support heeft zodat er geen problemen ontstaan wanneer 42 intern ook deze overstap gaat maken. Na wat onderzoek bleek dat het niet al te moeilijk was om support voor Java 8 te leveren. Ik moest mijn dependencies updaten. Ik had eerder nog niet gekeken naar mijn dependencies, dit omdat ze al geconfigureerd waren en ik nog geen nieuwe nodig had. Ik moest dus onderzoek doen om dit werkend te krijgen.

Zoals eerder verteld worden dependencies gemanaged via Maven. Elke dependency staat gedefinieerd in de 'pom.xml', hier zet je in welke dependency je wilt en welke versie van die dependency. Als je de applicatie dan voor het eerst runt zal hij de dependencies automatisch downloaden. Hetzelfde geldt als je een dependency versie update, Maven zal automatisch de nieuwe versie downloaden.

De applicatie updaten naar Java 8 was makkelijker dan gedacht, in het begin zag de pom.xml er ingewikkeld uit maar dit viel uiteindelijk mee. Ik werd namelijk in één keer geconfronteerd met alle soorten tags die de pom.xml bevat, bijvoorbeeld plug-ins en dependencies, deze hebben dan allemaal weer verschillende sub-tags. Het is gebruikelijk bij 42 om bovenaan de pom.xml alle versies neer te zetten, zo kan je snel de versies van je dependencies updaten zonder dat je heel je pom.xml door moet. Het bleek dat de applicatie op een oude versie van het Spring framework draaide. De huidige versie was '3.x.x', terwijl versie vier van het Spring framework al uit was. Dit is ook de reden waarom Java 8 het niet deed. Spring framework 3.x.x heeft namelijk geen support voor Java 8, om deze reden is die ook meteen geüpdate. Dit resulteerde in een aantal foutmeldingen bij andere dependencies, bijvoorbeeld Hibernate, omdat ze geen ondersteuning hadden voor de nieuwste Spring versie. Gelukkig kwam dit omdat deze dependencies ook allemaal oudere versies hadden, door ze te updaten waren er geen problemen meer. Een eis was wel dat de applicatie gecompiled werd op Java 7, dit was simpelweg gedaan door het build target op 1.7 te zetten in plaats van 1.8.

FRONT-END TOEVOEGINGEN EN FIXES

Zoals eerder beschreven was het, is het belangrijk om een stabiele versie van de applicatie op te leveren voor de studenten. Mijn begeleider zei dat het niet erg was als er nog wat bugs inzaten, hierdoor hadden ze goede opdrachten voor de studenten, maar de applicatie mocht natuurlijk geen grote problemen bevatten, dit betekent echter niet dat er geen extra functionaliteit is toegevoegd aan de applicatie.

Als eerste was er de wens om in de grafieken van het server overzicht ook versies neer te zetten. Zo kunnen de gebruikers makkelijk zien of een versie invloed heeft op de data. Een voorbeeld hiervan zou kunnen zijn dat door een Java update het CPU gebruik sterk is toegenomen. Door dit versienummer toe te voegen kunnen ze een dergelijke situatie direct herkennen en eventueel daarop baseren of ze een versie terug moeten draaien. Voor de grafieken wordt gebruik gemaakt van Highcharts. Highcharts is een plugin waarmee je makkelijk grafieken kan maken die tevens erg makkelijk zijn aan te passen naar de wensen van de gebruiker, [9] oorspronkelijk is Highcharts voor jQuery, maar tegenwoordig is er ook een stand alone versie. Hierdoor had Highcharts ook voor dit probleem een oplossing. In de grafieken kan je op een bepaalde tijd 'plotlines' neerzetten, dit is niets meer dan een streep met als label het versienummer. Er was wel een nadeel aan mijn implementatie. De grafieken werken via een Angular directive, hierdoor moest ik de versies voor elke grafiek opnieuw ophalen, omdat elke keer als deze directive gebruikt werd, de directive een aparte scope krijgt. Dit resulteert in een hoop onnodige calls richting de server. Hier kon ik nog geen goede oplossing voor bedenken en hier valt dus nog een verbeterslag te maken.

Ook kon de front-end verder verbeterd worden door validatie toe te voegen voor de data. Op dit moment werd er in de front-end niet gekeken naar de inhoud van de data wat betekend dat foutieve data zonder meldingen naar de back-end werd gestuurd. Aan de kant van de back-end wordt hier natuurlijk rekening mee gehouden, maar het is handig om dit van te voren ook al te doen. Zo behoed je de gebruiker van eventuele fouten. Hiervoor heeft AnuglarJS een ingebouwde functionaliteit genaamd formulier validatie. AngularJS kijkt of de input klopt met het type input dat opgegeven is. Als één van de waardes niet klopt zal de data niet naar de server verzonden worden. Ook wordt per input veld bijgehouden of deze foutief is ingevuld of niet, hierdoor is het mogelijk om een specifieke melding te geven per waarde die incorrect is. Dit is handig om de gebruiker extra feedback te geven.

Er is deze sprint nog veel tijd besteed aan het zoeken van een andere datum tijd picker. Dit omdat de huidige door de opdrachtgever als onprettig ervaren werd, omdat je het werkte met een systeem waar je door moest klikken naar de volgende eenheid. Helaas heb ik na veel zoeken geen geschikte vervanging gevonden. De meeste waren alleen date picker en andere werkte op de zelfde manier als de huidige. Hierdoor heb ik de oude teruggezet voor de Haagse Hogeschool.

Eén van de laatste taken voor deze sprint was het opslaan van gegevens in een sessie, dit zodat de gebruiker niet al zijn instellingen opnieuw hoeft te setten wanneer de pagina een keer herladen

[9] Highcharts: http://www.highcharts.com/

wordt. Dit is op twee manieren te doen. Door een AngularJS cookieStore te gebruiken of door een HTML 5 local storage te gebruiken. Allebei werken ze via keys waarbij elke key een bijbehorende value heeft. De keuze viel uiteindelijk voor de HTML 5 Local Storage. Deze is gekozen, omdat local storage veiliger en sneller is dan cookies. Zo wordt de data alleen maar opgevraagd als erom gevraagd wordt door de applicatie en kan je grotere stukken data opslaan dan met cookies. Hiernaast kan een webpagina alleen maar bij informatie dat door de webpagina opgeslagen is. [10]

Verder heb ik veel bugfixes gedaan in de front- en back-end. Hierbij ging het om kleine bugs. Zo werden sommige waardes niet goed geset en waren er andere waardes die niet goed opgehaald werden. Daarnaast heb ik nog gewerkt aan de rules compliance en heb ik meer javadoc toegevoegd. Vooral javadoc is handig voor studenten om te weten wat bepaalde methodes doen.

HAAGSE HOGESCHOOL

Koekoek wordt gebruikt bij het blok 'INF-E' op de Haagse Hogeschool, er werd mij gevraagd of ik de beheerder wilde helpen met het deployen van de Koekoek applicatie op de servers. Dit moest gedaan worden omdat elk groepje zijn eigen server krijgt waar ze steeds een nieuwe versie van de applicatie moeten gaan deployen. De basisversie van de applicatie moest standaard al op de server staan. Nadat de beheerder een plan had opgezet, deze lijkt veel op het deployment plan van Koekoek om naar de testomgeving te deployen, was het aan mij de taak om deze te controleren. Ik was vrij om het plan aan te vullen indien er stappen ontbraken of incorrect waren. Nadat ik het plan had doorlopen en verbeterd had was het eenvoudig om de rest van de applicaties uit te rollen. Om niet veel tijd hieraan te verliezen deed ik de ene helft en de beheerder de andere helft.

Verder zijn mij nog andere kleine zaken gevraagd, bijvoorbeeld het testen van verschillende services uit de ontwikkelstraat die opgezet werden door 42 voor de Haagse Hogeschool en het schrijven van de opdrachtomschrijving voor de opdracht van de eerste week. Op deze manier raakte ik meer betrokken bij het Haagse Hogeschool project. Ik vond dit een leuke taak, omdat ik het leuk vond om meer betrokken te zijn bij het project waar een applicatie wordt gebruikt dat door mij ontwikkeld is.

Reflectie

Deze sprint beschouw ik als één van de mindere sprints gedurende dit project. Omdat ik deze sprint in ging met de gedachten om een goede versie voor de studenten op te leveren heb ik voor mijn gevoel niet zoveel progressie geboekt op het gebied van nieuwe functionaliteiten. Dit omdat ik vooral veel bezig ben geweest met bug-fixes en het stabiel krijgen van de applicatie.

[10] HTML 5 localstorage: http://diveintohtml5.info/storage.html

SPRINT 7, AFRONDING

Sprint zeven is de laatste sprint van het project. Dit betekent dat ik mijn werkzaamheden af moet ronden en deze sprint naar een resultaat moet gaan waar de opdrachtgever tevreden mee is. Zoals eerder beschreven was, is de prioriteit van het project verschoven van de metingen naar het administratieve gedeelte. De bedoeling is dat deze applicatie Confluence gaat vervangen. Het was dan ook mijn doel deze sprint om deze functionaliteiten te gaan implementeren in Koekoek.

Analyse Confluence en Servercheck

Om te beginnen heb ik Confluence en Servercheck geanalyseerd. Met de term Servercheck in dit hoofdstuk wordt niet het programma bedoeld dat de gegevens pushed naar Koekoek. De opdrachtgever heeft op dit moment ook een zelfgeschreven applicatie die dezelfde naam draagt. Deze wordt op dit moment ook gebruikt om de status van servers te bekijken, verder heeft het geen administratieve functies. De reden waarom ik dit niet eerder heb gedaan is door het Haagse Hogeschool project, omdat dit tussendoor kwam ging hier mijn tijd eerst inzitten. Als eerste ging ik dus kijken wat de opdrachtgever nu precies in deze tabellen heeft staan en dit vergeleken met de huidige tabellen in Koekoek. De tabellen in Koekoek kwamen redelijk overeen met die van Servercheck. Waar ik achter kwam was dat nergens de persistent/status data wordt getoond in Koekoek. Hiervoor heb ik een tabel gemaakt in de overzichtspagina. Waarna ik nog een afspraak heb gemaakt met de opdrachtgever om de tabellen op de overzicht pagina te valideren en te bespreken welke data in de tabellen moest komen te staan.

Tijdens de afspraak bleek dat de huidige tabellen klopte. Nu waren er dus tabellen voor algemene informatie, prestatie, status en versie gegevens. Er was alleen de wens om sommige velden naar een van de andere tabellen te verplaatsen. Om deze data goed in de tabellen te krijgen heb ik aparte modellen gebruikt in de back-end. Dus niet modellen die één op één gemapped waren met de database.

Hierna heb ik gekeken wat er allemaal in Confluence mogelijk is waar Koekoek nog geen ondersteuning voor bied, hieruit bleek dat Koekoek sommige velden miste die je wel kon invullen in Confluence. Wat ik uiteindelijk heb gedaan is alle velden overgenomen van Confluence. Elk veld dat in Koekoek stond en niet in Confluence en niet wenselijk was door de opdrachtgever heb ik verwijderd. Wel is het mogelijk om bij elk veld (waar dit gevraagd werd door de opdrachtgever) een link op te nemen in plaats van een andere waarde. Hierdoor kan de opdrachtgever altijd naar een externe bron refereren.

Ook kan je in Confluence zogenoemde applicatie eigenschappen opgeven, dit zijn opmerkingen bij bepaalde applicaties die draaien op een server. Bijvoorbeeld welke sudo users aanwezig zijn in de database of een script om een applicatie te herstarten. Ook dit is dus verwerkt in Koekoek.

APPLICATIE AFRONDING

Om de applicatie af te ronden dit project moest er nog meer gedaan worden. Zo heb ik diverse aanpassingen gemaakt om de gebruiksvriendelijkheid te verhogen. Een voorbeeld hiervan is dat ik

een indicator heb toegevoegd die aangeeft of er data geladen wordt. In sommige situaties, zoals het bekijken van de prestaties van een server over een lange tijdsperiode, kan het relatief lang duren om deze data op te halen. Om feedback aan de gebruiker te geven dat de server nog bezig is, en er dus geen fout is opgetreden of iets dergelijks, is dit toegevoegd.

Ook had de opdrachtgever aangegeven dat het fijner zou zijn om via de URL al naar de server informatie pagina te kunnen navigeren.. In de huidige situatie ging je naar de server informatie pagina door middel van een 'id'. Het leek de opdrachtgever beter om dit te doen door middel van de host naam van een server. Deze is ook identiek, dus dit was geen probleem. Zo kunnen gebruikers gelijk via de url naar een server, omdat ze wel de host naam weten maar niet het gegenereerde id, dit verhoogd ook de gebruiksvriendelijkheid.

Zo waren ook nog andere requirements die niet nog geïmplementeerd waren die te maken hebben met administratie. Eén requirement was om in een notitie een link op te kunnen nemen en nieuwe regels te kunnen interpreteren. Dit zou betekenen dat ik een tekst parser moest gaan maken. Dit had ik nog nooit eerder gedaan en ik dacht dat dit aardig wat tijd zou gaan innemen. Bij het zoeken naar een AngularJS implementatie kwam ik op de directive 'TextAngular' [11]. TextAngular is een tekst editor voor AngularJS. Het was echter maar de vraag of ik deze wel wilde gaan gebruiken. Binnen Koekoek zou maar een zeer klein onderdeel van de plugin gebruikt worden. Doordat de plugin zoveel meer features aanbood dan ik nodig had werd het een relatief zware dependency, vooral omdat het ook maar op een enkele plek gebruikt zou worden. Toch heb ik deze plugin uiteindelijk gebruikt. Niet alleen omdat het me veel tijd scheelde, maar omdat de extra functies ook wel handig waren om de gebruiker meer flexibiliteit te geven. Denk hierbij aan standaard text editor functies. De functies kunnen naar wens aan en uit worden gezet. Uiteindelijk heb ik TextAngular ook niet op één plek gebruikt, zo wordt het nu ook gebruikt voor applicatie eigenschappen.

Tijdens het eerste deel van het project werd al duidelijk de versies van een server gegroepeerd moesten worden, wat te maken de leesbaarheid van de overzichtspagina. In de toekomst kunnen er veel verschillende versie soorten zijn, om deze goed in een tabel te tonen moet dit op één of andere manier gegroepeerd staan. De ene server draait bijvoorbeeld PostgreSQL, maar de andere MySQL en de andere weer MongoDB. De opdrachtgever wilde niet dat hier op de overzichtspagina drie kolommen voor kwamen in de versie tabel. Door de verschillende database management systemen aan een groep 'Database management systeem' toe te voegen kan je dit tegen gaan. Na een gesprek met de opdrachtgever bleek dat hij zelf de groepen wilde managen en dat het dus niet van te voren vast staat welke versie waar bij hoort. . Hier moest een functionaliteit voor ingebouwd worden. Dus ten eerste moet het mogelijk zijn om versie groepen aan te maken en te kunnen beheren en ten tweede moet een versie gekoppeld kunnen worden aan een groep. Om dit te realiseren is een aparte pagina gemaakt waar de gebruiker dit kan managen. Uit het normalisatie oogpunt is voor de versie groepen is een extra tabel in de database aangemaakt, deze heeft een relatie met de versies.

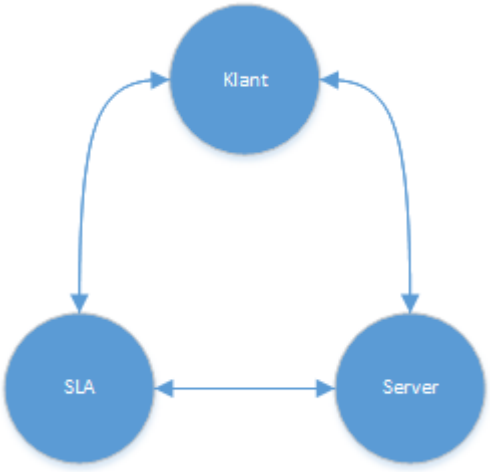
[11] Text angular: http://textangular.com/

Hierdoor moest de back-end ook het één het ander aangepast worden. Hierbij moet gedacht worden aan CRUD functionalteit voor versie groep en het correct opslaan van de nieuwe versiegroep bij een versie.

Verder stonden er nog andere requirements en issues die behandeld moesten worden, zo heb ik in eerdere sprints verteld dat de datum tijd picker niet goed was. Deze vond de gebruiker niet lekker werken. De eerste datum tijd picker had datum en tijd kiezen in één. Omdat dit niet als fijn werd beschouwd heb ik de date picker en time picker van Bootstrap gepakt. Eerst heb ik nog gezocht naar andere datum tijd pickers, maar ik kon geen geschikte vinden die geschreven was in AngularJS. De Bootstrap date en time picker heb ik door middel van een directive samengevoegd naar één inputveld waar je twee popups hebt. In de ene kan je een tijd kiezen en in de andere een datum, hiermee hoop ik de gebruikerservaring te verbeteren.

Daarnaast moest het mogelijk zijn om servers te kunnen activeren en deactiveren. De gebruiker moet dan ook een notitie invullen wanneer deze functionaliteit gebruikt wordt. Dit om aan te geven waarom het nodig was. Hierdoor moest er een boolean toegevoegd worden aan het server object. Hier heb ik hier een speciale beheer pagina voor gemaakt zodat dit centraal geregeld kan worden en je eenvoudig meerdere servers kan activeren/deactiveren zonder veel te navigeren. Eventueel had dit ook bij de server informatie pagina gekund. Hierna was het wel belangrijk dat ik ervoor zorgde dat bijvoorbeeld de overzichtspagina alleen maar actieve servers ophaalt, omdat inactieve servers niet te monitoren zijn. Verder zijn er nog kleine issues opgelost zoals in de status en prestatie tabel waarschuwingskleuren implementeren als bepaalde waardes over een grens gaan. Deze grenswaardes heb ik gehaald uit de Servercheck applicatie van mijn opdrachtgever, omdat deze daar al gedefinieerd stonden.

Als laatste moest de volgende cirkel gecreëerd worden op de pagina waar je klanten en SLA's kan beheren. Zo moest het mogelijk zijn dat om via een klant alle bijbehorende SLA's en servers te zien. Wanneer je vervolgens op een specifiek SLA klikt wordt de server selectie weer verkleind. Klik je vervolgens op een server dan zie je juist welke SLA's en klanten daarbij horen, zo kan de gebruiker eenvoudig zien wat waar bij hoort.



TESTEN

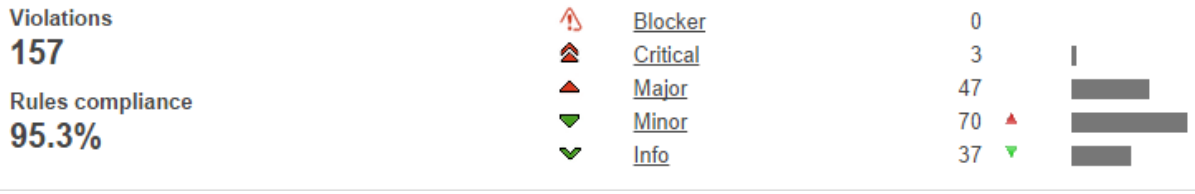
In sprint vier heb ik verteld dat ik mijn groovy script moest herschrijven omdat de testen het anders niet deden. Dit kwam omdat de testen via een in-memory database werken. In sprint vier heb ik toen geen tijd meer gehad om veel te testen en de sprints daarna heb ik hier ook niet aan gewerkt. De oude testen waren ook niet goed meer omdat een groot deel van de applicatie was herschreven, om deze rede heb ik dus ook alle testen verwijderd. Om het project toch goed af te ronden en de quota van 80% test coverage te halen ben ik eind deze sprint weer begonnen met het testen van de applicatie. Hoe getest wordt is eerder te lezen in dit verslag, ik zal dan ook gelijk naar het resultaat

gaan. Het uiteindelijke resultaat van het testen is niet op het niveau van wat het zou moeten zijn. Doordat mijn planning niet klopte en ik hierdoor tijdgebrek kreeg, is een test coverage behaald van 50%.

Hiernaast is er nog gewerkt om de rules compliance op het niveau van 95% of meer te krijgen. De meeste violations waren over missende javadoc dat ik niet goed bijgehouden had. Na dit toe te voegen en andere kleine problemen op te lossen had ik een rules compliance van 95.3%. Het verwachtte resultaat is dus behaald.

In de volgende figuren ziet u het resultaat van de test coverage en rules compliance. Zoals te zien is in het tweede figuur zijn de violations verdeeld in verschillende niveaus. Hierbij is critical de belangrijkste en deze tellen ook hert zwaarste. Bij major violations moet gedacht worden aan methode namen die teveel op elkaar lijken. Veel van deze violations zitten ook in de 'N2 API'. Ik heb wel geprobeerd deze op te lossen, maar ik wilde hier niet te veel tijd aan besteden., omdat de API niet in de scope van mijn project stond. Minor en info is voornamelijk Javadoc dat niet in het goede format staat, deze tellen ook bijna niet mee voor je rules compliance.

Figuur 17 Test coverage resultaat



Figuur 16 Rules compliance resultaat

Reflectie

Zoals eerder beschreven, stond deze sprint in het teken van het afronden van mijn project. Dit betekende voor mij dat de hoogste prioriteit, de administratie, moest werken nadat ik klaar was met deze sprint en niet onbelangrijk, dat de applicatie Confluence moest kunnen vervangen. Ik heb hier dus de focus op gelegd en ik vind dat dit goed gelukt is. Alle functionaliteiten van Confluence zitten nu in Koekoek verwerkt. Hetgene wat hier wel onder geleden heeft is het testen. Het resultaat is niet op het gewenste resultaat, maar ik kon hier ook niet langer meer aan werken dan ik heb gedaan om niet in de problemen te komen met mijn verslag.

PRODUCTEVALUATIE

Over het product zelf ben ik tevreden, het product wijkt wel af met wat als eerste instantie is beschreven in de opdrachtomschrijving. In de opdrachtomschrijving stond dat ik een tool zou opleveren die de servers van 42 zou monitoren. De applicatie doet dit wel, alleen kan het bijvoorbeeld nog niet met meerdere bronnen omgaan en word de data nog niet bewerkt. De reden hiervan was dat de wensen van de opdrachtgever tijdens het project meer richting de administratieve functie van de applicatie gingen. Maar er is wel over nagedacht hoe dit in de toekomst geïmplementeerd moet worden.

Over de werking van de applicatie ben ik zeer tevreden. Ik denk dat de applicatie nu goed alle informatie kan opslaan die op dit moment via administratie programma Confluence wordt opgeslagen. Ook er zit een goede basis in om de verschillende parameters die een server heeft te monitoren. Waar ik minder tevreden over ben is de code in de back-end. Op sommige plekken is dit nog erg rommelig en inconsistent, ik had niet veel tijd vrijgemaakt om oudere code te refactoren en had meer focus op nieuwe functionaliteit. Maar met de ervaring die ik nu heb zou ik de back-end code een stuk duidelijker en schoner kunnen maken. Dit ben ik ook zeker nog van plan te doen. Als laatste ben ik ontevreden over het test resultaat.

Over de kwaliteiten van de testen zelf ben ik tevreden. Ik heb bij de testen mocking toegepast om elke klasse individueel te testen en niet afhankelijk te maken van andere klassen, wat de wens was voor dit project. Over het resultaat ben ik iets minder tevreden, simpelweg omdat ik niet gehaald heb wat er van mij verwacht werd.

Ondanks deze mindere punten is de opdrachtgever tevreden met het geleverde resultaat en zal de ontwikkeling van Koekoek hierna niet stil komen te liggen.

BEROEPSTAKEN

In dit hoofdstuk zal in worden gegaan op de verschillende beroepstaken waar ik aan moest voldoen gedurende dit project. Per beroepstaak zal aangegeven worden wat ik heb gedaan om aan de beroepstaak te voldoen.

UITVOEREN ANALYSE DOOR DEFINITIE VAN REQUIREMENTS

Tijdens het begin van het project heb ik als eerste de bestaande requirements gecontroleerd of deze nog klopte. Begin sprint vier ben ik begonnen met het opstellen van extra requirements, omdat ik tot deze sprint alleen maar bezig was om bestaande functionaliteit te verbeteren. Hiernaast was het niet mogelijk om alle requirements aan het begin samen te stellen door de veranderende wensen van de opdrachtgever. Tijdens sprint vier heb ik de requirements samen gesteld uit informatie dat mij was gegeven uit eerdere gesprekken met de opdrachtgever. Tijdens deze gesprekken heb ik toekomstige wensen/requirements genoteerd, deze zijn pas nu verwerkt, omdat ik zoals verteld nog bezig was met bestaande functionaliteit verbeteren. Deze heb ik vervolgens laten valideren door mijn opdrachtgever. Omdat er per sprint requirements bij konden komen heb ik na iedere sprint demo de informatie geanalyseerd en omgezet naar nieuwe requirements indien er nieuwe informatie was. Hiernaast heb ik, indien het zinvol was, use-cases gemaakt om te verduidelijken hoe een requirement geïmplementeerd moet worden. Hiermee denk ik aan deze beroepstaak voldaan te hebben.

ONTWERPEN, BOUWEN EN BEVRAGEN VAN EEN DATABASE

Tijdens het project is de database meerdere malen veranderd. Hierbij is nagedacht over normalisatie en welke data relevant is om op slaan. De database zelf is gemaakt door middel van liquibase en groovy scripts. Deze tool zorgt ervoor dat je eenvoudig veranderingen in een database kan maken en dit ook bijgehouden wordt. Het bevragen van de database is gedaan door middel van een JPA Interface voor de standaard/basis query's. Denk hierbij aan CRUD functionaliteit. Hiernaast zijn er ook zelf ingewikkeldere query's geschreven als ik meer functionaliteit nodig had dan de JPA interface aanbood.

ONTWERPEN SYSTEEMDEEL

Tijdens het ontwerpen van de applicatie heb ik er rekening mee gehouden dat de applicatie goed object georiënteerd was. Ik heb gelet op het gebruik van abstracties en interfaces waar dit nuttig was. Ik heb continu gekeken tijdens het project of ik de applicatie hierop kon verbeteren. Hiernaast is er ook nagedacht over systeemdelen die in de toekomst nog geïmplementeerd moeten worden in de applicatie. Bijvoorbeeld hoe de metingen van de servers verwerkt en bewerkt moeten worden.

Het front-end ontwerp is gebaseerd op andere applicaties binnen 42, deze waren gemaakt door een front-end ontwikkelaar. Dit om zoveel mogelijk cohesie te creëren tussen de verschillende applicaties, wat de gebruiksvriendelijkheid ten goede komt. Verder geld dat ik voor diverse onderdelen verschillende opties heb overwogen en ik in overleg met de opdrachtgever of front-end ontwikkelaar de beste van die opties heb gekozen.

BOUWEN APPLICATIE

Aan de hand van de ontwerpen heb ik de applicatie gebouwd. Dit betreft de front- en back-end. Tijdens het ontwikkelen heb ik rekening gehouden met de complexiteit en uitbreidbaarheid van de code, maar ook aan de rules compliance. Hierdoor is er soms een refactoring van bestaande code uitgevoerd om de kwaliteit en leesbaarheid hiervan te verbeteren. Als ik tegen een probleem aanliep heb ik eerst bekeken wat de opties waren om dit op te lossen en wat de beste hier van was. Tevens is een groot deel van de code gedocumenteerd met Javadoc om het voor een toekomstige ontwikkelaar makkelijker te maken om de code eigen te maken.

Hiernaast heb ik de front-end code voor zichzelf laten spreken en zo clean mogelijk proberen te houden. Dit heb ik bereikt door het gebruik maken van AngularJS directives, hierdoor verminder je code duplicatie, omdat je herbruikbare web componenten bouwt. Hiernaast heb ik de HTML voor zichzelf laten spreken, dit betekent dat ik niet in de controllers of directives een deel van de DOM verander zonder dat dit duidelijk is. Ook heb ik dit bereikt door de AngularJS structuur aan te houden, zoals het data ophalen in de services.

UITVOEREN VAN EN RAPPORTEREN OVER HET TESTPROCES

Gedurende het project heb ik unit testen geschreven om de gemaakte functionaliteit te kunnen testen. Deze testen zijn bijgewerkt nadat ik een deel van de applicatie had herschreven. Voor het testen is ook gebruik gemaakt van het JMockit framework. Hierdoor kon ik zorgen dat elke klasse zo geïsoleerd mogelijk getest kon worden. Dit zorgt ervoor dat de testen minder vaak aangepast hoeven worden. Aan het einde van dit project heb ik een test coverage gehaald van 50%. Dit mede door het opnieuw moeten schrijven van de meeste testen door het refactoren van code. Op het einde heb ik een testrapport geschreven waarin wordt verteld wat er van mij verwacht werd, de uitvoering, het proces en de werkelijke resultaten.

BEHEREN EN DISTRIBUEREN VAN SOFTWARE

Tijdens het project was ik zelf verantwoordelijk voor het opzetten van een werkende versie van de applicatie op de testserver. Dit moest elke sprint worden gedaan. Hiernaast heb ik geholpen met het deployen van de applicatie op de servers voor de Haagse Hogeschool. Elke oude versie van de applicatie is bewaard in de repository van 42 om altijd snel terug te kunnen naar een oude versie. Dit voor het geval er iets mis mocht zijn. Voor dit proces is een deployment plan gemaakt. Hierin worden de stappen beschreven die uitgevoerd moeten worden om de applicatie op een server te deployen.

BIJLAGEN

BIJLAGE 1, OPDRACHTOMSCHRIJVING

Afstudeerplan
Informatie afstudeerder en gastbedrijf

Afstudeerblok: 2014-1.2 (start uiterlijk 12 mei 2014)
Startdatum uitvoering afstudeeropdracht: 12-05-2014
Inleverdatum afstudeerdossier volgens jaarrooster: 6 oktober 2014

Studentnummer: 10014845
Achternaam: dhr Klein
Voorletters: B
Roepnaam: Bastiaan
Adres: Buffelstraat 89
Postcode: 3064AA
Woonplaats: Rotterdam
Telefoonnummer: 010-4422293
Mobiel nummer: 06-28810382
Privé emailadres: bastiaanklein93@hotmail.com

Opleiding: Informatica
Locatie: Zoetermeer
Variant: voltijd

Naam studieloopbaanbegeleider: Vincent Broeren
Naam begeleidend examinator: Tim Cocx
Naam tweede examinator: Emeri Koenen

Naam bedrijf: 42 Enterprise Software Specialists
Afdeling bedrijf: -
Bezoekadres bedrijf: Koraalrood 33
Postcode bezoekadres: 2718 SB
Postbusnummer: 6003
Postcode postbusnummer: 2702 AA
Plaats: Zoetermeer
Telefoon bedrijf: +31 88 4242 042
Telefax bedrijf: +31 88 4242 099
Internetsite bedrijf: http://42.nl/

Achternaam opdrachtgever: dhr Bor
Voorletters opdrachtgever: RA
Titulatuur opdrachtgever: Ing.
Functie opdrachtgever: CTO
Doorkiesnummer opdrachtgever:
Email opdrachtgever: robert.bor@42.nl

Achternaam bedrijfsmentor1: dhr Bor
Voorletters bedrijfsmentor1: RA
Titulatuur bedrijfsmentor1: Ing.
Functie bedrijfsmentor1: CTO
Doorkiesnummer bedrijfsmentor1:
Email bedrijfsmentor1: robert.bor@42.nl

Doorkiesnummer afstudeerder:
Functie afstudeerder (deeltijd/duaal):

Titel afstudeeropdracht:
Het door ontwikkelen en het verbeteren van de kwaliteit van de server monitor.

Opdrachtomschrijving

1. Bedrijf

42 is in 2003 opgericht, en richtte zich toen hoofdzakelijk op ontwerp en ontwikkeling voor het Java platform. De laatste jaren is dat breder geworden, zo wordt inmiddels ook voor iOS ontwikkeld. Daarnaast is er een tak die zich bezighoudt met beheer van cloud-applicaties en -systemen.

42 is inmiddels ongeveer 40 man groot, en heeft op dit moment één vestiging, namelijk in Zoetermeer. De markt voor 42 is software-enigineering en -beheer op de platforms die 42 ondersteunt. Daarbinnen is geen specifieke focus op een bepaalde branch of sector.

42 zet zich vaak in voor Open Source en het bedrijf heeft inmiddels meerdere Open Source libraries op de wereld gezet.

2. Probleemstelling

42 heeft op dit moment op verschillende plekken remote servers draaien. De meeste staan bij cloudVPS. Deze zijn te monitoren via een monitoringsprogramma van cloudVPS zelf. Het probleem is dat deze monitor veel bugs bevat en het daardoor vaak lastig is om de juiste data af te lezen. Hierdoor kan niet goed gekeken worden of een server nog goed draait en wanneer een probleem is ontstaan. Ook laat deze monitor niet alle gegevens zien die gewenst zijn, zoals bijvoorbeeld het versienummer van de applicatie. Tenslotte wil 42 zich niet beperken tot één enkele hosting provider. In de huidige situatie impliceert dat dat automatisch gebruik gemaakt moet gaan worden van meerdere monitoring tools, namelijk voor iedere provider.

3. Doelstelling van de afstudeeropdracht

De doelstelling is om de monitor die op dit moment in ontwikkeling is door te ontwikkelen en de kwaliteit hiervan te verbeteren. De bestaande monitoring software is geschreven door twee stagiaires van de Haagse Hogeschool, te weten Tom van den Bulk en Kevin van Leeuwen. Zij hebben onderling de taken verdeeld in front- en backend, terwijl ikzelf beide aspecten zal moeten waarnemen. De bestaande software is nog niet operationeel. Met het uiteindelijke tool kan 42 de servers beter monitoren dan met het monitoringsprogramma van de VPS aanbieder zelf.

4. Resultaat

Het verwachtte resultaat is een werkende monitor die gebruikt kan worden om de diverse remote servers die het bedrijf heeft, te kunnen monitoren. Deze monitor moet verschillende parameters uitlezen, zoals bv CPU, load, network traffic en aantal processen. Parameters die afkomstig zijn uit de API(s) van de hosting provider(s) en parameters die de servers zelf doorsturen naar de database van 42.

5. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

Het bedrijf 42 werkt met de ontwikkelingsmethode Scrum. Dit is dus ook de ontwikkelingsmethode die ik zal gebruiken bij deze opdracht. Een sprint zal bestaan uit 10 werkdagen (2 weken). Per sprint zal het volgende opgeleverd worden:

- Demo met werkende versie van de software
- Retrospective
- Sprint planning

De volgende technieken zullen aan bod komen om de front-end van de applicatie te ontwikkelen.

- Hightcharts
- AngularJS
- HTML5

- CSS3
- Bower/Grunt (NTB)

De volgende technieken zullen aan bod komen om de backend van de applicatie te ontwikkelen.

- Spring
- Java
- JPA
- Hibernate
- Maven
- Junit
- JMockit
- PostgreSQL

Omdat zoals eerder gezegd de API van de hostingprovider niet alle gewenste gegevens kan aanbieden zal er ook nagedacht moeten worden over een service/programma die zelf alle gegevens pushed naar de centrale database/server van 42. Deze applicatie moet met een aantal dingen rekening houden. Zo moet als de master server er uit is, of als de applicatie voor één of andere reden geen connectie kan maken met de master server, zijn data op kunnen slaan. Deze buffer moet zodra de connectie weer beschikbaar is naar de server gestuurd worden. Ook zal de master server zelf een server toe moeten voegen zodra een server zich aanmeldt bij de master server. Hierdoor moet aan de andere kant rekening gehouden worden met een stukje beveiliging. Hoe wordt er voor gezorgd dat een nep applicatie geen connectie kan maken met de master server en neppe data pushen. Dus hoe indentificeren we of deze applicatie bij één van onze server hoort. Deze applicatie zal in tijdsintervallen data sturen. Er zal nagedacht moeten worden wat er gebeurd als de tijdsintervallen niet meer kloppen. Dus geen data elke minuut. Maar één keer om de minuut en dan opeens om de tien seconden. Natuurlijk moet er ook te zien zijn/een waarschuwing komen als een server er mogelijk uit ligt.

Verder moet gekeken worden hoe alle data wordt verzameld. Het ideale beeld is dat de data zo klaar staat dat het één op één naar de applicatie gepushed kan worden vanuit de server. Maar alle data die de servers sturen moet wel bewaard blijven. Er moet dus nagedacht worden over een universele databasestructuur die alle unieke aspecten van de verschillende bronnen behoud, maar voor elke bron de data kan opslaan die nodig is. Hierna zal de data bewerkt moeten worden, zodat het met zo min mogelijk bewerking naar de applicatie gepushed kan worden.

6. Op te leveren (tussen)producten

- Plan van aanpak
- Productbacklog
- Waar nodig documentatie van het systeem
    - Beschrijving architectuur
    - Deployment handleiding
    - Rationale document
    - Database ontwerp
    - Systeemontwerp, denk hierbij aan klassediagrammen en sequencediagrammen

7. Te demonstreren competenties en wijze waarop

1.4 Uitvoeren analyse door definitie van requirements:
De bestaande requirements zullen ten eerste geanalyseerd worden. Deze zullen opnieuw gevalideerd worden en zo nodig aangevuld. Daarna moet er gekeken worden welke al geïmplementeerd zijn en welke niet.

2.2: Ontwerpen, bouwen en bevragen van een database:
De bestaande database zal moeten uitgebreid en aangepast worden om de gegevens op te slaan die de opdrachtgever wilt. Tevens zal deze bevraagd worden door de back-end en zal er data inkomen van de applicatie die op de servers draait.

3.2: Ontwerpen systeemdeel:
Er zullen op het moment dat ik begin al systeemontwerpen zijn. Deze zullen zo nodig uitgebreid moeten worden. Tevens zullen er nieuwe gemaakt worden van de nieuwe software en van bestaande software dat indien nodig her geschreven zal moeten worden.

3.3: Bouwen applicatie:
Er zal door ontwikkeld moeten worden die de statussen van de verschillende virtuele servers van 42 laat zien. Hiervoor zal de front-end en de back-end aangepast moeten worden. Dit zal door mij ontwikkeld worden. Tevens moet er een applicatie ontwikkeld worden die data van de servers naar de database van 42 pushed.

3.5: Uitvoeren van en rapporteren over het testproces:
Bij 42 wordt er gewerkt volgens Test Driven Development. Dit houdt in dat er voor alle code unit tests geschreven moeten worden. Deze unit tests dienen vervolgens als hun eigen documentatie.

4.4: Beheren en distribueren van software:
Het uitrollen van nieuwe versies op de testomgeving zal door mij worden uitgevoerd.

BIJLAGE 2, PLAN VAN AANPAK

42

Koekoek

VPS Monitorings tool

Bastiaan Klein

20-6-2014

INHOUD

1. INLEIDING

Steeds meer bedrijven minder bedrijven hebben hun servers op fysieke servers staan binnen hun bedrijf. Bijna alles wordt gehost op servers bij de grote aanbieders. Denk hierbij bijvoorbeeld aan amazon. Dit is handig zodat je zelf niet meer de servers hoeft te draaien. En aangezien het altijd op de Cloud staat kan je zoveel capaciteit vragen als je wilt.

Als je het bij andere bedrijven hebt staan, betekend niet dat je niet wilt weten hoe het ervoor staat met je server. Gelukkig bieden deze aanbieders meestal wel een tool aan waarmee je jou servers kan monitoren. Dit geeft alleen weer het probleem dat je meerdere tools moet gebruiken zodra je bij meerdere providers servers hebt en dit is natuurlijk onhandig. Dit probleem gaan we aanpakken door alles samen te brengen naar één monitor.

Ten eerste zal de projectachtergrond besproken worden. Hierna word in de projectdefinitie de probleemstelling, doelstelling en projectdossier besproken. Verder worden de project organisatiestructuur behandeld en het project quality plan. Als laatste zullen de risico’s behandeld worden en een projectplanning.

2. PROJECTACHTERGROND

In dit hoofdstuk zal beschreven worden welke onderdelen allemaal uitgevoerd moeten worden in dit project. Tevens zal de noodzaak tot het opzetten van het project in dit hoofdstuk beschreven worden.

Uitleg van de context

Voor dit project zullen de volgende onderdelen opgeleverd worden:

- Planning per sprint
- Ontwerp database
- Functioneel ontwerp
- Technisch ontwerp
- Ontwikkelen applicatie
- Testrapportage
- Deployment handleiding
- Beschrijving architectuur

Planning per sprint

Per sprint zal een planning opgeleverd om te laten zien wat ik die sprint zal doen. Aan het eind zal blijken of ik deze planning behaald heb.

Ontwerp database

Het huidige ontwerp van de database zal geanalyseerd worden en zo nodig aangepast.

Functioneel ontwerp

In het functioneel ontwerp zullen alle requirements en functioneel ontwerp beschreven worden.

Technisch ontwerp

In het technisch ontwerp zullen verschillende diagrammen worden opgeleverd. Zo zal het technisch ontwerp bestaan uit design klassediagram van de applicatie. En zo nodig sequentie diagrammen.

Testrapportage

Dit zal gebeuren door unit testen te schrijven tijdens het coderen.

Deployment handleiding

In deze handleiding zal komen te staan wat er moet gebeuren als de applicatie naar live omgeving wordt gezet. Tevens zal deze bevatten wat er moet gebeuren als de applicatie geupdate word.

Noodzaak tot opzetten

Op dit moment heeft het bedrijf 42 bij verschillende hosting servers draaien voor klanten. Dit is handig, zodat je ze zelf niet meer fysiek hoeft te hosten. Maar ook al staan je servers ergens anders wil je alsnog kunnen monitoren om te kijken hoe ze het doen. Een probleem is alleen dat elke hosting provider hiervoor zijn eigen tool heeft en zelf bepaald welke data deze tool kan aanleveren.

Dit wil je niet, je wilt niet dat als je bij drie verschillende providers servers hebt je op drie verschillende tools moet kijken hoe het in elkaar zit. Daarom is er een monitor bedacht om alle servers bij elkaar te monitoren. Deze monitor zal ook extra informatie geven die de hosting providers zelf niet geven.

3. PROJECTDEFINITIE

In dit hoofdstuk wordt beschreven waarom het project opgezet is. Dus wat is het huidige probleem en wat is de doelstelling aan het einde van het project. Verder zullen alle documenten die aan het einde van het project opgeleverd worden neergezet worden.

Probleemstelling

Het probleem van dit project is dat het bedrijf 42 niet voor elke hosting provider die zij hebben een monitoring tool willen gebruiken. Dit is handig, omdat je liever alles centraal wilt hebben. Tevens leveren deze providers niet alle data die het bedrijf wilt monitoren. Denk hierbij aan het versienummer van de applicaties er draaien of de java versie enz.

Doelstelling

Het doel van dit project is om een monitor te ontwikkelen die alle gegevens van de virtuele servers naar één centrale plek voor 42 brengt. Hierdoor kan het bedrijf 42 op een centrale plek al hun servers monitoren en zo nodig ingrijpen. Tevens zal deze monitor data bevatten die de api's van de hosting providers niet aanleveren. Hierdoor zal alles centraal te zien zijn en kan iedereen ingrijpen indien nodig.

Project dossier/Scope

De volgende producten zullen opgeleverd worden aan de opdrachtgever:

- Planning per sprint
- Ontwerp database
- Functioneel ontwerp
- Technisch ontwerp
- Source code
- Testrapportage
- Deployment handleiding
- Beschrijving architectuur

4. PROJECT QUALITY PLAN

In dit hoofdstuk ga ik beschrijven wat de kwaliteitsverwachtingen zijn en hoe ik denk die kwaliteit te waarborgen.

Kwaliteitswaarborging

Om de kwaliteit te kunnen garanderen houd ik mij aan de volgende dingen:

- Veelvuldig communicatie met de opdrachtgever.
- Veelvuldig communicatie met de gebruikers.
- Elke 2 weken een review van mijn code en voortgang.
    - Op basis van de review kunnen er punten uit komen om de code te verbeteren.
- Test driven development.

5. RISICO'S

In dit hoofdstuk zullen de risico's op voorhand naar voren komen en wat ik kan doen om deze risico's op voorhand te vermijden.

Risico	1 – Teveel code moeten herschrijven
Omschrijving	Indien er in het huidige project te veel 'slechte' code is zal deze herschreven moeten worden.
Gevolgen	Indien er te veel code herschreven moet worden is er minder tijd tot het afmaken van het project, waardoor er een incompleet product opgeleverd zal worden.
Kans	De kans dat dit gebeurd is niet goed te voorspellen. Door eerdere ervaringen met mijn mede studenten denk ik dat de kans matig tot hoog is.
Risiconiveau	Matig - Hoog
Oplossing	Door eerst een code review te doen over heel de code. Hier moet ik al mijn ervaringen opschrijven en kijken wat relevant is om helemaal te herschrijven en wat niet. Dit in overleg met een senior java programmeur.

Risico	2 - Documentatie & afstudeerverslag bijhouden
Omschrijving	Vanuit 42 is er enkel vraag naar een haalbaarheidsstudie en een ontwerp voor de API. Echter wordt vanuit school verwacht dat er toch nog een hoop andere documentatie opgeleverd wordt.
Gevolgen	Indien ik teveel de focus leg op het ontwikkelen, omdat dit van mij verwacht wordt vanuit 42, zal ik achter komen te liggen op documentatiegebied en zal mijn uiteindelijke oplevering onvoldoende zijn vanuit het oogpunt van school.
Impact	Indien de documentatie en het verslag niet goed bijgehouden worden zal dit een zeer grote impact hebben op het uiteindelijke resultaat van het afstuderen.
Risiconiveau	Hoog
Oplossing	Door per week een vaste hoeveelheid tijd te besteden aan het maken/bijwerken van documentatie kan dit probleem voorkomen worden.

Risico	3 – Niet afwijken van het doel van het project
Omschrijving	Het doel van het project is om de kwaliteit van de code te verbeteren en de server monitor af te maken. Met de functionaliteiten die de opdrachtgever wilt. Dus met alle 'must-have' requirements.
Gevolgen	Indien er teveel tijd gestoken wordt in functionaliteiten die niet relevant zijn voor de server monitor. Maar meer aan de 'nice-to-have' dingen zal er minder tijd over zijn voor de requirements waar de applicatie echt aan moet voldoen. En dus zal deze niet af zijn aan het einde.
Impact	Indien het doel van het project niet bereikt wordt betekend dit dat 42 meer resources zal moeten gebruiken om dit doel alsnog te bereiken. Ook zou het een impact kunnen hebben op de beoordeling van mijn werk. Hiermee ligt de totale impact van dit risico hoog.
Risiconiveau	Matig
Oplossing	Door goed naar de productbacklog te kijken en binnen de planning de focus te leggen op vereiste onderdelen kan voorkomen worden dat er aan niet relevante functionaliteiten gewerkt wordt.

6. PLANNING

In dit hoofdstuk zal de planning van het project te vinden zijn.

Week	Periode	Activiteit(en)
1	12 t/m 16 mei	-Opstellen ontwikkelomgeving -Plan van Aanpak -Inwerken in spring/huidige code
2 t/m 14	19 mei t/m 15 september	-Ontwikkelen van server monitor in zes sprints*
15 t/m 17	15 september t/m 6 oktober	-Uitloopweken (extra ruimte) -Afronden afstudeerverslag

*Voor elke sprint zal een aparte planning gemaakt worden met taken die, die sprint uitgevoerd zullen worden. Elke sprint duurt 2 weken.

BIJLAGE 3, REQUIREMENTS

Koekoek

VPS MONITORINGS TOOL

BASTIAAN KLEIN

REQUIREMENTS

Nummer	Beschrijving	JIRA	Use-case
KOEK-1.	De user kan een lijst bekijken met de huidige status van alle servers.	KOEK-135	UC-1
KOEK-2.	De user kan een lijst bekijken met de huidige versies van alle servers.	KOEK-136	UC-1
KOEK-3.	De user kan een lijst bekijken met server informatie van alle servers.	KOEK-137	UC-1
KOEK-4.	Het systeem moet alle oorspronkelijke data van de variabelen gegevens opslaan	KOEK-138	-
KOEK-5.	Het systeem moet alle serverdata naar de zelfde tijdseenheid omzetten	KOEK-139	-
KOEK-6.	Het systeem moet per genormaliseerde data record opslaan wat de uiteindelijke waardes worden die getoond worden.	KOEK-140	-
KOEK-7.	Het systeem moet van de statische gegevens alleen de veranderingen opslaan.	KOEK-141	-
KOEK-8.	Het systeem moet van de statische gegevens de history bijhouden.	KOEK-142	-
KOEK-10.	De user kan verschillende variabelen gegevens van verschillende bronnen met elkaar vergelijken.	KOEK-144	UC-2
KOEK-11.	De user kan de history van users bekijken van een server	KOEK-145	UC-3
KOEK-12.	De user kan de history van versies bekijken van een server.	KOEK-146	UC-3
KOEK-13.	De user kan de history van de variabelen gegevens bekijken van een server	KOEK-147	UC-4
KOEK-14.	Het systeem mag niet stuk gaan als het foutieve of oudere JSON ontvangt	KOEK-148	-
KOEK-15.	Het systeem moet bij een foutieve of oudere JSON post zo veel mogelijk data proberen op te slaan.	KOEK-149	-
KOEK-16.	Het systeem moet een waarschuwing geven als het foutieve of oudere JSON verwerkt heeft.	KOEK-150	-
KOEK-17.	Het systeem moet alle informatie kunnen tonen van een server.	KOEK-151	UC-13
KOEK-18.	Het systeem moet waarschuwingen genereren wanneer het verschil tussen gegevens een bepaalde limiet bereikt.	KOEK-152	-
KOEK-19.	Het systeem moet automatisch een nieuwe server toevoegen als men hier voor het eerst data van krijgt	KOEK-153	-
KOEK-20.	Een user moet de informatie van een server kunnen bijwerken.	KOEK-154	UC-4
KOEK-21.	Het systeem moet CRUD functionaliteiten hebben voor Customers	KOEK-155	UC-5/ UC-6
KOEK-22.	Het systeem moet CRUD functionaliteiten hebben voor Suppliers	KOEK-156	UC-5/ UC-6
KOEK-23.	Het systeem moet CRUD functionaliteiten hebben voor SLA	KOEK-157	UC-5/ UC-6
KOEK-24.	Het systeem moet achteraf opnieuw alle metingen kunnen aggregeren (uit de ruwe data in het systeem)	KOEK-158	-
KOEK-25.	De user moet een notitie van een gebeurtenis kunnen aanmaken voor een server, deze moeten CRUD functionaliteit hebben.	KOEK-159	UC-5/ UC-6
KOEK-26.	Het systeem moet automatisch als er x tijd geen data uit bron x is gekomen de gegevens van bron y gebruiken.	KOEK-160	-
KOEK-27.	Het systeem zal op een tomcat server moeten kunnen draaien.	KOEK-161	-
KOEK-28.	Het systeem zal gebruik maken van een postgreSQL database	KOEK-162	-
KOEK-29.	Rules compliance moet 95% of hoger zijn	KOEK-163	-
KOEK-30.	Code test coverage moet 95% of hoger zijn	KOEK-164	-
KOEK-31.	De user kan een lijst bekijken met alle notities die aangemaakt zijn.	KOEK-165	UC-7
KOEK-32.	Er moet een mogelijkheid zijn in te kunnen loggen.	KOEK-166	UC-8
KOEK-33.	Er moet een mogelijkheid zijn uit te kunnen loggen.	KOEK-167	UC-9
KOEK-34.	De user moet een SSH sessie kunnen openen naar een server	KOEK-168	UC-10

Nummer	Beschrijving	JIRA	Use-case
KOEK-35.	De user moet standaard/bekende users kunnen aangeven in de applicatie.	KOEK-172	UC-11
KOEK-36.	De user moet standaard/bekende iptables knnen aangeven in de applicatie	KOEK-173	UC-11
KOEK-37.	De user moet servers op actief/non-actief kunnen zetten.	KOEK-187	UC-12
KOEK-38.	De user moet een lijst van applicatie informatie/eigenschappen kunnen zien.	KOEK-190	UC-13
KOEK-39.	De user kan een lijst bekijken met alle prestatie informatie van alle servers	KOEK-191	UC-1
KOEK-40.	Het systeem moet CRUD functionaliteit hebben op versie groepen	KOEK-192	UC-5/ UC-6
KOEK-41.	De user moet zelf een applicatie versie aan een versie groep kunnen koppelen	KOEK-193	UC-14
KOEK-42.	Het systeem moet CRUD functionaliteit hebben voor applicatie informatie/eigenschappen	KOEK-194	UC-5/ UC-6

BIJLAGE 4, USE-CASES

Koekoek

VPS MONITORINGS TOOL

BASTIAAN KLEIN

Naam	Lijst bekijken van server gegevens
ID	UC-1
Omschrijving	De gebruiker kan een lijst bekeken met gegevens van de server.
Requirements	KOEK-1 / KOEK-2 / KOEK-3
Actor	Gebruiker
Pre-condities	De gebruiker is ingelogd
Flow	1. De gebruiker selecteert 'Servers' in de header 2. De gebruiker selecteert welk lijst hij wilt bekijken. 3. Het systeem laat een lijst zien van de gewenste data.
Post-conditie	-

Naam	Variabelen/prestatie gegevens van verschillende servers vergelijken
ID	UC-2
Omschrijving	De gebruiker kan variabelen/prestatie gegevens van verschillende servers vergelijken
Requirements	KOEK-10
Actor	Gebruiker
Pre-condities	De gebruiker is ingelogd
Flow	1. De gebruiker selecteert 'Vergelijk data' om de header 2. De gebruiker selecteert van welke bron er data getoond moet worden. 3. De gebruiker selecteert een server 4. De gebruiker selecteert een metriek. 5. De gebruiker drukt op 'Voeg toe' 6. Het systeem laad de data in een grafiek.
Post-conditie	-

Naam	Geschiedenis bekijken
ID	UC-3
Omschrijving	De gebruiker kan de geschiedenis van users en versies bekijken van een server
Requirements	KOEK-11 / KOEK-12
Actor	Gebruiker
Pre-condities	De gebruiker is ingelogd De gebruiker bevind zich op de server informatie pagina
Flow	1. De gebruiker selecteert een tab. User of versies 2. De gebruiker selecteert ''Toon geschiedenis" 3. Het systeem laad de geschiedenis zien van de geselecteerde gegevens.
Post-conditie	-

Naam	Geschiedenis bekijken variabelen/prestatie gegevens
ID	UC-4
Omschrijving	De gebruiker kan de geschiedenis van variabelen/prestatie gegevens bekijken van een server
Requirements	KOEK-13
Actor	Gebruiker
Pre-condities	De gebruiker is ingelogd De gebruiker bevind zich op de server informatie pagina
Flow	1. De gebruiker selecteert “metrieken”. 2. De gebruiker selecteert een bron 3. De gebruiker selecteert een datum tijd. 4. De gebruiker druk op “Data” 5. Het systeem laad de gegevens in van die datum tijd.
Post-conditie	-

Naam	Server informatie bijwerken
ID	UC-4
Omschrijving	De gebruiker kan server informatie bijwerken
Requirements	KOEK-20
Actor	Gebruiker
Pre-condities	De gebruiker is ingelogd De gebruiker bevindt zich op de server informatie pagina
Flow	1. De gebruiker vult de gewenste gegevens in 2. De user drukt op “Opslaan” 3. Het systeem slaat de server op 4. Het systeem toont een melding dat de server is opgeslagen
Post-conditie	De server is geüpdate in de database
Alternative-flow	1. Het systeem geeft aan dat de gegevens niet valide zijn.

Naam	Stamtabel rij maken en updaten
ID	UC-5
Omschrijving	De gebruiker kan diverse stamtabellen beheren. Hier aanmaken en updaten
Requirements	KOEK-21 / KOEK-22 / KOEK-23 / KOEK-25 / KOEK-40 / KOEK-42
Actor	Gebruiker
Pre-condities	De gebruiker is ingelogd De gebruiker is op de gewenste beheer pagina
Flow	1. De gebruiker drukt op “Nieuw” of “Edit” 2. Het systeem toont een popup 3. De gebruiker vult de gewenste gegevens in 4. De gebruiker drukt op opslaan 5. Het systeem slaat het gewenste object op 6. Het systeem haalt de popup weg. 7. Het systeem laat een melding zien dat het object is opgeslagen
Post-conditie	Het object is aangemaakt/geüpdate in de database
Alternative-flow	1. Het systeem geeft aan dat de gewenste gegevens niet valide zijn De flow gaat verder vanaf stap 2.

Naam	Stamtabel rij verwijderen
ID	UC-6
Omschrijving	De gebruiker kan diverse stamtabellen beheren. Hier het verwijderen
Requirements	KOEK-21 / KOEK-22 / KOEK-23 / KOEK-25 / KOEK-40 / KOEK-42
Actor	Gebruiker
Pre-condities	De gebruiker is ingelogd De gebruiker is op de gewenste beheer pagina
Flow	1. De gebruiker drukt op “Verwijderen” . 2. Het systeem verwijderd het object uit de database. 3. Het systeem toont een melding dat het object met succes is verwijderd.
Post-conditie	Het object is verwijderd.
Alternative-flow	1. Het systeem geeft een melding dat het object niet verwijderd kan worden.

Naam	Lijst bekijken van alle notities bekijken
ID	UC-7
Omschrijving	De gebruiker kan een lijst bekeken met alle notities van alle servers
Requirements	KOEK-31
Actor	Gebruiker
Pre-condities	De gebruiker is ingelogd
Flow	1. De gebruiker selecteert ‘Activitystream’ in de header 2. Het systeem laat een lijst zien van de gewenste data.
Post-conditie	-

Naam	Inloggen
ID	UC-8
Omschrijving	De gebruiker kan inloggen in het systeem
Requirements	KOEK-32
Actor	Gebruiker
Pre-condities	
Flow	1. De gebruiker voert zijn username en wachtwoord in. 2. De gebruiker drukt op inloggen 3. Het systeem controleert de gegevens 4. Het systeem navigeert de gebruiker naar de activitystream pagina
Post-conditie	De gebruiker is ingelogd
Alternative-flow	1. Het systeem laat een melding zien dat het inloggen is fout gegaan met de bijbehorende fout.

Naam	Uitloggen
ID	UC-9
Omschrijving	De gebruiker kan uitloggen
Requirements	KOEK-33
Actor	Gebruiker
Pre-condities	De user is ingelogd.
Flow	1. De user drukt op uitloggen.
Post-conditie	De gebruiker is uitgelogd

Naam	SSH sessie
ID	UC-10
Omschrijving	De gebruiker kan een SSH sessie openen naar een server
Requirements	KOEK-34
Actor	Gebruiker
Pre-condities	De user is ingelogd. De user is op de server overzicht pagina Server URL is bekend
Flow	1. De user drukt op de hostnaam van een server
Post-conditie	Het OS opent een terminal met een SSH sessie.

Naam	Bekende objecten bekijken
ID	UC-11
Omschrijving	De gebruiker kan bekende iptables en user bekijken
Requirements	KOEK-35 / KOEK-36
Actor	Gebruiker
Pre-condities	De user is ingelogd. De user is op de server informatie
Flow	1. De gebruiker selecteert een tab. User of Iptables 2. De gebruiker selecteert ‘’Toon bekende” 3. Het systeem laad de gegevens en toont deze.
Post-conditie	-

Naam	Server actief/non-actief zetten
ID	UC-12
Omschrijving	De gebruiker kan servers op actief en non-actief zetten.
Requirements	KOEK-37
Actor	Gebruiker
Pre-condities	De user is ingelogd.
Flow	1. De gebruiker selecteert “Beheer” en vervolgens “Servers” in de header. 2. Het systeem laat een lijst met servers zien. 3. De gebruikers drukt een server aan om activere/deactiveren. 4. Het systeem toont een notitie popup 5. De gebruiker vult de gewenste gegevens in. 6. De gebruiker drukt op opslaan. 7. Het systeem geeft een melding dat de server is opgeslagen
Post-conditie	Server is geupdate in de database.
Alternative-flow	1. De gebruiker drukt de popup weg. 2. Het systeem zet de server weer terug naar zijn oude waarde.

Naam	Server informatie tonen
ID	UC-13
Omschrijving	De gebruiker kan alle informatie van een server bekijken
Requirements	KOEK-17 / KOEK-38
Actor	Gebruiker
Pre-condities	De user is ingelogd. De user is op de server overzicht pagina
Flow	1. De gebruiker dubbelklikt op een server rij.
Post-conditie	-

Naam	Versie koppelen aan een groep
ID	UC-14
Omschrijving	De gebruiker moet een versie kunnen koppelen aan een versie groep.
Requirements	KOEK-41
Actor	Gebruiker
Pre-condities	De user is ingelogd.
Flow	1. De gebruiker selecteert “Beheer” en vervolgens “Versies” in de header. 2. De gebruiker selecteert een versie groep bij een versie 3. De gebruiker drukt op opslaan. 4. Het systeem laat een melding zien dat de versies heeft bijgewerkt.
Post-conditie	Alle versies zijn geupdate met de nieuwe groep in de database.

BIJLAGE 5, DATABASE MODELLEN

Om de leesbaarheid van deze diagrammen te garanderen, zijn deze te vinden op de bijgevoegde USB-stick.

BIJLAGE 6, KLASSENDIAGRAMMEN

Om de leesbaarheid van deze diagrammen te garanderen, zijn deze te vinden op de bijgevoegde USB-stick.

BIJLAGE 7, TESTRAPPORT

Koekoek

VPS MONITORINGS TOOL

BASTIAAN KLEIN

In dit rapport zal het testproces van Koekoek VPS monitoringstool besproken worden. Hierbij word ingegaan op de verwachte resultaten van het testproces, daarna wordt besproken hoe en met welke technieken de testen zijn uitgevoerd. Als laatste wordt besproken hoe het testen verliep en wat de resultaten hier van waren.

Verwachte resultaten

Vanuit het bedrijf 42 wordt een minimale test coverage verwacht van 80%. Maar het gewenste niveau van iedere applicatie is 95% of hoger. Om te kijken of dit percentage gehaald is zal gebruikt worden gemaakt van de kwaliteitscontrole tool Sonar.

Te gebruiken technieken

De bovengenoemde test coverage moet behaald worden door het schrijven van unit testen. Hierbij zal gebruikt gemaakt worden van het test framework JUnit. Ook zal het jMockit framework gebruikt worden. JMockit staat toe om de testen te isoleren. Er is namelijk de wens dat elke test onafhankelijk is van andere klassen. JUnit is het standaard test framework voor het testen van Java code en wordt door 42 standaard toegepast op alle projecten. Voor mocken wordt gebruik gemaakt van Mockito of JMockit. Ik heb de keuze gemaakt voor JMockit omdat dit opgelegd werd door de opdrachtgever, omdat JMockit meer functies heeft dan Mockito.

De test coverage zal zoals eerder gezegd berekend worden via Sonar. Doordat de test coverage pas berekend word zodra er een wijziging is gemaakt in de remote repository en het builden van het project, heb ik nog een tool gebruikt die ook lokaal de test coverage kan berekenen. Dit om de development sneller te laten verlopen. Hiervoor is mijn keuze gevallen op Clover. Dit is een plugin voor de IDE Eclipse en werd aangeraden door een collega binnen het bedrijf 42. De uiteindelijke controle zal natuurlijk alsnog via Sonar verlopen.

Er is bij dit project geen sprake van Test Driven Development. De opdrachtgever vindt het niet belangrijk dat de code gebaseerd is op een eerder geschreven test. Hij vindt het alleen belangrijk dat er uiteindelijk wel een test gemaakt wordt.

Uitvoering

De uitvoering is voor het grootste deel verlopen zoals ik had verwacht. Er zijn gedurende de ontwikkeling testen geschreven om de klassen te testen. Hierbij is gebruik gemaakt van JMockit om testen te isoleren en ze niet afhankelijk te maken van andere klassen/testen. Denk bijvoorbeeld aan het mocken van een webclient om http calls te testen.

Resultaten

Het gewenste resultaat van 80% is uiteindelijk niet gehaald. Aan het einde van het project had ik niet genoeg tijd over om deze 80% te realiseren. In de onderstaande afbeelding kunt u het uiteindelijk resultaat zien. Helaas kan Sonar geen overzicht geven van de test coverage per package en/of klassen.

BIJLAGE 8, DEPLOYMENT PLAN

Koekoek

VPS MONITORINGS TOOL

BASTIAAN KLEIN

In dit document zult u alle stappen vinden die nodig zijn om de applicatie Koekoek met succes te deployen op de Koekoek server.

BUILDEN

Ten eerste zal de applicatie gebuild moeten worden. Dit kan met de volgende twee Maven commando's.

- **mvn release:prepare -Dusername= -Dpassword=**
- **mvn release:perform -Darguments="-Pdist"**

Met het eerste commando zorg je ervoor dat er jou applicatie gebuild en er een .war van gemaakt word. Met het tweede commando zorg je ervoor dat deze war geupload word naar de 42 repository.

Volgens download je de nieuwste build van de 42 repository.

DEPLOYEN

UPLOADEN VAN DE NIEUWE VERSIE NAAR JE EIGEN HOME DIRECTORY OP DE SERVER.

Ten eerste ga je naar de directory waar de waar gedownload is. Vanaf hier voer je het volgende commando uit:

```
scp koekoek-<release>.zip <user>@koekoek-test.42.nl:/home/<user>
```

UITPAKKEN VAN DE NIEUWE VERSIE.

Vervolgens pak je het bestand uit in een speciale map waarin alle oude releases bewaard worden. Dit zodat je altijd gelijk terug kan gaan mocht dit nodig zijn.

```
# log in op de server:
ssh <user>@koekoek-test.42.nl
# maak een directory aan:
cd /opt/koekoek/oude_releases
mkdir  koekoek-<release-nr>
# plaats in die nieuwe directory je zip-bestand uit je /home/<user> directory en unzip het daar
cd /opt/koekoek/oude_releases/koekoek-<release-nr>
cp /home/<user>/koekoek-<release>.zip .
unzip koekoek-<release>.zip
```

STOPPEN VAN DE TOMCAT SERVER

Hierna is het belangrijk dat je de tomcat server stopt. Omdat je anders je veranderingen niet gezien zullen worden door tomcat.

```
# verwijder ROOT.war
cd /opt/koekoek/base/webapps
ls -al
rm ROOT.war
# wachten totdat de map ROOT verwijderd is uit /opt/koekoek/base/webapps (ca 5 sec.)
ls -al

# als de map ROOT weg is kan tomcat gestopt worden
# controleer eerst dat tomcat nu draait
ps aux | grep koekoek
# je ziet nu een aantal entries, met als het goed is één met tomcat

# tomcat stoppen
sudo -u root /usr/bin/koekoek stop
# controleer dat tomcat nu gestopt is
ps aux | grep koekoek
```

WIJZIGEN VAN DE DATABASE

Indien er wijzigingen zijn gemaakt aan de database moeten deze veranderingen natuurlijk ook doorgevoerd worden. Dit gebeurd door de databasemigrator van Liquibase te draaien.

```
# kopieer je nieuwe changlogfiles uit /opt/koekoek/oude_releases/<nieuwe release>/db/changelogs
naar /opt/koekoek/db/changelogs
cd /opt/koekoek/db/changelogs
ls -al
cp /opt/koekoek/oude_releases/<nieuwe release>/db/changelogs/* .
ls -al
# voer de migratie uit
cd /opt/koekoek/db
sudo -u root ./migrate.sh
# controleer of de migratie goed is gegaan in de logfile
```

PLAATSEN VAN DE NIEUWE WAR FILE

Nu is het tijd om je nieuwe build van je applicatie in de apache folder te zetten. Tevens moet je rechten geven aan jou user om het bestand te maken uitpakken. Dit gebeurd zodra apache gestart word.

```
# kopieer de war file als ROOT.war en geef deze de juiste rechten
cd /opt/koekoek/base/webapps
cp /opt/koekoek/oude_releases/koekoek-<release-nr>/koekoek-<release-nr>.war ROOT.war
chown :developers ROOT.war
chmod 664 ROOT.war
```

STARTEN VAN JE APPLICATIE

```
# controleer eerst of de applicatie nog steeds gestopt is
ps aux | grep koekoek
# indien de applicatie toch draait eerst stoppen.
# als de applicatie niet draait kan hij gestart worden
sudo -u root /usr/bin/koekoek start
# controleer eerst of de applicatie gestart is
ps aux | grep koekoek
```

BEKIJKEN VAN DE LOGS

Om te kijken of alles goed is gegaan is het handig om even de log files te bekijken.

```
# logfiles staan in /opt/koekoek/base/logs
cd /opt/koekoek/base/logs
# interessante files zijn catalina.out, localhost.<datum>.log en localhost_access_log.<datum>.txt en
koekoek/logger.log

# bekijken kan met:
less <bestandsnaam>
pagina voorwaarts: f
pagina terug: b
naar einde: G
naar begin: g
less stoppen: q

# bestand volgen terwijl het loopt:
tail -f <bestandsnaam>
stoppen met ctrl-c
```

BIJLAGE 9, PRODUCT BACKLOG

Omdat deze bijlage te groot is voor dit document, kunt u deze vinden op de bijgeleverde USB stick.

BIJLAGE 10, BEDRIJFSEVALUATIE

DE HAAGSE HOGESCHOOL

Academie voor ICT & Media

Den Haag

Evaluatieformulier afstuderen

In te vullen door opdrachtgever c.q. bedrijfsmentor(en)

Student: Bastiaan Klein

Periode: Juni t/m Oktober 2014

Bedrijf c.q. instelling: Haagse Hogeschool

Bedrijfsmentor: Robert Bor

Plaats: Zoetermeer

Datum: 3 oktober 2014

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

Bastiaan heeft zich snel de domeinkennis verworven, waardoor hij prima in staat is gebleken om als klankbord te dienen voor de opdrachtgever.

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)?**

Bastiaan is een rustige persoonlijkheid die zich niet gek laat maken. Die rust brengt hij ook over op mensen die met hem samenwerken. Als hij zijn casus maakt, dan doet hij dat weloverdacht en met goed gekozen woorden

3. **Hoe heeft de student tijdens het uitvoeren van de opdracht gefunctioneerd?**

- **Qua verantwoordelijkheid** goed
- **Qua zelfstandigheid** goed
- **Qua planmatig werken** voldoende
- **Qua creativiteit** voldoende
- **Qua productiviteit** goed
- **Qua samenwerken met collega's** goed
- **Qua draagvlakontwikkeling** voldoende
- **Qua inspelen op bedrijfscultuur** goed
- **Qua rekening houden met de specifieke context van het bedrijf** goed
- **Qua het op gang brengen van de nodige veranderingen** voldoende

4. **Hoe beoordeelt u de kennis en kunde van de student in verhouding tot wat u verwacht van een bijna afgestudeerde?**

Bij binnenkomst wist Bastiaan vrijwel niets van onze technologie-stack. Deze barriere wist hij te overwinnen door hard te werken aan kennisverwerving. Aan het einde van de klus is hij zelfs in staat gebleken om andere mensen te helpen met problemen op het gebied van de stack.

5. Hoe beoordeelt u de kwaliteit van de opgeleverde (tussen)producten?

Koekoek is sterk verbeterd ten opzichte van het vorige product, edoch er is nog steeds een flinke slag te maken.

6. Bent u tevreden over het opgeleverde (eind)product?

- **In hoeverre heeft u gekregen wat is afgesproken?**

Gezien de wijzigende wensen (van onze kant), is zeker opgeleverd wat is afgesproken. De sprintafspraken werden over het algemeen prima gehaald.

- **In hoeverre voldoet het (eind)product aan uw verwachtingen?**

Het product is verbeterd en daar ben ik tevreden over.

- **Wat is de bruikbaarheid en onderhoudbaarheid hiervan?**

De code coverage mag nog wel wat hoger en er zijn veel constructies van het voorgaande team (stagiaires HHS) die vervangen dienen te worden. De toevoegingen van Bastiaan zijn kwalitatief van een hoger niveau.

- **Wat gebeurt er met het opgeleverde (eind)product?**

Dat dient nog een slag te ondergaan. Het product dient tevens als test-object voor de module INF-E Beheer van de Haagse Hogeschool.

- **Kunt u direct met het opgeleverde product aan de slag?**

Jazeker, dat doen we ook.

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

Bastiaan heeft niet alleen goed werk verricht met de opdracht, maar ook met de ondersteuning van de module INF-E Beheer, waar hij een sleutelrol heeft gespeeld in het faciliteren van de lessen en opdrachten.

8. Bent u bereid een volgende keer weer uw medewerking te verlenen aan het beschikbaar stellen van een afstudeerplaats (graag met toelichting)**?**

Ja.