# Ontwikkelen van datawarehousing & dashboard applicatie bij 42

Afstudeerverslag


Jordi Romeijn
11008865
Informatica, voltijd
Zoetermeer
07-06-2016

# Referaat

## Auteur

Jordi Romeijn

## Titel

Ontwikkelen van datawarehousing & dashboard applicatie bij 42 – Afstudeerverslag

## Afstudeerperiode

8 februari 2016 t/m 3 juni 2016

## Afstudeeromgeving

42 te Zoetermeer

## Instituut

De Haagse Hogeschool te Zoetermeer

## Opleiding

Informatica, voltijd, Zoetermeer

## Contactgegevens De Haagse Hogeschool

*Bezoekadres*
Bleiswijkseweg 37, 2712PB Zoetermeer

*Telefoonnummer*
+37 (0)70 445 7200

*Website*
http://www.dehaagsehogeschool.nl

## Contactgegevens 42

*Bezoekadres*
Koraalrood 33, 2718SB Zoetermeer

*Telefoonnummer*
+31 (0)88 4242 042

*Website*
http://www.42.nl

# Voorwoord

In dit document beschrijf ik mijn werkzaamheden en de keuzes die gemaakt zijn tijdens mijn afstuderen bij 42.

Ik heb hier 17 weken met veel plezier gewerkt en wil hen daar ook voor bedanken. Vooral de directe collegae die ik had zorgden ervoor dat ik een plezierige tijd had. Ook wil ik hen bedanken voor de hulp die zij mij gegeven hebben, zonder welke ik mijn opdracht niet succesvol had kunnen volbrengen. Specifiek zou ik mijn begeleider Rob en collega's Dave, Erik & Maarten willen noemen wie mij veel geleerd hebben over respectievelijk Java, Spring-Security, Sprint-Boot & JavaScript in de knowledge-sessions.

Ook wil ik de leraren op de HHS te Zoetermeer bedanken, zonder wiens lessen de afgelopen jaren ik niet de kennis zou hebben gehad om een opdracht zoals dit afstuderen te voltooien.

*Illustratie 1: Leviathan, rubber zeemonster.*

Ik wil mijn rubber eend bedanken voor het helpen bij het debuggen (Rubber Duck Debugging). Hij heeft goed naar me geluisterd. Hij is zo effectief geweest dat hij zelfs een eigen twitter-account heeft opgericht: @RubberLeviathan.

Ook de mensen die geholpen hebben met dit verslag te verbeteren ben ik dankbaar; de verschillende controles hebben hier een beter verslag van gemaakt.

Voor het begrijpen van dit document is een basale kennis van UML, Java & JavaScript vereist.

Zoetermeer
07-06-2016
Jordi Romeijn

# Inhoud

# Deel 1

## Achtergrondinformatie

In het eerste deel van dit afstudeerverslag vertel ik het doel en de structuur van dit document. Dit deel beschrijft wat de afstudeeropdracht inhield en waar ik de opdracht heb mogen uitvoeren. Daarnaast beschrijft het de methoden en technieken die gebruikt zijn gedurende het afstudeerproject.

# I. Inleiding

Dit document is het verslag van mijn afstuderen bij 42. Het doel van dit document is om een indruk te geven over de werkzaamheden die ik gedaan heb en de keuzes die tijdens het project gemaakt zijn te verantwoorden. Dit document is bedoeld voor mijn examinatoren om mijn afstuderen te kunnen beoordelen.

Dit document is opgedeeld in drie delen: Achtergrondinformatie, Projectuitvoer & Evaluatie en Referenties. In het deel Achtergrondinformatie wordt er verteld over het bedrijf, de opdracht, gebruikte methoden & technieken en wordt de 0-sprint besproken. Daarna wordt in Projectuitvoer de werkzaamheden en gemaakte keuzes tijdens het project besproken. Dit deel is opgesplitst in 8 hoofdstukken; per sprint wordt er apart naar de werkzaamheden en keuzes gekeken. Per sprint volgt mijn evaluatie over de huidige versie van (tussen)producten en processen. Het deel Evaluatie en Referenties gaat over mijn evaluatie over het gehele project en de uiteindelijk opgeleverde producten. In dit deel staat  beschreven hoe ik de beroepstaken wel of niet heb behaald. Ook staan hier de referenties van dit document opgesomd.

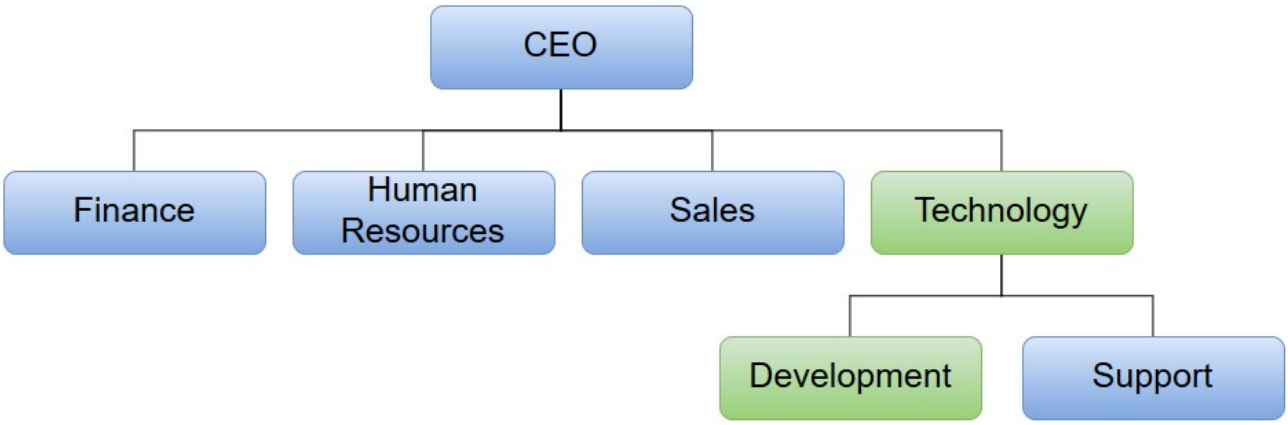
# II. Het Bedrijf

42 is opgericht in 2006 als een JAVA gespecialiseerd software bedrijf. Het bedrijf is gevestigd in Zoetermeer en telt ~42 man. Een belangrijk onderdeel van 42 is dat iedereen binnen het bedrijf een technische achtergrond heeft.

42 richt zich op het ontwikkelen van hoogwaardige oplossingen en diensten waarmee haar klanten hun werkprocessen en bedrijfsresultaten kunnen verbeteren. 42 ontwikkelt maatwerk Java-Enterprise producten, en doet dit bij voorkeur open source. De kernactiviteiten van 42 zijn: Software Development, Consultancy, Support, Audits, Analyse en Ontwerp.

Voorbeelden van bedrijven waar 42 op het moment al mee samenwerkt zijn de ANWB, de Belastingdienst en de Vrije Universiteit Amsterdam.

42 heeft een redelijk platte organisatiestructuur. Dit komt voort uit de filosofie van de directeur; hij heeft liever geen managementlaag. De mensen met een ‘managende’ rol hebben dit voornamelijk als bijrol en houden zich daarnaast ook bezig met ontwikkelen of support.




*Illustratie 2: Organogram. In het groen staan de afdelingen waar ik onder kom te vallen.*

Zoals te zien is in het organogram hierboven zijn er vier afdelingen: Finance, Human Resources, Sales & Technology. Onder Technology vallen vervolgens de afdelingen Development & Support. Ik kom te werken binnen Development; ik val daarmee ook onder Technology (groen aangegeven in het organogram).

42 heeft verschillende standaarden waar ik mij binnen dit project aan houd:

- Er wordt gewerkt met Scrum;
- Het Spring framework wordt gebruikt bij het programmeren in Java;
- Java code wordt getest met behulp van JUnit & JMockit;
- Bij het programmeren in JavaScript wordt er gebruik gemaakt van AngularJS;
- JavaScript code wordt getest met behulp van Jasmine;
- Er wordt gestreefd naar een 100% test coverage met unittests;
- Naast unittests worden er geen testcases gemaakt;
  - Er wordt ook geen testdocumentatie gemaakt;
- Er wordt gewerkt met Git;
- De desktop waarop ik kom te werken werkt op Linux.

Op de development afdeling wordt er gewerkt met scrum. Ik houd een agile methodiek aan met verschillende onderdelen van Scrum:

- Het project is een iteratief proces met sprints van twee weken (dit is ook wat 42 aanhoudt);
- Na afloop van elke sprint wordt er een sprint retrospective geschreven (door mijzelf);
- De product owner tijdens het project is de opdrachtgever;
- De stakeholders van dit project zijn mijn opdrachtgever en mijn begeleider; deze is tevens de architect. Later in het project (Sprint 3; §3.7) blijkt Operations/systeembeheer ook een stakeholder te zijn;
- Aan het eind van elke sprint wordt er een demo gegeven & een code review gehouden.

# III. De Opdracht

Het doel van de opdracht is om een nieuw software systeem te ontwikkelen. Het doel van dit systeem is om een centrale plaats te ontwikkelen voor het bekijken van bedrijfsgegevens. Deze bestaat op dit moment nog niet. Hierdoor moet er op verschillende (software) systemen binnen het bedrijf ingelogd worden om data van dat systeem te kunnen bekijken. Dit gaat ten koste van het overzicht en zorgt ervoor dat bepaalde data moeilijk aan elkaar gekoppeld kan worden.
Een aantal voorbeelden van bedrijfsgegevens zijn:

- Omzet: per klant, totaal, marges, debiteuren, crediteuren;
- Medewerkers per periode: aangenomen, uit dienst getreden, stagiairs, afstudeerders;
- Ziekteverzuim;
- Projecten: status, hoeveelheid;
- Sales: offertes gedaan, offertes geconverteerd;
- Inzetpercentages: per medewerker, totaal;
- SLA verplichtingen: behaald, niet behaald.

Een ander probleem is dat afspraken alleen per mail vastgelegd worden of onthouden worden door de betrokkenen. Er is geen centrale plaats en toetsing is soms te subjectief. Dit gaat ten koste van de efficiëntie en maakt het sturen van een groot bedrijf moeilijker. Indien de tijd het toelaat kan er nog nagedacht worden aan het implementeren van functionaliteit die deze toetsing aan het te maken systeem toevoegt.

Het doel van de opdracht is dat er twee nieuwe systemen komen. Systeem “Raaf” moet data van de andere systemen ophalen en zelf opslaan. Vervolgens moet een nieuw systeem, “Havik”, ontwikkeld worden die deze data laat zien samen met nieuwe formules die hierop toegepast kunnen worden. Een voorbeeld van zo'n nieuwe formule is dat het zichtbaar moet worden of het ziekteverzuim het afgelopen jaar is toegenomen of afgenomen (en met hoeveel procent). Kort gezegd moet Havik dus de front-end/GUI worden, en Raaf de back-end.

Na dit project is het mogelijk voor het management van 42 om een beter overzicht te verkrijgen van verschillende statistieken over het bedrijf. Afspraken kunnen beter getoetst worden en andere statistieken met betrekking tot personeel en omzet zullen beter en sneller beschikbaar zijn. Dit zorgt ervoor dat het management sneller kan zien hoe het bedrijf loopt en wat er niet goed gaat binnen het bedrijf. Anders gezegd, management kan efficiënter handelen op kernissues.

# IV. Methoden en Technieken

Gedurende het project werk ik op één van de computers gestationeerd bij 42. Hierbij heb ik toegang tot de verschillende systemen (en de source-code hiervan) die relevant zijn voor mijn project.

Een vergelijkbaar project zoals deze is in het verleden al eens uitgevoerd bij 42. Deze is destijds niet succesvol afgerond. Het project werd toen steeds groter, omdat er steeds meer functies bij de requirements kwamen. Door de sprints van twee weken forceer ik dat dit gedurende mijn project niet kan gebeuren; elke twee weken ligt er een werkend en getest product met een klein aantal nieuwe features ten opzichte van de voorgaande sprint.

De eerste week van het project zal de 0-sprint zijn; hierin probeer ik zoveel mogelijk de context van de opdracht duidelijk te maken en de eerste requirements op papier te krijgen (opnieuw om context te creëren).

De te maken systemen zullen in Java & JavaScript gemaakt worden. Om dit te programmeren wordt IntelliJ IDEA gebruikt als IDE. Kennis hiervan is binnenshuis bij 42 te vinden. De andere opties zouden NetBeans of Eclipse zijn; maar mijn begeleider heeft al snel laten merken dat deze onderdoen aan IntelliJ.
De makers van IntelliJ, JetBrains, hebben ook een (grote) plugin gemaakt voor Microsoft's Visual Studio genaamd Resharper. Met deze plugin heb ik in het verleden gewerkt; dit voegde veel functionaliteiten toe aan Visual Studio. Ik verwacht dat een complete IDE gemaakt door dezelfde makers veel overeenkomsten (qua functionaliteit) zal hebben met de plugin, waardoor ik al bekend zou zijn met meerdere van IntelliJ's functionaliteiten.

Bij 42 wordt er vrijwel niet met UML diagrammen gewerkt; wanneer dit wel gedaan wordt, wordt deze (meestal) reverse engineered door de IDE zelf. Omdat ik tijdens mijn project mijn UML diagrammen van te voren wil maken moest ik zelf op zoek naar een tool hiervoor. De computer die ik beschikbaar van 42 heb gekregen draait op Linux. Daardoor kan ik niet werken met de tools die ik in het verleden heb gebruikt, omdat deze alleen beschikbaar zijn voor Windows.

Ik heb besloten om met de online applicatie draw.io te werken. Deze is gratis, online (dus vereist geen installatie, maar is te openen in elke webbrowser) en bevat alle benodigde functies. Het opslaan gebeurt via een .xml bestand of direct als image (.png). Doordat draw.io online is zijn de diagrammen gemakkelijk te delen met anderen (bijvoorbeeld mijn begeleider). Deze hoeft om de diagrammen te kunnen bekijken en/of aan te passen geen apart UML programma te installeren.

In mijn afstudeerplan zijn vervolgens nog de volgende technieken benoemd: (de technieken die hierboven worden besproken zijn uit deze lijst gehaald)

- AngularJS;
- CSS;
- Grunt;
- Hibernate;
- HTML;
- Jmockit;
- JPA;
- Junit;
- LESS;
- Maven;
- Power;
- REST;
- SOAP;
- Spring;
- Spring-Boot.

Deze komen in het project zelf terug; indien er bijzonderheden zijn voorgekomen wordt dit bij het bijbehorende hoofdstuk verteld.

# V. Sprint 0: Opstarten

| Opstarten | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 | Sprint 7 | Sprint 8 |
|---|---|---|---|---|---|---|---|---|

In de 0-sprint zijn de eerste requirements opgesteld (§V.1). Daarnaast zijn er verschillende sessies geweest om Java/Spring/Spring-Boot kennis op te doen (§V.2) en heb ik de huidige systemen bekeken (§V.3). Als laatste volgt mijn evaluatie over de 0-sprint (§V.4).

## V.1. Requirements

Ter verduidelijking van het project zijn in deze sprint de eerste requirements opgesteld. Deze zijn uit de afstudeeropdracht behaald en uit een interview met de opdrachtgever. Tijdens het interview met de opdrachtgever is mij ook veel verteld over de huidige situatie. Dit komt terug in V.3. In de tabel hieronder valt een subset van de requirements te zien.

| *#* | *Requirement* | *Oorsprong* |
|---|---|---|
| 1.1. | Havik moet de data uit Raaf kunnen laten zien. | Afstudeeropdracht |
| 1.2. | Havik moet berekeningen kunnen toepassen op de data uit Raaf. | Afstudeeropdracht |
| 1.3.1. | Havik moet kunnen laten zien hoe het ziekteverzuim in het verleden is geweest. | Afstudeeropdracht |
| 1.3.2. | Havik moet het ziekteverzuim van verschillende jaren kunnen laten zien. | Afstudeeropdracht |
| 1.3.3. | Havik moet het ziekteverzuim van verschillende maanden kunnen laten zien. | Afstudeeropdracht |
| 1.4. | Havik moet gegevens uit het verleden kunnen opvragen. | Interview #1 Opdrachtgever |
| 1.5. | Sommige gegevens moeten handmatig ingevoerd kunnen worden. | Interview #1 Opdrachtgever |
| 1.6.1. | Havik moet kunnen laten zien hoeveel werknemers er in een jaar waren. | Interview #1 Opdrachtgever |
| 1.6.2. | Havik moet kunnen laten zien hoeveel werknemers er in een maand waren. | Interview #1 Opdrachtgever |
| 1.6.3. | Havik moet in de gevallen van 1.6.1. & 1.6.2. kunnen laten zien welke werknemers dit waren. | Interview #1 Opdrachtgever |
| 1.6.4. | Havik moet de hoeveelheid werknemers kunnen afzetten tegen de omzet in één grafiek. | Interview #1 Opdrachtgever |

*Tabel 1: Subset van de huidige requirements.*

> “De gegevens die bij 1.5. bedoeld worden zijn nog niet bekend; wanneer deze bekend zijn wordt deze requirement verder uitgebreid/verduidelijkt.”
>
> – Requirements V1.0.

Voorafgaand aan het interview met de opdrachtgever heb ik verschillende vragen opgesteld. Vervolgens heb ik het interview opgenomen zodat ik deze later opnieuw kan beluisteren. Dit heeft de volgende consequenties:

- Ik hoef geen notulen op te stellen tijdens het interview; ik kan mij hier volledig op focussen en mis geen informatie omdat ik bezig was met schrijven;
- Ik kan deze later in het project nog terugluisteren. Als er door nieuwe informatie mijn idee over een gezegd iets veranderd kan ik het originele gesprek terugluisteren; bij notulen is het opgeschreven met mijn originele idee in gedachten waardoor de notulen niet correct kunnen zijn; het opgenomen gesprek is echter altijd correct;
- Het beluisteren van een heel gesprek kost meer tijd als notulen doornemen.

Voor het opstellen van requirements luisterde ik achteraf het gesprek terug.

Er hebben twee interviews plaatsgevonden deze sprint; één met de opdrachtgever en één met de opdrachtgever & begeleider. De voorbereiding & opstellen van requirements gingen bij allebei hetzelfde (zoals hierboven uitgelegd). Het tweede gesprek (met de opdrachtgever & begeleider) zorgde ervoor dat beide betrokkenen met elkaar in discussie gingen; dit zorgde ook voor een hoop duidelijkheid & requirements.

Naast de requirements voor Havik zijn ook de eerste requirements voor Raaf, non-functional requirements en business rules opgesteld. Voor de volledige set zie bijlage A.

## V.2. Java, Spring & Spring-Boot

Tijdens de eerste weken van het project zijn er meerdere sessies geweest met mijn bedrijfsbegeleider over Java & Spring. Hierin heeft hij veel over deze onderwerpen uitgelegd en gedemonstreerd.

### V.2.1. Spring & Annotaties

Spring is een framework voor Java. Spring zorgt ervoor dat er veel dingen automatisch geregeld worden zonder dit zelf te hoeven programmeren; annotaties zijn hier een groot onderdeel van. Het onderstaande voorbeeld demonstreert de functie van de annotatie @RequestMapping.

```
@RequestMapping("/Demo")
public class AnnotationDemonstration {
    @RequestMapping(value = "/HelloWorld", method = RequestMethod.POST)
    public String helloWorld() { return "Hello World!"; }
}
```

*Illustratie 3: Demonstratie @RequestMapping.*

Zou deze code op een web server gezet worden zou de methode “helloWorld()” aan te spreken zijn vanaf een browser via de URL “localhost:8080**/demo/helloWorld**”. Naast de configuraties voor de web server is er geen andere configuratie nodig.

Een ander voorbeeld is het gebruik van @Bean & @Inject.

```
@Inject
private String helloWorld;


@Bean
private String string(){
    return "Hello World!";
}
```

*Illustratie 4: Demonstratie @Inject & @Bean.*

De @Bean annotatie maakt een zogenaamde Bean aan (in het voorbeeld hierboven een String met de waarde “Hello World!”). In andere klassen kan er nu een instantievariabele zijn van het type String met de annotatie @Inject. Deze variabelen worden vervolgens automatisch gevuld met de Bean van hetzelfde type. In het voorbeeld hierboven wordt de String helloWorld dus automatisch gevuld met de String Bean; ook wanneer deze in andere klassen zouden staan. In dit voorbeeld wordt een String als voorbeeld gebruikt; uiteraard werkt dit ook met zelfgemaakte klassen.

### V.2.2. Spring-Boot

Kennis van Spring-Boot heb ik opgedaan d.m.v. een sessie met een andere collega van 42 en zelf de beschikbare documentatie door te nemen.

Bij het opzetten van een nieuw project moet er (ook met het gebruik van Spring) nog veel geconfigureerd worden. Bijvoorbeeld database connecties en webservers. Spring-Boot neemt gedeeltes van deze configuratie over door deze automatisch toe te voegen. Spring-Boot gebruikt hier “opinionated views” voor. Bepaalde configuraties, zoals voor een database of webserver, worden dan automatisch met standaardwaarden geconfigureerd. Hierdoor hoef je als programmeur hier zelf geen tijd aan te spenderen. Zijn er echter opties die je toch wilt aanpassen kan dit ook. Het voordeel hiermee is dat de programmeur sneller kan beginnen met het programmeren, en in het begin niet opgehouden wordt door verschillende configuraties.

Een voorbeeld van de snelheid waarmee geprogrammeerd kan worden is een twitter bericht van Rob Winch, een project lead bij Spring.

Een tweet mag maximaal 140 karakters hebben. Door middel van alle auto-configuraties gegeven door Spring-Boot is er niet meer nodig dan 140 karakters om werkende code neer te zetten.

Zou deze code uitgevoerd worden is het mogelijk om naar de URL (“localhost:8080/”) te gaan, waar vervolgens “Hello World!” wordt laten zien.


*Illustratie 5: Spring-Boot Tweet.*

> Naast de configuraties voor de web server is er geen andere configuratie nodig.
>
> – §V.2.1.

In het hoofdstuk over Spring (§V.2.1.) stond een stuk code met de informatie dat er naast configuraties voor de web server geen andere configuraties nodig zijn. Bij het gebruik van Spring-Boot is het nog simpeler: deze configuraties zijn ook niet nodig. Simpelweg de referentie naar Spring-Boot toevoegen is genoeg; zonder Spring-Boot zouden zowel een referentie naar Spring als naar de gebruikte web server nodig zijn (en daarnaast nog de configuraties voor de web server).

# V.3. Huidige systemen

Tijdens de sessies van V.2. heeft mijn begeleider ook veel verteld over de opbouw van de systemen bij 42. In de tijd dat ik zelf naar de systemen heb gekeken zag ik dit ook terug en heb ik de connecties op zich beter kunnen bekijken. Deze bevindingen worden beschreven in de komende paragrafen.

## V.3.1. Result – Entity – Form

Objecten bestaan in drie versies in de code. Entity, Form & Result.

Een Entity in de code is zoals deze uit de database komt; alle velden zijn bereik- & aanpasbaar.

Een Result is zoals deze wordt doorgestuurd naar de gebruiker/web-browser. Een Result bevat vaak minder velden dan een Entity; belangrijke of geheime gegevens (bijvoorbeeld wachtwoorden) zijn hier uit gehaald. Hierdoor is dit veilig om naar de gebruiker terug te sturen zonder het gevaar te lopen dat de gebruiker bijvoorbeeld via de console van de browser deze gegevens kan lezen.

Het Form object is zoals deze voor de gebruiker toe te voegen of aan te passen is; bijvoorbeeld een formulier op een website. Dit kunnen andere gegevens zijn dan de gegevens bij Result. Door dit af te scheiden van Entity kunnen niet alle gegevens aangepast worden (data zoals een automatisch gegenereerd ID hoeft nooit aangepast te worden).

Forms & Results zijn kortlevend; ze dienen slechts als koppelstuk tussen de front- en backend.

De scheiding tussen Entities, Forms & Results zorgt ervoor dat data veiliger en minder foutgevoelig is.

## V.3.2. Repository – Service – Controller – AngularJS

Binnen de systemen wordt gewerkt met 3 “lagen”. De Repository, Service & Controller.

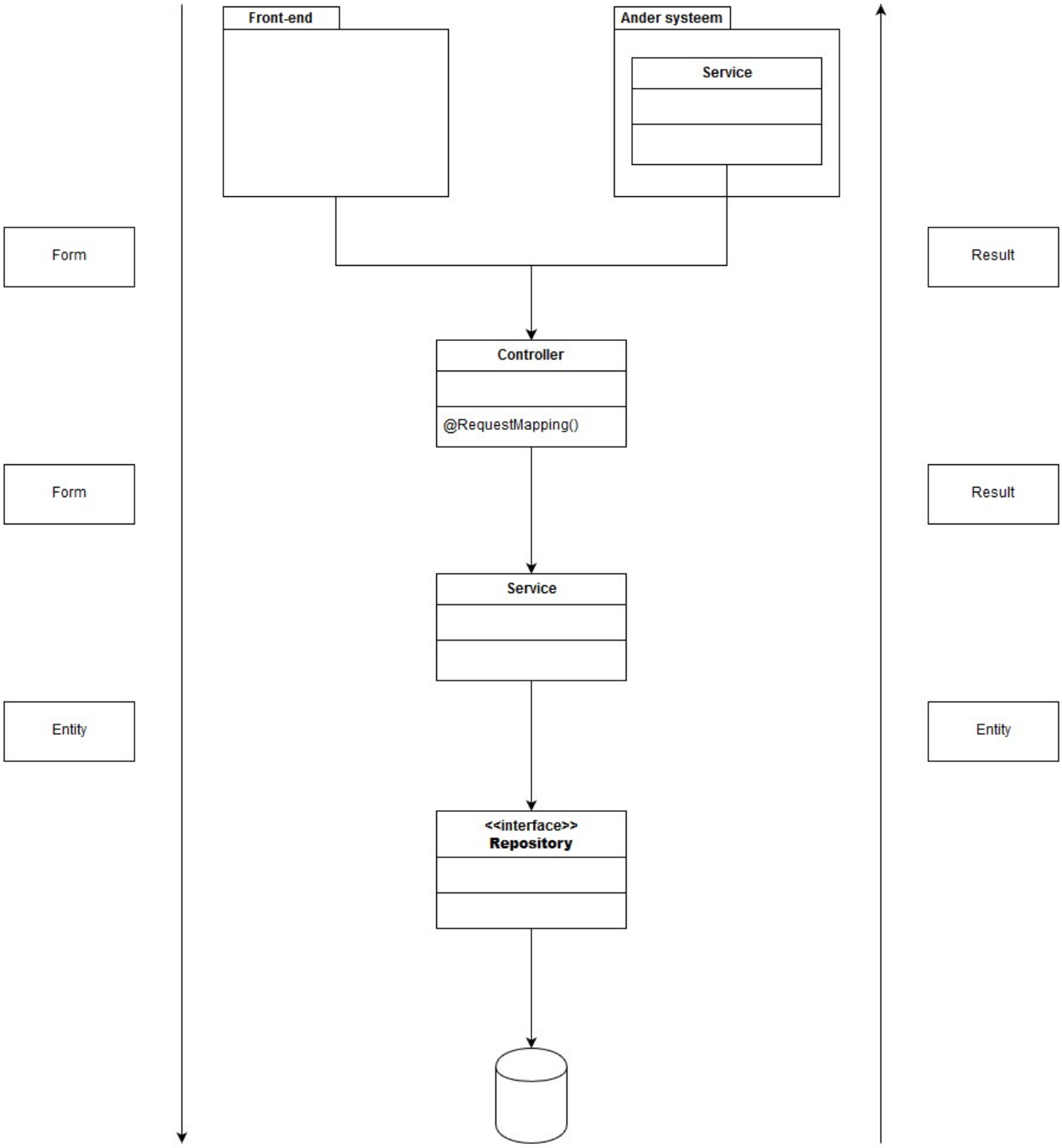
De Repository is de connectie tot de database. De objecten die hierbij meegestuurd worden, zowel naar een Repository toe als de uitgaande objecten, zijn Entities.

De Repository wordt (alleen) aangesproken door de Service klassen. In de Service laag gebeuren de conversies tussen Entities, Forms & Results. Naast de conversies bevatten de Service klassen de benodigde CRUD methodes.
Bevat een systeem geen directe connectie naar de database, maar wilt deze toch data opvragen dan zit de Service laag iets anders in elkaar (dit is het geval bij Havik). De Service klassen bevragen in dat geval de Controller van een ander systeem (Postduif) om de data op te halen uit de database.

De Controllers bevatten de methodes die door andere systemen/front-ends aangeroepen kunnen worden om data op te halen. Hierbij komen de requests direct uit de front-end of andere systemen. Voor de Controller zit hier geen verschil in. Binnen de Controller wordt er met zowel Forms als Results gewerkt. Er kan ook met primitives gewerkt worden in sommige gevallen. Een Read request voor een object met een int als identifier kan bijvoorbeeld deze int meesturen om het object op te halen. Hier wordt er dus een int naar de Controller gestuurd en een Result object teruggestuurd.

Voorbeeld van een Update request:

De front-end stuurt een Form object door naar de Controller. De Controller zorgt er vervolgens voor dat deze bij de juiste Service(-methode) terecht komt. De Service maakt hier vervolgens een Entity van en slaat deze op in de database via de Repository. Indien het object teruggestuurd moet worden maakt de Service vervolgens een Result van deze Entity. Via de reguliere return statements wordt deze teruggestuurd naar de front-end via de Controller.



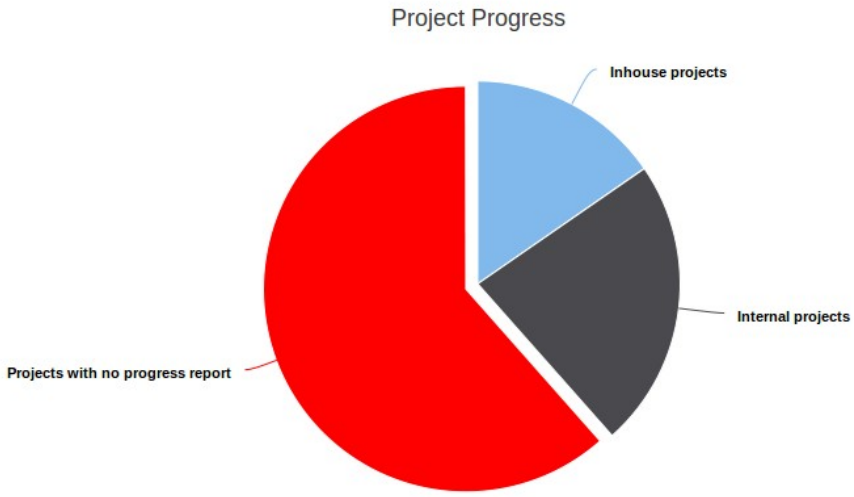

*Illustratie 6: De verschillende lagen en entiteits-vormen.*

## V.3.3. Postduif & Havik

De twee systemen waar ik het meest mee te maken ga krijgen zijn Postduif en Havik. Uiteraard heb ik de werking van deze twee systemen bekeken.

Postduif zelf bevat geen front-end;  Postduif is bedoeld om de connectie met de database te zijn voor de andere sytemen. De front-end die Postduif bevat is puur bedoeld voor development; de methodes exposed in de Controller kunnen hiermee getest worden. De opbouw van Postduif staat beschreven in §V.3.2.: een Repository laag voor de connectie met de database, een Service laag voor manipulatie en een Controller laag voor het ontsluiten van de REST API.

Havik is het systeem wat ik zou gaan maken tijdens dit project (samen met Raaf). Tijdens gesprekken met de opdrachtgever is mij verteld dat er al een gedeelte van Havik gemaakt is door een andere medewerker. In een meeting samen met deze medewerker heb ik de huidige functionaliteit van Havik kunnen zien. Na deze meeting heb ik zelf Havik op mijn PC geïnstalleerd en bekeken; er bestond meer functionaliteit dan zowel ik als mijn opdrachtgever & begeleider van tevoren dachten.




*Illustratie 7: Bestaande functionaliteit van Havik.*

De huidige versie van Havik bestaat uit twee systemen/Java-projecten: Havik-UI & Havik-Core. Havik-Core kan vergeleken worden met wat oorspronkelijk Raaf zou worden. De huidige functionaliteit van Havik bestaat al uit het laten zien van een aantal grafieken/tabellen. Deze gaan over de huidige situatie van de andere sytemen (bijvoorbeeld: hoeveel projecten lopen er op dit moment binnen 42). Havik-UI maakt gebruik van de openSource Library HighCharts; deze maakt het makkelijk om grafieken te maken. De code van de pie-chart op de vorige pagina staat hier naast. De code begint met enkele opties (waaronder het aangeven dat het een pie-chart is). Daaronder de data-series; deze wordt doorgegeven aan de function wanneer deze uitgevoerd wordt. Daaronder de URL's; hier wordt naartoe gelinked wanneer op de aparte data-series geklikt wordt. Hierdoor komt de gebruiker gelijk uit bij het systeem waar deze data van is met de queryparameters in de URL. Hierdoor kan de gebruiker meteen de detailgegevens zien over de aangeklikte data. De return statement als laatste geeft een gedeelte van de URL en de bovenstaande opbouw van de grafiek terug wanneer deze aangeroepen wordt.

```
var parse = function (data) {
    return {
    type: 'highchart',
    config: {
        options: {
            chart: {
                    plotBackgroundColor: null,
                    plotBorderWith: null,
                    plotShadow: false,
                    type: 'pie'
                }
            },
            series: [{
                name: 'Projects',
                data: [{
                    name: 'Inhouse projects',
                    y: data.inHouseProjects
                }, {
                    name: 'Internal projects',
                    y: data.internalProjects
                }, {
                    name: 'Projects with no progress report',
                    y: data.noProgress,
                    sliced: true,
                    selected: true,
                    color: 'red'
                }]
            }],
            title: {
                text: 'Project Progress'
            }
        },
        urls: [{
            label: 'Inhouse projects',
            url: '?inHouseOnly=true'
        }, {
            label: 'Internal projects',
            url: '?internalOnly=true'
        }, {
            label: 'Projects with no progress report',
            url: '?noProgressOnly=true'
        }]
    };
};
return {
    url: '/project-progress',
    parse: parse
};
```

*Illustratie 8: Voorbeeld code voor een Pie-Chart in HighCharts.*

Tijdens de volgende sprint zou er gesproken moeten worden over de functionaliteit die al bestaat in Havik met mijn opdrachtgever & begeleider. Er bestaat nu al een groot gedeelte van Havik (en wat Raaf zou worden), wat mijn project kan verkleinen. Een bijkomend punt is dat de systemen die ik zou gaan opzetten gebruik zouden maken van Spring-Boot, als test voor 42 of dit globaler ingezet zou moeten worden. Nu Havik al bestaat is het lastig (niet onmogelijk, maar dan moet er veel opnieuw gemaakt worden terwijl het al bestaat) om alsnog te gebruiken. Ook dit zou besproken moeten worden.

## V.4. Evaluatie

De huidige set van requirements is nog klein; mijn verwachting is dat uit de review door de opdrachtgever er juist veel bijkomen.

Java is een taal waar ik nog weinig ervaring mee heb. C# is een taal die ik echter beter beheers; al snel valt het op dat C# en Java ontzettend veel op elkaar lijken. In het Plan van Aanpak staat benoemd dat ik had verwacht dat het ontbreken van kennis op het gebied van Java een risico zal zijn; na deze ontdekking denk ik dat dit een veel kleiner risico is dan oorspronkelijk verwacht.

De huidige systemen zitten duidelijk in elkaar. Omdat elk systeem gebruik maakt van vrijwel dezelfde opbouw maakt dit sommige aspecten van het komende werk voorspelbaar. Het programmeren zal hierdoor simpeler zijn dan wanneer dit niet het geval was.

# Deel 2

## Opdrachtuitvoer

Het tweede deel van mijn verslag beschrijft op chronologische wijze hoe ik tot het eindresultaat ben gekomen. De eerder besproken componenten zullen hierbij soms ook genoemd worden. Hierbij vertel ik welke documenten opgesteld zijn, welke modellen ontworpen en welke systemen ontwikkeld. Hierbij wordt ook mijn gedachtegang en keuzebeslissingen uitgelegd.

# 1. Sprint 1: Requirements

| Opstarten | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 | Sprint 7 | Sprint 8 |
|---|---|---|---|---|---|---|---|---|

De eerste sprint draait om het requirements document (§1.1 & §1.4). Ondertussen heeft er het gesprek plaatsgevonden over de gevolgen dat Havik al bestaat (§1.2). Gedurende deze sprint heb ik ook mijn eerste demo gegeven (§1.3). Als voortzetting op de Java sessies besproken in §V.2. is er tijdens deze sprint een sessie geweest over JavaScript (§1.5). Als laatste volgt mijn evaluatie over deze sprint en zijn producten (§1.6).

In de eerste sprint is het de bedoeling om zo veel mogelijk requirements boven tafel te krijgen. Er wordt deze sprint dus geen software gemaakt. Dit is niet zoals het normaal loopt bij een Scrum-project. Voor dit project willen we (mijn begeleider & ik) allereerst een groot gedeelte van de backlog gemaakt hebben, zodat er zichtbaar is wat er uiteindelijk allemaal gemaakt kan worden. Vervolgens kunnen we hierna per sprint bekijken wat er gedaan gaat worden & kunnen er sprint-specifieke requirements opgesteld worden.

| *Activiteit* | *Datum* |
|---|---|
| Opstellen requirements | 15-18 & 22-25 februari |
| Opstellen definities | 18 & 24 februari |
| Opstellen use cases | 23 & 25 februari |
| Tijd die niet besteed kan worden aan requirements (bv. omdat een meeting nodig is voor verdere input) wordt besteed aan Spring, SpringBoot & Havik | - |
| Werken aan stageverslag | 19 & 26 februari |

*Tabel 2: Planning sprint 1 volgens het plan van aanpak.*

Requirements verzamelen een inefficiënt proces; het gebeurt regelmatig dat er niet door gewerkt kan worden, omdat er bijvoorbeeld op een gesprek met de opdrachtgever gewacht moet worden. Hierdoor blijft er regelmatig tijd over. Deze tijd is besteed aan het bestuderen van Spring, Spring-Boot & het al bestaande gedeelte van Havik. De bevindingen hiervan zijn onderdeel van de hoofdstukken §V.2 & §V.3.

## 1.1. Requirements V2.0

Om de requirements op te stellen zijn er meerdere gesprekken geweest met de opdrachtgever en de architect/begeleider (dezelfde persoon). De voorbereiding en het opstellen van de requirements zijn hetzelfde verlopen als in §V.1; ook deze interviews zijn opgenomen. Ondertussen heb ik de opmerkingen van de opdrachtgever ontvangen van de huidige versie (V1.0) van de requirements. Beide bronnen hebben voor een grote toevoeging aan requirements gezorgd; een subset van de functionele requirements valt hieronder te zien:

| # | | *Benodigd Requirement* | *Oorsprong* |
|---|---|---|---|
| 1.1. | | Havik moet grafieken kunnen laten zien. | Afstudeeropdracht |
| 1.2. | | Havik moet tabellen kunnen laten zien. | Afstudeeropdracht |
| 1.3.1. | | Het systeem moet kunnen filteren op werkmaatschappij. | Review #1 Opdrachtgever |
| 1.3.2.1. | 1.1. | In het geval van een staafdiagram moet het per werkmaatschappij aan- en uit te zetten zijn of deze zichtbaar is. | Interview #3 Opdrachtgever |
| 1.3.2.2. | 1.1. | In het geval van een lijndiagram moet het per werkmaatschappij en het totaal/gemiddelde aan- en uit te zetten zijn of deze zichtbaar is. | Interview #3 Opdrachtgever |
| 1.4.1.1. | 1.1. | Havik moet het ziekteverzuim over een periode van 10 jaar kunnen laten zien. | Review #1 Opdrachtgever |
| 1.4.1.2. | | Dit moet in een lijndiagram. | Interview #3 Opdrachtgever |
| 1.4.1.3. | | Hierbij wordt elk jaar gerepresenteerd door één lijn. | Interview #3 Opdrachtgever |
| 1.4.2.1. | 1.1. | Havik moet het ziekteverzuim per maand over een periode van een jaar kunnen laten zien. | Interview #3 Opdrachtgever |
| 1.4.2.2. | | De selectie moet dan voor/achteruit in de tijd kunnen scrollen. | Review #1 Opdrachtgever |
| 1.4.2.3 | | Dit moet in de vorm van een staafdiagram. | Review #1 Opdrachtgever |

*Tabel 3: Subset requirements V2.0.*

Om duidelijkheid te creëren bij zowel mijzelf als bij de opdrachtgever is er ook een lijst met definities opgesteld. Dit zijn termen die ikzelf niet kende (bv. FTE, grootboekrekening), of die onduidelijkheid zouden kunnen veroorzaken omdat ze openstaan voor interpretatie (bv. functie, widget) of op een andere manier onduidelijk zijn (bv. historische gegevens). Zie hier de volledige lijst hiervan:

| *Begrip* | *Afkorting* | *Definitie* |
|---|---|---|
| Full-Time Equivalent | FTE | Dienstverband van een medewerker ten opzichte van een volledige werkweek van 40 uur.<br>Een medewerker die 40 uur per week werkt staat gelijk aan 1 FTE; een medewerker die 32 uur per week werkt staat gelijk aan 0.8 FTE. |
| Functie | | Dat wat uiteindelijk culmineert in een widget. |
| Grootboekrekening | | Een herkenningsnummer voor boekhouding waar meerdere posten onder gegroepeerd kan worden. Iedere factuur is gekoppeld aan een grootboekrekening. |
| Statische/historische gegevens | | Gegevens die niet meer veranderen, meestal omdat er genoeg tijd over heen is gegaan dat deze niet meer hoeven, of zelfs mogen, aangepast te worden. |
| Verplichte verlofdagen | | Werkdagen die een medewerker verplicht vrij krijgt, bijvoorbeeld feestdagen. |
| Werkdagen | | De werkbare dagen in een week; kan per medewerker anders zijn. |
| Widget | | Een klein onderdeel van een webpagina die iets nuttigs met informatie opgehaald uit andere locaties doet. Geeft deze informatie vervolgens ook weer. |

*Tabel 4: Definities.*

Er was gepland dat er ook use cases gemaakt zouden worden. Omdat het te maken systeem echter een dashboard applicatie is zou dit weinig nuttigs toevoegen. De use cases die gemaakt zouden kunnen worden zou zijn van het inloggen (wat in totaal 1 stap is van de gebruiker; ook is dit al een bestaande functie met bestaande error handling). Andere use cases zouden zijn van het veranderen van bepaalde parameters bij bepaalde widgets (bijvoorbeeld een jaartal), maar dit is opnieuw één handeling; er is te weinig interactie met het systeem om hier nuttige use cases van te maken.

Daarnaast zijn de user stories wel bekend; dit maakt use cases nog minder interessant. (De opdrachtgever wilt een overzicht hebben van wat er binnen het bedrijf gebeurt zodat alle managers deze zelfde visie kunnen hanteren.)

## 1.2. Gevolgen Havik's bestaan

In §V.3.3 staat benoemd dat Havik al gedeeltelijk bestaat. Dit zou ervoor zorgen dat ik de systemen niet hoef op te starten, maar verder kan gaan op al bestaand werk. Dit is de oorzaak van meerdere gevolgen:

- Ik hoef het systeem niet vanaf begin af aan te maken;
- Ik ben gelimiteerd door de manier van coderen van mijn voorganger; een aanpassing kan oplopen tot een grote herschrijving van de code van mijn voorganger, wat ik wil voorkomen (minder tijd besteden aan het herschrijven van iets al bestaands is meer tijd die besteed kan worden aan het maken van mijn project);
- Omdat er nu geen tijd besteed hoeft te worden aan het opstarten van het systeem (de layout van de web-pagina is al bekend, algemene functionaliteiten die ik ook ga gebruiken zijn soms al gemaakt, etc.) kunnen er meer requirements behandeld worden/meer widgets & features gemaakt worden;
- De huidige opbouw van Havik zijn twee Java projecten: Havik-Core & Havik-UI. Havik-Core bevat al functionaliteit die in mijn project in Raaf zou vallen. Na conversatie met mijn begeleider hebben we besloten om Raaf te laten vallen en de functionaliteit die deze zou bevatten in Havik-Core te bouwen. Havik-UI wordt wat oorspronkelijk Havik zou zijn in dit project;
- Ik kan geen Spring-Boot gebruiken om het project op te starten.

Vooral dit laatste punt is van belang. 42 had voor ogen dat dit project ook voor hen een instap in Spring-Boot zou zijn. Zou Spring-Boot tijdens mijn project goed uitvallen zou 42 dit ook veel meer tijdens hun eigen projecten gebruiken. Nu ik dit niet hoef te gebruiken gaat dit niet meer op deze manier.

Er is nu daarom besloten dat ik een presentatie geef over Spring-Boot voor alle medewerkers van 42. Richting het einde van de derde sprint houdt 42 een Knowledge Transfer Meeting (KTM); hierin zal ik een presentatie geven over Spring-Boot en de voor-/nadelen hiervan.

## 1.3. Demo

Elke sprint wordt, zoals standaard bij Scrum, afgesloten met een demo. Voor deze demo had ik alleen een requirements document om te bespreken. Deze had ik echter vlak hiervoor opgestuurd naar mijn opdrachtgever & begeleider; dit resulteerde in een kort durende demo.

Er waren twee punten die hieruit kwamen als feedback. Als eerste moesten de requirements SMARTer. Ten tweede zou mijn begeleider het fijn vinden als de requirements ondersteund werden door nep-grafieken/illustraties als verduidelijking & visualisatie van de requirements.

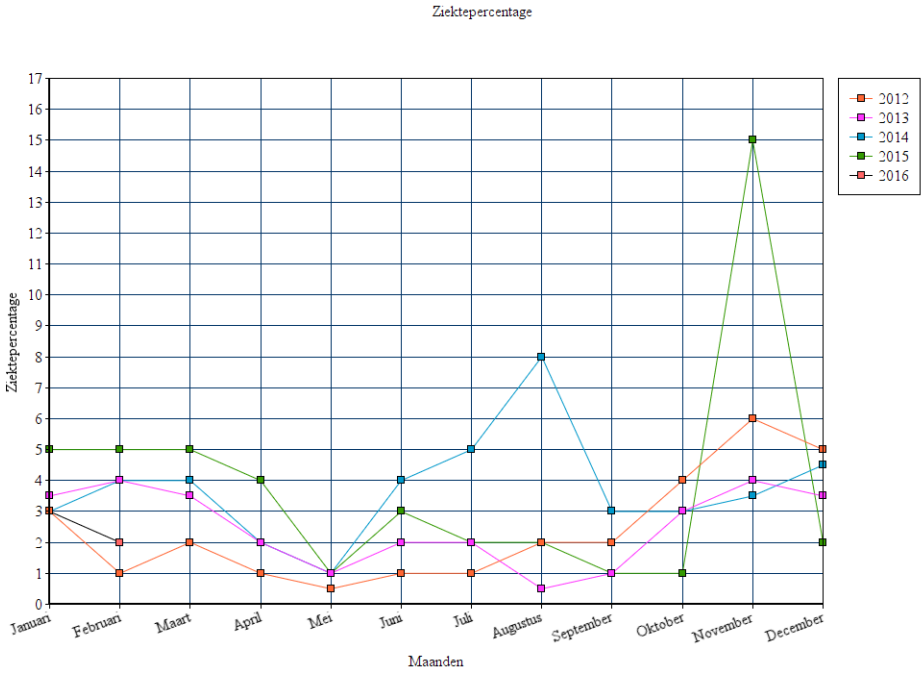
In de komende sprints zal er na de demo ook een code review gehouden worden. Hierin willen we het gemaakte product verbeteren d.m.v. de code op zich te verbeteren.

## 1.4. Requirements V3.0

Na de demo heb ik de feedback hieruit verwerkt; de requirements SMARTer maken & visualisatie toevoegen. Een voorbeeld hiervan is requirement 1.4.1.:

| # | | *Benodigd Requirement* | *Oorsprong* |
|---|---|---|---|
| 1.4.1.1. | 1.1. | Havik moet het ziekteverzuim over een periode van 10 jaar kunnen laten zien. | Review #1 Opdrachtgever |
| 1.4.1.2. | | Hierbij wordt het ziektepercentage tegenover maanden uitgezet. | Interview #3 Opdrachtgever |
| 1.4.1.3. | | Dit moet in een lijndiagram. | Interview #3 Opdrachtgever |
| 1.4.1.4. | | Hierbij wordt elk jaar gerepresenteerd door één lijn. | Interview #3 Opdrachtgever |

*Tabel 5: Requirement 1.4.1.2. is nieuw als resultaat om 1.4.1. SMARTer te maken.*




*Illustratie 9: Visualisatie requirement 1.4.1.*

De opdrachtgever gaat de requirements zelf nog volledig door lezen en hier feedback op geven; totdat deze goedkeuring is gekomen is het nog niet mogelijk om de requirements te prioriteren. Voorlopig heeft de opdrachtgever wel een voorkeur/aanraden aangegeven: om de volgende sprint de widget te maken m.b.t. de hoeveelheid werknemers of de widget m.b.t. ziekteverzuim. De uiteindelijke keuze hiertussen wordt later gemaakt en staat later beschreven (§2).

## 1.5. JavaScript

Tijdens dit project gaat er ook met JavaScript gewerkt worden. Omdat mijn kennis hiervan vrijwel niet bestaand is is er een sessie met één van de medewerkers van 42 (niet mijn begeleider, zoals bij de Java sessies) voortgekomen. Hierin zijn mij de belangrijkste onderdelen van JavaScript uitgelegd. JavaScript wordt in dit document niet uitgelegd; dit is een te groot onderwerp.

## 1.6. Evaluatie

De requirements zijn nu nog niet geprioriteerd. Voor de komende twee sprints heeft de opdrachtgever al aangegeven wat er kan gebeuren (de widgets voor medewerkersaantallen en ziekteverzuim), maar voor de sprints hierna moet de prioritering op orde zijn.

Het requirements proces zelf was duidelijk en volgens planning verlopen. De toevoeging van de voorbeeld grafieken creëerde inderdaad een stuk duidelijkheid bij de requirements.

De afwezigheid van de use cases op zich is jammer, maar deze zouden simpelweg geen nuttige informatie toevoegen aan dit project.

De kennis-sessies over JavaScript zijn goed verlopen, alhoewel dit wel een lastiger concept is dan Java. Tijdens het uitbreiden van de front-end zal moeten blijken of dit concept volledig over is gekomen; mijn verwachting is van wel. Blijk ik hier toch nog problemen mee te hebben dan kan ik “altijd” dezelfde collega om nog een sessie vragen.

# 2. Sprint 2: Medewerkers

| Opstarten | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 | Sprint 7 | Sprint 8 |
|---|---|---|---|---|---|---|---|---|

In §1.3 staat beschreven dat de opdrachtgever had aangegeven dat er voor deze sprint begonnen kon worden met de medewerkersaantallen of het ziekteverzuim; beide waren duidelijk gedefinieerd. Ik heb gekozen om te beginnen met de medewerkersaantallen. In beide gevallen is het zo dat er (natuurlijk) data moet worden laten zien in de vorm van een grafiek. Bij het ziekteverzuim is het echter zo dat er formules op de gegevens uitgevoerd moeten worden. Voor de eerste sprint waarin geprogrammeerd wordt leek het mij handig om te beginnen met alleen data-weergave, en de sprint hierna pas formules toe te voegen aan de werkzaamheden. Omdat ik dan al eerder een nieuwe grafiek heb toegevoegd zou dat gedeelte makkelijker/sneller moeten gaan, waardoor er meer tijd zou zijn voor het maken van de formules.

| *Activeit* | *Datum* |
|---|---|
| Sprint #1 Retrospective | 29 februari |
| Requirements m.b.t. werknemers | 29 februari – 3 maart, 7 – 10 maart |
| Ontwerpen systeemdeel (UML) | 29 februari, 1 maart |
| Bouwen systeemdeel | 1, 2, 3 maart |
| (Ontwerpen database) | 7 maart |
| (Aanpassen database) | 8 maart |
| Systeemdeel aanpassen voor samenwerking database | 7 – 10 maart |
| Werken aan stageverslag | 4 & 11 maart |

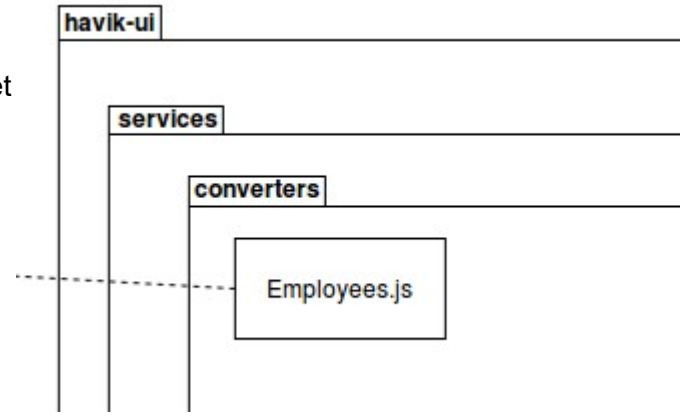
*Tabel 6: Planning sprint 2 volgens de sprint 1 retrospective.*

Voor het ontwikkelen van de eerste widget heb ik allereerst een UML klassendiagram opgesteld (§2.1). Hierna ben ik begonnen met het bouwen van het systeemdeel met mock-data (§2.2). Gedurende deze sprint ben ik ook bezig geweest met onderzoek naar Spring-Boot voor de presentatie die gegeven moet worden op de Knowledge Transfer Meeting (KTM) in de derde sprint (§2.3). Tijdens de sprint is er ook nog aan de requirements gewerkt (§2.4). Later is de mock-date in de widget vervangen door een in-memory database (§2.5). De afsluiting van de sprint gebeurt met de demo & code review (§2.6); hierna volgt nog mijn evaluatie over de gehele sprint (§2.7).

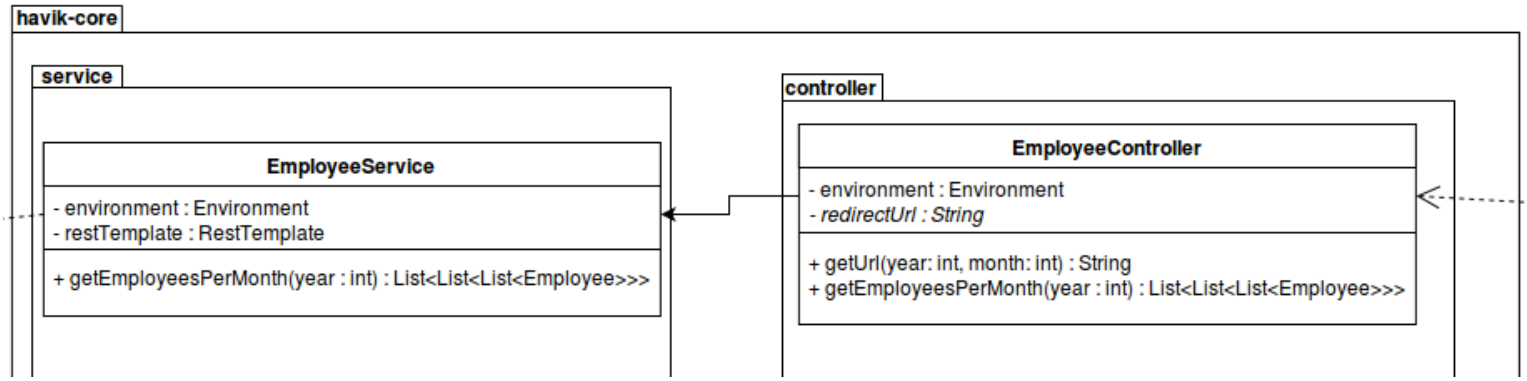
## 2.1. UML Klassendiagram

Er moeten in drie systemen toevoegingen gemaakt worden: Postduif, Havik-Core & Havik-UI. De toevoeging die op Havik-UI gemaakt moet worden is het toevoegen van een grafiek; hiervoor moet één bestand toegevoegd worden (Employees.js). Omdat JavaScript zelf geen klassen kent komen deze ook niet terug in het klassendiagram.

Deze klasse bevat vervolgens de URL om de medewerkers api van Havik-Core aan te spreken.




*Illustratie 10: Havik-UI.*



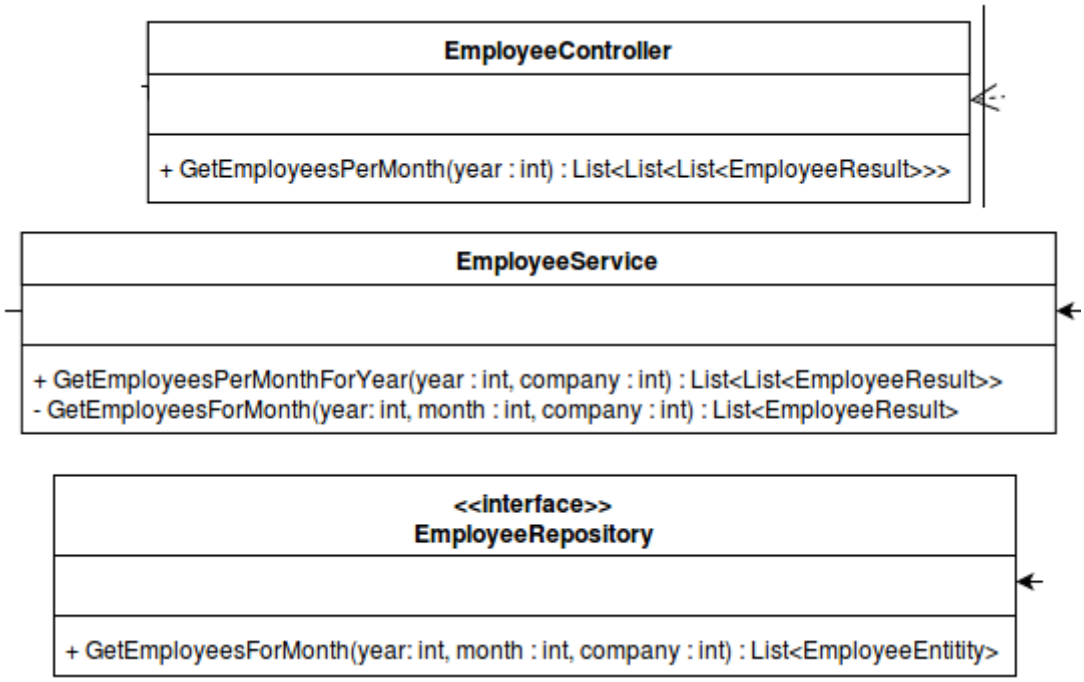

*Illustratie 11: Havik-Core.*

Hierin valt te zien dat er een List<List<List<Employee>>> teruggeven wordt naar de front-end. Deze is als volgt opgebouwd:

- De eerste list bevat alle werkmaatschappijen. Op dit moment zal de grootte van deze list dus meestal 2 zijn: 42 & Prepend;
- De tweede list bevat vervolgens de maanden; de grootte van deze is dus altijd 12;
- De derde list bevat vervolgens de werknemers.

Hierdoor zijn de werknemers per werkmaatschappij & per maand geselecteerd en te gebruiken in de front-end. De eerste widget laat de werknemersaantallen per maand zien in een staafdiagram (en per werkmaatschappij); door de bovenstaande verdeling is dit meteen op te halen.
Ook valt er op te merken dat er een List<Employees> teruggestuurd wordt en niet meteen een lijst met integers van de hoeveelheid werknemers in die maand. Hierdoor kan er op een balk geklikt worden en worden meteen de gegevens van de betreffende werknemers getoond; de database hoeft niet opnieuw bevraagd te worden.

In Postduif wordt de repository/database aangesproken om de medewerkers op te halen. Dezelfde opbouw als bij Havik-Core valt hier op te merken: de List<List<List<EmployeeResult>>>. In §V3.1. staat beschreven waarom er hier met EmployeeResults gewerkt wordt; wanneer deze doorgegeven worden via de REST api naar Havik-Core kan dit gemapped worden op de Employee klasse die daar gebruikt wordt.



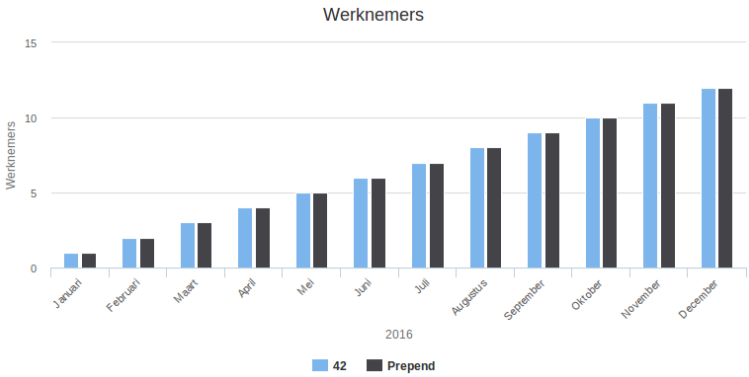

*Illustratie 12: Uitvergroting Postduif.*

## 2.2. Programmeren: Medewerkers widget

Na het maken van het klassendiagram ben ik begonnen aan het programmeren van de nieuwe widget. Beginnend bij het maken van de UI (in Havik-UI) was er veel te kopiëren van de andere al bestaande widgets. Kleine veranderingen bestonden vooral uit het fine-tunen van de grafiek: zetten dat het een staaf-diagram is, x-as waarden zetten & URL's corrigeren.

### 2.2.1. Havik-UI

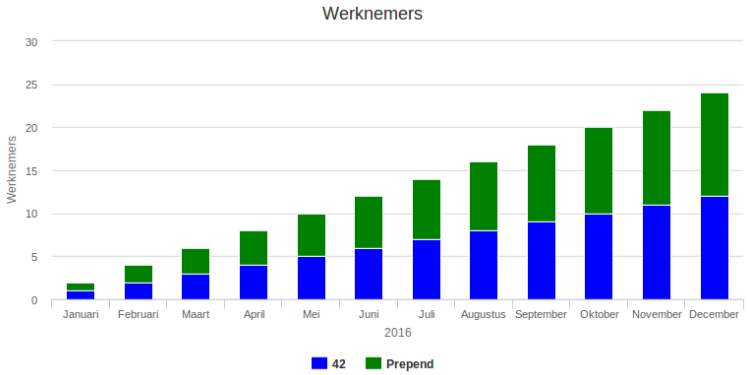
Om de grafiek vervolgens te bekijken was er uiteraard data nodig om te laten zien. Omdat ik Havik-Core nog niet hierin betrokken heb heb ik generieke/hardcoded data gemaakt in Havik-UI zelf. Na dit in de grafiek te plaatsen was de hiernaast staande illustratie het resultaat:




*Illustratie 13: Eerste versie medewerkers-widget.*

### 2.2.2. Havik-Core

De volgende stap was om de data uit Havik-Core te krijgen. Hiervoor zijn de controller & service aangemaakt in Havik-Core (zie klassendiagram in §2.1.). Vervolgens heb ik dezelfde data die hiervoor gebruikt werd in de UI gebruikt. Daarnaast heb ik de grafiek veranderd van een staaf naar een "stacked" staaf grafiek (naar aanleiding van tekeningen van de opdrachtgever).




*Illustratie 14: Data uit Havik-Core.*

## 2.2.3. Detailgegevens

Eén van de requirements is dat wanneer er op een kolom geklikt wordt er een scherm naar boven komt met gegevens over de werknemers die destijds werkten bij het bedrijf. Nog steeds werkend met gefingeerde data uit Havik-Core, heb ik deze data veranderd in Math.Random() data; hierdoor weet ik zeker dat wanneer er op een staaf geklikt wordt er de hoeveelheid werknemers uit de staaf gepakt wordt en niet ergens anders vandaan.

*Illustratie 15: Medewerkers-widget met random data & data display na klikken.*

Note: het detailscherm staat in het midden van het scherm: de zijkanten van de illustratie zijn er af gesneden.
Eerder staat beschreven dat de widget een aantal aanpassingen nodig had. Om dit scherm te kunnen maken was de grootste aanpassing. Het klikken op data is een functie die al gemaakt was. De bestaande functie zorgde er echter voor dat wanneer er op een staaf geklikt wordt dat er een nieuwe pagina opent welke naar het bijbehorende systeem gaat (Mus in dit geval). Deze functie is bij deze grafiek echter niet van toepassing, omdat de data uit Havik's eigen database gaat komen.

De huidige functionaliteit wordt op het moment dynamisch aan elke grafiek toegevoegd. Om dit te overriden heb ik een filter toegevoegd zodat grafieken met historische data deze functionaliteit niet krijgen. Een mooiere oplossing zou zijn om deze manier van toevoegen opnieuw te maken en per grafiek de correcte klik-functionaliteit toe te voegen. Dit zou echter meer tijd kosten. Deze keuze zou later besproken moeten worden met mijn begeleider en/of opdrachtgever.

## 2.2.4. Veranderen van jaartal

De volgende toevoeging was de mogelijk om het geselecteerde jaartal te kunnen veranderen. De volgende elementen zijn hiervoor toegevoegd:

- Een functie die de data in de grafiek veranderd; (zie §2.2.6.)
- Twee functies die aan de HighChart gebonden zitten
  - Deze zetten het cijfer/jaar (in de illustratie 2014) met één omhoog of omlaag en roepen vervolgens de functie van het eerste punt aan;
- Twee buttons in HTML die de functies van het vorige punt aanroepen.

Een requirement is dat de werkmaatschappijen apart aan/uit te zetten moeten zijn. Als er bijvoorbeeld op 42 geklikt wordt in de legenda dat deze wegvalt uit de grafiek. Dit is een functionaliteit dit al in de HighCharts library zelf zit; dit hoef ik niet meer toe te voegen.

*Illustratie 16: Keuze van jaartal.*

### 2.2.5. JavaScript: Promises

Bij het maken van de functie die de data van de grafiek veranderd moet er nieuwe data uit Havik-Core opgehaald worden. Dit gebeurt via een HTTP request. De functionaliteit om data op te halen bestond al; dit was al gemaakt door mijn voorganger. Deze functie deed echter meer dan alleen data ophalen; er was dus een nieuwe functie nodig die alleen data ophaalde. Deze heb ik gemaakt en vervolgens proberen te gebruiken.

Er deed zich toen een probleem voor wanneer ik deze gebruikte: de grafiek bleef leeg wanneer de data toegevoegd werd. In de code kwamen de termen “$q”, “deferred” & “promise” voorbij; drie termen die ik niet kende. Na hier op te zoeken en hier zelf kennis van proberen op te doen blijken deze drie termen veel met elkaar te maken te hebben; het concept zelf is echter vrij lastig. Om deze reden heb ik nog een kennis sessie met de JavaScript expert ingepland en gedaan.

*Voor u als lezer: promises zijn een vrij lastig aspect van JavaScript en dit kan ik niet op één pagina uitleggen. Het heeft te maken met asynchrone threads en functionaliteit die moet wachten tot zo’n asynchrone thread klaar is. Heeft u hier meer interesse in (het is zeker een interessant onderwerp) raad ik u aan om bij de literatuurlijst van dit document te kijken en de bijbehorende links te bestuderen.*

Kort & simpel samengevat: een $http call geeft een promise/werkt asynchroon. Wanneer deze promise terugkomt kan deze geresolved of gereject worden. Daarnaast bevat de promise de data die opgehaald is uit de REST API (via de $http call). Wanneer iets via een promise veranderd wordt is het niet meer handig om dezelfde variabelen aan te spreken buiten de promise; omdat deze asynchroon wordt veranderd weet je niet wanneer het welke data bevat.

*Deze samenvatting is alles behalve allesomvattend; voor meer info raad ik opnieuw aan om bij de websites in de literatuurlijst te kijken.*

Om de nieuwe functie om data te veranderen in de grafiek te laten werken moest deze code dus ook asynchroon/met deze promises gaan werken. Toen dit eenmaal toegevoegd was werd de data in de grafiek correct gewijzigd; de grafiek kon nu dus dynamisch veranderd worden.

## 2.3. Spring-Boot

Mijn begeleider had aangegeven dat hij de presentatie over Spring-Boot tijdens de demo van deze sprint wilde zien. Daarom heb ik deze gedurende deze sprint gemaakt. De presentatie is te vinden in de bijlagen (bijlage D).

## 2.4. Requirements

De opdrachtgever had een klein aantal opmerkingen/toevoegingen op het requirements document. Na deze toegevoegd te hebben moest de feedback van mijn begeleider/de architect nog ontvangen worden. Toen ik deze uiteindelijk ook had ontvangen was er geen tijd meer om deze te verwerken; dit zal in de volgende sprint moeten gebeuren.

## 2.5. Database

Verder gaande op het programmeren zou de volgende stap zijn om “echte” data te gebruiken. Hiervoor moet de data die Havik nodig heeft in de database moeten komen.

### 2.5.1. Design

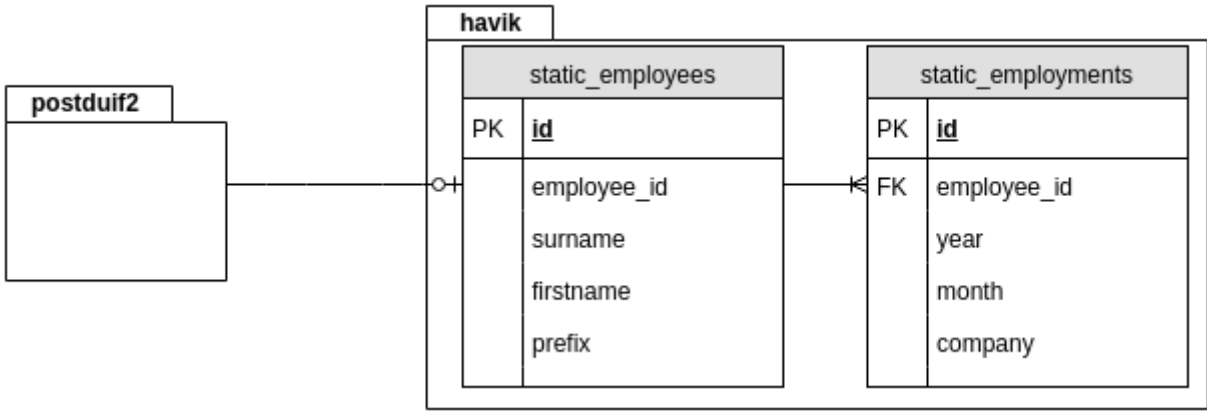
Voor deze widget zijn er twee mogelijkheden om de data op te slaan in de database.
De eerste is:

- Eén tabel;
- Hierin wordt vervolgens de medewerker met al zijn gegevens bijgehouden;
  - Ook worden zijn datums van in- & uitdiensttreding hierbij bijgehouden;
- Dit zorgt dat ervoor dat de database sneller is en het dure proces in de code zit;
  - Dit komt doordat alle medewerkers dan opgevraagd worden en er vervolgens in de code vergeleken moet worden of de datum die opgevraagd wordt tussen de in- & uitdiensttreding datums ligt;
- Het synchroniseren van de Havik database met de Postduif database is hierbij ook een duur proces. Elke medewerker moet vergeleken worden of deze nog bestaat en vervolgens of zijn in- & uitdiensttreding data nog klopt (en anders moet dit gecorrigeerd worden in Havik).

De tweede is:

- Twee tabellen;
- In de eerste tabel wordt de medewerker opgeslagen met al zijn gegevens;
- In de tweede tabel wordt vervolgens opgeslagen in welke maanden deze gewerkt heeft;
- Hierbij wordt er per maand één record per medewerker opgeslagen;
- Dit zorgt voor een grotere database;
  - De opdrachtgever heeft eerder aangegeven dat hij geen problemen heeft met hoeveel data er in de database bijkomt;
  - De architect heeft aangegeven dat de hoeveelheid data irrelevant weinig is; ~42 entrees per maand is niet schokkend veel;
- Dit zorgt ervoor dat het dure proces bij de database ligt, in plaats van bij de code;
  - Dit omdat er nu simpelweg één datum opgevraagd moet worden, waarbij de database hierop moet selecteren, niet de code.
- Het synchroniseren van databases is hierbij goedkoper. Elke medewerker die op het moment actief is krijgt een nieuw record in de tweede tabel; alle data die er al in staat hoeft niet vergeleken/aangepast te worden.

In overleg met de architect is er gekozen om voor de tweede methode te gaan. Het UML diagram hiervan is te zien in de volgende illustratie:




*Illustratie 17: Database design werknemers.*

De hoeveelheid gegevens bij de medewerkers (static_employees) is nog niet compleet.

Er moet nog met de opdrachtgever besproken worden welke gegevens er van een medewerker moeten worden laten zien wanneer er in de GUI klikt wordt. Mogelijk kan dit design aangepast worden naar aanleiding van deze requirement (omdat er data opgeslagen moet worden die de opdrachtgever graag terug ziet komen in de GUI).

## 2.5.2. Postduif in-memory database

Vervolgens kon dit geïmplementeerd worden in de al bestaande database. Voor nu implementeer ik dit in de database van Postduif. Zodra alles werkt kan dit in één keer gekopieerd worden naar Havik. Als er zich dan problemen voordoen in Havik is er een grote kans dat de code/implementaties zelf correct zijn, maar dat er bijvoorbeeld connecties niet goed zijn.

De bestaande in-memory database van Postduif wordt opgezet d.m.v. een .xml bestand. Hier heb ik mijn eigen tabellen aan toe kunnen voegen.

Het toevoegen van data werkt vervolgens via een .SQL bestand. Na zelf wat testdata te hebben verzonnen heb ik deze hier ook aan toegevoegd; dit zijn twee queries geworden.

```
<createTabel tableName="static_employees">
    <column name="id" type="bigint">
        <constraints primaryKey="true" nullable="false"/>
    </column>
    <column name="employee_id" type="bigint">
        <constraints nullable="false"/>
    </column>
    <column name="surname" type="varchar(30)">
        <constraints nullable="false"/>
    </column>
    <column name="firstname" type="varchar(30)">
        <constraints nullable="false"/>
    </column>
    <column name="prefix" type="varchar(10)"/>
</createTabel>

<createTabel>
    <column name="id" type="bigint">
        <constraints primaryKey="true" nullable="false"/>
    </column>
    <column name="employee_id" type="bigint">
        <constraints nullable="false"/>
    </column>
    <column name="year" type="int">
        <constraints nullable="false"/>
    </column>
    <column name="month" type="int">
        <constraints nullable="false"/>
    </column>
    <column name="company" type="int">
        <constraints nullable="false"/>
    </column>
</createTabel>
```

*Illustratie 18: Xml toevoegingen Havik-medewerkers.*

### 2.5.3. Programmeren

Tot dusverre werkt de widget met random data/getallen gegenereerd in Havik-Core. Nu de database in Postduif toereikend genoeg is om de widget te ondersteunen moeten er twee veranderingen plaatsvinden: de data moet uit Postduif komen i.p.v. Havik en de data moet uit de database komen i.p.v. gegenereerd te worden.

In de illustratie van het klassendiagram van Postduif (§2.1) staan de klassen en methodes die hiervoor benodigd waren. Het implementeren hiervan ging vrijwel probleemloos. Er lag nu een werkende medewerkers widget met data uit nieuwe tabellen in de Postduif database.

## 2.6. Demo & Code Review

Ook deze sprint wordt weer afgesloten met een demo. Gelijk aansluitend hieraan is de code review waarin mijn begeleider, en later eventueel andere experts, kijken naar de kwaliteit van de code.

### 2.6.1. Demo

Bij de demo waren zowel de opdrachtgever als mijn begeleider aanwezig. Het belangrijkste onderwerp was het feit dat het sprint doel niet behaald is. Het doel van de sprint was om een compleet werkende medewerkers widget te hebben liggen; dit ligt er echter niet. Wat er op dit moment ligt is een widget die gegevens uit de database van Postduif haalt, deze vervolgens kan laten zien, en indien er op geklikt wordt de detailgegevens laat zien. De bijbehorende diagrammen zijn hierbij ook opgesteld.

Wat er nog mist zijn de volgende elementen:

- De database moet in Havik zitten, niet in Postduif;
- Als er op data geklikt wordt van de huidige periode moet het detailschermp niet naar boven komen, maar moet er gelinkt worden naar een ander systeem;
- Naast de in-memory database moet er een PostgreSQL database gemaakt & gebruikt kunnen worden;
- De hele widget moet getest worden.

De opdrachtgever was toch tevreden over wat er tot dusverre lag; er is besloten dat er nog een sprint aan deze widget wordt besteed om de bovenstaande punten te verhelpen.

Ook heb ik de presentatie over Spring-Boot aan mijn begeleider gegeven. De presentatie op zich is goed, maar op de KTM staat er 15 minuten gereserveerd; hier kwam ik niet aan. Voordat de KTM plaatsvindt zou ik hier nog wat tijd aan moeten besteden om de presentatie langer te maken.

## 2.6.2. Code Review

Tijdens de code review zijn er meerdere punten uitgekomen die veranderd/verbeterd konden worden in de gemaakte code. De punten waren allemaal kleine verbeteringen; de code bevat geen extreem grote fouten of functionaliteit die gewoonweg opnieuw gemaakt zou moeten worden. Het grootste probleem is voorgelegd aan mijn begeleider:

> Bestaande Havik functionaliteit is dat er standaard een aantal URL's vastzitten aan de data in de grafieken. Dit zorgt ervoor dat wanneer er geklikt wordt op een dataset in een grafiek die over het "nu" gaat dat er doorgelinkt wordt naar het correcte systeem en de relevante gegevens gelijk wordt laten zien. Bij historische gegevens hoeft/kan dit echter niet, omdat de data hiervan in Havik zit en niet in Postduif.

Dit is bij de nieuwe widget op dit moment opgelost met een workaround.

> De ideale situatie voor het bovenstaande probleem zou zijn dat de huidige functionaliteit herschreven wordt.
>
> (…)
>
> De huidige manier werkt, maar mooier zou zijn om dit dynamisch in DashboardCtrl.js op te lossen (waar de URL klik-functie nu wordt geset) in plaats van om dit per historische grafiek handmatig te overriden en zelf te setten.
>
> – Code Review Sprint 2 – notulen

Ook wordt de List<List<List<StaticEmployee>>> veranderd naar integer waardes die teruggestuurd worden. Het klikken op een balk in de grafiek moet dan opnieuw gegevens opvragen, maar dit gebeurt waarschijnlijk te weinig om deze vertraging op te merken. Daarnaast zal het ophalen van de gegevens om de grafiek samen te stellen sneller zijn, wat meer baten oplevert dan het sneller kunnen laden van de gegevens wanneer er geklikt is.

## 2.7. Evaluatie

De feedback op de requirements van mijn begeleider was dat deze nog niet SMART genoeg waren. Allebei zaten we te denken aan dat deze vanaf scratch herschreven zouden worden i.p.v. de huidige nummering & style aan te houden. Dit zal waarschijnlijk de volgende sprint gebeuren.

> Wat er nog mist zijn de volgende elementen:
>
> • De database moet in Havik zitten, niet in Postduif;
>
> • Als er op data geklikt wordt van de huidige periode moet het detailscherm niet naar boven komen, maar moet er gelinkt worden naar een ander systeem;
>
> • Naast de in-memory database moet er een PostgreSQL database gemaakt & gebruikt worden;
>
> • De hele widget moet getest worden.
>
> – §2.6.1

> Waarom is dit tijdens de sprint niet gelukt: een verkeerde tijdsinschatting. Er zijn teveel nieuwe technieken voorbij gekomen waar ik geen rekening mee heb gehouden. Er is veel tijd besteed aan deze technieken te leren; nuttig bestede tijd, maar tijd die niet aan de widget besteed kon worden.
>
> Wat houdt dit in voor het project: er zal nog een sprint toegewijd moeten worden aan deze widget. Gedurende deze sprint kunnen de bovenstaande onderdelen geïmplementeerd worden.
>
> – Sprint 2 Retrospective

Het niet halen van het sprint-doel is jammer. Hierdoor kan er mogelijk aan het eind van het project één widget minder af zijn als “gepland”.

De presentatie van Spring-Boot moet nog iets uitgebreid worden. Ook zal ik mijn kennis moeten ophogen van Spring-Boot. De presentatie maak ik mij geen zorgen om; de vragen achteraf is waar ik problemen in voorzie. Omdat de enige kennis die ik heb theorie-kennis is, opgedaan vanuit de officiële reference guide en andere voorbeelden online, zullen praktijk-vragen wellicht lastig zijn. Er is geen tijd om een project op te starten om voldoende praktijk ervaring op te doen.
Ik heb de collega die mij een demo heeft gegeven in de 0-sprint (§V.2.2) gevraagd om met eventuele lastige vragen te helpen op de KTM. Hopelijk zorgt dit ervoor dat eventuele praktijk-vragen goed beantwoord kunnen worden.

# 3. Sprint 3: Medewerkers Database & Unittesting

| Opstarten | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 | Sprint 7 | Sprint 8 |
|---|---|---|---|---|---|---|---|---|

Ik ben begonnen met het verbeteren van de requirements (§3.1). Voordat ik begon met programmeren heb ik eerst het UML diagram van het systeem verbeterd (§3.2); de database was nog correct. Hierna ben ik verdergegaan met het programmeren hiervan (§3.3). Ondertussen zijn er nog meerdere kennis sessies geweest voor elementen die nu zouden terugkomen in het project (§3.4). Vervolgens kwam de KTM (§3.5). Het op een na laatste onderdeel van deze sprint was het testen van de widget (§3.6); de sprint is opnieuw afgesloten met de demo (§3.7). Het hoofdstuk wordt uiteraard afgesloten met de evaluatie over de sprint (§3.8).

| *Activeit* | *Datum* |
|---|---|
| Sprint #2 Retrospective | 14 maart |
| Requirements m.b.t. werknemers | 14 maart – 17 maart, 21 – 24 maart |
| Design systeemdeel aanpassen | 14, 15 maart |
| In-memory database naar Havik verplaatsen | 15, 16 maart |
| PostgreSQL database bouwen + met Havik laten werken | 16, 17 maart |
| Spring-Boot + KTM | 21, 22 maart |
| Testen (design + maken + uitvoeren) | 23, 24 maart |
| Werken aan stageverslag | 18 & 25 maart |

*Tabel 7: Planning sprint 3 volgens de sprint 2 retrospective.*

## 3.1. Requirements

Naar aanleiding van de feedback van mijn begeleider (en mijn eigen mening) heb ik het requirements document herzien. Nummeringen zijn aangepast/duidelijker gemaakt en de requirements zijn minder ambigu. De volledige nieuwe versie van requirements is te vinden in de bijlagen (bijlage E).

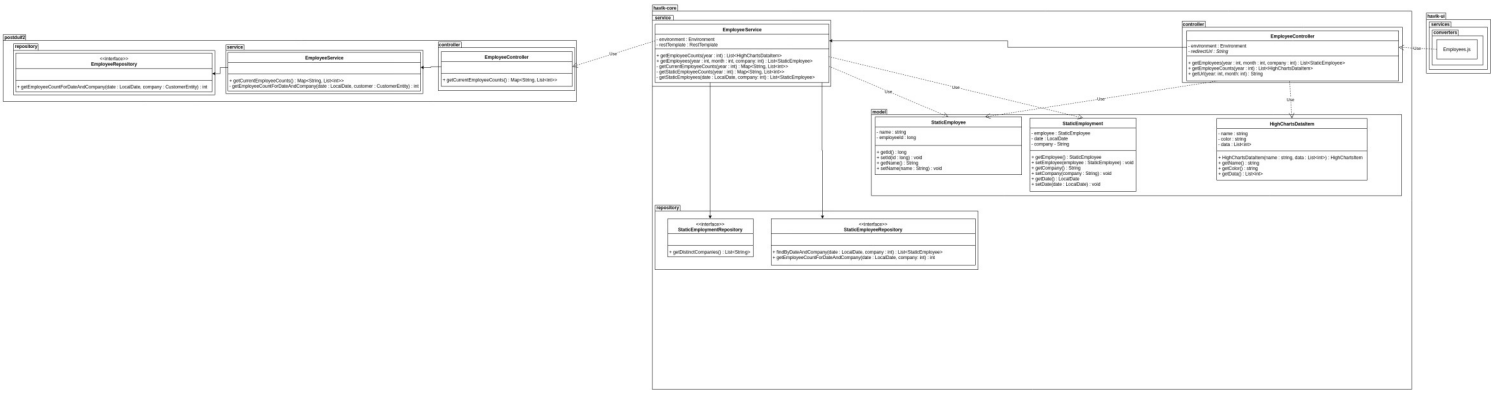
| # | Benodigd Requirement | | Oorsprong |
|---|---|---|---|
| 1.5. | 1.1. | Havik moet in een staafdiagram alle werknemers van een jaar kunnen laten zien. Hierbij wordt de hoeveelheid werknemers uitgezet tegenover de maanden. | Interview #1 Opdrachtgever<br>Interview #3 Opdrachtgever |
| | | Elke staaf is hierin opgedeeld in verschillende partities: één per werkmaatschappij. | Interview #3 Opdrachtgever |
| | | De werkmaatschappijen moeten apart aan en uit te zetten zijn. Ook moeten de werknemersgroepen apart aan en uit te zetten zijn. | Interview #3 Opdrachtgever<br>Review #1 Opdrachtgever |
| | | De selectie moet voor-/achteruit in de tijd kunnen scrollen met sprongen van één jaar (waardoor er op het scherm altijd een volledig jaar zichtbaar is, van januari t/m december). | Review #1 Opdrachtgever |
| | | De selectie moet niet verder terug in de tijd kunnen gaan dan 2003. | Interview #4 Opdrachtgever |
| | | De selectie mag ook niet verder in te tijd kunnen gaan dan het huidige jaar. | Interview #4 Opdrachtgever |
| | | Als er geklikt wordt op een staaf moet er een scherm zichtbaar worden die de werknemers van die maand van de bijbehorende werkmaatschappij laat zien. Zie de bijbehorende tabel welke gegevens er van de medewerkers moet worden laten zien. | Interview #3 Opdrachtgever<br>Interview #4 Opdrachtgever |
| | | Werknemers die later in de maand zijn aangenomen (dus niet de gehele maand hebben gewerkt) tellen mee voor de hele maand; deze hoeven niet speciaal aangegeven worden. | Interview #4 Opdrachtgever |

*Tabel 8: Requirement 1.5. uit Requirements V4.0.*

Zodra deze goedgekeurd zijn door zowel de opdrachtgever als mijn begeleider kunnen deze eindelijk geprioriteerd worden.

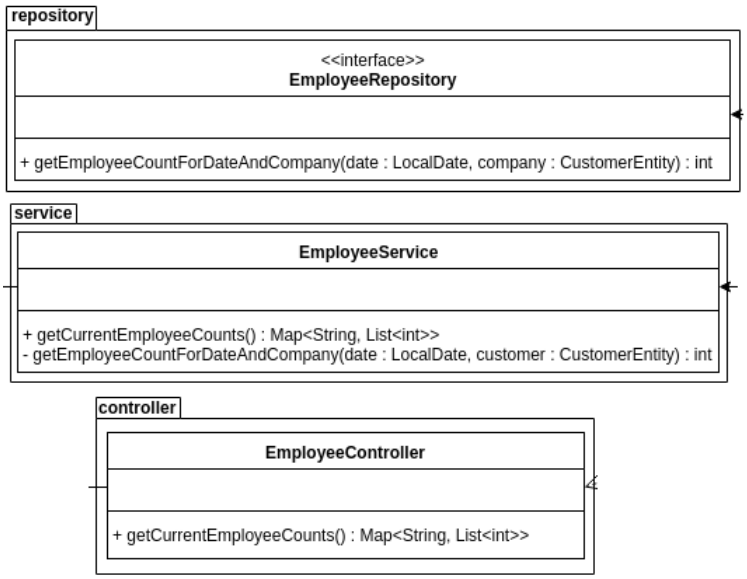
## 3.2. UML Diagrammen

Voordat ik verder ben gegaan met programmeren heb ik het UML diagram aangepast van de widget. Het diagram van de database is onveranderd gebleven; de data hierin is nog steeds correct en compleet.



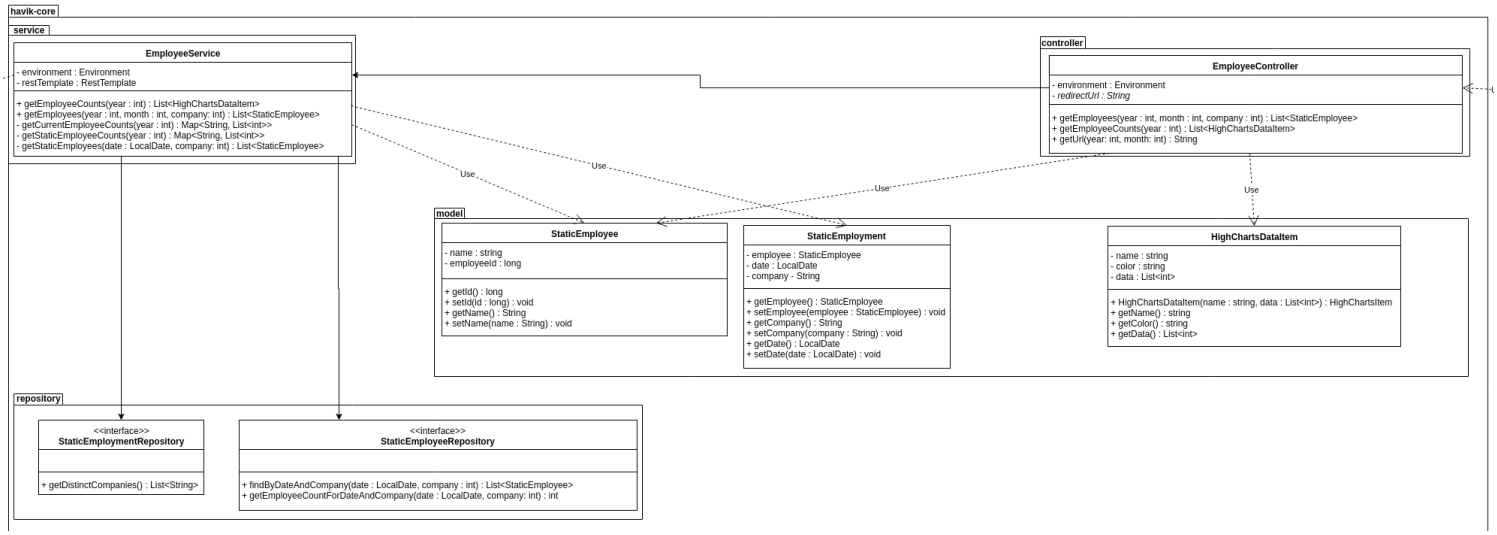
*Illustratie 19: Nieuwe versie van het klassendiagram. Deze is opgesplitst te zien op de huidige & komende pagina. Onder de diagrammen staan vervolgens de uitleg van dat gedeelte van het diagram.*

Het complete diagram valt in de illustratie hierboven te zien; deze is ook te vinden in de bijlagen (bijlage F). In Havik-UI zijn geen nieuwe klassen bijgekomen. Employee.js krijgt wel een verandering: optimalisatie voor HighCharts wordt hieruit gehaald en toegevoegd aan Havik-Core; dit op aanraden van mijn begeleider tijdens de laatste code review om zo veel mogelijk hiervan in




*Illustratie 20: Nieuwe versie van de toevoegingen aan Postduif.*

Java te houden.
De grootste verandering in Postduif is het weghalen van Employees; deze zijn niet nodig in Havik. Uit Postduif komen alleen de huidige werknemers; maar om deze te bekijken moet Havik deeplinken naar Mus; de Employees zelf worden dus nooit uit Postduif gehaald binnen deze widget. Daarnaast wordt er geen List<List<List<Employee>>> meer teruggegeven, maar een Map<String, <List<Integer>>. Hierbij is de key de naam van het bedrijf (“42 BV” of “Prepend” op dit moment) en de List bevat de hoeveelheid werknemers per maand.




*Illustratie 21: Nieuwe versie van de toevoegingen aan Havik-Core.*

In de illustratie hierboven valt de toevoegingen aan Havik-Core te zien. De volgende punten zijn interessant om op te merken:

- Er zijn twee repositories toegevoegd (linksonder); StaticEmployeesRepository & StaticEmploymentRepository. Deze zijn de voorbode van het verplaatsten van de de tabellen van Postduif naar de nieuwe database in Havik.
- Er zijn drie nieuwe enitities:
  - StaticEmployee & StaticEmployment: deze staan gelijk aan de records in de database;
  - HighChartsDataItem: een object zoals HighCharts deze in Havik-UI wilt ontvangen. Onderdeel van het verplaatsten van de optimalisatie voor HighCharts van Havik-UI naar Havik-Core.
- Er wordt niet meer met een List<List<List<>>> gewerkt, maar met een Map<String, List<Integer>>; dit is eerder in dit hoofdstuk uitgelegd.

## 3.3. Programmeren: Database

Nadat het klassendiagram verbeterd was kon de code hierop aangepast worden. Hiervoor moesten de gemaakte tabellen in Postduif verzet worden naar Havik-Core. Omdat Havik-Core zelf nog geen database (connectie) had moest deze nieuw gemaakt worden. De connecties konden gelegd worden d.m.v. klassen die ook in Postduif stonden; deze kopiëren en aanpassen zorgde ervoor dat de connectie werkend was. Er lag nu een werkende (maar lege) in-memory HSQL database.

Het vullen van de HSQL database wordt gedaan via twee SQL bestanden; de opbouw hiervan was ook af te leiden uit de opbouw van de bestanden in Postduif.

De volgende stap was om naast de in-memory database een PostgreSQL database op te zetten. Hiervoor hebben mijn begeleider en ik een sessie gehouden waarin we deze hebben opgezet en waarin ik voldoende uitleg over PostgreSQL heb gekregen om hier in de toekomst mee verder te gaan.

## 3.4. Kennis sessies

Gedurende deze sprint hebben er vier kennis sessies plaatsgevonden. De sessie over PostgreSQL staat al benoemd in §3.3.

Er is ook een sessie geweest over het testen in Java. Hierin hebben we de frameworks Junit & JMockit besproken. Er kwamen onderwerpen in terug zoals het niet bezitten van een Springcontext in de tests en het mocken van method calls. De basics van Java testing is mij hierin uitgelegd.

Er is ook een sessie geweest over de Beanmapper. Dit is een open-source project gemaakt door 42. Zoals eerder uitgelegd wordt er gewerkt met Forms, Entities & Results. De Beanmapper is een klasse/library die deze omzet naar de andere vormen. Er hoeven dus geen aparte methodes gemaakt te worden om deze drie klassen tussen elkaar om te zetten; daar is de Beanmapper voor gemaakt.

De laatste sessie van de sprint ging over het testen in Javascript. Deze sessie is niet gebeurd met mijn begeleider, maar met de andere collega waarmee ik al eerdere Javascript sessies heb gehad. Bij het testen van Javascript wordt het framework Jasmine gebruikt. Tijdens deze sessie zijn de basics van het testen van Javascript code mij uitgelegd.

## 3.5. Knowledge Transfer Meeting (KTM)

De Knowledge Transfer Meeting is een evenement waar het gehele bedrijf bij aanwezig is. Hierbij worden verschillende presentaties gegeven over nieuwe/interessante technieken die overwogen kunnen worden om binnen 42 te gaan gebruiken. In §1.2 staat geschreven dat ik hierop een presentatie zou geven over Spring-Boot omdat dit niet meer in mijn project terug zou komen.

De presentatie is goed verlopen, maar hierover meer in de evaluatie.

De presentatie zelf is terug te vinden in de bijlagen (Bijlage D).

## 3.6. Testen

Standaard bij 42 is om ‘alle’ methodes te testen; 100% test coverage dus, zolang de tests nuttig zijn. In Havik-Core heb ik unittests gemaakt die alle methodes in beide models & repositories en de controller test.

Voor de andere tests (Postduif & Havik-UI) heb ik uitleg nodig van anderen. De tests bij Postduif zijn groot en verwarrend; voor deze heb ik uitleg nodig van één van de makers. Aangezien deze maar één/twee keer per week aanwezig zijn zou dit pas de volgende sprint lukken. Hetzelfde geldt voor de tests van Havik-UI. Deze heb ik tijdens de code review (§3.7) besproken met mijn begeleider; om deze volledig te begrijpen/op verder te kunnen gaan heb ik meer uitleg nodig over wat er precies binnen Havik-UI gebeurt (mijn begeleider raadde ook aan om hulp te vragen aan de expert van Javascript).

## 3.7. Demo & Code Review

De demo is deze sprint anders verlopen; mijn begeleider en opdrachtgever waren niet tegelijk beschikbaar. Hierdoor heb ik de demo twee keer gegeven. Het nadeel hierbij is dat nieuwe punten die besproken worden dan niet met allebei tegelijkertijd wordt besproken; hierdoor krijg ik twee meningen i.p.v. meteen een consensus.

### 3.7.1. Demo

In de demo is besproken wat er nu af is en wat er de volgende sprint gaat gebeuren. De medewerkers widget is vrijwel af; wat er nog moet gebeuren zijn de volgende punten:

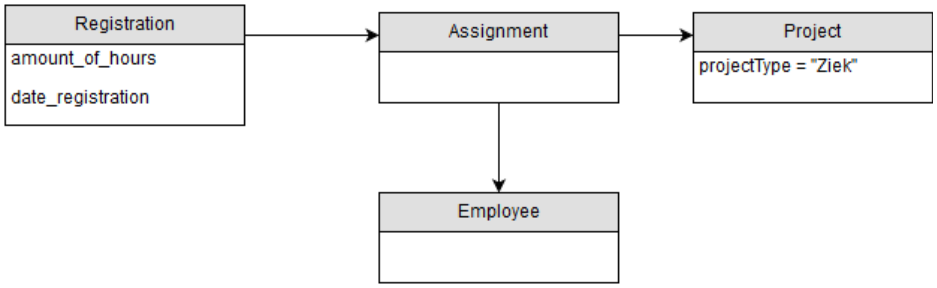
> • Deeplinken naar Mus;
>
> • Support voor medewerkers die niet op de eerste van de maand zijn begonnen;
>
> • Testen van Postduif-functioes & Havik-UI;
>
> • Hiervoor is hulp van respectievelijk Quadcore & Maarten nodig;
>
> • Punten die in de code review naar boven zijn gekomen
>
> – Sprint 3 Retrospective

Deze punten zijn op zich vrij klein, en kunnen in de volgende sprint opgelost worden; dit is besproken met zowel de opdrachtgever als begeleider. Dit is geen probleem.

Daarnaast is er besloten om de volgende sprint de widget te maken met betrekking tot Ziekteverzuim. Ook is er besproken om Havik binnenkort naar de test-omgeving te brengen. Hiervoor zou ik met Operations moeten kijken wat er allemaal benodigd is voor een deployment. Operations is nu dus een nieuwe stakeholder in dit project geworden.

Ik verwacht zelf dat de widget niet binnen één sprint gaat lukken; de nieuwe formule kan een probleem worden dat meer tijd nodig gaat hebben. Mijn begeleider verwacht dat het wel binnen één sprint te doen is. Met de juiste planning denk ik dat hij gelijk kan hebben; ik ga dus proberen om de widget binnen één sprint te maken.

Tijdens de demo met mijn begeleider hebben we een groot onderdeel voor het ziekteverzuim uitgedacht: de formule hiervoor en waar deze data in Postduif te vinden is. Benodigd voor het ziekteverzuim zijn de hoeveelheid uren dat iemand ziek is geweest en op welke datum. Ook de werknemer zelf is van belang uiteraard.
In de database is een apart project met als type “Ziek”; hier worden de ziekte-uren op geschreven. Aan dit project hangen Assignments. Aan een Assignment hangt vervolgens één medewerker en (meerdere) Registrations. Een Registration bevat de gegevens over wanneer en voor hoeveel uur (per dag).




*Illustratie 22: Database tracing voor ziekteverzuim.*

Voorbeeld: is medewerker A drie dagen ziek dan is dit zo opgebouwd:

- Er hangt één assignment aan het project “Ziek”;
- aan deze assignment hangt employee A;
- aan deze assignment hangen ook drie registrations; elk met één dag gespecificeerd en de hoeveelheid uren dat deze medewerker per dag ziek was (bij een contract van 40 uur waarschijnlijk dus 3x 8 uur).

## 3.7.2. Code Review

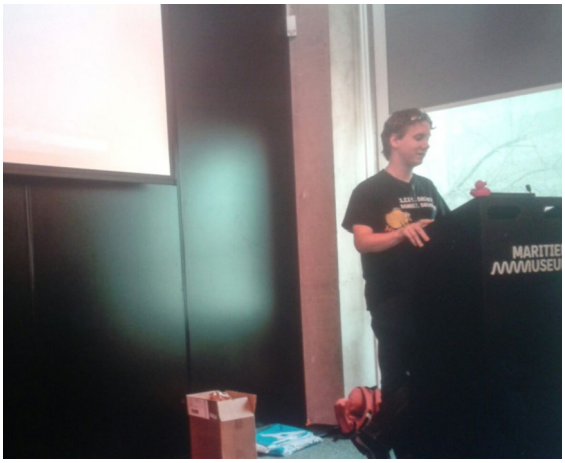
Tijdens de code review hebben we onder andere gekeken naar wat er toegevoegd gaat worden voor de komende widget. Daarnaast hebben we een onderdeel van de tests besproken; zoals in §3.6. geschreven staat moet voor de Javascript tests verder met de Javascript expert gesproken worden.

## 3.8. Evaluatie

Het herschrijven van de requirements was een grote verbetering. Het document/de requirements zelf zien er nu beter en duidelijker uit; ze zijn ook een stuk minder ambigu. Feedback moet jammergenoeg nog komen; gelukkig heb ik wel genoeg om verder te kunnen werken.

De knowledge sessions zijn nuttig geweest; ik heb veel kennis opgedaan. De Beanmapper uitleg was ben ik bang minder nuttig. De kennis op zich is goed om te hebben zodat ik snap wat er bij andere projecten/systemen gebeurt, maar binnen Havik wordt de Beanmapper niet gebruikt (deze is niet nodig). Javascript testing is opnieuw een lastig concept. Zoals in §3.6. is gebleken lastig genoeg dat ik hier toch wat extra info over moet hebben; in ieder geval over de tests die nu al bestaan en hoe deze werken.

De presentatie op de KTM ging meer dan goed. De presentatie zelf ging precies zoals verwacht; de vragen die gesteld werden waren ook goed te beantwoorden. Er werden enkele praktijk vragen gesteld; om deze goed te beantwoorden zou ik meer ervaring met Spring-Boot moeten hebben; deze kon ik zelf dus ook niet beantwoorden. Gelukkig werden deze meteen beantwoord door de collega (met meer ervaring) die ik speciaal voor dit soort vragen gevraagd had mij te helpen.



*Illustratie 23: Het geven van de presentatie op de KTM.*

De unittests zelf zijn voorlopig voldoende om Havik te testen. Zodra Havik in de test of acceptatie omgeving draait wordt deze uitgebreider getest door het gebruik doordat deze gebruikt wordt (acceptatie & explorerende tests).

Het is jammer dat de widget nog een klein aantal open punten heeft. Gelukkig zijn dit kleine punten die waarschijnlijk binnen een dag opgelost zijn, maar toch is het jammer dat dit niet 100% binnen de sprint is gelukt.

# 4. Sprint 4: Ziekteverzuim

| Opstarten | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 | Sprint 7 | Sprint 8 |
|---|---|---|---|---|---|---|---|---|

Deze sprint is toegewijd aan het maken van de volgende widget: het weergeven van het ziekteverzuim van één jaar.

| *Activiteit* | *Datum* |
|---|---|
| Sprint #3 Retrospective | 29 maart |
| Laatste punten medewerkers-widget* | 29 & 31 maart |
| Requirements m.b.t. ziekteverzuim | 30 maart & 4 april |
| UML diagrammen aanpassen n.a.v. nieuwe widget | 30 maart |
| Ontwikkelen front-end ziekteverzuim | 31 maart & 4 april |
| Tests ontwikkelen front-end ziekteverzuim | 31 maart & 4 april |
| Ontwikkelen back-end ziekteverzuim | 5, 6 & 7 april |
| Tests ontwikkelen back-end ziekteverzuim | 5, 6 & 7 april |
| Afstudeerverslag | 1 & 8 april |

*Tabel 9: Planning sprint 4 volgens de sprint 3 retrospective.*

Deze sprint ben ik begonnen met de laatste punten van de medewerkers widget af te maken (§4.1). Hierna heb ik de requirements voor de huidige sprint verder uitgewerkt (§4.2). Uiteraard moest het design aangepast worden om de nieuwe widget te kunnen accommoderen (§4.3). Hierna ben ik begonnen met het programmeren van de front-end (§4.4). Na de front-end gemaakt te hebben heb ik de back-end gemaakt (§4.5). Aan het einde van de sprint was er opnieuw een demo (§4.6). Dit hoofdstuk eindigt met mijn evaluatie (§4.7).

## 4.1. Medewerkers-widget

Aan de medewerkers-widget moesten nog drie dingen gebeuren:

- Deeplinken naar Mus;
- Support voor medewerkers die niet op de eerste van de maand zijn begonnen;
- Tests.

Voor de deeplink naar Mus is de klik methode in Havik-UI van de widget aangepast. Deze kijkt nu eerst of het huidige jaar wordt laten zien en de huidige maand is aangeklikt. Is dit het geval dan wordt er direct een nieuwe pagina geopend die opent bij Mus.

Voor het supporten van medewerkers die niet op de eerste dag van de maand zijn begonnen is de query aangepast die deze ophaalt. Deze heeft nu de zowel de eerste als de laatste dag van de maand nodig. Vervolgens moet de medewerker voor de laatste dag van de maand zijn aangenomen en na de eerste dag van de maand ontslag hebben genomen (of nog werkende zijn); zijn beide condities waar dan heeft deze minimaal één dag gewerkt in de maand en dan mag deze meegeteld worden.

```
SELECT COUNT(em.employee)
FROM Employment em
WHERE ( em.resignationDate IS NULL OR em.resignationDate > :firstDate)
AND em.commencementDate <= :lastDate
AND em.subsidiary = :subsidiary
```

*Illustratie 24: Postduif query: support voor medewerkers die niet op de eerste van de maand zijn begonnen.*

De query in Havik is hier ook op aangepast. Omdat de medewerkers in de database van Havik alleen maar worden opgeslagen wanneer deze werkzaam zijn (met de betreffende maand) was dit een kleinere aanpassing. De query accepteert nu ook een eerste en laatste dag van de maand, maar vergelijkt vervolgens of de datum waarmee de record is opgeslagen hier tussen valt. De datum dat iemand is begonnen met werken (of uit dienst is gegaan) wordt niet opgeslagen, dus hier kan/hoeft niet op gechecked te worden.

```
SELECT COUNT(se.employee)
FROM HistoricEmployee emp, HistoricEmployment se
WHERE se.date BETWEEN :firstDayOfMonth AND :lastDayOfMonth
AND se.company = :company
AND se.employee.employeeId = emp.employeeId
```

*Illustratie 25: Havik query: support voor medewerkers die niet op de eerste van de maand zijn begonnen.*

Voor de tests in Postduif had ik uitleg nodig van één van de makers hiervan (zoals beschreven in §3.6). Na deze gekregen te hebben heb ik deze ook gemaakt. Het grootste probleem was het aanmaken van data hiervoor. Omdat alle klassen ontzettend samenhangen (vrijwel alle klassen hebben objecten nodig van andere klassen om deze goed te kunnen gebruiken) is het opbouwen van data voor een test bijna net zo groot als de test zelf.

Ook voor de UI zijn bestaande tests aangepast/heb ik een nieuwe test gemaakt. Er bestond al een test die alle HighCharts methodes test; deze is aangepast om de nieuwe HighChart ook te testen. Ook is er een nieuwe methode bijgekomen die data opvraagt van Havik-Core. Deze methode wordt nu ook getest; wordt de correcte HTTP call gemaakt.

## 4.2. Requirements

Deze sprint heeft als doel een nieuwe widget. De eerste stap hiervoor is uiteraard opnieuw het duidelijk krijgen van de requirements; specifiek de requirements met betrekking tot deze widget. Hiervoor is er opnieuw een gesprek met de opdrachtgever voorgekomen. Het resultaat van dit gesprek, de requirements, is hieronder te vinden.

| # | | *Benodigd Requirement* | *Oorsprong* |
|---|---|---|---|
| 1.4. | 1.1. | Havik moet een staafdiagram kunnen weergeven die het ziektepercentage van één jaar laat zien. Hierbij wordt het percentage tegenover de maanden uitgezet. | Interview #3 Opdrachtgever<br>Interview #5 Opdrachtgever |
| | | Elke staaf is hierin opgedeeld in verschillende partities: één per werkmaatschappij. | Interview #3 Opdrachtgever<br>Interview #5 Opdrachtgever |
| | | De werkmaatschappijen moeten apart aan en uit te zetten zijn. | Interview #3 Opdrachtgever |
| | | De selectie moet voor-/achteruit in de tijd kunnen scrollen met sprongen van één jaar (waardoor er op het scherm altijd een volledig jaar zichtbaar is, van Januari t/m December). | Review #1 Opdrachtgever |
| | | De selectie moet niet verder terug in de tijd kunnen gaan dan 2003. | Interview #5 Opdrachtgever |
| | | De selectie mag ook niet verder in te tijd kunnen gaan dan het huidige jaar. | Interview #5 Opdrachtgever |
| | | Als er geklikt wordt op een staaf, welke historische data weergeeft, moet er een scherm zichtbaar worden die de zieke werknemers van die maand van de bijbehorende werkmaatschappij laat zien. Zie de bijborende tabel welke gegevens er van de medewerkers moet worden laten zien. | Interview #3 Opdrachtgever<br>Interview #5 Opdrachtgever |
| | | Wordt er vervolgens op één van de werknemers in dit scherm geklikt moet er een deeplink zijn naar systeem Mus. Bij Mus moet vervolgens automatisch ingelogd worden met dezelfde gegevens die gebruikt zijn om bij Havik in te loggen. Mus moet vervolgens uitkomen op de urenregistratiepagina van de specifieke werknemer waar op geklikt is voor die maand. | Interview #5 Opdrachtgever |
| | | Wordt er geklikt op een staaf die data van de huidige periode laat zien moet deze ook uitkomen bij Mus via een deeplink. Er moet bij Mus automatisch ingelogd worden met dezelfde inloggegevens die gebruikt zijn om bij Havik in te loggen. Mus moet vervolgens uitkomen op de pagina van het project “Ziekte”, waar alle medewerkers worden laten zien die ziek zijn. | Interview #5 Opdrachtgever |

*Tabel 10: Requirement 1.4 volgens Requirements V5.0.*

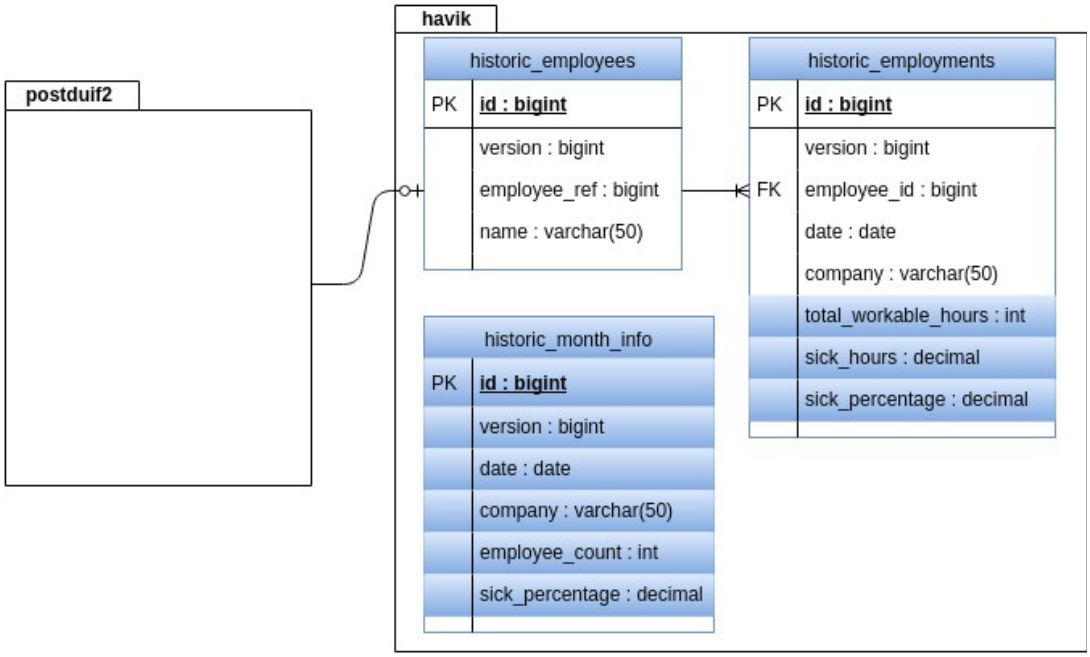
Mijn begeleider had hier vervolgens twee punten feedback op. Aangezien het gaat om percentages is een “stacked column” grafiek niet handig; percentages kunnen niet op deze manier opgeteld worden.
Ook zou mijn begeleider graag willen wanneer er op data van de huidige periode wordt geklikt er precies hetzelfde gebeurt als wanneer er op historische data wordt geklikt.
Wanneer deze punten opgelost zijn zouden de requirements goed zijn voor de requirements; alleen de feedback/goedkeuring van de opdrachtgever is eerst nog benodigd.

Beide punten zijn later besproken met de opdrachtgever; hij was het hier mee eens. De requirements zijn hierop aangepast.

## 4.3. Design

Na de requirements is het design aangepast om de nieuwe widget te kunnen accommoderen. Zowel het design van de database als het systeem is aangepast.

### 4.3.1. Database



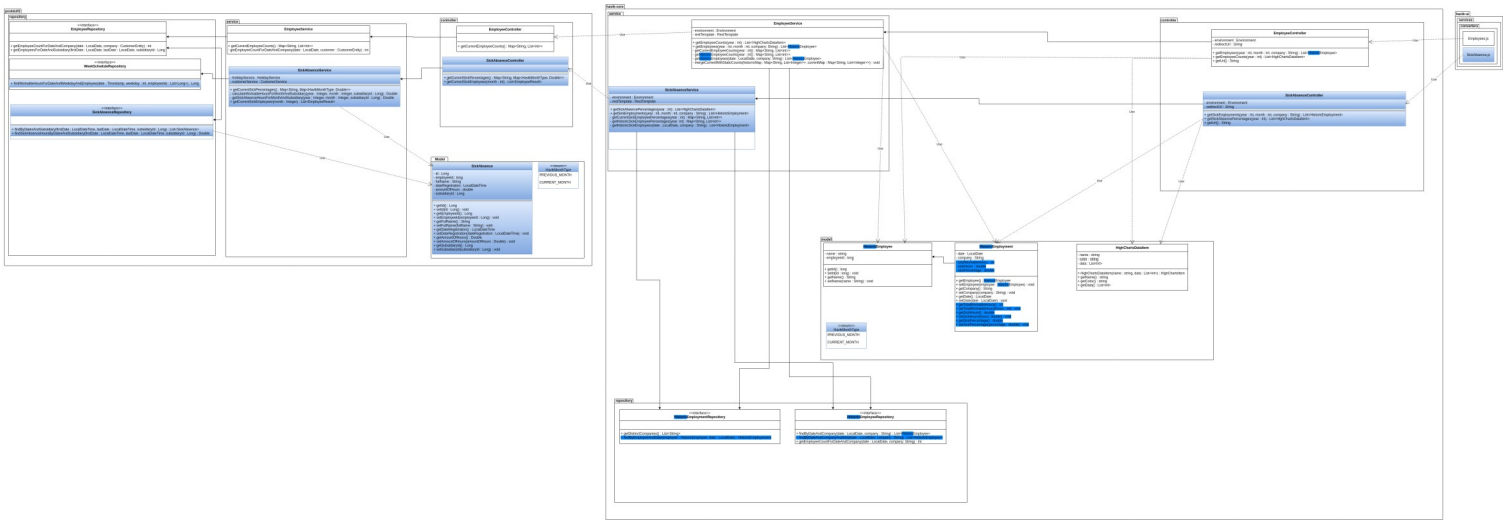

*Illustratie 26: Nieuw database design. Alles wat blauw gehighlight is komt er nieuw bij of wordt veranderd in het huidige systeem.*

De naamgeving is veranderd naar aanleiding van feedback van mijn begeleider: “static” is veranderd naar “historic” (bijvoorbeeld: “static_employees” naar “historic_employees”).
Daarnaast hebben employments drie nieuwe kolommen: per maand kan er per werknemer aangegeven worden hoeveel deze had kunnen werken en hoeveel deze ziek was.

Ook is er een nieuwe tabel: deze tabel zou de data per maand opslaan. Hierdoor zou er in Havik niet meer per medewerker de percentages berekend hoeven te worden (de werknemers aantallen kunnen hier ook aan toegevoegd worden, waardoor deze ook niet meer berekend hoeven te worden). Dit is een optie die eerst besproken moet worden met mijn begeleider/de architect; de uiteindelijke gegevens, het ziektepercentage, is ook uit te rekenen via de data per medewerker. Rechtvaardigt een nieuwe tabel hiervoor het versnellen van het ophalen van de gegevens voor de UI is een vraag die ik bij mijn begeleider/architect neer zou moeten leggen.

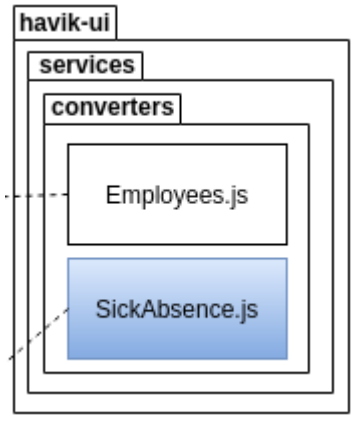
Het antwoord op deze vraag is benodigd voordat het design van Havik-Core volledig afgemaakt kan worden; dit zou namelijk een model toevoegen en veranderd zijn service. Deze zijn dus nog niet gemaakt; de rest van het design is wel gemaakt:

### 4.3.2. Systeemdeel



*Illustratie 27: Nieuw programmatuur design. Alles wat blauw gehighlight is komt er nieuw bij of wordt veranderd in het huidige systeem. Belangrijke onderdelen worden hieronder uitvergroot & beschreven. Voor een vergroting van het gehele diagram, zie bijlage G.*

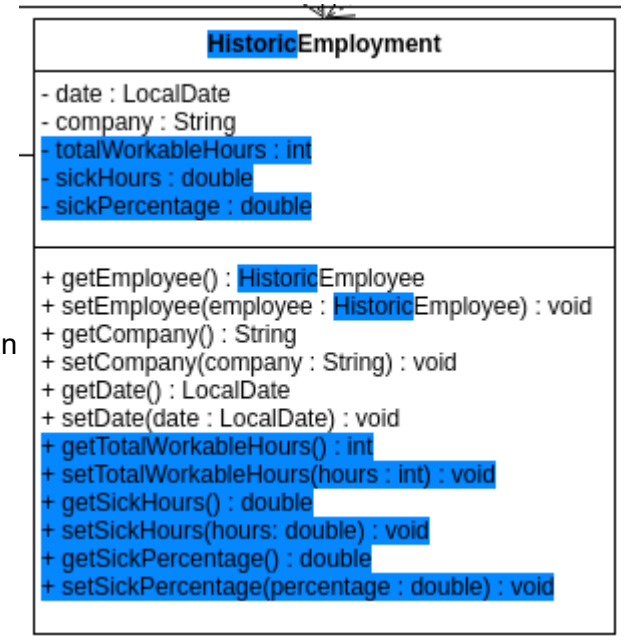
Voor de UI komt er nieuw bestand bij die de grafiek maakt. Zie de illustratie hiernaast




*Illustratie 28: Nieuwe Havik-UI.*

Voordat het grootste gedeelte van Havik-Core ontworpen kan worden is er feedback van mijn begeleider nodig met betrekking tot de tabel in database.
Wat wel ontworpen kan worden is de uitbreiding op HistoricEmployment. Deze moet met de database tabel mee veranderen, en komt er vervolgens dus zoals hiernaast uit te zien.

Verder kan ook de connectie naar Postduif ontworpen worden; dit is opnieuw niet afhankelijk van feedback van mijn begeleider.

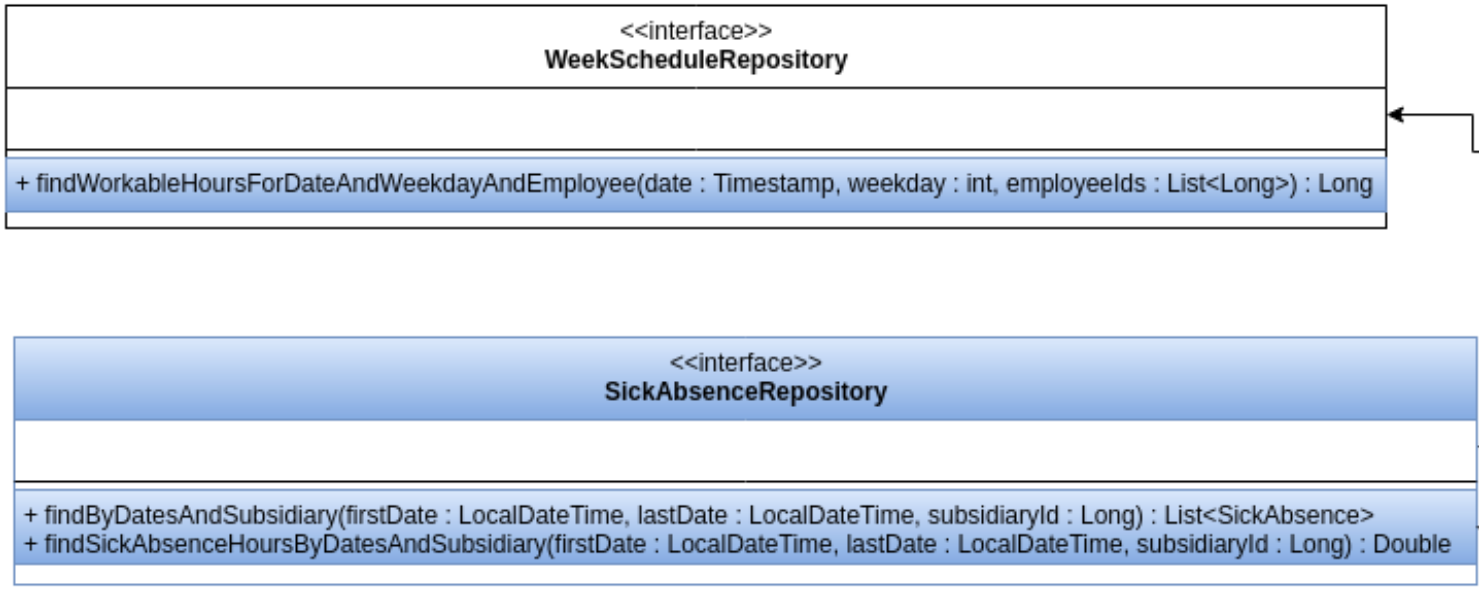



*Illustratie 29: Nieuwe versie HistoricEmployment.*


*Illustratie 30: Nieuwe klasse: SickAbsenceService.*

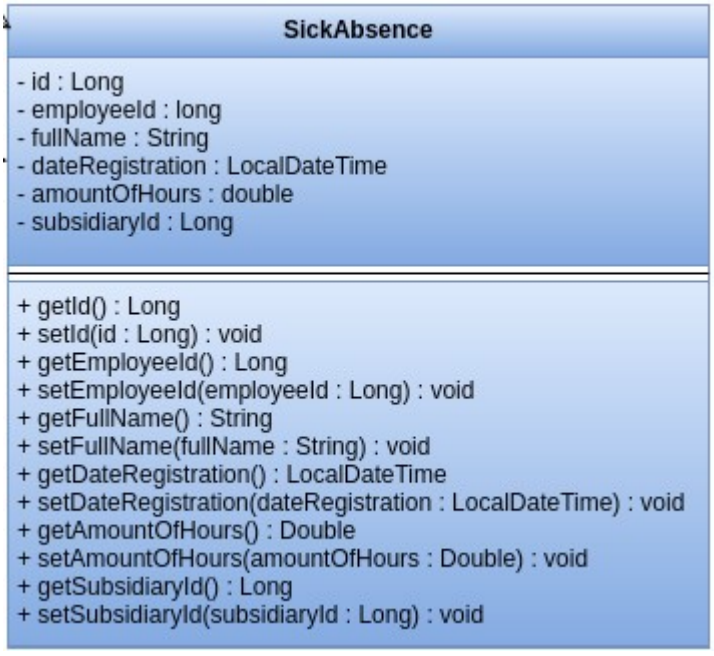
Voor deze connectie moet er een nieuwe Service komen. De methodes die percentages ophalen geven een Map<String, List<int>> terug; het idee hierbij is dat de String de naam van het bedrijf is en de List de percentages per maand bevat.

Bij het ophalen van de zieke werknemers zelf wordt er een HistoricEmployment teruggeven. Het idee hierbij is dat er in de UI meteen de relevante data (ziektepercentage per medewerker) beschikbaar is en de naam van de medewerker via de HistoricEmployee opgehaald kan worden die opgeslagen staat in de HistoricEmployment.




*Illustratie 31: Postduif repository aanpassingen/toevoegingen.*

De toevoegingen voor Postduif zijn ook ontworpen. Het probleem hiervoor is dat er niet één locatie is om de data op te halen. Er zijn meerdere tabellen waar data uit benodigd is; zie §3.7.1. Tijdens het ontwerpen van dit gedeelte van het systeem kwam dit probleem naar voren; hoe en waar zou dit opgehaald moeten worden. Hierbij werd ik op het idee gebracht om een database view aan te maken hiervoor. Hierbij hoort vervolgens een nieuwe repository (zie bovenstaand) en een nieuw model (zie hiernaast).
De verandering aan WeekScheduleRepository is voor het ophalen van de totale werkbare uren van een medewerker per dag. WeekSchedule is een bestaande tabel die per medewerker de hoeveelheid minuten (niet uren) per dag van de week (maandag, dinsdag, etc) opgeslagen heeft staan. Dit wordt per contract bijgehouden; veranderd het contract van een medewerker wordt er hier een record toegevoegd.




*Illustratie 32: Postduif SickAbsence.*

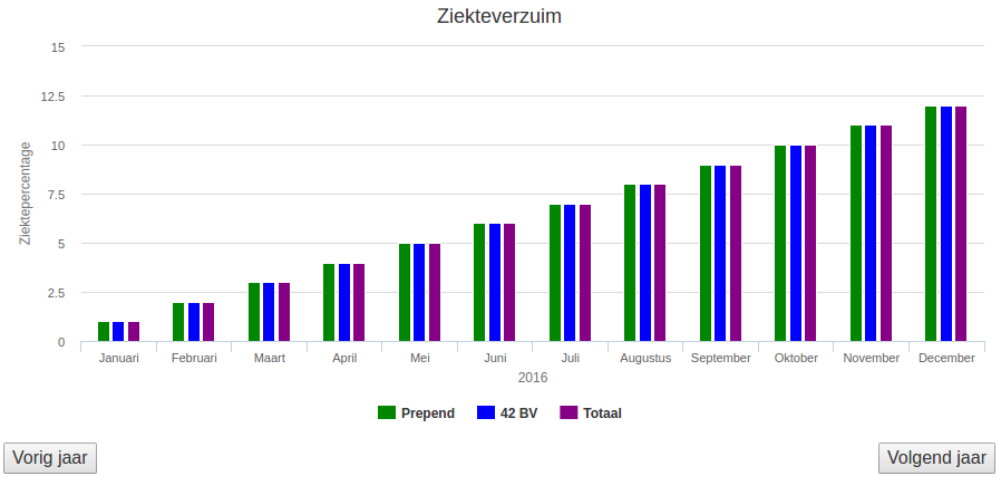
SickAbsence is de representatie van de view in het systeem. Alle info die benodigd is door Havik staat hierin opgeslagen; om welke werknemer het gaat, wanneer deze ziek was, voor hoeveel uur deze ziek was, en voor welk bedrijf deze destijds werkte.

## 4.4. Programmeren: Ziekteverzuim front-end

Met programmeren ben ik in de front-end/Havik-UI begonnen. Veel code was direct te kopiëren van de vorige (medewerkers) widget. Aanpassingen waren wel benodigd, zoals de URL waarvan data wordt opgehaald.

Om te checken of de nieuwe widget correct data laat zien heb ik vervolgens de Controller gemaakt in Havik-Core; deze geeft nu nog hardcoded data terug aan de UI. Aangezien de data correct werd weergegeven werkte deze dus correct.

Uiteraard is deze widget vervolgens toegevoegd aan de tests van de UI; ook dit was voor een groot gedeelte te kopiëren van de medewerkers-widget.




*Illustratie 33: Ziekteverzuim-widget met hardcoded data.*

## 4.5. Programmeren: Ziekteverzuim back-end

### 4.5.1. Havik-Core: Service

Verder programmeren aan Havik-Core was niet mogelijk; hiervoor moest ik allereerst feedback ontvangen van mijn begeleider/de architect over de extra tabel in de database. Om deze reden ben ik verder gegaan met het programmeren van de toevoegingen aan Postduif. Om deze connectie te leggen moest er wel wat gebeuren in Havik-Core; er moest een Service komen die deze connectie legt. De functionaliteit die de service voor nu moet hebben bestaat alleen uit twee API calls naar Postduif: voor het ophalen van de ziektepercentages en voor het ophalen van de daadwerkelijke medewerkers met tijden dat ze ziek waren.

### 4.5.2. Postduif: SickAbsence

Hierna konden de toevoegingen aan Postduif gemaakt worden. Het eerste grote probleem hierbij was het maken van de view voor SickAbsence. Er worden hiervoor veel tabellen gekoppeld tot één view; het uitzoeken van de exacte keys die gebruikt moesten worden kostte wat tijd. Een groter probleem was het uitzoeken van welk weekrooster (waarin staat opgeslagen hoeveel werkbare uren een persoon had per dag van de week) er bij welke ziekte-registratie behoorde. Uiteindelijk heb ik dit laten vallen en hier 2 aparte queries van gemaakt; deze 2 samenvoegen zou teveel tijd/werk gekost hebben.

```
SELECT r.id,
e.id AS employee_id,
CONCAT_WS(' ', pe.firstname, pe.prefix, pe.surname) AS full_name,
r.dateregistration as date_registration,
r.amountofhours as amount_of_hours,
emp.subsidiary_id
FROM Registration r
INNER JOIN Assignment a ON r.assignment_id = a.id
INNER JOIN Project p ON p.id = a.project_id
INNER JOIN ProjectType pt ON pt.id = p.projecttype_id
INNER JOIN Employee e ON e.id = a.employee_id
INNER JOIN Person pe ON e.person_id = pe.id
INNER JOIN Employment emp ON emp.employee_id = e.id
WHERE pt.type = 'Ziek'
AND r.dateregistration >= emp.commencement_date
AND (emp.resignation_date >= r.dateregistration OR emp.resignation_date IS NULL)
```

*Illustratie 34: view_sick_absence query.*

De uiteindelijke query is hierboven te zien. De belangrijke informatie wordt in de view bijgehouden:

- De werknemer met ID & naam;
- De datum dat deze ziek was en voor hoeveel uur (als iemand bijvoorbeeld halverwege de dag ziek naar huis gaat staat dit niet gelijk aan de hoeveelheid werkbare uren);
- Bij welke werkgever de medewerker destijds in dienst was.

De twee WHERE statements (exclusief “pt.type = ‘ziek’”) zijn benodigd om te bepalen voor welk bedrijf de werknemer destijds in dienst was. Dit is benodigd bij mensen die bij beide bedrijven (42 & Prepend) gewerkt hebben.

## 4.5.3. Postduif: WeekSchedule

Het tweede grote probleem was de query voor het uitzoeken van het weekrooster per datum. De tabel waarin dit opgeslagen staat (WeekSchedule) staat hiernaast als UML-diagram. De problemen die zich hierbij voordoen:

- Startdate: de startdatum wanneer een rooster in is gegaan. Een einddatum wordt echter niet bijgehouden. Er moest dus per dag bekeken worden wat het laatste voorgaande rooster was.
- Minutesmonday – minutessunday: de hoeveelheid minuten die een medewerker werkt volgens het rooster. Dit wordt per dag opgeslagen en kan per dag anders zijn. Omdat een maand niet standaard op dezelfde dag begint & eindigt moet er om een maand te berekenen dit per dag geëvalueerd worden.
- minutes_per_week: de hoeveelheid minuten die een medewerker in een volledige week zou moeten werken. Deze kan leeg zijn en in de test-database staat deze bij het merendeel niet ingevuld. Er van uitgaande dat de testdatabase van Postduif representatief is voor de echte productie database kan minutes_per_week dus niet gebruikt worden. Daarnaast doet ook bij deze het probleem voor dat een maand niet altijd op dezelfde dag begint/eindigt.

| WeekSchedule | |
|---|---|
| PK | **id : bigint** |
| | version : bigint |
| | startdate : timestamp |
| | minutesmonday : int |
| | minutestuesday : int |
| | minuteswednesday : int |
| | minutesthursday : int |
| | minutesfriday : int |
| | minutessaturday : int |
| | minutessunday : int |
| | remarks : varchar |
| | employee_id : bigint |
| | minutes_per_week : int |

*Illustratie 35: UML WeekSchedule.*

De uiteindelijke query staat hiernaast. Deze heeft 3 parameters nodig:

- Weekday: de dag van de week (1 = maandag; 7 = zondag); nodig om de correcte kolom op te halen;
- Date: de datum waar het om draait;
- EmployeeIds: de employees waar het om draait. Van te voren moet/kan de selectie gemaakt worden waar de uren van opgehaald worden; gefilterd op medewerkers die op die datum actief waren voor een bepaalde werkmaatschappij (of voor beide werkmaatschappijen).

Tot dusverre zijn alle queries die gemaakt zijn JPA queries geweest; dit is de eerste native SQL query. Dit komt omdat de JPA queries niet goed werkte met de SUM in combinatie met CASE en niet goed werkte met een subselect in een FROM statement.

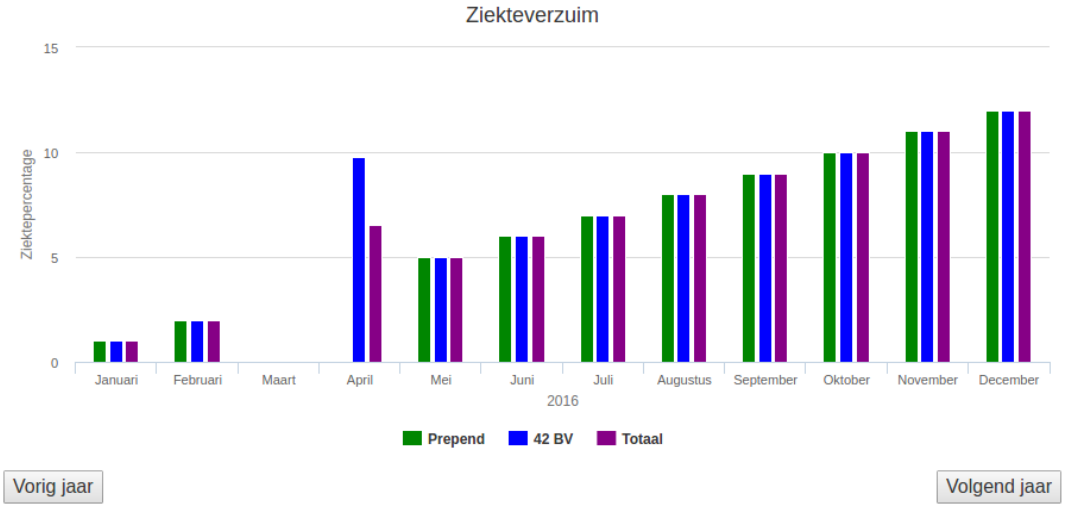
```
SELECT SUM( CASE :weekday
            WHEN 1 THEN ws.minutesmonday
            WHEN 2 THEN ws.minutestuesday
            WHEN 3 THEN ws.minuteswednesday
            WHEN 4 THEN ws.minutesthursday
            WHEN 5 THEN ws.minutesfriday
            WHEN 6 THEN ws.minutessaturday
            WHEN 7 THEN ws.minutessunday
            END)
FROM (  SELECT w.employee_id, MAX(w.startdate) AS max_startdate
        FROM WeekSchedule w
        WHERE w.startdate <= :date
        AND w.employee_id IN :employeeIds
        GROUP BY w.employee_id ) w
INNER JOIN weekschedule ws
ON ws.employee_id = w.employee_id
AND w.max_startdate = ws.startdate
```

*Illustratie 36: Uiteindelijke WeekSchedule query.*

## 4.5.4. Havik-Core

Nu de correcte data opgehaald werd kon deze teruggeven worden naar Havik-Core. Deze moet hier geconverteerd worden naar data die de front-end/Highcharts kan weergeven. Na dit geïmplementeerd te hebben ziet de widget er zoals hieronder uit:

De historische data is nog steeds hard-coded uit Havik-Core, maar de data van de huidige periode wordt correct uit Postduif opgehaald.




*Illustratie 37: Ziekteverzuim-widget met huidige periode-data.*

## 4.5.5. TODO

Het einde van de sprint is nu in zicht en er zijn nog verschillende onderdelen niet gemaakt:

> Functionaliteit die nog toegevoegd moet worden:
>
> - De in-memory database van Havik-Core moet uitgebreid worden om ziekteverzuim op te slaan;
> - De in-memory database van Havik-Core moet gebruike worden;
> - Beide bovenstaande punten gelden ook voor de PostgreSQL database;
> - De medewerkers die ziek waren moeten worden laten zien wanneer er op een staaf geklikt wordt;
> - Als er vervolgens op een medewerker geklikt wordt moet er via een deeplink een nieuw pagina bij Mus openen;
> - De widget moet goed getest worden; een aantal tests zijn al gemaakt, maar het grootste gedeelte moet nog getest worden.
>
> – Sprint 4 retrospective

## 4.6. Demo

In de demo is besproken wat er nu nog moet gebeuren voor de ziekteverzuim-widge. Deze punten zijn te veel & te groot; het doel om deze sprint de widget af te hebben is dus ook niet behaald. In de sprint retrospective (bijlage H) staat de exacte indeling van de sprint beschreven, samen met een uitleg waarom dit niet behaald is. De belangrijkste opsomming:

> Kort gezegd, er was teveel werk gepland in te weinig tijd.
>
> • Het afmaken van de medewerkers-widget heeft meer tijd gekost dan verwacht;
>
> • De requirements zijn wel volgens planning verlopen;
>
> • Het design is later nog veranderd door een beslissing over een extra tabel in de Havik database; dit heeft later in de sprint ook nog tijd gekost (ongepland);
>
> • De UI zelf had 2 dagen ingepland gekregen; deze moet nu nog getest worden. Had er inderdaad 2 dagen besteed kunnen worden aan de UI had dit waarschijnlijk af geweest;
>
> • De twee grote queries zijn twee onverwachte activiteiten die ongepland veel tijd hebben gekost;
>
> • De back-end heeft niet genoeg tijd kunne krijgen i.v.m. andere activiteiten die teveel tijd in beslag namen Hier zal nu nog wel een dag of 2-3 aan besteed moeten worden; Havik-Core + de database van Havik moeten aangepast worden voor de nieuwe widget en de tests voor Havik & Postduif moeten gemaakt worden.
>
> • De sprint was een dag korter i.v.m. Pasen.
>
> All in all: activiteiten hadden meer nodig dan gepland.
>
> – Sprint 4 retrospective

Er is nu afgesproken om de volgende sprint ook nog aan deze widget te spenderen. Daarnaast is er afgesproken om de volgende sprint de widget naar de test omgeving te deployen.

De code review die normaal volgt na de demo heeft nog niet plaatsgevonden; deze vindt zo vroeg mogelijk in de volgende sprint plaats.

## 4.7. Evaluatie

De medewerkers widget is af. Dit op zich is alvast een goed begin van de sprint en een goed resultaat.

De front-end van de ziekteverzuim widget is ook af; dit is ook een goede uitkomst van de sprint. Dat de back-end nog veel te doen heeft is echter een grote zorg. Uiteindelijk had ik toch misschien standvastiger moeten zijn en twee sprints voor de widget moeten inplannen (zie §3.7.1). Aan de andere kant is het een goede les “wat te doen als een sprintdoel niet behaald is”: goed evalueren wat er fout is gegaan, hiervan leren, en vervolgens doorgaan en dezelfde fout(en) niet nogmaals maken.

# 5. Sprint 5: Ziekteverzuim & Deployment

| Opstarten | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 | Sprint 7 | Sprint 8 |
|---|---|---|---|---|---|---|---|---|

> Het doel van de vijfde sprint is om de widget van de vorige sprint (ziekteverzuim) af te maken. De bijbehorende requirement hierbij is 1.4. Daarnaast wordt er deze sprint tijd besteed aan het deployen naar de testomgeving.
>
> – Sprint 4 Retrospective (planning Sprint 5)

| *Activiteit* | *Datum* |
|---|---|
| Code Review Sprint #4 | 11-4, 12-4 |
| Sprint #4 Retrospective | 11-4, 12-4 |
| Updaten Design (o.a. n.a.v. de code review) | 12-4 |
| Programmeren & testen ziekteverzuim-widget | 12-4, 13-4, 14-4 |
| Deployen naar testomgeving* | 18-4, 19-4, 20-4, 21-4 |
| Werken aan afstudeerverslag | 15-4, 22-4 |

*Tabel 11: Planning sprint 5 volgens de sprint 4 retrospective.*

De eerste activiteit van deze sprint was een code review (§5.1). Omdat de code review veel aanpassingen met zich mee bracht is hierna het design aangepast (§5.2); daarnaast heb ik ook de feedback van mijn begeleider binnengekregen over de extra tabel in de database. Na het design gemaakt te hebben ben ik de Havik-Core kant gaan programmeren van de ziekteverzuim-widget (§5.3). Hierna heb ik de functionaliteit ingebouwd dat er op staven in de grafiek geklikt kan worden (§5.4). Vervolgens zijn uiteraard de tests gemaakt om de nieuwe widget te testen (§5.5). Om van de alert boxes af te komen in de widget heb ik met mijn begeleider naar routing gekeken (§5.6). Gedurende de sprint heb ik ook gewerkt aan het deployen van Havik naar de test-omgeving (§5.7). De laatste activiteit van deze sprint is opnieuw de demo & code review (§5.8). Uiteraard volgt mijn evaluatie over deze sprint hierachter (§5.9).

## 5.1. Code Review Sprint #4

Tijdens de code review hebben we (mijn begeleider en ik) de code (uiteraard) aangepast. Meerdere methodes zijn kleiner gemaakt als resultaat van het splitsen van functionaliteit. Een methode is nu verantwoordelijk/bevat maar één functionaliteit (“separation of concerns”). In de meest extreme gevallen is één methode opgesplitst naar 4-5 methodes. Ook is er functionaliteit die regelmatig herhaald werd verplaatst naar een nieuwe klasse. Opnieuw komt “separation of concerns” hierbij kijken; de code is hierdoor ook overzichtelijker geworden.

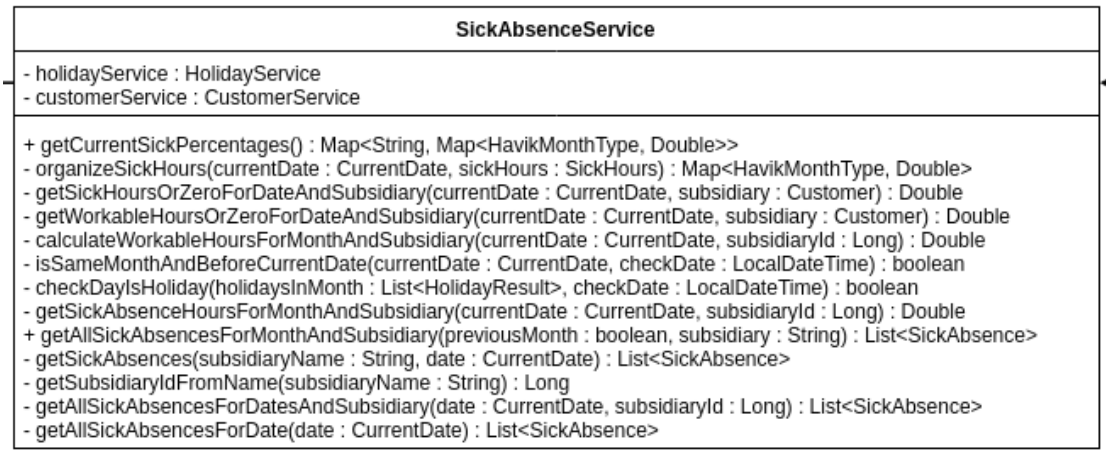
We hebben dit over twee sessies verspreid; tijdens de eerste sessie hebben we de methode “getCurrentSickPercentages()” van de klasse “SickAbsenceService” in Postduif behandeld. Na deze verbeterd te hebben heb ik zelf “getAllSickAbsencesForMonthAndSubsidiary()” zo ver mogelijk proberen te verbeteren. Vervolgens hebben we opnieuw met z’n tweeën naar deze methode gekeken om deze nog verder te verbeteren.

Deze veranderingen zijn terug te vinden in het klassendiagram (§5.2).

## 5.2. Design

Na de code review was het design verouderd. Zoals hierboven beschreven hebben we de klasse “SickAbsenceService” in Postduif vooral veranderd. Hieronder valt zowel de oude als de nieuwe versie van “SickAbsenceService” te zien.


*Illustratie 38: Oude versie SickAbsenceService.*



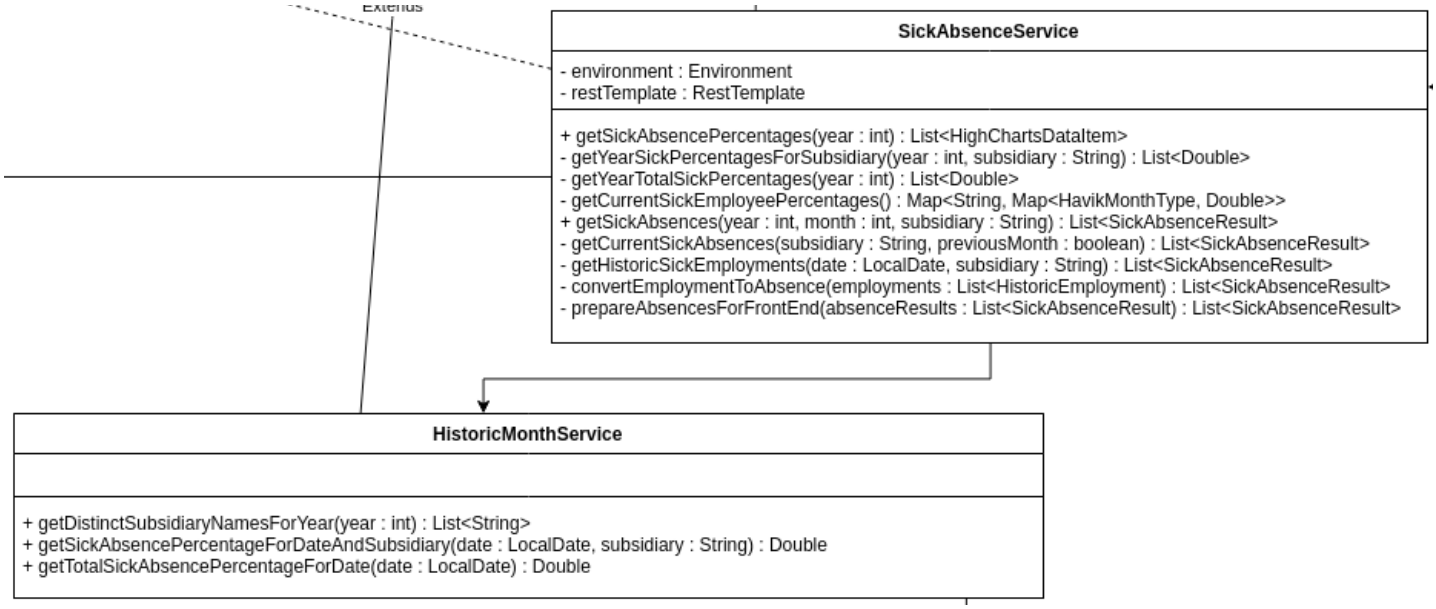

*Illustratie 39: Nieuwe versie SickAbsenceService*

Duidelijk is dat er veel nieuwe methodes zijn bijgekomen.

Naast het updaten van het klassendiagram is deze ook geupdate voor de functionaliteit in Havik-Core; feedback van mijn begeleider was eindelijk binnen. Hij stemde ook in met een nieuwe tabel in de database welke data per maand op slaat. Hiervoor moest er een nieuwe repository komen in Havik-Core; deze valt hiernaast te zien.


*Illustratie 40: Nieuwe repository.*

Ook moest hiervoor een nieuwe service bijgemaakt worden (“HistoricMonthService”) en moest “SickAbsenceService” (in Havik-Core) hierbij aangepast worden. Postduif had verder geen aanpassingen nodig; de front-end van Havik ook niet.




*Illustratie 41: Service klassen in Havik-Core.*

## 5.3. Programmeren: Ziekteverzuim Havik-Core percentages

Om de ziektepercentages uit Havik te laten zien heb ik allereerst de database uitgebreid; het design hiervoor had ik al in de vorige sprint gemaakt. Alleen de feedback van mijn opdrachtgever was nog nodig om te bepalen of dit een goede keuze was. Nu deze binnen is heb ik deze meteen geïmplementeerd.
Vervolgens heb ik de code geïmplementeerd zoals deze ook in het klassendiagram staat om de ziektepercentages op te halen. Deze methodes halen alleen de percentages op om deze vervolgens door te geven aan de UI.

| historic_month_info | |
|---|---|
| PK | **id : bigint** |
| | version : bigint |
| | date : date |
| | company : varchar(50) |
| | employee_count : int |
| | sick_percentage : decimal |

*Illustratie 42: Nieuwe tabel in Havik's database.*

## 5.4. Programmeren: Ziekteverzuim medewerkers

De volgende stap was om functionaliteit toe te voegen dat er op de staven geklikt kan worden. Wanneer er op een staaf geklikt wordt moeten de medewerkers die ziek waren worden laten zien met de hoeveelheid uren dat zij in die maand ziek waren. Het ophalen van deze informatie was de eerste stap hiervoor; hiervoor is geïmplementeerd wat er in het klassendiagram beschreven staat. Vervolgens het laten zien heb ik op dezelfde manier geïmplementeerd als de medewerkersaantallen, met de toevoeging dat het ziekteverzuim naast de naam kwam te staan; dit gebeurt met een alert box van de browser.

Nu de in-memory database van Havik geüpdatet is kon ook de PostgreSQL database uitgebreid worden met de nieuwe tabel. Na deze toegevoegd te hebben en hier data aan toegevoegd te hebben heb ik de applicatie uiteraard hiermee gedraaid; deze werkt nog precies naar behoren.

De volgende requirement die ik wilde behandelen was het klikken op deze gegevens; wanneer er op een medewerker geklikt wordt dat er doorgelinked wordt naar Mus waar de gebruiker vervolgens uitkomt op de pagina van deze medewerker. Hier liep ik tegen een probleem aan; alert boxes ondersteunen vrijwel geen markup (waaronder links/hrefs). Om dit op te lossen zal wat er nu in een alert box wordt laten zien op een compleet nieuwe pagina moeten komen. Omdat ik geen ervaring heb met het maken van multi-page websites met AngularJS heb ik hiervoor een knowledge session ingepland met mijn begeleider, om dit te leren. Omdat dit niet meteen hierna kon ben ik ondertussen aan een ander onderwerp begonnen: testen.

## 5.5. Tests

Van alle methodes die ik voor deze widget gemaakt heb ik unittests gemaakt. Dit uiteraard in Havik-UI, Havik-Core & in Postduif.

Tijdens het maken van deze tests bleek dat er inderdaad nog een bug in SickAbsenceService in Postduif zat. Door een naamgeving, waar ik achteraf gezien niet goed over nagedacht had, werd er een verkeerde check uitgevoerd.

```
private boolean isSameMonthAndBeforeCurrentDate(CurrentDate currentDate, LocalDateTime checkDate) {
    return checkDate.getMonthValue() == currentDate.getMonth() &&
            !(checkDate.getMonthValue() == /*currentDate.getMonth()*/ LocalDate.now().getMonthValue() &&
                    checkDate.getDayOfMonth() == /*currentDate.getDay()*/ LocalDate.now().getDayOfMonth());
}
```

*Illustratie 43: De methode met de bug.*

Het commentaar in de bovenstaande illustratie is wat de bug veroorzaakte; de code die erachter staat is de correcte code. Het valt te zien dat de naamgeving hier voor verwarring zorgt.

Nadat de unittests allemaal slaagden heb ik de data in de database ook zelf nog handmatig uitgerekend. Na bovenstaande verandering klopte wat ik berekend had met wat er in de GUI weergegeven werd.

## 5.6. Routing

In de session met mijn begeleider hebben we besproken wat routing inhoud en hoe dit werkt met AngularJS. Tijdens deze sessie hebben we gezamenlijk de routing gemaakt van de medewerkersaantallen widget. Nu ik weet hoe routing werkt heb ik vervolgens zelf de routing voor de ziekteverzuim widget kunnen maken.

Het oorspronkelijke probleem waar ik op stuitte was dat ik geen links kon toevoegen aan alert boxes voor het klikken op medewerkers. Nu deze informatie op een nieuwe pagina staat kan dit wel. Ik ben vervolgens in Mus gaan kijken naar welke URL's er gelinkt moet worden bij het klikken op een medewerker. Hier stuitte ik op een nieuw probleem. In de database van Havik, en in de code van het systeem tot dusverre, wordt gebruik gemaakt van het EmployeeId. In de URL's van de medewerkers wordt deze echter niet gebruikt; hier wordt het PersonId gebruikt. Na navraag gedaan te hebben bij de makers van Mus is het inderdaad niet mogelijk om via het EmployeeId bij de medewerkers uit te komen; er moet echt het PersonId gebruikt worden. Om dit aan te passen binnen Havik zou er veel moeten veranderen; de database moet hierbij uitgebreid/veranderd worden & de code moet ook dit nieuwe id gaan gebruiken. Omdat ik hier redelijk veel tijd aan zou moeten besteden wil ik dit eerst bespreken met mijn begeleider op de code review.

```
.state('employee-count', {
url: 'employee-count/:subsidiary/:year/:month',
templateUrl: 'views/employee-count.html',
controller: 'EmployeeCountCtrl as employeeCountCtrl',
resolve: {
    data: function($stateParams, EmployeeService){
        return EmployeeService.getEmployees(
            $stateParams.year,
            $stateParams.month,
            $stateParams.subsidiary);
        }
    }
});
```

*Illustratie 44: Routing voor de medewerkersaantallen widget.*

## 5.7. Deployment

Deze sprint zou er ook gewerkt worden naar het deployen naar de test-omgeving. Omdat dit mijn eerste deployment bij 42 zou zijn ben ik eerst langs systeembeheer gegaan om na te gaan wat hier voor nodig is.

Het aanmaken van de database tabellen moet via een .xml bestand gebeuren (zoals bij Postduif, zie §2.5.2) i.p.v. de .sql bestanden die nu gebruikt worden. Daarnaast moet er een proces plaatsvinden voor het vastleggen van de versies van verschillende libraries die gebruikt worden. Dit is echter een eenmalig proces. Aangezien Havik al eens eerder gedeployed is naar de test-omgeving wilde ik eerst overleggen met de vorige maker wat hiervan al gedaan is en wat hiervan nog gedaan moet worden. Op de dag dat deze afspraak ingepland stond meldde deze zich echter ziek; dit zal dus verplaatst moeten worden naar een later tijdstip.

De .xml bestanden voor de database heb ik wel gemaakt.

## 5.8. Demo & Code Review

Ook deze sprint is afgesloten met een demo & een code review.

### 5.8.1. Demo

Tijdens de demo is weer behandeld wat er deze sprint gebeurt is. De ziekteverzuim-widget is nu af. Het deployment is nu helaas niet gelukt (i.v.m. ziekte van een collega). Het deployen gaat nu zo vroeg mogelijk in de volgende sprint gebeuren.

Er zijn nu nog “maar” drie sprints tot het einde van het project. We hebben hier ook een globale planning voor gemaakt: de komende ($6^e$) sprint ga ik nog een widget maken, de $7^e$ sprint ga ik alles wat er nu ligt beter maken (refactoren, UI mooier maken, bug fixes, etc) en de laatste ($8^e$) sprint ga ik besteden aan mijn afstudeerverslag.

Ook hebben we afgesproken dat de requirements nu klaar zijn om geprioriteerd te worden; ook dit gaan we zo vroeg mogelijk in de volgende sprint doen. Tijdens de prioritering spreken we af welke widget er de komende sprint gemaakt gaat worden.

De opdrachtgever heeft zijn mening over het gehele project tot dusverre nog gegeven: hij is zeer tevreden. Hij benadrukte opnieuw dat hij liever iets minder widgets ziet die goed werken dan een groter maar niet goed werkend programma. Omdat dat is wat er nu ligt (twee widgets die goed werken) is hij zeer tevreden.

Uiteraard zijn alle afspraken die gemaakt zijn tijdens de demo in overleg gegaan met de opdrachtgever & begeleider.

### 5.8.2. Code Review

Tijdens de code review hebben we zoals gewoonlijk mijn gemaakte code bekeken & verbeterd. We hebben vooral naar de routing gekeken. Hierin zijn een aantal verbeteringen meteen geïmplementeerd. Ook hebben we gekeken naar het probleem wat in §5.6 benoemd staat; de link naar Mus kan niet gemaakt worden met de employeeIds, maar moet met de personIds. We hebben besproken wat ervoor nodig is om dit te veranderen in Havik, dat zijnde:

- de “historic_employees” tabel in de database;
- het model die bij deze tabel hoort in Havik-Core;
- redelijk veel code in Postduif; overal waar het employeeId opgehaald & doorgegeven wordt moet ook het personId opgehaald & doorgegeven worden;
    - dit zal veranderingen vereisen in models, services & repositories.

We hebben afgesproken dat dit, indien mogelijk, in de 7e sprint behandeld wordt; de komende (6e) sprint ga ik mij focussen op de nieuwe widget.

## 5.9. Evaluatie

Globaal genomen ging de sprint naar mijn idee goed. De widget is af en de eerste stappen naar deployment zijn gemaakt. Natuurlijk is het jammer dat het deployen nog niet gedaan is, maar dit is nog te overzien. Het belangrijkste is dat de widget af is en dat de opdrachtgever tevreden is.

Na de demo had ik naar de mening van mijn begeleider gevraagd over het gehele proces/project tot dusverre. Hij is ook tevreden. Hij gaf nog een verdere uitleg: er lag (voordat ik begon) al een Havik. Een Havik die is opgebouwd met een bepaalde filosofie; alle widgets halen data op die op dit moment van belang is; geen historische data dus. Wat ik heb gemaakt is een filosofie die hier tegenover staat: er moet juist historische data opgehaald worden. Dat het project tot nu toe goed verloopt terwijl dit zo tegenover elkaar staat is ook een punt waar hij tevreden over is. Daarnaast loopt het project volgens planning, wat er gemaakt is is van kwaliteit, etc.; verschillende punten waar hij tevreden naar kijkt.

Ook is er natuurlijk een leermoment in deze sprint te vinden; er werd een collega ziek op de dag dat ik wat met hem wilde overleggen. Er was echter geen tijd meer in de sprint om dit overleg later in de sprint te laten plaatsvinden. Het leerpunt is dan ook om activiteiten die afhankelijk zijn van andere mensen eerder te laten plaatsvinden; wordt er dan iemand ziek kan het later opgepakt worden.

Ook is mij tijdens deze sprint een baan aangeboden bij 42; dit is voor mij nog een bevestiging dat het project goed verloopt & dat 42 tevreden is over mij & mijn werkzaamheden.

# 6. Sprint 6: Klant Statistieken



> Het doel van de zesde sprint is om een nieuwe widget te maken. In overleg met de opdrachtgever is besloten om requirement 1.8 (volgens het requirements document V5.0 & V6.0) te maken; een top 10 van klanten welke de meeste uren toegewezen hebben gekregen van zowel de afgelopen 2 als 12 maanden (2 tabellen).
>
> Tijdens deze sprint vallen 3 vrije dagen; woensdag 27-4 (Koningsdag), donderdag 5-5 (Hemelvaart & Bevrijdingsdag), vrijdag 6-5.
>
> – Sprint 5 Retrospective (Planning Sprint 6)

Met het “overleg met de opdrachtgever” in de bovenstaande quote wordt het prioriteringsgesprek bedoeld.

| *Activiteit* | *Datum* |
|---|---|
| Requirements nieuwe sprint + prioriteren | 25-4, 26-4 |
| Sprint #4 Retrospective | 26-4 |
| Design | 26-4, 28-4 |
| Programmeren nieuwe widget, inclusief tests | 28-4, 2-5, 3-5, 4-5 |
| Werken aan afstudeerverslag | 29-4, 4-5 |

*Tabel 12: Planning sprint 6 volgens de sprint 5 retrospective.*

Mijn eerste activiteit deze sprint was het prioriteren van requirements (§6.1); hierin is ook bepaald welke widget er gemaakt gaat worden deze sprint. Vervolgens heb ik Havik gedeployed naar de testomgeving (§6.2). Hierna ben ik weer verder gegaan met de widget van deze sprint: allereerst het design maken (§6.3). Na het design ben ik dit uiteraard gaan programmeren (§6.4). De sprint is zoals gewoonlijk afgesloten met een demo & code review (§6.5). Als laatste volgt mijn evaluatie over deze sprint (§6.6).

## 6.1. Requirements

Als voorbereiding op de prioritering heb ik storypoints toegewezen aan alle requirements. Hierdoor kan de opdrachtgever een indicatie krijgen van hoe veel tijd alle requirements nodig gaan hebben om deze te implementeren. Mijn begeleider vertelde dat het toewijzen van storypoints normaal met het projectteam gedaan wordt. Aangezien ik deze luxe niet heb hebben we gekozen dat ik dit zelf ging doen.

In het prioriteringsgesprek hebben we vervolgens bepaald welke widget er deze sprint gemaakt gaat worden: de widget die bij de requirement hieronder hoort.

| # | Benodigd | Requirement | Oorsprong | Storypoints | Prioriteit |
|---|---|---|---|---|---|
| 1.8. | 1.2. | Havik moet op de locatie van één widget twee tabellen laten zien. | Prioritering | 8 | 1 |
| | | In beide tabellen moet de top 10 aan klanten worden laten zien met hun:<br>• klantnaam;<br>• hoeveelheid uren. | Interview #1 Opdrachtgever<br>Interview #3 Opdrachtgever<br>Prioritering | | |
| | | De eerste tabel laat deze gegevens zien over de afgelopen 12 maanden. De tweede tabel laat deze gegevens zien over de afgelopen 2 maanden. | Prioritering | | |
| | | De tabellen moeten gesorteerd worden op de hoeveelheid uren. | Prioritering | | |
| | | Het moet mogelijk zijn om werkmaatschappijen aan en uit te zetten. Hierbij worden uren bij een uitgezette werkmaatschappij niet meegenomen in de tabellen. | Interview #1 Opdrachtgever | | |
| | | Wanneer er geklikt wordt op de naam van een klant moet er een nieuw scherm openen waarin via een deeplink naar Kauw de gegevens van die klant wordt laten zien. | Prioritering | | |
| | | Het moet mogelijk zijn om de huidige bekeken maand (vanaf waar er 2/12 maanden worden teruggeteld) te veranderen. | Prioritering | | |
| | | Hiervoor moeten het jaar en de maand apart van elkaar veranderd kunnen worden. | Prioritering | | |

*Tabel 13: Requirement 1.8.*

Naast de widget voor deze sprint hebben we ook de andere requirements geprioriteerd. Er zijn 3 requirements geprioriteerd; verder gaand vanaf daar vond de opdrachtgever niet nuttig. De aanpak wanneer er verder gewerkt wordt met Havik (na dit project) krijgt dezelfde aanpak als dit project: per sprint wordt er bekeken wat er gemaakt gaat worden. Met die insteek hoeven ook niet alle requirements geprioriteerd te worden, alleen de belangrijkste zodat er wel meteen begonnen kan worden i.p.v. dat dit requirements proces opnieuw hieraan moet voorafgaan.

## 6.2. Deployment

Nu mijn collega weer beter was hebben we meteen een meeting ingeschoten over het deployen van Havik.

### 6.2.1. Meeting

Van het proces wat in §5.7 beschreven staat hoefde er niet meer te gebeuren; alles hiervan was al tijdens de vorige deployment (voor dit project) gedaan. Het enige wat er nog gedaan moest worden was het committen & pushen naar een nieuwe branch op git, waarna deze collega (met wie deze meeting was) dit nog zou reviewen, vervolgens mergen en dan kon het live gezet worden. Voor het live zetten op zich heb ik instructies gekregen over hoe dit moest.

Er kwamen nog enkele issues uit de review van mijn collega; na deze opgelost te hebben kon Havik eindelijk gedeployed worden.

### 6.2.2. Deployen naar testomgeving

Het deployen zelf is niet zonder slag of stoot verlopen. Verschillende factoren zorgden ervoor dat het deployen lastigen & langer verliepen dan normaal:

- Het was mijn eerste deployment; ik had hier geen ervaring mee;
- Het was de eerste Havik deployment met een database;
- Er zat een fout in de indeling van het Java-project.

Na een langdurige sessie met zowel iemand van systeembeheer als mijn begeleider is het uiteindelijk gelukt om Havik te deployen naar te testomgeving.

## 6.3. Design

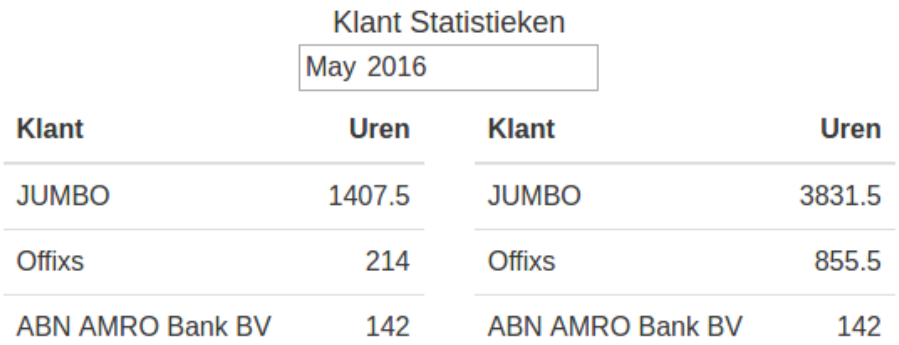
Het maken van het nieuwe design is opnieuw op dezelfde manier gebeurt als de voorgaande keren: allereerst deze updaten naar hoe de code nu is en vervolgens de nieuwe widget hier aan toevoegen.

## 6.4. Programmeren

Na het maken van het design ben ik dit gaan programmeren. Uiteraard komt dit op een andere branch op Git dan de branch die nu op de testomgeving staat.

### 6.4.1. Front-end

De nieuwe widget vereiste iets nieuws. De voorgaande twee widgets waren twee grafieken waarvan de HTML code al bestond. Deze nieuwe widget bestaat uit 2 naast elkaar geplaatste tabellen; deze HTML bestond nog niet. Voor deze widget heb ik voor het eerst tijdens dit project dus ook de HTML & CSS code van Havik-UI moeten aanpassen. Ook de Javascript code die hier bij hoort resulteert nu niet in een Highcharts object maar in een standaard Javascript object.




*Illustratie 45: De nieuwe widget.*

### 6.4.2. Back-end

Op aanraden van mijn begeleider heb ik van Havik-Core nu een doorgeefluik gemaakt van Postduif naar Havik-UI. Hierdoor werkt deze widget nu met correcte data. Omdat het synchronisatie proces tussen de Havik database en de Postduif database nog niet bestaat zou deze widget anders compleet niet werken (van de andere widgets werken de afgelopen twee maanden nog zonder synchronisatie). De bedoeling is dat de widget de afgelopen 2 maanden uit Postduif haalt en alle andere data uit de database van Havik. Maar de linker tabel laat de afgelopen twee maanden zien en de rechter de afgelopen twaalf maanden. Met alleen data van de afgelopen twee maanden beschikbaar zou de functionaliteit van deze widget ernstig afnemen, tot het punt dat deze niet meer nuttig is. Door de data voorlopig compleet uit Postduif te halen is deze iets trager, maar wel functioneel.

Postduif heeft nieuwe logica gekregen voor het ophalen van de top 10 aan klanten. De illustratie hierboven laat de nieuwe query zien die toegevoegd is aan de repository van klanten. Andere nieuwe logica in Postduif bestaat uit het converteren van deze return waardes naar een nieuw model object. Omdat de query zelf al de grootste logica bevat hoeft Postduif zelf weinig meer te doen voordat dit doorgegeven kan worden aan Havik.

```
@Query("SELECT SUM(r.amountOfHours) AS totalHours," +
        " c.clientName," +
        " c.id as clientId" +
        " FROM Registration r" +
        " JOIN r.assignment a" +
        " JOIN a.project p" +
        " JOIN p.customer c" +
        " WHERE c.subsidiary = FALSE" +
        " AND r.dateRegistration BETWEEN :firstDate AND :lastDate" +
        " GROUP BY c.id" +
        " ORDER BY totalHours DESC")
List<Object[]> findTopTenOfCustomersForPeriod(@Param("firstDate") LocalDateTime firstDate,
                                              @Param("lastDate") LocalDateTime lastDate,
                                              Pageable pageable);
```

*Illustratie 46: Nieuwe query in Postduif.*

## 6.5. Demo & Code Review

Zoals gewoonlijk is de sprint afgesloten met een code review & een demo.

### 6.5.1. Code Review

Het grootste punt wat in de code review voorbij is gekomen is het gebruik van <input type=”month”>. Dit is een nieuwe input type van HTML5. Deze is echter nog niet geïmplementeerd in de laatste versies van Firefox, Internet Explorer & Safari. Omdat Firefox & Safari veel gebruikt worden binnen 42 is dit een probleem. Tijdens de komende sprint ga ik deze input hierom vervangen door een Angular bootstrap input.

### 6.5.2. Demo

De sprintdoelen zijn behaald en de opdrachtgever is opnieuw/nog steeds tevreden. Er is afgesproken om de komende sprint zo veel mogelijk te verbeteren. Er komt geen nieuwe widget; in plaats hiervan wordt de style rechtgetrokken van de widgets en wordt lelijke code refactored.

## 6.6. Evaluatie

Het prioriteren van de requirements heeft minder opgeleverd dan ik had gehoopt/verwacht. Alleen de prioritering van de top ~3 aan requirements is minder dan ik had verwacht te doen. Deze manier van prioriteren is wel logischer, tijdens dit project is het ook ontzettend handig geweest om per sprint te bekijken wat er gedaan ging worden.

Er komt binnenkort een tech night aan; een avond vol met nieuwe technieken waar over na gedacht gaat worden om te gebruiken binnen 42. Een soort KTM in het klein. Ook hiervoor is mij weer gevraagd om een presentatie te geven; ditmaal over interactie met je IDE.

Het deployen was een erg lastig en lang proces. Wel ben ik blij dat ik deze ervaring nu ook eens gedaan heb; weten hoe er gedeployed wordt is goede kennis om te bezitten. De eerste keer Havik zien vanaf een andere URL dan “localhost” was ook leuk om te zien; daar doe ik het toch allemaal voor.

# 7. Sprint 7: Beautification

| Opstarten | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 | Sprint 7 | Sprint 8 |
|---|---|---|---|---|---|---|---|---|

> De komende sprint staat in het teken van het gelijktrekken van alle widgets. De layout, de looks & natuurlijk de code zelf worden deze sprint verbeterd. De bedoeling hiervan is dat de gebruiker hierdoor een betere ervaring heeft en toekomstige programmeurs sneller en beter met de code om kunnen gaan. Oplossen van bestaande bugs (zoals benoemd in de evaluatie) wordt hier ook in meegenomen.
>
> – Sprint 6 Retrospective (planning Sprint 7)

| *Activiteit* | *Datum* |
|---|---|
| Tests klant statistieken-widget | 09-05 |
| Sprint retrospective | 10-05 |
| Verbeteren Havik | 10-05 – 12-05, 17-05 – 19-05 |
| Presentatie voorbereiden tech-night | 10-05 – 12-05 |
| Afstudeerverslag | 13-05, 20-05 |

*Tabel 14: Planning sprint 7 volgens de sprint 6 retrospective.*

Deze sprint ben ik begonnen met het maken van de tests voor de laatste widget (§7.1). Hierna ben ik Havik gaan verbeteren (§7.2). De sprint is uiteraard weer afgesloten met een demo & code review (§7.3). Als laatste volgt opnieuw mijn evaluatie (§7.4).

## 7.1. Klant Statistieken Tests

Omdat ik tijdens de vorige sprint nog niet alle tests afgekregen heb kunnen krijgen heb ik daar gedurende deze sprint ook tijd aan moeten besteden. Dit is zonder problemen gelukt.

## 7.2. Beautification van Havik

Tijdens deze sprint heb ik de volgende punten verbeterd aan Havik:

- Alfabetische volgorde widgets;
- Alfabetische volgorde op detail-schermen;
- Correcte header voor “Totaal” op detail-schermen
- Tabel widget titels dezelfde grote als grafiek widgets;
- Numerieke gegevens in tabellen outlinen naar rechts;
- Titels toevoegen bij klant statistieken-widget;
- Month-picker vervangen;
- Autorisatie;
- School
- Optimalisatie factuur overzichten

– Sprint 7 demo

Het laatste punt in het bijzonder vereist meer uitleg. Het ophalen van factuur overzichten (twee widgets; waren al gemaakt voor ik aan dit project begon) kostte veel tijd. Het ophalen hiervan kostte op de testomgeving ~drie minuten. Dit was het proces wat het meeste tijd nodig had om Havik te kunnen laden; daarom ben ik deze gaan verbeteren. Als ik mijn lokale Postduif gebruik liet maken van de database van de testomgeving en mijn lokale Havik probeerde op te starten gaf deze na 4 minuten aan dat deze geen reactie terugkreeg van de server bij de requests van de factuur overzichten. Het ophalen van data kostte teveel tijd.

Wat Havik bij deze widgets deed was alle facturen ophalen en hier vervolgens een aantal filters op loslaten. Dit heb ik veranderd door deze code naar Postduif te verplaatsen en de filters van te voren al bij de query te gebruiken. Hierdoor worden niet meer alle facturen opgehaald, maar alleen de gebruikte facturen. Na deze veranderingen kostte het ophalen van de data (lokaal met de test-database) ~20 seconden; een grote verbetering tegenover 4+ minuten met een error.

Deze versie van Havik is ook naar de test omgeving gedeployed. Ditmaal was dit een simpeler proces dan de vorige keer (§6.2), omdat ik nu wist wat er gedaan moest worden. Het ophalen van alle data voor de widgets kostte bij de vorige versie ~drie minuten; met deze versie ~11 seconden.

## 7.3. Demo & Code Review

Zoals gebruikelijk is de sprint afgesloten met een demo & code review.

### 7.3.1. Demo

Tijdens de demo zijn er geen nieuwe afspraken gemaakt. Wat er nu nog gaat gebeuren in het project was al bekend:

- De komende sprint ga ik besteden aan mijn afstudeerdossier;
- In de laatste week ga ik een laatste presentatie geven (deze staat al ingepland);
- Opleverdocumentatie/de readme voor in Havik-Core moet nog gemaakt worden.

De opdrachtgever was opnieuw tevreden met hoe alles er nu uit ziet.

### 7.3.2. Code Review

Omdat dit de laatste code review van het project is hebben we deze anders aangepakt dan normaal. Als er nu nog grote punten uit zouden komen zou ik niet meer de tijd hebben om deze te verbeteren; vandaar dat we dit anders hebben gedaan. We hebben deze keer het .pom bestand veranderd naar de nieuwe standaard van 42. Hierdoor kan iemand die Havik van Git haalt deze direct gebruiken, zonder eerst nog (Spring-)profielen of databases te moeten activeren.

## 7.4. Evaluatie

Er is veel code veranderd deze sprint (ten goede). Havik is nu sneller en ziet er beter uit. Ik ben er tevreden over hoe deze nu loopt & er uit ziet. Het sprint-doel is dus goed gelukt in mijn ogen.

De verbetering in snelheid van Havik (van ~3 minuten naar ~11 seconden) is een resultaat waar ik ontzettend blij mee ben. Dit verbeterd de bruikbaarheid van Havik ontzettend; gebruikers hoeven niet eerst 3 minuten te wachten voordat ze Havik kunnen gebruiken.

# 8. Sprint 8: Afstudeerdossier

| Opstarten | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5 | Sprint 6 | Sprint 7 | Sprint 8 |
|---|---|---|---|---|---|---|---|---|

Deze sprint is toegewijd aan het afmaken van mijn afstudeerdossier zoals eerder afgesproken (met de opdrachtgever & begeleider). Er wordt hierin niet meer verder gewerkt aan Havik.

Laatste activiteiten die nog gedaan worden voor het project:

- De opleverdocumentatie wordt gemaakt;
- Er wordt een eindpresentatie gegeven aan alle medewerkers die Havik uiteindelijk gaan gebruiken.

Deze punten worden echter niet meer in dit verslag besproken; dit komt omdat deze punten na het opleveren van het afstudeerdossier gebeuren.

# Deel 3

## Evaluatie en referenties

Het derde deel van mijn verslag beschrijft mijn evaluatie over de uiteindelijk opgeleverde producten en het gehele proces van het project. Hierna staan de verschillende beroepstaken die aan bod zijn gekomen gedurende dit project benoemd met de motivatie waarom deze naar mijn idee wel of niet behaald zijn. Als laatste volgt het referentiemateriaal; een opsomming van de literatuurobjecten die gebruikt zijn gedurende dit project en de tabellen & illustraties die in dit document staan.

# Evaluatie

**Java**
Aan het begin van het project had ik de verwachting dat het werken met Java een probleem kon zijn omdat ik van de .NET kant kom. Achteraf gezegd is dit nooit een probleem geweest. De .net achtergrond heeft me zeer goed geholpen om Java snel op te pakken, en heeft zelfs een bijdrage geleverd binnen 42. (Tijdens de 5e sprint kwamen we erachter dat “//region”, een IDE comment van Visual Studio, ook werkt in IntelliJ IDEA.)

**Requirements**
Het requirements proces is wat slordig begonnen. Kijkend naar de kwaliteit van de requirements in het begin en halverwege is deze ontzettend verbeterd. Hier valt een goede groei in te zien.

Het prioriteren van requirements heb ik achteraf gezien te veel op gefocust. Halverwege het proces was dit niet nodig; en aan het einde was het ook maar een klein puntje. Omdat er per sprint bekeken wordt wat er gedaan gaat worden is een prioritering niet nodig.

**KTM**
De presentatie bij de KTM was zeer interessant. Een presentatie geven voor het gehele bedrijf was een ervaring die ik niet had verwacht toen ik aan dit project begon. Een leuke ervaring, maar ook zeer zeker leerzaam; en ervaring opdoen met presentaties geven voor grote groepen mensen kan ook nooit kwaad.

**Fixed-budget project**
Het was vreemd om per sprint te bekijken wat er gedaan ging worden. Op school heb ik vooral geleerd/gewerkt met “over ~8 weken ligt er ‘dit’”; van te voren vastgesteld dus (anders gezegd: fixed-price-projects). Het per sprint bekijken was daarom toch wel nieuw voor me; en ik vond het zeker wel prettig werken. Het is hiermee makkelijker om je aan te passen aan wat de opdrachtgever wilt en om een product op te leveren wat meer naar zijn wensen is.

**Methoden & Technieken**
Bij hoofdstuk IV. Methoden en Technieken staan verschillende technieken die langsgekomen zijn in dit project; deze waren vooraf opgesteld. Achteraf gesproken missen er een klein aantal:

- HSQLDB;
- PostgreSQL.

**Havik**
Ik ben tevreden met wat er opgeleverd gaat worden. Havik ziet er goed uit en is functioneel ook goed. Er zijn nog dingen die gedaan kunnen worden, maar dat zal altijd zo blijven (zoals bij elk software-product). Al met al ben ik tevreden met het proces & het product van dit project.

# Beroepstaken

Dit zijn de beroepstaken die in het originele afstudeerplan besproken zijn. In de volgende tabel wordt besproken waarom deze wel of niet behaald zijn:

| *Beroepstaak* | *Behaald (Y/N)* | *Motivatie* |
|---|---|---|
| 1.4. Uitvoeren analyse door definitie van requirements | Y | In het begin waren de requirements nog van mindere kwaliteit. Sinds het herschrijven in sprint 3 zijn de requirements van veel hogere kwaliteit; hoog genoeg om het halen van deze beroepstaak te kunnen rechtvaardigen. |
| 2.1. Opstellen gegevensmodel voor database | Y | Tijdens het gehele project heb ik meerdere malen een design voor de nieuwe database van Havik moeten ontwerpen/uitbreiden/verbeteren. |
| 2.2. Ontwerpen, bouwen en bevragen van een database | Y | De database van Havik heb ik zowel moeten ontwerpen, bouwen & bevragen. Hiernaast komt de database van Postduif bij die ik ook heb moeten bevragen. Ook heb ik een view moeten maken voor de database van Postduif die van meerdere tabellen gebruik maakt. |
| 3.2. Ontwerpen systeemdeel | Y | Het grootste gedeelte van Havik heb ik ontworpen; zie de verschillende bijbehorende hoofdstukken & klassendiagrammen in dit document. |
| 3.3. Bouwen applicatie | Y | Het doel van het project was het maken van de Havik applicatie. Het bouwen hiervan was uiteraard één van de grootste onderdelen hiervan. |
| 3.4. Initiëren en plannen van het testproces | N.V.T. | In het originele afstudeerplan is deze beroepstaak benoemd. Binnen 42 wordt er echter alleen unittests gebruikt. Uitgebreide testplannen, etc. worden hier (vrijwel) niet gemaakt. Het zou voor mijn project ook niet logisch geweest zijn om van 42's standaard af te wijken. |
| 3.5. Uitvoeren van en rapporteren over het testproces | N.V.T. | Zie motivatie van beroepstaak 3.4. |

# Literatuurlijst

Highcharts: http://www.highcharts.com
Spring: http://spring.io
Stack overflow: http://stackoverflow.com

**Havik:** https://Havik-test.42.nl

**Beanmapper:**
GitHub: http://github.com/42BV/beanmapper
Website: http://beanmapper.io

**JavaScript Promises:**
Andyshora: http://andyshora.com/promises-angularjs-explained-as-cartoon.html
HTML5rocks: http://www.html5rocks.com/en/tutorials/es6/promises/
Spring: http://spring.io/understanding/javascript-promises

# Tabellen & Illustraties

## Tabellen

# Illustraties