Data Analysis Portal

Afstudeerverslag

Auteur	Tim van Egmond (13078550)
opleiding	Informatica aan de Haagse Hogeschool
Opdrachtgever	CityTraffic
Plaats	Alphen aan den Rijn
Datum	1 Juni 2017
Versie	1.0

Referaat

Tim van Egmond

Ontwerpen en bouwen van een data-analysesysteem voor reparatie van tel-data.

Alphen a/d Rijn, CityTraffic, 2017

Afstudeerverslag van Tim van Egmond, geschreven in het kader van het afstuderen bij de opleiding Informatica aan de academie voor ICT & Media aan de Haagse Hogeschool.

Het verslag behandelt het (ontwikkel)proces dat is doorlopen tijdens de afstudeerperiode van Tim van Egmond bij CityTraffic te Alphen aan den Rijn. Deze opdracht stond in het teken van het ontwikkelen van een data-analysesysteem dat bedoeld is om errors in tel-data vroegtijdig te detecteren en zo te kunnen repareren.

Descriptoren:

- Afstudeeropdracht
- Kanban
- C# (.NET Core)
- Asp.Net Core
- Akka.NET
- Angular 2
- Angular Flex
- Angular material
- Node.js/NPM
- Webpack
- Autofac
- AutoMapper
- Serilogger
- Entity Framework Core
- HTML 5/CSS 3 (SASS)
- JSON
- Web applicatie (SPA)
- Actor model

Voorwoord

Dit verslag is tot stand gekomen naar aanleiding van de afstudeeropdracht van Tim van Egmond van de opleiding Informatica aan de academie voor ICT & Media aan de Haagse Hogeschool.

Van de lezer wordt verwacht dat hij/zij alle kennis van de opleiding informatica beheerst. Sommige stukken tekst kunnen door de lezer als complex gezien worden. Het is dan ook te raden dat de lezer enige kennis bezit van de gebruikte frameworks.

Graag wil ik CityTraffic bedanken voor de gezellige tijd en de gelegenheid die zij mij hebben geboden, om een afstudeeropdracht uit te voeren. Daarnaast wil ik mijn twee begeleiders bedanken, mvr. in ’t Veld en mvr. van der Hoek, voor de begeleiding die ze mij tijdens dit proces hebben getoond.

Tim van Egmond
Alphen aan den Rijn, 2017

Inhoudsopgaven

1. Inleiding

Tegenwoordig bestaat er nergens nog privacy en zeker niet als je buiten over straat loopt. In steden en winkelcentra hangen talloze camera's die alles in de gaten houden, zo ook de passantenstromen.

Dit project staat in het teken van data reparatie en error detectie. CityTraffic is een bedrijf dat mensen telt met behulp van sensoren. Nu komt het regelmatig voor dat deze sensoren geen goede of helemaal geen tellingen verstuurd hebben. Hoewel dit project niet in het teken staat de sensoren te verbeteren, is het wel de bedoeling dat deze problemen zo snel mogelijk opgelost worden. In dit project zal een analyse systeem gemaakt worden. Dit systeem zal fouten in de tellingen opsporen en werknemers in staat stellen, d.m.v. een webapplicatie, de fouten te verbeteren.

Binnen het project hebben twee onderzoeken plaatsgevonden. Het eerste onderzoek is het selecteren van een SPA framework, waarmee de webapplicatie gemaakt is. In dit onderzoek zijn meerdere frameworks met elkaar vergeleken, totdat er één overbleef. Het tweede onderzoek ging over Akka.NET, een framework waarmee multithreading op een ongewone manier wordt aangepakt. Dit onderzoek heeft ertoe geleid dat Akka.NET in de backend van de webserver is gebruikt om de performance van het systeem te waarborgen.

Bij de implementatie ligt een grote nadruk op error detectie. Hoe kunnen de foute tellingen op een efficiënte en correcte manier gedetecteerd worden? Hoe kunnen deze fouten aan de gebruikers getoond worden? En, hoe kan een gebruiker vervolgens deze fouten zo snel mogelijk oplossen?

Na afloop van de opdracht is een werkende demo opgeleverd van het gemaakte product. Dit product zal door het bedrijf verder ontwikkeld worden. Zodra de ontwikkeling dusdanig ver is dat het in productie genomen kan worden, zal het een gedeelte van het huidige systeem vervangen.

2. CityTraffic

PFM Footfall Intelligence is een bedrijf dat zich specialiseert in het tellen en volgen van mensen. Onder PFM vallen verschillende partners, elk gespecialiseerd in een bepaalde doelstelling. Eén van deze partners was CityTraffic.

CityTraffic is in 2010 opgericht en is sinds 2015 volledig overgenomen door PFM Footfall Intelligence. Citytraffic is een onderzoekbureau op het gebied van passantentellingen. De afgelopen jaren is het bedrijf flink gegroeid en is inmiddels actief in meer dan 120 Nederlandse en Belgische steden. In deze steden hebben ze een netwerk van sensoren aangelegd, om daar volcontinu de passantenstromen in winkelstraten mee te meten.

Telsystemen op basis van camera metingen genereren tellingen in de vorm van CSV-bestanden. Deze bestanden hebben de totaal tellingen van een camera. De sensoren die deze tellingen realiseren variëren van beeld-, infrarood- tot laser camera's (vanaf nu footfall devices genoemd).

Naast deze sensoren wordt ook gebruik gemaakt van zogenaamde hotspots, ook bekend als wifiscanners. Deze hotspots scannen de wifi en bluetooth mac-adressen van passanten. Het voordeel dat wifiscanners hebben over footfall devices is dat het mac-adres van een passant constant blijft. Doordat het mac-adres constant blijft, kunnen meerdere berekeningen/ analyses over deze data gemaakt worden, dit is wat de service uniek maakt. Met deze berekeningen kunnen bijv. de unieke aantallen bepaald worden. Hoeveel procent van de bezoekers is voor het eerst gedetecteerd, of juist een terugkerende bezoeker. Ook kunnen er berekeningen over meerdere wifiscanners uitgevoerd worden. Met dit soort berekeningen kan bijv. de looproutes van passanten bepaald worden.

De data, die de sensoren genereren, wordt nu in een database verwerkt tot rapportages. Rapportages kunnen op verschillende manieren aan de klant geleverd worden, aan de hand van wat de klant wil. Over het algemeen gebeurt dit in de vorm van met de hand samengestelde pdf-bestanden, zodat het voor elke klant een unieke look heeft. Deze bestanden worden periodiek aangeleverd en bevatten bijv. berekende maand of kwartaal data. Daarnaast kunnen klanten gebruik maken van een online portal genaamd SmartTrace. Op deze portal kan een klant al de (berekende) verzamelde data live bekijken. Ook kunnen hierin de onderliggende verhoudingen van de verschillende sensoren live bekeken worden.

Dankzij de verschillende producten die geleverd kunnen worden, kunnen gemeentes betere inschattingen maken. Zo kunnen ze bijvoorbeeld zien wat voor impact een bouwproject heeft op het gedrag van mensen. Daarnaast stelt het gemeentes in staat in te zoomen op de shopping gewoontes van mensen.

2.1. Organisatie

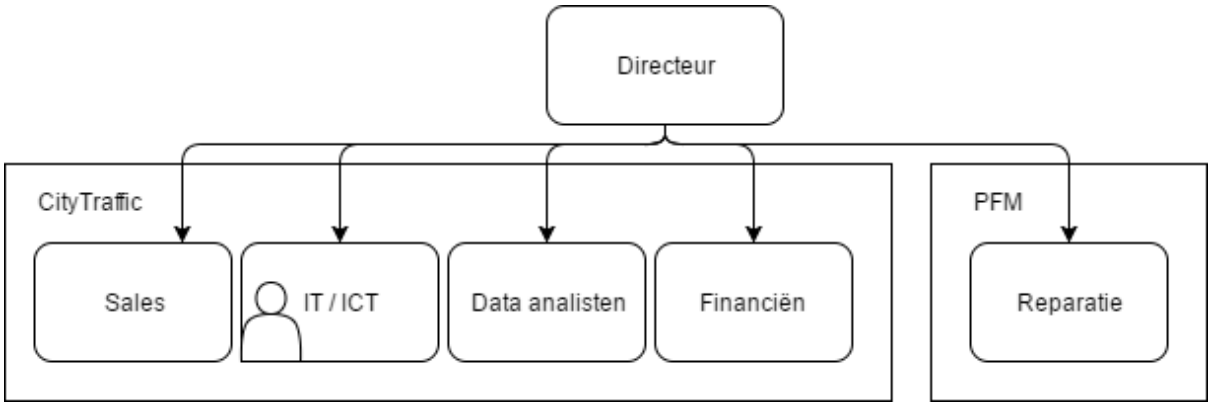
CityTraffic levert een compleet product aan haar klanten, zo is de IT-afdeling waar ik me bevind maar een klein gedeelte van de gehele organisatie. In figuur 1 is een organogram weergegeven, welke de hoofdafdelingen van het bedrijf weergeeft. In het figuur is te zien dat er één overkoepelende directeur is. Hij is verantwoordelijk voor alle bedrijven onder PFM Footfall Intelligence. Daarnaast is te zien dat CityTraffic bestaat uit meerdere afdelingen.

De sales afdeling van CityTraffic is officieel gevestigd in Amsterdam, dit is tevens het bezoekadres voor (potentiele) klanten. De overige werknemers van CityTraffic bevinden zich in Alphen aan den Rijn, het hoofdkantoor van PFM Footfall Intelligence. Vanwege het feit dat deze mensen zich ergens anders bevinden, heb ik ze niet leren kennen.

De data-analisten zijn, onder IT natuurlijk, de belangrijkste afdeling binnen het bedrijf. Deze werknemers zijn verantwoordelijk voor zowel het corrigeren van foutieve data, als het samenstellen van de rapportages. Deze taken worden door verschillende mensen uitgevoerd.

De reparatie en on site installatie van verschillende sensoren wordt nu uitgevoerd door een afdeling buiten CityTraffic. Deze dienst valt onder PFM en wordt door CityTraffic gebruikt voor haar klanten.

De afgelopen jaren is het bedrijf zich steeds meer op de IT en specifieker op automatisering gaan richten. Deze opdracht is een goede volgende stap in de automatisering van een proces.



Figuur 1: Organogram CityTraffic

2.2. Werkomgeving

De IT-afdeling bestaat uit vier vaste werknemers, met elk hun eigen rol binnen de afdeling. Bart, de CTO niet te verwarren met de CEO, heb ik maar even gezien. Het is zijn taak om projecten te begeleiden en nieuwe projecten op te zetten. Paktwis is verantwoordelijk voor de wifiscanner software, het OS zelf. Gerrit houd zich bezig met zowel softwareontwikkeling als systeembeheer. Chris, de bedrijfsmentor, is de hoofd softwareontwikkelaar. Hij houdt zich bezig met interne ontwikkeling en begeleid ook een extern ontwikkelteam.

Naast de vaste werknemers zijn er ook een aantal zzp'ers die regelmatig werk verrichten, zowel op kantoor als buiten. Deze personen zijn voornamelijk verantwoordelijk voor het operationeel houden van verschillende servers.

De afdeling werkt over het algemeen informeel, maar wel gestructureerd. Er wordt gebruik gemaakt van vaste tooling, welke het ontwikkelproces ondersteund. Veel van deze tooling, zoals Jira, Git en verschillende testomgevingen, worden intern gehost op virtuele omgevingen.

De sfeer op de werkvloer is erg prettig. Er is altijd tijd voor wat ontspanning tussendoor, maar dat betekent niet dat er niet hard gewerkt wordt.

3. Project

3.1. Aanleiding

Telsystemen generen tellingen, welke naar een server gestuurd worden. Aan de hand van deze tellingen worden reportages gegenereerd voor de klant. Nu kan het voorkomen dat een telsysteem om wat voor reden dan ook geen goede of helemaal geen tellingen naar de server gestuurd heeft. Zodra dit het geval is moet dit zo snel mogelijk gedetecteerd en gerepareerd worden. Data reparatie kan op dit moment plaatsvinden in meerdere omgevingen.

3.2. Probleemstelling

De verschillende vestigingen van PFM-Intelligence werken met verschillende softwarepakketten, waar tellingen in gerepareerd kunnen worden. De directeur wil graag dat iedereen met één standaardpakket werkt, om de data reparatie uit te voeren. In Nederland wordt met behulp van PFM totaal en Advantage II gewerkt en in Engeland met Advantage I. PFM totaal is opgesplitst in twee tools, één voor wifi en één voor tel data. In hoofdstuk 0 is meer te lezen over de beginsituatie. In de huidige omgevingen is nog te veel handmatig werk, wat veel tijd en geld kost.

3.3. Doelstelling

In deze opdracht zal een webapplicatie gemaakt worden, waar tel-data in gecorrigeerd kan worden. Deze applicatie zal een vervanger zijn voor alle huidige data reparatie software. De nieuwe applicatie dient slimmer te zijn dan de huidige omgevingen. Het moet de gebruiker suggesties kunnen geven wat wel en niet gerepareerd moet worden en hoe de reparatie er vervolgens uit dient te zien.

Performance is een belangrijk onderdeel binnen het project. Er zal onderzocht moeten worden hoe de applicatie schaalbaar gemaakt kan worden, zodat de applicatie om kan gaan met een grote hoeveelheid tel-data. Hiervoor zal gekeken worden naar frameworks als Akka.NET en Orleans. Deze frameworks lossen dit probleem mogelijk op door gebruik te maken van het actor model. De front-end van de webapplicatie zal gemaakt worden als een single page application. Zo kan ervoor gezorgd worden dat er zo veel mogelijk werk van de webserver uithanden gegeven kan worden. Er moet nog wel onderzocht worden van welk framework hiervoor gebruik gemaakt zal worden.

Voor de webapplicatie is het van belang dat deze uit te breiden is met de ondersteuning van andere bedrijfsprocessen. Het data correctie algoritme zal mogelijk omgaan met gevoelige informatie, zoals gescande mac-adressen. Hierdoor is het van belang dat er rekening gehouden wordt met de beveiliging. Het algoritme zelf zal door de data analisten opgesteld worden.

De webapplicatie moet gemaakt worden met de nieuwste standaarden en dient gebruiksvriendelijk te zijn op de volgende display types: Desktop PC (16:9 & 21:9), tablet en smartphones.

3.4. Resultaat

Nadat de opdracht is uitgevoerd zal een werkende demo van de webapplicatie met de data reparatie functie opgeleverd worden. Als besloten wordt dit project voort te zetten of in gebruik te nemen, zal het, het werk van de data analisten van CityTraffic verlichten.

4. Plan van aanpak

4.1. Ontwikkelmethode

De ontwikkelmethode die in dit project gebruikt wordt is Kanban. Kanban is de gebruikte ontwikkelmethodiek bij CityTraffic. Het bedrijf wil agile werken en andere methodieken als scrum hebben te veel overhead in een project dat door één man wordt uitgevoerd. Er is gekozen om agile te werken, zodat snel feedback gevraagd kan worden van de stakeholders. De requirements zullen in het begin niet allemaal bekend zijn. Hoewel Bart wel een globaal beeld heeft zullen de inhoudelijke pagina's van de grond af opgebouwd moeten worden in overleg met de stakeholders. Nog een reden is dat de stakeholders snel resultaat willen zien, hoewel dit ook mogelijk is met andere methodes, heeft agile hier de beste ondersteuning voor. Voor dit project zal in sprints gewerkt worden, welke een dynamische grootte hebben. Deze sprints mogen niet langer dan twee weken duren. Indien een taak langer dan twee weken in beslag neemt dient de taak opgesplitst te worden in sub-taken. Er wordt op deze manier gewerkt, zodat snel nieuwe afzonderlijke functionaliteiten aan de stakeholders opgeleverd kunnen worden.

4.2. Risico's

Het grootste risico is de beveiliging van de webserver. Doordat de webapplicatie in aanraking komt met gevoelige gegevens is het van uiterst belang dat deze gegevens niet in verkeerde handen belanden. Ook mogen alleen geselecteerde gebruikers bij de gegevens komen. Om dit risico zo goed mogelijk te tackelen zal een plan opgesteld worden, welke beschrijft hoe zowel de front-end als backend omgaat met de beveiliging. Hierin zal ook ingegaan moeten worden op de auditing, autorisatie en authenticatie. Indien de stakeholders dit probleem op een later moment willen tackelen, zal dit buiten de scope van het project vallen. Wel zal in de toekomst goed naar dit probleem gekeken moeten worden. Het product kan NIET live gaan voordat dit uitgewerkt is.

Een ander groot risico is de performance van de webserver. De server zal te maken krijgen met de verwerking van grote hoeveelheid data. In de toekomst zal de webserver mogelijk uitgebreid worden met andere bedrijfsprocessen, zoals locatie management, waardoor het verkeer met de server zal stijgen. Tijdens de ontwikkeling dient rekening gehouden te worden met de schaalbaarheid van de applicatie. Dit risico kan mogelijk verholpen worden met het gebruik van Akka.NET. Mocht dit niet het geval zijn, dan zal naar andere oplossingen gezocht moeten worden.

Het detectie/reparatie algoritme is nog niet bekend en moet nog worden onderzocht. Dit kan een grote impact hebben op de database dat gebruikt zal worden. Zodra het algoritme bekend is zal onderzocht moeten worden wat voor impact dit zal hebben op de gebruikte database(s).

4.3. Stakeholders

Dit project heeft vijf stakeholders, waarvan de meesten een unieke rol zullen spelen binnen dit project. Hieronder een korte introductie:

- Bart (CEO)
  Bart is de overkoepelende directeur van alle bedrijven die onder PFM vallen. Zijn rol binnen dit project is om de beslissende kracht te zijn. Daarnaast zal vooral zijn mening gevraagd worden als het gaat om de opmaak en lay-out van het te maken product.

- Koen (Data Analist)
  Koen is één van de twee data analisten binnen dit project. De rol van de data analist is het stellen van de eisen waar het programma aan moet voldoen. Hij is ook verantwoordelijk voor de inhoudelijke gegevens van de verschillende pagina's.

- Adrie (Data Analist)
  Adrie is de tweede data analist, welke betrokken is bij dit project. Hij is op dit moment part time in dienst en werkt drie middagen per week. Om deze reden zal hij niet bij elk gesprek aanwezig kunnen zijn.

- Gerrit (Ontwikkelaar/Systeembeheerder)
  Zoals eerder gezegd is Gerrit zowel ontwikkelaar als systeembeheerder. Hij is de originele ontwikkelaar van PFM totaal en heeft zo al ervaring met het data reparatie probleemstuk. Daarnaast bezit hij veel kennis over hoe op dit moment de verschillende systemen met elkaar werken. Zijn rol binnen dit project is het adviseren hoe het systeem in het totaal plaatje kan passen. Daarnaast heeft hij de meeste kennis voor wat betreft de huidige problemen die spelen binnen het gehele systeem.

- Chris (Ontwikkelaar)
  Naast de begeleidende rol heeft Chris ook de rol als stakeholder. Hoewel niet de originele ontwikkelaar, is hij op dit moment de hoofd ontwikkelaar van Advantage II, een omgeving waar dit project van afhankelijk is en eerder een poging voor data correctie en detectie in is gedaan.

4.4. Afspraken

Voordat aan het project begonnen is zijn meerdere afspraken gemaakt rondom het project. Naast onderstaande afspraken dient de code waar nodig ook gedocumenteerd te worden.

4.4.1. Definition of Done

Voor elke Kanban card geldt dat deze moet voldoen aan de definition of done voordat deze als klaar gezien kan worden. Hier volgt de algemene definition of done, voor afwijkende items zal de definition of done in Jira bijgehouden moeten worden. Nadat de requirements voor een card bekend zijn kan deze ontwikkeld worden. Voor elke feature moet gekeken worden of deze uitgewerkt dient te worden met een UML-diagrammen. Nadat de feature is ontwikkeld moet deze getest worden. Pas nadat de feature door de ontwikkelaar is goedgekeurd zal de feature door de stakeholders getest worden. Zodra de stakeholders de feature goedgekeurd heeft zal de bijbehorende card als done gezien worden.

4.4.2. Tools

Tijdens het project zal gebruik worden gemaakt van de volgende tools, tabel 1:

Tool	Beschrijving
Jira	Ticketsysteem dat zal worden gebruikt als Kanban bord
Draw.io	Webtool voor het maken van diagrammen
Microsoft Office	Alle documentatie wordt hiermee opgesteld
Visual Studio 2015	De Integrated development environment

Tabel 1: Gebruikte tools

4.4.3. Taal en libraries

Binnen PFM is het de voorkeur dat gewerkt wordt met C#. Daarnaast vindt de directeur het belangrijk dat er altijd met de nieuwste standaarden gewerkt wordt. Hierdoor zal de backend van de webserver gemaakt worden met behulp van ASP.NET Core. De voorkeur is dat deze code ook gecompileerd wordt met de dotnet core library, om zo cross platform te kunnen ondersteunen. De client zal gemaakt worden in een SPA framework, welke tijdens het ontwikkelproces gebruik maak van Node.js, NPM en Webpack.

4.4.4. Meetings

Na elke sprint dient er een acceptatie review gehouden te worden. In deze meeting krijgen stakeholders de gelegenheid om door de applicatie heen te klikken en feedback te geven. De (tussen) producten zullen, voordat er aan de implementatie begonnen wordt, met de begeleider besproken worden. Dit om ontwerpproblemen in een vroegtijdig stadium op te sporen.

4.5. Producten

Om dit project te realiseren zullen een aantal (tussen) producten gemaakt worden. Deze producten zijn op te delen in vier categorieën: Onderzoek, Requirements, Design en Testen. In de loop van het project zal bepaald worden uit welke deel producten deze rapporten zullen bestaan.

Op dit moment is het wel zeker dat er een onderzoek naar het te gebruiken SPA framework en Akka.NET zal worden gedaan. Het actor onderzoek zal voor Akka.NET worden gemaakt, dit is de enige library die op dit moment de dotnetcore runtime ondersteund. Ook zal er een performance test gemaakt worden voor het te maken algoritme.

4.6. Globale planning

De IT-afdeling wil van tevoren al een beeld hebben hoe de uiteindelijke applicatie technisch in elkaar zit, zoals welke frameworks er allemaal gebruikt worden. Hierdoor zullen de onderzoeken voor zowel front- als backend eerst uitgevoerd worden. Doordat de onderzoeken eerst uitgevoerd worden kan in een vroeg stadium rekening gehouden worden met de performance van het systeem. Het zal al snel duidelijk zijn of Akka.NET hier een geschikt framework voor is. In de onderzoek fase van het project zal niet in sprints gewerkt worden.

Tijdens de ontwikkeling zullen tussentijdse acceptatiegesprekken en interviews met de stakeholders plaatsvinden. In tabel 2 is een overzicht van de planning te zien. Nadat deze planning is uitgevoerd, zal niet aan alle eisen voldaan zijn, maar zal wel een basis van de uiteindelijke applicatie werkend zijn. Beveiliging is een groot risico, toch verwacht ik geen tijd te hebben dit in te bouwen in de applicatie. Wel zal tijdens de ontwikkeling rekening gehouden moeten worden dat dit een belangrijk punt is. Het detectie/reparatie algoritme is van tevoren niet bekent. Het is niet mijn taak dit algoritme te bedenken, enkel te implementeren. Het is mogelijk dat het algoritme niet op tijd uitgewerkt is, mocht dit het geval zijn dient er een wijziging in de planning plaats te vinden.

Week	Datum	Beschrijving
1	09-02-2017	Oriëntatie huidige situatie.
2	15-02-2017	Onderzoek SPA framework.
	23-02-2017	Opzet project in Visual Studio & Inrichting verschillende omgevingen (ontwikkel, test en database).
4	27-02-2017	Studeer over Akka.NET
	03-03-2017	Go / No Go gebruik Akka.NET
5	06-03-2017	Eerste gesprek met alle stakeholders. Opstellen en uitwerken van initiële requirements.
	09-03-2017	Ontwikkeling basis layout website.
7	20-03-2017	Acceptatietest layout website & verwerken van feedback.
	23-03-2017	Begin ontwikkeling data-correctie pagina’s.
10	10-04-2017	Acceptatietest data-correctie pagina’s & verwerken van feedback.
11	17-04-2017	Ontwikkeling automatische data-correctie methode.
13	01-05-2017	Performancetest automatische data-correctie methode.
14	08-05-2017	Implementeer Akka.NET in de backend van de website.
15	15-05-2017	Performancetest automatische data-correctie methode. Met Akka.NET implementatie.

Tabel 2: Globale planning

5. Beginsituatie

De scope van dit project is CityTraffic, wel wordt de beginsituatie van andere bedrijven van PFM besproken. De nieuwe applicatie zal een vervanging zijn voor alle huidige data reparatie software. Om deze reden zal de beginsituatie van al deze omgevingen kort beschreven worden.

Bij PFM Footfall Intelligence zijn drie omgevingen waar data reparatie plaats kan vinden:

1. **Advantage I**
   Advantage I is het originele systeem van PFM. Deze omgeving is ontwikkeld ver voordat CityTraffic onder de PFM-naam is gekomen. Het systeem is erg verouderd en werkt niet meer goed in moderne browsers. Tot ongenoegen van Bart, werkt PFM Engeland nog steeds met deze omgeving. Dit komt doordat zij van mening zijn dat Advantage II niet genoeg te bieden heeft, om de overstap te maken. Er ontbreken cruciale functionaliteiten.

2. **Advantage II**
   Ter vervanging van Advantage I is Advantage II ontwikkeld. Deze omgeving is in Nederland volop in gebruik, behalve het data reparatie onderdeel. De voornaamste reden dat dit onderdeel niet in gebruik genomen is, is omdat het systeem te traag is. Zodra dit probleem geconstateerd werd is besloten de ontwikkeling voor het data reparatie onderdeel stop te zetten. Verder is het systeem inmiddels ook verouderd.

3. **PFM Totaal**
   PFM Totaal functioneert o.a. als vervanging voor de missende data reparatie in Advantage II. PFM Totaal is een Windows desktopapplicatie welke twee verschillende omgevingen heeft voor wifi en footfall data.

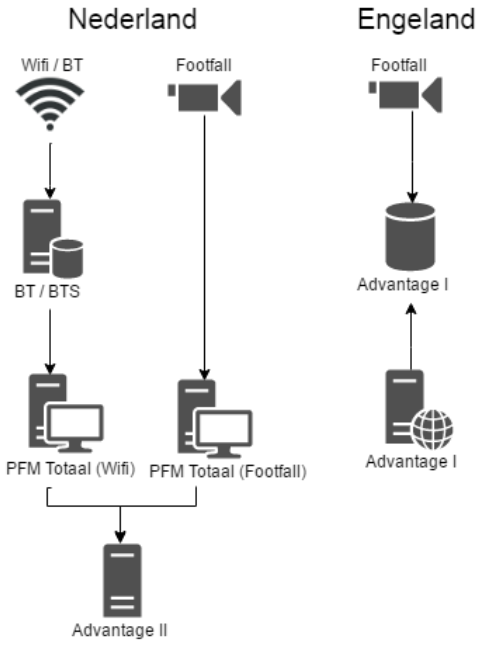
In het geval van CityTraffic wordt gebruik gemaakt van PFM Totaal, voor de data reparatie. Nadat de data gecorrigeerd en goedgekeurd is wordt deze data opgeslagen in Advantage II. Vanuit Advantage II worden de reportages gemaakt voor de klanten. Zowel advantage I als II hebben een webapplicatie waar klanten de tellingen op kunnen bekijken.

5.1. Datastroom

Footfall en wifidata wordt al op meerdere plekken verwerkt, voordat het in de advantage omgeving komt. In figuur 2 is een flowchart weergeven van de datastroom in de huidige situatie. Deze flowchart is op te splitsen in twee delen, een Nederlands en een Engels deel.

5.1.1. Nederland

Voor het Nederlandse gedeelte is te zien dat er zowel wifiscanners als footfall devices zijn. Wifiscanners leveren de tel data door middel van http-verzoeken aan de BT- of BTS-server. Deze twee servers zijn binnen PFM verantwoordelijk voor alle ruwe wifi en bluetooth data. Met ruwe data wordt elk gescande wifi en bluetooth mac-adres bedoelt. Deze data wordt vervolgens geaggregeerd op half uur basis en één keer per dag verzonden naar de wifi PFM Totaal omgeving. PFM Totaal zal één keer per dag alle tel-data (wifi & footfall) versturen naar Advantage II.



Figuur 2: Huidig situatie

Footfall data kan op meerdere manieren in de footfall omgeving van PFM Totaal terechtkomen. Voor dit project is het niet van belang deze verschillende wegen te kennen. Wel is het van belang te weten dat deze tel data van tevoren al geaggregeerd is, in de meeste gevallen op half uur basis. Net als bij wifi wordt de data één keer per dag gerepareerd in PFM Totaal. De gerepareerde data wordt vervolgens via ftp verzonden naar Advantage II verstuurd.

5.1.2. Engeland

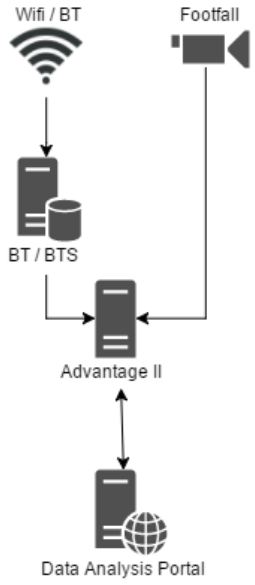
In Engeland wordt alleen gebruik gemaakt van footfall devices. Deze apparaten sturen hun tel-data rechtstreeks naar de SQL-database van Advantage I. De PFM-medewerkers in Engeland repareren deze data in de webapplicatie van Advantage I. Het voordeel van deze manier van werken is dat de klant gelijk nieuwe tellingen van de huidige dag in de webapplicatie kan zien. In Nederland pas nadat het door PFM-Totaal verstuurd is naar Advantage II.

5.2. Nieuw systeem

In het nieuwe systeem (figuur 3) zal de reparatie van PFM Totaal overbodig zijn. De wifi en footfall tellingen zullen ongeprepareerd opgeslagen worden in advantage II. Hierdoor zal de klant tellingen van de huidige dag sneller in de webapplicatie van Advantage II kunnen zien, net als de huidige situatie in Engeland. Engeland zal ook over moeten gaan op Advantage II.

Het nieuwe systeem, in figuur 3 Data Analysis Portal genoemd, zal gebruik maken van de Advantage II database. Vanuit deze database kan naar verwachting alle benodigde data opgehaald worden. De gerepareerde data moet ook opgeslagen worden in de advantage omgeving.

Het nieuwe systeem dient er rekening mee te houden dat de Advantage II omgeving in de toekomst veranderd of vervangen wordt. De originele ontwikkelaars van de Advantage omgeving zijn niet meer in dienst. Om die reden is het de wens dat de huidige structuur van de Advantage database niet veranderd. Zo kan voorkomen worden dat bestaande applicaties omvallen.



Figuur 3: Nieuw systeem

6. SPA framework

Voordat er iets gebouwd kan worden is het van belang dat er een keuze wordt gemaakt over het single page application framework dat gebruikt zal worden. Bij het opstellen van het plan van aanpak was het mijn idee gebruik te maken van een single page application. Ik wist dat snelheid een belangrijk onderdeel is van de webapplicatie en een SPA kan hier een belangrijke rol in spelen. Met het gebruik van een SPA zorg je ervoor dat zo veel mogelijk werk van de webserver in de browser plaatsvindt, zoals het renderen van de pagina's. Daarnaast kan een SPA de eindgebruiker de illusie geven dat alles snel is. Alles gebeurd in de browser, wat het navigeren tussen verschillende pagina's erg 'responsive' maakt. Dit geeft een betere user experience dan als de gebruiker moet wachten totdat de webserver een nieuwe pagina stuurt.

In overleg met de IT-afdeling zijn er een aantal eisen voor het framework opgesteld, dit zijn niet de enige eisen, maar wel de belangrijkste:

- Beschikt over lazy loading
  Een SPA stuurt in één keer een groot gedeelte of de gehele website naar de gebruiker, wat de initiële laad tijd van de website vergroot. Lazy loading zal mogelijk in de toekomst gebruikt worden om laadtijden te verbeteren.

- Ondersteuning voor TypeScript
  Typescript is een programmeertaal welke compileert naar JavaScript. Binnen PFM werken voornamelijk C# programmeurs. Binnen het bedrijf wordt verwacht dat de overbrugging van C# naar TypeScript makkelijker is dan naar JavaScript. Om deze reden is het de wens, mede als proef, te werken in TypeScript. Omdat TypeScript nog relatief nieuw is, is deze eis niet verplicht, maar gaat wel de voorkeur hiernaar uit.

- Ondersteuning voor 2-way databinding
  2-Way databinding is een standaard feature voor spa frameworks.

Naast bovenstaande eisen is de keuze gebaseerd op de populariteit, ondersteuning en de snelheid van het framework. Voor het onderzoek zijn onderstaande frameworks (tabel 3) geselecteerd. Deze frameworks waren van tevoren bekend bij de ontwikkelaars, maar er was binnen PFM nog niet eerder mee gewerkt.

	React	Angular 2	Ember	Vue.js	Aurelia
Bedrijf / sponsors	Facebook	Google	(meerdere)	(meerdere)	Blue Spire
Versie	15.4.2	2.4.7	2.11.0	2.1.10	1.0.8
Eerste uitgaven (volledig 1.0)	29 mei 2013	15 sep 2016	1 sep 2013	27 okt 2015	27 jul 2016
Taal	JSX	TypeScript	JavaScript	JavaScript	JavaScript

Tabel 3: Overzicht SPA frameworks

Nadat een selectie van de frameworks was gemaakt bleek al snel dat React geen ondersteuning heeft voor 2-way databinding en zal dus ook niet gebruikt worden. De overige requirements worden wel door elk framework ondersteund.

6.1. Community & Populariteit

De populariteit en omvang van het framework is van belang voor de verwachte ondersteuning van het framework op de lange termijn. Het is lastig om een inschatting te maken hoe populair iets is zonder een grote hoeveelheid data te verzamelen. Elk van de geselecteerde frameworks is open source en toegankelijk op GitHub, wat het onderzoek in staat stelt gegevens van GitHub te gebruiken. Er is voor gekozen een schatting te maken hoe populair een framework is door te kijken naar drie aspecten: Watchers, Stars en Forks.

Watchers zijn mensen die de repository volgen. Stars zijn mensen die de repository leuk vinden. Forks zijn mensen die de repository gekloond hebben, om zo zelfstandig iets door te kunnen ontwikkelen.

De resultaten van het onderzoek zijn in tabel 4 te vinden. Uit de rangschikking is te zien dat Angular 2 op dit moment het populairst is, gevolgd door Vue.js.

1	Angular 2
2	Vue.js
3	Aurelia
4	Ember

Tabel 4: Rangschikking populariteit

6.2. Snelheid

Snelheid is een risico binnen het project en er zal dus goed gekeken worden naar welk van de frameworks het snelst is. Gezien de tijd is het niet realistisch om voor elk framework een performance test te maken. Gelukkig is op internet genoeg informatie te vinden, zo ook performance testen. In het onderzoek zijn vele scenario's behandeld, welke te maken hebben met het manipuleren van data in tabellen. Voorbeelden zijn: het creëren/verwijderen van rijen en het wijzigen van de inhoud. [1]

6.2.1. Keyed & non-keyed

Niet elk framework werkt op dezelfde manier. Er is een splitsing te maken tussen keyed en non-keyed frameworks. Het gebruik van keyed of non-keyed heeft impact op de performance en usability van het framework. De verschillen zijn vooral te merken bij het gebruik van lijsten van data, bijv. een tabel met honderden of duizenden regels. [2]

Het verschil tussen keyed en non-keyed is het beheren van de DOM nodes op een pagina. Zodra iets in de controller veranderd, zal de keyed methode alle DOM nodes opnieuw creëren. De non-keyed methode zal de bestaande Dom nodes updaten. [2]

Voor dit project is besloten gebruik te maken van een keyed framework. Aurelia zal dus niet gebruikt worden. De keyed methode heeft een één op één relatie tussen de data in de controller (framework) en de DOM node. Dit zorgt ervoor dat wanneer de DOM node buiten het framework veranderd, dit geen impact heeft op de controller. Het voordeel hiervan is dat, wat de eindgebruiker ziet, altijd gelijk is aan wat er in de controller staat. Op deze manier kunnen geen onverwachte veranderingen naar de server gestuurd worden, of juist veranderingen verloren gaan.

6.2.2. Resultaten

Er zijn in totaal negen scenario's uitgevoerd tijdens de performance testen. Deze resultaten zijn te vinden in tabel 5 en zijn gegeven in milliseconden. In tabel 6 is het RAM gebruik van de frameworks gegeven in megabytes. [3]

	S1	S2	S3	S4	S5	S6	S7	S8	S9
Angular v2.4.3	178.12	188.54	9.86	3.89	11.34	48.68	1832.15	288.12	342.39
Ember v2.10.0-beta.3	309.09	283.64	14.91	5.94	15.26	53.36	2496.88	457.75	251.29
Vue v2.1.10	151.97	158.72	15.62	8.37	16.81	55.36	1551.08	369.41	248.823.33

Tabel 5: SPA framework performance tests

	Ready memory Memory usage after page load.	**Run memory** Memory usage after adding 1000 rows.
Angular v2.4.3	5.86	12.46
Ember v2.10.0-beta.3	10.76	21.20
Vue v2.1.10	3.65	8.89

Tabel 6: SPA framework RAM gebruik

Aan de hand van de resultaten kan gezien worden dat Ember op dit moment het minst presteert. In combinatie met de laagste scoren in de populariteit test is besloten Ember niet te gebruiken.

6.3. Uitkomst

Aan de hand van het onderzoek is besloten de front-end te maken in Angular 2. De keuze was tussen Angular 2 en Veu.js, welke beiden soortgelijk scoorde in alle categorieën. De keus voor Angular is gebaseerd op de toekomstplannen, hierin klinkt Angular een stuk aantrekkelijker. Kijkende naar het komende jaar lijkt het erop dat Angular 2 een actievere ontwikkeling heeft dan Vue.js. Dit kan betekenen dat Vue.js nu wel erg aantrekkelijk lijkt maar in de nabije toekomst misschien niet. Daarnaast is Angular 2 alleen in TypeScript beschikbaar, dus zal de support hier naar verwachting beter voor zijn dan Vue.js. [4] [5]

7. Opzet project

Het is de wens van de directeur dat de applicatie van de nieuwste technologieën gebruik maakt en daar hoort ook een moderne ontwikkelomgeving bij. In de planning is twee dagen gerekend voor de opzet van het project, echter nam dit veel meer tijd in beslag, een zeven dagen. Later in dit hoofdstuk is meer te lezen over de opgelopen vertraging.

7.1. Compilatie proces

Zoals eerder gezegd compileert TypeScript naar JavaScript en hier wordt de TypeScript compiler voor gebruikt. Er kan op twee verschillende manieren gecompileerd worden, namelijk: Ahead of Time (AOT) en Just in Time (JIT). Voor dit project zal alleen gebruik worden gemaakt van de JIT-compiler. Voornamelijk omdat deze makkelijker op te zetten is en de poging de AOT-compiler op te zetten mede verantwoordelijk is voor de opgelopen vertraging. Het grootste verschil tussen deze twee methodes is het moment waarop de code gecompileerd wordt van TypeScript naar JavaScript. Een ander verschil is het moment dat de Angular app gecompileerd wordt. [6]

Een Angular app bestaat voor een groot gedeelte uit, wat de Angular ontwikkelaars noemen, templates. Deze templates moeten vervolgens omgezet worden naar daadwerkelijk bruikbare code voor een browser. Dit proces noem ik de Angular (app) compilatie. [6]

- Just in Time (JIT)
  Met een JIT compilatie wordt de Angular app altijd voor elke gebruiker apart gecompileerd in de browser. Dit heeft als voordeel dat er gebruik kan worden gemaakt van Hot Module Replacement, hoofdstuk 7.2.1. [6]

- Ahead of Time (AOT)
  Wanneer gebruik wordt gemaakt van AOT-compilatie zal de gehele source (inclusief Angular app) één keer compileren in de ontwikkel omgeving. Dit maakt de laadtijden van de app een stuk sneller, dan met een JIT-compilatie. Omdat AOT al gecompileerd is hoeft de Angular compiler niet meegesleurd te worden naar de client, wat de bestanden een stuk kleiner maakt. Daarnaast maakt AOT gebruikt van tree shaking. Tree shaking is een Angular techniek, wat de source scant voor ongebruikte code en deze code verwijderd. [6]

  Een AOT-compilatie is ook veiliger, omdat de app niet gecompileerd hoeft te worden in de browser. Dit zorgt ervoor dat er minder momenten zijn waarin de code open is voor injectie van kwaadaardige code. [6]

De JIT-compiler is erg geschikt tijdens de ontwikkeling van de website, maar voor productie is AOT duidelijk meer geschikt. Tijdens dit project is beveiliging een belangrijk punt en hierdoor wilde ik graag een werkende AOT-compilatie hebben.

7.2. Webpack

In het plan van aanpak is te lezen dat er gebruik wordt gemaakt van Node.js, NPM en Webpack. Het gebruik van Node,js is simpel te verklaren, Angular heeft dit nodig om te werken. NPM is een package manager gebundeld in Node.js en wordt gebruikt om JavaScript dependencies (o.a. Angular zelf) te downloaden en te beheren.

Webpack is een module bundler voor moderne JavaScript applicaties, dit houdt in dat het verschillende losse bestanden samenvoegt tot één. Webpack wordt gebruikt zodat de uiteindelijke code die naar de browser gestuurd wordt, geoptimaliseerd is. De bestanden die gestuurd worden zijn zo klein mogelijk en bevatten alleen de daadwerkelijk gebruikte code van de app. Naast de

optimalisatie beschikt Webpack ook over Hot Module Replacement wat veel tijd bespaart tijdens de ontwikkeling.

7.2.1. Hot Module Replacement (HMR)

Een hulpmiddel waar tijdens dit project gebruik van is gemaakt is hot module replacement. Hot Module Replacement is een techniek om source aanpassingen runtime naar de browser te sturen. De client applicatie zal deze aanpassingen vervolgens, zonder de browser pagina te herladen, aan de gebruiker te tonen. Met het gebruik van HMR zorg je ervoor dat de webserver niet na elke verandering opnieuw opgestart hoeft te worden. Dit bespaart veel wachttijd tijdens de ontwikkeling. [7]

7.3. Configuratie

Er was moeite om webpack en het compilatie proces goed opgezet te krijgen, hierdoor is in overleg met Chris besloten te werken vanuit een template configuratie. Om zo niet meer tijd te verliezen dan al verloren was gegaan. Hier zal in de evaluatie op terug gekomen worden.

De website wordt gecompileerd in vier onderdelen: polyfill, vendor, client en chunk(s)

- Polyfill
  Een polyfill JavaScript file wordt gebruikt voor compatibiliteit patches. Zeker bij een SPA is het van belang dat de browser alles goed ondersteund. Jammer genoeg ontbreekt er in de praktijk het een en ander aan functionaliteit. Om een browser hier toch ondersteuning voor te geven worden polyfills gebruikt.

- Vendor
  Een vendor JavaScript file wordt gebruikt voor alle stukken code die gedeeld worden door zowel de Client als de Chunk. Met het gebruik van een vendor file hoeven deze stuken code niet meerdere keren gebundeld te worden.

- Client
  De client is de main (root) app, het hart van de applicatie.

- Chunk(s)
  Alle onderdelen die lazy loaded zijn worden gecompileerd als chunk. Een chunk zal pas naar de browser van de gebruiker verstuurd worden zodra de app het nodig heeft.

8. Akka.NET

De performance is van groot belang binnen dit project. De webserver zal een grote hoeveelheid data te verwerken krijgen, welke zo snel mogelijk verwerkt moet worden. Om het performance risico binnen het project te tackelen kan mogelijk gebruik gemaakt worden van Akka.NET. Chris wist van het bestaan van dit framework en wat het belooft. Binnen de IT-afdeling is nog nooit gewerkt met Akka.NET, of een soortgelijk framework, waardoor het gebruik van Akka.NET ook als test gebruikt zal worden of mogelijk meer systemen ontwikkeld kunnen worden met dit framework.

Het belangrijkste punt dat uit dit onderzoek moet blijken is de vraag:

"*Hoe zal Akka.NET geïntegreerd worden binnen dit project?*"

Uit het onderzoek kan ook blijken dat Akka.NET niet geschikt is binnen dit project. Mocht dit het geval zijn zal naar andere oplossingen gezocht moeten worden. In dit onderzoek wordt allereerst

achterhaald hoe Akka.NET te werk gaat om problemen op te lossen, die geclaimd worden opgelost te worden. Voordat aan dit onderzoek begonnen was, was mijn kennis van Akka.NET minimaal (wel van gehoord). Om deze reden is ervoor gekozen een bootcamp te volgen van Petabridge (bron: [8]). Deze bootcamp wordt aangeboden door de ontwikkelaars van Akka.NET en leert de deelnemer in drie verschillende units alles wat nodig is om een eerste project met Akka.NET op te zetten.

8.1. Algemene info

Dit hoofdstuk is bedoeld om mensen een eerste indruk te geven wat Akka.NET en het actor model is. Deze informatie is cruciaal voor de lezer om te weten, omdat het framework de structuur van een applicatie beïnvloed. Het exacte werken en gedrag van een actor zal op een later moment worden uitgelicht.

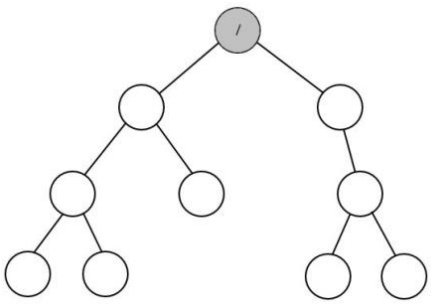
8.1.1. Wat is Akka.NET?

Akka.NET is een framework dat zich richt op schaalbare, fouttolerante real-time dataverwerking. Dit wordt bereikt door gebruik te maken van het actor model. Akka.NET is een .NET port van de scala variant Akka, welke al langere tijd succesvol wordt toegepast in grote productie omgevingen.

"We believe that writing correct, concurrent, fault-tolerant and scalable applications is too hard. Most of the time, that's because we are using the wrong tools and the wrong level of abstraction. Akka is here to change that." – Akka.NET

8.1.2. Actor systeem

Een actor is een object welke zowel status als gedrag bezit. Actoren kunnen door middel van berichten met elkaar communiceren, deze berichten kunnen zowel synchroon als asynchroon zijn. Naast berichten zijn er geen manieren om met actoren te communiceren, alles binnen een actor (attributen en methodes) zijn private. Doordat alles private is kan Akka ervoor zorgen dat thread-safety gewaarborgd is. Het actor systeem kan het best gezien worden als een groep mensen. Een persoon is de leider, die afzonderlijke taken verdeeld onder zijn team. Hierdoor ontstaat een hiërarchische structuur, deze structuur is gevisualiseerd in figuur 4. In dit figuur is de grijze actor met '/' de leider, deze actor wordt de root actor genoemd. [9]



Figuur 4: Het actor model gevisualiseerd

Het voordeel dat het actor systeem biedt over het traditionele multithreading is dat de ontwikkelaar niet na hoeft te denken over thread beheer, het actor systeem neemt deze taak op zich. Hierdoor kunnen actoren in één thread draaien of in honderd verschillende. [9]

8.1.3. Schaalbaarheid

Schaalbaarheid is een belangrijk aspect van Akka. In 8.1.2 is te lezen dat het actor model gezien kan worden als een groep mensen, met elk afzonderlijke taken. Als uitbreiding hierop kan ook ingebeeld worden dat er meerdere mensen zijn met dezelfde taak. Zodra de leider meerdere keren dezelfde taak krijgt kan deze taak verdeeld worden over de groep mensen die deze taak kunnen uitvoeren. In Akka kan dit ook, door middel van verschillende strategieën kunnen meerdere actoren aangemaakt worden met dezelfde taak. Dit principe wordt routing genoemd. [10]

8.1.4. Fouttolerantie

Fouttolerantie is het tweede belangrijke aspect van Akka, dit kan bereikt worden op meerdere manieren. Zoals in 8.1.2 te lezen is, is er een hiërarchische structuur van actoren. De relatie tussen actoren kan gezien worden als een child parent relatie. Elke parent is verantwoordelijk voor de child actoren, dit wordt gedaan door middel van de supervisor strategy. Zodra een child actor stopt met werken zal de parent beslissen wat er moet gebeuren en kan de child actor opnieuw opgestart worden. Alle berichten die de gefaalde actor gekregen heeft kunnen alsnog verwerkt worden door een andere actor, hierdoor zal er geen werk verloren gaan. [11]

8.2. Scenario's

Na de bootcamp uitgevoerd te hebben was het voor mij duidelijk dat Akka gebruikt kan worden binnen dit project, het leek mij ook verstandig dit te doen. Met de kennis die ik al had van de structuur van de database, kon ik voor mezelf een beeld vormen hoe het algoritme opgesplitst kan worden in actoren, meer in 12.3.2.

Met het gebruik van Akka kan geen inschatting gemaakt worden hoe snel het algoritme zal zijn, het algoritme was ook nog niet bekent. Wel kan er gesteld worden dat het algoritme met de business mee kan groeien. De schaalbaarheid die Akka biedt kan ervoor zorgen dat het algoritme niet exponentieel groeit naarmate de business groter wordt. Er zijn drie manieren hoe Akka geïntegreerd kan worden in de webserver.

8.2.1. Oplossing 1: Een nieuw systeem

De beste lange termijn oplossing is het maken van een nieuw systeem. Dit systeem kan van de grond af opgebouwd worden met Akka, waarmee een betrouwbare backend server gemaakt kan worden. In de huidige situatie wordt gebruik gemaakt van een mssql database. Databases staan erom bekent niet goed mee te schalen, zodra meer en meer query's tegelijk uitgevoerd moeten worden. Om het performance probleem helemaal op te lossen moet deze database afhankelijkheid weg. Deze oplossing is ver buiten de scope van dit project en zal daarom niet gebruikt worden.

8.2.2. Oplossing 2: Integreren huidig systeem zonder memory buffer

De snelste oplossing is om Akka te integreren in het huidige systeem zonder memory buffer. Akka zou hierin alleen gebruikt kunnen worden voor de taken die ook met multithreading opgelost kunnen worden. Dit komt voornamelijk doordat Akka alsnog afhankelijk zal zijn van de tijd dat het kost om query's en http requests, naar een api of database uit te voeren. In het geval dat een gebruiker enkel data uit de database op wil halen is het niet verstandig Akka te gebruiken. Akka kan de snelheid van de database niet verbeteren. Wel kan akka ervoor zorgen dat er meerdere database vragen tegelijk uitgevoerd worden.

8.2.3. Oplossing 3: Integreren huidig systeem met memory buffer

De applicatie kan ook ontwikkeld worden met een Akka memory buffer, waarin een kopie/gedeelte van de database in memory wordt bijgehouden. Deze buffer zal lijdend moeten zijn voor de database, wat risico's met zich mee brengt. Alle wijzigingen in de database moeten gesynchroniseerd worden met de buffer. Het voordeel dat je hieruit haalt is dat de applicatie een stuk sneller zal zijn dan met het gebruik van een database.

8.3. Advies & Uitkomst

Mijn advies is om voor oplossing 2 te kiezen. Aan deze oplossing zitten de minste risico's gebonden, er is geen aanpassing aan de huidige systemen voor nodig. Wel kan op deze manier het algoritme geoptimaliseerd worden, door op een gemakkelijke manier meerdere taken gelijktijdig te laten verlopen. Er kan op een veilige manier met multithreading gewerkt worden.

Op lange termijn zal ik wel kijken naar vervangingen voor het huidige systeem. Het lijkt mij niet wenselijk te werken met veel verschillende servers die afhankelijk zijn van elkaar. Zeker met de toenemende vraag naar realtime data en de slechte schaalbaarheid van databases, lijkt oplossing 1 een goede toekomstige investering.

Na de verschillende situaties besproken te hebben met Chris en Bart, is besloten mijn advies op te volgen verder te gaan met oplossing 2. Deze oplossing is het minst risico vol en kan als test functioneren of Akka een geschikt framework is om mogelijk meer software in te ontwikkelen.

9. Basis lay-out

Nadat beide onderzoeken afgerond waren en de ontwikkelomgeving opgezet was, kon begonnen worden aan de eerste echte sprint. Deze sprint was twee weken en staat in het teken van het ontwerpen en ontwikkelen van de basis lay-out voor de webapplicatie. Daarnaast is in deze sprint meerdere standaard benodigdheden ontwikkeld, welke noodzakelijk zijn voor de applicatie.

De eerste stap in deze sprint is het interviewen van de stakeholders, om de requirements voor de website te achterhalen. Tijdens dit gesprek waren er twee punten die ik duidelijk wilde krijgen. Allereerst was het belangrijk om te weten welke designtaal wordt aangehouden. Hiermee wordt bedoeld wat voor soort opmaak/stijl de verschillende html elementen krijgen, zodat alles een mooi geheel wordt. Ten tweede moet een beslissing gemaakt worden welke componenten er zijn en hoe deze eruit zien. Bij componenten kan gedacht worden aan de navigatie menu, toolbar en footer. Om de stakeholders te helpen in het maken van deze keuzes heb ik andere websites als voorbeeld gebruikt. Deze websites waren allen in Angular 2 ontwikkeld met als doelstelling te functioneren als dashboard, of als showcase voor verschillende html elementen zoals dropdown lijsten en toggle knoppen.

9.1. Opmaak

Uit het gesprek met de stakeholders zijn een aantal requirements gekomen betreffende de designtaal deze zullen in dit hoofdstuk behandeld worden.

Non Functional Software Requirements		
NFSR.3	The user interface is responsive. (mobile, tablet and desktop mode)	Should
NFSR.6	The ui uses the material design language	Must

Tabel 7: Opmaak requirements

- Angular material
  Er is ervoor gekozen dat de website een material design krijgt (NFSR.6). Nu was het een kwestie van wat voor library gebruikt zal worden om deze stijl te introduceren aan de website. Twee voornaamste keuzes zijn Angular Material en Bootstrap, beide goed te integreren in Angular. In overleg met Bart is er besloten gebruik te maken van Angular Material. De voornaamste reden voor deze keuze is dat het speciaal voor en door Angular wordt ontwikkeld, als de naam dat nog niet duidelijk maakte. Er wordt gedacht dat Angular Material een betere integratie in Angular heeft en zo een betere lange termijn oplossing is. Een nadeel voor nu is dat Angular Material nog in bèta fase is en er nog cruciale aspecten

missen, zoals datepickers en datatables, maar deze zijn wel te introduceren met 3rd-party alternatieven. Het risico dat ik aangaf, dat er mogelijk stukken code hierdoor herschreven moeten worden op een later tijdstip, vond Bart geen probleem.

- Angular Flex
  Angular Flex is een library ontwikkeld door het Angular team, gemaakt om ui componenten responsive te maken op verschillende scherm groottes (NFSR.3). Met deze library kan de wens om de site op zowel desktop, tablet en mobiel er goed uit te laten zien gewaarborgd worden.

- Material design icons
  Als icon set zal de material design icon set van Google gebruikt worden. Deze icon set wordt geadverteerd door Angular Material en past volledig bij de designtaal. Andere icon sets waar eventueel later ook support voor geleverd kan worden is Font Awseome en Ionicons.

9.2. UI Design

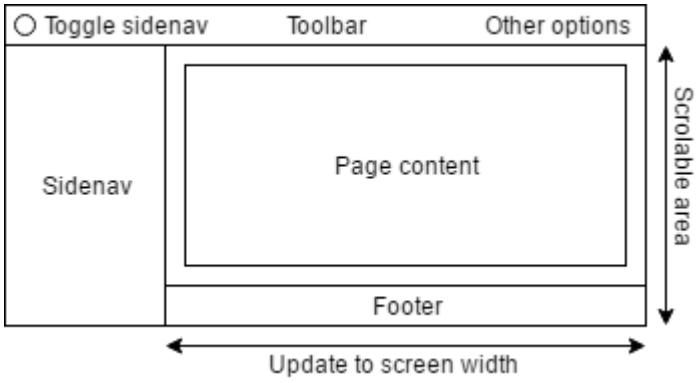
Nadat er een overeenkomst was over de look en feel van de website is besproken hoe de lay-out van de website eruit moet komen te zien.

Non Functional Software Requirements		
NFSR.5	The ui has a toolbar that is always on top	Must
NFSR.6	The ui has a navigation menu on the left of the page	Must
NFSR.7	The ui has a footer that is under the page content	Must

Tabel 8: UI Design requirements

Aan de hand van de voorbeelden die ik liet zien aan de stakeholders, is besloten een traditionele admin panel layout te maken. In figuur 5 is een schets van de admin panel te zien. Aan de linker kant van het scherm is het navigatie menu (sidenav, NFSR.6), dit menu is open en dicht te klappen met behulp van een knop linksboven in de toolbar. In desktopmodes zal de sidenav de pagina content naar rechts duwen, in de andere opties (tabled en mobiel) zal de sidnav over de pagina content getoond worden.

Verder zal op de toolbar (NFSR.5) twee knoppen verschijnen, een knop om te applicatie fullscreen te maken (toggle) en een knop om uit te loggen. De footer zal altijd aan de onderkant van de pagina komen te staan, onder de pagina content (NFSR.7). Ook als de pagina content niet scherm vullend is, moet de footer onderaan staan. Alleen de pagina content en de footer zitten in het gebied dat kan scrollen. De sidenav en toolbar zijn staties.

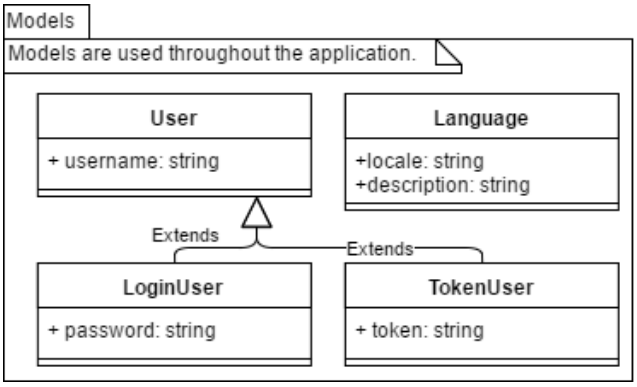


Figuur 5: Wireframe layout admin panel

9.3. Models

In de volgende hoofdstukken zal globaal de verschillende klassen diagrammen besproken worden. Deze diagrammen vormen de basis van de gehele app en zullen vaker aan bod komen in dit verslag.

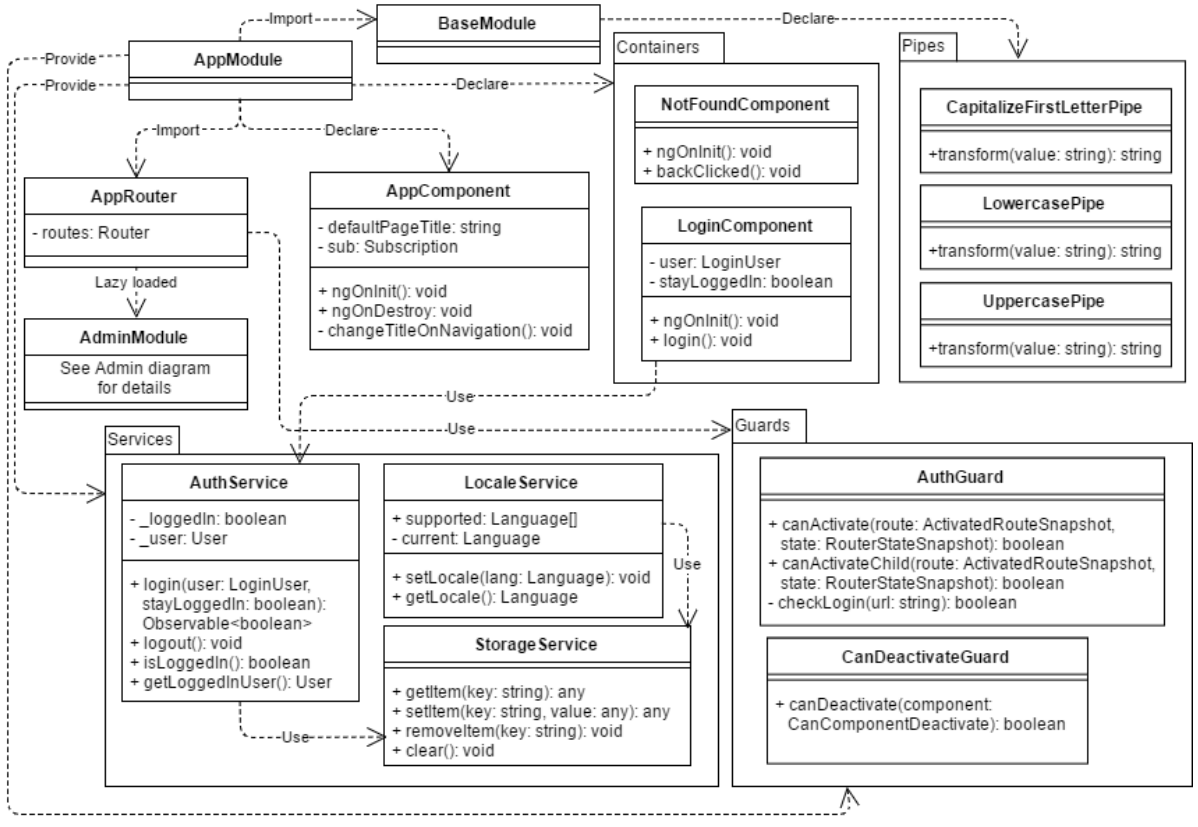
In Figuur 6 zijn alle models te zien die de basis van de app vormen. Ik had gelijk besloten de verschillende user objecten te scheiden, om zo gevoelige informatie niet te hoeven delen met componenten die dit niet hoeven te weten. Het TokenUser model is meer een placeholder op dit moment, omdat er nog geen beveiliging in de webserver zit. Wel is het zeker dat er met tokens gewerkt zal moeten worden, om zo user te authentiseren bij api calls. Op deze manier kan in een vroegtijdig stadium rekening gehouden worden met het beveiligings risico.



Figuur 6: App models klassen diagram

9.4. Root module

Figuur 7 geeft een overzicht van alle verschillende klassen weer, welke aanwezig zijn in de root van de app. Alles wat hierin staat is beschikbaar in de gehele applicatie. In dit diagram zijn alle klassen opgedeeld in de verschillende type klassen die Angular aanbiedt, de naamgeving van de klassen zijn hier ook op gebaseerd. De Angular klassen worden niet op een manier gebruikt (door Angular) waarbij er makkelijk een UML-diagram van gemaakt kan worden. Om deze reden zijn de klassen met elkaar verbonden hoe zo door de verschillende modules geïmporteerd en gedeclareerd worden. De admin module in dit diagram is niet uitgewerkt, deze staat uitgewerkt in het volgende diagram.



Figuur 7: App root klassen diagram

Zoals in figuur 7 te zien is bestaat de applicatie uit twee standaardpagina's, een login pagina en een not found (404) pagina. In het diagram staan deze in de containers package. De router zal de gebruiker automatisch naar het loginscherm verwijzen, maar zodra de user een niet bestaande url opgeeft zal een 404 error weergeven worden.

9.4.1. BaseModule

De app beschikt ook over een BaseModule, welke door elk ander module in de applicatie geïmporteert wordt. Deze module bestaat, zodat niet elke module de waslijst aan verschillende benodigdheden hoeft te importeren die alle nodig zijn om de app te runnen. Een module importeerd enkel de BaseModule en beschikt gelijk over alles dat in de BaseModule staat.

9.4.2. Authenticatie

De AuthService wordt op dit moment al gebruikt om zowel in te loggen als door de AuthGaurd. Op dit moment zet de AuthService enkel de interne state van de applicatie op ingelogd, zodra de login wordt aangeroepen. De AuthGuard checkt de login app state in de AuthService zodra de gebruiker naar de admin module probeert te gaan. Deze functionaliteit is gemaakt zodat de app al simuleert alsof de user daadwerkelijk inlogt.

9.4.3. Storage

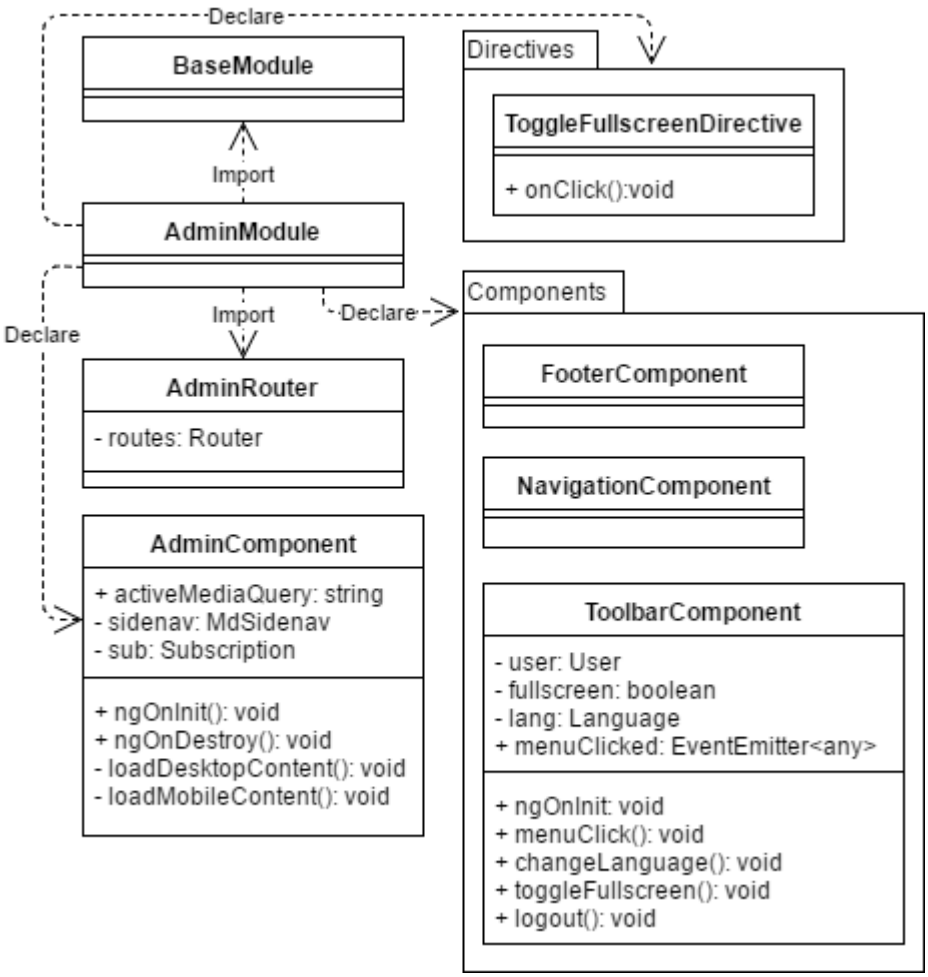
Er is ook een StorageService te zien. Deze service maakt gebruik van het window.localStorage object. Met dit object kunnen gegevens lokaal in de browser van de client opgeslagen worden. Deze gegevens zullen onthouden blijven, totdat de user de cache leegt. Op dit moment kan ervoor gekozen worden de login state hiermee lokaal in de browser te bewaren. Dit is natuurlijk niet permanent, maar wel erg handig tijdens development. Hierdoor hoef je als ontwikkelaar niet elke keer opnieuw in te loggen in combinatie met HMR.

9.4.4. Locale

De userinterface van de applicatie hoeft op dit moment alleen beschikbaar te zijn in het Engels. Wel is er een locale service opgezet, met de verantwoordelijkheid de locale te zetten in de applicatie. Uit ervaring weet ik dat lokalisatie/vertaling, toevoegen op een laat tijdstip in de ontwikkeling meer werk is dan als je hier vroeg mee begint. Om deze reden is ervoor gekozen gelijk in het begin gebruik te maken van ngx-translate. Ngx-translate is een library dat gebruikt kan worden om vertalingstabellen toe te voegen aan een Angular applicatie. Bij het opstarten van de applicatie zal de library een json bestand ophalen van de webserver waar alle ui strings in staan.

9.5. Admin Module

Figuur 8 is een uitbreiding op figuur 7, waar de AdminModule is uitgewerkt. Zoals in figuur 7 te zien is, is de AdminModule lazy loaded. Dat wil zeggend dat de module alleen van de server wordt opgehaald als dit nodig is, de router/guard is akkoord dat de gebruiker naar de admin pagina(s) mag.



Figuur 8: App admin lay-out klassen diagram

De AdminModule declareert zowel de footer, toolbar als de navigatie, welke worden gebruikt in de html van de AdminComponent. Op deze manier zijn de verschillende pagina's die onder de admin zullen vallen, zoals een dahboard, onafhankelijk van deze componenten. De componenten zullen altijd bestaan zolang de app binnen de admin module is. Angular zal deze componenten ook niet elke keer opnieuw hoeven te tekenen als de inhoudelijke pagina veranderd, wat de performance van de webapplicatie verbeterd.

9.5.1. AdminComponent

Het AdminComponent beschikt over twee load… methodes. Deze methodes bestaan zodat de instellingen rondom de navigatie goed worden gezet, zodra de app voor het eerst naar de admin pagina gaat. Deze instellingen bepalen o.a. of de content naar rechts wordt geduwd bij een openstaande navigatie.

9.5.2. Toolbar

In het ToolbarComponent is een EventEmitter te zien. De knop om de sidenav te openen en sluiten bevindt zich op de toolbar, maar de toolbar kan niet bij de navigatie komen. Om deze reden geeft de toolbar een event, welke wordt ondervangen in de AdminComponent. Dit component zal er vervolgens voor zorgen dat de navigatie geopend of gesloten wordt.

9.5.3. ToggleFullscreenDirective

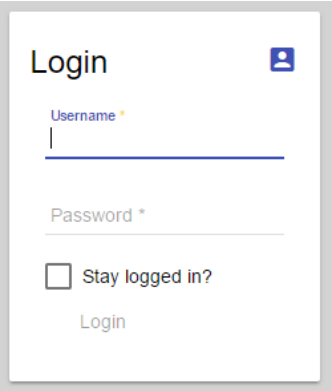
De ToggleFullscreenDirective is gemaakt, niet zozeer omdat dit een belangrijke requirement is, maar juist om ervaring op te doen in het maken van directives. Ik had nog nooit met Angular 2 gewerkt en het concept van directives was nieuw voor mij. Ik wil graag alle aspecten die Angular te bieden heeft kennen, om zo alle mogelijkheden van het framework te leren kennen.

9.6. Acceptatie

Nadat de applicatie was geïmplementeerd is een acceptatie review meeting gehouden met de stakeholders. Tijdens deze meeting waren de stakeholders positief over het bereikte resultaat, op de gebruikte kleuren en een te laag resolutie logo na. Tijdens het eerdere gesprek was er niks gezegd over de te gebruiken kleuren, dus dit was te verwachten. Ik had de standaard opmaak/thema van Angular Material gebruikt.

Ik had met de stakeholders afgesproken dat deze wensen in een toekomstige versie geïmplementeerd kan worden, maar dat dit naar mijn mening geen hoge prioritijd heeft. Deze sprint is dan ook als succesvol gezien en voldoet aan de defenition of done, de eisen waar de applicatie aan moest voldoen zijn verwerkt.

In figuur 9, 10 en 11 zijn screenshots weergeven van het resultaat.



Figuur 9: Login scherm

Figuur 10: Desktop admin dashboard scherm

Figuur 10 en 11 zijn twee screenshots van hetzelfde scherm, in een verschillende modus weergeven. (10 is desktop & 11 is mobiel/tablet) het verschil is de manier hoe het navigatiemenu opent. In de desktop modus zal het de content van de pagina naar links schuiven, terwijl het in mobiel/tabled juist over de content heen opent. Voor de rest lijken al deze afbeeldingen net of het een Android app is. Material design wordt ook op Android toegepast.

Figuur 11: Mobiel/tabled admin dashboard scherm

10. Globale navigatie

De volgende stap op de agenda, is het bespreken welke pagina's er zullen bestaan en wat de flow door deze pagina's zal zijn. Dit is geen volledige sprint van twee weken, maar is wel een afgezonderde doelstelling op de planning. Voor het uitwerken van de globale navigatie is één week gereserveerd.

10.1. Planning

Op dit punt van de afstudeerperiode wordt de originele planning gewijzigd. Op de planning staat dat vanaf nu gewerkt zal worden aan de datareparatie pagina's, maar in een gesprek met de stakeholders is besloten allereerst te focussen op error detectie. Ik wist dat er error detectie plaats moest vinden, om zo te bepalen welke data automatisch gerepareerd wordt, maar niet dat dit een grote afzonderlijke functie van de applicatie is. Ook is besloten voor nu te focussen op footfall apparaten en wifiscanners buiten de scope te laten. Voor wifiscanners is op dit moment nog veel onduidelijk, hoe deze data gerepareerd en gedetecteerd moet worden. Dit heeft geen effect op wat voor producten er opgeleverd zullen worden, alleen het uiteindelijke resultaat van de opdracht.

Week	Datum	Beschrijving
8	27-03-2017	Bespreken van de globale navigatie door de gehele applicatie.
9	03-04-2017	Opzetten van de webserver.
10	10-04-2017	Ontwikkelen van de error detectie algoritme.
12	24-04-2017	Performance test van het error detectie algoritme.
13	01-05-2017	Ontwikkelen van de error detectie pagina's.
14	08-05-2017	Ontwikkelen van de locatie en locatie detail pagina.
Indien tijd over		Ontwikkelen van de reparatie pagina

Tabel 9: Herziende planning

10.2. Requirements

De requirements in tabel 11 zijn uit twee gesprekken voortgekomen. Allereerst een gesprek waarin gebrainstormd is welke pagina's gemaakt moeten worden. Vanuit dit gesprek heb ik een aantal producten gemaakt, welke later behandeld zullen worden. Deze producten zijn vervolgens in een tweede gesprek gebruikt zodat iedereen een beeld had wat het plan is. Vanuit dit gesprek zijn de te maken pagina's en de navigatie tussen pagina's definitief gemaakt. De pagina's zelf moeten nog wel verder uitgewerkt worden, dit zal gebeuren zodra hieraan gewerkt wordt. Het is de bedoeling dat deze pagina's ontworpen worden met als doel het repareren van errors (NFSR.8). Hierdoor zal op de verschillende pagina's voornamelijk het aantal openstaande errors duidelijk moeten zijn.

User Requirements			
UR.4	UC.1	The user can see an overview of all errors	Must
UR.5	UC.1	The user can see an overview of all today errors	Must
UR.6	UC.2	The user can repair count data of footfall devices	Must
UR.7	UC.2	The user can repair count data of wifi devices	Should

Tabel 10: Navigatie requirements

Non Functional Software Requirements		
NFSR.8	The ui focusses on the amount of open (unresolved) errors	Must

Tabel 11: Navigatie requirements

10.3. Ondersteunende producten

Vanuit het gesprek met de stakeholders zijn twee ondersteunende producten opgesteld: een navigatiemap en mockups. Deze producten zijn bij een tweede gesprek over de globale navigatie gebruikt voor de beeldvorming van de stakeholders. Ook hebben deze producten mij geholpen in het begrijpen van de wensen van de stakeholders.

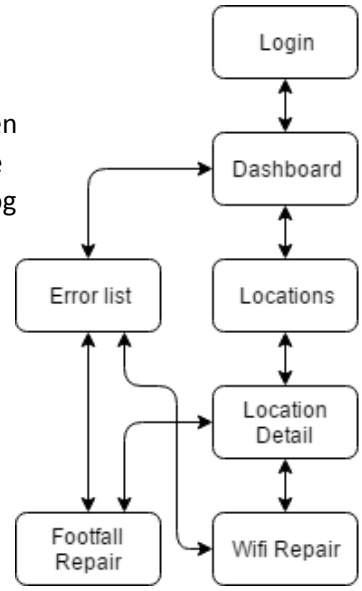
10.3.1. Navigatiemap

De navigatie door de applicatie is als volgt (figuur 12):

Zodra een gebruiker inlogt zal hij/zij naar het admin dashboard worden genavigeerd. Op het admin dashboard is een overzicht te zien van alle openstaande errors, zodat een gebruiker in één overzicht ziet of er nog reparaties moeten gebeuren.

M.b.v. het navigatie menu kan de gebruiker twee kanten op.

1. De error pagina.
   Vanuit de error pagina kan de gebruiker rechtstreeks naar de reparatie pagina, om de error te repareren.
2. De locatie pagina
   Vanuit de locatie pagina kan de gebruiker naar de locatie detail pagina, waarna naar een reparatie pagina gegaan kan worden.



Figuur 12: Navigatiemap

10.3.2. Error pagina

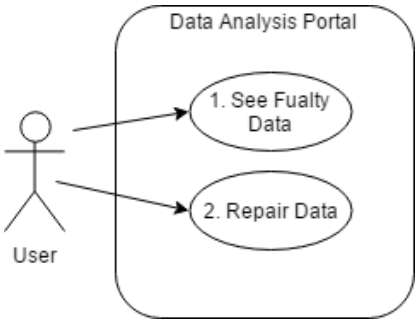
De error pagina bestaat uit een lijst van alle openstaande errors. In het navigatie menu staan twee knoppen om hier te komen: “Errors” en “Today Errors”. Het verschil tussen deze twee is de lijst van errors die getoond zal worden. De today errors geeft alleen een lijst van errors weer van gisteren en vandaag. De gewone error pagina toont een complete lijst van alle openstaande errors.

10.3.3. Locatie pagina

De locatie pagina bestaat uit een lijst van locaties, met het aantal openstaande errors. De gebruiker kan hier vervolgens een locatie kiezen, om zo een scherm te openen met meer detail. Deze detail pagina bevat een lijst van datums, waarop de locatie actief was met het aantal openstaande errors per datum. De gebruiker kiest vervolgens welke datum hij/zei wil repareren, waarna de reparatie pagina opent. Op deze pagina kan de gebruiker per apparaat per tijdsperiode (bijv. van 9:30 tot 10:00) de tel-data wijzigen.

10.4. Use cases

Vanuit de requirements zijn vervolgens de volgende use cases opgesteld, . Van deze use cases zijn ook use case beschrijvingen gemaakt. Deze beschrijvingen zijn niet compleet en zullen net als het diagram verder uitgewerkt worden, zodra de bijbehorende pagina’s ontwikkeld worden. De user requirements van tabel 11 bevatten UC.1 of UC.2, waaraan gezien kan worden bij welke Use Case het hoort.



Figuur 13: Use case diagram

11. Opzet webserver

Voordat aan het error detectie algoritme begonnen kan worden, moeten er meerdere dingen in de webserver gemaakt worden. Daar is deze sprint van één week voor bedoelt.

11.1. Requirements

De webserver moet meerdere type models gebruiken (NFSR.19 & 20). Het is op dit moment onzeker of de applicatie altijd Advantage II als database zal gebruiken. Mocht dit in de toekomst veranderen dan is het de bedoeling dat zo min mogelijk code herschreven hoeft te worden. Om deze reden is er besloten gebruik te maken van meerdere type models, meer in hoofdstuk 11.2.

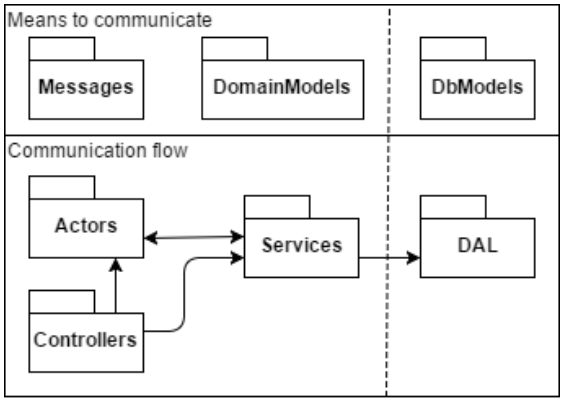
Non Functional Software Requirements		
NFSR.19	The server uses different types of models (database, domain)	Must
NFSR.20	The server uses DTO models	Should

Tabel 12: Webserver model requirements

11.2. Communicatiestroom

Het is belangrijk dat de webserver een lage koppeling heeft tussen de verschillende klassen en packages. In figuur 14 zijn de verschillende packages te zien, met de communicatie hiertussen.

- Controllers
  In controllers komen HTTP verzoeken binnen. Een controller mag alleen gebruik maken van een service, om data op te halen. Een controller kan gebruik maken van een actor, om bepaalde taken uit te laten voeren door het systeem, zoals de error detectie. Wanneer een HTTP verzoek binnen komt zal ASP.NET automatisch een object van de controller aanmaken. Dit gebeurd m.b.v. dependency injection.



Figuur 14: Webserver communicatie flow

- Actors
  Actors zijn afkomstig van Akka, waarin de error detectie wordt gemaakt. Het systeem en de actoren zelf kunnen met elkaar communiceren via berichten. In hoofdstuk 12.2 zal een voorbeeld gegeven worden hoe deze communicatie werkt. Actoren maken net als controllers gebruik van services om data op te halen.

- Services
  Services bestaan om de communicatie tussen het systeem en de database te regelen. Er zijn twee type models, domain models en database models. Database models worden door de data access layer (DAL) gebruikt voor de communicatie met de database. Binnen de webserver wordt gebruik gemaakt van domain models. Het is de taak van de service deze models om te zetten van database naar domain en vice versa.

- DAL
  De data access layer maakt gebruik van het entity framework om communicatie te leggen met de Advantage II database. Deze package beschikt enkel over de context waarmee verbinding gemaakt kan worden, een service krijgt de daadwerkelijke connectie.

11.3. Entity Framework Core

In het entity framework kunnen relaties worden vastgelegd in code. Deze relaties hoeven ook niet te bestaan in de database. In de applicatie heb ik meerdere van deze relaties vastgelegd, om zo gemakkelijker andere tabellen te kunnen bereiken. In het volgende voorbeeld zal een van deze relaties worden uitgelicht. Een locatie heeft een lijst van bijbehorende errors (één op veel).

In figuur 15 is de database model van een locatie te zien. Het laatste attribuut is een lijst van error database models genaamd LocErrors. In figuur 16 is te zien hoe de relatie is vastgelegd. Hier staat dat elk GenLocation object een lijst van errors heeft (HasMany). Voor elke van deze errors is er één locatie (WithOne). De error heeft een locatie ID (ErrLocID), waarmee de relatie gelegd kan worden (HasForeignKey).

```
public class GenLocation
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int LocId { get; set; }
    public string LocName { get; set; }
    public bool LocActive { get; set; }
    public virtual ICollection<GenErrors> LocErrors { get; set; }
}
```

Figuur 15: Locatie database model

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<GenLocation>()
        .HasMany(l => l.LocErrors)
        .WithOne(e => e.ErrLocation)
        .HasForeignKey(e => e.ErrLocId);
    ...
}
```

Figuur 16: EF locatie en error relatie vastlegging

11.4. Mapping van models (Automapper)

Om models om te zetten van het een naar het andere type object kan je zelf methodes schrijven. Dit geeft echter veel werk en kan slimmer opgelost worden. Ik had besloten gebruik te maken van een library genaamd automapper. Met deze library kan je relaties tussen models vastleggen en met behulp van een map een object omzetten. Het opzetten van de relaties werkt erg eenvoudig en is in figuur 17 te zien. In dit figuur wordt een database model GenErrors omgezet naar een domain model DetectedError. De eerste ForMember methode laat zien hoe de relatie ligt voor het ID attribuut. (d staat voor destination, o staat voor options en s staat voor source) Het DetectedError ID attribuut komt van de GenErrors ErrId.

De tweede ForMemeber methode werkt iets anders. Het probleem is dat het error type in GenErrors als byte wordt opgeslagen. In de domain models wilde ik graag gebruik maken van enums, voor deze vaste type errors. Voor deze mapping wordt gebruik gemaakt van een hulp methode te vinden in figuur 18. Deze methode zet het error type van het GenErrors model om in het enum type error, dat wordt gebruikt binnen het domain.

```
CreateMap<GenErrors, DetectedError>()
    .ForMember(d => d.ID, o => o.MapFrom(s => s.ErrId))
    .ForMember(d => d.Type, o => o.ResolveUsing(s => MapErrorType(s.ErrType)))
    ...
```

Figuur 17: Map van GenErrors naar detectedError

```
public static Models.ErrorType MapErrorType(byte type)
{
    switch (type)
    {
        case DAL.ErrorType.NoData: return Models.ErrorType.NoData;
        case DAL.ErrorType.PartlyZero: return Models.ErrorType.PartlyZero;
        case DAL.ErrorType.Deviation: return Models.ErrorType.Deviation;
        default: return Models.ErrorType.Unknown;
    }
}
```

Figuur 18: Map van Advantage error type naar domain error type

11.5. Service

Binnen een service heb ik besloten gebruik te maken van C# Task objecten. Door het gebruik van task objecten kan de data uit de database asynchroon worden opgehaald. Een object kan meerdere taken tegelijkertijd uitvoeren en vervolgens wachten totdat elke taak is uitgevoerd.

Ter ondersteuning hoe dit er in praktijk uitziet is in figuur 19 de GetLocation methode van de locatie service te zien. Deze methode haalt alle locaties uit de database op en geeft deze vervolgens terug in een lijst. In dit voorbeeld kan ook gezien worden hoe het locatie database model wordt omgezet in een domain locatie model.

Op regel 3 wordt het Task object aangemaakt, welke automatisch gestart wordt. Vervolgens wordt op regel 5, m.b.v. het data access layer (Context), alle locaties uit de database opgehaald. Elk van deze locaties wordt op regel 10 omgezet van een database model (GenLocation) naar een domain model (Location). Om vervolgens toegevoegd te worden aan de HashSet.

Binnen de services wordt voornamelijk gebruik gemaakt van HashSet's. Hier is voor gekozen, omdat een HashSet een constante performance heeft. In tegenstelling tot bijv. een normale List, welke een lineair stijgende lijn heeft, naarmate de lijst groter wordt. [12]

```
1  public async Task<ISet<Location>> GetLocations()
2  {
3      return await Task.Run(() =>
4      {
5          var genLocations = Context.GenLocation.Where(loc => loc.LocActive == true);
6          var locations = new HashSet<Location>();
7
8          foreach (GenLocation location in genLocations)
9          {
10             var model = Mapper.Map<GenLocation, Location>(location);
11             locations.Add(model);
12         }
13         return locations;
14     });
15 }
```

Figuur 19: Location service GetLocation methode

11.6. Dependency Injection (Autofac)

Er zijn maar drie dependency injection (DI) frameworks, welke door Akka ondersteund worden. [13] Eén van deze is Autofac. Er is geen speciale reden dat voor Autofac is gekozen, naar mijn weten hadden ze alle drie kunnen werken. De standaard Microsoft DI, waar ASP.NET gebruik van maakt, werkt i.i.g. niet in Akka. ASP.NET is te configureren dat het een andere DI framework gebruikt en werkt nu ook met Autofac.

Figuur 20 laat zien hoe een class geregistreerd wordt in Autofac. In dit voorbeeld wordt de LocationService geregistreerd als een interface (ILocationService). Alle services worden gebruikt als een interface. Dit om er verder voor te zorgen dat er binnen de applicatie geen afhankelijkheid van de Advantage database of bepaalde service ontstaat. Zodra de database vervangen wordt hoeft enkel een nieuwe versie van de service aangemaakt te worden, welke hetzelfde interface implementeert en registreert bij Autofac. Het strategy design pattern.

```
public class DIModule : Module
{
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterType<LocationService>().As<ILocationService>();
        ...
    }
}
```

Figuur 20: Dependency Injection setup

11.7. Logging (Serilog)

Een stuk code dat gemaakt is in Akka is lastig te debuggen. Je kan simpelweg een breakpoint zetten, maar omdat duizenden gelijke actoren tegelijkertijd uitgevoerd kunnen worden is het lastig te achterhalen waar je precies bent. Ik vind het makkelijker om de status van een actor naar bijv. de console te loggen en deze na afloop te doorlopen. Op deze manier kan achterhaalt worden welke actoren wel succesvol waren en welke niet.

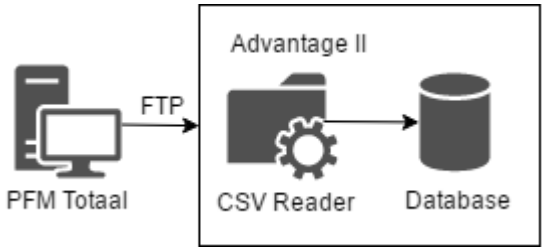
Een ASP.NET webapplicatie compileert naar een console applicatie, maar als deze applicatie wordt uitgevoerd in IIS (Internet Information Services), zal geen console openen. Alle berichten die een actor verstuurd zullen verloren gaan. Ik maak gebruik van Serilog een library dat naar een bestand op de disk kan schrijven. Serilog kan ook gebruikt worden i.c.m. Akka, waarmee vanuit actoren berichten verstuurd kunnen worden.

Deze library is gekozen, omdat er veel in is te configureren. Zo kan je bijv. de minimale log level per namespace instellen. Ook kan je gebruik maken van een rollingfile, welke per dag een nieuwe file aanmaakt en een maximale grootte heeft (geen logfiles van meer dan 1 GB).

12. Error detectie algoritme

Nu de webserver volledig is opgezet kan begonnen worden aan de error detectie zelf. Deze sprint zal twee weken duren, waarna de performance testen nog uitgevoerd moeten worden.

In Advantage II was al eerder een poging tot error detectie gemaakt, figuur 21. Vanuit PFM totaal worden csv bestanden over FTP verstuurd naar de Advantage II server. Deze server beschikt over een csv reader, welke periodiek checkt of er nieuwe csv bestanden zijn aangeleverd, om deze vervolgens in de database te verwerken. In deze csv reader vond de error detectie in plaats. De error detectie is lang geleden stopgezet, omdat de performance ver onder de maat was. Ik weet geen exacte cijfers, maar ik heb van Bart vernomen dat dit proces meerdere uren in beslag kon nemen.



Figuur 21: Advantage II error detectie

12.1. Requirements

In overleg met Bart en Chris is besloten de error detectie te baseren op de oude situatie, dezelfde type errors zullen gebruikt worden (Rule.7). Het voordeel dat dit met zich mee brengt is dat er geen database wijzigingen plaats hoeven te vinden. In tabel 13 staan een aantal requirements, welke later aan bod komen.

Business Rules		
Rule.4	Data is deviated when it is more than 20 % of the average.	Must
Rule.5	The average is calculated from the last 175 days	Must
Rule.6	The average can only use data from the same days of the week (example: Only mondays)	Must
Rule.7	The footfall error types are no data, partly zero and deviation	Must

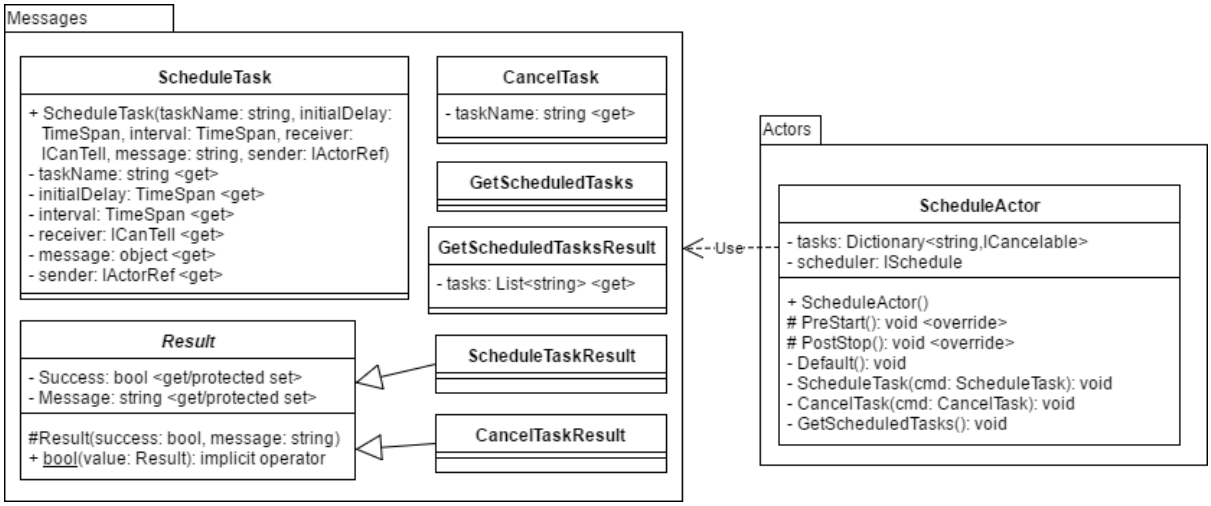
Business Requirements		
BR.9	The error detection algorithm runs periodically on the webserver	Should

Tabel 13: Error detectie requirements

12.2. Planner

Error detectie moet periodiek aangeroepen worden (BR.9). Dit heeft voor nu geen hoge prioriteit, maar kan later wel een grote impact hebben op de applicatie. Ik had besloten dit probleem gelijk aan te pakken, zodat dit geen risico wordt. Het periodiek aanroepen is met behulp van de Akka scheduler geïmplementeerd. De Akka scheduler kan periodiek of na een bepaalde wachttijd een bericht naar een actor versturen. De scheduler geeft vervolgens een CancelationToken terug voor het geval het bericht gecanceld moet worden.

In mijn applicatie heb ik een speciale ScheduleActor geïmplementeerd (figuur 22). Met behulp van deze actor kan vanuit het hele systeem, dat toegang heeft tot de actoren, nieuwe taken gepland en bestaande taken gecanceld worden. Het probleem dat je hiermee oplost is het opslaan van de CancelationTokens. Ergens in memory moeten deze objecten worden bijgehouden. In de messages package staan alle berichten, die door de schedule actor worden gebruikt.



Figuur 22: Schedule actor klassen diagram

12.2.1. Actor berichten

De actoren waar ik gebruik van maak zijn van het type ReceiveActor. De berichten waar dit type actor mee communiceert zijn eigenlijk gewoon klassen, welke niet verplicht zijn attributen te bevatten. De berichten die naar en van een actor worden verstuurd, worden door Akka geserialiseerd en gedeserialiseerd naar en van JSON.

Ter context hoe de berichten en de afhandeling hiervan werkt, is hieronder een simpel voorbeeld. Figuur 23 geeft de bijbehorende code weer van de gebruikte berichten. Deze zijn ook te zien in het klasse diagram van figuur 22. In dit voorbeeld wordt een lijst van taken opgehaald uit de schedule actor. De bijbehorende stappen staan in figuur 24. In dit voorbeeld is de schedule actor zelf al eerder aangemaakt.

```
public class GetScheduledTasks { }
public class GetScheduledTasksResult
{
    public GetScheduledTasksResult(List<string> tasks) { Tasks = tasks; }

    public List<string> Tasks { get; private set; }
}
```

Figuur 23: Get scheduled tasks actor berichten

```
1 var schedule = system.ActorSelection(ActorPaths.Schedule.Path);
2 var message = new GetScheduledTasks();
3 var task = schedule.Ask<GetScheduledTasksResult>(message);
4 task.Wait();
5 List<string> tasks = task.Result.Tasks;
```

Figuur 24: Ophalen van de lijst van taken uit de schedule actor

1. Krijg een referentie van de actor.
   Je mag NOOIT direct een relatie met een actor hebben, enkel een referentie daarnaartoe. In deze stap wordt een referentie aangemaakt naar de locatie van de schedule actor.
2. In deze stap wordt het bericht aangemaakt, welk van het type GetScheduledTask is.
3. Verstuur het bericht.
   Er zijn twee manieren om te communiceren Aks en Tell. In het voorbeeld wordt een Ask gebruikt. Met een Ask krijg je een Task object terug waarin het antwoord van de actor staat.
4. Wacht totdat de actor klaar is met werken.
5. Lees het antwoord van de actor. Deze is van het type GetScheduledTasksResult.

Figuur 25 laat zien wat de schedule actor doet. In de default methode (de default state) staat welke berichten worden afgehandeld. Er worden drie type berichten afgehandeld: ScheduleTask, CancelTask en GetScheduledTaks. In het laatste geval wordt de GetscheduledTasks methode aangeroepen. Deze methode geeft een bericht terug aan de zender van het type GetScheduledTasksResult. In dit object wordt een lijst van strings meegegeven, welke als key worden gebruikt in de dictionary genaamd tasks.

```
private void Default()
{
    Receive<ScheduleTask>(cmd => ScheduleTask(cmd));
    Receive<CancelTask>(cmd => CancelTask(cmd));
    Receive<GetScheduledTasks>(cmd => GetScheduledTasks());
}
private void GetScheduledTasks()
{
    Sender.Tell(new GetScheduledTasksResult(tasks.Keys.ToList()));
}
```

Figuur 25: Schedule actor code snipped

12.3. Ontwerp

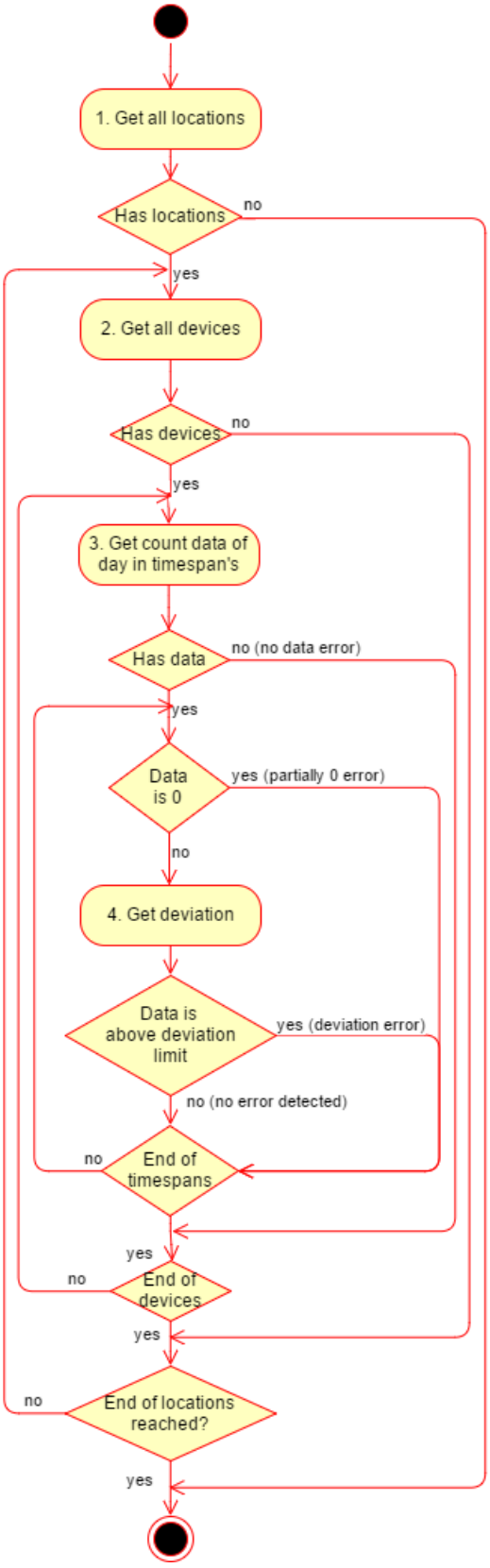
Er zijn veel ontwerpen gemaakt voor mijn beeldvorming hoe het algoritme geïmplementeerd kan worden. Deze diagrammen gaan steeds een laag dieper, om uiteindelijk tot een goed ontwerp te komen.

12.3.1. Flowchart

De eerste stap in het ontwerp is het in kaart zetten van de stappen die gemaakt moeten worden, om errors te kunnen detecteren. Deze stappen zijn als volgt:

1. Haal alle locaties op
2. Voor elke locatie haal de apparaten op.
3. Voor elk apparaat haal de data op.
   Alle tellingen zijn geaggregeerd in tijdspannen, van stadaart 30 minuten.
   a. Als voor één gehele dag geen data is, is er een ‘no data’ error.
   b. Als alleen een tijdspanne geen data heeft, is er op dat tijdslot een ‘partly zero’ error.
4. Per bestaande tijdspanne bereken de deviation. De diviation wordt berekend aan de hand van het gemiddelde dat, dat apparaat, op dat tijdslot, van die dag van de week, van de afgelopen 175 dagen heeft geteld. Diviation is dus het percentage dat de data afwijkt ten opzichte van het gemiddelde.
   a. Als de deviation groter of gelijk is aan 20%, is er een ‘deviation error’ op dat tijdslot.

In de huidige situatie zal de error detectie algoritme alleen checken voor errors op de dag voor vandaag.



Figuur 26: Flowchart error detection

12.3.2. Actor schaalbaarheid

In de flowchart zijn drie voor elke (foreach) acties te zien. Voor elke locatie, voor elk apparaat en voor elke tijdspanne (deviation berekening). Deze drie acties kunnen onderverdeeld worden in actoren. Je krijgt dan een locatie actor, een footfall actor (de scope is footfall devices) en een deviation actor.

Er zijn twee manieren waarop deze actoren schaalbaar gemaakt kunnen worden.

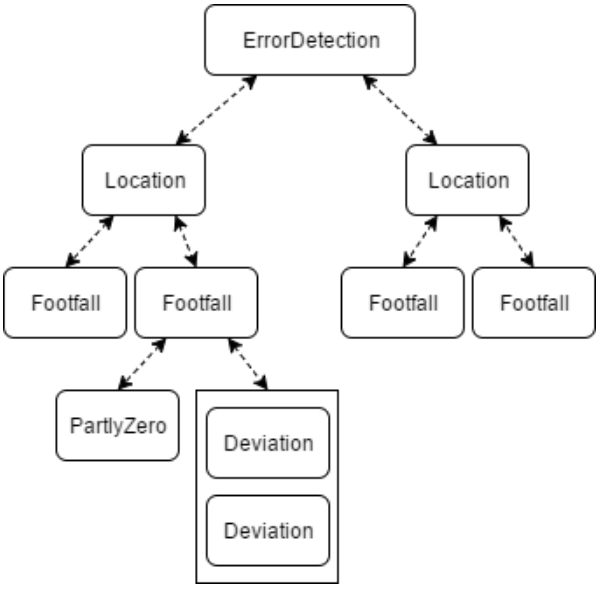
1. Gebruik maken van een router. (behandeld in 8.1.3)
2. Voor elk, bijv. locatie, een nieuwe actor aanmaken.

In figuur 26 is weergeven hoe het systeem met de business mee kan groeien.

Boven in het figuur staat de ErrorDetectie actor. Dit is de hoofd actor van het proces, waar de error detectie mee gestart kan worden. Deze actor zal voor elke locatie een Location actor aanmaken. (Elke locatie zal gelijktijdig checken voor errors)

Een Location actor zal voor de bijbehorende footfall devices een Footfall actor opstarten. (Elke footfall zal gelijktijdig checken voor errors)

Als de error detectie gestart wordt zullen er na het aanmaken van de Location en Footfall actoren al meer dan 5000 actoren actief zijn. Om het algoritme gedeeltelijk te limiteren, zal niet voor elke tijdspanne een actor aangemaakt worden.



Figuur 27: Actor schaalbaarheid

Een Footfall actor heeft één PartlyZero actor. Deze actor is verantwoordelijk voor het vinden van alle partly zero errors. Deze error check kan uitgevoerd worden door één actor, omdat het een relatief simpele loop door de tel-data is.

Een footfall device heeft meerdere Deviation actoren, welke gebruik maken van een router. De hoeveelheid Deviation actoren die per Footfall actor kunnen bestaan, is in te stellen in de configuratie.

De opzet hoe ik de actor indeel heeft als voordeel dat functionaliteit zo veel mogelijk wordt onderverdeeld in actoren. Dit zorgt ervoor dat het algoritme beter met fouten om kan gaan, hoofdstuk 8.1.4. Wanneer er iets fout gaat in bijv. de Deviation actor, zal enkel deze actor opnieuw opgestart moeten worden. De rest van het algoritme is hier onafhankelijk van. Mocht er in de toekomst nieuwe type errors toegevoegd worden, dan kunnen deze net als de PartlyZero of Deviation actor, aan het algoritme worden toegevoegd.

Een nadeel van deze opzet is de hoeveelheid query's. In deze opzet zal elke locatie apart de footfall devices ophalen, welke weer apart de tel-data ophaalt. Dit kan al snel oplopen tot erg veel query's die de database tegelijkertijd moet verwerken. Dit zijn echter wel kleine query's die over één tabel gaan.

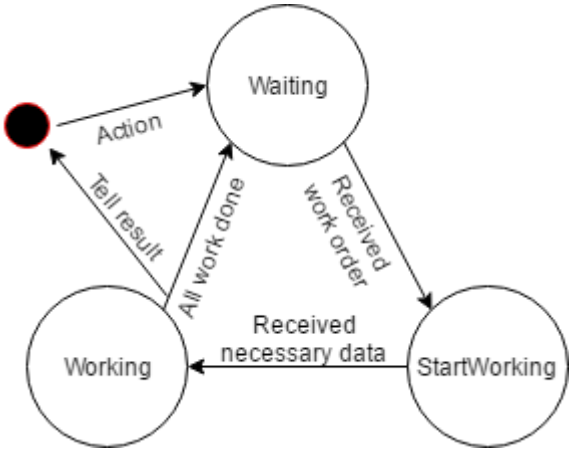
12.3.3. Actor state

Nu bekend is welke actoren er bestaan in het algoritme moet gekeken worden hoe de actoren zelf functioneren. In 12.2.1 is een voorbeeld weergeven hoe een actor berichten afhandelt. In het voorbeeld had de ScheduleActor alleen een Default state. De actoren in het error detectie algoritme hebben er drie. Elk van deze states heeft een eigen set van berichten welke afgehandeld kunnen worden. Zodra een actor een bericht krijgt, welke niet afgehandeld kan worden, zal dat bericht verloren gaan. Figuur 28 laat een generiek actor state diagram zien.

Zodra een actor wordt aangemaakt zal deze in de Waiting state komen. In deze state wacht de actor totdat hij iets te doen krijgt.

Zodra de actor een opdracht krijgt (bijv. start de error detectie), zal hij naar de StartWorking state gaan. In deze state wacht de actor totdat de benodigde data uit de database opgehaald is.

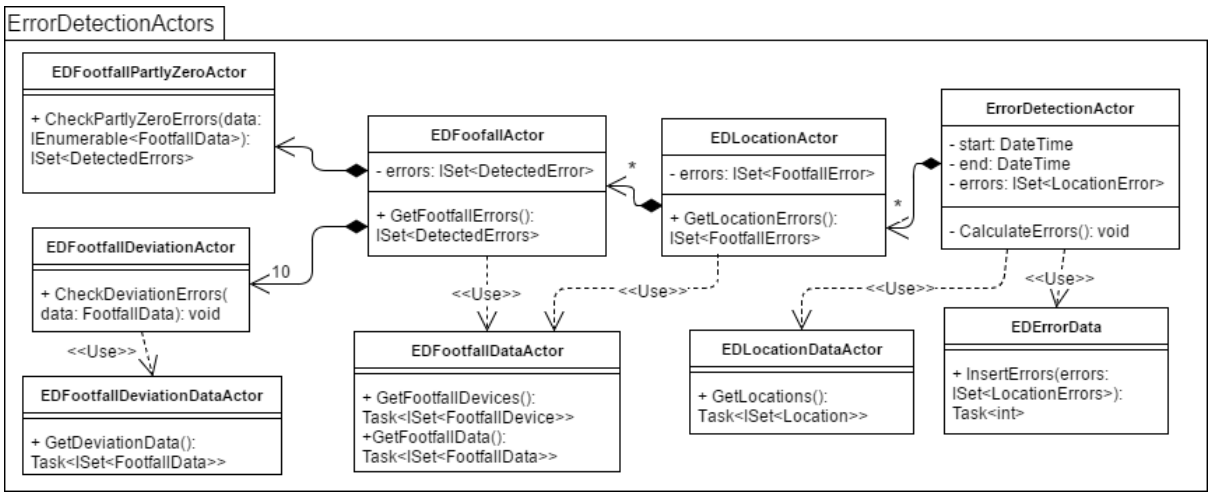
Wanneer de benodigde data is opgehaald kan de actor aan het werk gaan. Zodra de actor klaar is met werken wordt het resultaat teruggegeven aan de originele zender (de opdrachtgever van het start bericht)



Figuur 28: Actor state diagram

12.3.4. Class diagram

De schaalbaarheid is terug te zien in het klassen diagram, figuur 29. De error detectie actor heeft een lijst van locatie actoren, welke weer een lijst van footfall actoren heeft. De footfall actor heeft ook een vaste hoeveelheid deviation actoren. In het diagram zijn ook 'data' actoren te zien. Deze actoren zijn verantwoordelijk voor de interactie met de services van het systeem. Het wachten op deze actoren gebeurd in de StartWorking state.



Figuur 29: Error detection klasse diagram

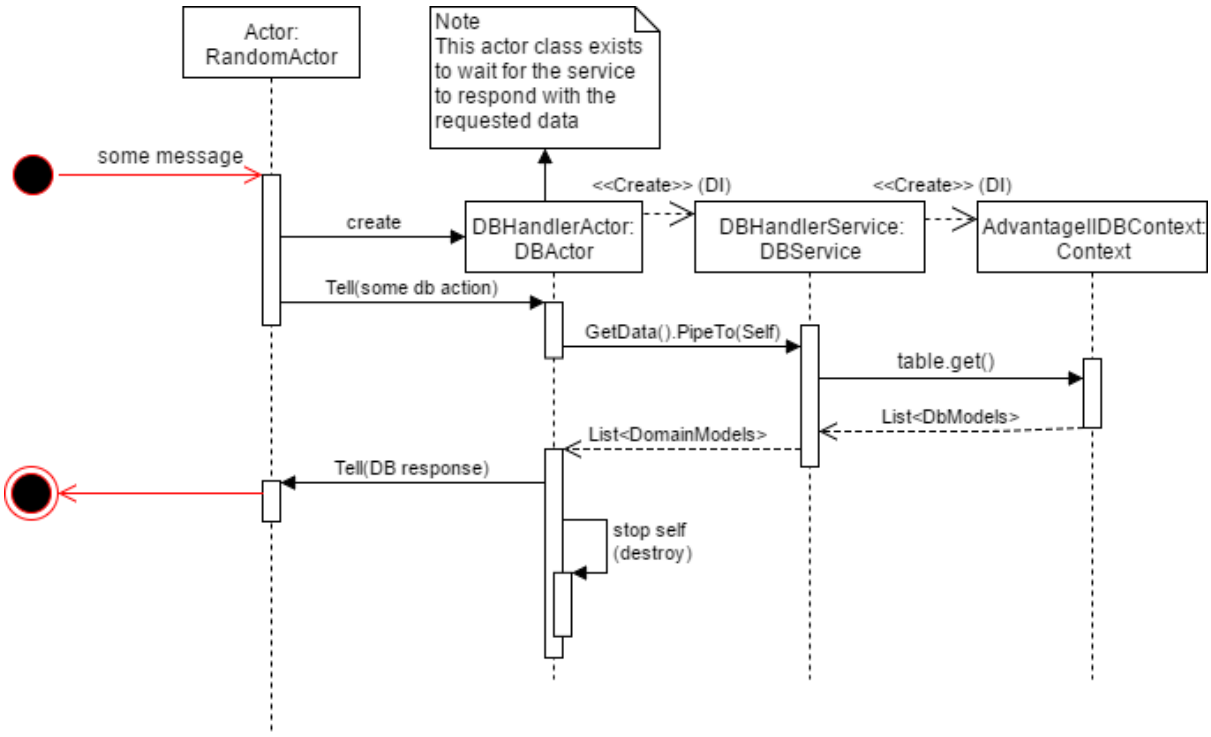
12.3.5. Sequence diagram

12.3.5.1. Database calls

Een actor kan data uit een service nodig hebben. Indien dit het geval is, zal het bevragen van de service in een speciale actor uitgevoerd worden. Dit wordt gedaan, om het algoritme nog fout toleranter te maken. Mocht er een fout ontstaan binnen de service, zal deze in de toegewezen actor afgehandeld kunnen worden.

Het bevragen van de database/service is weergeven in figuur 30. In dit generieke voorbeeld zijn de verschillende lagen van het systeem te zien. Het begint bij de RandomActor. Deze actor krijgt een bericht/actie binnen, waar extra data voor nodig is. Om deze data op te halen zal een HandlerActor worden aangemaakt, welke een handlerService geïnjecteerd krijgt door het dependency injection framework. Vanuit de HandlerActor wordt de service bevraagd. Het antwoordt van de service wordt naar de HandlerActor verstuurd (de PipeTo methode).

De service krijgt een context/connectie geïnjecteerd van de database. Met dit context object kan een select query uitgevoerd worden, waaruit een lijst van database models komt. De service zet deze models om naar domain models en geeft deze lijst aan de HandlerActor. Zodra de HandlerActor het resultaat heeft ontvangen, zal deze een bericht terugsturen naar de Random actor met het resultaat. Tot slot zal de HandlerActor zichzelf stoppen, om te vorkomen dat er memory leaks ontstaan. De laatste stap is niet verplicht als de HandlerActor als kind is aangemaakt van de RandomActor. Als dit het geval is zal de HandlerActor automatisch stopgezet worden, zodra de RandomActor afsluit.

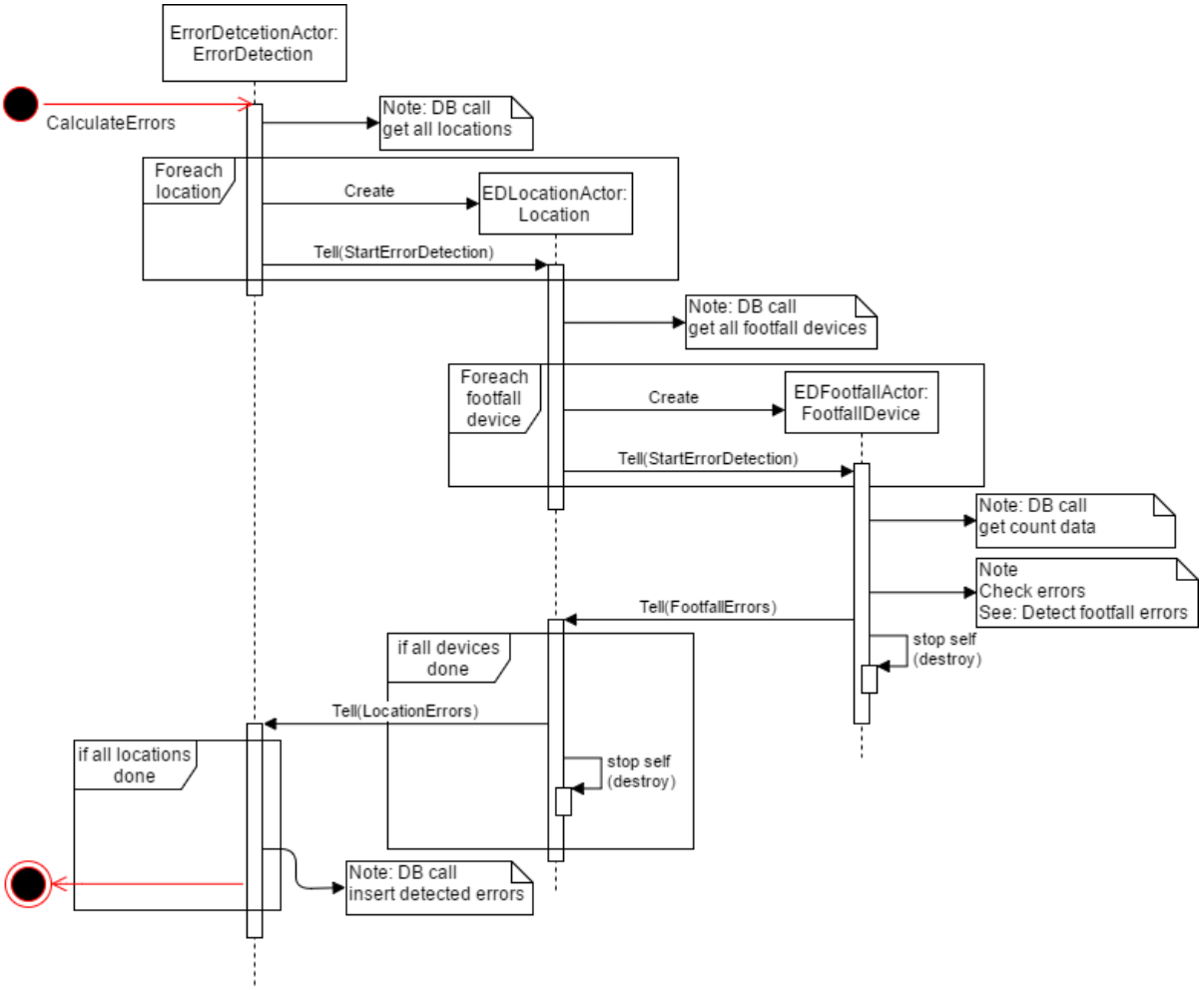


Figuur 30: Sequence diagram database calls

12.3.5.2. Error Detectie

Ter verdere ondersteuning is een sequence diagram gemaakt, dat weergeeft hoe de error detectie opstart (figuur 31). De error detectie actor zal door het systeem worden aangemaakt, zodra de webserver opstart, deze actor zal altijd leven. De error detectie kan gestart worden met een CalculateErrors bericht, waarna de locaties worden opgehaald en de locatie actoren aan het werk

gaan. Als een locatie actor alle errors gevonden heeft, zal het de errors teruggeven aan de ErrorDetectie actor. De error detectie actor is verantwoordelijk voor wat er met de gevonden errors gebeurd. In de huidige situatie zal het alle errors in één batch verwerken in de database.



Figuur 31: Sequence diagram error detectie

12.4. Unit testen

Voor de webserver is het van belang dat de data die verstuurd wordt richting de databases goed aankomt en intern verwerkt wordt. Om deze reden is de focus van de unittesten gericht op de services waar de webapplicatie gebruik van maakt. Naast de services worden ook de actoren binnen in de error detectie gecontroleerd. De unittesten zijn gemaakt met Visual Studio testtools en kunnen m.b.v. Visual Studio uitgevoerd worden.

12.4.1. Test setup

De webserver heeft meerdere dependencies welke opgezet moet worden, voordat een unittest uitgevoerd kan worden. Om dit proces zo gebruiksvriendelijk mogelijk te maken is er een Setup class aanwezig. Bij het uitvoeren van een test moet ervoor gezorgd worden dat van deze class de Initialize methode is aangeroepen. Zodra dit het geval is zorgt de class dat alles binnen de webserver getest kan worden.

De webserver maakt gebruik van de Advantage II database. Een groot deel van de webapplicatie is hiervan afhankelijk. Er is ervoor gekozen deze database te mocken in sqlite, om zo een consistent test dataset te hebben, welke niet te groot is.

12.4.2. Error detection algorithm

Het error detection algoritme is gemaakt m.b.v. Akka.NET, om deze reden kan het niet op een traditionele manier ge-unittest worden. Vanuit een unit test kan de inhoudelijke data van het algoritme niet bekeken worden. Ook kunnen individuele methodes van een Actor niet aangeroepen worden. Je kan enkel een bericht sturen en bekijken wat het antwoordt is van de actor. De error detectie is zo opgezet dat een actor altijd antwoordt met de gevonden errors. Hierdoor zijn verschillende datasets meegegeven aan de EDFootfallActor, zodra de actor antwoord geeft kan gekeken worden of de verwachte errors gevonden zijn.

12.5. Performance test

De performance testen behoren niet tot de error detectie sprint. Deze zijn in een volgende sprint van één week uitgevoerd. Wel is het onderdeel van het error detectie algoritme.

De performance testen zijn alle uitgevoerd op een test server met soortgelijke specs als op de live omgeving. De testomgeving maakt gebruik van een test database welke een exacte copy is van de live database. Tijdens de uitvoering van het algoritme is m.b.v. Windows taakbeheer en resource monitor de gebruikte hardware resources geobserveerd.

De tijden die gemeten worden komen direct uit de actoren. Elke actor bezit een C# StopWatch object, zodra de actor start wordt de stopwatch aangezet en zodra de actor klaar is wordt de stopwatch stopgezet. De resultaten van de stopwatch worden verstuurd naar een speciale performance overview actor. Deze actor is verantwoordelijk om alle tijden bij te houden en zodra de error detectie klaar is, de resultaten te printen in de log.

Om het opstarten van de error detectie gemakkelijker te maken, is er een speciale development pagina beschikbaar op de website. Op deze pagina kan de error detectie gestart worden en zodra deze klaar is, zullen de resultaten op de pagina getoond worden.

12.5.1. Scenario’s

De performance testen zijn uitgevoerd met verschillende scenario’s (tabel 14), oplopend in het aantal locaties. Elk van deze scenario’s zal vijf keer uitgevoerd worden. Van deze resultaten zal vervolgens een gemiddelde afgeleid worden, om eventuele beschikbare hardware resource fluctuaties zo goed mogelijk eruit te kunnen filteren.

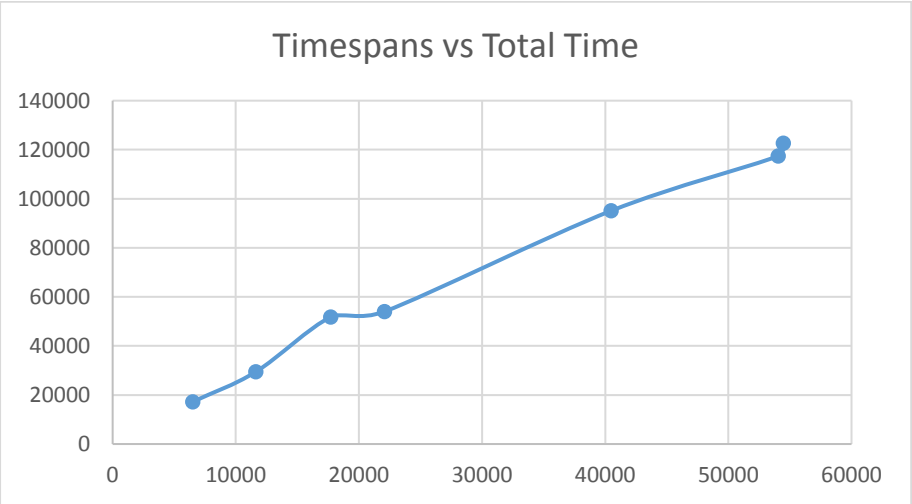
	Locaties	Footfall devices	Footfall met data	Time spans to check	Errors
1.	10	272	178	6516	3819
2.	25	702	354	11620	9418
3.	50	1346	564	17710	16524
4.	100	2049	700	22078	20804
5.	200	3131	1208	40480	33691
6.	300	3689	1548	54042	38834
7.	314	3745	1558	54474	38948

Tabel 14: Error detectie performance test scenario's

12.5.2. Resultaten

Het belangrijkste verschil tussen de verschillende scenario’s is de hoeveelheid data dat gecheckt moet worden voor errors. Dit is het gedeelte van het systeem dat de meeste tijd in beslag neemt. In

figuur 32 is een grafiek weergeven met het gemiddelde resultaat. In deze grafiek is de hoeveelheid tijdspannes (die gecheckt moeten worden) tegenover de totale tijd uitgebeeld.



Figuur 32: Tijdspanne (data dat gecheckt moet worden) tegenover de totale tijd

Het maakt niet uit hoe de resultaten op dit moment met elkaar vergeleken worden er is altijd een lineaire stijging te zien, zoals in figuur 32. Uit deze lineaire stijging kan geconcludeerd worden dat het hier om een O(N) algoritme gaat.

12.5.2.1. Resources

De observatie van de processor en ram geheugen is uitgevoerd tijdens het uitvoeren van scenario 7. De 0 sec markering is de status van de webserver voordat het error detectie algoritme is opgestart, de idle state. De webapplicatie maakt gebruik van een in memory cache, waar o.a. het entity framework gebruik van maakt. Hierdoor zal het ram gebruik snel omhoog gaan en actief blijven, het wordt mogelijk niet opgeruimd.

Zoals in tabel 15 is te zien gebruikt de webserver vanuit zichzelf niet veel resources, wel loopt het gebruik van resources snel op nadat de error detectie gestart is. Op peak momenten zijn er 44 threads actief, hoewel het niet zeker is hoeveel van deze threads actief bezig zijn met het detecteren van errors kan wel vrij zeker afgeleid worden dat het er in ieder geval 20 zijn. De CPU heeft op de testserver 4 cores, het is mogelijk dat Akka meer threads aanmaakt in het geval dat het aantal cores omhoog gaat.

	CPU Usage (%)	Threads	RAM (MB)
0 sec	0	18	77
10 sec	20	44	700
30 sec	11	43	1536
60 sec	33	41	1851
90 sec	38	43	1946
120 sec	27	44	981
150 sec	5	43	974
180 sec	0	25	974
210 sec	0	18	974

Tabel 15: Hardware resource observatie resultaten

De CPU usage is over het algemeen laag, rond de 30%, maar deze resultaten geven niet het gehele beeld. Het was duidelijk te zien dat op piekmomenten de CPU wel 90 tot 100 procent bereikte, dit heeft vermoedelijk te maken met de wachttijd van de database. Het kan voorkomen dat de database op bepaalde momenten veel deviation data query's behandelt en retourneert. In dit geval zal de deviation error checks veel actoren gelijktijdig aan het werk hebben, wat pieken veroorzaakt.

Tijdens de error detectie is goed te zien dat het geheugen gebruik oploopt, met ongeveer 2 gigabyte. Aan de hand van de status nadat de error detectie klaar is (210 sec) is af te leiden dat bijna 1

gigabyte daarvan in gebruik is door de in memory cache. Als de actoren zelf van het error detectie algoritme 1 gigabyte in gebruik nemen voor alle interne communicatie, is dit geen probleem. De server heeft meer dan genoeg RAM en meer RAM is makkelijk toe te voegen.

12.5.3. Verwachtingen

Op dit moment heeft zowel de actor werktijd als de query wachttijd een lineaire stijging, toch verwacht ik dat dit beter kan. Mijn verwachting is dat de actoren beter moeten kunnen schalen, waardoor de tijd dat de actoren bezig zijn (werken) minder zal worden. Met Akka kunnen applicaties gemaakt worden, welke goed van alle CPU cores gebruik kunnen maken, mijn verwachting is dan ook dat de huidige processor niet genoeg cores heeft om dit effect te zien. Huidige servers kunnen gemakkelijk meer dan 20 CPU cores hebben. Het zou ook kunnen dat Akka meer threads nodig heeft (huidig is 20), met meer threads kunnen meer actoren tegelijk actief zijn op de processor. Alle actoren die het algoritme gebruikt zitten vast aan deze twintig threads, met meer threads kunnen meer actoren daadwerkelijk werken.

12.5.4. Conclusie

Het algoritme heeft een duidelijke lineaire stijging. Dus aan de hand van deze resultaten kan gezegd worden dat het algoritme O(N) tijd in gebruik neemt. Het feit dat het algoritme er minder dan drie minuten over doet bij deze hoeveelheid data is al erg goed en is beter dan verwacht. De CPU usage is wel zorgwekkend, niet omdat het gemiddeld 30% is, maar juist omdat er pieken in zitten. Wel is het belangrijk dat er meer tests uitgevoerd worden. Deze tests moeten draaien op een betere server met meer CPU cores. Ook is het belangrijk dat de dataset veel uitgebreider wordt.

12.6. Toekomstige optimalisatie

In de toekomst kan de code beter geoptimaliseerd of gelimiteerd worden.

- Gebruik van streams
  Met streams kunnen data calls naar de database gelimiteerd worden, hiermee zal er minder druk op de database liggen. [14]

- Gebruik van dispatchers
  Op dit moment gebruikt het algoritme zo goed als alle cpu resources op peak momenten. Mogelijk wil je resources overhouden voor belangrijkere taken. Een displatcher kan toegewezen worden aan een groep van actoren. Met dispatchers kan je limitaties op de threadpool zetten. Hiermee kan je het aantal beschikbare threads instellen en hoelang elke actor mag werken, om zo elke actor een kans te geven iets te doen. [15]

- Router grootte
  De deviation heeft per footfall device 10 deviation actoren, dit is variabel. Ook kan de code gedeeltelijk herschreven worden om een limitatie te zetten op aantal locaties en footfall actoren. Op dit moment wordt voor elke locatie en apparaat een nieuwe actor opgestart. Met behulp van een router kan dit ook gelimiteerd worden tot een vaste hoeveelheid.

13. Error pagina’s

Nu het systeem errors kan detecteren is de vraag hoe deze errors weergeven kunnen worden aan de eindgebruiker. In deze sprint zijn zowel de error pagina’s als de admin dashboard gemaakt. Deze sprint was in totaal één week. Tijdens het opzetten van de globale navigatie is al eerder met de stakeholders besproken wat de verschillende pagina’s doen en kunnen. In deze sprint zullen de eerder en nieuwe achterhaalde requirements verwerkt worden tot een product.

13.1. Requirements

De belangrijkste requirements uit de interviews met de stakeholders m.b.t. de error pagina's zijn weergegeven in tabel 16. Alles beschreven in hoofdstuk 10.3.2 is nog van toepasing. Een toevoeging hierop is dat de 'today errors' en gewone 'errors' pagina's meerdere tabs zullen bezitten. Met deze tabs kan de gebruiker switchen tussen apparaten, waarvoor errors worden gedetecteerd, zoals footfall devices en scanners (NFSR.16). Op dit moment zal alleen de footfall tab daadwerkelijk inhoud hebben. Op het admin dashboard ziet de gebruiker de hoeveelheid openstaande errors (NFSR13). In 13.5 kan gezien worden hoe dit eruit ziet in het gemaakte product.

User Requirements			
UR.9	UC.3	The user can filter errors by error type	Should

Non Functional Software Requirements		
NFSR.13	The admin dashboard shows the open today and total of all footfall errors and footfall error types	Must
NFSR.16	The (today) error page uses deferent tabs for footfall, wifi and kpi errors	Must
NFSR.17	The (today) error page shows a list of all errors with the timestamp of the error, location name, device name and error type	Must
NFSR.18	The today error page only shows errors from today and yesterday	Must

Tabel 16: Gedetecteerde errors pagina requirements

13.2. Use case

Het use case diagram was al eerder weergeven (figuur 13), deze is nog steeds van toepassing. Nu is wel de use case beschrijving verder uitgewerkt, welke te zien is in Tabel 17: Bekijk gedetecteerde errors use-case beschrijvingtabel 17. Deze use case beschrijft wat het systeem en de gebruiker moeten doen, om de lijst van errors te tonen. Nadat deze use case is uitgevoerd kan de gebruiker de weergegeven errors filteren op error type (UR.9).

Name	See faulty Data
ID	UC.1
Description	Show a list of all faulty data
Primary Actor(s)	User
Secondary Actor(s)	-
Precondition(s)	1. The user is logged in
Main Flow	1. The user clicks on the "Errors" navigation item in the sidenav 2. The client navigates to the error page 3. The client requests all errors from the rest api 4. the rest api gets all errors from the Advantage II database 5. The rest api sends all errors to the client 6. The client shows the list of errors to the user
Postcondition(s)	1. the application show a list of all detected errors
Alternative Flow	1a. Today errors 1. The user clicks on "Today Errors"

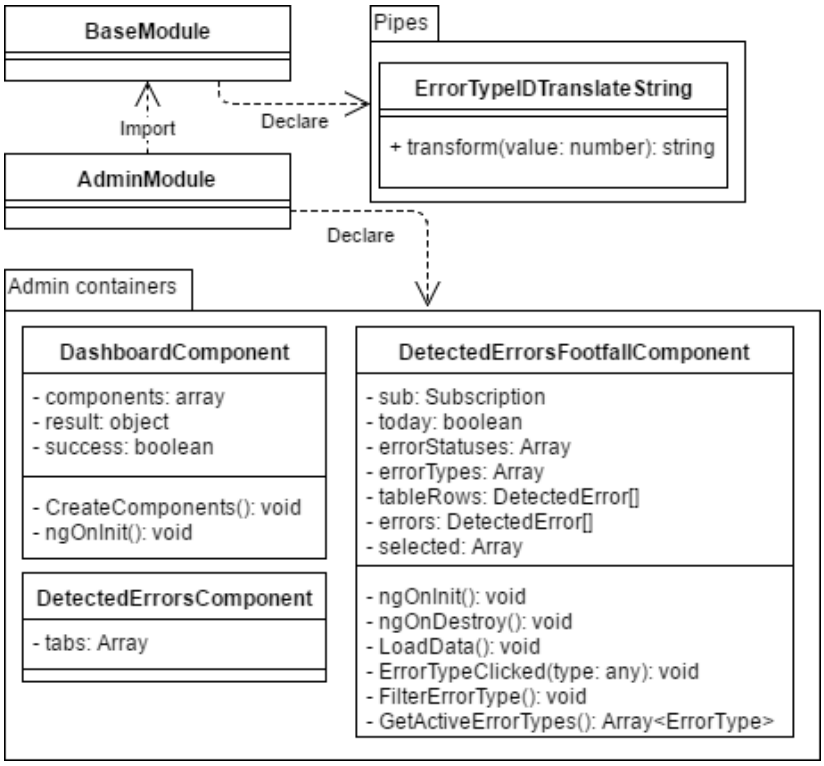
Tabel 17: Bekijk gedetecteerde errors use-case beschrijving

13.3. Ontwerp

Het gemaakte klasse diagram in figuur 33 is een uitbreiding op de eerder besproken admin lay-out klassen diagram (figuur 8). Voor de eindgebruiker zijn 'today errors' en 'errors' twee verschillende

pagina's. De pagina's laten een andere selectie van errors zien, anders dan dat zijn ze identiek. Om deze reden is de applicatie zo ontworpen dat beide pagina's dezelfde code gebruiken. In het diagram is dit terug te vinden in het DetectErrorsFootfallComponent. Tijdens het creëren van het component zal gekeken worden of het om 'today' gaat, om vervolgens een andere vraag naar de webserver te sturen. De webserver is verantwoordelijk voor de daadwerkelijke selectie van errors die onder 'today' vallen.

Het DetectedErrorsComponent is verantwoordelijk voor de toevoeging van tabs, waarmee de gebruiker kan navigeren tussen footfall, wifi en kpi.



Figuur 33: Gedetecteerde errors pagina's klasse diagram

13.4. Implementatie

De gebruiker van de applicatie wil zien op welk datum met tijdslot een error gedetecteerd is. Een probleem met datums is dat er niet een standaard manier is hoe deze geformuleerd wordt. (bijv. dd-mm-yyyy) In mijn implementatie heb ik besloten gebruik te maken van Moment. Moment is een javascript library om JavaScript Date objecten te lokaliseren naar een string. Binnen Angular kan Moment dit doen m.b.v. een pipe. Nu kan een Date object aan de gebruiker getoond worden als bijv. '1 mei 2017 12:00', zoals te zien is in figuur 35.

De gebruiker kan ook zien om welk type error het gaat (no data, partly zero of deviation). Vanuit de webserver wil je deze data niet als string naar de client sturen, maar juist als integer. De eindgebruiker wil niet zien 'error type 1', want hij/zei weet waarschijnlijk niet wat dat is. In figuur 33 is een ErrorTypeIDTranslateString pipe te zien, welke ik heb gemaakt om dit probleem op te lossen. Hieronder is de Angular pipeline te zien, de 'value' is de originele waarde (het type error als integer). Deze wordt door de ErrorTypeIDTranslateString gehaald, welke de integer omzet naar een vertaling's string. Met deze string kan de error type gelokaliseerd worden voor de eindgebruiker. Tot slot wordt van de eerste letter een hoofdletter gemaakt, zodat alles er naar wens uit ziet.

```
{{ value | errorTypeIDTranslateString | translate | capitalizeFirstLetter }}
```

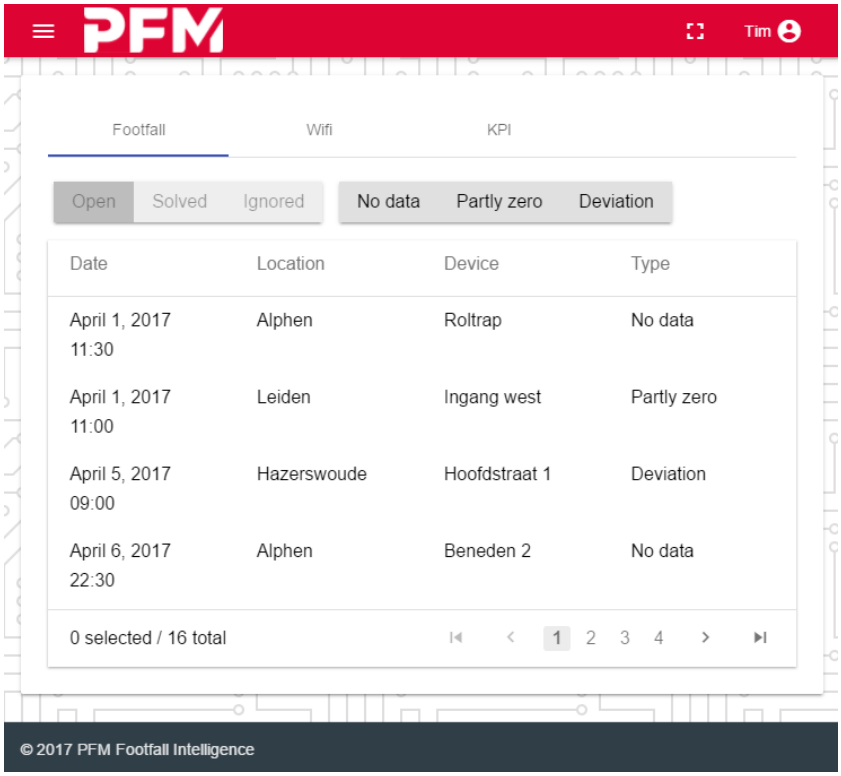
13.5. Acceptatie

Tijdens de acceptatie review waren de stakeholder opnieuw positief. Ik kreeg geen opmerkingen dat iets er niet goed uit zag, of anders moet. Wel kreeg ik een nieuwe wensen lijst mee, vol met uitbreidingen. Eén van deze nieuwe functionaliteiten is het negeren van een error. Op de error overzicht pagina (figuur 35) moet een functie worden ingebouwd, waarmee de geselecteerde errors de status 'ignore' kunnen krijgen. Wanneer dit het geval is zal de error niet meer weergegeven worden en telt deze niet meer mee op het dashboard.

Het admin dashboard, te zien in figuur 34, laat de hoeveelheid openstaande errors zien, zowel de totale hoeveelheid voor footfall devices, als de verschillende error types. De cijfers in het rood zijn de 'today' errors en de cijfers in het blauw de totalen.

Figuur 34: Admin dashboard

Het error overzicht (figuur 35) laat alle errors onder elkaar zien. De applicatie heeft de footfall tab geopend en laat alle type errors zien (het zijn toggle knoppen). De datum kolom is het tijdslot waarin de error gevonden is, gevolgd door de locatie naam, apparaat naam en error type. Onder in de tabel kan de gebruiker door verschillende tabel pagina's met gedetecteerde errors heen lopen.

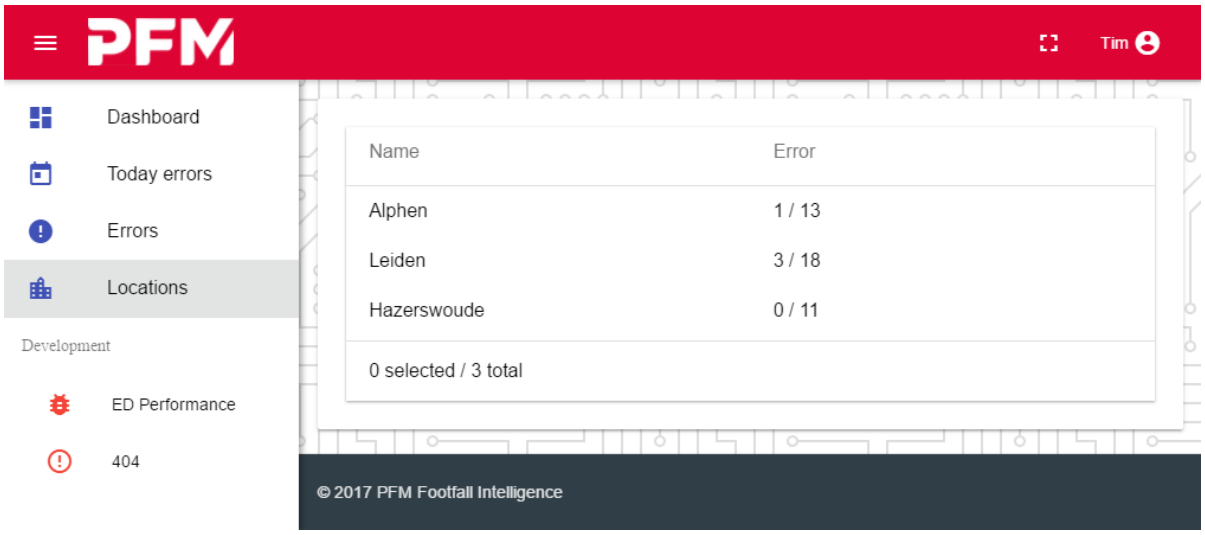


Figuur 35: Error overzicht pagina

14. Locatie pagina's

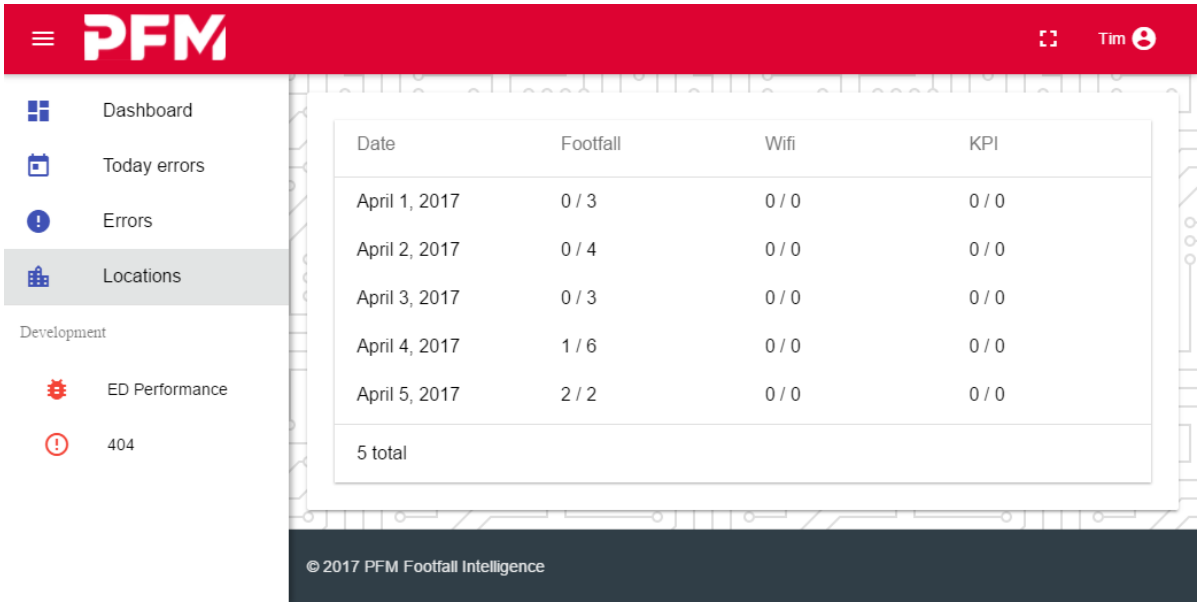
Tijdens de afstudeer periode zijn ook de locatie en locatie detail pagina's gemaakt. Deze sprint van een week is buiten de scope van dit verslag gelaten, omdat het weinig toegevoegde waarde biedt. Alles is op eenzelfde manier verlopen als de error pagina's sprint. Wel zal even kort het resultaat van deze sprint getoond worden.

De locatie pagina bevat een overzicht van alle locaties. In de error kolom staan het aantal open 'today' errors tegenover de totale hoeveelheid open errors. Naar aanleiding van de acceptatie meeting is besloten deze twee waardes te scheiden en in aparte kolommen te tonen. De gebruiker kan op een regel in de tabel klikken om voor die locatie de locatie detail pagina te openen.



Figuur 36: Locatie pagina

De detail pagina van een locatie geeft verschillende datums onder elkaar weer. Voor elke datum wordt per footfall, wifi en kpi het aantal errors weergegeven. De errors die worden getoond is het aantal openstaande errors tegenover de totale hoeveelheid. Als de gebruiker op één van deze kolommen klikt zal hij naar de reparatie pagina navigeren, welke in een volgende sprint ontwikkeld wordt.



Figuur 37: Locatie detail pagina

15. Evaluatie

15.1. Product

Applicatie

Na de opdracht uitgevoerd te hebben voldoet de applicatie niet aan de eisen zoals belooft in het afstudeerplan. De uiteindelijke applicatie beschikt niet over een datacorrectie functie.

Zoals in 10.1 is te lezen is besloten het project eerst te richten op de error detectie in plaats van de reparatie. Door deze keuze zijn de reparatie pagina's buiten de scope van het project beland. Wel voldoet de applicatie aan Chris zijn verwachtingen. Hij heeft laten weten dat het een goede basis is, waar zeker mee verder gewerkt zal worden.

De webapplicatie voldoet aan de Eisen van de stakeholders. De Stakeholders vinden het er erg mooi en modern uitzien, wat een doel van de webapplicatie is. Ook is de webapplicatie responsive gemaakt voor verschillende scherm formaten, zoals is te zien aan het navigatie menu 9.6. Hoewel er nog geen beveiliging in de website zit, wordt dit wel gesimuleerd. Tijdens het ontwerpen en bouwen van de webapplicatie is rekening gehouden met de beveiliging, door bijv. gebruik de maken van verschillende models in de client.

Een andere doelstelling is de gebruiksvriendelijkheid van de webapplicatie. Om de applicatie zo gebruiksvriendelijk mogelijk te maken is er gebruik gemaakt van verschillende hulpmiddelen, zoals Moment en een vertaal module.

Om de webapplicatie zo goed mogelijk uitbreidbaar te maken heeft de applicatie gedeelde modules en componenten, zoals de toolbar. Met deze gedeelde componenten kunnen op een later tijdstip gemakkelijk nieuwe pagina's worden toegevoegd.

Het error detectie algoritme presteert beter dan verwacht. Uit de performance test is gebleken dat alle huidige locaties in Advantage II binnen drie minuten getest kunnen worden op foutieve data. Bart was erg onder de indruk van dit resultaat en gaf aan dit algoritme verder uit te willen breiden.

Ik ben zelf erg tevreden over het eindresultaat. Hoewel niet alle doelstellingen zijn behaald, V.w.b. het datacorrectie gedeelte, ben ik van mening dat dit een goede basis opzet is. Over met name de server kant ben ik erg tevreden. Ik ben van mening dat de verschillende packages een duidelijke afsplitsing van taken hebben. Ook ben ik van mening dat het gebruik van verschillende frameworks en libraries de server ten goede komen, zoals automapper en serilog. Door het gebruik van deze libraries kan veel tijd bespaard worden tijdens de ontwikkeling.

Plan van Aanpak

Het plan van aanpak heeft mij en de stakeholders inzicht gegeven in de benodigde werkzaamheden. Hoewel de planning later in het project gewijzigd is, gaf het met name Chris inzicht in de doelstellingen die ik voor ogen had. Aan de hand van deze doelstellingen is vervolgens gewerkt.

SPA onderzoek

Het SPA onderzoek was erg belangrijk om een correcte keuze te kunnen maken in een SPA framework. Voorafgaand aan het onderzoek waren ik en de IT-afdeling niet op de hoogte van al deze frameworks. Naar onze mening is het ook goed, dat naar nieuwe frameworks als Aurelia is gekeken.

Dit onderzoek heeft inzicht gegeven in de verschillende beschikbare frameworks. Zonder dit onderzoek was mogelijk een verkeerde beslissing gemaakt, waardoor de performance mogelijk onder de maat zou zijn geweest.

Ik ben van mening dat zowel Angular 2 als Vue.js uitstekende keuzes waren geweest, maar met een blik op de toekomst Angular 2 net een voorsprong had.

Akka onderzoek

Het Akka.NET onderzoek heeft mij inzicht gegeven in de mogelijkheden van het framework. M.b.v. de uitgevoerde bootcamp heb ik geleerd hoe op een correcte manier met het framework gewerkt kan worden.

In de verschillende manieren waarop Akka geïntegreerd kan worden in het systeem was er een duidelijke beste keuze (voor dit project). Wel heeft het mij en Chris verder inzicht gegeven in de mogelijkheden van het framework. Op dit moment vinden er gesprekken plaats over een Advantage III. In dit nieuwe systeem zal Akka mogelijk weer een rol gaan spelen en misschien wel een complete integratie krijgen, zoals beschreven in één van de integratie mogelijkheden.

Requirements

De requirements zijn erg belangrijk geweest in dit project. Tijdens de interviews en acceptatiereviews hadden mensen snel de neiging nieuwe wensen te noemen. Deze wensen konden vervolgens alvast in Jira verwerkt worden voor toekomstige sprints. De requirements zullen een belangrijke rol blijven spelen, i.i.g. op Jira. De IT-afdeling vindt het dan ook belangrijk dat deze zijn bijgehouden, zeker nu besloten is de ontwikkeling voort te zetten.

Navigatiemap & Mockups

De navigatie map en mockups zijn in één sprint gemaakt. Deze producten hebben mij en de stakeholders geholpen met de beeldvorming hoe het uiteindelijke product eruit moet komen te zien. Deze producten zijn niet final, maar geven juist een goed start punt voor de verschillende verantwoordelijkheden van schermen binnen het systeem.

Use cases & beschrijvingen

Het use case diagram en beschrijvingen hebben voor mij weinig toegevoegde waarde gehad. Voor toekomstige ontwikkeling kunnen deze producten wel nuttig zijn. Naarmate de ontwikkeling van het project voort wordt gezet kunnen deze producten verder uitgewerkt worden. Zo heb je op een later moment naslag hoe het systeem te werk gaat.

Klasse diagram

Het klasse diagram is in twee delen op te splitsen, één voor de front-end (Angular) en één voor de backend. Deze diagrammen zijn goed ontvangen, het zal de toekomstige ontwikkeling gemakkelijker maken. Deze diagrammen hebben mij geholpen tot een goed ontwerp te komen. Ondanks dat het van Angular niet makkelijk is een uml diagram te maken, geeft het mij inzicht in de functionaliteit binnen elk component.

Sequence diagram

Sequence diagrammen zijn alleen gemaakt bij het ontwerpen van het error detectie algoritme. Deze hebben een ondersteunende rol gespeeld bij het ontwerpen van de interactie tussen actoren. Aan de hand van deze diagrammen is goed te zien hoe het algoritme mee kan schalen met de business. Deze diagrammen zullen voor het ontwikkelteam van belang zijn voor toekomstige uitbreidingen van het algoritme.

State diagram
Voor de verschillende actoren binnen het error detectie algoritme zijn state diagrammen gemaakt. Deze hebben net als sequence diagrammen een belangrijke rol gespeeld in het ontwerpen van het algoritme. Ze zijn dus net als sequence diagrammen van belang, voor toekomstige ontwikkelingen.

Testrapport
Ik ben niet tevreden over het testrapport. Voor de unit testen heb ik geen testontwerpen gemaakt, wat wel beloofd was in het afstudeerplan. Traditioneel unit testen was ook niet mogelijk, zoals uitgelicht in 12.4. Naar mijn menig heeft de opdrachtgever ook niet veel aan het rapport. Het beschrijft wel hoe te werk is gegaan, maar niet wat de testen zelf doen. Chris is toch blij met het rapport, omdat het toekomstige ontwikkelaars inzicht geeft in het unit testproces en de uitgevoerde performancetesten.

15.2. Proces
Planning
De originele planning is niet doorlopen, al vroeg in het project ging het mis met de opzet van het project. Voor deze taak waren origineel twee dagen gepland, maar dit zijn er zeven geworden. Tijdens deze sprint is de Webpack configuratie opgezet, iets dat ik nog nooit had uitgevoerd. Tijdens het maken van de planning had ik verwacht dat dit een simpele taak zou zijn, maar daar had ik me in vergist. Uiteindelijk is de keuze gemaakt vanuit een standaard configuratie te werken. Deze beslissing had ik eerder moeten maken, om niet meer tijd te verliezen. Het enige lichtpuntje van deze late beslissing is mijn huidige kennis van Webpack. Tijdens het configureren heb ik veel kennis opgedaan, betreft de werking en mogelijkheden van Webpack.

Tijdens de globale navigatie sprint is de originele planning compleet omgegooid, de redenen hiervoor staan beschreven in 10.1. Na de wijziging in de planning kon het systeem niet meer opgeleverd worden zoals in het afstudeerplan staat beschreven. Wel kunnen dezelfde producten opgeleverd worden en is het doel, een werkende demo van de applicatie opleveren, niet veranderd.

Na de wijziging in de planning zijn er geen problemen meer ontstaan. In tabel 18 is de uitgevoerde planning weergegeven, waarin de namen van de bijbehorende hoofdstukken staan vermeld.

Week	Beschrijving
1	Plan van Aanpak
1.5	Huidige situatie
2.5	SPA framework.
3.5	Opzet project
5	Akka.NET
6	Basis layout website.
8	Globale navigatie
9	Opzet webserver
10	Error detective algoritme
12	Error detectie performance testen
13	Error pagina’s
14	Locatie pagina’s

Tabel 18: Definitieve planning

Kanban

In Kanban is het gebruikelijk te werken met continuous delivery. In dit project wordt gewerkt met sprints, welke niet groter mogen zijn dan twee weken. De keuze voor deze manier van werken is in mijn ogen correct gebleken. Door in sprints te werken kunnen meerdere items samen worden gevoegd tot één. Het voordeel hiervan zijn de vaste contactmomenten met de stakeholders. Voorafgaand aan elke sprint is een interview gehouden met de stakeholders en een acceptatiereview, waarin de stakeholders het gemaakte product konden testen. Door deze vaste contact momenten wisten stakeholders waar ze aan toe waren.

Een groot onderdeel van Kanban is het Kanban bord, waar de verschillende issues op worden bijgehouden. Binnen dit project is Jira voor dit doeleinde gebruikt. Naar mijn mening heb ik dit niet goed genoeg bijgehouden. Hoewel de meeste issues, inclusief toekomstige, hierop vermeld staan, zijn deze niet in detail beschreven. (Toekomstige issues moeten nog verder besproken worden met de stakeholders.) Naar mijn mening had dit wel gemoeten, maar door tijdgebrek is dit achterwegen gelaten.

Voor elke Kanban card geldt een definition of done. Deze definition of done is tijdens het gehele proces aangehouden. Voor mij heeft het geen toegevoegde waarde gehad te werken volgens een definition of done. Ik weet voor mezelf wat de status van elke issue is. Wel geeft dit de stakeholders inzicht in de voortgang van het project. Naar mijn weten is niks gedaan met deze informatie. Door te werken met een definition of done merk ik wel dat er structuur in een project gebracht kan worden. Elke issue wordt op eenzelfde manier afgehandeld.

Meetings

De verschillende gesprekken die ik met de stakeholders heb gehad zijn naar mijn mening goed verlopen. Om requirements te achterhalen is voor elke sprint een gesprek ingepland met de stakeholders. Alle betrokkenen konden zo hun ideeën naar voren brengen en deze gelijk in de groep bespreekbaar maken. Door deze manier van werken kon snel overeenstemmingen gevonden worden tussen verschillende stakeholders. Een ander voordeel dat dit met zich mee brengt is de tijdsbesparing wat dit oplevert ten opzichte van één op één gesprekken met elke stakeholder.

De acceptatie reviews verliepen naar mijn mening ook goed. Ik merkte dat de stakeholders het leuk vonden tussentijdse resultaten te zien. Voor mij waren deze gesprekken ook erg belangrijk. Aan de hand van deze gesprekken wist ik zeker dat ik op de goede weg zat.

De gesprekken in de globale navigatie sprint zijn naar mijn mening het belangrijkst geweest. Als deze sprint er niet was geweest zou het verwachtingspatroon van de verschillende stakeholders er totaal anders uit hebben kunnen gezien. Juist omdat er in en vroegtijdig stadium afspraken zijn gemaakt is ervoor gezorgd dat iedereen op één lijn zit.

15.3. Beroepstaken

1.4 Uitvoeren analyse door definitie van requirements (niveau 3)

Naar mijn menig is het opstellen van de requirements uitgevoerd op niveau 3. Voor dit project is één set van requirements bijgehouden, welke naarmate het verloop van het project uitgebreid is. Dit project heeft vijf stakeholders (4.3), maar niet elke stakeholder heeft iets te zeggen over elk onderwerp. (Een data-analist heeft bijv. geen inspraak op de implementatie van Akka) Dit project heeft weinig tegengestelde requirements, de meetings zijn gezamenlijk gehouden. Hierdoor kon snel naar een definitieve set gewerkt worden. Alle requirements zijn door mij beheerd en gedefinieerd.

3.2 Ontwerpen systeemdeel (niveau 3)

Het ontwerpen van de applicatie is naar mijn mening op niveau vier uitgevoerd (drie was beloofd). Met name het error detectie algoritme is uitgebreid ontworpen. Tijdens het ontwerpen van het algoritme is begonnen met een flowchart (12.3.1), welke steeds verder is uitgewerkt tot een goed ontwerp. Er wordt voornamelijk op het schaalbaarheid aspect van het algoritme ingegaan. Het is belangrijk dat de software zo ontworpen wordt dat het kan meegroeien met de business, om zo in de toekomst de performance van het systeem ook zo hoog mogelijk te houden.

Voor het error detectie algoritme is ook ingegaan op de verschillende states, waar een actor zich in kan bevinden (12.3.3). In deze diagrammen staat beschreven welke actie een actor kan ontvangen om van state te veranderen en wat de flow tussen de verschillende states is.

Het is zeker dat de software in de toekomst gebruik zal moeten maken van tokens, om de user authenticatie te regelen. Hoewel de applicatie op dit moment geen beveiliging ingebouwd heeft, is in een vroegtijdig stadium hiermee rekening gehouden. De front-end van de applicatie maakt gebruik van verschillende type models, zodat niet elk component gebruikers data in kan zien 9.3.

De ontwerpen van de webserver maken gebruik van verschillende design patterns, zoals het strategy pattern en inversion of control (11.6). Alle ontwerpen zijn door mij zelfstandig gemaakt, de IT-afdeling had geen kennis van Akka. Zonder deze kennis is het lastig te begrijpen hoe de interactie tussen de actoren in het detectie algoritme werkt.

3.3 Bouwen applicatie (niveau 4)

Het bouwen van de applicatie is op het beloofde niveau vier uitgevoerd. De applicatie maakt gebruik van een meerdere frameworks, zowel in de front-end als de backend van de applicatie. In het verslag zijn meerdere voorbeelden weergegeven die demonstreren hoe deze frameworks werken, één daarvan is in hoofdstuk 12.2.1, waarin de interactie tussen actoren wordt uitgelicht.

Tijdens het bouwen van de applicatie zijn unit tests uitgevoerd, waar een test omgeving voor is opgezet (hoofdstuk 12.4.1). In deze test omgeving wordt de Advantage II database gemocked in sqlite, om zo een herbruikbare test dataset te creëren. De ontwikkelomgeving maakt gebruik van hulpmiddelen als hot module replacement (7.2.1). Met dit hulpmiddel kunnen source aanpassingen runtime naar de browser gestuurd worden.

Daarnaast maakt de applicatie gebruik van de bestaande Advantage II database, waar geen aanpassingen op zijn gemaakt. Om de applicatie hierop aan te laten sluiten, maar toch onafhankelijk te laten blijven, is gebruik gemaakt van verschillende type models. Dit staat beschreven in 11.1.

De source van de applicatie is gewaarborgd op git, hierbij wordt gebruik gemaakt van meerdere branches. Het bouwen van de applicatie is volledig door mij zelfstandig uitgevoerd. Binnen de IT-afdeling was nog nooit met Angular of Akka.NET gewerkt. Alle problemen waar ik tegenaan gelopen ben zijn door mij zelfstandig verholpen.

3.5 Uitvoeren van en rapporteren over het testproces (niveau 3)

Ik ben van mening dat ik niet heb voldaan aan deze beroepstaak. Er zijn voor dit project geen testontwerpen gemaakt, iets wat ik wel beloofd had. Wel zijn er unit testen uitgevoerd, voor verschillende services en de error detectie actoren (12.4). Naast de unit testen is er ook een performance test uitgevoerd, welke naar mijn mening geslaagd is.

Begrippen

2-Way databinding	Zodra een attribuut in de model/controller wijzigt, wordt de view automatisch geüpdatet.
Component (Angular)	Een component is een class met functionaliteit, welke alleen toegankelijk is door de bijbehorende html.
Dependency Injection (DI)	Dependency injection is een techniek waarbij een object de dependencies beheerd voor een ander object. Bijv. een controller die een service geïnjecteerd krijgt door het DI framework.
Designtaal (design language)	Een designtaal is een stijl waarin alles wordt ontworpen, zodat alles er consistent eruitziet.
Directive (Angular)	Een directive kan toegevoegd worden aan een DOM-element, om zo extra functie toe te voegen of het element zelf aan te passen.
Document Object Model (DOM)	Het DOM is een interface wat toegang bied tot HTML documenten. De data wordt opgeslagen als boomstructuur.
DOM Node	Een DOM node is een javascript object wat een HTML element representeert.
Dotnet Core (.NET Core)	Een geminimaliseerde versie van de .NET runtime welke zowel: Windows, Ubuntu als Mac ondersteund.
Footfall device (apparaat)	Een camera of ander soort sensor dat passanten telt. (geen wifi!)
Guard (Angular)	Een guard kan toegevoegd worden aan een bepaalde route in de router. Zodra een gebruiker deze route op wil gaan, zal de router vragen aan de guard of dat mag.
Hot module replacement (HMR)	Een webpack feature waarmee oude modules vervangen kunnen worden, zonder de browser te herladen.
JSON (JavaScript Object Notation)	JSON is een notatie wijze, om data te delen tussen verschillende systemen.
Lazy loading	Lazy loading is een techniek in web development, waarin content pas geladen wordt zodra het nodig is.
Material design	Material design is een designtaal ontwikkeld door Google. O.a. Android wordt op deze manier ontworpen.
Module (Angular)	Een module is een class welke alle type classes (componenten, pipes, enz.) declareert, zodat deze gebruikt kunnen worden. De verschillende classes kunnen alleen bestaan binnen de scope van de module.

Node.js	Node.js is een javascript runtime gebouwd op Chrome's V8 JavaScript engine. Node komt gebundeld met NPM en is nodig om voor Angular te programmeren.
NPM	NPM is een package manager voor JavaScript en wordt gebruikt om JavaScript dependencies te downloaden en te beheren.
Pipe (Angular)	Een pipe wordt gebruikt als data nog wat extra opmaak nodig heeft, voordat het aan de gebruiker getoond wordt op een pagina.
Router (Angular)	Routers bepalen wat er gebeurt zodra de url veranderd.
Service (Angular)	Een service bevat meestal functionaliteit dat gedeeld moet worden over meerdere componenten. Elk component heeft toegang tot de service, zolang het onder dezelfde scope valt.
Single page application (SPA)	Een website welke een enkele html pagina laad en deze d.m.v. JavaScript dynamisch aanpassen.
Webpack	Webpack is een module bundler voor JavaScript applicaties.
Wifi scanner	Een apparaat (omgebouwde router) met als functie wifi & bluetooth mac-adressen op te sporen en door te sturen naar een database/server.

Verwijzingen

[1] S. Krause, „JS web frameworks benchmark – Round 5,” 25 Januari 2017. [Online]. Available: http://www.stefankrause.net/wp/?m=201701. [Geopend 14 Februari 2017].

[2] S. Krause, „JS web frameworks benchmark – keyed vs. non-keyed,” 24 Januari 2017. [Online]. Available: http://www.stefankrause.net/wp/?p=342. [Geopend 14 Februari 2017].

[3] S. Krause, „Results for js web frameworks benchmark – round 5,” 25 Januari 2017. [Online]. Available: http://stefankrause.net/js-frameworks-benchmark5/webdriver-ts/table.html. [Geopend 14 Februari 2017].

[4] Brian, „Angular, React, and Vue: What’s Coming in 2017?,” 20 Januari 2017. [Online]. Available: https://angular.jsnews.io/angular-react-and-vue-whats-coming-in-2017-js-javascript-angular2-reactjs-vuejs/. [Geopend 15 Februari 2017].

[5] I. Minar, „Versioning and Releasing Angular,” 7 October 2016. [Online]. Available: https://angularjs.blogspot.nl/2016/10/versioning-and-releasing-angular.html. [Geopend 15 Februari 2017].

[6] „Ahead-of-Time Compilation,” Google, [Online]. Available: https://angular.io/docs/ts/latest/cookbook/aot-compiler.html. [Geopend 1 Maart 2017].

[7] GRardB, SpaceK33z, rouzbeh84 en sokra, „Hot Module Replacement,” Webpack, [Online]. Available: https://webpack.js.org/concepts/hot-module-replacement/. [Geopend 21 Februarie 2017].

[8] A. Stannard, „Akka.NET Bootcamp,” 18 Januari 2017. [Online]. Available: https://github.com/petabridge/akka-bootcamp. [Geopend 2 Maart 2017].

[9] „Actor Systems,” [Online]. Available: http://getakka.net/docs/concepts/actorsystem. [Geopend 8 Maart 2017].

[10] „Routers,” [Online]. Available: http://getakka.net/docs/working-with-actors/Routers. [Geopend 8 Maart 2017].

[11] „Fault Tolerance,” [Online]. Available: http://getakka.net/docs/Fault%20tolerance. [Geopend 8 Maart 2017].

[12] J. Bridgman, „HashSet vs. List performance,” 4 November 2015. [Online]. Available: https://stackoverflow.com/questions/150750/hashset-vs-list-performance. [Geopend 5 April 2017].

[13] nblumhardt, Danthar, sean-gilliam, skotzko, rogeralsing en kthompson, „Dependency Injection,” 10 februarie 2017. [Online]. Available: http://getakka.net/docs/Dependency%20injection. [Geopend 6 April 2017].

[14] „Streams introduction,” Akka.NET, [Online]. Available: http://getakka.net/docs/streams/introduction. [Geopend 29 April 2017].

[15] „Dispatchers,” Akka.NET, [Online]. Available: http://getakka.net/docs/working-with-actors/Dispatchers. [Geopend 29 April 2017].

Bijlagen

A.	Tim van Egmond afstudplan_2017-1.1
B.	Plan van Aanpak
C.	Voortgangsverslag
D.	Onderzoeksrapport SPA
E.	Onderzoeksrapport Akka.NET
F.	Requirements Discipline
G.	Design Discipline
H.	Testrapport
I.	Evaluatieformulier afstuderen