Afstudeerverslag

Data Repository

Glenn van Leeuwen
09094202
Informatica, voltijd
Delft
22-12-2016

Referaat

Dit afstudeerverslag beschrijft het proces dat Glenn van Leeuwen heeft doorlopen bij het ontwikkelen van een data repository service genaamd Data Repository, in de periode van 29 augustus 2016 t/m 22 december 2016 bij het bedrijf Ask Roger! te Delft.

Auteur	Glenn van Leeuwen
Instituut	De Haagse Hogeschool
Faculteit	Academie voor It & Design
Vestiging	Dutch Innovation Factory te Zoetermeer
Opleiding	Informatica
Vorm	Voltijd
Studentnummer	09094202
Afstudeerbedrijf	Ask Roger!
Locatie	Delft
Datum	22-12-2016

Voorwoord

Ik wil graag alle Rogers bedanken voor een goede tijd bij het bedrijf. De sfeer is goed, vooral als Cemal mensen een goede morgen wenst.

Daarnaast wil ik Chianne en Frank bedanken voor de begeleiding tijdens mijn stage, dit verliep soepel en heb niets te klagen gehad.

Ook wil ik Frank nog bedanken voor de nuttige feedback op m'n code en Giorgi voor de hulp bij het debuggen.

Ten slotte bedank ik Sergio voor de feedback over het afstudeerverslag en het proces eromheen, dit heeft ook enorm geholpen.

Oh and remember Nikos: One Piece is fake.

Inhoudsopgave

1 Inleiding

Ik heb een afstudeerstage gelopen bij het bedrijf Ask Roger!. Dit document is mijn verslag van deze stage, waarin ik de werkzaamheden beschrijf die ik gedaan heb en ik een verantwoording af leg van de gemaakte keuzes.
Ask Roger! heeft een softwareoplossing genaamd ToastAR, die bij binnenkomende oproepen in Skype For Business en IP telefooncentrales relevante klantgegevens toont in een pop-up venster. Als een gesprek plaatsvindt worden deze gegevens in meer detail weergegeven in een venster naast de chat of het (video)gesprek. Deze klantgegevens komen uit CRM en ERP systemen van de bedrijven zelf die de ToastAR gebruiken, deze systemen zijn lokaal aan het ToastAR systeem gekoppeld. ToastAR vraagt als client applicatie deze gegevens op via een Windows service genaamd HoastAR.
Dynamische data zoals bijvoorbeeld openstaande facturen wordt via web services opgehaald door HoastAR. Statische data zoals NAW gegevens worden anders opgehaald. Dit haalt HoastAR in de huidige situatie op via een andere Windows service genaamd MetaDirectory, een LDAP-directory service van het bedrijf Estos. Dit wordt gedaan zodat deze statische data vrijwel direct getoond kan worden door de ToastAR, dit is namelijk sneller dan web services. Deze snelle methode kan niet gebruikt worden voor dynamische data, vandaar het verschil in data opvragen door HoastAR.
De opdracht is het vervangen van deze directory service en deze nieuwe software toevoegen aan de HoastAR als een module hiervan.
Er is eerst een console applicatie gemaakt met de gewenste functionaliteiten alvorens dit te integreren in de HoastAR. Hierna is er een integratietest uitgevoerd om te zien of dit goed geïntegreerd is.
Het eindproduct kan contactgegevens ophalen uit externe bronnen, lokaal opslaan en deze via HoastAR leveren aan de ToastAR. De externe bronnen waaruit data opgehaald kan worden, zijn in de situatie aan het eind van de stage: CSV-bestanden en databases via een ODBC-koppeling. Verder kan ingesteld worden vanaf welk tijdstip en met welke interval na dit tijdstip deze gegevens worden opgehaald per databron.

In hoofdstuk 2 wordt de organisatie toegelicht van de opdrachtgever en de plaats van de afstudeerder daarin. Hierna wordt in hoofdstuk 3 de situatie toegelicht bij aanvang van de afstudeeropdracht. De werkzaamheden worden beschreven in hoofdstuk 4, met toelichting op en motivatie van de gemaakte keuzes. De werkzaamheden zijn onderverdeeld in de sprints die uitgevoerd zijn. De afwijkingen van het afstudeerplan worden hierna beschreven in hoofdstuk 5. De opgeleverde producten staan beschreven in hoofdstuk 6. De evaluatie van de gekozen aanpak, opgeleverde producten en de beroepstaken is erna te vinden in hoofdstuk 7. Ten slotte is er een literatuurlijst en een begrippenlijst achteraan dit document te vinden in hoofdstuk 8 en 9.

2 Organisatie

// Beschrijving van de organisatie van de opdrachtgever en de plaats van de afstudeerder daarin.

Ask Roger! is een bedrijf in de branche IT, Telecom en communicatie. Zij implementeren communicatiesystemen voor bedrijven. Dit zijn middelgrote en grotere bedrijven. Het aantal gebruikers per klant ligt tussen de 250 en 1000 gebruikers.

Voor het online samenwerken met collega's en klanten biedt Ask Roger! oplossingen als video conferencing en unified communications. Met video conferencing kan er vergaderd worden op afstand met behulp van videogesprekken. Bij unified communications draait het om tijd, plaats en apparaat onafhankelijk communiceren. Dit is niet één vast product maar een combinatie van meerdere oplossingen.

Ter verbetering van het klantcontact bij bedrijven biedt Ask Roger! oplossingen als CRM/ERP integratie en rapportage systemen. Bij CRM/ERP integratie worden klantgegevens uit deze systemen samengevoegd binnen één applicatie die is gekoppeld aan een IP telefooncentrale of aan Microsoft Skype for Business. Hiermee hebben bedrijven direct klantinformatie van de klant die hen belt of een chat start. Voor rapportage ontwikkelt Ask Roger! verschillende eigen oplossingen die bovenop basiselementen kunnen draaien, zoals een callcenter wallboard bovenop Cisco Unified Contact Center.

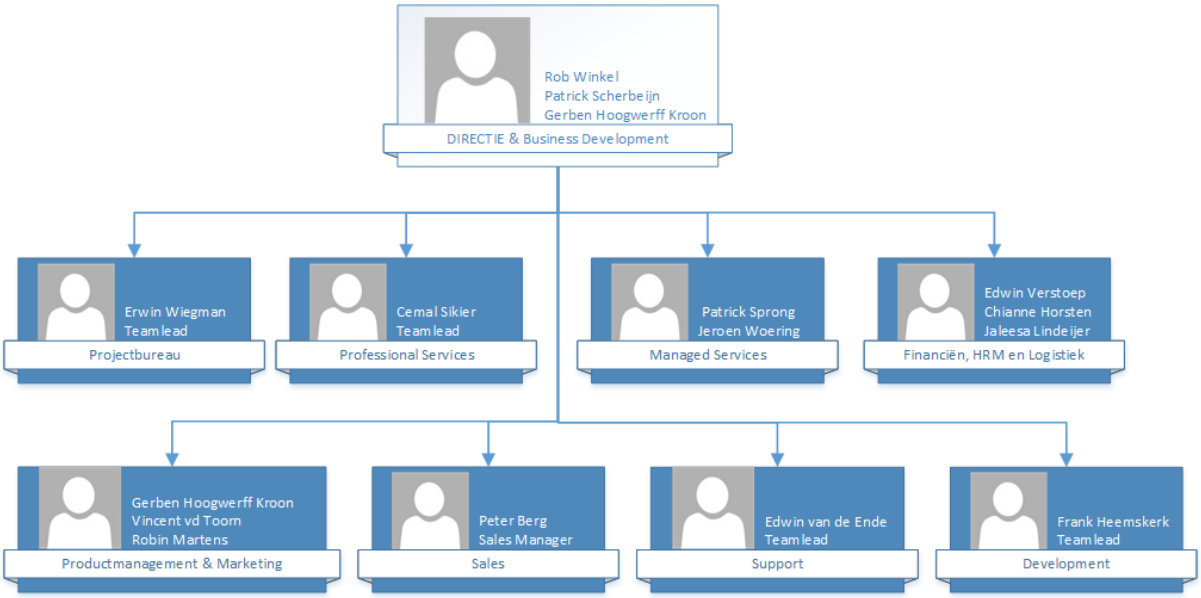
Voor een optimale bereikbaarheid van bedrijven biedt Ask Roger! oplossingen als omnichannel contact centers, vast-mobiel integratie en enterprise networks. Omnichannel contact centers zijn contact centers die niet alleen via een telefoon bereikbaar zijn, maar bijvoorbeeld ook via een browser, chat of video contact met een medewerker. Met vast-mobiel integratie maakt het niet uit of er een vaste of mobiele telefoon wordt gebruikt, gesprekken van klanten kunnen worden aangenomen onafhankelijk hiervan. Ask Roger! biedt het beheren en monitoren van netwerk infrastructuren aan onder het mom van enterprise networks.
Naast het aanleggen van bedrijfsnetwerken biedt Ask Roger! ook internet en telefonie netwerken aan bedrijven.

Verder heeft Ask Roger! consultants die advies op maat geven aan bedrijven. Ook leveren zij demo's en training sessies. Ten slotte heeft Ask Roger! ook een support afdeling waar support wordt geboden voor alle producten die bij het bedrijf worden afgenomen.

De focus ligt op robuuste systemen voor de lange termijn, het bedrijf wil een lange termijnrelatie aangaan met haar klanten. Het bedrijf is oplossingsgericht en zal niet snel aangeven dat iets onmogelijk is.

Ask Roger! heeft als klanten bijvoorbeeld de bedrijven en instellingen Corendon, Staatsbosbeheer, BPD, Nova College en Davo Wittebrug. Hiermee heeft Ask Roger! klanten in verscheidene branches.[1]

Ask Roger! is een groeiend bedrijf en zij verwachten deze groei voort te zetten de komende maanden. In september waren er ongeveer 40 medewerkers in dienst. Dit is ondertussen uitgegroeid tot ongeveer 50 medewerkers. Het bedrijf heeft op dit moment twee locaties: Houten en Delft, waarbij Delft de hoofdlocatie is. Er bevinden zich meerdere afdelingen binnen de organisatie.



Figuur 1. Organogram.

Een uitgebreide versie van het organogram is te vinden in bijlage 5.
Het bedrijf heeft als toplaag de afdeling Directie & Business Development. Deze afdeling bestaat uit drie personen.
In figuur 1 zijn de acht andere afdelingen te zien met bijbehorende team leiders of managers.

De opdracht heb ik uitgevoerd op de afdeling Development, deze afdeling bevindt zich alleen op de hoofdlocatie Delft. Er zijn ook andere afdelingen aanwezig op deze locatie. De afdeling Development bestaat uit 3 personen, waarbij ik de 4^{e} persoon op de afdeling ben. De werkdruk van de afdeling zonder mij bestaat uit 3 FTE. FTE is een eenheid die staat voor Full-time Equivalent, in het Nederlands wordt deze eenheid ook wel werktijdfactor genoemd. Een FTE van 1 staat gelijk aan de werkdruk van een fulltime werknemer.

3 Huidige situatie

// Beschrijving van de situatie bij aanvang van het afstuderen.

ToastAR is een applicatie die bij binnenkomende oproepen in Skype For Business of een IP telefooncentrale relevante klantgegevens toont in een pop-up venster. Deze klantgegevens komen uit CRM en/of ERP systemen van de bedrijven zelf die de ToastAR gebruiken, deze systemen zijn lokaal aan het ToastAR systeem gekoppeld. HoastAR is de server applicatie, ToastAR is de client applicatie. ToastAR heeft HoastAR nodig om data op te halen. Zie figuur 2 voor een duidelijk beeld van de samenhang van deze applicaties.

Via ToastAR kan ook gezocht worden in de contactinformatie afkomstig van een gekoppeld CRM of ERP systeem. Hiermee kan er naar personen worden gezocht en kan er uit de zoekresultaten een persoon worden geselecteerd, die dan gebeld kan worden.
Verder kan via de ToastAR een nummer of postcode aangeklikt worden in de applicatie, waarna met een hotkey (F8) die persoon gebeld wordt.

ToastAR is een oplossing die in eerste instantie ontwikkeld is voor gebruik in combinatie met Skype For Business. Dit is ondertussen uitgebreid voor IP telefonie via een TAPI koppeling. Telephony Application Programming Interface (TAPI) is een Microsoft Windows API dat telefonie mogelijk maakt via het Windows besturingssysteem. Toepassingen zoals Cisco Call Manager en Avaya Communication Manager kunnen via deze interface aangestuurd worden.[2]

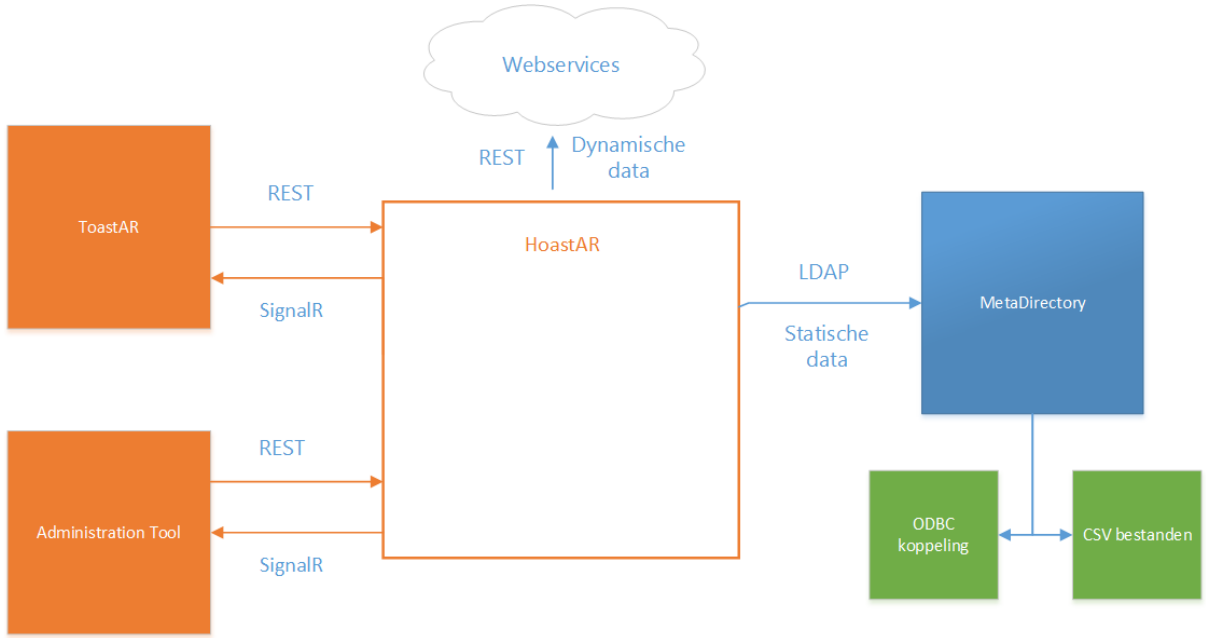
Wanneer er een inkomend telefoongesprek of inkomend chatbericht is, dan vraagt ToastAR informatie over deze beller of chatter op aan HoastAR. HoastAR haalt aan de hand van de Caller ID, postcode of andere kenmerken verdere informatie op uit het gekoppelde CRM of ERP systeem. Deze informatie wordt daarna geleverd aan ToastAR en getoond op het scherm in de vorm van een pop-up. In de pop-up is ook de mogelijkheid de telefoon op te nemen of de chat te openen.
Als de telefoon wordt opgenomen of de chat is geopend, dan wordt er een groter scherm weergegeven met relevante informatie over de beller of chatter.

Informatie aanvragen van ToastAR aan de HoastAR gaan via een Representational state transfer (REST) architectuur. Ook wel RESTful web services genoemd. Deze architectuur is een manier om interoperabiliteit te leveren tussen computersystemen over het internet.[3] Op deze manier kan ToastAR aanvragen doen aan HoastAR zonder fysiek in dezelfde omgeving te hoeven draaien.
Om data terug te sturen van HoastAR naar ToastAR wordt er gebruik gemaakt van ASP.NET SignalR. Dit is een ASP.NET library die het mogelijk maakt om data van de server kant te sturen naar een client als deze data beschikbaar komt.[4] Dit is ideaal voor dit systeem omdat de door ToastAR opgevraagde data niet altijd direct beschikbaar is.

HoastAR is ook bereikbaar voor een administratie tool van Ask Roger, deze maakt op dezelfde manier als ToastAR een connectie met HoastAR.

Statische data zoals adresgegevens en telefoonnummers worden door de HoastAR anders opgehaald dan dynamische data. Denk bij dynamische data aan data zoals openstaande facturen van klanten, waarbij het van belang is dat deze informatie actueel is.
Statische data wordt gerepliceerd uit de gekoppelde CRM en/of ERP systemen en lokaal opgeslagen. Deze data kan vrijwel direct geleverd worden aan ToastAR en is bij binnenkomende telefoongesprekken en chatberichten direct te zien in de pop-up. Het repliceren van de data gebeurt dagelijks. Deze snelle manier van informatie leveren aan ToastAR is dus niet geschikt voor dynamische data, omdat dit type data niet verouderd mag zijn.
Dynamische data wordt via RESTful web services opgevraagd en dit vergt meer tijd, deze informatie is dus later in ToastAR op het scherm te zien dan de statische gegevens.

De koppeling tussen de CRM en ERP systemen met klantinformatie en HoastAR verloopt in de huidige situatie via een directory service genaamd MetaDirectory van het bedrijf Estos.
Een directory service is een dienst die het mogelijk maakt om toegang te krijgen tot hiërarchisch georganiseerde gegevens die eventueel verspreid zijn opgeslagen in een computernetwerk.[5,6]
MetaDirectory kan gegevens uit bepaalde databronnen repliceren en lokaal opslaan. Deze bronnen zijn o.a. Comma Separated Value (CSV) bestanden en databases via een Open Database Connectivity (ODBC) koppeling. ODBC is een interface voor het benaderen van database management systemen.[7] Gegevens uit CRM en ERP systemen worden in de huidige situatie meestal geëxporteerd naar CSV bestanden alvorens deze bestanden worden gerepliceerd en opgeslagen door de MetaDirectory. De lokaal opgeslagen data wordt opgevraagd door HoastAR. Er kan ingesteld worden met welk interval de data wordt gerepliceerd door de MetaDirectory, in de huidige situatie is dit meestal een interval van een dag. HoastAR en MetaDirectory draaien beide als Windows Service, de communicatie tussen de HoastAR en de directory service verloopt via het Lightweight Directory Access Protocol (LDAP). LDAP is een protocol voor het benaderen van gedistribueerde directory services over een IP netwerk.[8]



Figuur 2. Systeem Context Diagram huidige situatie.

In figuur 2 is een overzicht te zien waarin de samenwerking van de systemen wordt weergegeven in een Systeem Context Diagram. De CRM en ERP systemen zijn gekoppeld aan de MetaDirectory via een ODBC koppeling of via geëxporteerde CSV bestanden zoals eerder beschreven. Het onderscheid in het type data is ook in het diagram weergegeven, de statische data wordt door de HoastAR opgehaald van de MetaDirectory via LDAP en de dynamische data via RESTful web services.

ToastAR, HoastAR en de administratie tool zijn geschreven in C# en maken gebruik van het .NET Framework.

In de HoastAR wordt gebruik gemaakt van de design pattern Dependency injection. Deze pattern implementeert het Inversion of Control principle, waarbij delen van een programma aangestuurd worden door een generiek framework. De modules van HoastAR hebben in hun constructor staan welke service zij nodig hebben en deze wordt geleverd bij het opstarten van deze modules. Een goed voorbeeld is de logger module die bijna in alle constructors aanwezig is om foutmeldingen te kunnen loggen. Om deze pattern toe te passen wordt er gebruik gemaakt van een open source dependency injector genaamd Ninject. Dit is een library voor .NET applicaties. Ninject is het framework dat gebruikt wordt om de modules binnen HoastAR aan te sturen.[9,10]

4 Werkzaamheden

// Beschrijving van de werkzaamheden met toelichting op en motivatie van de gemaakte keuzes.

Voor dit project heb ik gebruik gemaakt van een aangepaste versie van de ontwikkelmethodiek Scrum. Hierbij maak ik gebruik van sprints, iteraties van twee tot drie weken waarin ik delen van het project of systeemdelen in oplever. In de initiële planning waren deze sprints ingepland:

Sprint 0 (3 weken):

- Plan van Aanpak
- Globale Requirements vaststellen
- Systeem ontwerpen

Sprint 1 (2 weken):

- Basis van de data repository ontwerpen
- Basis van de data repository bouwen
- Basis van de data repository testen

Sprint 2 (2 weken):

- Lokale data opslag van de repository ontwerpen
- Lokale data opslag van de repository bouwen
- Lokale data opslag van de repository testen

Sprint 3 (2 weken):

- Replicator voor CSV databronnen ontwerpen
- Replicator voor CSV databronnen bouwen
- Replicator voor CSV databronnen testen

Sprint 4 (2 weken):

- Replicator voor Excel databronnen ontwerpen
- Replicator voor Excel databronnen bouwen
- Replicator voor Excel databronnen testen

Sprint 5 (2 weken):

- Replicator voor ODBC databronnen ontwerpen
- Replicator voor ODBC databronnen bouwen
- Replicator voor ODBC databronnen testen

Sprint 6 (4 weken):

- Custom replicator voor handmatige configuratie ontwerpen
- Custom replicator voor handmatige configuratie bouwen
- Custom replicator voor handmatige configuratie testen

Sprint 7(overige tijd):

- Replicator voor een specifiek CRM of ERP systeem ontwerpen

- Replicator voor een specifiek CRM of ERP systeem bouwen
- Replicator voor een specifiek CRM of ERP systeem testen”

De daadwerkelijk uitgevoerde sprints wijken af van dit plan. In hoofdstuk 6 worden de afwijkingen van het afstudeerplan duidelijk gemaakt.

De werkzaamheden zullen per uitgevoerde sprint toegelicht worden. Per sprint zal de sprint backlog te zien zijn onder het kopje Backlog en zullen de werkzaamheden in de sprint toegelicht worden onder het kopje Uitvoering. Hierna wordt de sprint review onder het kopje Review verwerkt, waarin de tussenresultaten worden geanalyseerd. De sprint retrospectives zijn uitgewerkt in hoofdstuk 7, Evaluatie. Voor meer informatie over Scrum, waar deze structuur in het verslag op gebaseerd is, zie hoofdstuk 4.1.2, bladzijde x.

De daadwerkelijk uitgevoerde sprints die worden beschreven in dit hoofdstuk zijn:

Sprint 0: Project Opstart. (3 weken)
Sprint 1: Basis van de applicatie. (2 weken)
Sprint 2: Scheduler. (2 weken)
Sprint 3: CSV Replicator. (2 weken)
Sprint 4: ODBC Replicator. (2 weken)
Sprint 5: HoastAR Integratie. (2 weken)
Sprint 6: Documentatie. (3 ½ weken)

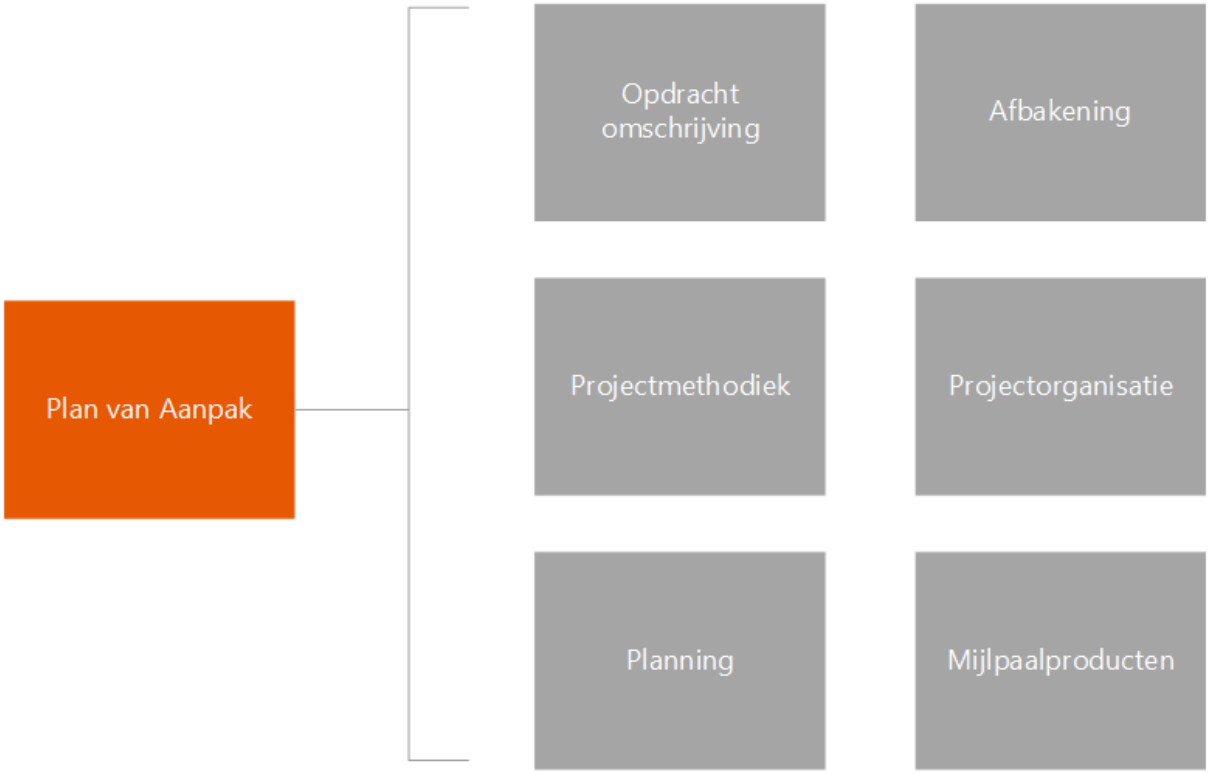
4.1 Sprint 0: Project Opstart

In de eerste sprint, sprint 0, heb ik de opstart gemaakt van het project. Voor deze sprint zijn 3 weken ingepland, de sprints hierna hebben een duur van twee weken. Hiervoor is gekozen omdat de opstart veel documentatie en ontwerpen bevat, wat naar verwachting meer tijd in beslag zou nemen dan een systeemdeel ontwikkelen en testen.

4.1.1 Backlog

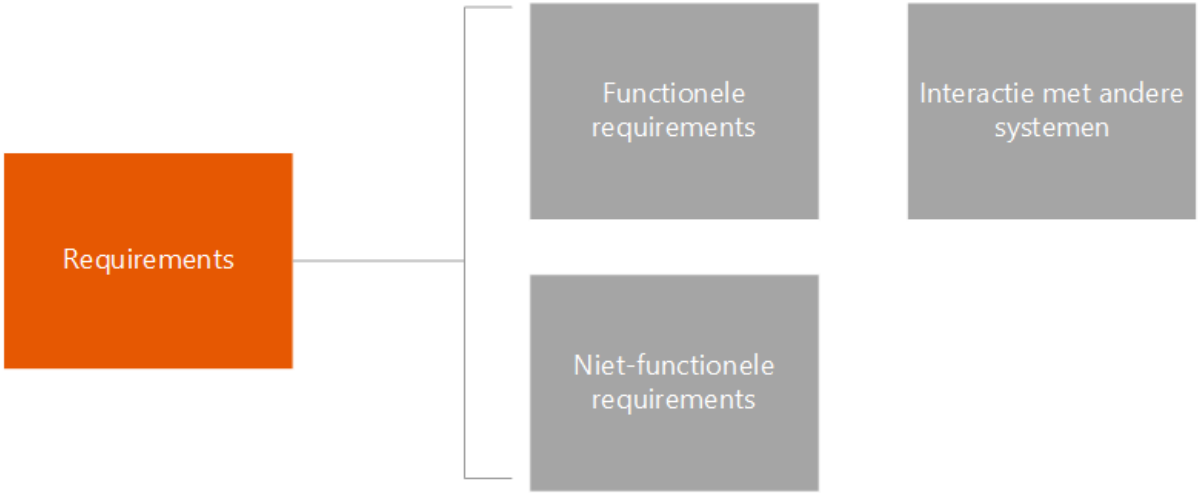
De volgende producten zijn toegevoegd aan de sprint backlog van Sprint 0:

Het Plan van Aanpak (PvA), de requirements voor dit project zijn vastgelegd en er is een eerste versie van het systeemontwerp gemaakt. Om een overzicht te hebben van de test soorten die uitgevoerd worden voor het project maak ik een Mastertestplan. De sprint backlog onderdelen zijn in figuur 3, 4, 5 en 6 te zien. Waarbij de backlog onderdelen in het oranje te zien zijn en in het grijs zijn deze verder onderverdeeld in taken die verricht moeten worden. Als alle taken volbracht zijn, is het sprint backlog onderdeel voltooid.



Figuur 3. Sprint 0 backlog PvA.

Het maken van het PvA is onderverdeeld in de volgende taken, het maken van de hoofdstukken: Opdracht omschrijving, Afbakening, Projectmethodiek, Projectorganisatie, Planning en Mijlpaalproducten.



Figuur 4. Sprint 0 backlog Requirements.

Het maken van het Requirements document bestaat uit het samenstellen van de Functionele requirements, de Niet-functionele requirements en het weergeven van de interacties van de data repository service met andere systemen.

Figuur 5. Sprint 0 backlog Klassediagram.

Het maken van het Klassediagram bestaat uit het ontwerpen van de klassen, eventuele attributen, en methodes hierin.

Figuur 6. Sprint 0 backlog Mastertestplan.

Bij het maken van het Mastertestplan worden de testsoorten beschreven die ik ga gebruiken tijdens dit project. Er zal per testsoort een beschrijving van de testsoort komen met mijn reden voor het gebruik van deze testsoort.

4.1.2 Uitvoering

In het PvA is een beknopte Probleembeschrijving en Organisatie beschrijving te vinden. Zie hoofdstuk 2 en 3 in dit verslag voor een uitgebreide beschrijving van de organisatie en de huidige situatie.

De doelstelling is in het PvA beschreven als: "Het doel van de opdracht is het ontwikkelen en het opleveren van een werkende data repository service die data uit externe bronnen lokaal opslaat en eventuele module(s) hiervoor ontwikkelen die data ophalen uit specifieke CRM of ERP systemen als hier nog tijd voor is."
Omdat niet zeker was of er genoeg tijd zou zijn om de modules te maken die data uit specifieke CRM of ERP systemen moeten repliceren heb ik dit op deze manier beschreven in de doelstelling. Het hoofd doel is de data repository service zelf. Deze data repository service zal verder in dit document Data Repository worden genoemd.

Het beoogde resultaat na een succes van deze opdracht heb ik in het PvA beschreven als: "Het resultaat is een interne service die het systeem geheel in eigen handen brengt van Ask Roger! en het bedrijf niet meer afhankelijk maakt van een externe service.
Hiermee kan er inde toekomst een pakket aangeboden worden aan klanten met geheel eigen software. Als de modules voor de verschillende CRM/ERP systemen ingebouwd zijn, kan er ook gedacht worden aan trial versies voor het softwarepakket. Daarnaast kan er ook gedacht worden aan de markt in het buitenland, omdat dan niet meer handmatig de koppeling gemaakt hoeft te worden bij de klant, dit wordt dan automatisch geregeld met behulp van de modules."
De opdracht is dus belangrijk voor het bedrijf. Door de software in eigen handen te hebben kunnen de huidige functionaliteiten die gebruikt worden uitgebreid worden

met replicatie mogelijkheden voor specifieke CRM en ERP systemen.
In de huidige situatie wordt contact data uit CRM en ERP systemen in de meeste gevallen eerst geëxporteerd naar een CSV bestand, alvorens dit geleverd wordt aan de Meta Directory. Dit wordt handmatig ingesteld in de huidige situatie.
Als dit geautomatiseerd wordt met behulp van de Data Repository kan daarna dus gedacht worden aan trial versies van deze software en de markt in het buitenland.

De afbakening van het project in het PvA luidt als volgt: "De opdracht beperkt zich tot de Data Repository, een module die data uit verschillende soorten databronnen lokaal opslaat en de data via de HoastAR aanbiedt aan de ToastAR applicatie. Er moet data vanuit de HoastAR opgevraagd kunnen worden waarmee het repliceren van de data uit een bepaalde bron ingepland kan worden."
Hiermee heb ik de scope van het project duidelijk vastgelegd om zo werk buiten deze scope te vermijden. Deze afbakening is gemaakt met behulp van de functionele requirements FR1, FR2, FR4 en FR5.

Code requirement	Beschrijving requirement
FR1	De Data Repository moet lokaal data gestructureerd kunnen opslaan.
FR2	De Data Repository moet data uit externe bron kunnen repliceren en deze lokaal opslaan.
FR4	De Data Repository moet lokale data kunnen leveren in een vooraf vastgesteld format. Dit is de ContactData klasse.
FR5	De Data Repository moet iedere replicatie kunnen inplannen aan de hand van instellingen geleverd vanuit HoastAR.

FR1 geeft aan dat er lokaal een database moet komen waarin gegevens in een bepaalde structuur opgeslagen moeten kunnen worden.

FR2 sluit aan op FR1, gerepliceerde data moet opgeslagen worden in deze lokale database.

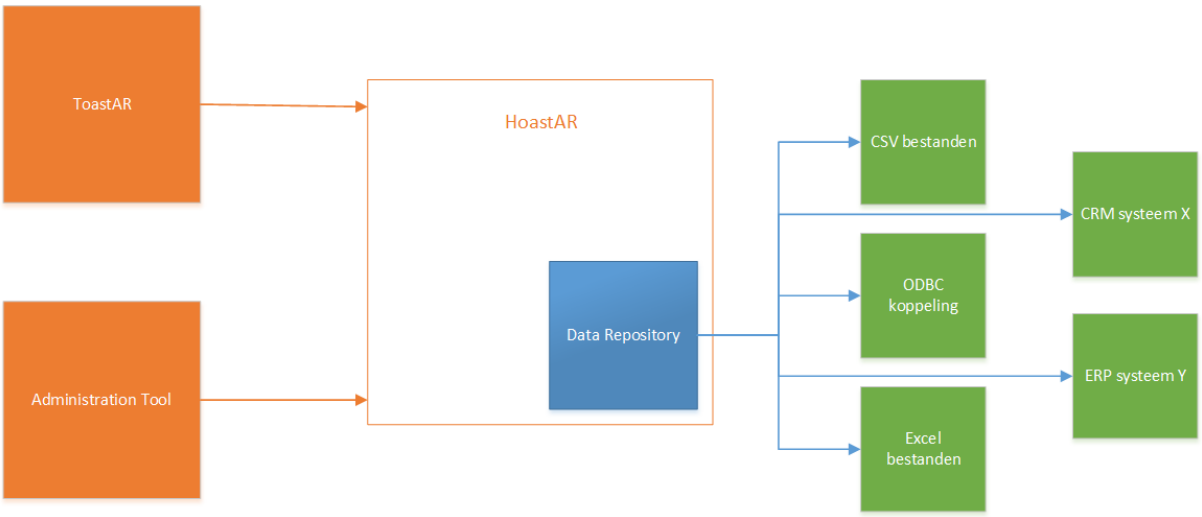
FR4 geeft aan in welke structuur de data van FR2 opgeslagen moet worden in de lokale database van FR1. Deze structuur is weergegeven in een klasse genaamd ContactData, waarbij de properties van deze klasse gezien moeten worden als gewenste kolommen in een database tabel. De gehele klasse is te zien in de bijlage van het Requirements document, in figuur 7 is een klein gedeelte te zien als voorbeeld.

```
public class ContactData
  {
    public string Zip { get; set; }
    public string City { get; set; }
    public string Company { get; set; }
    public string Custom0 { get; set; }
    public string Custom1 { get; set; }
```

Figuur 7. Deel van de ContactData klasse ter verduidelijking.

FR5 geeft aan dat er ook data de andere kant op gaat, namelijk gegevens vanuit HoastAR die de Data Repository nodig heeft om de replicaties in te kunnen plannen.

Om een duidelijk beeld te schetsen van de interactie van de Data Repository met andere systemen in de nieuwe situatie heb ik dit beschreven in het Requirements document met behulp van een Systeem Context Diagram. Hiermee wordt de afbakening ook duidelijker. Het gaat om de volgende beschrijving:
"De directory service wordt onderdeel van een Windows service genaamd HoastAR die de Data Repository intern zal aanroepen.
HoastAR zelf is bereikbaar voor client applicaties zoals ToastAR en een administrator tool.



Figuur 8. Systeem Context Diagram nieuwe situatie.

Hierboven is te zien wat de interactie is van de Data Repository met andere systemen en in welke context het zich bevindt. Zoals in deze figuur te zien is wordt de Data Repository een module van de HoastAR. De HoastAR wordt gebruikt als server van de client applicatie ToastAR en een Administration Tool. Alle gegevens die ToastAR en de administratie tool nodig hebben worden geleverd door de HoastAR.
Verder is te zien dat de Data Repository in de nieuwe situatie uit verscheidene databronnen data op kan halen."

Na de afbakening heb ik in het PvA beschreven wat de gebruikte projectmethodiek is voor dit project en waarom ik besloten heb dit te gebruiken, dit is hieronder te lezen.

"Tijdens het maken van het afstudeerplan heb ik gekeken naar verschillende ontwikkelmethodieken om te bepalen welke het best past bij de stageopdracht. Ten eerste is bepaald dat een iteratieve ontwikkelmethode de voorkeur heeft, sinds niet alle requirements vast staan bij aanvang van het project en er later nog requirements bij kunnen komen. Door een iteratieve methodiek te gebruiken kunnen requirements die later toegevoegd worden nog mee worden genomen in het huidige project en geïmplementeerd worden.
Verder is het bij deze opdracht gewenst incrementeel te ontwikkelen, om zo het softwareproduct in delen op te kunnen leveren. Als er een verkeerde inschatting is gemaakt tijdens de planning of als er zich onvoorziene vertragingen voordoen in het project kan op deze manier zeker zijn van werkende systeemdelen in plaats van een softwareproduct dat onvolledig is en misschien niet functioneert.

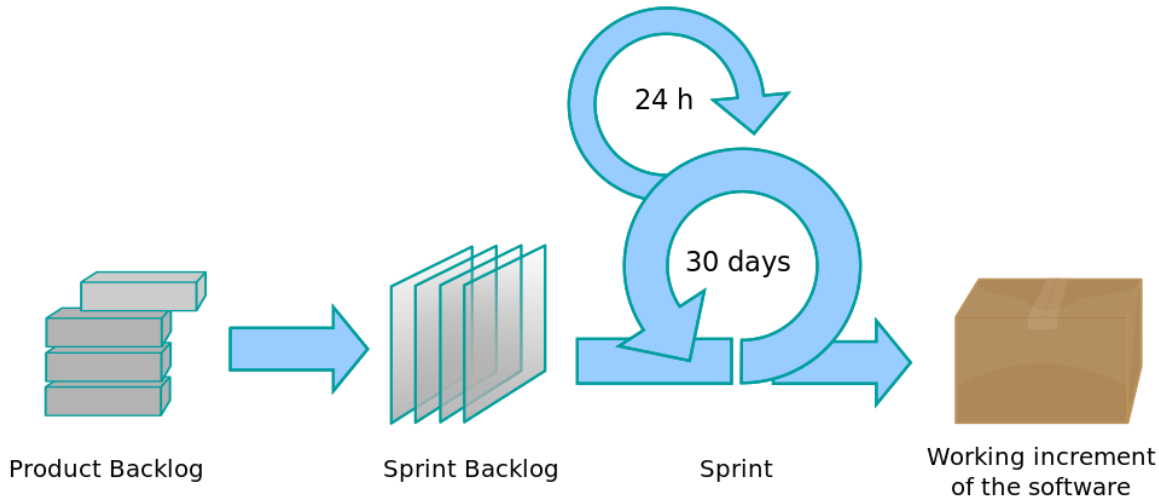
Bij het zoeken naar een incrementele en iteratieve software ontwikkelmethodiek vielen bepaalde methodieken zoals (Rational) Unified Process ((R)UP) af, sinds deze wel iteratief is maar niet incrementeel.
Er is uiteindelijk besloten om een Agile ontwikkelmethodiek te volgen, deze methodieken volgen het Manifest voor Agile Software Ontwikkeling[11], deze luidt als volgt:

- Mensen en hun onderlinge interactie boven processen en hulpmiddelen.
- Werkende software boven allesomvattende documentatie.
- Samenwerking met de klant boven contractonderhandelingen.
- Inspelen op verandering boven het volgen van een plan.

Hierbij wordt aangegeven dat dingen aan de rechterkant, zoals het volgen van een plan, niet onbelangrijk zijn.
De meeste Agile ontwikkelmethodieken focussen op incrementele oplevering van producten in korte iteraties.[12] Dit sluit daarom dus goed aan bij deze opdracht.

Het oog is hierbij gevallen op Scrum, een iteratieve en incrementele software ontwikkelmethodiek. Andere Agile methodieken vielen af omdat deze niet aansloten bij de opdracht. Bijvoorbeeld de ontwikkelmethodiek eXtreme Programming (XP) zou gebruik maken van Pair Programming, waarbij twee programmeurs samen achter één computer werken aan hetzelfde stuk code. Dit is onmogelijk sinds deze afstudeeropdracht individueel is.
Agile methodes die wel voldeden aan de eisen maar toch niet gekozen zijn, zoals DSDM, zijn afgevallen door persoonlijke voorkeur. Er is door mij al eerder ervaring opgedaan met Scrum en de werkwijze bevalt me.

Bij Scrum wordt het volgende proces gevolgd.[13]



Figuur 9. Het Scrum proces.

Een Scrum project bestaat uit meerdere iteraties waarin een werkend deel van de software wordt geleverd, wat Scrum dus iteratief en incrementeel maakt.

Iedere iteratie wordt een Sprint genoemd. Een sprint heeft een lengte van één tot vier weken.

De lijst met requirements van het project wordt de Product Backlog genoemd. Iedere sprint heeft een lijst met taken die tot een werkend systeemdeel moeten leiden, dit wordt de Sprint Backlog genoemd. Het is gebruikelijk aan het begin van de dag een meeting te houden met het team dat aan de sprint werkt, om zo de voortgang en de planning voor die dag door te nemen. Dit wordt ook wel de dagelijkse (Daily) Scrum of stand-up (meeting) genoemd.
Aan het eind van een sprint worden twee dingen gedaan, een Sprint Review en een Sprint Retrospective. In de Sprint Review wordt er bekeken welke geplande onderdelen af zijn gekregen en welke niet. Het is ook gebruikelijk een demo te geven aan de opdrachtgever van het werk dat af is deze sprint.
In de Sprint Retrospective wordt gereflecteerd op de uitgevoerde sprint, er worden twee vragen beantwoord:
Wat ging er goed deze Sprint?
Wat kan er verbeterd worden in de volgende Sprint?
Dit wordt gebruikt om een continue verbeterproces in stand te houden.

Binnen dit project zal er een aangepaste versie van Scrum worden gebruikt, de opdracht wordt immers door één developer uitgevoerd, namelijk ik als afstudeerder. In de planning is de sprint verdeling te zien voor dit project, verder wordt als Product Backlog het Requirements document gebruikt. Per Sprint zal er een Sprint Backlog gemaakt worden, met aan het eind van de Sprint een Review. De Retrospectives worden verwerkt in het Evaluatie hoofdstuk van het verslag. Er is afgesproken met de bedrijfsmentor om wekelijks de voortgang te bespreken, dit moment zal ook gebruikt worden als sprint review als er een sprint is afgerond.
Er wordt binnen de Development afdeling van Ask Roger! al gebruik gemaakt van stand-up meetings om het hele team van elkaar op de hoogte te houden waar aan gewerkt wordt en wat hierin de voortgang is. Ik doe hier ook aan mee om anderen op de hoogte te houden en mee te draaien in het Development team. Dit is ook handig om meer inzicht te krijgen in het HoastAR en ToastAR systeem van Ask Roger!."

Er is besloten geen User Stories of Unified Modeling Language (UML) use-cases op te stellen voor dit project. Dit verduidelijkt de requirements niet in dit project.

User stories zijn een toepassing binnen het Agile framework.[12] User stories zijn korte, simpele beschrijvingen van een functionaliteit, verteld vanuit het perspectief van de persoon die deze functionaliteit verlangt. Het is gebruikelijk dat zij de volgende structuur gebruiken: Als een <type gebruiker> wil ik <een bepaald doel>, < voor een bepaalde reden>.[14]

"Use-cases binnen UML zijn een middel om functionele eisen die gesteld worden aan het softwaresysteem weer te geven. Bij het maken van use-cases gaat men uit van de gebruiker van het systeem. Use-cases beschrijven hoe de gebruiker met het systeem om wil gaan."[15]

De uiteindelijke gebruikers van het systeem zouden gebruikers zijn van de ToastAR applicatie die via een interface bepaalde replicaties inplannen en zoeken naar contacten uit de lokale database, maar deze interface valt buiten de scope van het project. In figuur 8 op bladzijde 16 is dit goed te zien in het Systeem Context Diagram. De scope is de backend module Data Repository, die als module van HoastAR de HoastAR applicatie als gebruiker heeft.

Zowel user stories als use-cases worden verteld vanuit het perspectief van een gebruiker van het systeem om zo de requirements te koppelen aan de verschillende type gebruikers en hier meer duidelijkheid over te creëren. Sinds er maar één gebruiker is van de Data Repository in de huidige context, namelijk de HoastAR software, zou dit niet meer duidelijkheid scheppen.
De combinatie van het Requirements document met de requirements en de context beschrijving bij figuur 8 geeft voldoende duidelijkheid om geen Use Cases of User Stories nodig te hebben.

Verder is in het Projectmethodiek hoofdstuk het gebruik van Object-Oriented Design, Principles en Design Patterns beschreven en de reden hiertoe. Dit is hieronder beschreven.

"Naast deze ontwikkelmethodiek wordt er gebruik gemaakt van Object-Oriented Design (OOD). Dit gebruik ik omdat dit helpt een complex software probleem op te delen in kleinere onderdelen die ik per onderdeel gestructureerd kan oplossen, hiermee behoud ik makkelijker het overzicht. OOD focust zich op het concept van objecten en het toepassen hiervan in het ontwerpen van een applicatie. Het voordeel van het gebruik van objecten is dingen definiëren in het ontwerp die verantwoordelijk zijn voor zichzelf. Een object is iets met eigen verantwoordelijkheden. Een object weet wat voor type het is, de data in een object zorgt dat het object weet in welke toestand het zich bevindt en de code in het object zorgt dat het juist functioneert en de juiste functionaliteiten heeft.
Er kan ook gekeken worden naar objecten vanuit de verschillende perspectieven van Fowler[16]:

- Op een conceptueel niveau is een object een verzameling van verantwoordelijkheden.
- Op een specificatie niveau is een object een verzameling van methodes ofwel gedragingen die aangeroepen kunnen worden door andere objecten of het object zelf.
- Op een implementatie niveau is een object code en data en de computationele interacties tussen de twee.

Bij object georiënteerd ontwerpen komen klassen voort uit conceptuele objecten. Een klasse is een definitie van het gedrag van een object. Het bevat een beschrijving voor de data elementen die een object bevat, de methodes die een object bevat en de manier waarop de data elementen en methodes benaderd kunnen worden.

Omdat de data elementen van een object kunnen variëren kan ieder object van hetzelfde type andere data bevatten, maar zullen dezelfde functionaliteiten bevatten die gedefinieerd zijn door de methodes. Objecten zijn instanties van klassen.

Abstractie, encapsulatie en polymorfisme zijn drie belangrijke principes binnen OOD. Abstractie is een techniek om de complexiteit van computer systemen te arrangeren. Alleen relevante informatie wordt gebruikt in een bepaalde context, onderliggende details zijn te vinden op een lager abstractie niveau.
Encapsulatie is het verbergen van informatie binnen een object of module voor andere objecten en modules zodat deze informatie niet benaderd kan worden. Alleen informatie die door middel van een interface beschikbaar is gemaakt kan benaderd worden. Een interface beschermt de rest van een programma voor implementatie details die snel kunnen veranderen. Daarnaast is het een beveiliging van informatie, er wordt in een interface gekozen wat voor informatie andere objecten of modules mogen zien of gebruiken.
Polymorfisme is de mogelijkheid om iets conceptueels aan te roepen door middel van een abstracte referentie, maar verschillend gedrag terug te krijgen afhankelijk van de context.

Bij het ontwerpen van de applicatie en het herstructureren van de gemaakte code wordt er gebruik gemaakt van Design Patterns (DP) ofwel ontwerp patronen en Principles, principes binnen software ontwikkeling.
DP zijn herbruikbare ontwerp patronen die vaak voorkomende softwareproblemen binnen een bepaalde context oplossen. DP richten zich op een generale aanpak. Het zijn blauwdrukken om een ontwerp op te baseren, het zijn geen strikte regels waaraan gehouden moet worden.[17]

Principles zijn vuistregels die gevolgd kunnen worden om bepaalde problemen binnen het programmeren te voorkomen. Sommige DP zijn een implementatie van een bepaalde Principle. Een goed voorbeeld hiervan is al beschreven in de beschrijving van de huidige situatie, de Dependency Injection DP implementeert het Inversion of Control Principle.

Ten eerste zal ik in dit project de SOLID Principles volgen. Dit is een afkorting voor de vijf belangrijkste Principles binnen object-georiënteerd programmeren. Dit zijn:

- Single Responsibility Principle. Dit houdt in dat een klasse één verantwoordelijkheid moet hebben. Dit wordt ook wel het High Cohesion Principle genoemd, waarin de elementen binnen een klasse een hoge functionele verwantheid hebben aan elkaar ofwel een hoge cohesie. Dit moet de robuustheid van klassen bevorderen. De functionaliteiten van een klasse met meerdere verantwoordelijkheden kunnen makkelijker kapot gaan bij aanpassingen dan bij een klasse met één verantwoordelijkheid.
- Open Closed Principle. Dit houdt in dat klassen, modules en functies open moeten staan voor uitbreiding, maar gesloten voor verandering. Als een verandering aan een programma resulteert in ongewenste veranderingen in gerelateerde modules is dit slecht ontworpen, het programma is dan fragiel, onbuigzaam en onvoorspelbaar. Dit principe geeft aan dat modules ontworpen moeten worden met het doel dat deze nooit veranderen. Als er nieuwe functionaliteiten van een module bij komen, moet er nieuwe code

worden geschreven en mag de oude code niet worden aangepast. Om dit toe te passen spelen abstractie en polymorfisme een belangrijke rol.

- Liskov Substitution Principle. Dit houdt in dat klassen die afgeleid zijn van een basisklasse uitwisselbaar en vervangbaar moeten zijn voor hun basisklasse zonder veranderingen te veroorzaken aan het gedrag van het systeem. Het principe garandeert semantische interoperabiliteit van types in een hiërarchie.
- Interface Segregation Principle. Dit houdt in dat interfaces client specifiek moeten zijn. Clients moeten niet geforceerd worden om interfaces te implementeren die zij niet gebruiken. Dit principe voorkomt grote interfaces voor meerdere clients, waardoor die clients onnodige functies moeten implementeren die niet relevant zijn voor hun functioneren.
- Dependency Inversion Principle. Dit houdt in dat men afhankelijk moet zijn van abstracties en niet van implementaties. Details moeten afhankelijk zijn van abstracties, abstracties mogen niet afhankelijk zijn van details. Zo blijven modules op een hoger niveau onafhankelijk van implementatie details, dit bevordert Low Coupling.[18]

Als deze vijf principes gevolgd worden moeten deze samen zorgen voor een systeem dat makkelijk is te onderhouden en makkelijk is uit te breiden met nieuwe functionaliteiten in de toekomst.
Verder worden ook de volgende Principles toegepast:

- Don't Repeat Yourself (DRY). Dit houdt in dat ieder stukje informatie binnen een systeem maar één keer mag voorkomen. Het alternatief is gedupliceerde code dat op meerdere plekken voorkomt in de applicatie. Als deze code veranderd moet worden, moet de gedupliceerde code opgezocht worden en op meerdere plekken veranderd worden. Als het DRY principe gevolgd wordt hoeft code maar op één locatie aangepast te worden en blijft de code onderhoudbaar.
- Low Coupling ofwel Loose Coupling. Dit houdt in dat softwaremodules niet te veel om moeten gaan met andere modules, de modules moeten zo los ofwel onafhankelijk mogelijk van elkaar zijn. Coupling is de mate van afhankelijkheid tussen modules. Als de afhankelijkheid laag is dan is de kans klein dat andere modules aangepast moeten worden die afhankelijk zijn van een module als die module wordt aangepast, dit bevordert de onderhoudbaarheid van de code. Dependency Inversion is een specifieke vorm van decoupling, het streven naar een lage afhankelijkheid tussen modules.
- Abstractions Live Longer Than Details, ook wel Generalization Principle genoemd. Dit houdt in dat een algemene oplossing dat niet één maar meerdere problemen oplost beter is dan een specifieke oplossing. Specifieke oplossingen zijn meestal fragiel, bij veranderende requirements voldoen deze meestal niet meer. Een algemenere oplossing is stabieler en kan vaak hergebruikt worden voor meerdere situaties, dit voorkomt duplicatie en sluit goed aan bij het DRY principe.
- Crash Early, ook wel Fail Fast genoemd. Dit houdt in dat fouten in het systeem zo vroeg mogelijk opgespoord en gerapporteerd moeten worden om verborgen problemen met een applicatie te voorkomen. Verborgen

problemen kunnen tot ongewenst gedrag leiden van een systeem. Dit kan voorkomen worden door bijvoorbeeld validiteit van parameters te checken en Exceptions op te vangen en te loggen.[19]

Als deze principes toegepast worden dan bevorderen zij de onderhoudbaarheid van de code nog verder."

Ten slotte wordt in het Projectmethodiek hoofdstuk het gebruik van het .NET Framework beschreven en onderbouwd:
"Voor het ontwikkelen van de applicatie volg ik de Framework Design Guidelines voor het .NET Framework. Dit doe ik om een intern consistent programma te leveren die de best practices hanteert voor het .NET Framework. De Data Repository wordt een module van HoastAR, dit is geschreven in C# en maakt gebruik van het .NET Framework."[20,21]

Verder is er een Projectorganisatie hoofdstuk in het PvA te vinden met contactgegevens van bedrijfsmentoren, examinators en de afstudeerder zelf.
De mijlpaalproducten in het PvA staan met toelichting in hoofdstuk 6 in dit verslag.
Tot slot is er een planning te vinden waarin het project is onderverdeeld in sprints.
Met daarin de belangrijkste Sprint Backlog producten die per sprint aan bod komen:

"**Sprint 0 (3 weken):**

- Plan van Aanpak
- Globale Requirements vaststellen
- Systeem ontwerpen

Sprint 1 (2 weken):

- Basis van de data repository ontwerpen
- Basis van de data repository bouwen
- Basis van de data repository testen

Sprint 2 (2 weken):

- Lokale data opslag van de repository ontwerpen
- Lokale data opslag van de repository bouwen
- Lokale data opslag van de repository testen

Sprint 3 (2 weken):

- Replicator voor CSV databronnen ontwerpen
- Replicator voor CSV databronnen bouwen
- Replicator voor CSV databronnen testen

Sprint 4 (2 weken):

- Replicator voor Excel databronnen ontwerpen
- Replicator voor Excel databronnen bouwen
- Replicator voor Excel databronnen testen

Sprint 5 (2 weken):

- Replicator voor ODBC databronnen ontwerpen
- Replicator voor ODBC databronnen bouwen

- Replicator voor ODBC databronnen testen

Sprint 6 (4 weken):

- Custom replicator voor handmatige configuratie ontwerpen
- Custom replicator voor handmatige configuratie bouwen
- Custom replicator voor handmatige configuratie testen

Sprint 7(overige tijd):

- Replicator voor een specifiek CRM of ERP systeem ontwerpen
- Replicator voor een specifiek CRM of ERP systeem bouwen
- Replicator voor een specifiek CRM of ERP systeem testen"

In sprint 1 en 2 staat het maken van de basis van de applicatie ingepland. In de sprints hierna staan de verschillende replicators ingepland als hoofdproduct in de backlog voor deze sprints. Ik heb in de planning gekozen voor CSV, Excel en ODBC databronnen omdat deze functionaliteiten ook beschikbaar zijn in de huidige situatie, in de Meta Directory van Estos. Dit staat ook beschreven in het Requirements document:
"De data bronnen zijn CSV bestanden, Excel bestanden en systemen bereikbaar via ODBC. De huidige software die vervangen wordt bezit deze functies namelijk al."

In hoofdstuk 5, Afwijkingen afstudeerplan, wordt onder andere beschreven wat de uiteindelijke afwijkingen zijn in het uitgevoerde project ten opzichte van deze planning.

De requirements van de Data Repository heb ik in het Requirements document onderverdeeld in functionele requirements en niet-functionele requirements om de requirements overzichtelijker te maken.
"Functionele requirements zijn eisen aan het gedrag van het systeem, ze geven aan wat het systeem moet doen ofwel welke acties het moet uitvoeren. Niet-functionele requirements gaan over de kwaliteit van het systeem. Ze geven aan hoe goed het systeem moet werken."[22]

Alle functionele requirements zijn als volgt beschreven in het Requirements document:

- "FR1: De Data Repository moet lokaal data gestructureerd kunnen opslaan.
- FR2: De Data Repository moet data uit externe bron kunnen repliceren en deze lokaal opslaan.
- FR3: De Data Repository moet voorbereid zijn op uitbreidingen in de vorm van modules die data van specifieke externe data bronnen kunnen repliceren.
- FR4: De Data Repository moet lokale data kunnen leveren in een vooraf vastgesteld format. Dit is de ContactData klasse. (Zie bijlage)
- FR5: De Data Repository moet iedere replicatie kunnen inplannen aan de hand van instellingen geleverd vanuit HoastAR.
- FR6: De Data Repository moet een lijst met alle ingeplande replicaties kunnen leveren aan de HoastAR.

- FR7: De Data Repository moet iedere replicatie op een bepaalde start tijd kunnen uitvoeren met als optie een bepaald interval tussen de hierop volgende replicaties.
- FR8: De HoastAR moet kunnen achterhalen wat de specifieke gegevens zijn die ieder type replicatie module nodig heeft om te kunnen functioneren."

De meeste requirements zijn in sprint 0 vastgelegd, maar bijvoorbeeld FR6 t/m FR8 zijn in latere sprints erbij gekomen. De volgende niet-functionele requirements zijn vastgelegd in het Requirements document:

- "NFR1: Zoekopdrachten via de attributen SipAddress en Uri moeten binnen 0,1 seconde resultaat leveren op een database met 10.000 records.
- NFR2: Zoekopdrachten via de attributen Company en DisplayName moeten binnen 1 seconde resultaat leveren op een database met 10.000 records.
- NFR3: Tijdens het repliceren van data uit externe bronnen bij een normale serverload moet de lokale data 95% van de tijd beschikbaar zijn."

Hiermee is duidelijk dat zoeken op naam langer mag duren dan zoeken op telefoonnummers en e-mail adressen.
De functionele en niet-functionele requirements heb ik een codering gegeven om makkelijker te verwijzen naar een bepaalde requirement in dit verslag. Als bepaalde beslissingen of ontwerpen of implementaties gebaseerd zijn op een requirement zal er verwezen worden naar de requirement(s) in de vorm van een tabel met de requirement beschrijving(en), zoals bij de beschrijving van de afbakening van het PvA, om zo de traceerbaarheid van de requirements te waarborgen.

Verder heb ik de volgende punten onder het kopje Scope ondergebracht in het Requirements document:

- "De Data Repository wordt onderdeel van een Windows service die de Data Repository intern zal aanroepen. De Data Repository is niet vereist instellingen bij te houden voor de werking van de Windows service, maar moet wel zelf onderhoudend zijn.
- De beveiliging van de lokale data heeft een lagere prioriteit en valt buiten de scope."

Hiermee wordt benadrukt dat de Data Repository een module is van de HoastAR en zelf onderhoudend moet zijn, maar dat de HoastAR duidelijk zelf verantwoordelijk is voor het eigen functioneren.
Met dat de beveiliging buiten de scope valt betekent niet dat beveiliging vermeden moet worden, maar dat dit een lage prioriteit heeft. Bij het ontwerpen van de Data Repository is er rekening gehouden met de mogelijkheid om de lokale data in de toekomst alsnog te beveiligen.

De interactie met andere systemen dat ik heb beschreven in het Requirements document met behulp van een Systeem Context Diagram is bij de beschrijving van de afbakening uit het PvA al gebruikt als toelichting, dit is terug te lezen op bladzijde 16, in combinatie met figuur 8.

Verder heb ik ook nog het legacy systeem beschreven in het Requirements document, het systeem dat vervangen zal worden en hierdoor komt te vervallen:

"De Data Repository vervangt de huidige directory service genaamd MetaDirectory gemaakt door Estos die nu gebruikt wordt om data uit verschillende bronnen te repliceren en lokaal op te slaan."

Als de Data Repository geïntegreerd is als module van de HoastAR is er geen behoefte meer aan de Windows service MetaDirectory, dan heeft HoastAR de gewenste functionaliteiten zelf.

Voor het ontwerpen van de Data Repository heb ik een UML Klassediagram gemaakt. Hieronder is de theorie achter dit diagram te lezen.

"Een klassediagram toont de elementen waaruit het systeem bestaat en hun structurele, langdurige relaties. Het klassediagram bevat de klassen in het systeem, hun attributen, operaties en associaties. Het klassediagram is een model, het beschrijft de structuur waaraan de objecten uit die klassen gebonden zijn.

Een object is iets dat een zelfstandig bestaan leidt, in de werkelijkheid of in de geest. Belangrijk is dat het gezien kan worden als een zelfstandig iets. Over het algemeen is een object een afspiegeling van ofwel een fysiek ding uit de werkelijkheid, zoals een stoel of een auto, ofwel een concept uit die werkelijkheid, zoals een vakantie of een bankrekening.

Een klasse is een verzameling van objecten met overeenkomstige eigenschappen. De klasse beschrijving geeft de naam van de klasse en de beschrijving van de eigenschappen van de instanties. Deze eigenschappen worden verdeeld in attributen en operaties.

Een attribuut is informatie die door een object beheerd wordt. Bijvoorbeeld naam, geboortedatum, leeftijd en gewicht zijn attributen van objecten van de klasse Persoon. Attributen worden in het UML-klassediagram weergegeven in het middelste deel van de klasse, het bovenste deel bevat de naam van de klasse.

Een operatie, ofwel methode, beschrijft een service die een object levert. Een methode kan argumenten (parameters) hebben en een resultaat (returnwaarde) opleveren. Methodes worden in het UML-klassediagram weergegeven in het onderste deel van de klasse. Parameters worden genoteerd tussen ronde haken na de methodenaam. Het type van een eventuele returnwaarde wordt weergegeven achter een dubbele punt na de parameters.

Een associatie is een structurele relatie tussen twee klassen. Tussen twee klassen kunnen ook meerdere associaties liggen. In UML wordt een associatie aangegeven door een doorgetrokken lijn tussen twee klassen. De naam van de associatie wordt langs de lijn geschreven.
Bij een associatie-einde kan de multipliciteit weergegeven worden. De multipliciteit geeft het aantal instanties van de klasse weer dat de associatie aan dat associatie-einde mag bevatten. In UML kan de multipliciteit aangegeven worden door middel van cijfers, een sterretje of geen van beide. Als er geen multipliciteit is aangegeven bij een associatie-einde betekent dit een multipliciteit van één. Een sterretje betekent nul

of meerdere. Er kan een ander minimum in combinatie met een sterretje gebruikt worden, bijvoorbeeld 2..* betekent twee of meer."[23]

Ik heb gekozen om een klassediagram te maken om beter een structuur te kunnen visualiseren voor de applicatie tijdens het ontwerpen ervan. Daarnaast helpt een klassediagram het systeem te verduidelijken voor programmeurs die nog niet bekend zijn met de softwaremodule.
Het maken van een UML klassediagram sluit aan op OOD. UML wordt gebruikt om OOD grafisch weer te geven en het ontwerp makkelijker te begrijpen. Klassen uit het klassendiagram met bijbehorende attributen en methodes komen voort uit objecten op conceptueel niveau.[16,17,23] Het toepassen van OOD is dus ook een reden om een klassediagram te maken.

Omdat de Data Repository ontwerpen een iteratief proces is, worden er gedurende de verschillende sprints aanpassingen gedaan aan het UML klassediagram. Per sprint hoofdstuk in dit verslag worden soms delen van de toenmalige versie van het klassediagram gebruikt ter verduidelijking van het systeemdeel dat in die sprints gemaakt is. Om de laatste versie van het klassediagram in zijn geheel te bekijken verwijs ik je door naar bijlage 6.

Bij het maken van het Mastertestplan heb ik gebruik gemaakt van een template van TMAP. De relevante hoofdstukken verschillen per project sinds dit maatwerk is. Ik heb in de eerste versie beschreven dat ik gebruik maak van Unit Tests en de reden hiertoe, hieronder weergegeven.[24, 25]

"Het doel van een moduletest, ook wel unit test genoemd, is het testen van een specifiek onderdeel van een systeem. Een moduletest is een white-box test. Moduletesten richten zich op de elementaire bouwblokken in de code. Ze tonen aan dat de modules voldoen aan het technisch ontwerp.[1] Unit tests worden gemaakt in de vorm van code fragmenten die code van de te testen module aanroepen en controleren op een verwacht resultaat.

Als een systeemdeel getest is met unit tests kan dit systeemdeel eventueel hergebruikt worden met de zekerheid dat dit deel functioneert. Een ander voordeel is de her testbaarheid door middel van unit tests. Als er gewerkt is aan het systeem kan er snel getest worden of dit geen ongewenste effecten heeft gehad op eerder gemaakte modules van het systeem.

Voor de moduletesten is een Detailtestplan gemaakt, zie bijlage 8."

Het testplan is in latere sprints uitgebreid met andere testsoorten, de beschrijving hiervan is te vinden in de desbetreffende sprints.

4.1.3 Sprint 0 Review

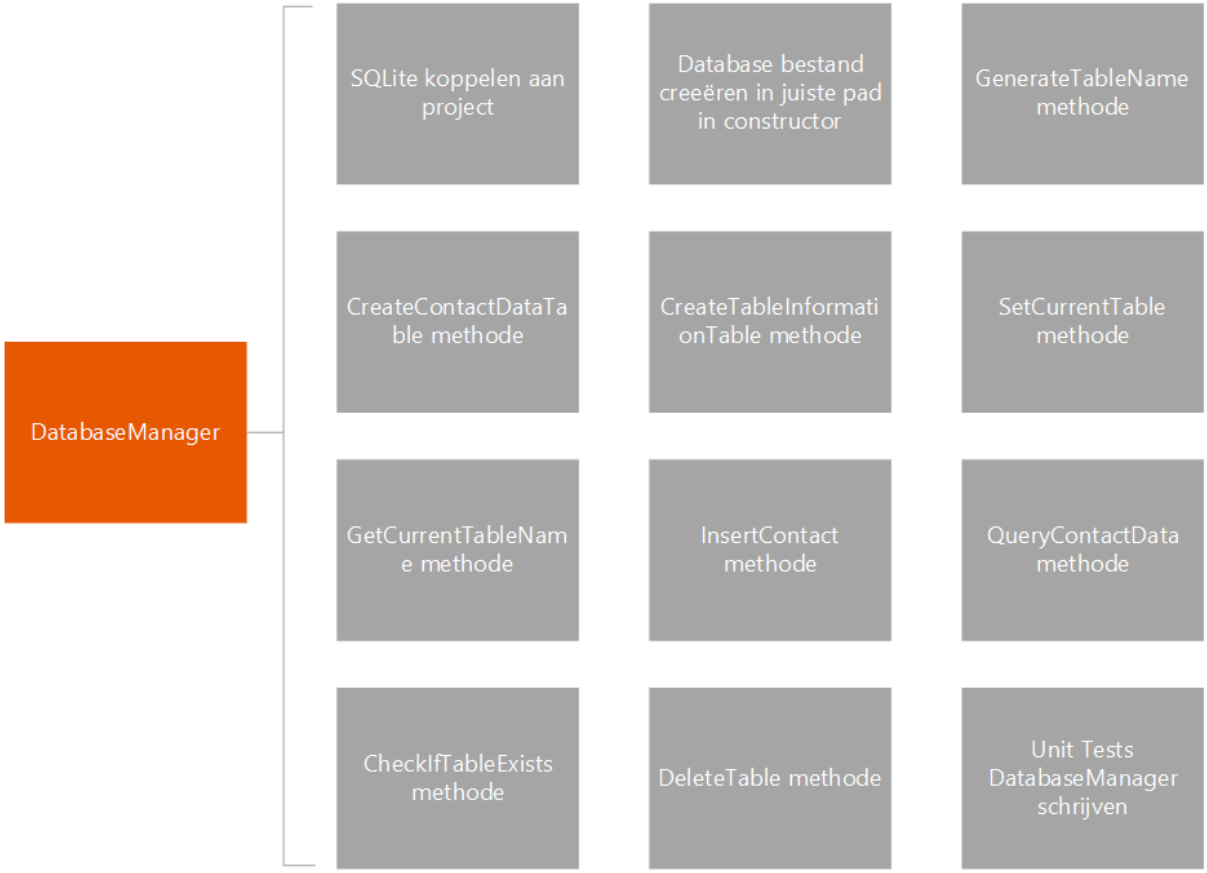
Uiteindelijk zijn alle geplande backlog producten voor sprint 0 op tijd af gekregen. Namelijk het PvA, het Requirements document en de eerste opzet van het Klassediagram. Dankzij deze documentatie en dit ontwerp kan er begonnen worden aan het ontwikkelen van het eerste systeemdeel in de volgende sprint.

4.2 Sprint 1: Basis van de applicatie

In sprint 1 is er gewerkt aan de basis van de applicatie. Deze bestaat uit de database, de replicatie planner en de klasse die de replicators aanroept. In de planning stond voor Sprint 2 de lokale dataopslag ontwerpen, bouwen en testen. Tijdens sprint 1 is hier al aan begonnen omdat de data opslag eigenlijk ook onderdeel is van de basis.

4.2.1 Backlog

De volgende backlog producten zijn toegevoegd aan Sprint 1:
De DatabaseManager klasse, de ContactData klasse, de ContactProvider klasse en de ReplicationScheduler klasse. Voor een uitleg van dit ontwerp, zie hoofdstuk 4.2.2. Verder is er ook gewerkt aan het Relational Representation Model (RRM) van de lokale dataopslag. De sprint backlog onderdelen zijn in figuur 10, 11, 12, 13 en 14 te zien.

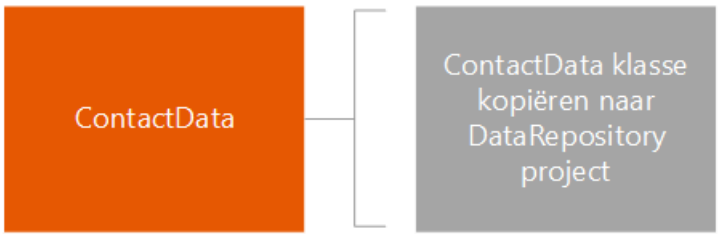


Figuur 10. Sprint 1 backlog DatabaseManager.

De DatabaseManager maken is onderverdeeld in een aantal taken. Ten eerste moet SQLite geïmplementeerd worden in de software om het te kunnen gebruiken in de

DatabaseManager klasse. Verder zal in de constructor gecheckt moeten worden of er al een database file aangemaakt is in de daarvoor gekozen map, zo niet moet deze daar aangemaakt worden. Dit zal in de constructor gecheckt worden, of aangeroepen worden vanuit de constructor, om zeker te zijn dat de klasse een database bestand heeft om mee te werken. Verder zijn er een aantal methodes die geïmplementeerd moeten worden:
De GenerateTableName methode zal een tabelnaam genereren en teruggeven aan de aanroeper van de methode.
CreateContactDataTable maakt een tabel aan met als tabelnaam de meegeleverde String. CreateTableInformationTable maakt de TableInformation tabel aan waarin de huidige tabellen worden bijgehouden van ieder domein.
SetCurrentTable zal een nieuwe tabel als huidige tabel van een bepaald domein instellen in de TableInformation tabel.
GetCurrentTableName verkrijgt de huidige tabelnaam van een bepaald domein uit de TableInformation tabel. InsertContact is de methode die een ContactData in een tabel zal zetten. QueryContactData is de methode die ContactData zal zoeken in de database binnen een bepaald domein aan de hand van bepaalde zoekargumenten. CheckIfTableExists kijkt of de tabel bestaat met de naam die is meegegeven aan deze methode. De DeleteTable methode verwijderd de tabel met de naam die is meegegeven aan de methode.

Daarnaast worden er unit tests geschreven voor de DatabaseManager klasse, om zo alle functionaliteiten te kunnen testen.

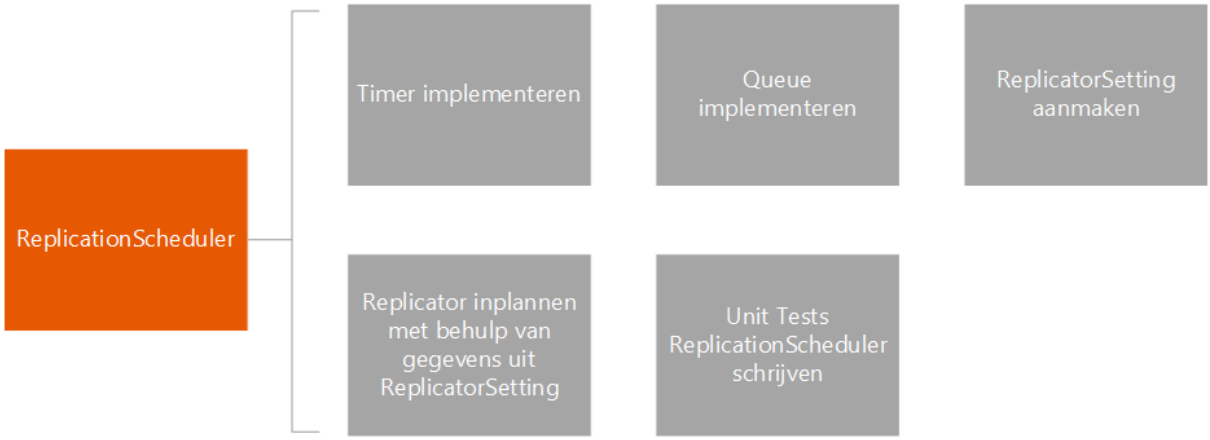


Figuur 11. Sprint 1 backlog ContactData.

De ContactData klasse zal vanuit HoastAR gekopieerd worden naar de console applicatie. Bij de integratie van de DataRepository in de HoastAR zal er weer verwezen worden naar de reeds bestaande ContactData klasse in HoastAR.

Figuur 12. Sprint 1 backlog ContactProvider

Om lokaal opgeslagen data in de vorm van ContactData te leveren aan HoastAR wordt een ContactProvider klasse gecreëerd. Hiervoor zal een interface gemaakt worden en een implementatie van de interface.



Figuur 13. Sprint 1 backlog ReplicationScheduler.

Bij het maken van de ReplicationScheduler zal er een timer geïmplementeerd worden om replicaties in te kunnen plannen. Naast deze timer zal er een queue systeem worden gemaakt om replicaties in een wachtrij te kunnen plaatsen.
Er zal een wrapper klasse worden gemaakt waarin alle instellingen worden bijgehouden die de scheduler nodig heeft om replicaties in te plannen. Met behulp van deze instellingen en de geïmplementeerde timer wordt de replicatie planner functionaliteit gemaakt.
Ten slotte zal voor deze klasse ook unit tests geschreven worden om de functionaliteiten te van de klasse te kunnen testen.

Figuur 14. Sprint 1 backlog RRM.

In het RRM zal de tabelstructuur van de lokale dataopslag uitgewerkt worden, hierin zijn de tabellen en hun attributen beschreven.

4.2.2 Uitvoering

De Data Repository wordt in de nieuwe situatie onderdeel van de Windows service HoastAR, in plaats van een externe service. In de huidige situatie moet de HoastAR communiceren met de MetaDirectory service via een protocol, dit is niet meer nodig bij de Data Repository.
De Data Repository zal een module worden van HoastAR en kan intern via C# code worden aangeroepen. Deze communicatie zal verlopen via verschillende interfaces, specifiek voor het soort communicatie tussen HoastAR en Data Repository. Dit zal ik doen om de Dependency Inversion Principle en Interface Segregation Principle te volgen.

Voordat ik kon beginnen met het implementeren van mijn ontwerp moest ik een keuze maken. Ik kon de Data Repository direct integreren in HoastAR tijdens het maken van de Data Repository of besluiten om de Data Repository eerst te maken in een losstaand project.
Ik heb besloten om de Data Repository eerst te maken in een apart project. Hiervoor heb ik gekozen omdat ik zo geheel kon focussen op de functionaliteiten van de Data Repository zonder last te hebben van onnodige complexiteit. Zolang ik interfaces maak voor communicatie met HoastAR maakt de onderliggende code van de Data Repository niet uit voor het integreren van de Data Repository module in HoastAR.

Het testen van de Data Repository door middel van systeemtesten is ook makkelijker in een apart project, omdat deze zelfstandig kan runnen zonder afhankelijk te zijn van de aanroep van HoastAR waar de Data Repository module in komt te staan. Er is gekozen om de Data Repository eerst te maken in een console applicatie. Hiervoor is gekozen omdat het testen van de functionaliteiten geen interface nodig heeft en de module gerund kan worden. Een systeemtest kan zo uitgevoerd worden voor de Data Repository zonder complexiteiten. Na het maken van Unit Tests voor gemaakte systeem delen heb ik gekozen om systeemtesten te doen na het maken van grote functionaliteiten van de Data Repository, zoals bijvoorbeeld replicaties kunnen inplannen. Ik heb hiervoor gekozen om zeker te zijn van het functioneren van het gemaakte systeem. Unit tests testen onderdelen van het systeem en het kan zich voordoen dat de individuele onderdelen die met unit tests getest worden goed functioneren, maar het geheel niet goed functioneert.
Om deze fouten op te vangen voer ik systeem testen uit. Ik heb dit ook toegevoegd en beschreven in het Mastertestplan:

"Het doel van een systeemtest is zoals de naam al aangeeft het hele systeem testen. De systeemtest is een black-box test. Testen gebeurt vaak aan de hand van de reeds ontwikkelde systeeminterfaces. De systeemtest toont aan dat het systeem werkt conform het functionele ontwerp.

Ik heb gekozen om systeem testen toe te passen om zeker te zijn dat het systeem werkt en geen fouten heeft die mogelijk langs de unit testen zijn geglipt."

Omdat ik besloten heb om de Data Repository module eerst in een console applicatie te maken en daarna in HoastAR te integreren is gekozen ook een systeem integratie test uit te voeren na deze integratie:

"Het doel van een systeem integratie test, ook wel ketentest genoemd, is het testen van het functioneren van een systeem in combinatie met andere systemen. Bij een ketentest wordt de samenwerking van het systeem met de aanliggende systemen getest. In de ketentest worden deze systemen vaak voor het eerst aan elkaar gekoppeld. De ketentest richt zich op het vinden van de fouten die ontstaan als systemen niet goed samenwerken.

Om zeker te zijn dat de DataRepository goed geïntegreerd wordt in de HoastAR en data via HoastAR aan de ToastAR geleverd kan worden wordt er een systeem integratie test uitgevoerd."

Zoals hierboven is te lezen zijn er dus twee testsoorten toegevoegd aan het Mastertestplan in deze sprint.

Naast het kiezen voor direct integreren of niet, moest ik deze sprint ook een keuze maken voor het soort database waar de lokale dataopslag in gedaan zou worden om aan requirement FR1 te kunnen voldoen.

Code requirement	Beschrijving requirement
FR1	De Data Repository moet lokaal data gestructureerd kunnen opslaan.

Ik kon zelf kiezen hoe ik de data lokaal zou opslaan. Bij het selectieproces moest ik een database vinden die past binnen de nieuwe context.
In de nieuwe situatie moet HoastAR de contact data uit externe bronnen ontvangen in een bepaald format.

Code requirement	Beschrijving requirement
FR4	De Data Repository moet lokale data kunnen leveren in een vooraf vastgesteld format. Dit is de ContactData klasse.

Daarnaast moet de lokaal opgeslagen data met een bepaalde snelheid geleverd kunnen worden.

Code requirement	Beschrijving requirement
NFR1	Zoekopdrachten via de attributen SipAddress en Uri moeten binnen 0,1 seconde resultaat leveren op een database met 10.000 records.
NFR2	Zoekopdrachten via de attributen Company en DisplayName moeten binnen 1 seconde resultaat leveren op een database met 10.000 records.

Om de data snel in dit format te kunnen leveren heb ik gekozen om de data op te slaan in dezelfde structuur als de ContactData klasse. Om geen tijd te verliezen met het omzetten van de data naar de gewenste structuur bij het leveren van deze data aan HoastAR, heb ik gekozen de klantdata uit externe bronnen na het repliceren om te zetten naar de ContactData structuur alvorens de data lokaal wordt opgeslagen.

Per databron zal er dus één tabel komen in de database in de structuur van de klasse ContactData. Met de bovenstaande context kon ik concluderen dat de Data Repository geen database management systeem (DBMS) nodig heeft die om moet kunnen gaan met een grote complexe data structuur. Hierdoor vielen grote DBMSen zoals SQL Server en Oracle af bij de keuze voor het soort database.

Daarnaast hebben geïntegreerde databases de voorkeur boven client-server databases. Alleen de Data Repository hoeft toegang te hebben tot de database, dus als enige client is er geen behoefte aan een DBMS die te bereiken is voor meerdere clients via een bepaald protocol. Bij een geïntegreerde database hoeft de Data

Repository ook niet met andere software buiten de Windows server HoastAR te communiceren om bij de lokaal opgeslagen data te komen, het database bestand kan direct benaderd worden door de Data Repository.

Bij het zoeken naar een geschikt geïntegreerd database systeem is mijn oog gevallen op SQLite. SQLite is open source, compacte, stabiele en autonome relationele database. Hiermee is het een goedkope oplossing die niet veel ruimte in beslag neemt en ook de gewenste database functies aan kan, er worden namelijk geen complexe database operaties uitgevoerd.[27]
Daarnaast is SQLite een zeer populaire geïntegreerde database engine. Het wordt gebruikt in ieder Android apparaat, iPhone en iOS apparaat, iedere Mac en Windows 10 machine, iedere Firefox, Chrome en Safari web browser en in nog veel meer toepassingen.[28]

Om de data op te slaan in de SQLite database met de structuur van ContactData is de volgende tabel structuur gemaakt en in gedocumenteerd het RRM:

"Hieronder is de tabelstructuur te zien van de lokale dataopslag van de Data Repository module van HoastAR.

Onderstreepte attributen zijn de Primary Keys van de tabellen.
Dikgedrukte attributen zijn attributen die geen NULL waarde mogen bevatten.

Tabel	Attributen
Contact*	(**Id**, Zip, City, Company, Custom0, Custom1, Custom2, Custom3, Custom4, Custom5, Custom6, Custom7, Custom8, Custom9, DisplayName, FirstName, LastName, **SipAddress**, Street, Uri, LocalContact, ExtensionUrl, ContactSource)
TableInformation	(**DC**, **CurrentTable**)

**Iedere bron waarvan data wordt gerepliceerd heeft een eigen tabel met contacten. De tabelnaam van deze tabel wordt gegenereerd door de Data Repository. Deze tabellen hebben dus een random naam om naam conflicten te vermijden met tabelnamen in de database.*

Zoals hierboven is te lezen, heeft iedere replicatie bron een eigen Contact tabel met attributen in de structuur van de klasse ContactData, om te voldoen aan requirement FR4. In de TableInformation tabel wordt bijgehouden wat de huidige tabelnaam is voor een specifieke databron."

Er is gekozen om geen Foreign Keys te laten afdwingen door de database zelf. Hier heb ik voor gekozen om complicaties te vermijden met het verwijderen en toevoegen van Contact tabellen door database constraints.
Om aan de requirement NFR3 te voldoen heb ik namelijk gekozen om bij het repliceren van data de nieuwe data in een nieuwe tabel te zetten, de nieuwe tabelnaam in de TableInformation tabel te zetten met een update en de oude tabel, als er een oude tabel is voor deze data bron, te verwijderen.

Code requirement	Beschrijving requirement
NFR3	Tijdens het repliceren van data uit externe bronnen bij een normale serverload moet de lokale data 95% van de tijd beschikbaar zijn.

SQLite zet een 'exclusive lock' op een database bestand als er data wordt weggeschreven naar dit bestand. Dit houdt in dat er heel even geen gegevens opgevraagd ofwel gelezen kunnen worden uit de database.[29]

Ik heb gekozen de nieuwe data in een nieuwe tabel te zetten om ingewikkelde rollbacks te vermijden als er fouten optreden bij het invoeren van nieuwe gerepliceerde data in de database. Dit probeer ik te vermijden om het database bestand zo min mogelijk in een locked toestand te hebben en de beschikbaarheid te verhogen.

Door gerepliceerde data in een nieuwe tabel te zetten blijft de data uit een bepaalde bron ook intern consistent, er wordt geen contact data stuk voor stuk geüpdatet in de huidige tabel waarbij er tijdens het updaten data uit de tabel opgevraagd kan worden dat op dat moment oude en nieuwe gegevens bevat.
Er wordt namelijk ContactData één voor één in de database toegevoegd om de exclusive lock niet te lang te laten duren en de beschikbaarheid te verhogen voor requirement NFR3. De beschikbaarheid van de data is in dit systeem belangrijker dan de duur van een replicatie. Replicaties van databronnen gebeuren in de huidige situatie dagelijks, in de nieuwe situatie wordt verwacht dat de replicaties over het algemeen dagelijks blijven gebeuren. Er wordt dus vaker data gelezen uit de lokale database dan naartoe weggeschreven.

Bij het ophalen van ContactData gegevens in de lokale database afkomstig van een bepaalde databron wordt de TableInformation tabel geraadpleegd om de huidige tabelnaam te achterhalen. Door deze tabelnaam te updaten in de TableInformation tabel na het invoeren van een nieuwe tabel kan er snel overgestapt worden naar de nieuwste contact informatie. Nieuwe data aanvragen worden zo direct doorverwezen naar de nieuwste tabel.

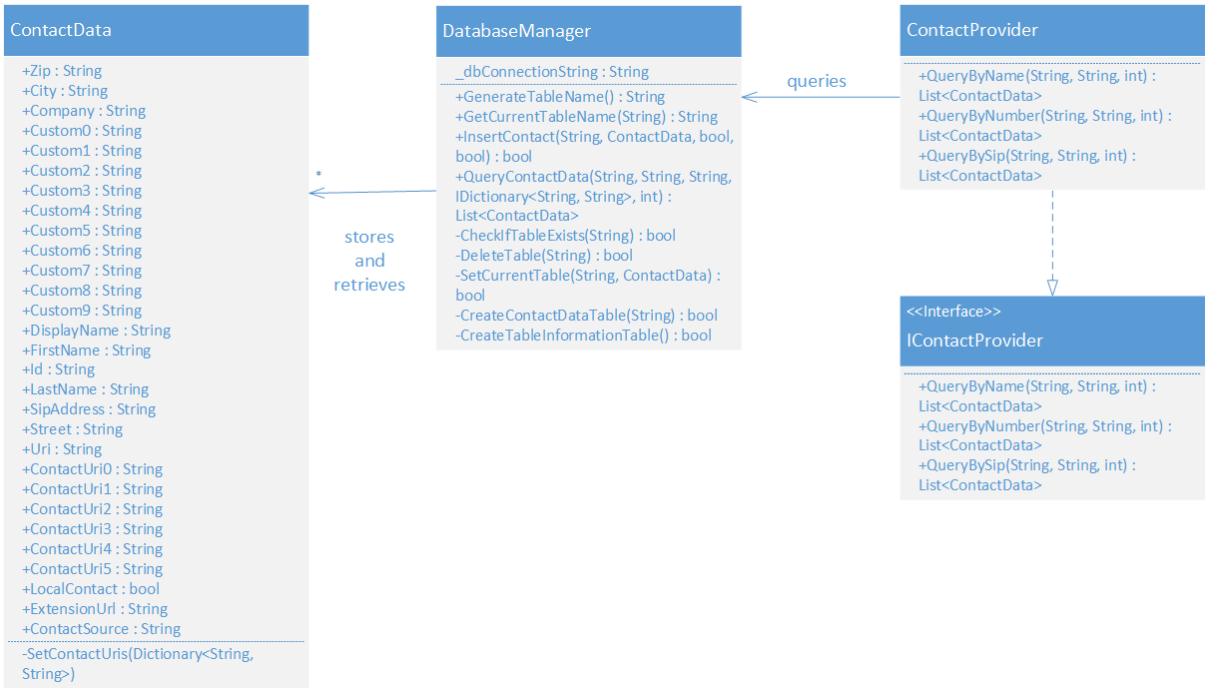
Om de data integriteit te behouden die Foreign Key constrains in database systemen geven zorg ik dat in de applicatie bij het omwisselen van de oude met de nieuwe tabel eerst wordt gekeken of de oude tabel bestaat. Als deze inderdaad bestaat sla ik de tabelnaam op in een variabele, update de tabelnaam in de InformationTable en verwijder ik de oude tabel. Dit wordt gedaan in de SetCurrentTable methode in de DatabaseManager klasse.

Als deze database oplossing onvoldoende blijkt te zijn in de toekomst qua concurrency tijdens het toevoegen van gerepliceerde data in de database, kan er gebruik worden gemaakt van de Write-Ahead Logging (WAL) optie van SQLite versie 3.7. Hiermee kan er tegelijkertijd data opgehaald en weggeschreven worden. Deze optie is 1 tot 2% langzamer voor applicaties die veel gegevens ophalen en minder wegschrijven, zoals dit project, vergeleken met de standaard instellingen van SQLite.[30]

Om aan de requirements FR1 en FR4 te voldoen zijn de klassen DatabaseManager, ContactData, IContactProvider en ContactProvider ontworpen in het klassendiagram.

Code requirement	Beschrijving requirement
FR1	De Data Repository moet lokaal data gestructureerd kunnen opslaan.
FR4	De Data Repository moet lokale data kunnen leveren in een vooraf vastgesteld format. Dit is de ContactData klasse.

Hieronder is de klassenstructuur te zien waarmee de lokale dataopslag en het aanbieden van data hieruit wordt geregeld.



Figuur 15. Deel van het initiële klassendiagram van de data opslag en aanbod.

Om aan de requirements FR1 te voldoen is de klasse DatabaseManager ontworpen. Ik ben als eerst begonnen met het maken van een implementatie van deze klasse omdat dit de basis vormt voor de Data Repository module.

In de DatabaseManager klasse wordt het lokale database bestand aangemaakt en worden alle queries op de database uitgevoerd.
Ik heb in de constructor van de DatabaseManager het maken van het database bestand geschreven. Hierbij wordt eerst gecheckt of het database bestand al bestaat, zo niet wordt deze aangemaakt. Het lokale database bestand wordt aangemaakt in een map binnen de installatiemap van HoastAR. Het volledige pad naar deze map wordt achterhaald met behulp van het opvragen van het pad naar het uitvoerende bestand ofwel de executable van de HoastAR, waarna dit pad wordt omgevormd naar de database map. Het is namelijk bekend waar de database map staat in relatie tot de bin map van de applicatie.
Door het pad van de database map te bepalen via de locatie van het uitvoerende

bestand maakt de mappenstructuur niet uit van de machine waarop HoastAR is geïnstalleerd. Op deze manier worden potentiële problemen voorkomen met het aanmaken van het database bestand.

Om de tabelnamen bij te houden per databron heb ik de CreateTableInformationTable methode geschreven, die een Boolean waarde teruggeeft waarmee wordt aangegeven of deze taak gelukt is of niet. De GenerateTableName genereert een random tabelnaam die gebruikt kan worden om een tabel aan te maken om ContactData in op te slaan voor gerepliceerde data. Deze tabel wordt aangemaakt via de CreateContactDataTable methode, die als parameter een string waarde nodig heeft. Deze waarde is de tabelnaam voor de nieuwe tabel. De methode GenerateTableName is een publieke methode, zo kan een andere klasse een nieuwe tabel aanmaken met de gegenereerde naam en weten in welke tabel contactinformatie toegevoegd zal worden. De methodes CreateContactDataTable en CreateTableInformationTable zijn privé. Ik heb hiervoor gekozen om andere klassen niet de mogelijkheid te geven direct nieuwe tabellen aan te kunnen maken, dit is immers de verantwoordelijkheid van de DatabaseManager klasse.
Hiermee voldoe ik aan het Single responsibility Principle.

Om ContactData van een gerepliceerde bron op te slaan in de database heb ik de publieke InsertContact methode gemaakt. Deze methode slaat één ContactData per keer op, de klasse die gerepliceerde ContactData wilt opslaan in de database zal deze methode meerdere keren aan moeten roepen om alle data op te slaan in de tabel. Ik heb hiervoor gekozen om de beslissing om contactinformatie stuk voor stuk op te slaan te waarborgen. Via de Boolean parameters wordt aangegeven of de ContactData de eerste of laatste in de collectie is of niet. Bij de eerste zal een tabel worden aangemaakt, dit wordt gedaan door intern de CreateContactDataTable aan te roepen. Als de laatste ContactData succesvol is opgeslagen in de database wordt de methode SetCurrentTable aangeroepen.
De SetCurrentTable methode zorgt ervoor dat de nieuwe tabel in de InformationTable tabel staat en de oude tabel verwijderd wordt, zodat ContactData aanvragen worden geleid naar de nieuwste tabel. Er wordt intern eerst de methode GetCurrentTableName aangeroepen om de huidige tabelnaam te achterhalen die hoort bij de respectieve databron. Deze naam wordt opgeslagen in een variabele, om later in de functie te gebruiken. Hierna wordt de TableInformation tabel geüpdatet met de nieuwe tabelnaam als er al een oude tabel was, zo niet wordt een nieuwe row toegevoegd in de tabel met de databron en bijbehorende tabelnaam. Als er een oude tabel is wordt deze hierna verwijderd via de privé methode DeleteTable, die als parameter de te verwijderen tabelnaam heeft.

De privé methode CheckIfTableExists wordt intern aangeroepen in de twee create methodes die tabellen aanmaken. CreateTableInformationTable en CreateContactDataTable geven beide namelijk een Boolean waarde terug die aangeeft of het aanmaken van de methode gelukt is. Omdat de SQLiteCommand.ExecuteNonQuery een waarde met de hoeveelheid aangepaste database rows teruggeeft en dit niet gebruikt kan worden om te checken of een tabel

aanmaken gelukt is, wordt dit gedaan in de CheckIfTableExists methode door middel van een query.

Om ContactData op te vragen uit de lokale database heb ik de methode QueryContactData gemaakt. In de parameters wordt aangegeven welke attributen geselecteerd moeten worden, wat de Where clausule is in de query, wat de waarden zijn voor in deze clausule in de vorm van een IDictionary en wat het maximum aantal resultaten zijn die moeten worden teruggegeven. In de methode wordt aan de hand van deze gegevens ContactData opgehaald uit de lokale database en teruggegeven.

Om aan de requirement FR4 te voldoen heb ik de klasse ContactProvider ontworpen en is de ContactData klasse gebruikt van HoastAR.

Code requirement	Beschrijving requirement
FR4	De Data Repository moet lokale data kunnen leveren in een vooraf vastgesteld format. Dit is de ContactData klasse.

Zoals in de beschrijving van de DatabaseManager klasse al een paar keer genoemd, om aan dit format ofwel structuur te voldoen wordt data geleverd in de vorm van ContactData. Gegevens uit de lokale database worden per contact in een ContactData object gezet, waarbij de attributen in de klasse gezet worden via de Set methodes van de desbetreffende properties.
Aan de ContactData klasse heb ik zelf niets ontworpen, deze klasse wordt al gebruikt in HoastAR.

Om contactgegevens te leveren aan HoastAR is er een interface gemaakt genaamd IContactProvider. Hiermee voldoe ik aan het Dependency Inversion Principle en het Interface Segregation Principle.

De methodes in deze interface zijn afgeleid van het volgende fragment uit het Requirements document:
"De interface waarmee HoastAR de data repository service zal ondervragen ligt al grotendeels vast. Er zal gezocht worden naar contacten via naam, telefoonnummer of SIP adres."

Om deze functionaliteit te kunnen leveren heb ik de methodes QueryByName, QueryByNumber en QueryBySip gezet in de interface.

De klasse die de IContactProvider interface implementeert heb ik ContactProvider genoemd. De drie methodes roepen intern de methode QueryContactData aan van de klasse DatabaseManager.

Bij het maken van methodes die collecties gebruiken als parameter of return type zijn de volgende Framework Design Guidelines gevolgd:

- "✓ **DO** use the least-specialized type possible as a parameter type. Most members taking collections as parameters use the IEnumerable<T>interface.

- **X DO NOT** return null values from collection properties or from methods returning collections. Return an empty collection or an empty array instead."[31]

Bij methodes die een Dictionary nodig hebben is bijvoorbeeld als parameter IDictionary gebruikt, om de parameters zo abstract mogelijk te houden. Dit is bijvoorbeeld toegepast bij de methode QueryContactData in de DatabaseManager klasse. Dit is ook weer toepassing van het Dependency Inversion Principle.

Als er alleen over een collectie heen geïtereerd hoeft te worden, dan wordt er gebruik gemaakt van Enumerable in plaats van bijvoorbeeld een List. Dit wordt gedaan om de immutability te waarborgen. Als een collectie aangepast kan worden is het mutable, zo niet heet dit immutable.
Verder wordt bij List, een implementatie van de interface IEnumerable, de collectie geladen in het geheugen. Door Enumerable te gebruiken worden lijsten met gegevens pas in het geheugen geladen als over de gegevens geïtereerd wordt.[32]

De gemaakte unit tests zijn gedocumenteerd in het document Detailtestplan moduletest. Er zijn hierbij logische en fysieke testgevallen beschreven. De beschrijving voor de fysieke testgevallen is als volgt:

"Om de fysieke testgevallen te beschrijven zal ik niet alle gemaakte unit tests naar dit document kopiëren om het aantal pagina's te beperken. Hieronder is de basis structuur van mijn unit tests te zien.

```
[TestMethod]
public void TestNaam()
{
    try
    {
        // Arrange

        // Act

        // Assert

    }
    finally
    {
        // Cleanup after test

    }
}
```

Figuur 16. Basis structuur unit test.

Als er operaties tijdens de unit test worden uitgevoerd die handmatige opschoon werkzaamheden vereisen wordt de bovenstaande structuur gebruikt. Denk hierbij aan unit tests die tabellen aanmaken in de database die na de test weer verwijderd moeten worden. In het try blok wordt de daadwerkelijke test uitgevoerd en het finally blok wordt na de test uitgevoerd.

In de meeste gevallen is er geen finally blok nodig om opschoon werkzaamheden te verrichten, dan wordt alleen de structuur gebruikt binnen het try blok.

Onder Arrange versta ik het initialiseren van klassen en het aanmaken van objecten die gebruikt zullen worden in de unit test om bepaalde functionaliteiten na te bootsen.
Onder Act versta ik het uitvoeren van de test door middel van het aanroepen van specifieke methodes.
Onder het Assert gedeelte wordt het resultaat onder het Act gedeelte van de test vergeleken met het verwachtte resultaat en dit bepaalt of de test is geslaagd of niet.

Een simpel voorbeeld van een gemaakte unit test is de GenerateTableNameNotNullOrEmpty test methode die het logische testgeval LT01 implementeert.

Code testgeval	Beschrijving testgeval
LT01	De gegenereerde tabelnaam mag niet een lege String zijn of een NULL waarde zijn.

Hieronder is de unit test te zien.

```
[TestMethod]
public void GenerateTableNameNotNullOrEmpty()
{
    // Arrange
    string actualResult;
    bool resultBool;
    DatabaseManager dbm = new DatabaseManager(_logger);

    // Act
    actualResult = dbm.GenerateTableName();
    resultBool = isNullOrEmpty(actualResult);

    // Assert
    Assert.IsFalse(resultBool);
}
```

Figuur 17. GenerateTableNameNotNullOrEmpty unit test.

Hier is goed te zien dat ik eerst een DatabaseManager klasse initialiseer onder Arrange en de klasse onder Act gebruik om de te testen methode aan te roepen. Onder Assert vergelijk ik het resultaat uit de methode."

Voor de DatabaseManager klasse zijn de volgende logische testgevallen opgesteld.

DatabaseManager

Code testgeval	Beschrijving testgeval
LT01	De gegenereerde tabelnaam mag niet een lege String zijn of een NULL waarde zijn.
LT02	Een ContactData moet correct opgeslagen kunnen worden in de database.
LT03	Er moet een nieuwe tabel aangemaakt worden voordat de eerste ContactData in een collectie van gerepliceerde data daarin opgeslagen wordt.
LT04	Na het invoeren van alle ContactData in een nieuwe tabel moet de nieuwe tabelnaam in de InformationTable tabel staan.
LT05	Als een oude tabel vervangen is voor een nieuwe tabel met de laatste versie van contactgegevens uit een databron, dan moet de oude tabel verwijderd worden uit de database.
LT06	Een verkeerde query moet een SQLiteException veroorzaken.
LT07	Een ContactData invoeren zonder een tabelnaam te leveren moet een Exception veroorzaken.
LT08	Bij het opvragen van de huidige tabelnaam van een bepaalde databron moet de juiste naam geleverd worden.
LT09	Bij het opvragen van de huidige tabelnaam uit een onbekende bron moet een lege String geleverd worden.
LT10	Bij het zoeken naar een ContactData in de database moet de correcte ContactData geleverd worden.
LT11	Het zoeken naar een ContactData in een niet bestaande tabel moet een SQLiteException veroorzaken.
LT12	Het zoeken naar een ContactData zonder alle benodigde parameters mee te leveren moet een Exception veroorzaken.
LT13	Bij het zoeken naar ContactData zonder gevonden resultaten moet een lege collectie geleverd worden.

Na de unit tests gemaakt te hebben heb ik een systeemtest uitgevoerd om de lokale dataopslag functionaliteiten te testen.

Ik heb na het maken en uitvoeren van de unit tests en systeemtest uitgevoerd, hieronder de beschrijving.

"Bij het maken van de applicatie heb ik een systeemtest uitgevoerd als ik een grote of belangrijke functionaliteit gemaakt had. Dit heb ik gedaan als vangnet om fouten op te sporen die niet opgevangen zijn door de gemaakte unit tests.

Bij systeemtesten wordt in principe een heel systeem getest. De systeemtest is een black-box test. Testen gebeurt vaak aan de hand van de reeds ontwikkelde systeeminterfaces. De systeemtest toont aan dat het systeem werkt conform het functionele ontwerp. De systeemtest is vaak het domein van de bouwende partij en wordt uitgevoerd voordat hij het systeem oplevert aan de acceptant.[26]

In sprint 1 heb ik een systeemtest uitgevoerd om de functionaliteiten van de lokale opslag te testen. Ik heb hierbij methodes uit de DatabaseManager aangeroepen vanuit de main methode van de console applicatie. Ik heb hierna het database bestand benaderd met een SQLite browser programma om te zien of de data daadwerkelijk werd opgeslagen in de database."

Na de systeemtest was ik begonnen met de DatabaseManager klasse refactoren. Hierbij heb ik een aantal methodes herschreven of geherstructureerd.

Bij het refactoren heb ik de methodes aangepast die queries opstellen en uitvoeren. Om SQL injection te voorkomen wordt nu gebruik gemaakt van geparameteriseerde queries. De queries worden vooraf bepaald en de parameters later toegevoegd met de SQLite.Command.Parameters.Add() methode.

Het bepalen van het database pad en het aanmaken van het database bestand heb ik in een aparte methode gezet die nu wordt aangeroepen in de constructor.

De daadwerkelijke connectie met de SQLite database bij het aanmaken, updaten en het verwijderen van tabellen vindt nu plaats in een aparte methode. Zie figuur 18. De methode ExecuteNonSelectQuery wordt hiervoor aangeroepen door methodes die tabellen aanmaken, data in tabellen updaten en tabellen verwijderen. Deze methode geeft een Boolean waarde terug om aan te duiden of de database operatie gelukt is.

```
-ExecuteNonSelectQuery(String,
IDictionary<String, object>, bool, String) :
bool
```

Figuur 18. ExecuteNonSelectQuery toevoeging aan het klassendiagram.

Met deze methode voldoe ik aan het Generalization Principle en volg ik het DRY principle.

4.2.3 Sprint 1 Review

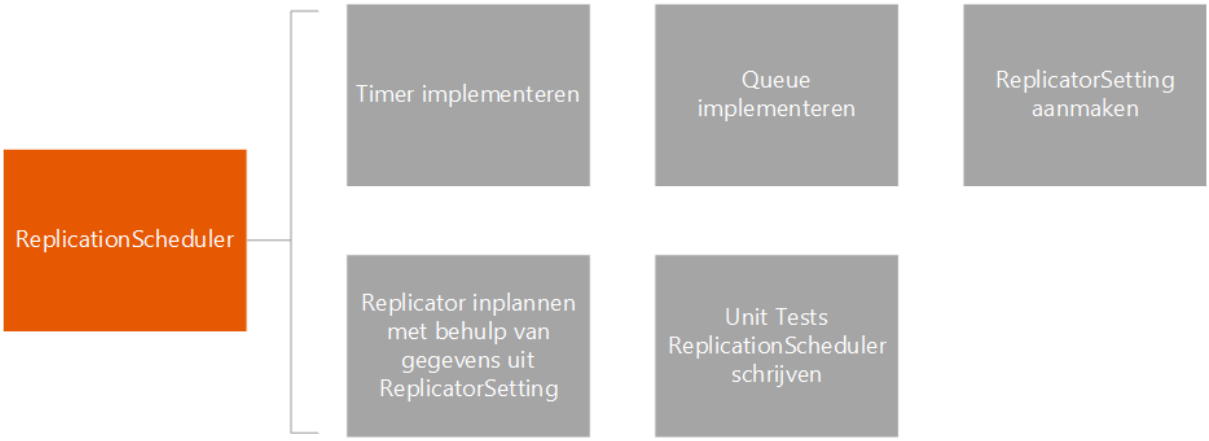
Er is in deze sprint al de lokale data opslag ontworpen, gemaakt en getest. Dit was in de originele planning ingepland voor sprint 2. Dit is wel ten koste gegaan van het ontwerpen, maken en testen van de ReplicationScheduler klasse die ook in de sprint backlog stond voor sprint 1 en het ontwerpen van de overige klassen die vallen onder de 'basis' van de applicatie. Het maken van de ReplicationScheduler en andere klassen wordt doorgeschoven naar sprint 2. De Unit Tests maken voor de ContactProvider klasse wordt ook doorgeschoven.

4.3 Sprint 2: Scheduler

Zoals ik in de sprint review van sprint 1 beschreven heb, heb ik in sprint 2 de overige klassen gepland die vallen onder de 'basis' van de applicatie. Onder de basis versta ik alle klassen buiten de klasse(n) verantwoordelijk voor het repliceren van data uit externe data bronnen.

4.3.1 Backlog

Het gaat hier specifiek om de klassen ReplicationScheduler, ReplicatorSetting en ReplicatorSettingsProvider.



Figuur 19. Sprint 2 backlog ReplicationScheduler.

Bij het maken van de ReplicationScheduler zal er een timer geïmplementeerd worden om replicaties in te kunnen plannen. Naast deze timer zal er een queue systeem worden gemaakt om replicaties in een wachtrij te kunnen plaatsen.
Er zal een wrapper klasse worden gemaakt waarin alle instellingen worden bijgehouden die de scheduler nodig heeft om replicaties in te plannen. Met behulp van deze instellingen en de geïmplementeerde timer wordt de replicatie planner functionaliteit gemaakt.
Ten slotte zal voor deze klasse ook unit tests geschreven worden om de functionaliteiten te van de klasse te kunnen testen.

Figuur 20. Sprint 2 backlog ReplicatorSettingsProvider.

Bij het maken van de ReplicatorSettingsProvider moet er een interface gemaakt worden en deze implementeren, daarnaast moet er een unit test voor geschreven worden.

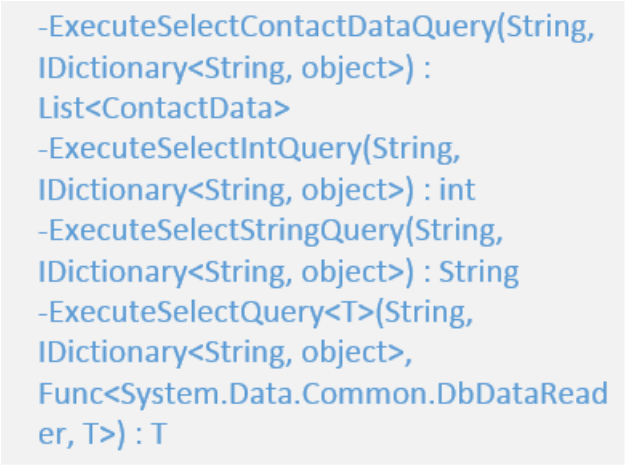
4.3.2 Uitvoering

Ik heb in het begin van sprint 2 eerst de unit tests geschreven voor de ContactProvider klasse. Hieronder zijn de logische testgevallen hiervan beschreven.

ContactProvider

Code testgeval	Beschrijving testgeval
LT14	Bij het zoeken op naam naar ContactData moet de correcte ContactData geleverd worden.
LT15	Bij het zoeken op nummer naar ContactData moet de correcte ContactData geleverd worden.
LT16	Bij het zoeken op SIP adres naar ContactData moet de correcte ContactData geleverd worden.

Hierna ben ik verder gegaan met het refactoren van de code in de DatabaseManager klasse. De daadwerkelijke connectie met de database bij het opvragen van gegevens uit tabellen heb ik nu ook uit de desbetreffende methodes gehaald en in een aparte methode gezet. Zie figuur 21.



Figuur 21. Select methodes toevoegingen aan het klassendiagram.

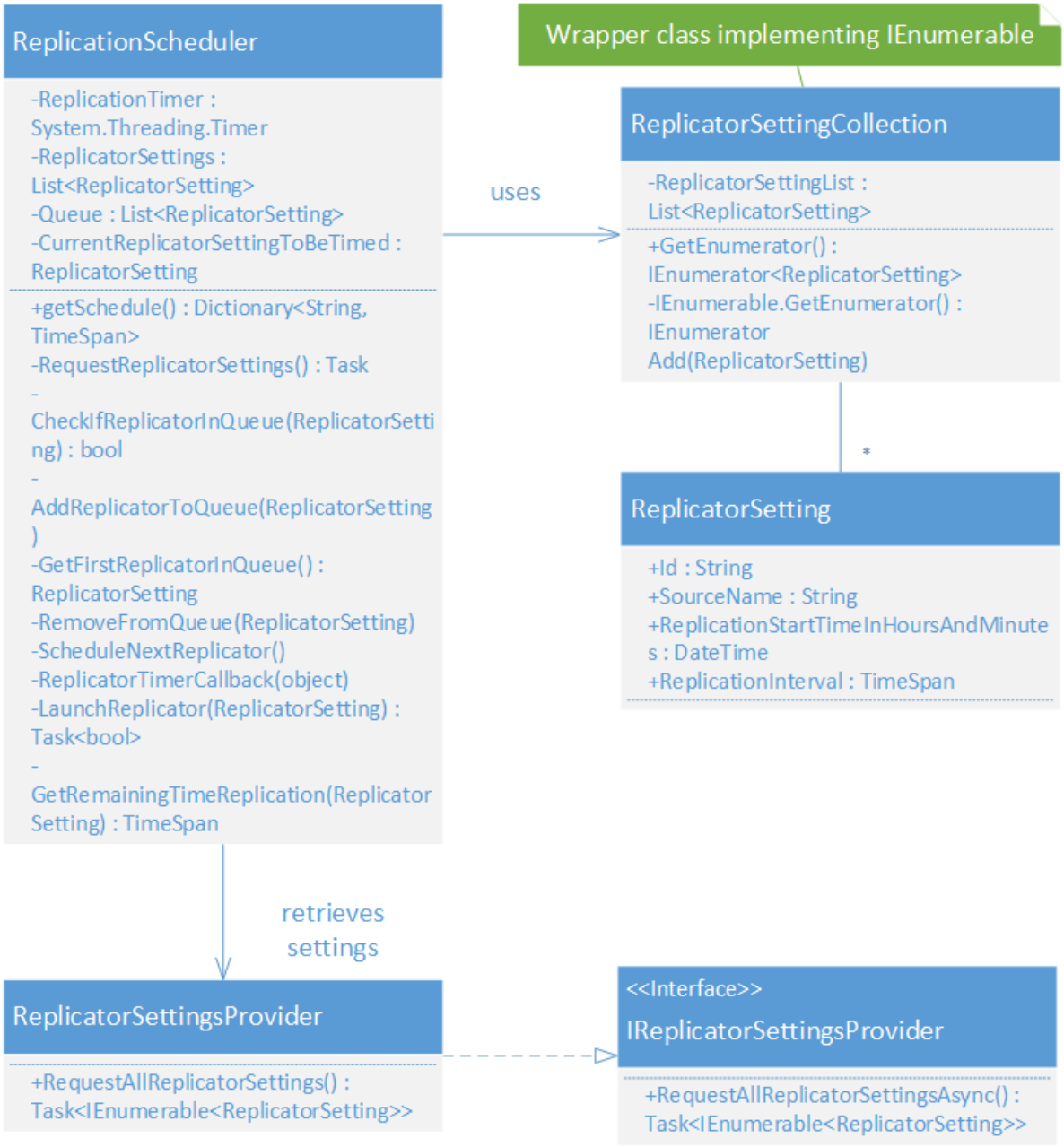
De methode ExecuteSelectQuery wordt aangeroepen door de methodes die data op willen vragen uit de database. Er kunnen meerdere types data gevraagd worden uit deze methode, dus is hier een zogenoemde Generic Method van gemaakt van het type T. Deze generieke methodes maken het mogelijk een abstractie te maken voor methodes die verschillende type waardes terug verwachten. Door een type als parameter mee te geven weet de methode welk type waarde teruggegeven moet worden.[33]
Daarnaast heeft deze methode nu ook een Func() delegate als parameter, dit is een manier om een taak binnen een methode te delegeren naar een andere methode. Een delegate is een type dat een referentie representeert naar een methode met een specifieke parameter lijst en return type.[34] De delegate methodes worden aangeroepen voor het uitlezen van de data reader. Na het uitlezen voegt de delegate deze data toe aan een lijst voor het specifieke type variabele(n) die uitgelezen zijn uit de database via de reader. Er zijn drie delegate methodes gemaakt: ExecuteSelectIntQuery, ExecuteSelectStringQuery en ExecuteSelectContactDataQuery voor de types data die opgehaald kunnen worden uit de database. Ik heb gebruik

gemaakt van delegates om zo de implementatie van ieder specifiek type gescheiden te houden van de database connectie logica. Als er een nieuw type verwacht wordt in de toekomst, kan hier een nieuwe delegate implementatie voor geschreven worden zonder aanpassingen te hoeven doen aan de database connectie code.[35]
Ook met deze methodes voldoe ik aan het Generalization Principle en volg ik het DRY principle.

Om aan de requirement FR5 te voldoen heb ik de klassen ReplicationScheduler, ReplicatorSetting, ReplicatorSettingCollection, IReplicatorSettingsProvider en ReplicatorSettingsProvider ontworpen.

Code requirement	Beschrijving requirement
FR5	De Data Repository moet iedere replicatie kunnen inplannen aan de hand van instellingen geleverd vanuit HoastAR.

De klassenstructuur van deze klassen is in figuur 22 weergegeven.



Figuur 22. Deel van het initiële klassendiagram van de klassen verantwoordelijk voor het inplannen van replicaties.

Om replicaties in te plannen heb ik de ReplicationScheduler klasse ontworpen. Om instellingen op te vragen vanuit HoastAR heb ik de interface IReplicatorSettingsProvider ontworpen, de klasse ReplicatorSettingsProvider implementeert deze interface. Om alle instellingen te versturen en ontvangen tussen meerdere klassen heb ik hier de klasse ReplicatorSetting voor gemaakt met daarin alle instellingen als attributen.
Om een collectie met ReplicatorSettings te hebben waar alleen over geïtereerd hoeft te worden heb ik de wrapper klasse ReplicatorSettingCollection gemaakt die IEnumerable implementeert.

Er zijn tijdens deze sprint requirements bij gekomen, het gaat hier om FR6 en FR7.

Code requirement	Beschrijving requirement
FR6	De Data Repository moet een lijst met alle ingeplande replicaties kunnen leveren aan de HoastAR.
FR7	De Data Repository moet iedere replicatie op een bepaalde start tijd kunnen uitvoeren met als optie een bepaald interval tussen de hierop volgende replicaties.

FR6 geeft aan dat de HoastAR een overzicht moet kunnen leveren met alle replicaties die ingepland staan.

FR7 houdt in dat een replicatie op een bepaald tijdsstip ingepland moet kunnen worden. Daarnaast moet er bij het inplannen van de daaropvolgende replicaties rekening worden gehouden met een bepaald interval.

Requirements FR6 en FR7 zijn erbij gekomen toen ik al bezig was aan het timer systeem binnen de ReplicationScheduler klasse.
In de klasse ReplicationScheduler wordt gebruik gemaakt van timers om op bepaalde momenten data te repliceren. Er zijn verschillende soorten timers beschikbaar in het .NET Framework, er moest een keuze gemaakt worden tussen:

- "System.Timers.Timer (this topic): fires an event at regular intervals. The class is intended for use as a server-based or service component in a multithreaded environment; it has no user interface and is not visible at runtime.
- System.Threading.Timer: executes a single callback method on a thread pool thread at regular intervals. The callback method is defined when the timer is instantiated and cannot be changed. Like the System.Timers.Timer class, this class is intended for use as a server-based or service component in a multithreaded environment; it has no user interface and is not visible at runtime.
- System.Windows.Forms.Timer: a Windows Forms component that fires an event at regular intervals. The component has no user interface and is designed for use in a single-threaded environment.
- System.Web.UI.Timer: an ASP.NET component that performs asynchronous or synchronous web page postbacks at a regular interval."[36]

Er is na de requirement FR7 gekozen voor de System.Threading.Timer timer. De eerste keuze was de System.Timers.Timer, maar die bleek alle exceptions te negeren in de elapsed event handler.[37] Hiermee kunnen bugs moeilijk opgespoord worden. Ook heeft System.Threading.Timer de optie om een start tijd in te stellen en System.Timers.Timer niet.

Verder gebruikt System.Timers.Timer intern een System.Threading.Timer, dus System.Timers.Timer is een wrapper klasse die ooit bedoeld is om het makkelijker te maken timers te gebruiken.[38]

De timer is een attribuut van de ReplicationScheduler klasse om te zorgen dat er altijd een referentie naar is. Als ik dit niet zou doen kan het zijn dat de garbage collector

van het .NET Framework de timer verwijdert uit het geheugen. Als dit gebeurt breekt het planningssysteem dat afhankelijk is van de timer.

De methode RequestReplicatorSettings vraagt alle instellingen op aan HoastAR via de RequestAllReplicatorSettings methode van de ReplicatorSettingsProvider klasse.
De GetSchedule methode heb ik gemaakt om aan FR6 te voldoen. Ik heb gekozen om bij deze methode per replicator de tijd die nog over is voor de volgende replicatie terug te geven in het type TimeSpan. De HoastAR heeft zo de mogelijkheid om via functies van deze klasse makkelijk de tijd om te zetten in dagen, uren, minuten etc. dat meer moeite zou kosten als de tijd gegeven zou worden in bijvoorbeeld een double.

De ScheduleNextReplicator methode is de methode waar daadwerkelijk de replicaties worden ingepland met behulp van de ReplicatorSetting attributen. De timer wordt in deze methode aangeroepen. ScheduleNextReplicator gebruikt intern de methode GetRemainingTimeReplication om de eerstvolgende replicatie te kunnen inplannen. Als de timer verloopt wordt de methode ReplicatorTimerCallback aangeroepen, deze roept op zijn beurt de methode LaunchReplicator aan om een specifieke replicatie te starten.

Om aan de requirement NFR3 te voldoen heb ik een queue systeem gemaakt in de ReplicationScheduler klasse.

Code requirement	Beschrijving requirement
NFR3	Tijdens het repliceren van data uit externe bronnen bij een normale serverload moet de lokale data 95% van de tijd beschikbaar zijn.

Om de database niet te veel te belasten met write operaties en de database locks te minimaliseren is het volgende gekozen. Er is een queue gemaakt waar replicators in gezet worden waar de start tijd van verstreken is, maar er al een andere replicatie bezig is. De geplande replicaties hoeven niet meteen uitgevoerd te worden, een minuut vertraging op een dagelijkse replicatie is bijvoorbeeld geen probleem.
Om mogelijk concurrency problemen op dit vlak compleet te vermijden is er dus gekozen om replicaties één voor één uit te voeren en hier een queue systeem voor te maken in de ReplicationScheduler.

Als er al een replicator bezig is met repliceren dan wordt de replicator die aan de beurt is in een queue gezet. Wanneer een replicator klaar is met repliceren en de data is opgeslagen in de lokale database, dan wordt de eerste replicator in de queue opgestart als er een queue is.

De queue zelf wordt intern bijgehouden in een List met ReplicatorSettings.
De methode CheckIfReplicatorInQueue wordt gebruikt om te weten of een specifieke replicatie in de queue staat door een ReplicatorSetting als parameter mee te geven.
De methode GetFirstReplicatorInQueue wordt gebruikt om de eerstvolgende replicator op te vragen in de queue. Als er geen queue is wordt een lege

ReplicatorSetting geleverd om de code te houden aan de Framework Design Guidelines.
Met de AddReplicatorToQueue en RemoveFromQueue methodes worden replicaties in en uit de queue gehaald door deze in en uit de List te halen.

Bij het maken van de unit tests voor de ReplicationScheduler en de ReplicatorSettingsProvider heb ik de volgende logische testgevallen beschreven.

ReplicationScheduler

Code testgeval	Beschrijving testgeval
LT17	Bij het opstarten van de ReplicationScheduler moet deze automatisch alle ReplicatorSettings ophalen en replicaties inplannen.
LT18	Bij het opvragen van het planningsschema moeten correcte tijden geleverd worden.
LT19	Als het tijd is om te repliceren moet een replicatie aangeroepen kunnen worden.

ReplicatorSettingsProvider

Code testgeval	Beschrijving testgeval
LT23	Er moeten ReplicatorSettings opgevraagd kunnen worden aan de HoastAR.

Na de unit tests heb ik wederom een systeemtest uitgevoerd om het planningssysteem te testen.

"In sprint 2 heb ik een systeemtest uitgevoerd om de functionaliteiten te testen van het replicatie planningssysteem. Ik heb dit gedaan door in de timer callback functie die wordt aangeroepen als een timer verloopt, een regel te schrijven naar het output scherm in de debugger. Daarnaast heb ik via de main methode van de console applicatie het replicatieschema opgevraagd en deze data ook naar het output scherm weg te schrijven in de debugger."

Na het maken van de unit tests voor de gemaakte klassen en het uitvoeren van de systeemtest heb ik een probleem waar ik op geattendeerd in de gemaakte unit tests van alle klassen tot dusver aangepakt.
Om in Unit Tests private methodes te testen in C# moet er gebruik worden gemaakt van Reflection. Ik heb geen andere methode gevonden hiervoor binnen C#. Het nadeel van deze methode is dat deze private methodes dan via een string met de naam erin worden opgeroepen. Hiermee gaat de onderhoudbaarheid van de applicatie sterk achteruit. Als bijvoorbeeld een methode naam van een private methode wordt veranderd, wordt dit niet automatisch doorgevoerd in de unit tests omdat die naam in een string staat en de compiler hier niets mee kan. Dan zou handmatig opgezocht moeten worden waar de methode wordt aangeroepen en daar

worden aangepast.
ik heb hierom gekozen om iets in te leveren op encapsulatie om dit probleem te verhelpen. Sommige private methodes zullen internal worden gemaakt zodat deze direct aan te roepen zijn in het Unit Test project. Er is namelijk expliciet toestemming gegeven aan het Unit Test project om de internals te zien. Hiermee is het ook in staat internal klassen direct aan te roepen.
Deze methodes blijven onbereikbaar buiten de assembly, net als de internal klassen. Hiermee zijn de kosten van de verminderde encapsulatie niet te hoog voor de verbetering van de onderhoudbaarheid van de code. Het veroorzaakt geen beveiligingsrisico, alleen is de code binnen de assembly niet dummy proof meer. Nu kunnen de methodes aangeroepen worden door iedere klasse binnen de assembly. Toekomstige ontwikkelaars die met deze code werken zullen hierop gewezen moeten worden.
Om onderscheid te houden tussen methodes die bedoeld zijn om aangeroepen te kunnen worden door andere klassen binnen de assembly en de methodes waarbij dit niet de intentie is, zijn deze methodes public. Hoewel er geen verschil is in de bereikbaarheid van public methodes en internal methodes binnen een internal klasse, wordt er op deze manier de intentie van de methodes duidelijker gehouden. Dit zal de encapsulatie niet garanderen, maar maakt de intenties duidelijker voor toekomstige ontwikkelaars.
Daarnaast wordt deze oplossing ook al gebruikt in de huidige HoastAR applicatie, waarbij de verschillende onderdelen van de HoastAR eigen assemblies hebben.

4.3.3 Sprint 2 Review

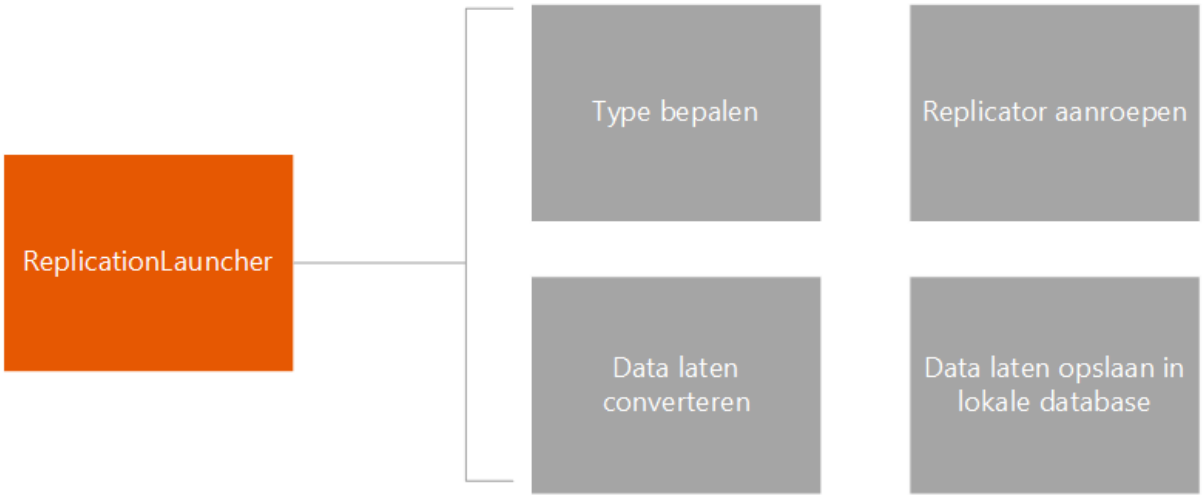
Sprint 2 is uiteindelijk een halve week uitgelopen ten opzichte van de planning mede dankzij het refactoren van de ReplicationScheduler klasse. De backlogproducten uit deze sprint zijn wel af gekregen.

4.4 Sprint 3: CSV replicator

In deze sprint is begonnen aan de replicatie functionaliteiten. Er is begonnen met de mogelijkheid om gegevens uit CSV bestanden te repliceren.

4.4.1 Backlog

Het gaat hier specifiek om de klassen ReplicationLauncher, IReplicator, CSVReplicator en ReplicatedDataConverter.

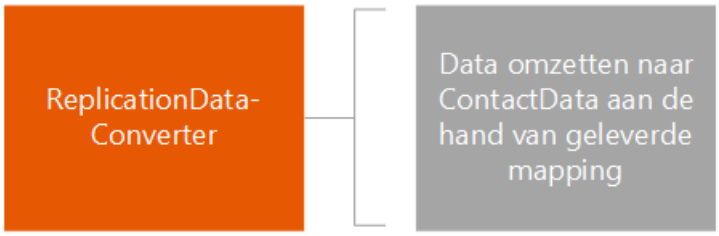


Figuur 23. Sprint 3 backlog ReplicationLauncher.

Bij het maken van de ReplicationLauncher klasse moet bepaald worden welk type replicator wordt aangeroepen. Daarnaast moet de replicator worden aangeroepen, data laten converteren en het laten opslaan.

Figuur 24. Sprint 3 backlog CSVReplicator.

De CSVReplicator moet de geleverde instellingen deserialiseren en de data kunnen repliceren.



Figuur 25. Sprint 3 backlog ReplicationDataConverter.

De data converter moet aan de hand van een mapping de geleverde data omzetten naar ContactData.

4.4.2 Uitvoering

Om aan de requirements FR2 en FR3 te voldoen heb ik gekozen om de interface IReplicator te maken.

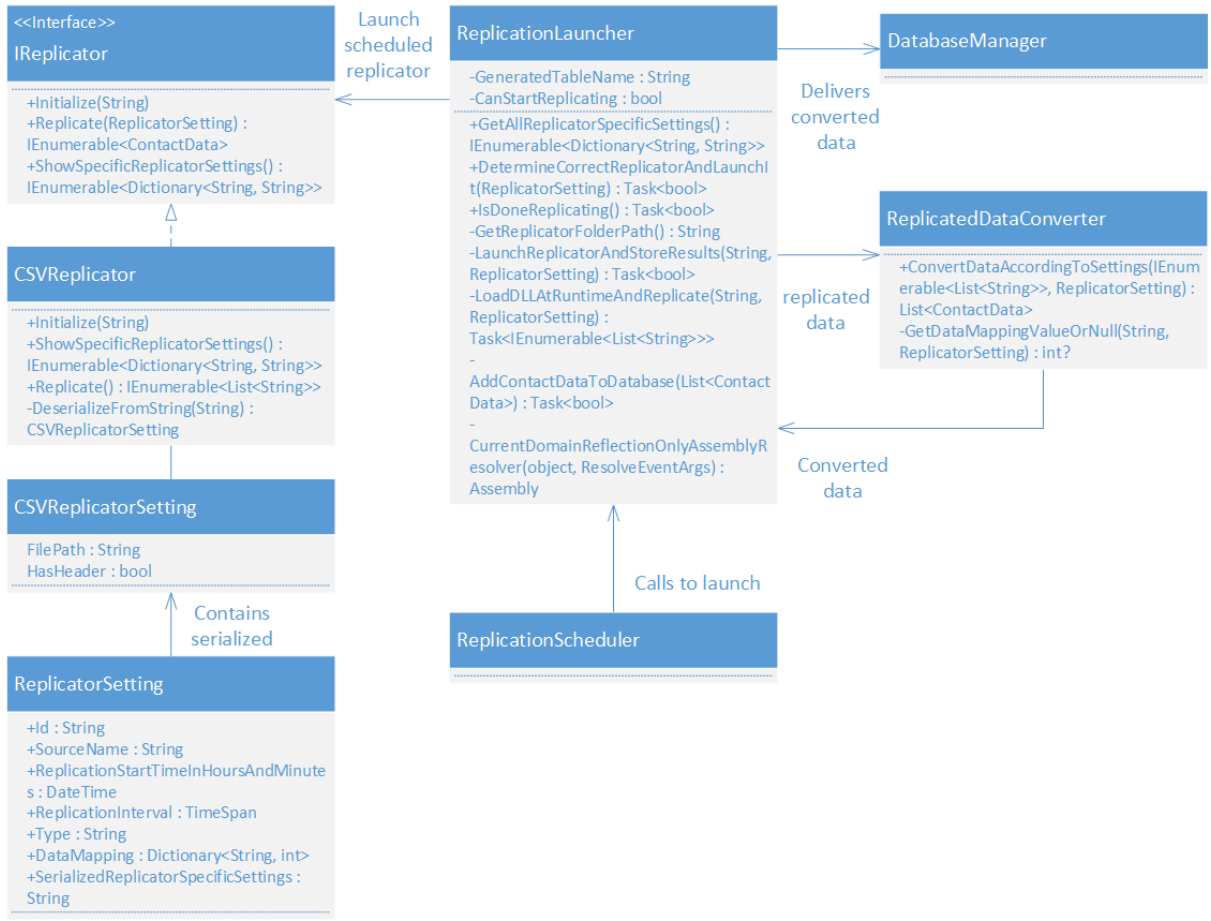
Code requirement	Beschrijving requirement
FR2	De Data Repository moet data uit externe bron kunnen repliceren en deze lokaal opslaan.
FR3	De Data Repository moet voorbereid zijn op uitbreidingen in de vorm van modules die data van specifieke externe data bronnen kunnen repliceren.

Ik heb hiervoor gekozen om de repliceermodules een vaste structuur te geven zodat de Data Repository afhankelijk is van deze abstractie in plaats van de implementaties van deze modules. Dit is gedaan om de Dependency Inversion Principle te volgen. Iedere repliceer module, hierbij Replicator genoemd, focust op het repliceren van data voor een specifiek type databron. De CSVReplicator heb ik bijvoorbeeld gemaakt om data te repliceren uit een CSV bestand als databron.

Ik heb gekozen om iedere replicator in een aparte Dynamic Linking Library (DLL) te zetten. Er zijn hier een aantal voordelen aan verbonden.[39]
Een groot voordeel is dat een replicator pas geladen wordt als deze nodig is. Zo hoeft de DataRepository niet alle replicators in te laden als deze wordt opgestart, en bespaart geheugen.
Verder kunnen veranderingen aan een replicator makkelijk gedaan worden zonder code in het hoofdprogramma aan te hoeven passen. Ook is het makkelijk nieuwe replicators te maken zolang deze de IReplicator interface implementeren. Ook hierbij zou de hoofdapplicatie niet aangepast hoeven worden bij iedere nieuwe replicator, dit garandeert de onderhoudbaarheid van de applicatie. Iedere replicator zou ook hergebruikt kunnen worden los van het DataRepository project als hier behoefte aan is.
Verder kan er zo gekozen worden specifieke replicators weg te laten uit bijvoorbeeld een light versie van het ToastAR systeem.

Iedere replicator implementeert de interface IReplicator. Om te zorgen dat deze interface op één plaats gedefinieerd is en alle replicators deze moeten implementeren is de interface in een eigen DLL gezet genaamd BaseReplicationAssembly. Alle replicator DLLs hebben een referentie naar deze assembly en hebben de BaseReplicationAssembly dus nodig om te functioneren, zij hebben anders geen definitie voor de interface die zij implementeren.

Hieronder is de structuur weergegeven in een deel van het klassendiagram met de klassen verantwoordelijk voor het repliceren van data.



Figuur 26. Deel van het initiële klassendiagram van de klassen verantwoordelijk voor het repliceren van data uit een CSV bestand.

De ReplicationLauncher is verantwoordelijk voor het aanroepen van het juiste type replicatie module, de terug geleverde data om laten zetten naar ContactData via de ReplicatedDataConverter klasse en deze geconverteerde data via de DatabaseManager op te slaan in de lokale database. De ReplicationLauncher wordt aangeroepen door ReplicationScheduler als de timer daarin verloopt en een replicatie gestart moet worden. De CSVReplicator implementeert de IReplicator interface en maakt gebruik van instellingen in de klasse CSVReplicatorSetting om te kunnen functioneren.

Het gebruik van de IReplicator interface en de verschillende type replicators is een toegepaste vorm van de Strategy Pattern. De ReplicationLauncher bepaalt de strategie aan de hand van het type in de ReplicatorSetting.

Het bepalen van het type replicator, het doorgeven van instellingen specifiek voor het type replicator en het omzetten van de data naar ContactData is mogelijk gemaakt door toevoegingen aan de ReplicatorSetting klasse.

Hieronder zijn de toevoegingen te zien.

```
+Type : String
+DataMapping : Dictionary<String, int>
+SerializedReplicatorSpecificSettings :
String
```

Figuur 27. Toegevoegde attributen ReplicatorSetting.

Aan de hand van het type in een ReplicatorSetting kan ReplicationLauncher bepalen welke replicator opgestart moet worden. De DataMapping wordt gebruikt zodat ReplicatedDataConverter kan zien welke attributen uit de databron naar welk attribuut in ContactData overgezet moet worden.

Als een replicator DLL wordt geïnstantieerd, worden de replicator specifieke instellingen meegegeven. Iedere replicator bezit een klasse zoals ReplicatorSettings met ieder een verzameling van instellingen die de replicator nodig heeft om te kunnen functioneren.
Zo heeft bijvoorbeeld de CSV replicator een klasse genaamd CSVReplicatorSettings die een string met een FilePath bevat. ReplicatorSettings heeft nu een string genaamd SerializedReplicatorSpecificSettings waarin deze specifieke setting klassen meegeleverd worden. De specifieke setting klasse wordt geserialiseerd opgeslagen in deze string. Als een replicator wordt aangeroepen, dan wordt deze string meegegeven aan de constructor of als parameter voor de initialize methode. De string wordt dan gedeserialiseerd en omgezet naar de specifieke setting klasse van de replicator.
Voor het serialiseren en deserialiseren is gekozen voor BinaryFormatter, die dit omzet naar en van binair formaat. Dit is een snelle en eenvoudige manier van serialiseren en dit formaat kan ook complexe objecten aan.

De methode IsDoneReplicating in ReplicationLauncher is een methode die door ReplicationScheduler aangeroepen kan worden om te achterhalen of de replicatie voorbij is, om dan een nieuwe replicatie op te starten.
De methode DetermineCorrectReplicatorAndLaunchIt bepaalt in de initiële versie van deze methode aan de hand het type uit ReplicatorSetting en een switch welke replicator opgestart moet worden. Als het replicator type bepaald is wordt de methode LaunchReplicatorAndStoreResults aangeroepen. Hierin wordt eerst de methode LoadDLLAtRuntimeAndReplicate aangeroepen, die de replicator DLL aanroept en de replicatie laat uitvoeren aan de hand van de meegeleverde settings. Daarna wordt in deze methode ConvertDataAccordingToSettings aangeroepen van de ReplicatedDataConverter klasse waarin de gerepliceerde data wordt meegegeven die de LoadDLLAtRuntimeAndReplicate heeft terug geleverd aan de Replicationlauncher. Nadat de gerepliceerde data is omgezet naar ContactData in de ReplicatedDataConverter klasse wordt de InsertContact methode aangeroepen in de DatabaseManager klasse. Hierna wordt in de IsDoneReplicating methode aangegeven dat de replicatie voltooid is. Bij exceptions binnen de replicator of launcher klasse wordt ook aangegeven in de IsDoneReplicating dat de replicatie voltooid is om de

scheduler klasse niet oneindig te laten wachten alvorens de volgende replicatie in de queue of de volgende ingeplande replicatie uit te laten voeren.

Bij de CSV replicator is gebruik gemaakt van een reeds bestaande CSV reader. Dit heb ik gedaan om op dit vlak het wiel niet opnieuw uit te hoeven vinden en tijd te besparen voor de andere replicators. Om niet afhankelijk te zijn van een library met onnodige functionaliteiten is er een reader gebruikt die werkt binnen één klasse. De reader was onderdeel van een CSV reader en writer, waarbij de writer niet nodig was in de replicator. De reader functionaliteiten en de interface die de reader en writer gebruikten zijn samengevoegd tot één reader klasse die de CSV replicator nu gebruikt om data uit CSV bronnen te halen.

Bij het maken van unit tests voor de ReplicationLauncher, CSVReplicator en de ReplicationDataConverter heb ik de volgende logische testgevallen opgesteld.

ReplicationLauncher

Code testgeval	Beschrijving testgeval
LT20	De CSV replicator DLL moet aangeroepen kunnen worden.
LT21	De CSV replicator moet correcte data leveren aan de ReplicationLauncher na het repliceren van de data uit een CSV bestand.
LT22	De gerepliceerde data moet correct zijn opgeslagen in de database.

ReplicationDataConverter

Code testgeval	Beschrijving testgeval
LT24	De geleverde data moet correct worden omgezet naar een collectie vol met ContactData.

Ik heb via een systeemtest de replicatie functionaliteit getest op de volgende manier:

"In sprint 3 heb ik een systeemtest uitgevoerd om de replicatie functionaliteiten te testen. Ik heb dit getest door via de main methode in de console applicatie het replicatieproces aan te roepen en via een SQLite browser programma het database bestand benaderd om te zien of de gerepliceerde data daadwerkelijk werd opgeslagen en op een correcte manier."

4.4.3 Sprint 3 Review

De replicatie functionaliteit is af gekregen met bijbehorende unit tests en een systeemtest. Er kan begonnen worden aan het volgende type replicator in de volgende sprint.

4.5 Sprint 4: ODBC replicator

Sprint 4 is toegewijd aan het maken van een repliceer module voor databronnen die benaderd kunnen worden via ODBC.

4.5.1 Backlog

De replicatiemogelijkheid wordt uitgebreid met de ODBCReplicator klasse.

Figuur 28. Sprint 4 backlog ODBCReplicator.

De ODBCReplicator moet de geleverde instellingen deserialiseren en de data kunnen repliceren.

4.5.2 Uitvoering

Tijdens de wekelijkse meeting met mijn bedrijfsmentor ben ik gewezen op de nadelen van het gebruik van een switch in de DetermineCorrectReplicatorAndLaunchIt methode in de ReplicationLauncher klasse. Bij iedere nieuwe replicator zou deze switch uitgebreid moeten worden met een nieuw type, dit zou de onderhoudbaarheid en uitbreidbaarheid van de code nadelig beïnvloeden.
Daarnaast volg ik op deze manier niet het Open Closed Principle, sinds de code in deze methode aangepast zou moeten worden bij iedere uitbreiding van nieuwe replicator types.

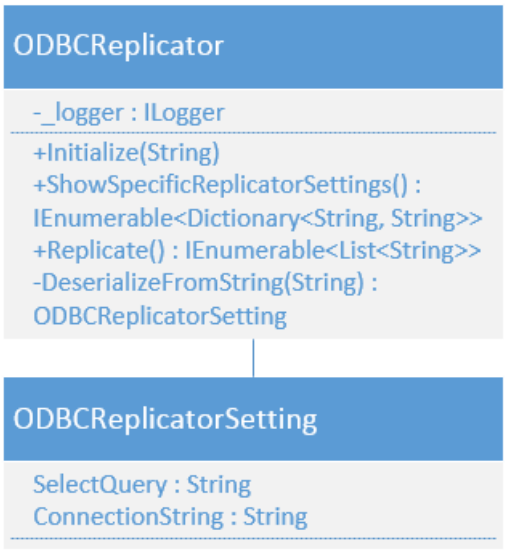
Om dit te verbeteren ben ik deze sprint begonnen met het refactoren van de ReplicationLauncher klasse.
In de Replicationlauncher is de wijze van het lanceren van replicators aangepast. De switch met types is verwijderd, en er wordt nu gekeken naar de beschikbare DLLs. Er wordt per DLL gekeken of dit het juiste type replicator is.

Bij ieder Replicator DLL project wordt een 'Post-build event command' geschreven die het DLL bestand van die replicator kopieert naar een map waar alle replicator DLLs in verzameld worden, deze map is Replicators genoemd. Dit commando wordt uitgevoerd nadat een replicator DLL project het DLL bestand aanmaakt.
Door alle replicator DLLs bij elkaar in een map te hebben kan makkelijker in de ReplicationLauncher klasse gezocht worden naar DLLs en over deze lijst geïtereerd worden, om zo het juiste type replicator te vinden en te instantiëren. De replicators worden zo dynamisch geladen i.p.v. in een switch te bepalen welk type geladen moet worden, om zo de code in de DetermineCorrectReplicatorAndLaunchIt methode niet uit te hoeven breiden bij iedere nieuwe replicator.
Er wordt aan ieder AssemblyInfo bestand van de replicator projecten een zogeheten Custom Attribute toegevoegd waarin het type van de replicator te zien is. Zo kan de ReplicationLauncher het juiste DLL bestand kiezen zonder eerst alle replicator assemblies in het geheugen te laden. De replicator assemblies worden geladen via de Assembly.ReflectOnlyLoad methode en er wordt over deze lijst geïtereerd om de juiste replicator aan te roepen. Als deze gevonden is wordt die replicator assembly daadwerkelijk geladen en geïnstantieerd.
De ReflectOnlyLoad methode zorgt ervoor dat het attribuut van de assemblies met

het type erin gelezen kan worden zonder eerst de assembly te instantiëren en in het geheugen te laden.

De methodes GetReplicatorFolderPath en CurrentDomainReflectionOnlyAssemblyResolver zijn toegevoegd aan de ReplicationLauncher klasse. De GetReplicatorFolderPath methode achterhaalt het pad naar de Replicator map zodat de DetermineCorrectReplicatorAndLaunchIt methode binnen deze map kan zoeken naar het juiste type DLL. De resolver methode is gemaakt om exceptions voor het missen van een assembly tijdens het laden van een replicator assembly met ReflectionOnlyLoad op te vangen en de missende assemblies ook te laden. ReflectionOnlyLoad laadt namelijk niet automatisch assemblies waar de te laden assembly van afhankelijk is, bijvoorbeeld de assembly met de IReplicator interface.

Om data te repliceren uit een bron via een ODBC koppeling zijn de volgende klassen ontworpen en toegevoegd aan het klassendiagram.



Figuur 29. Toevoeging ODBCReplicator aan klassediagram.

De ODBCReplicator klasse implementeert de IReplicator interface. Als instellingen gebruikt deze replicator een ConnectionString om een koppeling te kunnen maken met een specifieke database via ODBC. Daarnaast wordt een query meegegeven waarmee de data uit de desbetreffende database wordt opgevraagd. Om een connectie te maken via ODBC worden klassen gebruikt van System.Data.Odbc binnen het .NET Framework.

Voor het maken van de unit test voor de ODBCReplicator heb ik het volgende logische testgeval opgesteld

ODBCReplicator

Code testgeval	Beschrijving testgeval
LT25	De ODBC replicator moet correcte data leveren aan de ReplicationLauncher na het repliceren van de data uit een databron via een ODBC koppeling.

Na deze unit test heb ik ook een systeemtest uitgevoerd.

"In sprint 4 heb ik een systeemtest uitgevoerd om de replicatie functionaliteiten te testen van de ODBC replicator. Ik heb dit getest door via de main methode in de console applicatie het replicatieproces van het type ODBC aan te roepen en via een SQLite browser programma het database bestand benaderd om te zien of de gerepliceerde data daadwerkelijk werd opgeslagen en op een correcte manier."

Er is aan het eind van de sprint een begin gemaakt aan een ExcelReplicator. Hier heb ik het volgende ontwerp voor gemaakt.

Figuur 30. Ontwerp ExcelReplicator.

Voor de ExcelReplicator is er de optie om data uit Excel bestanden te lezen via OleDb. Daarnaast kan Microsoft.Office.Interop.Excel gebruikt worden, maar dat is niet beschikbaar zonder office te hebben geïnstalleerd. Er kan niet worden uitgegaan van een installatie van Microsoft Office op een server.
ik heb daarom gekozen om Excel bestanden uit te lezen met behulp van OleDb.

Uiteindelijk heb ik het ophalen van data via OleDb nog niet volledig afgekregen.

4.5.3 Sprint 4 Review

Het refactoren van de ReplicationLauncher klasse heeft een groot deel van de sprint gekost, maar ik heb de ODBCReplicator af kunnen krijgen. Verder heb ik al een groot deel gemaakt van de ExcelReplicator.

4.6 Sprint 5: HoastAR integratie

In deze sprint lag de focus op het integreren van de Data Repository, die op dat moment nog binnen de console applicatie stond, in HoastAR als module hiervan.

Tijdens de wekelijkse meeting was naar voren gekomen dat een replicator voor het CRM systeem TOPdesk een hogere prioriteit heeft dan een ExcelReplicator. Verder was de vraag wanneer ik de functionaliteiten die ik tot nu toe had zou gaan integreren in HoastAR. Ik heb besloten eerst te beginnen met het integreren van de Data Repository in HoastAR en in een latere sprint verder te gaan aan replicatie functionaliteiten.

4.6.1 Backlog

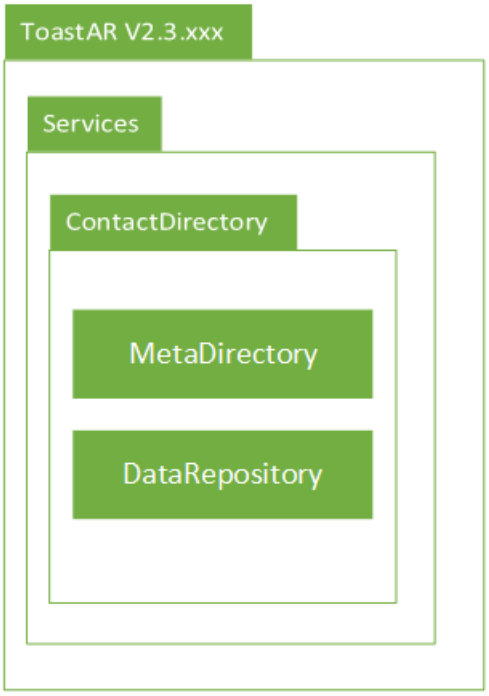
De HoastAR integratie heeft het volgende backlog product.

Figuur 31. Sprint 5 backlog HoastAR integratie.

De klassen moeten overgezet worden naar de juiste locatie, de module moet aangeroepen en opgestart kunnen worden en het logging systeem moet geïmplementeerd worden.

4.6.2 Uitvoering

Om de Data Repository te integreren heb ik de code van HoastAR ontvangen en hebben ik en mijn bedrijfsmentor de code doorlopen en heeft hij de structuur uitgelegd van HoastAR. Op deze manier wist ik hoe modules worden aangeroepen binnen HoastAR en waar de Data Repository module moest komen te staan binnen HoastAR. Hieronder is de locatie van de Data Repository module binnen de gehele ToastAR solution.



Figuur 32. Module locatie Data Repository binnen de HoastAR modulestructuur.

De Data Repository staat nu samen met de MetaDirectory module in de ContactDirectory module. De volledige namespace van deze module is Ar.ToastAr.Services.ContactDirectory.
Daarnaast wordt nu de ContactData klasse gebruikt die staat in de Ar.ToastAr.Model module. De ReplicatorSetting en replicator specifieke setting klassen staan nu ook in Ar.ToastAr.Model.
Het gebruik van de naam ToastAR is misschien verwarrend, maar in deze solution staan de ToastAR client, de HoastAR server en de administratie tool allemaal in aparte modules.

Tijdens de integratie kwam ik er achter dat de methode DetermineCorrectReplicatorAndLaunchIt in ReplicationLauncher nog geen unit test had om te testen of deze functionaliteit in de klasse wel werkt. De methode is in sprint 3 al beschreven en getest door middel van de systeemtest, maar ik heb na het refactoren van deze methode in sprint 4 in de systeemtest alleen de focus gelegd op de werking van het repliceren via een ODBC koppeling.

Het volgende logische testgeval is opgesteld voor de unit test voor de ReplicationScheduler klasse, waarmee deze methode ook getest wordt.

Code testgeval	Beschrijving testgeval
LT19	Als het tijd is om te repliceren moet een replicatie aangeroepen kunnen worden.

Na het maken van de unit test kwam ik er achter dat de methode niet functioneerde. Ik had in de methode een check staan of alle gevonden DLLs in de Replicator map wel de IReplicator interface implementeren. Daarna worden de DLLs ingeladen met ReflectionOnlyLoad en wordt de juiste DLL geselecteerd aan de hand van het type, zoals beschreven in het vorige hoofdstuk.
Ik ben veel tijd verloren met het zoeken naar een werkende oplossing om te proberen te achterhalen of een DLL een klasse heeft die IReplicator implementeert. Het idee van deze check was dat DLLs die eigenlijk geen replicator zijn niet ingeladen hoefden te worden via ReflectionOnlyLoad. Ik heb uiteindelijk besloten deze check ertussenuit te halen en direct de gevonden DLLs in de Replicator map in te laden via ReflectionOnlyLoad en al deze DLLs te checken voor een bepaald type. De kans is klein dat naast de BaseReplicationAssembly DLL met daarin de IReplicator interface andere DLLs in de Replicator map staan die geen replicators zijn. Verder duurde een oplossing vinden te lang en moest ik verder aan het integreren van de Data Repository in HoastAR.

Tijdens deze sprint is er nog een requirement bij gekomen, FR8.

Code requirement	Beschrijving requirement
FR8	De HoastAR moet kunnen achterhalen wat de specifieke gegevens zijn die ieder type replicatie module nodig heeft om te kunnen functioneren.

FR8 geeft aan dat de HoastAR moet weten wat de replicators nodig hebben aan specifieke instellingen om deze te kunnen leveren aan de replicators.

De methode GetAllReplicatorSpecificSettings in Replicationlauncher en ShowSpecificReplicatorSettings in de IReplicator interface zijn initieel gemaakt om aan deze requirement te voldoen. Bij de integratie in HoastAR is de ContactData klasse in HoastAR.Model gebruikt in plaats van mijn kopie van deze klasse. Omdat in de Model module van HoastAR klassen staan die vaak worden hergebruikt zoals setting klassen heb ik ReplicatorSettings en alle replicator specifieke setting klassen ook in de HoastAR.Model module verwerkt.
Op deze manier hebben andere modules binnen HoastAR ook toegang tot de setting klassen en voldoet het zo aan FR8. Om de replicator DLLs weer onafhankelijk te maken van HoastAR zou ik adviseren deze methodes te gebruiken en de replicator specifieke setting klassen weer terug in de DLL projecten te zetten.

Om de logger van HoastAR te gebruiken om exceptions naar toe te schrijven is de interface ILogging uit de Ar.ToastAr.Core.Logging module gebruikt.

Ninject zal bij het aanroepen van klassen kijken naar de constructor ervan en de benodigde klasse leveren. Om een logger te gebruiken heb ik in de klassen van de Data Repository die een logger nodig hebben ILogging in de constructor als parameter gezet.

Verder heb ik de klasse DataRepositoryInitializer aangemaakt. Als deze wordt aangeroepen door Ninject bij het opstarten van HoastAR, zal de klasse ReplicationScheduler aangeroepen worden. Op deze manier kan begonnen worden aan het inplannen van de replicaties.

De gemaakte unit tests heb ik aan de Ar.ToastAr.Test module toegevoegd.

Aan het eind van deze sprint was ik nog niet klaar met het integreren, het leveren van de settings vanuit HoastAR is nog niet compleet en ik heb dus nog niet kunnen testen of het repliceren werkt. Wel heb ik kunnen testen of het leveren van ContactData via HoastAR aan de ToastAR client werkt en dit is inderdaad het geval.

4.6.3 Sprint 5 Review

Ik ben veel tijd verloren aan het refactoren van de ReplicationLauncher en heb de Data Repository nog niet volledig kunnen integreren in HoastAR. De functionaliteit om data te leveren aan HoastAR, die op zijn beurt de data levert aan ToastAR werkt op dit moment wel.

4.7 Sprint 6: Documentatie

Tijdens mijn tussentijds assessment heb ik als advies gekregen mij geheel te focussen op de documentatie om dit te verbeteren en op tijd af te krijgen. Ik heb hierdoor geen tijd meer gehad om de integratie van de Data Repository af te krijgen.

4.7.1 Backlog

De documentatie bestaat uit het afstudeerverslag, afstudeerverslag bijlage document, testrapportage en alle overige documenten.

Figuur 33. Sprint 6 backlog documentatie.

De hoofdpunten zijn het herstructureren en het theoretisch inkaderen in het verslag.

4.7.2 Uitvoering

Ten eerste had ik meteen een week koorts te pakken na het tussentijds assessment, dit heeft mij een behoorlijke vertraging opgeleverd.

De belangrijkste feedback die ik heb ontvangen was het beter structureren van het afstudeerverslag, dit heb ik geïmplementeerd door per sprint een toelichting te geven van de werkzaamheden met onderbouwing.

Verder had ik als advies alles theoretisch in te kaderen, dit heb ik gedaan door het Plan van Aanpak uit te breiden en in het hoofdstuk projectmethodiek de theorie achter de gebruikte ontwikkelmethodiek, het klassendiagram, OOD, Principles en Design Patterns toe te lichten.
Ik heb dit in sprint 0 beschreven en in de sprints erna heb ik bij de gemaakte ontwerpkeuzes gerefereerd naar de requirements en deze achterliggende theorie.

Verder heb ik mijn testdocumentatie uitgebreid met een detailtestplan van de gemaakte unit tests en voor de systeemintegratietest, daarnaast heb ik ook de uitgevoerde systeem tests beschreven in een document.[40,41]

Voor het testen na de integratie is een Systeemintegratietest ontworpen.

"Het doel van een systeem integratie test, ook wel ketentest genoemd, is het testen van het functioneren van een systeem in combinatie met andere systemen. Bij een ketentest wordt de samenwerking van het systeem met de aanliggende systemen getest. In de ketentest worden deze systemen vaak voor het eerst aan elkaar gekoppeld. De ketentest richt zich op het vinden van de fouten die ontstaan als systemen niet goed samenwerken.

Om zeker te zijn dat de DataRepository goed geïntegreerd wordt in de HoastAR en data via HoastAR aan de ToastAR geleverd kan worden wordt er een systeem integratie test uitgevoerd."

Hierbij zijn de volgende logische testgevallen opgesteld.

Code testgeval	Beschrijving testgeval
LT01	De ToastAR client moet kunnen zoeken naar contacten, de contact data moet vanuit de lokale database in de Data Repository geleverd worden aan de hand van de zoektermen.
LT02	Er moet een replicatie ingepland kunnen worden via HoastAR en uitgevoerd worden.
LT03	Er moet via HoastAR de ingeplande replicaties opgevraagd kunnen worden.
LT04	Replicaties moeten gedaan kunnen worden voor CSV bestanden.
LT05	Replicaties moeten gedaan kunnen worden voor data bronnen bereikbaar via een ODBC koppeling.

Om de logische testgevallen uit te voeren heb ik deze fysieke testgevallen bedacht.

Code testgeval	Beschrijving testgeval
FT01	De lokale database van de Data Repository zal eerst gevuld worden met ContactData. Hierna moeten de ToastAR en HoastAR opgestart worden. Ten slotte moet er in de ToastAR client gezocht worden naar ContactData die is toegevoegd aan de lokale database.
FT02	Er bestaat nog geen user interface voor dus zal dit in HoastAR via code aangeroepen moeten worden via een interface van de Data Repository. Nadat de replicatie voltooid zou moeten zijn moet er in de lokale database gekeken worden of de data daadwerkelijk gerepliceerd en opgeslagen is.
FT03	Dit zal ook via code in de HoastAR aangeroepen moeten worden sinds hier ook geen user interface voor bestaat op dit moment.
FT04	Er moet via HoastAR eerst een replicatie ingepland worden van het type CSV. Nadat de replicatie voltooid zou moeten zijn moet er in de lokale database gekeken worden of de data daadwerkelijk gerepliceerd en opgeslagen is.
FT05	Er moet via HoastAR eerst een replicatie ingepland worden van het type ODBC. Nadat de replicatie voltooid zou moeten zijn moet er in de lokale database gekeken worden of de data daadwerkelijk gerepliceerd en opgeslagen is.

Het ontwerp van de applicatie heb ik per sprint toegelicht met delen van het klassendiagram ter verduidelijking.

4.7.3 Sprint 6 Review

Ik heb afgesproken de integratie van de Data Repository in HoastAR af te maken na het inleveren van mijn verslag. Ik zal hierna de systeemintegratietest uitvoeren waar ik een detailtestplan voor heb gemaakt.

5 Afwijkingen afstudeerplan

// Bespreking van afwijkingen ten opzichte van het afstudeerplan met motivatie.

Sprint 0 is later begonnen dan aangegeven in het afstudeerplan. De vertraging is veroorzaakt door de latere goedkeuring van het afstudeerplan, waardoor de stageplek ook niet meteen gereed was gemaakt bij het bedrijf. Er is uiteindelijk begonnen aan de stage op woensdag 7 september.

Verder is de sprint voor de ODBC replicator eerder gedaan dan de Excel replicator, om zo een complexe replicator te hebben gemaakt voordat ik daar geen tijd meer voor zou hebben in eventuele tijdsnood. Al zou ik alleen minder complexe replicators gemaakt hebben dan zou dit het oordeel voor de afstudeer opdracht negatief beïnvloeden.

Na het maken van de ODBC Replicator is er gekozen om het project te integreren in de HoastAR. Dit stond niet in het afstudeerplan omdat toen nog niet besloten was

om eerst de functionaliteiten te maken in een aparte console applicatie. Er is een integratietest gemaakt om zeker te zijn dat de gemaakte functionaliteiten goed werken na de integratie in HoastAR.

Verder was in het afstudeerplan geen rekening gehouden met het mogelijk aanpassen van het afstudeerplan na het tussentijds assessment. Ik heb hier de laatste paar weken veel tijd aan besteed.

6 Opgeleverde producten

// Opsomming van de opgeleverde (tussen)producten met toelichting.

Plan van Aanpak (PvA) – In het PvA is o.a. te zien wat de planning was voor het project, de opdracht omschrijving, projectmethodiek, projectorganisatie en mijlpaalproducten.

Requirements – Het Requirements document bevat de functionele en niet-functionele requirements voor de DataRepository.

Klassediagram – Het Klassediagram is een UML Class Diagram waarin de klassenstructuur van de Data Repository duidelijk wordt.

Relational Representation Model (RRM) – In het RRM is van de lokale data opslag te zien hoe de tabelstructuur eruit ziet.

Testplan – Het Mastertestplan bevat uitleg over de geplande tests voor het project.

Testdocumentatie – De detail testplan documenten bevat logische en fysieke testgevallen van een specifieke testsoort. Verder zijn de resultaten van de uitgevoerde systeem tests beschreven in het Systeemtest document.

Code – De daadwerkelijke code van het software project.

Afstudeerverslag – Dit verslag ter beoordeling van de afstudeerstage.

7 Evaluatie

7.1 Evaluatie gebruikte aanpak

// Evaluatie van de gebruikte aanpak tijdens de afstudeerperiode.

De keuze om de functionaliteiten eerst te maken in een console applicatie heeft voor en nadelen gehad. Het voordeel was dat er geheel gefocust kon worden op de functionaliteiten van de DataRepository zonder afgeleid te worden door HoastAR specifieke zaken. Het nadeel is dat de integratie in HoastAR tijd in beslag nam en getest moest worden of alle functionaliteiten nog functioneerden.

Als ik het project opnieuw zou doen, dan zou ik weer besluiten om de basis functionaliteiten eerst te maken in een console applicatie. Ik zou dan wel eerder deze functionaliteiten integreren, dit is niet op tijd afgekregen omdat hier te laat aan

begonnen was in dit geval. Het beste moment zou na het maken van de CSV replicator geweest zijn sinds toen de replicatie functionaliteit was gemaakt en getest. Alle basis functionaliteiten zaten toen in de Data Repository.

De ruime planning voor de project opstart in sprint 0 is een goede keuze geweest. Doordat ik later moest beginnen heb ik daar tijd mee verloren, zonder meteen de gehele planning aan te moeten passen.

Verder ben ik blij dat ik gekozen had om een agile ontwikkelmethodiek te gebruiken, zo kon ik requirements die later in het project erbij kwamen makkelijk integreren in de applicatie. Het gebruik van Scrum heeft me de optie gegeven om per sprint te bepalen wat de hoogste prioriteiten waren om te ontwikkelen voor het systeem en dit in de sprint backlog te plaatsen. Achteraf gezien heb ik beter gebruik moeten maken van deze optie en deze prioriteiten beter in kaart moeten brengen tijdens de wekelijkse meetings die ik had met mijn bedrijfsmentor. Hieronder heb ik beschreven wat hier de gevolgen van waren.

Door een communicatie fout van mijn kant heb ik tijd besteed aan het maken van een ExcelReplicator zonder te informeren of er geen ander type replicator was die een hogere prioriteit had voor het bedrijf. Het bleek dat een replicator die data uit het Top Desk CRM systeem repliceert een hogere prioriteit had.
Achteraf gezien zou ik toch geen tijd hebben gehad om deze replicator af te krijgen, sinds ik de week erna tijdens het tussentijds assessment te horen kreeg geheel te focussen op de documentatie, maar ik heb toch twee dagen aan het maken van de ExcelReplicator besteed en dit is zonde.

Ik had een systeemtest beter uit moeten voeren na het refactoren van de ReplicationLauncher klasse in sprint 4. In deze systeemtest had ik niet het complete systeem getest maar alleen gefocust op de werking van de ODBCReplicator, het repliceren en het juist opslaan van de gerepliceerde data. Omdat ik dit niet had gedaan heeft dit meer vertraging opgeleverd verderop in het project. Bij het integreren van de DataRepository in HoastAR tijdens sprint 5 kwam ik er achter dat er geen unit test was geschreven voor het bepalen van de juiste replicator DLL aan de hand van het type uit de ReplicatorSetting. Na het maken van de unit test kwam ik er achter dat dit nog niet goed functioneerde, ik heb hierdoor kostbare tijd verloren en heb de integratie daardoor niet af kunnen krijgen in sprint 5. De tijd die ik heb besteed aan het maken van de ExcelReplicator was dan waarschijnlijk voor een deel besteed aan het uitvoeren van deze systeemtest.

Als ik voortaan systeemtesten uitvoer ga ik alle functionaliteiten testen van een systeem in plaats van de focus leggen op een deel van het systeem dat net gemaakt is, om zo fouten zoals hierboven beschreven is te voorkomen.

Ten slotte misschien wel het pijnlijkste, ik heb het maken van dit verslag enorm onderschat. Ik had van tevoren niet verwacht zoveel tijd te besteden met het maken en verbeteren van alle documentatie. Mijn focus lag meer op het maken van de applicatie voor het bedrijf.
Ik had ingepland iedere vrijdag te werken aan mijn documentatie, maar ik had van

tevoren ook een sprint moeten toewijden in de planning aan het eind van het project. Uiteindelijk heb ik hier ook aan gewerkt in de laatste sprint, maar dit was ten koste gegaan van een goede afronding van het programma ontwikkelen. Het integreren van de Data Repository is niet gelukt voor het inleveren van dit verslag, ik heb afgesproken dit wel af te maken na het inleveren van dit verslag.

7.2 Evaluatie opgeleverde producten

// Evaluatie van de opgeleverde (tussen)producten.

Ik ben over het algemeen tevreden over de gemaakte software. Vóór de integratie voldeed het aan alle requirements, het kon data repliceren uit CSV bestanden en data bronnen bereikbaar via een ODBC koppeling. Ik keek al uit naar het maken van een replicator voor TOPdesk, maar het tussentijds assessment gooide roet in het eten. Als ik niet beoordeeld zou worden op mijn verslag maar op mijn gemaakte software dan zou ik tijd genoeg hebben gehad voor het integreren van de Data Repository in HoastAR en waarschijnlijk ook voor de TOPdesk replicator.

Sinds ik halverwege de integratie de focus moest leggen op mijn documentatie geeft mijn geleverde resultaat een bitter gevoel. Ik ben stiekem een perfectionist en voor mijn gevoel ben ik pas klaar na een voltooide en geteste integratie. Dit is ook één van de redenen waarom ik na dit verslag de integratie zal afmaken.

Bij het maken van de beschrijving van Sprint 1 in dit verslag ben ik erachter gekomen dat ik de InsertContact methode in de DatabaseManager klasse zou willen herschrijven als ik hier nog tijd voor had. In plaats van deze methode meerdere keren aan te laten roepen door een klasse die gerepliceerde data wilt opslaan, zou ik een collectie met alle ContactData uit die bron als parameter leveren om deze methode zelf verantwoordelijk te maken voor het stuk voor stuk invoeren van deze data in de database. Hiermee heeft deze methode geen Boolean parameters meer nodig en worden potentiële problemen voorkomen als niet goed wordt aangegeven of een in te voeren ContactData de eerste of de laatste in de collectie is of geen van beide. Dit is namelijk de functie van deze Booleans.
Ook is er na het herschrijven ook geen sprake meer van zogenoemde ‘control coupling’.[42] De volgorde in de uitvoer ofwel de ‘flow’ van de InsertContact methode wordt op dit moment beïnvloedt door middel van de Boolean parameters. Verder wordt niet het Single Responsibility Principle gevolgd in deze situatie, de DatabaseManager zou verantwoordelijk moeten zijn voor de manier van opslaan en niet de klasse die InsertContact aanroept.

Daarnaast zou ik in de methode QueryContactData beter de parameters willen checken om aan de Crash Early Principle te willen voldoen. Op dit moment worden de parameters niet eerst getest op validiteit voordat zij gebruikt worden. De Where clausule verdient ook extra checks tegen SQL injection die er op dit moment niet zijn.

Het configureren van een ODBC Data Source moet nu nog handmatig gedaan worden via de ODBC Data Source Administrator van Windows. Ik adviseer een uitbreiding op de Data Repository om dit programmatisch te doen in een nieuwere versie, om zo een koppeling van een databron via ODBC in een configuratie wizard te

kunnen maken. Zo kan de installatie en configuratie van ToastAR met bijbehorende HoastAR in de toekomst via zulke wizards. De SQLConfigDataSource Function kan hier mogelijk mee helpen.[43]

Ik zou ook nog een interface maken voor het aanleveren van ingeplande replicaties zoals beschreven in FR6. Op dit moment is de HoastAR afhankelijk van een implementatie in plaats van een abstractie, dit is in strijd met het Dependency Inversion Principle.

Het verslag zelf geeft hopelijk een goede indicatie van de complexiteit van de gemaakte software en ik heb mijn best gedaan de gemaakte keuzes goed te onderbouwen met referenties naar de requirements en de theorie die ik heb besproken in het PvA.

Verder heb ik in mijn perfectionistische bui de theorie waarschijnlijk te uitgebreid beschreven waardoor Sprint 0 een zeer groot hoofdstuk is geworden.

7.3 Evaluatie beroepstaken

// Evaluatie van de geselecteerde beroepstaken.

In het afstudeerplan had ik vier beroepstaken geselecteerd: de beroepstaken 1.4, 3.2, 3.3 en 3.5. Na de uitvoer van het project heb ik de volgende beroepstaken hieraan toegevoegd: 2.1, 2.2 en 3.4.

1.4 Uitvoeren analyse door definitie van requirements

Ik heb requirements gedefinieerd in het Requirements document. Deze heb ik opgesteld aan de hand van gesprekken met mijn bedrijfsmentor en feedback in onze wekelijkse voortgangsbesprekingen.
Het ontwikkelen van de applicatie was een iteratief proces en er zijn in latere iteraties requirements bij gekomen. Deze zijn toen ook in het Requirements document verwerkt. In dit verslag is naar bepaalde requirements gerefereerd bij de beschrijving van ontwerp keuzes die erop gebaseerd zijn. Zo is traceerbaar dat de requirements invloed hebben gehad op het ontwikkelen van de applicatie.

2.1 Opstellen gegevensmodel voor database

Bij het ontwikkelen van de DatabaseManager heb ik een opslag structuur ontworpen voor het lokaal opslaan van gerepliceerde data. Er is hierbij rekening gehouden met de requirements en de omgeving waarin de data opgeslagen wordt. In hoofdstuk 4.2.2 heb ik de opslag structuur beschreven.

2.2 Ontwerpen, bouwen en bevragen van een database

Voor het ontwerp van de lokale database heb ik een RRM gemaakt die ook in hoofdstuk 4.2.2 staat beschreven. De DatabaseManager klasse maakt een database bestand aan in de constructor, verder heb ik methodes gemaakt in deze klasse die de database bevragen en gegevens leveren aan andere klassen binnen de Data Repository. Daarnaast heb ik methodes gemaakt die tabellen aanmaken, updaten of verwijderen.

3.2 Ontwerpen systeemdeel

Ik heb de Data Repository module ontworpen voor de Windows service HoastAR. Ik heb per sprint in dit verslag het ontwerp toegelicht aan de hand van delen van het gemaakte UML klassediagram.

3.3 Bouwen applicatie

Na het ontwerpen van de Data Repository heb ik het ontwerp geïmplementeerd door middel van het bouwen van de applicatie. Ik heb delen van dit proces beschreven in de verschillende sprint hoofdstukken in dit verslag.

3.4 Initiëren en plannen van het testproces.

Ik heb tijdens sprint 0 een Mastertestplan document gemaakt die ik in sprint 1 verder heb uitgebreid. Hierin beschrijf ik mijn geplande test methoden voor het project. Verder heb ik een detailtestplan gemaakt voor de moduletests en de systeem integratie test.

3.5 Uitvoeren van en rapporteren over het testproces

Ik heb in iedere sprint na het maken van een systeemdeel unit tests geschreven en uitgevoerd. Dit heb ik voor een deel beschreven in het detailtestplan moduletest en vooral in dit verslag.
De uitgevoerde systeemtests staan beschreven in het Systeemtest document. De uitvoer van de systeem integratie test zal gedaan worden na het schrijven van dit verslag en is daarom nog niet over gerapporteerd.

8 Literatuurlijst

// Hieronder zijn de gebruikte bronnen weergegeven.

1. Ask Roger! – Specialist in Microsoft Skype for Business en Cisco Communicatie oplossingen. *Ask Roger!* http://www.askroger.nl/. Bekeken op 10-12-2016.
2. Telephony Application Programming Interface. *Wikipedia*. https://en.wikipedia.org/wiki/Telephony_Application_Programming_Interface. Voor het laatst aangepast op 1-12-2016. Bekeken op 12-12-2016.
3. Representational state transfer. *Wikipedia*. https://en.wikipedia.org/wiki/Representational_state_transfer. Voor het laatst aangepast op 12-12-2016. Bekeken op 12-12-2016.
4. SignalR. *ASP.NET*. https://www.asp.net/signalr. Bekeken op 12-12-2016.
5. Directoryservice. *Wikipedia*. https://nl.wikipedia.org/wiki/Directoryservice. Voor het laatst aangepast op 11-12-2013. Bekeken op 12-12-2016.

6. Directory service. *Wikipedia*. https://en.wikipedia.org/wiki/Directory_service. Voor het laatst aangepast op 26-11-2016. Bekeken op 12-12-2016.
7. Open Database Connectivity. *Wikipedia*. https://en.wikipedia.org/wiki/Open_Database_Connectivity. Voor het laatst aangepast op 9-12-2016. Bekeken op 12-12-2016.
8. Lightweight Directory Access Protocol. *Wikipedia*. https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol. Voor het laatst aangepast op 12-12-2016. Bekeken op 12-12-2016.
9. Dependency injection. *Wikipedia*. https://en.wikipedia.org/wiki/Dependency_injection. Voor het laatst aangepast op 11-12-2016. Bekeken op 12-12-2016.
10. Ninject – Open source dependency injector for .NET. *Ninject.org*. http://www.ninject.org/. Bekeken op 12-12-2016.
11. Manifest voor Agile Software Ontwikkeling. *Agile Manifesto*. http://agilemanifesto.org/iso/nl/manifesto.html. Gepubliceerd in 2001. Bekeken op 7-12-2016.
12. Agile software development. *Wikipedia*. https://en.wikipedia.org/wiki/Agile_software_development. Voor het laatst aangepast op 6-12-2016. Bekeken op 9-12-2016.
13. Scrum (software development). *Wikipedia*. https://en.wikipedia.org/wiki/Scrum_(software_development). Voor het laatst aangepast op 6-12-2016. Bekeken op 8-12-2016.
14. User Stories and User Story Examples by Mike Cohn. *Mountain Goat Software*. https://www.mountaingoatsoftware.com/agile/user-stories. Bekeken op 9-12-2016.
15. Warmer J, Kleppe A. Hoofdstuk 8: Use-Case-Diagram. In: *Praktisch UML*. 4e editie. Amsterdam: Pearson Education Benelux; 2008: 85-94.
16. Fowler M. *UML Distilled: A Brief Guide tot he Standard Object Modeling Language*. 3e Editie. Boston, MA: Addison-Wesley Professional; 2004.
17. Shalloway A, Trott JR. *Design Patterns Explained: A New Perspective on Object-Oriented Design*. 2e Editie. Pearson Education; 2004.
18. Martin RC. *Agile Software Development: Principles, Patterns, and Practices*. Pearson Education; 2003.
19. Hunt A, Thomas D. *The Pragmatic Programmer: From Journeyman to Master*. 18e Editie. Addison-Wesley; 2006.
20. Framework Design Guidelines. *msdn.microsoft.com* https://msdn.microsoft.com/en-us/library/ms229042(v=vs.110).aspx. Bekeken op 14-10-2016.
21. Cwalina K, Abrams B. *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries*. 2e Editie. Boston, MA: Addison-Wesley Professional; 2008.
22. de Swart N. Hoofdstuk 8: Software requirements. In: *Handboek Requirements. Brug tussen business en ICT*. Delft: Eburon Business; 2010: 67-77.

23. Warmer J, Kleppe A. Hoofdstuk 4: Klasse- en Objectdiagram. In: *Praktisch UML*. 4e editie. Amsterdam: Pearson Education Benelux; 2008: 17-52.
24. Nederlandstalige TMAP Downloads. *TMAP*. http://www.tmap.net/nederlandstalige-tmap-downloads. Bekeken op 15-12-2016.
25. de Grood D-J. Testplan. In: de Grood D-J. *TestGoal. Leerboek resultaatgedreven software testen.* 1e druk. Den Haag: Sdu Uitgevers bv; 2008: 123-143.
26. de Grood D-J. Aanpak: Testsoorten. In: de Grood D-J. *TestGoal. Leerboek resultaatgedreven software testen.* 1e druk. Den Haag: Sdu Uitgevers bv; 2008: 49-51.
27. About SQLite. *Sqlite.org*. https://www.sqlite.org/about.html. Bekeken op 23-9-2016.
28. Most Widely Deployed and Used Database Engine. *Sqlite.org* https://www.sqlite.org/mostdeployed.html. Bekeken op 16-12-2016.
29. File Locking And Concurrency In SQLite Version 3. Sqlite.org https://www.sqlite.org/lockingv3.html. Bekeken op 16-12-2016.
30. Write-Ahead Logging. *Sqlite.org*. https://www.sqlite.org/wal.html. Bekeken op 23-9-2016.
31. Guidelines for Collections. *msdn.microsoft.com*. https://msdn.microsoft.com/en-us/library/dn169389.aspx. Bekeken op 18-12-2016.
32. Enumerable Class (System.Linq). *msdn.microsoft.com*. https://msdn.microsoft.com/en-us/library/system.linq.enumerable(v=vs.110).aspx. Bekeken op 18-12-2016.
33. Generic Methods (C# Programming Guide). *msdn.microsoft.com*. https://msdn.microsoft.com/en-us/library/twcad0zb.aspx. Bekeken op 19-12-2016.
34. Func(T, Result) Delegate (System). *msdn.microsoft.com*. https://msdn.microsoft.com/en-us/library/bb549151(v=vs.110).aspx. Bekeken op 19-12-2016.
35. Delegates (C# Programming Guide). *msdn.microsoft.com*. https://msdn.microsoft.com/en-us/library/ms173171.aspx. Bekeken op 19-12-2016.
36. Timer Class. *msdn.microsoft.com*. https://msdn.microsoft.com/en-us/library/system.timers.timer(v=vs.110).aspx. Bekeken op 14-10-2016.
37. Timer.Elapsed Event. *msdn.microsoft.com*. https://msdn.microsoft.com/en-us/library/system.timers.timer.elapsed(v=vs.110).aspx. Bekeken op 21-10-2016.
38. Microsoft Reference Source .NET Framework 4.6.2 MyTimerCallback. *referencesource.microsoft.com*. https://referencesource.microsoft.com/#System/services/timers/system/timers/Timer.cs,dc67c9466604a3d9,references. Bekeken op 21-10-2016.

39. Advantages of Using DLLs. *msdn.microsoft.com*. https://msdn.microsoft.com/en-us/library/dtba4t8b.aspx. Bekeken op 11-11-2016.
40. de Grood D-J. Logisch testontwerp en testontwerptechnieken. In: de Grood D-J. *TestGoal. Leerboek resultaatgedreven software testen.* 1e druk. Den Haag: Sdu Uitgevers bv; 2008: 153-199.
41. de Grood D-J. Het fysieke testontwerp. In: de Grood D-J. *TestGoal. Leerboek resultaatgedreven software testen.* 1e druk. Den Haag: Sdu Uitgevers bv; 2008: 201-207.
42. Coupling (computer programming). *Wikipedia*. https://en.wikipedia.org/wiki/Coupling_(computer_programming). Bekeken op 18-12-2016.
43. SQLConfigDataSource Function. *msdn.microsoft.com*. https://msdn.microsoft.com/en-us/library/ms716476(VS.85).aspx. Bekeken op 20-12-2016.

9 Begrippenlijst

// Hieronder zijn een aantal begrippen uitgelicht waarvan de betekenis niet duidelijk kan zijn.

Begrip	Uitleg
CRM	Customer Relationship Management. In CRM systemen staat klantinformatie voor het bedrijf dat dit in beheer heeft.
ERP	Enterprise Resource Planning. ERP systemen worden in organisaties gebruikt ter ondersteuning van meerdere bedrijfsprocessen, waarbij de nodige data samengebracht wordt in dit systeem.