# Voorwoord

Voor u ligt het verslag van mijn afstudeerproject bij Tam Tam.

Ik wil Bart Manuel bedanken voor zijn rol als bedrijfsmentor en opdrachtgever. Je manier van begeleiden heeft me veel geleerd.

Bart Venderbosch wil ik bedanken voor zijn rol in de laatste fase van het project. Onze dagelijkse stand-up meetings hebben me erg goed geholpen.

Tot slot wil ik mevrouw Lousberg en meneer Breukel bedanken voor hun begeleiding vanuit school. Ik heb jullie betrokkenheid bij deze begeleiding erg gewaardeerd.

# Inhoudsopgave


Inleiding ..... 1

1 Werkomgeving ..... 2

1.1 Tam Tam ..... 2

1.2 Mixit ..... 2

1.3 Afdeling E-business ..... 2

1.4 Mijn plaats ..... 2

2 Opdracht ..... 3

2.1 Probleemstelling ..... 3

2.2 Doelstelling ..... 4

3 Methodes en technieken ..... 5

3.1 Projectmethode ..... 5

3.2 Architectual Description ..... 7

3.3 Versiebeheer ..... 7

3.4 Ontwikkelomgeving ..... 8

3.5 Testen ..... 8

4 Project fases en planning ..... 9

4.1 Onderzoek huidige situatie ..... 9

4.2 Architectual description ..... 9

4.3 Implementatie koppeling ..... 9

4.4 Testen koppeling ..... 9

4.5 Project planning ..... 9

5 Onderzoek huidige situatie ..... 10

5.1 Interne systemen Tam Tam ..... 10

5.2 Rol CRM systeem ..... 11

5.3 Werking Microsoft Dynamics CRM koppeling ..... 13

6 Functional view ..... 14

6.1 Functionele omschrijving ..... 14

6.2 Functionele en niet-functionele eisen ..... 14

6.3 Onderzoek koppelingsmogelijkheden Capsule CRM ..... 15

6.4 Functioneel ontwerp ..... 15

7 Information view ..... 18
7.1 Datastructuur Capsule API ..... 19
7.2 C# implementatie datastructuur ..... 20
7.3 Relevante datastructuur Tam Tam ..... 21
7.4 Ontwerpen transformatie ..... 22
8 Concurrency view ..... 26
8.1 Verwerking create / update events ..... 27
8.2 Verwerking delete events ..... 28
8.3 Voorkoming concurrency problemen ..... 29
9 Development view ..... 30
9.1 Onderzoek werking Capsule API ..... 30
9.2 Ontwerp implementatie Capsule API ..... 31
9.3 Onderzoek Entity Framework ..... 31
9.4 Ontwerp implementatie schaduwdatabase ..... 32
10 Structuur implementatie ..... 33
10.1 Overzicht structuur ..... 33
10.2 TamTam.Capsule.Data ..... 33
10.3 TamTam.Capsule.SyncService ..... 34
10.4 Tamtam.Capsule.SyncServiceInstaller ..... 34
10.5 TamTam.Capsule.TestApp ..... 35
10.6 Vergaderingen ..... 35
10.7 Planning en sprints ..... 36
11 Implementatie Capsule API ..... 37
11.1 Overzicht implementatie ..... 37
11.2 Ophalen data ..... 37
11.3 Implementatie Capsule dataklasses ..... 37
11.4 Conversie naar klasses ..... 38
11.5 Datamutaties ..... 39
11.6 Conversie naar XML ..... 39
12 Implementatie schaduwdatabase ..... 40
12.1 Opzet ..... 40
12.2 Data toevoegen ..... 40
12.3 Data opvragen ..... 41
12.4 Data wijzigen / verwijderen ..... 42

13 Analyse gegenereerde database ..... 43
13.1 Structuur ..... 43
13.2 Keys en indexen ..... 44
13.3 Datatypes ..... 44
13.4 Data Annotations ..... 45
13.5 Conclusie ..... 45
14 Implementatie SyncService ..... 46
14.1 Opzet ..... 46
14.2 Timers ..... 46
14.3 Foutenafhandeling ..... 46
14.4 Implementatie Tam Tam dataklasses ..... 47
14.5 Verwerking toegevoegde en gewijzigde data ..... 48
14.6 Verwerking verwijderde data ..... 48
14.7 Het 'shortName' probleem ..... 48
15 Testplan ..... 50
15.1 Verlenging project ..... 50
15.2 Opdrachtomschrijving ..... 50
15.3 Opzet testplan ..... 51
15.4 Kwaliteitsattributen ..... 51
15.5 Keuze testsoorten ..... 52
15.6 Testaanpak unittests ..... 53
15.7 Testaanpak integratietests ..... 54
15.8 Testomgeving ..... 55
15.9 Producten ..... 55
15.10 Sprints ..... 55
16 Implementatie unittests ..... 56
16.1 Opzet ..... 56
16.2 Werking unittests ..... 56
16.3 Objecten vergelijken ..... 56
16.4 Voorbeeld unittest ..... 57
16.5 Geen 'datadriven testing' ..... 58
16.6 Tests Synchronizer ..... 58

17 Implementatie integratietests ..... 60
17.1 Aanpak integratietests ..... 60
17.2 Voorbeeld testscript ..... 60
17.3 FillDatabaseFromAPI test ..... 61
18 Testrapport ..... 62
18.1 Resultaten unittests ..... 62
18.2 Resultaten integratietests ..... 62
18.3 Suggesties verdere tests ..... 62
19 Evaluatie ..... 64
20 Beroepstaken ..... 66
20.1 Overzicht beroepstaken ..... 66
20.2 Demonstratie beroepstaken ..... 66
21 Literatuurlijst ..... 67

# Inleiding

Tijdens mij afstudeerproject bij Tam Tam heb ik een koppeling gemaakt tussen Capsule CRM en de interne systemen van Tam Tam.

In dit verslag beschrijf ik het proces van de totstandkoming van deze koppeling. Ook worden de (tussen)producten inhoudelijk besproken.

De opbouw van het verslag is als volgt.
In hoofdstuk 1 wordt de werkomgeving besproken. Vervolgens is in hoofdstuk 2 t/m 4 het plan van aanpak opgenomen.

In hoofdstuk 5 t/m 18 wordt het proces en de (tussen)producten besproken. Hierin komen respectievelijk het onderzoek naar de huidige situatie, de ontwerpfase, de implementatiefase en de testfase aan de orde.

Tot slot wordt er in hoofdstuk 19 een evaluatie gedaan en in hoofdstuk 20 aangetoond hoe de gekozen beroepstaken zijn toegepast binnen het project.

# 1 Werkomgeving

In dit hoofdstuk wordt de werkomgeving besproken.

## 1.1 Tam Tam

Tam Tam behoort tot de top van de Nederlandse full-service internetbedrijven. Tam Tam levert oplossingen op het gebied van online marketing en communicatie, portals en klantspecifieke applicaties. Tam Tam biedt haar klanten verschillende diensten: consultancy, projectmanagement, interaction design, application development en webmasterdiensten, zoals site-analyse.

Tam Tam werkt onder andere voor de volgende klanten:  De Belastingdienst, Bouwfonds, Eneco, Giro 555, Holland Casino, De Hypotheker, Isala Klinieken, Kennisnet, Ministerie van VWS, Mojo Concerts, Oxxio, ProRail, Tebodin, TU Delft, Vrije Universiteit Amsterdam.

Bij Tam Tam zijn ruim negentig medewerkers in dienst.

## 1.2 Mixit

Mixit is een dochterbedrijf van Tam Tam. Mixit richt zich specifiek op oplossingen met betrekking tot online samenwerken. Een voorbeeld hiervan is een bedrijfsportal met Google Apps integratie.

## 1.3 Afdeling E-business

E-bussiness is een afdeling van Tam Tam. Bij E-bussiness ligt de focus op het bouwen van webapplicaties. Veelal zijn deze webapplicaties zogenaamde ‘mijn-omgevingen’. Een voorbeeld hiervan is de ‘mijn Hypotheker’-website.

## 1.4 Mijn plaats

Ik heb mijn afstudeeropdracht uitgevoerd in dienst van Tam Tam. Ik heb dit gedaan op de afdeling E-bussiness. Dit is ook de afdeling waar mijn opdrachtgever en bedrijfsmentor voornamelijk werkt.

Bij het begin van mijn afstuderen bleek dat de opdracht gewijzigd was (zie hoofdstuk 2). Door deze veranderingen voerde ik mijn opdracht niet uit voor Tam Tam, maar in eerste instantie voor het dochterbedrijf Mixit.
Bart Manuel bleef echter mijn opdrachtgever. Hierdoor had ik in de praktijk weinig met Mixit te maken.

# 2 Opdracht

De volgende hoofstukken zijn overgenomen uit het Plan van Aanpak. Zie de bijlagen voor het volledige Plan van Aanpak.

In dit hoofdstuk wordt de opdracht besproken.

## 2.1 Probleemstelling

**Oorspronkelijke probleemstelling**

Bij Tam Tam maakt men momenteel gebruik van het Customer Relation Management (CRM) systeem van Microsoft: 'Microsoft Dynamics CRM 4.0'.

Men is niet tevreden met dit CRM systeem. Hier heeft men verschillende redenen voor:

1. Het systeem nodigt niet uit om te gebruiken.
2. Het systeem is alleen te gebruiken in Internet Explorer. Hierdoor is het systeem alleen onder Windows te gebruiken.
3. Het systeem heeft geen goede smartphone/tablet apps.

Vanwege deze redenen wil men graag overstappen op een SaaS[1] CRM systeem wat wel aan deze voorwaarden voldoet. Men heeft zich al globaal georiënteerd op de verschillende mogelijkheden. Men is erg enthousiast over Highrise[2].
Het nieuwe systeem zal echter wel, net als het huidige systeem, aangesloten moeten worden op de ESB (Enterprise Service Bus). Ook zal de data uit het oude systeem overgezet moeten worden naar het nieuwe systeem.

**Wijziging probleemstelling**

Aan het begin van het project bleek dat de probleemstelling gewijzigd was.

De wijziging van de probleemstelling werd door twee dingen veroorzaakt:

1. Door het horen van negatieve ervaringen met Highrise was men hier niet langer enthousiast over.
2. Het dochterbedrijf Mixit (zie hoofdstuk 1) was al overgestapt op het gebruik van het SaaS CRM systeem 'Capsule CRM'.

Naar aanleiding van deze veranderde situatie werd een nieuwe probleemstelling opgesteld.

**Gewijzigde probleemstelling**

Bij Mixit is men overgegaan op het SaaS CRM systeem 'Capsule CRM'[3].

Bij de overgang heeft men de data uit Microsoft Dynamics CRM geëxporteerd naar Excel formaat en geïmporteerd in Capsule CRM.

Er is echter nog geen koppeling tussen Capsule CRM en de interne systemen van Mixit.

Tam Tam wil in een later stadium mogelijk ook overgaan op het gebruik van Capsule.

[1] Software as a Service. Dit is software die als online dienst wordt aangeboden. Hierdoor wordt de klant zaken als installatie, onderhoud en beheer uit handen genomen.
[2] http://highrisehq.com
[3] http://capsulecrm.com

## 2.2 Doelstelling

**Oorspronkelijke doelstelling**

Een succesvolle overgang uitvoeren naar een SaaS CRM systeem aan de hand van drie stappen:

1. Bepalen welk SaaS CRM systeem het best geschikt is voor Tam Tam.
2. Een koppeling tussen het gekozen SaaS CRM systeem en de interne systemen van Tam Tam ontwerpen en implementeren.
3. De migratie naar dit nieuwe CRM systeem ontwerpen en uitvoeren.

**Wijziging doelstelling**

Door de gewijzigde probleemstelling verandert ook de doelstelling.

**Gewijzigde doelstelling**

Een koppeling tussen Capsule CRM en de interne systemen van Mixit ontwerpen en implementeren.

Aangezien de interne systemen van Mixit een kopie zijn van de systemen van Tam Tam, zal wanneer Tam Tam overgaat op het gebruik van Capsule CRM, de koppeling ook voor Tam Tam gebruikt kunnen worden.

# 3 Methodes en technieken

In dit hoofdstuk worden de methoden en technieken besproken.

## 3.1 Projectmethode

Deze paragraaf de gebruikte projectmethode: scrum.

### 3.1.1 Wat is scrum?

In de volgende paragrafen zal wordt kort besproken wat scrum precies is.

**Definitie**

Scrum is een iteratieve projectbeheersingsmethode. Scrum wordt vaak gebruikt bij agile software ontwikkeling.

**Project backlog**

Elk scrumproject heeft een 'project backlog'.
De project backlog beschrijft de taken die binnen het project uitgevoerd moeten worden. Verder worden de resulterende producten van deze taken beschreven.

**Sprints**

Bij scrum wordt het project opgedeeld in sprints.
Een sprint heeft doorgaans een lengte van 1 tot 4 weken. Alle sprints binnen een project hebben dezelfde lengte.
Aan het begin van een sprint wordt de 'sprint backlog' opgesteld. Voor de sprint backlog worden 1 of meer taken van de project backlog opgesplitst en gedetailleerder beschreven.

**Daily scrum**

Bij scrum kan gewerkt worden met een dagelijkse vergadering, de 'daily scrum'.
Deze vergadering heeft een maximale duur van 15 minuten.
Tijdens de vergadering beantwoordt elk teamlid de volgende vragen:

- Wat heb je gisteren gedaan?
- Wat ga je vandaag doen?
- Zijn er 'impediments' (belemmeringen die de uitvoering van het werk in de weg staan)?

**Rollen**

Scrum onderscheid verschillende rollen:

| Rol | Omschrijving |
|---|---|
| Scrummaster | De scrummaster zorgt ervoor dat het team zijn werk goed kan doen. Hij zorgt dat 'impediments' uit de weg worden geruimd. |
| Product owner | De product owner is de 'stem van de klant'. Hij is er verantwoordelijk voor dat de klant krijgt wat hij wil. Hij bereikt dit door de taken te omschrijven en prioriteit toe te kennen.<br>Als de klant betrokken genoeg is kan deze rol door de klant zelf vervuld worden. Wanneer dit niet het geval is wordt de rol door een teamlid vervuld. |
| Team | Het team voert het daadwerkelijke werk uit. Een scrum team bestaat doorgaans uit 5-9 personen. |

### 3.1.2 Scrum binnen het project

**Overzicht rolverdeling**

Tijdens het project zullen de volgende personen betrokken zijn:

| Wie | Rol |
|---|---|
| Bart Manuel | Opdrachtgever,<br>Hoofdarchitect interne systemen Tam Tam/ Mixit |
| Mark Lagendijk | Scrummaster, product owner en team (projectleider, architect, developer, tester) |
| Mike Noordermeer, Marco Krikke | Developer klantenportal |

**Rol Mark Lagendijk**

Zoals in de bovenstaande tabel te zien is zal ik alle drie de primaire scrum rollen vervullen: scrummaster, product owner en team.

Hier volgt per rol de motivatie voor deze keuze:

- Team:
  Aangezien ik de enige is die het werk daadwerkelijk uitvoert, ben ik (volgens de definitie binnen scrum) ook het enige teamlid.

- Scrummaster:
  De scrummaster is ervoor verantwoordelijk dat eventuele impediments uit de weg worden geruimd.
  Ik ervaar zelf deze eventuele impediments, het is daarom ook logisch dat ik ervoor verantwoordelijk ben dat deze worden opgelost.

- Product owner:
  De product owner houdt zich tot op detailniveau bezig met het bepalen van de uit te voeren taken. Verder bepaalt hij de prioriteit van deze taken.
  Het vervullen van deze rol eist redelijk veel tijd. Wanneer deze rol niet door mij zou worden vervuld, zou degene die de rol vervult minimaal een dagelijkse vergadering (daily scrum) met mij moeten hebben.
  De opdrachtgever is, als partner van Tam Tam, erg druk bezet. Ik heb er daarom voor gekozen om de opdrachtgever niet te vragen of hij deze rol wilde vervullen, maar hem zelf te vervullen.
  Het gevaar van zelf de product owner rol vervullen is dat de opdrachtgever niet meer genoeg bij het proces betrokken wordt. Wanneer dit gebeurt, bestaat het gevaar dat het uiteindelijke product niet overeen komt met de wensen van de opdrachtgever. Om dit te voorkomen moet de opdrachtgever bij alle belangrijke beslissingen betrokken worden.

**Rol Bart Manuel**

Bart Manuel zal naast de rol van opdrachtgever, ook de rol van systeem architect vervullen. Bart is binnen Tam Tam de opdrachtgever voor de ontwikkeling van de verschillende interne systemen. Vanwege zijn technische kennis van de interne systemen is hij ook stakeholder in de rol hoofdarchitect.

**Rol Mike Noordermeer en Marco Krikke**

Tijdens de implementatiefase van het project zullen Mike Noordermeer en Marco Krikke bij het project betrokken zijn. Als developers van de interne systemen van Tam Tam / Mixit zullen zij technisch advies geven tijdens de implementatie.
Verder zullen zij eventuele, voor de implementatie van de koppeling benodigde, wijzigingen aan de interne systemen van Mixit uitvoeren.

**Sprints**
In het de ontwerpfase van het project zal niet met sprints worden gewerkt. Deze fase zal meer volgens de klassieke watervalmethode worden uitgevoerd.
De reden hiervoor was de keuze om überhaupt een aparte ontwerpfase te hebben. Bij scrum wordt doorgaans het ontwerp tegelijk met de implementatie gemaakt. Vanwege de complexiteit van de koppeling was echter besloten een eerst een architectual description te maken.

Tijdens de development- en testfase van het project zal wel met sprints worden gewerkt. Deze sprints zullen een duur hebben van 1 week.
Er is gekozen voor een sprintlente van 1 week vanwege de duur van het afstudeerproject. Wanneer sprints langer dan 1 week zouden duren, zouden er maar enkele sprints zijn. Daardoor zouden de voordelen van het hebben van sprints verdwijnen.

## 3.2 Architectual Description
Het ontwerp van de koppeling zal worden gemaakt in de vorm van een zogenaamde 'architectual description'. Dit zal worden gedaan volgens de methode zoals beschreven in het boek 'Software Systems Architecture'.

**Viewpoints**
Het systeem zal beschreven worden vanuit verschillende 'viewpoints'. Elk viewpoint belicht een anders aspect van het systeem.
Er zal een selectie worden gemaakt uit de volgende viewpoints:

- Functional
- Information
- Concurrency
- Development
- Deployment
- Operational

**Perspectives**
Ook zullen er 1 of meerdere 'perspectives' worden toegepast. Voorbeelden van perspectives zijn 'evolution' en 'availability and resilience'.

**Modellen en diagrammen**
De modellen en diagrammen zullen met UML gemaakt worden.
Indien het de begrijpbaarheid van de Architectual Description ten goede komt, kan er ook gebruik worden gemaakt van vrije diagrammen. Aangezien er geen niet-technische stakeholders bij het project betrokken zijn, zal zoveel mogelijk UML gebruikt worden.

## 3.3 Versiebeheer
Binnen Tam Tam worden verschillende versiebeheersystemen gebruikt.
In het verleden gebruikte men 'Vault'. Enkele jaren geleden is men overgestapt op het gebruik van SVN. Bij deze overstap zijn bestaande projecten echter niet overgezet.

De code van de interne systemen van Tam Tam bevindt zich hoofdzakelijk in Vault. De code van nieuwe interne projecten wordt echter bij voorkeur in SVN gezet.

In overleg met de opdrachtgever is besloten om de code van de koppeling in SVN te zetten.

## 3.4 Ontwikkelomgeving

Bij Tam Tam worden  de meeste projecten met het .NET platform van Microsoft gemaakt. Hierbij wordt gebruik gemaakt van de IDE 'Visual Studio' en de programmeertaal 'C#'. Voor het maken van websites / webapplicaties maakt men gebruik van ASP.NET en ASP.NET MVC.

Vanwege de ervaring met C#.NET zijn de interne systemen ook hiermee gemaakt. Vanuit deze situatie volgt als vanzelfsprekend de eis om de koppeling ook met C#.NET te maken.

## 3.5 Testen

Voor de testfase zal het boek 'TestGoal' als handboek gebruikt worden.
Er zullen verschillende soorten tests uitgevoerd worden.

**Unittests**

Voor het testen van de koppeling zullen unittests gemaakt worden.
Met unittests kunnen verschillende onderdelen van een systeem afzonderlijk getest worden.
Een ander voordeel van unittests is dat ze geautomatiseerd uitgevoerd kunnen worden, bijvoorbeeld bij elke build van het systeem. Hierdoor zijn unittests zeer geschikt als regressietest.

**Integrationtests**

Naast unittests zal er ook een integratietest uitgevoerd worden.
Unittests zijn zeer geschikt om functionaliteit van afzonderlijke modules te testen. Ze testen alleen niet de samenwerking tussen de verschillende modules. Om de samenwerking tussen de modules te testen zal daarom gebruik gemaakt worden van integrationtests.

# 4 Project fases en planning

In dit hoofdstuk worden de verschillende fases van het project worden besproken. Vervolgens wordt in 4.5 de globale planning van het project weergegeven.

## 4.1 Onderzoek huidige situatie

Allereerst zal de huidige situatie onderzocht moeten worden.

Dit onderzoek zal zich met name richten op de koppeling tussen Microsoft Dynamics CRM en de interne systemen van Tam Tam. De te bouwen koppeling zal dezelfde functionaliteit moeten bieden als deze bestaande koppeling.

## 4.2 Architectual description

Het ontwerp van de koppeling zal worden gemaakt in de vorm van een 'architectual description' (zie 3.2).

Bij het opstellen van de requirements zal het onderzoek naar de bestaande koppeling als uitgangspunt worden genomen.

Er zal worden onderzocht hoe de koppeling met Capsule CRM kan worden gebouwd met dezelfde functionaliteit als de bestaande koppeling. Op basis van dit onderzoek zal het verdere ontwerp worden gemaakt.

## 4.3 Implementatie koppeling

De koppeling zal worden geïmplementeerd met C#.NET (zie 3.4).
Bij de implementatie zal worden uitgegaan van de architectual description.

## 4.4 Testen koppeling

De koppeling zal ook formeel getest worden. Hiervoor zullen er verschillende soorten tests gemaakt worden (zie 3.5).

## 4.5 Project planning

In deze paragraaf wordt de globale planning voor het project weergegeven.

| Wanneer | Wat |
|---|---|
| Week 1 t/m 3 | Opstellen Plan van Aanpak.<br>Onderzoek huidige situatie. |
| Week 4 t/m 7 | Opstellen Architectual Description. |
| Week 8 t/m 12 | Implementeren koppeling. |
| Week 13 t/m 15 | Opstellen testplan.<br>Testen koppeling. |
| Week 1 t/m 17 | Opstellen afstudeerverslag. |

# 5 Onderzoek huidige situatie

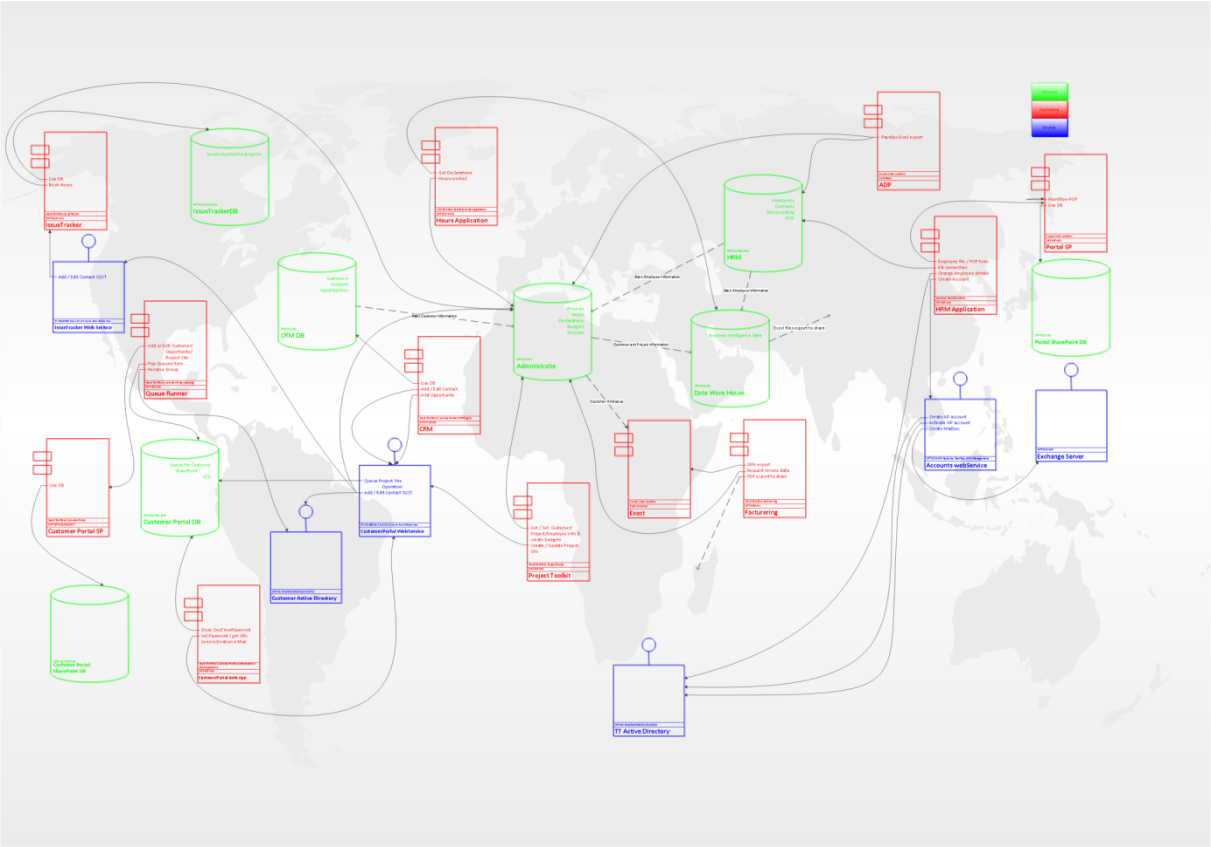
In dit hoofdstuk wordt het onderzoek naar de huidige situatie beschreven.

## 5.1 Interne systemen Tam Tam

De opdracht gaat over het maken van een koppeling tussen Capsule CRM en de interne systemen van Tam Tam / Mixit.
In de opdrachtomschrijving bleek al dat de koppeling aangesloten moet worden op de ESB van Tam Tam. De koppeling zal dus niet direct met andere interne systemen communiceren, maar alleen met de ESB. Hierdoor zijn deze interne systemen, vanuit de koppeling gezien, verder niet interessant.

Het is echter wel interessant om een idee te krijgen van het geheel waarvan de koppeling deel zal maken. Er is daarom een kort onderzoek gedaan naar wat dit zoal voor systemen zijn.
Dit onderzoek is gebeurd op basis van de documentatie van een eerdere afstudeerder. Deze afstudeerder had de opdracht om de huidige architectuur in kaart te brengen en een voorstel tot verbetering in te dienen.

Hieronder volgt zijn overzichtsdiagram van de huidige architectuur. NB: om het diagram leesbaar te maken zou het op a3 formaat moeten worden uitgeprint. Het diagram dient hier slechts ter illustratie van de complexiteit van de interne systemen.




Hieronder worden verschillende van de in het diagram getoonde systemen besproken. Deze lijst is niet volledig, maar is bedoeld om een beeld van de situatie te schetsen.

**Portal**
De portal is de ruggenmerg van het intranet van Tam Tam. Verschillende applicaties van Tam Tam, die hieronder beschreven worden, zijn onderdeel van de portal.

**Urenboekapplicatie**
In de urenboekapplicatie kunnen medewerkers aangeven hoeveel ze gewerkt hebben en aan welke projecten / overige werkzaamheden ze deze tijd besteed hebben.
Ook de administratie van de reiskosten en aanwezigheid bij de lunch gebeurt met behulp van deze applicatie.

**HRM-applicatie**
De HTM-applicatie is gemaakt om de werkzaamheden van de HRM afdeling te vergemakkelijken. In de applicatie kunnen nieuwe medewerkers worden aangemaakt en de voortgang van medewerkers worden bewaakt. Een belangrijk onderdeel van de applicatie is het weergeven van verschillende data, zoals welke verjaardagen komen eraan of wanneer loop iemands contract af.

**Project toolkit**
In de project toolkit worden projecten beheerd. Projectleiders kunnen projecten aanmaken en de budgetten voor hun projecten beheren.
Wanneer een project wordt aangemaakt wordt er automatisch een bijbehorende projectsite gecreëerd.

**Facturatie-applicatie**
Met de facturatie-applicatie kan de Finance afdeling de facturen voor een bepaalde maand laten aanmaken. Deze facturen worden dan gegenereerd in de vorm van PDF-bestanden. Ook wordt er een XML-bestand gegenereerd, wat in de boekhoudingssoftware kan worden geïmporteerd.

**ADP**
ADP is een applicatie waarin de salarisadministratie kan worden bijgehouden. Werknemers kunnen op een website inloggen om hun salarisstroken te bekijken.

**Customer portal**
Via de customer portal kunnen medewerkers en klanten de verschillende projectsites bezoeken. De medewerkers kunnen documenten over het project opslaan. De klant kan onder andere op deze documenten reageren en zo op een eenvoudige manier samenwerken met de medewerkers.

Ook de opportunitysites zijn via deze portal te bereiken. Deze zijn vergelijkbaar met de projectsites, behalve dat ze alleen toegankelijk zijn voor medewerkers van Tam Tam / Mixit.

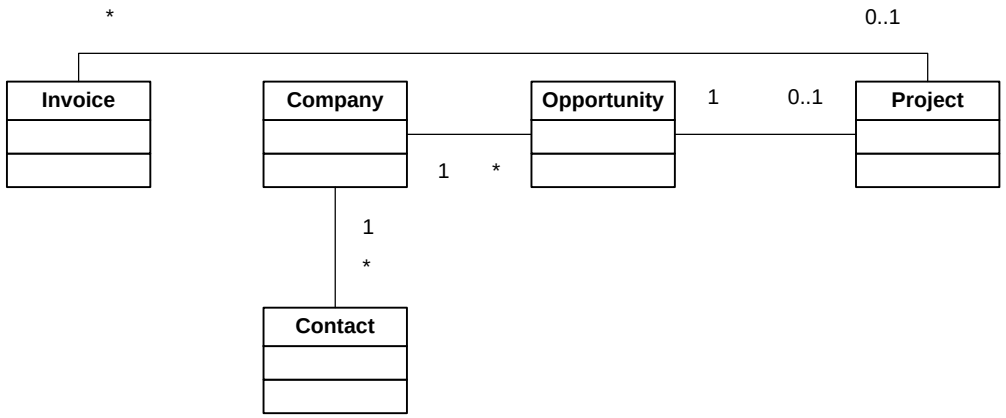
## 5.2 Rol CRM systeem
Tam Tam en Mixit gebruiken hun CRM systeem op een vrij basale manier. Dit was de reden dat het voor Mixit geen problemen opleverde om vanaf, het erg uitgebreide, Microsoft Dynamics CRM over te stappen op het, vrij eenvoudige, Capsule CRM.

**CRM entiteiten**
De volgende entiteiten worden bijgehouden in het CRM systeem. De tabel toont de naamgeving van Tam Tam, Microsoft Dynamics CRM en Capsule CRM voor deze entiteiten:

| Tam Tam | MS Dynamics CRM | Capsule CRM | Definitie |
|---|---|---|---|
| Company | Account | Organisation | (potentieel) klantbedrijf |
| Contact | Contact | Person | werknemer van klantbedrijf |
| Opportunity | Opportunity | Opportunity | potentiele opdracht |

**Rol CRM entiteiten**

Hoewel dit slechts drie entiteiten zijn spelen ze wel een belangrijke rol in de datastructuur van Tam Tam. Onder andere de entiteit 'Project' en 'Invoice' (factuur) zijn ervan afhankelijk. Het volgende klassendiagram geeft dit weer:




Ik heb dit diagram opgesteld aan de hand van de beschrijvingen van de interne systemen uit 6.1. Het diagram laat zien hoe een belangrijke entiteit als 'project' afhankelijk is van de entiteiten uit het CRM systeem. Hierdoor zijn veel van de interne systemen direct of indirect afhankelijk van de date uit het CRM systeem.
Het diagram toont uiteraard slechts een klein gedeelte van de datastructuur van Tam Tam. Het dient slechts ter illustratie van de genoemde afhankelijkheid.

Een mooi voorbeeld van de afhankelijkheid van de data uit het CRM is de Customer Portal. Medewerkers (Contacts) van een klantbedrijf (Company) kunnen inloggen op de Customer Portal en krijgen dan de projectsites van hun bedrijf te zien.

**Events CRM entiteiten**

Voor de werking van verschillende applicaties is het nodig dat er bepaalde acties worden uitgevoerd op het moment dat er data in het CRM systeem toegevoegd, gewijzigd of verwijderd wordt.

Zo moet er bijvoorbeeld een opportunitysite worden aangemaakt wanneer er een Opportunity is toegevoegd.
Een ander voorbeeld is het aanmaken van een Contact. Een Contact moet na het aanmaken aan een Active Directory worden toegevoegd zodat hij kan inloggen op de Customer Portal.

Dit wordt gerealiseerd door middel van een koppeling tussen het CRM systeem en de ESB. Deze koppeling zorgt ervoor dat wanneer er data wijzigt in het CRM systeem, deze wijzigingen worden doorgegeven aan de ESB.
De ESB zorgt er vervolgens voor dat de wijzigingen aan de interne systemen worden doorgegeven

zodat de juiste acties kunnen worden ondernomen (bijvoorbeeld het aanmaken van een opportunitysite).

## 5.3 Werking Microsoft Dynamics CRM koppeling

De te bouwen koppeling moet vergelijkbare functionaliteit gaan bieden als de bestaande koppeling met Microsoft Dynamics CRM.
Als hulpmiddel voor het bepalen van de functionele eisen is er onderzoek gedaan naar de werking van deze koppeling. Hierbij is gekeken naar de code van de koppeling.

Allereerst is onderzocht welke mogelijkheden Microsoft Dynamics CRM biedt om systemen aan het CRM systeem te koppelen.
Vervolgens is onderzocht hoe de koppeling van deze mogelijkheden gebruik maakt.

**Koppelingsmogelijkheden Microsoft Dynamics CRM**

Microsoft Dynamics CRM bleek verschillende functies te hebben die een rol spelen bij het maken van een koppeling:

- Ondersteuning voor zelfgemaakte plugins.
  Door zelf een plugin te schrijven kan de gebruiker eigen code koppelen aan bepaalde gebeurtenissen (‘events’) binnen het systeem.
- Er zijn events voor alle entiteiten die in het CRM systeem kunnen worden opgeslagen. Voor elke entiteit zijn er 3 soorten events: create, update en delete.
- De gegevens van een event (bijvoorbeeld: welke ‘Company’ is gewijzigd en wat zijn de wijzigingen?) worden meegegeven aan de gekoppelde code.
- De eigen code kan een foutmelding laten tonen. De wijziging(en) van het event worden dan niet opgeslagen.

**Werking Microsoft Dynamics koppeling**

De koppeling bleek vrij eenvoudig te werken.
De koppeling is gemaakt in de vorm van een plugin. De plugin koppelt code aan alle events die een bijbehorende orchestratie[4] in het ESB hebben. De gekoppelde code doet het volgende:

1. De code roept een orchestratie van de ESB aan met de gegevens van het event.
2. Wanneer de orchestratie een foutmelding teruggeeft, wordt de foutmelding aan de gebruiker getoond.

[4] Naam van een methode / procedure van een ESB

# 6 Functional view

In dit hoofdstuk wordt het functionele ontwerp besproken.

## 6.1 Functionele omschrijving

In Capsule CRM kan data worden toegevoegd, gewijzigd of verwijderd. Wanneer er een dergelijke gebeurtenis ('event') plaatsvind moeten er bijbehorende acties in de interne systemen van Tam Tam uitgevoerd worden.
Het doel van de te bouwen oplossing is een koppeling vormen tussen Capsule CRM en de interne systemen van Tam Tam. Deze koppeling herkent de events in Capsule CRM en zorgt ervoor dat de bijbehorende acties in de systemen van Tam Tam uitgevoerd worden.

## 6.2 Functionele en niet-functionele eisen

Na het uitvoeren van het onderzoek naar de bestaande koppeling (6.3) en het onderzoek naar de koppelingsmogelijkheden van Capsule CRM (7.3) is er een gesprek met de opdrachtgever gevoerd voor het bepalen van de functionele en niet-functionele eisen. Op basis van de bevindingen van de onderzoeken en dit gesprek zijn de volgende eisen opgesteld:

| | **Eigenschap** | **Beschrijving** |
|---|---|---|
| 1 | Herkenning events | Het systeem herkent het toevoegen, wijzigen en verwijderen (create, update, delete) van data in Capsule CRM. Het gaat hierbij om de volgende entiteiten: 'Organisation', 'Person' en 'Opportunity'. |
| 2 | Herkenning noodzaak orchestratie | Het systeem is in staat te bepalen wanneer er op basis van een event een actie in de systemen van Tam Tam uitgevoerd moet worden. |
| 3 | Aanroepen orchestraties | Het systeem is in staat om deze acties uit te laten voeren, doormiddel van het aanroepen van de juiste 'orchestratie' van de ESB. |
| 4 | Snelheid create / update events | Het herkennen en verwerken van create en update events gebeurt binnen 5 minuten na het optreden van het event. |
| 5 | Snelheid delete | Het herkennen en verwerken van delete events gebeurt minimaal 1 keer per dag. |
| 6 | Foutbestendigheid | Het systeem is foutbestendig. Wanneer er een fout optreedt tijdens het verwerken van een event, blijft het systeem wel draaien. |
| 7 | Rollback | Het systeem is in staat om een event terug te draaien, wanneer een orchestratie van het ESB een foutmelding geeft. |
| 8 | Foutmelding | Wanneer er een rollback plaatsvindt, stelt het systeem de gebruiker hiervan op de hoogte. |
| 9 | Vervangbaarheid | Bij het ontwerp van het systeem is rekening gehouden met vervangbaarheid. Het vervangen van de koppeling door een vergelijkbare koppeling met een ander SaaS CRM systeem dient zo eenvoudig mogelijk te zijn. |

**Toelichting**

Eis 1 t/m 3 en 8 zijn functionele eisen die afgeleid zijn van de werking van de bestaande koppeling.
Eis 6 en 9 zijn kwalitatieve eisen die in het gesprek met de opdrachtgever aan de orde kwamen.
Eis 7 is een nieuwe functionele eis van de opdrachtgever.
Eis 4 en 5 komen voort uit het onderzoek naar de koppelingsmogelijkheden van Capsule CRM. Op basis van dit onderzoek werd duidelijk dat het verwerken van de create en update events wezenlijk anders werkt dan het verwerken van delete events.

## 6.3 Onderzoek koppelingsmogelijkheden Capsule CRM

Na het onderzoeken van de Microsoft Dynamics koppeling en het opstellen van de functionele eisen is er onderzoek gedaan naar de koppelingsmogelijkheden van Capsule CRM.

Capsule CRM heeft geen ondersteuning voor plugins. De enige beschikbare manier om eigen systemen te koppelen bleek de Capsule API te zijn.

Deze API bleek de volgende eigenschappen te hebben:

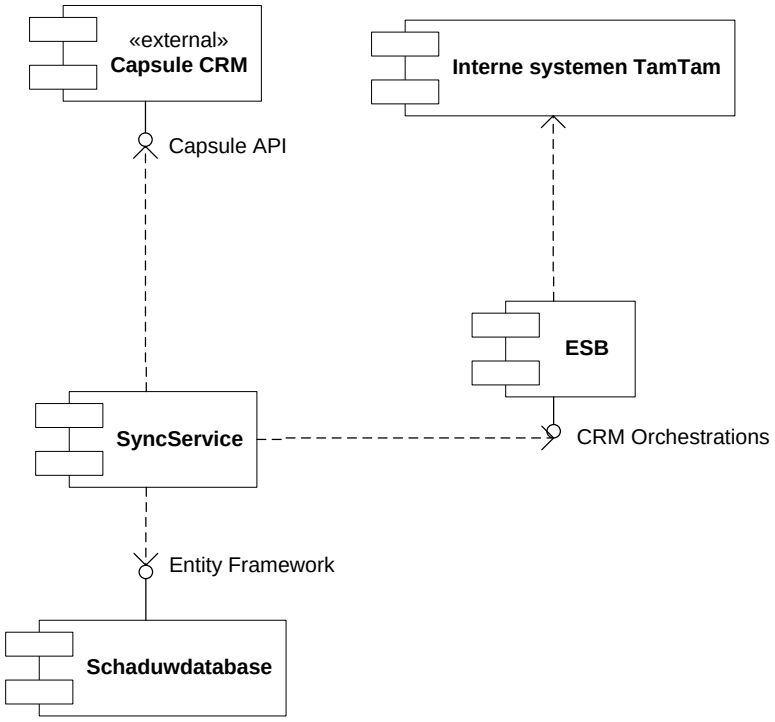
- De API kan gebruikt worden data uit het CRM systeem op te vragen. Een applicatie kan bijvoorbeeld *alle Organisations* opvragen.
- De API houdt geen historie bij. Hierdoor kan er niet via de API worden opgevraagd of er data verwijderd is, of welke wijzigingen er hebben plaatsgevonden
- De API biedt wel de mogelijkheid om alleen nieuwe of gewijzigde data op te halen. Hiermee kan een applicatie bijvoorbeeld *alle Organisations* opvragen die *sinds 5 december 10:00* aangemaakt of gewijzigd zijn. Er is dan echter nog niet duidelijk welke nieuw zijn en welke gewijzigd. Ook is van de gewijzigde items niet bekend wat de wijziging was.
- Via de API kan data toegevoegd, gewijzigd of verwijderd worden.

## 6.4 Functioneel ontwerp

Voor het opstellen van het functioneel ontwerp zijn zowel de functionele eisen als de resultaten van het onderzoek naar de koppelingsmogelijkheden van Capsule CRM, als uitgangspunt genomen.

### 6.4.1 Componentdiagram

Het functioneel ontwerp bestaat uit het volgende componentendiagram met beschrijving:




**Toelichting**

De koppeling bestaat uit de twee componenten ‘SyncService’ en ‘schaduwdatabase’. Verder is te zien

dat de SyncService met drie andere componenten communiceert: Capsule CRM, de Schaduwdatabase en de ESB.
De keuze voor dit ontwerp wordt verder toegelicht aan de hand van de beschrijving van de verschillende componenten.

### 6.4.2 Capsule CRM

Dit is het Capsule CRM system. Het CRM system is te benaderen via een API.
Uit onderzoek naar de mogelijkheden van de API bleek dat het niet mogelijk is om de events die hebben plaatsgevonden op te vragen. Het is alleen mogelijk om data zelf op te vragen. Wel is het mogelijk om alleen die data op te vragen die sinds een bepaalde tijd toegevoegd / of gewijzigd is.

### 6.4.3 Schaduwdatabase

**De uitdaging**
Er moest dus een oplossing bedacht worden om toch te kunnen bepalen welke events er hebben plaatsgevonden. Uiteindelijk is als oplossing de schaduwdatabase bedacht.
Het idee is simpel: wanneer je wilt bepalen welke veranderingen een situatie ondergaan heeft kan dit gebeuren door de beginsituatie te vergelijken met de huidige situatie.

**Voorbeeld**
Om 13:00 ligt er een bal in het midden van een voetbalveld.
Om 13:10 ligt de bal in een van de doelen.
Conclusie: in de 10 verstreken minuten is de bal vanaf het midden van het veld verplaatst naar het doel.

Zoals te zien is betekend dit wel dat de gebeurtenissen worden samengevat. Het feit dat de bal mogelijk verschillende keren heel het veld rond gegaan is wordt buiten beschouwing gelaten.

**Werking schaduwdatabase**
Het idee van het gebruik van de schaduwdatabase berust op dit concept. De schaduwdatabase bevat de situatie van de data in Capsule CRM op moment ‘A’. Wanneer we op moment ‘B’ de data uit Capsule CRM vergelijken met de schaduwdatabase, kunnen we bepalen welke events er hebben plaatsgevonden.
Door de veranderingen ook weer op de data in de schaduwdatabase toe te passen kan even later het proces weer herhaald worden.

**Performance**
Om te bepalen naar de haalbaarheid van dit concept is er onderzocht hoeveel data er in Capsule CRM is opgeslagen. Uit dit onderzoek bleek dat er niet meer dan enkele honderden Organisations zijn opgeslagen. Elke Organisation heeft slechts enkele People. De hoeveelheid Opportunities was beperkt tot minder dan honderd.

Aangezien deze aantallen in verhouding erg laag liggen en er geen aanleiding is om rekening te houden met explosieve groei, worden er geen performance problemen verwacht.

### 6.4.4 SyncService

De koppeling bestaat uit de SyncService in combinatie met de schaduwdatabase.
De SyncService bepaald periodiek of er data in Capsule is toegevoegd, gewijzigd of verwijderd. Er zijn hierbij twee onafhankelijke verwerkingscycli:

- De herkenning en verwerking van create en update events. Deze cyclus heeft een interval van maximaal 5 minuten.
- De herkenning en verwerking van delete events. Deze cyclus heeft een interval van maximaal 24 uur.

Het herkennen van de events gebeurt met behulp van een schaduwdatabase. Voor elk event wordt de bijbehorende orchestratie van de ESB aangeroepen. Na het uitvoeren van de orchestraties worden de veranderingen ook doorgevoerd in de schaduwdatabase.
Wanneer het uitvoeren van een orchestratie mislukt, voert de SyncService een rollback uit door de wijziging in Capsule ongedaan te maken. Dit proces wordt beschreven in hoofdstuk 9.

# 7 Information view

In dit hoofdstuk worden de datamodellen besproken die een rol spelen binnen de koppeling. Dit zijn de volgende modellen:

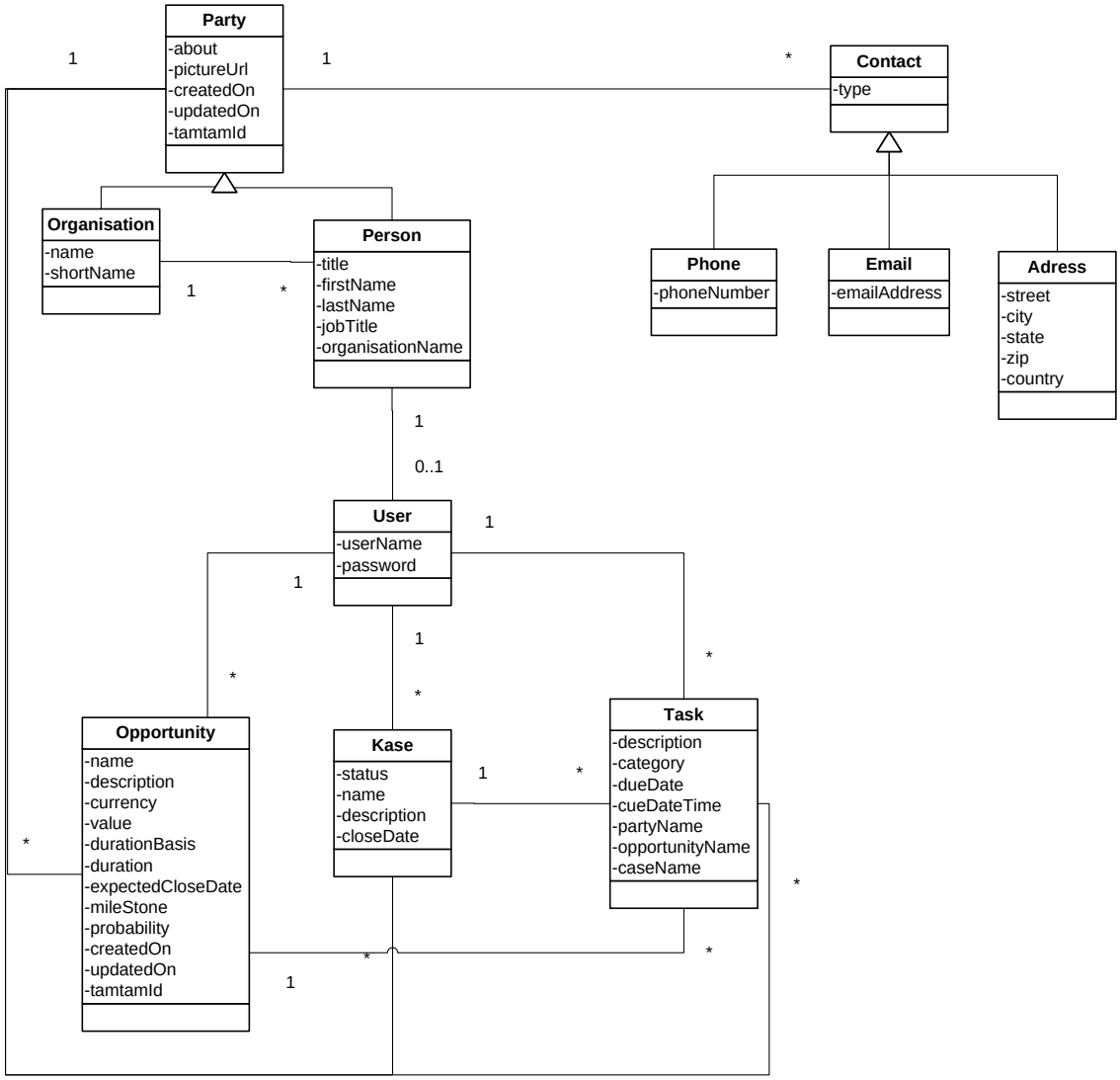
1. Het datamodel van Capsule CRM (8.1).
2. De implementatie van dit model in C# code (8.2).
3. Het datamodel van de relevante Tam Tam klasses (8.3).

Verder wordt de transformatie vanaf de geïmplementeerde Capsule klasses naar de Tam Tam klasses weergegeven (8.4).

## 7.1 Datastructuur Capsule API

De eerste stap was het in kaart brengen van de datastructuur van de Capsule API.

Deze stap is uitgevoerd op basis van reverse engineering met behulp van de documentatie van de Capsule API[5]. Als primary en foreign keys worden numerieke ids gebruikt. Deze zijn weggelaten in dit diagram.




**Het shortName property**

In het diagram is te zien dat Organisation een 'shortName' property heeft. Dit property zit standaard niet in Capsule CRM, maar is door Mixit zelf toegevoegd. Zoals de naam al zegt bevat dit veld een korte versie van de bedrijfsnaam. Het veld heeft een maximum van 20 karakters.

**Het tamtamId property**

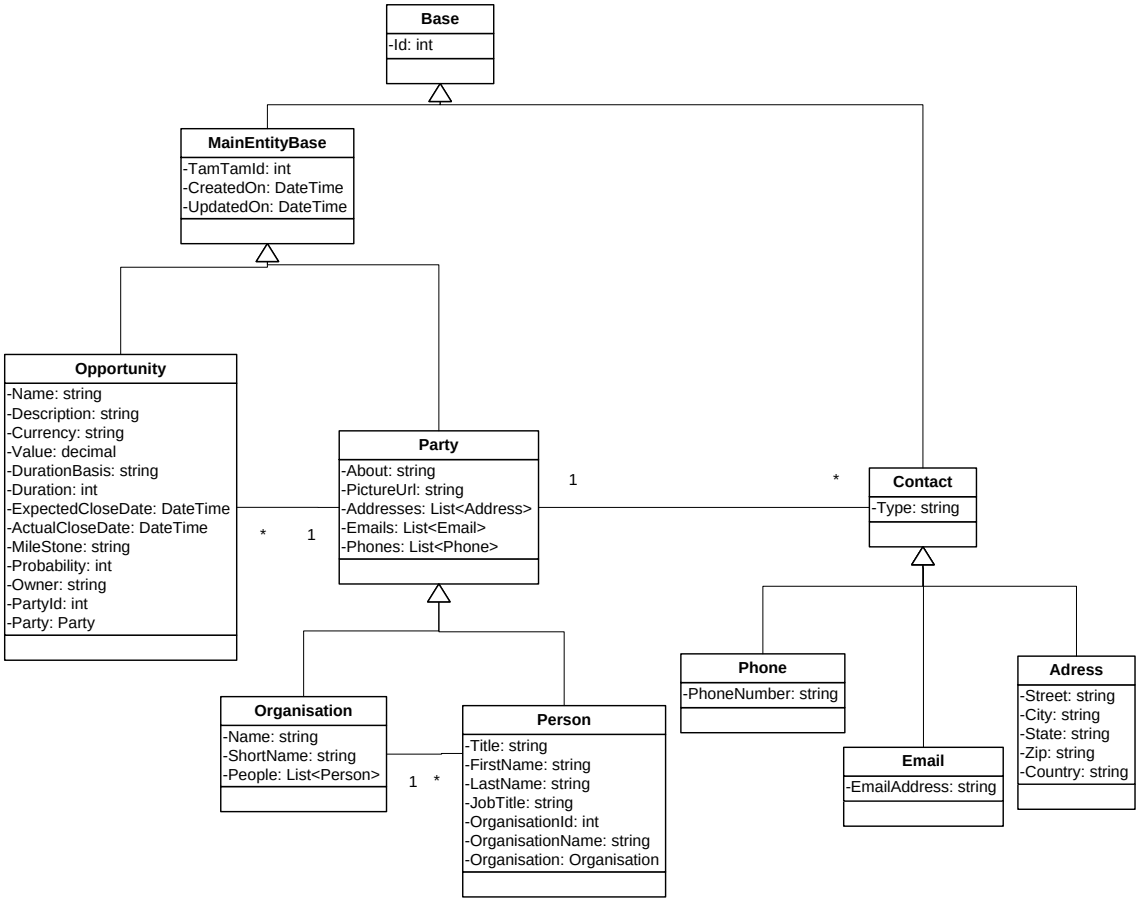
Zowel Organisation als Person en Opportunity hebben het 'tamtamId' property. Dit is ook een toegevoegd veld. Dit veld is toegevoegd voor de werking van de koppeling.
De reden voor dit veld is dat wanneer het standaard 'id' veld gebruikt zou worden er problemen

[5] http://capsulecrm.com/help/page/api_gettingstarted

zouden kunnen ontstaan in de database(s) van de interne systemen van Tam Tam. Het zou namelijk kunnen gebeuren dat een id al voorkomt in deze database(s).
Om dit te voorkomen is het tamtamId veld toegevoegd. Bij het aanmaken van een object zorgt de ESB voor het genereren van dit id. De ESB geeft het id terug aan de koppeling, die vervolgens het id weer in Capsule plaatst.

## 7.2 C# implementatie datastructuur

De tweede stap was het ontwerpen van de C# implementatie van de in kaart gebrachte datastructuur van de Capsule API. Deze implementatie is nodig om de data uit Capsule CRM te kunnen gebruiken in de koppeling.




**Selectie klasses**

In de functionele eisen was vastgesteld dat het bij het herkennen van de events gaat om de klassen ‘Opportunity’, ‘Organisation’ en ‘Person’. Omdat de klassen ‘User’, ‘Kase’ en ‘Task’ geen rol spelen zijn deze niet meegenomen in het ontwerp van de C# implementatie.

De alle Contact klasses worden ook geïmplementeerd. Deze klasses zijn een onderdeel van de Organisation en Person klasse: bij het opvragen van een Organisation worden ook alle bijbehorende contactinformatie meegegeven.

**Keys**
De primary keys en foreign keys waren in het eerste diagram achterwege gelaten. In dit diagram zijn deze wel toegevoegd. Omdat elke klasse een ‘id’ property als primary key heeft, is de basisklasse ‘Base’ toegevoegd.

**Base klasse**
Het zou overdreven zijn om alleen omdat alle klasses een ‘id’ hebben hiervoor de ‘Base’ klasse te introduceren. De Base klasse heeft echter een ander voordeel. Doordat alle dataklasses de properties en methodes van de Base klasse overerven, kan op eenvoudige wijze worden afgedwongen dat alle dataklasses een bepaalde methode implementeren.

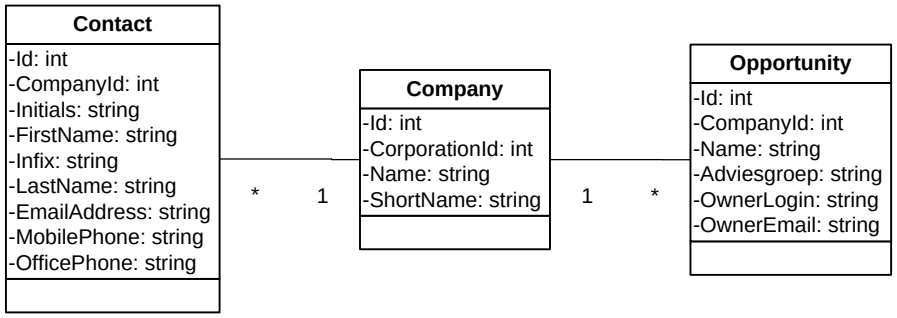
**Afgeleide attributen**
Aan de klasse ‘Party’ zijn de attributen ‘addresses’, ‘emails’ en ‘phones’ toegevoegd. Dit is de implementatie van de link naar ‘Contact’.
Er is gekozen voor 3 afzonderlijke ‘List’s om in de code eenvoudiger gebruik te kunnen maken van deze ‘Contact’s.

Aan de klasse ‘Organisation’ is het attribuut ‘people’ toegevoegd. Dit is de implementatie van de link naar ‘Person’.

## 7.3 Relevante datastructuur Tam Tam

De derde stap was het in kaart brengen van de relevante datastructuur van Tam Tam.

Deze stap is uitgevoerd door te kijken naar de parameters van de orchestraties van de ESB. Op basis van deze parameters is het volgende klassendiagram opgesteld:

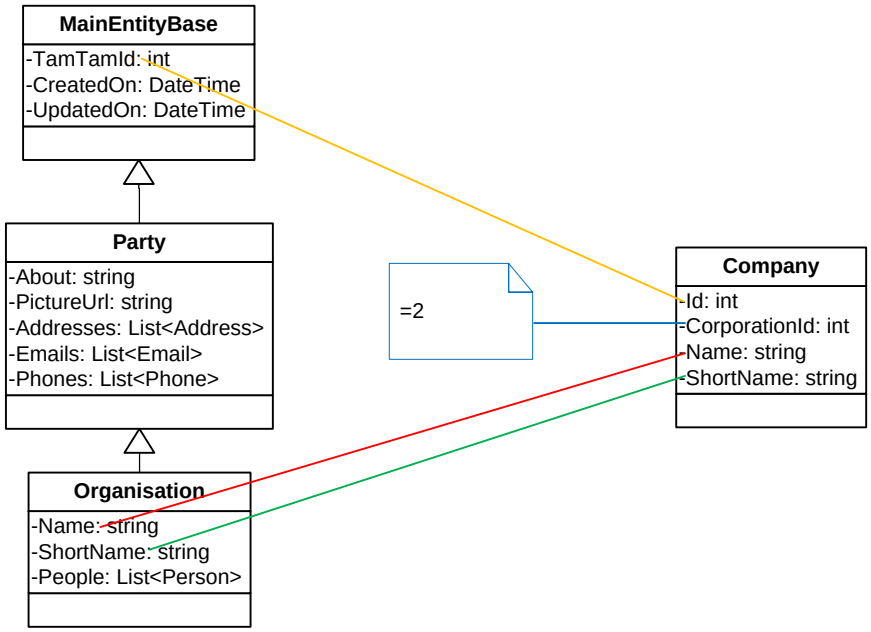


## 7.4 Ontwerpen transformatie

De laatste stap was het ontwerpen van de transformatie van de geïmplementeerde datastructuur van de Capsule API naar de datastructuur van Tam Tam.

### 7.4.1 Organisation -> Company

Het volgende diagram toont de transformatie van Organisation naar Company:




**Id**

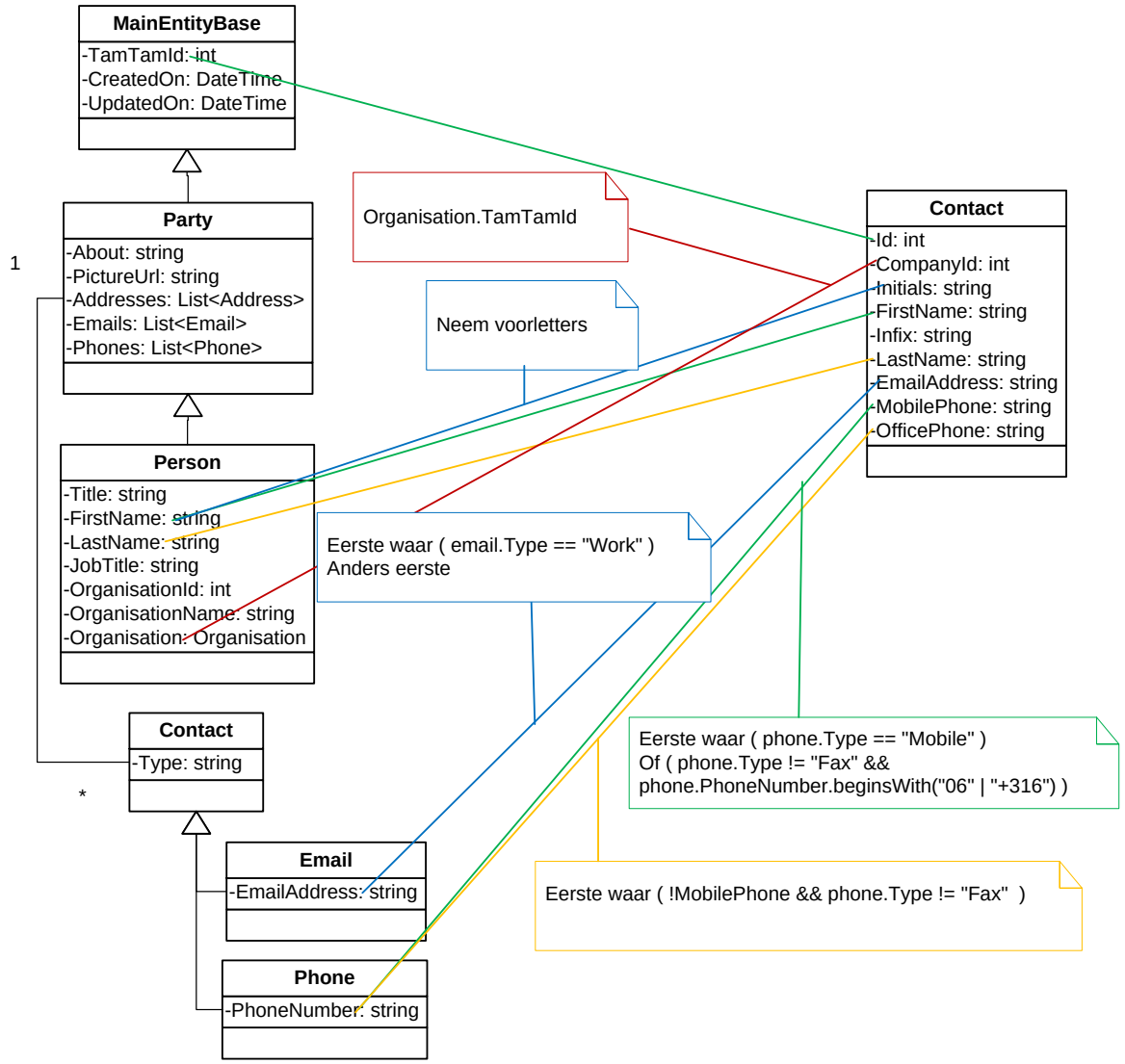
Zoals te zien is wordt het 'Id' property van Company gevuld met het 'TamTamId' van de Organisation. Dit zien we ook terug bij de andere transformaties.

**CorporationId**

Het CorporationId van een Company wordt door de interne systemen gebruikt om de CRM data van Mixit te onderscheiden van de data van TamTam. Het CorporationId van Mixit is '2'. De koppeling zal daarom elke Company een CorporationId van '2' geven.

### 7.4.2 Person -> Contact

Het volgende diagram toont de transformatie van Person naar Contact:




**CompanyId**

Omdat alle 'Id's op de 'TamTamId' properties gebaseerd zijn, moet voor het vullen van de CompanyId van een Contact, ook het TamTamId van de bijbehorende Organisation gebruikt worden.

**Initials**

In de Tam Tam structuur worden ook de voorletters van een Contact bijgehouden. Aangezien deze niet in Capsule worden bijgehouden moest hier een oplossing voor verzonnen worden.
In overleg is besloten om het Initials property te vullen op basis van de beginletter(s) van de voornaam/voornamen van de de Person.

**EmailAddress**

De Capsule structuur staat meerdere e-mailadressen toe terwijl de Tam Tam structuur maar 1 emailadres toestaat.

In overleg is besloten om het emailadres van het type ‘Work’ emailadres de voorkeur te geven. Wanneer dit niet aanwezig is wordt het eerste emailadres gebruikt.

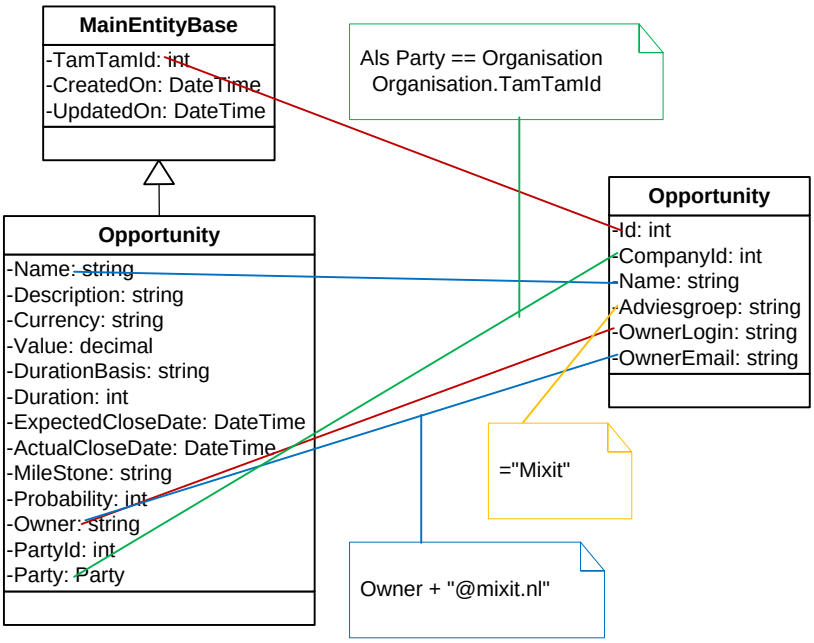
**MobilePhone**
De Tam Tam structuur houdt twee telefoonnummers bij: MobilePhone en OfficePhone. Binnen de Capsule structuur kunnen ook meerdere telefoonnummers worden bijgehouden. Een ‘Phone’ kan van het type ‘Mobile’, ‘Office’ of ‘Fax’ zijn. Het is echter ook mogelijk dat er geen type is opgegeven.

Uit onderzoek naar de data in Capsule bleek dat het type van een Phone vaak ontbreekt. Aangezien er geen manier in Capsule geen manier is om af te dwingen dat een Phone altijd een Type krijgt, is ervoor gekozen om het veld MobilePhone te vullen met nummer van de eerste Phone die:

- Het type ‘Mobile’ heeft.
- Of een Mobiel nummer heeft (beginnend met ’06 of ‘+316’)

**OfficePhone**
Het OfficePhone property wordt op eenzelfde manier gevuld. Het property wordt gevuld met het nummer van de eerste Phone die geen MobilePhone is en niet van het type ‘Fax’ is.

### 7.4.3 Opportunity -> Opportunity
Het volgende diagram toont de transformatie van (Capsule) Opportunity naar (Tam Tam) Opportunity:




**CompanyId**
Binnen de TamTam structuur is aan elke Opportunity aan een Company gekoppeld. Binnen de Capsule structuur is er echter aan elke Opportunity of een Organisation, of een Person gekoppeld. Dit verschil kan problemen opleveren. Wanneer een Opportunity aan een Person gekoppeld is in plaats van een Organisation, kan het Opportunity niet als Tam Tam Opportunity worden aangemaakt.

In de praktijk bleken alle Opportunities aan Organisations gekoppeld te zijn. Er kan echter niet van worden uitgegaan dat dit altijd zo zal zijn.

Bij de transformatie wordt gecontroleerd of het Party van de Opportunity een Organisation is. Alleen als dat het geval is, wordt de TamTamId van het Party gebruikt om het CompanyId property te vullen.

**Adviesgroep**
Binnen de Tam Tam structuur heeft elke Opportunity een 'Adviesgroep' property. Dit property bestond (nog) niet in Capsule CRM. Er waren twee mogelijke oplossingen:

1. Het property in Capule CRM toevoegen als 'custom field'.
2. Altijd 'Mixit' als waarde gebruiken.

In overleg is besloten om de tweede optie toe te passen. Dit is mogelijk omdat Mixit momenteel uit maar 1 adviesgroep bestaat. Wanneer dit in de toekomst veranderd zal de code dus moeten worden aangepast naar de eerste optie.

**OwnerEmail**
Binnen de Tam Tam structuur wordt het e-mailadres van de 'Owner' van een Opportunity bijgehouden.
Dit emailadres is af te leiden uit het 'Owner' property van de Capsule Opportunity. Dit property is namelijk gevuld met het emailadres van de betreffende gebruiker, maar dan zonder de toevoeging '@mixit.nl'. Dit komt doordat voor de usernames in Capsule het emailadressen gebruikt worden.

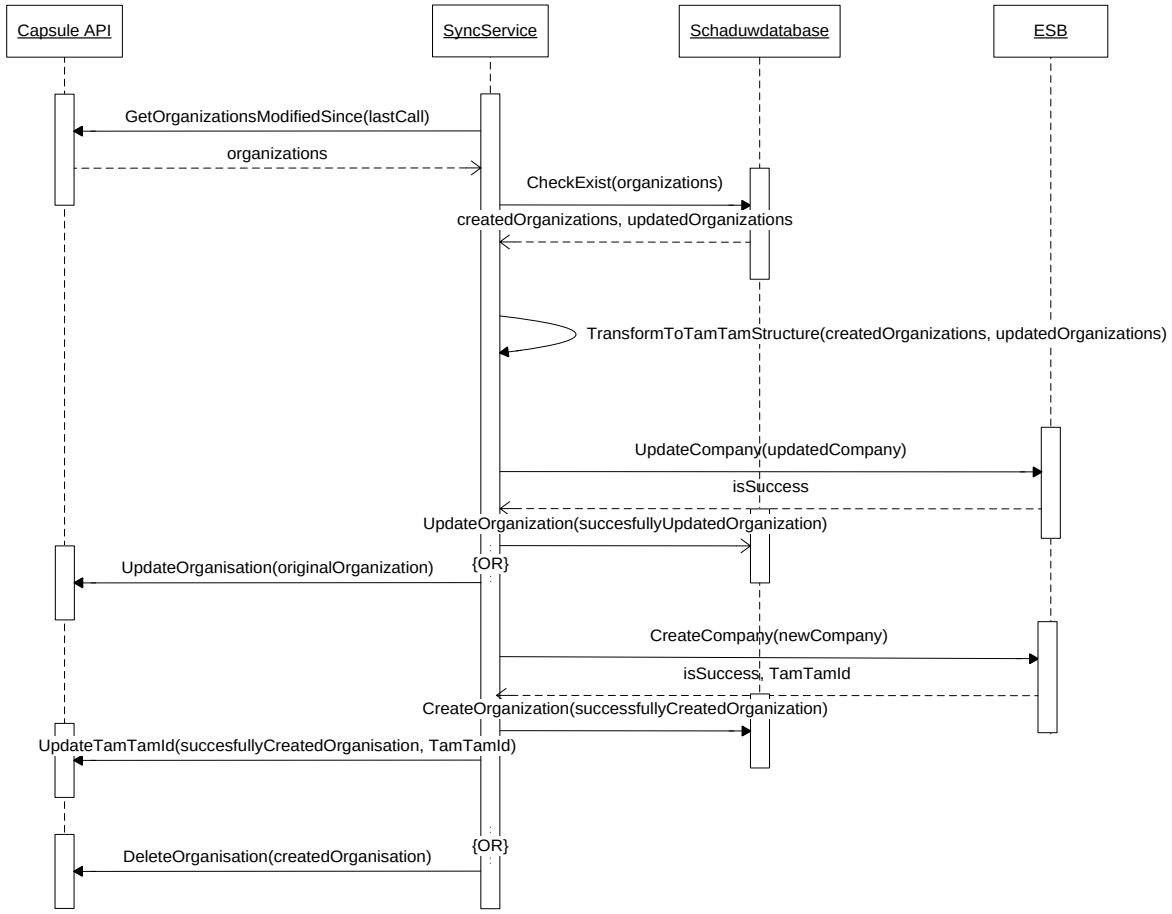
# 8 Concurrency view

Op basis van de functionele view zijn twee sequencediagrammen opgesteld. Deze sequencediagrammen beschrijven het proces voor het verwerken van de events voor de Organisations. Het verwerken van de Persons en Opportunities gebeurt op een zelfde manier.

Deze sequencediagrammen laten zien hoe de koppeling samenwerkt met de verschillende systemen. In 9.3 worden twee concurrency problemen besproken. De problemen worden uitgelegd en er wordt toegelicht hoe in de gemaakte sequencediagrammen de problemen verholpen zijn.

## 8.1 Verwerking create / update events

Het eerste diagram beschrijft het verwerken van create en update events:

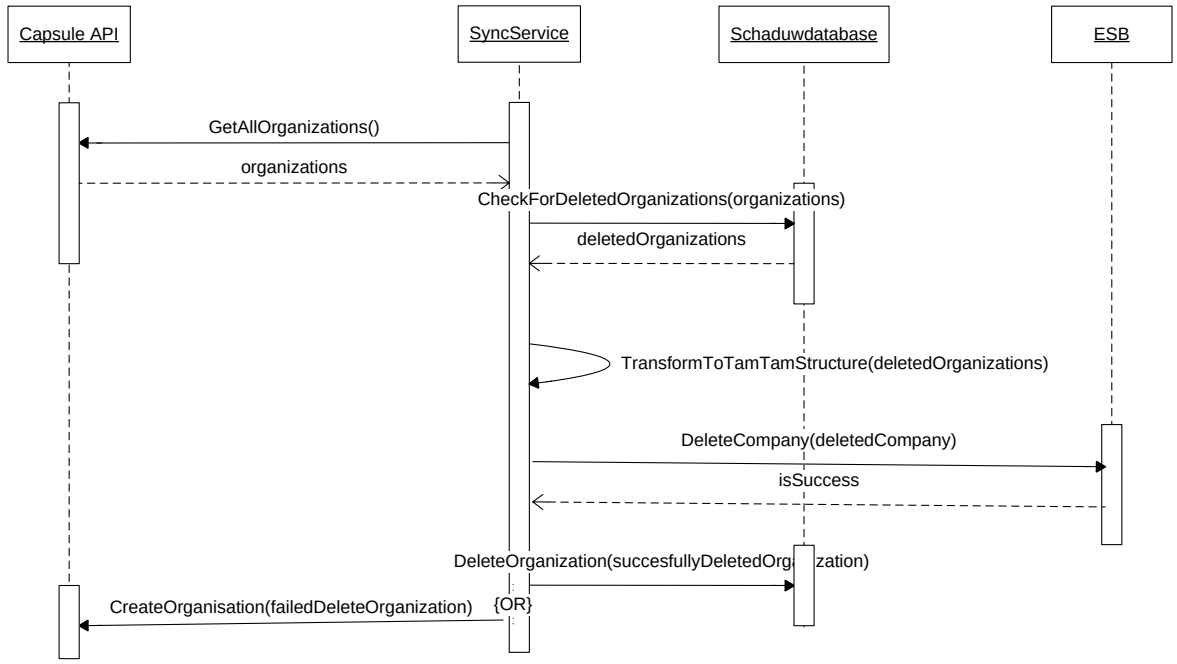



**Tekstuele beschrijving**

1. De SyncService vraagt de Organisations op die sinds het tijdstip van de laatste verwerking zijn aangemaakt / gewijzigd.
2. De SyncService kijkt welke van de opgehaalde Organisations in de schaduwdatabase voorkomen. Op basis hiervan wordt bepaald welke Organisations nieuw zijn en welke gewijzigd.
3. De Organisations worden omgezet naar de Tam Tam datastructuur (Companies).
4. Bij de omzetting van gewijzigde Organisations wordt ook het origineel omgezet. De resulterende Companies worden vergeleken. Alleen wanneer deze niet gelijk zijn is er sprake van een Company update event.
5. Voor elke updatedCompany wordt de UpdateCompany orchestratie van de ESB aangeroepen.
6. Wanneer de orchestratie succesvol is uitgevoerd, wordt de bijbehorende updatedOrganisation opgeslagen in de schaduwdatabase.
   Als er een fout is opgetreden wordt via de Capsule API het origineel van de updatedOrganisation hersteld.
7. Voor elke newCompany wordt de CreateCompany orchestratie van de ESB aangeroepen.
8. Wanneer de orchestratie succesvol is uitgevoerd, wordt de bijbehorende createdOrganisation opgeslagen in de schaduwdatabase. De SyncService zorgt ervoor dat het teruggegeven ‘TamTamId’ in Capsule CRM wordt opgeslagen.
   Als er een fout is opgetreden wordt de Organisation via de Capsule API verwijderd.

## 8.2 Verwerking delete events

Het tweede diagram beschrijft het verwerken van delete events:




**Tekstuele beschrijving**

1. De SyncService haalt alle Organisations op via de Capsule API.
2. De SyncService kijkt of de schaduwdatabase Organisations bevat die niet voorkomen in de opgehaalde Organisations. Dit zijn de verwijderde Organisations.
3. De Organisations worden omgezet naar de Tam Tam datastructuur (Companies).
4. Voor elke deletedCompany wordt de DeleteCompany orchestratie van de ESB aangeroepen.
5. Wanneer de orchestratie succesvol is uitgevoerd, wordt de bijbehorende deletedOrganisation verwijderd uit de schaduwdatabase.
   Als er een fout is opgetreden wordt de deletedOrganisation via de Capsule API opnieuw aangemaakt.

## 8.3 Voorkoming concurrency problemen

**Hoe worden problemen met grensgevallen voorkomen?**

Via de API kunnen nieuw aangemaakte en gewijzigde Organisations worden opgehaald. Hierbij wordt de tijd/datum meegegeven van sinds wanneer de Organisations aangemaakt of gewijzigd zijn.
Op deze manier kan er, bijvoorbeeld elke minuut, gecontroleerd worden of er nieuwe en / of gewijzigde Organisations zijn. Wanneer echter als tijd exact een minuut eerder wordt opgegeven zouden er problemen met grensgevallen kunnen ontstaan.

Voorbeeld:

1. De koppeling vraagt om 13:00:00 de wijzigingen van de afgelopen minuut op en verwerkt deze.
2. Om 13:00:01 wordt er een nieuwe Organisation aangemaakt.
3. De koppeling vraagt om 13:01:02 de wijzigingen van de afgelopen minuut op.

In het voorbeeld is te zien dat zodra het opvragen een (afgeronde) seconde te laat zou gebeuren, het mogelijk wordt dat events niet verwerkt worden.

De oplossing voor dit probleem is een tijd/datum gebruiken die ruim eerder (bijvoorbeeld 10 seconden) is dan de tijd/datum van de laatste controle. Hierdoor worden er echter mogelijk wel Organisations teruggegeven waarvan het event al verwerkt is. Vanwege de werking van het herkenningsproces van de events vormt dit echter geen probleem:

- De Organisations worden gezien als updatedOrganisations (de nieuw aangemaakte Organisations zijn immers al opgeslagen in de schaduwdatabase).
- Bij het omzetten naar Companies blijkt dat de Company niet veranderd is ten opzichte van de huidige versie. Hierdoor worden deze Companies niet meegenomen bij het uitvoeren van de UpdateCompany orchestratie.

**Hoe wordt voorkomen dat de rollback van een event als nieuw event wordt gezien?**

Bij het uitvoeren van de rollback wordt de wijziging van een event ongedaan gemaakt via de API. De API heeft echter niet de mogelijkheid om aan te geven dat het om een rollback gaat. Vanuit Capsule gezien is de rollback dus een gewoon nieuw event.
Er moet worden voorkomen dat de rollback ook nieuwe aanroepen van de ESB orchestraties veroorzaakt.

De oplossing is dat de wijzigingen van events alleen in de schaduwdatabase worden verwerkt, wanneer er geen rollback is. Hierdoor heeft een event vanuit het persperctief van de schaduwdatabase alleen plaatsgevonden wanneer de orchestratie succesvol was.
Wel zorgt de rollback ervoor dat de Organisation waarop de rollback is gedaan wordt meegegeven bij de volgende keer dat nieuwe / gewijzigde Organisations worden opgehaald. Vanwege de werking van het herkenningsproces van de events vormt dit echter geen probleem:

- De Organisations worden gezien als updatedOrganisations (de nieuw aangemaakte Organisations zijn immers al opgeslagen in de schaduwdatabase).
- Bij het omzetten naar Companies blijkt dat de Company niet veranderd is ten opzichte van de huidige versie. Hierdoor worden deze Companies niet meegenomen bij het uitvoeren van de UpdateCompany orchestratie.

# 9 Development view

In dit hoofdstuk wordt het ontwerp van de implementatie van de Capsule API en het ontwerp van de implementatie van de schaduwdatabase besproken.
Verschillende delen zijn overgenomen uit de architectual description.

## 9.1 Onderzoek werking Capsule API

Om een API te kunnen gebruiken moet deze API worden geïmplementeerd. Dit houdt in dat er code geschreven wordt die de functies van de API beschikbaar maakt.
Vaak zijn zulke implementaties als library beschikbaar bij de eigenaar van de API. Bij Capsule bleken er echter alleen implementaties voor de talen PHP en Ruby beschikbaar te zijn, een C# library ontbrak. Hierdoor moest ik zelf een implementatie van de API ontwerpen en implementeren.

In het kader van dit ontwerp is er eerst onderzoek gedaan naar de werking van de API. De bevindingen van dit onderzoek zijn ook opgenomen in de architectual description.

**RESTful HTTP API**

De Capsule API is een zogenaamde RESTful HTTP API. Dit houdt in dat de Capsule API benaderd wordt via HTTP requests.
Er kan hierbij gebruik gemaakt worden van de vier HTTP methodes 'GET', 'POST','PUT', en 'DELETE'. De actie die uitgevoerd dient te worden wordt gespecificeerd door deze methode. Respectievelijk kan hierdoor data opgehaald, aangemaakt, gewijzigd en verwijderd worden.
Om aan te geven welke data opgehaald / gemanipuleerd moet worden wordt gebruik gemaakt van de URL.
De URL die hiervoor gebruikt dient te worden kan worden opgedeeld in vier verschillende onderdelen:

1. De basis url
2. Het datatype van de op te halen / te manipuleren data
3. Het id van de op te halen / te manipuleren data (optioneel)
4. Parameters die de selectie van de data beperken (optioneel)

De basis URL is de 'locatie' van da API. In het geval van MixIt is dit:
https://mixit.capsulecrm.com/api/

Het datatype geeft aan wat voor soort data we willen ophalen / manipuleren. Als we alle organisaties willen ophalen wordt de URL:
https://mixit.capsulecrm.com/api/organisation

Wanneer we een specifieke organisatie willen ophalen of wanneer we een organisatie willen wijzigen of verwijderen moet het id van de organisatie toegevoegd worden. De URL wordt dan bijvoorbeeld:
https://mixit.capsulecrm.com/api/organisation/1234

Het is mogelijk om de selectie van de data te beperken door het toevoegen van parameters. Deze parameters worden in de querystring gezet. Wanneer we alle organisaties willen ophalen die aangemaakt of gewijzigd zijn sinds 1 november wordt de URL:
https://mixit.capsulecrm.com/api//api/organisation?lastmodified=20111101T000000

**XML API**

De Capsule API maakt gebruik van XML om data weer te geven en te ontvangen. De structuur van deze data wordt omschreven in de Information View.

## 9.2 Ontwerp implementatie Capsule API

**HTTP requests**

.NET biedt standaard componenten voor het uitvoeren van HTTP requests. Met behulp van deze componenten kunnen de requests aan de Capsule API uitgevoerd worden.

**XML data**

Ook voor het werken met XML data biedt .NET standaard componenten. Door gebruik te maken van deze componenten kan de XML data worden ingelezen in C# classes en visa versa. De classes dienen geïmplementeerd te worden volgens het ontwerp in de Information View.
Elke klasse krijgt een constructor die op basis van een XmlNode het object kan aanmaken.

## 9.3 Onderzoek Entity Framework

Voor de benadering van de database is gekozen voor het ORM (object-relational mapping) framework 'Entity Framework'.

In het kader van de implementatie van de schaduwdatabase is er onderzoek gedaan naar de drie verschillende mogelijke strategieën voor het gebruik van het Entity Framework:

1. Database First
2. Model First
3. Code First

**Database First**

Bij de Database First benadering worden op basis van een database bijbehorende C# classes aangemaakt.

**Model First**

Bij de Model First benadering wordt in Visual Studio het datamodel ontworpen. Op basis van dit datamodel worden vervolgens zowel de C# modellen als de databasetabellen aangemaakt.

**Code First**

Deze optie is nieuw in het Entity Framework. Bij deze benadering wordt op basis van bestaande C# classes een database aangemaakt.

## 9.4 Ontwerp implementatie schaduwdatabase

Er is voor gekozen om de schaduwdatabase te implementeren met de Code First aanpak.

Het doel van de schaduwdatabase is de data uit Capsule CRM opslaan. De structuur van de schaduwdatabase is hierdoor afhankelijk van de Capsule API.
Wanneer er gekozen zou worden voor een Database First of Model First aanpak, zou de implementatie van de Capsule API afhankelijk worden van de implementatie van de schaduwdatabase. Dit zou ontwerptechnisch een verkeerde keuze zijn, omdat er een onnodige wederzijdse afhankelijkheid zou ontstaan:

- De implementatie van de Capsule API is afhankelijk van de schaduwdatabase omdat de “Models” (data klassen) hierop gebaseerd zijn.
- De implementatie van de schaduwdatabase is afhankelijk van de implementatie van de Capsule API omdat hij gevuld wordt met die via de API verkregen wordt.

Deze wederzijdse afhankelijkheid zou het moeilijker maken om deze twee componenten te implementeren en te onderhouden. Door voor de Code First aanpak te kiezen vervalt deze afhankelijkheid:

- Voor de implementatie van de Capsule API worden ‘met de hand’ data classes geschreven.
- De implementatie van de schaduwdatabase is afhankelijk van de implementatie van de Capsule API. De schaduwdatabase wordt automatisch gegenereerd op basis van de geschreven data classes.

# 10 Structuur implementatie

De volgende hoofdstukken bespreken de implementatie van de koppeling.
In dit hoofdstuk wordt de structuur van de code besproken.

## 10.1 Overzicht structuur

Zoals in het plan van aanpak besproken is zou de koppeling met C#.NET gemaakt worden. Het programma Visual Studio wordt hierbij als IDE gebruikt.

**Solution en projects**

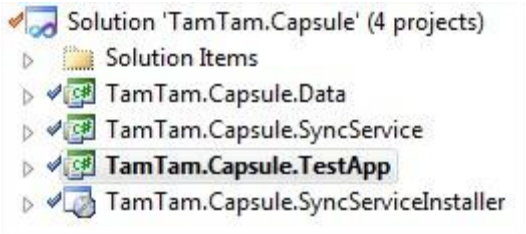
Binnen Visual Studio wordt gewerkt met een 'solution' die wordt onderverdeeld in 'projects'.

Een project is een verzameling van (code) bestanden die samen 1 resultaat produceren. Dit resultaat kan verschillende vormen hebben. Het kan bijvoorbeeld een applicatie zijn ('.exe'-bestand) of een code library ('.dll'-bestand).

Een solution groepeert verschillende projects die met elkaar te maken hebben. Een solution kan dus verschillende applicaties bevatten.
Als deze applicaties een gemeenschappelijke basis hebben wordt de code van die basis doorgaans ondergebracht in 1 of meerdere code library projects.

**Solution van de koppeling**

De volgende afbeelding is een screenshot van de solution van de koppeling.




Als naam voor de solution is gekozen voor 'TamTam.Capsule'. De naam van een solution is tevens de 'namespace' voor de code binnen de solution.
De naam geeft aan dat het gaat om code van Tam Tam, die betrekking heeft op Capsule.

De koppeling is opgedeeld in 4 verschillende projects. In de volgende paragrafen wordt kort besproken waar de verschillende projects voor dienen en waarom voor deze verdeling is gekozen.

## 10.2 TamTam.Capsule.Data

Het TamTam.Capsule.Data project is een code library project. In dit project is de zogenaamde datalaag van de koppeling ondergebracht. Het project bestaat uit de volgende elementen:

1. De dataklasses van zowel de Capsule datastructuur als de Tam Tam datastructuur.
2. De implementatie van de communicatie met de Capsule API.
3. De implementatie van de schaduwdatabase en de communicatie met de schaduwdatabase.

Punt 1 en 2 worden in hoofdstuk 12 besproken en punt 3 wordt in hoofdstuk 13 besproken.

**Waarom een code library?**

Zoals hierboven al besproken wordt de gemeenschappelijke basis van projects binnen een solution vaak ondergebracht in code library projects.

De te maken koppeling zou echter maar uit 1 applicatie bestaan. Er is echter toch gekozen om de genoemde elementen onder te brengen in een code library project.

De reden voor deze keuze is dat de datalaag wel gebruikt worden door de koppeling, maar ook gebruikt zouden kunnen worden door een andere applicatie.
Door de datalaag in een apart project onder te brengen ontstaat er meer cohesie op projectniveau. Ook wordt er hierdoor rekening gehouden met mogelijke toekomstige applicaties die ook van de datalaag gebruik willen maken.

## 10.3 TamTam.Capsule.SyncService

Het TamTam.Capsule.SyncService project is de kern van de koppeling.

**Taken SyncService**
De SyncService is verantwoordelijk voor het bepalen van de wijzigingen (add / edit / delete) in de data in Capsule CRM. De SyncService moet deze wijzigingen doorvoeren in de schaduwdatabase en de bijbehorende orchestraties van de ESB aanroepen.
Dit proces is beschreven 8.1 en 8.2.

**Windows service**
De SyncService is een Windows service. Een Windows service is een applicatie of proces wat op de achtergrond draait. In tegenstelling tot een gewone applicatie heeft een Windows service heeft dus geen GUI.
Voor het uitvoeren van periodieke taken wordt vaak zo'n Windows service gebruikt. Een andere mogelijkheid is een console-applicatie.

Op basis van onderzoek op internet bleek dat het een Windows service beter geschikt is dan een console applicatie. Een Windows service zou beter in staat zijn om langdurig periodiek een taak uit te voeren dan een console applicatie. Ook is een Windows service beter veilig op te starten en af te sluiten.
Om deze redenen is ervoor gekozen de koppeling een Windows service te maken.

De implementatie van de SyncService wordt besproken in hoofdstuk 15.

## 10.4 Tamtam.Capsule.SyncServiceInstaller

Zoals al gezegd is de SyncService een Windows-service-applicatie. In tegenstelling tot bijvoorbeeld een console-applicatie of een Windows-forms-applicatie kan een Windows-service-applicatie niet direct vanuit Visual Studio worden opgestart om de applicatie te testen.

**Installer**
Een Windows-service-applicatie moet eerst geïnstalleerd worden. Voor dit installeren is een installer nodig.
Deze installer kan worden gemaakt door met Visual Studio een installatieproject toe te voegen aan de solution. Vervolgens wordt de output van een Windows-service-applicatie project als inhoud voor de installer geselecteerd.
Wanneer dit gedaan is kan de installer vanuit Visual Studio gecreëerd en uitgevoerd worden.

## 10.5 TamTam.Capsule.TestApp

**Debuggen**

Bij het schrijven van de code voor een applicatie is het gebruikelijk om nieuwe code te debuggen. De nieuwe code wordt dan in debug-modus uitgevoerd. Met behulp van de debugger kan de developer zien of de code het goed doet en wat er fout gaat.
De fouten die hij op deze manier tegenkomt worden direct verbeterd. Hij blijft dit proces herhalen totdat de code werkt.
Vaak zijn er meerdere van deze iteraties nodig voordat de code in orde is. Het is daarom van belang dat de code eenvoudig uit te voeren en te debuggen is.

**Testapplicatie**

Omdat een Windows-service-applicatie geen GUI heeft en niet met '1 druk op de knop' te debuggen is, is ervoor gekozen om een aparte testapplicatie te maken.
Deze applicatie is een eenvoudige Windows-forms-applicatie. Op het form worden verschillende knoppen geplaatst waarmee bepaalde functies gekoppeld worden. Dit kunnen zowel functies uit de datalaag als uit de SyncService zijn.
Op deze manier kunnen deze functies eenvoudig gedebugged worden.

## 10.6 Vergaderingen

Tot zover hadden er geen wekelijkse vergaderingen plaatsgevonden. De vergaderingen met de opdrachtgever waren veelal ad-hoc. In deze vergaderingen werden wel de technische aspecten besproken, maar ik betrok de opdrachtgever niet echt bij het verdere proces en de planning.

**Developer meetings**

Tijdens de implementatiefase hebben er wekelijkse vergaderingen plaatsgevonden met Mike Noordermeer en Marco Krikke. Als developers van de interne systemen van Tam Tam waren zij goed in staat om feedback te geven op mijn voortgang.

**Aanpassingen webservice**

In de architectual description werd besproken dat de koppeling zou communiceren met de ESB. Voor elke wijziging zou er een methode van de ESB moeten worden aangeroepen, een zogenaamde orchestratie.

Op basis van zo'n aanroep moeten de juiste acties in de interne systemen van Tam Tam worden uitgevoerd. De orchestratie regelt dit door een methode van een interne webservice aan te roepen. Deze webservice is vervolgens verantwoordelijk voor het uitvoeren van de juiste acties.

In de eerste bespreking met de developers kwam de architectual description aan de orde. Ze hadden geen aanmerkingen op mijn gemaakte ontwerp. Wel bleek dat de genoemde webservice zou moeten worden aangepast om mijn ontwerp mogelijk te maken. Dit had vooral te maken met het 'tamtamId' probleem (zie 7.1).

Er werd besloten dat Mike op een later tijdstip, zodra hij hier de tijd voor zou hebben, de benodigde veranderingen zou uitvoeren. Aangezien het daadwerkelijke aanroepen van de orchestraties binnen het laatste onderdeel van de implementatie gedaan zou worden en dit technisch niet zo spannend is, voorzag ik hier geen problemen mee.

## 10.7 Planning en sprints

Voor de implementatiefase heb ik geen gedetailleerde backlog gemaakt. De sprintplanning was vrij globaal. De gedachte hierachter was dat deze vrij globale sprintplanning specifiek genoeg was om uitvoerbaar te zijn.
Deze gedachte bleek te kloppen. Het volgen van de planning leverde geen problemen op.

**Voortgang**

Het was inmiddels week 9. Ik liep hiermee dus een week achter op de project planning. Wat erger was, was dat ik ook nog niet met het afstudeerverslag was begonnen. Omdat ik achter liep op de planning richtte ik me echter volledig op de inhoudelijke kant van het project, met de gedachte dat het verslag later wel zou komen.

**Sprintplanning**

| Sprint | Beschrijving |
|---|---|
| 1 | Capsule API. Tijdens deze sprint zullen de benodigde methodes van de Capsule API geïmplementeerd worden. Hiervoor is het ook nodig dat de Capsule dataklasses (zie 7.2) geïmplementeerd worden. |
| 2 | Capsule Context. Tijdens deze sprint zal de schaduwdatabase en de communicatie met deze database geïmplementeerd worden. |
| 3 en 4 | SyncService en SyncServiceInstaller. Tijdens deze sprints zal de SyncService en zijn installer geïmplementeerd worden. Ook zullen de hiervoor benodigde Tam Tam dataklasses (zie 7.3) geïmplementeerd worden. |

# 11 Implementatie Capsule API

In dit hoofdstuk wordt de implementatie van de Capsule API beschreven. Alleen die elementen van de API die nodig zijn voor het functioneren van de SyncService zijn geïmplementeerd.

## 11.1 Overzicht implementatie

De implementatie van de Capsule API moest de volgende twee taken kunnen uitvoeren:

1. Het ophalen van data en deze data inlezen in dataklasses.
2. Het uitvoeren van datamutaties (toevoegen / wijzigen / verwijderen).

De implementatie van het eerste punt wordt beschreven in 12.2 t/m 12.4 en het tweede punt in 12.5 en 12.6.

## 11.2 Ophalen data

De eerste stap voor de implementatie van de Capsule API was code schrijven voor het uitvoeren van datarequests aan de Capsule API.

Uit de sequencediagrammen in 8.1 en 8.2 volgt dat er voor elke entiteit waarvan de events verwerkt worden twee data request methodes moeten zijn. Een voor het ophalen van alle data en een voor het ophalen van de data die sinds een bepaald tijdstip toegevoegd / gewijzigd is.

De Capsule API geeft XML data terug. Voordat deze data gebruikt kan worden moet deze worden omgezet naar dataklasses.

## 11.3 Implementatie Capsule dataklasses

Voor het gebruik van de data die opgehaald wordt via de hierboven beschreven methodes moet de data worden omgezet naar dataklasses. Hiervoor was in de architectual description al een klassendiagram gemaakt (zie 7.2).

Op basis van dit klassendiagram zijn de klasses geïmplementeerd.

## 11.4 Conversie naar klasses

Nu de dataklasses geschreven waren moest de code voor het omzetten van de XML geschreven worden. Hier volgt een voorbeeld van de XML beschrijving van een Person:

```
<person>
  <id>13413797</id>
  <contacts>
    <email>
      <id>24697697</id>
      <emailAddress>naam@voorbeeld.nl</emailAddress>
    </email>
    <phone>
      <id>24697698</id>
      <phoneNumber>123-5667285</phoneNumber>
    </phone>
    <website>
      <id>24697699</id>
      <type>Work</type>
      <webAddress>http://www.voorbeeld.nl</webAddress>
      <webService>URL</webService>
      <url>http://www.voorbeeld.nl</url>
    </website>
  </contacts>

<pictureURL>https://d365sd3k9yw37.cloudfront.net/a/1326731384/theme/default/images/per
son_avatar_70.png</pictureURL>
  <createdOn>2011-08-01T15:13:34Z</createdOn>
  <updatedOn>2011-08-01T15:18:36Z</updatedOn>
  <firstName>Jan</firstName>
  <lastName>Pieterse</lastName>
  <organisationId>13413796</organisationId>
  <organisationName>Voorbeeld Organisatie</organisationName>
</person>
```

De implementatie is gedaan door aan alle klasses een constructor toe te voegen met als parameter een XmlNode. Voor het ophalen van de waardes uit deze XmlNode is een ‘helper’ klasse gemaakt. Met deze klasse kunnen waardes uit de XML data worden uitgelezen. Deze waardes worden vervolgens in het bijbehorende property gestopt.

Bij de Person klasse ziet dit er als volgt uit:

```
public Person(XmlNode node)
    : base(node)
{
    Title = XmlHelper.GetString(node, "title");
    FirstName = XmlHelper.GetString(node, "firstName");
    LastName = XmlHelper.GetString(node, "lastName");
    JobTitle = XmlHelper.GetString(node, "jobTitle");
    OrganisationId = XmlHelper.GetNullableInt(node, "organisationId");
    OrganisationName = XmlHelper.GetString(node, "organisationName");
}
```

Zoals hierboven te zien is wordt ook de constructor van de basisklasse (in dit geval ‘Party’) aangeroepen. In deze constructor worden de properties van de basisklasse uitgelezen. Door voor deze constructie te kiezen wordt voorkomen dat deze code in elke subklasse voorkomt.

## 11.5 Datamutaties

Voor de rollback functionaliteit is het noodzakelijk dat er data via de API kan worden toegevoegd, gewijzigd en verwijderd.

Bij de requests voor het toevoegen en wijzigen van data moet de data (bijvoorbeeld een Person) worden meegegeven in XML-formaat. Daarom moest er ook code geschreven worden voor de conversie van de dataklasses naar XML.

## 11.6 Conversie naar XML

Bij de conversie van XML naar de dataklasses zagen we dat er gebruik werd gemaakt van een 'XmlNode'. Voor de omzetting naar XML zou je daarom verwachten dat het hier ook handig is om gebruik te maken van de XmlNode klasse.

Na wat zoeken op internet bleek echter dat hier beter een nieuwe klasse voor gebruikt kon worden: de 'XElement' klasse. De reden hiervoor is dat het genereren van een XmlNode erg bewerkelijk is. De XElement klasse is echter wel eenvoudig te gebruiken.
Als voorbeeld nemen we weer de Person klasse:

```
public override XElement ToXElement()
{
    XElement element = this.GetPartyXElement("person");
    element.Add(
        new XElement("title", Title),
        new XElement("firstName", FirstName),
        new XElement("lastName", LastName),
        new XElement("jobTitle", JobTitle),
        new XElement("organisationId", OrganisationId)
    );

    return element;
}
```

Zoals te zien is wordt ook hier weer gebruik gemaakt van de overervingstructuur. De 'GetPartyXElement'-methode wordt aangeroepen om de properties uit Party basisklasse om te zetten. Vervolgens worden de Person properties toegevoegd.

Aan elke dataklasse is een 'ToXElement' methode toegevoegd. Vervolgens is de volgende methode aan de 'Base' klasse toegevoegd om van elke klasse ook een XML-string op te kunnen halen in plaats van een XElement:

```
public string ToXmlString()
{
    return this.ToXElement().ToString();
}
```

# 12 Implementatie schaduwdatabase

In dit hoofdstuk wordt de implementatie van de schaduwdatabase en de communicatie met deze database beschreven.

## 12.1 Opzet

Zoals in 10.3 al te lezen was, heeft Carlos Sardo tijdens de ontwerpfase geholpen met het kiezen van de aanpak voor het implementeren van de communicatielaag met de database.
In 10.4 wordt de keuze voor de Code-First aanpak van het Entity Framework beschreven. Een van de voordelen van deze aanpak was dat de database automatisch op basis van de dataklasses gegenereerd zou worden.

Omdat ik zelf nog geen ervaring had met het opzetten van een communicatielaag op basis van het Entity Framework riep ik opnieuw de hulp van Carlos in. Hij wees me op enkele tutorials waarin dit wordt uitgelegd.

Vervolgens heeft hij me geholpen bij de opzet. Dit resulteerde in de volgende code:

```
public class CapsuleContext : DbContext
{
    public DbSet<Person> People { get; set; }
    public DbSet<Organisation> Organisations { get; set; }
    public DbSet<Address> Addresses { get; set; }
    public DbSet<Email> Emails { get; set; }
    public DbSet<Phone> Phones { get; set; }
    public DbSet<Website> Websites { get; set; }
    public DbSet<Opportunity> Opportunities { get; set; }

    …

}
```

In bovengenoemde code definiëren we de klasse ‘CapsuleContext’ die de ‘DbContext’-klasse van Entity Framework overerft.
Verder wordt er voor elke klasse waarvan we data in de database willen kunnen opslaan een property van het type ‘DbSet’ gedefinieerd.

Naast deze code moest de locatie van de database nog worden aangegeven in een configuratiebestand.

Meer was er niet nodig. Bij het builden van de solution werd nu automatisch de database gegenereerd op basis van de Capsule dataklasses.

Zie voor de het volgende hoofdstuk voor een analyse van de gegeneerde database.

## 12.2 Data toevoegen

In de vorige paragraaf zagen we dat er voor de opzet van het Entity Framework maar enige regels code nodig waren. De genoemde code zorgt er echter niet alleen voor dat de database gegenereerd wordt, maar is ook de basis voor de communicatie met de database.

**Code voor toevoegen**

Wanneer we bijvoorbeeld een Person willen opslaan in de database kan dit met de volgende code:

```
CapsuleContext context = new CapsuleContext();
context.People.Add(person);
context.SaveChanges();
```

Zoals te zien is vormt de CapsuleContext hier de basis. Met behulp van de context kan data opgevraagd, toegevoegd, gewijzigd en verwijderd worden.

**Testen van toevoegen**
Om te controleren of het opslaan van objecten in de database nu ook echt werkte, moest dit getest worden.

Er waren twee mogelijke manieren om dit te doen:

1. Zelf wat testdata verzinnen.
2. De bestaande data uit Capsule gebruiken.

Ik heb voor de tweede optie gekozen. Doordat de CapsuleAPI al geïmplementeerd was, was het minder werk om de data uit Capsule te gebruiken voor het testen. Een ander voordeel is dat je op deze manier direct de echte situatie test.

De test is uitgevoerd door een methode te maken die eerst alle data ophaalt doormiddel van de CapsuleAPI. Vervolgens wordt de CapsuleContext gebruikt om de opgehaalde data op te slaan in de database.

Het uitvoeren van de methode verliep zonder problemen, er trad geen exception op. Uit controle bleek dat de data inderdaad in de database was opgeslagen.
Er bleek alleen een probleem te zijn: de waardes van de id kolommen klopten niet.

**Probleem met ids**
De ids bleken gevuld te worden door middel van auto-increment. Hierbij werden de ingevulde waardes genegeerd en begon bij elke tabel de nummering bij 1.

Na wat onderzoek op internet bleek dit een veelvoorkomend probleem te zijn. Wanneer je niets aangeeft gaat het Entity Framework ervan uit dat een numeriek ‘id’ property een auto-increment kolom moet worden.
De oplossing was om met een ‘Data Annotation attribuut’ aan te geven dat je de ids zelf vult (zie 14.4 voor meer informatie over Data Annotations).

## 12.3 Data opvragen

Het opvragen van data gebeurt met behulp van LINQ.

Met LINQ kunnen queries geschreven worden in C#. Het is dus niet langer nodig om met C# code SQL queries samen te stellen. Doordat LINQ veel op SQL lijkt is het erg snel te leren.

**Voorbeeld**
Een voorbeeld van zo'n LINQ query zien we in de volgende methode:

```
public List<Person> GetDeletedPeople(List<Person> allPeople)
{
    List<int> allPersonIds = GetIds(allPeople.ToList<Base>());

    return (
        from p in this.People
        where !allPersonIds.Contains(p.Id)
        select p
    ).ToList();
}
```

Deze methode moet een lijst van People teruggeven die verwijderd zijn.
Dit gebeurt door de lijst met alle personen, die uit Capsule is opgehaald, te vergelijken met de personen in de database. Wanneer een persoon niet meer in Capsule voorkomt, maar nog wel in de database staat, is hij blijkbaar verwijderd (zie ook 9.2).

In de methode zien we dat er op basis van de Capsule lijst met personen eerst een lijst van de id's van die personen wordt gemaakt.
Vervolgens wordt er met behulp van een LINQ query die deze lijst gebruikt de verwijderde personen teruggegeven. Dit gebeurt door alle personen uit de database te selecteren waarvan het id niet voorkomt in de lijst.

## 12.4 Data wijzigen / verwijderen

De code voor het wijzigen en verwijderen van data is pas tijdens de implementatie van de SyncService (hoofdstuk 15) gemaakt en getest. Omdat het toch onder de implementatie van de schaduwdatabase valt, wordt het hier besproken.

Voor het wijzigen of verwijderen van data kan, net als bij het toevoegen, gewoon methode van de context gebruikt worden.

**Probleem met Contact klasses**
Tijdens het testen van het wijzigen en verwijderen bleek echter dat de informatie uit de Contact klasses (Phone, Email, etc) voor problemen zorgde.

Bij het testen van het wijzigen van People / Organisations viel op dat wijzigingen aan informatie van een van de Contact klasses niet verwerkt werd.
Bij het testen van het verwijderen People / Organisations bleek dat dit helemaal niet werkte. Er trad een exception op die vermelde dat de Person niet verwijderd mocht worden, wegens een 'foreign key constraint'.

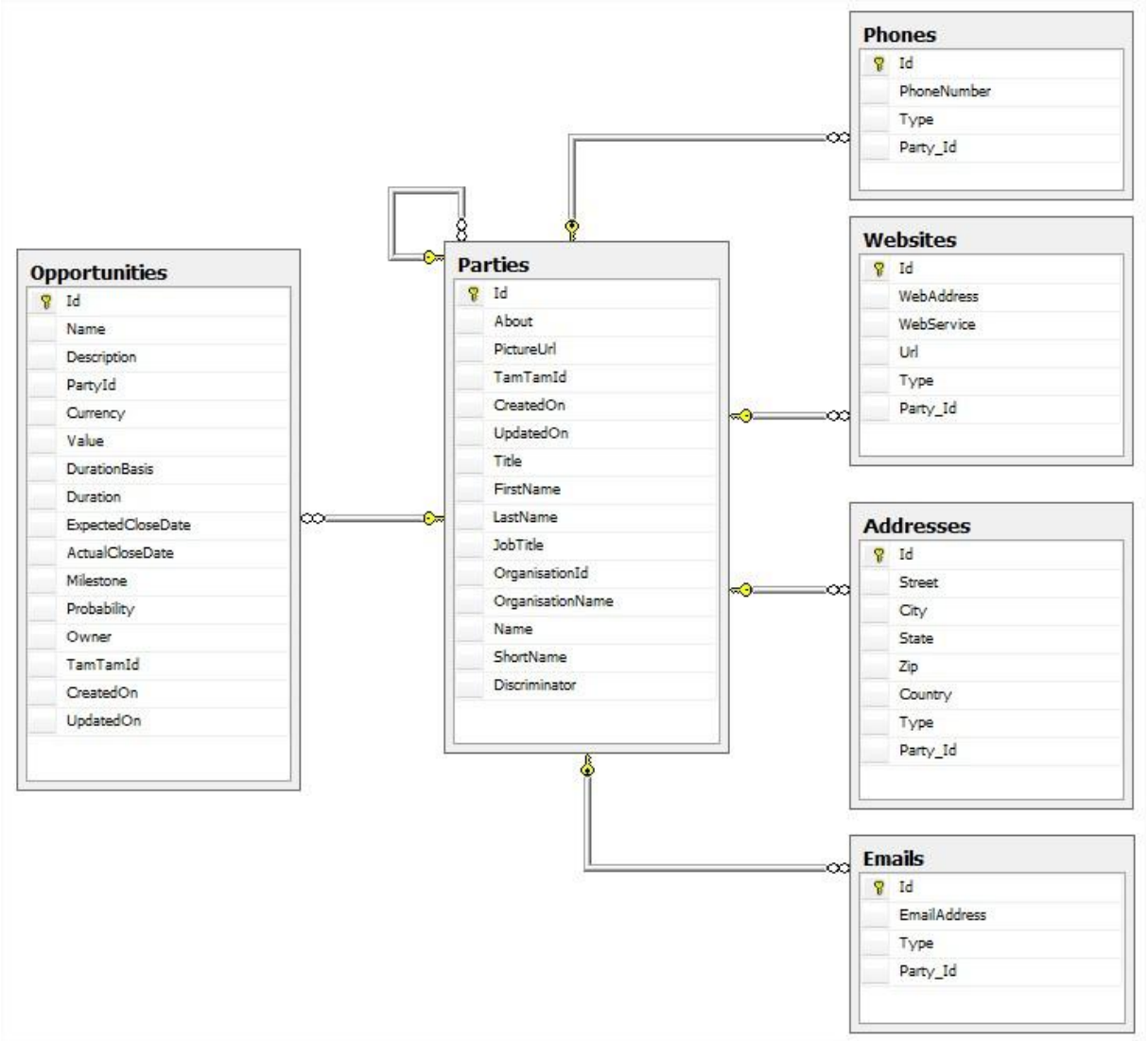
Uit navraag bij Carlos bleek dat dit een bekend probleem is. Op het moment dat een object lijsten met andere objecten bevat, moet bij wijzigingen 'handmatig' worden aangegeven wat de veranderingen in zo'n lijst zijn. Dit houdt in dat doormiddel van code moet worden aangegeven welke items uit de lijst toegevoegd, gewijzigd en verwijderd zijn.
Nadat ik een functie had gemaakt die dit deed, was het probleem inderdaad opgelost.

# 13 Analyse gegenereerde database

In dit hoofdstuk wordt de door het Entity Framework gegenereerde database geanalyseerd.

## 13.1 Structuur

De gegenereerde database had de volgende structuur:




Als we het bovenstaande model vergelijken met het klassediagram uit 8.2 vallen enkele verschillen op.

**Minder tabellen dan klasses**

Het eerste wat opvalt, is dat de volgende klasses geen eigen tabel hebben gekregen:

- Base
- MainEntityBase
- Contact
- Organisation
- Person

De velden van Base, MainEntityBase en Contact zijn opgenomen in de tabel van de subklasse. Het voordeel van deze aanpak is dat bij het selecteren van de data van een object, er geen data uit verschillende tabellen hoeft worden samengevoegd.
Zo kan een Opportunity nu direct geselecteerd worden uit de Opportunities tabel zonder dat de data moet worden samengevoegd met bijbehorende data uit een MainEntity en Base tabel.

**Parties tabel**
De Organisation klasse en Person klasse hebben samen 1 tabel gekregen: de Party tabel. Deze tabel bevat kollommen voor alle velden van beide klasses.
Ook bevat de tabel een 'discriminator' kolom. Deze kolom wordt gebruikt om aan te geven of rij van de tabel een Organistation of een Person is.

Op het eerste gezicht is niet gelijk duidelijk waarom het Entity Framework voor deze aanpak gekozen heeft.
Wanneer we echter het klassediagram uit 7.2 bekijken zien we de oorzaak. De Contact klasse is immers verbonden met de Parties klasse. Door een Parties tabel te maken in plaats van afzonderlijke Organisations en People tabellen, kan in de verschillende Contact tabellen 'Party_id' als foreign key gebruikt worden.

Toch zou het wel mogelijk zijn om twee afzonderlijke tabellen te gebruiken. De verschillende Contact tabellen zouden dan twee foreign key kolommen krijgen: 'Organisation_id' en 'Person_id'.

## 13.2 Keys en indexen

**Primary keys**
Het Entity Framework heeft correct herkend dat het 'id' veld wat we aan elke klasse gegeven hebben de primare sleutel is.

**Foreign keys**
Ook de foreign keys zijn goed herkent. Zowel het veld 'PartyId' van Opportunity als het veld 'OrganisationId' van Person zijn als foreign key gedefinieerd.

Bij de verschillende Contact klasses heeft het Entity Framework een 'Party_id' kolom toegevoegd. Door middel van deze foreign key kolom wordt de relatie met Party gelegd.

**Indexen**
Voor zowel de primary keys als de foreign keys zijn indexen ingesteld. Verder zijn er geen indexen.

## 13.3 Datatypes

Wanneer we de datatypes van de properties in de code vergelijken met de datatypes van de velden zien we dat het Entity Framework de volgende mapping heeft toegepast:

| C# | Microsoft SQL Server |
|---|---|
| int | int |
| decimal | decimal |
| DateTime | datetime |
| string | nvarchar(max) |

Het enige wat opvalt, is dat voor strings nvarchar(max) wordt gebruikt. Ten opzichte van 'nvarchar' met een expliciete waarde, bijvoorbeeld nvarchar(255), heeft dit een voordeel en een nadeel.

Het nadeel is dat de kolom niet meer indexeerbaar is.
Het voordeel is dat de kolom meer data kan bevatten (tot 2gb)[6].

In ons geval maakt het weinig uit. We hebben geen indexen op tekstkolommen nodig, maar hebben ook geen velden die extreem grote hoeveelheden tekst bevatten.

## 13.4 Data Annotations

Wanneer het Entity Framework bepaalde aspecten van de database niet goed / zoals gewenst genereerd, kan dit gedrag worden bijgestuurd met Data Annotations.
Met deze Data Annotations kan extra informatie over een klasse of property van een klasse worden gedefinieerd. Het Entity Framework gebruikt deze informatie bij het genereren van de database.

Voorbeelden hiervan zijn het bepalen van welke tabellen er gegenereerd worden, of het instellen van een maximale lengte voor een property.

## 13.5 Conclusie

Het concept van een database genereren op basis van dataklasses was nieuw voor me. Het verbaasde me hoe goed het Entity Framework hiertoe in staat is. Naast het probleem met de primary key (zie 13.2) was de database direct bruikbaar.

Omdat we niet met grote hoeveelheden data te maken hebben en de database momenteel goed presteert, is het niet nodig om extra performance optimalisaties te onderzoeken.

Bij een applicatie die zelf de data van gebruikers moet valideren is het goed als er ook validatie op database niveau gebeurt. Het gaat dan niet alleen om maximale veldlengtes, maar ook om andere constraints.
Aangezien onze database de data uit een bestaande database moet overnemen, is het niet wenselijk dat de database zelf verdere validatie uitvoert. Dit zou niets toevoegen, maar wel problemen kunnen opleveren.

We kunnen dus concluderen dat het Entity Framework op basis van onze dataklasses een adequate database heeft gegenereerd. De database is direct bruikbaar en heeft geen aanpassingen nodig.

[6] http://msdn.microsoft.com/en-us/library/ms186939.aspx

# 14 Implementatie SyncService

In dit hoofdstuk wordt de implementatie van de SyncService besproken.

## 14.1 Opzet

In 11.3 werd uitgelegd waarom de SyncService de vorm zou krijgen van een Windows service.
Vervolgens is in 11.4 uitgelegd dat hier ook een installer voor gecreëerd moest worden.
Over het aanmaken van het SyncService project en de installer valt verder niets te vermelden.

## 14.2 Timers

In de eisen (7.2) stond dat de verwerking van create en update events binnen 5 minuten door de koppeling verwerkt moeten worden. Voor delete events was het voldoende om ze 1 keer per dag te verwerken.

Na onderzoek op internet bleek dat voor het periodiek uitvoeren van een actie het best de 'Timer' klasse gebruikt kan worden. Wanneer er verschillende acties een verschillend interval hebben moet voor elke actie een afzonderlijke Timer worden gebruikt.

**Interval**

Bij het bepalen van de intervallen van de twee methodes is zowel rekening gehouden met de genoemde eisen als de belasting van de Capsule API.

Voor de verwerking van de delete events moeten *alle* Organisations, People en Opportunities opgehaald worden. Om niet voor onnodige belasting van de server van de Capsule API te zorgen is er daarom gekozen om de delete events slechts 1 keer per dag te verwerken.

De verwerking van create en update events moest binnen 5 minuten gebeuren. Voor de verwerking van deze events hoeft alleen die data die daadwerkelijk is aangemaakt of gewijzigd worden opgevraagd. Omdat de hiervoor benodigde requests in verhouding voor weinig belasting zorgen, is er voor gekozen om de create en update events elke minuut te verwerken.
Op deze manier zal de verwerking van deze events gemiddeld een halve minuut nadat ze hebben plaatsgevonden gebeuren.

## 14.3 Foutenafhandeling

In de eisen (7.2) stonden drie eisen die betrekking hebben op hoe de koppeling met fouten omgaat:

1. Foutbestendigheid:
   Het systeem is foutbestendig. Wanneer er een fout optreedt tijdens het verwerken van een event, blijft het systeem wel draaien.

2. Rollback
   Het systeem is in staat om een event terug te draaien, wanneer een orchestratie van het ESB een foutmelding geeft.

3. Foutmelding
   Wanneer er een rollback plaatsvindt, stelt het systeem de gebruiker hiervan op de hoogte.

**Verval rollbackeis**

Tijdens de bespreking met de developers had ik het over de implementatie van de rollback. Naar

aanleiding hiervan bespraken we of het eigenlijk wel zinnig was om deze rollback functionaliteit te implementeren. We kwamen tot de conclusie dat dit niet het geval was:

1. Wanneer er een rollback nodig zou zijn, is er al iets misgegaan aan de ESB-kant. Dit probleem zal in de meeste gevallen toch al met de hand opgelost moeten worden. Het voegt dan niets toe als de wijziging aan de Capsule-kant automatisch is teruggedraaid.
2. De rollback functionaliteit is redelijk complex en wordt alleen gebruikt in het geval dat een fout optreedt. De kans is groot dat er bij de implementatie mogelijke situaties over het hoofd worden gezien. Hierdoor zou een rollback mogelijk voor extra fouten kunnen zorgen.

**Wijziging foutmeldingseis**

Naar aanleiding hiervan werd ook de foutmeldingseis besproken.
Er werd besloten dat Mike de foutmeldingsfunctionaliteit aan de kant van de ESB zou implementeren. De reden hiervoor is verantwoordelijkheid: wanneer de ESB een actie uitvoert waarin een fout optreedt, is de ESB ook verantwoordelijk voor het melden van deze fout.

De foutmeldingseis voor de koppeling werd gewijzigd in:
'Wanneer er een fout optreedt in de koppeling, wordt deze fout gelogd'.

**Implementatie foutlogging**

Voor de implementatie van het loggen van fouten heb ik op internet opgezocht hoe dit het best gedaan kan worden. Kan dit het beste in een eenvoudig tekst bestand gedaan worden, of is er een betere manier?

In de discussies die ik vond werd aangeraden de 'EventLog' te gebruiken. Met de EventLog kunnen verschillende types berichten gelogd worden, zoals: information, warning en error.
Vervolgens kan de standaard met Windows meegeleverde 'EventViewer'-applicatie gebruikt worden om de log te bekijken. Hierbij kan er ook voor gekozen worden om bijvoorbeeld alleen error-berichten weer te geven.

Op basis van deze 'EventLog' klasse heb ik de foutlogging functionaliteit geïmplementeerd. In de verdere code van de SyncService zijn 'try..catch' blokken gebruikt, waarbij in de 'catch' de exception wordt gelogd. Dit zorgt ervoor dat wanneer er een fout optreedt deze fout kan worden gelogd en de applicatie verder kan gaan.
Hierdoor wordt ook de foutbestendigheidseis verwezenlijkt.

## 14.4 Implementatie Tam Tam dataklasses

Voordat met de data vanuit Capsule een methode van de ESB aangeroepen moet de data eerst geconverteerd worden naar de Tam Tam structuur. Het klassediagram van de relevante dataklasses is te zien in 7.3.

Op basis van dit klassediagram is zijn de klasses geïmplementeerd. Vervolgens is er aan elke klasse een constructor toegevoegd waarmee een nieuwe instantie van de klasse kan worden aangemaakt op basis van een Capsule object.
In de constructor wordt vervolgens de conversie van de Capsule structuur naar de Tam Tam structuur uitgevoerd. Deze conversie was al beschreven in de architectual description (zie 7.4).

## 14.5 Verwerking toegevoegde en gewijzigde data

Het sequencediagram in 8.1 laat zien hoe de toegevoegde en gewijzigde data verwerkt wordt. Dit proces is voor zowel Organisations als People en Opportunities geïmplementeerd.

Bij de implementatie is er 1 methode gemaakt die dit proces uitvoert:
'CheckAndProcessModifiedItems'. Deze methode roept vervolgens drie methodes aan voor de afzonderlijke verwerking van Organisations, People en Opportunities.
Deze drie methodes voeren vervolgens het proces uit zoals dat in het sequencediagram beschreven wordt, met uitzondering van de rollback.

## 14.6 Verwerking verwijderde data

Het proces van de verwerking van de verwijderde data is te zien in het diagram in 8.2. Net als bij de verwerking van de gewijzigde en toegevoegde data is er 1 hoofdmethode gemaakt:
'CheckAndProcessDeletedItems'.
Deze methode roept de drie methodes voor de verwerking van verwijderde Organisations, People en Opportunities aan.

## 14.7 Het 'shortName' probleem

In 8.1 werd het 'shortName' veld al besproken. Er werd hier uitgelegd dat shortName in Capsule werd toegevoegd, omdat het veld nodig was voor de interne systemen van Tam Tam.

**Het probleem**
Wat ik tijdens de ontwerpfase niet wist, was dat shortName door verschillende van deze systemen als primary key gebruikt wordt. Het gevolg hiervan is dat er geen Company mag bestaan waarbij de shortName niet is ingevuld.

**Geen simpele oplossing**
De oplossing hiervoor leek simpel: het shortName veld in Capsule moet verplicht zijn. Het probleem is alleen dat custom fields in Capsule niet verplicht kunnen worden gemaakt. Er is daardoor geen manier om af te dwingen dat elke Organisation ook daadwerkelijk een ingevulde shortName zal hebben.

Het probleem wat hier uit voort komt is dat er wel Organisations kunnen bestaan zonder shortName, maar geen Companies zonder shortName.

**De oplossing**
Ik stelde als oplossing voor om de SyncService pas de CreateCompany orchestratie te laten uitvoeren wanneer een Organisation een shortName heeft.
Wanneer de shortName ontbreekt wordt de Organisation wel opgeslagen in de schaduwdatabase, maar wordt de CreateCompany orchestratie niet uitgevoerd. Wanneer de Organisation gewijzigd wordt, en na de wijziging wel een shortName bevat, wordt alsnog de CreateCompany orchestratie aangeroepen.

Dit werkt echter ook door op de Contacts en Opportunities. Wanneer de Company waar een Contact of Opportunity bij hoort nog niet is aangemaakt, kan de Contact of Opportunity ook nog niet worden aangemaakt.
De oplossing hiervoor is om bij het verwerken van de create / update events van Persons en

Opportunities de CreateContact en CreateOpportunity orchestraties alleen uit te voeren als de bijbehorende Organisation is aangemaakt.

Vervolgens moet wanneer bij een Organisation de shortName later wordt ingevuld op dit moment ook de CreateContact en CreateOpportunity orchestratie worden aangeroepen voor alle People en Opportunities die aan de Organisation gekoppeld zijn.

# 15 Testplan

In dit hoofdstuk wordt het testplan besproken.

Allereerst wordt echter de voortgang van het project besproken

Vervolgens begint elke paragraaf, met uitzondering van 15.3, begint met een gedeelte uit het testplan. Waar nodig wordt er nog een toelichting gegeven.

Zie bijlage B voor het gehele testplan.

## 15.1 Verlenging project

Inmiddels was het week 15. Ik liep dus nog steeds een week achter op de planning. Ik was echter ook nog steeds niet begonnen met het afstudeerverslag.

**Reflectie**

Ik besefte me dat het niet zou lukken om het project in de gestelde tijd af te krijgen. Ook besefte ik me dat ik eigenlijk veel eerder al had moeten ingrijpen en aan het verslag had moeten beginnen.

Ik kwam erachter dat er hier twee oorzaken voor waren:

1. Doordat ik niemand bij het proces en de planning had betrokken had ik hier zelf ook minder overzicht op.
2. Ik had het verslag uitgesteld wegens faalangst.

**Tussentijds assesment**

Tijdens het tussentijds assesment heb ik om verlenging van het project gevraagd. Aangezien er nog geen verslag was (qua documentatie alleen de architectual description) werd, besloten om een voorwaardelijke verlenging (van 10 weken) te geven. Ik zou me eerst de resterende weken van de oorspronkelijke projectduur op het verslag moeten richten. Vervolgens zou aan het eind van deze weken op basis van het gemaakte gedeelte van het verslag definitief worden besloten om de verlenging door te laten gaan of niet.

**Daily scrum**

Ik besloot om maatregelen te nemen om te voorkomen dat ik door mijn faalangst opnieuw niet aan het verslag zou werken. Ik heb aan een collega (Bart Venderbosch) gevraagd of hij elke dag een daily scrum (zie 3.1.1) met mij wilde houden. Hij stemde hier in toe en de rest van het project hebben we elke dag een daily scrum gehouden.

**Testfase**

De testfase is in de verlenging uitgevoerd.

## 15.2 Opdrachtomschrijving

Het doel van de testfase is de kwaliteit van de huidige en toekomstige code van de Capsule koppeling te verhogen. De tests moeten niet alleen gericht zijn op het opsporen van fouten in de huidige code maar ook op het voorkomen van regressiefouten.

Bij voorkeur wordt er zoveel mogelijk gebruik gemaakt van automatisch en snel uit te voeren tests.

**Toelichting**
De opdrachtomschrijving richt zich op het testen van de code.
Naast het testen van code is het echter ook mogelijk om documenten en diagrammen te testen. Er wordt dan bijvoorbeeld gekeken naar dingen als de onderlinge consistentie.

Bij Tam Tam ligt de nadruk echter minder op formele documentatie en meer op een praktische en tastbare aanpak. Vanwege deze reden is er gekozen om de tests ook zo praktisch en tastbaar mogelijk te maken

## 15.3 Opzet testplan

Bij het maken van het testplan moest een selectie worden gemaakt van de paragrafen die in het plan zouden worden opgenomen.

**Inhoud**
Als uitgangspunt is het inhoudsopgavetemplate van TestGoal[7] gebruikt. Uit dit template is een selectie van paragrafen gemaakt.
Er is voor gekozen om het testplan zo eenvoudig en beknopt mogelijk te houden. Op basis van de opdrachtomschrijving is bepaald wat voor tests er uitgevoerd gaan worden. Vervolgens zijn alleen die paragrafen opgenomen die noodzakelijk waren voor het plannen van deze tests.

**Geen detailtestplannen**
In TestGoal worden de concepten 'mastertestplan' en 'detailtestplan' beschreven. Het idee is dat er 1 mastertestplan wordt gemaakt en vervolgens per testsoort een detailtestplan. De structuur van de detailtestplannen is hierbij gelijk aan het mastertestplan, met als enig verschil dat de inhoud is toegespitsts op de betreffende testsoort.
Hoewel er gekozen is om twee testsoorten te gebruiken is er maar 1 testplan gemaakt. Er is hiervoor gekozen vanwege de eerder genoemde aanpak om het plan zo beknopt mogelijk te houden.

## 15.4 Kwaliteitsattributen

Als basis voor de teststrategie is een aantal kwaliteitsatributen geselecteerd. Er is hier uitgegaan van het kwaliteitsattributenoverzicht van TestGoal[8].

**Functionaliteit**

- Accuracy (nauwkeurigheid)
- Compliance (conformiteit)

**Betrouwbaarheid**

- Maturity (volwassenheid)

De tests zullen er op gericht zijn om de koppeling op deze punten te verbeteren.

**Toelichting**
Er is gekozen om de paragraaf 'kwaliteitsattributen' op te nemen om hiermee het doel van de tests

[7] TestGoal, pagina 124
[8] TestGoal, pagina 129

extra duidelijk te maken. De geselecteerde kwaliteitsattributen zijn goed toepasbaar bij het testen van code:

- Nauwkeurigheid:
  Door de code van bijvoorbeeld het converteren van XML naar de Capsule dataklasses te testen kunnen onnauwkeurigheden worden opgespoord en verholpen.

- Conformiteit:
  Wanneer de code van de transformatie van de Capsule dataklasses naar de Tam Tam dataklasses wordt getest, wordt het verwachte resultaat bepaald aan de hand van de modellen uit de architectual description. Op deze manier kunnen eventuele inconsistenties tussen het model en de implementatie gevonden worden.

- Volwassenheid:
  Door het testen van code kunnen 'kinderziektes' worden gevonden en verholpen. Op deze manier wordt de code volwassener.

## 15.5 Keuze testsoorten

Uit de testopdracht volgt dat unittests de voorkeur hebben.

**Unittests**

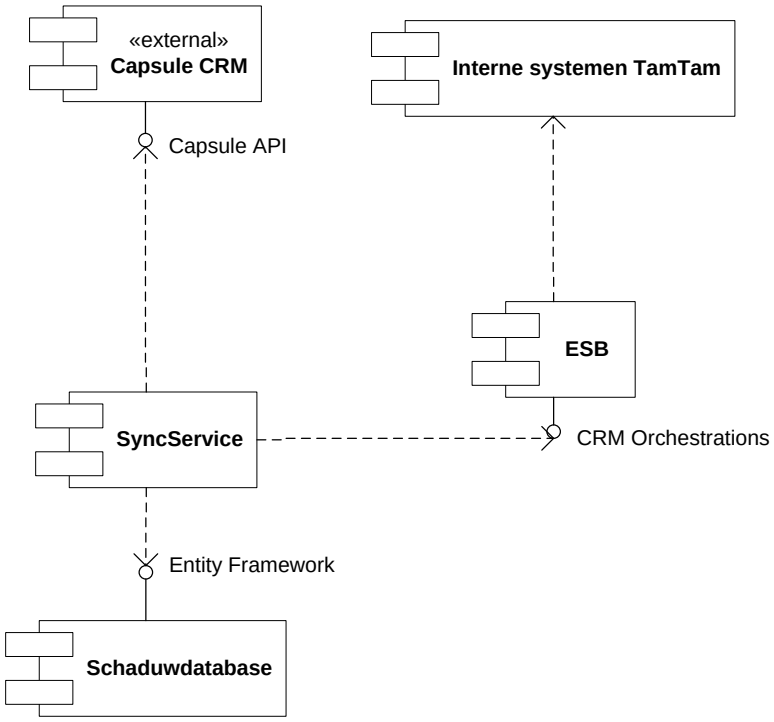
Unittests kunnen automatisch en snel uitgevoerd worden. Hierdoor zijn unittests zeer geschikt als regressietests. Ook zal de kwaliteit van de huidige code verhoogd worden door het maken van unittests.
Verder zijn unittests er op gericht om tijdens de verdere ontwikkeling voortdurend te worden uitgevoerd. Hierdoor kunnen regressiefouten snel worden opgespoord en verholpen.

Unittests hebben echter ook een beperking. Doordat ze voortdurend moeten worden uitgevoerd is het belangrijk dat ze snel uit te voeren zijn.
Om dit te bereiken wordt code die gebruik maakt van databases of andere systemen vervangen door een zogenaamde 'stub'[9]. Hierdoor zijn unittests dus niet geschikt voor het testen van de code die de communicatie met de database of andere systemen verzorgd.

**Communicatie met andere systemen**

Op het componentdiagram van de koppeling zien we met welke andere systemen de koppeling te maken heeft:

[9] Een vervangende klasse die de oorspronkelijke klasse simuleert.




Zoals te zien is werkt de SyncService samen met de volgende systemen:

- Capsule CRM, via de Capsule API.
- De schaduwdatabase, benaderd via het Entity Framework.
- De ESB, via de CRM Orchestrations.

**Integratietests**
Om de code die de communicatie met de database / andere systemen uitvoert toch te kunnen testen zullen er integratietests gemaakt worden.
Deze tests zullen niet automatisch uitgevoerd kunnen worden. Zowel het uitvoeren als het controleren van de integratietests zal handmatig gedaan moeten worden.

## 15.6 Testaanpak unittests

Voor het opstellen van de testaanpak van de unittests is gekeken welk gedeelte van de code met unittests getest kan worden.
In 15.5 werd al uitgelegd dat de code die de communicatie met andere systemen regelt niet met unittests getest zal worden.

**Testframework**
De unittests zullen gemaakt worden met het testframework van Microsoft. Dit testframework is geïntegreerd met Visual Studio en is daardoor gemakkelijk te gebruiken.

**Conversie en transformatie**
Uit analyse van de code bleek dat de volgende onderdelen zonder het gebruik van stubs te testen is:

- De conversie van de Capsule data van XML naar de C# dataklasses.
- De transformatie vanuit de Capsule dataklasses naar de Tam Tam dataklasses.

Voor het testen van deze code zullen unittests geschreven worden.

**Synchronizer**
Er zal ook geprobeerd worden om unittests voor de Synchronizer klasse te schrijven.
De Synchronizer klasse is verantwoordelijk voor de synchronisatie van de data in Capsule met de schaduwdatabase. Daarnaast is de klasse ook verantwoordelijk voor het uitvoeren van de juiste orchestraties van de ESB.

De klasse is dus afhankelijk van alle drie de in 15.5 genoemde externe systemen.
Om toch unitttests voor de klasse te kunnen schrijven zullen er stubs voor de klasses gemaakt moeten worden die de communicatie met deze systemen uitvoeren.

Aangezien ik nog niet veel ervaring met geautomatiseerd testen heb, is lastig in te schatten hoeveel tijd het kost om de benodigde stubs te schrijven. Wanneer blijkt dat het te veel tijd kost, is het beter om de Synchronizer klasse door middel van integratietests te testen.

**Toelichting**
Er is gekozen om gebruik te maken van het testframework van Microsoft. Uit onderzoek bleek dat er twee veelgebruikte testframeworks zijn voor het testen van C# code: NUnit en het framework van Microsoft.
Aangezien ik geen tijd had om deze frameworks uitgebreid te vergelijken heb ik voor het framework van Microsoft gekozen. De reden om voor het framework van Microsoft te kiezen is dat dit framework standaard beschikbaar is in Visual Studio en ook geïntegreerd is met Visual Studio. Het leek me dat het daarom minder tijd zou kosten om de unittests met dit framework op te zetten.

## 15.7 Testaanpak integratietests
Het doel van de integratietests is die delen van de code te testen, die niet met unittests getest konden worden.
In 15.5 werd al aangegeven dat het hier om de communicatie met de volgende drie systemen gaat: de Capsule API, de schaduwdatabase en de ESB.

**TestApp**
Voor het maken van de integratietests is het nodig dat er een manier is om de te testen code uit te voeren. Tijdens het ontwikkelen is hiervoor al de ‘TestApp’ gemaakt.
De TestApp bestaat uit een formulier met verschillende knoppen. Met elke knop kan een bepaalde methode worden uitgevoerd. De ontwikkelaar gebruikt vervolgens de debugger om te inspecteren of de code goed werkt.

Voor de integratietests zal de TestApp als basis worden gebruikt. Elke test zal bestaan uit een knop in de TestApp, in combinatie met een testscript waarin beschreven wordt hoe de test moet worden uitgevoerd.

**Capsule API**
De communicatie met de Capsule API wordt beschreven in de ‘Capsule API’-klasse. Deze klasse heeft verschillende methodes waarmee data uit het Capsule CRM systeem worden opgevraagd.
Voor elk van deze methodes zal een test worden gemaakt.

**Schaduwdatabase**
Voor de communicatie met de schaduwdatabase wordt de Code-First aanpak van het Entity Framework gebruikt. Dit gebeurt in de ‘CapsuleContext’-klasse.

Voor de volgende dingen zal een test gemaakt worden:

- Het genereren van de database.
- Het aanmaken, wijzigen en verwijderen van Organisations, People en Opportunities.
- Het vullen van de database met alle data vanuit Capsule CRM.

**ESB**

De communicatie met de ESB zal niet met integratietests getest worden. Er zijn hier verschillende redenen voor:

- De koppeling moet zorgen dat de orchestraties met de juiste parameters worden aangeroepen, dit wordt echter al getest met de unittests voor de transformatie.
- Wanneer dit getest zou worden, moet 'met de hand' testdata uit de interne systemen worden opgeruimd.

Vanwege deze redenen zal de communicatie met de ESB pas bij de live-gang getest worden.

**Synchronizer**

Wanneer het te veel tijd kost om de Synchronizer-klasse met unittests te testen, zal dit met integratietests moeten gebeuren.
Omdat de communicatie met de ESB niet in de integratietests wordt getest, zal er hiervoor een stub worden gebruikt.

## 15.8 Testomgeving

In het testplan worden de testomgeving / benodigdheden voor het uitvoeren van de unittests en de integratietests beschreven. Er is hierbij gekozen om de testomgeving zo eenvoudig mogelijk te houden. Als uitgangspunt is genomen dat de tests op de computer van de developer uit te voeren moeten zijn.

## 15.9 Producten

Het uitvoeren van de testfase levert de volgende producten op:

- Unittests.
- TestApp met knoppen voor de verschillende integratietests.
- Testscripts voor de verschillende integratietests.
- Een testrapport met de resultaten en bevindingen naar aanleiding van het uitvoeren van de tests.

## 15.10 Sprints

| Sprint | Beschrijving |
|---|---|
| 1 | Implementatie en uitvoeren unittests. Unittests voor de in 15.6 beschreven onderdelen. De unittests van de Synchronizer worden alleen uitgevoerd als het binnen sprint haalbaar is. |
| 2 | Implementatie en uitvoeren integratietests. Integratietests voor de in 15.7 beschreven onderdelen. Wanneer de unittests van de Synchronizer niet haalbaar worden, worden hier nog integratietests voor gemaakt. |

# 16 Implementatie unittests

In dit hoofdstuk wordt de implementatie van de unittests besproken.

## 16.1 Opzet

In het testplan was gekozen voor het testframework van Microsoft. Een van de voordelen van dit framework is de integratie met Visual Studio.
Door deze integratie is een nieuw testproject eenvoudig aan te maken. Dit gebeurt door een nieuw project aan de solution toe te voegen en te kiezen voor het type 'Test project'.

Het aangemaakte project bevat voorbeelden en uitleg over het gebruik van het testframework.

**Indeling**

Een testproject bestaat uit 1 of meerdere testklasses. Elk van deze klasses heeft 1 of meerdere testmethodes.

Ik heb ervoor gekozen om elke groep tests een eigen klasse te geven. Op deze manier zijn de tests duidelijk gegroepeerd. Een ander voordeel is dat de tests binnen een klasse los van de andere tests kunnen worden uitgevoerd.

## 16.2 Werking unittests

Elke unittest is op te delen in drie stappen:

1. Arrange
2. Act
3. Assert

**Arrange**

In de 'arrange' of 'inrichten' stap worden de voorbereidingen voor het uitvoeren getroffen. In deze stap worden onder andere de variabelen die tijdens het uitvoeren gebruikt worden geïnitialiseerd.

**Act**

In de 'act' of 'uitvoeren' stap wordt de methode die het onderwerp van de tests is uitgevoerd.

**Assert**

In de 'assert' of 'beweringen' stap worden er 1 of meerdere beweringen over de uitkomst van de test gedaan. Als deze beweringen kloppen is de test geslaagd.
Wanneer een test niet geslaagd is betekend dit dat er of een fout in de test zit, of in de geteste code.

## 16.3 Objecten vergelijken

In het testplan werd genoemd dat de conversie van XML naar de Capsule dataklasses getest zou worden. Ook zou de transformatie van Capsule dataklasses naar Tam Tam dataklasses getest worden.

**Controle conversie**

Bij deze tests is het resultaat van de uitvoering een object. Om te controleren of de test geslaagd is moet er gecontroleerd worden of dit object correct is.
Dit kan gedaan worden door tevens een object aan te maken met de verwachte eigenschappen. Vervolgens wordt de uitkomst vergeleken met dit object.

**Aanpak vergelijking**

Er is echter geen standaard manier om in C# twee objecten met elkaar te vergelijken.
Wanneer je dit toch wilt doen moet je een van de twee volgende methodes gebruiken:

1. Zelf alle waardes van de properties van de objecten met elkaar vergelijken.
2. Een klasse gebruiken die de properties van twee willekeurige objecten met elkaar kan vergelijken.

Aangezien de eerste methode erg bewerkelijk is heb ik voor de tweede methode gekozen. Ik heb hier echter niet zelf een klasse voor geïmplementeerd maar ben opzoek gegaan naar een bestaand component.

Mijn zoektocht leverde de 'Compare NET objects'-klasse op. Deze klasse bleek zelfs specifiek voor het vergelijken van objecten in testen gemaakt te zijn.
Wanneer twee objecten niet gelijk aan elkaar blijken te zijn worden de verschillen opgenomen in het testresultaat. Hierdoor is eenvoudiger te achterhalen waar de fout zit.

## 16.4 Voorbeeld unittest

Met behulp van het genoemde component zijn de tests voor conversie van XML naar de Capsule dataklasses en de transformatie van de Capsule dataklasses naar de Tam Tam dataklasses geïmplementeerd.
Als voorbeeld kijken we naar de test van de conversie naar de 'Address' klasse.

```
[TestMethod]
public void AddressDeserialization()
{
    //arrange
    XmlNode addressNode =
StringToXmlNode("<address><id>532</id><type>Office</type><street>Herenweg
165</street><city>Rotterdam</city><zip>2976 ZV</zip></address>");

    //act
    Address resultAddress = new Address(addressNode);

    //assert
    Address expectedAddress = new Address()
    {
        Id = 532,
        Type = "Office",
        Street = "Herenweg 165",
        City = "Rotterdam",
        Zip = "2976 ZV"
    };
    Assert.IsTrue(compareObjects.Compare(resultAddress, expectedAddress),
compareObjects.DifferencesString);
}
```

In de bovenstaande code zijn de drie stappen gemarkeerd.

In de 'arrange' stap wordt er een XmlNode aangemaakt die een adres bevat.
Vervolgens wordt in de 'act' stap een 'Address' aangemaakt op basis van deze XmlNode.
Ten slotte wordt in de 'assert' stap de gewenste uitkomst gedefinieerd en, met behulp van een instantie van de 'Compare NET objects'-klasse, vergeleken met de daadwerkelijke uitkomst.

## 16.5 Geen 'datadriven testing'

Bij het onderzoeken van de mogelijkheden van het testframework van Microsoft kwam ik het principe 'datadriven testing' tegen.

**Wat is datadriven testing?**

Het idee van datadriven testing is dat een test meerdere keren wordt uitgevoerd met verschillende data. De begindata en gewenste uitkomst staan dan niet zoals in het voorbeeld in de vorige paragraaf in de test zelf gedefinieerd, maar worden dynamisch geladen.

**Werking**

Voor het gebruik van datadriven testing met het testframework van Microsoft maak je allereerst een databron aan. Dit kan een comma-seperated-file, een XML-bestand of zelfs een database zijn.

Vervolgens definieer je meerdere 'rows' in de databron. Bij het testen wordt de testmethode telkens aangeroepen met de data uit 1 row.
Elke waarde van de row is echter wel een string. De testmethode moet daarom eerst de waardes uit de row omzetten naar het juiste type.

**Waarom niet gebruikt?**

Ik heb gekozen om geen gebruik te maken van de mogelijkheid tot datadriven testing.

Datadriven testing zou wel een meerwaarde hebben. Dat een test met bepaalde testdata geen fouten oplevert, betekend niet dat de test met andere testdata ook foutloos zal verlopen.
In het geval van white-box testing kan je zelfs de testdata bepalen op basis van de code. Wanneer je dit doet kun je ervoor zorgen dat alle verschillende paden van een functie doorlopen worden.

Het implementeren van datadriven testing kost echter aanzienlijk meer tijd. Dit komt vooral doordat je een databron moet gebruiken en ervoor moet zorgen dat de data omgezet wordt naar het juiste type. Dit zorgt voor extra complexiteit.

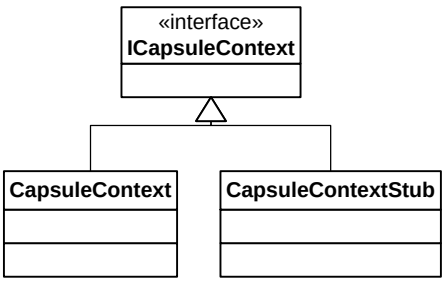
Omdat ik deze extra tijd niet had, heb ik besloten om geen datadriven testing toe te passen.

## 16.6 Tests Synchronizer

Om de Synchronizer-klasse te kunnen testen zouden er drie 'stubs' gemaakt moeten worden. Deze stubs zouden de functionaliteit van de CapsuleContext, CapsuleAPI en ESB klasse moeten simuleren.

**Implementatie stubs**

Het volgende diagram geeft de structuur weer die nodig is wanneer je een stub wilt aanmaken:




Allereerst moet er een interface worden aangemaakt op basis van de bestaande klasse. In het diagram is dit de 'CapsuleContext'-klasse. In deze interface worden de definities van de publieke

methodes van de bestaande klasse overgenomen.
In de bestaande klasse wordt aangegeven dat hij de aangemaakte interface implementeert.

Vervolgens kan overal waar een instantie van de klasse gebruikt wordt, het type worden gewijzigd naar de het type van de interface. Als een methode bijvoorbeeld als parameter 'CapsuleContext context' heeft, wordt dit gewijzigd in 'ICapsuleContext context'.

Als laatste kan de stub zelf worden geïmplementeerd. Hierbij moet de stub alle methodes van de echte klasse implementeren. Wanneer een methode niet nodig is voor het uitvoeren van de tests kan de inhoud van de methode echter gewoon leeggelaten worden.

**Te complex**
Tijdens het opstellen van het testplan had ik er al rekening mee gehouden dat het mogelijk te veel tijd zou kosten om de tests voor de Synchronizer klasse te maken.

Ik was al bezig met de implementatie toen ik erachter kwam dat het niet alleen te veel tijd zou gaan kosten om de tests te maken, maar dat ze ook te complex zouden zijn.

Om de tests te kunnen maken zou het volgende nodig zijn:

1. Een stub van de CapsuleContext. Deze stub moet een 'in-memory' database bevatten. Het opvragen, toevoegen, wijzigen en verwijderen van data moet gewoon werken.
2. Een stub van de CapsuleAPI. Deze stub moet instelbare properties hebben op basis waarvan de resultaten van de verschillende methodes bepaald wordt.
3. Een stub van de ESB service. Deze stub moet registreren welke orchestraties er zijn uitgevoerd en met welke parameters.
4. Testdata voor de CapsuleContextStub.
5. Testdata voor de CapsuleAPIStub.

De testdata zou hierbij zou gedefinieerd moeten worden dat elk type event (create, update delete) minstens 1 keer voorkomt voor elk datatype (Organisation, Person, Opportunity).

Kort samengevat kwam het er op neer dat zowel de stubs zelf, als de testdata redelijk complex zouden zijn.
De Synchronizer-klasse is zelf echter niet erg complex. De complexiteit van de Synchronizer is vooral verwerkt in de klasses die de Synchronizer gebruikt (CapsuleContext en CapsuleAPI). Deze code zou al in andere tests getest worden.

Het leek me geen goed idee om erg complexe tests te maken om eenvoudige code te testen. De kans is dan groot dat er fouten in de tests komen. De tests worden dan onbetrouwbaar.
Ook bestond het risico dat de imlementatie te veel tijd zou gaan kosten.

Ik heb er daarom voor gekozen om de Synchronizer klasse te testen met behulp van integratietests.

# 17 Implementatie integratietests

In dit hoofdstuk wordt de implementatie / het schrijven van de integratietests besproken.

## 17.1 Aanpak integratietests

In het testplan werd de aanpak voor het maken van de integratietests al besproken. De tests zouden bestaan uit een knop per te testen methode. De instructies voor het uitvoeren van de tests zouden beschreven worden door middel van testscripts.

**Stijl testscripts**

Normaal gesproken worden testscripts erg expliciet geschreven. De instructies zijn dan zo duidelijk en eenvoudig dat 'iedereen' de tests kunnen uitvoeren.
Mijn testscripts zijn echter op de (toekomstige) developers van de koppeling gericht. Ik heb er daarom voor gekozen om de testscripts ook op het 'niveau' van een developer te schrijven. Door hiervoor te kiezen zullen de developers de testscripts minder snel als 'kinderachtig' ervaren en eerder geneigd zijn de tests uit te voeren.

**Implementatie**

De implementatie van de integratietests verliep zonder problemen.
Voor verschillende te testen methodes was al een knop aanwezig in de TestApp. Voor deze tests hoefde dus alleen het testscript nog geschreven te worden.

## 17.2 Voorbeeld testscript

Als voorbeeld volgt hier het testscript van de 'GetAll' methodes van de CapsuleAPI-klasse:

"

Dit testscript beschrijft het testen van de verschillende 'GetAll'-methodes. Als voorbeeld wordt hier de 'GetAllOrganisations' methode genomen.

1. Start de TestApp met debugging (2.1).
2. Klik een 'GetAllOrganisations'-knop.
3. Stap met F10 over de code heen, tot het einde van de 'Click' procedure.
4. Controleer steekproefsgewijs de inhoud van de gevulde 'organisations' variabele. Vergelijk meerdere items met de corresponderende items in Capsule CRM.

"

In het bovenstaande testscript is te zien dat het script op developers is gericht. Voor het uitvoeren van de test moet de debugger van Visual Studio gebruikt worden.
Verder is te zien dat niet alle stappen tot in detail beschreven zijn. Er wordt bijvoorbeeld niet uitgelegd hoe de developer de inhoud van de 'organisations' variabele moet vergelijken met de corresponderende items in Capsule CRM.

Zie bijlage D voor de overige testscripts.

## 17.3 FillDatabaseFromAPI test

In elke integratietest wordt 1 bestaande methode getest. De code voor deze tests bestaat dan ook alleen uit het aanroepen van de te testen methode.

Een uitzondering hierop is de 'FillDatabaseFromAPI'-test. In de code voor deze test voor alle relevante data uit Capsule CRM opgehaald via de API. Vervolgens wordt deze data via de CapsuleContext-klasse opgeslagen in de schaduwdatabase.

In deze test wordt de samenwerking van de CapsuleAPI- en CapsuleContext-klasses getest. De functionaliteit van de test (het vullen van de database met de huidige data uit Capsule CRM) is ook nodig voor het uitvoeren van de tests van de Synchronizer en zal ook worden gebruikt bij de deployment van de koppeling.

# 18 Testrapport

Dit hoofdstuk bevat het testrapport naar aanleiding van de uitvoer van de unittests en integratietests.

## 18.1 Resultaten unittests

De implementatie en uitvoer van de unittests ging gelijk op.
Dit komt doordat bij de ontwikkeling van de unittests de testcode zelf ook getest moet worden. Wanneer een test niet succesvol wordt uitgevoerd hoeft dit niet per se te komen door een fout in de geteste code, maar kan dit ook komen door een fout in de test zelf.

**Tests transformatie**

Bij het uitvoeren van de tests voor de transformatie vanaf de Capsule dataklasses naar de Tam Tam dataklasses kwamen enkele bugs aan het licht. Deze bugs waren eenvoudig te verhelpen. De enige reden voor het bestaan van de bugs was dat er tijdens de implementatie de code nog niet goed getest was.

**Voordeel formeel testen**

Dit voorbeeld laat echter wel een belangrijk voordeel van formeel en gestructureerd testen zien. Wanneer het testen alleen op een informele manier tijdens het implementeren van de code gebeurt, kan het overzicht over welke code al getest is eenvoudig verloren worden. Ook zijn de tests die gedaan zijn niet goed herhaalbaar.
Bij formeel testen is dit probleem er niet. Het is dan duidelijk wat er getest is en op welke wijze. Ook zijn de tests herhaalbaar.

## 18.2 Resultaten integratietests

Het uitvoeren van de integratietests heeft geen bugs of andere problemen aan het licht gebracht. Dit is te verklaren door het feit dat een groot gedeelte van deze tests ook al tijdens de ontwikkeling van de betreffende methodes is uitgevoerd. De problemen die toen ontdekt werden, waren tijdens de officiële testfase dus al opgelost.

## 18.3 Suggesties verdere tests

De geïmplementeerde en uitgevoerde unittests en integratietests hebben samen een hoog code coverage percentage.
Met de unittests zijn de belangrijkste dataconversie methodes getest. Zowel de conversie van XML naar de Capsule dataklasses als de transformatie van Capsule dataklasses naar Tam Tam dataklasses zijn op deze manier getest.
In de integratietests zijn alle publieke methodes van de drie belangrijkste logica klasses van de koppeling getest: Synchronizer, CapsuleContext en CapsuleAPI.

**Datadriven testing**

De unittests zouden nog te verbeteren zijn door datadriven testing toe te passen. Door dezelfde tests met verschillende data uit te voeren, zouden de tests robuuster en betrouwbaarder worden.

De verschillende testdata zou kunnen worden bepaald op basis van de te testen methodes. Op deze manier zou het code coverage percentage van de geteste conversiemethodes (en de hulpmethodes die deze methodes gebruiken) naar 100% gebracht kunnen worden.

**Testen bij deployment**
Bij de integratietests is ook de communicatie met de API van Capsule en de database getest. De samenwerking van deze twee kwam aan de orde in de 'FillDatabaseFromAPI'-test.

Het uitvoeren van de verschillende CRM orchestraties van de ESB is echter nog niet getest. Het is daarom van belang dat dit bij de deployment van de koppeling uitgebreid getest zal worden.
Er is een redelijke kans dat tijdens dit testen nog enkele bugs of andere problemen gevonden zullen worden die zullen moeten worden opgelost.

NB: de deployment van de koppeling en de bijbehorende tests hebben op moment van schrijven nog niet plaatsgevonden en worden daarom niet in dit verslag beschreven. De reden hiervoor is dat Mike Noordermeer wegens omstandigheden nog geen tijd had gehad om de aanpassingen aan de webservice (zie 10.6) af te maken.

Momenteel is hij hiermee bezig. De deployment en bijbehorende tests zullen plaatsvinden in de weken tussen het inleveren van het verslag en de afstudeerzitting.

# 19 Evaluatie

Ik heb mijn afstudeerproject bij Tam Tam als erg positief ervaren. Ik denk dat ik veel geleerd heb, zowel op technisch vlak als op het vlak van algemene vaardigheden.

**Resultaat**

De koppeling is ontworpen, geïmplementeerd en getest. Het enige wat nog moet gebeuren is de deployment en de bijbehorende tests (zie 18.3). Vervolgens moeten de eventuele bugs die hierbij naar boven komen nog worden opgelost. Dit is bij elkaar nog 2 a 3 dagen werk. Dit zal worden uitgevoerd in de periode tussen het inleveren van het verslag en de afstudeerzitting.

Zowel de opdrachtgever als ik zijn tevreden over het gemaakte product. Ik ben van mening dat het project als succesvol kan worden beschouwd.

**Technische kennis**

Door het uitvoeren van het project heb ik nieuwe technische kennis van verschillende onderwerpen opgedaan:

- Het ontwerpen van de architectuur van een redelijk complex systeem.
- Het gebruiken van het Entity Framework.
- Het implementeren van een web-API in C#.
- Het maken en uitvoeren van unittests en integratietests.

**Projectaanpak**

Bij de projectaanpak ben ik in de valkuil gestapt die ik zelf al voorzag (3.1.2). Doordat ik zelf de rol van product owner vervulde betrok ik de opdrachtgever niet genoeg bij het project. De opdrachtgever vond dit geen probleem omdat hij inhoudelijk wel bij het project betrokken was.

Doordat ik hem echter qua proces en planning niet genoeg bij het project betrok, kreeg ik hier ook geen feedback op. Hierdoor miste ik het overzicht en bleef ik het schrijven van het verslag voor me uitschuiven.
Wanneer ik de opdrachtgever wel bij het proces en de planning had betrokken, had het me kunnen helpen om eerder en beter te reageren op het feit dat het project uitliep.

**Zelfcorrectie**

Om ervoor te zorgen dat het ondanks mijn faalangst met schrijven, toch zou lukken om het afstudeerverslag goed af te krijgen, had ik Bart Venderbosch gevraagd om elke dag een daily scrum met mij te houden (zie 15.1).

Deze daily scrums hebben mij goed geholpen tijdens de laatste fase van het project. Het zorgde ervoor dat ik de planning niet meer uit het oog verloor. Ook hielp het me als ‘stok achter de deur’ bij het schrijven van het verslag.

Ik heb hiervan geleerd dat ik er in de toekomst voor moet zorgen dat er altijd iemand betrokken is bij het proces en de planning.

**Oriëntatie beroepspraktijk**

Tijdens het uitvoeren van mijn project ben ik erachter gekomen dat ik liever frontend (web)development doe dan puur backend development. Bij het programmeren miste ik het aspect van het bezig zijn met een gebruikersinterface.

Ik heb bij Tam Tam al een gesprek gevoerd over de mogelijkhede en er is een functie als frontend developer voor mij beschikbaar. Na het afronden van het afstuderen zal er een vervolggesprek zijn waarna ik hoogstwaarschijnlijk bij Tam Tam in dienst zal treden.

# 20 Beroepstaken

In dit hoofdstuk wordt aangetoond waar de geselecteerde beroepstaken in het project zijn toegepast.

## 20.1 Overzicht beroepstaken

De volgende tabel geeft de voor gekozen beroepstaken weer:

| Code | Beschrijving | Niveau |
|---|---|---|
| ~~1.3~~ | ~~Selecteren van standaardsoftware~~ | ~~3~~ |
| 2.3 | Uitvoeren van gegevensconversie | 3 |
| 3.1 | Ontwerpen softwarearchitectuur | 4 |
| 3.3 | Bouwen applicatie | 3 |
| 3.4 | Initiëren en plannen van het testproces | 3 |

## 20.2 Demonstratie beroepstaken

**Selecteren van standaardsoftware**

Deze beroepstaak is niet toegepast binnen het project. De reden hiervoor was dat aan het begin van het project bleek dat de opdracht gewijzigd was (zie 2).

**Uitvoeren van gegevensconversie**

In de koppeling komen twee verschillende gegevensconversies voor:

1. De conversie van de XML van Capsule CRM naar de Capsule dataklasses.
2. De conversie van de Capsule dataklasses naar de Tam Tam dataklasses.

Deze conversies zijn zowel ontworpen (zie 7.4 en 9.2) als geïmplementeerd (zie 11.4 en 14.4).

**Ontwerpen softwartearchitectuur**

De beroepstaak ‘ontwerpen softwarearchitectuur’ is uitgevoerd met het maken van de architectual description. Deze wordt in beschreven in hoofdstuk 6 t/m 9.

**Bouwen applicatie**

Deze beroepstaak is uitgevoerd met het implementeren van de koppeling. Dit is beschreven in hoofdstuk 10 t/m 14.

**Initiëren en plannen van het testproces**

Deze beroepstaak is opgenomen vanwege het vervallen van de beroepstaak ‘selecteren van standaardsoftware’. Het testproces wordt beschreven in de hoofdstukken 15 t/m 18. Hoofdstuk 15 beschrijft het plannen van dit proces.

# 21 Literatuurlijst

Derk-Jan de Grood. ‘TestGoal’. Sdu Uitgevers, Den Haag, 2008.

Nick Rozanski en Eoin Woods. ‘Software Systems Architecture’. Pearson Education, Verenigde Staten, 2007.

# Bijlagen

De bijlagen bestaan uit de volgende onderdelen:

A. Plan van Aanpak
B. Architectual Description
C. Testplan
D. Testscripts

# Bijlage A: Plan van Aanpak

# Inhoudsopgave

Inleiding ..... 2

1 Opdracht ..... 3

1.1 Probleemstelling ..... 3

1.2 Doelstelling ..... 4

2 Afbakening ..... 5

3 Bedrijfsstandaards ..... 6

3.1 Versiebeheer ..... 7

3.2 Ontwikkelomgeving ..... 8

4 Methodes en technieken ..... 7

4.1 Projectmethode ..... 7

4.2 Architectual Description ..... 7

4.3 Testen ..... 8

5 Project backlog x ..... 9

6 Project planning x ..... 7

# Inleiding

Dit is het plan van aanpak voor het afstudeerproject van Mark Lagendijk bij Tam Tam. In dit document worden de gebruikelijke onderdelen van het plan van aanpak besproken, van opdrachtomschrijving tot planning.

# 1 Opdracht

In dit hoofdstuk wordt de opdracht besproken..

## 1.1 Probleemstelling

**Oorspronkelijke probleemstelling**

Bij Tam Tam maakt men momenteel gebruik van het Customer Relation Management (CRM) systeem van Microsoft: 'Microsoft Dynamics CRM 4.0'.

Men is niet tevreden met dit CRM systeem. Hier heeft men verschillende redenen voor:

4. Het systeem nodigt niet uit om te gebruiken.
5. Het systeem is alleen te gebruiken in Internet Explorer. Hierdoor is het systeem alleen onder Windows te gebruiken.
6. Het systeem heeft geen goede smartphone/tablet apps.

Vanwege deze redenen wil men graag overstappen op een SaaS[10] CRM systeem wat wel aan deze voorwaarden voldoet. Men heeft zich al globaal georiënteerd op de verschillende mogelijkheden. Men is erg enthousiast over Highrise[11].
Het nieuwe systeem zal echter wel, net als het huidige systeem, aangesloten moeten worden op de ESB (Enterprise Service Bus). Ook zal de data uit het oude systeem overgezet moeten worden naar het nieuwe systeem.

**Wijziging probleemstelling**

Aan het begin van het project bleek dat de probleemstelling gewijzigd was.

De wijziging van de probleemstelling werd door twee dingen veroorzaakt:

3. Door het horen van negatieve ervaringen met Highrise was men hier niet langer enthousiast over.
4. Het dochterbedrijf Mixit was al overgestapt op het gebruik van het SaaS CRM systeem 'Capsule CRM'.

Naar aanleiding van deze veranderde situatie werd een nieuwe probleemstelling opgesteld.

**Gewijzigde probleemstelling**

Bij Mixit is men overgegaan op het SaaS CRM systeem 'Capsule CRM'[12].

Bij de overgang heeft men de data uit Microsoft Dynamics CRM geëxporteerd naar Excel formaat en geïmporteerd in Capsule CRM.

Er is echter nog geen koppeling tussen Capsule CRM en de interne systemen van Mixit.

Tam Tam wil in een later stadium mogelijk ook overgaan op het gebruik van Capsule.

[10] Software as a Service. Dit is software die als online dienst wordt aangeboden. Hierdoor wordt de klant zaken als installatie, onderhoud en beheer uit handen genomen.
[11] http://highrisehq.com
[12] http://capsulecrm.com

## 1.2 Doelstelling

**Oorspronkelijke doelstelling**

Een succesvolle overgang uitvoeren naar een SaaS CRM systeem aan de hand van drie stappen:

4. Bepalen welk SaaS CRM systeem het best geschikt is voor Tam Tam.
5. Een koppeling tussen het gekozen SaaS CRM systeem en de interne systemen van Tam Tam ontwerpen en implementeren.
6. De migratie naar dit nieuwe CRM systeem ontwerpen en uitvoeren.

**Wijziging doelstelling**

Door de gewijzigde probleemstelling verandert ook de doelstelling.

**Gewijzigde doelstelling**

Een koppeling tussen Capsule CRM en de interne systemen van Mixit ontwerpen en implementeren.

Aangezien de interne systemen van Mixit een kopie zijn van de systemen van Tam Tam, zal wanneer Tam Tam overgaat op het gebruik van Capsule CRM, de koppeling ook voor Tam Tam gebruikt kunnen worden.

# 2 Afbakening

Het project zal zich richten op het ontwerpen, implementeren en testen van de koppeling tussen Capsule CRM en de interne Mixit systemen.
Het risico bestaat dat de interne systemen van Mixit in mindere of meerdere mate moeten worden aangepast voordat de koppeling er op aangesloten kan worden. In dit geval zullen deze aanpassingen niet door de student worden uitgevoerd, maar door Mike Noordermeer en/of Marco Krikke.

# 3 Project fases

In dit hoofdstuk worden de verschillende werkzaamheden van het project besproken.

## 3.1 Onderzoek huidige situatie

Allereerst zal de huidige situatie onderzocht moeten worden.

Dit onderzoek zal zich met name richten op de koppeling tussen Microsoft Dynamics CRM en de interne systemen van Tam Tam. De te bouwen koppeling zal dezelfde functionaliteit moeten bieden als deze bestaande koppeling.

## 3.2 Architectual description

Het ontwerp van de koppeling zal worden gemaakt in de vorm van een 'architectual description' (zie 5.2).

Bij het opstellen van de requirements zal het onderzoek naar de bestaande koppeling als uitgangspunt worden genomen.

Er zal worden onderzocht hoe de koppeling met Capsule CRM kan worden gebouwd met dezelfde functionaliteit als de bestaande koppeling. Op basis van dit onderzoek zal het verdere ontwerp worden gemaakt.

## 3.3 Implementatie koppeling

De koppeling zal worden geïmplementeerd met C#.NET (zie 4.2).
Bij de implementatie zal worden uitgegaan van de architectual description.

## 3.4 Testen koppeling

De koppeling zal ook formeel getest worden. Hiervoor zullen er verschillende soorten tests gemaakt worden (zie 5.3).

# 4 Methodes en technieken

In dit hoofdstuk worden methodes en technieken, die ten behoeve van het uitvoeren van de opdracht gebruikt zullen worden, besproken.

## 4.1 Projectmethode

Het project zal scrum als projectbeheersingsmethode gebruiken.

### 4.1.1 Wat is scrum?

In de volgende paragrafen zal wordt kort besproken wat scrum precies is.

**Definitie**

Scrum is een iteratieve projectbeheersingsmethode. Scrum wordt vaak gebruikt bij agile software ontwikkeling.

**Project backlog**

Elk scrumproject heeft een 'project backlog'.
De project backlog beschrijft de taken die binnen het project uitgevoerd moeten worden. Verder worden de resulterende producten van deze taken beschreven.

**Sprints**

Bij scrum wordt het project opgedeeld in sprints.
Een sprint heeft doorgaans een lengte van 1 tot 4 weken. Alle sprints binnen een project hebben dezelfde lengte.
Aan het begin van een sprint wordt de 'sprint backlog' opgesteld. Voor de sprint backlog worden 1 of meer taken van de project backlog opgesplitst en gedetailleerder beschreven.

**Daily scrum**

Bij scrum kan gewerkt worden met een dagelijkse vergadering, de 'daily scrum'.
Deze vergadering heeft een maximale duur van 15 minuten.
Tijdens de vergadering beantwoordt elk teamlid de volgende vragen:

- Wat heb je gisteren gedaan?
- Wat ga je vandaag doen?
- Zijn er 'impediments' (belemmeringen die de uitvoering van het werk in de weg staan)?

**Rollen**

Scrum onderscheid verschillende rollen:

| Rol | Omschrijving |
|---|---|
| Scrummaster | De scrummaster zorgt ervoor dat het team zijn werk goed kan doen. Hij zorgt dat impediments uit de weg worden geruimd. |
| Product owner | De product owner is de 'stem van de klant'. Hij is er verantwoordelijk voor dat de klant krijgt wat hij wil. Hij bereikt dit door de taken te omschrijven en prioriteit toe te kennen.<br>Als de klant betrokken genoeg is kan deze rol door de klant zelf vervuld worden. Wanneer dit niet het geval is wordt de rol door een teamlid vervuld. |
| Team | Het team voert het daadwerkelijke werk uit. Een scrum team bestaat doorgaans uit 5-9 personen. |

### 4.1.2 Scrum binnen het project

**Overzicht rolverdeling**

Tijdens het project zullen de volgende personen betrokken zijn:

| Wie | Rol |
|---|---|
| Bart Manuel | Opdrachtgever,<br>Hoofdarchitect interne systemen Tam Tam/ Mixit |
| Mark Lagendijk | Scrummaster, product owner en team (projectleider, architect, developer, tester) |
| Mike Noordermeer, Marco Krikke | Developer klantenportal |

**Rol Mark Lagendijk**

Zoals in de bovenstaande tabel te zien is zal ik alle drie de primaire scrum rollen vervullen: scrummaster, product owner en team.

Hier volgt per rol de motivatie voor deze keuze:

- Team:
  Aangezien ik de enige is die het werk daadwerkelijk uitvoert, ben ik (volgens de definitie binnen scrum) ook het enige teamlid.

- Scrummaster:
  De scrummaster is ervoor verantwoordelijk dat eventuele impediments uit de weg worden geruimd.
  Ik ervaar zelf deze eventuele impediments, het is daarom ook logisch dat ik ervoor verantwoordelijk ben dat deze worden opgelost.

- Product owner:
  De product owner houdt zich tot op detailniveau bezig met het bepalen van de uit te voeren taken. Verder bepaald hij de prioriteit van deze taken.
  Het vervullen van deze rol eist redelijk veel tijd. Wanneer deze rol niet door mij zou worden vervuld, zou degene die de rol vervuld minimaal een dagelijkse vergadering (daily scrum) met mij moeten hebben.
  De opdrachtgever is, als partner van Tam Tam, erg druk bezet. Ik heb er daarom voor gekozen om de opdrachtgever niet te vragen of hij deze rol wilde vervullen, maar hem zelf te vervullen.
  Het gevaar van zelf de product owner rol vervullen is dat de opdrachtgever niet meer genoeg bij het proces betrokken wordt. Wanneer dit gebeurt, bestaat het gevaar dat het uiteindelijke product niet overeen komt met de wensen van de opdrachtgever. Om dit te voorkomen moet de opdrachtgever bij alle belangrijke beslissingen betrokken worden.

**Rol Bart Manuel**

Bart Manuel zal naast de rol van opdrachtgever, ook de rol van systeem architect vervullen. Bart is binnen Tam Tam de opdrachtgever voor de ontwikkeling van de verschillende interne systemen. Vanwege zijn technische kennis van de interne systemen is hij ook stakeholder in de rol hoofdarchitect.

**Rol Mike Noordermeer en Marco Krikke**

Tijdens de implementatiefase van het project zullen Mike Noordermeer en Marco Krikke bij het project betrokken zijn. Als developers van de interne systemen van Tam Tam / Mixit zullen zij technisch advies geven tijdens de implementatie.
Verder zullen zij eventuele, voor de implementatie van de koppeling benodigde, wijzigingen aan de interne systemen van Mixit uitvoeren.

**Sprints**
In het de ontwerpfase van het project zal niet met sprints worden gewerkt. Deze fase zal meer volgens de klassieke watervalmethode worden uitgevoerd.
De reden hiervoor was de keuze om überhaupt een aparte ontwerpfase te hebben. Bij scrum wordt doorgaans het ontwerp tegelijk met de implementatie gemaakt. Vanwege de complexiteit van de koppeling was echter besloten een eerst een architectual description te maken.

Tijdens de development- en testfase van het project zal wel met sprints worden gewerkt. Deze sprints zullen een duur hebben van 1 week.
Er is gekozen voor een sprintlente van 1 week vanwege de duur van het afstudeerproject. Wanneer sprints langer dan 1 week zouden duren, zouden er maar enkele sprints zijn. Daardoor zouden de voordelen van het hebben van sprints verdwijnen.

## 4.2 Architectual Description

Het ontwerp van de koppeling zal worden gemaakt in de vorm van een zogenaamde 'architectual description'. Dit zal worden gedaan volgens de methode zoals beschreven in het boek 'Software Systems Architecture'[13].

**Viewpoints**
Het systeem zal beschreven worden vanuit verschillende 'viewpoints'. Elk viewpoint belicht een anders aspect van het systeem.
Er zal een selectie worden gemaakt uit de volgende viewpoints:

- Functional
- Information
- Concurrency
- Development
- Deployment
- Operational

**Perspectives**
Ook zullen er 1 of meerdere 'perspectives' worden toegepast. Voorbeelden van perspectives zijn 'evolution' en 'availability and resilience'.

**Modellen en diagrammen**
De modellen en diagrammen zullen met UML gemaakt worden.
Indien het de begrijpbaarheid van de Architectual Description ten goede komt, kan er ook gebruik worden gemaakt van vrije diagrammen. Aangezien er geen niet-technische stakeholders bij het project betrokken zijn, zal zoveel mogelijk UML gebruikt worden.

## 4.3 Versiebeheer

Binnen Tam Tam worden verschillende versiebeheersystemen gebruikt.
In het verleden gebruikte men 'Vault'. Enkele jaren geleden is men overgestapt op het gebruik van SVN. Bij deze overstap zijn bestaande projecten echter niet overgezet.

De code van de interne systemen van Tam Tam bevindt zich hoofdzakelijk in Vault. De code van nieuwe interne projecten wordt echter bij voorkeur in SVN gezet.

[13] Software Systems Architecture

In overleg met de opdrachtgever is besloten om de code van de koppeling in SVN te zetten.

## 4.4 Ontwikkelomgeving

Bij Tam Tam worden de meeste projecten met het .NET platform van Microsoft gemaakt. Hierbij wordt gebruik gemaakt van de IDE 'Visual Studio' en de programmeertaal 'C#'. Voor het maken van websites / webapplicaties maakt men gebruik van ASP.NET en ASP.NET MVC.

Vanwege de ervaring met C#.NET zijn de interne systemen ook hiermee gemaakt. Vanuit deze situatie volgt als vanzelfsprekend de eis om de koppeling ook met C#.NET te maken.

## 4.5 Testen

Voor de testfase zal het boek 'TestGoal'[14] als handboek gebruikt worden.
Er zullen verschillende soorten tests uitgevoerd worden.

**Unittests**

Voor het testen van de koppeling zullen unittests gemaakt worden.
Met unittests kunnen verschillende onderdelen van een systeem afzonderlijk getest worden.
Een ander voordeel van unittests is dat ze geautomatiseerd uitgevoerd kunnen worden, bijvoorbeeld bij elke build van het systeem. Hierdoor zijn unittests zeer geschikt als regressietest.

**Integrationtests**

Naast unittests zal er ook een integratietest uitgevoerd worden.
Unittests zijn zeer geschikt om functionaliteit van afzonderlijke modules te testen. Ze testen alleen niet de samenwerking tussen de verschillende modules. Om de samenwerking tussen de modules te testen zal daarom gebruik gemaakt worden van integrationtests.

[14] TestGoal, Derk-Jan de Grood

# 5 Project planning

In dit hoofdstuk wordt de globale planning voor het project weergegeven.

| Wanneer | Wat |
|---|---|
| Week 1 t/m 3 | Opstellen Plan van Aanpak.<br>Onderzoek huidige situatie. |
| Week 4 t/m 7 | Opstellen Architectual Description. |
| Week 8 t/m 12 | Implementeren koppeling. |
| Week 13 t/m 15 | Opstellen testplan.<br>Testen koppeling. |
| Week 1 t/m 17 | Opstellen afstudeerverslag. |

# Bijlage B: Testplan

# Inhoudsopgave

Inleiding ..... 1

1 Opdrachtomschrijving ..... 2

2 Testbasis ..... 2

3 Teststrategie ..... 3

3.1 Kwaliteitsatributen ..... 3

3.2 Keuze testsoorten ..... 3

3.3 Testaanpak unittests ..... 4

3.4 Testaanpak integratietests ..... 5

3.5 Testomgeving unittests ..... 6

3.6 Testomgeving integratietests ..... 6

4 Producten ..... 7

# Inleiding

Dit is het testplan voor de testfase van de Capsule CRM koppeling. In hoofdstuk 1 wordt de testopdracht geformuleerd. De teststrategie wordt besproken in hoofdstuk 3.

# 1 Opdrachtomschrijving

Het doel van de testfase is de kwaliteit van de huidige en toekomstige code van de Capsule koppeling te verhogen. De tests moeten niet alleen gericht zijn op het opsporen van fouten in de huidige code maar ook op het voorkomen van regressiefouten.

Bij voorkeur wordt er zoveel mogelijk gebruik gemaakt van automatisch en snel uit te voeren tests.

# 2 Testbasis

Zowel de architectual description als de code zelf zullen als testbasis gebruikt worden. Er zal dus sprake zijn van white-box testing.
Door gebruik te maken van white-box testing zal er gerichter getest kunnen worden. Verder is echte black-box testing überhaupt niet mogelijk, omdat de tester ook de developer is die de code geschreven heeft.

# 3 Teststrategie

In dit hoofdstuk wordt de teststrategie besproken.

## 3.1 Kwaliteitsatributen

Als basis voor de teststrategie is een aantal kwaliteitsatributen geselecteerd. Er is hier uitgegaan van het kwaliteitsattributenoverzicht van TestGoal[15].

**Functionaliteit**

- Accuracy (nauwkeurigheid)
- Compliance (conformiteit)

**Betrouwbaarheid**

- Maturity (volwassenheid)

De tests zullen er op gericht zijn om de koppeling op deze punten te verbeteren.

## 3.2 Keuze testsoorten

Uit de testopdracht volgt dat unittests de voorkeur hebben.

**Unittests**

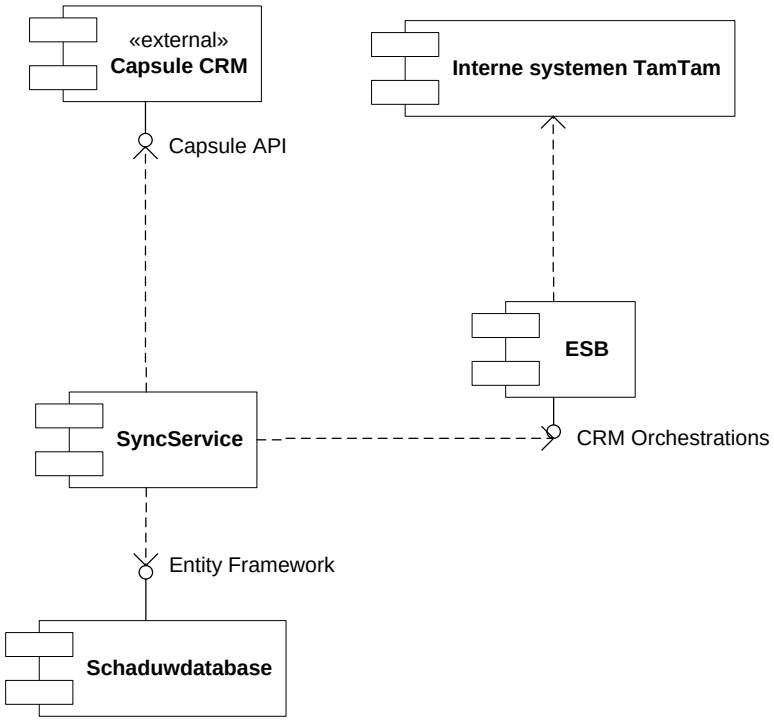
Unittests kunnen automatisch en snel uitgevoerd worden. Hierdoor zijn unittests zeer geschikt als regressietests. Ook zal de kwaliteit van de huidige code verhoogd worden door het maken van unittests.
Verder zijn unittests er op gericht om tijdens de verdere ontwikkeling voortdurend te worden uitgevoerd. Hierdoor kunnen regressiefouten snel worden opgespoord en verholpen.

Unittests hebben echter ook een beperking. Doordat ze voortdurend moeten worden uitgevoerd is het belangrijk dat ze snel uit te voeren zijn.
Om dit te bereiken wordt code die gebruik maakt van databases of andere systemen vervangen door een zogenaamde 'stub'[16]. Hierdoor zijn unittests dus niet geschikt voor het testen van de code die de communicatie met de database of andere systemen verzorgd.

**Communicatie met andere systemen**

Op het componentdiagram van de koppeling zien we met welke andere systemen de koppeling te maken heeft:

[15] TestGoal, Derk-Jan de Grood, pagina 129
[16] Een vervangende klasse die de daadwerkelijke klasse simuleert.




Zoals te zien is werkt de SyncService samen met de volgende systemen:

- Capsule CRM, via de Capsule API.
- De schaduwdatabase, benaderd via het Entity Framework.
- De ESB, via de CRM Orchestrations.

**Integratietests**
Om de code die de communicatie met de database / andere systemen uitvoert toch te kunnen testen zullen er integratietests gemaakt worden.
Deze tests zullen niet automatisch uitgevoerd kunnen worden. Zowel het uitvoeren als het controleren van de integratietests zal handmatig gedaan moeten worden.

## 3.3 Testaanpak unittests

Voor het opstellen van de testaanpak van de unittests is gekeken welk gedeelte van de code met unittests getest kan worden.
In 3.2 werd al uitgelegd dat de code die de communicatie met andere systemen regelt niet met unittests getest zal worden.

**Testframework**
De unittests zullen gemaakt worden met het testframework van Microsoft. Dit testframework is geïntegreerd met Visual Studio en is daardoor gemakkelijk te gebruiken.

**Conversie en transformatie**
Uit analyse van de code bleek dat de volgende onderdelen zonder het gebruik van stubs te testen is:

- De conversie van de Capsule data van XML naar de C# dataklasses.
- De transformatie vanuit de Capsule dataklasses naar de Tam Tam dataklasses.

Voor het testen van deze code zullen unittests geschreven worden.

**Synchronizer**
Er zal ook geprobeerd worden om unittests voor de Synchronizer klasse te schrijven.
De Synchronizer klasse is verantwoordelijk voor de synchronisatie van de data in Capsule met de schaduwdatabase. Daarnaast is de klasse ook verantwoordelijk voor het uitvoeren van de juiste orchestraties van de ESB.

De klasse is dus afhankelijk van alle drie de in 3.2 genoemde externe systemen.
Om toch unitttests voor de klasse te kunnen schrijven zullen er stubs voor de klasses gemaakt moeten worden die de communicatie met deze systemen uitvoeren.

Aangezien ik nog niet veel ervaring met geautomatiseerd testen heb, is lastig in te schatten hoeveel tijd het kost om de benodigde stubs te schrijven. Wanneer blijkt dat het te veel tijd kost, is het beter om de Synchronizer klasse door middel van integratietests te testen.

## 3.4 Testaanpak integratietests

Het doel van de integratietests is die delen van de code te testen, die niet met unittests getest konden worden.
In 3.2 werd al aangegeven dat het hier om de communicatie met de volgende drie systemen gaat: de Capsule API, de schaduwdatabase en de ESB.

**TestApp**
Voor het maken van de integratietests is het nodig dat er een manier is om de te testen code uit te voeren. Tijdens het ontwikkelen is hiervoor al de 'TestApp' gemaakt.
De TestApp bestaat uit een formulier met verschillende knoppen. Met elke knop kan een bepaalde methode worden uitgevoerd. De ontwikkelaar gebruikt vervolgens de debugger om te inspecteren of de code goed werkt.

Voor de integratietests zal de TestApp als basis worden gebruikt. Elke test zal bestaan uit een knop in de TestApp, in combinatie met een testscript waarin beschreven wordt hoe de test moet worden uitgevoerd.

**Capsule API**
De communicatie met de Capsule API wordt beschreven in de 'Capsule API'-klasse. Deze klasse heeft verschillende methodes waarmee data uit het Capsule CRM systeem worden opgevraagd.
Voor elk van deze methodes zal een test worden gemaakt.

**Schaduwdatabase**
Voor de communicatie met de schaduwdatabase wordt de Code-First aanpak van het Entity Framework gebruikt. Dit gebeurt in de 'CapsuleContext'-klasse.

Voor de volgende dingen zal een test gemaakt worden:

- Het genereren van de database.
- Het aanmaken, wijzigen en verwijderen van Organisations, People en Opportunities.
- Het vullen van de database met alle data vanuit Capsule CRM.

**ESB**
De communicatie met de ESB zal niet met integratietests getest worden. Er zijn hier verschillende redenen voor:

- De koppeling moet zorgen dat de orchestraties met de juiste parameters worden aangeroepen, dit wordt echter al getest met de unittests voor de transformatie.
- Wanneer dit getest zou worden, moet ‘met de hand’ testdata uit de interne systemen worden opgeruimd.

Vanwege deze redenen zal de communicatie met de ESB pas bij de live-gang getest worden.

**Synchronizer**
Wanneer het te veel tijd kost om de Synchronizer-klasse met unittests te testen, zal dit met integratietests moeten gebeuren.
Omdat de communicatie met de ESB niet in de integratietests wordt getest, zal er hiervoor een stub worden gebruikt.

## 3.5 Testomgeving / benodigdheden unittests

Voor het uitvoeren van de unittests is alleen Visual Studio 2010 nodig. De tests worden uitgevoerd door de solution in Visual Studio te openen en de optie ‘Run all tests in solution’ te kiezen.

## 3.6 Testomgeving / benodigdheden integratietests

Voor het uitvoeren van de integratietests is het volgende nodig:

- Visual Studio 2010
- Een Microsoft SQL Server instantie, bij voorkeur lokaal.
- Microsoft SQL Server Management Studio.
- Internetverbinding
- Een Capsule CRM account.

De tests worden uitgevoerd met behulp van de Test App die vanuit Visual Studio opgestart wordt.

Hierbij is een Microsoft SQL server instantie is nodig voor het aanmaken en gebruiken van een testversie van de schaduwdatabase.
De testdatabase en inhoud van de database wordt gecontroleerd via Microsoft SQL Server Management Studio.

De internetverbinding is nodig voor de communicatie met de Capsule API.

Het Capsule CRM account is nodig om in te kunnen loggen in Capsule. De tester kan vervolgens de data in Capsule vergelijken met verschillende testresultaten.

# 4 Producten

Het uitvoeren van de testfase levert de volgende producten op:

- Unittests.
- TestApp met knoppen voor de verschillende integratietests.
- Testscripts voor de verschillende integratietests.
- Een testrapport met de resultaten en bevindingen naar aanleiding van het uitvoeren van de tests.

# Bijlage D: Testscripts

# Inhoudsopgave

Inleiding ..... 2
1 Algemeen ..... 3
1.1 Benodigdheden ..... 3
1.2 Voorbereiding ..... 3
1.3 Werking testscripts ..... 3
1.4 Testdata in Capsule CRM ..... 3
2 Capsule API ..... 4
2.1 Starten TestApp met debugging ..... 4
2.2 GetAll methodes ..... 4
2.3 GetModified methodes ..... 4
2.4 Get methodes ..... 4
3 Capsule Context ..... 5
3.1 Starten TestApp ..... 5
3.2 Het (opnieuw) genereren van de database ..... 5
3.3 FillDatabaseFromAPI methode ..... 5
3.4 Create methodes ..... 6
3.5 Update methodes ..... 6
3.6 Delete methodes ..... 6
4 SyncService ..... 7
4.1 CheckAndProcessModifiedItems methode ..... 7
4.2 CheckAndProcessDeletedItems methode ..... 7

# Inleiding

Dit document bevat de testscripts voor het uitvoeren van de integratietests van de Capsule CRM koppeling. De testscripts zijn bedoelt om de developer van de koppeling te helpen om gestructureerd te testen.

In hoofdstuk 1 worden algemene instructies gegeven. In hoofdstuk 2 t/m 4 staan de testscripts voor het testen van de Capsule API, Capsule Context en SyncService.

# 1 Algemeen

In dit hoofdstuk wordt algemene informatie gegeven ten behoeve van het uitvoeren van de testscripts.

## 1.1 Benodigdheden

Voor de uitvoer van de tests zijn de volgende dingen nodig:

- Visual Studio 2010
- Een Microsoft SQL Server instantie, bij voorkeur lokaal.
- Microsoft SQL Server Management Studio.
- Internetverbinding
- Een Capsule CRM account.

## 1.2 Voorbereiding

De TestApp staat standaard ingesteld op het gebruik van een lokale database. Deze database wordt automatisch gegenereerd met de naam 'CapsuleContext'.
Pas de ConnectionString in het app.config bestand aan om een andere database te gebruiken.

## 1.3 Werking testscripts

De testscripts zijn opgedeeld in 3 testgroepen:

1. Capsule API
2. Capsule Context
3. SyncService

Elke groep heeft een corresponderend tabblad in de TestApp. Op elk tabblad staan meerdere knoppen. Met elke knop kan een bepaalde functionaliteit getest worden.

Voor elke groep is er ook een hoofdstuk in dit document. In elke punt van het hoofdstuk wordt een test, of een onderdeel van een test beschreven.
De tests zijn puntsgewijs beschreven. Een punt kan ook verwijzen naar een andere test of testonderdeel.

## 1.4 Testdata in Capsule CRM

In verschillende tests wordt geïnstrueerd om data in Capsule CRM toe te voegen, te wijzigen of te verwijderen.
Let op dat dit alleen zonder bijwerkingen kan worden gedaan zolang de koppeling nog niet gedeployed is. Wanneer deze tests na het deployen moeten worden uitgevoerd zal de gedeployde koppeling tijdelijk moeten worden uitgeschakeld. Dit dient uiteraard op een gecoördineerde wijze te gebeuren, zodat de gebruikers tijdens de test geen wijzigingen uitvoeren.

# 2 Capsule API

In dit hoofdstuk staan de testscripts waarmee de CapsuleAPI-klasse wordt getest.

## 2.1 Starten TestApp met debugging

Dit punt bevat instructies voor het starten van de TestApp met debugging. In de testscripts wordt naar dit punt verwezen.

1. Open de TamTam.Capsule solution.
2. Ga naar de code van CapsuleTestForm in het TamTam.Capsule.TestApp project.
3. Zet een breakpoint in de 'StartDebugging' procedure.
4. Start het TestApp project in debugging modus.

## 2.2 GetAll methodes

Dit testscript beschrijft het testen van de verschillende 'GetAll'-methodes. Als voorbeeld wordt hier de 'GetAllOrganisations' methode genomen.

5. Start de TestApp met debugging (2.1).
6. Klik een 'GetAllOrganisations'-knop.
7. Stap met F10 over de code heen, tot het einde van de 'Click' procedure.
8. Controleer steekproefsgewijs de inhoud van de gevulde 'organisations' variabele. Vergelijk meerdere items met de corresponderende items in Capsule CRM.

## 2.3 GetModified methodes

Dit testscript beschrijft het testen van de verschillende 'GetModified'-methodes. Als voorbeeld wordt hier de 'GetModifiedOrganisations' methode genomen.

1. Start de TestApp met debugging (2.1).
2. Maak 1 of meerdere Organisations aan via Capsule.
3. Klik een 'GetModfiedOrganisations'-knop.
4. Stap met F10 over de code heen, tot het einde van de 'Click' procedure.
5. Controleer dat de 'organisations' variabele de nieuw aangemaakte / gewijzigde Organisations bevat.
6. Wijzig de aangemaakte Organisations via Capsule.
7. Voer stap 3 t/m 5 nog eens uit.

## 2.4 Get methodes

Dit testscript beschrijft het testen van de verschillende 'Get'-methodes. Als voorbeeld wordt hier de 'GetOrganisation' methode genomen.

1. Start de TestApp met debugging (2.1).
2. Selecteer een Organisation in Capsule. Noteer het id wat in de url staat.
3. Vul het 'Capsule id' tekstveld in de TestApp met het genoteerde id.
4. Klik de 'GetOrganisation'-knop.
5. Stap met F10 over de code heen, tot het einde van de 'Click' procedure.
6. Controleer dat de 'organisation' variabele de geselecteerde Organisation bevat.

# 3 Capsule Context

In dit hoofdstuk staan de testscripts waarmee de CapsuleContext-klasse wordt getest.

## 3.1 Starten TestApp

Dit punt bevat de instructies voor het starten van de TestApp. In de testscripts wordt naar dit punt verwezen.

1. Open de TamTam.Capsule solution
2. Start het TestApp project in debugging modus

## 3.2 Het (opnieuw) genereren van de database

Dit testscript beschrijft het (opnieuw) genereren van de schaduwdatabase. Andere testscripts die een nieuwe (lege) database nodig hebben verwijzen naar dit testscript.

1. Start de TestApp (3.1).
2. Open de SQL Server instantie in Management Studio.
3. Controleer of de database 'CapsuleContext' al bestaat. Zo ja, klik dan op de 'DropDatabase'-knop.
4. Klik de 'CreateDatabaseIfNotExists' knop.
5. Open de (opnieuw) aangemaakte CapsuleContext database en stel vast dat de volgende tabellen zijn aangemaakt:
    - Addresses
    - Emails
    - Opportunities
    - Parties
    - Phones
    - Websites

## 3.3 FillDatabaseFromAPI methode

Dit testscript beschrijft het vullen van de CapsuleContext database met alle relevante data uit Capsule CRM.

1. Start de TestApp (3.1).
2. Maak de database (opieuw) aan (3.2).
3. Klik de 'FillDatabaseFromAPI'-knop.
4. Open de CapsuleContext database en controleer de inhoud van de tabellen. Stel steekproefsgewijs vast dat deze overeenkomt met de data in Capsule CRM.

## 3.4 Create methodes

Dit testscript beschrijft het testen van de verschillende ‘Create’-methodes. Als voorbeeld wordt hier de ‘CreateOrganisation’-methode genomen.

1. Start de TestApp (3.1).
2. Maak de database (opieuw) aan (3.2).
3. Klik de ‘CreateOrganisation’-knop.
4. Controleer dat de Organisation in de CapsuleContext database is opgeslagen. Vergelijk de opgeslagen Organisation met de ‘organisation‘-variabele uit de ‘buttonCreateOrganisation_Click’-methode.

## 3.5 Update methodes

Dit testscript beschrijft het testen van de verschillende ‘Update’-methodes. Als voorbeeld wordt hier de ‘UpdateOrganisation’-methode genomen.

1. Voer de Create methodes test uit (3.4).
2. Klik de ‘UpdateOrganisation’-knop.
3. Stel vast dat de Organisation in de database gewijzigd is. Vergelijk de Organisation met de ‘organisation‘-variabele uit de ‘buttonUpdateOrganisation_Click’-methode.

## 3.6 Delete methodes

Dit testscript beschrijft het testen van de verschillende ‘Delete-methodes. Als voorbeeld wordt hier de ‘DeleteOrganisation’-methode genomen.

1. Voer Create methodes test uit (3.4). Voer optioneel de Update methodes test uit (3.5).
2. Klik de ‘DeleteOrganisation’-knop.
3. Stel vast dat de Organisation verwijderd is uit de database.

# 4 SyncService

In dit hoofdstuk staan de testscripts waarmee de SyncService-klasse wordt getest.
De ESB is hier vervangen door een stub die als dummy fungeert. Er wordt hierdoor niet getest of de juiste orchestraties worden aangeroepen.

## 4.1 CheckAndProcessModifiedItems methode

Dit testscript test de 'CheckAndProcessModifiedItems'-methode. Hierbij wordt het ophalen van de wijzigingen en het verwerken van deze wijzigingen in de Capsule Context database getest.

1. Start de TestApp in debugmodus (2.1).
2. Zorg dat de database gevuld is met de data uit Capsule CRM (3.3).
3. Maak 1 of meerdere Organisations, People en Opportunities aan in Capsule CRM.
4. Controleer de datum + tijd in het 'Modified since'-tekstveld. Als het goed is bevat deze de datum van 'vandaag' en het tijdstip van enkele minuten voor het starten v/d test.
5. Klik op de 'CheckAndProcessModifiedItems'-knop.
6. Controleer in de CapsuleContext database dat de nieuw aangemaakte / gewijzigde items zijn opgeslagen.
7. Wijzig 1 of meer van de aangemaakte Organisations, People en Opportunities.
8. Voer stap 4 t/m 6 nog eens uit.

## 4.2 CheckAndProcessDeletedItems methode

Dit testscript test de 'CheckAndProcessModifiedItems'-methode. Hierbij wordt het verwerken van verwijderede items getest.

1. Voer de test van de 'CheckAndProcessModifiedItems'-methode uit (4.1).
2. Verwijder 1 of meerdere van de aangemaakte items.
3. Klik op de 'CheckAndProcessDeletedItems'-knop.
4. Controleer dat de verwijderde items ook uit de database zijn verwijderd.