Afstudeeropdracht	Exact Online Client SDK
Auteur	A.S.R. Brandt
Plaats	Delft
Inleverdatum	27 maart 2014
Examinator 1	Dhr. B. Kuiper
Examinator 2	Dhr. G.M. Tuk
Bedrijfsmentor	Dhr. E. Wieringa

Inhoud

Deel 1: Afstudeerverslag

Deel 2: Plan van Aanpk

Deel 3: Master Testplan

Deel 4: Detail Testplan

Deel 5: Developer Documentatie

Deel 6: Goedgekeurde Afstudeerplan

Afstudeerverslag

Exact Online Client Software Development Kit

Dankwoord

Graag wil ik het Connectivity Team, en in het bijzonder Els Gootjes, en Cihan Duruer bedanken voor hun begeleiding mijn afstuderen bij Exact. Niet eerder heb ik zulke behulpzame, geduldige en gemotiveerde collega's gehad. Voornamelijk door hun hulp en inzicht heb kan ik stellen dat mijn afstudeerstage bij Exact één van de leerzaamste ervaringen is geweest die ik tot nu toe bij een bedrijf heb gehad. Ook wil ik Edgar Wieringa en Jurjen Boss bedanken voor hun begeleiding en de mogelijkheid om af te studeren bij Exact.

Inhoud

1 Inleiding 2
2 Organisatie 3
2.1 Beschrijving organisatie 3
2.2 Situatie aanvang afstuderen & toekomstvisie 3
2.3 Plek van de student binnen de organisatie 4
3 Huidige structuur 5
3.1 RESTful Application Programming Interface 5
3.2 Open Data Protocol 6
3.3 Authenticatie: OAuth 2.0 6
4 Huidige Project Management Methode 8
5 Opdrachtomschrijving 9
5.1 Exact Online Software Development Kit 9
5.2 Verantwoordelijkheden 10
5.3 Planning 10
6 Gekozen aanpak 11
6.1 Keuze Soort Ontwikkelmethode 11
6.1.1 Spiraal model 11
6.1.2 Waterval Methode 11
6.1.3 Iteratieve Methode 12
6.1.4 Conclusie & Keuze 12
6.2 Keuze specifieke Software Ontwikkel Methode 12
6.2.1 Unified Process 12
6.2.2 Test Driven Development 13
6.2.3 Scrum 13
6.2.4 Conclusie & Keuze 14
6.3 Keuze Testmethode 14
6.3.1 Testen tijdens ontwikkeling 15
6.3.2 Testmethode 15
6.4 Ontwikkeling 16
7 Werkzaamheden Week 1 & 2 – Oriëntatie 17
8 Werkzaamheden Week 3 & 4 – Sprint 0 18

9 Werkzaamheden Week 5 & 6 – Sprint 1 20
9.1 Automated Testing 20
9.2 Lagen Exact Online Software Development Kit 21
10 Werkzaamheden Week 7 & 8 – Sprint 2 22
10.1 Vertalen van Json Response uit de API 22
10.2 Updaten en Verwijderen van Entiteiten uit Exact Online 23
10.3 Code Review 23
10.4 Demo Stakeholers SDK 24
11 Werkzaamheden Week 9 & 10 - Sprint 3 25
11.1 Research naar de oData SDK 25
11.2 Updaten van entiteiten 25
11.3 Uitvoeren van een oData Query 26
12 Werkzaamheden Week 11 & 12 - Sprint 4 27
12.1 Refactoring SDK Architecture 27
12.2 Entity Controller 28
12.3 Singleton Pattern 28
12.4 Conclusie 29
13 Werkzaamheden Week 13 & 14 - Sprint 5 30
13.1 Beperkingen JSON.NET 30
13.2 Opschonen van de API Response met APIResponseStripper 30
13.3 Bug: DateTime Format met JavaScriptSerializer 33
13.4 Conclusie 33
14 Werkzaamheden Week 15 - Sprint 6 34
14.1 Modificeren van gekoppelde entiteiten 34
15 Werkzaamheden Week 16 & 17 - Sprint 7 36
15.1 Create Entity with Collection 36
15.2 REST Principles for PUT 37
15.3 Async Functionality 37
16 Werkzaamheden Week 18 & 19 - Sprint 8 39
16.1 Developer Documentatie 39
16.2 REST Principles Linked Entities voor PUT 39
16.3 Beschikbaar stellen van de models 41

17 Werkzaamheden Week 20 42
17.1 Compleet maken Unit Tests 42
17.2 Ondersteunen gebruikers in bèta fase 44
18 Afwijkingen afstudeerplan 45
18.1 Software Ontwikkel Methode & Ontwerpen Systeemdeel 45
18.2 TestGoal 45
19 Opgeleverde producten 46
20 Evaluatie aanpak 48
20.1 Communicatie met stakeholders 48
20.2 Opstellen documentatie 48
20.3 Scrum voor SDK Project 48
20.4 Testing & Test Driven Development 48
21 Evaluatie opgeleverde producten 50
21.1 Plan van Aanpak 50
21.2 Detail Ontwerp 50
21.3 Master- en Detail Test Plan 50
21.4 Software Development Kit 50
21.5 Regressie Testen 51
21.6 Sprint Review 51
21.7 Developer Documentation 51
22 Begrippen- en Afkortingenlijst 53
23 Wijze van aantonen competenties 54
23.1 Ontwerpen Systeemdeel - Complexiteit: Complex niveau: 4 54
23.2 Initiëren en plannen van het testproces – Complexiteit: Lastig, niveau: 3 54
23.3 Bouwen applicatie – Complexiteit: Complex, niveau: 4 55

1 Inleiding

Dit afstudeerverslag is geschreven naar aanleiding van het afstuderen van Aschwin Brandt (de student) bij Exact voor de afstudeeropdracht 'Exact Online Client Software Developent Kit'. Deze Software Development Kit (SDK) is in een periode van 20 weken ontworpen en gebouwd voor third party developers om op een makkelijke manier een koppeling te kunnen maken tussen hun applicatie en Exact Online.

Het afstudeerverslag is onderdeel van het afstudeerdossier en zal samen met de bijlagen, gebruikt worden door de begeleidend docenten om te helpen bij het beoordelen van de student. Het afstudeerverslag bestaat uit verschillende onderdelen die in aparte hoofdstukken zijn beschreven. In het eerste deel van het verslag wordt de situatie bij aanvang van het afstuderen en de gewenste situatie beschreven. Vervolgens kan terug worden gelezen welke technieken er momenteel gebruikt worden, hoe deze werken, en wordt de gekozen aanpak en de beschrijving van de werkzaamheden beschreven. Ook zal uitleg worden gegeven over de afwijkingen met betrekking tot het afstudeerplan.

Na het lezen van het afstudeerverslag heeft de lezer inzicht in de opdracht, het doel, de diepgang van de opdracht, de werkzaamheden, ondervonden problemen en gemaakte keuzes van de student, en kan een eerste beoordeling worden gegeven. Na deze beoordeling zal, mits deze positief is, een voordracht worden gegeven van de student waar in deze een onderdeel van de opdracht zal uitlichten en de een kans heeft vragen van de examinatoren te beantwoorden. Na de voordacht zal een definitieve beoordeling worden gegeven voor het afstuderen.

2 Organisatie

In dit hoofdstuk wordt de organisatie en de huidige situatie met betrekking tot het afstuderen beschreven. Na dit hoofdstuk is duidelijk wat de doelstelling van Exact Online is, wat de situatie was bij aanvang van het afstuderen, en wat de rollen van de student binnen de organisatie en haar doelstellingen zijn geweest.

2.1 Beschrijving organisatie

Exact Holding is een beursgenoteerd Nederlands softwarebedrijf dat is opgericht in 1984. Momenteel heeft Exact 1900 werknemers die verspreid zitten over vestigingen in 20 landen. Exact richtte zich van oorsprong alleen op boekhoudsoftware voor midden- en kleinbedrijf, maar heeft over de jaren heen meerdere producten ontwikkeld die ook andere bedrijfsprocessen ondersteunen (projectmanagement, fabricage, klantenservice, salarisadministratie etc.). Het doel van Exact is om ondernemers te helpen hun doelen te realiseren door er onder andere voor te zorgen dat de ondernemer 24/7 per dag overzicht heeft in de (financiële) situatie van zijn of haar bedrijf.

Exact is sinds 2005 bezig met de cloud oplossing 'Exact Online'. Hiermee kunnen ondernemers hun boekhouding doen via een webbrowser zonder eerst hard- en software aan te schaffen en te installeren. Alle data wordt opgeslagen in datacenters die door partners van Exact worden beheerd, waardoor deze data altijd en overal beschikbaar is. Het enige dat de ondernemers nodig hebben is een computer met een webbrowser en een internetverbinding.

2.2 Situatie aanvang afstuderen & toekomstvisie

Om het mogelijk te maken voor partners, dochterbedrijven of business partners om data uit Exact Online te integreren met hun eigen applicaties, heeft Exact diverse Application Programming Interfaces (API's) ontwikkeld. Deze zijn gemaakt en worden onderhouden door het Connectivity Team.

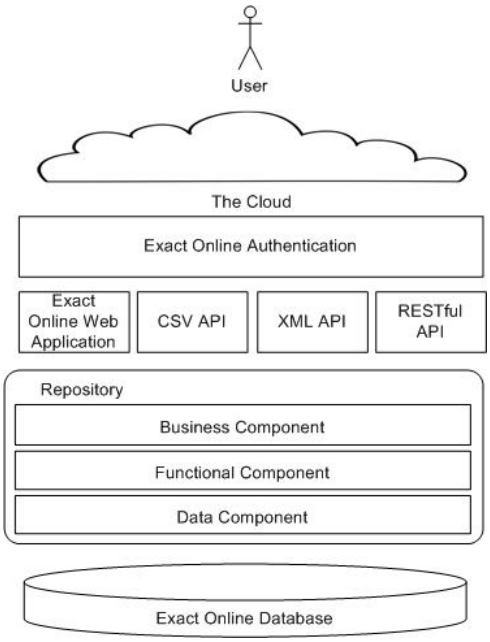
Het probleem met de API's is dat deze niet gemakkelijk zijn in het gebruik. De ontwikkelaars moeten zelf requests opbouwen en responses interpreteren, ze moeten weten welke velden een bepaalde resource/entiteit heeft, wat voor acties ze kunnen uitvoeren op een resource etc. Ook moeten ze kennis hebben van REST, oData, OAuth en Json. Daarnaast zijn de aanroepen erg gevoelig voor typfouten omdat naamgeving erg belangrijk is, en worden de REST principes niet altijd goed gehanteerd wat veel dataverkeer oplevert. Voornamelijk het werken met representaties van data wordt vaak vergeten. Zo kan het voorkomen dat een third party developer altijd de hele resource opvraagt terwijl deze slechts twee velden nodig heeft. Om het voor de third party developers makkelijk te maken om te werken met de API, en om het correct gebruiken van REST af te dwingen moest een Software Development Kit worden ontwikkeld.

De vraag naar een SDK zal eind mei ook toenemen. Met de release van de AppCentre (een platform waar third parties hun applicatie kunnen verkopen) zal er naar alle waarschijnlijkheid een toename zijn in applicaties die werken met Exact Online.

2.3 Plek van de student binnen de organisatie

Tijdens het afstudeertraject was de student onderdeel van het ‘Connectivity Team’. Dit is een Scrum Team van 5 personen, dat verantwoordelijk is voor de bouw en onderhoud van de Exact Online Application Programming Interface (API). Daarnaast heeft het Connectivity Team de bouw van het App Centre als verantwoordelijkheid, en zal zij in de toekomst de Exact Online SDK beheren.

3 Huidige structuur



Afbeelding 1: Huidige structuur Exact Online

In Afbeelding 1 wordt de huidige structuur van de Exact Online web applicatie en de verschillende API's en hun onderliggende componenten weergegeven in een lagendiagram. De onderste laag is de Exact Online database waarin alle data wordt opgeslagen. De Repository laag bestaat uit verschillende componenten die de business logica maken. De laag daarboven maakt deze data en business logica beschikbaar voor gebruikers (via de Exact Online Web Applicatie) en de verschillende API's. Voor de afstudeeropdracht (die wordt beschreven in hoofdstuk 5) zal worden gewerkt met de Exact Online RESTful API en de Authentication laag. Deze worden in de onderstaande hoofdstukken toegelicht.

3.1 RESTful Application Programming Interface

Exact heeft vier manieren om data uit de Exact Online database te benaderen: De Exact Online Web Applicatie en drie API's: CSV, XML & REST. De RESTful API is de nieuwste API ontwikkeld door het Connectivity Team. Deze wordt in dit hoofdstuk behandeld, en zal in de rest van het verslag worden gerefereerd als 'de API' of 'Exact Online API'.

REST staat voor 'Representational State Transfer' en is een architectuur stijl gebaseerd op het http protocol om voornamelijk prestaties, schaalbaarheid, eenvoud en overdraagbaarheid te bevorderen[1]. REST lijkt veel op http, maar is in feite een structuur die de schaalbaarheid en fout tolerantie van een systeem garandeert. In veel gevallen wordt http gebruikt in combinatie met REST, maar andere protocollen zoals SMTP kunnen ook gebruikt worden[2]. Ook is een REST API stateless, wat inhoud dat elke aanroep naar de server alle benodigde informatie moet bevatten en dus geen gebruik kan maken van opgeslagen informatie op de server zoals een sessie[3].

Exact gebruikt de REST architectuur voor één van haar API's omdat oData hier gebruik van maakt (zie hoofdstuk 3.2). Op deze manier kunnen applicaties worden gebouwd die gebruik maken van data uit Exact Online. Via https wordt een connectie gemaakt met de API en worden commando's uitgevoerd om data op te halen of te manipuleren. De voordelen van een REST API zijn:

[1] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
Intro into REST http://youtu.be/YCcAE2SCQ6k
What is REST? http://www.restapitutorial.com/lessons/whatisrest.html
[2] http://restcookbook.com/Miscellaneous/rest-and-http/
[3] REST Architectural Style http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

- **Eenvoud**: Slechts een aantal commando's, gemakkelijk leesbare URI's
- **Schaalbaarheid**: Door de eenvoud en scheiding van verantwoordelijkheden wordt de complexiteit en daarmee de effectiviteit en performance verbeterd. Ook is REST stateless waardoor er geen informatie opgeslagen hoeft te worden
- **Representaties van de data**: Er kunnen delen van data worden opgestuurd of opgevraagd waardoor bandbreedte wordt bespaard
- **Losse koppeling**: De API kan door alle soorten applicaties gebruikt worden

3.2 Open Data Protocol

Open Data Protocol (oData) is een web protocol voor het bevragen en updaten van data. Het protocol is van oorsprong ontwikkeld om standaard Create, Read, Update en Delete (CRUD) commando's uit te voeren op data. oData gebruikt REST voor alle requests[4]. Een voordeel van oData is dat deze een uniforme manier biedt voor het aanroepen van data. Hierdoor kan iedereen met kennis van dit protocol gemakkelijk de aangeboden data uit elke API die dit protocol ondersteund ophalen.

Ook heeft oData een gemakkelijk te begrijpen query syntax waarmee clients, op een vergelijkbare manier als queries op een database databases, data kunnen opvragen. Voor de Exact API wordt oData gebruikt om de data uit Exact Online gemakkelijk beschikbaar te maken voor externe partijen. Omdat oData een REST-based data service is zijn de aanroepen via http URI's. Zo'n URI bevat de volgende onderdelen:

- **Service Root**: De URI die verwijst naar de datasource
- **Resource Path**: Het specifieke entiteit die opgevraagd moet worden
- **Query Options**: Eventuele (filter) opties

Een voorbeeld van API requests wordt getoond in Tabel 1. Hier is de uiteindelijke URI opgesplitst in 'service root', 'resource path' en 'query options' om de leesbaarheid te verbeteren.

Service Root	Resource Path	Query Options
https://start.exactonline.nl/api/v1/499156/	financial/GLAccounts	
https://start.exactonline.nl/api/v1/499156/	Accounts	?$filter=Code+eq+'123
https://start.exactonline.nl/api/v1/499156/	Accounts	?$select=Code,Name

Tabel 1

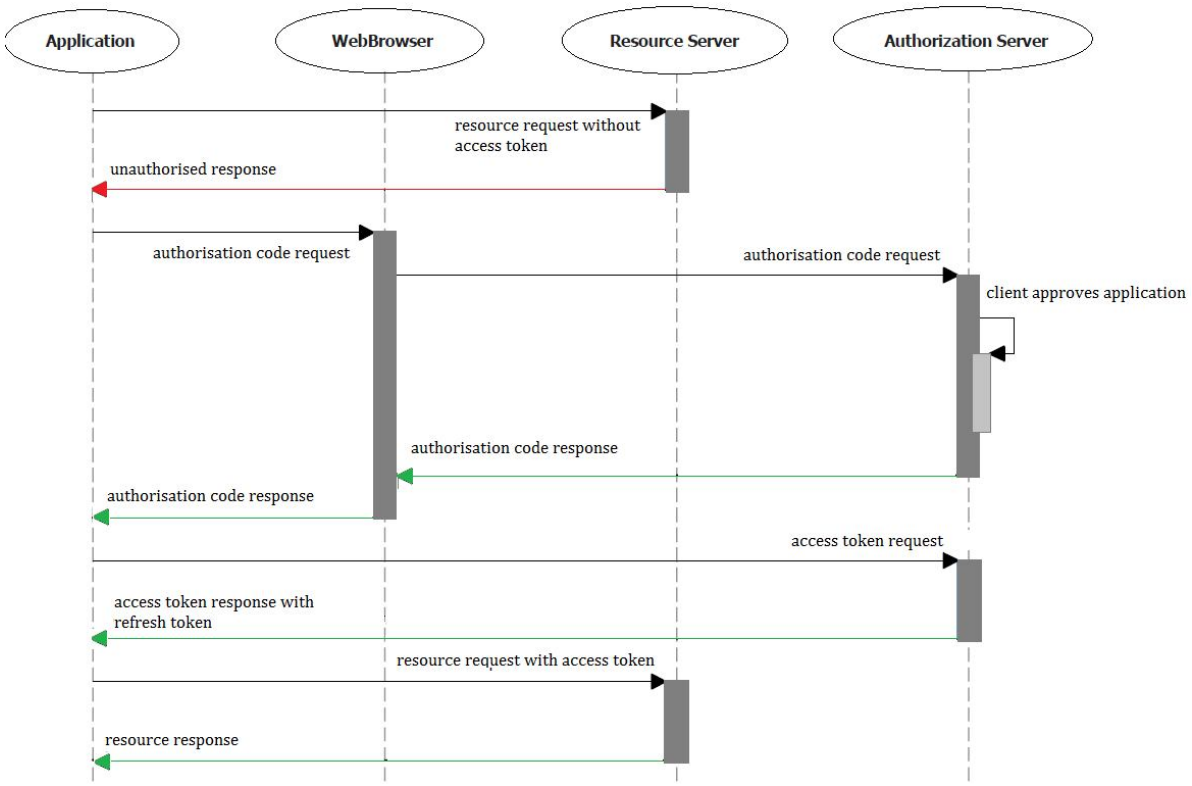
3.3 Authenticatie: OAuth 2.0

OAuth 2.0 is een standaard voor autorisatie waarmee derden toegang kunnen krijgen tot resources zonder dat de gebruiker een username en password hoeft te geven. De 'resource owner' specificeert welke gegevens beschikbaar zijn en hoelang deze benaderd mogen worden. oAuth 2.0 verschilt van de eerste versie van oAuth in het gebruikersgemak en het gebruik van SSL.

oAuth wordt gebruikt in Exact Online om applicaties van derden toegang te geven tot de data van één klant. Deze toegang kan alleen verleend worden door de klant zelf. Deze registreert een nieuwe

[4] http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html
Afbeelding: https://start.exactonline.nl/docs/HlpDocument.aspx

applicatie in zijn/haar omgeving waar onder andere een return url wordt teruggegeven waar de ‘access token’ naar teruggestuurd moet worden. Vervolgens krijgt deze een ‘ClientID’ en een ‘Client Secret’ die in de applicatie die toegang wil hebben tot de gegevens moeten worden geregistreerd. Zodra dit is gebeurd kan de applicatie een authenticatie request doen, en daarna een access token opvragen bij Exact Online (Afbeelding 2). Deze access token wordt vervolgens door de applicatie gebruikt om zichzelf te autoriseren.



Afbeelding 2: Overzicht OAuth Authenticatie Request

Het voordeel van oAuth is dat een gebruiker geen gebruikersnaam en wachtwoord meer hoeft op te geven van zijn/haar omgeving voor het delen van data. Ook kan deze op elk moment besluiten om de rechten van de derde in te trekken. De nadelen van het gebruik van oAuth liggen voornamelijk bij de programmeur. Deze moet nogal wat stappen ondernemen om de autorisatie voor elkaar te krijgen. Ook is oAuth moeilijk toe te passen in applicaties anders dan web applicaties.

4 Huidige Project Management Methode

Scrum is de dominante projectmanagementmethode die wordt gebruikt binnen Exact om software te ontwikkelen. Binnen Scrum wordt gewerkt in sprints van 2 tot 4 weken. Het voordeel hiervan is dat er voor de stakeholders snel duidelijkheid is over de voortgang, er korte lijnen worden gehouden met de stakeholders en teamwork centraal staat[5].

Exact gebruikt sprint van twee weken. Elk van deze sprints wordt voor aanvang met het team gepland in een 'grooming' sessie. In deze sessie wordt door de Product Owner samen met het team bepaald welke user stories (functionaliteiten) uit het sprint backlog (wenslijst) worden opgepakt in de volgende sprint, en wordt er een schatting gedaan van de benodigde tijd uitgedrukt in storypoints. Deze user stories en hun geschatte tijd worden in een sprint backlog geplaatst en worden weergegeven op het scrum board (zie Afbeelding 3). Op dit bord wordt de vooruitgang en taakverdeling getoond.

Afbeelding 3: Scrumboard Connectivity

Het Scrum proces wordt begeleid door de Scrum Master. Binnen het Connectivity team zorgt hij ervoor dat er sprint reviews worden gehouden en Scrum goed wordt uitgevoerd.

Links op het Scrum board staan op de gele stickynotes de userstories. Rechtsboven op een userstory staat het aantal storypoints waarop de user story is geschat, rechtsonder het workitem id voor een koppeling met Team Foundation Service van Microsoft en linksonder staat een letter die de userstory identificeert (A-Z). Rechts naast de user story staan de verschillende taken die moeten gebeuren. Dit zijn meestal development, test, code review, merge en PO check, maar kunnen per user story ook verschillen.

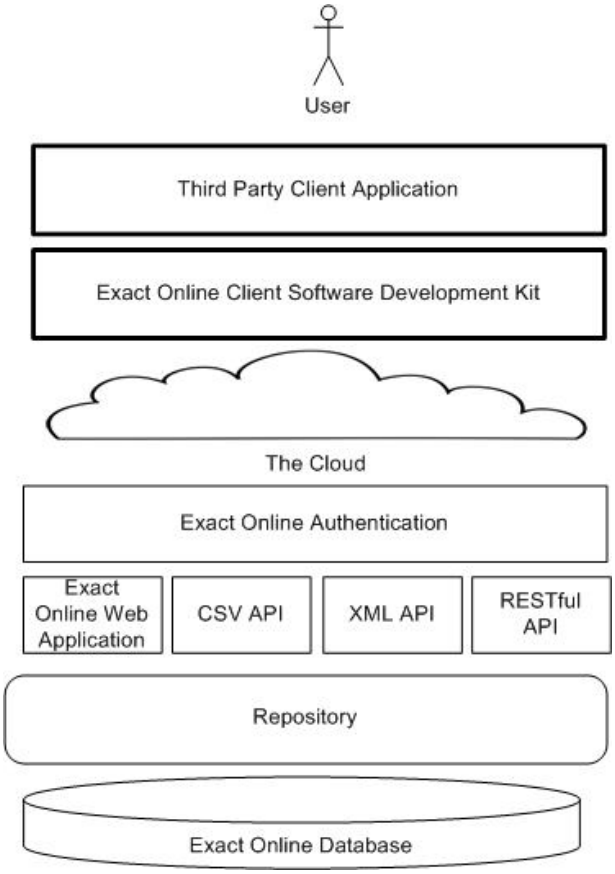
Het Connectivity Team heeft ook een Product Owner. Deze is de verantwoordelijke voor het product en de beheerder van het product backlog (gehele wenslijst). Hij heeft contact met de stakeholders over de eisen en wensen en zorgt ervoor dat het Scrum team deze correct implementeert.

[5] https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf

5 Opdrachtomschrijving

De afstudeeropdracht bestond uit het ontwerpen, bouwen en testen van de Exact Online Client Software Development Kit (SDK). De SDK is een verzameling van hulpmiddelen die het ontwikkelproces voor externe ontwikkelaars versimpelt. In de afstudeeropdracht is alleen een SDK gebouwd voor de taal C#. De onderstaande hoofdstukken beschrijven de verschillende onderdelen van de opdracht.

5.1 Exact Online Software Development Kit



Afbeelding 4: Gewenste Situatie Exact Online Client SDK

De SDK die voor Exact ontwikkeld is, is een verzameling van classes die de ontwikkelaar kan helpen in het doen van aanroepen naar de API voor het ophalen van gegevens uit de web applicatie Exact Online. In Afbeelding 4 wordt de plek van de SDK in de huidige omgeving getoond.

De SDK (in feite ook een API) verschilt van de Exact Online API doordat deze taal specifiek is, en veel stappen die de ontwikkelaar moet doen uit handen neemt of versimpelt. Waar de API alleen requests afhandelt en Json teruggeeft, doet de SDK de requests voor de ontwikkelaar en converteert deze de API response naar een C# object.

Voor Exact is de SDK opgeleverd in de vorm van een dynamic link libary (DLL). Deze kan geïmporteerd worden in een C# of VB.NET ontwikkelomgeving en vervolgens direct gebruikt worden.

De voordelen van een SDK zijn dat de software ontwikkelaar geen kennis hoeft te hebben van de interne werking van de code, maar alleen hoeft te weten welke functionaliteit (methodes en classes) de SDK aanbiedt, en welke waardes deze teruggeeft. Op deze manier kunnen er snel heel complexe programma's worden geschreven. Een nadeel van een SDK is dat er niet gemakkelijk aanpassingen gedaan kunnen worden door derden. Een class kan wel worden uitgebreid, maar omdat de interne werking niet zichtbaar is voor de ontwikkelaar kunnen bijvoorbeeld private methods niet worden gebruikt. De externe ontwikkelaar heeft dus te maken met de beperkingen van de SDK en is afhankelijk van Exact voor het toevoegen van extra functionaliteit. Voorafgaand aan het afstuderen is een opdrachtomschrijving opgesteld. De eisen uit de opdrachtomschrijving zijn:

- De SDK moet worden gebouwd in de programmeertaal C#.
- De mogelijke structuur van de SDK moet worden onderzocht door te kijken naar concurrerende en bewezen SDK's zoals die van Inuit, Facebook, Twitter, etc.

- De SDK moet worden opgeleverd in de vorm van een DLL die door de third party developer geïmporteerd kan worden in zijn/haar C# project.
- De SDK moet kunnen reageren op wijzigingen in de API door een architectuur te ontwerpen die gemakkelijk wijzig- en onder houdbaar is.
- Voor elke functionaliteit moeten er regressie testen (zoals unit- en integratie tests) worden opgeleverd die gebruikt kunnen worden bij verdere ontwikkeling.
- De SDK moet de meest gebruikte scenario's van third party ontwikkelaars ondersteunen
    - Maken van entiteiten
    - Lezen van entiteiten
    - Wijzigen van entiteiten
    - Verwijderen van entiteiten
- De architectuur van de SDK moet geschikt zijn voor gebruik in andere programmeertalen
- Ontwikkelaars ondersteunen in de bèta fase van de SDK

5.2 Verantwoordelijkheden

De student was verantwoordelijk voor de voortgang en de ontwikkeling van de SDK. In de eerste weken heeft hij de planning gemaakt en de user stories voor de eerste sprints opgesteld en geprioriteerd. De juiste software ontwikkelmethode is gekozen, en er is gekeken naar de mogelijkheden en architectuur van bestaande SDK's. Vervolgens is een eerste versie van de architectuur opgesteld en is begonnen aan de ontwikkeling. Over de duur van het project heeft de student de architectuur uitgebreid en aangepast waar nodig en de documentatie mee laten veranderen. Testing was onderdeel van de software ontwikkel methode en is gaandeweg gedaan.

5.3 Planning

In de tabel hieronder staat de planning zoals die is opgesteld in het plan van aanpak weergegeven. De beschrijving van de acht sprints is in één rij beschreven om ruimte te besparen. Per sprint werd een nieuwe versie van de SDK opgeleverd. De indeling van de functionaliteit per sprint werd in overleg met de stakeholders vastgesteld voorafgaand aan de sprint.

Datum	Fase	Opmerking
28-okt 4-nov	Oriëntatiefase	Inwerken bij Exact. Oriënteren op de opdracht & maken van Plan van Aanpak.
11-nov 18-nov	Sprint 0	Detailontwerp, Master Test Plan en Product Backlog met daarin de user stories.
25-nov 17-maart	Sprint 1 t/m 8	Elke twee weken een nieuwe versie van de SDK, na de laatste sprint de oplevering.

Tabel 2: Sprintplanning Project Client SDK

6 Gekozen aanpak

Dit hoofdstuk beschrijft de gekozen aanpak met betrekking tot de gekozen methodes. In de onderstaande hoofdstukken wordt eerst de keuze voor het soort methode beschreven (waterval of iteratief), vervolgens welke specifieke software ontwikkelmethode daarbij is gekozen en welke testmethode is gekozen. Bij elke keuze wordt de achterliggende redenering beschreven.

6.1 Keuze Soort Ontwikkelmethode

Binnen Exact wordt al jaren de agile software ontwikkelmethode Scrum gebruikt. In de opdracht omschrijving stond aangegeven dat hiervan afgeweken mag worden mits hier noodzaak voor was. Om die reden zijn de andere soorten ontwikkelmethodes ook in overweging genomen. Deze worden beschreven in de onderstaande hoofdstukken.

6.1.1 Spiraal model

Het spiraal model stelt een product in staat om te groeien over de loop van het project. Eerst wordt een concept van de requirements gemaakt, een concept van de werkzaamheden, een requirements plan en vervolgens een eerste prototype. Deze kan geverifieerd en gevalideerd worden. Vervolgens wordt een ontwikkeling plan opgesteld, een 2^{e} prototype gemaakt en de stappen opnieuw doorlopen[6]. Het prototype wordt gebruikt om de requirements te specifiëren. Nadat er een operationeel prototype is opgeleverd wordt een detail ontwerp gemaakt, en de stappen doorlopen om het uiteindelijke product te kunnen releasen.

Het spiraal model is niet gekozen omdat de SDK in delen opgeleverd kan worden waardoor er sneller een werkbaar product is. Ook zijn de requirements vooraf niet volledig bekend, en is het specifiëren van de requirements dus een onmogelijke taak.

6.1.2 Waterval Methode

De watervalmethode heeft de volgende voordelen[7]:

- Alle eisen en wensen zijn aan het begin van het project bekend en gedetailleerd beschreven
- De te nemen stappen/fases zijn vooraf bekend
- Er hoeft niet nagedacht te worden over keuzes uit vorige fases

Omdat voor de Software Development Kit niet alle eisen vooraf bekend zijn.

Een voordeel van de waterval methode is dat alleen aan het begin van het project over eisen en wensen wordt nagedacht en hier goede afspraken over worden gemaakt. Dit is echter ook een groot nadeel van waterval omdat er moeilijk wijzigingen kunnen worden gedaan in het ontwerp. Ook kost het bij de watervalmethode veel tijd voordat er een werkbaar product kan worden getoond omdat er eerst een uitgebreide requirements analyse gedaan moet worden, worden ontwikkeld en worden getest voordat er iets opgeleverd kan worden. Hierdoor ziet de eindgebruiker het product pas op een later stadium van het project waardoor eventuele fouten die zijn ontstaan door miscommunicatie of verkeerde

[6] http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/
[7] http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/

interpretatie van de requirements moeilijker worden hersteld. Het testen van het product gebeurd ook pas in een late fase van het project waardoor dit proces erg tijdsintensief is (het hele product moet immers in één keer getest worden).

6.1.3 Iteratieve Methode

Bij een iteratieve methode is de planning minder strikt. Als er iteratief gewerkt wordt kunnen stappen worden herhaald en kunnen foute aannames snel worden hersteld. De opdrachtgever kan eens in de 2 tot 4 weken een werkbare versie van het product testen, en de communicatie met de klant of eindgebruiker is over het algemeen veel beter dan bij de watervalmethode omdat dit over de gehele duur van het project gebeurd.

6.1.4 Conclusie & Keuze

Uiteindelijk is gekozen voor een iteratieve aanpak. Dit omdat er binnen de organisatie waarde wordt gehecht aan communicatie met de eindgebruiker of productmanager zodat het product helemaal aansluit bij hun eisen en wensen. In een watervalmethode is de kans op een product wat dit niet doet een stuk groter dan bij een iteratieve aanpak omdat er alleen in het begin van het project contact is. Ook is de Software Development Kit een product wat goed gebruikt zou kunnen worden in een vroeg stadium van ontwikkeling en kan groeien in functionaliteit over de duur van het project. Het regelmatig opleveren van een product (onderdeel van iteratief werken) zou dus goed gebruikt kunnen worden in dit project. Een van iteratief afgeleide ontwikkelmethode is agile. Deze zal samen met de iteratieve methodes worden behandeld in de volgende hoofdstukken.

6.2 Keuze specifieke Software Ontwikkel Methode

Nadat de soort ontwikkelmethode was gekozen, moest een software ontwikkelmethode die zowel geschikt was voor het project als voor gebruik binnen de organisatie worden uitgezocht. Hiervoor is eerst een lijst gemaakt met methodes die binnen de categorie agile/iteratief vallen om een overzicht te krijgen van de methodes die toegepast zouden kunnen worden. Een aantal van deze methodes vielen echter meteen af omdat deze niet mogelijk zijn binnen het project. Dit waren:

- **Extreme Programming (XP)**: Het project wordt uitgevoerd door één persoon. XP moet altijd in paren worden gedaan.
- **Dynamic Software Development Method (DSDM)**: viel ook meteen af omdat hier continue samengewerkt moet worden met de eindgebruiker wat niet mogelijk is binnen dit project.
- **Rapid Application Development (RAD)**: Omdat deze veel feedback van de eindgebruiker nodig heeft.

Uit een lijst met software ontwikkelmethoden zijn de volgende drie geselecteerd aan de hand van inzicht in het project en eisen en wensen van de opdrachtgever: Unified Process, Test Driven Development en Scrum. Deze methodes worden in de onderstaande hoofdstukken tegen elkaar afgewogen.

6.2.1 Unified Process

UP heeft als grote voordeel dat ongeveer alles aan een project is uitgedacht (documentatie, fases, gebruikerscontact, testen) en dat deze goed aan te passen is op een specifiek project. Voorafgaand aan het project kunnen de artifacts die noodzakelijk zijn voor het project (bijvoorbeeld een specifiek

diagram) worden gekozen en ingepland. Ook kunnen fases worden herhaald waardoor deze artifacts over het hele project groeien en worden verfijnd.

In eerste instantie viel de keuze op UP, maar in overleg met de Product Owner en Manager Product Management bleek dat ze vonden dat deze methode teveel overhead had en dat de voordelen van deze methode ook in hun huidige software ontwikkel methode terug komen. Uit het gesprek bleek dat er keuze was tussen het gebruik van UP en deze aanpassen om het in de organisatie te laten werken, of het toevoegen van benodigde onderdelen voor het project aan hun huidige software ontwikkelmethode. Doorslaggevende factoren waren:

- Moeilijk kunnen aansluiten bij Connectivity Team
- Moeilijk kunnen aansluiten bij de huidige gang van zaken in ontwikkeling
    - Grooming
    - Sprint Retrospectieve
    - Reviews
    - Sprint Planning
- Nadruk op documentatie binnen UP omdat deze snel verouderd en meestal niet wordt gelezen

Uiteindelijk is dus niet gekozen voor Unified Process.

6.2.2 Test Driven Development

Een eis uit de opdrachtomschrijving (Hoofdstuk 5) was dat er uitvoerbare unittesten/regressietesten gemaakt moeten worden. Test Driven Development (TDD) dwingt het maken van testen af door alle testen te schrijven voorafgaand aan het maken van code. De volgende stappen worden doorlopen:

1. **Testing:** Tests maken die functionaliteit test
2. **Developing:** Functionaliteit realiseren
3. **Test Run:** Tests draaien
4. **Refactor:** Indien test slaagt: Refactor fase

Binnen de organisatie is het afgelopen jaar druk gewerkt aan het maken van regressietesten en het verbeteren van de code coverage. Langzamerhand worden steeds meer projecten (zoals het App Centre) gebouwd volgens TDD. Een nadeel van TDD is dat dit een informele methode is en over veel onderdelen (bijvoorbeeld planning) niet goed is nagedacht. TDD is goed voor een aantal dingen maar is opzichzelfstaand niet geschikt voor de ontwikkeling van de SDK.

6.2.3 Scrum

Scrum is een agile methode waarin met sprints wordt gewerkt. Binnen Exact wordt gewerkt in sprints van 2 weken. Alle eisen en wensen (user stories) en hun acceptatie criteria (de eisen waaraan een user story moet voldoen om goedgekeurd te worden) worden voorafgaand aan een sprint op volgorde van prioriteit ingepland. In de sprint worden per user story de volgende stappen doorlopen:

1. **Development**: Ontwikkelen van de code
2. **Testing**: Schrijven van automatische testen die de oplossing testen en een manier bieden om in de toekomst snel te kunnen zien of een oplossing nog werkt.

3. **Code Review**: De gemaakte code wordt gecontroleerd door een andere ontwikkelaar.
4. **Product Owner Check**: De gemaakte functionaliteit wordt gecontroleerd door de product owner. Deze kijkt of de functionaliteit daadwerkelijk doet wat deze moet doen.
5. **Merge door merge manager**: De merge manager voegt de code toe aan de development branche en controleert op eventuele conflicten.

Scrum heeft dezelfde voordelen als vele andere iteratieve methodes. Een bijkomend voordeel is dat binnen Scrum de nadruk wordt gelegd op teamwork en communicatie door middel van dagelijkse meetings, en wordt de voortgang zichtbaar gemaakt door middel van een Scrumboard waarop alle user stories (taken) staan en de voortgang wordt weergegeven. Zo is iedereen van het team altijd op de hoogte van de voortgang en de knelpunten. Een nadeel van Scrum is dat er standaard de nadruk ligt op zo min mogelijk overhead en er eigenlijk geen documentatie gemaakt moet worden. Het maken van documentatie kan echter wel worden toegevoegd als userstory aan de product backlog mocht hier behoefte aan zijn.

6.2.4 Conclusie & Keuze

Uiteindelijk is er geen keuze gemaakt tussen Scrum of Test Driven Development (TDD). TDD kan goed toegepast worden als software ontwikkelmethode binnen Scrum (wat eigenlijk meer dient als raamwerk voor het beheren van softwareontwikkeling). Alle onderdelen van Scrum zullen dus worden toegepast, maar de code zal worden ontwikkeld volgens TDD.

Een nadeel van Scrum in dit project is dat het niet volledig correct kan worden toegepast. Scrum is bedoeld voor multidisciplinaire teams die werken aan één product. Voor dit project zal één ontwikkelaar werken aan een apart product, maar wel onderdeel uitmaken van een Scrumteam (Connectivity Team). Deze keuze is gemaakt omdat de SDK en API erg nauw verbonden zijn, en omdat het Connectivity Team een groot belang heeft bij de SDK en hier inspraak in moet hebben. De Product Owner van het Connectivity team zal ook niet volledig zijn rol aannemen voor het SDK project omdat dit een technischere achtergrond heeft, en daardoor meer technische user stories heeft. De rol van product owner zal daarom gedeeltelijk worden overgenomen door een ontwikkelaar van het Connectivity Team.

6.3 Keuze Testmethode

Omdat gekozen is voor een agile software ontwikkelmethode moest een testmethode worden gevonden die toegepast kon worden in een agile project. Verscheidende bronnen zijn geraadpleegd om hier een beeld van te krijgen. Waar het in de grote lijnen op neer komt is dat testing in de sprint gebeurd, en dat dit zowel door een Quality Engineer (QA) als een ontwikkelaar gedaan kan worden[8]. Een Scrum team bestaat over het algemeen uit een verzameling van personen met elk hun eigen expertise, en hoewel bij Scrum het werk onderling gemakkelijk uitgewisseld kan worden, blijven taken als testen veelal toch bij de QA liggen.

Bij testen in een agile project worden vooraf geen (uitgebreide) testplannen geschreven omdat het project ‘lean’ moet blijven. Per sprint wordt gekeken naar de functionaliteit en hoe deze getest kan

[8] Testing in Agile with TMap NEXT http://www.scribd.com/doc/166653157/Testing-Agile-With-TMap-NEXT

worden[9]. Vaak gebeurd dit pas nadat de code geschreven is, en de ontwikkelaar tijdens het proces al meerdere keren, zonder een test plan en/of testontwerp techniek, getest heeft of de code/module waaraan gewerkt werd het deed. Omdat dit dubbel werk is zal in het volgende hoofdstuk Test Driven Development in overweging worden genomen.

6.3.1 Testen tijdens ontwikkeling

Binnen de organisatie is er behoefte om het testproces te veranderen om een betere testdekking (hoeveelheid code die getest wordt) te krijgen en een goed beeld te krijgen van de risico's van verschillende onderdelen. De testdekking zou groter worden door gebruik te maken van Test Driven Development (TDD) omdat hier een testgeval gemaakt moet worden voordat een stuk code geschreven wordt. Ook de code die geschreven zou worden zou van betere kwaliteit zijn, en minder tijd kosten om te realiseren. Omdat er een betere manier gevonden moest worden om code te schrijven is in overleg met het management besloten om deze methode te gebruiken binnen Scrum om te zien of en hoe dit in de toekomst toegepast zou kunnen worden binnen de organisatie.

6.3.2 Testmethode

Een probleem binnen de organisatie blijft echter dat de testen voor een groot gedeelte gemaakt worden uit de losse pols. Dit wordt gedaan door ervaren developers en quality engineers waardoor de kwaliteit van de test code in de meeste gevallen goed is, maar er kan niet met zekerheid worden gezegd of dit daadwerkelijk ook zo is. Het kan voorkomen dat, omdat er geen standaard methode of test ontwerp techniek wordt gebruikt om te testen, de testgevallen niet geschikt zijn het specifieke deel code of dat er onderdelen van de software wordt overgeslagen.

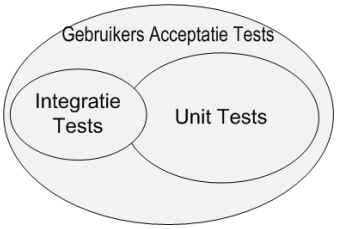
Als testmethode is daarom in eerste instantie gekozen voor TestGoal. Deze bleek echter niet voldoende te zijn als referentie omdat deze bepaalde onderdelen (zoals agile testen, en het maken van een testrisico analyse) niet goed beschreef. Ook werd vanuit de Quality Engineer van Exact opgemerkt dat een testmethode voornamelijk handig is voor systemen die niet geautomatiseerd getest kunnen worden omdat deze testen dan in één keer doorlopen worden en met zekerheid gesteld moet worden dat de testdekking voldoende is. Dit omdat je handmatige testen het liefst zo min mogelijk wil herhalen.

Voor de SDK is voor een op maat gemaakte oplossing gekozen. Alle tests kunnen geautomatiseerd worden gedraaid en zijn onderverdeeld in de volgende drie categorieën:

- **Gebruikers Acceptatie Tests**: Testen van de user story/functionaliteit
- **Integratie Tests**: Voor testen van de communicatie tussen SDK en de Exact Online API
- **Unit Test**: Voor het testen van interne werking van één specifieke module

Per testcategorie zijn er testen geschreven die minimaal één correct scenario, en één incorrect scenario testen. Deze scenario's zijn opgesteld aan de hand van de acceptatie criteria van elke user story. Er is niet gebruik gemaakt van een specifieke test ontwerp techniek voor het opstellen van testen. Per methode of functionaliteit is gekeken naar de invoer en mogelijke uitvoer voor het opstellen van de test.

[9] Google Tech Talks: Agile Testing http://www.youtube.com/watch?v=bqrOnIECCSg



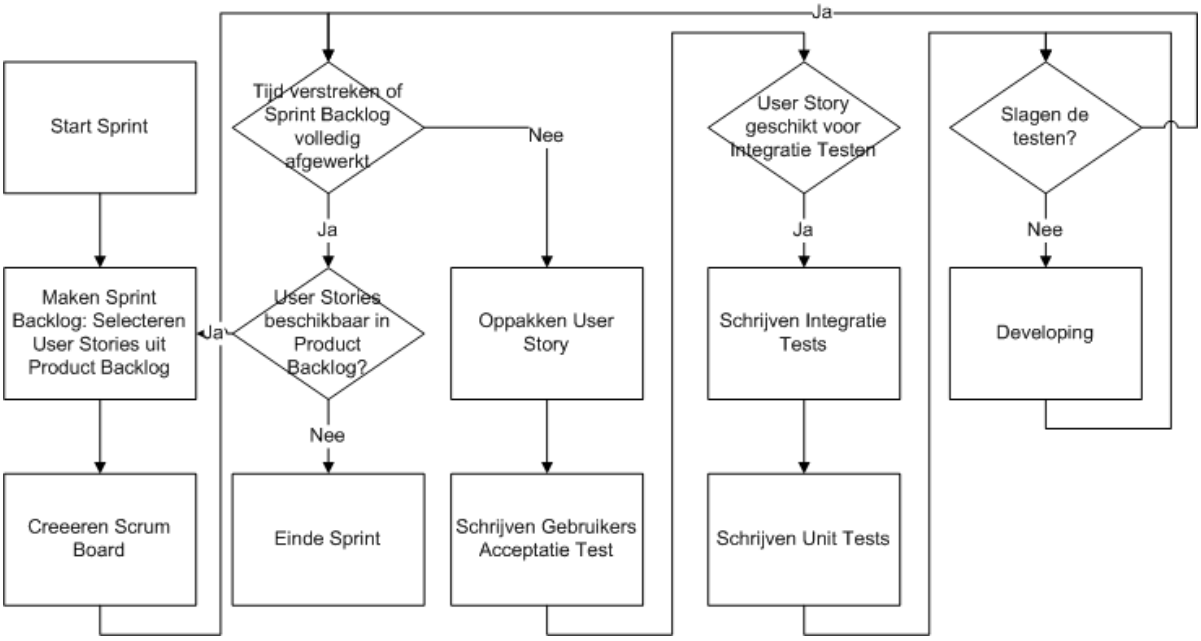
Afbeelding 5: Model overlap testsoorten

Ondanks dat er met Test Driven Development werd gewerkt is er niet voor elke methode een test geschreven. Dit omdat dit niet nodig was voor kleine niet complexe methodes en omdat er tussen de drie test categorieën soms overlap zat. Zo werd de werking van de SDK (een combinatie van de Unit- en Integratie Tests) op een globaal niveau ook getest door de gebruikers acceptatie testen, en werden delen van de onderdelen die eigenlijk door een integratietest getest zouden worden ook door een unit test opgevangen. In Afbeelding 5 is een indicatie van de overlap tussen testsoorten weergegeven.

Uiteindelijk zijn er voor de SDK meer dan 85 unit tests, 17 integratie tests en 21 gebruikers acceptatietests geschreven.

6.4 Ontwikkeling

De ontwikkeling van de SDK aan de hand van Scrum en TDD ging per sprint op de volgende manier (zie Afbeelding 6).



Afbeelding 6: Ontwikkelproces SDK

De user stories zijn geprioriteerd aan de hand van eigen inzicht, feedback van het Connectivity team en de Product Owner en Product Manager. Niet elke user story was geschikt voor integratietesten, dus per user story is gekeken of hier behoefte voor was. Voor de andere user stories is per story gekeken naar de testbaarheid, en de behoefte voor de testdiepte. De actie 'Developing' representeert het codeerproces. Dit gebeurde zoveel mogelijk volgens de volgende principes:

- Schrijf voor hergebruik
    - Losse koppeling: Classes weten zo min mogelijk van elkaar

    - Hoge cohesie: Eén functie of methode doet één ding
- Documentatie in code door middel van comments
- Functies en benaming van de functies zo duidelijk mogelijk
- Leesbaar/Gemakkelijk te begrijpen voor andere ontwikkelaars

7 Werkzaamheden Week 1 & 2 – Oriëntatie

Week één en twee stonden in het teken van de oriëntatie, kennismaking en hadden als resultaat een eerste versie van het plan van aanpak. De eerste dagen is gewerkt aan het eigen maken van de kennis over Exact door middel van zelfstudies die verplicht zijn voor alle nieuwe medewerkers. Ook is kennis opgedaan over huidige technieken (REST, oData, en oAuth) en zijn gesprekken gevoerd met het management, de bedrijfsmentor en de product owner over het te realiseren product. Het plan van aanpak en de keuzes daarin zijn besproken halverwege de tweede week. Tijdens dit gesprek is gepraat over de eventuele keuze voor UP als software ontwikkel methode (zie hoofdstuk 5). Ook is gekeken naar de SDK van de concurrentie: Inuit om inspiratie op te doen voor de opdracht.

Nadat de juiste software ontwikkelmethode was gekozen moest ook kennis worden opgedaan over testing in agile. Dit is gebeurd aan de hand van TestGoal, en artikelen over agile testing die door een aantal quality engineers waren gerefereerd. Nadat voldoende kennis was opgedaan is een opzet voor het master test plan en het detail ontwerp gemaakt.

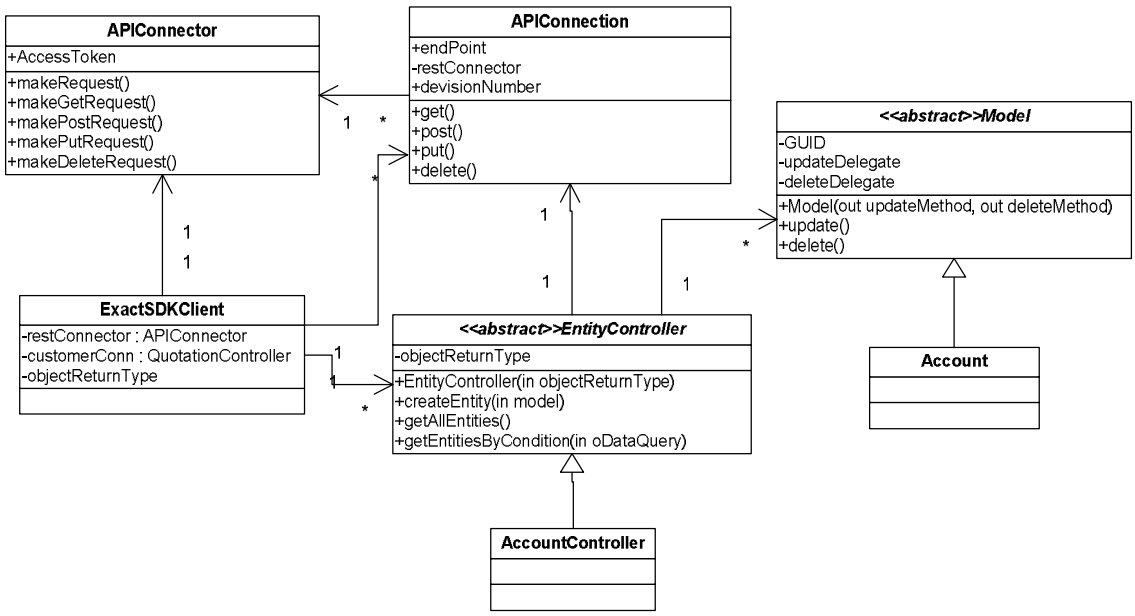
8 Werkzaamheden Week 3 & 4 – Sprint 0

Sprint 0 wordt binnen Exact gebruikt als opstartfase voor een project. In deze print kunnen de voorbereidingen worden getroffen die nodig zijn om het project te starten. In sprint 0 van het SDK project zijn de volgende producten opgeleverd:

- Het plan van aanpak
- Een deel van de user stories
- Acceptatie criteria
- Een eerste versie van het detail ontwerp
- Het master test plan
- Prototype

Om de documenten zo goed mogelijk te maken en om een idee te krijgen van de mogelijke architectuur is gekeken naar de SDK's van Facebook en van Twitter. De Facebook is gebruikt om een kleine web applicatie te maken die alle vrienden ophaalt van iemand die op deze web applicatie inlogt met zijn Facebook ID via oAuth. Het plan van aanpak, de user stories en acceptatie criteria zijn vervolgens met de quality engineer van het Connectivity Team besproken, en aan de hand van de feedback verfijnd.

De eerste versie van het detail design moest zo worden gemaakt dat er de eerste sprint (sprint 1) mee gewerkt kan worden, maar alleen maar noodzakelijke aspecten beschrijft. Binnen Exact wordt namelijk veel waarde gehecht aan lean, en documenten die nooit meer worden gelezen moeten niet worden gemaakt. Om die reden bevatte deze alleen een globaal class diagram met de verschillende classes en hun associaties, en wat attributen en methodes om een idee te geven over hun onderdeel/taken in het geheel (zie het concept class diagram in Afbeelding 7).



Afbeelding 7: Concept Class Diagram

In sprint 0 is verder ook gewerkt aan het prototype. Dit prototype kon alleen nog standaard acties uitvoeren op de API: toevoegen, wijzigingen, verwijderen en updaten van entiteiten van het type GLAccount (grootboek accounts). Deze functionaliteiten van het prototype werden direct getest in de unit tests waardoor ook hier ervaring mee opgedaan is. Het documenteren en programmeren heeft veel invloed gehad op het nog beter begrijpen van de opdracht.

Aan het einde van sprint 0 was genoeg kennis opgedaan, en genoeg gemodelleerd (detail ontwerp) om te kunnen starten met sprint 1. De documenten die in deze sprint zijn opgeleverd kunnen worden teruggevonden in de bijlage.

9 Werkzaamheden Week 5 & 6 – Sprint 1

In het eerste deel van de sprint is veel aandacht besteed aan het detail test plan die per sprint gemaakt moest worden. Door onvolledige en onduidelijke beschrijving in het boek TestGoal over het doen van een Test Risico analyse is hier relatief lang aan gewerkt (4 dagen). Een risicoanalyse wordt voornamelijk gebruikt als er niet voldoende tijd is om te testen om uit te vinden welke onderdelen het best getest kunnen worden in de beschikbare tijd, wat in het geval van Test Driven Development (TDD) met Scrum (de gekozen software ontwikkel methodes) niet het geval is. Door TDD worden alle features getest, en door Scrum worden steeds de belangrijkste functionaliteiten gerealiseerd. Wel is het belangrijk dat wordt gekeken naar de hele testbasis en de risico's ervan, en naar hoe de constraints getest dienen te worden, maar dit hoeft slechts één keer gedaan te worden voor de hele SDK. Het opstellen van een detail ontwerp bleek ook veel overhead te hebben omdat per sprint eerst een analyse gedaan moest worden voor de te testen onderdelen waardoor veel tijd verloren zou gaan. Uiteindelijk is hierom gekozen om het maken van een detail test plan per sprint te laten vervallen. Een reden die ook mee telde was de documenten niet door andere ontwikkelaars gebruikt of gelezen zouden worden.

9.1 Automated Testing

Nadat was begonnen aan de realisatie van de user stories bleek het testen moeilijker te zijn dan gedacht. Werken volgens TDD klinkt in theorie eenvoudig, maar het is toch vrij moeilijk om dit consistent te doen. Als ontwikkelaar heb je de neiging om direct te gaan bouwen, maar om TDD goed toe te passen moet je eerst de tijd nemen om na te denken over hoe je de functie gaat testen, en om de test te bouwen.

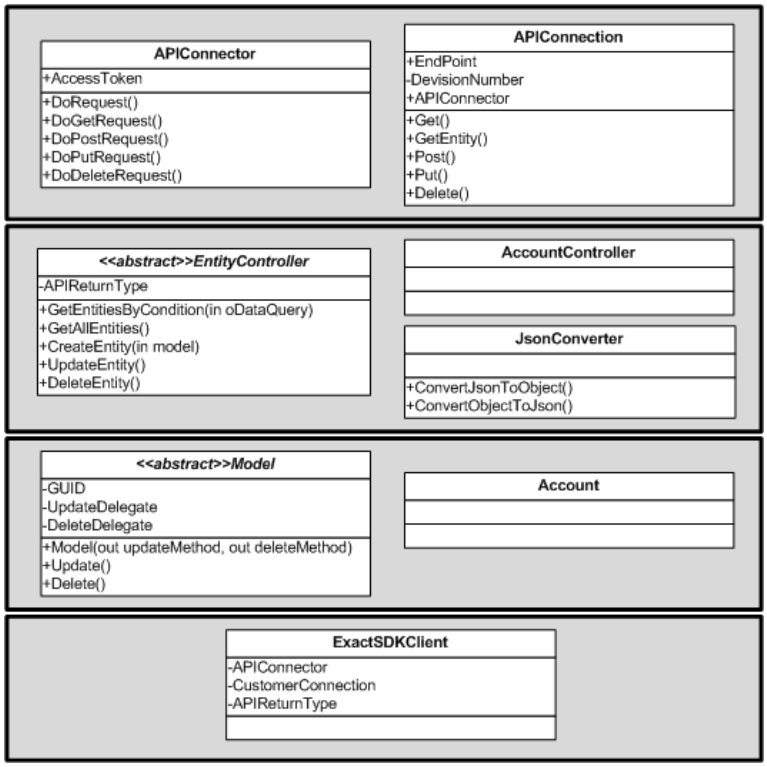
Na een aantal keer per ongeluk toch code schrijven voor dat de testgevallen zijn geschreven is de ontwikkelmethode consistenter toegepast. Al vrij snel waren er veel testen gerealiseerd, maar de verschillende testen hadden veel overlap, en de verkeerde dingen werden getest. Dit kwam omdat in unit testen met de API gewerkt werd, waardoor dit meer een integratie test werd. Ook bleken de gebruikers acceptatietesten moeilijk te realiseren zijn en veel overlap te hebben met de unit en integratietesten. Met hulp van de quality engineer van Exact is geleerd hoe correct unit en integratietesten testen geschreven moeten worden, hoe gewerkt moet worden met mock objecten, en wat de coding guidelines zijn van Exact wat betreft methodenamen voor unittesten. In Tabel 3 staat een voorbeeld van een unit test weergegeven.

```
[TestCategory("Unit Test")]
[TestMethod(), ExpectedException(typeof(ArgumentException))]
public void APIConnectionConstructor_InitializeWithEmptyConnector_Fail()
{
       APIConnection c = new APIConnection(null, "financial/GLAccounts");
}
```

Tabel 3 Voorbeeld Unit Test

De testmethodes zijn als volgt opgebouwd: Naam van de class, de methode, wat deze test doet en de verwachte uitkomst. De tests in deze sprint waren relatief eenvoudig en bestonden uit user- en integratietesten. De gebruikers acceptatietesten waren niet van toepassing omdat nog geen gebruikersfunctionaliteit is gerealiseerd.

9.2 Lagen Exact Online Software Development Kit



Afbeelding 8: Concept Lagendiagram SDK

Met de eerste versie van de SDK is ook het detail ontwerp uitgebreid. In Afbeelding 8 is deze weergegeven. De class 'ExactSDKClient' is de front controller en zal uiteindelijk gebruikt worden door de user om de functionaliteit van de API aan te roepen. De andere classes realiseren de functionaliteit. In sprint 1 is voornamelijk gewerkt aan de APIConnector en APIConnection class die op een laag niveau communiceerde met de API. Het manipuleren (wijzigen, aanmaken en verwijderen) van data is toegevoegd, en ook het ophalen van een collectie van Account entiteiten in Json formaat is gerealiseerd. EntityController en de Model classes worden in een later stadium geïmplementeerd.

Nadat sprint 1 was afgelopen is gekeken naar de hoeveelheid storypoints die gerealiseerd waren. Dit zodat aan de hand hiervan gekeken kon worden hoeveel storypoints er in de volgende sprint gepland konden worden. Dit waren er uiteindelijk een stuk of 8. Voor sprint 2 zijn er 9 gepland.

10 Werkzaamheden Week 7 & 8 – Sprint 2

Aan het begin van sprint 2 is een (Engelstalige) demo gegeven aan het Connectivity Team over de voortgang en de architectuur van de Software Development Kit. Na deze demo was een mogelijkheid tot vragen stellen. In sprint 2 zijn verder de volgende taken uitgevoerd:

- Vertalen van Json naar C# Objecten
- Bedrijfsbezoek begeleidend docent
- Code Review door Connectivity Team

10.1 Vertalen van Json Response uit de API

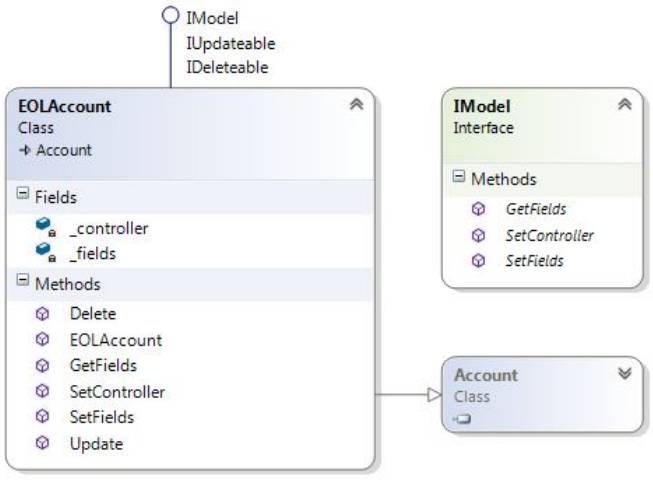
De API retourneert entiteiten in Json formaat, en om het gemakkelijk te maken voor externe partijen om te werken met deze entiteiten moeten deze geconverteerd worden naar C# objecten. Om klein te beginnen, maar niet in een later stadium te struikelen over verborgen functionaliteiten zoals linked entities (entiteiten die een verwijzing hebben naar andere entiteiten (bijvoorbeeld: Account heeft meerdere instanties van BankAccounts), is begonnen met de 'Account' entiteit. Dit is een entiteit die de gegevens van klanten van een bedrijf bevat.

Het 'Model' (de classes die de Exact Online entiteiten representeren) hoefde niet zelf opgebouwd te worden, want deze worden ook al gebruikt om de API te realiseren. Van de bestaande models is een DLL gemaakt die vervolgens geïmporteerd kon worden in de SDK. Voor deserializen van objecten is gebruik gemaakt van de bestaande open source library JSON.NET. Hoewel deze een groot deel van het deserializen op zich kon nemen schoot deze te kort in het werken met datum velden. In de API worden data (datums) in EPOCH vorm (Unix Timestamp) teruggegeven, maar moet deze in Microsoft EDM (Entity Data Model) worden meegegeven. Om dit voor elkaar te krijgen is een subclass geschreven van JsonConverter en is het converteren van data specifiek voor de API toegevoegd.

De Exact Online API retourneert een response die genest is in een 'd' en 'result' tag afhankelijk van de soort response (JsonObject of JsonArray). De JSON.NET libary kon niet omgaan met deze response, dus om de correcte Json aan te leveren is een class geschreven die de response stript van de onnodige tags. Ook bleek deserializen van gekoppelde objecten (bijvoorbeeld de BankAccount objecten die gekoppeld zijn aan Account) niet te kunnen door het formaat waarin de objecten zijn geretourneerd. Omdat de User Story 'Get collection of all account entities in C# objects' eigenlijk uit teveel sub user stories bestond en de doorlooptijd te lang was is besloten om de linked entities niet te realiseren.

10.2 Updaten en Verwijderen van Entiteiten uit Exact Online

Nadat het deserializen van objecten werkte was de volgende stap het kunnen updaten en verwijderen van de entiteiten. Om het voor de gebruiker/ontwikkelaar gemakkelijk te maken om dit te doen was het idee om deze functionaliteit te implementeren door de 'Delete' of 'Update' methode van een entiteit te kunnen aanroepen (weergegeven in Tabel 4 in Hoofdstuk 10.4).



Afbeelding 9: Betrokken classes voor update- en delete functionaliteit

Om dit voor alle entiteiten te implementeren op een elegante manier moest eigenlijk de superclass worden overschreven. Omdat gebruik werd gemaakt van de models uit Exact Online die geen superclass hebben moesten de classes worden uitgebreid met een andere class. Dit kon alleen ook niet omdat de classes al uitgebreid/extended waren. Om ervoor te zorgen dat alle Exact Online Models toch één interface implementeerden is de 'Account' Class overerft en heeft deze overgeërfde class een IModel interface geïmplementeerd (Afbeelding 9). Dit was uiteindelijk ook geen elegante oplossing, want in deze situatie moeten alle models worden overgeërfd wat veel werk en vooral onderhoudswerk is, maar bood wel de flexibiliteit die nodig was om de functionaliteit te realiseren, en zorgde ervoor dat er geen aanpassingen gedaan hoefde te worden aan de models. Met de kennis van toen leek dit de beste aanpak.

10.3 Code Review

In de code review is door de begeleidende software ontwikkelaar van het Connectivity team gekeken naar de opbouw van de SDK en naar hoe gecodeerd werd. Deze was op een goede manier opgezet, maar er miste nog een aantal dingen. Zo moest meer worden gekeken naar het uiteindelijke gebruik van de SDK (dus hoe de externe ontwikkelaars bijvoorbeeld gegevens gaan opvragen) zodat dit op een makkelijke manier gedaan wordt. Tijdens de code review is daarom gezocht naar een manier waarop oData functionaliteit geïmplementeerd kon worden in de SDK. Bij toeval is gestuit op een bestaande oData SDK genaamd 'Simple.OData.Client'. Deze kon heel waarschijnlijk ook gebruik worden voor de Exact API. Indien deze voldoet aan de eisen zou dit de realisatie van de Exact SDK onnodig maken en veel tijd schelen, dus hier moet naar worden gekeken. Deze activiteit staat gepland voor sprint 3.

Het stuiten op een bestaande SDK die het ontwikkelen van de eigen SDK onnodig zou kunnen maken was nooit gebeurd als er research was gedaan naar bestaande oplossingen in de eerste fase van dit project. Dit was toen niet gedaan omdat het geen onderdeel was van de opdracht. Er is wel gekeken naar SDK's van verschillende grote (concurrerende) bedrijven om inspiratie op te doen over de bouw van de Exact SDK. Het zoeken naar bestaande oplossingen moet in de toekomst dus wel worden gedaan.

10.4 Demo Stakeholers SDK

Aan het einde van sprint 2 is een Engelstalige demo gegeven over de voortgang van de realisatie van de SDK aan de stakeholders. Dit waren in totaal ongeveer 20 man waaronder een aantal die ingebeld waren vanuit Amerika. Deze demo was een onderdeel van de sprint review waarin ook het App Centre werd gepresenteerd door het Connectivity Team.

In de demo zijn een aantal scenario's behandeld die door de eindgebruiker veel gebruikt zouden worden: het gemakkelijk werken met API via de APIConnector en APIConnection classes, het ophalen van entiteiten in dynamic objects, en het ophalen wijzigen, verwijderen van Account entiteiten. Een voorbeeld van de laatste functionaliteit staat hieronder weergegeven in Tabel 4.

```
TestObjectsCreator toc = new TestObjectsCreator();// Mock object
APIConnection conn = new APIConnection(toc.APIConnector(), toc.UriCrmAccount());
AccountController ac = new AccountController(conn);

// Get Account from Exact Online to alter
List<IModel> accounts = ac.Get("Name+eq+'Test Name UAT'");
Account testAccount = (Account)accounts[0];
Guid testAccountGUID = testAccount.ID;

// Change description of testAccount
string newName = "Test Name UAT2";
testAccount.Name = newName;
testAccount.Update();
```

Tabel 4: Voorbeeld getoonde functionaliteit demo

Na de tweede sprint is door de demo veel feedback ontvangen van de stakeholders en zijn de taken voor de volgende sprint opgesteld.

11 Werkzaamheden Week 9 & 10 - Sprint 3

Sprint 3 was een relatief korte sprint door de feestdagen. In totaal is er 6 dagen gewerkt, waarvan twee dagen maar 7 uur. In sprint drie zijn er drie user stories gepland:

- Research naar een oData SDK
- Automatisch updaten van een account entity nadat deze is gewijzigd
- Opvragen van een specifieke collectie van entiteiten na het uitvoeren van een oData query

De voortgang en de ontwerpbeslissingen worden hieronder beschreven.

11.1 Research naar de oData SDK

In de vorige sprint is gestuit op een bestaande oData SDK (Simple.OData.Client), met een aantal heel mooie features om data op te halen. Simple.OData.Client bleek zo volledig dat deze eventueel gebruikt zo kunnen worden in plaats van de zelf gemaakte Exact Online SDK wat veel tijd zou kunnen schelen. Om een goede keuze te kunnen maken moest worden gekeken of deze software daadwerkelijk toegepast kan worden op de bestaande API.

Na het testen van Simple.OData.Client bleek echter dat deze SDK erop rekent dat de oData standaard volledig wordt ondersteund. Dit doet de Exact Online API (door toevoeging van extra attributen) niet. Ook is er geen support voor oAuth 2.0, kunnen er geen extra headers worden meegestuurd voor basic authentication, is er weinig documentatie en voorbeelden beschikbaar en moeten de namen van de models exact overeen komen om deze correct te kunnen mappen. Dit is bij veel entiteiten niet het geval. Uiteindelijk is daarom besloten niet gebruikt te maken van Simple.OData.Client en gekozen om de mooie features van de oData SDK te gebruiken als voorbeeld voor de Exact Online SDK.

11.2 Updaten van entiteiten

Het kunnen updaten van een entiteit was al mogelijk, maar om een stap voor de gebruiker/ontwikkelaar weg te nemen zou het handig zijn als dit automatisch gebeurt nadat deze een entiteit wijzigt. De user story die hiervoor gemaakt is hield in dat na een wijziging van een veld, de controller de entiteit automatisch binnen 5 seconden update. Het automatisch updaten van entiteiten zou ook inhouden dat een entiteit bewust zou worden van het veld dat gewijzigd wordt. De API waar de SDK mee communiceert, kan omgaan met delen van entiteiten die worden opgestuurd zodat bandbreedte wordt bespaard, dus deze functionaliteit zou ook een toename in prestaties betekenen.

Na gekeken te hebben naar mogelijke oplossingen bleek er geen goede oplossing beschikbaar. Om te kunnen weten welke velden veranderen moet per veld een eventhandler worden toegevoegd wat voor de verschillende models van Exact Online veel werk zou zijn. Een andere mogelijke oplossing is dat een entiteit geserialized zou worden en dat de originele waarde hiervan wordt vergeleken met de huidige waarde. Ook hiervoor geldt helaas dat dit wegens de hoeveelheid entiteiten en velden die deze entiteiten hebben teveel werk zou zijn voor alle models. Uiteindelijk is gekozen om de user story uit te stellen en aan een user story met hogere prioriteit te werken.

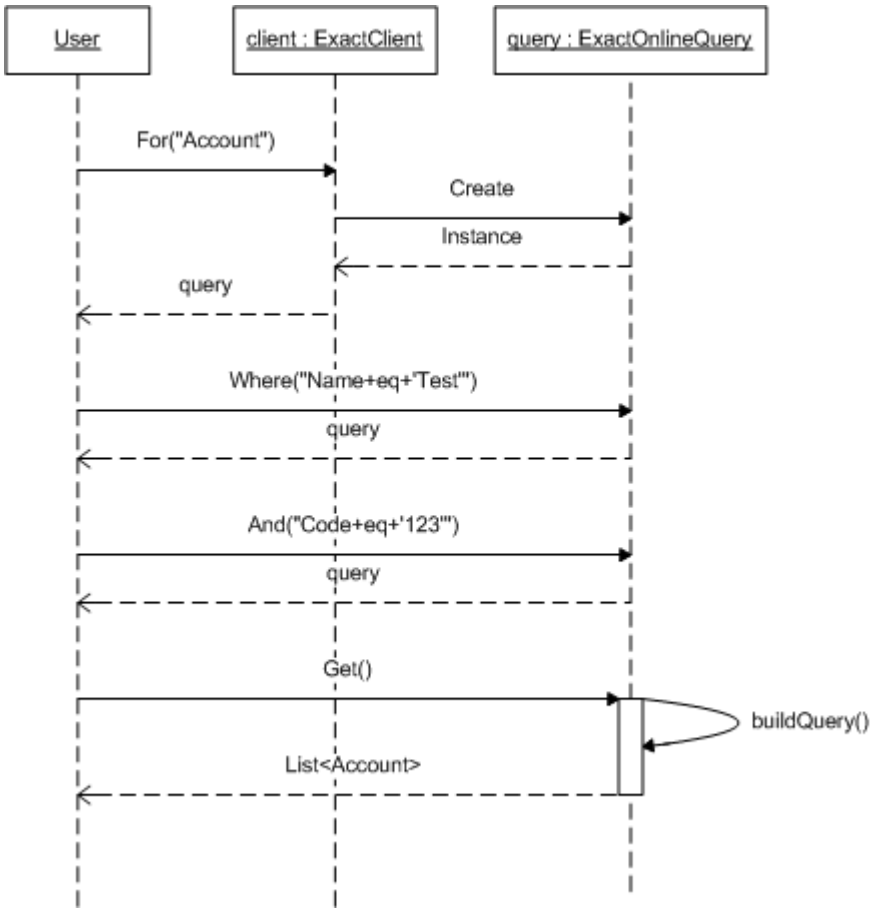
11.3 Uitvoeren van een oData Query

In hoofdstuk 11.1 staat beschreven dat voor deze functionaliteit gekeken is naar de oData SDK. Deze heeft een gebruiksvriendelijke manier voor het ophalen van entiteiten en het uitvoeren van filters op die entiteiten (Tabel 5).

```
var accounts = client.For("Account")
               .Where("Name+eq+'Test Testname'")
               .And("Code+eq+'123'")
               .Get();
```

Tabel 5: Voorbeeld uitvoeren filters

De bovenstaande functionaliteit is geïmplementeerd met behulp van de nieuwe helper class 'ExactOnlineQuery' en de front controller 'ExactClient'. Zie het sequentie diagram (Afbeelding 10) hieronder voor interne werking.



Afbeelding 10: Sequentiediagram opbouwen query

De afbeelding geeft de werking weer zonder het tonen van hoe de juiste controller bij de instantie van ExactOnlineQuery wordt gevonden en hoe de instantie van ExactOnlineQuery met deze controller communiceert.

De methode 'And()' kan meerdere keren worden uitgevoerd om een query op te bouwen. De methode 'Where()' maar één keer, en moet worden uitgevoerd voordat er een 'And()' wordt uitgevoerd.

Voor de ExactOnlineQuery class zijn enkel integratie- en gebruikersacceptatietesten gemaakt. Er is functionaliteit die door middel van unit tests getest zou kunnen worden, maar deze is private, en wordt indirect ook getest in de andere testsoorten. Ook is het risico van deze functionaliteit relatief laag omdat dit slechts het opbouwen van een string is.

Na deze sprint was er, door te kijken naar andere oData SDK's genoeg kennis opgedaan over architectuur om verder te gaan met de ontwikkeling van de Exact Online Client SDK.

12 Werkzaamheden Week 11 & 12 - Sprint 4

In sprint 4 zijn de volgende taken gepland:

- User Stories:
  - Veranderen/Refactoren van de architectuur
    - Werken met generieke types
    - Entity controllers voor het beheren van de state van entiteiten
    - Toevoegen van het Singleton Pattern voor de controllers
  - Realiseren van gekoppelde entiteiten
  - Documentatie up to date krijgen
- Sprint Review Demo voor Stakeholders
- Test Code Review door de quality engineer

Het uitgevoerde werk en de gemaakte keuzes zijn hieronder beschreven.

12.1 Refactoring SDK Architecture

In sprint 3 is ontdekt dat de architectuur niet volledig conform het Model View Controller (MVC) design pattern is, en dat de oplossing om een entiteit over te erven en hier code aan toe te voegen voor update, delete en het registeren van read en write velden geen goede oplossing is. Dit omdat het de onderhoudbaarheid en uitbreidbaarheid van het project niet ten goede komt. In die situatie zou voor elk type entiteit een eigen subclass gemaakt moeten worden. Hier was toen voor gekozen omdat er vanuit werd gegaan dat de models uit de API niet aangepast moeten worden. In sprint 3 is door het Connectivity Team aangegeven dat, indien dit voor de SDK nodig is, er een superclass toegevoegd mocht worden. Hierin kon de globale functionaliteit zoals update en delete geplaatst worden.

Samen met het Connectivity Team is echter een nog betere oplossing bedacht. Als de models aangepast mogen worden kunnen hier ook attributen aan toegevoegd worden die aangeven welke velden schrijfbaar zijn, en wat de primary key/id is. Op deze manier hoeft er geen code toegevoegd te worden aan de models (wat het MVC design pattern in stand houdt) en kan kan de controller generiek gemaakt worden. Deze hoeft immers niet meer af te weten van de lees- en schrijfbaarheid van de velden van een specifieke entiteit. Ook heeft het toevoegen van attributen aan de model het voordeel dat dit handig is voor het Connectivity Team, omdat zij ook moeten weten welk veld lees- en schrijfbaar is. Een voorbeeld van de toegevoegde attributen is terug te vinden in Tabel 6.

```
<DataServiceKey("ID")>
Partial Public Class Account
       Public Property [AddressLine1] As String
       <TypeOfField(FieldType.ReadOnly)>
       Public Property [Created] As Date?
End Class
```

Tabel 6: Voorbeeld attributen voor models

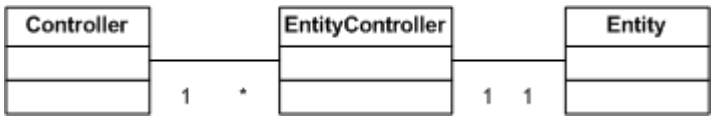
Het attribuut ‘DataServicekey’ geeft aan welk veld van de Account model de ID (soort primary key) is. In dit voorbeeld is het ID veld weggelaten. Om aan te geven dat een veld alleen leesbaar is wordt de attribuut ‘TypeOfField()’ gebruikt. Deze attributen worden door middel van C# reflection (een manier om

de meta-data, eigenschappen en functies van een object te weten at runtime[10]) gebruikt in de Controller, EntityController en ExactOnlineJsonConverter om de lees-, schrijf-, en verwijder functionaliteit te realiseren. Daarnaast zijn de Controller en EntityController C# Generic Gemaakt waarmee ze geen vast type hebben. Hierdoor is er maar één Controller en één EntityController class nodig om voor alle soorten Controllers en EntityControllers objecten te generen. Dit kwam de onderhoudbaarheid van de code ten goede.

12.2 Entity Controller

In de laatste sprint review is feedback ontvangen over de implementatie van de REST principes. In de demo werd bij het updaten van een entiteit de hele entiteit naar de API opgestuurd (inclusief de niet gewijzigde velden) wat niet conform REST is en bandbreedte kost.

Na het refactoren van de architectuur, het toepassen van generieke types en reflection was ook de mogelijkheid ontstaan om de state (de huidige combinatie van velden en hun waardes) van een entiteit op te slaan.



Afbeelding 11: Nieuwe architectuur Controller

Hoewel deze functionaliteit nog niet volledig is gerealiseerd, is wel de architectuur opgezet zodat dit in de toekomst gedaan kan worden. Een controller gaat in deze situatie niet meer over het updaten en deleten van een entiteit, maar dit wordt gedaan door een entity controller. De controller beheert nu alleen de verzameling van entity controllers die één entiteit beheren (Afbeelding 11)

Dit heeft de volgende voordelen:

- Beter overzicht en onderhoudbaarheid: Een controller bevat minder logica
- Betere scheiding van functionaliteit: De logica is beter verdeeld over de verschillende classes
- De update en delete functionaliteit wordt nu geheel opgevangen door de controller in plaats van het model zoals ook beschreven in hoofdstuk 12.1
- De state van een entiteit kan worden bewaard waardoor achterhaald kan worden welke velden aangepast zijn

Na het toevoegen van de EntityController wordt bij het ophalen van entiteiten uit Exact Online per entiteit een instantie gemaakt van EntityController die de state van de entiteit beheert.

12.3 Singleton Pattern

Met het toevoegen van de Entity Controllers moet zeker worden gesteld dat er maar één instantie van de EntityController class bestaat per entity. Als dit niet zo is dan kan dit negatieve effecten hebben op de performance en kunnen er fouten optreden na het updaten of deleten van een entiteit. Na de refactoring zijn de controllers generiek, en is er geen standaard manier om te garanderen dat er maar één instantie is van een bepaald type controller. Om dit te realiseren is de class 'ControllerSingleton'

[10] http://msdn.microsoft.com/en-us/library/ms173183.aspx

toegevoegd. Deze class is zelf een Singleton (een class waar maar één instantie van mag bestaan), en is bedoelt om instanties van Controllers van een bepaald type terug te geven.

12.4 Conclusie

Deze sprint is voornamelijk gewerkt aan het refactoren van de code. Dit bleek uiteindelijk een erg grote user story met meer storypoints dan verwacht. Omdat het belangrijk was dat de architectuur goed was voordat verder gegaan wordt aan het uitbreiden van de functionaliteit, heeft het refactoren voorrang gekregen op de overige user stories.

13 Werkzaamheden Week 13 & 14 - Sprint 5

In sprint 5 waren drie user stories gepland:

- Realisering van koppeling tussen entiteiten
- Updaten van documentatie met nieuwe ontwerpbeslissingen
- Manier vinden om aan te geven dat bepaalde entiteiten alleen lees- en/of wijzig baar zijn

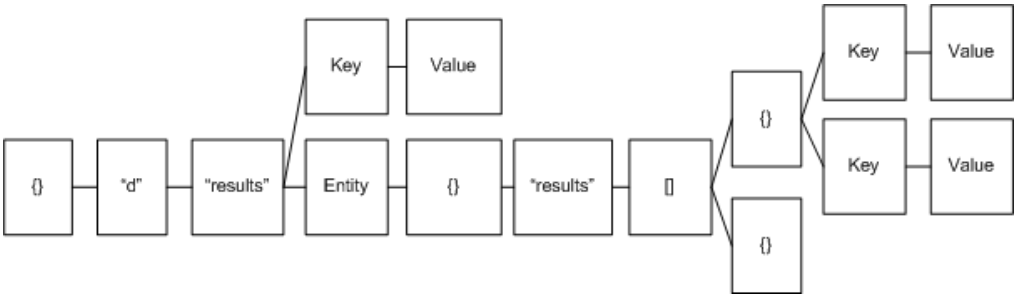
Door onvoorziene omstandigheden stond deze sprint in het teken van gekoppelde entiteiten. Eén entiteit (bijvoorbeeld SalesInvoice) kan een collectie van meerdere entiteiten bevatten (bijvoorbeeld SalesInvoiceLines). In de onderstaande hoofdstukken zullen de ondervonden problemen en gemaakte keuzes worden behandeld.

13.1 Beperkingen JSON.NET

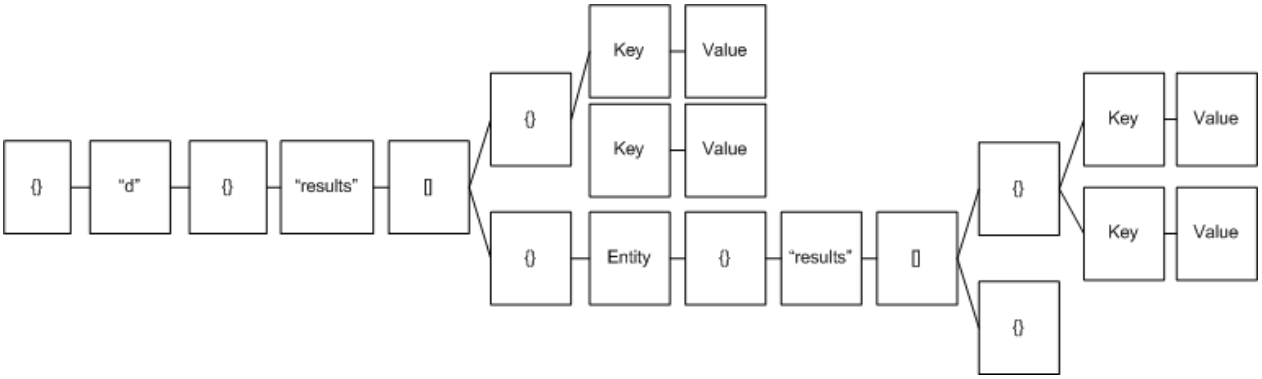
JSON.NET biedt een standaard oplossing voor het vertalen van gekoppelde json objecten naar echte C# objecten. Dit bleek de software alleen niet te ondersteunen voor IQueryable. Omdat de model classes eigenlijk niet aangepast mogen worden was gezocht naar een methode om dit toch mogelijk te maken voor JSON.NET. Eén methode was om de 'JsonConverter' class over te erven en deze implementeren zodat deze wel om kan gaan met dit type collectie. De realisatie van deze functionaliteit bleek echter erg ingewikkeld, en toen dit tijdens een standup meeting naar voren kwam, bleek dat het Connectivity Team ook graag wil overstappen van IQueryable naar IENumerable verzamelingen in de models. Dit mocht dus veranderd worden, waarna verder gewerkt kon worden aan de gekoppelde entiteiten functionaliteit.

13.2 Opschonen van de API Response met APIResponseStripper

De API kan twee soorten responses terug geven: een enkele entiteit (Afbeelding 12) of een collectie van entiteiten (Afbeelding 13). Deze responses bevatten naast een Json object of een Json array ook andere tags zoals "d" en "results". Om de response voor het JSON.NET framework bruikbaar te maken moest er code worden geschreven die de response opschoont.



Afbeelding 12: Enkele entiteit

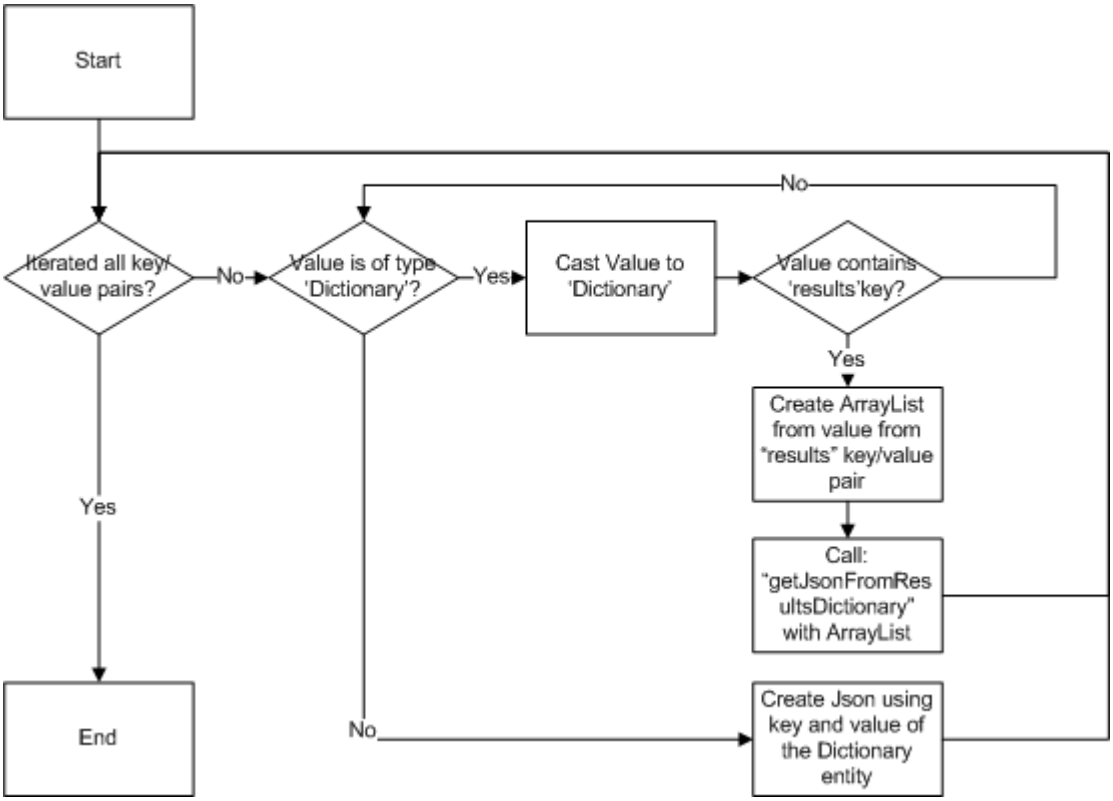


Afbeelding 13: Verzameling entiteiten

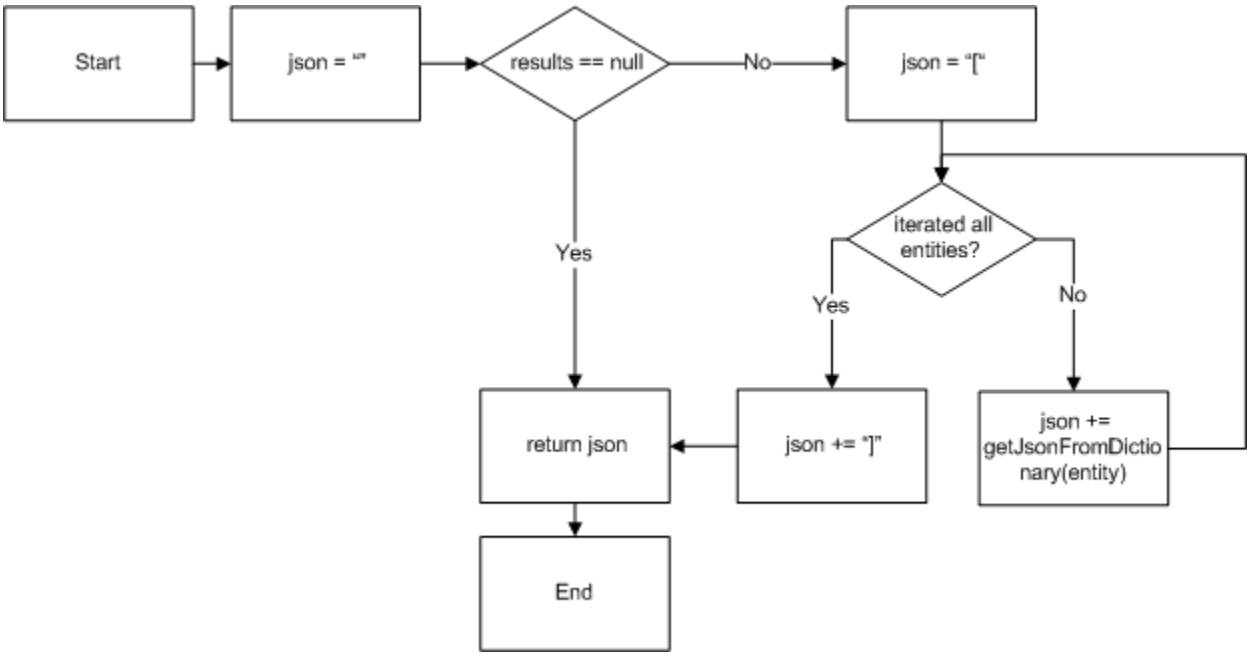
Overbodige tags werden al verwijderd door de eerste versie van APIResponseStripper (nu APIResponseCleaner), maar moeten nu ook om kunnen gaan met overbodige "results" tags in geneste velden (voor linked entities). Om dit voor elkaar te krijgen wordt gebruik gemaakt van de JavaScriptSerializer class. Dit is een onderdeel van het .NET Framework waarmee net zoals met het JSON.NET framework een string omgezet kan worden naar een object. Aan de hand van deze class wordt de string geïnterpreteerd en een 'Dictionary' object opgebouwd.

Dit Dictionary object wordt vervolgens gebruikt om een valide Json string op te bouwen. Dit gebeurt via een complex recursief algoritme. Afhankelijk van het soort response wat wordt verwacht (één entiteit of een collectie van entiteiten) wordt de methode 'GetJsonObject()' of 'GetJsonArray()' van de class APIResponseCleaner aangeroepen.

Binnen die methodes wordt een 'Dictionary' object gemaakt van de API response aan de hand van een overgeërfde JavaScriptSerializer class. Vervolgens wordt (afhankelijk van de soort response) uit dit 'Dictionary' object de inhoud van de "d" en/of "results" tag gehaald en wordt één van de onderstaande methodes aangeroepen. 'GetJsonFromDictionary()' voor een enkel object (zie Afbeelding 14) of 'GetJsonFromResultsDictionary()' voor een collectie (Afbeelding 15).



Afbeelding 14: versimpelde weergave van de methode GetJsonFromDictionary()



Afbeelding 15: versimpelde weergave van de methode GetJsonFromResultsDictionary()

Zoals gezien kan worden in Afbeelding 14 en Afbeelding 15 roepen de verschillende methodes elkaar ook aan afhankelijke van de soort collecties die de key/value pairs in de Dictionary objecten bevatten. Nadat

alle waardes en collecties binnen de Dictionary objecten zijn doorlopen wordt een Json string (zonder de overbodige tags) teruggegeven aan de aanroepende methode. Deze kan vervolgens worden gebruikt om een C# object op te bouwen van het juiste type.

13.3 Bug: DateTime Format met JavaScriptSerializer

De JavaScriptSerializer die gebruikt werd om de API Response te interpreteren en om te zetten naar een Dictionary object bleek de datum velden automatisch te interpreteren. Dit was echter niet de bedoeling en leverde fouten op in de JSON.NET module die de response uiteindelijk moest omzetten naar een Exact Online object. Omdat er geen makkelijke manier was om dit uit te zetten is een eigen versie gemaakt van JavaScriptConverter waarvan een instantie meegegeven kan worden aan de JavaScriptSerializer. Deze converter doet hetzelfde als de standaard JavaScriptConverter, maar verandert de DateTime weer terug naar het EPOCH timestamp.

13.4 Conclusie

Aan het einde van sprint 5 is net de 'Linked Entities' functionaliteit opgeleverd. Helaas zonder create, update en delete functionaliteit. Ook was de bedachte oplossing voor het opschonen van de API response niet ideaal omdat het veel CPU kost. Deze verbeteringen/toekomstige user stories zullen in één van de volgende sprints worden opgepakt.

14 Werkzaamheden Week 15 - Sprint 6

In verband met vakantie is in sprint 6 maar één week gewerkt. De volgende userstories moesten worden gerealiseerd:

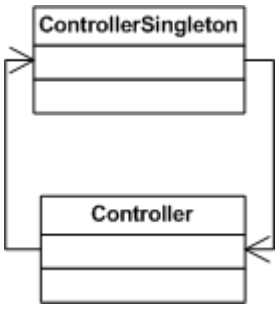
1. Modificeren van gekoppelde entiteiten
2. Realiseren van (school) documentatie
3. Beperkingen voor wijzingen en verwijden van entiteiten die dit niet ondersteunen

Punt twee (realiseren van de (school) documentatie) hield in dat het afstudeerverslag moest worden bijgewerkt zodat deze gereed was voor de voorlopige beoordeling. Dit had veel overlap met het realiseren van de documentatie voor de software ontwikkelaars binnen Exact omdat de interne werking van de SDK wordt uitgelegd. Wegens tijdtekort is echter alleen gewerkt aan de documentatie voor het afstudeerverslag.

Punt drie was een relatief gemakkelijke user story waar attributen zijn toegevoegd aan het model die beschrijven of een entiteit creëer- wijzig- en/of verwijde baar is. Deze user story zal niet worden beschreven in dit hoofdstuk.

14.1 Modificeren van gekoppelde entiteiten

Nadat het opvragen van gekoppelde entiteiten werkte (sprint 5), moesten deze ook gemodificeerd kunnen worden (toevoegen, wijzigen en verwijderen). Om dit te realiseren moeten deze entiteiten, net zoals 'normale' entiteiten worden beheerd door een 'EntityController'. Dit is gedaan met behulp van de methode 'AddEntityToManagedEntitiesCollection' van de Controller class.



Afbeelding 16: Probleem cirkel associatie

Om de design principle losse koppeling door te voeren moest een architecturale beslissing gemaakt worden. Een Controller mocht immers voorheen niets afweten van andere Controller instanties en de ControllerSingleton instantie. Deze informatie is wel nodig om een entiteit te kunnen registreren bij de bijbehorende Controller. Een probleem wat echter optreedt als Controller mag afweten van ControllerSingleton is een cirkel associatie (zie Afbeelding 16).

Uiteindelijk is gekozen om te werken met een delegate die wordt ingesteld door de ControllerSingleton. Door middel van deze delegate kan een Controller instanties ophalen van een ander type Controller zonder dat deze kennis heeft van de ControllerSingleton. Het registeren van deze delegate en het ophalen van de juiste Controller wordt op de volgende manier gedaan binnen de methode 'GetController' van de class ControllerSingleton (Tabel 7).

```
public IEntityManager GetEntityManager(Type type)
{
	MethodInfo method = typeof(ControllerSingleton).GetMethod("GetController");
	MethodInfo genericMethod = method.MakeGenericMethod(new Type[] { type });
	IEntityManager controller = (IEntityManager)genericMethod.Invoke(this, null);
	return controller;
}
```

```
public IController<T> GetController<T>()
{
       string typename = getTypename<T>();
       Controller<T> returncontroller = (Controller<T>)_controllers[typename];

       … Additional Logic …

       returncontroller = new Controller<T>(conn);
       returncontroller.GetManagerForEntity = new ManagerForEntity(GetEntityManager);
}
```

Tabel 7: Voorbeeldcode registeren delegate & ophalen juiste EntityManager

De method 'GetEntityManager' wordt geregistreerd als delegate code door deze toe te wijzen aan de property/delegate 'GetManagerForEntity'. Als de delegate (GetManagerForEntity) wordt aangeroepen wordt in dit geval dus eigenlijk de methode 'GetEntityManager' uitgevoerd.

In het geval van een entiteit van type SalesInvoice die gekoppelde entiteiten heeft van het type SalesInvoiceLine, zal de Controller van type SalesInvoice de delegate 'GetEntityManager' aanroepen met het type SalesInvoiceLine als parameter. Deze roept de methode 'GetController' op de volgende manier aan: GetController<SalesInvoiceLine>() en krijgt de Controller van het type SalesInvoiceLine terug.

15 Werkzaamheden Week 16 & 17 - Sprint 7

In sprint 7 is aan de volgende user stories gewerkt:

1 oAuth Module
2 Documentatie SDK
3 Create Entity with Collection
4 REST Principles for PUT
5 Expired Access Token/Refresh Token Delegate
6 Performance verbeteren/ Async Functionality

Punt 1 is uiteindelijk vervallen wegens versieproblemen van de te gebruiken modules. De user story had een te lage prioriteit om hier veel tijd in te steken. Punt 2 (Documentatie van de SDK) is gemaakt voor de overdracht van de SDK aan de organisatie aan het eind van het afstudeertraject. De documentatie beschrijft de functionaliteiten, en de interne werking van de SDK om deze functionaliteit te realiseren. Punt 5 (Access Token Delegate) is een stuk code wat uitgevoerd kan worden als er een unauthorised exception optreedt. Deze code moet een string waarde teruggeven die de nieuwe access token bevat. De overige user stories worden in de onderstaande hoofdstukken beschreven.

15.1 Create Entity with Collection

Voorheen was het al mogelijk om entiteiten met gekoppelde entiteiten te maken. Dit gebeurde door een entiteit te maken, en hier daarna entiteiten aan te koppelen. Het was alleen nog niet mogelijk om een entiteit met één of meer gekoppelde entiteiten in één keer aan te maken. Voor entiteiten als factuur is het niet mogelijk om eerst een factuur te maken, en daarna pas een factuurregel toe te voegen, en moet de factuur inclusief minimaal één regel samen worden opgestuurd naar de Exact Online API.

Om deze functionaliteit te maken moest er onderscheid gemaakt worden in het genereren van Json code voor het creëren of wijzigen van een linked entity. Bij het creëren hoeft er namelijk in het geval van een factuurregel geen factuur ID worden meegegeven omdat de factuur nog niet bestaat. Dit is gerealiseerd door een nieuw fieldtype toe te voegen aan de models genaamd 'ReferenceField'. In de class 'ExactOnlineJsonConverter' wordt aan de hand van de variabele _JsonForUpdating en de methode 'IsWriteField' een selectie gemaakt van velden waarvoor Json gemaakt moet worden. In Tabel 8 is een versimpelde versie van de code die dit realiseert weergegeven.

```
private Boolean IsWriteField(PropertyInfo pi)
{
	Attribute[] attributes = pi.GetCustomAttributes().Where(x =>
	x.GetType().Equals(typeof(SDKFieldType))).ToArray();

	Boolean returnValue = true;
	foreach (SDKFieldType field in attributes)
	{
		if (
			(field.TypeOfField == FieldType.ReadOnly) ||
			(field.TypeOfField == FieldType.ReferenceField && SkipReferenceField)
		)
		{
			returnValue = false;
```

```
                }
        }

        return returnValue;
}

public override void WriteJson(JsonWriter writer, object value, JsonSerializer serializer)
{
        // Get all key/value pairs of the entity fields
        PropertyInfo[] writeableFields = value.GetType().GetProperties()
                                        .Where(x => IsWriteField(x)).ToArray();

        … Additional Code …

}
```

Tabel 8: Voorbeeldcode schrijfbare velden ophalen

In de methode 'IsWriteField' wordt door middel van een LINQ query alle attributen van het veld van het die het type 'SDKFieldType' zijn opgehaald. Deze geven het type (ReadOnly/ReferenceField) weer. Vervolgens worden deze attributen doorlopen en wordt gecontroleerd of deze 'ReadOnly' of 'ReferenceField' zijn. Dit zijn momenteel de enige veldtypes, maar om fouten in de toekomst te voorkomen (door bijvoorbeeld een nieuw type veld) is deze check toegevoegd. In de methode 'WriteJson' wordt de methode 'IsWriteField' in een LINQ query gebruikt om alle schrijfbare velden terug te geven. Deze worden later gebruikt om de Json op te bouwen.

15.2 REST Principles for PUT

Een groot voordeel van REST is dat er delen van entiteiten opgestuurd kunnen worden bij het aanmaken of wijzigen van een entiteit. Zo hoeven alleen velden te worden opgestuurd waarvan daadwerkelijk de waarde wordt ingesteld of gewijzigd wat bandbreedte bespaart. Deze functionaliteit wordt gerealiseerd door de class EntityController, waar de originele waardes van een entiteit worden bewaard, en de class EntityConverter. Binnen de EntityConverter wordt door gebruik van JSON.NET en de custom converter 'ExactOnlineJsonConverter' de Json van de gewijzigde velden opgebouwd. De methode 'WriteJson' (Tabel 8) bevat ook een statement waar, indien er Json opgebouwd moet worden voor een update, een selectie wordt gemaakt van gewijzigde velden (Tabel 9).

```
// Check if this is an object where updating json have to be created
if (_JsonForUpdating)
{
        writeableFields = writeableFields.Where(x => IsUpdatedField(value, x)).ToArray();
}
```

Tabel 9: Voorbeeldcode schrijven gewijzigde velden

In de methode 'IsUpdatedField' wordt de oude en de nieuwe entiteit met elkaar vergeleken. Door het toevoegen van deze code wordt alleen van de gewijzigde velden Json geschreven en wordt bandbreedte bespaard.

15.3 Async Functionality

De ondersteuning van asynchrone requests (voor GET) komt vanuit de eis om de prestaties van de SDK te verbeteren. Het viel in demo's op dat het ophalen van entiteiten veel tijd koste. Voorheen werd gedacht

dat dit kwam door het vertalen van de API response naar een object, maar nadat de performance van de verschillende onderdelen is gemeten bleek de meeste tijd verloren te gaan met het doen van een aanroep en het wachten op een response. Afhankelijk van de soort applicatie die een third party developer wil maken kan deze eigenschap de bruikbaarheid van de applicatie verminderen. Om die reden is besloten om het asynchroon doen van een aanroep te ondersteunen in de API.

Voor het testen van de asynchrone functionaliteit is een unit test geschreven die 5 keer dezelfde aanroep deed en in totaal 250 account entities ophaalde uit Exact Online. Hoewel de asynchrone functionaliteit deze requests tegelijkertijd uitvoerde en er een zichtbare toename in prestaties gezien kon worden, leverde de nieuwe functionaliteit wel problemen op met de EntityControllers. Er mag maar één instantie van een EntityController zijn per entiteit, maar doordat de entiteiten tegelijkertijd worden geregistreerd leverde dit fouten op. Uiteindelijk is besloten om bij het asynchroon ophalen van entiteiten geen entiteiten te registeren en dit over te laten aan de ontwikkelaar.

16 Werkzaamheden Week 18 & 19 - Sprint 8

In de laatste sprint werd gewerkt naar de oplevering van de Software Development Kit. Bij aanvang zijn meerdere gesprekken gevoerd met de stakeholders over hun eisen en wensen en de mogelijkheden qua functionaliteiten. De volgende dingen waren gepland:

- Developer Documentatie opleveren (Documentatie voor toekomstige ontwikkelaars over de architectuur, en de procedures voor het aanpassen en updaten van de SDK)
- REST Principes toepassen voor PUT actie op Linked Entities: Linked Entities werden nog gezien als volledig nieuwe entiteit tijdens updaten (omdat objecten moeilijk met elkaar vergeleken kunnen worden). Ook voor Linked Entities moesten alleen de gewijzigde velden opgestuurd worden.
- Updaten van alle Model Classes voor gebruik in de SDK: Toevoegen van attributen die zowel het entiteit als de read only velden beschrijven.
- Ondersteunen gebruikers in bèta fase

Naast de bovenstaande punten is ook support geleverd de eerste gebruikers van de SDK (het Connectivity Team). Zij maakte voor hun eigen product (Een AppCentre waar third party developers hun applicaties op kunnen zetten en verkopen) een demo applicatie die gebruik maakt van de SDK. Tijdens de ontwikkeling van deze app hadden zij vragen en vonden ze kleine bugs die opgelost moesten worden. In de onderstaande hoofdstukken worden de belangrijkste onderdelen beschreven.

16.1 Developer Documentatie

Omdat na de afstudeerperiode niet meer gewerkt zal worden voor Exact moet de code goed worden overgedragen. In een gesprek met de Product Owner en het Connectivity Team werden eisen en wensen voor de documentatie opgesteld. De (Engelstalige) developer documentatie moest aan de volgende eisen voldoen:

- Beschrijven van globale architectuur
- Beschrijven van complexe functionaliteiten
- Beschrijven van veel voorkomende scenario's: Toevoegen en wijzigen van entiteiten

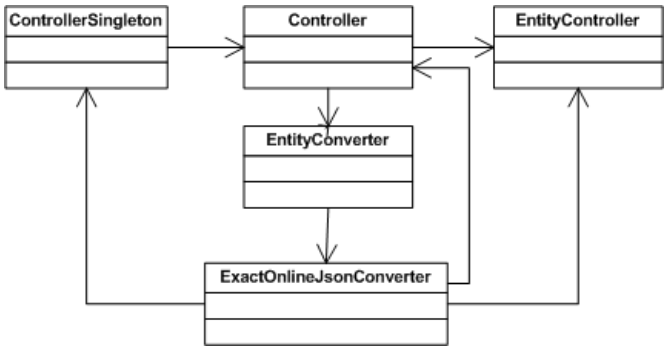
Het verslag is geverifieerd met een collega die beperkte kennis had van de opdracht/SDK. Aan de hand van een vragenlijst met veel voorkomende vragen en scenario's, is aan hem gevraagd om met behulp van de documentatie antwoord te geven op die vragen. Op deze manier kon snel getoetst worden of de documentatie volledig en overzichtelijk genoeg was en kon ook direct een deel van de kennis worden overgebracht. Aan de hand van de feedback is het document uitgebreid. Vervolgens is de documentatie door nog 3 personen (inclusief de product owner) doorgenomen en opgeleverd. De developer documentatie kan teruggevonden worden in de bijlage.

16.2 REST Principles Linked Entities voor PUT

In de vorige sprint zijn de REST principles (het alleen schrijven/opsturen van gewijzigde velden) bij PUT/Update toegepast op entiteiten. Voor gekoppelde entiteiten (Linked Entities) was deze functionaliteit er nog niet. Dit kwam omdat de vorige versie van de code werkte door middel van het

vergelijken van de waardes van velden met die van de originele entiteit. Indien deze verschilden was er sprake van een gewijzigd veld, en werd Json code voor dit veld opgebouwd. Voor gekoppelde entiteiten werd dit nog niet ondersteund omdat dit een lijst was met objecten. Een object uit deze lijst verschilde altijd met de een object uit de originele lijst. Dit omdat als er twee objecten met exact dezelfde waardes in de velden worden vergeleken door middel van een 'Equals' methode, deze altijd verschillen (het zijn immers twee verschillende objecten). Om die reden moest een manier worden bedacht om te controleren of het object/entiteit was gewijzigd. Dit is gedaan door het toevoegen van de methode 'IsUpdated' aan de EntityController class. Deze zou dan op de volgende manier aangeroepen kunnen worden:

1. Doorloop alle gekoppelde entiteiten. Per entiteit:
    a. Haal de juiste EntityController op aan de hand van het ID van de entiteit
    b. Voer 'IsUpdated' methode uit. Indien updated:
        i. Roep ExactOnlineJsonConverter aan om het entiteit naar Json te converteren (recursieve methode)



Afbeelding 17: Cirkel associatie

Om deze functionaliteit te realiseren zou de ExactOnlineJsonConverter moeten afweten van 3 classs: de SingletonController (voor het ophalen van de juiste Controller class), de Controller voor het ophalen van de juiste EntityController, en de EntityController voor het uitvoeren van de IsUpdated functie. Dit wat resulteren in een cirkel associatie en hoge koppeling (zie Afbeelding 17).

Een andere oplossing was het gebruik maken van delegates. In hoofdstuk 14.1 (Modificeren van gekoppelde entiteiten) is beschreven hoe een cirkel associatie wordt ontweken tussen de koppeling Controller en ControllerSingleton met behulp van de delegate 'ManagerForEntity'. Deze werd gebruikt om een instantie die de interface IEntityManager implementeerde terug te krijgen.

Voor het genereren van Json voor Linked Entities in de PUT functie is ook gebruik gemaakt van een delegate waardoor de class ExactOnlineJsonConverter geen kennis meer hoefde te hebben van de andere classs. Deze delegate heet 'GetEntityController' en wordt ingesteld in de Controller class. Omdat de Controller ook niet af mag weten van de ControllerSingleton (zoals beschreven in hoofdstuk 15.2) is ervoor gekozen om de methode die wordt uitgevoerd als de 'GetEntityController' delegate wordt aangeroepen, de 'GetManagerForEntity' delegate aan te laten roepen waarmee de Controller van het gevraagde type wordt opgehaald. De volgende dingen gebeuren:

1. De class ExactOnlineJsonConverter wil de juiste EntityController hebben voor een gegeven entiteit en roept de 'GetEntityController' delegate aan die een EntityController teruggeeft.
2. In de 'GetEntityController' delegate wordt code aangeroepen die staat in de Controller class

3. In de methode in de Controller class wordt de delegate ‘GetManagerForEntity’ aangeroepen waarmee een instantie van een Controller wordt teruggegeven van het benodigde type
4. Vervolgens vraagt deze de benodigde EntityController op aan de Controller uit de vorige stap en retourneert hij deze aan de code die de ‘GetEntityController’ delegate uit stap 1 aanriep.

Door deze ontwerpkeuze hoeven de classs niets af te weten van elkaar, maar enkel te weten welk type de delegate retourneert.

16.3 Beschikbaar stellen van de models

Het originele plan was het delen van de models/classes tussen de API en de SDK. De achterliggende gedachte hierachter was dat dit gunstig zou zijn voor eventuele wijzigingen in de models aan de kant van de API omdat deze dan zonder veel moeite ook snel konden worden toegevoegd aan de SDK. In een laat stadium bleek dat dit voor de SDK toch geen slimme oplossing was. De API heeft veel referenties naar andere Dynamic Link Libraries (dlls) die dan ook moeten worden meegestuurd. Zo zou bijvoorbeeld de Exact.Core dll en de Exact.Web.UI dll moeten worden meegestuurd naar de third party developer. Om dit te voorkomen is gekozen om de models wel te gebruiken, maar deze vooraf te strippen van alle referenties naar andere dlls. Uiteindelijk een script geschreven voor het opbouwen van de model classes die de volgende stappen doorloopt:

1. Verwijderen van overbodige tags
2. Toevoegen SDK Action attribuut: Acties die op een resource/entiteit uitgevoerd kunnen worden
3. Toevoegen van de Read Only Field attributen: Velden Die alleen leesbaar zijn

17 Werkzaamheden Week 20

In de laatste week zijn de puntjes op de i gezet. Maandagochtend is begonnen met de twee wekelijkse sprint review demo waar de gerealiseerde functionaliteit van de laatste sprint is behandeld, en de stakeholders op de hoogte zijn gesteld van de stand van zaken van de SDK en de plannen voor de toekomst. Voor de rest stonden in deze week de volgende punten in de planning:

- Model Creëer script afmaken met nieuwe read-only velden
- Ondersteuning gebruikers in bèta fase
- Tests beter leesbaar maken
- REST voor POST/Create realiseren
- Toevoegen functionaliteiten (oData)
- Compleet maken van Unit Tests
- Documentatie bijwerken
- Workshop
- Oplevering SDK

De workshop van de SDK is gegeven als officiële overdracht van de SDK. In de twee uur durende workshop is de interne werking uitgelegd aan het hele team, en is getoond hoe wijzigingen gedaan kunnen worden aan het model. Ook was er een gelegenheid tot het stellen van vragen. Verder zijn in de laatste dagen heel veel kleine taken opgepakt. Het onderstaande hoofdstuk beschrijft het interessantste aspect van deze week.

17.1 Compleet maken Unit Tests

Wegens grote tijdsdruk is TDD in de laatste sprint niet voor elke user story goed toegepast. Het realiseren van de functionaliteit voor de demo had een hoge prioriteit omdat deze functionaliteit getoond moest worden aan de stakeholders. De volgende tests moesten gemaakt worden:

1 Meesturen van een lege GUID (deze mag niet geschreven worden)
2 Schrijven gewijzigde velden van een linked entity
3 Json voor gewijzigde velden

In Tabel 10 worden de gewijzigde velden getest. De eerste tests, test of de gewijzigde velden worden geschreven. In de tweede tests wordt getest of de niet gewijzigde velden niet worden geschreven, en in de 3^{e} test wordt getest of het gewijzigde veld, wat read-only is, niet wordt geschreven.

```
[TestMethod()]
[TestCategory("Unit Test")]
public void EntityConverter_ConvertObjectToJson_ForAlteredFields_Succeed()
{
       Account newAccount = new Account() { Name = "New Account"};
       EntityConverter ec = new EntityConverter();
       var expected = "{\"Name\":\"New Account\"}";

       var result = ec.ConvertObjectToJson<Account>(new Account(), newAccount, null);
       Assert.AreEqual(expected, result);
}
```

```
[TestMethod()]
[TestCategory("Unit Test")]
public void EntityConverter_ConvertObjectToJson_ForNoAlteredFields_Succeed()
{
	Account oldAccount = new Account() { Name = "New Account" };
	Account newAccount = new Account() { Name = "New Account" };

	EntityConverter ec = new EntityConverter();
	var expected = "";

	var result = ec.ConvertObjectToJson<Account>(oldAccount, newAccount, null);
	Assert.AreEqual(expected, result);
}

[TestMethod()]
[TestCategory("Unit Test")]
public void EntityConverter_ConvertObjectToJson_WithReadonlyFields_Succeed()
{
	Account newAccount = new Account() { AccountManagerHID = 10 };

	EntityConverter ec = new EntityConverter();
	var expected = "";

	var result = ec.ConvertObjectToJson<Account>(new Account(), newAccount, null);
	Assert.AreEqual(expected, result);
}
```

Tabel 10: Test voor gewijzigde velden

Voor het testen van een gewijzigd linked entity veld is de volgende test geschreven (Tabel 11). In deze unit test wordt eerst een object van het type SalesInvoice aangemaakt. Vervolgens wordt hier een SalesInvoiceLine object aan toe gevoegd en beide objecten geregistreerd in een EntityController object. De SalesInvoiceLine wordt gewijzigd en van de SalesInvoice wordt de Json opgevraagd. De methode verwacht dat de opgevraagde Json de waarde heeft die de SalesInvoiceLine gekregen heeft nadat deze geregistreerd was in de EntityController. Als dit zo is slaagt de test.

```
[TestCategory("Unit Test")]
[TestMethod()]
public void EntityConverter_ConvertExistingLinkedObjectToJson_Succeed()
{
	// Create Object
	SalesInvoice newInvoice = new SalesInvoice();
	newInvoice.InvoiceID = new Guid("4f68481a-7a2c-4fbc-a3a0-0c494df3fa0d");

	List<SalesInvoiceLine> invoicelines = new List<SalesInvoiceLine>();
	SalesInvoiceLine newInvoiceLine = new SalesInvoiceLine();
	newInvoiceLine.Description = "NewInvoiceForEntityWithCollection";
	invoicelines.Add(newInvoiceLine);
	newInvoice.SalesInvoiceLines = invoicelines;

	EntityController entityController = new EntityController
		(newInvoice, "ID", "4f68481a-7a2c-4fbc-a3a0-0c494df3fa0d",
		new MockObjects.MAPIConnection(), null);

	// Change description invoiceline
	newInvoiceLine.Description = "ChangedNewInvoiceForEntityWithCollection";
```

```
        EntityConverter ec = new EntityConverter();
        GetEntityController deleg = new GetEntityController(GetEntityController);
        string json = ec.ConvertObjectToJson<SalesInvoice>
               ((SalesInvoice)entityController.OriginalEntity, newInvoice, deleg);

        string expected = "{"SalesInvoiceLines": [{"Description":
        "ChangedNewInvoiceForEntityWithCollection"}]}";

        Assert.AreEqual(expected, json);
}
```

Tabel 11: Testen gewijzigd veld in linked entity

Na de toevoeging van de testen was de testdekking weer goed. Na een wijziging in één van de functies kan nu aan de hand van de regressietesten snel worden gevalideerd of de functies nog correct werken.

17.2 Ondersteunen gebruikers in bèta fase

In de laatste sprint was de eerste gebruiker (een developer uit het Connectivity Team) een aantal keer ondersteund in de werkzaamheden. Voor de sprint review wilde het Connectivity Team een demo app maken die tijdens op de App Centre werd geplaatst. Deze demo app kon wenskaarten maken waarvan de contactgegevens van klanten werd opgehaald uit Exact Online via de SDK en deze wenskaarten opgeslagen konden worden als document in de Exact Online omgeving van de klant.

Tijdens de bèta fase kwamen een aantal dingen naar boven die onduidelijk waren. Dit had voornamelijk te maken met het instellen van de endpoint van de API (de URI waar deze naar moet verwijzen), en de eisen waaraan deze moest voldoen. Ook bleek dat het instellen van de access token niet zo duidelijk was, en werden kleine foutjes gevonden. Tijdens de normale werkzaamheden zijn dus ook bugs geregistreerd en uitleg gegeven over de hoe en wat, wat betreft de SDK. In de tijd die over was zijn deze fouten verbeterd en is ook nog extra functionaliteit (oData functies als top en select) toegevoegd aan de SDK om de gebruiker tegemoet te komen en een volledig project op te leveren.

18 Afwijkingen afstudeerplan

Dit hoofdstuk beschrijft de afwijkingen ten opzichte van het afstudeerplan die voorafgaand aan de afstudeerstage is opgesteld. Dit plan dient ter beoordeling van de afstudeeropdracht, maar wordt in veel gevallen gaande wijs veranderd naarmate de opdracht vordert. Dit hoofdstuk beschrijft de wijzigingen, zodat de examinatoren kunnen beoordelen of de afstudeeropdracht nog steeds aan de eisen voldoet.

18.1 Software Ontwikkel Methode & Ontwerpen Systeemdeel

Voor aanvang van het afstuderen leek de opdracht uitermate geschikt te zijn voor Unified Process (UP) of Test Driven Development (TDD). In de eerst twee weken, die diende als oriëntatie, is een voorstel gedaan om de SDK te ontwikkelen aan de hand van deze methodieken, maar bleek dat de binnen de organisatie weinig vraag is naar zwaar gedocumenteerde systemen. Documentatie kost veel tijd om te maken en veroudert snel. Sinds 2009 wordt gewerkt binnen Exact gewerkt volgens Scrum, dat net als UP ook een iteratieve ontwikkel methode is. In hoofdstuk 6 is de keuze en de beredenering achter de keuze voor de software ontwikkel methode beschreven. TDD kan goed worden toegepast in Scrum. Momenteel worden user stories gemaakt die elk een aantal taken hebben. Dit zijn meestal:

- **Development**: Realiseren van de user story
- **Testing**: Testen van de user story
- **(Code-)Review**: (Laten) Controleren van de code door een andere developer
- **PO Check**: Controleren van de gerealiseerde functionaliteit door de product owner
- **Merging**: Toevoegen van de gerealiseerde code aan de development branche in de source control tool “Team Foundation Services” van Microsoft

Het testen gebeurt op het moment pas na de development van de functionaliteit maar is, voor de ontwikkeling van de SDK, verplaatst naar de eerste taak. Voor een overzicht van het ontwikkelproces zie Afbeelding 6 in hoofdstuk 6.4.

18.2 TestGoal

Testing was een erg belangrijk onderdeel voor de SDK. Om dit goed te kunnen doen was vooraf bedacht dat er daarom een testmethodiek gebruikt zou worden om kwaliteit te garanderen. Omdat de student nog niet veel ervaring had met verschillende testmethodes, en omdat TestGoal gericht is op resultaat (wat binnen de organisatie erg belangrijk is) zou deze methodiek in eerste instantie worden gebruikt. De technieken en oplossingen die TestGoal aandraagt bleken echter niet goed toepasbaar te zijn in een agile/lean software ontwikkelmethode omdat er vooraf veel documenten/ontwerpen gemaakt moeten worden, en er weinig beschreven werd over het testproces in een agile project. Om die reden is hier, na het maken van een mater- en detail test plan, van afgezien. Test Driven Developement, (test)code reviews, en de user stories en acceptatie criteria zijn gebruikt om de kwaliteit te garanderen en fouten te mogelijke fouten te ontdekken en kwaliteit te garanderen. Met hulp van de quality engineer van Exact is bepaald welke testen moeten worden geschreven en hoe gewerkt kan worden met mockobjecten en het unit test framework van Microsoft. Ook is besproken hoe de verdeling van de verschillende testsoorten (unit-, integratie- en gebruikers acceptatietesten) moest zijn en welke testen tenminste geschreven moesten worden om correct te testen.

19 Opgeleverde producten

Product	Datum	Beschrijving
Oriëntatiefase	N.v.t.	Inwerken bij Exact
Plan van Aanpak	08-11-2013	PVA met planning van project
Sprint 0		
Master Test Plan	22-11-2013	Beschrijving van Testproces voor project
Detail Ontwerp	22-11-2013	Beschrijving Testproces binnen sprint 1
Detail Test Plan V0	22-11-2013	Ontwerp van SDK voor sprint 1
Product Backlog	22-11-2013	User Stories voor de SDK
Sprint Backlog Sprint 1	22-11-2013	User Stories voor sprint 1
Sprint 1		
Demo	03-12-2013	Korte demo over voortgang voor team
Exact Online Client SDK V0	06-12-2013	Eerste versie van de SDK: REST requests
Unit Tests SDK V0	06-12-2013	
Integratietests SDK V0	06-12-2013	
User Acceptance Tests SDK V0	06-12-2013	
Sprint 2		
Sprint Backlog Sprint 2	10-12-2013	
Exact Online Client SDK V1	23-12-2013	Werken met Account entities
Unit Tests SDK V1	23-12-2013	
Integratietests SDK V1	23-12-2013	
User Acceptance Tests SDK V1	23-12-2013	
Demo Sprint Review	23-12-2013	Demo voor stakeholders
Sprint 3		
Sprint Backlog Sprint 3	24-12-2013	
Exact Online Client SDK V2	07-01-2013	oData functionaliteit toegevoegd
Unit Tests SDK V2	07-01-2013	
Integratietests SDK V2	07-01-2013	
User Acceptance Tests SDK V2	07-01-2013	
Developer Documentation V1	07-01-2013	Vertaling van detail design naar documentation
Sprint 4		
Sprint Backlog Sprint 4	07-01-2014	
Exact Online Client SDK V3	20-01-2014	
Unit Tests SDK V3	20-01-2014	
Integratietests SDK V3	20-01-2014	
User Acceptance Tests SDK V3	20-01-2014	
Sprint 5		
Sprint Backlog Sprint 5	21-01-2014	
Exact Online Client SDK V4	30-01-2014	Ivm. Vakantie op donderdag
Unit Tests SDK V4	30-01-2014	
Integratietests SDK V4	30-01-2014	
User Acceptance Tests SDK V4	30-01-2014	
Sprint 6		
Sprint Backlog Sprint 6	10-02-2014	
Exact Online Client SDK V5	17-02-2014	
Unit Tests SDK V5	17-02-2014	

Integratietests SDK V5	17-02-2014	
User Acceptance Tests SDK V5	17-02-2014	
Demo Sprint Review	17-02-2014	Demo voor stakeholders
Sprint 7		
Sprint Backlog Sprint 7	17-02-2014	
Exact Online Client SDK V6	03-03-2014	
Unit Tests SDK V6	03-03-2014	
Integratietests SDK V6	03-03-2014	
User Acceptance Tests SDK V6	03-03-2014	
Developer Documentation V1	03-03-2014	Beschrijving scenario's overdracht
Demo Sprint Review	03-03-2014	Demo voor stakeholders
Sprint 8		
Sprint Backlog Sprint 8	03-03-2014	
Exact Online Client SDK V7	17-03-2014	
Unit Tests SDK V7	17-03-2014	
Integratietests SDK V7	17-03-2014	
User Acceptance Tests SDK V7	17-03-2014	
Developer Documentation V2	17-03-2014	Volledige documentatie na review
Demo Sprint Review	17-03-2014	Demo voor stakeholders
Sprint 9		
Sprint Backlog Sprint 8	21-03-2014	
Exact Online Client SDK V8	21-03-2014	Definitieve oplevering
Unit Tests SDK V8	21-03-2014	
Integratietests SDK V8	21-03-2014	
User Acceptance Tests SDK V8	21-03-2014	
Workshop	21-03-2014	Workshop voor Connectivity Team

20 Evaluatie aanpak

Dit hoofdstuk beschrijft de evaluatie van de student met betrekking tot de aanpak van de afstudeeropdracht. In de onderstaande hoofdstukken worden de belangrijkste onderdelen van deze opdracht geëvalueerd.

20.1 Communicatie met stakeholders

De communicatie met de stakeholders gedurende het project was goed. Aan het begin van het project is duidelijk aangegeven wat het plan van aanpak was, en gedurende het project is regelmatig gepraat over de voortgang met het Connectivity Team. Naar het einde van het project toe bleek dat voornamelijk de business stakeholders meer gesproken hadden mogen worden om in een eerder stadium duidelijk te stellen wat er precies opgeleverd kon worden. Deze gesprekken gebeurde ook tijdens het project, maar in informele vorm (bijvoorbeeld direct na een sprint review, een presentatie waar het product werd getoond). Hierdoor werden er niet genoeg concrete afspraken gemaakt en was er sprake van onuitgesproken verwachtingen en aannames aan beide kanten. Deze hebben nooit problemen opgeleverd, maar hadden beter gemanaged kunnen worden.

20.2 Opstellen documentatie

Door het niet vooraf opstellen van gedetailleerde documentatie konden veel verschillende oplossingen worden geprobeerd. Het niet vooraf maken van documentatie gaf geen problemen in ontwerp, maar de behoefte aan diagrammen en tekeningen werd wel beduidend groter bij complexere onderdelen zoals recursieve methoden of het in kaart brengen van de architectuur van de SDK. Ook het communiceren van de architectuur naar collega's werd een stuk makkelijker door het maken van een diagram. Uiteindelijk is daarom een document opgesteld waarin de architectuur van de SDK gedocumenteerd staat (zie hoofdstuk 21.7).

20.3 Scrum voor SDK Project

Scrum was achteraf de juiste methode voor de ontwikkeling van de SDK binnen Exact De nadelen waren alleen dat, doordat er niet in een echt Scrumteam gewerkt werd (het was een aparte, individuele opdracht), een aantal dingen zoals de grooming, planning en retrospective niet 100% correct konden worden doorgevoerd. Ook was een Scrumboard en een stand-up meeting wel handig om de voortgang te laten zien, maar er was maar één persoon met taken op het bord. Wel kon veel worden geleerd van de ontwikkelaars van het Connectivity Team en waren ze zeker betrokken bij de opdracht. De processen zoals grooming, retrospectieve en sprintplanning zijn wel ervaren door mee te kijken met het Connectivity Team.

20.4 Testing & Test Driven Development

Test Driven Development binnen Scrum was een erg goede keuze. Door TDD toe te passen werd hoge kwaliteit code geschreven en werden de design principes beter gehanteerd. De keuze om geen vaste testmethode te gebruiken bleek achteraf geen beperking. Per user story kon aan de hand van de beschrijving en de acceptatie criteria op inzicht van de ontwikkelaar goede regressietesten worden geschreven. Deze testen beschreven zichzelf door duidelijke naamgeving waardoor ze ook duidelijk zijn voor andere ontwikkelaars.

Wat beter had gekund was het consistent volgen van TDD. Als er deadlines waren of een oplossing uitgeprobeerd moest worden werden niet altijd eerst testen geschreven. Dit was dan wel een bewuste keuze, maar hierdoor hadden gaten kunnen vallen in de testdekking omdat het achteraf maken van testen kan worden vergeten. Door het noteren van een to do list van de te maken testen is dit niet gebeurd en zijn de risico's zo klein mogelijk gehouden.

21 Evaluatie opgeleverde producten

In de onderstaande hoofdstukken worden de opgeleverde producten geëvalueerd. Dit zijn de producten in hun uiteindelijke vorm. Een overzicht van tussenproducten kunt u terugvinden in hoofdstuk 19, en de beschrijving van het proces om tot de producten te komen in hoofdstuk 8 t/m 16.

21.1 Plan van Aanpak

In de oriëntatie fase (de eerste twee weken bij Exact) is de opdracht geanalyseerd en is er een plan van aanpak (PVA) opgesteld. De eerste versie van het PVA beschreef de aanpak met gebruik van Unified Process als software ontwikkelmethode en TestGoal als test methodiek. Deze versie is besproken met de manager, product manager en begeleidend developer waarin ze de keuze voor UP en TestGoal in twijfel trokken. Uiteindelijk bleek Scrum ook net zo goed toe te passen op het project en is daarvoor gekozen (hoofdstuk 6.1.4). Nadat een nieuwe versie van het plan van aanpak was gemaakt is deze nog een keer besproken met de stakeholders, en het onderdeel testen ook met de Quality Engineer. Aan de hand van de laatste opmerkingen is een definitief PVA gemaakt. Door feedback van de stakeholders is de kwaliteit en de juistheid van het PVA gegarandeerd, en is de opdracht ook duidelijker en concreter gemaakt.

21.2 Detail Ontwerp

In sprint 0 is een (Engelstalig) detailontwerp gemaakt waarin de user stories, constaints en de eerste versie van de SDK is beschreven. Het detailontwerp bevatte een eerste versie van een class diagram en de beschrijving van twee classes die gebruikt werden voor communicatie met de API. Dit detailontwerp werd wel gebruikt in het verificatieproces met de Quality Engineer, maar verder door niemand gelezen. Binnen de organisatie is geen vraag naar gedetailleerde ontwerpen omdat ontwikkelaars deze liever niet lezen, het teveel overhead (extra werk) is en deze snel verouderen. Ook bleken de vooraf gemodelleerde oplossingen toch niet één op één vertaald te kunnen worden en was het niet nodig architecturale beslissingen met een grote groep mensen te delen. Voor verdere sprints is daarom afgeweken van vooraf modelleren. Wel is een ‘Developer Documentation’ document gemaakt. Deze wordt beschreven in hoofdstuk 21.7.

21.3 Master- en Detail Test Plan

Het master- en detail test plan is alleen in sprint 0 en sprint 1 gebruikt. Later bleek dat dit, net als het detail ontwerp, teveel overhead was wat niet geschikt was voor een agile/lean project. Ook bleek dat veel testen goed aan de hand van het inzicht van de developer geschreven konden worden, en dat het om die reden zonde was van de beschikbare tijd om eerst een test plan te schrijven. Uiteindelijk is dus alleen het Master Test Plan en één detail testplan geschreven. Voor een voorbeeld van het Master Test Plan en het detail test plan zie de bijlage. De manier waarmee uiteindelijk getest is staat beschreven in hoofdstuk 6.3.

21.4 Software Development Kit

De Software Development Kit is gegroeid van een klein product met beperkte functies, naar een groot product met veel functionaliteit. Door een agile methode te gebruiken en de functionaliteit veel te controleren bij de stakeholders is een product gemaakt wat goed aansluit bij hun eisen en wensen. Ook technisch gezien voldoet het project aan de eisen. De principes van REST worden doorgevoerd waardoor

er geen overbodig dataverkeer is, en kan er worden bijgehouden elke velden van welke entiteiten veranderen. Ook is het makkelijk om de SDK te onderhouden en nieuwe entiteiten toe te voegen, er wordt rekening gehouden met read only velden, Controllers zijn generiek, de code volgt de design principles en de gebruiker (third party developer) wordt goed ondersteund in zijn of haar werkzaamheden. Dit door een gemakkelijke gebruikersinterface en documentatie in de vorm van intellisense (een manier voor de ontwikkelaar om beschrijvingen van classes en methodes te zien). Ook is de SDK in bèta fase getest door middel van een demo applicatie voor het Exact App Centre waardoor ook feedback van de gebruikers is meegenomen in het uiteindelijke product. De quality engineer, begeleidend developer en de product owner waren tevreden over de kwaliteit van het afgeleverde product.

21.5 Regressie Testen

Om de kwaliteit te verbeteren en zo veel mogelijk fouten te voorkomen moest er worden getest. Deze tests moesten automatisch kunnen worden uitgevoerd zodat na een wijziging snel gekeken kan worden of de verschillende onderdelen nog steeds correct functioneren. De test code is gecontroleerd door de Quality Engineer op juistheid en correcte testdekking (of alle belangrijke onderdelen wel getest zijn). Uiteindelijk zijn er 21 gebruikers acceptatie testen, 85 unit tests en 17 integratietesten geschreven die scenario's testen die voor komen in de SDK. Van de unit en integratietest zijn per methode ten minste één goed scenario (happy flow) en één slecht scenario getest. Voor complexe methodes die meerdere resultaten kunnen hebben zijn meer dan twee testgevallen geschreven.

De regressietesten kwamen erg goed van pas bij het doorvoeren van wijzigingen in architectuur. Indien er een paar tests ineens niet meer slaagden kon snel worden gezien waar de oorzaak van de fout lag en kon snel tot een oplossing worden gekomen. Ook taken als refactoring (onderdeel van Test Driven Development) konden makkelijker worden doorgevoerd omdat de testgevallen geautomatiseerd waren.

Het maken van regressietesten was een goede keuze voor een product als de SDK en heeft de kwaliteit van de code (voornamelijk de leesbaarheid, en scheiding van verantwoordelijkheid) helpen te verbeteren, en het aantal fouten in weten te perken.

21.6 Sprint Review

In de sprint review is de SDK getoond aan de stakeholders door middel van een demo. Omdat de SDK een moeilijk product is om te tonen is dit gedaan door middel van een presentatie waarin de voortgang werd behandeld, en een demo waarin de stakeholders konden zien hoe de SDK gebruikt kon worden om verschillende scenario's te realiseren. Deze demo werd in het Engels gegeven.

Sommige van deze sprint reviews waren goed, maar sommige waren erg lastig te begrijpen voor de stakeholders en geïnteresseerden. Ook was het moeilijk om een interessante demo te geven nadat veel functionaliteit al gecreëerd was en er alleen aan details gewerkt moest worden.

21.7 Developer Documentation

De developer documentation is naar het einde van het project toe gemaakt om de kennis van de SDK te kunnen overdragen aan het Connectivity Team. De documentatie bevat een globale beschrijving van de

functionaliteit van de SDK, de architectuur, de oplossingen voor problemen en een introductie over hoe te testen en te onderhouden.

Om de documentatie helemaal te laten aansluiten bij de eisen en wensen van het Connectivity Team is deze getest op een developer die nog geen kennis had van de interne werking van de SDK. Aan de hand van een vragenlijst over de globale structuur en veel voorkomende scenario's is gecontroleerd of de developer deze vragen met alleen de hulp van het document kon beantwoorden. De bevindingen van die test zijn meegenomen in de nieuwe versie die nogmaals is gecontroleerd door de Quality Engineer en de Product Owner. Hiermee is een voldoende beschrijving gegarandeerd. De developer documentatie kan worden gevonden in de bijlage.

22 Begrippen- en Afkortingenlijst

Afkortingen & begrippen	Beschrijving	Aanvullende Informatie
SDK	Software Development Kit	Verzameling van tools voor bouwen van software
API	Application Programming Interface	Een manier voor externe om delen van
oAuth	Open Authentication	Protocol voor authenticatie
EOL	Exact Online	Product van Exact
Json	Java Script Object Notation	Manier om een object te schrijven
C#	C Sharp	Een programmeertaal van Microsoft
TDD	Test Driven Development	Een Software Ontwikkelmethode
UP	Unified Process	Een Software Ontwikkelmethode
XP	Extreme Programming	Een Software Ontwikkel methode
App Centre	Een product van Exact	Website waar apps op kunnen worden verkocht
REST	Representational State Transfer	Een web protocol
oData	Open Data Protocol	Een protocol voor het opvragen van data
DSDM	Dynamic Software Development Method	Een Software Ontwikkel Methode
RAD	Rapid Application Development	Een Software Ontwikkel Methode
Linked Entity	Gekoppelde Entiteit	Een entiteit die gekoppeld is met een andere entiteit. In database termen wordt dit een associatie genoemd.
Lean	Lean Software Development	Manier om software te schrijven met zo min mogelijk overhead (documenten, analyses etc.)

23 Wijze van aantonen competenties

De student heeft voorafgaand aan het afstuderen drie beroepstaken geselecteerd die tijdens het afstudeertraject zijn bewezen. Van elk van deze beroepstaken staan in de onderstaande hoofdstukken beschreven hoe de student heeft aangetoond dat hij deze de beroepstaak beheerst en waar de moeilijkheden lagen.

23.1 Ontwerpen Systeemdeel - Complexiteit: Complex niveau: 4

Het ontwerpen van een systeemdeel is over de hele loop van het afstuderen gebeurd. In sprint 0 is een conceptueel class diagram opgesteld die heeft gediend als blauwdruk voor de eerste versie van de SDK. Omdat de methode Scrum werd gebruikt en het project zo lean mogelijk gehouden moest worden is er geen uitgebreide documentatie geschreven en is vooraf niet in detail nagedacht over details in het ontwerp. Per onderdeel wat gaandeweg is toegevoegd aan de SDK zijn wel altijd de volgende punten toegevoegd voor het hanteren van een goede architectuur:

- Functionele decompositie: Scheiding tussen verantwoordelijkheden
- Losse koppeling tussen classes door:
    - Toepassing van interfaces
    - Gebruik maken van delegates
- Toepassing van het Model (View) Controller design pattern
- Toepassing van Singleton Pattern
- Review van class diagram door Quality Engineer

Doordat er geen definitieve architectuur is opgesteld voorafgaand aan het ontwikkelen konden verschillende oplossingen worden uitgeprobeerd. Gaandeweg is zo gekozen voor de meest geschikte oplossing in een bepaalde situatie. Door het continue schrijven en herschrijven van de code met oog op toepasbaarheid, kwaliteit en leesbaarheid is uiteindelijk een robuuste architectuur opgezet. Voorbeelden hiervan zijn de generieke controllers (hoofdstuk 12.1) en de entity controllers (hoofdstuk 12.2). Een globaal overzicht van de uiteindelijke architectuur is beschreven in de documentatie (hoofdstuk 16.1).

De moeilijkheden in het ontwerpen van een systeemdeel lagen niet in de opgezette architectuur, maar in het blijven hanteren van de eerder beschreven principes bij het toevoegen van nieuwe functionaliteiten. In het geval van de EntityControllers verschoof de verantwoordelijkheid van het beheren van entiteiten van de Controller naar de EnityController waardoor opnieuw nagedacht moest worden over de scheiding. Ook was het genereren van Json voor gewijzigde velden een lastige opgave omdat de JsonConverter toegang moest hebben tot class als EntityControllers. Dit is uiteindelijk opgelost door middel van delegates.

23.2 Initiëren en plannen van het testproces – Complexiteit: Lastig, niveau: 3

Deze beroepstaak is op de volgende manier bewezen:

- Uitzoeken testen in een agile project *
- Schrijven van een mastertestplan die het testtraject voor testen in agile beschrijft
- Schrijven van een detail test plan die de testen per sprint beschrijft**

- Communicatie en overeenstemming met Quality Engineer van het Connectivity Team over testaanpak
- Bedenken testdekking voor de SDK en onderverdeling in verschillende categorieën.
- Opstellen van regressietesten voor:
    - Gebruikers Acceptatie Testen
    - Integratie Testen
    - Unit Testen
- Werken met MockObjecten ***
- Werken met het Test Framework van Microsoft
- Toepassen van Test Driven Development
- Bepalen juiste testaanpak per user story

* Omdat testen in een agile project niet goed beschreven was in de beschikbare literatuur moest zelf worden uitgezocht hoe dit op de beste manier gedaan kon worden. Om een geschikte manier te vinden is gekeken naar hoe agile testing werd toegepast in de organisatie, en is gekeken naar hoe dit beter kon. Deze bevindingen zijn toegepast in het project en zijn gaandeweg verfijnd. Ook is de onderverdeling van verschillende soorten testen voor een goede testdekking zelf bedacht en gehanteerd. Omdat de SDK geen standaard applicatie is maar een framework, is ook gekeken hoe onderdelen als gebruikers acceptatietesten konden worden opgepakt.

** Het detail test plan wat per sprint gemaakt zou worden is uiteindelijk vervallen wegens teveel overhead en niet voldoende vraag binnen de organisatie. De student heeft per user story gekeken naar de testbaarheid en heeft per user story een collectie met testen opgezet die deze user story testen. Om de kwaliteit en testdekking te garanderen is de aanpak besproken met de quality engineer.

*** Door toepassing van Test Driven Development is de code zo geschreven dat deze gemakkelijk getest kan worden. In sommige situaties waar delen van de implementatie niet publiek gemaakt moesten worden is op een andere manier getest. Dit gebeurde veelal door middel van MockObjecten. Zo werden aanroepen naar de API getest door een mock object te maken die dezelfde interface implementeerde als de APIConnector. MockObjecten werden voornamelijk toegepast om delen van de SDK te testen die gekoppeld waren met andere delen van de SDK (het doen van aanroepen waar een aanroep class noodzakelijk was, het converteren van Json waar een Controller class voor nodig was, etc.)

De beroepstaak is niet compleet conform de beschrijving uit het document ‘Beroepstaken van de opleiding ICT en Media’ uitgevoerd. De uitgevoerde taken hadden zowel elementen van de beroepstaak ‘Initiëren en plannen van het testproces’ en ‘Uitvoeren van en rapporteren over het testproces’. Voor de laatstgenoemde zijn testscripts gemaakt, is het .NET Unit Test Framework gebruikt, zijn de testen volgens planning uitgevoerd en zijn aan de hand van uitvoeren van test runs conclusies getrokken.

23.3 Bouwen applicatie – Complexiteit: Complex, niveau: 4

De beroepstaak bouwen van een applicatie is op de volgende manier bewezen:

- Programmeren in C# en Visual Basic
- Toepassing van C# ‘Reflection’
- Werken met delegates
- Werken met generieke types (C# generic types)
- Werken met asynchrone functies / multithreading
- Werken met Team Foundation Service voor beheren van de code
- Verbeteren van kwaliteit code door middel van code reviews

- Recursieve methodes schrijven voor opschonen van Json
- Werken met LINQ voor het vermijden van geneste for en foreach loops
- Schrijven van webrequest classes die communiceren met de Exact Online RESTful API
- Toevoegen van interfaces voor het ontkoppelen van implementatie, maken van tests
- Overerven van bestaande classes uit JSON.NET en .NET Framework voor eigen implementatie

De moeilijkheden van de beroepstaak 'Bouwen applicatie' zaten in het bouwen van software die gebruikt gaat worden door andere ontwikkelaars en in de toekomst waarschijnlijk open source wordt. Om die reden moet de interface en interne werking van de SDK van hoge kwaliteit zijn. Als ze dit niet zijn is er namelijk een grote kans dat ontwikkelaars afzien van het gebruik van de SDK voor hun applicatie. Daarom moest veel worden stilgestaan bij ontwerp en ontwikkelkeuzes en zijn grote beslissingen overlegd met de ontwikkelaars van het Connectivity Team.

Een ander moeilijk onderdeel was de recursieve functies voor het opschonen van de API response. Deze code moest kunnen omgaan met twee verschillende soorten results (één entiteit en een collectie) en een nieuwe Json string opbouwen. De problemen hiervan lagen bij het niet kunnen overzien van de structuur van de API response. Toen uiteindelijk een boomdiagram was getekend werd dit een stuk duidelijker en kon een goede oplossing worden bedacht.

Plan van Aanpak

Software Development Kit Exact Online

Auteur	Aschwin Brandt
Datum	12 november 2013
Plaats	Delft

Contents

1 Inleiding 1

2 Opdrachtomschrijving 2

2.1 Projectopdracht 2

2.2 Projectgrenzen 2

3 Betrokkenen 3

3.1 Stakeholders 3

3.2 Projectorganisatie 3

4 Methoden en Technieken 4

4.1 Software ontwikkelmethode 4

4.2 Testmethodiek 5

5 Op te leveren producten 6

5.1 Sprint 0 6

5.2 Sprint 1 t/m 8 6

6 Planning 7

7 Kwaliteit 8

7.1 Unit Testing & Gebruikers acceptatietests 8

7.2 Feedback 8

7.3 Integratie Testing 8

8 Risicoanalyse 9

9 Kosten en baten analyse 10

1 Inleiding

Exact biedt externe partijen ondersteuning in het bouwen van software dat gebruik kan maken van de gegevens en functionaliteit van Exact Online. Dit zodat het voor deze partijen gemakkelijker en aantrekkelijker wordt om een koppeling te maken van hun software naar Exact Online. Dit zal uiteindelijk het gebruik van Exact Online kunnen vergroten en de marktpositie van Exact kunnen verbeteren.

Momenteel liggen er plannen op tafel voor de ontwikkeling van een nieuw 'Developer Documentation Platform'. Hiermee kunnen externe partijen snel online kennis maken met de Application Programming Interface om snel een beeld te krijgen van de mogelijkheden. Dit documentation platform bestaat uit tekst/documentatie en een web applicatie waarmee in de browser opdrachten uitgevoerd kunnen worden op de API.

Om het naast het 'Developer Documentation Platform' nog makkelijker te maken om applicaties te ontwikkelen, zijn er ook plannen voor de realisatie van een Software Development Kit (SDK), die het ontwikkelaars nog makkelijker moet gaan maken om hun applicaties te maken.

Het plan van aanpak beschrijft de stappen die nodig zijn om tot de realisatie van de SDK te komen. Het plan van aanpak is zowel voor de ontwikkelaar als de betrokkenen geschreven en behandelt de producten, risico's en de kosten en baten analyse. Het plan van aanpak zal tijdens het project worden gebruikt als referentie voor wat betreft de aanpak van het project.

2 Opdrachtomschrijving

In dit hoofdstuk wordt de opdracht voor de realisatie van de SDK beschreven. In de onderstaande hoofdstukken kunt u de uitgebreide projectopdracht met probleem- en doelstelling, de beschrijving van het eindproduct en de projectgrenzen terugvinden.

2.1 Projectopdracht

De doelstelling van dit project is om het werken met data uit Exact Online gemakkelijker te maken voor ontwikkelaars. Dit door het aanleveren van een SDK waarmee gemakkelijk queries uitgevoerd kunnen worden op Exact Online en makkelijk te gebruiken objecten terug te krijgen. Op deze manier hoeven ontwikkelaars zich geen zorgen te maken over het authenticatieproces en het parsen van JSON objecten of XML files.

De software moet zo worden ontworpen dat toekomstige aanpassingen en uitbreidingen snel kunnen worden doorgevoerd. Hierdoor kan snel worden ingesprongen op wijzigingen in de API en de eisen en wensen van de klant. Na de oplevering van de SDK kunnen externe ontwikkelaars een dynamic link library (DLL) importeren en direct werken met objecten die terug worden gegeven uit Exact Online.

2.2 Projectgrenzen

Omdat de doorlooptijd van het project korter is dan de tijd die staat voor de opdracht (beschreven in de projectopdracht) zal het project enkel de volgende onderdelen bevatten:

- Onderzoek naar de SDK structuur uitgewerkt in een detailontwerp.
- De SDK zal enkel de taal C# ondersteunen maar ontworpen worden met het oog op multi level platform. Het ontwerp zal dus geen taalgebonden elementen bevatten.
- Wijzigingen in de RESTful API (parameters die gewijzigd worden) zullen worden ondersteund. De unit testen die worden gemaakt zullen falen en/of de code zal build errors geven.
- Unit Testen zullen (waar mogelijk) worden gemaakt.
- De SDK zal de meest gebruikte ontwikkel scenario's ondersteunen (uitgeschreven in user stories).
- Een externe ontwikkelaar hoeft anders dan een private en een public key in te voeren geen kennis te beschikken over oAuth, of handelingen uit te voeren voor het maken van een connectie met de API.
- De SDK kan na het doen van een query zowel C# objecten als JSON objecten teruggeven.

3 Betrokkenen

In dit hoofdstuk zullen de betrokkenen bij het project worden beschreven. In de onderstaande hoofdstukken staan zowel de stakeholders, als de projectorganisatie en hun rollen en verantwoordelijkheden beschreven.

3.1 Stakeholders

Naam	Functie	Verantwoordelijkheden
Aschwin Brandt	Software Developer	Design & Development SDK
Jurjen Boss	Product Manager Connectivity	Product Owner SDK. Bepaalt samen met de developer de features en de planning van die features per sprint.
Edgar Wieringa	Manager Product Management	Bepaalt samen met de product owner en de developer de planning en aanpak van het project.
Connectivity Team	Software Developer	Aanspreekpunt technische vragen & vragen m.b.t. de API
Cihan Duruer	Software Developer	Eerste aanspreekpunt technische vragen, vragen m.b.t. de API & ontwerpbeslissingen.

3.2 Projectorganisatie

Product Owner: Deze vertegenwoordigd de opdrachtgever en beheert de product backlog. De product owner bepaalt de prioriteit van de features en plant deze samen met het development team in in de sprint backlog. Ook is hij het aanspreekpunt voor de developer wat betreft vragen over de functionaliteit.

Developer: De developer is onderdeel van het Connectivity Scrum Team, maar zal aan een apart stuk software werken. De developer zal verantwoordelijk zijn voor het specifieren van de requirements en deze aanleveren aan de product owner, het opstellen van een master- en detailtestplan, een detailontwerp voor de realisatie van de SDK, de realisatie van de code en de unit tests.

4 Methoden en Technieken

Omdat er maar één persoon aan het product gaat werken zal geen gebruik worden gemaakt van een project management methode. De hoeveelheid werk om dit goed toe te kunnen passen staat niet in verhouding met de voordelen hiervan. De software ontwikkelmethode zal zorgen voor de benodigde structuur zodat er systematisch wordt gewerkt, en de oplevering van tussenproducten zal plaatsvinden. De keuze voor de te gebruiken software ontwikkelmethode en hoe deze zal worden toegepast zal hieronder worden beschreven. Ook zal de keuze voor de testmethode worden toegelicht.

4.1 Software ontwikkelmethode

Voor de realisatie van de SDK is gekozen voor een agile software ontwikkel methode. Dit heeft de volgende redenen:

- Niet alle requirements hoeven vooraf bekend te zijn.
- Er kan iteratief en incrementeel worden gewerkt. Dit stelt de ontwikkelaar in staat de juiste oplossing te vinden en geleidelijk functionaliteit toe te voegen.
- Na elke iteratie wordt een uitvoerbaar systeem opgeleverd waardoor vaak feedback kan worden gegeven.
- Functionaliteit van de RESTful API wordt continue uitgebreid en verbeterd. Hierop moet op ingesprongen kunnen worden.

De geprefereerde software ontwikkelmethode binnen de organisatie is Scrum. Deze heeft als nadeel dat er geen vaste vorm is van documenteren, maar dit wordt binnen de organisatie toch niet veel gedaan. De gedachtegang hierachter is dat documenten snel verouderen, en vanuit de code kan ook documentatie worden gemaakt. Ook heeft Scrum het grote nadeel dat het opstellen van een test plan geen concreet onderdeel is van de ontwikkelmethode. Het testplan, en eventuele andere benodigde documentatie, kan echter wel worden toegevoegd naar de eisen van het project en van de product manager.

De voordelen van de flexibiliteit, het vaak kunnen opleveren van een werkbaar product, en het team waar het project in uitgevoerd kan worden wegen hier echter meer dan genoeg tegenop. In dit plan van aanpak zal worden geschreven hoe onderdelen als documentatie en testplan zullen worden opgevangen.

Voor de ontwikkelmethode Scrum zal na de realisatie van het plan van aanpak de volgende stappen worden doorlopen:

1. Product Backlog User stories maken: functionele en niet functionele eisen van de SDK.
2. Gebruikers acceptatietests: Bepalen wanneer de SDK voldoet aan de eerder opgestelde eisen.
3. Opstellen detailontwerp van systeem.
4. Opstellen Master Testplan.
5. Per Sprint
    a. Detail Testplan
    b. Uitvoerbare testgevallen die falen *
    c. Werkende code

d. Refactoren
e. Mergen

* Om de kwaliteit van de code zo hoog te houden zal een onderdeel van Test Driven Development worden toegepast. Voordat er code wordt geschreven zal eerst worden gekeken naar het detail testplan en de testbaarheid van het onderdeel. Vervolgens zal indien mogelijk een unittest worden gemaakt die faalt, waarna de code zal worden gemaakt. Als de unittest slaagt zal de code worden herstructureerd om de leesbaarheid en onder houdbaarheid te verbeteren (refactoring).

4.2 Testmethodiek

Voor de testmethodiek is de keuze gevallen op TestGoal. Dit heeft als voornaamste reden omdat de ontwikkelaar hier bekend mee is en dat het, net zoals Scrum, een business/resultaat gedreven aanpak heeft. Er zal dus vooraf worden gekeken naar de risico's voor de business en hoe de beschikbare test ontwerptechnieken het best kunnen worden toegepast om deze risico's te klein mogelijk te houden.

Omdat gebruik gemaakt gaat worden van Scrum kan er niet volgens het V-Model worden getest. Er worden geen uitgebreide requirements opgesteld, en er zal geen functioneel en technisch ontwerp worden gemaakt. Er zal dus gebruikt gemaakt moeten worden van agile testen. De user stories en het detailontwerp zullen als input dienen voor het master testplan waarin het globale testproces voor het hele project staat uitgeschreven. Per sprint zal een kort detail testplan worden geschreven waar de eerder gekozen ontwerptechnieken zullen worden toegepast op de sprint specifieke functionaliteit.

5 Op te leveren producten

Dit hoofdstuk beschrijft de op te leveren producten. Omdat binnen Scrum geen planningen worden gemaakt voor periodes langer dan 4 weken, zullen alleen de globale producten worden beschreven die per sprint opgeleverd zullen worden.

5.1 Sprint 0

Het doel van sprint 0 is om de benodigde stappen te maken die nodig zijn om aan de realisatie van het systeem te beginnen. In deze sprint zal de nadruk liggen op de requirements van de SDK die zullen worden uitgeschreven in user stories en uiteindelijk het product backlog zullen vormen. Ook zullen acceptatietesten worden gemaakt waaraan gemeten kan worden of de functionaliteit correct is gerealiseerd, en zal een globaal plan worden geschreven over hoe getest gaat worden gedurende het project. In het onderstaande overzicht zijn alle producten weergegeven die deze fase worden gemaakt.

- Backlog met user stories
- Gebruikers acceptatietesten
- Detailontwerp SDK
- Master Testplan

5.2 Sprint 1 t/m 8

Voorafgaand aan een sprint zal eerst een sprint planning worden gedaan waarin de prioriteit van de user stories wordt herzien en een planning wordt gemaakt voor de komende sprint. Deze planning zal in de sprint backlog komen te staan. Vervolgens zal bij aanvang van de sprint een detail testplan worden opgesteld waarin uitgeschreven staat of en hoe de specifieke features getest moeten worden. Daarna zullen deze testen gemaakt worden, waarna begonnen zal worden aan het ontwikkelproces zoals beschreven in hoofdstuk 4.1.

Elke sprint heeft de volgende producten

- Detailtestplan
- Ge-update detailontwerp
- Regressietesten/Unit Tests
- C# Library die geïmporteerd kan worden in Visual Studio

6 Planning

Date	Week	Fase	Opmerking
28-okt	1	Oriëntatiefase	Inwerken bij Exact. Oriënteren op de opdracht & maken van Plan van Aanpak.
4-nov	2		
11-nov	3	Sprint 0	Detailontwerp, Master Test Plan en Product Backlog met daarin de user stories.
18-nov	4		
25-nov	5	Sprint 1	
2-dec	6		
9-dec	7	Sprint 2	
16-dec	8		
23-dec	9	Sprint 3	
30-dec	10		
6-jan	11	Sprint 4	
13-jan	12		
20-jan	13	Sprint 5	
27-jan	14		
3-feb	15	Sprint 6	
10-feb	16		
17-feb	17	Sprint 7	
24-feb	18		
3-mrt	19	Sprint 8	
10-mrt	20		
17-mrt	21	*	Vakantie: Deze wordt (in delen) tussen 25 november 2013 en 24 maart 2014 opgenomen.

7 Kwaliteit

Omdat de Software Development Kit een product is dat onder andere door externe ontwikkelaars gebruikt gaat worden is het belangrijk dat het product kwalitatief goed is. Ontwikkelaars zullen redelijk kritisch zijn over producten waar ze verstand van hebben, en het aanleveren een product dat veel fouten bevat kan negatieve effecten hebben voor hun bereidwilligheid om het te gebruiken. In dit hoofdstuk zal worden beschreven hoe het risico op slechte kwaliteit zal worden beperkt.

7.1 Unit Testing & Gebruikers acceptatietests

Voordat een stuk code geschreven wordt zal eerst minimaal één unittest worden geschreven die deze code test. De unittest is een whitebox test die per methode wordt geschreven om de interne werking van de methode te testen. Per methode zal over het algemeen een test worden geschreven die als invoer goede en foute waardes heeft om te zien hoe de code hier op reageert. Deze test kan later gebruikt worden om uitbreidingen en aanpassingen van de SDK geautomatiseerd te kunnen testen.

Door middel van een variant van unit testing zal (waar mogelijk) per feature een geautomatiseerde gebruikers acceptatietest worden gemaakt. Die is een blackbox test die controleert of het systeem reageert zoals de gebruiker verwacht (requirements). Net als de unit tests moeten deze gebruikers acceptatietesten geautomatiseerd uitgevoerd kunnen worden zodat in een later stadium uitbreidingen en aanpassingen snel getest kunnen worden. De geautomatiseerde gebruikers acceptatietest lijken op integratietests

7.2 Feedback

Binnen Scrum is veel mogelijkheid tot feedback. Na elke sprint is een sprint review waarin besproken wordt wat er goed en slecht ging en worden concrete afspraken gemaakt voor de volgende sprint. Binnen de sprint moet naast de daily Scrum en de feedback die terug kan worden gegeven op elke versie van de SDK, ook feedback komen op de documenten als het master testplan, het detailtestplan, detailontwerp en de unit tests. Deze zijn allemaal vatbaar voor fouten door verkeerde interpretatie van de ontwikkelaar, dus moeten voor zover mogelijk binnen de organisatie moeten worden getoetst. Ook zal de gerealiseerde code moeten worden nagekeken door een expert om te kijken of dit voldoet aan de richtlijnen binnen Exact en of deze code niet verkeerd is opgesteld. Daarom is afgesproken dat er korte lijnen gehouden worden tussen de developer en de begeleiding zodat direct feedback gegeven kan worden op gemaakte keuzes.

7.3 Integratie Testing

Voor de gebruikers is het belangrijk dat de SDK blijft functioneren na eventuele toevoegingen of wijzigingen. Om die reden zullen er testen worden geschreven die gebruik maken van de functionaliteiten van de SDK. Deze code simuleert de scenario's die uitgevoerd kunnen worden door de gebruikers en verifieert of de uitkomst gelijk is aan de verwachte uitkomt. Zo kan geautomatiseerd worden gezien of de SDK nog backwards compatible is na een wijziging.

8 Risicoanalyse

Risico	Gevolgen	Herkenningspunten	Preventie maatregelen	Herstelprocedure
Ontwerp is niet volledig of niet goed opgezet.	De uiteindelijke architectuur van de SDK is niet goed, waardoor programmeren lang duurt. SDK onduidelijk voor eindgebruikers.	Planning loopt uit. Uitbreiden van functionaliteit is moeilijk. Eindgebruikers willen niet werken met SDK.	Ontwerp baseren op succesvolle SDK's. Ontwerp continue laten controleren.	Ontwerp herzien in teammeeting. In overleg met team project refactoren.
Geplande functionaliteit is moeilijk te timeboxen.	Minder functionaliteit in het eindproduct.	Geplande functionaliteit overschrijdt herhaaldelijk de geplande tijd.	In teamverband functionaliteit plannen. Functionaliteiten opdelen in kleine delen.	Indien nodig meer ontwikkelaars op het project zetten.
Door onduidelijke en/of onvolledige requirements kan slecht de testdekking en risico's bepaald worden.	Geen reëel beeld van de testdekking, waardoor risico's gelopen kunnen worden op fouten.	Hoge mate van fouten in het live product.	Na maken sprint backlog een testplan maken waarin risico's en testdekking bepaald worden en deze worden gedekt met een testmethode.	Testplan achteraf maken en test- en verbeterfase inlasten.
Groot deel van de code is niet testbaar door middel van unittests.	De code kan niet geautomatiseerd getest worden waardoor het testen handmatig moet gebeuren.	Er is sprake van een gebruikers acceptatietest of systeem integratietest die niet met unittests getest kan worden.	Vooraf kijken naar de testbaarheid van code die gemaakt wordt in het proof of concept en kijken naar het detailontwerp. Aan de hand hiervan een testplan opstellen.	Methodes refactoren. Een andere testontwerp technieken moeten de testen overnemen.
In de test/bèta fase wordt het product niet gebruikt, waardoor geen feedback wordt gegeven.	Geen duidelijk beeld van de volledigheid van de functionaliteiten van de SDK.	Weinig/geen feedback.	Partners van Exact contacteren over de mogelijkheden tot testen/gebruiken v.d. SDK	Alsnog contact opnemen met de eindgebruikers.

9 Kosten en baten analyse

De return of investment van de Exact Online SDK is moeilijk te meten. Uiteindelijk hoopt Exact door het ontwikkelen van de SDK het aantal vragen m.b.t. de API te laten afnemen en dat meer ontwikkelaars producten gaan maken die de functionaliteit van Exact Online gebruikt. Dit zorgt er uiteindelijk voor dat Exact een prominente rol gaat spelen in veel applicaties doordat het als basis dient voor die applicaties. Dit brengt uiteindelijk meer klanten en meer data naar Exact waar in de toekomst veel dingen mee gedaan kunnen worden (Data mining etc.).

Indien de SDK deze verwachtingen waar maakt wegen de kosten en baten zeer goed tegen elkaar op. Eén persoon zal over de periode van 17 weken namelijk relatief goedkoop de software ontwikkelen. Als dit door een full time medewerker gedaan zou worden zouden de risico's een stuk groter zijn indien het project zou falen.

Master Test Plan

Exact Online Software Development Kit

Author	Aschwin Brandt
Team	Connectivity
Location	Exact Software Delft
Date	26 november 2013

Contents

1 Introduction 1
2 Agile Testing in Scrum and Test Driven Development 2
2.1 Sprint 0 2
2.2 Sprint 1 to 9 2
3 Test Methods 3
3.1 Unit Tests 3
3.2 (Automated) User Acceptance Tests 3
4 Detail Test Plan 4
4.1 Intake Test base 4
4.2 Risk analyses 4
4.3 Test Coverage 5
4.4 Test Design Methods 5

1 Introduction

This document is created for the developers, quality engineers and other stakeholders to specify the test process for the Exact Online Client Software Development Kit.

The goal of test process is to ensure that the Software Development Kit will work properly, and to prove to the management and product owner that that the requirements are met. Because the SDK is a product for expert users with high demands, as little as possible bugs must be present at launch. The SDK must be easy to work with so that a lot of people are going to use it to build software for Exact Online. The test process will give a good indication about the functioning of the code and the amount of errors, and will also help the developer in the development of the software.

The SDK will be developed using an agile software development method and because of this, the test processes must be applied accordingly. This document will explain how this is done, and will give a global view about the test process for the SDK. The master test plan will describe the global process and the test procedure. The master test plan will function as a blueprint for the detail test plan that will be created at the start of each sprint.

2 Agile Testing in Scrum and Test Driven Development

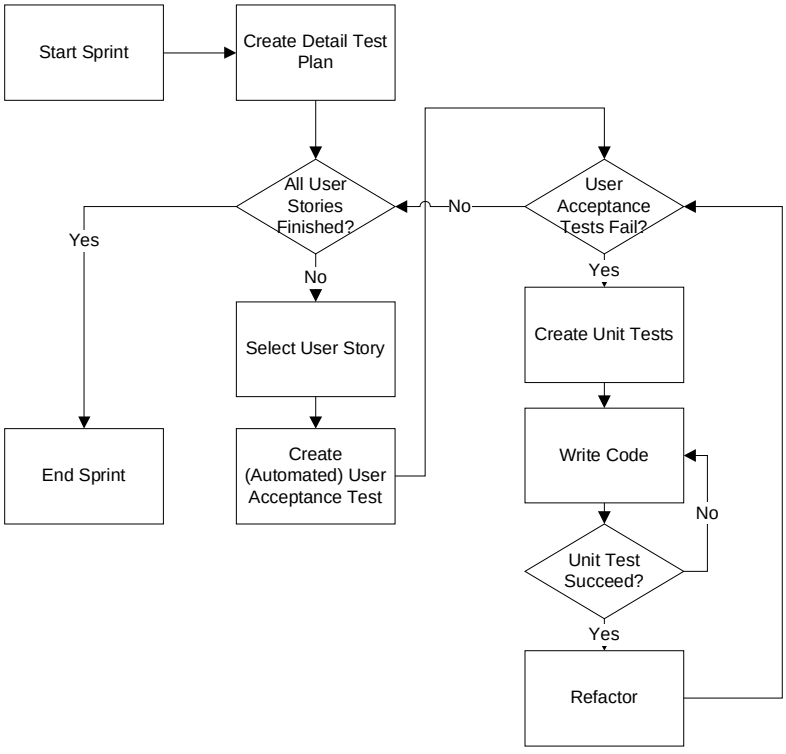
Testing in an agile environment can't be done by the normal 'V-Model' because the documents made in that model (functional design, technical design, etc.) aren't provided. Because of this, the test method must be tailored to the specific software development method of the project: Scrum and Test Driven Development.

Agile requires continues testing because each sprint will contain different user stories, and each user story requires a different test approach. The master test plan will only contain a global plan for the creation of the tests. Per sprint the developer/tester must look at the specific user stories and create a tailored test plan (detail test plan) accordingly. In the chapters below the details for the testing within the sprints will be explained.

2.1 Sprint 0

In sprint 0 all the necessary documents for the creation of the SDK are being created. Normally after the orientation phase, the documents specifying requirements and design will be almost done, and can be used for creating tests. With agile, these documents are just detailed enough for the first sprint (sprint 1). Only the Master Test Plan that describes the global test process for the whole project will be finished.

2.2 Sprint 1 to 9



In the beginning of every sprint a detail test plan is created. In this test plan the user stories in the form of sprint backlog items will be analyzed for risks, and the test coverage will be determined. After that the correct test method will be chosen, and the tests will be set up. For each user story an automated user acceptance test will be created. This possibly can't be done for every user story (non-functional or user story constraints for example), so in the detail test plan these user stories and the user acceptance tests will be described also.

Figure 1

Figure 1 shows the test process of every sprint.

3 Test Methods

This chapter covers the test methods that will be used during the development of the SDK. In the chapters below the methods are described in detail.

3.1 Unit Tests

Unit tests are a white box test that test specific units of code. Because this is a white box test, the internal operation of the methods is known, and tests can be built in such a way that possible faults in the operations can be tested. The collection of unit tests that are built each sprint can be used in the future to automatically test code after an addition or a change. The tests are build using the test design methods described in chapter 3.4.

3.2 (Automated) User Acceptance Tests

The user acceptance tests simulate the scenarios described in the user stories and test if the acceptance criteria are met. Each user story should have at least one user acceptance test. This must be preferably an automated test that can be added to the collection of regression tests. If this isn't possible (for example with the user story constraints) the test must be issued on paper and must be tested manually at the end of each sprint. This to ensure that the user story is implemented correctly and/or that the old user stories aren't compromised by additions or changes to the system.

The user acceptance test tests at least two scenarios: the right way and the wrong way. Each test must be described. The tests that can be automated are described in the code and manual tests must be described using the table below.

#	Post conditions	Action	Expected Result	Observed result	Test failed?

4 Detail Test Plan

For each sprint a detail test plan will be created to specify the test base, make a risk analyses, find the proper test coverage, select the appropriate test methods and create the logical tests that describe what and how to be tested. This chapter will describe which parts the detail design will consist of and what steps need to be taken each phase to create the detail test plan.

4.1 Intake Test base

The test base for this project will consist of a detail design with user stories and acceptance criteria. This data will be used to create all the tests for this project. Every phase the test base will be analyzed to find risks and decide proper test coverage. Since the test base is relatively small it's difficult to find the risks and find the appropriate test method. The steps that need to be taken to cover the risks a small test base brings will be described in the chapter below.

4.2 Risk analyses

With the risk analyses the risks will be found and ordered. This must be done to get a better understanding of the risks and to find the proper test coverage and test methods. After the risk analyses, all stakeholders will know which parts of the system are important and must be tested, and what parts are optional or can be left untested. In this way the tester will enforce that the critical parts of the system will be tested, but that the test process overall won't take too long.

The input for the risk analyses is the test base. For this project the test base will consist out of the user stories and acceptance criteria made each sprint. These user stories will be translated to system parts that are showed in a test tree so that attention points are visible for the stakeholders. This test tree will be used as a checklist and as input for finding relative importance.

In Figure 2 an example of the test tree is showed.

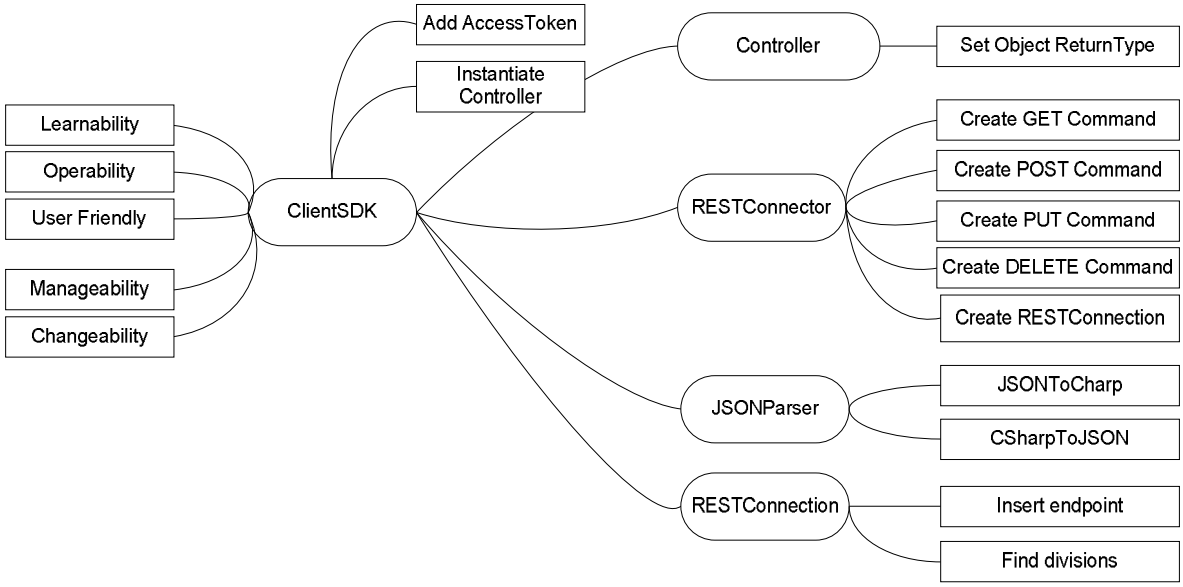


Figure 2

After the test tree is created, the stakeholders will be asked to assess the specific parts and make a statement about the risks. The results of this analysis will show the relative importance and will be used to prioritize the features by risk. This estimation will be written in the table below.

Risk area	Stakeholder 1	Stakeholder 2	Stakeholder 3
Total			

Each stakeholder will estimate the risk of each risk area with a number. This numbers can be 9 for critical importance of the functioning of the system, 5 for an important function. Errors and bugs are allowed as long as there is a workaround, 3 for non-critical functions where errors are only annoying, but don't compromise the intended result and 1 for risk areas that isn't necessary for the proper functioning of the system. With this analysis all risk areas can still be important for the system but the relative importance will be found.

4.3 Test Coverage

Using the test tree and the ordered risk areas, the test coverage will be decided. This coverage must be made to ensure that the features with high risks are covered, but that no time is wasted testing features with low or no risk. The test coverage will do a statement about which parts will be included and which part will be excluded from the tests, what risks this will bring and what the consequences are.

4.4 Test Design Methods

For each test method (user acceptance test/unittest/etc) and functionality/user story there are a variety of test design methods that can be used to create a module test (white box). These methods are:

- Syntax Testing
- Equivalence partitioning
- Boundary Value Analysis
- Cause Effect graphing
- State Transition
- Process Cycles Tests / Algorithm test
- Load Test
- Stress Test
- Reliability Test
- Exploratory Testing

Depending on the user stories and their risks, appropriate test design methods will be selected and developed in the form of logical test cases which describes what and how a risk area must be tested. The detail test plan will show what must be tested (logical test design). The code and the manual test cases show how this is done (in code documentation).

Detail Test Plan

Exact Online Software Development Kit Sprint 1

Author	Aschwin Brandt
Team	Connectivity
Location	Exact Software Delft
Date	-

Contents

1 Introduction 1
2 Risk analyses 2
2.1 Intake Test base 2
2.2 Test Coverage 3
3 User Acceptance Tests 5
3.1 Standard Users Stories 5
3.2 Constraints 6
4 Unit Tests 8
4.1 Test Design Methods 8

1 Introduction

This document is created for the developers, quality engineers and other stakeholders to specify the test process for the Exact Online Client Software Development Kit in sprint 1. This sprint will contain the user stories that form the base (framework) of the SDK and therefore must be tested properly. The detail test plan will describe how the testing is done in sprint 1, what the risks are, the decisions about the test depth and how the user acceptance tests and unit tests are done. After the sprint is done, a test report will describe the test results of tests in this document.

2 Risk analyses

With the risk analyses the risks will be found and ordered. This must be done to get a better understanding of the risks and to find the proper test coverage and test methods. After the risk analyses, all stakeholders will know which parts of the system are important and must be tested, and what parts are optional or can be left untested.

2.1 Intake Test base

The test base for this project is relatively small. These because the project is created using an agile method and not all the requirements are known up front. The test base will consist out a detail design with user stories, acceptance criteria and a class diagram with documentation about the internal working. In the overview below these parts are described and there risks, and activity to counter these risks are assessed.

Finding	Risk	Activity
User Stories, Acceptance Criteria & Constraints		
Only specify the users' point of view.	Details about the implementation are not described therefore this implementation is left to the developers perspective.	
Only specify what a user wants and why.	Details about what to do on a wrong action are not described (alternative flow).	Create comprehensive user acceptance tests which test all possible paths.
Don't describe which criteria must be met before a user story can be executed.	Some features will not work before other dependencies are instantiated.	Describe these criteria in the user acceptance tests.
Don't describe any triggers that can trigger actions in the SDK.	Events like lost connection, expired access token, etc. are not described.	Find and tests these triggers in the user acceptance tests.
Class Diagram		
Not all classes are showed. Only the abstract of controller and model.	The abstracts are built with the Account entities in mind. Therefore generic functions of other entities can be forgotten.	Model other entities with high amount of logic (coupled classes for example) also.
Not all methods and properties are showed.	This could detriment the code coverage.	Use Test Driven Development to ensure code coverage.
The internal communication between instances is not showed in a sequence diagram.	Algorithm must be created by the developer. He/she has to think about the process and the calls between classes. This can make for code/tasks that aren't properly divided between components.	Develop using the class diagram as reference.

Table 1

2.2 Test Coverage

In sprint 1 the basic functionality of the Software Development Kit will be developed. This functionality will function as a sort of framework for future functions. Therefore all the functionality must be tested properly. Because the software development Test Driven Development is applied, all user stories will have unit tests and user acceptance tests where possible. To find the risk areas in the SDK, a test tree is created. This tee is displayed in figure 1.

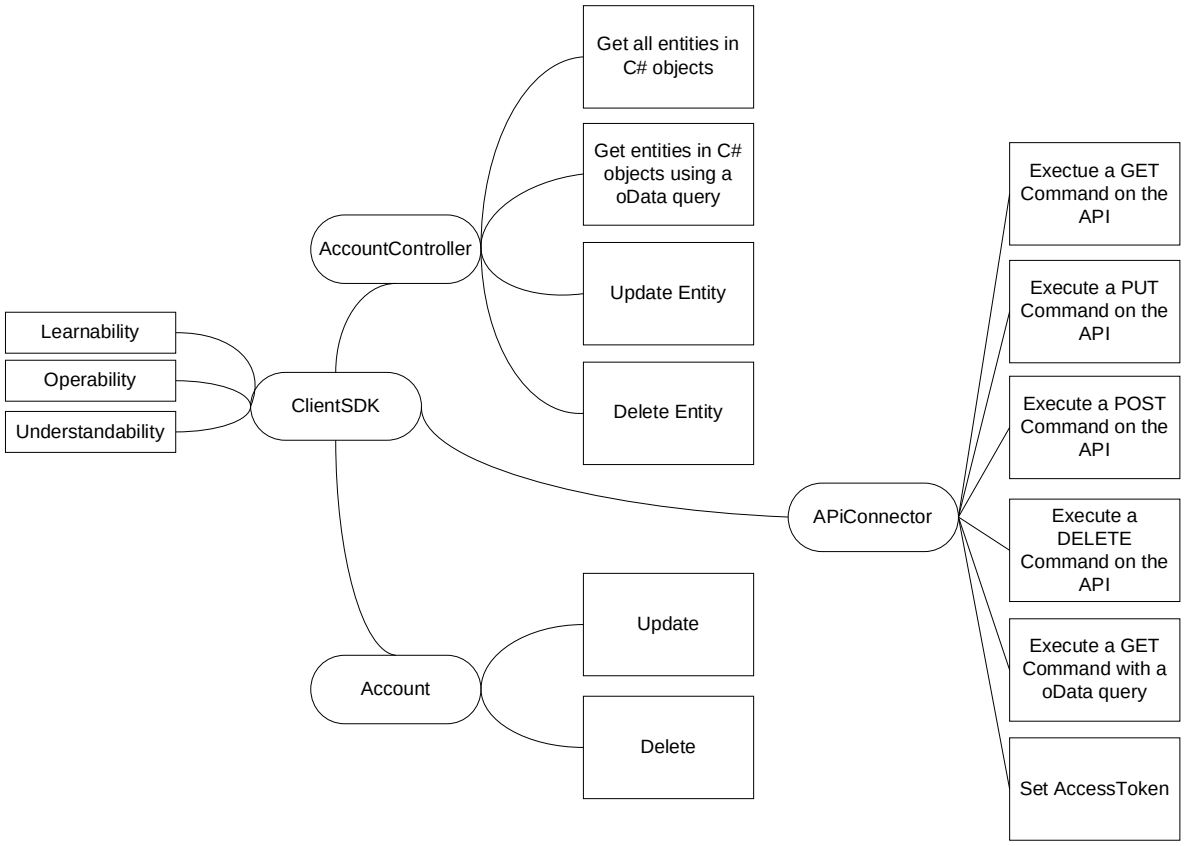


Figure 3

The depth (and strength) of the tests must be specified. In table 2 is specified what functions have a high chance of errors. Each function can have a number that indicates the chance for errors in that function. These numbers can be 9, 5, 3 and 1. The further is the highest amount of risk, and the latter the lowest. Using this table, the correct test design method is selected.

Risk area	Risk Level
Connection with the API	
Execute a GET Command on the API	
Execute a PUT Command on the API	9
Execute a POST Command on the API	9
Execute a DELETE command on the API	
Set AccessToken	
Functionalities on Accounts	
Get all entities in C# objects	9
Get entities in C# objects using oData query	3
Update Entity	3
Delete Entity	
Functions of a specific account	
Update	
Delete	
Total	**33**

Table 2

3 User Acceptance Tests

The user acceptance tests simulate the scenarios described in the user stories and test if the acceptance criteria are met. Each user story should have at least one user acceptance test. This must be preferably an automated test that can be added to the collection of regression tests. In the chapters below the user acceptance tests are described for the user stories and the constraints.

3.1 Standard Users Stories

The user acceptance tests of standard user stories will be automated were possible. Each user story will be set up using the user story description and the acceptance criteria. In the table 3, the user stories and their coupled user acceptance tests are displayed. The right column displays if a user story can be automatically tested using a user acceptance test.

#	As a user of the SDK I can	Acceptance criteria	Automatically Testable?
1.	Get a text response in JSON format from the API after executing a REST GET call so that I can read data from Exact Online.	The user retrieves a string in JSON format.	Yes
2.	Execute a PUT, DELETE and POST command on the API so that I can manipulate data in Exact Online	The user can update, delete and create data in Exact Online	Yes
3.	Get a collection of all the account entities in JSON format so that I can access the data without knowing the API	Collection of entities contains equal number of entities that are returned from the API parsed in a C# Dynamic class so that all the attributes are available.	Yes
4.	Get a specific collection of JSON account entities by using a oData query so that I can get specific entities without knowing oAuth	Collection of entities contains equal amount of entities that are returned with a raw REST/oData request	Yes
5.	Get a collection of all account entities in C# objects so that I can access the data without knowing the API	Collection of entities contains equal amount of entities that are returned from the API	Yes
6.	Get a specific collection of C# accounts entities by using a oData query so that I can get specific entities without knowing oAuth	Collection of entities contains equal amount of entities that are returned with a raw REST/oData request	Yes
7.	Alter the state of an account entity in the database by altering the fields of the C# entity so that I don't have to worry about connecting with the API via oAuth and REST to update the entity	5 seconds after updating the entity, the state will be changed in the database. When requesting the object with API the state is equal to the state of the C# object	Yes
8.	Alter the state of an account entity in the database by manually calling the update method of that entity so that I	Directly after updating the entity, the state will be changed in the database. When requesting the entity via	Yes

	know that the update is made directly	the API, the state is equal to the state of the C# object	
9.	Delete an account entity by manually calling the delete method of that entity so that I can delete entities from Exact Online	Directly after deleting the entity won't exist in Exact Online	Yes

Table 3

3.2 Constraints

In table 4 the user acceptance tests for the non-functional elements are displayed. These can't be tested automatically because these test make a statement about the usability, operability etc.

#	User Story (as a user I can)	Post conditions	Action	Expected Result	Observed result	Test failed ?
1.	Get entities from Exact Online without knowing the specific steps (oAuth2.0, SSL, REST) to take so that I can easily access data from Exact Online	Valid access token	User must only specify the accesstoken and do getAllEntities() methos of the accountController.	getAllEntities returns all Account entities of that specific user.		
2.	Use the SDK functionality after importing a DLL and entering the key so that I can start using the SDK with the minimum amount of effort	DLL & Valid access token	Add DLL as reference. Instantiate ExactClientSDK with the access token.	User can do read, update and delete on Account entities in Exact Online		
3.	Execute a oData query on the API without specifically connecting to the API so that I don't have to create the connection myself	Referenced DLL Valid access token	Set endpoind use APIConnection class and then use get with a query to execute a query on the API.	Result is returned.		
4.	Write most of my code using the standard XML documentation (IntelliSense) so that I don't have to read the	Valid access token, ExactClient class instantiated.	User types 'accountController.g'	Options 'getAllEntities()', 'getEntity(guid)', and 'getEntities(query)' are displayed.		

	documentation to execute specific tasks					
5.	Easily access all the functionality of the API via a REST connector. For example: connector.get() connector.post() etc.	Valid access token, Instantiated APIConnector class	Use makeGetRequest, makePostRequest, makePutRequest and makeDeleteRequest methods	User can access all functionality of the API using these commands.		
6.	Get JSON entities back after executing queries so that I can parse the entity to a format I like	Valid access token, instantiated APIConnector class	Execute APIConnector methods (makeGetRequest, MakePostRequest etc.)	Return value is string which contains JSON		

Table 4

4 Unit Tests

Unit tests are a white box test that test specific units of code. Because this is a white box test, the internal operation of the methods is known, and tests can be built in such a way that possible faults in the operations can be tested. The collection of unit tests that are built each sprint can be used in the future to automatically test code after an addition or a change.

4.1 Test Design Methods

For each test method (user acceptance test/unit test/etc.) and functionality/user story there are a variety of test design methods that can be used. These methods are:

1. **Syntax Testing**: Tests fields and data elements.
2. **Equivalence partitioning**: Tests functional decisions.
3. **Boundary Value Analysis**: Test functional decisions at boundary values.
4. **Cause Effect graphing**: Small system parts with high risk.
5. **State Transition**: Test transitions in states
6. **Process Cycles Tests**: Tests use cases and suitability of the system
7. **Algorithm Test**: Tests the scenarios in a system and if functions are properly communicating

For each system part/risk area an appropriate test design method must be selected. This depends on the amount of risk and the type of functionality. Table 5 shows the functionality that must be tested and the test types that will be used. To show the possible risk the column 'Implementation Details' is added to show what types of data or structure is applicable.

User story/Risk Area	Implementation Details	Risk Level	Test Type
Execute a PUT Command on the API	Valid Token, creating endpoint string, passing correct values	9	Cause and Effect graphing
Execute a POST Command on the API	Valid Token, creating endpoint string, correct values	9	Cause and Effect graphing
Get all entities in C# objects	Parsing from JSON string to C# object	9	Boundary Value analysis
Get entities in C# objects using oData query	Parsing and sending oData query	3	Syntax Tests
Update Entity	Setting update delegate, parsing from C# object to JSON string	3	Syntax Tests

Table 5

Detail Design Exact Online Software Development Kit

Author	Aschwin Brandt
Location	Exact Software Delft
Date	26 november 2013
Version	1.0

Version Management

Version	Date	Author	Changes
1.0	26-11-13	A.S.R. Brandt	Initial Document

Table of Contents

1 Introduction 2
2 Requirements 3
2.1 User Stories Sprint 1 3
2.2 Constraints 4
3 Class Diagram 5
3.1 ExactSDKClient Class 6
3.2 RESTConnector & RESTConnection Class 6
3.3 EntityController & Model Class 6

1 Introduction

This document describes the details of the Exact Online Software Development Kit (SDK) design and will be used by the lead developer to communicate the features, design decisions and internal operations with the stakeholders. The detail design can also be used by future developers to extend the functionality of the SDK or execute updates in the code or the design.

The detail design of the SDK will grow over the course of the project, and must be kept up to date by the designers. Therefore the detail design is a sprint backlog item that must be delivered. Because the document is part of an agile project, only the absolute necessary parts will be described in this document. Details about the internal operations of the SDK will be documented in the code, the code comments and in the regression tests. After the initial detail design is approved the Master Test Plan will be created. After that the first sprint will start.

2 Requirements

The user stories describe the functionality that a user wants in the system. The user stories are a part of the Scrum method and describe the wanted functionality of a system in the language of the user. User stories come with acceptance criteria that describe which criteria must be met before the user story has been fulfilled. For the SDK most of the user stories will look like the list below:

As a developer I (can)

- Get all the objects back
- get a list C# entities back after executing queries
- get a list of JSONS entities back after executing queries
- can delete an entity by using its delete method
- can update an entity by using its update method
- can create an entity using a controllers create method
- don't have to manually update altered entities

Per entity (account, company, invoice, invoice item, etc.) other functionality must be created. Not every entity can be created be read, updated or deleted.

The user stories are split in to two types: User stories and constraints. The further will contain specific functions for the system, and the latter will describe the non-functional elements like performance and usability.

2.1 User Stories Sprint 1

The user stories describe the desired functionality. User stories 1 to 4 are necessary for all the functionality in het SDK and are a part of the framework. These will have high priority for sprint 1. After that the CRM 'Accounts' part of the SDK will be created.

#	**As a user of the SDK I can**	**Acceptance criteria**
10.	Get a text response in JSON format from the API after executing a REST GET call so that I can read data from Exact Online.	The user retrieves a string in JSON format.
11.	Execute a PUT, DELETE and POST command on the API so that I can manipulate data in Exact Online	The user can update, delete and create data in Exact Online
12.	Get a collection of all the account entities in JSON format so that I can access the data without knowing the API	Collection of entities contains equal number of entities that are returned from the API parsed in a C# Dynamic class so that all the attributes are available.
13.	Get a specific collection of JSON account entities by using a oData query so that I can get specific entities without knowing oAuth	Collection of entities contains equal amount of entities that are returned with a raw REST/oData request
14.	Get a collection of all account entities in C#	Collection of entities contains equal amount of

	objects so that I can access the data without knowing the API	entities that are returned from the API
15.	Get a specific collection of C# accounts entities by using a oData query so that I can get specific entities without knowing oAuth	Collection of entities contains equal amount of entities that are returned with a raw REST/oData request
16.	Alter the state of an account entity in the database by altering the fields of the C# entity so that I don't have to worry about connecting with the API via oAuth and REST to update the entity	5 seconds after updating the entity, the state will be changed in the database. When requesting the object with API the state is equal to the state of the C# object
17.	Alter the state of an account entity in the database by manually calling the update method of that entity so that I know that the update is made directly	Directly after updating the entity, the state will be changed in the database. When requesting the entity via the API, the state is equal to the state of the C# object

2.2 Constraints

The constraints describe the non-functional elements of the SDK. These are described in the same way as the user stories, but don't have acceptance criteria, and are mandatory for all the user stories.

#	**As a user of the SDK I can**
1.	Get entities from Exact Online without knowing the specific steps (oAuth2.0, SSL, REST) to take so that I can easily access data from Exact Online
2.	Get entities from Exact Online without knowing the API structure so that I don't have to read documentation for common scenarios (for example: controller.get(query))
3.	Use the SDK functionality after importing a DLL and entering the key so that I can start using the SDK with the minimum amount of effort
4.	Execute a oData query on the API without specifically connecting to the API so that I don't have to create the connection myself (for example: controller.get(query))
5.	Write most of my code using the standard XML documentation (IntelliSense) so that I don't have to read the documentation to execute specific tasks
6.	Easily access all the functionality of the API via a REST connector. For example: connector.get() connector.post() etc.
7.	Get JSON entities back after executing queries so that I can parse the entity to a format I like

3 Class Diagram

The class diagram (figure 1) shows the internal structure of the SDK. Per class the associations, multiplicities, public methods and properties are showed. Where needed further explaining is done in the subchapters.

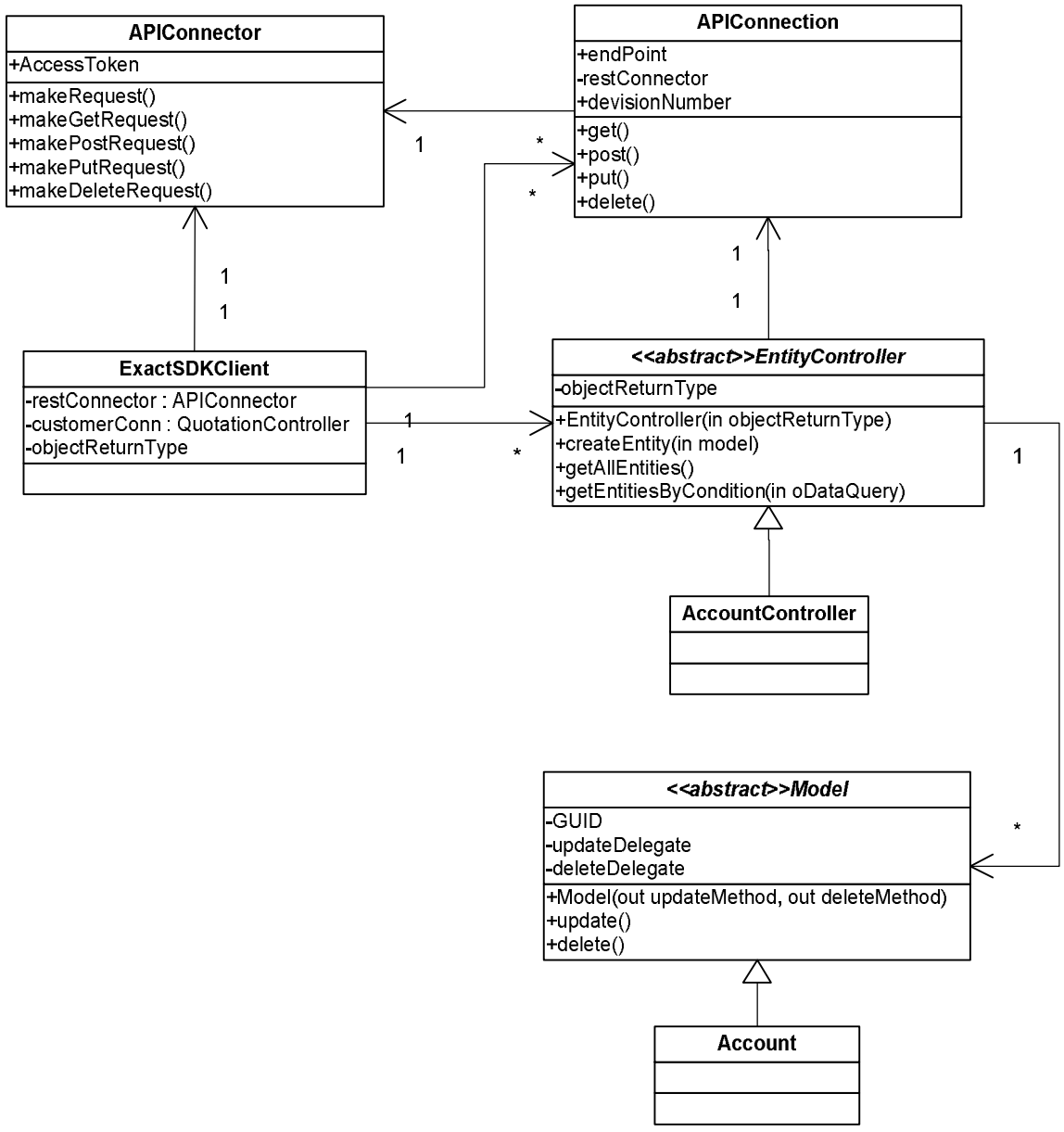


Figure 4

3.1 ExactSDKClient Class

This class is the primary class for the users. After instantiating an object of this type the user has access to all the functionality of the SDK. Within this class instances of RESTConnector and Controller will be created and managed. In this way the user can easily create REST calls to the API or alter the data in Exact Online via the appropriate controller.

3.2 RESTConnector & RESTConnection Class

This class is used to execute oData and REST commands directly on the API. The RESTConnector is created by an instance of ExactSDKClient and can also be used to create specific RESTConnection instances. These RESTConnection instances can be used to access specific parts of the Exact Online API like Invoices, GLAccounts or Accounts and let the user access these instances without having to specify the endpoint and request type each call.

Each Exact Online user has a specific division number. This is the number of the administration of the user. A user can have multiple administrations. If the user has only one administration this will be used in all the RESTConnecion instances. If a user has more than one administration a specific administration must be specified when initiating a connection.

3.3 EntityController & Model Class

The EntityController class is an abstract class. This class controls all instances of the Model class that are associated. For example AccountController has a collection of Account classes that represent the accounts in Exact Online for a specific user. When the user changes a value in the Account object, the AccountController takes care of the connection with the API via the RESTConnection class to make sure these local changes are also made in Exact Online.

The EntityController can return two types of entities: JSON or C#. The return type is specified with the parameter 'objectReturnType' that the user must specify in the ExactSDKClient class. The user can also manually update or delete an entity of the Account type. Because of the design pattern Model-Controller the Model does not know about the Controller. To get around this, there are two delegates for update and delete that must be specified when the Model get initiated.

Exact Online Software Development Kit Documentation

Author	Aschwin Brandt
Location	Exact Software Delft
Date	21-03-2014
Version	6.0

Version Management

Version	Date	Author	Changes
1.0	06-01-13	A.S.R. Brandt	Initial version of documentation (branch of deprecated detail design)
2.0	21-01-14	A.S.R. Brandt	Update documentation after refactoring phase of the SDK
3.0	27-02-14	A.S.R. Brandt	Updated the documentation
4.0	04-03-14	A.S.R. Brandt	Updated documentation after review of S. Hemati
5.0	18-03-14	A.S.R. Brandt	Added creating models script description
6.0	20-02-14	A.S.R. Brandt	Removed reference field due to updating in architecture

Table of Contents

1. Introduction 2
2. Functionalities 3
   - 2.1 Connect to the API 3
   - 2.2 Connect to specific parts of the API with friendly methods 3
   - 2.3 Deserializing Json To C# Objects 3
   - 2.4 Serializing C# Objects to Json 4
   - 2.5 ExactClient 4
3. Tests for the Software Development Kit 5
   - 3.1 User Acceptance Tests 5
   - 3.2 Integration Tests 5
   - 3.3 Unit Tests 5
   - 3.4 Performance Tests 5
4. Basic Structure Software Development Kit 6
   - 4.1 Software Development Kit layers 6
   - 4.2 Authenticating with Exact Online 6
5. Get Collection of Entities 7
6. Managing Entities: Internal working Controller & EntityController 9
   - 6.1 Controller: Managing Entities 9
   - 6.2 ControllerSingleton: Managing Controllers 9
   - 6.3 EntityController: Managing Entity State 10
7. Exact Online Models 11
   - 7.1 Attributes and Field Types 11
   - 7.2 Scenario: Adding new entity 11
   - 7.3 Scenario: Changing Exact Online API Entities 12
8. Converting Entities: Serializing- and de-serializing Json 13
   - 8.1 Convert Json to Object 13
   - 8.2 Convert Object to Json 13
   - 8.3 Errors & Exceptions 13
9. API Responses 14
   - 9.1 API Response Structure 14
   - 9.2 API Response Cleaner 15

9.3 Linked Entities 16
9.4 Scenario: Exceptions & Error Codes 16
10. Calling the Exact Online API 17
10.1 Scenario: Usage APIConnection & APIConnector 17
10.2 Scenario: Expired Access Token 17
11. Generating Models 18
12. Delivering new Version of the SDK 18

1 Introduction

This document describes the details of the Exact Online Software Development Kit (SDK) architecture and can be used by future developers as a reference guide to get a global understanding of the structure and internal working of the SDK. In this way they can get up to date quick and make changes to the SDK easily and faster. The documentation includes information about the functionalities of the SDK, the global structure, the test process, the different layers and details about specific implementation for parts that are difficult to understand.

As the SDK is created within an agile project, only the absolute necessary parts to understand the SDK are described. For details about the implementation, the developer must use the code and the automated test cases. This document is set up to provide information to help in the following scenarios:

- The developer wants to maintain or improve the SDK
- An entity from the model must be updated
- The oData query returns the wrong result
- Data isn't converted in the right way to entities
- Unknown exceptions are thrown (for example: internal server exception)

In the first part of the documentation the functionalities, different types of tests and the test process are described. The chapters after that will describe the SDK top down where the first chapters will describe how the functionality that the third party developer uses is built. The last chapters will describe the communication with the API.

2 Functionalities

This chapter describes all the functionalities of the Software Development Kit to create a good context for the developer about what features are necessary and which ones ware not. The SDK is built to simplify working with the Exact Online Application Programming Interface (API) and each layer that is created takes away common work for software developers that want to use date from Exact Online. The chapters below show all the functionality that is currently part of the Software Development Kit and how it's used.

2.1 Connect to the API

In its most basic form the SDK lets the developer execute commands in an easy way. This happens by using the class 'APIConnector'. In Tabel 12 an example is showed.

```
var conn = new ApiConnector(accesstoken, accesstokenDelegate);
string uri = "https://start.exactonline.nl/api/v1/499156/";
string filter = ""; // oData Filter
string expand = ""; // Field to Expand

conn.DoGetRequest(uri, filter, extend);
conn.DoPutRequest(endpoint, putdata);

string guid = "3c534e79-c4fe-44d2-9765-00b30573c2de";
conn.DoDeleteRequest("crm/Accounts" + "(guid'" + guid + "')");

string postdata = "";
conn.DoPostRequest("crm/Accounts" + "(guid'" + guid + "')", postdata);
```

Table 6

2.2 Connect to specific parts of the API with friendly methods

Connect to specific parts of the API with friendly methods to do get, post, put & delete operations. Table 7 has an example of how to do this. The 'ID' is the name of the unique field. 'AccessToken', 'GUID' and 'Data' are fake values.

```
ApiConnector _connector = new ApiConnector("FakeAccessToken", null);
APIConnection _conn = new APIConnection(null,
"https://start.exactonline.nl/api/v1/499156/crm/Accounts");

_conn.Get("");
_conn.GetEntity("ID", "3c534e79-c4fe-44d2-9765-00b30573c2de ");
_conn.Put("ID", "3c534e79-c4fe-44d2-9765-00b30573c2de ", "Data");
_conn.Post("Json Formatted Data");
_conn.Delete("ID ", "3c534e79-c4fe-44d2-9765-00b30573c2de ");
```

Table 7

2.3 Deserializing Json To C# Objects

Converting raw Json to typed Objects from the Exact Online API is done by using the JSON.NET JsonConvert class. <T> is the specific type the json (json in string format) is converted to.

```
JsonConvert.DeserializeObject<T>(json);
```
Table 8

2.4 Serializing C# Objects to Json

Converting raw Json to typed Objects is done by using the JSON.NET JsonConverter. For correct interpreting of the DateTime and ReadOnly values the ExactOnlineJsonConverter is used.

```
ExactOnlineJsonConverter converter = new ExactOnlineJsonConverter();
JsonConverter[] converters = new JsonConverter[1];
converters[0] = converter;
var json = JsonConvert.SerializeObject(entity, converters);
```
Table 9

2.5 ExactClient

The ExactClient provides an easy way to do all the above steps without thinking about internal operations via ExactClient class which does the work for communication with the specific controllers. In Table 10 and Table 11 the actions that can be performed by using ExactClient are displayed.

```
var client = new ExactClient("http://start.exactonline.nl/", "AccessToken");

// Add SalesInvoice to Database
SalesInvoice newInvoice = new SalesInvoice();
client.For<SalesInvoice>().Insert(ref newInvoice);
```
Table 10

```
// Read Account
var accounts = client.For<Account>().Get();

// Update Account
var account = client.For<Account>().GetEntity("c20a5590-c605-4f59-8fbb-112ee142bc59");
client.For<Account>().Update(account);

// Delete account
client.For<Account>().Delete(newAccount);
```
Table 11

In the case an access token is expired dedicated code can be called. This is done by setting the 'AccessTokenManagerDelegate'. If this delegate is set, it will be called when the SDK is unauthorized to access the API.

3 Tests for the Software Development Kit

The SDK is developed using Test Driven Development (TDD) to ensure good test coverage and testable code. This means that for each user story automated tests are created. In the chapters below the different type of tests en their interpretation are described.

3.1 User Acceptance Tests

The User Acceptance Tests (UAT) is the test that tests the functionality for the user. Because the SDK is made for developers the user stories that are created are tested. For the user acceptance tests a different project is created. Each test is in its own file which is named <User Story TFS Number><Short Description of Functionality>. For example: “184244_GetSpecificCollectionUsingOData”.

3.2 Integration Tests

The integration tests are tests that test the connection with the API. For each functionality of the SDK a API call must be made. In the integration tests these calls are tested. The integration tests are only created for the classes that communicate with the API (APIConnector and APIConnection).

3.3 Unit Tests

Unit Tests test different methods of the code. The unit tests are only created for the public methods that contain complex logic. Because the UAT tests the functionality, and unit tests test the methods that make this functionality, these tests have some overlap. The unit tests are especially handy for testing if a method is working correctly and finding broken functionality after maintenance or refactoring.

3.4 Performance Tests

The performance tests are tests made by the developer to test the runtime of different parts of the code. These tests and their runtime are computer depended, so these tests and their execution time differ per computer. The performance tests were created to find bottlenecks in performance and don’t have to be maintained or updated.

4 Basic Structure Software Development Kit

This chapter describes the structure of the Software Development Kit (SDK) for understanding the basics about the place of the SDK within the Exact environment. In the chapters below the specific layers of the SDK are showed, and global information about the SDK are provides.

4.1 Software Development Kit layers

In **Error! Reference source not found.** the different layers in the Exact Online environment and the place of the SDK in this environment is showed.

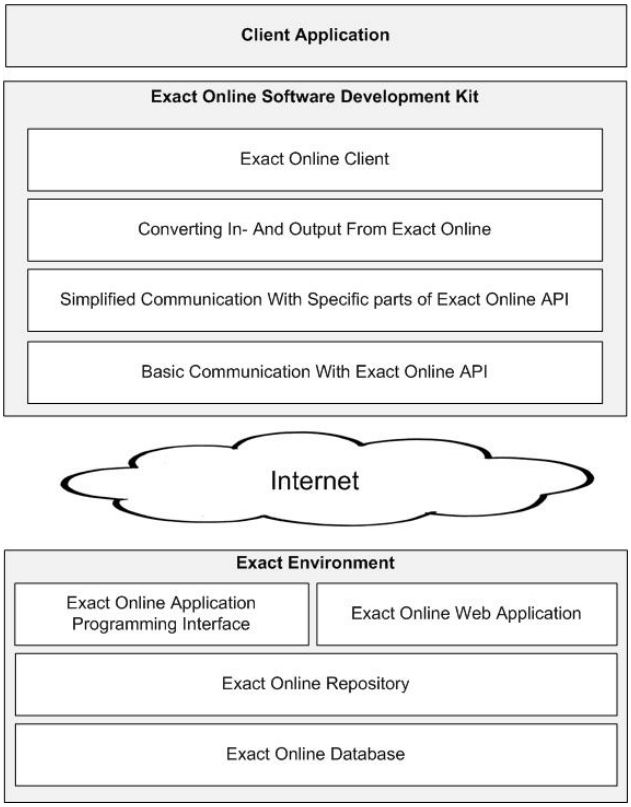


Figure 5

The SDK simplifies accessing data from Exact Online. The classes for basic communication & simplified communication with specific pars of, Exact Online API (APIConnector and APIController) form the basis for the SDK. Via these classes operations can be performed on the API. These operations are: GET, POST, PUT and DELETE (see chapter 5 'Manipulate Data Using REST Calls' for details about the internal operations).

The layer on top of that (Converting In- And Output from Exact Online) translates the responses (formatted in Json) to objects that the third party developer (user of the SDK) can work with. This can be C# 'dynamic' objects and typed objects from the Exact Online Web API Models.

The top layer, 'Exact Online Client', provides the third party developer with all the handles to perform operations on the Exact Online API in an easy way. The Exact Client calls the sub layer which calls the layer below to perform actions. Each layer can only call the layer below it. The layered approach makes the SDK better to maintain and more easy to understand.

4.2 Authenticating with Exact Online

For authentication with Exact Online the developer needs to specify the following things: The Exact Online Web URL for example https://start.exactonline.nl/ and a valid access token. Getting and refreshing the access token after it's expired is the task of the developer. For this task a different module is created that's outside the scope of the Exact Online SDK.

5 Get Collection of Entities

The main class for the developer to work with is 'ExactClient'. This class will let the developer access all the functionality of the SDK in an easy way. In the table (Tabel 12) below an example is showed of how the developer can get entities from Exact Online.

```
var accounts = client.For<Account>()
       .Where("Name+eq+'Test Testname'")
       .And("Code+eq+'123'")
       .Get();
```

Tabel 12

ExactClient uses the following classes are used to get a collection of entities (Figure 6).

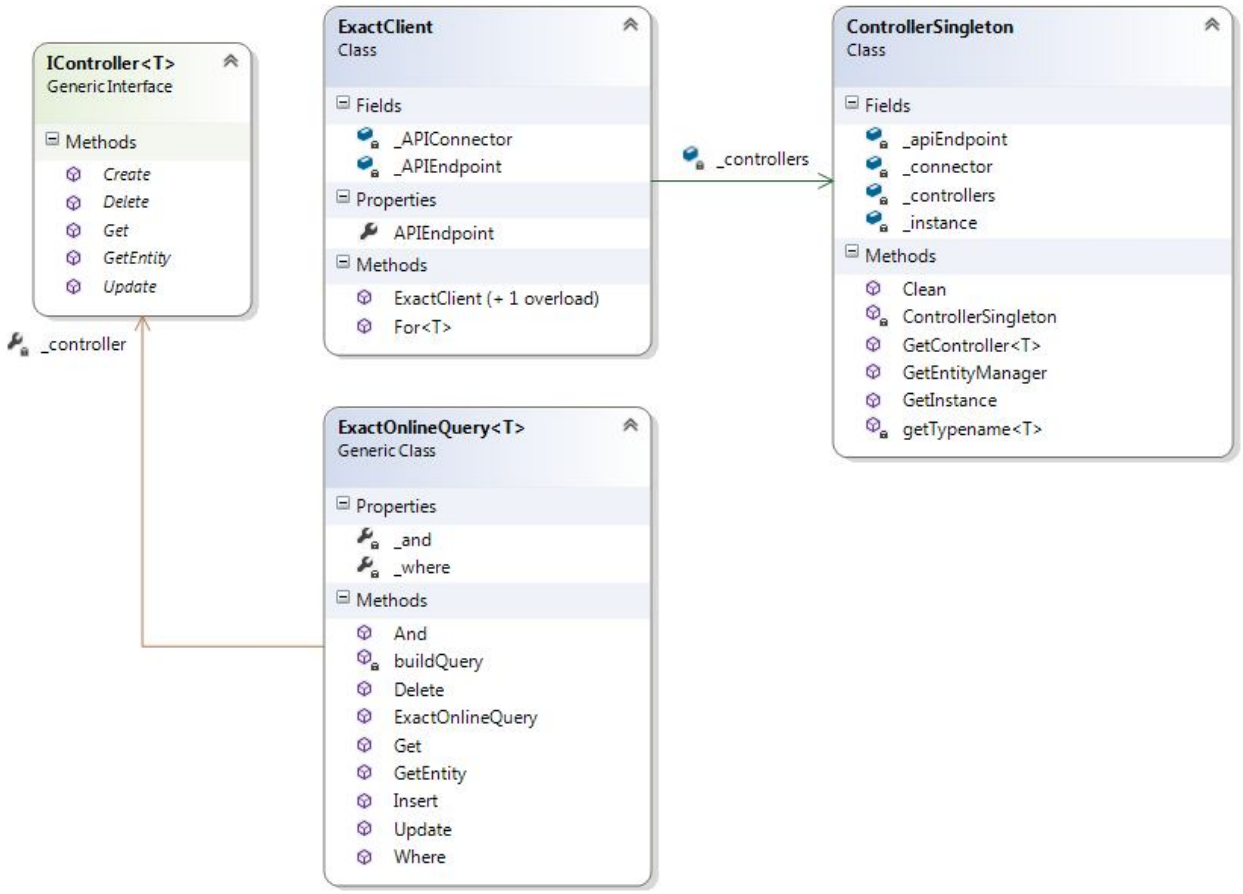


Figure 6

The sequence executed is displayed in the sequence diagram below (Figure 7).

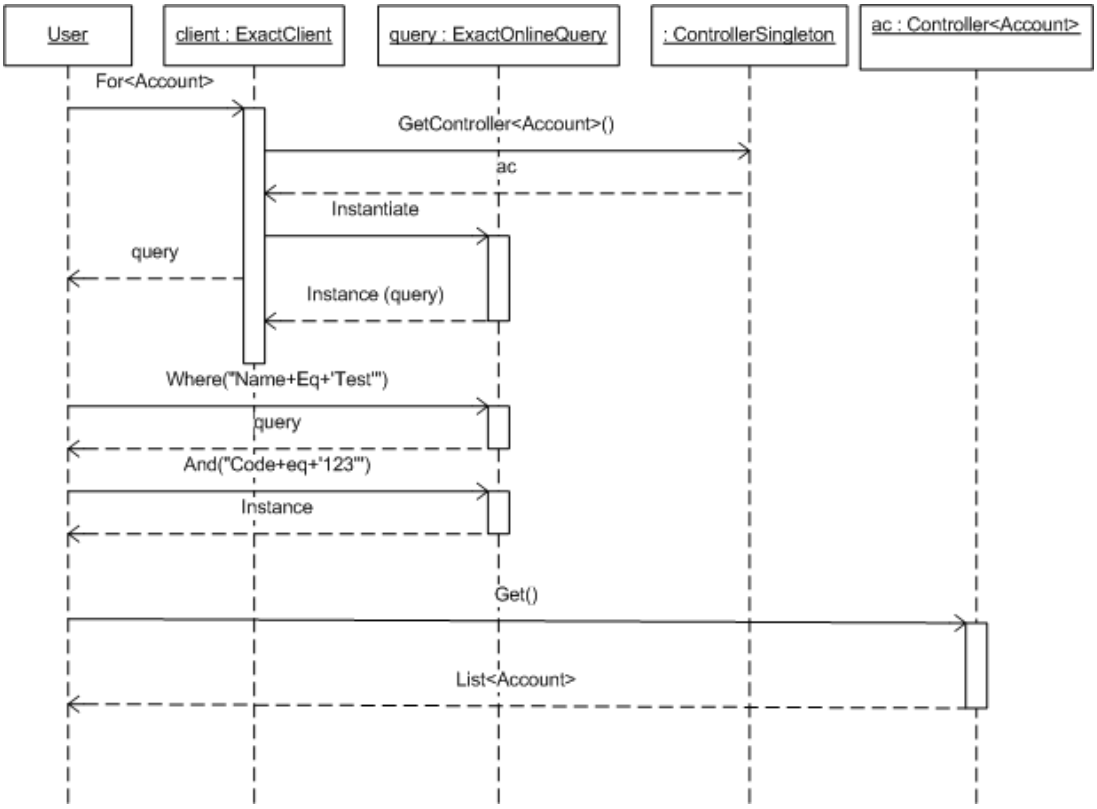


Figure 7

6 Managing Entities: Internal working Controller & EntityController

Figure 8 shows two classes: Controller<T> and EntityController<T>. These classes are used to manage the Exact Online Entities in the SDK. In the chapters below details about the implementation are provided.

6.1 Controller: Managing Entities

The Controller class is used to get, create, update and delete entities. The Controller works together with EntityController to create this functionality. The Controller class is responsible for managing a collection of entities by managing a collection of EntityController objects. For managing entities the following classes and interfaces are necessary (see Figure 8).

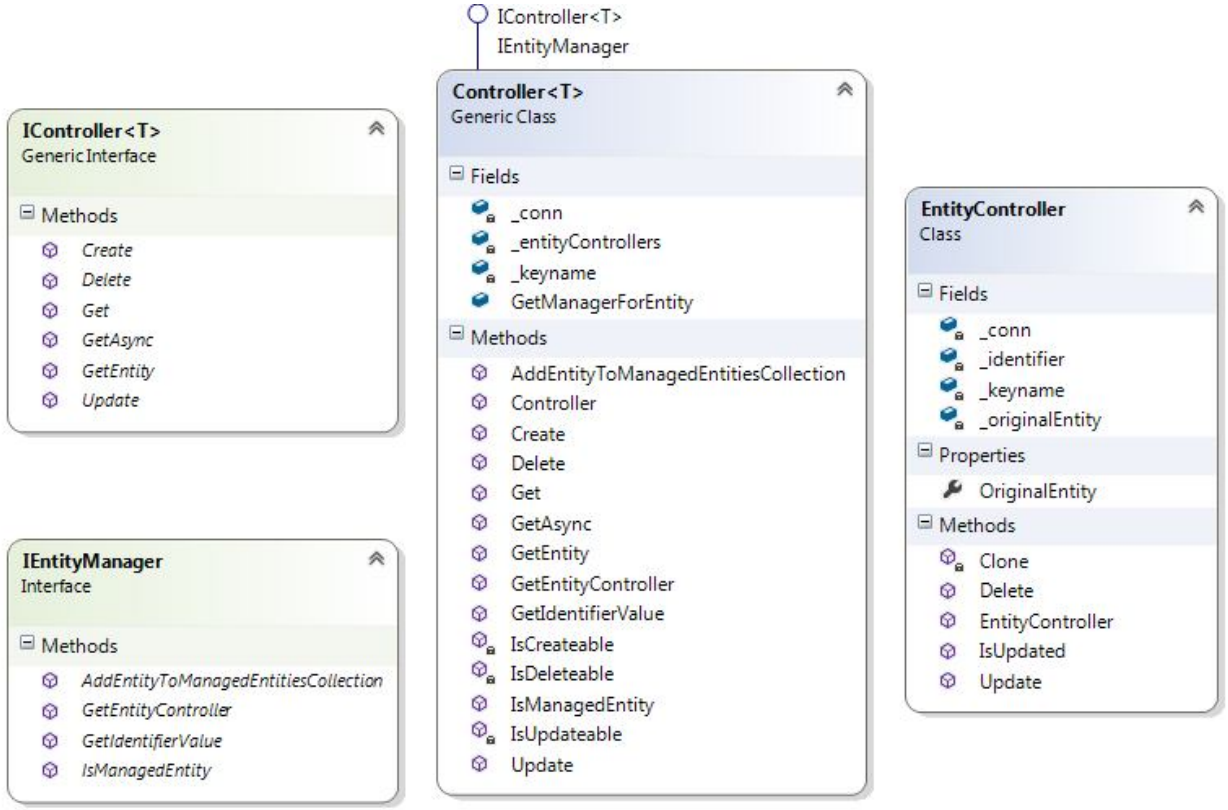


Figure 8

6.2 ControllerSingleton: Managing Controllers

The 'ControllerSingleton' class is used for managing all the different types of 'Controller' classes. Per entity type only one controller may manage the entities. This is to prevent conflicts that result from not updated entity states. The ControllerSingleton is responsible for instantiating Controllers of the requested type if they not yet exist, and for returning instances of the requested type. When instantiating a new instance of Controller the delegate 'ManagerForEntity' is set. In this way a Controller of and entity that has linked entities can request the correct Controller for the linked entities.

6.3 EntityController: Managing Entity State

To make it possible to only send altered/updated fields of an entity back to the API when updating the entity, the state of the entities must be managed. This is done by the EntityController class. For each entity that was requested by the developer using the Controller class, an instance of EntityController is created. These instances are managed by the Controller in the _entityControllers field. This is a Hashtable with the unique identifier as Key and the instance of EntityController as value. The Hashtable gets filled within he Get() and GetEntity() method by calling AddEntitiyToManagedEntitiesCollection() for each entity that is requested.

Within the EntityController a copy of the original entity is saved. This copy is used to create Json for updating only the altered fields of an entity. When an update is performed the current entity and the original entity gets sent to the method 'ConvertObjectTOJsonForUpdating()' of the EntityConverter class.

7 Exact Online Models

The Exact Online Models/Entities are classes that represent the entities in the Exact Online Web Application. These classes are reused from the Exact Online API that can be used by the (third party) developer to create new entities in an easy way. By using the Models the naming of fields is known and the developer doesn't have to know which fields there are in the entity and fewer mistakes can be made. To use the Exact Online API Models attributes that describe the type of entity and the type of fields must be added. The attributes and the procedure for adding new entities are described in the chapters below.

7.1 Attributes and Field Types

The table below (Table 7) shows the different types of attributes that specify fields of the models. The Models and their entities are written in Visual Basic because the API is also written in this language.

Description	Attribute
Supported Actions on the Entity	`<SupportedActionsSDK(True, True, True, True)>`
The field that contains the unique id	`<DataServiceKey("")>`
Specifies a read only field	`<SDKFieldType(FieldType.ReadOnly)>`

Table 12

- **Supported Actions**: These are the actions that can be performed on the entity (resource in API). This can be Create, Read, Update and/or Delete.
- **DataServiceKey**: Specifies the field that contains the unique ID. For the most entities this is the 'ID' field. This field will not be written to on create or update.
- **Readonly fields**: Specifies if a field can only be read. If a field is read-only the JsonConverter will not create Json for this field.

An example of the way these attributes must be implemented is shown in Table 13.

```
<DataServiceKey("ID")>
<SupportedActionsSDK(True, True, True, True)>

       Public Class SalesInvoiceLine

              <SDKFieldType(FieldType.ReadOnly)>
              Public Property ID As Guid

              Public Property InvoiceID As Guid
       End Class
```

Table 13: Example Attributes in SalesInvoiceLine

7.2 Scenario: Adding new entity

To make a new entity available in the Exact Online API the following actions must be taken:

1. Add class to "Exact.Web.Api.Models" namespace
2. Specify the actions that can be performed on the entity (Create, Read, Update & Delete)
3. Specify the read-only & reference fields

4. Add endpoint to ControllerSingleton class

To get the complete overview of the process to update the SDK see chapter 11.

7.3 Scenario: Changing Exact Online API Entities

When a change in the API entities is done, the according C# classes must be updated also for the SDK to work correctly with these entities. In the enumeration shown below the actions for each scenario are described:

- **Added new Entity**: Add new C# class representing the entity & specify field types (chapter 7.1)
- **Deleted Entity**: Delete associated C# class
- **Added Field**: Add field in C# class & specify the field type (chapter 7.1)
- **Change Field**: Change field in C# class & specify the field type (chapter 7.1)

8 Converting Entities: Serializing- and de-serializing Json

The SDK works closely with the Exact Online API to modify data. The API works with Json, so for each action that can be executed Json must be read or created. To do this the framework JSON.NET is used. The chapters below describe this process.

8.1 Convert Json to Object

To Convert Json to a C# Object with the correct type the following steps must be taken:

1. Get the response from the API
2. Cleanup the response from the API
3. Call EntityConvert.ConvertJsonToObject<T>() method with the clean response
    a. Call JsonConvert.DeserializeObject<T>() with the Json
    b. Return the return value of the method in step a

8.2 Convert Object to Json

To convert an object to Json the following steps need to be taken:

1. Get the entity
2. Call EntityConverter.ConvertObjectToJson<T>() with the entity as parameter
    a. Instantiate new JsonConverter
    b. Call JsonConvert.SerializeObject with the entity and the JsonConverter as parameter
    c. Return return value of method in step b

JsonConverter: JSON.NET doesn't provide a way to convert an object to Json that is compatible with the API. The JsonConverter must take the following things in account:

- **DateTime Fields**: These must be formatted as 0:yyyy-MM-ddTHH:mm
- **Reference Field**: These must only be written on update
- **ReadOnly Field**: These must not be written
- **Writing Linked Entities**: For every linked entity the JsonConverter must be called recursively

8.3 Errors & Exceptions

DateTime Exception: The API Returns a DateTime field in EPOCH timestamp ('/Date(1267747200000)/'). If the incorrect JavaScriptConverter is used in the APIResponseCleaner class (chapter 9) the JavaScriptSerializer class converts the EPOCH time to the timestamp in the format of the current language settings. To ensure correct formatting the JSSDateTimeConverter (subclass of JavaScriptConverter) class must be used in the methods GetJsonObject() and GetJsonArray() in the APIResponseCleaner class.

9 API Responses

A big part of the SDK is converting the API response (Json) to a C# object. The problem only, is that the API adds additional tags to the response. To convert the response to a C# object these tags have to be removed. The chapters below describe the two different types of responses and the way to cleanup these responses.

9.1 API Response Structure

The API Response exists out of multiple parts. Each of these parts are displayed in table below.

Part	Description
{}	Json Object
[]	Json Array
Key	The key describing the Json field
Value	The value inside the Json field

If a single entity is returned from the API it has the following structure (Figure 9). The whole object is formatted in Json. The actual Json Object is saved in the value of “results”.

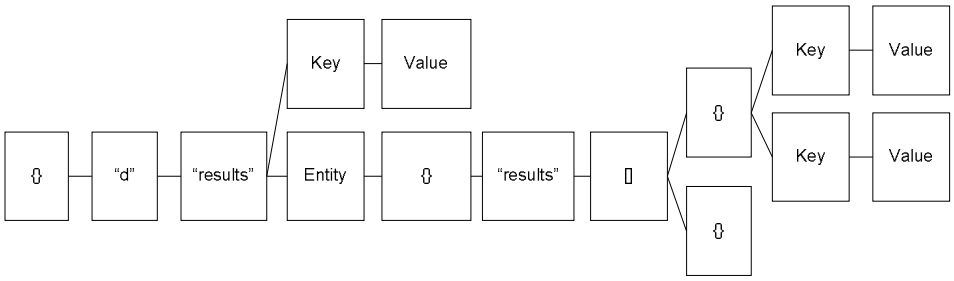


Figure 9

A collection of entities is returned by the API as a Json Array. This Json Array is stored in the “results” value. An example of the structure is displayed below (Figure 10):

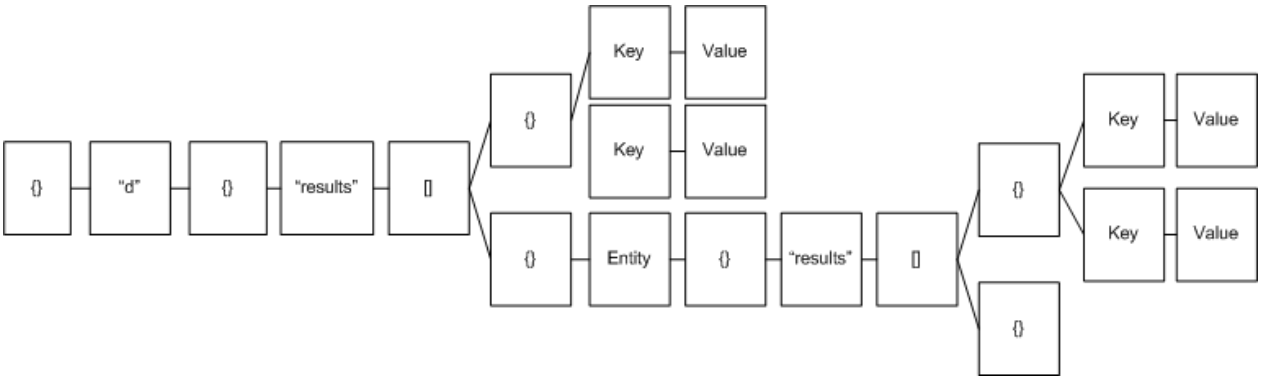


Figure 10

9.2 API Response Cleaner

'APIResponseCleaner' is a class where the response from the Exact Online API gets cleaned up. The Json string is deserialized using the JavaScriptSerializer class. This class creates a dictionary object with the other objects representing the Json. This object is used to create new (clean) json. Depending on the type of response (single entity or a collection of entities) a different flow is run through. This flow starts with 'GetJsonArray()' or 'GetJsonObject()' which both call dedicated methods that cleanup the response.

In picture Figure 11 and Figure 12 the application flow is displayed.

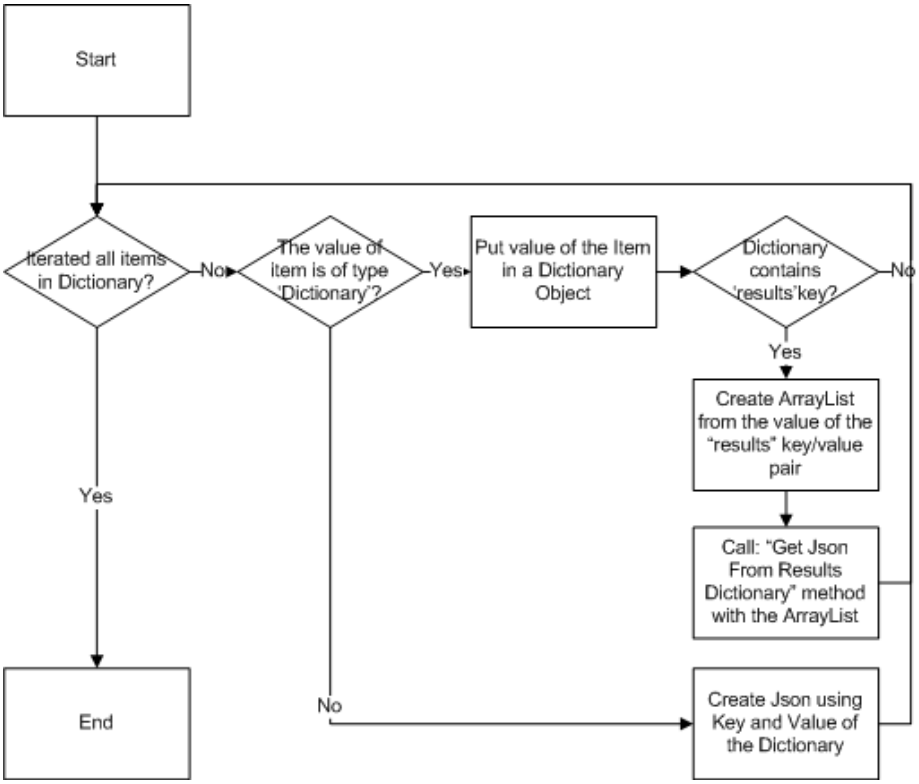


Figure 11: Get Json Object

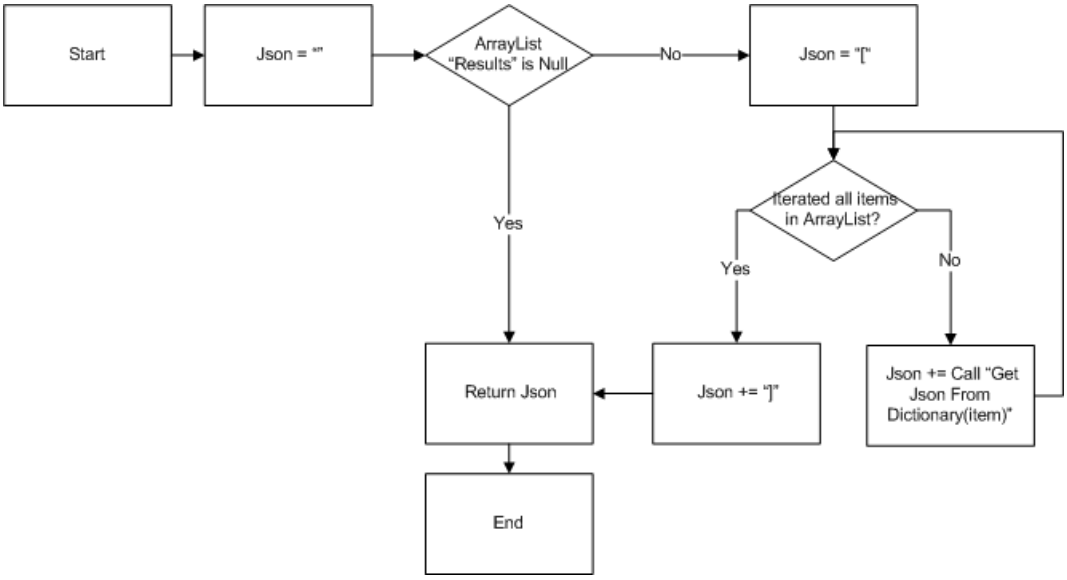


Figure 12: Get Json Array

9.3 Linked Entities

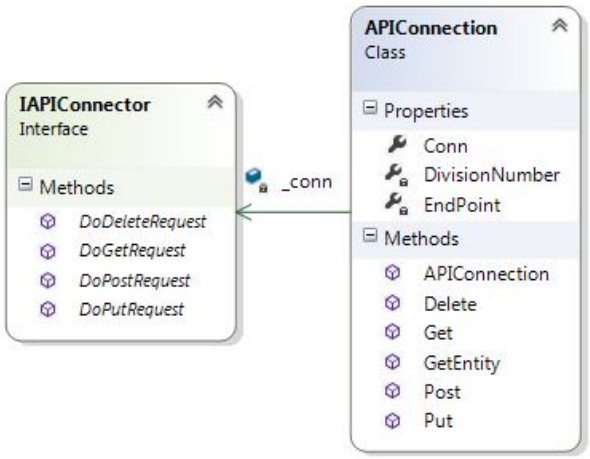
Linked Entities (for example: BankAccounts that are coupled to Account) are stored in the value of the specific Json field (for example the BankAccounts name/value pair) in the form of a Json Array. This Json Array is set within a "results" value.

9.4 Scenario: Exceptions & Error Codes

The APIResponseCleaner class depends on a Json Response described in chapter 0. This response comes from the Exact Online API and can only be invalid in the following scenarios:

- **The EOL API Changed the response structure**: Changes in the response cleaner must be made.
- **The API does not return Json**: The HTTP 'Content-Type' Attribute is set wrong.

10 Calling the Exact Online API



The base layer, described in chapter 4.1, consist out of APIConnector and APIConnection that are used to perform operations on the REST API. The class APIConnector uses an endpoint in the form of a uniform resource identifier (URI) to perform the HTTP requests GET, POST, PUT and DELETE to read, create, update or delete data in the API. APIConnection is the class that uses an instance of APIConnector to execute HTTP requests on a specific part of the API. In this way the developer doesn't have to specify the end point for each operation (read or update) for each type of entity.

Figure 13

10.1 Scenario: Usage APIConnection & APIConnector

In the table below (table 1) an example of a GET request with the APIConnector is given. First the endpoint will be specified and an instance of APIConnector will be instantiated. After that the request to a specific endpoint with possible parameters (empty in this case) can be done.

```
string endpoint = "https://start.exactonline.nl/api/v1/ 499156/crm/Accounts";
APIConnector conn = new APIConnector(endpoint);
conn.DoGetRequest(endpoint, string.Empty);
```

Table 14

In table 2 an example of the implementation of APIConnection is showed.

```
string endpoint = "https://start.exactonline.nl/api/v1/ 499156/crm/Accounts";
APIConnector connector = new APIConnector(endpoint);
APIConnection conn = new APIConnection(connector, endpoint);

string result = conn.Get(string.Empty);
```

Table 15

10.2 Scenario: Expired Access Token

When an access token is expired the AccessTokenManagerDelegate gets fired. In this code the user can refresh the access token. After this code is executed the latest request is sent to the API. This is tried 5 times. If the request returns 'Unauthorized' after that, the code sends an Exception to the developer.

11 Generating Models

The models are generated using a script that uses the Exact Online API Model classes and two CSV files. This script:

- Strips all the unnecessary tags
- Removes namespaces and comments
- Converts IQueryable to IEnumerable
- Adds attributes that describe the actions that can be performed on the entity (CRUD)
- Add read-only fields to the models

To do all the above the Console App 'AddSdkActionsToModels' has to be executed. The following steps have to be taken to do this correctly:

1. Create CSV Files*
    - 'Readonly Fields.csv' for with all the read-only fields
    - 'Entities.csv' for the actions that can be performed on the entities
2. In the code: Specify the path that the models are placed in
3. Run the program

After the program has finished please verify the modified date and time of the models, and check which ones are changed to find possible entities that are skipped.

*The CSV Files should be formatted in the following way:

Name	Format	Example
Entities.csv	Entity; Endpoint; Actions;	AccountClass;crm/AccountClasses;GET;
Readonly Fields.csv	Entity; Fieldname;	Absence;AbsenceTypeDescription;

Warning: The line should end with a semicolon ';'

12 Delivering new Version of the SDK

For delivering a new version of the SDK the following steps must be taken:

1. Writing User Acceptance Tests for new functionality
2. Writing Unit Tests for new methods
3. Writing Integration Tests (if necessary)
4. Add endpoint to ControllerSingleton class
5. Writing new code
6. Testing code with tests from step 1-3
7. If Succeed: Build Project & Export DLL
8. Update Documentation

Afstudeerplan

Informatie afstudeerder en gastbedrijf

Afstudeerblok: 2013-2.2 (start uiterlijk 18 november 2013)

Startdatum uitvoering afstudeeropdracht: 28 oktober 2013

Inleverdatum afstudeerdossier volgens jaarrooster: 24 maart 2014

Studentnummer: 10073272

Achternaam: dhr. Brandt

Voorletters: A.S.R.

Roepnaam: Aschwin

Adres: Bosboom Toussaintplein 159

Postcode: 2624DK

Woonplaats: Delft

Telefoonnummer: 06 15 60 58 42

Mobiel nummer: 06 15 60 58 42

Privé emailadres: aschwin.brandt@gmail.com

Opleiding: Informatica

Locatie: Den Haag

Variant: Voltijd

Naam studieloopbaanbegeleider: R. Bechet-Tjoonk

Naam begeleidend examinator: B. Kuiper

Naam tweede examinator: G. Tuk

Naam bedrijf: Exact

Afdeling bedrijf: Connectivity

Bezoekadres bedrijf: Molengraaffsingel 33

Postcode bezoekadres: 2629 JD

Postbusnummer:

Postcode postbusnummer:

Plaats: Delft

Telefoon bedrijf: 015 - 711 51 00

Telefax bedrijf: 015 - 711 51 10

Internetsite bedrijf: http://www.exact.nl

Achternaam opdrachtgever: dhr. Wieringa

Voorletters opdrachtgever: E.

Titulatuur opdrachtgever: ing.

Functie opdrachtgever: Manager Product Management

Doorkiesnummer opdrachtgever: 015 711 50 05

Email opdrachtgever: edgar.wieringa@exact.com

Achternaam bedrijfsmentor: dhr. Wieringa

Voorletters bedrijfsmentor: E.

Titulatuur bedrijfsmentor: ing.

Functie bedrijfsmentor: Manager Product Management

Doorkiesnummer bedrijfsmentor: 015 711 50 05

Email bedrijfsmentor: edgar.wieringa@exact.com

Doorkiesnummer afstudeerder:

Functie afstudeerder (deeltijd/duaal): Voltijd

Titel afstudeeropdracht: Exact Online - Client Software Development Kit Development

Opdrachtomschrijving

1. **Bedrijf**

Exact Holding is een beursgenoteerd Nederlands Softwarebedrijf dat is opgericht in 1984. Momenteel heeft Exact 1900 werknemers die verspreid zitten over vestigingen in 20 landen. Exact richtte zich in het begin alleen op boekhoudsoftware voor midden- en kleinbedrijf, maar heeft over de jaren heen meerdere producten ontwikkeld die ook andere bedrijfsprocessen ondersteunen (projectmanagement, fabricage, klantenservice, salarisadministratie etc.). Het doel van Exact is om ondernemers helpen hun doelen te realiseren door er onder andere voor te zorgen dat de ondernemer 24/7 per dag overzicht heeft in de (financiële) situatie van zijn of haar bedrijf.

Exact is sinds 2005 bezig met de cloud oplossing Exact Online. Hiermee kunnen ondernemers hun producten gebruiken via de webbrowser zonder eerst hard- en software aan te schaffen en te installeren. Alle data wordt opgeslagen in datacenters die door partners van Exact worden beheerd, waardoor deze data altijd en overal beschikbaar is.

Exact Online heeft een API waarmee externe software ontwikkelaars data kunnen opvragen. Deze koppeling wordt met name gebruikt voor mobiele en cloud applicaties en wordt beheerd en uitgebreid door de afdeling ‘Connectivity’ naar de eisen en wensen van deze externe partijen. De ontwikkeling gebeurt in sprints van twee weken volgens de software ontwikkeling methode Scrum.

2. **Probleemstelling**

Exact biedt externe partijen ondersteuning in het bouwen van software dat samenwerkt met Exact Online. Dit zodat het voor deze partijen gemakkelijker en daardoor aantrekkelijker wordt om een koppeling met een ander softwarepakket te ontwikkelen. Momenteel liggen er plannen op tafel voor de ontwikkeling van een nieuw 'Developer Documentation Platform'. Hiermee kunnen externe partijen gemakkelijk online kennis maken met de Application Programming Interface (API) om snel een beeld te krijgen krijgen van de mogelijkheden. Dit gaat zowel door documentatie in de vorm van tekst gebeuren, als het aanleveren van een web applicatie waarmee in de browser opdrachten uitgevoerd kunnen worden op de API.

Om het voor externe software partijen nog makkelijker te maken om applicaties te ontwikkelen die communiceren met Exact Online moet, naast de eerder beschreven onderdelen, ook een Software Development Kit (SDK) worden gemaakt die de ontwikkelaars kan helpen in het realiseren van hun software. Deze SDK zal geschreven worden in C#, en zal dienen als wrapper op de Exact Online API die het communiceren met de API versimpelt. De SDK zal opgeleverd worden als DLL.

Momenteel kan er wel al gecommuniceerd worden met delen van Exact Online via de API, maar dit is nog relatief moeilijk en tijdsintensief want het vergt veel kennis over protocollen als SOAP, JSON, REST en de API zelf. Na realisatie van de SDK kunnen externe ontwikkelaars de SDK importeren en direct werken met objecten die terug worden gegeven uit Exact Online zonder zich teveel zorgen te hoeven maken over al deze stappen.

De opdracht voor de realisatie van de Software Development Kit voor Exact Online staat in de planning voor begin volgend jaar (2014), en heeft daarom een lage urgentie. Dit is echter wel een opdracht waar veel vraag naar is en veel bedrijven profijt van kunnen hebben, dus de opdracht heeft een hoge prioriteit.

3. **Doelstelling van de afstudeeropdracht**

Op 24 maart 2014 is gerealiseerd: een detail ontwerp die opbouw* van de hele SDK beschrijft en (een deel van) de code van de SDK met unittests zoals afgesproken met de opdrachtgever.

*Het is waarschijnlijk niet mogelijk om alle functionaliteit van de API te realiseren in de daarvoor beschikbare tijd. Om die reden is besloten om wel de structuur/opbouw van de SDK voor zover nodig (het is niet nuttig om elke class te modelleren als dit slechts meer van hetzelfde is) helemaal uit te werken.

4. **Resultaat**

Na realisatie van de Software Development Kit kunnen externe partijen gemakkelijker software ontwikkelen dat gebruikt maakt van de functionaliteiten van Exact Online. Omdat het makkelijker wordt om met de data van Exact Online te werken doordat er geen kennis meer nodig is over oAuth, JSON etc. zullen meer bedrijven hier interesse in hebben, wat uiteindelijk de marktpositie van Exact kan versterken. Aanpassingen en uitbreidingen kunnen door de architectuur gemakkelijk en snel worden doorgevoerd waardoor nieuwe eisen en wensen van externe partijen wat betreft functionaliteit snel kunnen worden toegevoegd, en aanpassingen die in de API worden gedaan snel ook in de SDK kunnen worden doorgevoerd. Het snel kunnen inspringen op nieuwe eisen en wensen van externe software ontwikkelaars verhoogt de tevredenheid, wat uiteindelijk hun relatie met Exact, en de bereidwilligheid om software te bouwen die de markpositie van Exact versterkt vergroot.

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

Werkzaamheden	**Aantal werkdagen**
Oriënteren op de opdracht / Inwerken bij bedrijf & Plan van aanpak opstellen	5
Opstellen mastertestplan	10
Opstellen detailtestplan	5
Opstellen detailontwerp	15
Realiseren van framework	5
Realiseren van de regressietesten*	10
Realiseren implementatie van de software	10

Opleveren beta versie	5
Beheren van wijzigingen en door ontwikkelen van de SDK	5
Totaal	**70**

* Aan de hand van de nu beschikbare informatie zal de ontwikkelmethode waarschijnlijk Test Driven Development of Unfied Process zijn. Het idee is om eerst een framework van lege classes en methoden te bouwen die nog geen body hebben, en aan de hand daarvan regressietesten/unit testen te schrijven. Dit omdat er geen grafische interface wordt ontwikkeld waaraan ontleed kan worden of de ontwikkelde software correct werkt, omdat de software stabiel moet draaien en zo min mogelijk fouten moet bevatten, en er herbruikbare tests moeten worden geschreven die kunnen worden gebruikt als de SDK wordt uitgebreid of aangepast.

6. Op te leveren (tussen)producten

- Plan van aanpak
- Mastertestplan
- Detailtestplan
- Detailontwerp
    - Lagendiagram voor toelichting ontwikkeling SDK
    - Class Diagram voor weergeven architectuur en gekozen design patterns
    - (Indien nodig) Sequentie Diagram voor belangrijke/ingewikkelde instantie constructies
    - Proces Cyclus Diagram voor belangrijke/ingewikkelde features
- Regressietesten/Unit Tests
- Geïmplementeerde methodes
- C# Library die geïmporteerd kan worden in een C# project

7. Te demonstreren competenties en wijze waarop

3.2 Ontwerpen Systeemdeel - Complexiteit: Complex niveau: 4

Voor de beroepstaak ‘Ontwerpen Systeemdeel’ zal de student de structuur van de te realiseren Software Development Kit ontwerpen. Er zal worden nagedacht over hoe deze het beste gerealiseerd

kan worden voor optimale uitbreidbaarheid en onder houdbaarheid en zal worden gekeken welke design patterns hieraan kunnen bijdragen. De student zal kijken naar de decompositie van de verschillende functionaliteiten in de Exact Online API en zal aan de hand van deze decompositie een structuur opzetten die het beste aansluit bij de huidige API. Ook zal worden gekeken naar logische indeling van de verschillende functies, het toepassen van een scheiding van model en controller, en zal rekening gehouden moeten worden met de cashing eigenschappen van de RESTful API.

De beroepstaak 'Ontwerpen Systeemdeel' zal worden bewezen door middel van het detailontwerp waarin de structuur door middel van diagrammen en tekstuele toelichting wordt beschreven. De te gebruiken diagrammen zullen onder andere een lagendiagram, class diagram met uitleg van de te gebruiken design patterns, sequentie diagram en proces cyclus diagram zijn.

3.4 Initiëren en plannen van het testproces – Complexiteit: Lastig, niveau: 3

Voordat aan de codering moet worden begonnen, moet omdat volgens Test Driven Development (TDD) zal worden gewerkt, eerst een mastertestplan worden geschreven waarin de strategie van het hele testtraject staat uitgeschreven. Hierin wordt onder andere behandeld welke kwaliteitsattributen getest moeten worden, welke risico's er liggen, welke testontwerp technieken gebruikt zullen worden etc. Vervolgens moet per testsoort een detailtestplan worden opgesteld. Hierin worden uitspraken gedaan over hoe de verschillende onderdelen getest moeten worden.

TestGoal zal waarschijnlijk worden gebruikt als uitgangspunt voor het ontwerpen en uitvoeren van het testproces. De te gebruiken testontwerptechnieken zullen bekend worden na de orientatie op de opdracht in de eerste week waar gekeken gaat worden naar de testbaarheid van verschillende onderdelen van het te realiseren systeem, de risico's en de eisen van de stakeholders met betrekking tot het testproces. De nadruk zal liggen op de herhaalbaarheid van de testen, het testframework dat voor de ontwikkeling en de doorontwikkeling van de SDK gebruik zal gaan worden, en de testen die beschrijven wanneer de gerealiseerde code goed is voor productie.

3.3 Bouwen applicatie – Complexiteit: Complex, niveau: 4

Aan de hand van de eerder gemaakte ontwerpen zal de student de software realiseren. Eerst zal hij het bedachte framework uitschrijven zonder de implementatie van de methodes. Vervolgens zal hij

aan de hand van de methodes unit tests opstellen die gebruikt worden om te controleren of de methode werkt. Daarna zal hij de methodes op volgorde van prioriteit implementeren.

Het bouwen van de applicatie zal gaan volgens de software ontwikkelmethode Test Driven Development (TDD). De code zal worden geschreven in C# gebruik makend van het .NET Framework en Visual Studio. Omdat het Software Development Kit moet worden doorontwikkeld in de toekomst is het noodzakelijk om te werken met een versiebeheertool.

Tijdens het bouwen van de applicatie moet gewerkt worden met technieken als oAuth, zal rekening gehouden moeten worden met de software architectuur Representational State Transfer (REST) die wordt gebruikt voor de Exact Online API. Ook zal worden gebruikt met datacollecties en met LINQ voor het bevragen van die datacollecties.