- **STUDENT:** Nora Kamoen
- **OPLEIDING:** Professionele Bachelor Elektronica-ICT, afstudeerrichting ICT

- **TITEL BACHELORPROEF:** BACKENDONTWIKKELING VAN BEHEERSOFTWARE VOOR REALDOLMEN EDUCATION IN .NET MET EF CODE FIRST, REPOSITORY PATTERN EN AUTOFAC

- **TREFWOORDEN:** Backend, WebAPI, .NET, Dependency Injection, Repositories, EF Code First, DTO's, Owin, XUnit, Dacpac

- **STAGEPLAATS:** RealDolmen
- **STAGELEID(ST)ER**: Joris Maervoet
- **STAGEMENTOR:** Tom Eeraerts

SAMENVATTING

Deze bachelorproef behandelt de ontwikkeling van een backend-applicatie voor RealDolmen Education voor het beheer van cursussen. Een belangrijk aspect hiervan is de modulaire opbouw. Ook is het noodzakelijk om de gehele ontwikkeling voldoende af te stemmen op de business kant zodat het resultaat inzake de datastructuur en functionaliteit voor hen het meest optimaal is. De volledige ontwikkeling gebeurt in .NET gebruik makend van EF Code First, Repository Pattern en Dependency Injection.

Voor de ontwikkeling wordt vertrokken vanuit de principes van Domain Driven Design en wordt gebruik gemaakt van Entiteiten, Aggregate Roots, Repositories en Services. Binnen de entiteiten wordt een opdeling gemaakt in Aggregate Roots en losse entiteiten. De Aggregate Roots zijn 'Course', 'Company' en 'Session'. Binnen elke entiteit zijn ook velden aanwezig die voorzien in logging en soft delete. De resulterende entiteiten worden dan omgezet naar een database gebruik makend van Entity Framework Code First. Op deze manier is het heel eenvoudig aanpassingen te maken in de datastructuur.

De toegang tot de database wordt voorzien door Repositories. Hierbij wordt vertrokken van een generieke Repository waarvan alle andere kunnen overerven. Wanneer een andere implementatie nodig is kan deze uitgewerkt worden in de specifieke Repository. Binnen de repositories zijn methodes aanwezig voor alle CRUD-acties.
Doorheen de volledige applicatie wordt gebruik gemaakt van Dependency Injection aan de hand van Autofac. Alle dependencies worden geregistreerd aan de hand van hun interfaces in een container. Autofac levert gebruik makend van deze container de juiste dependencies aan de juiste klassen.

Voor de WebAPI wordt gebruik gemaakt van Owin middleware. Hierdoor kan de API draaien op eender welke server. Alle configuraties binnen de API dienen dan toegevoegd te worden aan de Owin middleware zodat de server deze zo kan consulteren.
Binnen de controllers zijn GET, POST, DELETE en PATCH-methodes aanwezig. Deze methodes geven enkel DTO's terug naar de client. Zo heeft de front-end enkel wat nodig is.

De code wordt tevens uitvoerig getest gebruik makend van het XUnit framework voor .NET en Owintesting. Binnen XUnit worden testklassen geïnitialiseerd aan de hand van een constructor en afgebroken via de IDisposable interface. Er kan ook een onderscheid gemaakt worden tussen Facts en Theories. Theories kunnen variabele data meegeleverd krijgen binnen de test.
Voor het deployment wordt gebruik gemaakt van Visual Studio Team Services en een Dacpac voor het online brengen van de database.

Woord vooraf

Met het uitwerken van deze bachelorproef rond ik mijn opleiding 'Professionele Bachelor Elektronica-ICT' met afstudeerrichting ICT aan de Odisee Hogeschool af. Om deze bachelorproef tot stand te brengen liep ik stage binnen de dienst RealDolmen Education van het bedrijf RealDolmen. De ontwikkeling van deze bachelorproef was een proces van vallen en opstaan maar met de hulp en steun van verscheidene mensen heb ik deze tot een goed einde kunnen brengen.

Als eerste wens ik mijn stagementor Tom Eeraerts, ICT-Trainer binnen RealDolmen Education, te bedanken. Doorheen de stage zorgde hij ervoor dat ik genoeg uitdaging had aan de opdracht en een hele hoop kennis kon opdoen aan de hand van de nieuwe gebruikte technologieën. Niet enkel op technisch vlak stond hij voor me klaar maar hij was er ook om de stress te verlichten en me moed te geven wanneer het wat minder ging.

Naast Tom Eeraerts wil ik ook Tom Knockaert bedanken, Unit Manager Education & Technical Writers binnen RealDolmen. Ook hij stond gedurende het verloop van de stage klaar met raad en daad over de ontwikkeling van de opdracht. Hij ontfermde zich over de business kant van mijn opdracht en wist elk onderdeel duidelijk aan me uit te leggen.

Tevens wens ik de heer Joris Maervoet, Lector opleiding professionele bachelor Elektronica-ICT, te bedanken. Als stagebegeleider gaf hij doorheen de ontwikkeling van mijn bachelorproef uitgebreide feedback, ondersteuning en de nodige begeleiding om deze bachelorproef te maken tot wat ze nu is.

Ook wil ik mijn ouders, zus en vriend bedanken voor de onvoorwaardelijke steun en het eeuwige geduld tijdens deze drukke periode. Zij stonden steeds voor me klaar om de druk te verlichten en me ervan te verzekeren dat ik goed bezig was.

Tot slot wens ik Tom Eeraerts, Tom Knockaert, Joris Maervoet en mijn mama, Michèle Baeck, nog eens extra te bedanken voor de tijd die zij besteed hebben aan het nalezen van mijn bachelorproef.

Inhoud

Woord vooraf 3

Inhoud 5

Lijst met gebruikte afkortingen 9

1 Voorstelling van het stagebedrijf 11

1.1 Ontstaan en geschiedenis 11

1.2 Over RealDolmen 11

1.3 Visie en missie 12

1.3.1 Visie 12

1.3.2 Missie 12

1.4 Bedrijfsorganisatie 13

1.4.1 Vestigingen 13

1.4.2 Organisatie 13

2 Bachelorproefopdracht 15

2.1 Startsituatie 15

2.2 Gewenste situatie 15

2.2.1 Technologisch luik 16

3 Actieplan 17

3.1 Gannt chart 18

4 Voorstudie 23

4.1 Oude applicatie 24

4.2 Domain-Driven Design 25

4.2.1 In het kort 26

4.3 Entiteiten 27

4.3.1 Aggregate roots 27

4.4 Database changes 28

4.4.1 Change tracking 28
4.4.2 Soft Deletes 29
4.4.2.1 Soft delete database interceptor 29
4.5 Entity Framework Code First..30
4.5.1 Eager loading vs. Lazy loading 30
4.6 Repository Pattern 31
4.6.1 Unit Of Work 32
4.6.2 Data Mapper 32
4.7 Dependency Injection 33
4.7.1 IoC verduidelijking..33
4.7.2 Autofac..34
4.8 Owin 36
4.9 Testing 37
4.9.1 XUnit 37
4.9.2 Owintesting 37
4.9.3 Mocking via Moq 38
4.10 Dacpac 39
5 Praktische uitwerking..41
5.1 Domeinlaag..42
5.1.1 Entiteiten 42
5.1.1.1 Functionaliteit 43
5.1.2 EF Code First..44
5.2 Repositories 47
5.3 Soft Delete en Logging 49
5.3.1 Evolutie soft delete..49
5.3.2 Evolutie Logging 49
5.4 Data Transfer Objects 51
5.5 WebAPI 53
5.5.1 Controllers..53
5.5.2 Put vs. Patch..53
5.5.3 RESTful API..54
5.5.4 Owin..55

5.6 Services 57

5.6.1 Opbouw Aggregate Root services 57

5.6.2 CourseCategory service 58

5.7 Dependency Injection 59

5.8 Testen 61

5.8.1 Tests 61

5.8.1.1 XUnit 61

5.8.1.2 Owintesting 62

5.8.2 Mocking database 63

5.9 Application Insights 64

5.10 Visual Studio Team Services 66

5.10.1 Continuous Integration 66

5.10.2 Continuous Delivery 66

5.10.3 Agile 66

Algemeen besluit 69

Literatuurlijst 71

Bibliografie 73

Bijlagen 77

B-1 Entiteiten v1.0 78

B-2 Entiteiten v2.0 81

B-3 Entiteiten v3.2 85

B-4 Entiteiten v4.0 89

Lijst met gebruikte afkortingen

API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CRUD	Create, Read, Update, Delete
DDD	Domain Driven Design
DTO	Data Transfer Object
EF	Entity Framework
IoC	Inversion of Control
TFS	Team Foundation Server
UOW	Unit of Work
VB	Visual Basic

1 Voorstelling van het stagebedrijf

1.1 Ontstaan en geschiedenis

RealDolmen is ontstaan in 2008 na de fusie van Real Software en Dolmen. Beide bedrijven richtten zich reeds in het verleden op consultancy en het aanbieden van ICT-gerichte oplossingen voor andere bedrijven.

Real Software werd opgericht in 1986 door Rudy Hageman en Leo Meuris maar ging ten onder na de overname van ‘TAVA Technologies’, een Amerikaans softwarebedrijf. In 1999 verliet Leo Meuris het bedrijf en in 2004 haalde ‘The Gores Group’ de meerderheid van de aandelen binnen om vervolgens in 2008 te fusioneren met Dolmen tot RealDolmen [1].

Dolmen Computer Applications werd opgericht in 1982 door de Colruyt-groep. In ’97 werd Dolmen afgesplitst van Colruyt en werd het personeel dat enkel voor Colruyt werkte ondergebracht in een nieuw dochterbedrijf ‘InfoCo’ dat heden ten dage nog steeds actief is. Dolmen deed verschillende overnames onder meer van GSE, Datasoft Solutions en JConsults international. In 2004 was Colruyt als bedrijf niet langer meer aandeelhouder van Dolmen maar bezat de familie Colruyt wel nog steeds 40% van de aandelen. In 2008 fusioneerde Dolmen dan met Real Software en ontstond RealDolmen met de familie Colruyt aan het hoofd [2].

1.2 Over RealDolmen

RealDolmen is een bedrijf actief binnen de ICT-sector. Hun troef is dat ze niet de focus leggen op de technologie zelf maar op hun klanten en wat die verwachten van de gewenste technologie. Vanuit dit gegeven werkt RealDolmen naar de juiste toepassing van de technologie.

RealDolmen is ervan overtuigd dat ICT het werk eenvoudiger en aangenamer moet maken en ervoor moet zorgen dat hun klant efficiënter te werk kan gaan. Ze zijn ervan overtuigd dat wanneer ICT eenvoudiger gemaakt wordt het vanzelf ook efficiënter wordt. Om ervoor te zorgen dat ICT perfect werkt voor hun klanten staan ze ook dicht bij hen. Ze leren hun klanten beter kennen zodat ze ook hun drijfveren leren kennen waarom ze komen werken en wat ze verlangen van hun job.

RealDolmen is een echte expert in ICT-technologieën en de uitvoering van ICT-projecten en outsourcingstrajecten. Hiervoor neemt RealDolmen de complexiteit van het ICT-domein weg waardoor klanten zich kunnen focussen op de hoofdtaak van het bedrijf waarin ze werken en hiermee ook hun werk aangenamer wordt [3].

Het bedrijf is opgebouwd uit vier grote pijlers:

- IT outsourcing [4]: binnen de activiteit van Business solutions biedt RealDolmen eindoplossingen met eigen software of softwarepakketten aangeleverd door derden.
- Professional services [5]: onder Professional services worden zowel softwareontwikkeling als analyse, testing, integratie en infrastructuuractiviteiten gerekend. Ook biedt RealDolmen hierbij courseware, ontwikkelingsmethodologieën, projectmanagement methodologieën, software building blocks en dergelijke aan.
- Products & Licenses [6]: deze activiteit zorgt voor de aanlevering van hardware producten en softwarelicenties binnen domeinen als data center, front-end, networking & security, hardware en software procurement en Unified Communications.
- Consulting: binnen deze tak biedt RealDolmen ondersteuning voor 'Business & IT Alignment', 'Hybrid Cloud', 'Data Insights', 'Agile Business Processes', 'Engaged Workplace' en 'Customer Centricity'.

1.3 Visie en missie

1.3.1 Visie

De visie van RealDolmen wordt als volgt op de site beschreven [1a]:

"In de lokale markten en domeinen waarin we actief zijn, willen we de referentie zijn inzake geïntegreerde oplossingen die de gehele ICT-levenscyclus ondersteunen.

- Referentie: de preferentiële en betrouwbare keuze voor klanten, partners en werknemers
- Lokaal: nabijheid tot onze klanten in de Benelux
- Geïntegreerde oplossingen: volledige ICT-aanbod dat de hele levenscyclus bestrijkt, inclusief infrastructuur, toepassingen en communicatie
- Gehele ICT-levenscyclus: het ondersteunen van alle plan-build-run (design-deploy-maintain) activiteiten."

1.3.2 Missie

Naast de visie heeft RealDolmen ook een heel duidelijke missie beschreven op hun site [1a]:

"We make ICT work for your business"

1.4 Bedrijfsorganisatie

1.4.1 Vestigingen

RealDolmen heeft verschillende vestigingen in België, Nederland en ook Luxemburg. De hoofdzetel bevindt zich in Huizingen en deze stuurt de operaties binnen België en Nederland [7].

In België werkt RealDolmen via regionale satellietkantoren.

De internationale dochterondernemingen zijn

- Real Solutions DA in Luxemburg: vooral actief in de financiële sector.

1.4.2 Organisatie

RealDolmen is opgedeeld in verschillende afdelingen of sectoren. Hoofdzakelijk biedt RealDolmen consultancy in de brede zin van het woord. Niet enkel het ontwikkelen van applicaties maar ook integratie van verschillende software- of hardware pakketten als het ontwikkelen van bedrijfsnetwerken en een goede manier voor het beheren ervan worden hierin aangeboden.

De drie grote pijlers binnen het consultancy gegeven zijn: Microsoft die alles omtrent .NET, Azure, SharePoint en andere Microsoft toepassingen voor zich neemt en binnen deze sector experts aflevert. Tevens is er ook een grote afdeling die alles omtrent Java-applicaties verzorgt en een afdeling die alles inzake CRM behandelt.

Algemeen is er een verzameling aan developers, analisten, projectmanagers, systeembeheerders, architecten, netwerkspecialisten, … Een heel breed gamma aan functies die allemaal een plaats vinden binnen de structuur van RealDolmen.

Een andere afdeling is RealDolmen Education. Deze voorziet cursussen, opleidingen en certificaten in verschillende vakken binnen IT. Ze voorzien opleidingen voor hun eigen personeel maar tevens ook aan externe bedrijven. Binnen deze afdeling doorloop ik mijn stage.

2 Bachelorproefopdracht

Deze bachelorproef behandelt de ontwikkeling van een backend applicatie voor RealDolmen Education. Deze applicatie is verantwoordelijk voor het beheren van

- Cursussen
- Sessies
- Klanten (bedrijven) met hun personeel dat deelneemt aan de cursussen
- Trainers
- Locaties
- Het cursusmateriaal
- …

2.1 Startsituatie

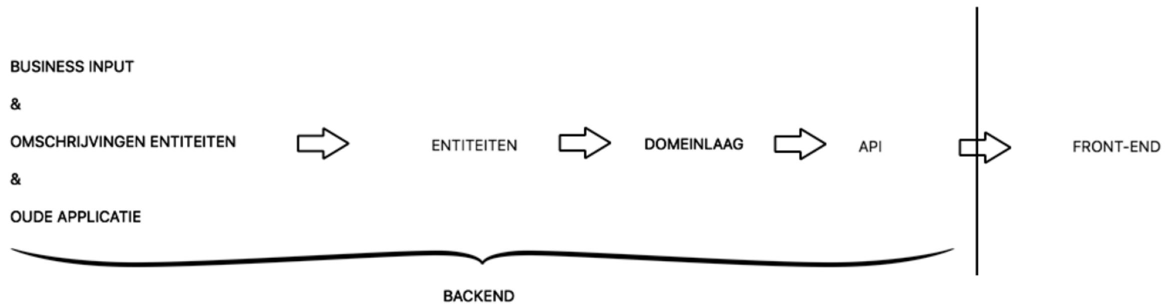
RealDolmen Education heeft reeds een bestaande applicatie voor het beheren van de cursussen. Echter voldoet deze applicatie niet aan de huidige noden omdat deze niet modulair is opgebouwd. Hiernaast is de structuur van de applicatie verouderd daar RealDolmen Education als dienst een andere richting uit wil voor het beheren van de verschillende klanten, deelnemers en instructeurs.

Om de nieuwe applicatie tot stand te brengen kan vertrokken worden van de oude structuur en omschrijvingen van de gewenste situatie omtrent de verschillende entiteiten. Ook de verschillende functionaliteiten dienen terug te komen in de nieuwe applicatie met hier en daar de nodige veranderingen of verbeteringen.

2.2 Gewenste situatie

Een belangrijk aspect binnen de nieuwe applicatie is dat deze modulair dient te zijn. Dit is voor de dienst RealDolmen Education heel belangrijk daar de applicatie moet meegroeien met de bedrijfsstructuur. Ook dient de API toegankelijk te zijn voor verschillende andere applicaties zoals het planningsbord waardoor het essentieel is om de entiteiten aan de basis van de applicatie goed uit te werken.

Het analyse-design-proces start vanuit de klant of RealDolmen Education zelf. Het is belangrijk om van in het begin van de ontwikkeling af te stemmen met diegenen die de applicatie zullen gebruiken en hier de entiteiten op af te stemmen. Verder kan vanuit deze entiteiten vertrokken worden om de domeinlaag te schrijven en vervolgens ook de API die de aanlevering en verwerking zal verzorgen naar de front-end toe. Dit proces wordt geïllustreerd in Figuur 1.



Figuur 1 Flow Bachelorproefopdracht (backend)

Het front-end gedeelte zal uitgewerkt worden door een andere stagiair van een andere school. Dit brengt ook een extra communicatieaspect met zich mee dat in rekening gehouden moet worden doorheen de hele ontwikkeling.

2.2.1 Technologisch luik

De applicatie wordt ontwikkeld met behulp van het .NET framework. Om de database tot stand te brengen wordt gebruik gemaakt van het Entity Framework Code First aangevuld met het Repository Pattern om te voorzien in de datacommunicatie. Om binnen de applicatie de dependencies te onderhouden wordt gebruik gemaakt van Dependency Injection aan de hand van Autofac.

Tijdens de ontwikkeling wordt gebruik gemaakt van Visual Studio Team Services om te voorzien in Continuous Integration en Continuous Delivery met hierin ook een Agile werkomgeving ingebouwd.

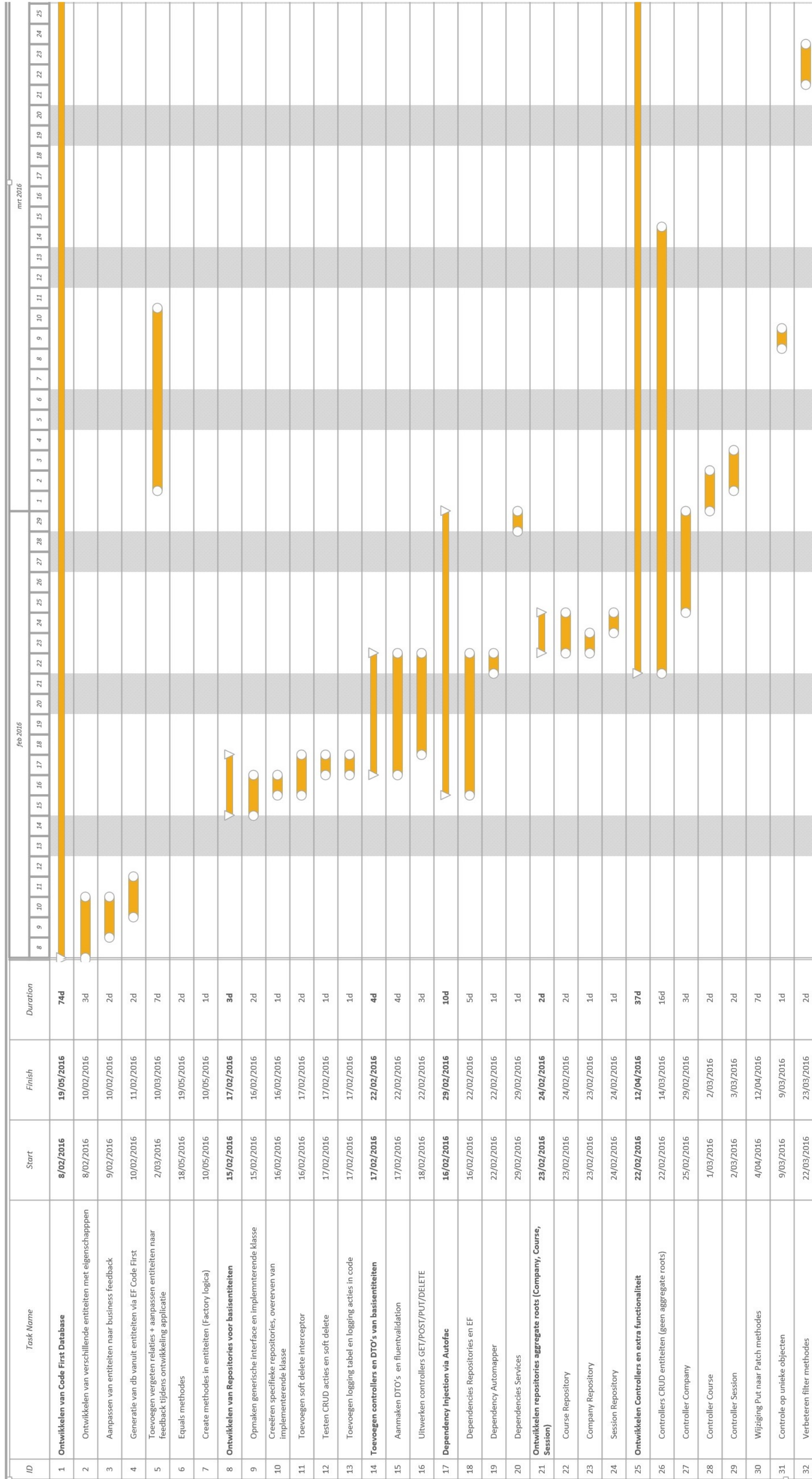
3 Actieplan

ACTIEPLAN				
Student(e): Nora Kamoen			Groep: 3ICT5	
Stageplaats: RealDolmen				
Stageleid(st)er (interne promotor): Joris Maervoet				
Stagementor (externe promotor): Tom Eeraerts				
Stap	**Inhoud**	**Streef-datum**	**Werkelijke datum**	**Opvolging**
1	Uitwerken 'Tropix' entiteiten	09/02/16	10/02/16	
2	Database via code first EF	11/02/16	11/02/16	
3	Onderzoek naar implementatie technologieën	Permanent	11/02/16	Repository Pattern, Dependency Injection, XUnit tests, soft delete, change tracking
4	Repository Pattern structuur	15/02/16	15/02/16	Structuur repositories met Aggregate roots
5	Toevoegen van XUnit tests	Permanent	03/03/16	Controllers, services, repos, mocking db
6	Generic Repository (en Unit of Work) voor basis entiteiten	18/02/16	17/02/16	Generic Repository met implementerende klassen per basis entiteit + Unit Of Work
7	Controllers en DTO's voor basis entiteiten	24/02/16	22/02/16	Controllers met CRUD acties en DTO's omzetten via Automapper
8	Interfaces hoofd repositories	03/03/16	25/02/16	CRUD Interfaces en klassen + CRUD child entities in entity klasse.
9	Controllers Aggregate roots	10/03/16	28/04/16	Later af door refactoring/verbetering
9	Autofac configureren	15/03/16	22/02/16	Dependency injection toegevoegd via modules
10	Toevoegen logica afhankelijk van vraag vanuit front-end	22/03/16	20/04/2016	Logica eisen kwamen pas later naar boven omwille van bugs en het voorzien van de basis
11	db logging en soft delete	29/03/16	24/02/16	Soft delete (db interceptor) + logging bij save
12	Bestaande data overzetten	07/04/16	19/05/2016	Migratie via console client (terwijl testing API)

Tabel 1 Actieplan

3.1 Gannt chart

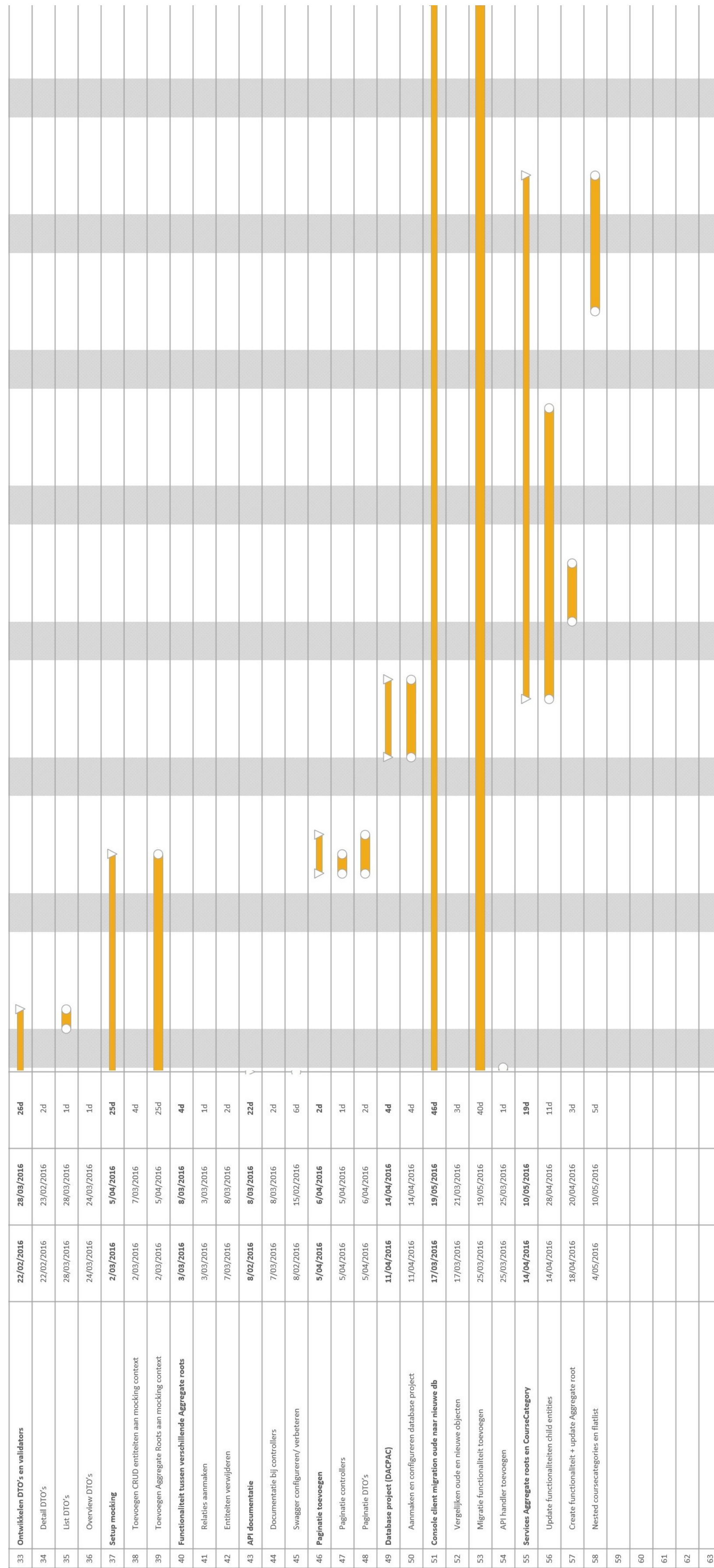
Onderstaande Gannt chart biedt een ruimer overzicht van het verloop van de verschillende taken in het actieplan.



ID	Task Name	Start	Finish	Duration
1	**Ontwikkelen van Code First Database**	**8/02/2016**	**19/05/2016**	**74d**
2	Ontwikkelen van verschillende entiteiten met eigenschapppen	8/02/2016	10/02/2016	3d
3	Aanpassen van entiteiten naar business feedback	9/02/2016	10/02/2016	2d
4	Generatie van db vanuit entiteiten via EF Code First	10/02/2016	11/02/2016	2d
5	Toevoegen vergeten relaties + aanpassen entiteiten naar feedback tijdens ontwikkeling applicatie	2/03/2016	10/03/2016	7d
6	Equals methodes	18/05/2016	19/05/2016	2d
7	Create methodes in entiteiten (Factory logica)	10/05/2016	10/05/2016	1d
8	**Ontwikkelen van Repositories voor basisentiteiten**	**15/02/2016**	**17/02/2016**	**3d**
9	Opmaken generische interface en implemnterende klasse	15/02/2016	16/02/2016	2d
10	Creeëren specifieke repositories, overerven van implementerende klasse	16/02/2016	16/02/2016	1d
11	Toevoegen soft delete interceptor	16/02/2016	17/02/2016	2d
12	Testen CRUD acties en soft delete	17/02/2016	17/02/2016	1d
13	Toevoegen logging tabel en logging acties in code	17/02/2016	17/02/2016	1d
14	**Toevoegen controllers en DTO's van basisentiteiten**	**17/02/2016**	**22/02/2016**	**4d**
15	Aanmaken DTO's en fluentvalidation	17/02/2016	22/02/2016	4d
16	Uitwerken controllers GET/POST/PUT/DELETE	18/02/2016	22/02/2016	3d
17	**Dependency Injection via Autofac**	**16/02/2016**	**29/02/2016**	**10d**
18	Dependencies Repositories en EF	16/02/2016	22/02/2016	5d
19	Dependency Automapper	22/02/2016	22/02/2016	1d
20	Dependencies Services	29/02/2016	29/02/2016	1d
21	**Ontwikkelen repositories aggregate roots (Company, Course, Session)**	**23/02/2016**	**24/02/2016**	**2d**
22	Course Repository	23/02/2016	24/02/2016	2d
23	Company Repository	23/02/2016	23/02/2016	1d
24	Session Repository	24/02/2016	24/02/2016	1d
25	**Ontwikkelen Controllers en extra functionaliteit**	**22/02/2016**	**12/04/2016**	**37d**
26	Controllers CRUD entiteiten (geen aggregate roots)	22/02/2016	14/03/2016	16d
27	Controller Company	25/02/2016	29/02/2016	3d
28	Controller Course	1/03/2016	2/03/2016	2d
29	Controller Session	2/03/2016	3/03/2016	2d
30	Wijziging Put naar Patch methodes	4/04/2016	12/04/2016	7d
31	Controle op unieke objecten	9/03/2016	9/03/2016	1d
32	Verbeteren filter methodes	22/03/2016	23/03/2016	2d

33	**Ontwikkelen DTO's en validators**	**22/02/2016**	**28/03/2016**	**26d**
34	Detail DTO's	22/02/2016	23/02/2016	2d
35	List DTO's	28/03/2016	28/03/2016	1d
36	Overview DTO's	24/03/2016	24/03/2016	1d
37	**Setup mocking**	**2/03/2016**	**5/04/2016**	**25d**
38	Toevoegen CRUD entiteiten aan mocking context	2/03/2016	7/03/2016	4d
39	Toevoegen Aggregate Roots aan mocking context	2/03/2016	5/04/2016	25d
40	**Functionaliteit tussen verschillende Aggregate roots**	**3/03/2016**	**8/03/2016**	**4d**
41	Relaties aanmaken	3/03/2016	3/03/2016	1d
42	Entiteiten verwijderen	7/03/2016	8/03/2016	2d
43	**API documentatie**	**8/02/2016**	**8/03/2016**	**22d**
44	Documentatie bij controllers	7/03/2016	8/03/2016	2d
45	Swagger configureren/ verbeteren	8/02/2016	15/02/2016	6d
46	**Paginatie toevoegen**	**5/04/2016**	**6/04/2016**	**2d**
47	Paginatie controllers	5/04/2016	5/04/2016	1d
48	Paginatie DTO's	5/04/2016	6/04/2016	2d
49	**Database project (DACPAC)**	**11/04/2016**	**14/04/2016**	**4d**
50	Aanmaken en configureren database project	11/04/2016	14/04/2016	4d
51	**Console client migration oude naar nieuwe db**	**17/03/2016**	**19/05/2016**	**46d**
52	Vergelijken oude en nieuwe objecten	17/03/2016	21/03/2016	3d
53	Migratie functionaliteit toevoegen	25/03/2016	19/05/2016	40d
54	API handler toevoegen	25/03/2016	25/03/2016	1d
55	**Services Aggregate roots en CourseCategory**	**14/04/2016**	**10/05/2016**	**19d**
56	Update functionaliteiten child entities	14/04/2016	28/04/2016	11d
57	Create functionaliteit + update Aggregate root	18/04/2016	20/04/2016	3d
58	Nested coursecategories en flatlist	4/05/2016	10/05/2016	5d
59				
60				
61				
62				
63				

Odisee HOGESCHOOL



33	**Ontwikkelen DTO's en validators**	**22/02/2016**	**28/03/2016**	**26d**
34	Detail DTO's	22/02/2016	23/02/2016	2d
35	List DTO's	28/03/2016	28/03/2016	1d
36	Overview DTO's	24/03/2016	24/03/2016	1d
37	**Setup mocking**	**2/03/2016**	**5/04/2016**	**25d**
38	Toevoegen CRUD entiteiten aan mocking context	2/03/2016	7/03/2016	4d
39	Toevoegen Aggregate Roots aan mocking context	2/03/2016	5/04/2016	25d
40	**Functionaliteit tussen verschillende Aggregate roots**	**3/03/2016**	**8/03/2016**	**4d**
41	Relaties aanmaken	3/03/2016	3/03/2016	1d
42	Entiteiten verwijderen	7/03/2016	8/03/2016	2d
43	**API documentatie**	**8/02/2016**	**8/03/2016**	**22d**
44	Documentatie bij controllers	7/03/2016	8/03/2016	2d
45	Swagger configureren/ verbeteren	8/02/2016	15/02/2016	6d
46	**Paginatie toevoegen**	**5/04/2016**	**6/04/2016**	**2d**
47	Paginatie controllers	5/04/2016	5/04/2016	1d
48	Paginatie DTO's	5/04/2016	6/04/2016	2d
49	**Database project (DACPAC)**	**11/04/2016**	**14/04/2016**	**4d**
50	Aanmaken en configureren database project	11/04/2016	14/04/2016	4d
51	**Console client migration oude naar nieuwe db**	**17/03/2016**	**19/05/2016**	**46d**
52	Vergelijken oude en nieuwe objecten	17/03/2016	21/03/2016	3d
53	Migratie functionaliteit toevoegen	25/03/2016	19/05/2016	40d
54	API handler toevoegen	25/03/2016	25/03/2016	1d
55	**Services Aggregate roots en CourseCategory**	**14/04/2016**	**10/05/2016**	**19d**
56	Update functionaliteiten child entities	14/04/2016	28/04/2016	11d
57	Create functionaliteit + update Aggregate root	18/04/2016	20/04/2016	3d
58	Nested coursecategories en flatlist	4/05/2016	10/05/2016	5d
59				
60				
61				
62				
63				

Figuur 2 Gannt chart

4 Voorstudie

Deze studie wordt opgebouwd aan de hand van de workflow voor het ontwikkelen van de applicatie. Hierbij wordt vertrokken vanuit een korte schets van de reeds bestaande applicatie en diens tekortkomingen gevolgd met een omschrijving van de basisprincipes van Domain Driven Design. Hierna worden de principes van entiteiten besproken en de omzetting hiervan naar een database gebruik makende van Entity Framework Code First. Deze technologieën zijn opgegeven door RealDolmen Education.

Binnen het databasegegeven worden enkele voorbeelden van change tracking en soft deletes besproken, de uiteindelijke keuze voor de gebruikte technologie gebeurt in samenspraak. Voor de dataverwerking wordt het Repository Pattern aangebracht door het bedrijf. Verder in de voorstudie wordt de Dependency Injection technologie Autofac besproken alsook het OWIN framework.

Wat betreft de echte implementatie en deployment, wordt het gebruik van Dacpac besproken en ook het gebruik van XUnit tests om de applicatie grondig te kunnen testen alvorens de applicatie finaal af te leveren.

4.1 Oude applicatie

De reeds bestaande applicatie is opgebouwd in Visual Basic (verder in de tekst zal ik hiernaar verwijzen als 'VB'). De applicatie bestaat slechts uit een solution met hierin wel verscheidene projecten maar alle interne communicatie gebeurt onderling. Er is gebruik gemaakt van een eenvoudige structuur waarin de verschillende usercontrols opgebouwd zijn uit .xaml bestanden met elk hun achterliggende code in VB die de afhandeling van userinteractie voor zich neemt. Deze manier van werken zorgt ervoor dat er geen concrete grens is tussen front-end en backend. Alle verwerking van de data gebeurt rechtstreeks achter de formulieren en wordt niet doorgestuurd naar een service, een API of een tussenliggende logicalaag. Dit vormt een eerste verschil bij de nieuwe applicatie daar de opdracht opgesplitst is in een duidelijke backend en front-end. Naar functionaliteit toe is vrijwel alles reeds aanwezig in de oude applicatie, de grootste veranderingen slaan op de data.

Binnen de oude applicatie is er in de database door de jaren heen heel wat veranderd en zijn tabellen bij gecreëerd omdat de applicatie het niet toeliet om de bestaande structuur zelf te wijzigen. Ook deze dubbelzinnigheden dienen dus weggewerkt te worden.

In het nieuwe proces zal er gewerkt worden volgens Entity Framework Code First maar dit aspect wordt later in de voorstudie uitgelegd (zie 4.5). Bij de structuur van de database of de entiteiten dienen een aantal zaken sterk gewijzigd te worden, dit vooral naar inhoudelijke tekortkomingen toe. Vanuit de business is er nood aan een andere structuur en ook door wijzigingen binnen RealDolmen als bedrijf zijn we genoodzaakt het hele concept opnieuw uit te denken. Deze verandering zal vooral zichtbaar zijn in de database maar ook in de toevoeging van bepaalde functionaliteiten die vasthangen aan nieuwe data en veranderingen in structuur van de applicatie en het gebruik van nieuwe technologieën van .NET.

4.2 Domain-Driven Design

Deze sectie beschrijft de basisprincipes van het ‘Domain-Driven Design’ ofte DDD. Wanneer men strikt volgens de regels van DDD wil werken bij het ontwikkelen van een applicatie vindt men alle volgende componenten terug met de correcte interactie, deze worden geïllustreerd in Figuur 3. Gezien de complexiteit van DDD beperkt de huidige studie zich tot het gebruiken van een ‘Ubiquitous Language’, ‘Entities’, ‘Aggregates’, ‘Repositories’ en ‘Services’ en dit alles onderhouden via ‘Continuous Integration’.

Figuur 3 Elementen DDD [8]

4.2.1 In het kort

Bij DDD wordt steeds vertrokken vanuit het domein en diens domeinlogica. Vanuit dit domein wordt dan de structuur van de volledige applicatie bepaald en hoe alle componenten met elkaar zullen interageren. Ook is het belangrijk dat bij de ontwikkeling iedereen in het team volgens dezelfde termen communiceert. Dit wordt gedefinieerd onder de term 'Ubiquitous language' en is ook een van de punten die tijdens de uitwerking van de opdracht gehandhaafd wordt.

DDD maakt gebruik van verscheidene 'Building Blocks'. Deze worden weergegeven in de bovenste cluster van bovenstaande figuur (Figuur 3). Elke 'Building Block' heeft zijn eigen karakteristieken waaronder [9]:

- Entity: een object dat gedefinieerd wordt door diens identiteit (unieke objecten) en niet door diens attributen.
- Value Object: een object met attributen maar zonder specifieke identiteit (niet uniek)
- Aggregate: een verzameling van objecten die afhankelijk zijn van een 'root entity'. Bij het gebruiken van een Aggregate wordt elke verandering doorgevoerd bij alle afhankelijke objecten en is deze structuur dus consistent.
- Domain event: een object dat een gebeurtenis omvat.
- Service: services worden gebruikt om bepaalde bewerkingen bij te houden die niet specifiek tot een object behoren.
- Repository: een verzameling van methodes voor het vergaren van objecten en bewerkingen hierop.
- Factory: deze bevat methodes voor het aanmaken van objecten.

4.3 Entiteiten

Entiteiten zijn het eerste deel binnen DDD die besproken zullen worden. Deze studie en analyse betreft de ontwikkeling van de verschillende entiteiten nodig om de applicatie op te bouwen. Deze dienen naadloos bij elkaar aan te sluiten en ook een goede opdeling te bevatten waardoor elke entiteit op zichzelf degelijk opgebouwd is. De techniek om te starten met het uitdenken van entiteiten op een manier dat deze volledig gericht zijn op de business kant is overgenomen uit het 'Domain Driven Design'.

Entiteiten omvatten alle mogelijke velden van een object, deze worden gebruikt voor het aanmaken van databasetabellen via Entity Framework Code First (zie 4.5), alsook een unieke identificatie. Wanneer er dan een veld binnen het object gewijzigd wordt blijft het object nog steeds hetzelfde dankzij dit unieke veld. Bijvoorbeeld wanneer men een entiteit 'Persoon' heeft met een naam en voornaam en tevens ook een identificatienummer dan kan de naam van de persoon veranderd worden maar blijft de persoon nog steeds dezelfde [10].

4.3.1 Aggregate roots

Bij het ontwikkelen van de entiteiten binnen een domeinmodel zijn er steeds tal van relaties tussen de verschillende entiteiten. Zo kan het ook zijn dat er een soort van 'hoofdobjecten' ontstaan die onderliggende objecten omvatten waarbij deze enkel kunnen bestaan via het hoofdobject. Een concreet voorbeeld hiervan is een huis met kamers. Een huis kan op zichzelf bestaan en bevat kamers, maar kamers kunnen niet bestaan zonder het huis. Deze logica kunnen we doortrekken bij het ontwikkelen van de entiteiten en ons model, de hoofdobjecten worden hierbij 'Aggregate roots' genoemd [11].

Deze objecten zullen telkens een reeks child entities omvatten en de toegang tot deze onderliggende entities zal in principe enkel via de root verlopen.

4.4 Database changes

Naast de gewone entiteiten en hun attributen die nodig zijn binnen de applicatie is het ook belangrijk om een zekere logging bij te houden van wat er gebeurt met alle records alsook een vorm van soft deletes zodat er ondanks het verwijderen van bepaalde entiteiten nog steeds een geschiedenis zichtbaar is van de records die 'verwijderd' zijn.

4.4.1 Change tracking

SQL Server heeft zelf een change tracking systeem wat elke Create, Update en Delete operatie zal bijhouden in een logging tabel. Hierin worden de Primary Key van het aangepaste record bijgehouden en enkele velden die weergeven wie dit record wanneer gewijzigd heeft en door welke operatie [12]. Change tracking kan echter ook zelf geïmplementeerd worden door het gebruik van een script met triggers. In dit script kan een logging tabel aangemaakt worden en kunnen de velden gekozen worden die bijgehouden moeten worden. De triggers kunnen Create, Update en Delete zijn. Wanneer Delete gebeurt aan de hand van een soft delete dan is dit geen Delete trigger maar een Update trigger, dit kan opgevangen worden door een extra Stored Procedure die hiervoor een record zal toevoegen in de logging tabel. Mogelijke velden voor de logging tabel zijn: ID, TableName, PrimaryKey, ChangedField, ChangedBy, AddedOn, ChangedOn, DeletedOn [13].

Het aanmaken van een logging tabel kan ook manueel gebeuren op dezelfde wijze als de toevoeging van andere entiteiten. Het change tracking systeem kan doorgevoerd worden door extra functionaliteit voorafgaand aan de SaveChanges() methode met daarin een systeem wat overloopt welke records toegevoegd of gewijzigd zijn. Op deze manier is het niet nodig om extra scripts te voorzien of Stored Procedures.

Door de logging apart te houden in een andere tabel en niet in de originele tabellen als extra velden kan alle analytische data gegroepeerd worden en vergemakkelijkt het proces om te bekijken welke tabellen of records het minst of meest gewijzigd worden, wie de data het meeste gebruikt en dergelijke meer. Wanneer dit in de originele entiteiten zelf gehouden wordt komen er veel meer queries aan te pas om hetzelfde resultaat te bereiken. De logging tabel kan zo ook gelinkt worden aan een web interface zodat analyses ook snel en gebruiksvriendelijk gemaakt kunnen worden [12].

Echter wanneer de verschillende acties enkel geregistreerd worden in een aparte tabel kan geen rekening gehouden worden met de deleteactie daar deze gebeurt aan de hand van een soft delete. Een ander nadeel is dat het specifieke record ID telkens 0 is bij een Create actie en hierbij niet geweten is om welk record het gaat. Om deze redenen is een betere optie de loggingvelden in elke entiteit apart toe te voegen. De functionaliteit om de loggingvelden te vullen verplaatst dan van de SaveChanges() methode naar elke aparte Delete, Update of Create methode.

4.4.2 Soft Deletes

Om geen data te 'verliezen' kan gebruik gemaakt worden van soft deletes. Zo kunnen records ook gemakkelijk terug gehaald worden zonder terug te keren naar een back-up. Soft deletes kunnen bijgehouden worden door een extra veld in elke tabel waar aan de hand van een bit bijgehouden wordt of het record verwijderd is.

Bij het ophalen van data dient dit bit steeds gecontroleerd te worden in de queries wat meer werk met zich meebrengt. Een andere manier kan zijn aan de hand van partitionering. Hierbij wordt de tabel opgesplitst wat het opvragen van de data makkelijker en zekerder maakt. Wanneer in een query het controleren van het 'verwijderd'-bit vergeten zou worden kan dit grote gevolgen hebben voor het programma [14].

Naast het partitioneren is een nog betere manier het gebruiken van een database interceptor.

4.4.2.1 Soft delete database interceptor

Het principe van een database interceptor is dat deze een CRUD-actie kan manipuleren en hierdoor een andere actie kan uitvoeren in plaats van de standaard actie. Dit kan heel handig zijn voor soft deletes. Wanneer een soft delete database interceptor gebruikt wordt dan zal het plaatsen van een flag aanzien worden als een echte delete. Als gevolg hiervan zullen get acties de zogezegd verwijderde records ook niet meer ophalen.

Een soft delete database interceptor zal een softdeletecolumn bepalen aan de hand van de benaming hiervan. Binnen deze kolom zal dan een bit geplaatst worden dat aanduidt of het record al dan niet verwijderd is. Telkens wanneer een delete actie uitgevoerd wordt naar de database zal deze actie onderschept worden en kan de interceptor de uiteindelijke actie uitvoeren.

De interceptor dient niet enkel geconfigureerd te worden maar deze dient ook bij alle identiteiten geregistreerd te worden alsook in de context van de database. Slechts dan zal de interceptor overal doorgevoerd worden [15].

Het grootste nadeel aan een soft delete interceptor is dat deze niet zal werken wanneer gebruik gemaakt wordt van Aggregate Roots. Wanneer een child entiteit verwijderd wordt dan zal enkel de link tussen de entiteiten 'null' worden maar zal het record niet als verwijderd geregistreerd worden. Om deze reden is het bij het gebruik van aggregate roots aangewezen om zelf manueel de soft delete te voorzien door het setten van een flag in een updatemethode in plaats van een delete actie uit te voeren.

4.5 Entity Framework Code First

Wanneer men alle entiteiten binnen het domeinmodel heeft ontwikkeld kunnen deze gebruikt worden om een effectieve database te creëren via het Entity Framework (of EF) Code First principe. Hierbij dient men van elke entiteit een klasse te maken waarin alle velden aangemaakt worden en ook alle relaties gedefinieerd worden. Wanneer men enkel gebruik maakt van entiteiten en niet van value objects kan een extra basisentiteit ontwikkeld worden die het identificatienummer zal beheren alsook de flag voor de soft delete.

Om van de entiteiten ook echte tabellen te maken heeft het Entity Framework een context klasse nodig waarin alle entiteiten gedefinieerd worden als DbSets. In deze contextklasse kunnen ook initiële instellingen vastgelegd worden of specifieke tabelnamen en samenstellingen voor bijvoorbeeld veel-op-veel relaties. Op deze manier kan alles naar persoonlijke wensen vastgelegd worden [16].

EF Code First maakt gebruik van 'Migrations' om de feitelijke database aan te maken of eventuele wijzigingen in de opbouw van de entiteiten door te voeren. Deze werkmethode laat ook toe om terug te keren naar vorige versies van de database zonder dat dit conflicten genereert zoals wanneer men een tabel rechtstreeks uit de database wil verwijderen. Entity Framework handelt alles in een logische volgorde af en regelt alles via de opgegeven connectiestring uit het configuratiebestand [17].

Wanneer er binnen de applicatie gebruik gemaakt dient te worden van de (al dan niet) reeds aangemaakte database heeft men enkel een instantie van de context nodig en kan alles uitgewerkt worden op dezelfde manier als wanneer men gebruik zou maken van een reeds bestaande database en hier connectie mee zou maken. Het enige verschil is dat de structuur bepaald wordt door de aangemaakte entiteiten in code in plaats van het rechtstreeks aanmaken in de SQL-managementstudio. Dit biedt als voordeel dat de database veel meer modulair is en zo ook de applicatie in zijn geheel [16].

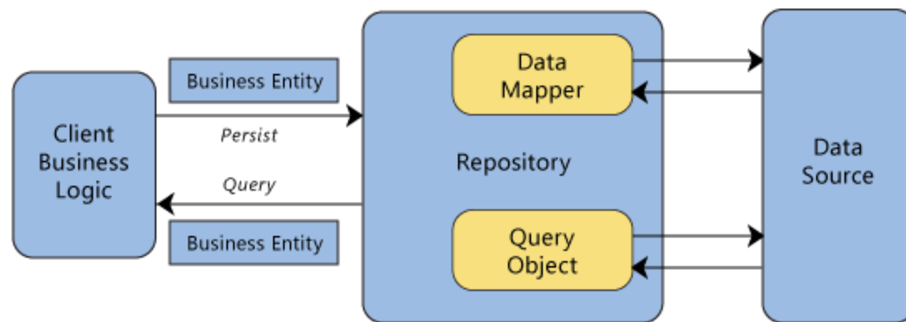
4.5.1 Eager loading vs. Lazy loading

Standaard wordt binnen EF Lazy loading voorzien. Lazy loading laat toe dat wanneer een entiteit ingeladen wordt de relaties nog niet actief meegenomen worden maar slechts een verwijzing zonder effectief object. De objecten worden dan pas ingeladen wanneer hier expliciet naar gevraagd wordt wat ervoor zorgt dat er geen onnodige data ingeladen wordt met een verbeterde performantie als gevolg. Echter is Lazy loading niet steeds de beste oplossing wanneer entiteiten echt afhangen van elkaar zoals bij Aggregate roots.

Binnen de ontwikkeling van de nieuwe applicatie wordt actief gekozen voor het gebruiken van Aggregate roots daar we de basislijnen van DDD willen volgen en dit ook een betere structuur voorziet binnen het domeinmodel. Dit zorgt ervoor dat Eager loading meer gewenst is. Bij het gebruik van Eager loading worden alle child entiteiten onmiddellijk ingeladen wanneer de root opgehaald wordt, dit veroorzaakt een groter resultaat maar een zekerheid dat het object volledig is met al diens relaties. Eager Loading kan men inschakelen door het gebruiken van Include clausules bij de queries [18].

4.6 Repository Pattern

Bij de ontwikkeling van een applicatie is het aangewezen deze op te delen in verschillende lagen onder de vorm van verschillende projecten of een grondige structuur aan de hand van mappen. Zo kan er een onderscheid gemaakt worden tussen alles betreffende data, logica en communicatie naar front-end toe. Een manier om dit onderscheid te maken is het gebruiken van repositories of het toepassen van het Repository Pattern. Repositories vormen hierbij de logicalaag direct na de data. Dit patroon wordt geïllustreerd in Figuur 4.



Figuur 4 Repository Pattern [19].

Na het ontwikkelen van de verschillende entiteiten, en dit te gebruiken voor het creëren van een database via EF Code First, volgt de logica. Alle bewerkingen op deze data en het doorgeven ervan naar de toplaag (communicatie met client zoals API) gebeurt in repositories. Hierin bevinden zich alle methodes voor Get, Delete, Update en Create acties. Een belangrijk gegeven is dat deze repositories enkel de methodes mogen omvatten die ook echt nodig zijn. Wanneer er van een entiteit enkel Get-acties nodig zijn dan kan men dus hiervoor een aparte Repository voorzien waarin enkel Getmethodes toegankelijk zijn.

Veel entiteiten hebben nood aan identieke methodes. Hiervoor kan een basis generische Repository voorzien worden waarin standaard de nodige methodes aanwezig zijn. Wanneer een entiteit meer nodig heeft dan enkel deze basismethodes kunnen extra specifieke methodes toegevoegd worden in diens specifieke Repository.

Enkele voordelen van het Repository Pattern zijn:

- Gemakkelijker testen van de functionaliteit daar deze volledig afgescheiden is van de data.
- Management verloopt centraal.
- Zorgt voor mapping tussen data en business entiteiten.
- Functionaliteit is gemakkelijk aan te passen daar dit geen invloed heeft op de structuur van de data entiteiten.
- Zorgt voor een zekere abstractie naar de client toe omwille van het mapping gegeven.

Bij het uitwerken van een applicatie volgens het Repository Pattern worden er van elke Repository een interface en een implementerende klasse voorzien. Ook dit draagt toe tot de modulariteit van het programma. Wanneer er extra functionaliteit gewenst is of wanneer een bepaalde implementatie gewijzigd dient te worden dan kan dit in een handomdraai door in de interface signaturen toe te voegen of in de implementerende klassen de functionaliteit te wijzigen. Alle implementerende klassen zullen de toevoegingen

overnemen. Wanneer de implementaties veranderen hoeft de rest van het programma dit niet te weten daar nog steeds de originele signaturen gebruikt worden. Dit zorgt voor een zekere abstractie naar de klassen toe die deze interface benutten [20].

4.6.1 Unit Of Work

Binnen het Repository Pattern kan ook een Unit of Work (of UOW) aangemaakt worden. Deze klasse is een soort verzamelaar waarin alle repositories geregistreerd worden. Deze klasse bevat ook een specifieke Save methode en een instantie van de context. De UOW kan doorheen de applicatie gebruikt worden om de nodige repositories aan te spreken. De sterkte hiervan is dat er slechts 1 instantie aangemaakt wordt van de context en hierdoor zeker geen verkeerde of dubbele connecties plaatsvinden maar ook dat de save methode ervoor zorgt dat alles correct doorgegeven wordt naar de feitelijke database.

Een Unit Of Work kan aanzien worden als een transactie met hierbinnen alle repositories. Pas wanneer de Save methode aangeroepen wordt zullen alle lopende wijzigingen doorgevoerd worden naar de database [19].

4.6.2 Data Mapper

Een extra functie van repositories is het gegeven dat de eigenlijke databasestructuur en opbouw van entiteiten onzichtbaar is voor de client. Het enige wat de client te zien krijgt zijn business entiteiten en het omvormen hiervan kan ook opgenomen worden in de repositories [21]. Het omvormen van de entiteiten kan aan de hand van een data mapper zoals ‘Automapper’.

Bij het gebruiken van Automapper is het nodig om van alle mogelijke business entiteiten objecten te voorzien met de nodige velden, deze objecten worden Data Transfer Objects of DTO’s genoemd. Deze DTO’s worden gebruikt om door te geven naar de client en zo enkel die informatie door te spelen die de client nodig heeft zonder de volledige innerlijke datastructuur bloot te geven. Automapper zal alle geregistreerde objecten automatisch mappen aan de hand van de naamgeving of specifieke combinaties die vastgelegd kunnen worden bij de configuratie [22].

Mappings kunnen in twee richtingen gebeuren zowel van data entiteit naar DTO als vice versa. Wanneer men een mapping doorvoert van een DTO naar een data entiteit moet men in rekening houden dat men hiermee niet het feitelijke object uit de database haalt maar enkel een mapping uitvoert. Het is dus nodig om het feitelijke object nog op te halen wanneer er bewerkingen moeten doorgevoerd worden in de database.

4.7 Dependency Injection

Wanneer applicaties een zekere omvang aannemen is het niet onwaarschijnlijk dat verschillende klassen gebruik maken van andere, dit zijn dependencies. Zonder deze dependencies kan de klasse niet alle functionaliteit uitvoeren die nodig is. Het aanmaken van dependencies kan gebeuren in de constructoren waar er objecten aangemaakt worden van de nodige klassen. Echter is dit een omslachtige manier van werken daar telkens al deze instanties manueel dienen te worden aangemaakt en meegegeven. Dit kan ook tot dubbele instanties leiden binnen de applicatie. Hierbij ligt de verantwoordelijkheid van het aanmaken van deze objecten ook telkens bij de klasse die een object aanmaakt en diens methodes oproept. Deze werkwijze kan volledig omgedraaid worden aan de hand van dependency injection. Dit zorgt voor een 'Inversion of Control' of IoC.

4.7.1 IoC verduidelijking

IoC kan op een gemakkelijke manier vergeleken worden met het dagelijkse leven, namelijk het plannen van een reis. Het typische verloop van het plannen van een reis gaat als volgt:

- De bestemming bepalen samen met de periode
- De nodige informatie over vluchten vergaren via een vliegtuigmaatschappij
- Een taxi maatschappij contacteren voor vervoer tussen thuis en de luchthaven
- Tickets ophalen voor de vlucht en in de taxi stappen.

Het nadeel aan deze manier van plannen is dat wanneer de vliegtuigmaatschappij verandert van telefonisch contact naar elektronisch (mail en site), de klant zich moet aanpassen aan deze nieuwe situatie en een nieuw proces moet aanleren.

Wanneer men echter volgens het IoC principe wil redeneren dan gaat het plannen van een reis als volgt:

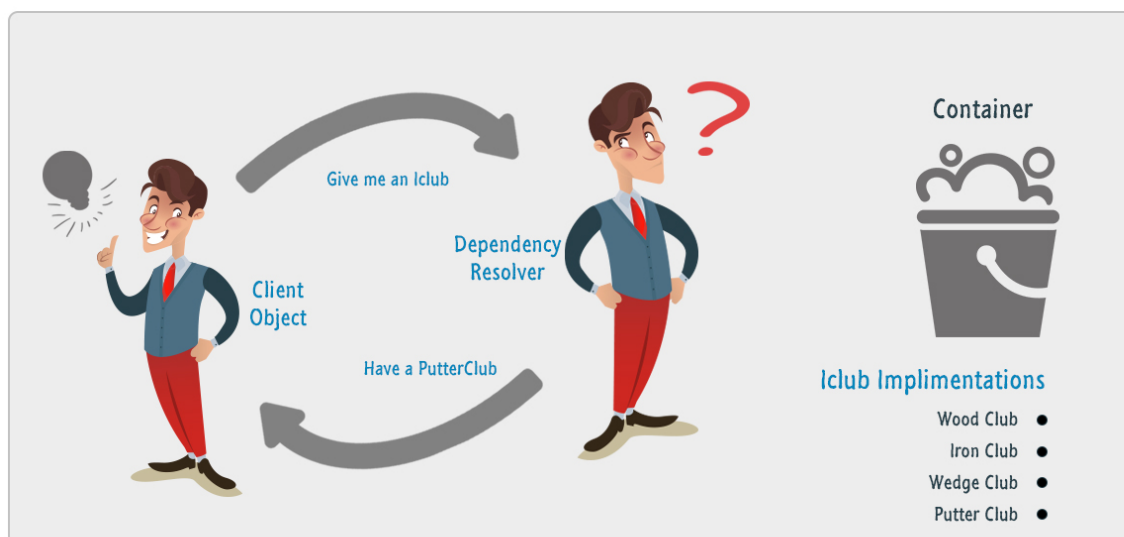
- Een administratief centrum contacteert de klant en zal voor hem de nodige info opzoeken over mogelijke vluchten bij een bepaald of verschillende vliegtuigmaatschappijen.
- Hetzelfde administratief centrum boekt een taxi als vervoer naar de luchthaven.
- De tickets worden opgestuurd en de taxi staat klaar voor de klant op het gewenste moment.

De voordelen bij IoC is dat het reisbureau verantwoordelijk is voor alle boekingen. Wanneer er iets verandert bij de taximaatschappij dan hoeft de klant zich hierover geen zorgen te maken. Ook wanneer het reisbureau diens communicatiekanaal veranderd heeft dit amper gevolgen voor de klant daar het administratief centrum zelf contact opneemt en niet andersom [23].

Bij het gebruiken van een IoC systeem zullen de dependencies ingevuld worden door een container of externe klasse. Hierbij is er een mogelijkheid om de gebruikte objecten aan te maken in de constructor maar een efficiëntere werkwijze is om hiervoor een framework te gebruiken. Een mogelijke bibliotheek dat deze werking voorziet is Autofac.

4.7.2 Autofac

Autofac is een mechanisme dat de dependency injection voor zich neemt en werkt aan de hand van een container. Om te starten met Autofac dient een containerbuilder aangemaakt te worden waarin alle dependencies geregistreerd worden. Deze registraties gebeuren voor alle interfaces die de verschillende klassen gebruiken in hun constructor. Aan de hand van deze registraties worden de nodige dependencies ingevoegd bij het creëren van de klasse. Andere uitwerkingen van dependency injection zijn via Property Injection of Method Injection. Binnen deze applicatie is het beperkt tot Constructor Injection daar zo goed als alle dependencies steeds door de hele klasse gebruikt worden en niet enkel in een methode.



Figuur 5 Voorbeeld Dependency resolver en container zoals Autofac [24].

Interfaces kunnen meegegeven worden aan de hand van :Interface in de klasse signatuur, alsook via parameters in de constructor van de klasse. Autofac zal dan aan de hand van diens container de nodige instanties meegeven bij de creatie van de klasse [25]. Hierdoor is het niet langer nodig dat elke klasse zijn dependencies zelf aanmaakt door middel van declaraties in de constructor zoals 'Interface int = new Interface();'. Al deze instanties worden nu meegegeven wat Inversion of Control tot gevolg heeft.

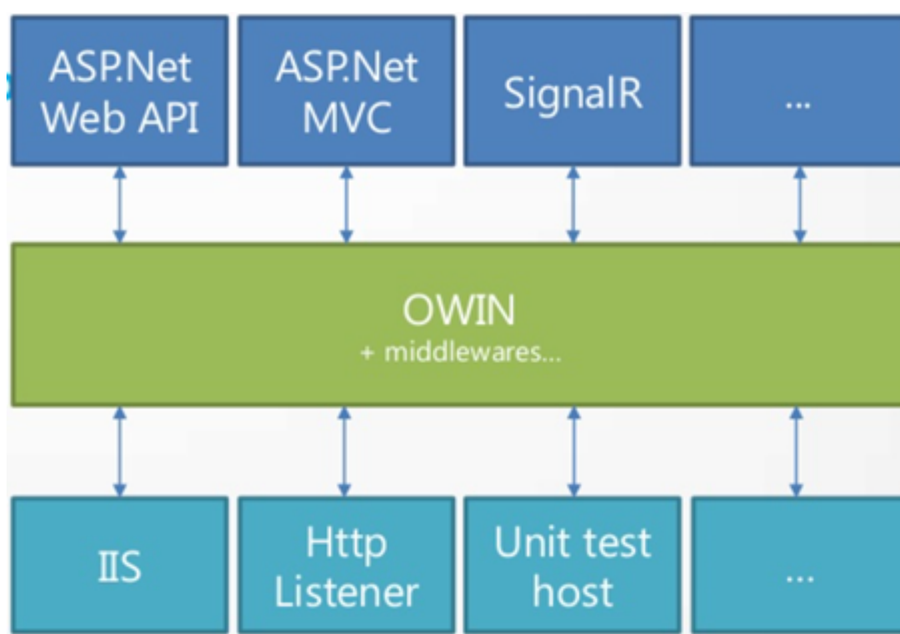
Men kan ervoor kiezen om alle dependencies manueel te registreren in de container, dit zijn in het Repository Pattern alle repositories, de configuratie van de mapper, configuratie van EF met diens context en entiteiten alsook eventuele services. Naarmate een applicatie groeit, kunnen deze registraties heel omvangrijk worden en bestaat de mogelijkheid dat er enkele vergeten worden of verkeerd geregistreerd worden. Om dit te voorkomen kan van modules gebruik gemaakt worden, deze modules zoeken binnen een namespace naar specifieke bestanden (afhankelijk van naamgeving) en registreren hiervan de nodige interfaces. Wanneer er repositories of services bijkomen, zullen deze automatisch toegevoegd worden aan de container zonder extra notaties. Modules worden op hun beurt opgenomen in de ContainerBuilder en deze container wordt opgeslagen op een globaal niveau binnen de applicatie daar deze van overal toegankelijk dient te zijn om de dependencies te resolven [26].

Een groot voordeel van deze registratie aan de hand van de interfaces is dat de implementaties van interfaces heel gemakkelijk aanpasbaar zijn. Een toevoeging binnen de interface wordt gemakkelijk doorgevoerd naar de implementerende klassen. Ook kunnen de implementaties gewijzigd worden naar een nieuwe situatie zonder dat andere klassen hier weet van moeten hebben, zij maken nog steeds gebruik van dezelfde aanroepingen en resultaten daar de interface signaturen niet veranderen. Dankzij het gebruik van interfaces zijn ook alle implementerende klassen direct toegankelijk via de methodes uit de interface [25].

4.8 Owin

Owin is een specificatie voor webapplicaties en diens implementatie is Katana. Owin dient als een middleware tussen bijvoorbeeld een .NET API en eender welke webserver. Bij standaard .NET APIs wordt gebruik gemaakt van IIS-server en worden alle instellingen hierop afgestemd. Wanneer men gebruik maakt van deze standaard configuratie zijn alle mogelijke instellingen aanwezig en worden alle mogelijke services ingeladen, dit is niet altijd nodig en soms is het gewenst een andere webserver te benutten in plaats van IIS-server, daarom kan het gebruik van Owin een betere optie zijn [27].

Bij het gebruiken van Owin wordt de webserver losgekoppeld van de applicatie en is het niet langer belangrijk welke server uiteindelijk gebruikt zal worden. Het vormt met name een middleware waarin alle configuraties van de applicatie in geregistreerd worden. Binnen deze registraties behoren onder meer Autofac, Automapper, WebAPI en alle andere nodige configuraties. Bij het runnen van de applicatie op een server zal de server zelf ophalen wat hij op dat moment nodig heeft via deze middleware [28]. De plaatsing van de Owin middleware wordt verduidelijkt in Figuur 6.



Figuur 6 Owin middleware [29]

Wanneer gebruik gemaakt wordt van Owin zal de API niet langer het Global.asax bestand benutten maar in plaats hiervan een eigen Startup klasse voorzien waarin alle initiële instellingen voor de applicatie gemaakt worden alsook alle registraties van deze instellingen binnen de IAppBuilder [30].

De IAppBuilder die gebruikt wordt voor het registreren van de instellingen maakt gebruik van de http-configuratie die ingesteld wordt via een Webapiconfig bestand. In dit bestand worden alle routes binnen de API aangemaakt en toegankelijk gemaakt binnen de applicatie. Doordat de IAppBuilder hiervan gebruik maakt kunnen al deze routes ook toegankelijk gemaakt worden in de middleware en kan de applicatie naadloos aansluiten bij de server.

4.9 Testing

4.9.1 XUnit

XUnit testing is een framework van unit testing voor .NET. Het grootste verschil met het klassieke Unit test framework is dat de tests geïnitialiseerd worden via de constructor en afgebroken via een implementatie van IDisposable. Andere Unit test frameworks maken gebruik van ‘setupfixtures’ en ‘teardown’ attributen, dit is niet langer nodig bij XUnit.

Naast deze andere manier van opstarten en afbreken van de testklassen bestaat binnen XUnit ook het verschil tussen Facts en Theories. Bij Facts wordt in de test slechts 1 mogelijkheid van data meegegeven aan de test en is er ook slechts 1 uitkomst mogelijk. Bij Theories echter kan er dynamisch data meegegeven worden en kan deze data ook testresultaten omvatten. Op deze manier kunnen meer verschillende casussen bekeken worden en hiermee ook meerdere lijnen code getest worden dan bij het schrijven van een Fact. Het gebruik van Theories leidt dus tot een beter zicht op de totale werking van het programma.

Het aangeven of een test een Fact of Theory is, gebeurt aan de hand van attributen. Deze kunnen ook gebruikt worden voor Skip en Time-out. Skip zal de test overslaan en Time-out zal de test doen falen wanneer de tijdslimiet overschreden is. Deze extra functionaliteit geeft de mogelijkheid om te zien welke stukken code eventueel te veel tijd in beslag nemen [31].

4.9.2 Owintesting

Naast klassieke XUnit tests kan ook gebruik gemaakt worden van Owintesting. Hierbij is er de mogelijkheid om een API uitvoerig te testen als een client. Het voordeel hierbij is dat ook de connectie met de API getest wordt en zo eventuele problemen opgemerkt kunnen worden.

Owintesting maakt gebruik van de Startup klasse van de API. Deze wordt gebruikt om de connectie te maken en een TestServer op te zetten. Verder kunnen Owintests afgehandeld worden gebruik makend van het XUnit framework of een ander testing framework [32].

4.9.3 Mocking via Moq

Mock objecten of mocking bestaat erin bestaande objecten te simuleren en hun gedrag over te nemen.

Bij het schrijven van tests voor een backend met database is het aangewezen om deze database te mocken daar er dan geen nutteloze data wordt weggeschreven naar de werkelijke database. Een mogelijk framework hiervoor is Moq.

Moq biedt een goede structuur om een mocking database op te bouwen. Deze database wordt dan volledig in-memory gecreëerd. Moq is het enige framework dat een volledige ondersteuning biedt voor Linq en Lambda expressies waardoor het perfect kan omgaan met de reeds aanwezige functionaliteit.

Het opzetten van de database gebeurt aan de hand van een setup klasse. Hierin worden eerst alle tabellen gevormd en dit aan de hand van JObjecten. Deze JObjecten worden gecreëerd aan de hand van Json bestanden waarin een basis geleverd wordt van objecten voor de database. De JObjecten worden gebruikt om tabellen te vormen en relaties te leggen tussen voorgaande. De gevormde tabellen worden op hun beurt toegevoegd aan een mockingcontext en deze context kan dan gebruikt worden in de XUnit tests [33].

4.10 Dacpac

Eens de applicatie vorm krijgt is het ook belangrijk om deze in zijn volledigheid te kunnen deployen, dit zijnde ook met de database. Om de database te deployen kan gebruik gemaakt worden van een Dacpacbestand waarin de volledige databasestructuur in vastgelegd wordt.

Een Dacpacbestand wordt opgebouwd uit SQL Server Objects. De developer creëert een Server Data Tool database project met hierin de lokale database, wanneer dit project built zal een Dacpac gebouwd worden. Om telkens dit project ook up to date te houden wordt gebruik gemaakt van de 'schema compare' tool. Hierin kan de source database geselecteerd worden en het project als doel. Wanneer de vergelijking uitgevoerd wordt zullen alle wijzigingen opgelijst worden en kan de developer zelf beslissen welke doorgevoerd moeten worden in het project. Het is aangewezen deze vergelijking steeds te maken na het doorvoeren van een migration in het EF Code First project [34].

Bij deployment kan het Dacpacbestand opgezocht worden en kan vanuit dit bestand een database gemaakt worden aan de hand van de XML-objecten die hierin gedefinieerd staan. Binnen TFS-server kan hiervoor een buildstep ingevoerd worden dat de Dacpac naar de gewenste locatie en server zal deployen. Het servertype van bestemming kan tevens ook in het database project vastgelegd worden zodat de juiste structuur van objecten vastgelegd wordt in het Dacpacbestand alsook de juiste configuraties [35].

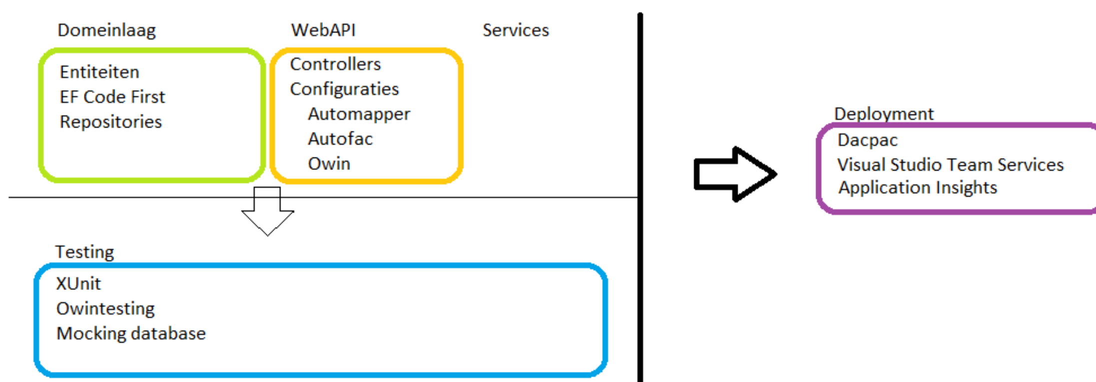
5 Praktische uitwerking

In dit deel wordt elke stap binnen de ontwikkeling van de backend overlopen. Voor elk van deze stappen worden de gebruikte technieken alsook eventuele evoluties en wijzigingen behandeld.

Allereerst wordt de domeinlaag besproken. Hierin komt de ontwikkeling van de verschillende entiteiten aan bod alsook de omzetting hiervan naar een database via EF Code First en bijkomende database configuraties zoals logging en soft deletes. Naast de entiteiten wordt ook de ontwikkeling van de verschillende DTO's behandeld. De uitwerking van de repositories komt hierin ook aan bod om te voorzien in alle datacommunicatie.

Vervolgens komt de uitwerking van de API aan bod met alle controllers en nodige configuraties omtrent mappings, dependency injection via Autofac en ten slotte ook Owin. Naast de controllers en overige configuraties worden ook de aangemaakte services behandeld. Hierna volgt alles omtrent testing en deployment aan de hand van Dacpac en Visual Studio Team Services.

Onderstaand schema geeft de verschillende onderdelen weer die aan bod zullen komen. De voorstelling ervan illustreert tevens de workflow.



Figuur 7 Weergave workflow en besproken onderwerpen

5.1 Domeinlaag

5.1.1 Entiteiten

De eerste stap in de ontwikkeling van de backend is het uitdenken van de entiteiten. Hierbij wordt vertrokken vanuit de bestaande database en een korte uitleg over welke entiteiten welke eigenschappen zullen moeten hebben.

Aangezien de structuur binnen RealDolmen Education aanzienlijk verandert voor het komende boekjaar heeft dit ook invloed op de entiteiten zoals de 'BusinessOwners'. In de oude applicatie is dit een aanzienlijke tabel terwijl deze nu slechts enkele records zal bevatten en ook anders opgebouwd zal worden. Ook het concept van 'Companies' als klanten is nieuw. Vroeger konden enkel de personen klanten zijn en niet elke persoon behoorde noodzakelijk tot een bedrijf. Dit laatste is nu echter wel het geval. Alle 'Persons' en 'Instructors' zullen ook uit dezelfde velden bestaan en hierdoor samengevoegd worden tot 1 tabel in tegenstelling tot de oude database opstelling.

Naast de wijzigingen in de structuur is er ook een grote opkuis van dubbele tabellen of het samenvoegen ervan. In de oude applicatie was er geen plaats voor een degelijke uitbreiding of het aanpassen van de datastructuur daar dit een hele hoop veranderingen vergt in de code en veel fouten tot gevolg heeft, hierdoor zijn er in de loop van de jaren extra kleine tabellen toegevoegd die nu genormaliseerd moeten worden.

Het hele proces van de ontwikkeling van de entiteiten ging door verscheidene stadia. Na elke aanpassing of evolutie werd deze gecommuniceerd naar de personen die het programma zullen gebruiken alsook het hoofd van RealDolmen Education. Door deze terugkoppeling zit iedereen in dezelfde stroom en zijn de uiteindelijke entiteiten de meest correcte vorm met de beste benadering van wat de businesskant wenst.

In bijlage bevinden zich 3 versies van de ontwikkeling die de evolutie weergeven van begin tot einde tijdens het ontwikkelingsproces voor de entiteiten (documenten *"B-1 Entiteiten v1.0", "B-2 Entiteiten v2.0"* en *"B-3 Entiteiten v3.2"*). Echter zijn er tijdens de ontwikkeling van de applicatie nog enkele entiteiten licht gewijzigd zoals 'Session' waar alles omtrent 'evaluation' geschrapt is en er in plaats van het veld 'Code' twee velden met 'Year' en 'SequenceNumber' toegevoegd zijn. Aan de hand van deze twee laatste velden wordt de code gemaakt om door te sturen naar de front-end. De finale versie van de verschillende entiteiten zijn ondergebracht in "*"B-4 Entiteiten v4.0"*.

Tijdens dit ontwikkelingsproces werden ook de Aggregate Roots bepaald zijnde 'Course', 'Company' en 'Session' met dan telkens hun child entities. Deze keuze werd gemaakt om een vaste structuur te behouden en duidelijk te maken dat voorgaande entiteiten de basis vormen van het programma. Door dit onderscheid te maken kan verder in de uitwerking van de code ook een vaste structuur aangehouden worden.

De 'Company' is een volledig nieuw concept en zorgt voor een goede flow in het beheren van de cursisten, instructeurs en de verscheidene locaties en factureringsadressen. Alle personen en adressen zullen telkens toebehoren aan een bedrijf waardoor er nergens losse flodders voorkomen en er een goed overzicht bestaat van wie waar toebehoort.

Bij de ‘Course’ entiteit zijn de child entities ‘Descriptions’ (met ‘Topics’), ‘Materials’ en ‘Remarks’. Alle objecten van deze soort kunnen niet bestaan zonder een ‘Course’ en zouden dan ook niet vrij aangemaakt mogen worden. Hiertussen bevinden zich dus harde links. De entiteit ‘Remarks’ komt ook terug binnen de ‘Session’ en de ‘Company’, een Remark zal dus telkens tot een van de drie Aggregate Roots behoren maar nooit tot meer dan één hiervan.

Bij de ‘Session’ als Aggregate root behoren: ‘SessionDay’, ‘Attachment’ en ‘Audience’. Hierbij geldt dezelfde denkwijze als bij de child entities van Company en Course. Alle overige relaties tussen verschillende objecten zijn niet afhankelijk van elkaar. Hiertoe behoren de verschillende categorieën, types, talen, statussen en dergelijke.

Omdat elke entiteit standaard een ‘ID’, ‘RowVersion’ en een ‘IsActive’ veld bevat zijn deze apart geplaatst in een ‘BaseEntity’ waar vervolgens alle entiteiten van overerven. Naast deze basisentiteit was initieel ook een ‘Logging’ entiteit toegevoegd. Hierin bevonden zich de velden ‘ID’, ‘Table’, ‘Record_ID’, ‘CreatedOn’ en ‘UpdatedOn’. In deze entiteit zullen alle wijzigingen binnen de database bijgehouden worden. Meer hierover wordt uitgelegd onder 5.3 Soft Delete en Logging. Echter is dit tegen het einde van de opdracht gewijzigd en worden de logging velden in elke entiteit bijgehouden. Daar deze over alle entiteiten nodig zijn, zijn deze velden ondergebracht in de BaseEntity.

5.1.1.1 Functionaliteit

Binnen alle Aggregate Roots en andere entiteiten die geen child entity zijn, zijn een aantal methodes aanwezig binnen de klasse van de entiteit. Al deze klassen bevatten een statische Create methode waarin een object van de entiteit wordt aangemaakt gebruik makende van de nodige meegegeven parameters. Dit is volgens het principe van Factories, echter zijn er geen aparte klassen aangemaakt naast de entiteiten om deze Create methodes in onder te brengen.

De Aggregate Roots hebben naast deze Create methode ook methodes voor het toevoegen, updaten en verwijderen van child entities binnen het object. Deze methodes worden genuttigd vanuit de services van de Aggregate Roots. De services worden verder in de praktische uitwerking verwerkt onder 5.6 Services.

5.1.2 EF Code First

Na het uitdenken van de verschillende entiteiten en deze op punt te stellen dienen ze omgezet te worden naar een werkelijke database en dit gebeurt aan de hand van het Entity Framework Code First. Per entiteit wordt een klasse voorzien met daarin de nodige eigenschappen en relaties. De relaties worden 'virtual' gemaakt omdat dit in het geval van Lazy Loading toelaat om de relaties onmiddellijk in te laden en de data op te halen wanneer nodig. Eager Loading aan de tegenhand zorgt ervoor dat alle vernoemde relaties ook onmiddellijk meegenomen worden en deze objecten dan ook onmiddellijk ter beschikking staan.

Elke entiteit bestaat standaard uit een 'ID', een 'RowVersion' en een 'IsActive' veld dat garant staat voor een soft delete. Daar deze velden door alle entiteiten genuttigd worden kunnen deze apart geplaatst worden in een 'BaseEntity' klasse waarna alle andere entiteiten hiervan kunnen overerven. Door deze velden af te splitsen wordt dubbel werk vermeden en zijn deze velden aanwezig in elke klasse die hiervan overerft. Wanneer een extra algemeen veld toegevoegd moet worden kan dit gemakkelijk in de BaseEntity bijgevoegd worden. Het veld 'RowVersion' staat garant voor concurrency controle. De configuratie hiervoor wordt vastgelegd binnen de datacontext. In Figuur 8 wordt de BaseEntity weergegeven. Later in de ontwikkeling werd er ook voor gekozen om de loggingvelden hierin onder te brengen. Deze werden dan naadloos overgenomen door alle andere entiteiten en zo ook toegevoegd aan de uiteindelijke database zonder extra werk te verrichten.

```
public abstract class BaseEntity {

    0 references | - changes | -authors, -changes
    public BaseEntity() {
        IsActive = true;
    }

    218 references | - changes | -authors, -changes
    public int ID { get; set; }
    109 references | 0/3 passing | - changes | -authors, -changes
    public bool IsActive { get; set; }

    22 references | - changes | -authors, -changes
    public DateTime? CreatedOn { get; set; }
    17 references | - changes | -authors, -changes
    public DateTime? UpdatedOn { get; set; }
    23 references | - changes | -authors, -changes
    public DateTime? DeletedOn { get; set; }

    [Required()]
    62 references | - changes | -authors, -changes
    public string ChangedBy { get; set; }

    [Timestamp]
    46 references | - changes | -authors, -changes
    public byte[] RowVersion { get; set; }
}
```

Figuur 8 BaseEntity met algemene velden

Naast alle verschillende entiteiten dient er tevens een context aangemaakt te worden. Binnen deze contextklasse wordt de naam van de databaseconnectie meegegeven in de constructor waardoor tijdens het lopen van de applicatie telkens naar de juiste connectie verwezen wordt in het app.config bestand. In de constructor kunnen ook enkele configuraties vastgelegd worden waaronder het uitschakelen van Lazy Loading. Deze uitschakeling zorgt ervoor dat niet op elk moment alle verwijzingen naar relaties mee ingeladen worden maar enkel de pure objecten zelf. Wanneer deze relaties wel nodig zijn bijvoorbeeld bij Aggregate roots, kunnen deze expliciet ingeladen worden aan de hand van Eager Loading. Voorbeelden hiervan volgen onder 5.2 Repositories. In de contextklasse wordt ook de methode OnModelCreating() overschreven. Hierin wordt een veel op veel relatie vastgelegd die een andere benaming heeft dan de entiteiten zelf. Standaard worden deze veel-op-veel tabellen automatisch aangemaakt als de relaties de namen dragen van de entiteiten. Een enkele veel-op-veel is hier een uitzondering van en dat is die tussen een Course en Instructors, Instructors zijn ‘Person’ objecten dus herkent hij deze relatie niet als een veel-op-veel aangezien gebruik gemaakt wordt van de ‘Instructors’ benaming. In Figuur 9 staat de configuratie voor deze veel-op-veel relatie.

```
modelBuilder.Entity<Person>()
    .HasMany<Course>(s => s.Courses)
    .WithMany(c => c.Instructors)
    .Map(cs =>
    {
        cs.MapLeftKey("InstructorId");
        cs.MapRightKey("CourseId");
        cs.ToTable("CoursesInstructors");
    });
```

Figuur 9 Configuratie veel-op-veel relatie

Wanneer alle entiteiten uitgewerkt zijn in verschillende klassen en de nodige relaties ook aangemaakt zijn dienen al deze klassen toegevoegd te worden aan de context. Door de entiteiten hierin toe te voegen weet EF Code First van welke klassen tabellen gemaakt moeten worden. Op deze manier kan de ‘BaseEntity’ uitgesloten worden uit de feitelijke database maar zullen deze velden wel telkens via overerving in alle tabellen terugkomen. De registratie van enkele entiteiten wordt weergegeven in Figuur 10.

```
0 references | Nora Kamoen, 63 days ago | 1 author, 4 changes | 4 work items
public virtual DbSet<Address> Addresses { get; set; }
0 references | Nora Kamoen, 68 days ago | 1 author, 2 changes | 3 work items
public virtual DbSet<Attachment> Attachments { get; set; }
0 references | Nora Kamoen, 68 days ago | 1 author, 2 changes | 3 work items
public virtual DbSet<Audience> Audiences { get; set; }
0 references | Nora Kamoen, 68 days ago | 1 author, 2 changes | 3 work items
public virtual DbSet<BusinessOwner> BusinessOwners { get; set; }
0 references | Nora Kamoen, 68 days ago | 1 author, 2 changes | 3 work items
public virtual DbSet<Company> Companies { get; set; }
```

Figuur 10 Registratie entiteiten in context

Een andere instelling dat eerder al vermeld werd is deze voor de concurrency. Deze werd ingevoegd om database concurrency tegen te gaan en ervoor te zorgen dat twee personen geen data kunnen aanpassen die sinds de laatste request reeds gewijzigd is. De configuratie hiervoor bestaat erin in de ModelBuilder methode het RowVersion veld te

registreren als ConcurrencyToken (Figuur 11). Bij elke SaveChanges wordt deze RowVersion dan vergeleken en wanneer deze gewijzigd is, zal dit een error veroorzaken. Deze error kan dan afgehandeld worden in de API.

```
modelBuilder.Entity<Address>()
    .Property(p => p.RowVersion).IsConcurrencyToken();
modelBuilder.Entity<Attachment>()
    .Property(p => p.RowVersion).IsConcurrencyToken();
modelBuilder.Entity<Audience>()
    .Property(p => p.RowVersion).IsConcurrencyToken();
modelBuilder.Entity<BusinessOwner>()
    .Property(p => p.RowVersion).IsConcurrencyToken();
modelBuilder.Entity<Company>()
    .Property(p => p.RowVersion).IsConcurrencyToken();
modelBuilder.Entity<CourseCategory>()
    .Property(p => p.RowVersion).IsConcurrencyToken();
modelBuilder.Entity<Course>()
    .Property(p => p.RowVersion).IsConcurrencyToken();
```

Figuur 11 ConcurrencyToken registratie

Nadat alle entiteiten geregistreerd zijn en ook alle configuraties aangemaakt zijn, kunnen deze klassen omgevormd worden tot een feitelijke database aan de hand van migrations. Om dit mogelijk te maken dient een eerste commando uitgevoerd te worden om migraties mogelijk te maken. Daarna kan bij elke wijziging in de entiteiten een nieuwe migration uitgevoerd worden waarna die kan doorgevoerd worden naar de feitelijke database. De nodige commando's hiervoor zijn:

```
PM> Enable-Migrations -StartUpProjectName Realdolmen.Education.Tropix.Api -ProjectName Realdolmen.Education.Tropix.data
```

```
PM> Add-Migration -StartUpProjectName Realdolmen.Education.Tropix.Api -ProjectName Realdolmen.Education.Tropix.Data -Name RowVersion
```

```
PM> Update-Database -StartUpProjectName Realdolmen.Education.Tropix.Api -ProjectName Realdolmen.Education.Tropix.data
```

Figuur 12 Commando's migrations EF

Binnen de applicatie kan dan gebruik gemaakt worden van de context en de verschillende entiteitklassen om acties uit te voeren van of naar de database. Deze acties en verwerkingen gebeuren in de Repositories.

5.2 Repositories

Nu de domeinlaag werd opgebouwd, kan de toegang tot de databank worden uitgewerkt. Om dit volledig af te scheiden in aparte klassen binnen de domeinlaag is gekozen voor Repositories. Binnen deze Repositories worden voor alle entiteiten query's verzameld voor het ophalen, bewerken en versturen van data naar de database.

Omdat de lijst van entiteiten vrij lang is en velen hiervan identiek dezelfde acties nodig hebben, zijnde basis CRUD-acties, kunnen deze gegroepeerd worden in een generieke Repository. Binnen deze Repository worden generieke methodes voorzien voor Get, Add, Update en Delete acties. Om deze toepasbaar te maken op alle mogelijke entiteiten wordt voor het generische type <TEntity> de entiteit 'BaseEntity' gekozen daar alle entiteiten hiervan overerven. Hierdoor kan tevens telkens de ID genuttigd worden zonder extra castings uit te voeren bijvoorbeeld bij de methode GetById(int id), alsook het veld 'IsActive' is hierdoor toegankelijk en kan genuttigd worden binnen de methode Delete(). Naast deze twee velden zijn ook de loggingvelden toegankelijk waardoor deze bij elke actie mee ingevuld kunnen worden.

De generieke Repository vertrekt vanuit een interface met hierin alle signaturen voor de basismethodes. Deze signaturen worden weergegeven in Figuur 13. Naast deze interface bestaat een implementerende klasse die al deze methodes uitwerkt met de nodige controles.

```
46 references | Nora Kamoen, 7 days ago | 1 author, 2 changes | 2 work items
public interface IGenericBaseRepository<TEntity> where TEntity : BaseEntity {

    1 reference | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    Expression<Func<TItem, bool>> PropertyEquals<TItem, TValue>(string property, TValue value);

    51 references | 0/15 passing | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    IQueryable<TEntity> GetTable();

    86 references | 0/2 passing | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    Task<TEntity> GetById(int id);

    26 references | 0/1 passing | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    void Add(TEntity entity);

    14 references | 0/4 passing | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    void Update(TEntity entity);

    13 references | 0/5 passing | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    void Delete(TEntity entity);

    16 references | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    Task Delete(int id);

    53 references | 0/13 passing | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    Task SaveAsync();
}
```

Figuur 13 Interface generische Repository

Alle specifieke Repositories van de verschillende entiteiten zullen op zich ook overerven van de implementerende klasse van de generieke Repository. Wanneer een bepaalde entiteit extra methodes nodig heeft kunnen deze toegevoegd worden in de specifieke Repository. Alsook kan een verschillende implementatie van een methode doorgevoerd

worden door middel van een override, dit is vooral gewenst bij de Aggregate Roots. Om deze overschrijving toe te laten worden alle methodes in de generieke Repository virtual gemaakt.

De Aggregate Roots bezitten naast de verwijzing naar de generieke Repository ook een eigen interface. Binnen deze interface worden alle extra methodes gedefinieerd die de implementerende klasse dan kan uitwerken. Daar de Aggregate Roots nood hebben aan het inladen van hun child entities wordt er gebruik gemaakt van Eager Loading. Om dit mogelijk te maken dienen de Get methodes uit de generieke Repository overschreven te worden met een specifieke methode waarin de nodige Include() clausules opgenomen zijn. Een voorbeeld hiervan is binnen de Company een Include voor Employees, Addresses en Remarks, weergegeven in Figuur 14.

```
/// <summary> Gets the table of the Company and all its child entities.
26 references | 4/4 passing | Nora Kamoen, 7 days ago | 1 author, 4 changes | 6 work items
public override IQueryable<Company> GetTable() {
    var company = context.Set<Company>()
        .Include(c => c.Addresses)
        .Include(c => c.Remarks)
        .Include(c => c.Employees);
    return company;
}
```

Figuur 14 Override GetTable methode Aggregate Root Company

Naast deze override hebben de repositories van Aggregate Roots ook extra methodes voor het ophalen van enkel hun child entities of 1 enkele child entity. Deze methodes worden dan eerst toegevoegd in de interface waarna de implementerende klasse de functionaliteit hiervoor zal bezitten. Een voorbeeld van deze extra methodes in de interface van de 'Company' entiteit wordt getoond in Figuur 15.

```
10 references | Nora Kamoen, 7 days ago | 1 author, 17 changes | 13 work items
public interface ICompanyRepository : IGenericBaseRepository<Company> {
    //add necessary method signatures
    2 references | Nora Kamoen, 8 days ago | 1 author, 2 changes | 3 work items
    IQueryable<Person> GetInstructors();
    5 references | Nora Kamoen, 8 days ago | 1 author, 3 changes | 4 work items
    IQueryable<Address> GetAddresses(int companyId);
    3 references | Nora Kamoen, 8 days ago | 1 author, 2 changes | 2 work items
    IQueryable<Remark> GetRemarks(int companyId);
    4 references | Nora Kamoen, 23 days ago | 1 author, 2 changes | 3 work items
    IQueryable<Person> GetEmployees(int companyId);
    5 references | Nora Kamoen, 21 days ago | 1 author, 1 change | 1 work item
    Task<Address> GetAddress(int id);
    8 references | Nora Kamoen, 20 days ago | 1 author, 1 change | 1 work item
    Task<Person> GetPerson(int id);
}
```

Figuur 15 Interface Company Aggregate Root: extra methodes

Alle mogelijke bewerkingen van en naar de database zullen via deze Repositories verlopen. Op deze manier wordt alle datacommunicatie volledig afgescheiden van de business laag of API maar blijven al deze bewerkingen strikt in de domeinlaag waar ook de entiteiten en datacontext zich bevinden.

5.3 Soft Delete en Logging

Het bedrijf wil alle data behouden die aangemaakt is binnen de applicatie omwille van data integriteit waardoor geen gebruik gemaakt kan worden van de echte delete functionaliteit. Alle records moeten bestaand blijven in de database maar er moet wel een indicatie zijn of deze records al dan niet actief zijn binnen de huidige situatie. Om dit tot stand te brengen wordt er gebruik gemaakt van soft delete.

5.3.1 Evolutie soft delete

Deze soft delete gebeurde initieel aan de hand van een database interceptor. Hierbij werd elk delete commando opgevangen en werd een veld binnen het object op ‘false’ geplaatst wat duidelijk maakte dat dit object verwijderd was. Om dit veld te bepalen werd gebruik gemaakt van een data attribuut. Binnen de configuratieklassen van de database interceptor werd enerzijds dit attribuut opgezocht binnen de entiteiten om de juiste kolom te bepalen en anderzijds werd een queryvisitor aangemaakt om de deleteactie te onderscheppen en hierbij enkel de bepaalde kolom te wijzigen. Wanneer dan in de applicatie records worden opgevraagd, zal de queryvisitor ook genuttigd worden waardoor het resultaat enkel deze records zal bevatten waarbij de kolom op ‘true’ staat.

Echter deze soft delete database interceptor is niet efficiënt wanneer men gebruik maakt van Aggregate Roots. Wanneer men een Aggregate Root wenst te verwijderen wordt deze verwijderd gebruik makend van het soft delete veld maar worden alle relaties met child entities op NULL geplaatst waardoor men alle linken kwijt is. Hierdoor werd afgezien van het gebruik van een soft delete database interceptor. In plaats hiervan wordt het ‘IsActive’ veld manueel op ‘false’ gezet bij een delete actie. Op deze manier is het ook mogelijk om verwijderde records nog steeds op te halen en door te sturen naar de front-end wanneer nodig. Een delete actie verandert hierdoor naar een update van het object en niet in een echte verwijder actie.

5.3.2 Evolutie Logging

RealDolmen Education wil ook een overzicht bewaren van wanneer welke records aangemaakt of gewijzigd zijn en door wie. Om dit tot stand te brengen kan men gebruik maken van een change tracking systeem dat reeds aanwezig is binnen SQL of kan men zelf een loggingtabel of logging velden toevoegen in de database. Binnen deze applicatie werd voor de tweede optie gekozen daar het logging gegeven persoonlijk aangepast kan worden en alle gegevens ook in de bestaande database aanwezig blijven en niet elders opgeslagen worden.

De eerste uitwerking hiervan maakte gebruik van een aparte loggingtabel. Deze tabel bestaat uit volgende velden:

- ID
- Table
- Record
- CreatedOn
- UpdatedOn
- ChangedBy

Om de logging tot stand te brengen werd een extra functionaliteit ingevoegd alvorens de SaveChangesAsync() methode aan te roepen. Hierin wordt gekeken welke records gewijzigd zijn sinds de laatste situatie aan de hand van de ChangeTracker. Alle records die hierin aanwezig zijn worden overlopen en bekeken of deze zich in de 'Added' of 'Modified' state bevinden. Enkel deze twee staten zijn nodig aangezien er nooit records verwijderd worden. Respectievelijk wordt er dan in de loggingtabel een record toegevoegd waarbij het veld 'CreatedOn' of 'UpdatedOn' ingevuld wordt met het huidige tijdstip. Verder wordt ook de tabelnaam van het record ingevuld alsook het ID (dit is 0 bij Added) en de gebruikersnaam van de persoon die dit record aangepast heeft.

```
public async Task SaveAsync() {
    var changeSet = this.context.ChangeTracker.Entries();

    if (changeSet != null) {
        foreach (var entry in changeSet.Where(c => c.Entity is BaseEntity && c.State == EntityState.Added)) {
            Logging log = new Logging();
            log.CreatedOn = DateTime.Now;
            log.ChangedBy = userService.GetUserId();
            log.RecordID = Convert.ToInt32(((BaseEntity)entry.Entity).ID);
            log.Table = this.context.GetTableName(entry);
            context.Logs.Add(log);
        }
        foreach (var entry in changeSet.Where(c => c.Entity is BaseEntity && c.State == EntityState.Modified)) {
            Logging log = new Logging();
            log.UpdatedOn = DateTime.Now;
            log.ChangedBy = userService.GetUserId();
            log.RecordID = Convert.ToInt32(((BaseEntity)entry.Entity).ID);
            log.Table = this.context.GetTableName(entry);
            context.Logs.Add(log);
        }
    }

    try {
        await this.context.SaveChangesAsync();
    } catch (DbUpdateException dbu) {
        var exception = HandleDbUpdateException(dbu);
        throw exception;
    } catch (Exception e) {
        Debug.WriteLine(e.Message);
        throw;
    }
}
```

Figuur 16 Functionaliteit logging voor SaveChangesAsync()

Echter wanneer men deze uitwerking gebruikt weet men nooit precies welk record wanneer toegevoegd is aangezien hier telkens 0 als ID ingevuld wordt. Om deze reden werd gekozen om de aparte tabel te vervangen door extra velden in elke aparte entiteit. Daar deze velden in elke entiteit terugkomen kunnen deze toegevoegd worden via de BaseEntity.

Om de logging uiteindelijk tot stand te brengen wordt in de verschillende Add, Update en Delete acties extra code toegevoegd die het veld 'CreatedOn', 'UpdatedOn' of 'DeletedOn' samen met 'ChangedBy' zullen initialiseren. Nu het ook mogelijk is om delete acties apart te loggen, wordt een extra veld 'DeletedOn' voorzien. Door gebruik te maken van extra velden in elke tabel is er een volledig overzicht welke acties voor het laatst uitgevoerd zijn op elk specifiek record.

5.4 Data Transfer Objects

Data Transfer Objects of DTO's zijn de uiteindelijke objecten die via de API doorgestuurd of ontvangen zullen worden naar of van de client applicatie(s). Deze worden ontwikkeld omdat het niet steeds nodig is om de volledige entiteiten door te sturen en op deze manier ontstaat er ook een soort abstractie tussen business en database. Deze DTO's bevinden zich op hun beurt in een ander project binnen de solution daar deze niet volledig tot de datalaag behoren maar ook niet tot de API, ze vormen een soort van schakel.

In eerste instantie werd voor de DTO's een kopie voorzien van elke entiteit zonder enkele relaties die overbodig zijn voor de front-end. Voorbeeld van dergelijke relatie is de relatie binnen 'Remark' naar 'Company', 'Course' en 'Session' daar deze relatie reeds aanwezig is in de Aggregate Root zelf. Naargelang de ontwikkeling van de applicatie vordert en ook de front-end duidelijker wordt zijn tal van DTO's bijgevoegd en gewijzigd. Hierin wordt dan een onderscheid gemaakt tussen

- Volledige DTO's met alle velden aanwezig
- Overzicht DTO's die de algemene velden weergeven en voornamelijk gebruikt worden voor Post acties
- Lijst DTO's die enkel naam en eventueel code omvatten, hetgeen ideaal is voor overview pagina's waar bijvoorbeeld alle courses in een lijst worden weergegeven.
- Functionele DTO's zoals paginering [36] (Figuur 17), concurrency, geneste lijst.

```
public class PaginationDTO<T> where T : BaseDTO {
    3 references | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    public int TotalCount { get; set; }
    3 references | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    public double TotalPages { get; set; }
    10 references | 4/4 passing | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    public List<T> ListEntities { get; set; }
}
```

Figuur 17 PaginationDTO gebruikt voor Getmethodes

Elke DTO heeft een eigen validatie klasse waarin gedefinieerd wordt welke velden verplicht zijn en waaraan deze moeten voldoen. Op deze manier is het mogelijk om een veld te laten valideren aan de hand van een Regex of algemene regels zoals groter dan, kleiner dan, not null enzovoort. Om deze validatie tot stand te brengen werd gebruik gemaakt van FluentValidation [37]. Hiermee kan voor elk veld van het object een regel voorzien worden waaraan voldaan moet worden. Zo'n regel kan ook een eigen methode omvatten waarin de waarde van dit veld kan gecontroleerd worden. Telkens wanneer een DTO verstuurd wordt naar de API kan deze validatie toegepast en gecontroleerd worden aan de hand van de ModelState.IsValid eigenschap en hierdoor kan een aangepaste error teruggegeven worden bij faling. Aan elke validatieregel kan ook een aangepast bericht toegevoegd worden. Dit bericht wordt dan verzameld in de fouten binnen ModelState en kan zo ook doorgegeven worden naar de client.

```
class CourseDTOValidator : AbstractValidator<CourseDTO>{

    0 references | Nora Kamoen, 5 hours ago | 1 author, 4 changes | 4 work items
    public CourseDTOValidator() {
        RuleFor(c => c.Code).Must(CodeValidator).WithMessage("Give valid code for course");
        RuleFor(c => c.Title).NotEmpty();
        RuleFor(c => c.Duration).NotEmpty().GreaterThan(0);
        RuleFor(c => c.Responsible).NotNull();
        RuleFor(c => c.BusinessOwner).NotEmpty();
        RuleFor(c => c.CourseCategories).NotNull();
        RuleFor(c => c.Price).GreaterThanOrEqualTo(0);
    }

    /// <summary> Validates the code.
    1 reference | Nora Kamoen, 5 hours ago | 1 author, 4 changes | 5 work items
    public bool CodeValidator(string code) {
        if (code != null && !code.Equals("")) {
            string pattern = "^[a-zA-Z0-9]{0,6}$";
            if (Regex.IsMatch(code, pattern))
                return true;
            else
                return false;
        }
        return true;
    }
}
```

Figuur 18 Voorbeeld Validator CourseDTO volgens FluentValidation

Wanneer DTO's ontvangen of verzonden dienen te worden moeten deze omgezet worden van of naar de oorspronkelijke entiteit. Deze omzettingen kunnen manueel gemaakt worden door elk veld apart toe te kennen maar dit kan ook een stuk eenvoudiger door gebruik te maken van een mapping bibliotheek. Binnen de ontwikkeling van deze backend werd gebruik gemaakt van Automapper. Om de mogelijke mappings aan te maken dienen deze geregistreerd te worden in een configuratiebestand. Voor elke mogelijke mapping moet hierin een regel toegevoegd worden waarin vastgelegd wordt wat het oorspronkelijke object en het doelobject is. Wanneer de mappings in beide richtingen mogelijk moeten zijn kan de methode ReverseMap() toegevoegd worden achter de regel.

Sommige mappings bevatten een bewerking op velden van het oorspronkelijke object zoals bij de entiteit 'Company'. Hierbij is er een mapping nodig waarbij er slechts één adres moet toegevoegd worden aan de DTO, het primair adres. Om dit mogelijk te maken kan gebruik gemaakt worden van de methodes ForMember() en MapFrom() waarin dan het doelveld bepaald wordt en de bewerking op het oorspronkelijke veld.

```
CreateMap<Company, CompanyCompleteDTO>().ReverseMap();
CreateMap<Company, CompanyListDTO>().ReverseMap();
CreateMap<CompanyDTO, CompanyListDTO>().ReverseMap();
CreateMap<Company, CompanyDTO>()
    .ForMember(dest => dest.Address, (IMemberConfigurationExpression<Company> comp) => comp
    .MapFrom(src => src.Addresses.Where(a => a.IsPrimary).FirstOrDefault())).ReverseMap();
CreateMap<Person, PersonDTO>()
    .ForMember(p => p.Languages, (IMemberConfigurationExpression<Person> person) => person
    .MapFrom(p => p.Languages)).ReverseMap();
CreateMap<Person, PersonNameDTO>().ReverseMap();
CreateMap<Session, SessionCompleteDTO>()
    .ForMember(s => s.Code, (IMemberConfigurationExpression<Session> sess) => sess
    .MapFrom(src => string.Concat(src.Year, "-", src.SequenceNumber))).ReverseMap();
CreateMap<Session, SessionDTO>()
    .ForMember(s => s.Code, (IMemberConfigurationExpression<Session> sess) => sess
    .MapFrom(src => string.Concat(src.Year, "-", src.SequenceNumber))).ReverseMap();
CreateMap<Session, SessionListDTO>().ReverseMap();
```

Figuur 19 Voorbeelden registraties Automapper

5.5 WebAPI

Na de uitwerking van de volledige domeinlaag kan gewerkt worden aan de WebAPI. Hierbinnen dienen de verschillende controllers aangemaakt en uitgewerkt te worden alsook de nodige configuraties. Tijdens de ontwikkeling zijn verschillende beslissingen gemaakt omtrent de http-methodes die toegankelijk zijn alsook de toegang tot de API via de standaard IIS-server of Owin.

5.5.1 Controllers

Voor elke Aggregate root en alle overige entiteiten die geen child entity zijn van ‘Course’, ‘Session’ en ‘Company’ is er een controller aanwezig binnen de API. Via deze controllers is het mogelijk om DTO’s op te halen, te versturen en aan te passen. In totaal zijn er zo elf controllers aanwezig voor volgende objecten:

- BusinessOwner
- Course
- CourseCategory
- Company
- MaterialCategory
- Session
- State
- SessionType
- Language
- EvaluationQuestionOption
- EvaluationQuestion

Alle andere entiteiten worden via de controllers van de Aggregate Roots behandeld. Op deze wijze blijft de denkwijze van de Aggregate Root sterk aanwezig waarbij alle toegang tot de child entities via de root entiteit verloopt.

5.5.2 Put vs. Patch

Oorspronkelijk waren in alle controllers methodes aanwezig voor Get, Post, Delete en Put. Elke controller heeft een Get methode om alle objecten van deze entiteit uit de database op te halen en ook een Get methode voor een specifiek object. Hiernaast heeft elke controller een Post methode om een object toe te voegen en een Delete methode om een gewenst object te verwijderen. In het begin van de ontwikkeling had elke controller een Put methode maar deze is later vervangen door Patch daar dit voordeliger is voor data transfers.

Binnen de Put methode werden alle velden van het object overlopen en geüpdatet en werd ook steeds het volledige object verzonden naar de API. Echter is dit een heel overladen manier wanneer men te maken heeft met grotere objecten en Aggregate Roots. Hierbij dienen dan ook telkens alle child entities verzonden te worden naar de API ook wanneer hier niets aan veranderd is. Om deze overladen wijze te vermijden werd later overgeschakeld naar de Patch methode. Binnen de Patch methode worden enkel deze velden of objecten verzonden die gewijzigd, toegevoegd of verwijderd dienen te worden. Om dit mogelijk te maken wordt gebruik gemaakt van een JsonPatchDocument volgens het

Marvin.JsonPatch package [38]. In dit patchdocument wordt voor elke wijziging een regel voorzien die als volgt opgebouwd is:

{ “Op”: “Replace”, “Path”: “/Name”, “Value”: “RealDolmen” }

Het veld “Op” staat voor het type operatie dat nodig is om de wijzigingen door te voeren. Mogelijkheden zijn Replace, Add, Remove, Copy. Het “Path” geeft weer waar in het object de verandering dient doorgevoerd te worden en “Value” geeft de gewenste waarde mee. De value van de operatie kan ook een object zijn zoals een “Address” binnen het “Company” object. Om het patchdocument toe te passen wordt gebruik gemaakt van de methode ApplyTo() waarbij het doelobject dient meegegeven te worden. Aangezien gebruik gemaakt wordt van DTO’s om de objecten uit te wisselen met de client, moet dit patchdocument ook toegepast worden op een DTO-object. Om hierna de wijzigingen door te voeren op het uiteindelijke object uit de database wordt gebruik gemaakt van services bij Aggregate Roots om ook alle child objecten te overlopen en up te daten.

```
MaterialCategoryDTO dto = mapper.Map<MaterialCategory, MaterialCategoryDTO>(cat);
matCatPatchDocument.ApplyTo(dto);
```

Figuur 20 Toepassing JsonPatchDocument

5.5.3 RESTful API

Een andere wijziging die gemaakt is tijdens de ontwikkeling van de API is de evolutie naar RESTful API. Voorheen waren methodes aanwezig om relaties aan te maken tussen objecten zoals AddPersonToCompany en DeletePersonFromCompany waaraan dan het Person object en de CompanyID meegegeven diende te worden, echter zijn deze methodes niet RESTful. Dit betekent dat de methode betrekking heeft op twee soorten objecten en tevens niet automatisch herkend kan worden als een ‘Post’, ‘Get’, ‘Delete’ of ‘Patch’ methode. In een RESTful API bevat elke controller enkel routes met entiteiten in de benaming en geen werkwoorden zoals ‘GET’, ‘UPDATE’, ‘DELETE’. Bij de eerder vermelde methodes waren de routes ‘api/Companies/{id}/Persons/Add’ en ‘api/Companies/{id}/Persons/Delete’. Om dit te vermijden werd elke controller opnieuw bekeken en gereduceerd tot slechts twee Get methodes (alles en één entiteit), één delete methode en één Patch methode. Al deze voormelde methodes hebben enkel betrekking op de hoofdentiteit van de controller. Relaties worden dan niet langer via aparte routes gemaakt maar wel via de patch methode. Op deze manier zijn alle controllers RESTful. De verschillende http-methodes worden op deze manier ook automatisch herkend via het voorgaande ‘Post’, ‘Get’, ‘Delete’ of ‘Patch’ woord.

Naast de vijf standaardmethodes bevatten de controllers van Aggregate Roots ook Get methodes om de lijsten van hun child entities op te halen. Dit om te vermijden dat al deze child entities telkens met de Aggregate Root meegegeven worden wat soms tot heel grote objecten kan leiden. Bijvoorbeeld bij ‘Company’ zijn er aparte methodes om de ‘Persons’ of ‘Employees’ en ‘Addresses’ op te halen van een specifiek bedrijf. Wanneer deze telkens zouden meegegeven worden met de Get methodes van de Company zelf zou men een lijst van alle bedrijven met daarin telkens een lijst van verscheidene adressen en een grote lijst van werknemers doorsturen. Dit kan heel snel oplopen en is ook niet steeds nodig. Bijvoorbeeld in het overzicht van de Companies is enkel de naam van het bedrijf en eventuele code nodig en bij andere weergaves zijn dan enkel de Employees van een

bepaald bedrijf nodig. Door deze data te scheiden van elkaar in de API-routes wordt telkens enkel opgehaald wat nodig is en ontstaat er geen overhead.

Door extra Get methodes toe te voegen voor child entities wordt er ingegaan tegen de principes van Aggregate Roots in DDD. Daarbij is het de bedoeling om de child entities enkel via de root entity te benaderen en deze niet afzonderlijk op te halen. Echter is dit in deze situatie ver van optimaal en werd gekozen voor extra Get methodes om de werking van het programma te verbeteren.

5.5.4 Owin

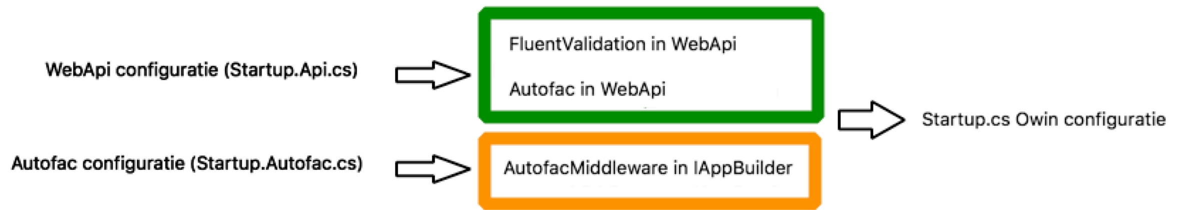
Binnen de applicatie werd vooreerst vertrokken vanuit de standaard WebAPI configuraties gebruik makende van IIS-server. Hiervoor werden alle instellingen bijgehouden in het Global.asax bestand. Om de mogelijkheid open te laten om over te schakelen naar een andere webserver wordt deze configuratie vervangen door Owin.

Owin maakt gebruik van een eigen configuratie bestand wat de Global.asax vervangt. Dit bestand heet Startup.cs en krijgt een klasse attribuut wat indiceert dat deze klasse de startup configuratie bevat. Binnen deze klasse worden alle nodige configuraties gegroepeerd die op hun beurt kunnen gebruikt worden door de webserver. Binnen het configuratie bestand wordt gebruik gemaakt van IAppBuilder om alle configuraties aan te linken.

De nodige configuraties in de backendontwikkeling zijn deze voor Autofac, WebAPI, FluentValidation en de standaard Cors configuratie. Al deze dienen aangemaakt te worden in het Startup bestand net zoals deze vroeger werden aangemaakt in Global.asax.

Om Autofac te configureren binnen de Owin WebAPI wordt eerst het Autofac configuratiebestand aangeroepen waarna de resulterende container met alle geregistreerde dependencies wordt toegevoegd aan de IAppBuilder. Door middel van deze toevoeging vormt de Autofac configuratie ook een soort Middleware.
Ook voor WebAPI dient de configuratie toegevoegd te worden aan de IAppBuilder instantie. Dit gebeurt door eerst een instantie te maken van het Startup.Api.cs bestand en hiervan de resulterende HttpConfiguration toe te voegen aan de Owin configuratie. Het resultaat van de API-configuratie kan tevens ook gebruikt worden om Autofac aan de WebAPI te linken via de HttpConfiguration. Binnen het specifieke API-configuratie bestand worden alle routes binnen de API geregistreerd alsook de Cors configuratie waarin bepaald wordt welke acties mogelijk zijn en van welke bron requests mogen gemaakt worden naar de API.
Om ook de validatie van DTO’s, gebruik makend van FluentValidation, toegankelijk te maken wordt ook terug gebruik gemaakt van het API-configuratie bestand om deze instellingen toe te voegen aan de WebAPI.

De verschillende configuraties en hun link naar de uiteindelijke Owin configuratie wordt geïllustreerd in Figuur 21



Figuur 21 Owin configuratie

Naast de toevoeging van de verschillende configuraties die nodig zijn voor de functionaliteit van de applicatie wordt ook alles omtrent authenticatie toegevoegd aan de Owin configuratie. Deze instellingen behoorden niet tot de opdracht maar werden toegevoegd door de opdrachtgever zelf. Voor de authenticatie wordt gebruik gemaakt van een TokenAuthentication waarbij de gebruiker opgehaald wordt aan de hand van een CloudConfigurationManager.

5.6 Services

Om de controllers niet te overladen en slechts de functionaliteit te behouden voor het afhandelen van requests, wordt alle code verantwoordelijk voor het aanmaken van objecten of het verhandelen van updates binnen de patch methode verplaatst naar Services. Deze services zijn tevens ook afgesplitst in een ander project. Elke Aggregate Root heeft zo een eigen service. Binnen deze services zijn methodes aanwezig om het volledige object en ook de child entities hiervan te controleren en up te daten, gebruik makende van de DTO waarop het patchdocument op toegepast is.

Bij elke service wordt vertrokken van een interface waarin alle signaturen van de methodes genoteerd worden. Deze interface wordt dan geïmplementeerd in de klasse van de service. Zo zijn er in de applicatie drie gelijkaardige services aanwezig, één voor elke Aggregate Root zijnde 'Course', 'Company' en 'Session'. Naast deze drie services werd nog een extra aangemaakt voor de 'CourseCategory'. Dit omdat er gebruik gemaakt wordt van een geneste lijst om deze door te sturen naar de front-end zodat daar de verschillende categorieën gemakkelijk in een boomstructuur weergegeven kunnen worden.

5.6.1 Opbouw Aggregate Root services

Elke Service bevat een Updatemethode die alle velden van het object zal updaten gebruik makende van de DTO die doorgestuurd wordt vanuit de controllers. Om ook alle child entities up te daten wordt hiervoor gebruik gemaakt van extra Updatemethodes. Hieraan wordt dan telkens de lijst met DTO's meegegeven alsook het doelobject, de Aggregate Root.

```
public interface ISessionService {

    2 references | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    List<Remark> UpdateRemarks(List<RemarkDTO> dto, Session session);
    2 references | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    List<Attachment> UpdateAttachments(List<AttachmentDTO> dto, Session session);
    2 references | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    Task<List<SessionDay>> UpdateSessionDays(List<SessionDayDTO> dto, Session session);
    2 references | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    Task<Audience> UpdateAudience(AudienceDTO dto, Session session);
    2 references | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    Task<Session> UpdateSession(SessionCompleteDTO dto, Session session);
    2 references | Nora Kamoen, 11 days ago | 1 author, 1 change | 1 work item
    Task<Session> CreateSession(SessionDTO dto);
}
```

Figuur 22 Voorbeeld update methodes in Interface SessionService

Binnen deze extra updatemethodes voor de child entities wordt steeds de DTO lijst vergeleken met de lijst in de Aggregate Root. Wanneer een object wel aanwezig is binnen de DTO en niet in de Aggregate root dan wordt deze toegevoegd, is het in beide aanwezig wordt het vergeleken via een Equals() methode en geüpdatet wanneer verschillend. Wanneer het object wel aanwezig is in de Aggregate Root maar niet in de DTO dan zal dit object verwijderd worden aan de hand van een soft delete (IsActive = false). Om de objecten toe te voegen, up te daten of te verwijderen uit de Aggregate Root, wordt extra functionaliteit gebruikt binnen de klasse van de entiteit zelf.

Deze vergelijkingen van de child entities gebeuren aan de hand van foreach iteraties. Om de vergelijking te maken werd bekeken of dit ook via de Except() methode zou kunnen, echter werkt dit enkel goed voor het toevoegen van objecten aan de lijst. Ook voor het verwijderen is extra code nodig aangezien dit een update van het veld 'IsActive' vereist. Om objecten up te daten is ook nog steeds extra logica nodig waardoor het gebruik van Except() geen tot weinig voordelen biedt.

Naast de updatemethodes bevatten deze services ook een Create() methode dat gebruik maakt van de Create() methode in de root entiteit zelf. Binnen deze methode worden de nodige velden gedeclareerd gebruik makende van de DTO dat doorgestuurd wordt alsook berekeningen waar nodig zoals 'Year' binnen het Session object.

5.6.2 CourseCategory service

De vierde service, deze voor de CourseCategory bevat slechts één updatemethode daar er enkel gebruik gemaakt wordt van de CourseCategory entiteit. Andere methodes binnen deze service zijn om van een gewone List<> een geneste lijst te maken en vice versa. Hiervoor werd gebruik gemaakt van een nieuwe DTO, de 'NestedDTO'. Deze DTO bevat telkens een object en een lijst met 'children'.

Om de geneste lijst te maken worden eerst uit de oorspronkelijke lijst alle 'Root' categories opgehaald, dat zijn deze zonder parentcategory. Vanuit deze roots worden dan de children opgehaald en van hieruit verloopt het verdere proces recursief. Zo kan elke childcategory opnieuw children bevatten. De resulterende geneste lijst wordt dan gebruikt om naar de front-end te sturen zodat deze daar in een boomstructuur kan weergegeven worden.

Wanneer de volgorde in de lijst verandert in de front-end wordt de volledige geneste lijst teruggestuurd naar de backend. Daar wordt van de geneste lijst terug een platte lijst gemaakt door hetzelfde proces te volgen maar deze dan telkens toe te voegen aan een gewone lijst. Wanneer dit proces afgerond is gaat elk object door een Equals() methode en wordt het vergeleken met het oorspronkelijke. Wanneer er iets gewijzigd is, zoals de parentcategory, wordt dit geüpdatet en kan daarna terug een geneste lijst gemaakt worden met het resultaat en de aangepaste volgorde.

5.7 Dependency Injection

Doorheen de volledige applicatie wordt gebruik gemaakt van dependency injection. Om hierin te voorzien wordt gekozen voor Autofac die gebruik maakt van een ContainerBuilder. Hiervoor dienen eerst de nodige packages geïnstalleerd te worden waarna alle registraties toegevoegd kunnen worden.

Alle Autofac configuraties worden gemaakt binnen Startup.Autofac.cs. Als eerste wordt hierin een ContainerBuilder aangemaakt waarin alle nodige dependencies in geregistreerd zullen worden. De eerste registratie is deze van de verschillende controllers, deze worden opgehaald door middel van reflectie via de methode GetExecutingAssembly(). Na de API controllers wordt ook de WebAPI Filter Provider toegevoegd aan de container gebruik makend van de HttpConfiguration.

Wanneer alle WebAPI onderdelen toegevoegd zijn kunnen alle verschillende repositories, services en Entity Framework onderdelen geregistreerd worden. Aangezien dit een variabel aantal bestanden kan zijn wordt dit het gemakkelijkst toegevoegd door middel van modules. Binnen deze modules worden alle bestanden van een bepaalde Assembly overlopen en deze die eindigen op een bepaalde benaming worden toegevoegd aan de ContainerBuilder.

De eerste Module is deze van de repositories en wordt weergegeven in Figuur 23. Hierbij worden alle bestanden binnen de RealDolmen.Education.Tropix.Data namespace overlopen en enkel deze geregistreerd die eindigen op “Repository”. Ook wordt hierin gekeken dat enkel de interfaces hiervan toegevoegd worden aangezien Autofac de dependencies legt aan de hand van interfaces en hun implementerende klassen. Daar de implementerende klassen ook variabel zijn wordt enkel gebruik gemaakt van de interfaces waardoor er steeds toegang is tot de verschillende implementerende klassen.

```
2 references | Nora Kamoen, 69 days ago | 1 author, 3 changes | 4 work items
public class RepositoryModule : Autofac.Module {

    1 reference | Nora Kamoen, 69 days ago | 1 author, 3 changes | 4 work items
    protected override void Load(ContainerBuilder builder) {
        builder.RegisterAssemblyTypes(Assembly.Load("RealDolmen.Education.Tropix.Data"))
            .Where(t => t.Name.EndsWith("Repository"))
            .AsImplementedInterfaces()
            .InstancePerLifetimeScope();
    }
}
```

Figuur 23 Module repositories

Een tweede module is de ServiceModule. Hierin worden alle verschillende services geregistreerd aan de hand van hun interface. Om deze interfaces te verzamelen worden alle bestanden binnen RealDolmen.Education.Tropix.Services overlopen en deze toegevoegd die eindigen op “Service”.

Als laatste module is er de Entity Framework Module. Hierin wordt de datacontext geregistreerd en eventueel ook het Unit Of Work wanneer deze aangemaakt is. Deze registraties gebeuren wel manueel daar dit slechts één bestand inhoudt.

Naast de verschillende modules dienen tevens ook de verschillende mapping configuraties toegevoegd te worden. Dit gebeurt door het registreren van het AutoMapper configuratiebestand. Ook deze registratie wordt manueel toegevoegd via de Resolve() methode.

Wanneer alle registraties genoteerd zijn, al dan niet aan de hand van modules, kan de ContainerBuilder omgezet worden naar een feitelijke container. Deze resulterende container wordt dan gebruikt binnen de Owin configuratie om een AutofacMiddleware te vormen.

5.8 Testen

Om de functionaliteit van de applicatie op punt te stellen en ook foutloos te maken wordt gebruik gemaakt van testen. Enerzijds wordt gebruik gemaakt van XUnit tests om pure klassen te testen en specifieke methodes en anderzijds worden ook Owintests toegevoegd waarbij ook de API functionaliteit en de toegang hiertoe getest wordt.

Naast de verschillende testen wordt tevens een Mocking database voorzien om ervoor te zorgen dat geen nutteloze data naar de database verzonden wordt. Een andere reden om de database te mocken is om de tests ook toegankelijk te maken binnen de Visual Studio Team Services.

5.8.1 Tests

5.8.1.1 XUnit

Binnen de XUnit tests werden testen toegevoegd die zowel de controllers testen alsook de repositories en services. Om deze tests tot stand te brengen dienen eerst de nodige packages geïnstalleerd te worden waarna de verschillende testklassen aangemaakt kunnen worden.

Elke testklasse wordt geïnitialiseerd aan de hand van een constructor. Hier worden voor de tests ook manueel alle dependencies aangemaakt en niet via Autofac. Naast de constructor heeft elke testklasse ook standaard een Dispose() methode daar de testklassen de IDisposable interface implementeren. Deze constructor en Dispose methode vervangen de setup en teardown fixtures uit het klassieke Nunit framework.

Alle methodes krijgen een Fact (voorbeeld in Figuur 24) of Theory (voorbeeld in Figuur 25) attribuut wat ervoor zorgt dat de methode aanzien wordt als een testmethode. Bij Theory dient telkens data meegegeven te worden met de methode aan de hand van het InlineData attribuut. Deze data wordt dan doorgegeven aan de methode aan de hand van parameters en zo kan een enkele methode verschillende tests uitvoeren voor elke lijn InlineData. Wanneer men het project built verschijnt aan elke testmethode een symbool waarbij men ervoor kan kiezen om de test te runnen of te debuggen. Alle tests kunnen ook in een keer geselecteerd worden om uit te voeren via de test explorer, hier staan alle testmethodes opgelijst. Wanneer een test slaagt krijgt deze een groen symbool, wanneer hij faalt een rood.

```
[Fact]
| 0 references | 0 changes | 0 authors, 0 changes
public async void GetCompanies() {
    var result = await contr.GetCompanies("r");
    OkNegotiatedContentResult<List<CompanyListDTO>> conResult = Assert.IsType<OkNegotiatedContentResult<List<CompanyListDTO>>>(result);
    foreach (CompanyListDTO name in conResult.Content) {
        var addresses = await contr.GetAddresses(name.ID);
        OkNegotiatedContentResult<List<AddressDTO>> adResult = Assert.IsType<OkNegotiatedContentResult<List<AddressDTO>>>(addresses);
        var employees = await contr.GetEmployees(name.ID);
        OkNegotiatedContentResult<PaginationDTO<PersonDTO>> empResult = Assert.IsType<OkNegotiatedContentResult<PaginationDTO<PersonDTO>>>(employees);
    }
}
```

Figuur 24 Voorbeeld testmethode Fact

```
[Theory]
[InlineData("RealDolmen De Pinte", "Grote steenweg 30", "De Pinte", "9840", "België", false)]
[InlineData("RealDolmen Kontich", "Appelstraat 10", "Kontich", "1240", "België", false)]
0 references | 0 changes | 0 authors, 0 changes
public async Task PatchCompanyTest(string name, string street, string city, string zipcode, string country, bool primary) {
    var result = await contr.GetCompany(1);
    OkNegotiatedContentResult<CompanyDTO> compResult = Assert.IsType<OkNegotiatedContentResult<CompanyDTO>>(result);
    var addresses = await contr.GetAddresses(compResult.Content.ID);
    OkNegotiatedContentResult<List<AddressDTO>> adResult = Assert.IsType<OkNegotiatedContentResult<List<AddressDTO>>>(addresses);
    int count = adResult.Content.Count;

    AddressDTO ad = new AddressDTO();
    ad.City = city;
    ad.Street = street;
    ad.Zipcode = zipcode;
    ad.Country = country;
    ad.IsPrimary = primary;
    ad.Name = name;
```

Figuur 25 Voorbeeld testmethode Fact met InlineData

Alle XUnit tests worden tevens ook uitgevoerd wanneer de code gebuild wordt in de Visual Studio Team Services. Aangezien alle tests betrekking hebben op data kunnen deze tests online enkel slagen wanneer daar ook een database aanwezig is, echter is dit niet het geval bij een gewone build daar enkel de code toegankelijk is. Om dit probleem op te lossen wordt gebruik gemaakt van een mocking database, dit wordt besproken onder 5.8.2 Mocking Database.

5.8.1.2 Owintesting

Een tweede reeks testen werd gemaakt aan de hand van Owintesting. Dit laat toe om tests te maken die zich gedragen als een API client. Hiervoor wordt gebruik gemaakt van de Startup klasse om verbinding te maken met de API.

Binnen de Owintests worden alle verschillende routes die betrekking hebben op de Aggregate Roots ‘Course’, ‘Company’ en ‘Session’ getest. Dit zijn zowel de Get, Add, Delete en Patch routes. Naast alles van deze drie entiteiten worden ook andere entiteiten aangemaakt of gewijzigd die nodig zijn om een Aggregate Root aan te maken zoals ‘CourseCategory’, ‘SessionState’ enzovoort. Elke route wordt geconsulteerd op dezelfde wijze als een client dit zou doen. Zo is het ook mogelijk om de juiste foutmeldingen door te krijgen en de verschillende response messages te testen. Het voordeel hieraan is dat bij faling de API nog steeds gedebugd kan worden.

Een voorbeeld test binnen Owintesting wordt weergegeven in Figuur 26. Hierin wordt een Delete actie uitgevoerd alsook een Get om het resultaat te controleren daar een Delete een NoContent als resultaat geeft.

```
[Fact]
| 0 references | Nora Kamoen, 18 days ago | 1 author, 3 changes | 2 work items
public async void DeleteCompanyTest() {
    using (var server = TestServer.Create<Startup>()) {
        var result = await server.HttpClient.DeleteAsync("api/companies/1");

        var resultAfterDelete = await server.HttpClient.GetAsync("api/companies/1");

        CompanyDTO originalcompany = resultAfterDelete.Content.ReadAsAsync<CompanyDTO>().Result;
        Assert.Equal(false, originalcompany.IsActive);
    }
}
```

Figuur 26 Voorbeeld Owintest Delete en Get route

5.8.2 Mocking database

Om de tests ook online in de Visual Studio Team Services te laten slagen en ervoor te zorgen dat de testdata niet telkens in de database terecht komt wordt een mocking database aangemaakt. Hiervoor wordt allereerst gebruik gemaakt van Json bestanden waarin van elk type entiteit enkele objecten gedefinieerd zijn.

Om de database op te bouwen worden de verschillende objecten uit de Json bestanden gelezen en omgevormd naar echte objecten die verzameld worden in lijsten. Om ook de relaties te maken tussen de verschillende objecten dienen deze manueel geconfigureerd te worden. Deze relaties worden gelegd aan de hand van de opgestelde lijsten vanuit de Json bestanden met behulp van linq queries. Dit wordt weergegeven in Figuur 27 en Figuur 28.

```
var languages = new List<Language>();
foreach (var l in mockLanguages["Languages"]) {
    Language result = Language.CreateLanguage((string)l["Name"]);
    result.ID = (int)l["ID"];
    languages.Add(result);
}
```

Figuur 27 Inlezen en omvormen Json objecten

```
foreach (var item in persons) {
    item.Courses = courses.Where(x => x.Responsible.ID == item.ID).ToList();
    item.Languages = languages.Where(x => x == item.Languages.ToList().Where(l => l.ID == x.ID)).ToList();
}
```

Figuur 28 Vastleggen van relaties

Gebruik makend van deze lijsten worden dan DBSet() instanties gemaakt die vervolgens kunnen toegevoegd worden aan de mocking context.

In de verschillende tests wordt steeds gebruik gemaakt van de mockingcontext aan de hand van diens Setup klasse. Deze context wordt dan meegegeven aan de verschillende repositories waardoor de data niet langer naar de echte database verzonden wordt maar naar de in-memory mocking database.

Door gebruik te maken van een mocking context dienen enkele dingen in code aangepast te worden zoals de toegang tot de verschillende tabellen in de database. Dit kan niet langer door de naam van de tabel maar gebeurt via Set<Type>(). Ook is de CurrentValues() functionaliteit niet toegankelijk in een mocking database. Dit moet of vervangen worden in een mocking methode of aangepast worden in de echte code om dezelfde functionaliteit te kunnen aanbieden bij een mocking database.

5.9 Application Insights

Als laatste wil RealDolmen Education ook een overzicht van het verkeer naar de API. Dit kan verwezenlijkt worden aan de hand van de toevoeging van Application Insights. Dit is een service die Microsoft aanbiedt en die heel eenvoudig kan toegevoegd worden aan een standaard WebAPI. Binnen Application Insights worden alle requests geregistreerd en ook de nodige foutmeldingen bijgehouden. Hierdoor is het mogelijk om een gedetailleerd overzicht te verkrijgen van welke routes druk bezocht zijn en welke problemen opleveren.

Om deze service toe te voegen dienen de nodige packages geïnstalleerd te worden. Wanneer men gebruik maakt van de standaard opstelling van WebAPI, deze waarbij gebruik gemaakt wordt van IIS-server, dienen geen extra configuraties gemaakt te worden. Echter omdat in deze applicatie gewerkt wordt met Owin, dient de Application Insights toegevoegd te worden aan de Owin Middleware. Deze toevoeging gebeurt als eerste door het aanmaken van een TelemetryClient. Deze Client registreert alle verkeer over de API [39].

Om ervoor te zorgen dat er een uitgebreide rapportering gebeurt binnen de Application Insights dienen deze expliciet gelogd te worden wanneer men gebruik maakt van Owin. Dit gebeurt aan de hand van een InsightsReportMiddleware die gebruik maakt van de TelemetryClient. In deze klasse wordt voor elke request de nodige data opgehaald en toegevoegd aan de TelemetryClient.

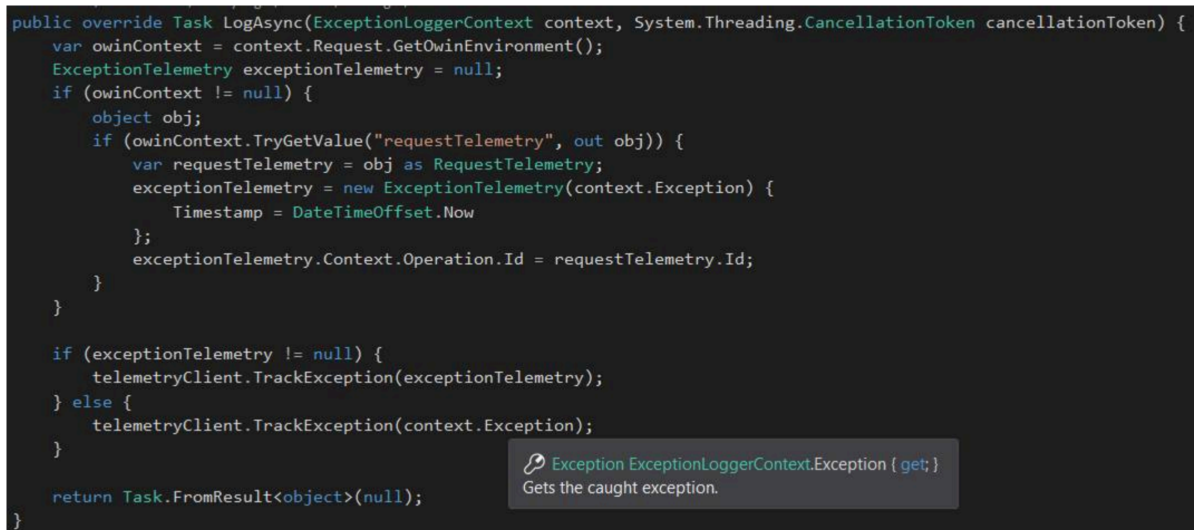
```
    public async Task Invoke(IDictionary<string, object> environment) {
        var ctx = new OwinContext(environment);
        var rt = new RequestTelemetry() {
            Url = ctx.Request.Uri,
            HttpMethod = ctx.Request.Method,
            Name = ctx.Request.Path.ToString(),
            Timestamp = DateTimeOffset.Now
        };
        environment.Add("requestTelemetry", rt);

        var sw = new Stopwatch();
        sw.Start();
        await next(environment);
        sw.Stop();

        rt.ResponseCode = ctx.Response.StatusCode.ToString();
        rt.Success = ctx.Response.StatusCode < 400;
        rt.Duration = sw.Elapsed;
        telemetryClient.TrackRequest(rt);
    }
}
```

Figuur 29 Logging van requests voor Application Insights

In bovenstaande figuur (Figuur 29) wordt de logging van alle requests weergegeven. Dit gebeurt aan de hand van een StopWatch om de verstreken tijd tijdens de request ook te registreren. Hiermee wordt extra informatie doorgegeven over welke requests lang duren waardoor tijdens de ontwikkeling ook hiervoor verbeteringen kunnen doorgevoerd worden.

Wanneer een fout optreedt in de API en dus de request niet slaagt, wordt hiervoor een aparte afhandeling voorzien. Deze gebeurt binnen AiExceptionLogger. Voorgaande klasse wordt toegevoegd aan de HttpConfiguration als een service in de Owin Startup.cs klasse. Binnen de AiExceptionLogger klasse wordt de OwinContext opgehaald en wordt deze

gebruikt om de huidige request op te halen. Wanneer deze request een exceptie als resultaat heeft wordt deze request in de TelemetryClient gelogd als een exceptie, anders volstaat de gewone registratie via de InsightsReportMiddleware.



Figuur 30 ExceptionLogger methode voor Application Insights

In bovenstaande methode (zie Figuur 30) binnen de AiExceptionLogger klasse wordt voor elke Exception binnen de huidige OwinContext het tijdstip bijgehouden alsook de ID. Echter wanneer de ExceptionTelemetry de exceptie niet kon loggen wordt deze toegevoegd aan de TelemetryClient met behulp van de TrackException() methode en de expliciete ophaling van de exceptie via de OwinContext.

5.10 Visual Studio Team Services

5.10.1 Continuous Integration

De volledige ontwikkeling van de applicatie wordt opgevolgd door het gebruik van Git als version control systeem. Om te voorzien in Continuous Integration of CI wordt per wijziging in de ontwikkeling van de applicatie de aanpassingen ingecheckt en doorgestuurd naar de TFS-server waar op gepaste wijze de volledige applicatie gebuild wordt en er een rapport beschikbaar gesteld wordt met de resultaten.

Binnen het buildproces zijn ook de tests opgenomen die geen betrekking hebben op Owintesting. Alle resultaten van deze tests kunnen dan achteraf ook bekeken worden in het web platform. Om alle stappen in de ontwikkeling duidelijk bij te houden wordt er ook gebruik gemaakt van de Agile werkomgeving binnen TFS-server. Hierin werd gewerkt in sprints van twee weken. Meer over deze Agile werkomgeving wordt besproken onder 5.10.3 Agile.

5.10.2 Continuous Delivery

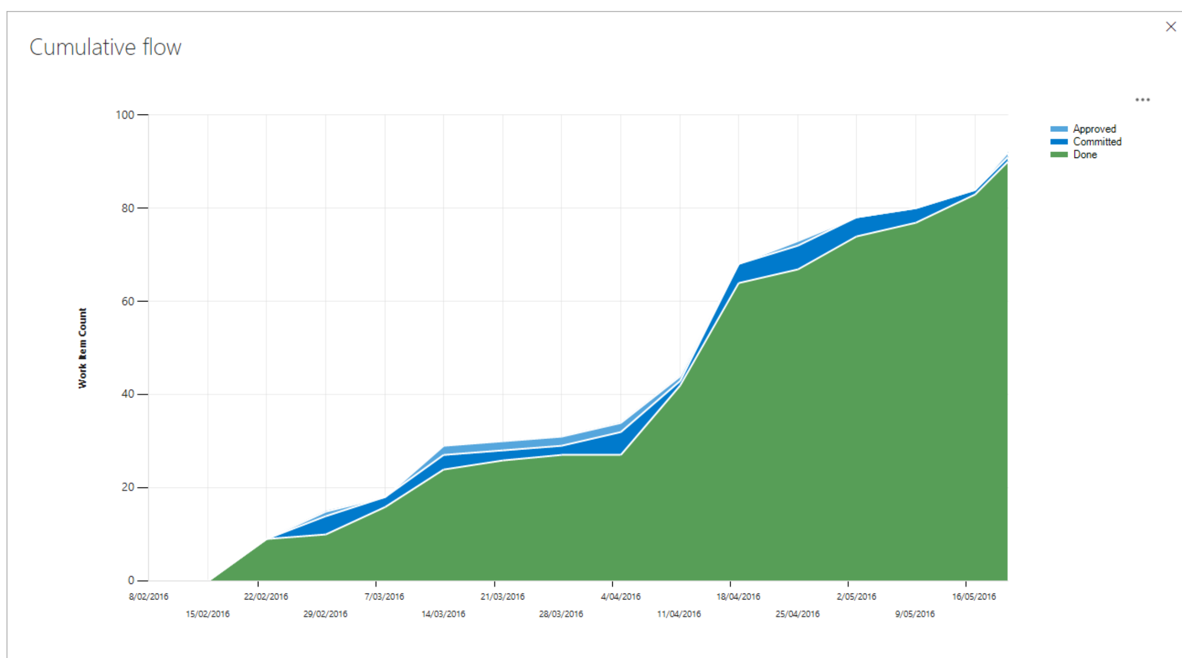
Naast de CI werd ook voorzien in Continuous Delivery of CD. Elke grote deeltaak werd opgesplitst in een nieuwe branch. Wanneer deze deeltaak afgewerkt was of het nodig was om de gemaakte verbeteringen online te brengen zodat de front-end ontwikkeling verder kon, werd deze aparte branch gemerged en werd hiervan een release gemaakt. Binnen deze release werd ook gebruik gemaakt van de Dacpac om de database up to date te houden.

Deze Dacpac is afkomstig van het database project. Hierin wordt de aangemaakte database via Entity Framework gebruikt. Telkens wanneer een migration toegepast wordt binnen het project dient ook een schema compare uitgevoerd te worden op dit databaseproject. Binnen deze schema compare wordt de database vergeleken met het database project en kunnen eventuele wijzigingen doorgevoerd worden naar het project toe. Wanneer deze update gebeurd is dient het project gebuild te worden en wordt er op zijn beurt een Dacpac gecreëerd waarmee bij deployment de onlinedatabase gebouwd kan worden.

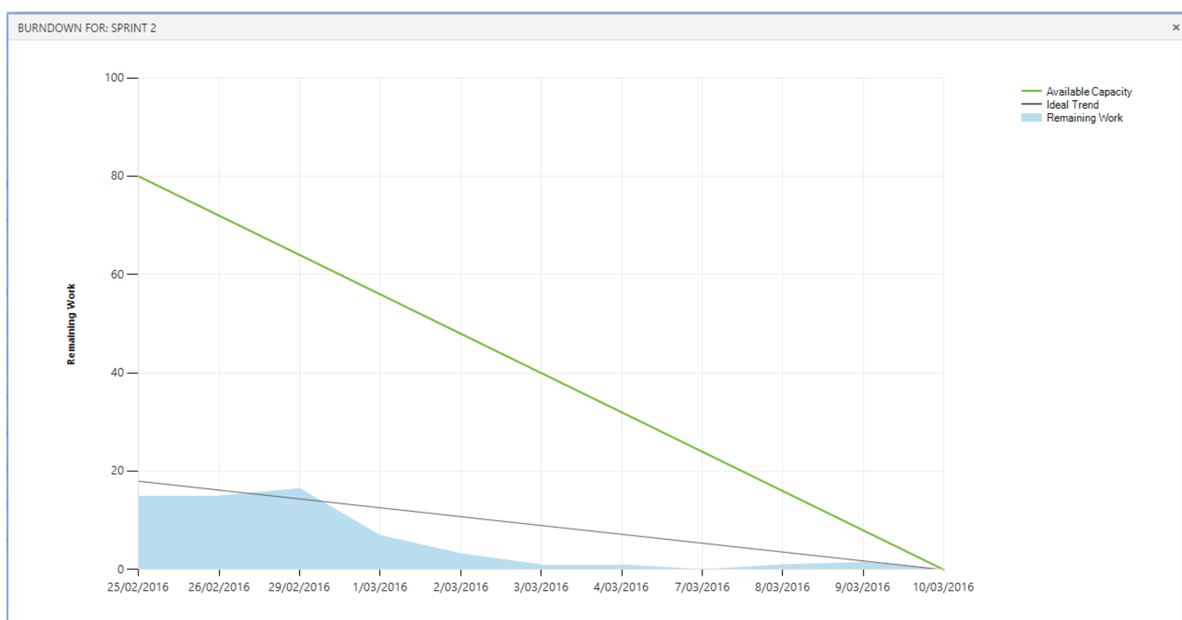
5.10.3 Agile

Binnen de TFS-server wordt ook gebruik gemaakt van de Agile werkomgeving. De volledige ontwikkelingsperiode is opgesplitst in sprints van twee weken waarin de nodige backlogitems aangemaakt werden. Per backlogitem werden ook alle kleine deeltaken toegevoegd met een schatting van de tijd dat deze in beslag zullen nemen alsook een toewijzing aan de persoon die deze dient uit te voeren. Bij elke checkin die dan gebeurt vanuit Visual Studio kunnen deze taken toegevoegd worden. Hierdoor kan in de code history de evolutie bekeken worden met telkens het juiste item hieraan gekoppeld.

Naar aanleiding van het gebruik van de backlogitems met de taken wordt per sprint ook een burndown bijgehouden dat de evolutie weergeeft van hoeveel taken er reeds voltooid zijn in de lopende sprint. Naast deze grafiek is er ook een algemene weergaven van alle backlogitems over de verschillende sprints heen waarin weergegeven wordt hoeveel er reeds voltooid zijn en hoeveel er nog in ontwikkeling zijn. In onderstaande figuren wordt de breakdown van sprint 2 weergegeven alsook het algemeen verloop van alle backlogitems.

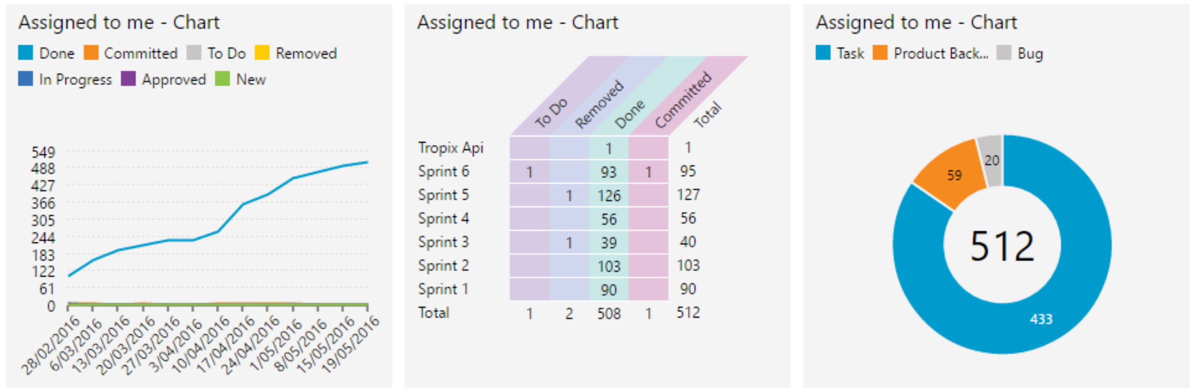


Figuur 31 Algemeen verloop backlogitems



Figuur 32 Burndown sprint 2

Voor de specifieke taken worden ook grafieken bijgehouden die weergeven hoeveel taken in welke sprint voltooid zijn, hoeveel backlogitems aangemaakt zijn en hoeveel bugs gerapporteerd zijn binnen het systeem. Op deze manier kan doorheen de ontwikkeling bijgehouden worden welke sprints veel taken bevatten en hoeveel bugs opgelost dienden te worden alsook hoeveel taken verwijderd werden enzovoort. Deze grafieken worden weergegeven in Figuur 33.



Figuur 33 Vooruitgang weergegeven in grafieken

Algemeen besluit

Bij deze bachelorproefopdracht was mijn grootste algemene doelstelling het aanleren van nieuwe technologieën. Mijn verwachtingen hieromtrent zijn volledig ingelost. Ik heb verscheidene nieuwe technologieën aangeleerd binnen WebAPI alsook het gebruik van Dependency Injection was volledig nieuw voor me. Naast de verschillende technologieën heb ik ook heel wat geleerd rond het gestructureerd opbouwen van een applicatie. Nu ben ik ervan overtuigd dat ik een applicatie kan opbouwen volgens de 'regels van de kunst'.

Toen ik begon aan deze opdracht verwachtte ik om aan het einde een feilloze functionaliteit te kunnen aanleveren voor hetgeen ik ontwikkeld heb. Echter is dit niet gelukt aangezien dit ook afhankelijk is van de evolutie binnen de front-end. Hierbij zijn er nog enkele fouten naar boven gekomen wanneer mijn opdracht reeds afgelopen was. Aan de hand van dit resultaat en mijn terugblik op de samenwerking tussen mezelf en de andere stagiair kan ik besluiten dat dit deels vermeden kon worden door een betere communicatie en frequenter samen te zitten om de zaken te overlopen.

Aansluitend bij vorige vaststelling kan ik bijvoegen dat over het algemeen het contact met collega's vlot en constructief verliep. Echter was de rapportering van mijn kant uit soms te weinig. Dit werkpunt werd aangehaald bij een tussentijdse evaluatie waardoor ik de kans had om dit bij te schaven in de laatste weken. Mijn eigen vorderingen kon ik wel steeds evalueren aan de hand van de Agile werkomgeving. Op deze manier had ik een goed overzicht van de taken die reeds afgerond waren alsook van deze die nog in de wachtrij stonden. Deze vorderingen werden ook wekelijks binnen het team overlopen tijdens een stand-up meeting waardoor iedereen een duidelijk zicht had op de vooruitgang. Na elke sprint vond ook een sprintreview plaats om na te gaan of het beoogde resultaat bereikt was.

De backend die ik ontwikkeld heb heeft als doel om in verschillende applicaties gebruikt te worden in de toekomst. Dit onder meer in de vernieuwde 'Tropix'-applicatie maar ook in het planningsbord, de website enzovoort. Daar dit voor verschillende toepassingen gebruikt dient te worden en tevens moet meegroeien met de bedrijfsstructuur was het belangrijk deze modulair op te bouwen. Mijn ontwikkeling geeft een grondige basis mee maar kan nog niet opgenomen worden voor de dagelijkse werking binnen het bedrijf. Hiervoor was de opdracht van te grote omvang en is dus nog verdere uitwerking noodzakelijk.

Wanneer ik mijn bereikte resultaat met een kritisch oog bekijk moet ik vaststellen dat sommige codefragmenten kwalitatiever geschreven kunnen worden. De oorzaak hiervan is een gebrek aan ervaring alsook het feit dat al deze technologieën een nieuwe manier van werken met zich meebrengen.

Doorheen het proces was ik genoodzaakt me een weg te banen in de nieuwe technologieën. Hierbij verplichtte ik mezelf om deze op zelfstandige wijze onder de knie te krijgen daar ik ervan overtuigd ben dat ik hierdoor het meeste bijleer. Wanneer er toch enkele zaken onduidelijk bleven kon ik alsnog te rade bij mijn stagementor, Tom Eeraerts.

Ook wanneer er fouten opdoken in het programma probeerde ik die zo veel mogelijk zelf te ontleden en op te lossen.

Tijdens de gehele periode heb ik ook nader kennis gemaakt met RealDolmen als bedrijf. Dankzij de gezonde bedrijfscultuur en de collegiale sfeer voelde ik me hier thuis en opgenomen in de groep. Dit werd versterkt door de ondersteuning en waardering voor mijn werk van de verschillende collega's.

Literatuurlijst

[1a] RealDolmen, „Over RealDolmen," 2014. [Online]. Available: http://www.realdolmen.be/nl/about. [Geopend 9 maart 2016].

Bibliografie

[1] Wikipedia, „Real Software," Wikimedia Foundation, Inc., 23 november 2014. [Online]. Available: https://nl.wikipedia.org/wiki/Real_Software. [Geopend 28 maart 2016].

[2] wdp, „Dolmen kort," 14 augustus 2007. [Online]. Available: http://www.standaard.be/cnt/bs1g0ana. [Geopend 28 maart 2016].

[3] RealDolmen, „Over RealDolmen," 2014. [Online]. Available: http://www.realdolmen.be/nl/about. [Geopend 9 maart 2016].

[4] RealDolmen, „Business Solutions," 30 september 2014. [Online]. Available: http://www.realdolmen.be/nl/services-solutions/business-solutions. [Geopend 28 maart 2016].

[5] RealDolmen, „Professional Services," 30 september 2014. [Online]. Available: http://www.realdolmen.be/nl/services-solutions/professional-services. [Geopend 28 maart 2016].

[6] RealDolmen, „Infrastructure Products," 20 augustus 2014. [Online]. Available: http://www.realdolmen.be/nl/services-solutions/infrastructure-solutions. [Geopend 28 maart 2016].

[7] RealDolmen, „Organisatie," 2014. [Online]. Available: http://www.realdolmen.be/nl/about/organization. [Geopend 9 maart 2016].

[8] Domain Laguage, Inc., „Domain-Driven Design," 2016. [Online]. Available: https://domainlanguage.com/ddd/patterns/. [Geopend 23 maart 2016].

[9] Wikipedia, „Domain-Driven Design," 11 februari 2016. [Online]. Available: https://en.wikipedia.org/wiki/Domain-driven_design. [Geopend 2 april 2016].

[10] P. Brown, „What is the difference between Entities and Value Objects," 30 april 2014. [Online]. Available: http://culttt.com/2014/04/30/difference-entities-value-objects/. [Geopend 15 februari 2016].

[11] M. Verraes, „Related Entities vs Child Entities," 30 december 2013. [Online]. Available: http://verraes.net/2013/12/related-entities-vs-child-entities/. [Geopend 23 februari 2016].

[12] MSDN Microsoft, „Track Data Changes (SQL Server)," 2014. [Online]. Available: https://msdn.microsoft.com/en-us/library/bb933994(v=sql.120).aspx. [Geopend 10 februari 2016].

[13] J. Galloway, „Adding simple trigger-based auditing to your SQL Server database," 21 januari 2008. [Online]. Available: http://stackoverflow.com/questions/7268350/how-to-log-data-changed-in-sql-server-tables-which-approach-is-better. [Geopend 10 februari 2016].

[14] K. Little, „How to decide if you should use table partitioning," 6 maart 2012. [Online]. Available: https://www.brentozar.com/archive/2012/03/how-decide-if-should-use-table-partitioning/. [Geopend 11 februari 2016].

[15] R. B. Paruchuri, „Soft Deleting Entities Cleanly Using Entity Framework 6 Interceptors," 28 augustus 2015. [Online]. Available: http://www.codeguru.com/csharp/csharp/soft-deleting-entities-cleanly-using-entity-framework-6-interceptors.html. [Geopend 15 ferbuari 2016].

[16] MSDN Microsoft, „Code First to a New Database," 2015. [Online]. Available: https://msdn.microsoft.com/en-us/data/jj193542.aspx. [Geopend 9 februari 2016].

[17] MSDN Microsoft, „Code First Migrations," 2015. [Online]. Available: https://msdn.microsoft.com/en-us/data/jj591621.aspx. [Geopend 10 februari 2016].

[18] MSDN Microsoft, „Loading Related Entities," Microsoft., 2015. [Online]. Available: https://msdn.microsoft.com/en-us/data/jj574232.aspx. [Geopend 24 februari 2016].

[19] M. Durrant, „EF Code First with the Repository and Unit of Work Patterns," 21 juni 2012. [Online]. Available: http://www.mattdurrant.com/ef-code-first-with-the-repository-and-unit-of-work-patterns/. [Geopend 17 februari 2016].

[20] MSDN Microsoft, „The Repository Pattern," 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/ff649690.aspx. [Geopend 11 februari 2016].

[21] M. Fowler, „Data Mapper," [Online]. Available: http://martinfowler.com/eaaCatalog/dataMapper.html. [Geopend 11 februari 2016].

[22] J. Bogard, „Getting Started Guide," 28 januari 2016. [Online]. Available: https://github.com/AutoMapper/AutoMapper/wiki/Getting-started. [Geopend 18 februari 2016].

[23] D. Nene, „A beginners guide to Dependency Injection," 1 juli 2005. [Online]. Available: http://www.theserverside.com/news/1321158/A-beginners-guide-to-Dependency-Injection. [Geopend 11 februari 2016].

[24] J. G. Nair, „Autofac An Open-Source Dependency Injection (DI)," 13 juli 2015. [Online]. Available: http://blog.logiticks.com/autofac-an-open-source-dependency-injection-di/. [Geopend 7 april 2016].

[25] Autofac Contributors, „Getting Started," 2014. [Online]. Available: http://autofac.readthedocs.org/en/latest/getting-started/index.html. [Geopend 11

februari 2016].

[26] Autofac Contributors, „Modules," 2014. [Online]. Available: http://docs.autofac.org/en/latest/configuration/modules.html. [Geopend 22 februari 2016].

[27] A. Nemes, „OWIN and Katana part 1: the basics," 14 april 2014. [Online]. Available: http://dotnetcodr.com/2014/04/14/owin-and-katana-part-1-the-basics/. [Geopend 8 februari 2016].

[28] A. Abel, „What's this Owin Stuff About," 20 mei 2014. [Online]. Available: https://coding.abel.nu/2014/05/whats-this-owin-stuff-about/. [Geopend 23 februari 2016].

[29] J. Corioland, „Using OWIN tot test yout web api controllers," 1 april 2014. [Online]. Available: http://www.juliencorioland.net/archives/using-owin-to-test-your-web-api-controllers#.VyYCJaOLTxs. [Geopend 1 mei 2016].

[30] J. Atten, „ASP.NET Web Api 2.2: Create a Self-Hosted OWIN-Based Web Api from scratch," 11 januari 2015. [Online]. Available: http://johnatten.com/2015/01/11/asp-net-web-api-2-2-create-a-self-hosted-owin-based-web-api-from-scratch/. [Geopend 1 maart 2016].

[31] B. Hall, „Introduction to XUnit," 6 januari 2008. [Online]. Available: http://blog.benhall.me.uk/2008/01/introduction-to-xunit/. [Geopend 11 februari 2016].

[32] P. Thiagarajan, „Unit testing OWIN applications using TestServer," 26 november 2013. [Online]. Available: https://blogs.msdn.microsoft.com/webdev/2013/11/26/unit-testing-owin-applications-using-testserver/. [Geopend 2 mei 2016].

[33] D. Cazzulino, „Moq 4," Github, 11 februari 2016. [Online]. Available: https://github.com/moq/moq4. [Geopend 2 maart 2016].

[34] msdn Microsoft, „Data-tier Applications," 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/ee210546.aspx. [Geopend 10 maart 2016].

[35] Msdn Microsoft, „Extract, Publish, and Register .dacpac Files," 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj860455(v=vs.103).aspx. [Geopend 10 maart 2016].

[36] J. Chase, „AngularJS with Web API: server side paging," 24 juni 2015. [Online]. Available: https://code.msdn.microsoft.com/AngularJS-with-Web-API-43e5de16. [Geopend 8 maart 2016].

[37] J. Skinner, „FluentValidation," 21 Augustus 2015. [Online]. Available: https://github.com/JeremySkinner/FluentValidation/blob/master/README.md. [Geopend 15 februari 2016].

[38] K. Dockx, „JSON patch (JsonPatchDocument) RFC 6902 implementation for .NET,” 26 april 2016. [Online]. Available: https://github.com/KevinDockx/JsonPatch. [Geopend 4 april 2016].

[39] Trethaller, „How to link exceptions to requests in Application Insights on Azure?,” 12 juni 2015. [Online]. Available: http://stackoverflow.com/questions/30485385/how-to-link-exceptions-to-requests-in-application-insights-on-azure. [Geopend 2 mei 2016].

Bijlagen

B-1 Entiteiten v1.0

B-2 Entiteiten v2.0

B-3 Entiteiten v3.2

B-4 Entiteiten v4.0

B-1 Entiteiten v1.0

Binnen deze eerste versie van de entiteiten werd vertrokken van een algemene omschrijving en de oude database. Deze velden werden samengevoegd met onderstaande entiteiten tot gevolg.

COURSE

string Code
string title
int duration
double? pricePerPerson
Boolean active
Boolean new
Boolean onWebsite
Language language
Instructor responsible
List<Instructor> instructors
Boolean ready
List<Remark> remarks
List<Description> descriptions
List<Material> materials
BusinessOwner businessowner
Audience audience
Platform platform
List<Category> categories

DESCRIPTION

Int ID
Language language
Audience audience
string? prerequisites
string goal
string method
string description
string? URL
List<Topic> topics

TOPIC

Int ID
String name
Int? parentTopic_ID

Platform {Java, Microsoft, generic,…}

Enum Language {en, fr, nl}

Audience {eindgebruiker, developer, IT pro, …}

BUSINESS OWNER

Static string code
String name

MATERIAL

Int ID
String name

CATEGORY

Int ID
String name
Int parentCategory_ID

SESSION

String Code
string title
Type sessionType
Language language
Instructor instructor
Address location
List<Remark> remarks
Food food
Status status
List<Evaluation> evaluations
Double costs
List<SessionDay>
sessionDays

ADDRESS

Int ID
String name
String address
String country
Int zipcode

REMARK

Int ID
String text

ATTACHMENT

Int ID
String name
String? URL

ACOUNTING

Int ID
Int projectnumber
BusinessOwner
businessowner
Address address

CUSTOMER

Int ID
String name
List<Person> students
Accounting accounting
List<Address> address
Person contact
List<Remark>? Remarks
Boolean intern

Type {Open Kalender, Intra-Company}

Enum Language {en, fr, nl}

Enum Food {koud, warm, geen}

Enum Status {voorlopig, bevestigd, geannuleerd}

EVALUATION

Int ID
String question
Int order
List<Option> options
String? Comment

OPTION

Int ID
String description
Int order
Bool checked

SESSIONDAY

Int ID
Datetime start
Datetime end
Instructor instructor

PERSON Int ID String firstname String lastname String email	
INSTRUCTOR Int ID List<Language> language Boolean gender Person person Organization organization Boolean active	**ORGANIZATION** int ID String name

B-2 Entiteiten v2.0

Binnen deze tweede versie werd de inhoud van de verschillende entiteiten herzien alsook een herverdeling naar Aggregate Roots toe.

COURSE
code
title
duration
price
active
new
onWebsite
language
Instructor responsible
List<Instructor> instructors
ready
List<Remark> remarks
List<Description> descriptions
List<Material> materials
Businessowner businessowner
Audience audience
Platform platform
List<Category> categories

DESCRIPTION
ID
Language language
Audience audience
prerequisites
goals
methods
shortDescription
longDescription
webDescription
content
URL
List<Topic> topics

TOPIC
ID
name
Topic parentTopic

PLATFORM
ID
name

LANGUAGE {en,fr,nl}

TARGETAUDIENCE
ID
name

BUSINESSOWNER
code
name

MATERIAL
ID
info
available
MaterialCategory category

MATERIALCATEGORY
ID
name

COURSECATEGORY
ID
name
CourseCategory parentCategory

SESSION
code
title
sessionType
Language language
Instructor instructor
Address location
List<Remark> remarks
Food food
Status status
List<Evaluation> evaluations
costs
List<SessionDay> sessionDays
List<Client> customers

ADDRESS
ID
name
street
city
zipcode
country

REMARK
ID
text

ATTACHMENT
ID
name
URL

LANGUAGE {en, fr, nl}

FOOD {warm, broodjes, geen}

STATUS {voorlopig, bevestigd, geannuleerd}

SESSIONDAY
ID
start
end
Instructor instructor

EVALUATION

ID
Scorekey key
Questionoption? option
comment?
score?

EVALUATIONQUESTION

ID
Scorekey key
question
EvaluationCategory category
Language language
order
Evaluationtype type
List<Questionoption>? Options

QUESTIONOPTION

ID
description
Language language
order

EVALUATIONTYPE {options, standard, comment}

EVALUATIONCATEGORY {comment, course, trainer, organization, overall}

SCOREKEY

ID
name

CLIENTCOMPANY

ID
name
Person responsible
List<Address> addresses
List<Remark> remarks
isIntern
List<Accounting> accountings

ACCOUNTING

ID
projectnumber
Businessowner businessowner
Address address

CLIENT

ID
Clientcompany company
List<Person> students

PERSON

ID
firstname
name
email
telephone

INSTRUCTOR

ID
Person person
List<Language> languages
gender
active
Organization organization

ORGANIZATION

ID
name
Address address

B-3 Entiteiten v3.2

Met deze derde versie werd uiteindelijk vertrokken om de applicatie op te bouwen. Hierin liggen de finale Aggregate Roots vast en zijn alle enumeraties ook vervangen door tabellen.

COURSE

code
title
duration
price
active
new
onWebsite
ready
Person responsible
List<Person> instructors
List<Remark> remarks
List<Description> descriptions
List<Material> materials
Businessowner businessowner
CourseCategory categories

DESCRIPTION

ID
Language language
audience
prerequisites
goals
methods
shortDescription
longDescription
URL
List<Topic> topics
isExternal

TOPIC

ID
name
Topic? parentTopic

LANGUAGE

ID
name

BUSINESSOWNER

code
name
projectnumber
article

MATERIAL

ID
info
MaterialCategory category

MATERIALCATEGORY

ID
name

COURSECATEGORY

ID
displayName
functionalName
CourseCategory? parentCategory

SESSION
code
title
SessionType type
Language language
Address location
List<Remark> remarks
Food food
costs
Status status
List<SessionDay> sessionDays
Company organizer
List<Attachment> attachments
Audience audience
evaluationStudentGood
evaluationStudentBad
evaluationStudentNeutral
evaluationInstructor
evaluationEvaluator
evaluationSynopticScore

ADDRESS
ID
name
street
city
zipcode
country
isPrimary

REMARK
ID
text

ATTACHMENT
ID
name
URL

LANGUAGE
ID
name

FOOD
ID
name

STATUS
ID
name

SESSIONDAY
ID
start
end
Person instructor

SESSIONTYPE
ID
name

AUDIENCE
ID
participantsKnown
participantsTotal
List<Person> participants
Company company
projectnumber
invoiceID
invoiceAmount

EVALUATIONQUESTION

ID
feedbackKey
category
presentationorder
Language language
content
type
List<Evaluationquestionoption> options

EVALUATIONQUESTIONOPTION

ID
presentationorder
content

SCOREDFEEDBACK

ID
Session session
content
InDepth
ConceptsAndPractice
WellPlaced
InstructorKnowledge
InstructorPresentation
InstructorInteraction
WellStructured
Classroom
Organization
CoureMaterials
GoalAttained
Global

COMPANY	**PERSON**
ID	ID
name	firstname
Person contact	name
List<Address> addresses	email
List<Remark> remarks	telephone
isIntern	Company company
List<Person> employees	isInstructor
BTWnumber	List<Language> languages
	gender
	isActive

B-4 Entiteiten v4.0

Deze entiteiten zijn de finale versie. De verschillende wijzigingen die hierin aangebracht zijn waren als gevolg van ontwikkelingen in het programma of noden vanuit de business kant.

COURSE
code
title
duration
price
active
new
onWebsite
ready
Person responsible
List<Person> instructors
List<Remark> remarks
List<Description> descriptions
List<Material> materials
Businessowner businessowner
List<CourseCategory> categories

DESCRIPTION
ID
Language language
audience
prerequisites
goals
methods
shortDescription
longDescription
URL
List<Topic> topics
isExternal

TOPIC
ID
name
Topic? parentTopic

LANGUAGE
ID
name

BUSINESSOWNER
ID
code
name
projectnumber
article

MATERIAL
ID
info
MaterialCategory category

MATERIALCATEGORY
ID
name

COURSECATEGORY
ID
displayName
functionalName
CourseCategory? parentCategory
displayOrder

SESSION
year
sequencenumber
title
SessionType type
Language language
Address location
Address invoiceAddress
List<Remark> remarks
Food food
Status status
List<SessionDay> sessionDays
Company organizer
List<Attachment> attachments
Audience audience
Course course

ADDRESS
ID
name
street
city
zipcode
country
isPrimary

REMARK
ID
contents

ATTACHMENT
ID
name
URL

LANGUAGE
ID
name

FOOD
ID
name

STATUS
ID
name

SESSIONDAY
ID
start
end
Person instructor

SESSIONTYPE
ID
name

AUDIENCE
ID
participantsKnown
participantsTotal
List<Person> participants
Company company
projectnumber
invoiceID
invoiceAmount

COMPANY

ID
name
List<Address> addresses
List<Remark> remarks
isIntern
List<Person> employees
BTWnumber

PERSON

ID
firstname
name
jobTitle
email
telephone
Company company
isInstructor
isContact
List<Language> languages
gender

EVALUATIONQUESTION

ID
feedbackKey
category
presentationorder
Language language
content
type
List<Evaluationquestionoption> options

EVALUATIONQUESTIONOPTION

ID
presentationorder
content

SCOREDFEEDBACK

ID
Session session
content
InDepth
ConceptsAndPractice
WellPlaced
InstructorKnowledge
InstructorPresentation
InstructorInteraction
WellStructured
Classroom
Organization
CoureMaterials
GoalAttained
Global