Afstudeerverslag

Het documenteren van RESTful Web APIs

****  **Auteur:** E. Verboekend (10065466)  
**Opleiding:** Informatica aan De Haagse Hogeschool  
**Afstudeerperiode:** mei 2014 – september 2014  
**Examinatoren:** E.M. van Doorn  
 A.M.J.J. Lousberg  
  
**Bedrijf:** ENOVATION   
**Plaats:** Capelle a/d IJssel  
**Bedrijfsmentor:** A. van ‘t Hart  
 **Datum:** 6 oktober 2014 **Versie:** 1.0

Referaat

Afstudeerverslag, Erwin Verboekend, Het documenteren van RESTful Web APIs, ENOVATION, De Haagse Hogeschool, 6 oktober 2014.

Dit afstudeerverslag beschrijft het proces rondom de afstudeeropdracht, Het documenteren van RESTful Web APIs, die gedurende de periode van mei 2014 t/m september 2014 in opdracht van het bedrijf ENOVATION is uitgevoerd. De opdracht is gerelateerd aan de opleiding Informatica aan De Haagse Hogeschool.

Descriptoren:

* Documentatie
* RESTful Web API
* Test Driven Development
* Spring Framework
* Swagger
* API Blueprint
* RAML
* Proof of concept

Voorwoord

Voor u ligt het afstudeerverslag dat ik gedurende mijn afstudeerperiode bij het bedrijf ENOVATION heb opgesteld.   
In de periode van mei 2014 tot en met september 2014 heb ik binnen het bedrijf ENOVATION een opdracht uitgevoerd, betreffende een onderzoek naar het documenteren van RESTful Web APIs en het ontwikkelen van een proof of concept.

Ik zou van tevoren graag een aantal mensen willen bedanken die de realisatie van deze opdracht mogelijk hebben gemaakt. Allereerst ben ik ENOVATION dankbaar voor het beschikbaar stellen van de afstudeeropdracht en wil ik Albert van ’t Hart bedanken voor de begeleiding die hij mij gedurende mijn afstudeerperiode heeft gegeven. Verder wil ik ook graag alle collega’s van ENOVATION hartelijk bedanken voor de gezelligheid en de manier waarop zij me binnen het bedrijf op mijn gemak lieten voelen.

Daarnaast wil ik de heer E.M. van Doorn en mevrouw A.M.J.J. Lousberg bedanken voor de begeleiding die zij mij vanuit De Haagse Hogeschool hebben gegeven.

Ik wens u heel veel plezier bij het lezen van mijn afstudeerverslag!

Erwin Verboekend  
Capelle a/d IJssel, 6 oktober 2014.

Inhoudsopgave

[1 Inleiding 1](#_Toc400308453)

[2 Over het bedrijf 2](#_Toc400308454)

[2.1 Algemeen 2](#_Toc400308455)

[2.2 Organogram 2](#_Toc400308456)

[2.3 Werkwijze 2](#_Toc400308457)

[3 Theoretisch kader 3](#_Toc400308458)

[3.1 Application Programming Interface 3](#_Toc400308459)

[3.2 Representational State Transfer 3](#_Toc400308460)

[4 Opdrachtomschrijving 7](#_Toc400308467)

[4.1 Aanleiding 7](#_Toc400308468)

[4.2 Probleemstelling 7](#_Toc400308469)

[4.3 Doelstelling 7](#_Toc400308470)

[4.4 Vraagstelling 7](#_Toc400308471)

[4.5 Risicofactoren 8](#_Toc400308472)

[5 Plan van aanpak 9](#_Toc400308473)

[5.1 Onderzoeksmethoden 9](#_Toc400308474)

[5.2 Technieken 9](#_Toc400308475)

[5.3 Aanpak 10](#_Toc400308476)

[5.4 Planning 10](#_Toc400308480)

[5.5 Op te leveren producten 11](#_Toc400308481)

[6 Het interview 12](#_Toc400308482)

[6.1 Voorbereiding 12](#_Toc400308483)

[6.2 Het proces van ENOVATION 12](#_Toc400308484)

[6.3 De technieken van ENOVATION 12](#_Toc400308485)

[6.4 De eisen van ENOVATION 13](#_Toc400308486)

[7 Afbakening en planning 15](#_Toc400308487)

[8 Het literatuuronderzoek 16](#_Toc400308490)

[8.1 De selectie van mogelijkheden 16](#_Toc400308491)

[8.2 Swagger 18](#_Toc400308492)

[8.3 API Blueprint 19](#_Toc400308493)

[8.4 RAML 21](#_Toc400308494)

[8.5 Conclusie 23](#_Toc400308495)

[9 Proof of concept 24](#_Toc400308496)

[9.1 Inleiding 24](#_Toc400308497)

[9.2 Voorbereiding 24](#_Toc400308498)

[9.2.1 Het opzetten van de ontwikkelomgeving 24](#_Toc400308499)

[9.2.2 Aanpak 26](#_Toc400308500)

[9.3 Ontwerp 28](#_Toc400308501)

[9.3.1 Het ontwerpen van het proof of concept 28](#_Toc400308502)

[9.3.2 Het ontwerpen van de RAML definitie 33](#_Toc400308503)

[9.4 Testen 34](#_Toc400308504)

[9.4.1 Het opzetten van de testomgeving 34](#_Toc400308505)

[9.4.2 Het opstellen van de testcases 35](#_Toc400308506)

[9.5 Bouwen 40](#_Toc400308507)

[9.5.1 Het bouwen van het proof of concept 40](#_Toc400308508)

[9.5.2 Het implementeren van Swagger 46](#_Toc400308509)

[9.5.3 Het genereren van de RAML documentatie 48](#_Toc400308510)

[9.5.4 Het genereren van de Swagger documentatie 50](#_Toc400308511)

[10 Advies 52](#_Toc400308512)

[11 Evaluatie 54](#_Toc400308513)

[11.1 Productevaluatie 54](#_Toc400308514)

[11.2 Procesevaluatie 54](#_Toc400308515)

[11.3 Verantwoording competenties 55](#_Toc400308516)

[Bijlage A: Literatuurlijst 57](#_Toc400308522)

[Bijlage B: Afkortingenlijst 58](#_Toc400308523)

[Bijlage C: Het interview 59](#_Toc400308524)

# 1 Inleiding

Dit afstudeerverslag beschrijft het proces dat ik gedurende mijn afstudeerperiode bij het bedrijf ENOVATION heb doorlopen. Alle keuzes die ik gemaakt heb, zullen in dit verslag worden beschreven en toegelicht. Op basis hiervan kunnen de examinatoren bepalen of ik de van tevoren opgestelde competenties op het juiste niveau beheers.

In hoofdstuk 2 wordt er wat achtergrondinformatie gegeven over het bedrijf ENOVATION. Hierdoor zal duidelijk worden wat het bedrijf zoal doet en wat mijn positie binnen het bedrijf gedurende mijn afstudeerperiode is.

In hoofdstuk 3 wordt er aandacht bestaat aan de theorie rondom de opdracht. Om bekend te worden met de verschillende termen, zullen deze uitgebreid worden besproken.

Hoofdstuk 4 is gewijd aan het bespreken van de opdracht. In dit hoofdstuk worden onder andere de aanleiding, probleemstelling en doelstelling van de opdracht besproken.

Vervolgens wordt er in hoofdstuk 5 beschreven hoe de opdracht uitgevoerd ging worden. Hierbij komen de verschillende methoden en technieken aan bod en zal er een planning worden gegeven van de periode waarin de opdracht is uitgevoerd.

Hoofdstuk 6 beschrijft de werkzaamheden die gedurende het begin van de afstudeerperiode zijn uitgevoerd.

Als vervolg op hoofstuk 6 worden in hoofdstuk 7 de afbakening en de volledige nieuwe planning van de opdracht besproken.

Hoofdstuk 8 en 9 beschrijven de werkzaamheden die na de afbakening zijn uitgevoerd. De hoofdstukken zijn gewijd aan het literatuuronderzoek en het ontwikkelen van het proof of concept.

In hoofdstuk 10 wordt beschreven wat het advies over de verschillende mogelijkheden voor het documenteren van een RESTful Web API is.

# 2 Over het bedrijf

International

## 2.1 Algemeen

ENOVATION is een internationaal ICT-bedrijf dat in 1983 is opgericht en sinds 2013 onderdeel uitmaakt van de VANAD Group. Met ruim 30 jaar ervaring op het gebied van IT-innovatie streeft ENOVATION naar het effectief verbinden van mensen, systemen en organisaties in de zorg, transport & logistiek en zakelijke dienstverlening. Ze focussen zich voornamelijk op het leveren van diensten op het gebied van communicatie en data-integratie. ENOVATION heeft momenteel meer dan 180 medewerkers verspreid over Nederland, het Verenigd Koninkrijk en China. Het hoofdkantoor van het bedrijf is gevestigd op de campus in Capelle aan den IJssel.

De oplossingen die ENOVATION levert, worden voornamelijk aangeboden als SaaS-diensten. SaaS staat voor Software as a Service en betekent dat de software niet bij de klant geïnstalleerd wordt, maar bij de leverancier van de software. Hierdoor is de leverancier verantwoordelijk voor de installatie, het onderhoud en beheer en kan de klant de software over het internet benaderen.

## 2.2 Organogram

Op de onderstaande afbeelding is een schematisch overzicht van de bedrijfsstructuur van ENOVATION te zien.

ENOVATION

Development

Product Marketing

Professional Services

Product Development

Technical Support

Sales

International

Gedurende mijn afstudeerperiode was ik werkzaam op de afdeling Development.

## 2.3 Werkwijze

ENOVATION ontwikkelt zijn producten met behulp van de methode Scrum. Softwareontwikkeling met Scrum wordt vaak gezien als een methodiek, echter kan het beter gezien worden als een manier voor het managen van een complex project. Tijdens Scrum leveren teams in korte sprints, die een vaste lengte hebben van 1 tot 4 weken, werkende producten op. Tijdens het ontwikkelen van de producten ligt de focus op snelle communicatie, teamwork en grote betrokkenheid van de teamleden. Door het gebruik van korte sprints is Scrum zeer geschikt voor projecten waarin de wensen en eisen op het laatste moment nog kunnen wijzigen.

# 3 Theoretisch kader

In dit hoofdstuk wordt er aandacht besteed aan de theorie rondom de opdracht. De termen die tijdens de opdracht aan bod komen, worden hieronder besproken om bekend te worden met de verschillende termen.

## 3.1 Application Programming Interface

Een Application Programming Interface (API) is een interface met een verzameling van regels en specificaties op basis waarvan een applicatie kan communiceren met een andere applicatie. Hierdoor kent de ene applicatie geen details van de implementatie, maar kan de functionaliteiten van de andere applicatie wel gebruiken. Een API kan worden gezien als een contract tussen de client en de server. Een API die over het internet kan worden benaderd, wordt aangeduid als Web API. Wanneer er met een Web API gecommuniceerd wordt, wordt er vaak gestructureerde data uitgewisseld door middel van een standaard formaat. Twee veel voorkomende formaten zijn Extensible Markup Language (XML) en JavaScript Object Notation (JSON).

Het is hedendaags steeds gebruikelijker dat organisaties beschikken over een Web API. Hierdoor is het voor derde partijen mogelijk om een applicatie te ontwikkelen die gebruik maakt van de functionaliteiten die de applicatie van de organisatie te bieden heeft. Daarnaast wordt het combineren van gegevens uit verschillende bronnen buitengewoon eenvoudig. Josh Walker, een analist van de organisatie Forrester Research in Cambridge, zei ooit:

“Building an application with no APIs is basically like building a house with no doors.”

*- Josh Walker –*

Wat Josh Walker hiermee wil zeggen, is dat een applicatie zonder API volledig ontoegankelijk is voor de buitenwereld. Tegenwoordig leven we juist in een wereld waarin we de meest uiteenlopende gegevens willen combineren en het liefst zo snel en simpel mogelijk.

## 3.2 Representational State Transfer

Representational State Transfer (REST) is een architectuurstijl die in 2000 voor het eerst is beschreven door Roy Thomas Fielding. Alhoewel de meningen erg verdeeld zijn over wat REST precies inhoudt, houden we gedurende deze opdracht de beschrijving aan zoals Fielding die in zijn doctoraatstudie *Architectural Styles and the Design of Network-based Software Architectures* (R. T. Fielding, 2000) gaf. Volgens de doctoraatstudie bestaat REST uit een set van criteria die toegepast worden op de elementen binnen een architectuur. De elementen die hiermee worden bedoeld, zijn:

|  |  |
| --- | --- |
| **Element** | **Omschrijving** |
| Componenten | Een component zorgt ervoor dat de data-elementen bruikbaar zijn voor zijn interface en andersom |
| Connectoren | Een connector is een element dat zorgt voor de communicatie tussen componenten |
| Data | Een data-element is een element met informatie die door middel van een connector naar een component kan worden gestuurd of van een component kan worden ontvangen |

De zes criteria die op deze drie elementen worden toegepast, zijn:

|  |  |
| --- | --- |
| * Client-Server * Stateless * Cacheable | * Uniform Interface * Layered System * Code-on-Demand |

### 3.2.1 Client-Server

Het criterium Client-Server heeft betrekking op de scheiding tussen de verantwoordelijkheden van de kant van de client en de kant van de server. Door het scheiden van de verantwoordelijkheden kunnen beide kanten zich onafhankelijk van elkaar ontwikkelen en wordt de schaalbaarheid van het systeem verbeterd.

### 3.2.2 Stateless

Het criterium Stateless geeft aan dat de communicatie tussen de client en server volledig stateless moet zijn. Dit houdt in dat de informatie, die nodig is voor de server om de request (aanvraag naar een server) te begrijpen, zich in de request moet bevinden. De server mag hierbij geen gebruik maken van informatie die aan de kant van de server is opgeslagen.

### 3.2.3 Cacheable

Het criterium Cacheable verplicht de ontwikkelaar om van alle responses (antwoord van een server) aan te geven of het cacheable of dan wel niet cacheable is. Wanneer een response cacheable is, betekent dit dat de response na de request aan de kant van de client wordt opgeslagen. Vervolgens kan de kant van de client er bij soortgelijke request voor kiezen om de opgeslagen data terug te geven zonder dat er opnieuw een request gestuurd hoeft te worden. Dit heeft echter als nadeel dat de opgeslagen data mogelijk kan verschillen van de data die zou worden teruggegeven op het moment dat de request wel verstuurd zou worden.

### 3.2.4 Uniform Interface

Het criterium Uniform Interface is de basis voor elk ontwerp van een REST service. Het criterium deelt de architectuur op in verschillende delen die onafhankelijk van elkaar ontwikkeld kunnen worden. Het criterium wordt gevormd door de volgende vier richtlijnen:

* Identification of resources
* Manipulation of resources through representations
* Self-descriptive messages
* Hypermedia as the engine of application state

Om de richtlijnen goed te begrijpen, is het van belang om te weten wat de begrippen *resource* en *representation* precies inhouden.

De volgende teksten komen uit het boek “*RESTful Web Services*” (L. Richardson & S. Ruby, 2007).

A resource is anything that’s important enough to be referenced as a thing in itself. If your users might “want to create a hypertext link to it, make or refute assertions about it, retrieve or cache a representation of it, include all or part of it by reference into another representation, annotate it, or perform other operations on it”, then you should make it a resource.

Usually, a resource is something that can be stored on a computer and represented as a stream of bits: a document, a row in a database, or the result of running an algorithm. A resource may be a physical object like an apple, or an abstract concept like courage, but (as we’ll see later) the representations of such resources are bound to be disappointing.

Here are some possible resources:

* Version 1.0.3 of the software release
* The latest version of the software release
* The first weblog entry for October 24, 2006
* A road map of Little Rock, Arkansas
* Some information about jellyfish
* A directory of resources pertaining to jellyfish
* The next prime number after 1024
* The next five prime numbers after 1024
* The sales numbers for Q42004
* The relationship between two acquaintances, Alice and Bob
* A list of the open bugs in the bug database

When you split your application into resources, you increase its surface area. Your users can construct an appropriate URI and enter your application right where they need to be. But the resources aren’t the data; they’re just the service designer’s idea of how to split up the data into “a list of open bugs” or “information about jellyfish.” A web server can’t send an idea; it has to send a series of bytes, in a specific file format, in a specific language. This is a representation of the resource. A resource is a source of representations, and a representation is just some data about the current state of a resource. Most resources are themselves items of data (like a list of bugs), so an obvious representation of a resource is the data itself.

#### Identification of resources

De eerste richtlijn geeft aan dat alle resources geïdentificeerd moeten worden door een Uniform Resource Identifier (URI). Een URI vormt het adres van de resource die achter de URI schuilgaat. Met behulp van specifieke protocollen is er via het netwerk interactie mogelijk met de resource. Informatie die niet geïdentificeerd wordt door een URI is geen resource.

#### Manipulation of resources through representations

De tweede richtlijn vertelt ons dat de kant van de client de resources niet direct kan benaderen. De resources kunnen alleen gemanipuleerd worden door het sturen van representations naar de server.

#### Self-descriptive messages

De derde richtlijn verplicht alle requests om self-descriptive te zijn. Dit komt erg overeen met het criterium Stateless en betekent dat alle requests alle informatie moeten bevatten om te begrijpen wat er met de request moet gebeuren.

#### Hypermedia as the engine of application state

De richtlijn Hypermedia as the Engine of Application State (HATEOAS) geeft aan dat de client op basis van de responses moet kunnen bepalen welke acties hij/zij kan ondernemen. Dit kan bijvoorbeeld als de representation van een resource de URIs bevat van andere resources.

### 3.2.5 Layered System

Het criterium Layered System verplicht een ontwikkelaar om de softwarearchitectuur op te delen in verschillende lagen. Hierbij is het van belang dat elk component niet verder kan kijken dan de directe laag waarmee hij probeert te communiceren. Door de communicatie te limiteren tot één laag wordt de complexiteit van de software beperkt.

### 3.2.6 Code-on-Demand

Code-on-Demand is het enige criterium in REST dat optioneel is en stelt de ontwikkelaar in staat om functionaliteiten uit te breiden door bijvoorbeeld JavaScript-scripts uit te voeren aan de kant van de client. Gezien het criterium optioneel is en verder niet ter sprake zal komen, laat ik het bij deze uitleg.

Wanneer een Web API aan de eerste vijf criteria voldoet, mag de Web API gekarakteriseerd worden als RESTful.

# 4 Opdrachtomschrijving

In dit hoofdstuk wordt de opdracht nader toegelicht aan de hand van de aanleiding, probleemstelling en het doel van het onderzoek.

## 4.1 Aanleiding

ENOVATION wil in de toekomst graag overstappen naar één Healthcare Platform waarin de alle bestaande en toekomstige healtcare diensten via RESTful Web APIs worden aangeboden. Door de diensten via RESTful Web APIs aan te bieden, is het voor andere leveranciers van software mogelijk om zelf een systeem te ontwikkelen dat gebruik maakt van de diensten van ENOVATION (in het verloop van dit verslag zullen zulke leveranciers aangeduid worden als klanten). Voorbeelden van zo’n systeem zijn bijvoorbeeld een elektronisch patiëntendossier (EPD), een huisarts informatiesysteem (HIS) of een elektronisch voorschrijfsysteem (EVS). Voor de klanten moet het eenvoudig en duidelijk zijn hoe de aangeboden RESTful Web APIs in hun systeem geïmplementeerd kunnen worden. Een goede documentatie van de RESTful Web APIs is daarom heel erg belangrijk. Wanneer de documentatie te wensen overlaat, zullen de klanten niet optimaal gebruik kunnen maken van de functionaliteiten die de oplossingen van ENOVATION te bieden hebben.

## 4.2 Probleemstelling

De documentatie die ENOVATION hedendaags van zijn diensten kan opleveren, bestaat vaak uit slechts enkele eenvoudige HTML-pagina’s die door de ontwikkelaars met de hand opgesteld zijn. Dit heeft nadelige gevolgen voor zowel de ontwikkelaars van ENOVATION als voor de klanten die gebruik willen maken van de diensten van ENOVATION. Allereerst is het voor ENOVATION lastig om zijn RESTful Web APIs op een consequente manier te documenteren en kost het veel tijd en moeite om die documentatie synchroon te houden met de code. Daarnaast is het voor de klanten niet eenvoudig om op basis van de met de hand opgestelde HTML-pagina’s te begrijpen hoe zij de RESTful Web API kunnen gebruiken.

## 4.3 Doelstelling

Het doel van deze opdracht is om ENOVATION te adviseren bij het kiezen van een oplossing voor het eenvoudig en consequent documenteren van zijn RESTful Web APIs, waardoor de documentatie makkelijker synchroon te houden is en veel tijd en kosten bespaard zullen worden. Het advies zal bestaan uit een adviesrapport waarin de resultaten van het onderzoek worden geëvalueerd en een proof of concept (POC) waarin één of meerdere oplossingen uitgewerkt worden om de potentie van de oplossing(en) te onderbouwen.

## 4.4 Vraagstelling

De hierboven beschreven problemen leiden tot één centrale hoofdvraag:

*Welke mogelijkheden voor het documenteren van een RESTful Web API sluiten het best aan op de manier waarop ENOVATION zijn RESTful Web APIs ontwikkelt en de door ENOVATION gestelde eisen?*

In de hoofdvraag wordt er gebruik gemaakt van het zinsdeel ‘*de manier waarop*’. In de hierboven genoemde hoofdvraag wordt hiermee het volgende bedoeld:

* Het proces dat ENOVATION hanteert tijdens het ontwikkelen van zijn RESTful Web APIs
* De technieken die ENOVATION toepast tijdens het ontwikkelen van zijn RESTful Web APIs

Uit ervaring was van tevoren bekend dat het documenteren van een RESTful Web API op verschillende stadia van het ontwikkeltraject kan gebeuren. Om uiteindelijk met een goed en volledig advies te komen, was het van belang dat er rekening werd gehouden met de huidige werkwijze van ENOVATION.

Om de hoofdvraag te kunnen beantwoorden, is de hoofdvraag onderverdeeld in vier deelvragen:

1. Hoe verloopt het proces van ENOVATION tijdens het ontwikkelen van zijn RESTful Web APIs?
2. Welke technieken past ENOVATION toe bij het ontwikkelen van zijn RESTful Web APIs?
3. Welke eisen stelt ENOVATION aan het documenteren van een RESTful Web API?
4. Welke mogelijkheden bestaan er voor het documenteren van een RESTful Web API?

## 4.5 Risicofactoren

Tijdens het uitvoeren van de opdracht kunnen omstandigheden een ongunstige invloed hebben op de uitvoering van de opdracht. Met behulp van de opdrachtomschrijving kunnen deze omstandigheden van tevoren worden erkend, waardoor de impact van de omstandigheden beperkt kan blijven. Hieronder worden de erkende omstandigheden en hun maatregelen besproken.

#### Onvoldoende kennis van de technieken

Wanneer er tijdens het interview technieken naar voren komen waar ik weinig tot geen ervaring mee heb, kan dit mogelijk leiden tot problemen tijdens het ontwikkelen van een proof of concept. Het zou bijvoorbeeld zo kunnen zijn dat ik een proof of concept moet ontwikkelen met een programmeertaal waar ik niet bekend mee ben. Wanneer dit voorval optreedt, moet ik op tijd besluiten om hulp te vragen aan mijn bedrijfsmentor of mijn collega’s op de Development afdeling.

#### Onvoldoende inzicht tijdens het beoordelen van de mogelijkheden

Wanneer het tijdens het beoordelen van de mogelijkheden voor het documenteren van een RESTful Web API blijkt dat ik onvoldoende inzicht heb om de waarde van de mogelijkheid te bepalen, moet ik dit gelijk bekend maken bij de opdrachtgever. In overleg met de opdrachtgever kan vervolgens bepaald worden of de mogelijkheden van belang zijn of niet.

# 5 Plan van aanpak

In dit hoofdstuk staat beschreven welke methoden en technieken ik heb toegepast en de werkwijze die ik heb aangehouden tijdens het uitvoeren van deze opdracht.

## 5.1 Onderzoeksmethoden

Tijdens het uitvoeren van het onderzoek is er gebruik gemaakt van twee verschillende onderzoeksmethoden. Deze onderzoeksmethoden worden hieronder beschreven.

#### Interview

Een interview is een methode om kennis te vergaren van één of meerdere personen. Het kent drie verschillende vormen, namelijk een ongestructureerd, halfgestructureerd en gestructureerd interview. Voor deze opdracht is er gekozen om een halfgestructureerd interview te houden met de stakeholders van ENOVATION, omdat een interview de mogelijkheid geeft om door te vragen over een bepaald onderwerp in tegenstelling tot bijvoorbeeld een enquête. Bij het houden van een halfgestructureerd interview ligt het onderwerp van tevoren vast, maar alle vragen en de volgorde waarin de vragen gesteld gaan worden niet. Hierdoor is er genoeg ruimte voor eigen inbreng tijdens het interview. Tijdens het interview zou duidelijk worden welke technieken ENOVATION gebruikt bij het ontwikkelen van zijn RESTful Web APIs en welk proces hierbij gehanteerd wordt.

#### Literatuuronderzoek

Om te achterhalen wat er al over het documenteren van een RESTful Web API bekend was, is er een literatuuronderzoek uitgevoerd. Tijdens een literatuuronderzoek wordt bestaande literatuur geanalyseerd en samengevat. Voor het literatuuronderzoek zijn verschillende bronnen gebruikt, waaronder boeken en de zoekmachine Google Scholar. Het literatuuronderzoek zou inzicht geven in welke oplossingen er zijn voor het documenteren van een RESTful Web API en welke van deze oplossingen mogelijk het best aansluiten op de omgeving van ENOVATION. Voor dit onderzoek is er expliciet gekozen voor een literatuuronderzoek, omdat de geschiktheid van een oplossing deels gebaseerd moest worden op de ervaringen van anderen. Gezien de ervaringen van anderen geen uitsluiting konden geven of de oplossingen geschikt zouden zijn voor de omgeving van ENOVATION zou een proof of concept meer duidelijkheid geven.

## 5.2 Technieken

#### 100 punten methode

De 100 punten methode is een methode die gebruikt kan worden bij het prioriteren van bijvoorbeeld eisen in een groepsomgeving. Tijdens het toepassen van de 100 punten methode krijgen alle personen die betrokken zijn bij het opstellen van de eisen 100 punten die zij over de eisen kunnen verdelen. Des te meer punten zij toekennen aan een eis, hoe meer waarde zij aan de eis hechten. Een andere methode voor het prioriteren van eisen is de MoSCoW-methode. Bij het toepassen van de MoSCoW-methode worden de eisen onderverdeeld in vier verschillende groepen die de mate van prioriteit aangeven, namelijk: must haves, should haves, could haves en won’t haves. Aangezien de 100 punten methode beter en makkelijker toe te passen is voor het prioriteren van de eisen van meerdere stakeholders en dit tijdens deze opdracht het geval is, heb ik ervoor gekozen om de 100 punten methode te gebruiken.

## 5.3 Aanpak

### 5.3.1 Het interview

De opdracht begint met een oriëntatiefase waarin het interview met ENOVATION opgesteld wordt. Met behulp van het resultaat van het interview konden de eerste drie deelvragen beantwoord worden, namelijk:

1. Hoe verloopt het proces van ENOVATION tijdens het ontwikkelen van zijn RESTful Web APIs?
2. Welke technieken past ENOVATION toe bij het ontwikkelen van zijn RESTful Web APIs?
3. Welke eisen stelt ENOVATION aan het documenteren van een RESTful Web API?

De antwoorden op de hierboven genoemde vragen zouden als input dienen voor het literatuuronderzoek.

### 5.3.2 Het literatuuronderzoek

Het literatuuronderzoek zal inzicht geven in welke oplossingen er bestaan voor het documenteren van een RESTful Web API. Omdat er hoogstwaarschijnlijk talloze oplossingen zijn, zullen alleen de oplossingen die aansluiten op de ontwikkelomgeving en de eisen van ENOVATION nader onderzocht worden. De selectie van oplossingen zal voornamelijk gebaseerd worden op de ervaringen van anderen. Dit geeft echter nog geen uitsluiting of de oplossingen ook daadwerkelijk geschikt zijn voor ENOVATION. Daarom worden er twee of meer oplossingen gekozen die toegepast worden op een proof of concept. Hoeveel oplossingen dit zullen zijn, hangt af van de resultaten van het literatuuronderzoek.

### 5.3.3 Het proof of concept

Om te bepalen of de mogelijkheden die tijdens het literatuuronderzoek naar voren zijn gekomen ook daadwerkelijk in staat zijn om het probleem dat in paragraaf 4.2 wordt beschreven voor ENOVATION op te lossen, zal een proof of concept worden ontwikkeld. Het proof of concept bestaat uit een RESTful Web API die in grote lijnen ontwikkeld wordt met behulp van de technieken en processen die ENOVATION hanteert.

## 5.4 Planning

In de onderstaande tabel is de initiële planning van de opdracht weergegeven.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Week** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| Oriëntatie | x | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Onderzoeksplan |  |  | x | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Interview |  |  |  | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Literatuuronderzoek |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Onderzoeksrapport |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Proof of concept |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Adviesrapport |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Afstudeerverslag |  |  |  |  |  |  |  |  |  | x | x | x | x | x | x | x | x | x | x | x |

Zoals in de tabel te zien is, is er voor enkele werkzaamheden nog geen planning gemaakt. Dat komt omdat de planning voor deze werkzaamheden pas gemaakt kon worden, nadat het interview was gehouden en er inzicht was verkregen in de omvang van de opdracht. In het verloop van de uitvoering van de opdracht is er een nieuwe en uitgebreidere planning gemaakt.

## 5.5 Op te leveren producten

De onderstaande producten worden opgeleverd:

* Onderzoeksplan
* Onderzoeksrapport
* Proof of concept
* Adviesrapport
* Afstudeerverslag

# 6 Het interview

In dit hoofdstuk worden de werkzaamheden rondom het interview en de resultaten van het interview beschreven.

6.1 Voorbereiding  
Voorafgaand aan het halfgestructureerde interview zijn er enkele vragen opgesteld waarmee, zoals in het plan van aanpak staat beschreven, inzicht verkregen zou worden in welke technieken ENOVATION gebruikt bij het ontwikkelen van zijn RESTful Web APIs en welk proces hierbij gehanteerd wordt. De vragen die van tevoren zijn opgesteld, zijn te vinden in de bijlagen.

Het interview is op 23 mei 2014 gehouden met Solution Architect Albert van ’t Hart en Software Architect Gluay Heiden. Beiden zijn werkzaam bij ENOVATION en hebben ervaring met het ontwikkelen van RESTful Web APIs.

## 6.2 Het proces van ENOVATION

Allereerst was het van belang dat de mogelijkheden voor het documenteren van een RESTful Web API zouden aansluiten op de manier waarop ENOVATION te werk gaat tijdens het ontwikkelen van zijn RESTful Web APIs. Tijdens het ontwikkelen van nieuwe software wordt er vaak onderscheid gemaakt tussen twee verschillende strategieën, namelijk top-down en bottom-up.

#### Top-down

Bij het toepassen van de top-down strategie wordt het te ontwikkelen systeem eerst opgebroken, waardoor inzicht wordt verkregen in alle onderdelen van het systeem. De onderdelen worden hierbij benoemd zonder in te gaan op de details. Vervolgens worden er van de onderdelen modellen opgesteld en wordt er steeds verder naar de technische details toegewerkt. Aan het eind worden alle onderdelen samengevoegd tot één product. Een belangrijk aspect van de top-down strategie is dat het coderen wordt uitgesteld totdat er een bepaald niveau van detail is bereikt.

#### Bottom-up

De strategie bottom-up richt zich voornamelijk op het coderen en testen in een vroeg stadium van een project. Dit betekent dat er gelijk begonnen kan worden met het coderen en testen zodra het eerste onderdeel van het systeem benoemd is. Een nadeel van de bottom-up strategie is dat er tijdens het coderen nog niet goed nagedacht is over hoe het onderdeel met de rest van het systeem gaat samenwerken. Hierdoor zou een project veel ingewikkelder kunnen uitvallen dan men van tevoren dacht, waardoor een hoop extra tijd en kosten verloren gaat.

Uit het interview is voortgekomen dat ENOVATION begint met het maken van een gedetailleerd ontwerp van de RESTful Web API voordat er daadwerkelijk met het coderen wordt begonnen. Hieruit kon ik concluderen dat ENOVATION de top-down strategie toepast.

## 6.3 De technieken van ENOVATION

Tijdens het ontwikkelen van een RESTful Web API maakt ENOVATION gebruik van verschillende technieken. Onder deze technieken worden de ontwikkeltools, frameworks en standaarden verstaan. Aangezien de mogelijkheden voor het documenteren van een RESTful Web API mogelijk ook gebruik maakten van verschillende tools en standaarden konden de technieken van ENOVATION bepalend zijn tijdens het maken van een goede selectie. De technieken die tijdens het interview naar voren zijn gekomen, staan in de tabel op de volgende bladzijde met een omschrijving weergegeven.

|  |  |
| --- | --- |
| **Techniek** | **Omschrijving** |
| Java | Java is een objectgeoriënteerde programmeertaal. |
| Spring Framework | Spring Framework (Spring) is een framework dat gericht is op de ontwikkeling van software in de programmeertaal Java |
| SoapUI | SoapUI is een applicatie voor het testen van web services met een SOA of REST architectuur |
| Apache Maven | Apache Maven is een tool voor Java-projectmanagement en geautomatiseerde softwarebouw. Door middel van een project object model (POM) wordt de volgorde van het bouwen aangegeven |
| Spring Tool Suite | Spring Tool Suite is een op het ontwikkelprogramma Eclipse gebaseerd ontwikkelprogramma dat op maat gemaakt is voor het ontwikkelen van Spring applicaties |
| Apache Subversion | Apache Subversion is een open source versiebeheersysteem |
| Git | Git is net als Apache Subversion een versiebeheersysteem (ENOVATION bevindt zich momenteel in een overstap van Subversion naar Git) |
| AngularJS | AngularJS is een open source web applicatie framework dat de ontwikkeling van single-page applicaties ondersteunt. Hiervoor wordt alleen gebruik gemaakt van HTML, CSS en JavaScript. Door het toepassen van het MVC-model wordt de ontwikkeling en het testen vergemakkelijkt |
| Twitter Bootstrap | Twitter Bootstrap is een front-end framework. Het framework bestaat onder andere uit op HTML en CSS gebaseerde design templates |
| Linux CentOS | CentOS is een linuxdistributie voor serversystemen en is gebaseerd op Red Hat Enterprise Linux (RHEL) |
| Jenkins CI | Jenkins Continuous Integration (Jenkins CI) is een applicatie die herhalende opdrachten bewaakt. Onder de herhalende opdrachten verstaan we bijvoorbeeld het bouwen van een software project |

Uit de tabel is op te maken dat ENOVATION zijn RESTful Web APIs bouwt met behulp van Apache Maven en vervolgens ontwikkelt in de programmeertaal Java. Hierbij wordt er gebruik gemaakt van het Spring Framework en een ontwikkelprogramma dat hier direct op aansluit, namelijk Spring Tool Suite. Het testen van de RESTful Web APIs gebeurt aan de hand van testcases die opgesteld worden in het programma SoapUI. Als het aankomt op graphical user interfaces (GUI), maakt ENOVATION gebruik van AngularJS en Twitter Bootstrap. Het literatuuronderzoek zal echter pas aantonen of deze twee technieken ook daadwerkelijk belangrijk zijn voor deze opdracht.

## 6.4 De eisen van ENOVATION

Naast het proces en de technieken waren ook de overige eisen die ENOVATION stelde belangrijk. De eisen dienden om erachter te komen wat ENOVATION belangrijk vond bij het documenteren van een RESTful Web API. De eisen die naar voren kwamen zijn zoals eerder is beschreven, geprioriteerd met behulp van de 100 punten methode. Ze zijn samen met een toelichting weergegeven in de tabel op de volgende bladzijde.

|  |  |  |
| --- | --- | --- |
| **Eis** | **Toelichting** | **Toegekende punten** |
| De documentatie moet dicht bij de API blijven | De documentatie van de RESTful Web APIs moet dicht bij de API blijven, zodat te allen tijde de juiste documentatie snel geraadpleegd kan worden | 20 |
| De documentatie moet minder bewerkelijk zijn | Het opstellen van de documentatie moet minder arbeidsintensief en tijdrovend zijn dan in de huidige situatie | 10 |
| De documentatie moet op een consequente manier kunnen worden opgesteld | Het opstellen van de documentatie moet volgens een vast plan verlopen. Hierdoor ontstaat er een standaard voor de documentatie van de RESTful Web API van ENOVATION | 10 |
| De documentatie moet rijk en duidelijk zijn | De documentatie wordt gezien als rijk en duidelijk wanneer:   * Alle HTTP responsecodes worden beschreven * Alle beschikbare parameters worden beschreven * Alle constraints worden beschreven * De documentatie code voorbeelden bevat * De RESTful Web API vanuit de documentatie te testen is | 30 |
| De documentatie moet synchroon zijn met de code | De documentatie van de RESTful Web APIs moet te allen tijde synchroon zijn met de code van de desbetreffende API | 30 |

De toegekende punten die in de tabel te zien zijn, zijn de punten die de stakeholders gezamenlijk hebben toegekend aan de eisen, omdat er tijdens het toekennen van de punten nauwelijks verschillen waren tussen de stakeholders.

# 7 Afbakening en planning

In dit hoofdstuk worden de afbakening en de volledige planning van de opdracht besproken.

## 7.1 Afbakening

Na de oriëntatiefase, het opstellen van het onderzoeksplan en het houden en uitwerken van het interview is gebleken dat het literatuuronderzoek en het uitwerken van het proof of concept meer tijd in beslag zou nemen dan ik van tevoren had gedacht. Dit kwam omdat er tijdens het oriënteren al was gebleken dat er een kleine selectie van mogelijkheden was die erg populair waren. Het was dus noodzakelijk dat deze mogelijkheden nog dieper onderzocht moesten worden. Daarnaast heb ik op aanraden van mijn bedrijfsmentor meer tijd voor het realiseren van het proof of concept ingepland. De bedrijfsmentor raadde dit aan, omdat het ontwikkelen van een RESTful Web API meer inhield dan alleen het bouwen en hij vond dat ik hier voldoende tijd voor moest inplannen.

De opdracht bestond in eerste instantie uit twee verschillende onderdelen, namelijk een onderdeel over het documenteren van een RESTful Web API en een onderdeel over het in kaart brengen van het gebruik van een RESTful Web API. Door de hierboven beschreven problemen klopte de initiële planning totaal niet meer en was er veel minder tijd beschikbaar voor het tweede onderdeel. Een alternatief voor dit probleem was het volledig afronden van het eerste onderdeel en het tweede onderdeel uitvoeren zover als ik kon. Echter heeft de opdrachtgever aangegeven dat het eerste onderdeel belangrijker was, omdat het resultaat van dit onderdeel in de toekomst van invloed kon zijn op het ontwikkelproces van ENOVATION. De opdrachtgever heeft daarom geadviseerd om me volledig te richten op het onderdeel over het documenteren van een RESTful Web API en het tweede onderdeel te laten vallen.

Toen er besloten was om het tweede onderdeel compleet te laten vallen, kon ik een nieuwe planning maken die volledig op het eerste onderdeel gericht was.

## 7.2 Planning

In de onderstaande tabel is de planning voor het eerste onderdeel weergegeven. Hierbij is er meer tijd ingepland voor het literatuuronderzoek en het ontwikkelen van het proof of concept. Aangezien ik in de eerste vier weken bezig ben geweest met oriënteren, het opstellen van een onderzoeksplan en het houden en uitvoeren van het interview, kon ik pas in week 5 beginnen met het literatuuronderzoek.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Week** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| Literatuuronderzoek |  |  |  |  | x | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Onderzoeksrapport |  |  |  |  |  |  | x | x |  |  |  |  |  |  |  |  |  |  |  |  |
| Proof of concept |  |  |  |  |  |  |  |  | x | x | x | x | x |  |  |  |  |  |  |  |
| Adviesrapport |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | x | x |  |  |
| Afstudeerverslag |  |  |  |  |  |  |  |  |  | x | x | x | x | x | x | x | x | x | x | x |

# 8 Het literatuuronderzoek

In dit hoofdstuk worden de resultaten van het uitgevoerde literatuuronderzoek besproken en zullen de mogelijkheden die op het proof of concept zullen worden toegepast naar voren komen.

## 8.1 De selectie van mogelijkheden

Om na te gaan welke mogelijkheden er bestaan voor het documenteren van een RESTful Web API is er een snelle scan uitgevoerd. Tijdens deze scan werd er met behulp van de Google Search Engine gezocht naar alle verschillende mogelijkheden. Gezien niet alle mogelijkheden in aanmerking konden komen, werden de mogelijkheden in eerste instantie op basis van de volgende criteria beoordeeld:

* De populariteit van de oplossing
* De mate waarin er nog gewerkt wordt aan het verbeteren van de mogelijkheid
* De eerste indruk
* De compatibiliteit met de technieken van ENOVATION

Tijdens het uitvoeren van de snelle scan is er gebruik gemaakt van de volgende zoektermen:

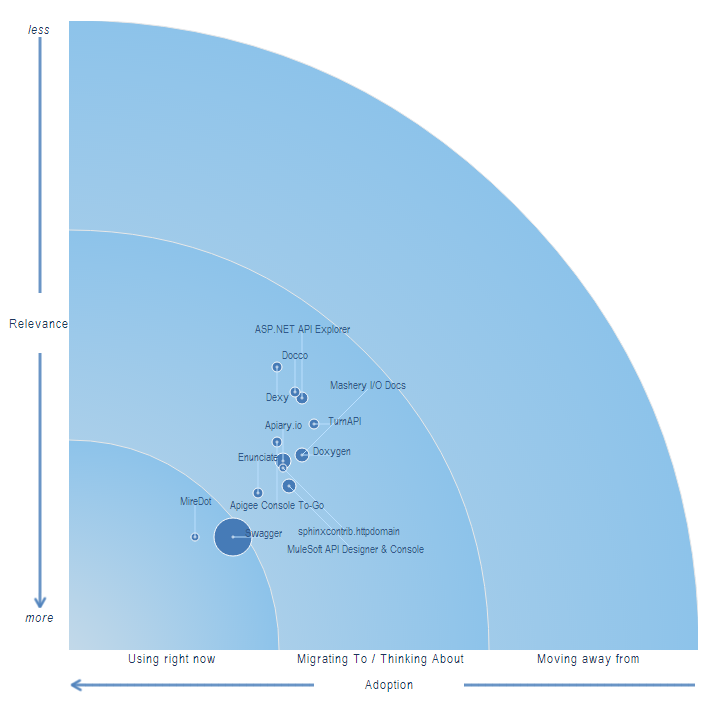
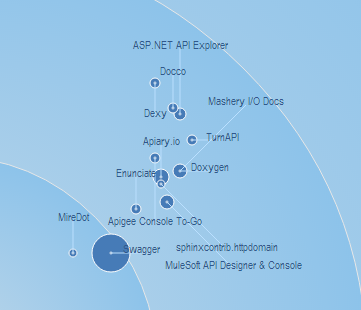
|  |
| --- |
| **Zoekterm** |
| RESTful Web API |
| RESTful API |
| API documentation |

Het eerste literatuurstuk dat naar voren kwam, was de blog *Don’t Make The Same Mistake Twice* (2012). In de blog wordt er onderzoek gedaan naar acht verschillende mogelijkheden voor het documenteren van een REST API die ontwikkeld is met behulp van het Spring Framework. De conclusie die in de blog wordt getrokken, is heel interessant voor dit literatuuronderzoek, want ENOVATION maakt ook gebruik van het Spring Framework. De mogelijkheden die in de blog aan bod komen, zijn:

* SpringDoclet
* RESTdoclet
* wsdoc
* Wadl
* Swagger
* I/O Docs
* enunciate
* API Blueprint

De twee mogelijkheden die volgens de blog uiteindelijk als beste naar voren kwamen, zijn Swagger en I/O Docs.

Op de webpagina *How do you document your APIs?* (2013) worden nog enkele andere mogelijkheden genoemd. Daarnaast is er ook een diagram te zien waarin ontwikkelaars kunnen aangeven welke mogelijkheid zij gebruiken. Dit diagram is te zien in figuur 2 (How do you document your APIs, 2013).

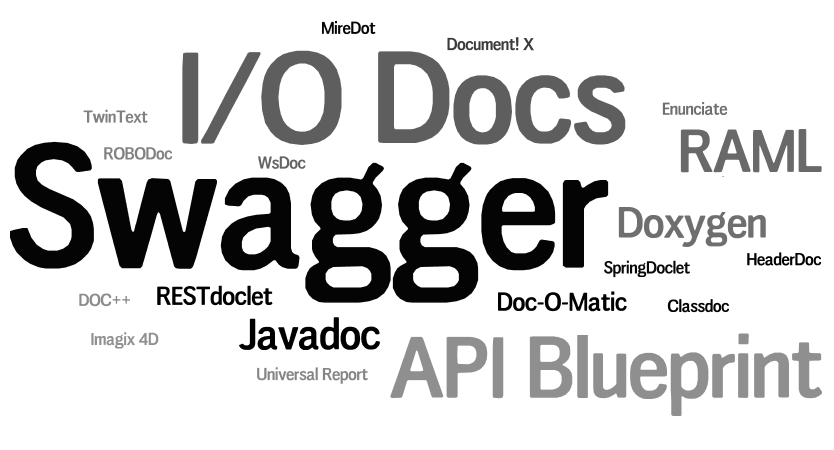


Figuur 1: welke mogelijkheden worden het meest gebruikt?

In het diagram staat de bruikbaarheid van de mogelijkheid tegenover het niveau van integratie afgebeeld. Hierbij kent het niveau van integratie drie verschillende waarden, namelijk *nu in gebruik*, *migreren naar de mogelijkheid* en *overstappen op een andere mogelijkheid.* De drie waarden zijn in dezelfde volgorde weergegeven op de X-as. Op de Y-as staat de bruikbaarheid met bovenin de waarde *weinig* en onderin de waarde *meer*. De grootte van de cirkels staat voor het aantal stemmen. Hoe groter de cirkels, des te meer stemmen voor die mogelijkheid.

Het diagram laat zien dat Swagger, I/O Docs, API Blueprint (Apiary.io) en RAML (MuleSoft API Designer & Console) de meeste stemmen hebben gekregen. Hoewel dit niet gelijk betekende dat dit ook de meest gebruikte mogelijkheden waren, ben ik er op basis van de verhoudingen met de andere mogelijkheden toch van uitgegaan.

Alle mogelijkheden die tijdens de scan zijn beoordeeld, zijn in de onderstaande word cloud weergegeven. De weergegeven grootte schaalt met de populariteit van de oplossing.



Zoals in de word cloud te zien is, zijn Swagger, I/O Docs, API Blueprint en RAML de populairste mogelijkheden van dit moment. Eveneens zijn dit de mogelijkheden die tijdens de snelle scan naar voor kwamen als de mogelijkheden met de rijkste en duidelijkste documentatie. De scan heeft echter ook aangetoond dat I/O Docs en Swagger qua gebruik en resultaat erg veel op elkaar lijken. Om ervoor te zorgen dat er genoeg aandacht wordt besteed aan verschillende mogelijkheden, is ervoor gekozen om slechts één van de twee mogelijkheden verder te onderzoeken. De keuze is vanwege de grotere community gevallen op Swagger. Dit betekende dat de mogelijkheden Swagger, API Blueprint en RAML nader onderzocht zouden worden.

## 8.2 Swagger

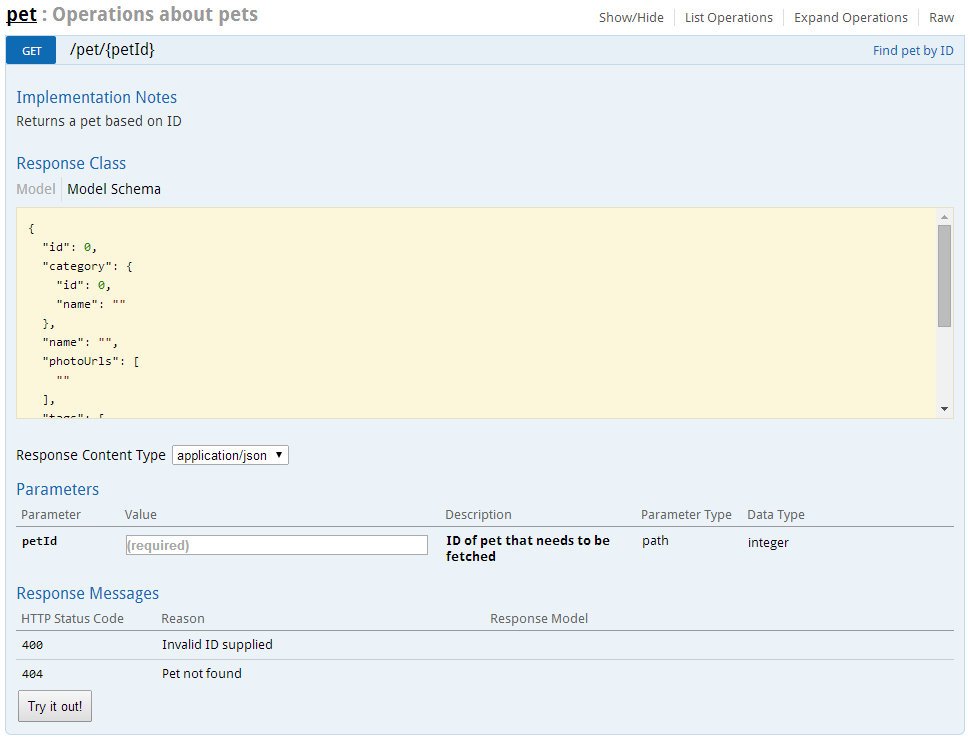
Swagger is “*a specification and complete framework implementation for describing, producing, consuming, and visualizing RESTful web services*” (Swagger: A simple, open standard for describing REST APIs with JSON, 2014). Enkele specificaties van Swagger zijn te zien in de onderstaande tabel.

|  |  |
| --- | --- |
| **Formaat** | JSON / XML |
| **Licentie** | ASL 2.0 |
| **Beschikbaar op** | GitHub |
| **Gesponsord door** | Reverb |
| **Laatste versie** | 1.2 |
| **Eerste commit** | Juli 2011 |
| **Design strategie** | Bottom-up |

Zoals in de tabel te zien is, maakt Swagger gebruik van de bottom-up strategie. Dit houdt in dat Swagger pas aan de orde komt wanneer de code is opgesteld. Bij het implementeren van Swagger wordt de code voorzien van annotaties. Deze annotaties worden gedurende het compilen omgezet naar een van tevoren gedefinieerd formaat, namelijk JSON of XML. Dit formaat wordt vervolgens gedistribueerd via een URL. Door het distribueren van het formaat is het voor iedereen mogelijk om te begrijpen hoe de RESTful Web API in elkaar zit. Om het formaat om te zetten naar een formaat dat voor een mens beter te begrijpen is, bestaat er een tool genaamd Swagger UI.

Swagger UI is onderdeel van het Swagger project en bestaat uit een verzameling van HTML, JavaScript en CSS assets. Swagger UI kan op basis van de URL, die Swagger gebruikt om het formaat te distribueren, een rijke en visuele documentatie genereren. Doordat het slechts uit enkele HTML, JavaScript en CSS assets bestaat, is het mogelijk om Swagger UI op elke server of lokale machine te laten draaien.

Een voorbeeld van een documentatie die door Swagger UI gegenereerd is, is te zien in het onderstaande figuur.



Figuur 2: voorbeeld van een gegenereerde documentatie door Swagger UI

De elementen die voor alle requests gegenereerd worden, zijn:

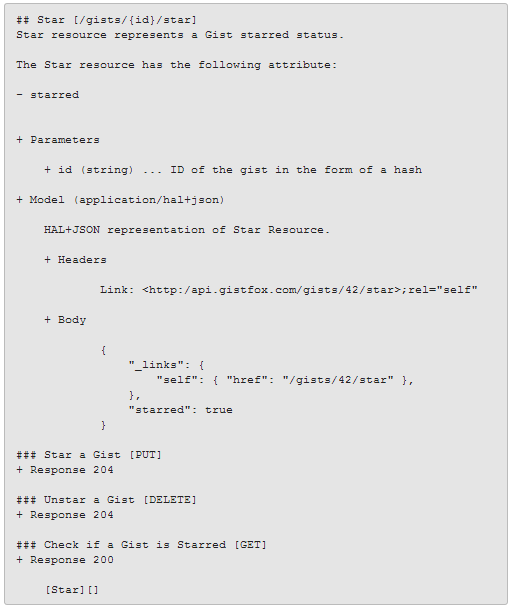
* Een beschrijving van wat de request doet
* De default response body (zowel Model als Schema)
* Alle parameters die verplicht of optioneel worden meegegeven
* Alle responses die afwijken van het default (inclusief een reden en de model of schema)
* De mogelijkheid om de request eenvoudig vanuit de documentatie te testen

## 8.3 API Blueprint

API Blueprint is een op Markdown-gebaseerde taal voor het beschrijven van een API. Markdown is een platte tekst syntax die eenvoudig omgezet kan worden naar bijvoorbeeld HTML. Doordat de blueprints uit platte tekst bestaan, kunnen ze makkelijk gelezen en begrepen worden. Het uitwisselen van de blueprints onder de ontwikkelaars levert dus geen enkel probleem op. Enkele specificaties van API Blueprint staan in de onderstaande tabel weergegeven.

|  |  |
| --- | --- |
| **Formaat** | Markdown |
| **Licentie** | MIT |
| **Beschikbaar op** | GitHub |
| **Gesponsord door** | Apiary |
| **Laatste versie** | 1A3 |
| **Eerste commit** | April 2013 |
| **Design strategie** | Top-down |

Het ontwikkelen van een RESTful Web API met behulp van API Blueprint begint met het opstellen van een blueprint (top-down). Een voorbeeld van hoe zo’n blueprint eruit kan zien, is te zien op het onderstaande figuur.



Figuur 3: voorbeeld van een API Blueprint

Na het opstellen van een blueprint zorgen verschillende tools ervoor dat er vloeiend naar een andere ontwikkelfase kan worden overgegaan. Enkele tools die hierbij kunnen helpen, worden hieronder kort toegelicht.

#### Dredd

Dredd is een tool die kan worden gebruikt om de implementatie van een API op basis van de blueprint te testen. Met behulp van Dredd is het mogelijk om de blueprint eenvoudig te linken aan een CI systeem, zoals Travis CI of Jenkins. Hierdoor heb je te allen tijde een blueprint die up-to-date is.

#### Iglo

Iglo is een tool waarmee een blueprint omgezet kan worden naar HTML. Hierdoor krijg je een formaat die voor een mens makkelijk te begrijpen is.

#### Aglio

Aglio is net als iglo een tool om blueprints om te zetten naar HTML. Aglio is slechts wat uitgebreider met verschillende thema’s.

#### Contractor

Contractor is een tool waarmee een mock server voor een API kan worden opgezet op basis van de blueprint. Contractor scant de routes en zorgt ervoor dat de juiste responses, die in de blueprint zijn gedefinieerd, worden teruggeven.

#### API Blueprint Mock Server

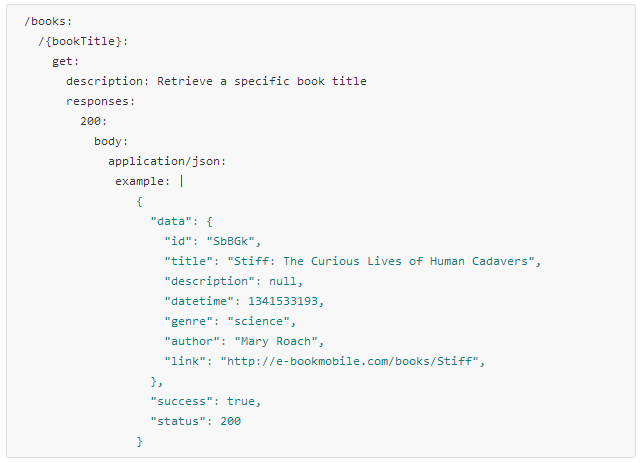
API Blueprint Mock Server is net als Contractor een tool waarmee een API getest kan worden met behulp van een mock server. API Blueprint Mock Server is gebaseerd op Java en kan op elke lokale machine gedraaid worden.

## 8.4 RAML

RAML staat voor RESTful API Modeling Language en is net als API Blueprint een manier om een API te beschrijven zodat het voor een mens te begrijpen is. RAML is gebaseerd op YAML en de legt de nadruk op het beschrijven resources, methods, parameters en responses. YAML staat voor YAML Ain’t Markup Language en is een voor een mens makkelijk te begrijpen standaard voor het serialiseren van data. De specificaties van RAML staan in de onderstaande tabel weergegeven.

|  |  |
| --- | --- |
| **Formaat** | YAML |
| **Licentie** | ASL 2.0 / TM |
| **Beschikbaar op** | GitHub |
| **Gesponsord door** | MuleSoft |
| **Laatste versie** | 0.8 |
| **Eerste commit** | September 2013 |
| **Design strategie** | Top-down |

Uit de hierboven genoemde specificaties blijkt dat RAML werkt met behulp van de top-down strategie. Het proces begint net zoals bij API Blueprint met het maken van een beschrijving van de RESTful Web API. In RAML wordt zo’n beschrijving ook wel een RAML definitie genoemd. Een RAML definitie bestaat vaak uit één bestand met de extensie *.raml*. Het is echter ook mogelijk om in een RAML definitie naar externe .raml bestanden te linken, waardoor de definitie uit meerdere bestanden bestaat. Hierdoor is het mogelijk om elementen uit de beschrijving her te gebruiken in definities van andere RESTful Web APIs. Een voorbeeld van hoe een simpele RAML definitie eruitziet, is te zien in het onderstaande figuur.



Figuur 4: voorbeeld van een RAML definitie

Hoewel RAML een vrij nieuwe mogelijkheid is voor het documenteren van een RESTful Web API, bestaan er al veel interessante open source tools die het ontwikkelproces van ENOVATION goed zouden kunnen ondersteunen. De tools die interessant zijn voor ENOVATION worden hieronder weergegeven.

#### API Designer

API Designer is een web-based API ontwikkel tool. Het bestaat uit een RAML editor met een geïntegreerde RAML Console (API Console).

#### API Console

De API Console betreft een grafisch user interface om de structuur en belangrijkste patronen van een door RAML gedefinieerde API visueel weer te geven. Het zorgt voor een interactieve documentatie van de API.

#### API Notebook

API Notebook is een op JavaScript gebaseerde web-based workspace, waarin APIs live getest en verkend kunnen worden. Het testen van de APIs gebeurt door middel van opgestelde use cases die vervolgens eenvoudig kunnen worden opgeslagen in Markdown formaat.

#### SoapUI RAML Plugin

De SoapUI RAML plugin maakt het mogelijk om een RAML definitie te importeren in SoapUI, waardoor het mogelijk is om de APIs te testen.

#### RAML to HTML

Met RAML to HTML kan een RAML definitie worden omgezet naar een enkele HTML pagina die de documentatie van de API bevat.

#### RAMLang

RAMLang is een command-line tool waarmee eenvoudig AngularJS services gecreëerd kunnen worden op basis van een RAML definitie.

## 8.5 Conclusie

Tijdens het literatuuronderzoek is er een selectie gemaakt van drie verschillende mogelijkheden voor het documenteren van een RESTful Web API. Deze drie mogelijkheden zijn aan het eind van het literatuuronderzoek nader onderzocht. De mogelijkheden die tot de selectie behoren zijn Swagger, API Blueprint en RAML. Als we de drie mogelijkheden op basis van de strategieën met elkaar gaan vergelijken, is Swagger de enige mogelijkheid die werkt met behulp van de bottom-up strategie. Hoewel in paragraaf 6.2 is aangegeven dat uit het interview is gebleken dat ENOVATION gebruik maakt van de top-down strategie, is Swagger vanwege de rijke en duidelijke documentatie toch een manier die mogelijk een nieuwe kijk kan geven op het proces van ENOVATION.

API Blueprint en RAML zijn daarentegen twee mogelijkheden die allebei gebruik maken van de top-down strategie. Beide mogelijkheden vereisen dat er aan het begin van het ontwikkeltraject een beschrijving van de RESTful Web API wordt opgesteld. Zowel API Blueprint als RAML kent een eenvoudige syntax voor het maken van deze beschrijving. Dit heeft als groot voordeel dat de beschrijving voor iedereen makkelijk te lezen en te begrijpen is. Daarnaast beschikken beide mogelijkheden over handige tools die een grote bijdrage konden leveren aan het ontwikkelproces.

In overleg met de opdrachtgever is ervoor gekozen om twee mogelijkheden met een totaal verschillende strategie te gebruiken bij de ontwikkeling van het proof of concept, omdat de opdrachtgever het belangrijk vond dat er ook gekeken werd naar een strategie die niet direct bij het proces van ENOVATION aansloot. Dit betekende dat Swagger in ieder geval gebruikt zou worden bij de ontwikkeling van het proof of concept en er nog een keuze gemaakt moest worden tussen API Blueprint en RAML. Zoals in het literatuuronderzoek te zien is, hebben API Blueprint en RAML een vergelijkbare werkwijze. Uiteindelijk heb ik voor RAML gekozen, omdat RAML over meer tools beschikt die aansluiten op de technieken van ENOVATION, zoals de SoapUI RAML Plugin en RAMLang. Daarnaast is RAML vrij nieuw en is de kans groot dat de community nog veel meer tools voor RAML ontwikkelt.

De twee mogelijkheden die op het proof of concept toegepast zouden worden, waren dus Swagger en RAML.

# 9 Proof of concept

Dit hoofdstuk bespreekt de werkzaamheden rondom het ontwikkelen van het proof of concept en het toepassen van de mogelijkheden die in het vorige hoofdstuk naar voren zijn gekomen.

## 9.1 Inleiding

Een proof of concept is een project waarmee de potentie van een bepaalde methode of idee wordt gedemonstreerd. Tijdens deze opdracht werd met het proof of concept de potentie van Swagger en RAML onderzocht. Hoewel een proof of concept vaak een klein project en een versimpelde versie van de werkelijkheid is, was het echter voor deze opdracht wel gewenst dat het proof of concept voldoende complexiteit zou bevatten. In overleg met de opdrachtgever is besloten dat het proof of concept complex genoeg zou zijn wanneer het proof of concept twee of meer entiteiten zou hebben die onderling een relatie hebben. Een entiteit is iets wat een eigen bestaan heeft, zoals bijvoorbeeld een auto. Alles wat bij die specifieke auto hoort en geen eigen bestaan heeft, wordt gezien als een value attribuut. Ik heb voor twee entiteiten gekozen, waarmee bijna elk complex systeem te maken heeft, namelijk de gebruikers en groepen. Aangezien het proof of concept ging over de potentie van RAML of Swagger en niet over het ontwikkelen van een nieuwe RESTful Web API, heb ik besloten om het ontwerp te baseren op een standaard voor het managen van gebruikers en groepen, namelijk SCIM.

#### SCIM

System for Cross-domain Identity Management (SCIM) is een standaard en heeft als doel het beheren van gebruikers in de cloud te vergemakkelijken. Dit probeert SCIM te bereiken door de manier waarop gebruikers en groepen worden gepresenteerd te standaardiseren. Hierdoor kan het verplaatsen van gebruikers in en om de cloud sneller, makkelijker en goedkoper gebeuren.

## 9.2 Voorbereiding

Voordat ik kon beginnen met het ontwikkelen van het proof of concept, was het belangrijk dat ik een geschikte ontwikkelomgeving had opgezet en een aanpak had opgesteld.

### 9.2.1 Het opzetten van de ontwikkelomgeving

Om een geschikte ontwikkelomgeving op te zetten, was het van belang dat ik eerst naging welke technieken onderdeel uit moesten maken van de ontwikkelomgeving. Deze technieken heb ik op basis van drie verschillende bronnen opgesteld, namelijk op basis van de technieken die ENOVATION gebruikt bij het ontwikkelen van zijn RESTful Web APIs, de technieken die tijdens het literatuuronderzoek naar voren zijn gekomen en de technieken waarvan ik dacht dat ik ze nodig zou hebben.

Aangezien ik de werkwijze van ENOVATION in een zo’n groot mogelijke mate probeerde te kopiëren, heb ik eerst gekeken naar welke technieken van ENOVATION onderdeel uit zouden gaan maken van de ontwikkelomgeving. Hiervoor heb ik gebruik gemaakt van de tabel in paragraaf 6.3. De technieken die uiteindelijk onderdeel waren van de ontwikkelomgeving, zijn Spring Tool Suite, Apache Maven, SoapUI en Git.

#### Spring Tool Suite

Uit de technieken van ENOVATION blijkt dat ENOVATION gebruik maakt van het Spring Framework. Spring Tool Suite (STS) is een ontwikkelprogramma dat speciaal op maat gemaakt is voor het ontwikkelen van Spring applicaties. Aangezien de ervaringen van ENOVATION met STS erg goed zijn, heb ik ervoor gekozen om dit programma ook te gaan gebruiken bij het ontwikkelen van het proof of concept.

#### Apache Maven

Apache Maven is een techniek die door ENOVATION gebruikt wordt voor het managen van een Java-project en het automatiseren van de softwarebouw. Door middel van Maven is het voor ENOVATION mogelijk om binnen enkele minuten een werkende applicatie draaiende te krijgen. Gezien er maar een beperkte tijd beschikbaar was voor het ontwikkelen van een proof of concept, was het zeer gunstig als de ontwikkeling van het proof of concept snel kon beginnen. Het gebruik van Apache Maven was voor deze opdracht dus zeer voordelig.

#### SoapUI

ENOVATION gebruikt voor het testen van zijn RESTful Web APIs het programma SoapUI. SoapUI is een gratis open source programma voor het testen van APIs, waarmee door middel van een GUI snel tests gemaakt en herhaaldelijk uitgevoerd kunnen worden. SoapUI is zeer geschikt voor het testen van RESTful Web APIs en sluit goed aan op de aanpak die in de volgende paragraaf wordt besproken.

#### Git

Git is een versiebeheersysteem en wordt gebruikt voor het managen van de broncode. ENOVATION maakt momenteel gebruik van een ander versiebeheersysteem genaamd Subversion, maar bevindt zich midden in het proces om over te stappen naar Git. Dit is dan ook de reden waarom ik gedurende deze opdracht gebruik maak van Git in plaats van Subversion.

Vervolgens heb ik gekeken naar de verschillende technieken die tijdens het literatuuronderzoek naar voren zijn gekomen. De technieken die uiteindelijk onderdeel uitgemaakt hebben van de ontwikkelomgeving, zijn SoapUI RAML plugin, RAML to HTML en SwaggerUI.

#### SoapUI RAML Plugin

De SoapUI RAML Plugin is een plugin voor SoapUI en maakt het mogelijk om een RAML definitie te importeren in SoapUI. Hierdoor kunnen de testcases en mock server op basis van de RAML definitie automatisch worden gegenereerd. De plugin zorgde voor een vloeiende overgang van de ontwerpfase naar de testfase.

#### RAML to HTML / SwaggerUI

RAML to HTML en SwaggerUI zijn allebei tools die tijdens het literatuuronderzoek naar voren zijn gekomen en kunnen worden gebruikt om de documentatie van een RESTful Web API om te zetten naar een voor de mens makkelijker te begrijpen formaat, namelijk HTML. Gezien het belangrijk was dat de documentatie ook voor de klanten te begrijpen zou zijn, heb ik ervoor gekozen om beide tools onderdeel uit te laten maken van de ontwikkelomgeving. Hierdoor was ik uiteindelijk in staat om te kijken welke rol de tools konden spelen tijdens het uitwisselen van de documentatie.

Als laatst heb ik gekeken naar de technieken waarvan ik vond dat ze onderdeel uit moesten maken van de ontwikkelomgeving. Deze technieken waren Draw.io en MySQL Server.

#### Draw.io (UML)

Unified Modeling Language (UML) is modelleertaal die gebruikt kan worden voor het ontwerpen van een systeem en biedt een verzameling van diagrammen waarmee de structuur en het gedrag van een systeem grafisch weergegeven kan worden. Voor het ontwerpen van het proof of concept heb ik specifiek gekozen voor UML, omdat UML de standaard is die ik op school heb geleerd en waarmee ik dus al de nodige ervaring had. Voor het maken van de diagrammen is de applicatie Draw.io gebruikt. Draw.io is een gratis online applicatie voor het opstellen van diagrammen. Het grote voordeel van Draw.io is dat het een online applicatie betreft die een connectie heeft met Google Drive. Hierdoor zijn de opgestelde diagrammen op alle apparaten met een internetverbinding bereikbaar.

#### MySQL Server (MySQL)

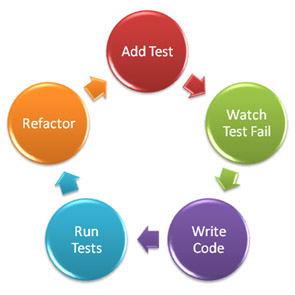
Het opslaan van de data van de RESTful Web API kon op verschillende manieren worden gedaan. Twee manieren hiervoor zijn het opslaan van de data in een database (extern) en het hardcoded (intern) opslaan van de data. Tijdens het hardcoded opslaan van de data, wordt de data opgeslagen in de broncode van het systeem. Voor het ontwikkelen van het proof of concept heb ik ervoor gekozen om een MySQL Server op te zetten en de data op te slaan in een MySQL database, omdat het zeer ongebruikelijk is om data hardcoded op te slaan en een database net even wat extra complexiteit aan de RESTful Web API geeft. Daarnaast heb ik gekozen voor een MySQL database in plaats van bijvoorbeeld een Oracle database, omdat ik in de afgelopen jaren ruime ervaring opgedaan heb met MySQL en hier dus al reeds bekend mee was.

### 9.2.2 Aanpak

Zoals in paragraaf 2.3 staat beschreven maakt ENOVATION gebruik van Scrum. Helaas maakte ik tijdens het uitvoeren van deze opdracht geen onderdeel uit van een team. Hierdoor was het voor mij niet mogelijk om Scrum te gebruiken. Na het interview was wel bekend dat ENOVATION begint met het maken van testcases en het opzetten van een mock server met behulp van SoapUI. Het beginnen met het maken van testcases wordt ook wel Test Driven Development (TDD) genoemd en is een ontwikkelmethode die goed geschikt is voor het ontwikkelen van RESTful Web APIs. Daarom heb ik ervoor gekozen om TDD te gaan gebruiken bij het ontwikkelen van het proof of concept.

#### Test Driven Development

Test Driven Development (TDD) is een ontwikkelmethode waarbij een korte ontwikkelcyclus steeds herhaald wordt. Een schematische weergave van de ontwikkelcyclus is te zien in figuur 1 (When Is It OK Not To TDD?, 2010).



Figuur : de ontwikkelcyclus van TDD

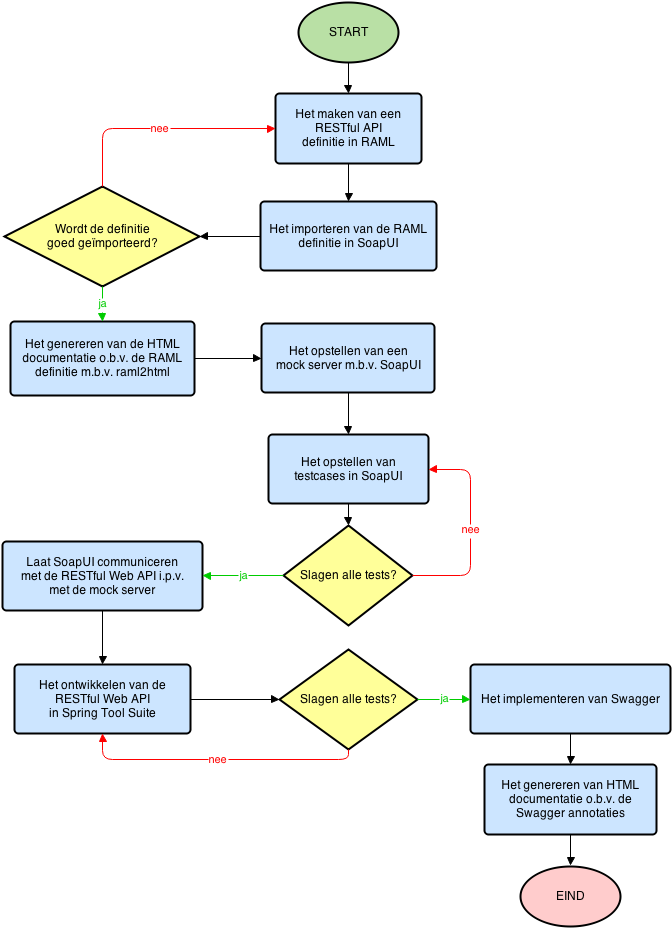
De ontwikkelaar begint het proces met het schrijven van een test die bijvoorbeeld een gewenste nieuwe functie moet testen (**Add Test**). Hierbij is het van belang dat er nog geen code is opgesteld en de test dus in eerste instantie faalt (**Watch Test Fail**). Vervolgens begint de ontwikkelaar met het opstellen van een minimum aantal regels code (**Write Code**) waarmee de test in staat is te slagen (**Run Tests**). Als laatste stap in de cyclus wordt de code opnieuw gestructureerd waardoor het voldoet aan de standaard (**Refactor**).

Zoals in het literatuuronderzoek is gebleken maken Swagger en RAML allebei gebruik van een andere strategie. Daarom was het belangrijk om te weten in welke volgorde de twee mogelijkheden toegepast moesten worden. Aangezien RAML wordt toegepast in de ontwerpfase en Swagger pas nadat er gecodeerd is, heb ik ervoor gekozen om eerst het ontwerp te maken waarbij RAML werd toegepast. Vervolgens kon de RESTful Web API worden ontwikkeld en als laatst Swagger worden toegepast.

Als eerst zou er dus een ontwerp gemaakt moeten worden van de RESTful Web API dat bestond uit het maken van een UML diagram van de database en een RAML definitie. Aangezien ik zoals hierboven staat beschreven van tevoren heb bepaald dat ik Test Driven Development ging gebruiken, moesten er na het ontwerp testcases worden opgesteld. Deze testcases moesten opgesteld worden met behulp van het programma SoapUI. Door de SoapUI RAML plugin zou ik vervolgens in staat zijn om de RAML definitie in te laden en automatisch een mock server op te zetten. Door middel van de mock server kon ik dan al de gewenste responses opstellen en de RESTful Web API testen voordat er ook maar één regel code getypt was. Op het moment dat alle testcases slaagden, kon ik beginnen met het ontwikkelen van de RESTful Web API in Spring Tool Suite. Door SoapUI vervolgens te laten communiceren met de RESTful Web API in plaats van de mock server, zou ik na het toevoegen van een functionaliteit en het nog een keer uitvoeren van de testcases in SoapUI direct terugkoppeling krijgen. Wanneer een testcase niet zou slagen, wist ik dat ik de code moest aanpassen. Wanneer alle testcases zouden slagen, was het tijd om naar de volgende fase over te gaan, namelijk het toepassen van Swagger.

#### Een schematische weergave van de aanpak

De hierboven beschreven aanpak heb ik uitgewerkt in een flowchart. Een flowchart is een schematische weergave van het proces, waarin de verschillende stappen door middel van pijlen in chronologische volgorde zijn weergegeven. De flowchart die ik heb gemaakt, heeft geholpen bij het volgen van de juiste stappen en is te zien in figuur 6.



Figuur 6: een flowchart van het ontwikkeltraject

## 9.3 Ontwerp

### 9.3.1 Het ontwerpen van het proof of concept

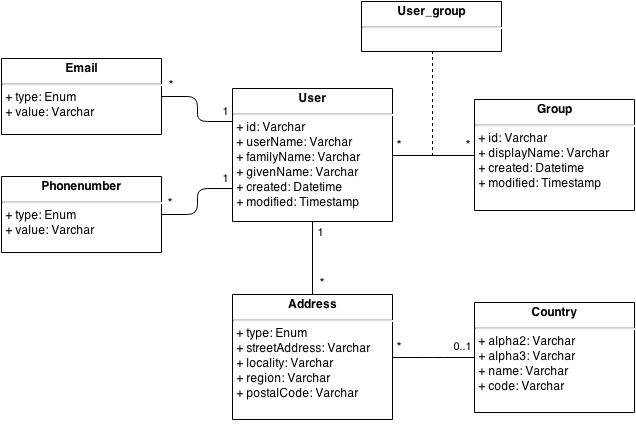
Zoals in de inleiding van dit hoofdstuk stond beschreven, zou het proof of concept gebaseerd worden op SCIM. SCIM probeert door middel van enkele schema’s de manier waarop gebruikers en groepen gepresenteerd worden te standaardiseren. Een schema beschrijft het formaat van de data en kan worden gebruikt voor de validatie van de data. Twee voorbeelden van hoe een user en een group gepresenteerd kunnen worden, zijn te zien in figuur 7 en 8 (SCIM: System for Cross-domain Identity Management, 2014).



Figuur 7: een voorbeeld van een user presentatie Figuur 8: een voorbeeld van een group presentatie

In de hierboven genoemde figuren is te zien dat SCIM de gebruikers en groepen in een JSON-formaat presenteren. Elke presentatie bevat een attribuut ‘schemas’ waarvan de waarde verwijst naar een schema. Dit schema bevat vervolgens welke attributen de presentatie kan bevatten en welke attributen de presentatie verplicht moet bevatten. Hierdoor kan de data op basis van het schema gevalideerd worden. Gezien ENOVATION nog geen ervaring heeft met het toepassen van schema’s, heb ik ze tijdens het ontwikkelen van het proof of concept ook achterwege laten.

SCIM bestaat slechts uit twee verschillende entiteiten, namelijk gebruikers en groepen. In figuur 8 is te zien dat een groep meerdere gebruikers (members) kan hebben. Dit duidt erop dat er een relatie bestaat tussen de gebruikers en groepen. Aangezien de attributen *phoneNumbers* en *emails* slechts over een value en een type beschikken en niet over een *id*, zijn dit value attributen en geen entiteiten. Op basis van de voorbeelden en de specificatie op de officiële website van SCIM, heb ik met behulp van Draw.io een database ontwerp gemaakt. Het ontwerp van de database is te zien in figuur 9.



Figuur 9: ontwerp van de database

In figuur 9 is te zien dat het database ontwerp uit zeven tabellen bestaat, namelijk:

* User
* Email
* Address
* Phonenumber
* Country
* Group
* User\_group

Hierbij zijn *User* en *Group* de tabellen waarin de gebruikers en groepen worden opgeslagen. De relatie tussen een gebruiker en een groep, wordt opgeslagen in de tabel *User\_group*. De tabellen *Email*, *Phonenumber*, *Address* en *Country* vormen samen de value attributen van een gebruiker.

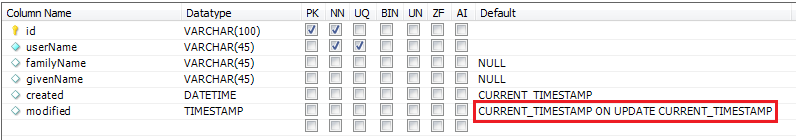
De entiteiten User en Group beschikken allebei over een attribuut ‘id’. De waarde van dit attribuut wordt gebruikt om een gebruiker of groep in de RESTful Web API te identificeren. Het toekennen van een uniek nummer aan bijvoorbeeld een gebruiker is een veel gebruikte manier voor identificatie. Het probleem hiervan is echter dat twee verschillende databases niet gemakkelijk samengevoegd kunnen worden, omdat er dan mogelijk nummers dubbel voorkomen. Een oplossing hiervoor is het gebruik van Universally Unique Identifiers (UUID). Volgens de specificatie van UUID (UUIDs and GUIDs, 1998):

A UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDs. To be precise, the UUID consists of a finite bit space. Thus the time value used for constructing a UUID is limited and will roll over in the future (approximately at A.D. 3400, based on the specified algorithm). A

UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network.

Zoals in de tabellen Email, Phonenumber en Address te zien is, beschikken alle drie de tabellen over een attribuut *‘type*’. Dit attribuut heeft als type Enumeration (afgekort: Enum) . Dit betekent dat de waarde van het attribuut overeen moet komen met één van de waarden die van tevoren worden vastgelegd. Enkele waarden die bijvoorbeeld voor het attribuut ‘type’ in de tabel Email worden vastgelegd, zijn ‘home’ en ‘work’.

De tabellen User en Group beschikken allebei over de attributen *created* en *modified*. Dit zijn de attributen waarin wordt opgeslagen wanneer een entiteit is aangemaakt of voor het laatst is aangepast. Het is belangrijk om op te merken dat de attributen van een ander type zijn. Het attribuut ‘created’ is namelijk van het type Datetime en ‘modified’ van het type Timestamp. Dit heeft ermee te maken dat een attribuut van het type Timestamp automatisch de waarde van de huidige tijd en datum kan aannemen op het moment dat de entiteit wordt aangepast. De standaard waarde van het attribuut is dan:

  
Gezien dit bij een attribuut van het type Datetime op een lastigere manier mogelijk is, heb ik voor het type Timestamp gekozen.

Wanneer de figuren en het database ontwerp goed moet elkaar vergeleken worden, is te zien dat niet alle attributen terugkomen in het database ontwerp. Dit komt omdat het proof of concept gebaseerd is op SCIM en het niet de bedoeling was om SCIM één op één over te nemen. De attributen die wel in het ontwerp staan, waren naar mijn mening genoeg om het proof of concept voldoende complexiteit te geven.

Toen het database ontwerp opgesteld was, kon ik gaan kijken naar hoe het proof of concept er straks uit moest gaan zien. Hiervoor was het belangrijk om te kijken naar de entiteiten binnen het proof of concept en op welke manier deze entiteiten beheerd konden worden. Eén van de principes van REST is dat het verder bouwt op het protocol HTTP. Dit betekent dat er met een RESTful Web API gecommuniceerd kan worden door middel van HTTP requests en HTTP responses. Elke request moest gebruik maken van één van de methoden die HTTP te bieden heeft. De vier meest voorkomende methoden zijn:

* **GET**

*De GET methode wordt gebruikt voor het ophalen van een bestaande resource.*

* **POST**

*De POST methode wordt gebruikt voor het aanmaken van een nieuwe resource.*

* **PUT**

*De PUT methode wordt gebruikt voor het aanpassen van een bestaand resource of het aanmaken van een nieuwe resource op het moment dat de resource nog niet bestaat.*

* **DELETE**

*De DELETE methode wordt gebruikt voor het verwijderen van een bestaande resource.*

Voor een volledige beschrijving van HTTP verwijs ik u door naar het RFC-document van HTTP dat te vinden is in de literatuurlijst.

Een RESTful Web API bepaalt op basis van de HTTP methode wat er moet gaan gebeuren. Het resultaat hiervan wordt vervolgens als HTTP response teruggegeven. Voor het ontwerpen van het proof of concept was het van belang om te bedenken welke HTTP requests er mogelijk waren en wat de verschillende responses hierop konden zijn. In de onderstaande tabellen zijn deze verschillende requests en responses weergegeven.

#### Responses endpoint: /users

|  |  |
| --- | --- |
| **Request** | **Response** |
| |  |  | | --- | --- | | HTTP Method | GET | | |  |  | | --- | --- | | HTTP Status Code | 200 OK | | Body | Presentatie van alle users | |
| |  |  | | --- | --- | | HTTP Method | POST | | Body | Presentatie van een user | | |  |  | | --- | --- | | HTTP Status Code | 201 CREATED | | Headers | Location | | Body | Presentatie van de user | |
| |  |  | | --- | --- | | HTTP Status Code | 400 BAD REQUEST | | Body | Presentatie van een error | |

#### Responses endpoint: /users/{user\_id}

|  |  |
| --- | --- |
| **Request** | **Response** |
| |  |  | | --- | --- | | HTTP Method | GET | | |  |  | | --- | --- | | HTTP Status Code | 200 OK | | Headers | Location | | Body | Presentatie van de user | |
| |  |  | | --- | --- | | HTTP Status Code | 404 NOT FOUND | | Body | Presentatie van een error | |
| |  |  | | --- | --- | | HTTP Method | PUT | | Body | Presentatie van een user | | |  |  | | --- | --- | | HTTP Status Code | 200 OK | | Headers | Location | | Body | Presentatie van de user | |
| |  |  | | --- | --- | | HTTP Status Code | 400 BAD REQUEST | | Body | Presentatie van een error | |
| |  |  | | --- | --- | | HTTP Status Code | 404 NOT FOUND | | Body | Presentatie van een error | |
| |  |  | | --- | --- | | HTTP Method | DELETE | | |  |  | | --- | --- | | HTTP Status Code | 204 NO CONTENT | |
| |  |  | | --- | --- | | HTTP Status Code | 404 NOT FOUND | | Body | Presentatie van een error | |

#### Responses endpoint: /groups

|  |  |
| --- | --- |
| **Request** | **Response** |
| |  |  | | --- | --- | | HTTP Method | GET | | |  |  | | --- | --- | | HTTP Status Code | 200 OK | | Body | Presentatie van alle groups | |
| |  |  | | --- | --- | | HTTP Method | POST | | Body | Presentatie van een group | | |  |  | | --- | --- | | HTTP Status Code | 201 CREATED | | Headers | Location | | Body | Presentatie van de group | |
| |  |  | | --- | --- | | HTTP Status Code | 400 BAD REQUEST | | Body | Presentatie van een error | |

#### Responses endpoint: /groups/{group\_id}

|  |  |
| --- | --- |
| **Request** | **Response** |
| |  |  | | --- | --- | | HTTP Method | GET | | |  |  | | --- | --- | | HTTP Status Code | 200 OK | | Headers | Location | | Body | Presentatie van de group | |
| |  |  | | --- | --- | | HTTP Status Code | 404 NOT FOUND | | Body | Presentatie van een error | |
| |  |  | | --- | --- | | HTTP Method | PUT | | Body | Presentatie van een group | | |  |  | | --- | --- | | HTTP Status Code | 200 OK | | Headers | Location | | Body | Presentatie van de group | |
| |  |  | | --- | --- | | HTTP Status Code | 400 BAD REQUEST | | Body | Presentatie van een error | |
| |  |  | | --- | --- | | HTTP Status Code | 404 NOT FOUND | | Body | Presentatie van een error | |
| |  |  | | --- | --- | | HTTP Method | DELETE | | |  |  | | --- | --- | | HTTP Status Code | 204 NO CONTENT | |
| |  |  | | --- | --- | | HTTP Status Code | 404 NOT FOUND | | Body | Presentatie van een error | |

In de tabellen wordt er gebruik gemaakt van de term ‘endpoint’. Een endpoint vormt een toegangspunt tot de RESTful Web API waarachter resources schuil gaan. Zoals in de tabellen te zien is, bestaan er vier verschillende endpoints:

* /users
* /users/{user\_id}
* /groups
* /groups/{group\_id}

De hierboven genoemde endpoints zijn gebaseerd op de manier waarop het in REST wordt aanbevolen. Hierbij moeten *{user\_id}* en *{group\_id}* vervangen worden door het id van de resource (bijvoorbeeld: */users/90347d30-495e-11e4-916c-0800200c9a66*). Doord de endpoints op deze manier op te stellen, werd er voldaan aan het criteria Identification of resources.

In de linker kolom zijn de verschillende HTTP requests weergegeven met ieder een HTTP methode en eventueel een body. Aan de rechter kant van iedere request zijn alle mogelijke HTTP responses op die request weergegeven. Daarbij heeft iedere response een status code en eventueel de headers en een body.

### 9.3.2 Het ontwerpen van de RAML definitie

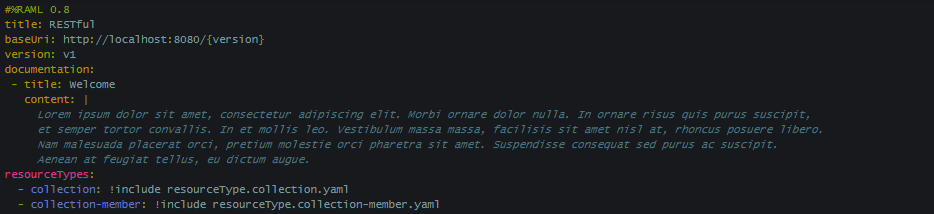
In het literatuuronderzoek is gebleken dat er tijdens het toepassen van RAML eerst een RAML definitie wordt gemaakt van de RESTful Web API. Hierdoor kun je eenvoudig in kaart brengen hoe het ontwerp van een RESTful Web API eruitziet. Het was tijdens deze opdracht dan ook van belang dat ik een RAML definitie zou maken van het proof of concept.

Om een goede RAML definitie te kunnen opstellen, heb ik de officiële website van RAML geraadpleegd. Op de website werd stap voor stap uitgelegd hoe je een definitie moest opstellen en welke syntax je hierbij moest hanteren. Een klein stukje van de RAML definitie van het proof of concept is te zien in figuur 10.



Figuur 10: een stukje uit de RAML definitie van het proof of concept

Zoals in het bovenstaande figuur te zien is, is het een hele overzichtelijke syntax die makkelijk te begrijpen is. In het figuur worden van alle mogelijke requests op de endpoints */users* en */users/{user\_id* de responses gedefinieerd. Daarnaast wordt voor allebei de endpoints een type gedefinieerd. In deze RAML definitie verwijst het type naar een resourceType die bovenaan de RAML definitie gedefinieerd wordt, zoals te zien is in figuur 11.



Figuur 11: de definitie van de resourceTypes

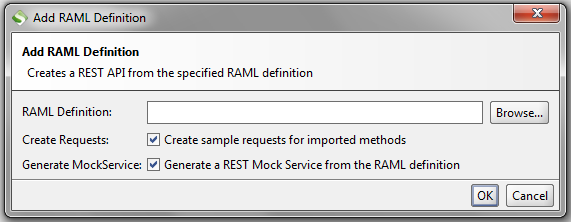
In figuur 11 is te zien dat de resourceTypes *collection* en *collection-member* verwijzen naar een extern bestand. Door het verplaatsen van definities naar externe bestanden, is het mogelijk om de definities ook in andere projecten te gebruiken. Het hergebruiken van al bestaande definities zorgt ervoor dat een RAML definitie nog sneller kan worden opgesteld. Tijdens het maken van de RAML definitie heb ik geprobeerd om zoveel mogelijk te definiëren in externe bestanden, omdat het een groot voordeel van RAML is en omdat ik wilde aantonen hoe makkelijk het te realiseren is.

## 9.4 Testen

Zoals in de aanpak beschreven staat, zou ik gedurende de ontwikkeling van het proof of concept gebruik maken van TDD. Dit betekende dat ik na het maken van een ontwerp moest beginnen met het opzetten van een testomgeving en het opstellen van testcases.

### 9.4.1 Het opzetten van de testomgeving

Voor het testen van het proof of concept heb ik SoapUI in combinatie met de SoapUI RAML plugin gebruikt. Hierdoor was ik in staat om de RAML definitie die ik in de ontwerpfase had opgesteld te importeren in SoapUI (figuur 12).



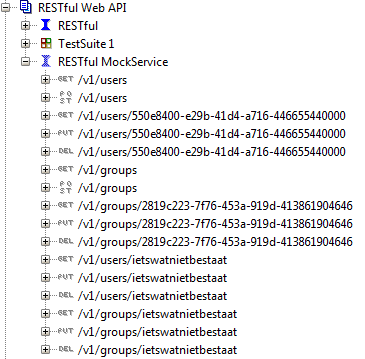
Figuur 12: het importeren van de RAML definitie

Vervolgens kon SoapUI op basis van de RAML definitie automatisch een mock server opzetten. Aangezien de plugin nog volop in ontwikkeling is, verliep het importeren niet helemaal vlekkeloos. Zo werd bijvoorbeeld {user\_id} in een endpoint niet ondersteund. Om dit op te lossen heb ik deze dynamische ids vervangen door een door mij willekeurig gekozen id. Dit betekende wel dat alleen de user met die specifieke id via de mock server benaderd kon worden. Om het niet bestaan van een user of group te simuleren, heb ik de volgende endpoints ook opgenomen in de mock server:

* /users/ietswatnietbestaat
* /groups/ietswatnietbestaat

Door het toevoegen van de bovenstaande endpoints was ik in staat om een testcase op te stellen voor als een user of group niet gevonden kon worden.

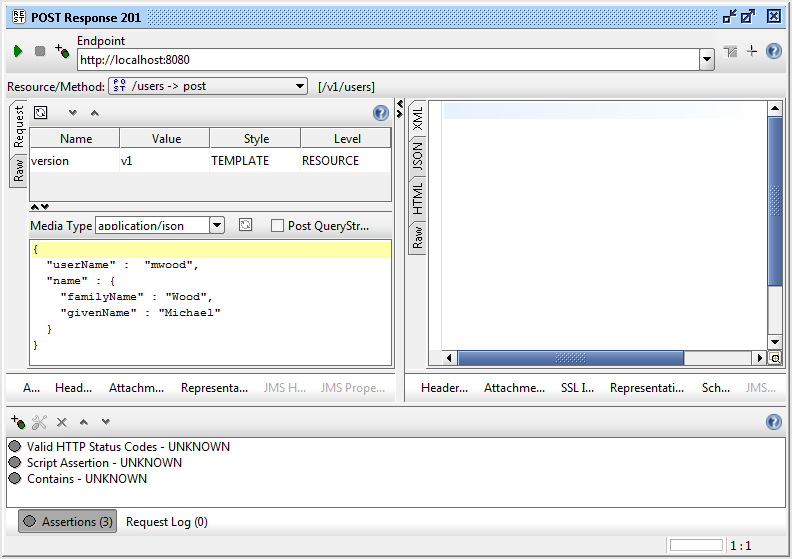
In het onderstaande figuur is te zien hoe de mock server er uiteindelijk uitzag.



Figuur 13: een mock server in SoapUI

### 9.4.2 Het opstellen van de testcases

Toen de mock server volledig was opgezet, was het tijd om de verschillende testcases op te stellen in SoapUI. Met een testcase wordt onderzocht of het systeem onder bepaalde omstandigheden het gewenste gedrag vertoont. Aangezien het proof of concept uit vier endpoints bestond, heb ik vier testcases opgesteld waarin teststeps verantwoordelijk waren voor het testen van alle mogelijke responses binnen dat endpoint. In het onderstaande figuur is te zien hoe een teststep werd opgesteld in SoapUI.

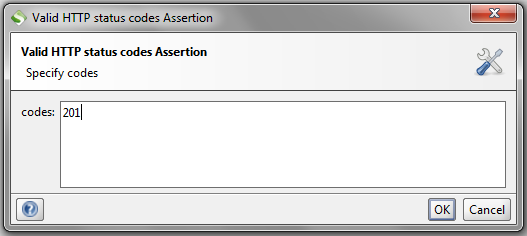


Figuur 14: het opstellen van een testcase in SoapUI

Bij het opstellen van een teststep in SoapUI, wordt als eerst een request opgesteld die bij het uitvoeren van de teststep verzonden wordt naar de RESTful Web API of naar de mock server als dit zo is ingesteld. Vervolgens worden er zogenaamde ‘assertions’ toegevoegd. Een assertion is een bewering over de response en tijdens het uitvoeren van de teststep wordt nagegaan of deze bewering klopt. In figuur 14 zijn onderin drie verschillende assertions te zien. Hoewel SoapUI een aantal verschillende soorten assertions kent, heb ik voor het testen van het proof of concept alleen gebruik gemaakt van de volgende drie:

#### Valid HTTP Status Codes

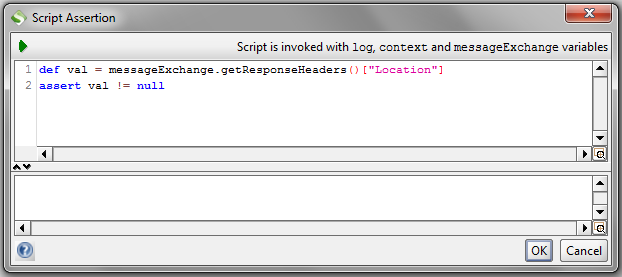
Bij een Valid HTTP Status Codes Assertion wordt de HTTP Status Code van de response gecontroleerd. De teststep slaagt alleen wanneer de HTTP Status Code overeenkomt met één van de codes die in de assertion gedefinieerd staan. Een voorbeeld van hoe een Valid HTTP Status Codes Assertion in SoapUI wordt toegevoegd, is te zien in figuur 15.



Figuur 15: een voorbeeld van een Valid HTTP Status Codes Assertion

#### Script Assertion

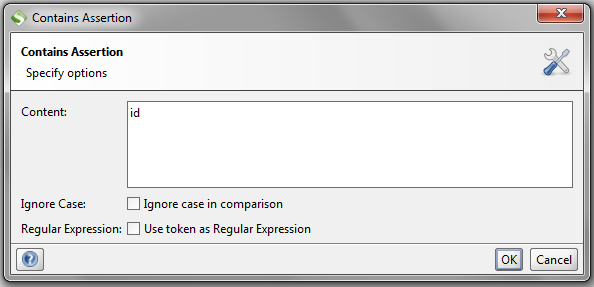
Met een Script Assertion is het mogelijk om de response te controleren op basis van een zelfgemaakt script. Het script wordt opgesteld met behulp van de scripttaal Groovy en zorgt ervoor dat de response op een nog uitgebreidere manier gecontroleerd kan worden. Zo is het bijvoorbeeld mogelijk om de headers te controleren of te kijken of de response tijd onder een bepaald limiet blijft. In figuur 16 is een voorbeeld te zien van een Script Assertion, waarbij er gecontroleerd wordt of de header ‘location’ bestaat.



Figuur 16: een voorbeeld van een Script Assertion

#### Contains

Een Contains Assertion controleert of de body van de response bepaalde begrippen bevat. De teststep slaagt op het moment dat de in de assertion gedefinieerde begrippen gevonden worden in de body. Hierbij is het mogelijk om tijdens het zoeken geen rekening te houden met hoofd- en kleine letters en om gebruik te maken van een reguliere expressie (patroon). Een voorbeeld van hoe een Contains Assertion in SoapUI kan worden toegevoegd, is te zien in figuur 17.



Figuur 17: een voorbeeld van een Contains Assertion

In de volgende tabellen zijn de assertions te zien die ik voor alle verschillende responses heb opgesteld.

#### TestCase endpoint: /users

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **HTTP Status Code** | **Assertions** |
| GET | 200 | |  |  | | --- | --- | | Valid HTTP Status Code | 200 | | Contains | id | |
| POST | 201 | |  |  | | --- | --- | | Valid HTTP Status Code | 201 | | Script | Header location | | Contains | id | |
| 400 | |  |  | | --- | --- | | Valid HTTP Status Code | 400 | | Contains | description | |

#### TestCase endpoint: /users/{user\_id}

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **HTTP Status Code** | **Assertions** |
| GET | 200 | |  |  | | --- | --- | | Valid HTTP Status Code | 200 | | Script | Header location | | Contains | id | |
| 404 | |  |  | | --- | --- | | Valid HTTP Status Code | 404 | | Contains | description | |
| PUT | 200 | |  |  | | --- | --- | | Valid HTTP Status Code | 200 | | Script | Header location | | Contains | id | |
| 400 | |  |  | | --- | --- | | Valid HTTP Status Code | 400 | | Contains | description | |
| 404 | |  |  | | --- | --- | | Valid HTTP Status Code | 404 | | Contains | description | |
| DELETE | 204 | |  |  | | --- | --- | | Valid HTTP Status Code | 204 | | Contains | description | |
| 404 | |  |  | | --- | --- | | Valid HTTP Status Code | 404 | | Contains | description | |

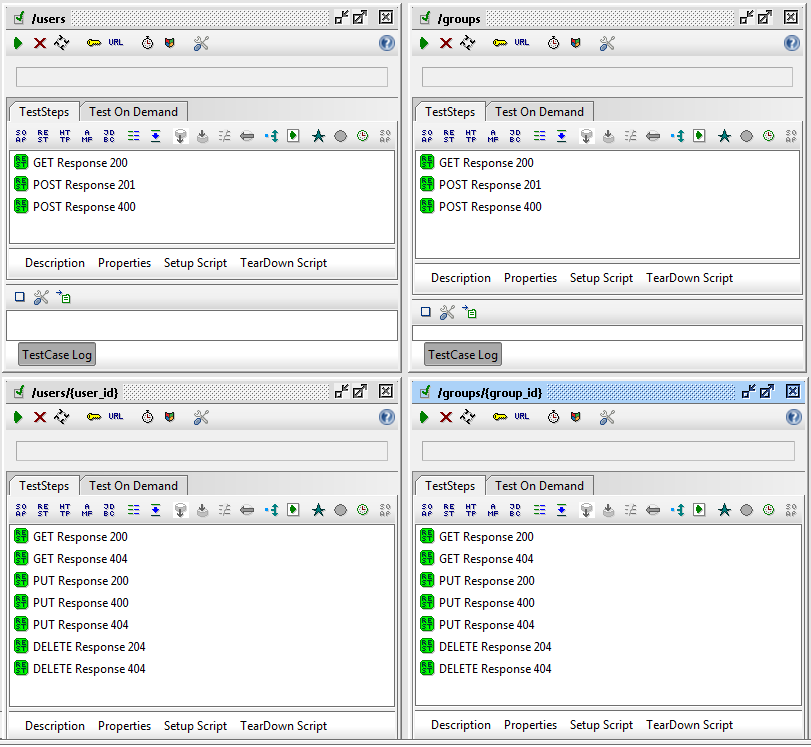
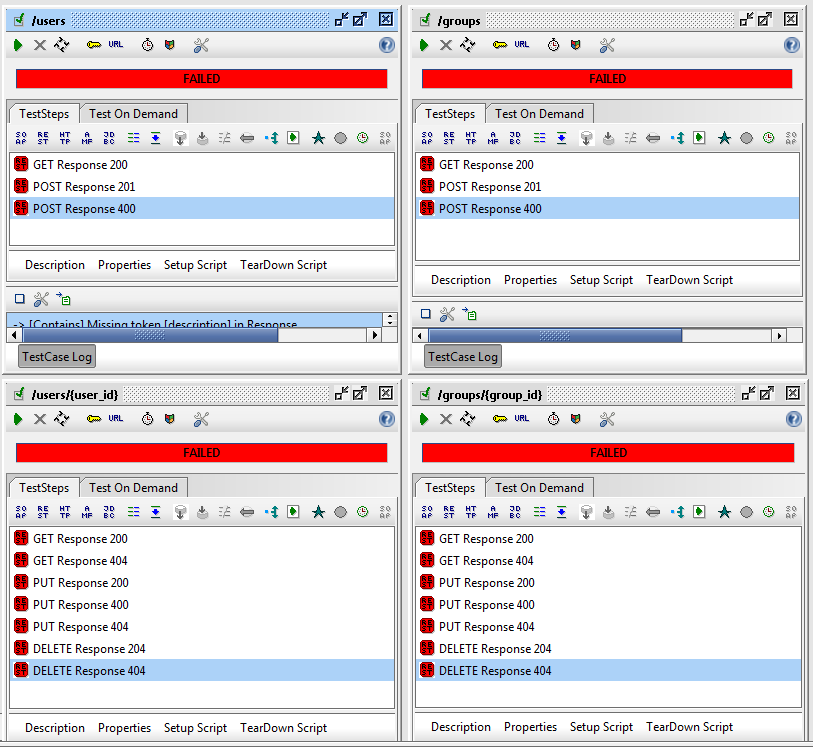
#### TestCase endpoint: /groups

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **HTTP Status Code** | **Assertions** |
| GET | 200 | |  |  | | --- | --- | | Valid HTTP Status Code | 200 | | Contains | id | |
| POST | 201 | |  |  | | --- | --- | | Valid HTTP Status Code | 201 | | Script | Header location | | Contains | id | |
| 400 | |  |  | | --- | --- | | Valid HTTP Status Code | 400 | | Contains | description | |

#### TestCase endpoint: /groups/{group\_id}

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **HTTP Status Code** | **Assertions** |
| GET | 200 | |  |  | | --- | --- | | Valid HTTP Status Code | 200 | | Script | Header location | | Contains | id | |
| 404 | |  |  | | --- | --- | | Valid HTTP Status Code | 404 | | Contains | description | |
| PUT | 200 | |  |  | | --- | --- | | Valid HTTP Status Code | 200 | | Script | Header location | | Contains | id | |
| 400 | |  |  | | --- | --- | | Valid HTTP Status Code | 400 | | Contains | description | |
| 404 | |  |  | | --- | --- | | Valid HTTP Status Code | 404 | | Contains | description | |
| DELETE | 204 | |  |  | | --- | --- | | Valid HTTP Status Code | 204 | | Contains | description | |
| 404 | |  |  | | --- | --- | | Valid HTTP Status Code | 404 | | Contains | description | |

Toen alle testcases in SoapUI waren opgesteld, was het tijd om de testcases te laten communiceren met de opgezette mock server. Aangezien de mock server in SoapUI standaard op het adres <http://localhost:8080>draait en de testcases hier al mee wilden communiceren, hoefde ik niks aan de instellingen te veranderen. Na het uitvoeren van de testcases bleek dat alle testcases slaagden (figuur 18). Op het moment dat ik de mock server stop zette en de testcases nog een keer uitvoerde, faalden alle testcases (figuur 19).

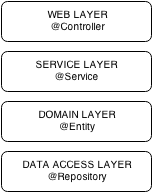
Figuur 18: het uitvoeren van de testcases met een draaiende Figuur 19: het uitvoeren van de testcases zonder een mock  
 mock server server

## 9.5 Bouwen

### 9.5.1 Het bouwen van het proof of concept

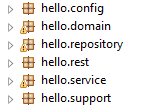
Het proof of concept heb ik gebouwd met behulp van Apache Maven en Spring Tool Suite. Aangezien ik nog geen ervaring had met het ontwikkelen van een RESTful Web API, ben ik op zoek gegaan naar guides die mij konden helpen bij het opzetten van een project. De guide die ik uiteindelijk gebruikt heb, heet *Building a RESTful Web Service* en is een guide van Spring.io. Met behulp van de guide was ik in staat om in een zeer korte tijd een kleine RESTful Web API op te zetten die ik vervolgens zelf kon uitbreiden.

In de guide werd er slechts gebruik gemaakt van twee componenten, namelijk een model en een controller. Een model is verantwoordelijk voor de data van een systeem. Het kan uit een enkel object of soms ook uit een structuur van verschillende objecten bestaan. Een controller is verantwoordelijk voor het verwerken van de requests die door de gebruiker van het systeem worden gemaakt en heeft de kennis hoe hij deze requests moet verwerken.

Een RESTful Web API die gebruik maakt van het Spring Framework is opgebouwd uit vier verschillende lagen. Deze vier lagen zijn in het figuur hiernaast weergegeven.

Zoals in het figuur te zien is, bestaat een RESTful Web API in Spring uit een web, service, domain en data access laag. De web laag uit bestaat uit de controllers die in Spring geannoteerd worden met @Controller en zijn zoals eerder besproken verantwoordelijk voor het afhandelen van de requests. In de service laag bevinden zich de services die in Spring geannoteerd worden met @Service. De services vormen een koppeling tussen de web laag en de data access laag en zijn verantwoordelijk voor de business logica. In de domain laag bevinden zich de modellen van de RESTful Web API. Als laatst hebben we nog de data access laag die uit verschillende repositories bestaat. Een repository wordt in Spring geannoteerd met @Repository en is verantwoordelijk voor de communicatie met de database.

Op basis van de hierboven beschreven lagen, heb ik een mappenstructuur gemaakt voor het proof of concept. Deze mappenstructuur was als volgt:



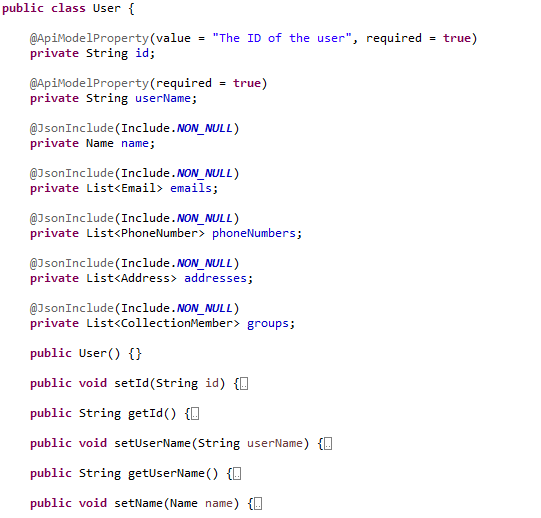
De map *Config* was bedoeld voor configuratie bestanden en de map *Support* voor overige bestanden die de processen konden ondersteunen. Aangezien er in Spring 4 een nieuwe soort controller was, namelijk de RestController in plaats van een Controller, heb ik voor de naam *Rest* gekozen. De map Rest bevatte alle controllers van het proof of concept.

Als eerst heb ik voor alle tabellen die in het database ontwerp te zien zijn, een model gemaakt. Hierdoor werd de structuur in de map Domain als volgt:

**Domain**

* User
* Address
* Email
* PhoneNumber
* Group

Elk model bevatte de attributen die ook in het ontwerp te zien zijn. Een voorbeeld hiervan is te zien in het onderstaande figuur.



Figuur 20: een deel van het user model

In het figuur is te zien dat een User over een attribuut van het type *Name* beschikt. Dit type verwijst naar een ander model en bevat de attributen *givenName* en familyName. Ik heb gekozen om hiervoor een apart model te maken, omdat beide attributen te maken hebben met de naam van de user. Daarnaast worden daardoor ook de attributen in een response overzichtelijker weergegeven.

Vervolgens was het tijd om de controllers, services en repositories aan te maken. Wanneer er gebruik wordt gemaakt van het Spring Framework is het gebruikelijk om voor elke entiteit een eigen controller, service en repository aan te maken. Voor het proof of concept betekende dit dat er een controller, service en repository aangemaakt moest worden voor de entiteiten User en Group. In de onderstaande figuren is te zien hoe de controller, service en repository van de User ongeveer geworden zijn.

#### UserController



Figuur 21: een deel van de UserController

Figuur 21 toont een deel van de controller van de entiteit User. Zoals te zien is, wordt er in de controller veel gebruik gemaakt van annotaties die in de code aangeduid worden met een @ ervoor). De annotatie @RequestMapping(value = “/v1/users”) zorgt ervoor dat alle requests op de endpoint */v1/users* verwerkt worden door de functies binnen de UserController. Om aan te geven welke functie verantwoordelijk is voor het verwerken van welke request, wordt er ook gebruik gemaakt van de @RequestMapping annotatie. Zo geeft bijvoorbeeld de annotatie @RequestMapping(method = RequestMethod.GET) aan dat de functie die eronder staat aangeroepen wordt wanneer er een HTTP GET request wordt verzonden naar de endpoint /v1/users. De functie die wordt aangeroepen, raadpleegt vervolgens de UserService om de request verder te verwerken. Tijdens het verwerken van de request was het belangrijk dat er geen gebruik werd gemaakt van data die aan de kant van de server was opgeslagen. Wanneer dit wel het geval zou zijn, was het proof of concept niet meer stateless.

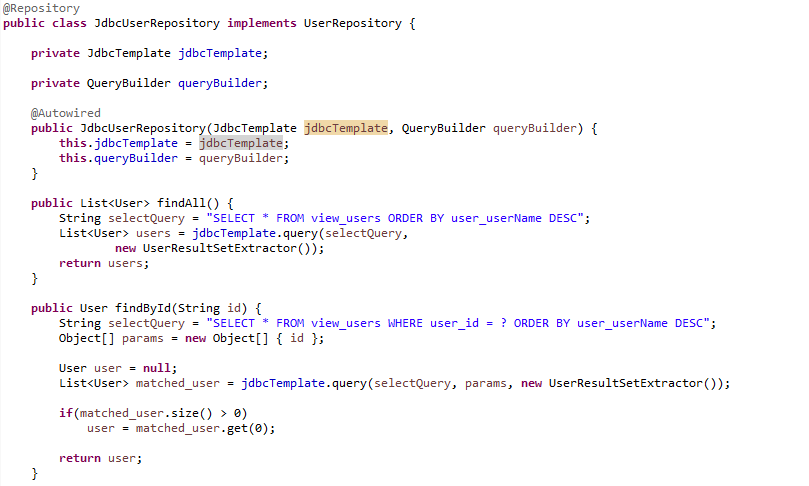
#### UserService



Figuur 22: een deel van de UserService

De UserService is een koppeling tussen de UserController en de UserRepository en is verantwoordelijk voor de business logica. Dit wil onder andere zeggen dat de service bepaalt wat voor soort response er wordt teruggegeven. Wanneer een functie binnen de service vanuit de controller wordt aangeroepen, probeert de service de benodigde data te verkrijgen door een functie binnen de UserRepository aan te roepen. De service bepaalt vervolgens op basis van de teruggekregen data of de response een foutmelding of gewoon de data moet bevatten.

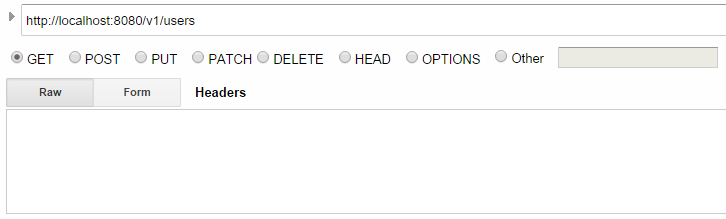
#### UserRepository



Figuur 23: een deel van de UserRepository

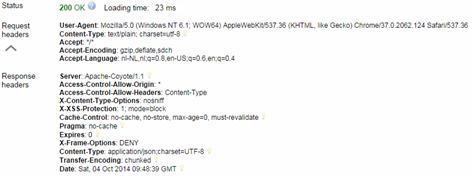
Figuur 23 toont een deel van de UserRepository. Een repository is zoals eerder beschreven verantwoordelijk voor de directe communicatie met de database. Dit betekent dat in de repository de SQL queries worden opgesteld en de database wordt bevraagd. De enige verantwoordelijkheid van een repository is het teruggeven van het antwoord van de database. Enkele fouten die optreden tijdens het bevragen van de database worden afgehandeld door de service.

Na het toevoegen van een nieuwe functionaliteit werden telkens de testcases in SoapUI opnieuw uitgevoerd om te kijken of het proof of concept naar verwachting reageerde. Daarnaast zijn de functionaliteiten getest met behulp van een Google Chrome extensie genaamd Advanced REST Client. Met behulp van de Advanced REST Client was het mogelijk om een request te verzenden naar het proof of concept en de response goed te analyseren. Een voorbeeld van een request in de Advanced REST Client is te zien in het onderstaande figuur.



Figuur 24: een request met de Advanced REST Client

In het voorbeeld wordt een HTTP GET request verzonden naar <http://localhost:8080/v1/users>. Hier geeft het proof of concept vervolgens een response op. De response is te analyseren in het onderstaande figuur.





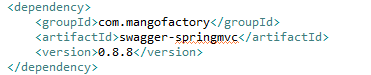
Figuur 25: een responsebody met de Advanced REST Client

De response heeft een HTTP Status Code van 200 OK. Dit betekent dat het proof of concept de request succesvol heeft verwerkt. In dit geval bevat de response ook een body waarin alle users in een JSON-formaat zijn weergegeven.

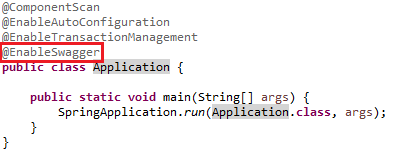
Toen alle functionaliteiten waren toegevoegd en getest met behulp van de testcases in SoapUI en de Advanced REST Client, was het tijd om Swagger te implementeren.

### 9.5.2 Het implementeren van Swagger

Het implementeren van Swagger was relatief eenvoudig. Om gebruik te kunnen maken van alles wat Swagger te bieden heeft, waren er in het proof of concept slechts enkele aanpassingen nodig. Als eerst moest er in de POM een dependency worden toegevoegd:



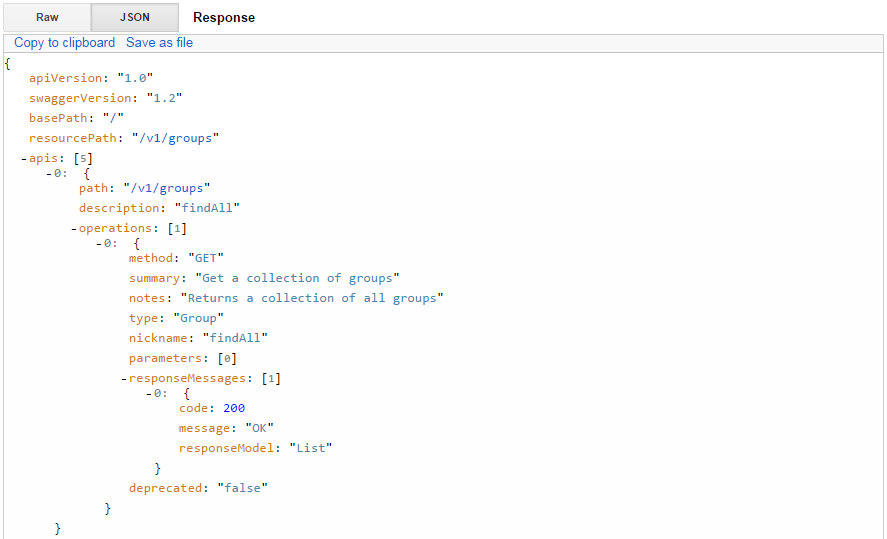
Door het toevoegen van de dependency aan de POM kon ik gebruik maken van de verschillende Swagger annotaties. De Swagger annotaties worden net als in Spring aangeduid met een @. De eerste annotatie die ik aan de main functie van het proof of concept moest toevoegen, was @EnableSwagger. Dit is te zien in het onderstaande figuur.



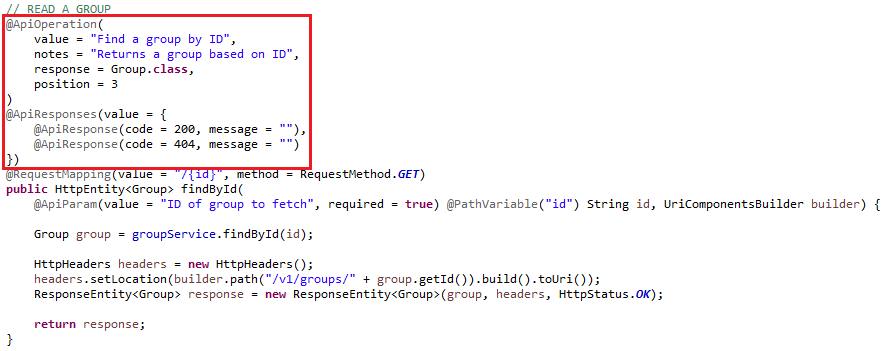
Door het toevoegen van @EnableSwagger werd tijdens het compilen van het proof of concept de broncode door Swagger gescant. Op basis van deze scan werd een documentatie gegenereerd en gedistribueerd op <http://localhost:8080/api-docs>. Wanneer er een HTTP GET request werd gemaakt naar dit adres, bevatte de body van de response de onderstaande inhoud in JSON-formaat.



Vervolgens kon er ook een HTTP GET request gemaakt worden naar <http://localhost:8080/api-docs/v1/groups>. Een deel van het resultaat hiervan was als volgt:



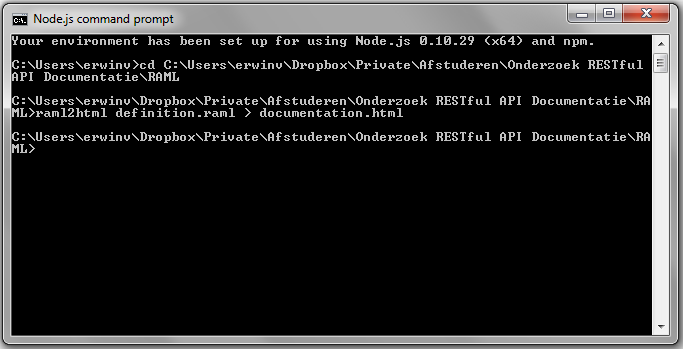
Zoals hierboven te zien is, bevatte de body de mogelijke requests en de bijbehorende responses van de endpoint die je in het adres hebt vermeld. Deze vorm van documentatie was naar eigen voorkeur uit te breiden door in de broncode nog meer gebruik te maken van Swagger annotaties. In het volgende figuur is een voorbeeld gegeven van hoe de Swagger annotaties gebruikt werden.



Figuur 26: een voorbeeld van Swagger annotaties

### 9.5.3 Het genereren van de RAML documentatie

Het genereren van de RAML documentatie heb ik gedaan met de tool RAML to HTML. RAML to HTML draaide op Node.js. Node.js is een softwareplatform waarop men applicaties kan ontwikkelen en draaien. De applicaties, zoals RAML to HTML, zijn geschreven in JavaScript en worden uitgevoerd in de JavaScript-engine van Node.js zelf. De programma’s kunnen worden uitgevoerd door middel van een command-line, zoals te zien is in het onderstaande figuur.



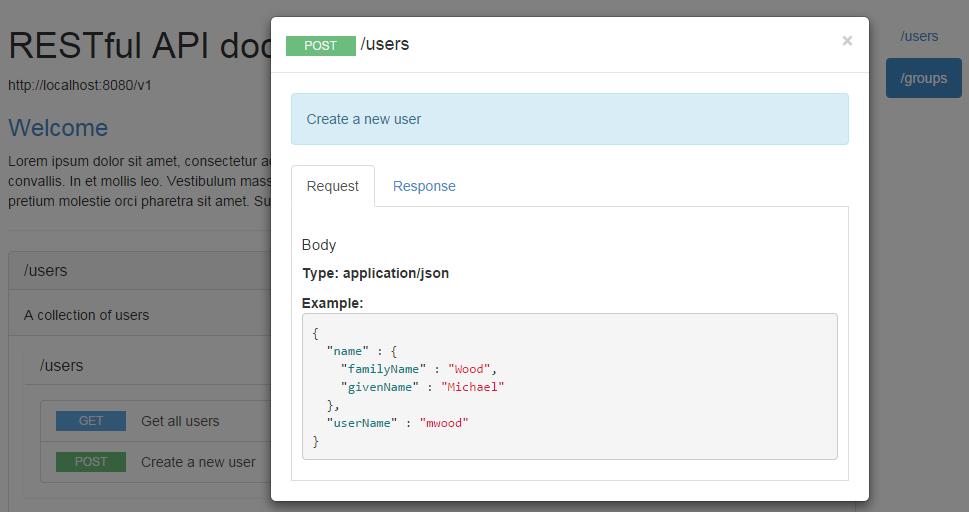
Figuur 27: het uitvoeren van het commando

Voor het omzetten van een RAML definitie naar HTML werd het volgende commando uitgevoerd:

raml2html de\_raml\_definitie.raml > de\_html\_documentatie.html

Vervolgens werd in de desbetreffende map een bestand genaamd *de\_html\_document.html* aangemaakt. Hoe RAML to HTML de RAML definitie van het proof of concept heeft omgezet, is te zien op de volgende bladzijde.





Figuur 28: het resultaat van RAML to HTML

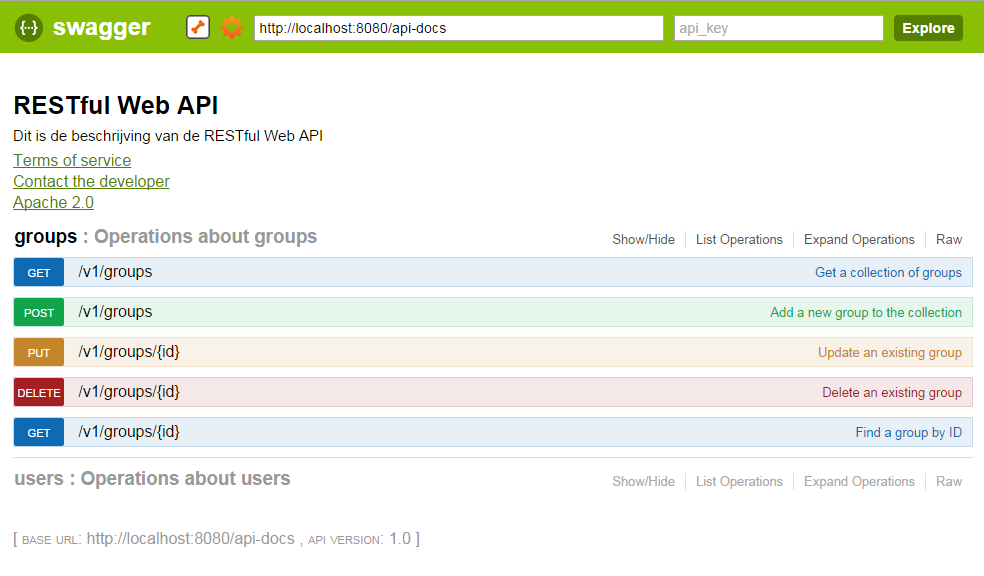
De gegenereerde documentatie zag er erg overzichtelijk en strak uit. Helaas was de tool RAML to HTML nog in volle ontwikkeling, waardoor een aantal definities, die wel in de RAML definitie stonden, niet werden omgezet door de tool. Daarnaast is de documentatie vrij simpel. Hierdoor is de documentatie wel makkelijk te begrijpen, maar ontbreken er een hoop belangrijke aspecten, zoals bijvoorbeeld de response modellen/schema’s. Dit zorgt ervoor dat het voor een klant mogelijk lastig is om te bepalen hoe de modellen er precies uitzien. Het was helaas ook niet mogelijk om de RESTful Web API vanuit de documentatie te testen, waardoor een klant zijn eigen requests moest opstellen met een externe tool zoals bijvoorbeeld de Advanced REST Client. Naast de overzichtelijke visuele weergave levert de tool RAML to HTML niet veel meer dan de RAML definitie zelf. Aangezien de tool nog in ontwikkeling was, zou dit in de toekomst misschien kunnen veranderen.

### 9.5.4 Het genereren van de Swagger documentatie

Zoals in paragraaf 9.2.1 beschreven is, werd de Swagger documentatie gegenereerd met behulp van SwaggerUI. Nadat ik SwaggerUI van GitHub had gehaald, was het slechts een kwestie van het openen van de index pagina van SwaggerUI. Op de index pagina kon ik bovenin aangeven op welk adres Swagger de documentatie had gedistribueerd. Dit is te zien in het onderstaande figuur.

C:\Users\Erwin\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Word\swaggerui_explore.png

Op het moment dat ik op de knop ‘Explore’ drukte, zette SwaggerUI de documentatie in JSON-formaat om naar visuele lijst van alle endpoints en hun mogelijke requests. Het resultaat hiervan is te zien in het volgende figuur.

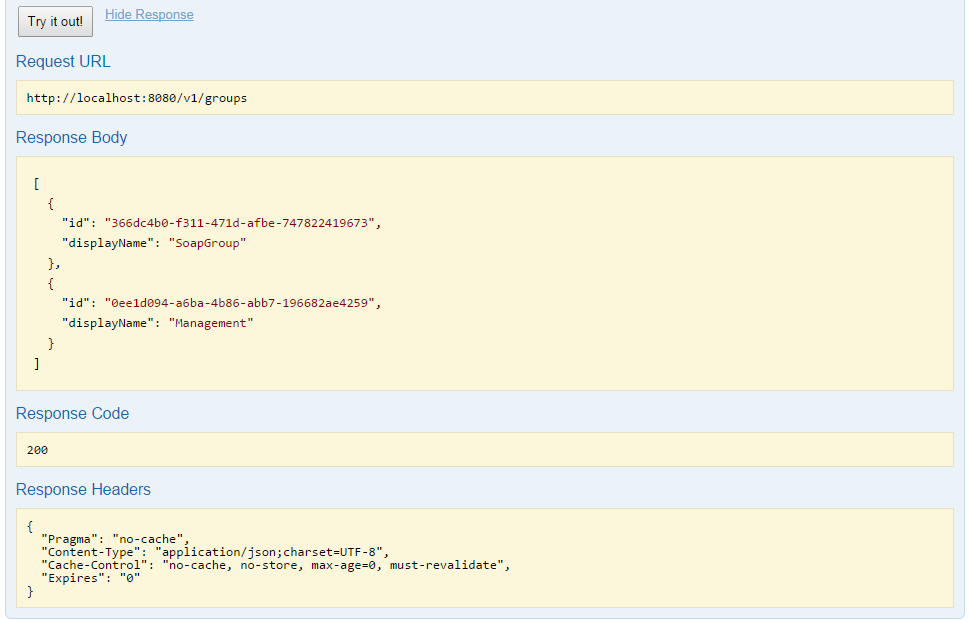


Wanneer ik op een request drukte, klapte de request open en werden een extra beschrijving, het response model/schema en alle mogelijke responses weergegeven (figuur 29).



Figuur 29: een request in SwaggerUI

Zoals in het figuur te zien is, staat er onderaan een knop genaamd ‘Try it out!’. Door op deze knop te drukken, was ik vanuit de documentatie in staat om de request op de RESTful Web API te testen. De response van de request werd vervolgens netjes weergegeven (figuur 30).



Figuur 30: het testen van een request in SwaggerUI

De figuren tonen aan dat SwaggerUI in staat is om een hele rijke documentatie te genereren. De documentatie is daarnaast niet alleen erg overzichtelijk, maar ook heel makkelijk te begrijpen. Hoewel ik SwaggerUI lokaal heb gedraaid, is het ook mogelijk om SwaggerUI op een server te laten draaien. Hierdoor is het ook voor klanten mogelijk om de documentatie van de RESTful Web API te bekijken en de RESTful Web API eenvoudig uit te proberen.

# 10 Advies

In dit hoofdstuk wordt beschreven wat het advies is over de verschillende mogelijkheden voor het documenteren van een RESTful Web API.

Op basis van mijn bevindingen ben ik tot een advies gekomen die passend is bij het bedrijf ENOVATION. Dit wil zeggen dat het advies zo dicht mogelijk bij de eisen en het proces van het bedrijf aansluit.

Zoals in de opdracht staat beschreven, is er een literatuuronderzoek gedaan naar de mogelijkheden voor het documenteren van een RESTful Web API. Twee van de mogelijkheden die naar voren zijn gekomen, zijn vervolgens toegepast op een proof of concept. Deze twee mogelijkheden waren RAML en Swagger. Aangezien RAML gebruik maakt van de top-down strategie, was dit de mogelijkheid die als eerst werd toegepast.

Mijn eerste bevindingen waren dat RAML gebruik maakt van een zeer eenvoudige syntax. Deze syntax was binnen een mum van tijd te begrijpen. Dit zorgt ervoor dat je zelfs als je de syntax niet kent heel snel een RAML definitie kan opstellen. Een ander groot voordeel van RAML is dat de community heel veel open source tools ontwikkelt. Met deze tools is het bijvoorbeeld mogelijk om een mock server op te zetten of de definitie om te zetten naar HTML formaat. Hierbij komen we ook gelijk bij het nadeel van RAML. De tool die gebruikt wordt om de definitie om te zetten naar HTML formaat is nog vol in ontwikkeling. Dit heeft als gevolg dat nog niet alle functionaliteiten die je in een definitie kwijt kunt, worden herkend door de tool. Daarnaast is het HTML formaat dat wordt gegenereerd niet heel erg uitgebreid en is het niet mogelijk om de RESTful Web API vanuit de HTML pagina te testen. Dit zou echter in de toekomst kunnen veranderen, want RAML is nog redelijk nieuw en de beschikbare tools worden steeds beter.

Swagger is een mogelijkheid die gebruik maakt van de bottom-up strategie. Dit betekent dat Swagger pas geïmplementeerd kon worden op het moment dat er al met het bouwen van het proof of concept was begonnen. Dit sluit niet echt aan op de werkwijze van ENOVATION en bovendien zorgt dit ervoor dat er pas in een laat stadium een documentatie van de RESTful Web API beschikbaar is. Swagger was echter wel heel makkelijk en snel te implementeren. Net als RAML was het bij Swagger ook mogelijk om de documentatie om te zetten naar HTML formaat. Om dit te realiseren moest er gebruik worden gemaakt van SwaggerUI. De documentatie die SwaggerUI genereerde was veel uitgebreider en duidelijker dan die van RAML. Daarnaast was het ook mogelijk om de RESTful Web API vanuit de HTML pagina te testen. Helaas beschikt Swagger niet over een reeks van open source tools. Hierdoor word je eigenlijk gedwongen om de werkwijze van Swagger te hanteren en is het moeilijk om daar ook maar deels vanaf te wijken.

De plus- en minpunten van RAML en Swagger staan in de onderstaande tabel nog even kort opgesomd.

|  |  |
| --- | --- |
| **RAML** | **Swagger** |
| + Top-down strategie | + Makkelijk te implementeren |
| + Zeer eenvoudige syntax | + Uitgebreide en mooie documentatie |
| + Veel open source tools beschikbaar | + RESTful Web API vanuit documentatie te testen |
| - Geen uitgebreide documentatie | - Bottom-up strategie |
| - RESTful Web API niet vanuit documentatie te testen | - Niet echt overzichtelijk in de broncode |

RAML is een vrij nieuwe mogelijkheid voor het documenteren van RESTful Web APIs. Daar staat echter wel tegenover dat RAML heel veel gebruik maakt van open source tools. ENOVATION maakt zelf ook heel veel gebruik van open source tools en ziet dit toch als een sterk pluspunt. Aangezien RAML een grote community heeft, verwacht ik dat RAML flink zal groeien en dat de kwaliteit van de beschikbare tools snel zal verbeteren. RAML is een mogelijkheid die direct aansluit op de werkwijze van ENOVATION. Hierdoor kan RAML binnen een handomdraai in gebruik worden genomen en zijn waarde bewijzen. Mijn advies is dus dat ENOVATION in de toekomst moet gaan overwegen om zijn RESTful Web APIs te documenteren met behulp van RAML.

# 11 Evaluatie

In dit hoofdstuk kijk ik terug op de producten die ik heb opgeleverd en de manier waarop deze producten tot stand zijn gekomen.

## 11.1 Productevaluatie

Hoewel ik natuurlijk minder producten heb opgeleverd, omdat ik het tweede onderdeel wegens beperkte tijd volledig heb laten vallen, ben ik erg tevreden over de producten die ik heb opgeleverd voor het eerste onderdeel. De producten die ik heb opgeleverd, zijn het onderzoeksplan, onderzoeksrapport, proof of concept en het adviesrapport. Doordat ik me aan het begin van de opdracht goed georiënteerd heb, was ik in staat om een duidelijk plan te maken. In dit plan heb ik beschreven hoe ik tot een goed eindproduct kon komen, namelijk het adviesrapport.

Het product waar ik iets minder tevreden over ben, is het onderzoeksrapport en dan voornamelijk de uitwerking van het literatuuronderzoek. Tijdens deze opdracht ben ik erachter gekomen dat ik het erg moeilijk vind om goed onderzoek te doen. Hierdoor heb ik het gevoel dat ik het literatuuronderzoek wat beter uit had kunnen werken. Daarnaast ben ik vergeten om het interview op te nemen, waardoor ik het interview niet zo goed kon uitwerken als ik wilde. Ik ben echter wel van mening dat de uitkomst ondanks de wat mindere uitwerking van het literatuuronderzoek hetzelfde geweest zou zijn.

Het product waar ik dan wel zeer tevreden over ben, is het proof of concept. Het proof of concept was voor deze opdracht heel erg belangrijk, omdat op basis van het proof of concept een goed advies gegeven kon worden. Ik ben in staat geweest om een complex proof of concept te ontwikkelen op de manier waarop ENOVATION dat ook doet. Hierdoor hadden mijn bevindingen tijdens het ontwikkelen van het proof of concept veel waarde en kon ik een advies geven waar ENOVATION ook daadwerkelijk iets mee kon. Het ontwikkelen van het proof of concept en het toepassen van RAML en Swagger heeft voor een echte uitdaging gezorgd.

## 11.2 Procesevaluatie

Ik ben over het algemeen zeer tevreden over het proces dat ik tijdens deze opdracht heb doorlopen. Het is jammer dat ik het tweede onderdeel heb moeten laten vallen, maar hierdoor was ik wel in staat om heel duidelijk proces op te stellen voor het eerste onderdeel. Gedurende het uitvoeren van deze opdracht heb ik ook erg veel contact gehad met de opdrachtgever. Hierdoor waren we allebei op de hoogte van de stand van zaken en konden we samen oplossingen bedenken voor eventuele problemen. Ik vind het jammer dat ik Scrum heb niet kunnen toepassen gedurende deze opdracht, omdat ik niet in team werkte. Ik denk dat ik ook een hoop had kunnen leren van collega’s.

Ik ben aangenaam verrast over hoe prettig ik het ontwikkelproces vond. Het toepassen van Test Driven Development bij het ontwikkelen van het proof of concept heeft mij heel erg goed geholpen bij het hanteren van het proces. Met behulp van TDD kreeg ik tijdens het ontwikkelen steeds feedback op de dingen die ik had gedaan. Hierdoor kon ik telkens met een goed gevoel door met de volgende stap.

Wanneer ik in de toekomst nog een soortgelijke opdracht zou moeten uitvoeren, zal ik zeker overwegen om het op precies dezelfde manier aan te pakken zoals ik nu heb gedaan. Het werken in een team zal ik overigens wel sneller de voorkeur geven.

## 11.3 Verantwoording competenties

Voordat ik begon met het uitvoeren van deze opdracht is er een aantal competenties opgesteld waaraan gedurende de opdracht voldaan moest gaan worden. Hieronder worden deze competenties nogmaals besproken en geef ik aan waarom ik aan de competenties voldaan heb.

### 11.3.1 Selecteren methoden, technieken en tools

Deze competentie omvat het selecteren van methoden, technieken en tools en het opzetten van een ontwikkel-, beheer-, test- en productieomgeving.

Tijdens het uitvoeren van deze opdracht heb ik verschillende methoden, technieken en tools geselecteerd. Allereerst heb ik ervoor gekozen om een literatuuronderzoek te doen en een interview te houden. Met behulp van deze twee methoden ben ik in staat geweest om tot een selectie van mogelijkheden voor het documenteren van een RESTful Web API te komen. Twee van deze mogelijkheden zijn vervolgens toegepast tijdens het ontwikkelen van het proof of concept. De methoden, technieken en tools die ik geselecteerd heb voor het ontwikkelen van een proof of concept komen voort uit het literatuuronderzoek en het interview.

Voor het ontwikkelen van het proof of concept heb ik ook verschillende omgevingen opgezet. Als eerst heb ik met behulp van SoapUI een testomgeving opgezet. Vervolgens heb ik de ontwikkelomgeving opgezet en alle benodigde tools en technieken onderdeel uit laten maken van deze omgeving. De tools en technieken die nodig waren voor het ontwikkelen van het proof of concept waren bijvoorbeeld een MySQL Server, Spring Tool Suite en Apache Maven.

### 11.3.2 Ontwerpen, bouwen en bevragen van een database

Deze competentie omvat het transformeren van een gegevensmodel naar een vorm die geïmplementeerd kan worden in een database en het bevragen van een database.

Nadat ik het gegevensmodel had ontwerpen en de lokale MySQL Server had opgezet, heb ik op basis van het gegevensmodel een Relationeel Implementatie Model (RIM) gemaakt. Dit model was vervolgens te implementeren in het databasemanagementsysteem (DBMS). Het proof of concept kon de database vervolgens bevragen door middel van de SQL queries die had opgesteld. Tijdens het bevragen van de database werden er altijd meerdere tabellen geraadpleegd.

### 11.3.3 Ontwerpen systeemdelen

Deze competentie omvat het zodanig beschrijven van de systeemdelen zodat het bouwen van het systeemdeel mogelijk is.

Aangezien het bouwen van een proof of concept niet het doel van deze opdracht was, heb ik het ontwerp van het proof of concept uiteindelijk gebaseerd op een standaard, namelijk SCIM. Hierdoor is het ontwerpen van de systeemdelen minder groot uitgepakt dan ik van tevoren had gedacht. Daarentegen heb ik echter wel het gedrag van de verschillende systeemdelen beschreven. Tijdens het ontwerpen van de systeemdelen heb ik ook goed gezorgd voor een duidelijke scheiding tussen de lagen.

### 11.3.4 Bouwen van systeemdelen

Deze competentie omvat het bouwen en documenteren van de systeemdelen en het samenstellen van de systeemdelen tot een werkende applicatie.

Tijdens het bouwen van de systeemdelen heb ik gebruik gemaakt van de opgezette ontwikkelomgeving. Spring Tool Suite was onderdeel van deze ontwikkelomgeving en werd gebruikt voor het ontwikkelen van de systeemdelen. De systeemdelen zijn ontwikkeld met behulp van de programmeertaal Java en het Spring Framework. Gedurende de ontwikkeling van de systeemdelen heb ik goed rekening gehouden met de complexiteit, testbaarheid en hergebruik van de systeemdelen. Alle complexe systeemdelen heb ik voorzien van een goede en duidelijke documentatie.

### 11.3.5 Uitvoeren van en rapporteren over het testproces

Deze competentie omvat het opstellen van een testscript en het uitvoeren van het testplan.

Gedurende de ontwikkeling van het proof of concept heb ik gebruik gemaakt van Test Driven Development. Voordat ik begon met het bouwen van het proof of concept heb ik verschillende testcases opgesteld die de kwaliteit van het proof of concept zouden waarborgen. De testcases en een mock server zijn vervolgens opgesteld in het programma SoapUI. De testcases hebben hoofdzakelijk de functionaliteit van het proof of concept getest en de rapportage betrof het volledige systeem.

# Bijlage A: Literatuurlijst

Building a RESTful Web Service - Spring.io. Geraadpleegd op <http://spring.io/guides/gs/rest-service/>

Fielding, R.T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Geraadpleegd op <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>

Hypertext Transfer Protocol -- HTTP/1.1. Geraadpleegd op <http://www.w3.org/Protocols/rfc2616/rfc2616.html>

Kasarkod, J. (2013, 5 november). How do you document your APIs? Geraadpleegd op <http://www.infoq.com/research/api-documentation>

Lane, K. (2012, 27 juni). Flexible and Collaborative API Design and Documentation with Apiary.io | ProgrammableWeb. Geraadpleegd op <http://www.programmableweb.com/news/flexible-and-collaborative-api-design-and-documentation-apiary.io/2012/06/27>

Lensmar, O. L. E. (2014, 27 maart). Another API-Blueprint, RAML and Swagger Comparison. Geraadpleegd op <http://www.slideshare.net/SmartBear_Software/api-strat-2014metadataformatsshort>

Mead, N. (2006, 26 september). Requirements Prioritization Introduction. Geraadpleegd op <https://buildsecurityin.us-cert.gov/articles/best-practices/requirements-engineering/requirements-prioritization-introduction>

Mombrea, M. (2014, 28 februari). API documentation made beautiful with Apiary.io | ITworld. Geraadpleegd op <http://www.itworld.com/development/407333/api-documentation-made-beautiful-apiaryio>

RAML – RESTful API modeling language. Geraadpleegd op <http://raml.org/docs.html>

Rest Api Documentation | Don’t Make The Same Mistake Twice [Blog post]. (2012, 6 augustus). Geraadpleegd op [http://mestachs.wordpress.com/2012/08/06/rest-api-documentation/](http://mestachs.wordpress.com/2012/08/06/rest-api-documentation/%20)

Richardson, L., & Ruby, S. (2007). *RESTful Web Services*. California, United States of America: O'Reilly Media.

SCIM: System for Cross-domain Identity Management (2014). Geraadpleegd op <http://www.simplecloud.info/>

Swagger: A simple, open standard for describing REST APIs with JSON. (2014). Geraadpleegd op <https://helloreverb.com/developers/swagger>

Tools to generate beautiful web api documentation [Blog post]. (2013, 15 februari).  
Geraadpleegd op <http://www.mattsilverman.com/2013/02/tools-to-generate-beautiful-api-documentation.html>

Van de Woude, Y. (2013, 18 mei). Rest Api Documentation | Don't Make The Same Mistake Twice [Blog reactie]. Geraadpleegd op [http://mestachs.wordpress.com/2012/08/06/rest-api-documentation/#comment-2638](http://mestachs.wordpress.com/2012/08/06/rest-api-documentation/%23comment-2638)

When Is It OK Not To TDD? [Illustratie]. (2010, 1 juli). Geraadpleegd op <http://gamesfromwithin.com/wp-content/uploads/2010/07/tdd_cycle.jpg>

Wikipedia. (2005, 21 februari). Comparison of documentation generators. Geraadpleegd op <http://en.wikipedia.org/wiki/Comparison_of_documentation_generators>

# Bijlage B: Afkortingenlijst

|  |  |
| --- | --- |
| **Afkorting** | **Omschrijving** |
| API | Application Programming Interface |
| DBMS | Databasemanagementsysteem |
| GUI | Graphical User Interface |
| JSON | JavaScript Object Notation |
| POC | Proof of concept |
| RAML | RESTful API Modeling Language |
| REST | Representational State Transfer |
| RFC | Request for Comments |
| SCIM | System for Cross-domain Identity Management |
| STS | Spring Tool Suite |
| TDD | Test Driven Development |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier |
| XML | Extensible Markup Language |
| YAML | YAML Ain’t Markup Language |

# Bijlage C: Het interview

**Welk proces hanteert ENOVATION bij het ontwikkelen van zijn producten?**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

**ENOVATION hanteert enkele technieken bij het ontwikkelen van zijn RESTful Web APIs. Welke technieken zijn dit precies?**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

**Ik neem aan dat de documentatie van de RESTful Web APIs moet aansluiten bij deze technieken. Waar moet de documentatie naar jullie mening nog meer aan voldoen?**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

**Wat is de verdeling als we de eisen gaan prioriteren met behulp van de 100 punten methode?**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |