Professionele Bachelor Toegepaste Informatica

Dotoforfelix

Thomas Galicia

Promotoren:

Serge Liberloo
Arno Barzan

Eppix
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Dotoforfelix

Thomas Galicia

Promotoren:

Serge Liberloo
Arno Barzan

Eppix
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Via deze weg wil ik graag Serge Liberloo en Gilian Hoskens willen bedanken voor de leuke samenwerking. Doorheen de stage hebben beide heren gezorgd voor een aangename en leerrijke werksfeer. Ze hebben beide hun kennis en ervaring gedeeld met mij. Binnen Eppix hebben ze mij goed ondersteund en begeleid, hierdoor heb ik veel nieuwe kennis opgedaan. Verder heeft dit ertoe geleid dat ik uit mijn fouten heb kunnen leren.

Ten slotte wil ik mijn hogeschoolpromotor Arno Barzan en taallector Kim Sleurs bedanken voor alle feedback en opvolging. De feedback was ten alle tijden zeer constructief en leerrijk.

Abstract

De stage is uitgevoerd bij Eppix. Het bedrijf biedt onder andere backend-ontwikkeling aan voor systemen in het Java Spring-framework. Deze stage bij Eppix is onderverdeeld in een stageopdracht en een onderzoeksonderwerp. De stageopdracht heeft als doel een bestaande statische website om te vormen naar een webapplicatie die extra functionaliteiten aanbiedt aan de eindgebruikers.

De website in kwestie is bedoeld voor Dotoforfelix. Dit is een groep wandelaars die jaarlijks een sponsortocht organiseert ten voordele van het Felix Project. Het Felix Project is een organisatie die zich in zet voor zwerfkatten in België. Dotoforfelix wandelt jaarlijks de Dodentocht in Bornem, de opbrengsten van deze tocht gaan integraal naar het Felix Project. De huidige website is heel statisch en de meeste processen moeten handmatig uitgevoerd worden door de beheerders.

Via de nieuwe webapplicatie kunnen deelnemers zelf hun deelname registreren en beheren. Dit zorgt ervoor dat een deel van de processen niet meer door de beheerders moet uitgevoerd worden. Verder zijn er ook extra functionaliteiten toegevoegd voor het kerncomité en de administrators zodat zij de verschillende sponsortochten kunnen aanmaken en beheren.

Dit systeem is mogelijk gemaakt door een webapplicatie geschreven in het Angular-framework die communiceert met een RESTful API geschreven in het Java Spring-framework.

Het onderzoeksonderwerp van deze stage is autorisatie aan de hand van de OAuth-standaard. Er wordt nagegaan wat OAuth is en hoe de standaard in elkaar zit. Verder wordt er ingegaan op hoe deze standaard op een generieke manier geïmplementeerd kan worden in een aparte autorisatie-server, zodat deze server gebruikt kan worden in verschillende *microservice* omgevingen of applicaties.

Ten slotte is deze generieke autorisatie-server gebruikt om de toegang tot de RESTful API van de webapplicatie af te schermen zodat deze enkel beschikbaar is voor gebruikers met de juiste toegangsrechten.

Inhoudsopgave

Lijst van gebruikte figuren

Lijst van gebruikte afkortingen

1. XP	Extreme programming
2. DevOps	Development and Operations

Inleiding

Het Felix project is een organisatie die “zich inzet om het aantal zwerfkatten op een efficiënte manier tot een minimum te beperken. Zij proberen dit doel te bereiken door onder meer sterilisaties en castraties uit te voeren op zwerfkatten zodat het aantal zwerfkatten tot een minimaal beperkt blijft” [1].

De kosten van deze processen kunnen hoog oplopen, dus organiseert Dotoforfelix jaarlijks een sponsortocht om een deel van de kosten te dekken. Deze sponsortocht is het wandelen van de jaarlijkse Dodentocht [2] in naam van Dotoforfelix.

Via de website dotoforfelix.be kunnen mensen zich registreren en vervolgens zichzelf opgeven als wandelaar of volger. Het doel van deze sponsortocht is dat andere mensen de wandelaars of volgers kunnen sponsoren, bij wandelaars gaat het om een zelfgekozen bedrag per gewandelde kilometer. De uiteindelijke opbrengst van deze sponsortocht gaat integraal naar het Felix project.

I. Stageverslag

1 Bedrijfsvoorstelling

1.1 Eppix

Eppix is een bedrijf dat oorspronkelijk begonnen is met een focus op Java en iOS-ontwikkeling, maar vandaag de dag zijn ze uitgebreid naar een volwaardig software ontwikkelingsbedrijf en bieden ze ook Angular aan. Verder bieden ze consultancy aan om bedrijven verder te helpen bij het opzetten van een *DevOps* en/of *deployment* omgeving. [3]

Het doel van het bedrijf is om andere bedrijven te helpen met het ontwikkelen van een systeem of applicatie zodat het oorspronkelijke bedrijf de focus kan leggen op het eindproduct en zich geen zorgen moet maken over een bepaald onderdeel van dit eindproduct.

Binnen het bedrijf hechten ze belang aan openheid, eerlijkheid, respect en het leerproces. Ze nemen zelf ook de tijd om nieuwe technologieën bij te leren zodat ze deze ook aan klanten kunnen aanbieden. Ten slotte hanteren ze binnen Eppix moderne methodologieën zoals scrum, agile en XP.

2 Stageopdracht

2.1 De probleemstelling

De huidige website van Dotoforfelix geeft de nodige informatie weer, maar bevat weinig functionaliteiten voor de eindgebruikers. Het grootste deel van de processen gebeurt door de nodige data door te mailen naar de administrators, zij zullen dan op een manuele manier deze data verwerken.

Verder hebben administrators ook extra functionaliteiten nodig zoals bijvoorbeeld het promoveren van een gewone gebruiker naar de administrator rol.

2.2 Doelstelling

Deze stageopdracht bestaat uit verschillende delen. Ten eerste is er het herwerken van de Dotoforfelix website, deze wordt omgezet naar een Angular webapplicatie. Deze applicatie biedt de oorspronkelijke – en nieuwe functionaliteiten aan.

Verder maakt deze applicatie gebruik van een RESTful API om data op te vragen en aan te passen. Bovendien is deze API afgeschermd zodat de data en operaties enkel toegankelijk is voor gebruikers met specifieke permissies. Dit is verwezenlijkt aan de hand van een autorisatie-server die ervoor zorgt dat gebruikers enkel toegang krijgen tot de data of operaties die voor hun bedoeld is.

2.3 Componenten

Dit hoofdstuk bevat een uitgebreidere uitleg van alle verschillende onderdelen die tijdens de stageopdracht aan bod komen.

2.3.1 Webapplicatie

De webapplicatie maakt gebruik van het Angular-framework. De webapplicatie bevat dezelfde functionaliteiten als de oorspronkelijke Dotoforfelix website; maar bepaalde manuele acties die vooraf via e-mail uitgevoerd werden kunnen nu via de webapplicatie gerealiseerd worden. Ten slotte is de webapplicatie uitgebreid met nieuwe functionaliteiten die bedoeld zijn voor de administrators zodat zij het jaarlijkse Dodentocht evenement en de gebruikers kunnen beheren.

2.3.2 Dotoforfelix RESTful API

De RESTful API zorgt voor het aanbieden van data aan de webapplicatie. Verder zorgt deze ervoor dat vanuit de webapplicatie data aangepast of gecreëerd kan worden. De toegang tot deze API is afgeschermd zodat bepaalde data of operaties enkel toegankelijk zijn voor gebruikers met de correcte gebruikersrollen. Op deze manier wordt ervoor gezorgd dat bepaalde operaties enkel door bijvoorbeeld administrators uitgevoerd kunnen worden.

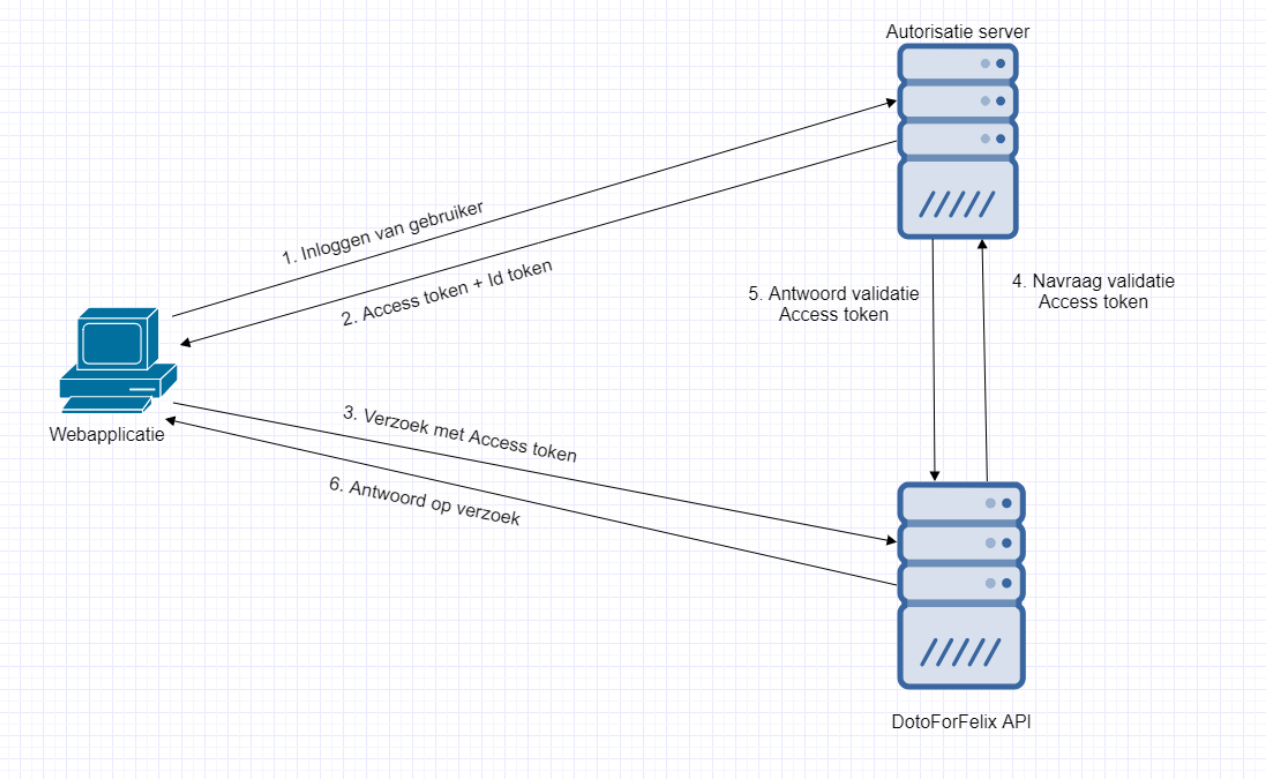
2.3.3 Autorisatie-server

De autorisatie-server staat in voor het valideren van de inloggegevens van eindgebruikers. Verder dient deze server voor het aanmaken en valideren van toegangs*token*s. De Dotoforfelix API gebruikt dit toegangs*token* om te bepalen of een gebruiker toegang heeft tot bepaalde data of bepaalde operaties. Deze autorisatie-server is generiek, zodat hij gebruikt kan worden in andere omgevingen/applicaties. Zowel de autorisatie-server als de Dotoforfelix API is geschreven in Java met behulp van het Spring Boot-framework.

2.4 Implementatie

2.4.1 Algemeen

Het hele systeem bestaat uit drie verschillende componenten die met elkaar communiceren. De webapplicatie communiceert zowel met de autorisatie-server als de Dotoforfelix API omdat deze gescheiden zijn van elkaar. De onderstaande afbeelding illustreert de algemene flow van hoe de verschillende componenten met elkaar communiceren wanneer een gebruiker data probeert op te vragen.



Figuur 1 Algemene flow bij aanvraag data aan de Dotoforfelix API

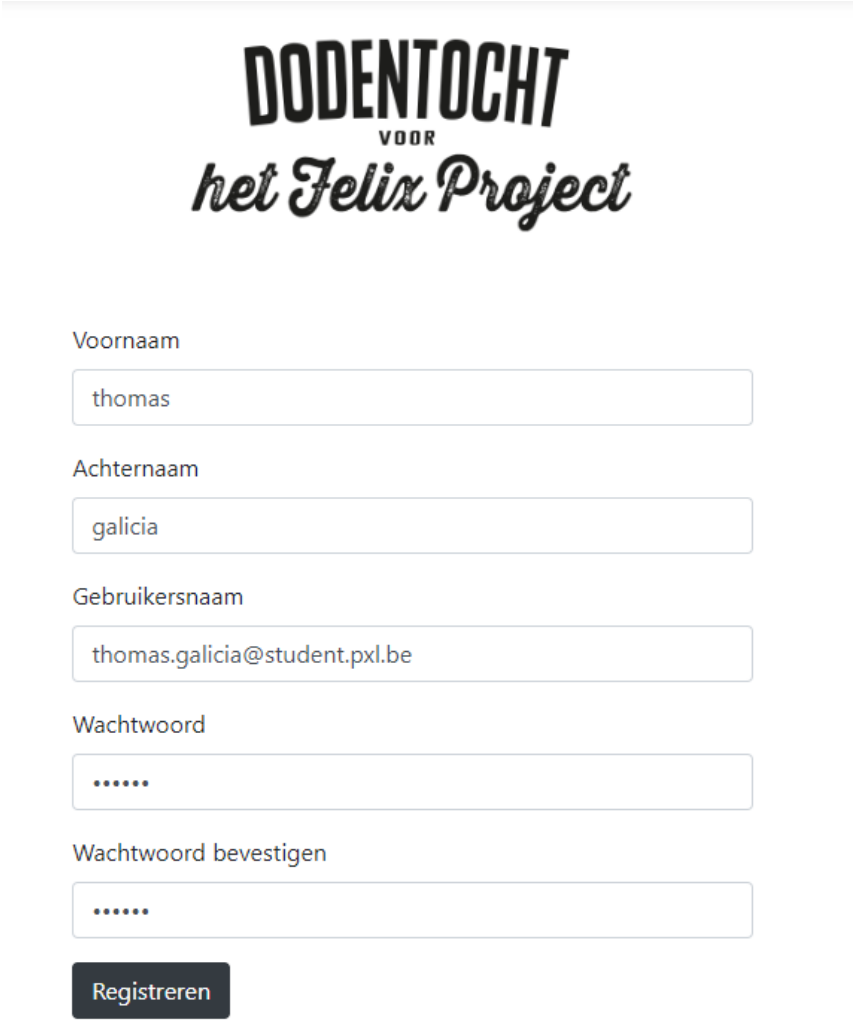
2.4.2 Dotoforfelix webapplicatie & API

De webapplicatie bestaat uit twee delen, enerzijds is er het algemene deel waar elke gebruiker toegang tot heeft, dit bevat de algemene informatie zoals bijvoorbeeld wie de organisatie is en welke wandelaars of volgers er meedoen aan de huidige editie. Anderzijds is er het dashboard dat enkel toegankelijk is voor ingelogde gebruikers, hier kan de eindgebruiker verschillende zaken bekijken zoals zijn eigen deelname van alle jaren. Het dashboard wordt voor leden van het kerncomité en administrators ook verder uitgebreid met extra functionaliteiten zodat de verschillende edities aangemaakt en beheerd kunnen worden. Deze applicatie communiceert met de Dotoforfelix API om data op te vragen, bewerken of toe te voegen.

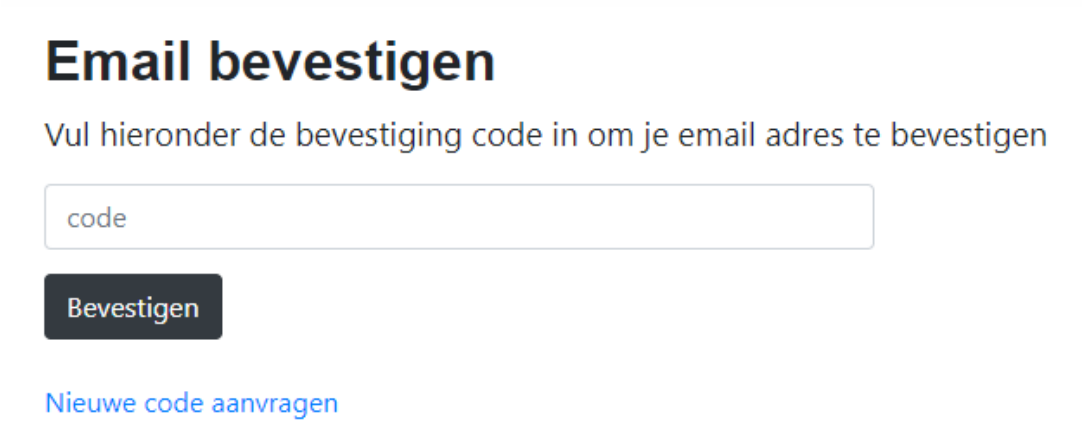
2.4.2.1 Registratie

Het registratieproces bestaat uit drie stappen. Ten eerste moet de eindegebruiker zijn algemene gegevens invullen (naam, voornaam, e-mail, wachtwoord). Deze gegevens worden vervolgens opgeslagen op de autorisatie-server. Indien de gebruiker succesvol geregistreerd is op deze server, zal hij/zij naar de volgende stap omgeleid worden. Op dit scherm moet de gebruiker een code invullen die via e-mail ontvangen is. Indien de code geldig is, wordt de gebruiker verder omgeleid naar de laatste stap van het registratie proces. Als de code ongeldig is, kan de gebruiker een nieuwe code aanvragen. Via deze stap bevestigd de gebruiker zijn e-mailadres.

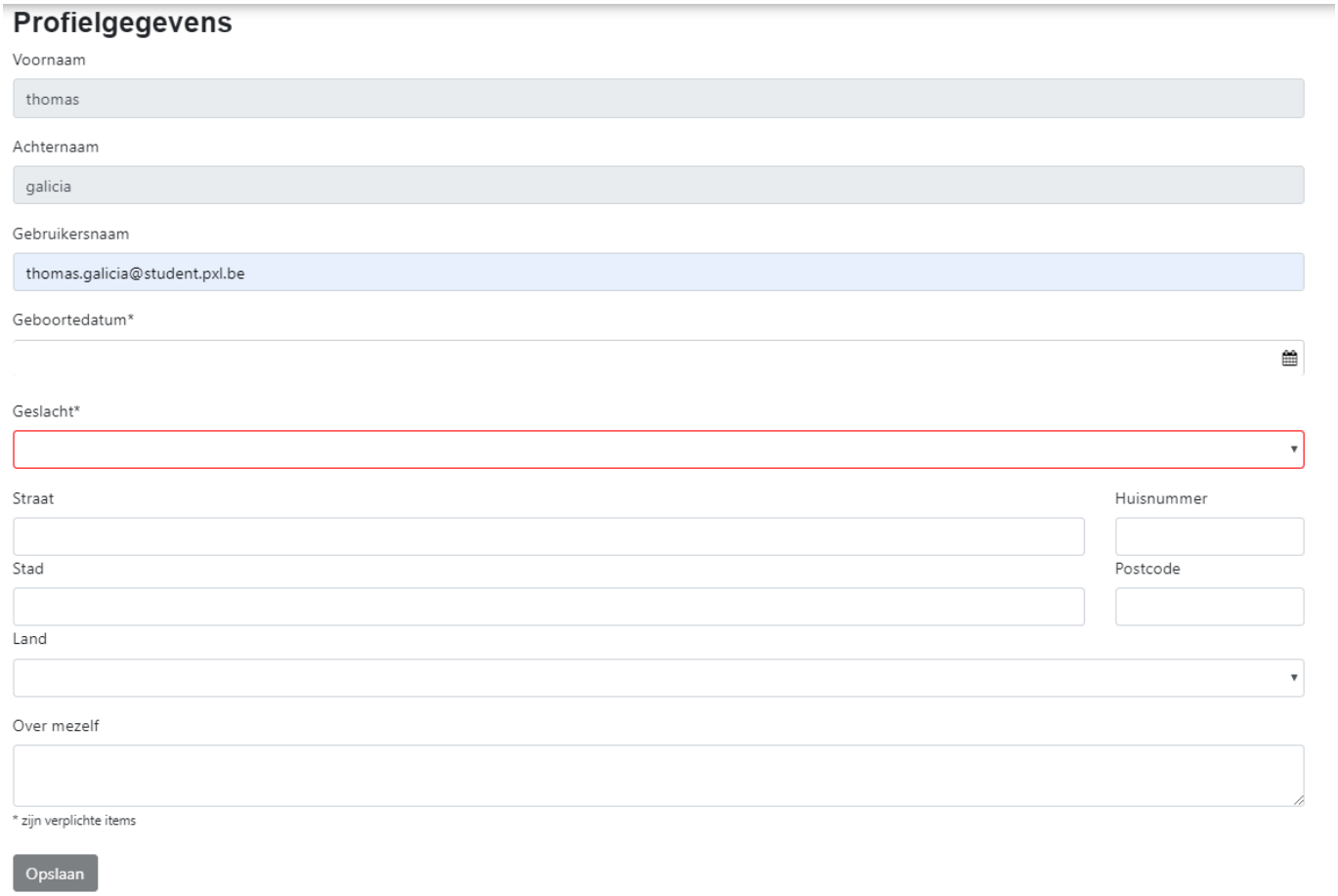
Ten slotte is er de laatste stap van het registratie proces, hierbij gaat de eindgebruiker zijn basisgegevens aanvullen met extra informatie zoals bijvoorbeeld adres, geboortedatum en geslacht; ook heeft de gebruiker hier de mogelijkheid om een profielfoto te uploaden en eventueel bij te knippen indien deze foto te groot is. Het registratie proces bestaat uit twee stappen waar de eindgebruiker persoonlijke informatie moet invullen, het eerste deel van de informatie dient voor de autorisatie-server. Het tweede deel is bedoeld voor de Dotoforfelix API. In de onderstaande afbeeldingen (figuren 2,3,4) ziet u de verschillende fases van het registratieproces in volgorde.



Figuur 2 Registratie deel 1, het invullen van basisgegevens van de nieuwe gebruiker



Figuur 3 Registratie deel 2, bevestigen van e-mailadres m.b.v. de ontvangen code in mailbox



Figuur 4 Registratie deel 3, aanvullen van gebruikersgegevens met extra informatie

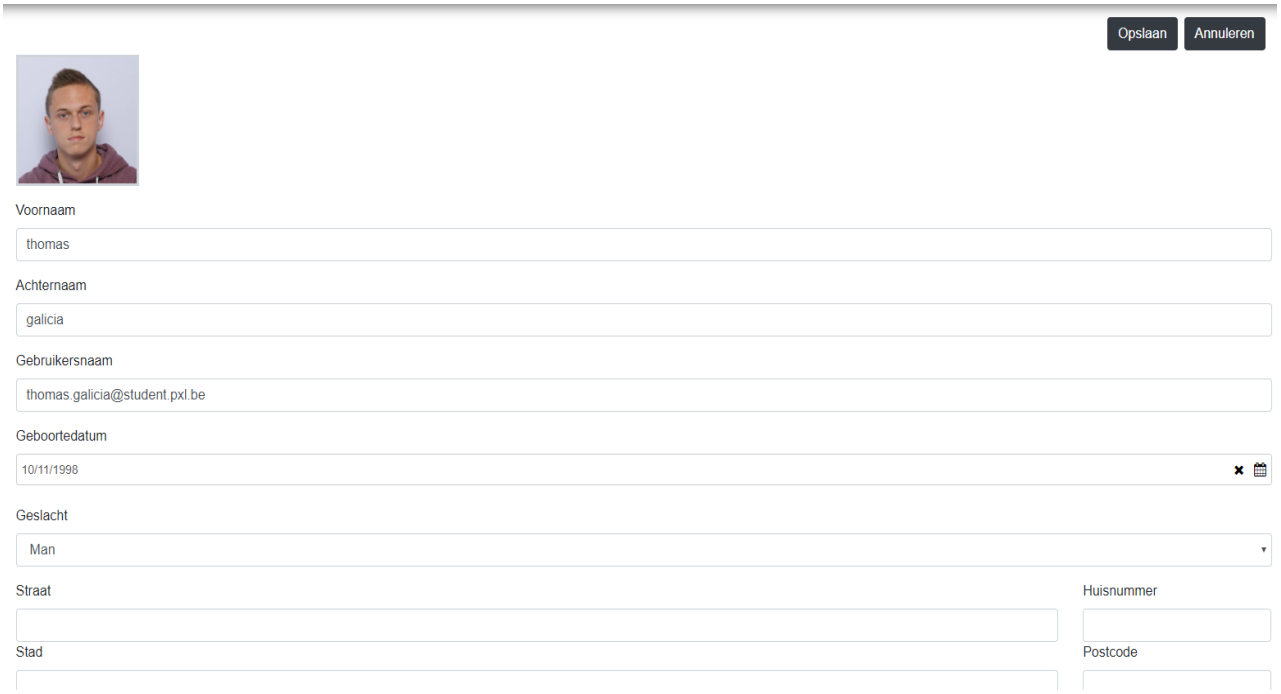
2.4.2.2 Inloggen

Indien de gebruiker al een account heeft, kan hij/zij inloggen doormiddel van zijn e-mailadres en wachtwoord, zoals figuur 5 hieronder aantoont. Indien de gebruiker nog geen account heeft, kan onderaan het scherm gekozen worden om zichzelf te registreren. Wanneer een gebruiker probeert aan te melden en hij/zij stap 2 of 3 van het registratie proces overgeslagen heeft, zal de gebruiker niet kunnen inloggen. In de plaats wordt hij/zij omgeleid naar de stap die overgeslagen was. Zo kan geen enkele stap van het registratie proces omzeild worden.

Figuur 5 Loginscherm voor bestaande gebruikers

2.4.2.3 Profiel beheren

Via de profiel pagina kan een gebruiker zijn eigen profielinformatie bekijken. Indien hij/zij deze gegevens wil aanpassen is dit mogelijk via de “Wijzig” knop boven- of onderaan de pagina. Wanneer een gebruiker in de “Wijzig” modus zit, zal hij/zij de mogelijkheid krijgen om zijn/haar gegevens aan te passen. Ook de profielfoto kan in “Wijzig” modus veranderd worden. Indien de foto te groot is kan deze bijgeknipt worden in de applicatie, dit wordt gevisualiseerd in figuur 7. Ten slotte heeft de gebruiker de mogelijkheid om zijn/haar wijzigingen op te slaan of te annuleren.



Figuur 6 Profiel pagina in "wijzig" modus

Figuur 7 Image cropper om profiel foto bij te snijden

2.4.2.4 Bedrijven

De applicatie bevat ook de mogelijkheid om een bedrijf te koppelen aan een gebruiker. Indien gewenst kan een gebruiker sponsoren in naam van zijn/haar bedrijf. Achteraf kunnen de bedrijfsgegevens gebruikt worden om een btw-attest aan te vragen. Het toevoegen en bijwerken van bedrijfsgegevens is heel gelijkaardig aan de profielgegevens. Ten slotte is er de mogelijkheid voor gebruikers om hun bedrijf te verwijderen uit de applicatie.

2.4.2.5 Dashboard

Eenmaal een gebruiker succesvol is ingelogd, kan hij/zij navigeren naar het dashboard. Op dit dashboard bevinden zich de meeste functionaliteiten van de applicatie. Indien een niet ingelogde bezoeker probeert te navigeren naar het dashboard, zal de bezoeker omgeleid worden naar de inlog pagina. Het dashboard bestaat uit verschillende tabbladen; deze tabbladen zijn afhankelijk van de rol van de gebruiker zichtbaar of niet. De administrator zal toegang hebben tot meer tabbladen in vergelijking met een gewone gebruiker. Bij bepaalde tabbladen bevindt er zich aan de linkerkant van het dashboard een item dat gebruikt kan worden om een jaartal te selecteren.

2.4.2.6 Evenement creëren

Als administrator is het mogelijk om een evenement te creëren via het dashboard. Het is wel enkel mogelijk om het evenement van dit jaar of volgend jaar te creëren. Dit laat toe om het evenement van volgend jaar op voorhand al te creëren. Het maken van een nieuw evenement gaat enkel wanneer alle vorige evenementen gesloten zijn. Het kan niet voorkomen dat er twee actieve evenementen zijn. Bij het aanmaken van een nieuw evenement is de administrator verplicht om drie datums in te vullen nl. de datum van het evenement, de datum waarop de registratie van de wandelaars eindigt en de datum waarop de registratie van de volgers eindigt. Verder heeft de administrator de mogelijkheid om het vast sponsorbedrag voor volgers in te stellen. Indien er een fout optreedt zal er telkens een gepaste foutmelding gegeneerd worden.

2.4.2.7 Evenement beheren

Het tabblad “Events” biedt een overzicht van de verschillende evenementen, dit tabblad is enkel toegankelijk voor de leden van het kerncomité en de administrators. In dit tabblad krijgen ze een overzicht van het evenement van het geselecteerde jaartal. Deze pagina heeft als hoofddoel rapportage, want het bevat alle hoofdinformatie over het evenement (aantal wandelaars, aantal volgers, datums en sponsorbedragen). Voor leden van het kerncomité biedt deze pagina een overzicht van het geselecteerde evenement.

Administrators hebben nog bijkomende functionaliteiten, zij kunnen het evenement sluiten indien het evenement open staat en de gewandelde afstanden van alle wandelaars ingevuld zijn. Bij het sluiten van een evenement worden er automatisch e-mails verstuurd naar alle sponsors. In deze e-mail bevindt zich een overzicht met alle deelnemers die zij gesponsord hebben. Verder worden deze e-mails ook verstuurd naar de administrators zodat zij op de hoogte gebracht worden van alle sponsors. Ten slotte hebben de administrators buiten het sluiten van een evenement ook de mogelijkheid om het openstaande evenement aan te passen, hier kunnen zij de drie datums van het evenement bewerken (datum van de wandeltocht, einddatum registratie van wandelaars en einddatum registratie van volgers). Verder kunnen zij aansluitend het vast sponsorbedrag voor volgers aanpassen, dit is wel enkel mogelijk wanneer nog geen enkele volger gesponsord is voor deze editie. De onderstaande afbeelding (figuur 8) weergeeft het tabblad vanuit het perspectief van een administrator.

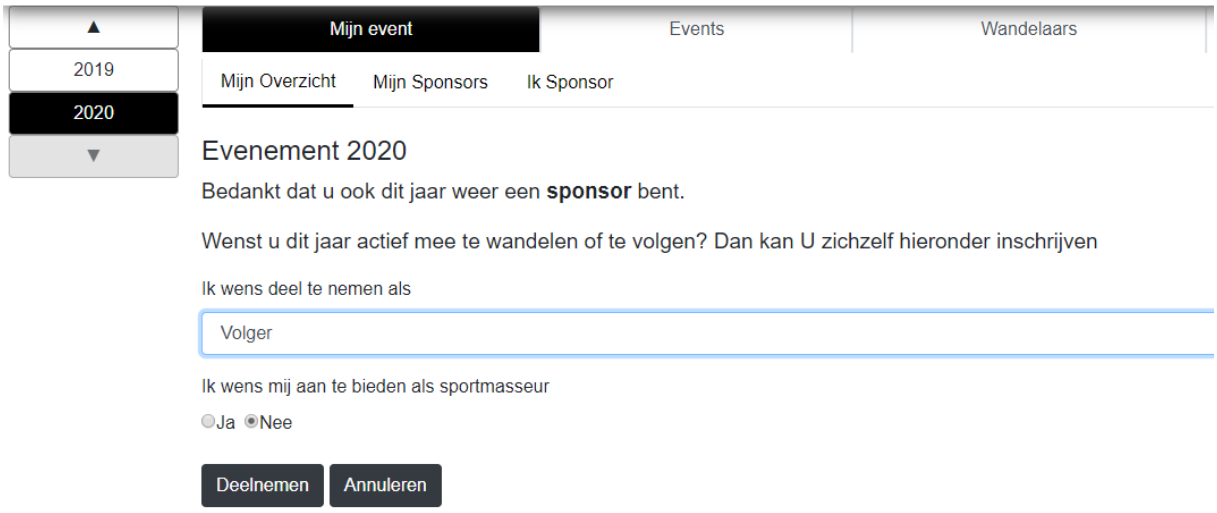
Figuur 8 Events tabblad vanuit administrator perspectief

2.4.2.8 Deelname registreren

Doormiddel van het sub tabblad “Mijn overzicht” in het tabblad “Mijn event” kan een gebruiker in combinatie met de jaarselector, aan de linkerkant van het dashboard, zijn/haar deelname bekijken van het geselecteerde jaartal. Indien de gebruiker het huidige evenementjaar selecteert en hij/zij niet ingeschreven is voor dit evenement, krijgt de gebruiker de mogelijkheid om zijn/haar deelname te registreren zoals aangetoond wordt in figuur 9 en 10.

Tijdens de registratie dient de gebruiker als eerste aan te geven of hij/zij wenst deel te nemen als wandelaar of volger. Vervolgens zal naargelang het gekozen deelnemerstype extra informatie ingevuld kunnen worden zoals bv. het registratienummer (als wandelaar).

Figuur 9 Registreren nieuwe deelname als wandelaar



Figuur 10 Registreren van nieuwe deelname als volger

2.4.2.9 Deelname beheren

Indien de gebruiker al deelneemt aan de huidige editie, kan hij/zij, vanuit hetzelfde tabblad "Mijn Overzicht", bepaalde gegevens van de deelname aanpassen via het bewerk icoontje. Dit laat toe dat een gebruiker zichzelf op voorhand kan inschrijven en dat hij/zij later het registratienummer kan invullen eenmaal hij/zij dit nummer ontvangen heeft van de officiële Dodentocht organisatie. Ten slotte heeft de eindgebruiker ook de mogelijkheid om zijn/haar deelname op te zeggen via de knop "Deelname annuleren", zoals hieronder geïllustreerd. Hierbij wordt een extra pop-up scherm getoond dat vraagt om de annulatie te bevestigen. Indien een deelnemer, die reeds gesponsord wordt, zijn/haar deelname opzegt, dan worden alle sponsors van deze deelnemer via een automatische e-mail op de hoogte gebracht dat hun sponsoring niet meer doorgevoerd wordt.

Figuur 11 Overzicht van bestaande deelname

2.4.2.10 Overzicht van sponsors

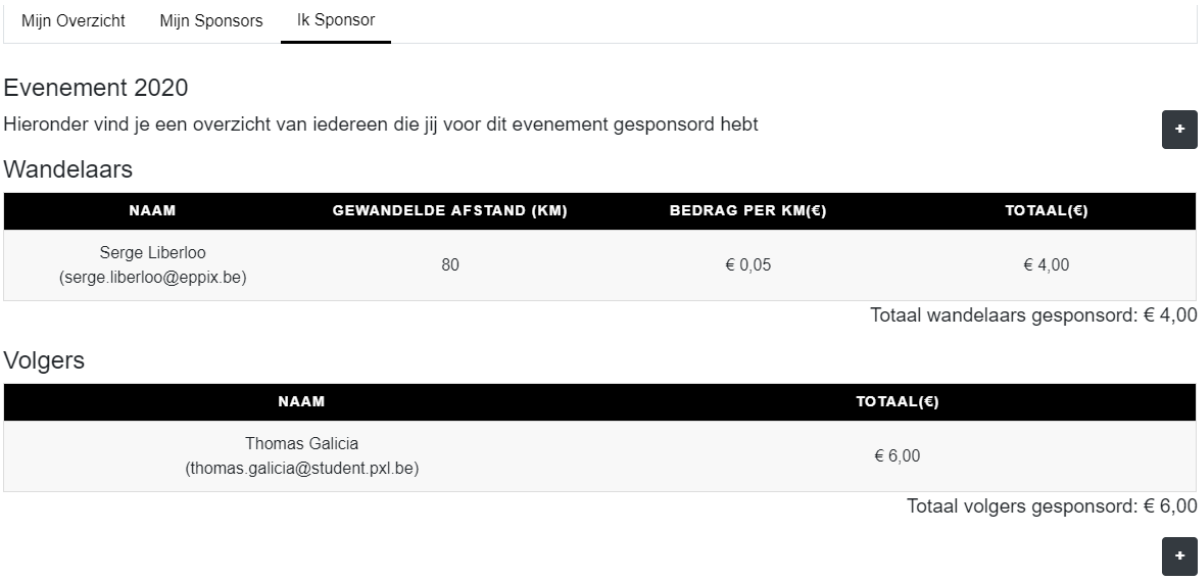
In de applicatie is het mogelijk om deelnemers te sponsoren en om gesponsord te worden als wandelaar of volger. Het sub tabblad “Mijn sponsors” biedt een overzicht van alle personen die de deelname van de huidige gebruiker steunt, de onderstaande figuur (figuur 12) biedt een visuele representatie. Dit tabblad bevat ook een jaarselector zodat de gebruiker ook zijn sponsors van de vorige edities kan raadplegen.

Figuur 12 Overzicht van alle sponsors die de deelnemer steunen

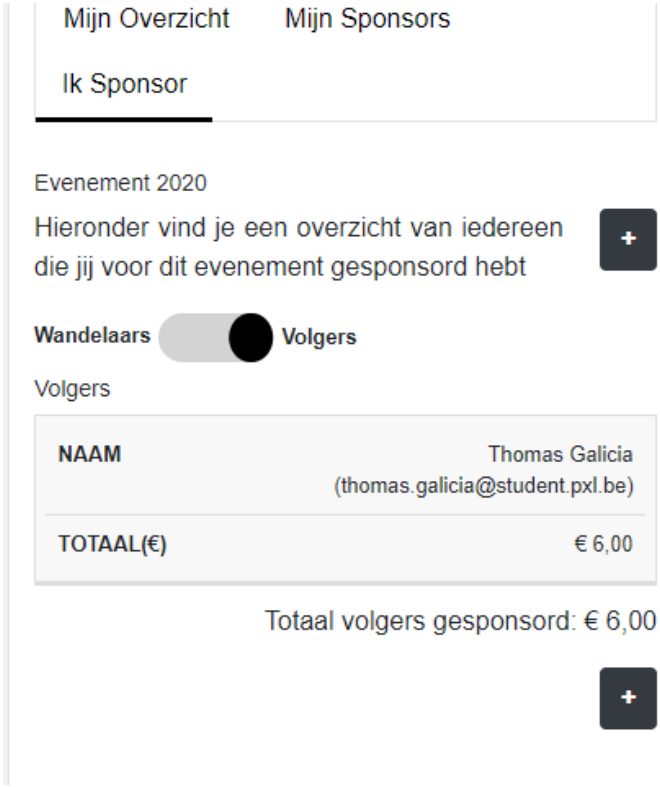
2.4.2.11 Sponsoring toevoegen

Naast het sub tabblad “Mijn Sponsors” bevindt zich het tabblad “Ik Sponsor”. Via dit tab krijgt de gebruiker een overzicht van alle deelnemers die hij/zij steunt. Het overzicht is onderverdeeld in twee verschillende tabellen: nl. een tabel met de wandelaars die de gebruiker sponsort en een tabel met volgers die de gebruiker sponsort. Deze tabellen worden op een desktop onder elkaar weergegeven. In een mobiele omgeving wordt er telkens één tabel getoond en kan er gewisseld worden aan de hand van een *toggle* knop. Beide overzichten worden hieronder weergegeven ter verduidelijking (figuur 13 en 14).

Boven- en onderaan de tabellen bevindt zich een knop met het plus symbool. Via deze knop kan de gebruiker een nieuwe sponsoring toevoegen. Zoals hieronder aangetoond wordt in figuur 15 en 16, dient de gebruiker te selecteren welke deelnemer hij/zij wil sponsoren. Indien de geselecteerde deelnemer een wandelaar is, zal de gebruiker zelf een bedrag kunnen invullen. Dit bedrag is het aantal (in euro) waarvoor hij/zij de wandelaar wil steunen per gewandelde kilometer. Als de geselecteerde deelnemer een volger is, wordt automatisch het vast bedrag voor volgers ingevuld. Dit bedrag kan niet aangepast worden door de gebruiker. Wanneer de gebruiker zijn/haar nieuwe sponsoring wil opslaan, zal er een extra venster weergegeven worden dat vraagt om de sponsoring te bevestigen. In dit venster wordt aangegeven tot welke bedrag de sponsoring maximaal kan oplopen. Dit venster is bedoeld om duidelijk aan te geven dat wandelaars per kilometer gesponsord worden en dat het totaalbedrag kan oplopen naargelang de effectief gewandelde kilometers.



Figuur 13 Overzicht van sponseringen die gebruiker uitvoert (desktopversie)



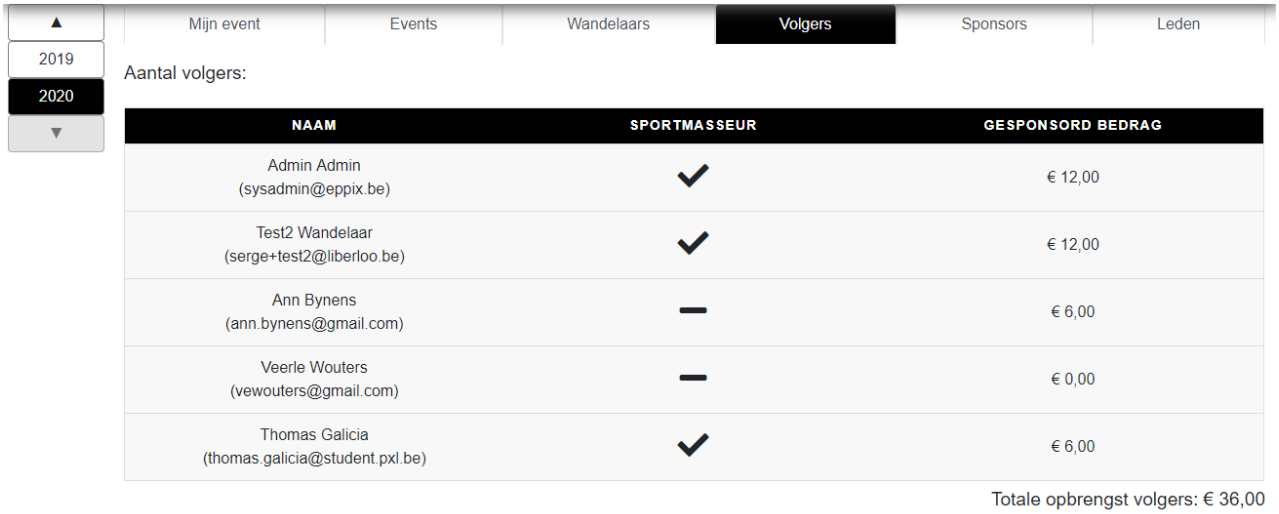
Figuur 14 Overzicht van sponseringen die gebruiker uitvoert (mobiele versie)

Figuur 15 Toevoegen van nieuwe sponsoring voor een wandelaar

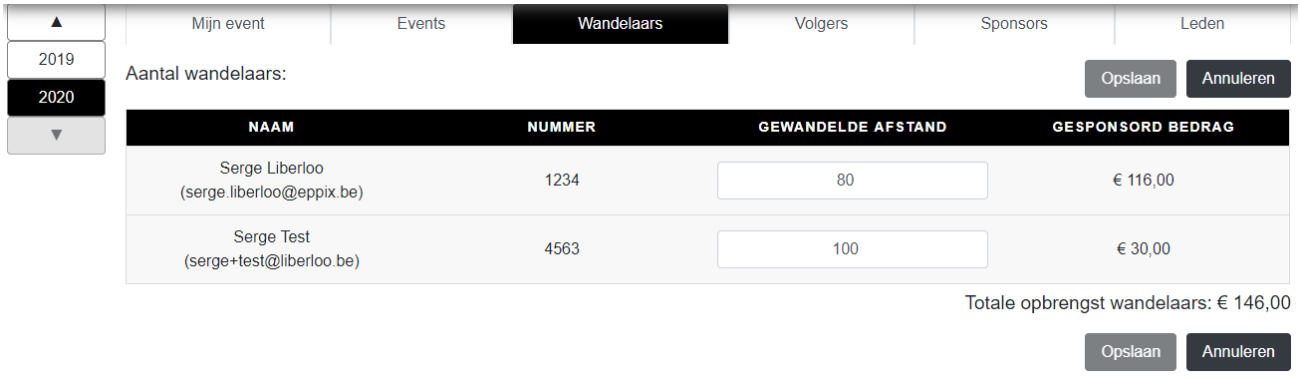
Figuur 16 Toevoegen van nieuwe sponsoring voor een volger

2.4.2.12 Deelnemersbeheer

Met de tabbladen “Wandelaars” en “Volgers” krijgen kernleden en administrators een duidelijk overzicht van alle deelnemers van een specifiek jaar. Het tabblad “Wandelaars” bevat onder andere de gewandelde afstand van elke wandelaar en het totaal gesponsord bedrag voor diezelfde wandelaar. Indien het geselecteerde evenement het open evenement is, kunnen administrators de gewandelde afstand van de wandelaars bewerken. Hierdoor moeten de gewandelde kilometers niet meer doorgemaild worden van wandelaars naar administrators (zie figuur 18 hieronder). Het tabblad “Volgers” biedt een overzicht van de volgers van een specifiek evenement. Zoals hieronder geïllustreerd wordt (figuur 17) krijgt een administrator of kernlid een overzicht van alle volgers. Per volger is aangegeven voor hoeveel hij/zij gesponsord wordt en of hij/zij zichzelf aanbiedt als sportmasseur of niet.



Figuur 17 Overzicht van alle volgers van het geselecteerde evenement



Figuur 18 Overzicht van alle wandelaars van het geselecteerde evenement in edit modus

2.4.2.13 Beheren van sponsors

Sponsors hebben niet de mogelijkheid om hun eigen gesponsorde bedragen te bewerken. Indien een sponsor een verkeerd bedrag heeft ingevuld of de verkeerde deelnemer heeft gesponsord kan hij/zij contact opnemen met een van de administrators. Via het tabblad “Sponsors” krijgen de administrators en de leden van het kerncomité een overzicht van alle gesponsorde bedragen van het geselecteerde evenement. Van het openstaande evenement kan een administrator de gesponsorde bedragen aanpassen (enkel bij wandelaars) of de sponsoring volledig verwijderen. Elke sponsoring voor een wandelaar is voorzien van twee pictogrammen nl. één voor de sponsoring te bewerken en één om de sponsoring te verwijderen, dit wordt in de onderstaande afbeelding verduidelijkt. Een sponsoring bedoeld voor een volger bevat maar één pictogram nl. om de sponsoring te verwijderen, een sponsoring voor een volger kan niet bijgewerkt worden omdat dit een vast bedrag bevat. Tijdens het bijwerken van een sponsoring is het niet mogelijk om andere items te bewerken of verwijderen; verder is het enkel mogelijk om het bedrag per kilometer aan te passen. Pas als de wijzigingen opgeslagen of geannuleerd zijn, heeft de administrator de mogelijkheid om een andere sponsoring bij te werken of te verwijderen. Voor leden van het kerncomité biedt dit tabblad enkel een overzicht, zij hebben geen extra kolom met acties (bijwerken of verwijderen).

Figuur 19 Overzicht gesponsorde bedragen van het openstaande evenement

2.4.2.14 Aanpassen gebruikersrollen

Via het tabblad “Leden” kan een administrator gebruikersrollen aanpassen. De administrators kunnen via dit tabblad een rol toevoegen of - verwijderen van een bepaalde gebruiker. Deze functionaliteit bevindt zich op de autorisatie-server, maar is afgeschermd en kan enkel binnen in de microservice omgeving aangevraagd worden. De applicatie communiceert de verandering van gebruikersrol met de Dotoforfelix API. Deze API zal de aanpassing doorgeven aan de autorisatie-server. De autorisatie-server en de Dotoforfelix API delen een *secret* dat enkel zij kennen. Zonder dit *secret* kan de actie niet opgevraagd worden op de autorisatie-server.

2.4.2.15 Toegangsmanagement

Op de Dotoforfelix API zijn bepaalde *resources* afgeschermd. Deze *resource*s kunnen enkel aangevraagd worden met de juiste gebruikersrollen. De webapplicatie zal een *access token* meesturen indien hij een van deze afgeschermde *resource*s wil opvragen. Om de toegang tot bepaalde *resource*s af te schermen is er gebruik gemaakt van de standaard sec*uri*ty die het Spring-framework aanbiedt. De standaard sec*uri*ty van Spring bevat al heel wat functionaliteiten, maar het gebruik van *access token*s is niet inbegrepen in de standaard implementatie. Om Spring sec*uri*ty compatibel te maken met *access token*s is er een extra middleware nodig.

Deze middleware wordt in Java onder andere geregeld aan de hand van filters. Deze filters worden uitgevoerd bij een HTTP-*request* naar de RESTful API. De filter wordt aangeroepen voordat de uiteindelijke doel*resource* of -operatie bereikt wordt. Voor het ondersteunen van *access token*s in Spring sec*uri*ty is een extra filter nodig, deze filter zal het *access token* uit de *authorization header* van de HTTP-*request* halen en vervolgens doorsturen naar de autorisatie-server. De autorisatie-server zal dit *access token* aanpakken, valideren en ten slotte decoderen. Het gedecodeerde *token* wordt teruggegeven als antwoord op de HTTP-*request*.

Voor het afschermen van een bepaalde *resource* kan er gebruik gemaakt worden van de “Secured” annotatie van Spring sec*uri*ty. Met deze annotatie kunnen bepaalde rollen meegegeven worden. Indien de *resource* aangevraagd wordt, zal de “Secured” annotatie op voorhand een klasse genaamd “Sec*uri*tyContextHolder” raadplegen. Deze klasse bevat kortweg de gebruiker die de HTTP-*request* uitvoert, van deze gebruiker worden de rollen opgevraagd. Indien de gebruiker een van de rollen die gespecifieerd zijn in de “Secured” annotatie bezit, dan zal de toegang tot de *resource* verleend worden. Indien de gebruiker niet een van de aangegeven rollen bezit, dan zal de toegang tot de *resource* geweigerd worden.

Als de standaard implementatie van Spring sec*uri*ty gebruikt wordt, zal de gebruiker automatisch in de “Sec*uri*tyContextHolder” gestopt worden. In het geval van *access token*s moet de gebruiker en zijn rollen expliciet in de “Sec*uri*tyContextHolder” gestopt worden. Dit gebeurt in de zelfgeschreven filter aan de hand van het gedecodeerde *access token*. Ten slotte moet de zelfgeschreven sec*uri*ty filter nog toegevoegd worden aan de *filter chain* van Spring sec*uri*ty. Door de filter toe te voegen aan de Spring sec*uri*ty *filter chain* wordt hij onderdeel van Spring sec*uri*ty.

2.4.3 Autorisatie-server

Deze autorisatie-server is generiek, dit wil zeggen dat deze verschillende applicaties moet ondersteunen en uitgerold kan worden in diverse microservice omgevingen.

2.4.3.1 Opstartproces

De server ondersteunt het gebruik van rollen. Doordat deze server generiek is moeten de verschillende rollen van de verschillende applicaties configureerbaar zijn. Dit wordt gedaan aan de hand van een JSON-bestand, in dit bestand kan de systeem administrator rollen definiëren. Het configuratie bestand bevat de namen van de verschillende rollen, optioneel kan er ook via een extra parameter aangegeven worden of de rol een standaard rol is.

Verder wordt er tijdens het opstarten van de autorisatie-server een initieel account gemaakt, dit geeft de systeem administrator de mogelijkheid om automatisch een administrator account aan te maken op de server. Dit administrator account is bovendien configureerbaar via een bestand. In dit bestand bevindt zich het e-mailadres, passwoord en de rol van het initiële account.

Tijdens het opstarten zal eerst het JSON-bestand gebruikt worden om de verschillende rollen uit te lezen en ze vervolgens op te slaan in de database. Hierna zal het tweede configuratie bestand gebruikt worden om het initiële account aan te maken in de database. Nadat dit account is aangemaakt, wordt er een bestand weggeschreven op het filesysteem om aan te geven dat dit account gecreëerd is. Dit bestand voorkomt dat het initieel account opnieuw wordt aangemaakt indien de oorspronkelijke gegevens in de databank zijn gewijzigd. Hieronder vindt u een afbeelding van beide configuratiebestanden ter verduidelijking.

```
{
  "roles": [
    {"role":  "ROLE_SPONSOR", "defaultRole":  true},
    {"role":  "ROLE_CORE_MEMBER"},
    {"role":  "ROLE_ADMINISTRATOR"}
  ]
}
```

Figuur 20 Configuratiebestand rollen

```
bootstrap.admin.email = sysadmin@eppix.be
bootstrap.admin.password = *****
bootstrap.admin.role = ROLE_ADMINISTRATOR
```

Figuur 21 Configuratiebestand voor het initieel account

2.4.3.2 E-mail verificatie

Bij het registreren van een nieuwe gebruiker voorziet de server e-mail-verificatie. Deze verificatie gebeurt aan de hand van een code die opgeslagen is in de databank. Indien de gebruiker succesvol is opgeslagen in deze databank, wordt er een verificatie-e-mail gestuurd. Deze e-mail bevat een code die de gebruiker kan ingeven op de doelapplicatie. Indien de gebruiker de juiste code aan de autorisatie-server aflevert, zal het account van de gebruiker beschikbaar worden. Ten slotte is ervoor gezorgd dat de e-mail enkel gestuurd wordt indien de gebruiker succesvol is toegevoegd aan de databank. Dit is gerealiseerd aan de hand van transacties binnen Java.

2.4.3.3 Gebruikersrollen migreren

De server ondersteunt het migreren van rollen van een gebruiker. Via de autorisatie-server kan een specifieke rol toegevoegd of afgenomen worden van een gebruiker. Deze functionaliteit is afgeschermd zodat enkel betrouwbare applicaties of servers deze actie kunnen uitvoeren. Deze actie kan nooit rechtstreeks aangevraagd worden aan de autorisatie-server. De server deelt een bepaald *secret* met andere servers binnen de *microservice*-omgeving. Op de autorisatie-server is een aparte REST-controller aanwezig met acties die enkel aan de hand van dit gedeelde *secret* kunnen uitgevoerd worden. Op deze manier zijn bepaalde functionaliteiten van de server afgeschermd van de buitenwereld en kunnen ze enkel tussen *microservices* gebruikt worden.

2.4.3.4 Toegangsmanagement

Voor het beheren van de toegang tot de Dotoforfelix API wordt er gebruikt gemaakt van *access token*s. Dit *token* bevat bepaalde informatie die gebruikt wordt om na te gaan of de gebruiker toegang heeft tot een bepaalde *resource*. In deze autorisatie-server wordt er gebruik gemaakt van het JWT-*token* formaat voor de *access tokens*. JWT-*token*s zijn zeer bruikbaar omdat ze *self-contained* zijn, dit wil zeggen dat alle nodige informatie in het *token* gestopt kan worden en dat er geen bijkomende database interactie nodig is. Een JWT-*token* bestaat uit 3 delen nl. een *header*, *payload* en *signature*. [4]

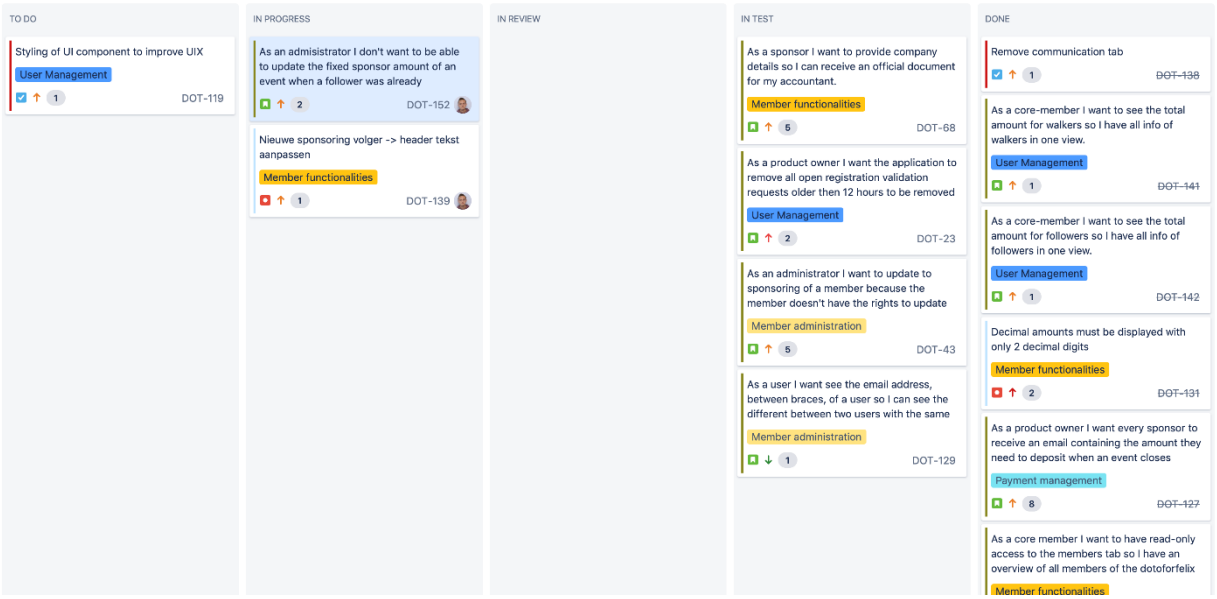
De header is het eerste deel van het *token*, hier bevindt zich het signeer algoritme en het type van *token* (= JWT). Na de header volgt de payload, dit bevat zogenaamde *claims*. Er bestaan drie soorten *claims*. De geregistreerde *claims* zijn op voorhand vastgelegd. Dit zijn gestandaardiseerde waardes die elk dienen voor bepaalde informatie bv. De *claim* “iss” staat voor de *issuer* van het *token*, dit is de partij die het *token* heeft uitgedeeld. De twee andere soorten zijn de publieke – en private *claims*. Dit zijn zelfgekozen waardes die gebruikt kunnen worden indien de *claim* niet aanwezig is in de geregistreerde *claims.*

De publieke en private zijn allebei zelfgekozen, het verschil tussen de twee is dat publieke claims *collision resistant* moeten zijn. Deze publieke claims zijn bedoeld voor publiek gebruik, hierdoor moeten ze goed gedocumenteerd zijn zodat er geen conflicten of botsingen kunnen optreden met andere claims. Private claims zijn enkel gekend tussen de producent en de consument van het *token*. De producent en consument weten op voorhand welke zelfgekozen *claims* er aanwezig gaan zijn in het uiteindelijke *token*. In deze autorisatie-server is er gebruik gemaakt van een private claim zodat de gebruikersrollen mee in het *access token* opgenomen kunnen worden.

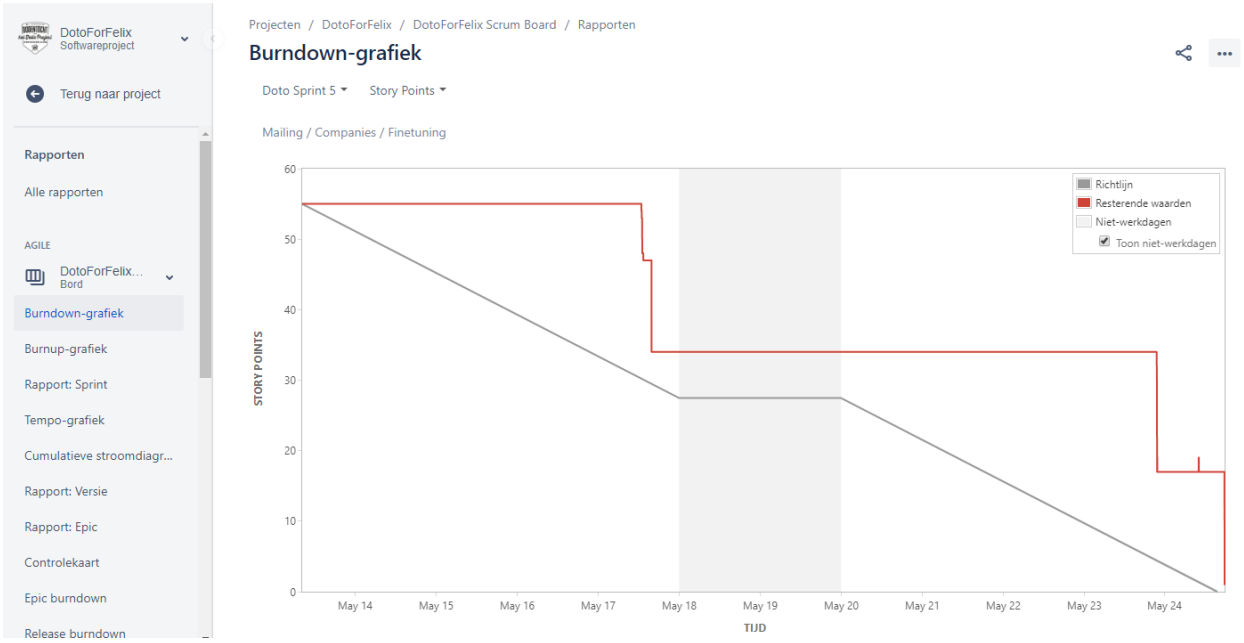
Ten slotte is er het laatste deel van het JWT-*token*, dit is de *signature*. Een JWT-*token* kan geëncrypteerd worden indien de data zeer gevoelig is, maar dit is niet noodzakelijk. Het *token* kan ook gesigneerd worden aan de hand van de *header*, *payload* en een *secret* in combinatie met een signeer algoritme. Wanneer er gekozen wordt om het *token* enkel te signeren en niet te encrypteren kan de data die zich in het *token* bevindt gedecodeerd en bekeken worden door iedereen. Dit komt omdat de *header* en *payload* simpelweg base64 gecodeerd zijn. De handtekening wordt gebruikt om te verifiëren dat het *token* wel degelijk afkomstig is van een bepaalde producent en ook om na te gaan dat het *token* niet aangepast is door iemand.

2.5 Issue-tracking

Door deze stage heen is er gebruik gemaakt van JIRA. Dit platform laat toe om alle verschillende functionaliteiten neer te schrijven en zo een *backlog* te creëren. Dankzij dit platform is de webapplicatie op een *agile* manier ontwikkeld aan de hand van *sprints*. Hieronder ziet u een voorbeeld van een *sprint* in JIRA (figuur 22), dankzij dit platform is er een duidelijk overzicht van de functionaliteiten waar in de huidige *sprint* de focus op gelegd wordt. Verder biedt JIRA extra ondersteuning aan voor reflecteren. M.a.w. het bespreekbaar maken, binnen het team, van de gevolgde methodologieën en gebruikte technologieën tijdens de voorbije *sprints*. Binnen JIRA kunnen er verschillende grafieken geraadpleegd worden, zoals bijvoorbeeld in figuur 23, die de efficiëntie van het ontwikkelingsproces in kaart brengen. Deze grafieken kunnen bijdragen aan het leerproces van de ontwikkelaar, hieruit kunnen lessen getrokken worden en kan achteraf gevalideerd worden of het ontwikkelingsproces wel degelijk een vooruitgang heeft geboekt. Ten slotte is er ook de mogelijkheid om JIRA te koppelen aan andere producten zoals Bitbucket zodat het project op een makkelijke manier beheerd kan worden.



Figuur 22 Scrum bord in JIRA



Figuur 23 Burndown grafiek in JIRA

2.6 Documentatie

Voor documentatie is binnen de stage gebruik gemaakt van Confluence. Dit platform biedt de mogelijkheid om technologieën, onderzoek en functionaliteiten op een gestructureerde manier te documenteren. Deze documentatie kan op een later moment geraadpleegd worden door iedereen binnen het bedrijf. Confluence kan ook aan JIRA gekoppeld worden zodat ze samen gebruikt kunnen worden.

2.7 Toekomstperspectief

Momenteel biedt de webapplicatie de basisfunctionaliteiten aan, maar deze applicatie kan in de toekomst verder uitgebreid worden. Een goede uitbreiding is het toevoegen van een betalingssysteem zodat de gesponsorde bedragen rechtstreeks via de webapplicatie overgedragen kunnen worden aan het Felix project. Verder zou het aantal gewandelde kilometers automatisch ingevuld kunnen worden door deze data op te halen bij de organisatie van de Dodentocht. Deze functionaliteit is echter afhankelijk van de beschikbare APIs van de organisatie.

2.8 Reflectie

Deze stageopdracht heeft gezorgd dat mijn bestaande kennis rond Angular, Java en Spring Boot verder uitgebreid is. De kennis die ik doorheen mijn opleiding opgedaan had rond deze technologieën was zeer toepasselijk, de voorkennis zorgde voor een vlotter verloop van de stage. Hierdoor kon de focus op andere zaken gelegd worden. Bovendien heeft deze opdracht mij nieuwe technologieën en tools bijgebracht die ervoor zorgen dat ik efficiënter te werk kan gaan. Doorheen deze stage heb ik duidelijk geleerd hoe de 3 lagen (controller, service en repository) in elkaar zitten en welke verantwoordelijkheden door welke laag gedragen wordt. Dit heeft ertoe geleid dat de API op een gestructureerde manier is opgebouwd en dat er wederom efficiënter gewerkt is. Het gebruik van Liquibase heeft het doel van database migraties aangetoond. Tussen de verschillende versies van het eindproduct is het datamodel meermaals aangepast, Liquibase zorgt ervoor dat deze aanpassingen op een vlotte en foutloze manier toegepast kunnen worden. Dit doel zal ik meepakken in mijn verdere carrière omdat dit zeer belangrijk is. Ik ben ten slotte zeer tevreden over het eindproduct dat ik gerealiseerd heb, deze opdracht heeft ervoor gezorgd dat ik op een relatief korte tijd een grote stap als ontwikkelaar heb kunnen zetten.

II. Onderzoekstopic

1 OAuth

1.1 Vraagstelling

Binnen het bedrijf is er grote vraag naar een generieke autorisatie-server zodat deze gebruikt kan worden voor verschillende applicaties of omgevingen binnen Eppix. Deze vraag zal beantwoord worden aan de hand van het implementeren van OAuth.

Concreet komen de volgende vragen aan bod:

- Welke OAuth flow kan gebruikt worden in de verschillende soorten omgevingen?
- Wat is het verschil tussen OAuth 1.0 en 2.0?
- Welke versie van OAuth biedt de meeste beveiliging, versie 1.0 of 2.0?
- Is OAuth enkel voor externe partijen bedoeld of kan het ook binnen een eigen microservice omgeving geïmplementeerd worden?
- Hoe worden de verschillende rollen binnen de autorisatie-server gemaakt en beheerd zodat ze adaptief zijn ten opzichte van de doelapplicatie?

1.2 Onderzoeksmethode

In dit onderzoek wordt er gekeken naar de manier waarop de OAuth-standaard werkt en hij op een generieke manier geïmplementeerd kan worden. Verder wordt er ook onderzocht welke versie van de standaard de beste is.

Voor het onderzoek naar de implementatie van OAuth wordt gebruik gemaakt van de officiële documentatie van OAuth 1.0 (RFC 5849) en 2.0 (RFC 6749). Deze documentatie dient als basis, van waaruit, indien mogelijk, een test project gemaakt wordt voor de verschillende onderdelen of implementaties.

Indien de standaard-documentatie onvoldoende informatie oplevert om een test-implementatie te maken, worden andere betrouwbare bronnen geraadpleegd om toch genoeg informatie te verzamelen voor een test-implementatie.

Na een test-implementatie van een onderdeel zal ditzelfde onderdeel indien nodig geïmplementeerd worden in de uiteindelijke stageopdracht. Na de uiteindelijke implementatie wordt er ook nog een extra verificatie stap uitgevoerd waarbij de implementatie stap per stap wordt nagekeken om zeker te zijn dat de standaard correct is toegepast.

Ten slotte worden voor de vergelijking tussen de versies van de standaard de voor- en nadelen van de versies ten opzichte van elkaar afgewogen. Verder zullen de verschillen in de versies ook geanalyseerd worden om een inzicht te krijgen in de reden waarom deze versies verschillen van elkaar. Op deze manier kunnen eventuele zwakke punten in de oudere versie aangetoond worden.

1.3 Uitwerking onderzoek

1.3.1 Introductie

OAuth is een open standaard die dient om een gebruiker/applicatie toegang te geven tot bepaalde informatie zonder dat er wachtwoorden uitgewisseld moeten worden. Het is een populaire standaard die vaak gebruikt wordt om bepaalde data toegankelijk te maken voor externe partijen.

In een niet-OAuth-situatie zou een *client* informatie verkrijgen door middel van de gebruikers gegevens van de eindgebruiker (vb. gebruikersnaam en wachtwoord). Dit zorgt voor verschillende problemen:

- De doelserver moet gebruikersgegevens bijhouden van de eindgebruikers om deze te valideren.
- De doelserver moet wachtwoorden ondersteunen.
- De *client* kan aan de hand van de juiste gegevens van de eindgebruiker op elk moment de gegevens/acties op de doelserver aanspreken. De doelserver heeft geen controle over hoelang de *client* toegang heeft tot de middelen en er is geen controle over wat de client met de gegevens van eindgebruiker mag uitvoeren/opvragen.

OAuth pakt deze problemen aan door een autorisatie-laag toe te voegen. Binnen OAuth wordt er een verschil gemaakt tussen de eindgebruiker en de *client*. De *client* zal in de meeste gevallen binnen OAuth geen gebruik maken van de inloggegevens van de eindgebruiker. De *client* zal in de plaats toegang vragen aan de eindgebruiker voor bepaalde zaken. Als de gebruiker instemt, dan zal de *client* een toegangs*token* ontvangen dat hij kan gebruiken om deze bepaalde zaken op te vragen of uit te voeren op de doelserver. In dit onderzoeksonderwerp is er gebruik gemaakt van OAuth versie 2.0.

1.3.2 Terminologieën

Binnen OAuth 2.0 zijn er verschillende rollen: [5]

- ***Resource owner***: Dit is de eindgebruiker. Deze entiteit zal toestemming geven aan de *client* zodat de *client* bepaalde data kan opvragen of bepaalde operaties kan uitvoeren.
- ***Resource server***: Dit is de server die de data van de *resource owner* bevat, deze zal ook het toegangs*token* van de *client* gebruiken om te bepalen of de *client* toegang heeft tot de data of de rechten heeft om de actie uit te voeren.
- ***Client***: De applicatie/het systeem dat de data gaat opvragen of de operatie gaat uitvoeren in naam van de eindgebruiker/*resource* owner.
- ***Authorization server***: Deze server staat in voor het aanmaken van de toegangs*token* nadat de eindgebruiker/*resource* owner succesvol geauthentiseerd is.

Verder zijn er binnen OAuth 2.0 nog een paar algemene terminologieën:

- ***Authorization* grant**: Dit is het vragen van toestemming aan de eindgebruiker/*resource* owner. Hij kan hiermee akkoord gaan of deze weigeren.
- ***Access Token***: Dit is het toegangs*token* dat de *client* zal gebruiker om de data op te vragen of de actie uit voeren. Dit *token* is verkregen nadat de eindgebruiker zijn toestemming heeft verleend. Aan de hand van dit *token* weet de doelserver welke rechten de *client* heeft.
- ***Redirect Uri***: deze url zal gebruikt worden door de *authorization* server om bepaalde data door te geven aan de *client* tijdens de OAuth flow.
- **Scope**: een parameter die de *client* bezorgd aan de autorisatie-server om aan te geven welke permissies/data hij wil verkrijgen van de eindgebruiker.
- ***Client ID***: Een *id*entifier die aangeeft welke applicatie het is. Deze wordt meestal gebruikt wanneer je communiceert met externe partijen. Bij de externe partij zal je een applicatie/project moeten registreren. Na de registratie zal je een *client ID* en *client secret* ontvangen.
- ***Client secret*:** Een gedeeld geheim tussen de applicatie en autorisatie-server. Dit *secret* mag niet publiek bekendgemaakt worden. Het *secret* wordt gebruikt als extra beveiliging bijvoorbeeld tijdens het uitwisselen van autorisatie-code voor de *access token* in de *authorization* code flow.

1.3.3 Grant-types

Binnen OAuth 2.0 zijn er een vijftal mogelijke grant-types/flows die gebruikt kunnen worden. Een grant-type beschrijft een manier waarop OAuth toegepast kan worden. De verschillende grant- types zijn bedoeld voor verschillende situaties omdat niet elke grant-type toegepast kan worden in elke situatie.

1.3.3.1 *Authorization* code grant-type

Het *authorization* code grant-type is een veel gebruikt type. Het wordt gebruikt om een eigen API open te stellen voor externe partijen of om een API van een externe partij te gebruiken.

In dit grant-type zal de *client* de eindgebruiker doorverwijzen naar de autorisatie-server van de organisatie. Bij het doorverwijzen naar deze autorisatie-server zal de *client* bepaalde parameters meegeven zoals de *redirect uri*, scope en het *client ID*. De eindgebruiker zal vervolgens zijn inloggegevens van de organisatie invullen op de autorisatie-server van diezelfde organisatie.

Indien de inloggegevens correct zijn, zal de autorisatie-server een autorisatie-code genereren. Vervolgens verwijst de autorisatie-server de gebruiker terug door naar de oorspronkelijke *client* aan de hand van de *redirect uri*. In deze *redirect uri* zal ook de autorisatie-code zitten, waarschijnlijk in de vorm van een query parameter zodat deze toegankelijk is voor de *client*.

Vervolgens zal de *client* deze autorisatie-code opsturen naar de autorisatie-server samen met het *client secret* en diezelfde *redirect uri*. De autorisatie-server zal deze gegevens ontvangen en vervolgens gaat hij de autorisatie-code nakijken. Indien de code, *client secret* en *redirect uri* geldig zijn en de combinatie van de drie geldig is, dan zal de autorisatie-server een *access token* genereren en bezorgen aan de *client*. Dit *access token* kan de *client* dan gebruiken om data op te vragen aan de *resource* server.

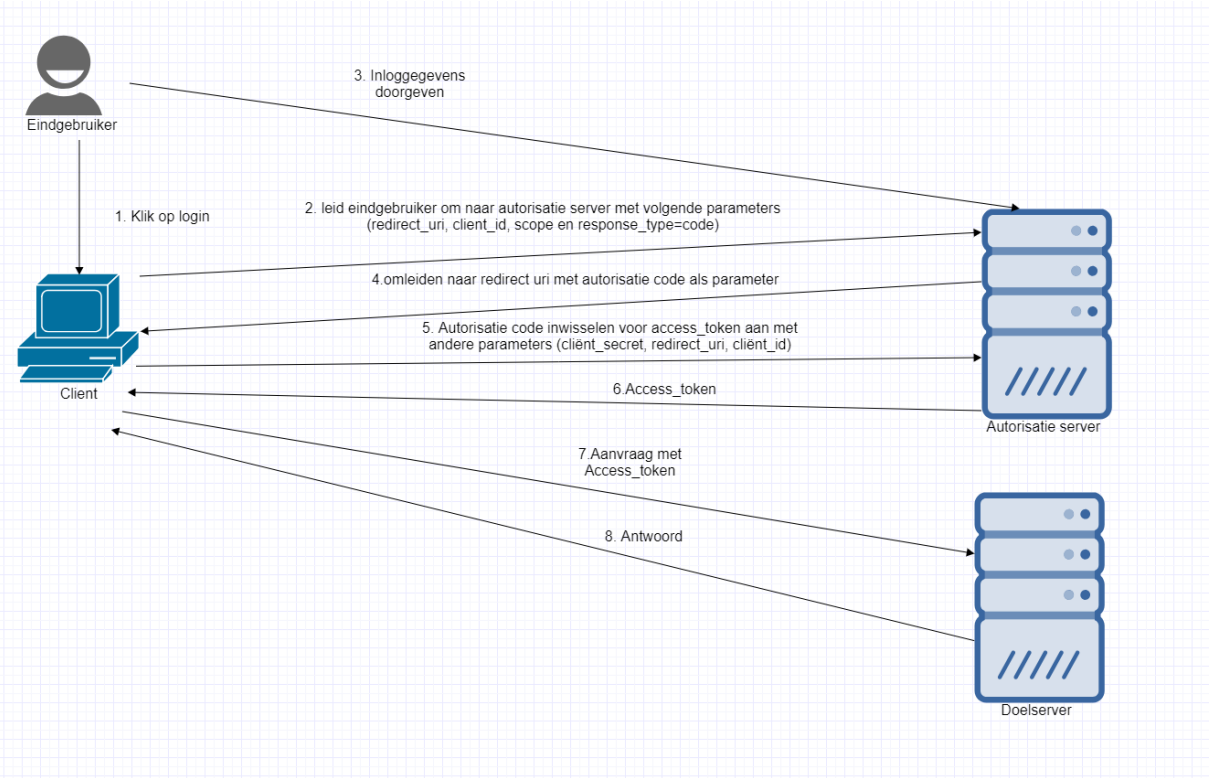
Dit grant-type wordt veel gebruikt om verschillende redenen. Ten eerste wordt de gebruiker doorverwezen van de *client* naar de autorisatie-server van de organisatie. Hierdoor bezorgt de gebruiker zijn inloggegevens die hij bezit van die organisatie rechtstreeks aan de autorisatie-server van diezelfde organisatie. De *client* gaat nooit deze inloggegevens ontvangen.

Ten tweede maakt dit grant-type gebruik van een hybride aanpak; dit type maakt gebruik van zowel een back- als frontchannel.

Een backchannel is een goed beveiligd kanaal tussen twee eindpunten, bijvoorbeeld twee servers. Een frontchannel is een minder beveiligd kanaal; een voorbeeld hiervan is een internetbrowser. In een internetbrowser kan je verschillende elementen zoals netwerkverkeer bekijken.

In dit grant-type gebeurt het grootste deel op het frontchannel. Het uitwisselen van de autorisatie - code voor het uiteindelijke *access token* gebeurt op het backchannel. De reden hiervoor is enerzijds dat het backchannel veiliger is en deze stap moet ook zo veilig mogelijk verlopen, anderzijds kan gevoelige informatie die optreedt in deze stap zoals het *client secret* op de server bewaard worden en moet dit niet in bijvoorbeeld de browser bijgehouden worden.

De autorisatie-code kan niet ingewisseld worden zonder het bijhorende *client secret* en; daarom is het belangrijk dat dit *secret* op een veilige plaats wordt bewaard. Om deze reden gebeurt het uitwisselen van de autorisatie-code op het backchannel, omdat dit zorgt voor een veilige opslagplaats. Deze grant-type staat hieronder voorgesteld aan de hand van een afbeelding waarbij de verschillende stappen geïllustreerd worden.

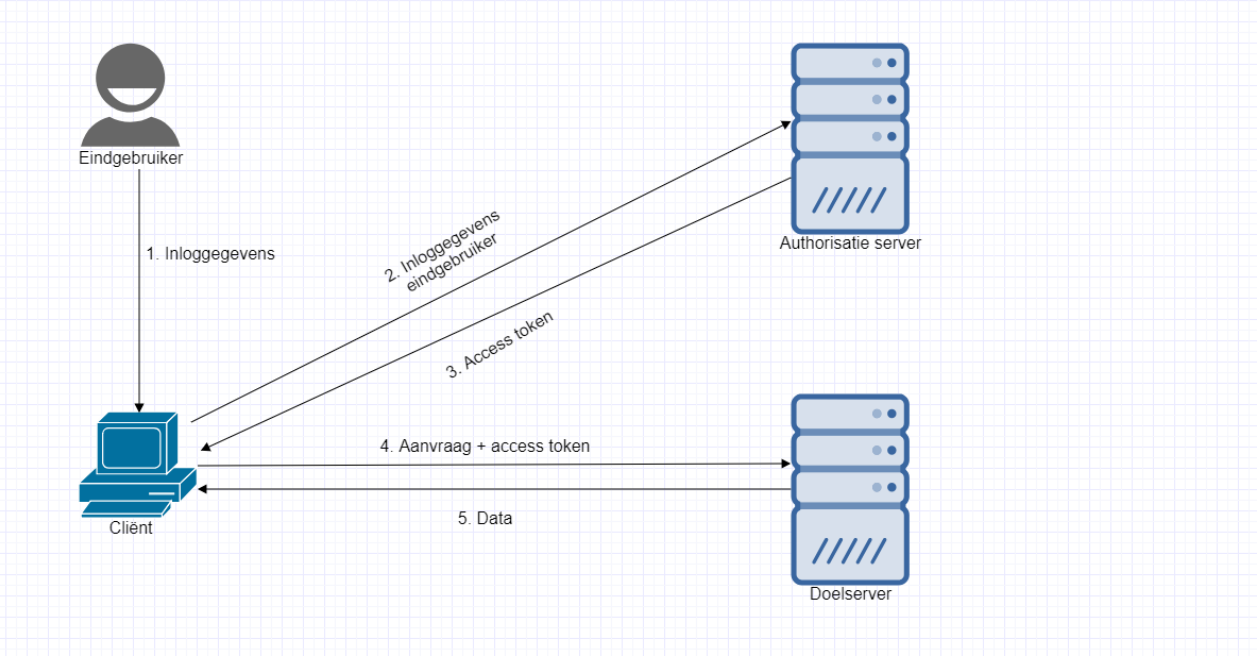


Figuur 24 OAuth 2.0 authorization code grant-type normale workflow

1.3.3.2 Password grant-type

Bij het *password* grant-type ruilt de *client* de inloggegevens van de eindgebruiker in voor een *access token*. Deze aanpak is eigenlijk wat OAuth wou oplossen; daarom wordt dit grant-type ook enkel gebruikt voor eigen services omdat deze te vertrouwen zijn. Om externe partijen toe te laten om een service te gebruiken of bepaalde data op te vragen mag dit grant-type niet gebruikt worden.

In dit type geeft de eindgebruiker zijn inloggegevens aan de *client* zoals hieronder aangetoond. Deze *client* gaat deze gegevens gebruiken in combinatie met een eventueel *client ID* en/of *client secret* om zich te authentiseren bij de autorisatie-server. Wanneer de inloggegevens succesvol gevalideerd zijn, zal de autorisatie-server een *access token* genereren en dit teruggeven aan de *client*. De *client* kan dit *access token* vervolgens gebruiken om de doelserver aan te spreken.



Figuur 25 Password grant-type voor lokaal inloggen met OAuth 2.0

1.3.3.3 Implicit grant-type

De *implicit* grant-type is een vereenvoudigde versie van de *authorization* code grant-type. Bij dit type wordt het *access token* rechtstreeks teruggegeven indien de gebruikersgegevens correct zijn. Het uitwisselen van de autorisatie-code voor het *access token* wordt in dit type niet uitgevoerd, hierdoor is het een stuk onveiliger.

Dit type wordt ook over het algemeen niet aangeraden om te gebruiken. Een van de enigste toepassingen om deze grant-type toe te passen is wanneer je een Single Page Application hebt of een publieke *client*.

Een beter alternatief voor de implicit grant-type is het gebruik van de *authorization* code grant- type. Het probleem dat hier optreedt is dat we gebruik maken van een publieke *client* dus data zoals het *client secret* is niet goed afgeschermd. Om dit probleem aan te pakken bestaat er een extensie voor de *authorization* code grant-type, deze extensie heeft als naam “Proof Key for Code Exchange”.

Bij deze extensie wordt er in het begin twee zaken opgesteld nl. een *code verifier* en een *code challenge*. De *code verifier* is een willekeurige string tussen 43 en 128 karakters lang. Deze *code verifier* wordt vervolgens gebruikt om een *code challenge* te maken.

Wanneer de publieke *client* de code challenge heeft gemaakt, zal hij de autorisatie-server aanspreken. Hierbij stuurt hij diezelfde code challenge mee. Wanneer de autorisatie-code uitgewisseld moet worden voor het uiteindelijke *access token,* zal de *client* de *code verifier* mee opsturen naar de autorisatie-server.

Deze *code verifier* zal gebruikt worden om de code challenge van stap 1 na te kijken. Indien de code challenge geldig blijkt te zijn zal het *access token* aangemaakt worden en teruggegeven worden aan de publieke *client*.

Deze extensie is gemaakt om ervoor te zorgen dat publieke *client*s nog altijd de *authorization* code grant-type kunnen gebruiken. De publieke *client* kan geen gebruik maken van een vast *client secret* omdat de *client* het *secret* niet veilig kan opslaan. Via de code challenge en code verifier wordt er eigenlijk een tijdelijk *secret* gemaakt om ervoor te zorgen dat de autorisatie-code niet door iemand anders gestolen kan worden en vervolgens ingeruild kan worden voor het *access token*.

1.3.4 OpenID Connect

OpenID Connect is een extra laag bovenop OAuth. Gewone OAuth dient om *clients* toegang te geven tot bepaalde data of om toestemming te geven om bepaalde operaties uit te voeren. Het bevat dus geen gebruikersinformatie. OpenID Connect voegt een authenticatie laag toe aan OAuth zodat de doelapplicatie deze gebruikersinformatie kan raadplegen.

OpenID Connect voegt voornamelijk twee zaken toe aan OAuth. Ten eerste een *id token* en eventueel een *userinfo endpoint*. Het *id token* is een extra *token* dat de *client* zal terugkrijgen samen met het oorspronkelijke *access token*. Dit *id token* zal basisinformatie bevatten over de eindgebruiker zoals bijvoorbeeld naam en voornaam.

De tweede voornaamste toevoeging van OpenID Connect is een *userinfo endpoint*. Dit *endpoint* kan door de *client* aan de hand van het *access token* aangesproken worden om extra gebruikers informatie op te vragen indien de basisinformatie, die zich in het *id token* bevindt, niet genoeg zou zijn. Om OpenID Connect te gebruiken zal er als *scope* “openId” meegegeven worden aan de autorisatie-server. De autorisatie-server zal dan de gewone OAuth flow doorlopen en zowel een *acccess* - als *id token* genereren.

1.3.5 Literatuurstudie: OAuth versie 1.0 versus versie 2.0

1.3.5.1 Wat is het verschil tussen OAuth 1.0 en 2.0?

Het doel van OAuth is om een applicatie toe te laten om data op te halen bij een externe partij. Voor dat de OAuth-standaard bestond moest de gebruiker zijn inloggegevens van de externe partij delen met de applicatie. De applicatie zou dan data ophalen bij de externe partij aan de hand van de inloggegevens van de gebruiker.

In OAuth 1.0 zijn er verschillende terminologieën die aan bod komen:

- *Client*/*Consumer*: Dit is de applicatie die probeert de data op te halen bij externe partij

- Server/Serviceprovider: Verwijst naar een http-server die om kan gaan met de OAuth standaard. Dit is de eventuele server van de externe partij waarvan de *client* data wil opvragen.

- *Protected resource*: Verwijst naar de *resource* die afgeschermd is op de server. Deze *resource* kan enkel opgevraagd worden met de juiste permissies.

- *Resource owner*: De eindgebruiker die bepaald of de *client* gebruik mag maken van de *resource*.

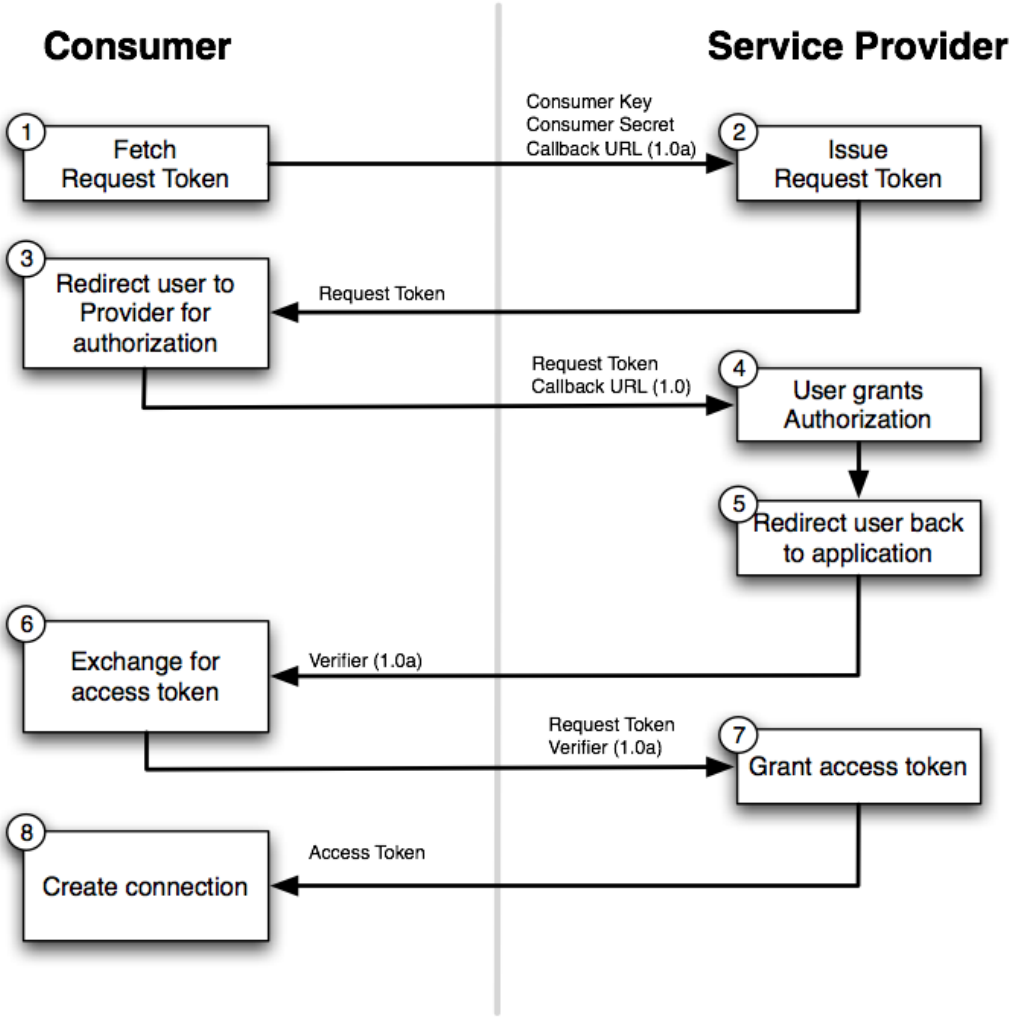
- *Consumer key* en - *secret*: Dit zijn gegevens die de *client* gaat gebruiken om zichzelf te authentiseren.

- *Request token*: Dit is een tijdelijk *token* dat door de server aangemaakt wordt op het begin van de OAuth workflow. De *client* gaat doorheen de OAuth 1.0 flow dit *token* gebruiken om de server aan te spreken.

- *Access token*: Dit is een *token* dat de server zal genereren indien het *request token* geldig is. Via dit *access token* kan de *client* de nodige *resource*s opvragen aan de server.

De OAuth 1.0 workflow:

1) De *client* vraagt een *request token* aan met behulp van de consumer key en - *secret*.
2) De server/serviceprovder valideert de *consumer key* en – *secret*. Indien deze geldig zijn zal er een *request token* gegenereerd worden. Dit *request token* wordt afgeleverd aan de *client* doormiddel van een *redirect*.
3) De *client* zal het *request token* in ontvangst nemen. Vervolgens gaat de *client* de gebruiker terug omleiden naar de serviceprovider. Tijdens dit omleiden stuurt de *client* het *request token* mee om zichzelf te authentiseren.
4) De gebruiker vult zijn inloggegevens in bij de serviceprovider. Indien deze inloggegevens correct zijn en de gebruiker toestemming heeft verleend dat de *client* toegang heeft tot zijn/haar data, wordt de gebruiker terug omgeleid naar de *client* met een verificatie code.
5) De *client* stuurt de verificatie code samen met het *request token* naar de serviceprovider. Daar zullen zowel de verificatie code als het *request token* nagekeken worden. Indien beide correct zijn zal er een *access token* gegenereerd worden en teruggegeven worden aan de *client*.
6) De *client* vraagt de nodige *resource*s op met behulp van het *access token*.



Figuur 26 OAuth 1.0 workflow voor het aanvragen van een access token

“OAuth 2.0 is een compleet nieuwe standaard. Het is niet *backward compatible* met OAuth 1.0.” [6] De focus van OAuth 2.0 was voornamelijk het versimpelen van de ontwikkeling aan de *client* kant. Verder biedt OAuth 2.0 ook meerdere flows. Deze verschillende flows hebben allemaal hun eigen situaties waarin ze gebruikt kunnen worden. Hierdoor is OAuth 2.0 bruikbaar in webapplicaties, desktopapplicaties en mobiele applicaties. OAuth 1.0 is niet aangepast aan bv. mobiele applicaties.

OAuth 2.0 is zoals eerder vermeld een nieuwe standaard en geen uitbreiding op versie 1.0. Versie 2.0 behandeld bepaalde problemen en uitdagingen van 1.0. Ten eerste steunt 2.0 op het gebruik van HTTPS; dit zorgt ervoor dat niet elke *request* gesigneerd moet worden. Ten tweede zorgen de verschillende flows in OAuth 2.0 ervoor dat er ondersteuning is voor niet-browser applicaties zoals mobiele applicaties. Versie 1.0 is voornamelijk ontworpen voor web-toepassingen. Dit leidt tot verschillende problemen op desktop – en mobiele applicaties.

Verder is er bij OAuth 2.0 een duidelijke scheiding gemaakt tussen de verschillende rollen. Er is bijvoorbeeld een autorisatie-server en een *resource*-server terwijl er in versie 1.0 enkel gesproken wordt over een server. Ten slotte zijn de *access tokens* tussen beide ook verschillend. Bij versie 1.0 is er geen duidelijkheid over de levensduur, in de nieuwe versie is er wel de mogelijkheid om een levensduur toe te voegen aan het *access token* waardoor het na een bepaalde tijd onbruikbaar is. Dankzij de toevoeging van deze levensduur is er ook een extra *token* bijgekomen in versie 2.0, nl. het refresh *token* waarmee een nieuw *access token* aangevraagd kan worden.

Verder staan in de bron de grote lijnen van OAuth 2.0 uitgelegd, maar hier is in deze literatuurstudie niet verder op ingegaan omdat deze zaken al zijn toegelicht in andere onderdelen van het onderzoek.

1.3.5.2 Okta: de verschillende onderdelen van OAuth in beide versies

"OAuth 2.0 is een volledig nieuwe versie ten opzichte van OAuth 1.0. Het enige dat beide versies delen met elkaar zijn de algemene doelen en gebruikerservaring" [7]. OAuth 1.0 is een veelgebruikte standaard die een lange tijd gebruikt werd. Na verloop tijd werd het wel duidelijk dat de standaard bepaalde uitdagingen met zich meebracht. Uit deze uitdagingen is OAuth 2.0 gestart.

Het eerste verschil tussen beide versies omvat de terminologieën en rollen. In versie 2.0 zijn er vier duidelijke rollen nl. *client*, autorisatie-server, *resource* server en *resource owner*. Deze begrippen komen in versie 1.0 ook wel voor, maar onder een andere naam. Ook is er in OAuth 1.0 geen duidelijke scheiding tussen autorisatie-server en *resource* server. Dit is een van de grote verschillen tussen de versies.

Bij versie 2.0 is er gekozen om de server van versie 1.0 op te splitsen in 2 aparte servers. Dit zorgt ervoor dat ze fysiek ook apart kunnen bestaan. Dit is een mogelijkheid maar geen verplichting volgens de standaard. Verder zorgt het er ook voor dat deze servers apart ontwikkeld en beheerd kunnen worden. Het opsplitsen in twee servers zorgt ervoor dat er een autorisatie-server ontstaat die enkel verantwoordelijk is voor het beheren van de gebruikers, het aanmaken en valideren van de *access token*s. [8]

Versie 1.0 is ook moeilijker te implementeren doordat er rekening moet gehouden worden met cryptografie omdat deze versie niet steunt op HTTPS. Verder zorgt versie 1.0 voor moeilijkheden bij het maken van een *request* met behulp van een *access token*. In deze versie moet er in de *authorization* header van de *request* verschillende parameters meegegeven worden. De verschillen tussen de HTTP-*requests* worden hieronder geïllustreerd. [9]

```
curl --get 'https://api.twitter.com/1.1/statuses/show.json' \
--data 'Id=210462857140252672' \
--header 'Authorization: OAuth oauth_consumer_key="xRhHSKcKLl9VF7fbyP2eEw",
oauth_nonce="33ec5af28add281c63db55d1839d90f1",
oauth_signature="oBO19fJO8imCAMvRxmQJsA6IdXk%3D", oauth_signature_method="HMAC-
SHA1", oauth_timestamp="1471026075", oauth_token="12341234-
ZgJYZOh5Z3ldYXH2sm5voEs0pPXOPv8vC0mFjMFtG", oauth_version="1.0"'
```

Figuur 27 HTTP-request met OAuth 1.0

```
curl https://api.example.com/profile
-H "Authorization: Bearer XXXXXXXXXXX"
```

Figuur 28 HTTP-request met OAuth 2.0

Versie 2.0 introduceerde het principe van *bearer token*s, dit zorgde ervoor dat enkel het *access token* in de *authorization* header van de *request* aanwezig moest zijn. [10]

Een ander verschil is dat OAuth 1.0 was ontstaan met drie verschillende flows, maar naargelang de standaard zich verder ontwikkelde zijn deze drie flows samengevoegd tot een enkele flow die bedoeld was voor alle *use cases*. Na verloop van tijd toen versie 1.0 meer en meer geïmplementeerd was, werd ook duidelijk dat deze enkele flow voor beperkingen zorgde op vlak van gebruikerservaring. Versie 2.0 lost dit probleem op door gebruik te maken van verschillende flows die elk bedoeld zijn voor bepaalde situaties. [11]

Ten slotte is het centraliseren van autorisatie in grotere systemen een uitdaging in de oude versie. In versie 1.0 wordt er gebruik gemaakt van de *client credentials* om de inkomende *request*s te verifiëren, dit zorgt ervoor dat de endpoints die beveiligd zijn toegang moeten hebben tot deze *client credentials* om ze te valideren. Hierdoor is het moeilijk om de endpoints en de autorisatie te scheiden van elkaar omdat ze beide toegang moeten hebben tot dezelfde informatie. Bij versie 2.0 is dit wel mogelijk omdat de autorisatie-server de *client credentials* nakijkt tijdens het aanvragen van een *access token*. De beveiligde endpoints gebruiken enkel het *access token* waardoor ze geen rekening moeten houden met andere data. [12] Hierdoor kunnen de *endpoints* en de autorisatie-server gescheiden worden en kan de autorisatie-server indien nodig gecentraliseerd worden.

1.3.5.3 Welke versie gebruiken voor implementatie?

OAuth 1.0 is een bekende standaard die opkwam tussen 2007 en 2008. Versie 1.0 was een veel gebruikte standaard. Om deze versie te implementeren moet er veel rekening gehouden worden met cryptografie, dit zorgt voor uitdagingen als ontwikkelaar.

De komst van versie 2.0 zorgde ervoor dat de standaard makkelijker te implementeren was, dit zorgde wel voor een afname inzake beveiliging.

Wanneer we de versies met elkaar vergelijken wordt het duidelijk dat ze van elkaar verschillen. Het eerste verschil tussen beide versies is de beveiliging op vlak van transport van data. Versie 1.0 steunt niet op het gebruik van HTTPS, dit is de voornaamste reden waarom deze versie zo hard steunt op het gebruik van cryptografie. Bij versie 2.0 wordt er wel gesteund op het gebruik van HTTPS voor beveiligde communicatie, dit zorgt er wel voor dat indien deze versie gebruikt wordt in een niet HTTPS - of fout geconfigureerde omgeving, de standaard zeer kwetsbaar is aan bv. een man-in-the-middle aanval.

Versie 2.0 maakt ook gebruik van *bearer token*s, dit zorgt ervoor dat deze versie makkelijker te implementeren is. Het gebruik van *bearer token*s zorgt er wel voor dat het niveau van beveiliging ten opzichte van versie 1.0 lager is omdat deze *token*s gestolen of gekopieerd kunnen worden.

Verder zorgt de nieuwe versie voor meer flexibiliteit doordat er verschillende workflows aanwezig zijn, die elk bedoeld zijn voor specifieke situaties. Versie 1.0 omvat slechts één workflow die voornamelijk bedoeld is bij de implementatie van *web client*s. Dit zorgt voor beperkingen of uitdagingen bij, bijvoorbeeld mobiele applicaties. Ten slotte is er in versie 2.0 meer duidelijkheid en flexibiliteit op vlak van de server(s) die aan bod komen. In deze versie is er een onderscheid gemaakt tussen de autorisatie-server en *resource* server, dit zorgt ervoor dat ze losgekoppeld kunnen worden van elkaar.

Indien een ontwikkelaar moet kiezen tussen beide versies, moet hij/zij rekening houden met eventuele externe partijen. Veel grote bedrijven zoals bv. Google zijn volledig afgestapt van versie 1.0 en laten deze versie niet meer toe. Als ontwikkelaar is het wel nog mogelijk om versie 1.0 te gebruiken binnen een eigen autorisatie-server, maar de algemene trend voor dit geval is om OAuth

2.0 met een cryptografie extensie te gebruiken. Dit is een persoonlijke afweging die de ontwikkelaar moet maken. [13]

1.3.5.4 Conclusie

Beide versies vertrekken vanuit dezelfde doelen zoals bijvoorbeeld het aanspreken van externe partijen vanuit een *client* zonder dat deze *client* de inloggegevens van de eindgebruiker bij de externe partij nodig heeft. Dit zorgt er ook voor dat beide versies op algemeen niveau redelijk wat gelijkenissen vertonen. Dit is wel enkel het geval op algemeen niveau. Wanneer er dieper ingegaan wordt op beide versies, wordt al snel duidelijk dat ze volledig verschillen. Versie 1.0 zorgt voor meer beveiliging dankzij de extra cryptografie, maar ook voor meer uitdagingen inzake implementatie.

Versie 2.0 zorgt ervoor dat de implementatie versimpeld wordt, maar dit zorgt wel voor een afname inzake beveiliging. Verder is het ook duidelijk dat versie 2.0 afhankelijk is van bestaande beveiliging nl. HTTPS. Indien HTTPS niet aanwezig zou zijn, mag versie 2.0 niet uitgerold worden omdat er dan een te groot beveiligingsrisico aanwezig is. Versie 1.0 is niet afhankelijk van HTTPS omdat het gebruikmaakt van cryptografie, maar dit zorgt wel voor een extra uitdaging.

Verder zorgt de nieuwe versie voor meer flexibiliteit en een duidelijk onderscheid tussen de verantwoordelijkheden van de verschillende componenten. Hierdoor is de nieuwe versie meer bruikbaar in een microservice omgevingen ten opzichte van versie 1.0, want de autorisatie-server kan makkelijker afgezonderd worden van de *resource* server. Bijkomend kan in versie 2.0 de autorisatie-server gecentraliseerd worden zodat deze gebruikt kan worden voor verschillende applicaties of *resource* servers. Bovendien bevat versie 2.0 verschillende workflows die elk ontworpen zijn voor verschillende situaties. Dit draagt weer bij aan het feit dat versie 2.0 makkelijker te implementeren is en flexibeler is ten opzichte van versie 1.0.

Ten slotte is er het gebruik van *bearer token*s in de nieuwe versie die net als andere zaken in versie 2.0 zorgt voor een makkelijkere -, maar ook een minder beveiligde implementatie. Dit is weer een voorbeeld waar duidelijk wordt dat versie 2.0 minder beveiligd is, maar deze versie van de standaard probeert deze afname in beveiliging te compenseren door nieuwe functionaliteiten toe te voegen. In dit geval is dat het toevoegen van een levensduur aan de *bearer token*s waardoor deze maar voor een bepaalde tijd bruikbaar zijn.

1.3.5.5 Reflectie

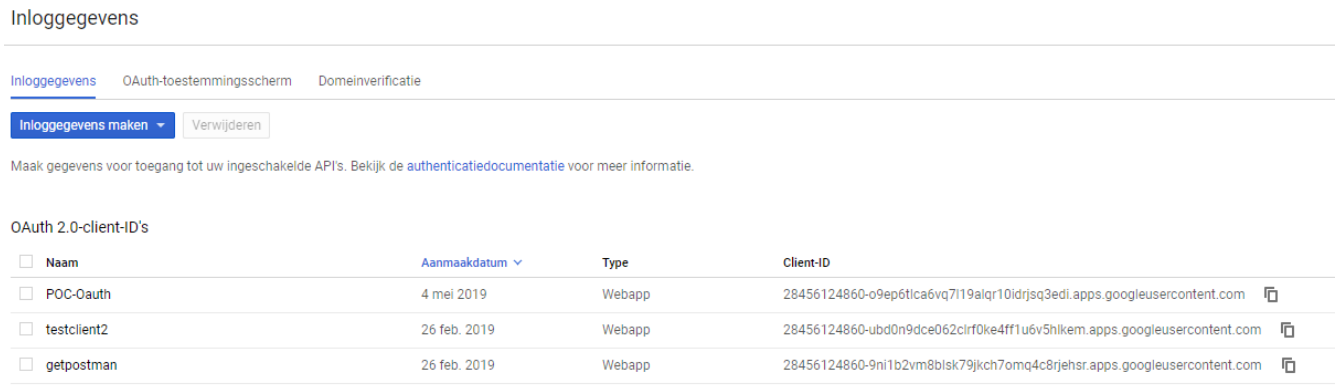
De bronnen komen op alle vlakken overeen met elkaar. Ze vullen elkaar ook aan, niet elke bron gaat even diep in op elk stuk in vergelijking met de andere bronnen. Dit toont wel aan dat het belangrijk is om verschillende bronnen te raadplegen. Deze literatuurstudie geeft meer achtergrond over het onderzoeksonderwerp en toont ook aan dat OAuth 2.0 niet de enige standaard is die geïmplementeerd kan worden. Verder kunnen vanuit de verschillen die in deze literatuurstudie zijn aangehaald, bepaalde vragen gesteld worden die een ontwikkelaar aan het begin van een OAuth gerelateerd project kan stellen om te bepalen welke versie hij/zij gaat implementeren. Voor mezelf was het zoeken naar een onderwerp voor deze literatuurstudie de grootste uitdaging. OAuth is een autorisatie standaard waarbij er geen alternatieven zijn. Daarom dat ik gekozen heb om OAuth intern te vergelijken inzake de verschillende versies. Hierbij heb ik de bovenstaande bronnen geraadpleegd, ik heb ze geanalyseerd. Ten slotte heb ik de structuren en gespreksonderwerpen met elkaar vergeleken. De bronnen vatten dezelfde algemene werking van OAuth samen maar elke bron bevat op zijn beurt een extra hoofdstuk dat in de andere bronnen weinig of niet aan bod komt. Deze hoofdstukken heb ik vervolgens gebruikt om deze literatuurstudie aan te vatten. Ondanks dat de bronnen voor het grote deel dezelfde informatie deelde was het mogelijk om extra onderdelen toe

te voegen door middel van de verschillen tussen de bronnen. Ik heb deze literatuurstudie aangepakt vanuit de verschillen en niet de gelijkenissen. Dit zorgt voor een andere kijk en geeft naar mijn mening en vollediger resultaat.

1.3.6 Proof of concept: Inloggen met Google

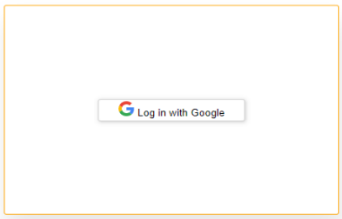
Veel applicaties laten vandaag de dag toe om in te loggen via een externe partij zoals Google of Facebook. Dit zorgt ervoor dat gebruikers niet per applicatie aparte inloggegevens moeten beheren. In deze testopstelling zal de gebruiker inloggen bij Google met zijn/haar inloggegevens, vervolgens zal er op basis van de gegevens die de gebruiker bij Google bezit een account aangemaakt worden in de databank. Dit *proof of concept* focust zich enkel op Google omdat zij de OAuth standaard volledig implementeren. Facebook bevat een eigen adaptatie die gebaseerd is op OAuth. Dit zorgt voor een andere manier van werken omdat de standaard niet volledig geïmplementeerd is.

Google maakt gebruik van de *authorization* code grant-type (zie 1.3.3.1), dit type is het meest gebruikt om OAuth met externe partijen toe te passen. Voor dit grant-type is een *client ID* en – *secret* nodig bij de externe partij. Dit kan bemachtigd worden via de *developers console* van Google [14]. Via deze console kan een ontwikkelaar het *client ID* en – *secret* genereren. De onderstaande afbeelding illustreert hoe deze *developers console* er uit ziet.



Figuur 29 Developers console Google waar client ID en secret aangevraagd kunnen worden

Deze testopstelling maakt gebruik van een RESTful API geschreven in Java met behulp van het Spring Boot-framework. Verder is er een Angular-webapplicatie aanwezig die zal communiceren met de RESTful API. De webapplicatie bevat in deze opstelling een pagina met daarop een knop om in te loggen via Google, zoals geïllustreerd op de onderstaande afbeelding.



Figuur 30 Proof of concept: mogelijkheid om in te loggen met Google

Indien de gebruiker op deze knop klikt zal aan de RESTful API het adres opgevraagd worden van de autorisatie-server van Google. Dit is een vast adres dat gevonden kan worden aan de hand van het *discovery document* dat Google aanbiedt [15]. Dit adres bevat enkele parameters zoals het *client ID*. Omdat deze gegevens bijgehouden worden op RESTful API is het de verantwoordelijkheid van de API om het adres op te bouwen en af te leveren aan de webapplicatie. Eenmaal het adres is afgeleverd aan de webapplicatie zal deze de gebruiker omleiden naar het adres. Vervolgens komt de gebruiker terecht op de autorisatie-server van Google waar de inloggegevens aangevuld moeten worden.

Indien de inloggegevens correct zijn zal aan de gebruiker toestemming gevraagd worden om bepaalde data of acties te delen met de doelapplicatie. Als de gebruiker toestemming geeft zal hij/zij terug omgeleid worden met de autorisatie-code naar de webapplicatie. Deze applicatie zal de autorisatie-code opsturen naar de RESTful API, op deze API wordt deze code uitgewisseld voor een *access* – en *id token* bij Google. Deze uitwisseling gebeurt op de API omdat hier gevoelige data zoals het *client sercret* veilig opgeslagen kan worden. Na het ontvangen van het *access* – en *id token* kan de data uit het *id token* gehaald worden omdat dit een JWT-*token* is. De gebruikersdata die in het *id token* zit kan vervolgens gebruikt worden om een nieuwe gebruiker in de databank aan te maken of om de bestaande gebruiker uit de databank te halen. Met deze gegevens kan de RESTful API ten slotte een *access token* genereren voor de webapplicatie.

1.3.7 Toekomstperspectief

Deze autorisatie-server dient momenteel als basis. Vanuit deze basis kan de autorisatie-server verder uitgebreid worden naargelang de doelapplicatie. In de toekomst kan deze server gebruikt worden om op een gecontroleerde manier externe partijen toe te laten. Om deze functionaliteit te verwezenlijken zal er waarschijnlijk gewerkt moeten worden met *scopes* in plaats van gebruikersrollen. Hiervoor zal de autorisatie-server uitgebreid moeten worden met de authorization code grant-type (zie 1.3.3.1).

Conclusie

Dit onderzoek heeft het belang van OAuth aangetoond. Via OAuth kan de toegang tot een RESTful API gecontroleerd worden. Een API kan met behulp van deze standaard publiek uitgerold worden, maar ondertussen zorgt de standaard ervoor dat de beheerders of eigenaars van de API zelf de touwtjes in handen hebben i.v.m. de toegang tot de API. OAuth kan gebruikt worden in een eigen omgeving; verder biedt het mogelijkheid aan een API open te stellen naar externe partijen en bepaald OAuth wat de externe partij in naam van de eindgebruiker mag uitvoeren op de API.

Voor de keuze tussen OAuth versie 1 en 2 kan er best gekeken worden naar de nieuwste versie. Versie 1 zorgt voor meer beveiliging dankzij encryptie, maar dit voordeel kan evengoed in versie 2 toegepast worden. Versie 2 is vandaag de dag een betere optie omdat het makkelijker te implementeren is en de meeste bedrijven ondersteunen alleen de nieuwste versie. Verder zorgt het voor een betere architectuur doordat er een duidelijk onderscheid is tussen de *resource* - en autorisatie-server. Versie 2 biedt de mogelijkheid om te werken met een gecentraliseerde autorisatie-server die afzonderlijk ontwikkeld en uitgerold kan worden. Bovendien is de nieuwste versie bruikbaarder omdat ze verschillende strategieën aanbiedt naargelang het soort applicatie dat ontwikkeld wordt.

Ten slotte moet er in deze conclusie duidelijk bij vermeld worden dat OAuth een autorisatie standaard is. Het wordt vaak gebruikt voor authenticatie, maar hier is het niet voor ontworpen. Indien er toch nood is aan een authenticatie laag kan er gekeken worden naar de implementatie van OpenID connect.

Reflectie

Deze stage heeft naar mijn mening een grote meerwaarde voor afstuderende studenten. Via de stage leer je op korte tijd veel zaken bij die van pas komen wanneer je na de stage in het werkveld terecht komt. Het realiseren van dit project van start tot einde heeft gezorgd voor een unieke ervaring met veel leuke momenten. Het onderzoeksonderwerp was voor mij als ontwikkelaar zeer gepast. Het beschermen van een API is naar mijn mening even belangrijk als het ontwikkelen van de API zelf. Het is een onderwerp waar in mijn ogen niet veel aandacht aan besteed wordt terwijl vandaag de dag alles rond privacy en het afschermen van data draait.

Ik heb dankzij de stageopdracht mijn kennis in Java, Angular en Spring Boot verder kunnen uitbouwen. Bovendien heb ik de mogelijkheid gehad om nieuwe technologieën of onderwerpen te gebruiken zoals bijvoorbeeld Liquibase voor databank migraties in Java. Deze nieuwe technologieën tonen op momenten zaken aan die je als ontwikkelaar niet wist dat ze ontbraken. Ten slotte zorgde de ondersteuning die ik van Eppix gekregen heb ervoor dat mijn algemene vaardigheden als ontwikkelaar gestegen zijn. Binnen Eppix hebben ze mij goed ondersteund en hebben ze mij op de juiste momenten op mijn fouten gewezen zodat ik hieruit kon leren. Ik ben op bepaalde momenten tegen fouten of problemen aangelopen, maar dankzij de ondersteuning die je krijgt los je deze zaken op en leer je eruit. Over het algemeen liep deze stage zeer vlot, maar er zijn altijd momenten dat het een beetje minder vlot gaat. Op deze momenten moet je jezelf kunnen motiveren en moet je eventueel terugkijken waarom de zaken niet lopen zoals verwacht. Hieruit kan je dan je lessen trekken en werken aan deze aandachtspunten. Deze twaalf weken hebben gezorgd voor een unieke en leerrijke ervaring waarvan ik met veel trots terugkijk op het resultaat dat ik heb neergezet samen met het team van Eppix.

Bibliografie

[1] „Het Felix Project," [Online]. Available: http://www.hetfelixproject.be/profiel/profiel.htm. [Geopend 16 Maart 2019].

[2] Dodentocht, „Dodentocht," /, [Online]. Available: http://www.dodentocht.be/nl.html. [Geopend 13 Mei 2019].

[3] „Eppix website," Eppix, [Online]. Available: https://www.eppix.be/. [Geopend 27 Februari 2019].

[4] Auth0, „jwt.io," Auth0, [Online]. Available: https://jwt.io/introduction/. [Geopend 5 Mei 2019].

[5] „OAuth 2.0: roles (RFC 6749)," [Online]. Available: https://tools.ietf.org/html/rfc6749#section-1.1. [Geopend 26 Februari 2019].

[6] P. Ramchandani, „Packtpub: What's the difference between OAuth 1.0 and OAuth 2.0?," Packtpub, 13 Juni 2018. [Online]. Available: https://hub.packtpub.com/what-is-the-difference-between-oauth-1-0-and-2-0/. [Geopend 6 April 2019].

[7] „OAuth: Differences Between OAuth 1 and 2," Okta, [Online]. Available: https://www.oauth.com/oauth2-servers/differences-between-oauth-1-2/. [Geopend 6 April 2019].

[8] „OAuth: Separation of Roles," Okta, [Online]. Available: https://www.oauth.com/oauth2-servers/differences-between-oauth-1-2/separation-of-roles/. [Geopend 6 April 2019].

[9] „OAuth: Authentication and Signatures," Okta, [Online]. Available: https://www.oauth.com/oauth2-servers/differences-between-oauth-1-2/authentication-and-signatures/. [Geopend 6 April 2019].

[10] „OAuth: Bearer Tokens," Okta, [Online]. Available: https://www.oauth.com/oauth2-servers/differences-between-oauth-1-2/bearer-tokens/. [Geopend 6 April 2019].

[11] „OAuth: User Experience and Alternative Token Issuance Options," Okta, [Online]. Available: https://www.oauth.com/oauth2-servers/differences-between-oauth-1-2/user-experience-alternative-token-issuance-options/. [Geopend 6 April 2019].

[12] „OAuth: Performance at Scale," Okta, [Online]. Available: https://www.oauth.com/oauth2-servers/differences-between-oauth-1-2/performance-at-scale/. [Geopend 6 April 2019].

[13] S. E. Team, „Synopsys: What's the difference? OAuth 1.0 vs OAuth 2.0," Synopsys, 11 Maart 2016. [Online]. Available: https://www.synopsys.com/blogs/software-security/oauth-2-0-vs-oauth-1-0/. [Geopend 7 April 2019].

[14] Google, „Google developers console," Google, [Online]. Available: http://console.developers.google.com. [Geopend 19 Mei 2019].

[15] Google, „Google discovery document,” Google, [Online]. Available: https://developers.google.com/identity/protocols/OpenIDConnect#discovery. [Geopend 05 Mei 2019].