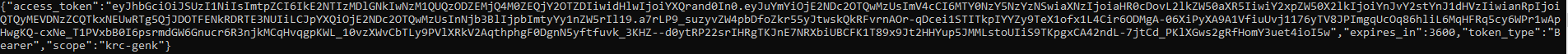
*PE Yellow Team*  
  
  
**Ster 1: lokale console applicatie**  
  
In eerste instantie hebben we een eigen client toegevoegd aan de identity server met zijn eigen client\_secret.  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Om een globale configuratie te creëren voor onze lokale applicatie hebben we deze credentials geplaatst in een aparte json file.   
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
We hebben daarna enkele methodes toegevoegd om allereerst het token op te gaan halen. Het eerste probleem waar we tegenaan liepen, was dat we onze baseURL hadden ingesteld als een secure URL, namelijk https. De error die hierdoor werd opgegooid was de volgende:  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
Aangezien het in deze opdracht nog niet direct de bedoeling was om de communicatie tussen de services via HTTPS te laten verlopen, hebben we er gewoon een simpel hypertekst transfer protocol van gemaakt.   
  
De methode GetToken gaat voor ons een WebRequest uitvoeren. Hier stellen we het content type en de methode in. We gaan namelijk aan de hand van een form onze credentials meegeven en een post uitvoeren op de URL die ons een token uitleent waarbij we ons kunnen authenticeren aan onze API. Hier hebben we wat research moeten doen om de form data op een correcte manier mee te geven in onze request. Het probleem waarbij we hier tegenaan liepen, was voornamelijk de encodering die we hadden toegepast, namelijk de ASCII encodering. We hebben hierbij wat hulp op het internet gezocht om te bepalen hoe we ons form correct konden meegeven. Een werkende oplossing vonden we door officiële Microsoft docs te raadplegen, namelijk:  
<https://docs.microsoft.com/en-us/dotnet/framework/network-programming/how-to-send-data-using-the-webrequest-class>  
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Eens we deze problemen hadden opgelost, was het enkel nog een kwestie om ons token uit de response te halen. Dit hebben we kunnen bekomen door onderstaande methode te gebruiken om het token uit de response te parsen. Het token zit omvat in een object waarbij het token de value is van de key ‘access\_token’.  
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Het bearerToken wordt via ‘GetTokenFromResponse’ opgeslaan in de class variable ‘bearertoken’, die we dan makkelijk kunnen aanspreken in ons programma. Dan konden we uiteindelijk ons Main programma uitschrijven. In de Main halen we eerst het token op door de methode ‘GetToken’, waarbij we onze globale ingestelde configuratie (credentials) meegeven.   
  
We laten de user een stad / postcode ingeven. Deze input geven we mee als parameter in de request, het endpoint naar onze seatholders. Om deze request te laten slagen, geven we het bearerToken mee in de headers.  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
We lezen de response uit tot het einde, met onderstaande screenshot als uiteindelijke resultaat:  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
  
  
**Ster 2: Auth0 Client Credentials flow**  
  
  
We hebben voor deze opdracht gebruik gemaakt van Auth0, als tussenliggende service voor het uitlenen van een token en te kunnen authenticeren aan onze API. De stappen hiervoor waren zeer duidelijk, binnen Auth0 konden we een nieuwe API aanmaken, waarbij we een naam moesten instellen en een identifier.  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
  
De identifier hierbij was wel zeer belangrijk. Dit is iets wat we initieel mis hadden gedaan. Zoals te zien in de placeholder is het natuurlijk belangrijk om als identifier jouw eigen API mee te geven die je wilt benaderen. Dus in ons geval was dat <http://localhost:5000/api/>. Wij hebben onze API dan ook de naam ‘localhost:5000’ gegeven.   
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
  
Eens deze API was opgezet, konden we een test uitvoeren om Auth0 tokens op te halen voor onze applicatie. We hebben hierbij gebruik gemaakt van Postman om deze verbinding uit te testen.  
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving

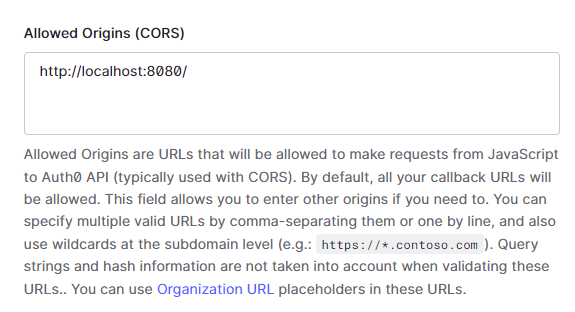
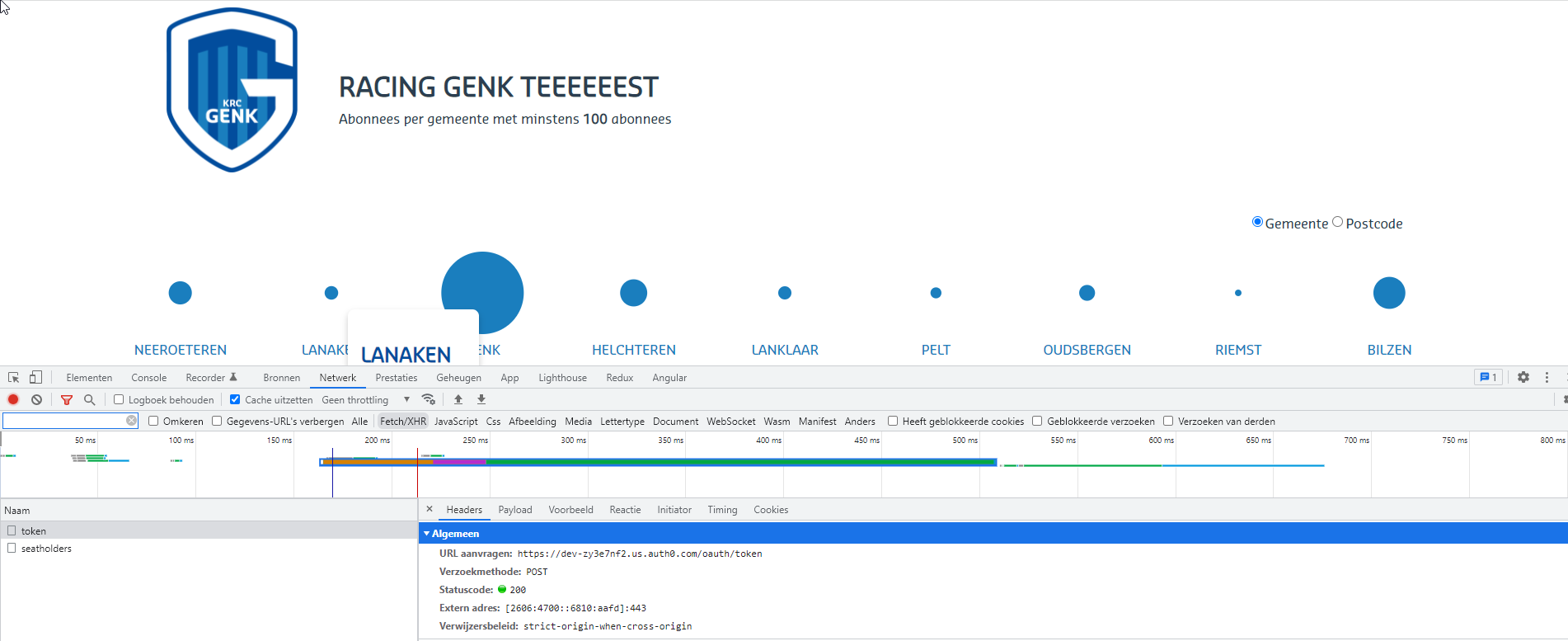
Het enige verschil hierbij was dat we bovenop de client\_id, de client\_secret en het grant\_type ook ons audience moesten meegeven. De data moest op dezelfde manier worden doorgegeven, namelijk als x-www-form-urlencoded. We kregen zo als response het token terug, uitgeleend door Auth0.  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
De eerste stappen op het platform van Auth0 waren zeer duidelijk en verliepen vlot. Maar eens we dit gingen implementeren in ons project, merkten we toch dat we enkele zaken nog moesten aanpassen. We hebben onder andere in Startup.cs de ‘ConfigureServices’ moeten aanpassen om nu gebruik te maken van de Auth0 Client Credentials Flow, in plaats van de flow van de identityserver. We hebben hierbij de manier van authentiseren moeten veranderen, alsook het authenticatieschema. Deze informatie konden we ook terugvinden onder het kopje ‘quickstart’ van onze API. Maar dit vergde toch wat meer uitzoekwerk.  
Afbeelding met tekst

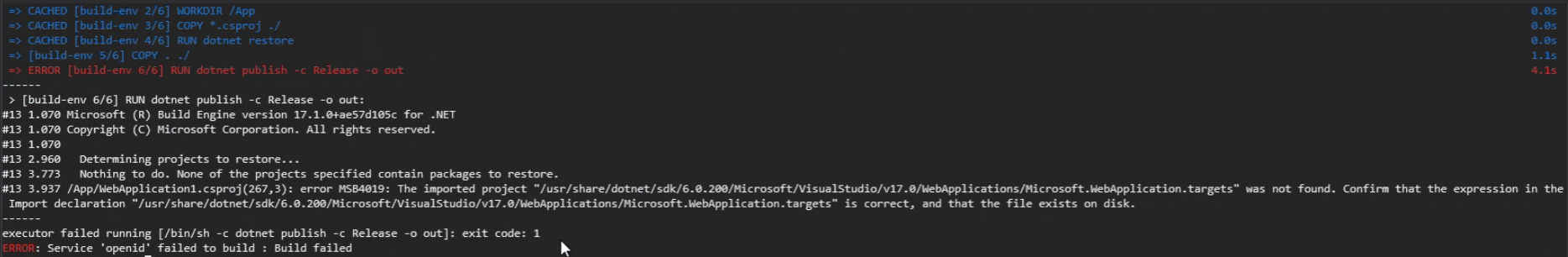
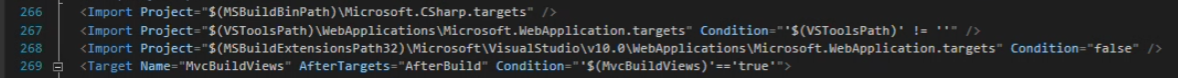
Automatisch gegenereerde beschrijving  
  
  
Onze lokale console applicatie was vrij gemakkelijk om aan te passen, hierbij moesten we niet veel veranderen. Ons username en password werden nu respectievelijk het client\_id en het client\_secret van Auth0. De baseUrl moesten we ook nog veranderen naar het baseUrl van Auth0.  
Afbeelding met tekst

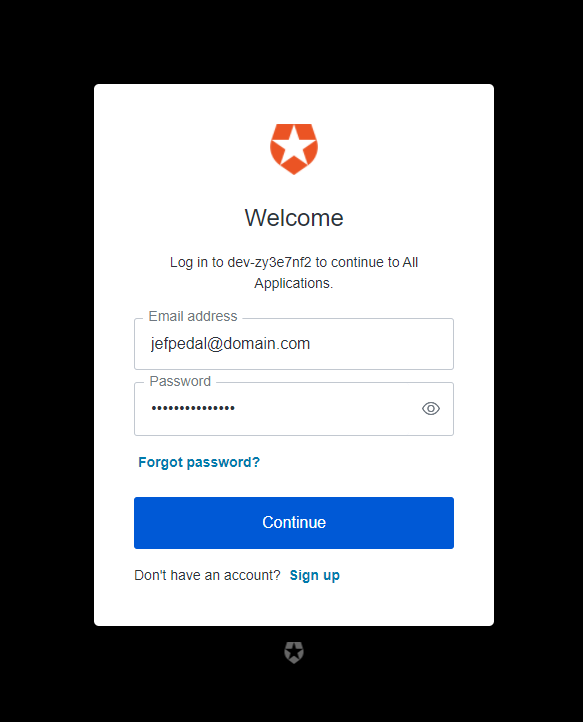
Automatisch gegenereerde beschrijving  
  
In de form data moesten we nog een extra veld toevoegen, namelijk het audience veld. Ons API endpoint van de resources die we graag willen benaderen. Afbeelding met tekst

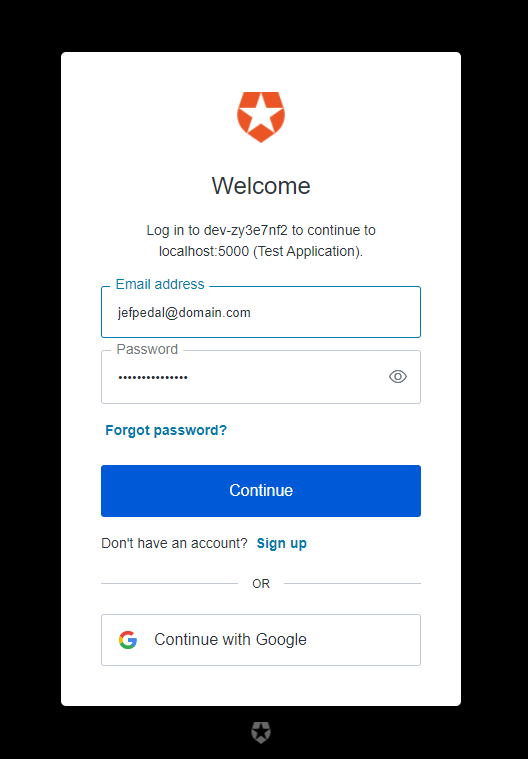
Automatisch gegenereerde beschrijving  
  
Het gebruik maken van de Auth0 Client Credentials Flow in onze console applicatie ging vrij vlot:  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Maar de frontend werkte precies nog niet zo goed mee. Na wat opzoekwerk op het internet vonden we dat we binnen onze applicaties op het Auth0 platform ook Allowed Origins kunnen toevoegen. Eens dit in orde was, werkte de frontend ook vlot met deze flow.  
  
  
  
Zoals te zien in onderstaande screenshot maakt de frontend nu ook gebruik van de Auth0 Client Credentials Flow:  
  
  
We merkten naderhand dat de identityserver volledig overtollig was geworden. We hebben dan ook de docker-compose.yml naar behoren aangepast, zodat deze service niet meer hoefde mee te draaien in onze applicatie:  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
  
  
**Ster 3: OpenID connect**   
  
  
Dit was voor ons de meest uitdagende van de opdrachten binnen Yellow Team. Er waren hierbij twee grote struikelblokken die we ondervonden tijdens het uitwerken van deze opdracht:  
  
1. De verscheidenheid aan opties om gebruik te maken van OpenID Connect in combinatie met Auth0 en .NET.  
2. Het meedraaien van de MVC app in docker.  
  
We hadden besloten om verder gebruik te maken van Auth0 en een oplossing te zoeken om binnen Auth0 gebruik te maken van OpenID Connect binnen onze MVC app. We kwamen hierbij uit op verschillende bronnen en verschillende manier om dit te implementeren:  
  
a) <https://auth0.com/docs/authenticate/identity-providers/enterprise-identity-providers/oidc>  
b) <https://auth0.com/docs/get-started/applications/configure-applications-with-oidc-discovery>  
c) <https://auth0.com/docs/quickstart/webapp/aspnet-owin/01-login>  
d) <https://auth0.com/docs/quickstart/webapp/aspnet-core-2/03-authorization>  
  
We hebben testjes opgezet aan de hand van al deze bronnen. We hebben allereerst een Open ID Connect verbinding aangemaakt op het Auth0 platform (bron a):  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
  
In een poging om deze OpenID connection te gebruiken in een MVC App hebben we de documentatie gevolgd in bron b. Aangezien deze documentatie het voorbeeld toont van een App\_Start\Startup.Auth.cs file, namen we aan dat dit moest geïmplementeerd worden in een .NET Entity Framework project. Tijdens het opzetten van deze test kwamen we dan uit bij de grote uitdaging om dit project laten mee te draaien in docker:  
  
Het probleem hierbij was dat hij binnen de linux container ging zoeken naar een Microsoft bestand, namelijk Microsoft.WebApplication.targets. Zoals vermeld in de csproj file van dit Entity Framework project.  
  
Na talloze uren zoeken en uittesten om toch te trachten dit project laten mee te draaien in docker, besloten we om opnieuw te beginnen en nieuwe testjes op te zetten. Zo kwamen we uit bij de laatste twee bronnen, waarbij bron d het dichtst aanleunde bij wat we nodig hadden. Dit voorbeeld toonde een simpele MVC app die gebruik ging maken van Auth0 OpenID Connect.  
  
We hadden ook reeds vastgesteld dat we binnen Auth0 users konden toevoegen aan een database die Auth0 voor ons beschikbaar stelt. Met behulp van deze users konden we ook authenticeren aan de hand van een Username-Password-Authentication, die standaard beschikbaar is binnen Auth0:  
Afbeelding met tekst

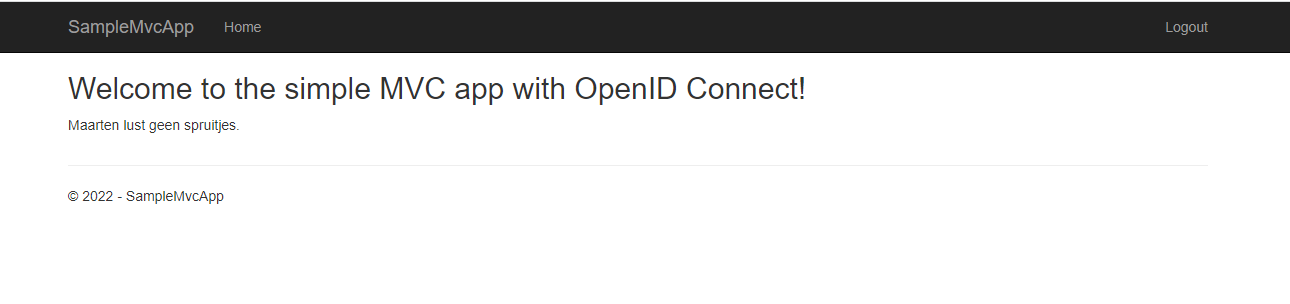
Automatisch gegenereerde beschrijving  
  
  
  
 We merkten op dat dit project, in het voorbeeld van bron d precies gebruik maakte van deze openID connectie:  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Door deze connectie te implementeren in onze MVC applicatie, kwam hetzelfde inlogscherm tevoorschijn om te kunnen authentiseren met onze app:  
  
  
Pas na het inloggen, werden we verder doorgeleid naar onze applicatie (zoals ook te zien in de afbeelding: Log in to … to continue to localhost:5000 (onze Applicatie).  
  
Om deze verbinding volledig correct te leggen moesten we binnen Auth0 ook nog enkele zaken instellen binnen onze applicatie, zoals bijvoorbeeld de allowed callback URLs en allowed logout URLs. Aangezien onze MVC app draait op localhost:3000, moesten we deze URLs als callback en logout toevoegen:  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Om de autorisatie volledig compleet te maken moesten we dan nog het Authorize attribuut toevoegen aan de HomeController, anders werd er niet gevraagd naar het externe inlogscherm van Auth0, omdat authenticatie natuurlijk niet nodig zou zijn.  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Na het succesvol inloggen met behulp van het tussenliggend Auth0 inlogscherm konden we dan terug worden omgeleid naar onze MVC app, zoals gedefinieerd in de callback URL, en de message zien op het home scherm : ‘Maarten lust geen spruitjes.’

  
Daar bovenop draaide dit project ook meteen vlot mee in docker:  
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
  
Omwille van deze redenen hebben we uiteindelijk besloten om verder te gaan met de documentatie uit bron d  
  
  
  
**Ster 4: OAuth Authorization Code Flow.**Om deze ster uit te voeren, konden we verder gaan op het voorbeeld uit bron d (ster 3).  
  
We hebben allereerst permissies toegevoegd aan onze Auth0 API:  
**Afbeelding met tekst

Automatisch gegenereerde beschrijving**  
  
We moesten ook deze instellingen aanzetten binnen onze API, om role based access control te activeren:  
**Afbeelding met tekst

Automatisch gegenereerde beschrijving**  
Omdat we enkel door deze stappen uit te voeren, er niet in slaagden om de permissies mee te sturen in het token besloten we om verder te werken met de rollen zoals vermeld in de documentatie.   
  
We maakten de rollen aan binnen Auth0 en wezen deze toe aan onze users:  
**Afbeelding met tekst

Automatisch gegenereerde beschrijving**  
  
Binnen de rollen kenden we de eerder aangemaakte permissies toe die we nodig hadden. Zoals we dit ook vaak zien in de praktijk. Autorisatie gebeurt vaak op basis van rollen, die eventueel ook geneste permissies kunnen bevatten, dus vonden we het een goed idee om hiermee verder te gaan.  
**Afbeelding met tekst

Automatisch gegenereerde beschrijving**  
**Afbeelding met tekst

Automatisch gegenereerde beschrijving**  
Jef Pedal kreeg de admin role toegewezen, met de advanced permission:  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
John Doe kreeg de User role toegewezen, met de basic permission:  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
Zo gingen we verder met de documentatie uit bron d, die ons vertelde dat we een rule moesten toevoegen binnen Auth0 zodat de rollen werden toegevoegd aan het Id Token om deze beschikbaar te stellen in onze backend:  
**Afbeelding met tekst

Automatisch gegenereerde beschrijving**  
  
Het was voor ons eerst onduidelijk waar we precies een rule konden aanmaken, dus volgden we ook nog volgende documentatie om ons even wegwijs te maken:  
<https://auth0.com/docs/customize/rules/create-rules>  
  
Daarna volstond het om gewoon een empty rule toe te voegen met de javascript code zoals vermeld in de documentatie:  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
We moesten hierbij in onze backend ook nog de OIDC authentication handler registration aanpassen om deze te informeren welke claim in het Token de informatie voor de rollen bevat:  
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
  
  
Vervolgens was het enkel nog een kwestie om de Authorize attributen aan te passen in de HomeController en in de nieuw aangemaakt AboutController.  
  
De HomeController werd toegankelijk gemaakt voor iedereen, en de AboutController enkel voor Jef Pedal (de admin).  
  
Afbeelding met tekst, oranje

Automatisch gegenereerde beschrijving  
  
Afbeelding met tekst, oranje, schermafbeelding

Automatisch gegenereerde beschrijving  
  
Afbeelding met tekst

Automatisch gegenereerde beschrijving  
  
  
