

**Software testing**

API

**Bart Meeremans**

bachelor – Toegepaste informatica

**Software testing**

Inhoudsopgave

[Intro 4](#_Toc92882113)

[Wat is mijn api? 5](#_Toc92882114)

[API gebruiken 6](#_Toc92882115)

[Get request 6](#_Toc92882116)

[Recipe 6](#_Toc92882117)

[Category 7](#_Toc92882118)

[Get request with ID 7](#_Toc92882119)

[Recipe 7](#_Toc92882120)

[Category 7](#_Toc92882121)

[Post request 8](#_Toc92882122)

[Recipe 8](#_Toc92882123)

[Category 9](#_Toc92882124)

[Put request 9](#_Toc92882125)

[Recipe 9](#_Toc92882126)

[Category 10](#_Toc92882127)

[Delete 10](#_Toc92882128)

[Recipe 10](#_Toc92882129)

[Category 10](#_Toc92882130)

[Tests recipe 11](#_Toc92882131)

[Simpele get test 11](#_Toc92882132)

[(cursus voorbeeld) 11](#_Toc92882133)

[(eigen api) 12](#_Toc92882134)

[Get test aan de hand van ID 14](#_Toc92882135)

[Post test 19](#_Toc92882136)

[Collection level tests 22](#_Toc92882137)

[Put test 23](#_Toc92882138)

[Delete test 24](#_Toc92882139)

[Collection runner 25](#_Toc92882140)

[Environments 25](#_Toc92882141)

[Post request 26](#_Toc92882142)

[Get request 26](#_Toc92882143)

[Put request 26](#_Toc92882144)

[Delete request 27](#_Toc92882145)

[Alle requests runnen 28](#_Toc92882146)

[Tests category 29](#_Toc92882147)

[Collection level tests 29](#_Toc92882148)

[Get test 30](#_Toc92882149)

[Post test 32](#_Toc92882150)

[Put test 33](#_Toc92882151)

[Delete test 34](#_Toc92882152)

[Collection runner 35](#_Toc92882153)

[Collection runner van een volledige crud(recipe/category) 39](#_Toc92882154)

[Resultaat in JSON 44](#_Toc92882155)

[Newman 67](#_Toc92882156)

[Newman instaleren 68](#_Toc92882157)

[Collection runnen via newman 68](#_Toc92882158)

# Intro

Voor software testing is het de bedoeling dat ik mezelf een manier van testing eigen ga maken. En om dit te bereiken moeten we zelf een cursus zoeken met betrekking tot een testing manier die ons interesseert.

Persoonlijk vindt ik API’s wel interessant en dus heb ik gekozen om met postman te testen, mede omdat ik vorig jaar voor het vak business application development een c# rest applicatie heb gemaakt die ik nu dan zou kunnen testen.

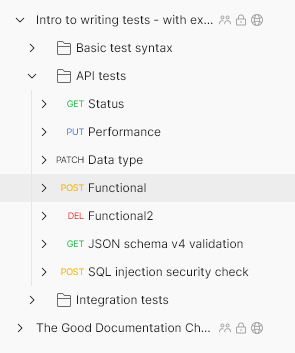
Als cursus heb ik gekozen voor de cursus die postman zelf aanbied.

<https://www.postman.com/webinars/continuous-testing/>

Maar Ik haal ook wat informatie uit de cursus die door de docent is aangeboden.

Om te beginnen heb ik natuurlijk postman geïnstalleerd ik heb hier gekozen voor de desktop applicatie omdat ik de browser omgeving minder overzichtelijk vindt.

Ik ben begonnen met het openen van een postman template die een basis introductie geeft tot testing. (intro to writing tests).



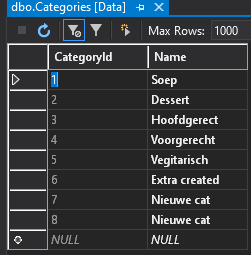
Vervolgens heb ik zelf een workspace aangemaakt voor het testen van mijn eigen api.

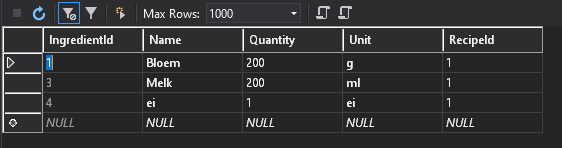
# Wat is mijn api?

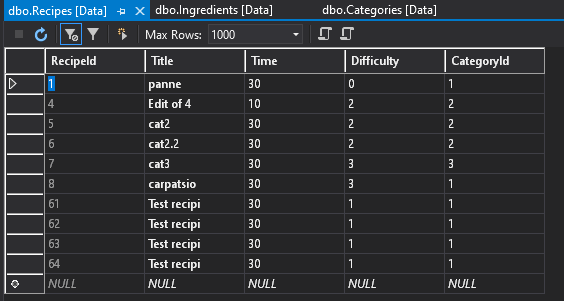
De api die ik ga testen noemt recipe API. Het is een programma die ik als opdracht in 2ti heb gemaakt. De API werkt met entity framework die recepten in een database opslaat.



De database bestaat uit 3 tabelen: recipes, categories en ingredients.







Er zijn ook 2 controllers aanwezig namelijk voor category en recipe. Deze beschikken over de CRUD operaties en deze kan ik dus ook gaan testen.



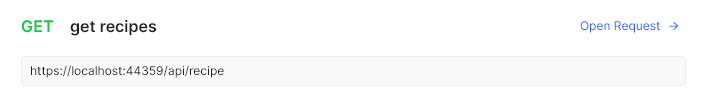
# API gebruiken

Om te beginnen heb ik een collection aangemaakt in postman waar ik mijn CRUD operaties kan uitvoeren.

In postman is het mogelijk om API documentatie te laten generen dit heb ik dan ook gedaan voor het verduidelijken van mijn API.

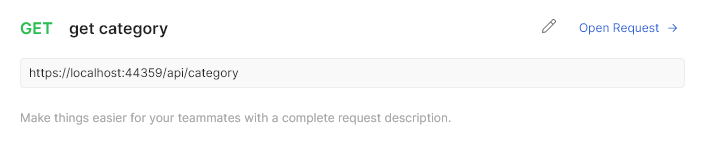
## Get request

### Recipe



Wanneer ik een get request doe van recipe krijg ik een lijst van de recepten met hun respectievelijke info

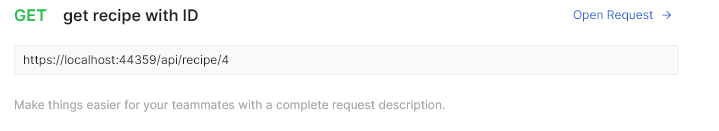
### Category



Ook bij een get van category krijgen we een lijst van alle categorieën.

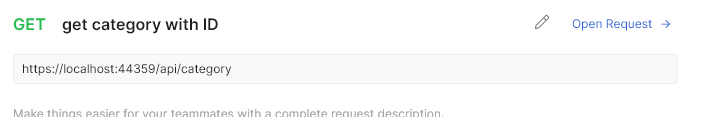
## Get request with ID

### Recipe



Wanneer we een get request doen met een ID in de link zal de api het opgevraagde recept returnen in een JSON formaat.

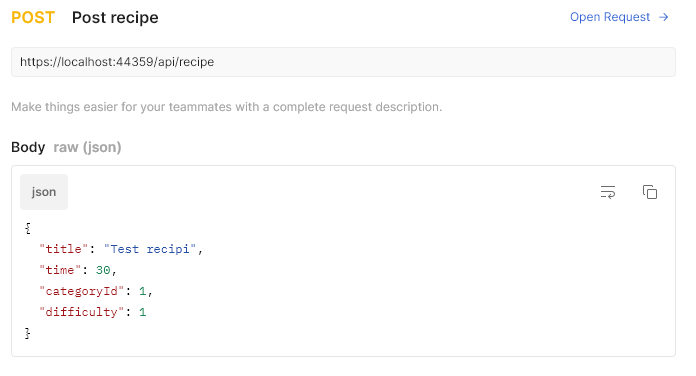
### Category



Wanneer we een get request doen met een ID in de link zal de api de opgevraagde categorie returnen in een JSON formaat.

## Post request

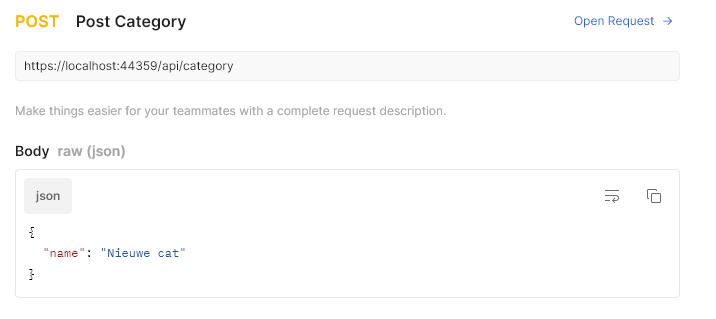
### Recipe



Voor het aanmaken van een recept moeten we een body meegeven in een JSON formaat.

Als response returned de API het aangemaakte object in een JSON formaat.

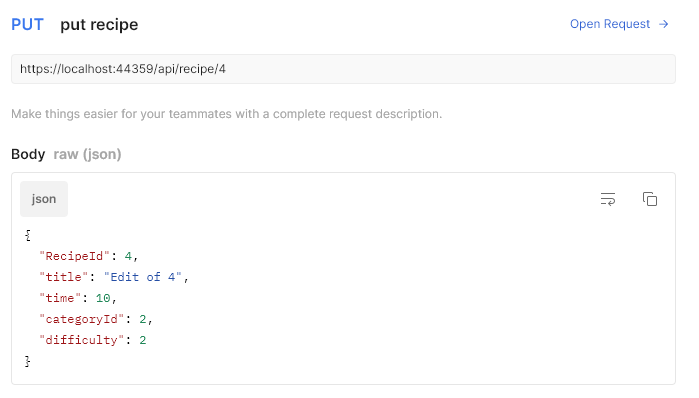
### Category



Als response ontvangen we het aangemaakt object in een JSON formaat.

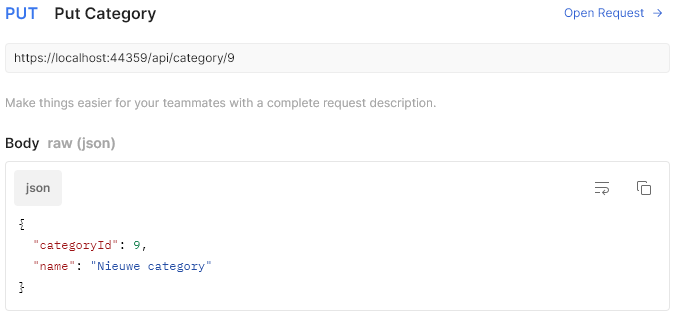
## Put request

### Recipe



Respone is een JSON object dat we net hebben aangemaakt.

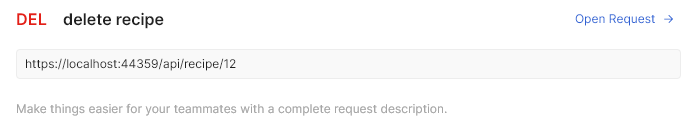
### Category



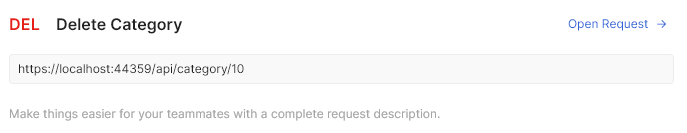
Response is een no content 204 code.

## Delete

### Recipe



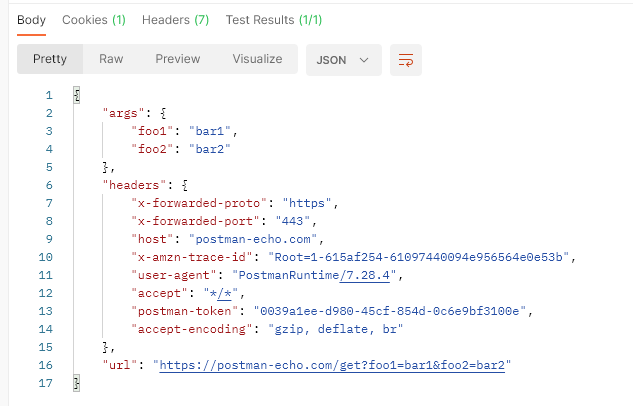
### Category

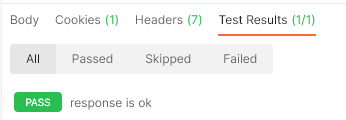


# Tests recipe

## Simpele get test

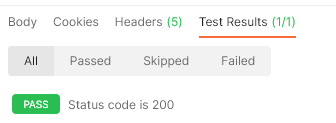
### (cursus voorbeeld)

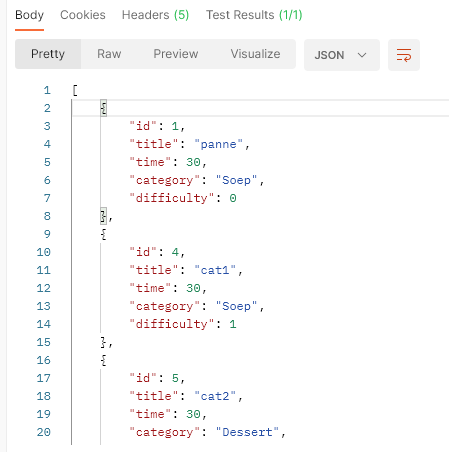


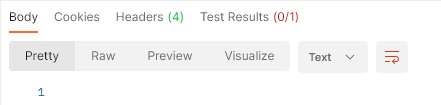
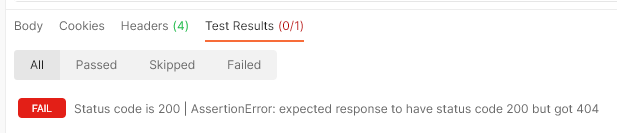


### (eigen api)

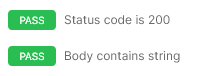
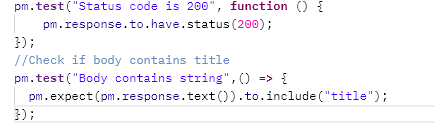
Wanneer we hier een get request zullen doen zal de test gaan controlleren of de status code 200 wordt terug gegeven. Deze status code wilt zeggen dat een request gelukt is. Aan de hand van deze test kunnen we dus gaan nakijken of we wel verbinding hebben met de API en of onze URL wel klopt.



Indien ik de URL verander naar iets foutief faalt de test.  

We kunnen ook andere voorwaarden schrijven in de test, voor het volgende voorbeeld stel ik ook de eis dat het opgehaalde resultaat een title zal bevaten.



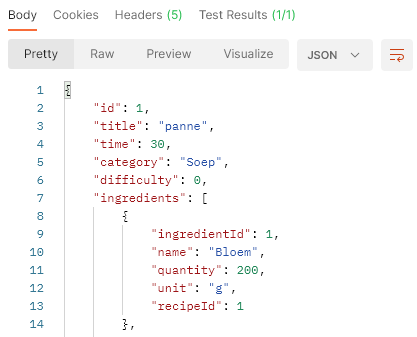
Wanneer we bijvoorbeeld eisen dat de body van het opgevraagde object een woonplaats bevat, zal de test een fail teruggeven.

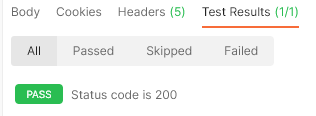


## Get test aan de hand van ID

Net zoals de vorige test gaan we eerst gaan kijken of we wel een object kunnen opvragen aan de hand van een status code namelijk 200.

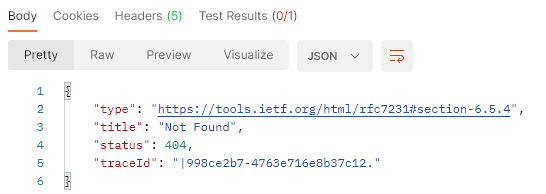
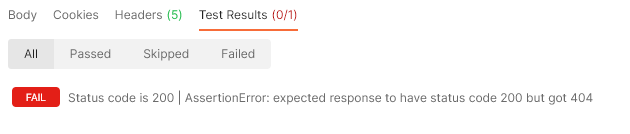






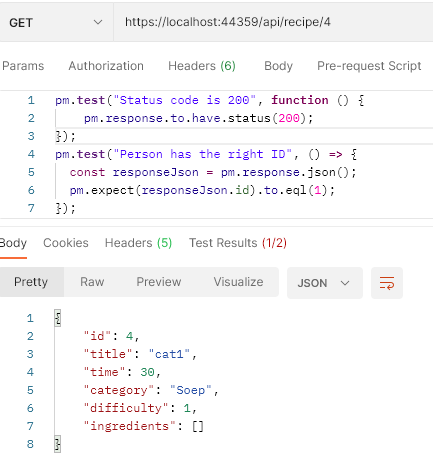
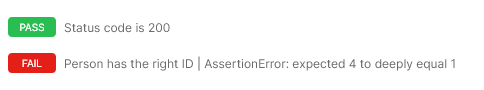
Wanneer ik een ID meegeef dat niet bestaat zal de test foutief zijn.



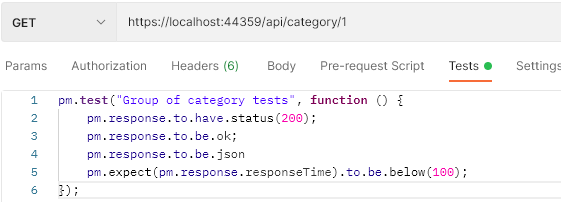
Nu moeten we wel nog gaan controlleren of we wel degelijk het juist object terug krijgen. Dit kunnen we doen door na te gaan of het ID van het gereturnede object wel het zelfde is dan het ID In de URL.

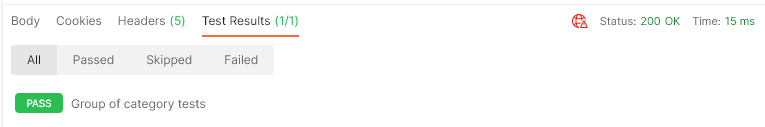


Wanneer we bijvoorbeeld in de URL object 4 gaan opvragen, maar in de test schrijf ik dat ik 1 verwacht zal de test falen.

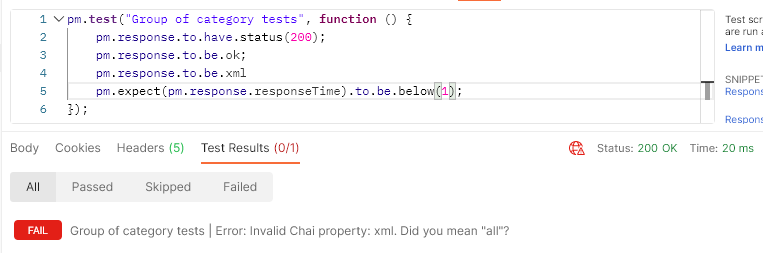
We kunnen ook zien dat het wel gelukt is om het object op te halen door de status code. Maar het is wel niet het ID dat de 2de test verwahcte.

We hebben ook de mogelijkheid om meerdere voorwaarden te schrijven in 1 enkele test.

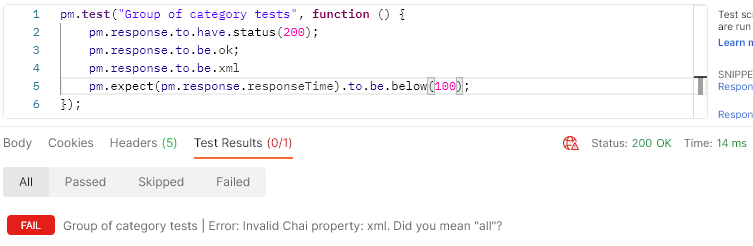


Bij deze test zal een categorie opgevraagd worden, er wordt gekeken naar de status code, of de request gelukt is maar ook of de response wel in het verwachte resultaat is. Als laatste gaan we ook nog kijken of de response wel snel gebeurt. 

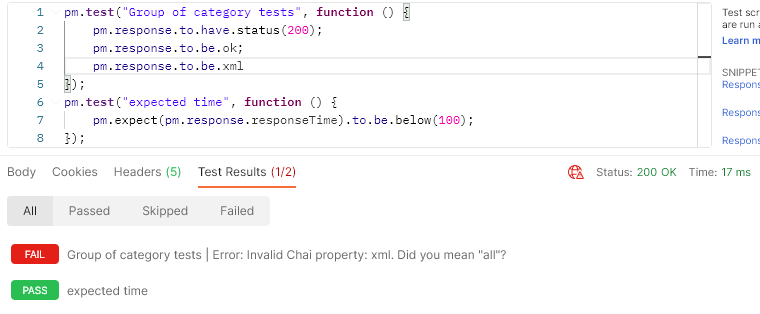
We kunnen in dit geval zien dat de tests geslaagd zijn omdat de status 200 ok is en ook de response time is 15ms.



In dit geval failed de test omdat de time 20ms is wat dus meer is dan 1, maar er is nog een probleem. In deze test wordt namelijk XML verwacht maar de api returned json en toch wordt dit niet in de fail code gezegt. Dit komt omdat de time test als laatste gebeurt



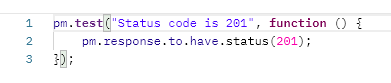
Indien we de expected time wel mogelijk maken gaan we zien dat we nu wel zien dat het foute formaat teruggestuurd wordt.

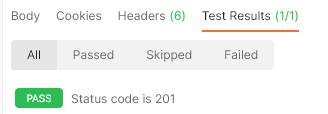
Hieruit kan je concluderen dat testen in 1 enkele group zetten ook risicos heeft. 

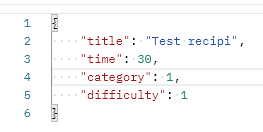
Dit is dus een betere optie omdat we nu duidelijk kunnen zien wat er gaat faalen. Het heeft echter ook geen nut om de response statusen te splitsen omdat deze oftewel allemaal gaan werken of juist helemaal niet.

## Post test

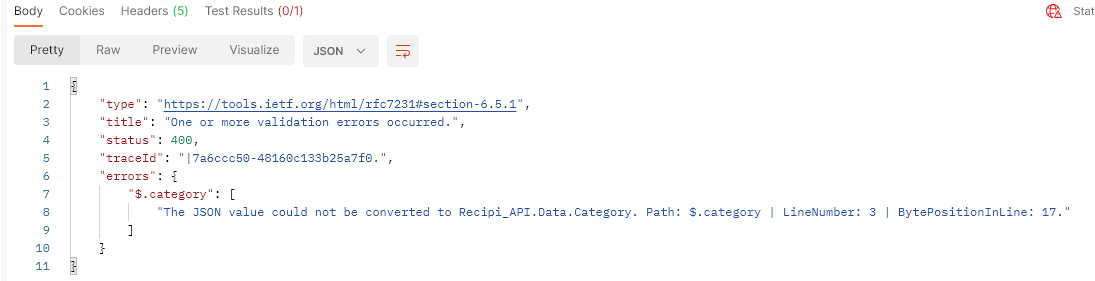


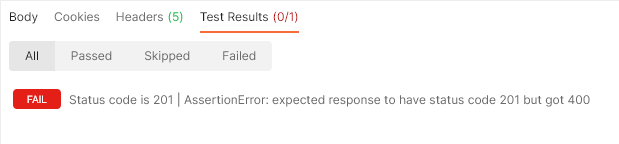


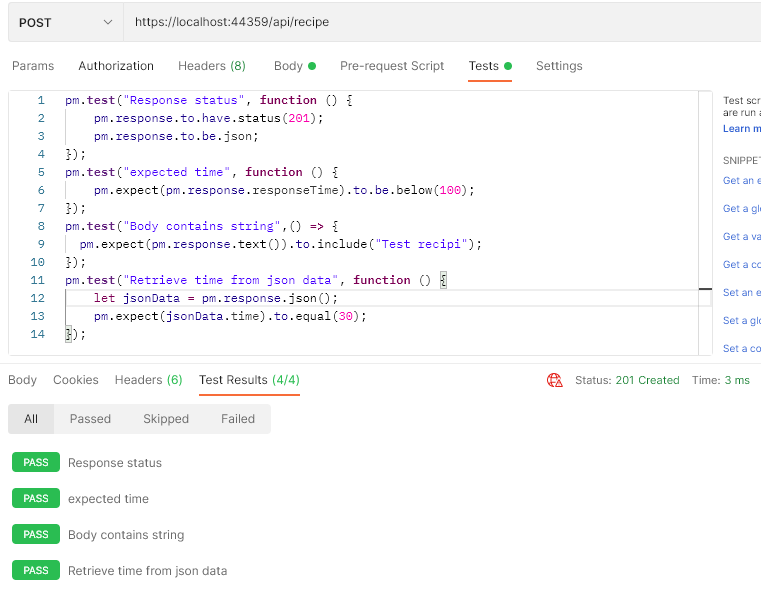
Indien een gegeven body onvolledig of fout is returnt de test een fout. 

In de body is category meegegeven inplaats van categoryId





Ook deze testen kunnen we gaan uitbreiden.



Hier gaan we dus controleren het request wel kan worden uitgevoerd dit doe ik in de groep “reponse status”. Vervolgens wordt er gekeken of het request wel in minder dan 1s gebeurt.

En in de laatste 2 groepen kan je de 2 verschillende manieren zien om gegevens na te kijken die zijn toegevoegd.

In de groep “Body contains string” ga ik gewoon kijken of er in de response tekst is waar “Test recipi” in voorkomt. Ik zou hier dus ook bijvoorbeeld “T” kunnen zetten en van zodra de response body een t bevat zal de test slagen .

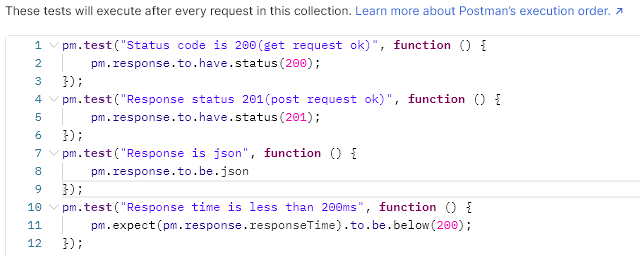
In de group “Retrieve time from json data” kunnen we echter gebruik gaan maken van JSON. Eerst maak ik een json object aan en stel ik het gelijk aan de ontvangen data. Hier kan ik dan ook mee gaan werken, ik zal gaan kijken of het json object een variabelen “time” bevat. En dan vervolgens ook kijken of deze gelijk is aan 30.

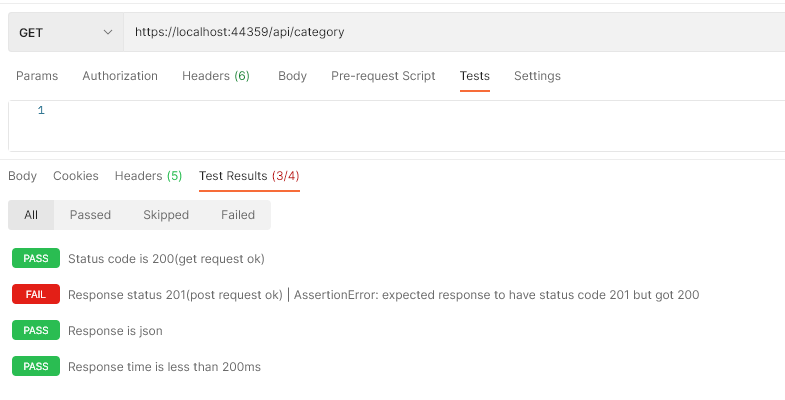
In de 2de methode zal je dus egt een exacte waarde gaan verwachten terwijl in het eerste voorbeeld je value er gewoon moet in voorkomen.

Er is wel nog 1 probleem, namelijk deze verwachte waarden zijn hard coded.

## Collection level tests

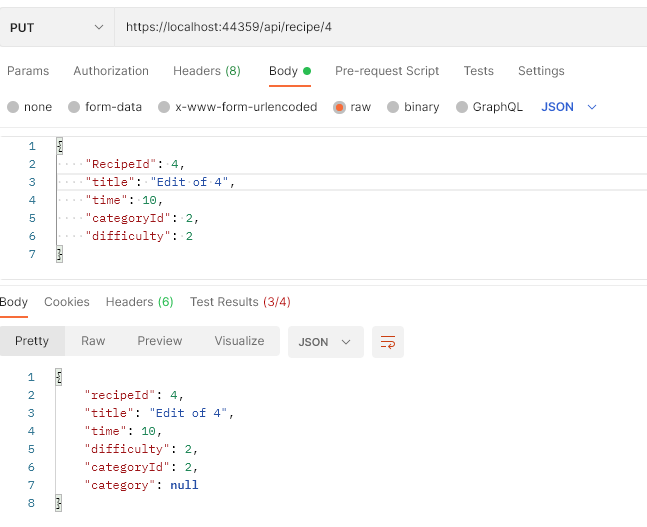
Er zijn natuurlijk tests die we altijd willen gaan uitvoeren. Bijvoorbeeld kijken welke status code we terug krijgen, of controlleren welk bestandsformaat gereturned wordt. Het is in postman echter mogelijk om tests te gaan schrijven op een collection niveau.

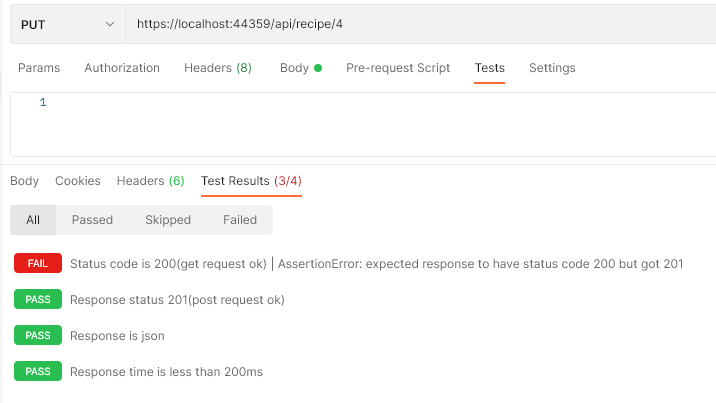




Wanneer ik nu een api call ga doen in mijn eigen collection zullen de 4 tests automatisch worden gedaan. Dit is handig voor testen die we steeds zullen nodig hebben.

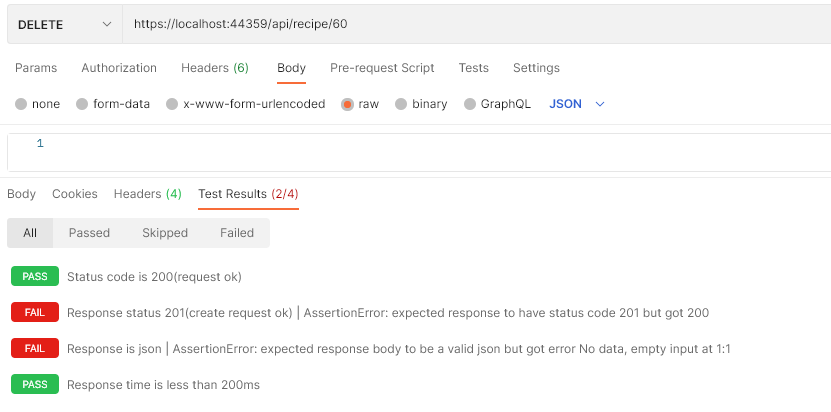
## Put test

Voor het zenden van een pout request zal er in de link een ID worden verwacht. En in de body worden de nieuwe gegevens verwacht.



En ook al heb ik zelf geen tests geschreven, toch zal ik test results terug krijgen dankzij de tests die ik op collection niveau heb geschreven.

## Delete test



Ook hier zullen door de collection tests de 4 default tests worden uitgevoerd.

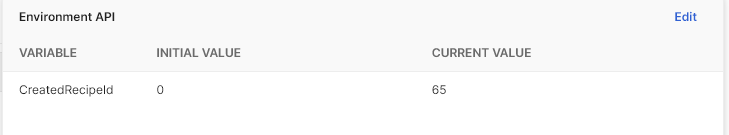
## Collection runner

Dit is een handige tool die ons toelaat om bepaalde tests te gaan uitvoeren rechtstreeks achter elkaar. Dit is handig wanneer we bijvoorbeeld willen testen of ik zo wel een recipe kan aanmaken, vervolgens dit kan updaten en tot slot ook kan verwijderen.

Dit is al een eerste kleine vorm van API testing automization omdat we ons aangemaakte object gaan bijhouden en vervolgens met deze data gaan werken

### Environments

Om te beginnen ben ik een environment gaan aanmaken in postman die ik vervolgens ga gebruiken om een ID op te slaan van een Recipe dat ik net gemaakt heb.



### Post request



Om te beginnen maak ik een nieuwe variabelen aan die het ontvangen JSON object gaat lezen.

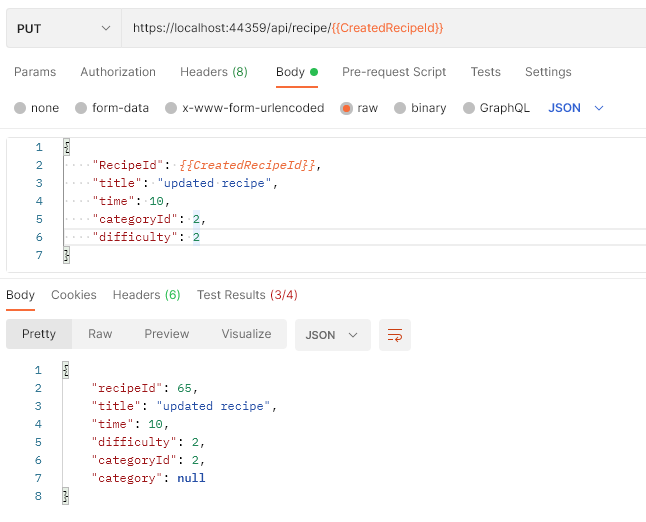
Vervolgens zet ik in de environment de variabelen gelijk aan de ontvangen data zijn ID.

### Get request

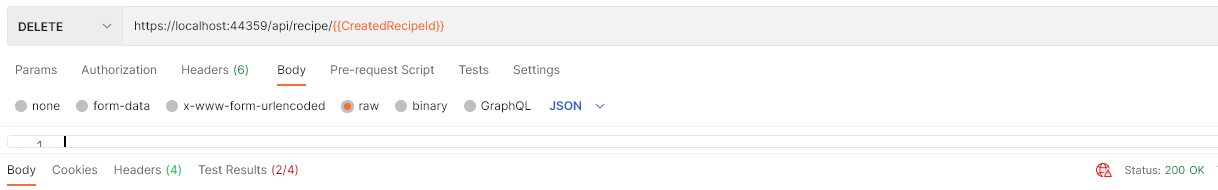


Nu kunnen we gebruik maken van het CreatedRecipeId om het nieuwe aangemaakte object te gaan opvragen met een get request.

### Put request

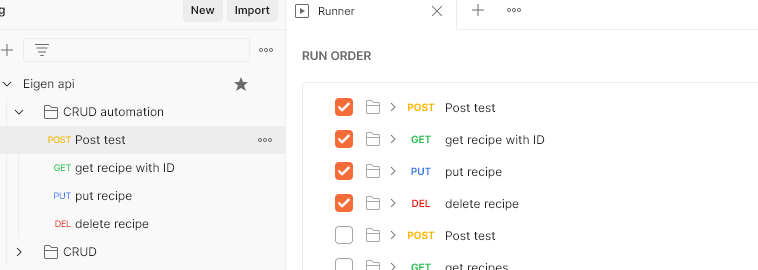
Deze variabelen kunnen ook gebruikt worden in de body van een request. In dit geval zal ik een put request doen op het laatst aangemaakte object.

### Delete request

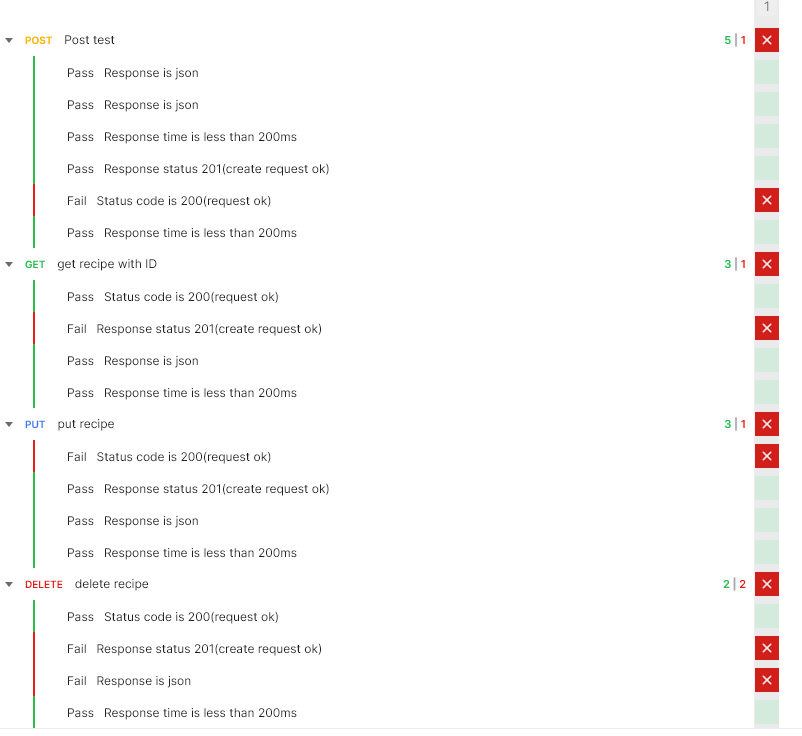


Als laatste in de crud operaties hebben we enkel nog het delete request, dit gaan we dan ook testen met ons aangemaakte object. Ook hier kunnen we da environment variabele gebruiken.

### Alle requests runnen



Vervolgens heb ik in de collection runner de 4 tests geselecteerd.

Dit is het resultaat dat we bekomen.

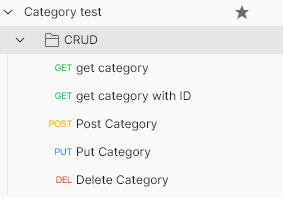
We kunnen hier zien dat de tests die horen te slagen gelukt zijn.

De gefaalde tests zijn tests die we verwachten te falen zo zal bijvoorbeeld een post request enkel verwachten een status code 201 te ontvangen en niet 200.

Op deze manier kunnen we eenvoudig en overzichtelijk gaan controleren of we een crud operatie kunnen uitvoeren. En dit op een geautomatiseerde manier.

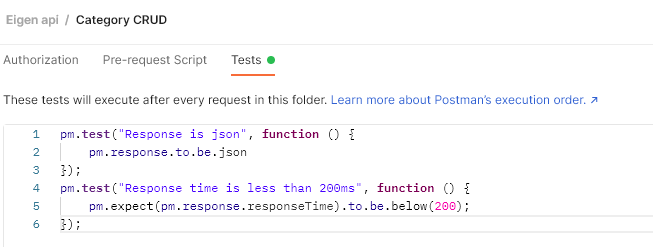
# Tests category

Om te beginnen ben ik een nieuwe folder gaan aanmaken voor de testen van category.

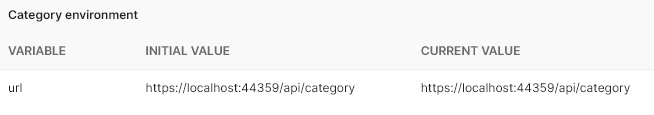


## Collection level tests

Ook bij categorie zullen er enkele testen zijn die we voor alle request kunnen doen. Voor deze is het dan ook makkelijk om ze op een collection niveau te schrijven zodat we ze niet moeten over typen voor alle testen. Voor alle tests gaan we nagaan of ze in minder dan 200ms kunnen gerealiseerd worden en of ze geretourneerd worden als JSON.



Een andere manier om herhaling te verminderen is door een environment aan te maken en hier de URL te vermelden zodat we ook deze makkelijk kunnen hergebruiken zonder het te moeten herhalen.



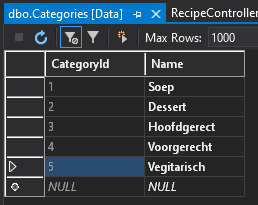
## Get test

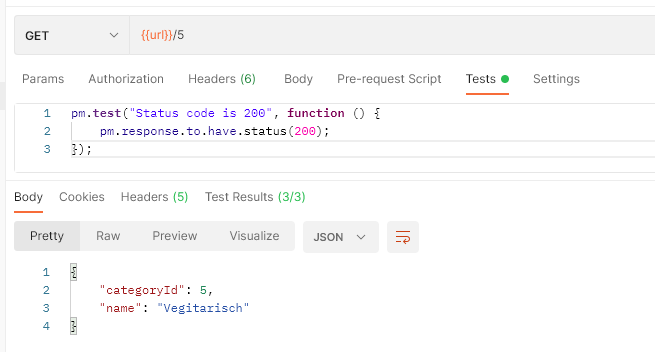


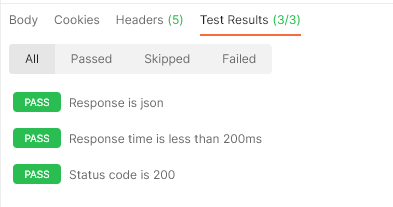
Door de automatiseringen die ik op collection niveau heb gedaan kan ik nu door enkel {{url}} te typen gaan testen of een get van categorie succesvol is en gebeurd in minder dan 200ms en of dit een JSON returned.

Het enige wat ik zelf nog heb toegevoegd is een controle van de status code. We kunnen ook zien dat het request wel degelijk de juiste informatie opvraagt.

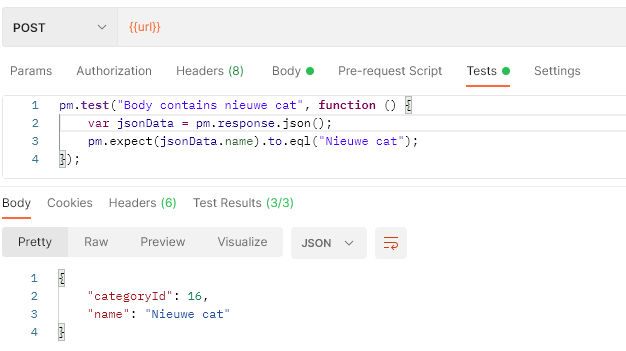


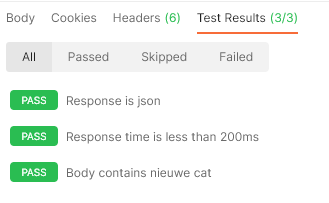


Ook een get request aan de hand van een ID kan makelijk getest worden. 

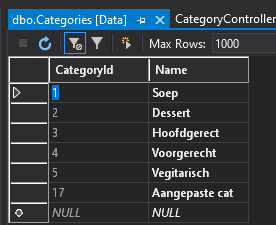


## Post test

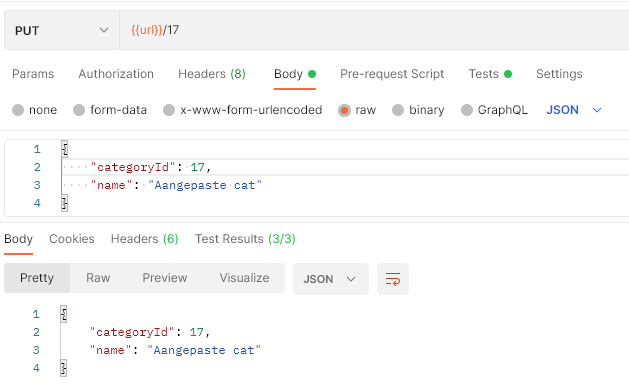


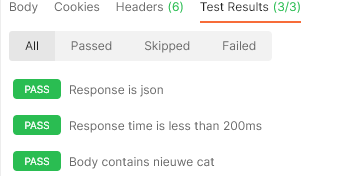


Ook bij de post test kunnen we onze URL variabele gebruiken. En ook de collection tests zullen automatisch worden uitgevoerd, ik heb dan ook nog een test bijgeschreven die nagaat of in de gereturnde JSON de name wel overeenkomt met de name dat we willen.

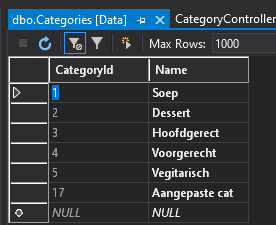


## Put test

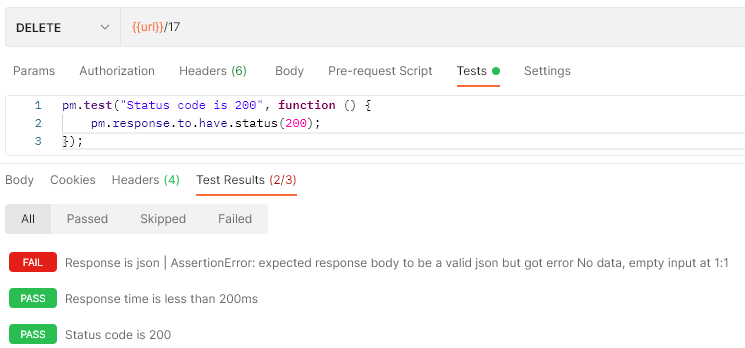




Ook hier is door het voorbereidend werk amper werk om de test uit te voeren. Door de URL variabelen en de collection tests.

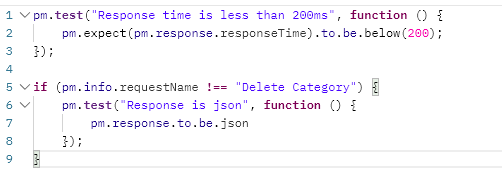


## Delete test



Wanneer we de tests gaan runnen kunnen we ook wel zien dat de collection tests hier niet zullen werken. Dit komt omdat de delete call geen data returned. Een oplossing zou zijn om de delete request in een andere folder te stoppen maar dit kan naar mijn mening wat verwarrend zijn.

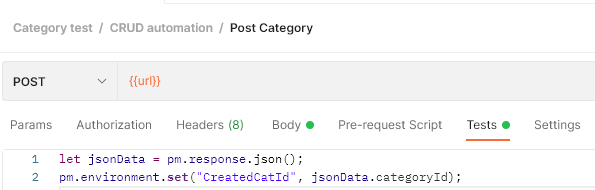
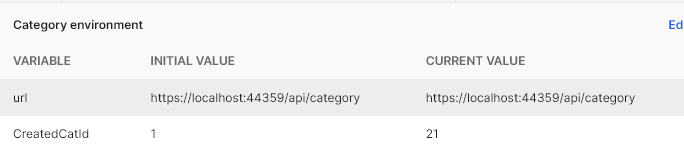
Een betere oplossing in dit geval is gaan zeggen in de collection tests dat de json test enkel moet gebeuren bij de tests die niet Delete Category noemen.



Wanneer we nu de delete test runnen zal de json test niet uitgevoerd worden.

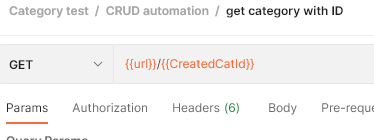


## Collection runner

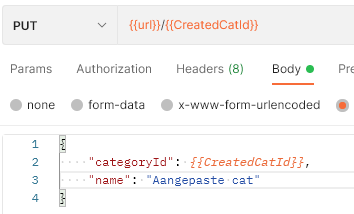
Om te beginnen zal ik eerst een environment variabelen moeten aanmaken om het ID van de aangemaakte categorie op te slaan. 

Vervolgens slaag ik het ID op van een aangemaakte categorie die ik vervolgens kan gebruiken in de volgende tests.

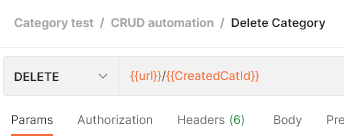
Opvragen van het net gecreerde object.



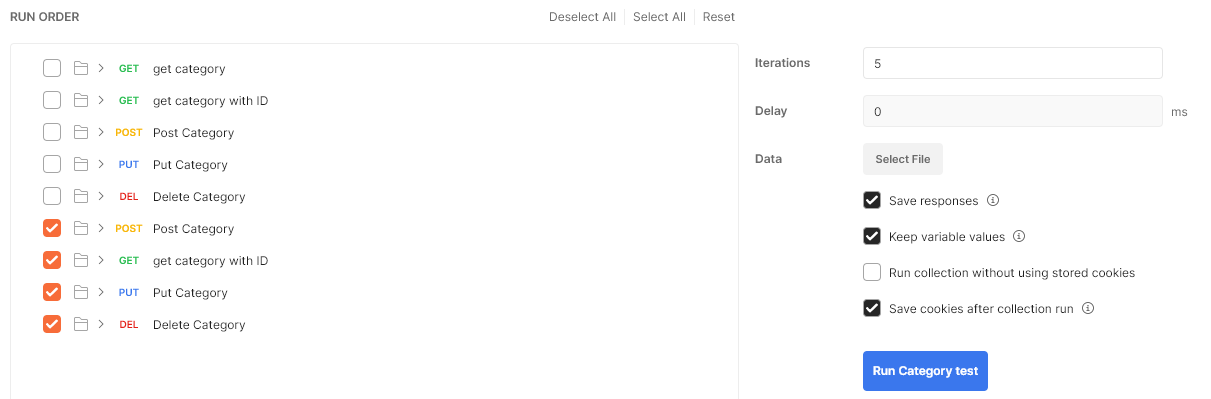
Aanpassen van het object



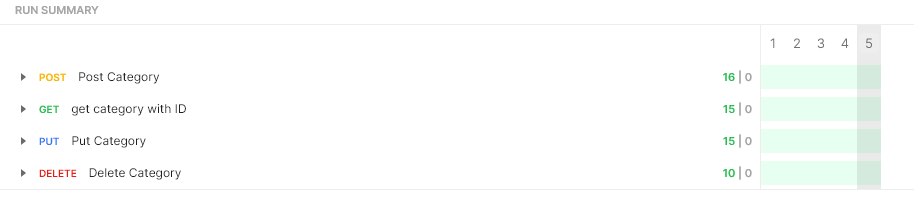
Deleten



Vervolgens gaa ik de collection 5x uitvoeren en telkens de uitslag opslaan.



Hier kan ik zien dat de testen 5x gelukt zijn



Onder view summary kan ik de 5 iteraties gaan inspecteren samen met de response bodies

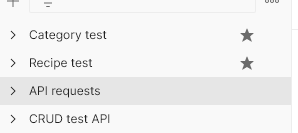
Zo kan ik vaststellen dat alles is geslaagd zonder problemen.

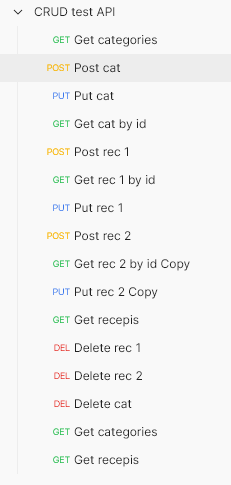


# Collection runner van een volledige crud(recipe/category)

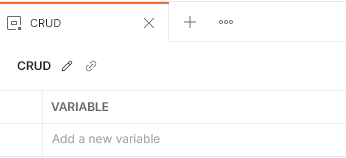
In dit onderdeel ga ik een volledige cyclus doorlopen van het aanmaken van een nieuwe categorie en hier ook wat gerechtjes aan toevoegen. Met de kennis die ik verkregen heb door de vorige onderdelen van deze paper zou dit geen probleem mogen zijn.

Om mijn omgeving probeer en georganiseerd te houden heb ik voor deze tests een nieuwe collectie aangemaakt namelijk CRUD test API.

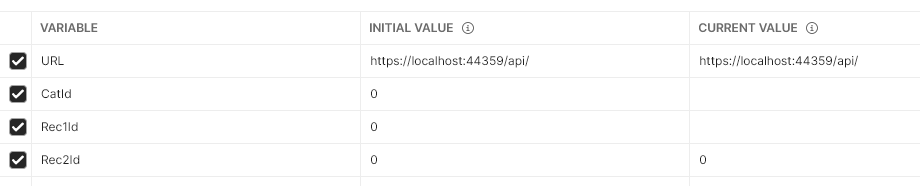


Vervolgens ben ik gaan na denken welke request tests ik allemaal kan uitvoeren en die heb ik dan ook allemaal aangemaakt. Momenteel zijn deze requests nog leeg maar op deze manier heb ik wel al een duidelijk overzicht van wat ik allemaal moet implementeren.

Het is natuurlijk mijn bedoeling om zo snel en efficiënt mogelijk te werken, en om dit te kunnen doen zal ik ook een environment nodig hebben.

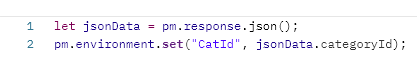
ook deze is initieel leeg en zal ik aanvullen wanneer nodig.

Enkele variabelen die ik zal nodig hebben is natuurlijk de basis URL, het id van de aangemaakte categorie en van de recepten.



De volgende stap is om elke request te gaan invullen dit aan de hand van de environment variabelen.



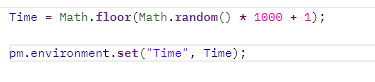
Ik moet antuurlijk ook het ID opslaan wanneer ik iets aanmaak. 



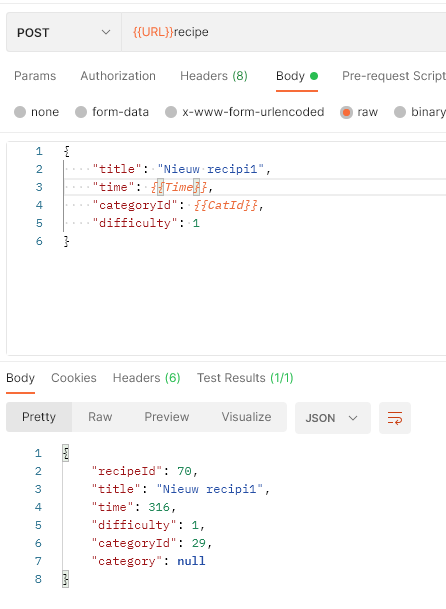
En dit vervolgens gebruiken. Dit heb ik gedaan voor elke request in de collection.

Momenteel heb ik dus nog geen tests geschreven in deze collection.

Voor de tijd van een recept gaa ik gebruik maken van de pre request script. Dit heb ik tot nu toe nog niet gebruikt, maar met deze feature kunnen we code gaan utivoeren alvorens een request gedaan wordt.



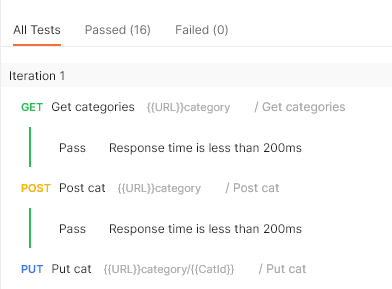
In dit geval wordt een random waarde tussen 1 en 1000 gecreëerd en vervolgens ingevuld in de variabele Time die ik heb toegevoegd aan de environment.



Op deze manier heb ik nu een recept aangemaakt dat bij de vorige categorie hoort met een random tijdsduur.

Wanneer ik elke request heb ingevuld met de juiste waarden heb ik een simpele response test geschreven op collection niveau. Dit heb ik gedaan zodat ik eens kan gaan kijken of elke request welk wordt uitgevoerd.

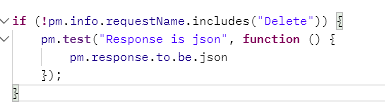




Elke test is gelukt dus dit wil ook zeggen dat elke request is gelukt.

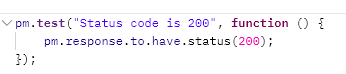
Nu is het tijd om wat betere tests te gaan schrijven om alles grondiger te testen.

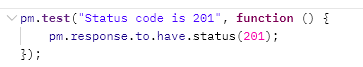
Net zoals de vorige collection runne zou ik hier graag nagaan of er wel json gereturned wordt. Dit zou het geval moeten zijn voor 15 van de 16 testen behalve bij de delete van categorie. Dus ook deze test zou ik graag op collection niveau doen.

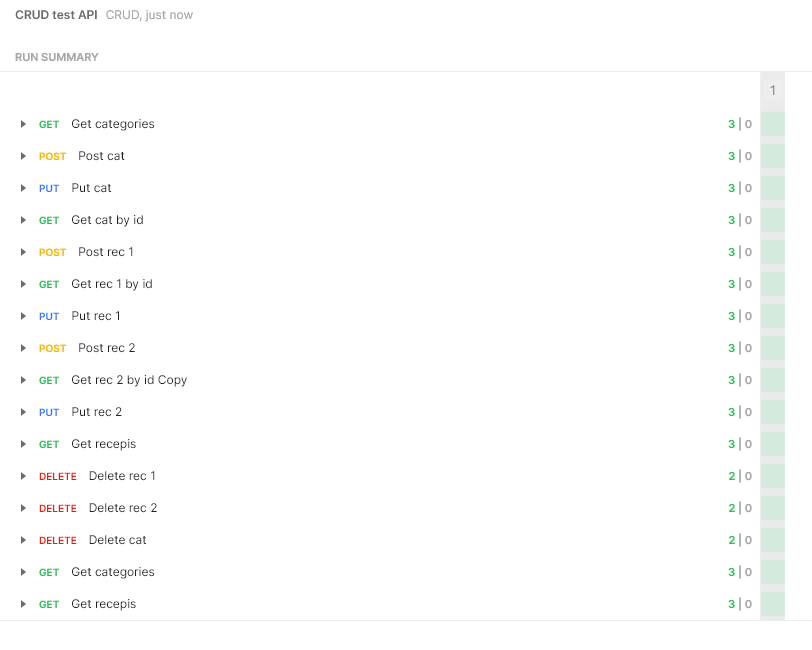


Zodat dit geen fout zou opleveren heb ik wat code geschreven dat zegt dat deze test enkel gedaan moet worden wanneer het niet gaat over een delete request.

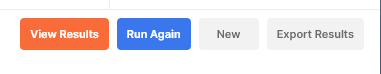
Vervolgens heb ik bij elke test gekeken of de status code correct is. Bij create en put moet dit 201 returnen. En bij get en delete 200.







De collection blijkt te slagen samen met alle tests.



Het is ook mogelijk om dit te exporteren naar een json file.

## Resultaat in JSON

Op deze manier is het makkelijk om all de tests en requests weer teg even samen met de response.

{

"id": "2dd3e31f-2abd-4899-ba36-3c71218ad9b3",

"name": "CRUD test API",

"timestamp": "2021-12-09T11:43:34.351Z",

"collection\_id": "15613414-9d16102b-a6bb-465f-9dec-b47c92697535",

"folder\_id": 0,

"environment\_id": "16c1249a-7000-49c8-812f-a5d025c04907",

"totalPass": 45,

"totalFail": 0,

"results": [

{

"id": "3e155315-c3a3-4ac0-adeb-09e063920393",

"name": "Get categories",

"url": "https://localhost:44359/api/category",

"time": 14,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

14

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

}

]

},

{

"id": "2f834540-50f9-4f20-a5f9-7c20217482cc",

"name": "Post cat",

"url": "https://localhost:44359/api/category",

"time": 7,

"responseCode": {

"code": 201,

"name": "Created"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 201": {

"pass": 1,

"fail": 0

}

},

"times": [

7

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

}

]

},

{

"id": "04d3dc5d-b225-4d71-b69e-9ec17855ac02",

"name": "Put cat",

"url": "https://localhost:44359/api/category/33",

"time": 5,

"responseCode": {

"code": 201,

"name": "Created"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 201": {

"pass": 1,

"fail": 0

}

},

"times": [

5

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

}

]

},

{

"id": "e9eef9ea-9cc7-45ae-b689-5143512c97b8",

"name": "Get cat by id",

"url": "https://localhost:44359/api/category/33",

"time": 4,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

4

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

}

]

},

{

"id": "11469ded-75b6-46fd-840c-7cd5a4fa2c07",

"name": "Post rec 1",

"url": "https://localhost:44359/api/recipe",

"time": 4,

"responseCode": {

"code": 201,

"name": "Created"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 201": {

"pass": 1,

"fail": 0

}

},

"times": [

4

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

}

]

},

{

"id": "5716bea9-e357-4e6c-b70a-5a0ea026daed",

"name": "Get rec 1 by id",

"url": "https://localhost:44359/api/recipe/78",

"time": 50,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

50

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

}

]

},

{

"id": "cecfb919-3a5b-404b-8650-364b336ef506",

"name": "Put rec 1",

"url": "https://localhost:44359/api/recipe/78",

"time": 6,

"responseCode": {

"code": 201,

"name": "Created"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 201": {

"pass": 1,

"fail": 0

}

},

"times": [

6

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

}

]

},

{

"id": "d2ebba16-d143-4b16-b2ff-a9b54993c9d7",

"name": "Post rec 2",

"url": "https://localhost:44359/api/recipe",

"time": 5,

"responseCode": {

"code": 201,

"name": "Created"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 201": {

"pass": 1,

"fail": 0

}

},

"times": [

5

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

}

]

},

{

"id": "e3d0f755-fcf3-4cde-9b05-1296a75b0f17",

"name": "Get rec 2 by id Copy",

"url": "https://localhost:44359/api/recipe/79",

"time": 4,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

4

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

}

]

},

{

"id": "570ac58c-92f5-4a6a-b403-2d924569817f",

"name": "Put rec 2",

"url": "https://localhost:44359/api/recipe/79",

"time": 8,

"responseCode": {

"code": 201,

"name": "Created"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 201": {

"pass": 1,

"fail": 0

}

},

"times": [

8

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 201": true

}

]

},

{

"id": "a5662892-2e91-4150-9edc-f8f341878f41",

"name": "Get recepis",

"url": "https://localhost:44359/api/recipe",

"time": 3,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

3

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

}

]

},

{

"id": "ff5ffa23-f779-4e59-a8c6-eb45486eb25f",

"name": "Delete rec 1",

"url": "https://localhost:44359/api/recipe/78",

"time": 3,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

3

],

"allTests": [

{

"Response time is less than 200ms": true,

"Status code is 200": true

}

]

},

{

"id": "1fcf7da4-616d-4830-b2cc-13214938e37d",

"name": "Delete rec 2",

"url": "https://localhost:44359/api/recipe/79",

"time": 4,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

4

],

"allTests": [

{

"Response time is less than 200ms": true,

"Status code is 200": true

}

]

},

{

"id": "1d67f7a7-f5ff-4a47-8f1f-c8f8cbb2bab4",

"name": "Delete cat",

"url": "https://localhost:44359/api/category/33",

"time": 4,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

4

],

"allTests": [

{

"Response time is less than 200ms": true,

"Status code is 200": true

}

]

},

{

"id": "71938eca-0fd3-47b6-944d-94e06328729d",

"name": "Get categories",

"url": "https://localhost:44359/api/category",

"time": 3,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

3

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

}

]

},

{

"id": "43af6a31-af57-4b50-8ef3-82e2ca257067",

"name": "Get recepis",

"url": "https://localhost:44359/api/recipe",

"time": 5,

"responseCode": {

"code": 200,

"name": "OK"

},

"tests": {

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

},

"testPassFailCounts": {

"Response time is less than 200ms": {

"pass": 1,

"fail": 0

},

"Response is json": {

"pass": 1,

"fail": 0

},

"Status code is 200": {

"pass": 1,

"fail": 0

}

},

"times": [

5

],

"allTests": [

{

"Response time is less than 200ms": true,

"Response is json": true,

"Status code is 200": true

}

]

}

],

"count": 1,

"totalTime": 129,

"collection": {

"requests": [

{

"id": "3e155315-c3a3-4ac0-adeb-09e063920393",

"method": "GET"

},

{

"id": "2f834540-50f9-4f20-a5f9-7c20217482cc",

"method": "POST"

},

{

"id": "04d3dc5d-b225-4d71-b69e-9ec17855ac02",

"method": "PUT"

},

{

"id": "e9eef9ea-9cc7-45ae-b689-5143512c97b8",

"method": "GET"

},

{

"id": "11469ded-75b6-46fd-840c-7cd5a4fa2c07",

"method": "POST"

},

{

"id": "5716bea9-e357-4e6c-b70a-5a0ea026daed",

"method": "GET"

},

{

"id": "cecfb919-3a5b-404b-8650-364b336ef506",

"method": "PUT"

},

{

"id": "d2ebba16-d143-4b16-b2ff-a9b54993c9d7",

"method": "POST"

},

{

"id": "e3d0f755-fcf3-4cde-9b05-1296a75b0f17",

"method": "GET"

},

{

"id": "570ac58c-92f5-4a6a-b403-2d924569817f",

"method": "PUT"

},

{

"id": "a5662892-2e91-4150-9edc-f8f341878f41",

"method": "GET"

},

{

"id": "ff5ffa23-f779-4e59-a8c6-eb45486eb25f",

"method": "DELETE"

},

{

"id": "1fcf7da4-616d-4830-b2cc-13214938e37d",

"method": "DELETE"

},

{

"id": "1d67f7a7-f5ff-4a47-8f1f-c8f8cbb2bab4",

"method": "DELETE"

},

{

"id": "71938eca-0fd3-47b6-944d-94e06328729d",

"method": "GET"

},

{

"id": "43af6a31-af57-4b50-8ef3-82e2ca257067",

"method": "GET"

}

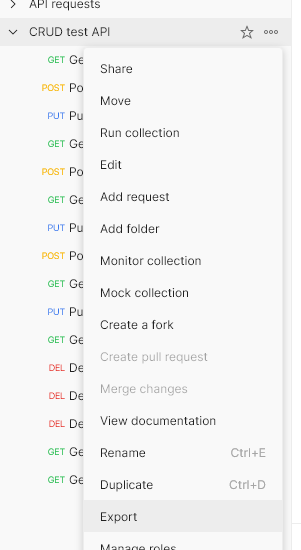
]

}

}

# Newman

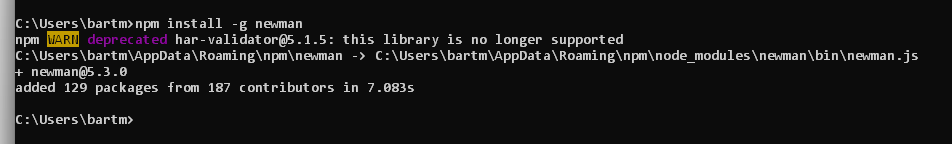
Om te beginnen zal ik net zoals de tests mijn collection exporten naar een json file.



Vervolgens moet ik het gaan installeren.

## Newman instaleren

Eerst moest ik na gaan of ik wel NPM had geïnstalleerd dit was het geval dus kon ik vervolgens newman instaleren aan de hand van commands .



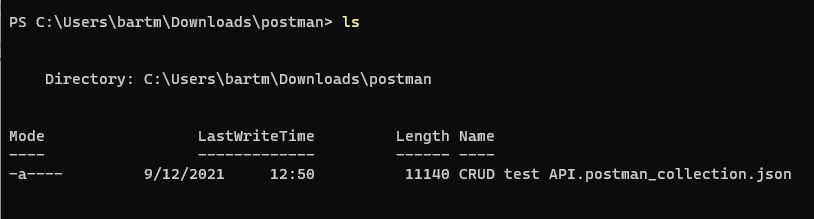
Met het volgende commando kan ik na gaan of newman wel degelijk geïnstalleerd is.



Dit is inderdaad het geval en zo kan ik zien dat ik momenteel versie 5.3.0 heb geïnstalleerd.

## Collection runnen via newman

Vervolgens ben ik naar de directory van het geëxporteerde json bestand gegaan.



Dan heb ik het comando ingegeven om het bestand te runnen via newman.





Jammer genoeg kent newman geen environment variabelen dus kunnen we enkel test runnen die we gehard code hebben. Dit komt omdat deze enkel worden opgeslagen binnen postman.