

**Software testing**

API

**Bart Meeremans**

bachelor – Toegepaste informatica

**Software testing**

Inhoudsopgave

[Normale tests 4](#_Toc85992123)

[Simpele get test 4](#_Toc85992124)

[(cursus voorbeeld) 4](#_Toc85992125)

[(eigen api) 5](#_Toc85992126)

[Get test aan de hand van ID 7](#_Toc85992127)

[Post test 13](#_Toc85992128)

[Collection level tests 16](#_Toc85992129)

[Put test 17](#_Toc85992130)

[Delete test 18](#_Toc85992131)

[Collection runner 18](#_Toc85992132)

[Environments 18](#_Toc85992133)

[Post request 19](#_Toc85992134)

[Get request 19](#_Toc85992135)

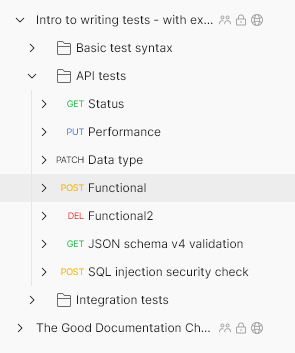
[Put request 19](#_Toc85992136)

[Delete request 20](#_Toc85992137)

[Alle requests runnen 21](#_Toc85992138)

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

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

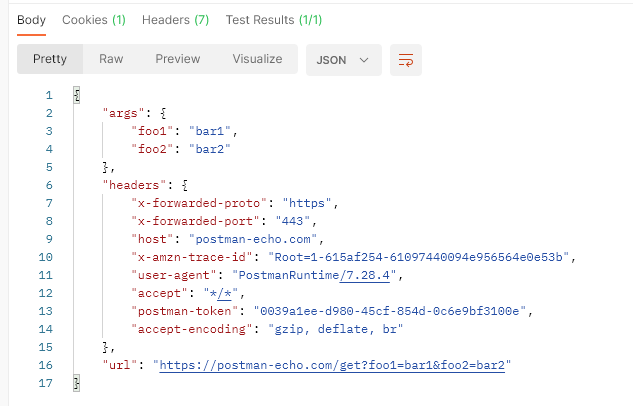


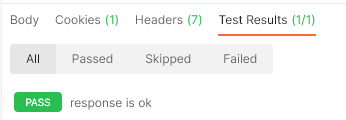
Vervolgens ben ik in mijn workspace zelf wat testjes gaan schrijven om de basis van API testing onder de knie te krijgen.

# Normale tests

## Simpele get test

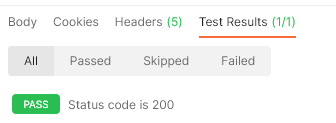
### (cursus voorbeeld)

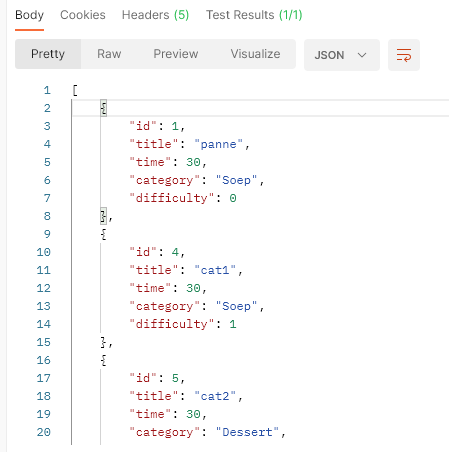


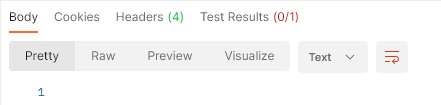
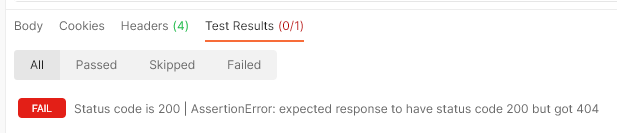


### (eigen api)

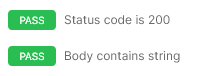
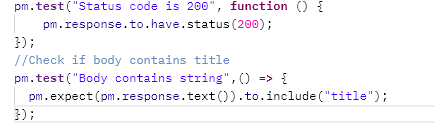
Wanneer we hier een get request zullen doen zal de test gaan controlleren of de status code 200 wordt terug gegeven. Deze status code wilt zeggen dat een request gelukt is. Aan de hand van deze test kunnen we dus gaan nakijken of we wel verbinding hebben met de API en of onze URL wel klopt.



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.



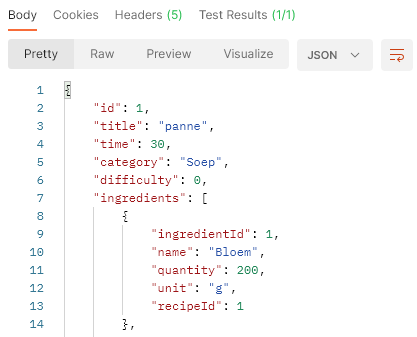
Wanneer we bijvoorbeeld eisen dat de body van het opgevraagde object een woonplaats bevat, zal de test een fail teruggeven.

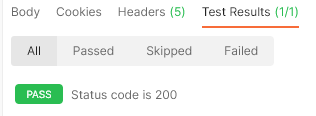


## Get test aan de hand van ID

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.

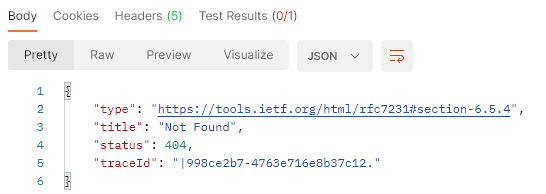
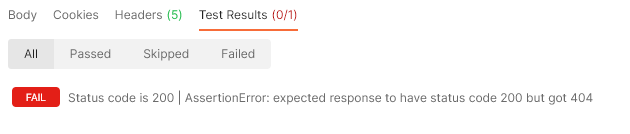






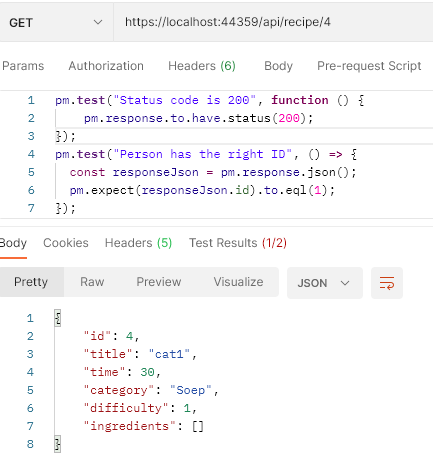
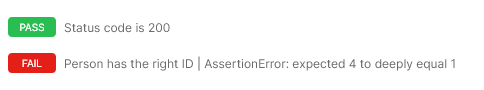
Wanneer ik een ID meegeef dat niet bestaat zal de test foutief zijn.



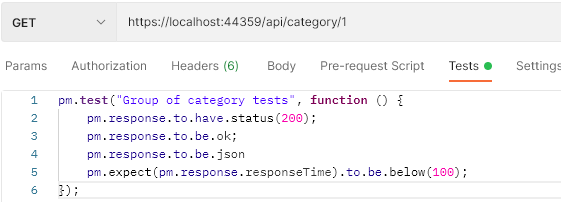
Nu moeten we wel nog gaan controlleren of we wel degelijk het juist object terug krijgen. Dit kunnen we doen door na te gaan of het ID van het gereturnede object wel het zelfde is dan het ID In de URL.

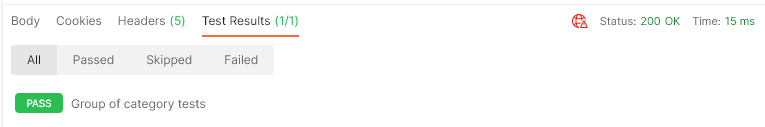


Wanneer we bijvoorbeeld in de URL object 4 gaan opvragen, maar in de test schrijf ik dat ik verwacht zal de test falen.

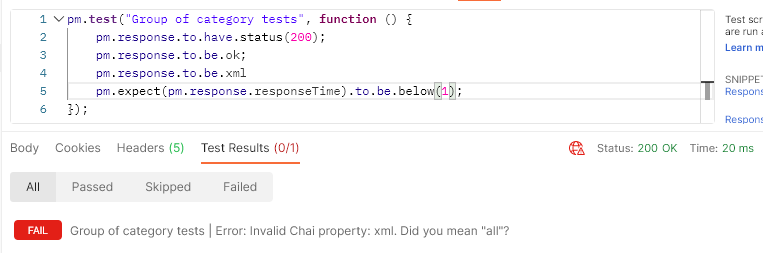
We kunnen ook zien dat het wel gelukt is om het object op te halen door de status code. Maar het is wel niet het ID dat de 2de test verwahcte.

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

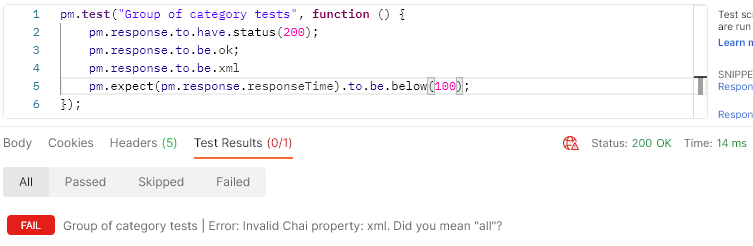


Bij deze test zal een categorie opgevraagd worden, er wordt gekeken naar de status code, of de request gelukt is maar ook of de response wel in het verwachte resultaat is. Als laatste gaan we ook nog kijken of de response wel snel gebeurt. 

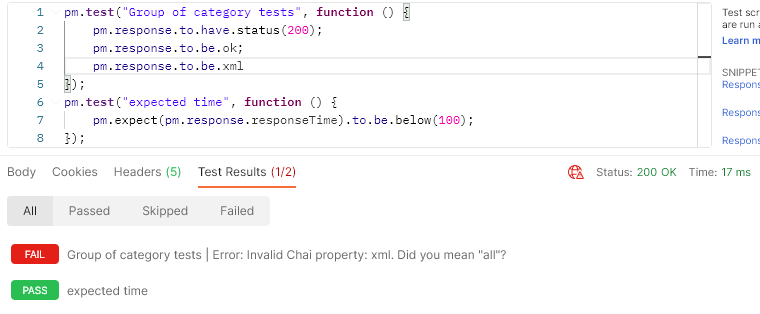
We kunnen in dit geval zien dat de tests geslaagd zijn omdat de status 200 ok is en ook de response time is 15ms.



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



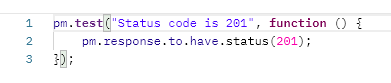
Indien we de expected time wel mogelijk maken gaan we zien dat we nu wel zien dat het foute formaat teruggestuurd wordt.

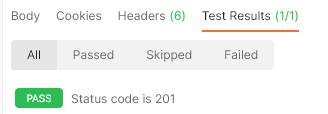
Hieruit kan je concluderen dat testen in 1 enkele group zetten ook risicos heeft. 

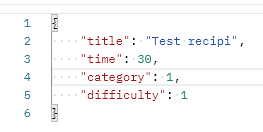
Dit is dus een betere optie omdat we nu duidelijk kunnen zien wat er gaat faalen. Het heeft echter ook geen nut om de response statusen te splitsen omdat deze oftewel allemaal gaan werken of juist helemaal niet.

## Post test

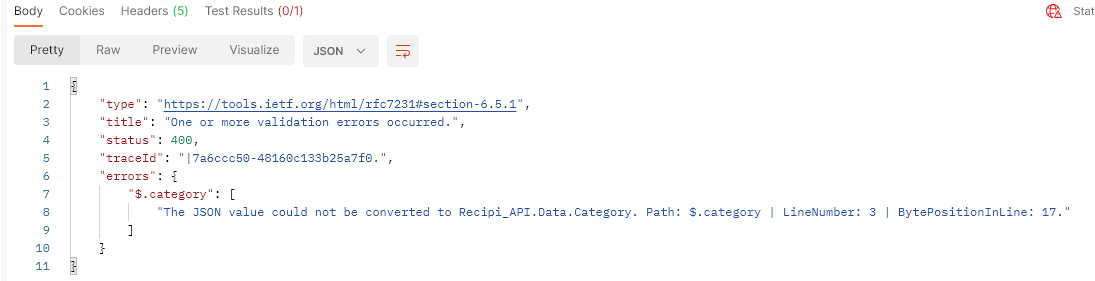


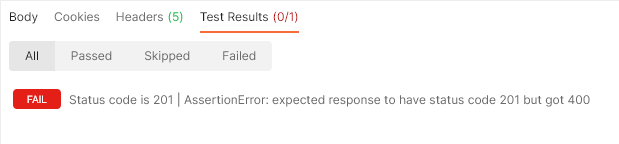


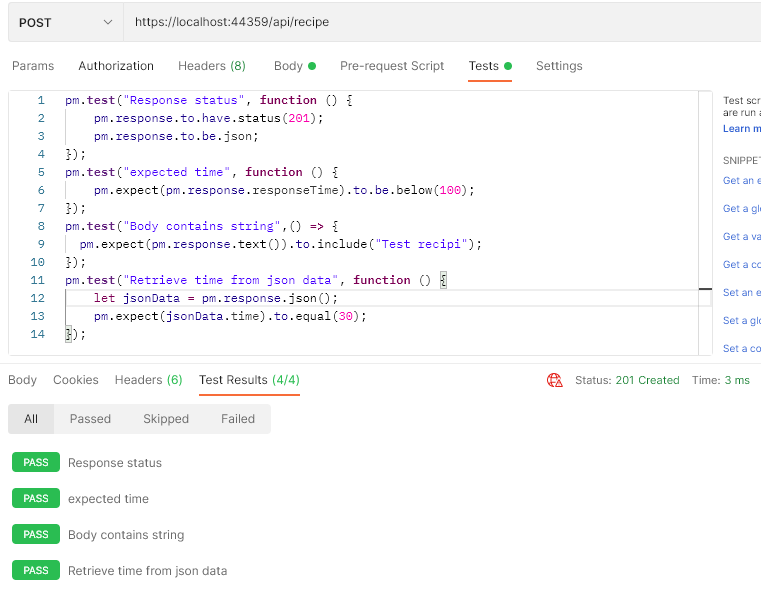
Indien een gegeven body onvolledig of fout is returnt de test een fout. 

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 .

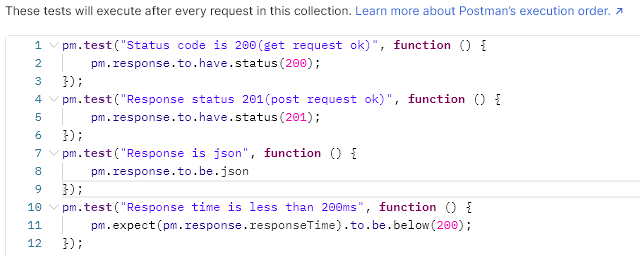
In de group “Retrieve time from json data” kunnen we echter gebruik gaan maken van JSON. Eerst maak ik een json object aan en stel ik het gelijk aan de ontvangen data. Hier kan ik dan ook mee gaan werken, ik zal gaan kijken of het json object een variabelen “time” bevat. En dan vervolgens ook kijken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

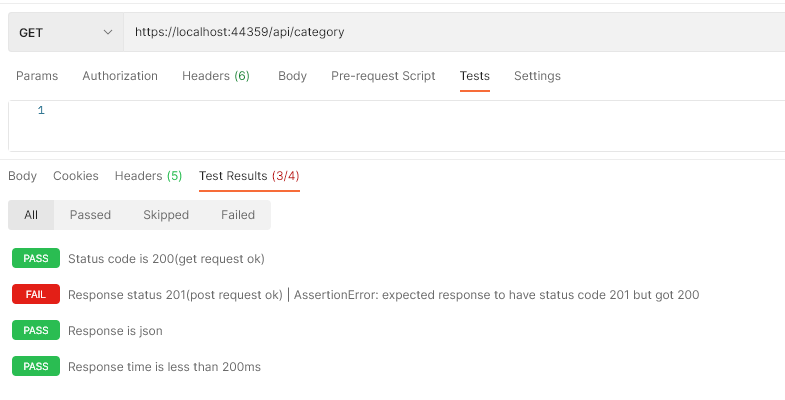
In de 2de methode zal je dus egt een exacte waarde gaan verwachten terwijl in het eerste voorbeeld je value er gewoon moet in voorkomen.

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

## Collection level tests

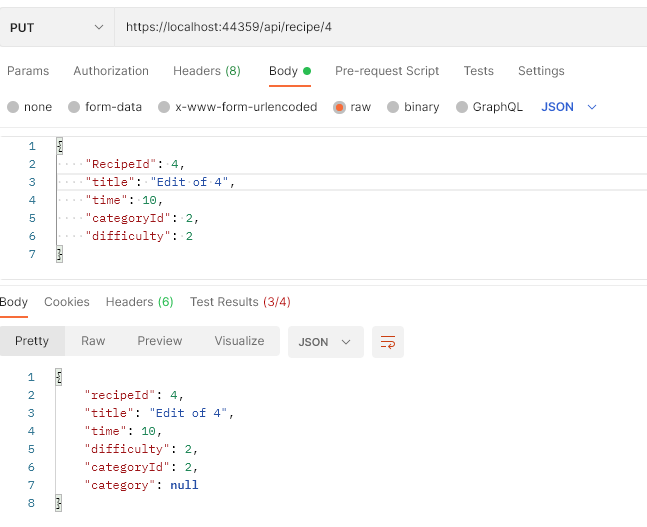
Er zijn natuurlijk tests die we altijd willen gaan uitvoeren. Bijvoorbeeld kijken welke status code we terug krijgen, of controlleren welk bestandsformaat gereturned wordt. Het is in postman echter mogelijk om tests te gaan schrijven op een collection niveau.

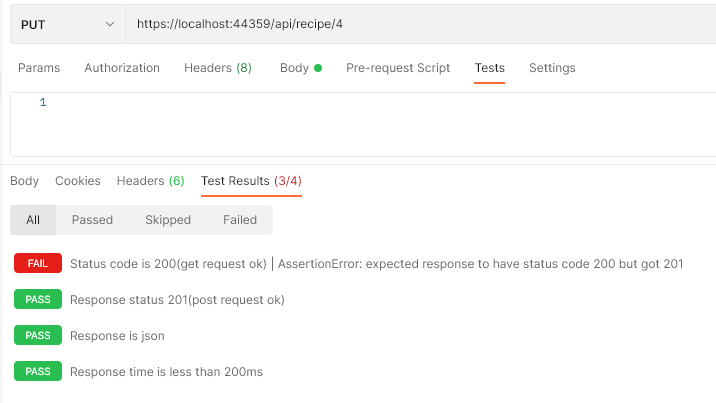




Wanneer ik nu een api call ga doen in mijn eigen collection zullen de 4 tests automatisch worden gedaan. Dit is handig voor testen die we steeds zullen nodig hebben.

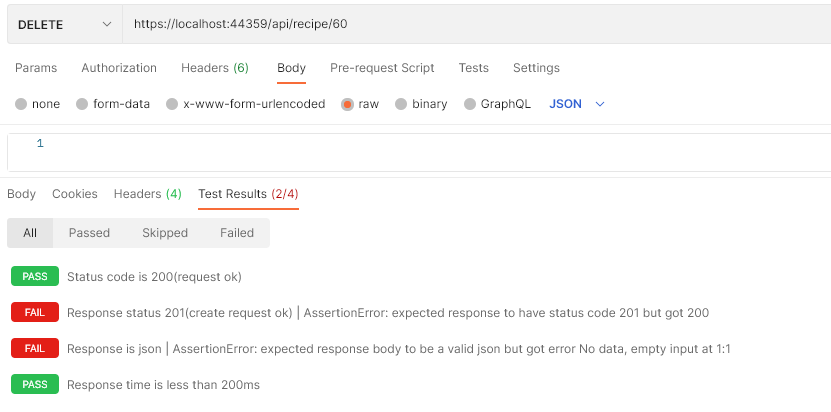
## Put test

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.

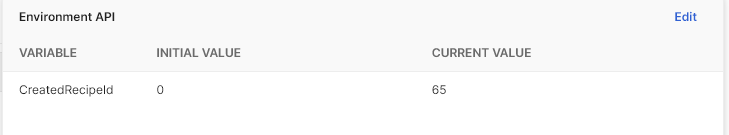
## Collection runner

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.

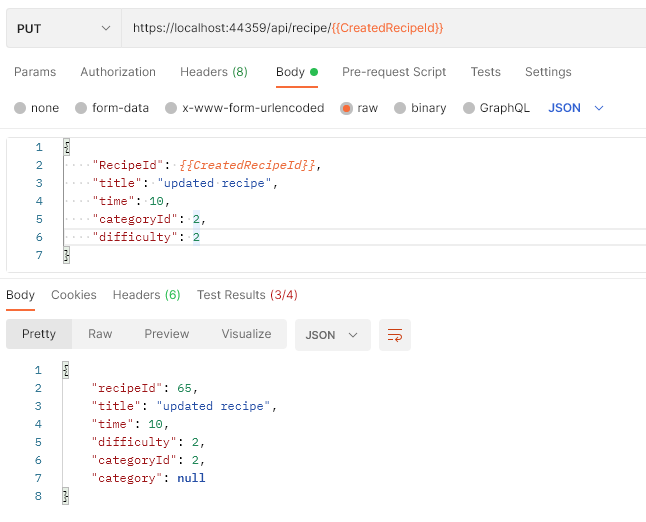
Vervolgens zet ik in de environment de variabelen gelijk aan de ontvangen data zijn ID.

### Get request

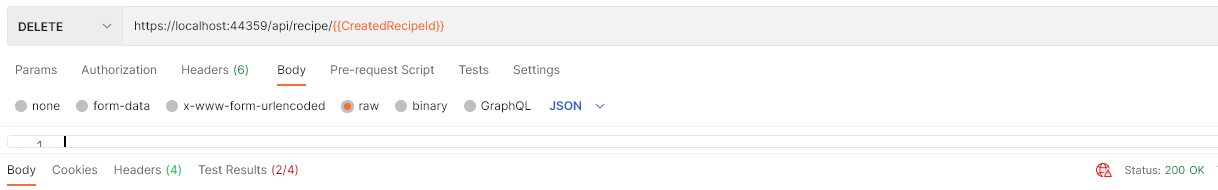


Nu kunnen we gebruik maken van het CreatedRecipeId om het nieuwe aangemaakte object te gaan opvragen met een get request.

### Put request

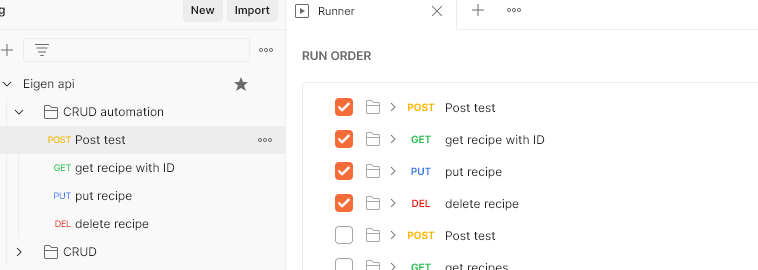
Deze variabelen kunnen ook gebruikt worden in de body van een request. In dit geval zal ik een put request doen op het laatst aangemaakte object.

### Delete request

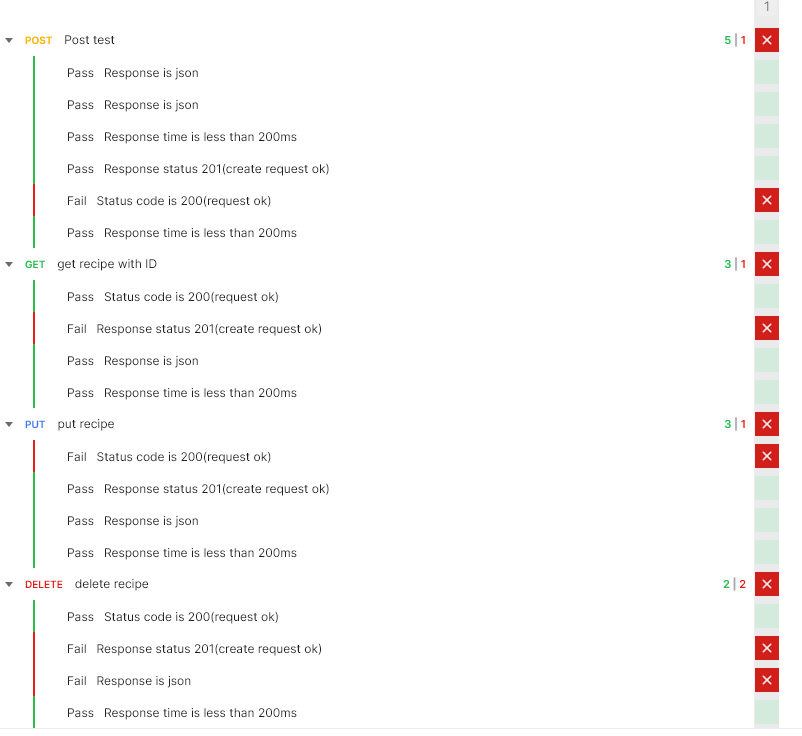


Als laatste in de crud operaties hebben we enkel nog het delete request, dit gaan we dan ook testen met ons aangemaakte object. Ook hier kunnen we da environment variabele gebruiken.

### 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.