

**ASP.NET Core Web API testen met Postman**

Deneyer Sander

3 Toegepaste Informatica

Contents

[Intro 3](#_Toc85721847)

[Setup 3](#_Toc85721848)

[Postman: The Complete Guide – REST API Testing 3](#_Toc85721849)

[Introduction and first steps in Postman 3](#_Toc85721850)

[Creating with API requests 4](#_Toc85721851)

[Requests 4](#_Toc85721852)

[Responses 6](#_Toc85721853)

[Saving requests 7](#_Toc85721854)

[Writing tests and scripts 8](#_Toc85721855)

[Een eerste simpele test 8](#_Toc85721856)

[De tabel Categories testen 9](#_Toc85721857)

[Writing tests and scripts using variables 12](#_Toc85721858)

[API workflow using Github & Twilio 12](#_Toc85721859)

[Advanced assertions 12](#_Toc85721860)

[Automatically running tests 12](#_Toc85721861)

[Running Newman with other CI servers/tools 12](#_Toc85721862)

[Workflows and Scenarios 12](#_Toc85721863)

[Data driven tests: Running a request multiple times with different data sets 13](#_Toc85721864)

[Team Collaboration 13](#_Toc85721865)

[Mock servers 13](#_Toc85721866)

[File uploads (testing, automatic uploads, uploading multiple files) 13](#_Toc85721867)

[Authentication / Authorization 13](#_Toc85721868)

[Advanced topics and user questions 13](#_Toc85721869)

[New features in Postman 13](#_Toc85721870)

[JavaScript fundamentals 13](#_Toc85721871)

[Postman Developer Challenge 2021 13](#_Toc85721872)

[Bonus section 13](#_Toc85721873)

[Timetable 14](#_Toc85721874)

# Intro

Als opdracht voor het vak software testing moeten we zelf onze test omgeving zoeken en wat we willen testen. Na even te denken heb ik ervoor gekozen om een REST-API te tesen via het programma postman. Ik heb hiervoor geopteerd door enkele zaken.

Ten eerste is het testen van API’s in postman nieuw voor mij. Alhoewel ik postman zelf al eens heb gebruikt was dit maar vlugtig en niet zo uitgebreid zoals ik het nu wil doen.

Ten tweede kan ik zo een eerder geschreven API testen dat we voor het vak ‘Advanced Programing Concepts’ moesten maken. Doot dit te doen kan ik mijn eigen werk testen, en dus zien hoe goed deze taak was. En kan ik eventueel ook nog fouten uit het programma halen zodat ik hieruit kan leren zowel voor dit vak als het andere.

Om dit allemaal te kunnen leren heb ik ervoor gekozen om een cursus aan te kopen op Udemy (Postman: The Complete Guide – REST API Testing). De weg dat ik zelf afleg tijdens deze cursus wordt hieronder beschreven.

# Setup

Om een API te kunnen testen heb ik natuurlijk een API nodig en een API test platform. Voor de API gebruik ik een al eerder door mezelf gemaakt programma. En het test platform zal Postman zijn.

Toen ik eerst postman wou installeren, installeerde het als een app. Geen kwaad zo je denken, maar als je dit dan opende dan ging het naar een web based platform dat ik nogal omslachtig vond. Maar omdat ik al eens eerder had gebruik gemaakt van postman wist ik dat er een een echte applicatie van bestond. En na altijd geredirect te worden naar de app vond ik het ineens. Hierna was het gewoon gemakkelijk downloaden en snel installeren.

# Postman: The Complete Guide – REST API Testing

## Introduction and first steps in Postman

In de cursus content hebben ze eerst eerst een overview gegeven van wat we kunnen verwachten van de cursus en wat we niet/wel kunnen verwachten van dit.

Hierna wordt er overlopen hoe je een account kunt maken en hoe je postman moet downloaden. Eens postman is opgestart wordt het programma getoond en wordt er al direct een simpele request gemaakt. Eens we het antwoord hebben gekregen van de request wordt hier van alles over gezegd.

Hierna werd het al een beetje complexer en heeft hij uitgelegd wat parameters zijn en hoe je ze kan gebruiken.

Een request (met bepaalde parameters) kan ook altijd opgeslagen worden via postman. Maar dit kan niet zomaar we moeten eerst een collection maken om dit in op te slaan.

Een request kan meer dan alleen een GET zijn, hij heeft ook al getoond hoe je met postman een POST request kan maken, opstellen en weer oplsaan.

Hierna begon hij met iets belangrijks uit te leggen, hoe je via variabelen bepaalde data uit een response body kan gebruiken in een ander request zoals bv een post. Dit gebeurdt aan de hand van code.

Tenslotte heeft hij ook nog getoond hoe je gemakkelijk een simpele test kan opstellen via de snippets.

Via de Newman kan je bepaalde requests van één collection laten runnen na elkaar. Dit zorgt ervoor dat je verschillende requests/testen kunt combineren in een workflow.

Op het einde van de Intro heeft hij ook nog in het kort uitgelegd wat automated testing is.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

## Creating with API requests

### Requests

Een htpp request bestaat uit vier delen:

* Een request methode
* Een url waar het request moet naartoe gaan
* Headers
* Body

Graphical user interface, text, application, Teams

Description automatically generatedGraphical user interface, text

Description automatically generated

De request methode bij Postman zijn makkelijk aan te duiden, de voornaamste zijn GET, POST, PUT, DELETE. Over alle andere methodes wordt momenteel nog niks over gezegd omdat dit direct een ander niveau dan wat ze ons proberen te leren in deze cursus.

Een gemakkelijk feature van postman is dat het de vorige url adressen onthoud dat je al hebt gebruikt. Hierdoor kan je dus snel werken zonder steeds de juiste url op te hoeven zoeken.

Headers zijn ook een deel van het request. Er zijn al standaard headers dat postman aanbied maar je kan er ook nog altijd zelf meegeven.

Onder body kan je verschillende opties selecteren. 

Hierin kan je verschillende waardes meegeven aan uw request. En dit zijn dan eigenlijk post requests. Je kan bijvoorbeeld voor een Voornaam & Achternaam meegeven. Om gemakkelijk te zien wat we gaan meegeven gaan we werken met Hookbin. Hiermee gaan we kunnen zien wat we allemaal doorsturen.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email, website

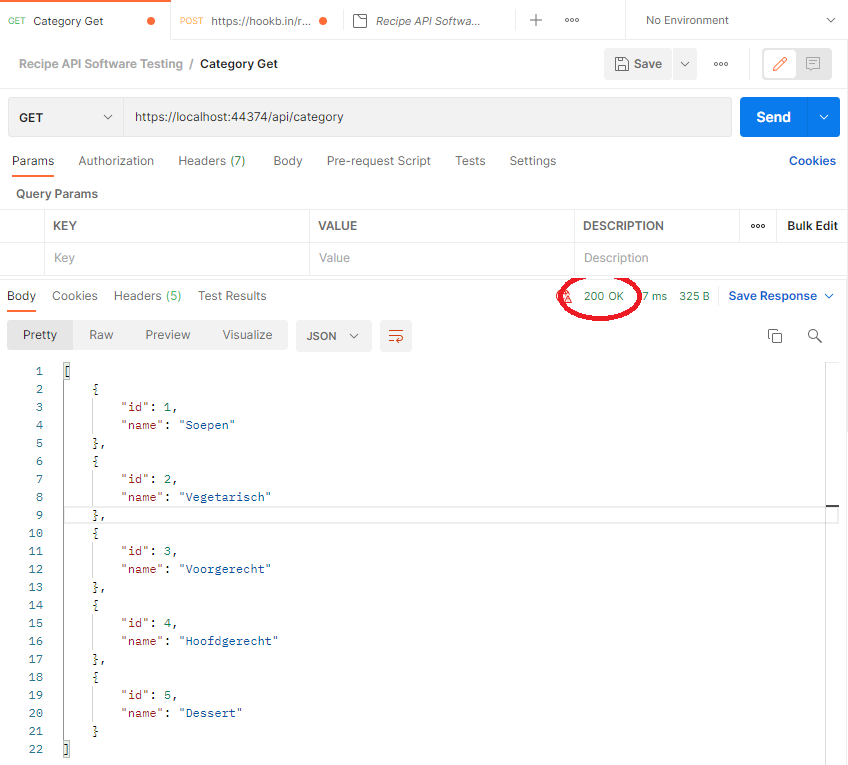
Description automatically generated

Je kan dus zien op de eerste screenshot dat ik via postman een PUT heb gedaan met een voornaam en achternaam in de Body. Op de tweede screenshot kan je daadwerkelijk zien dat de website Hookbin dit heeft ontvangen.

Het verschil tussen de form-data optie en de x-www-form-urlencoded is dat hetgeen wat je doorstuurd web encoded wordt. Dit betekent dat bijvoorbeeld spaties worden encoded. Bij Raw kan je kiezen dat je in Tekst schrijft, JavaScript, JSON, HTML of XML.

### Responses

De response is eigenlijk het gedeelte dat de api antwoord. Wanneer de code begint met een twee, dan weet je dat alles oké is gegaan. Wanneer de server code terugstuurt dat begint met een 4 dan weet je dat je iets fout hebt gedaan. En wanneer het begint met een 5 dan is er een probleem met de server.

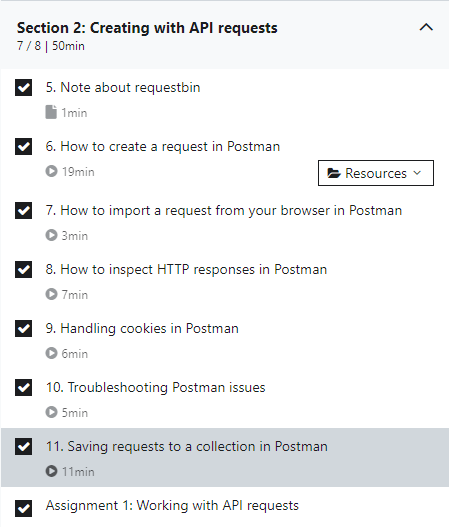


Zoals je nu kan zien hebben we een simpele GET request gemaakt waar we vragen achter /category. En in het programma staat dit gecodeert dat het een get all returnt. Je ziet dus 5 verschillende gerechten dat standaard waardes zijn. Wat je ook kan zien is dat de code 200 werd teruggegeven dus alles is in orde en het duurde 77ms om een antwoord te krijgen. Je kan het antwoord ook in verschillende formaten laten weergeven zoals JSON, XML, HTML en Tekst. En ten slotte kan je de response ook nog opslaan. Wat ook handig is dat je naast de response code kan zien hoe groot de response is.

### Saving requests

Text

Description automatically generatedJe kan verschillende requests opslaan door ze in een collectie te zetten. Een collectie bestaat normaal uit verschillende bijhorende requests. Zelf heb ik al een collectie gemaakt voor dit vak met al één request in dat ik hierboven heb getoond. Om requests nog beter te beheren kan je ook folders maken in een collectie maar dit is nog niet toepasselijk voor mij.



## Writing tests and scripts

### Een eerste simpele test

Wat is een test? Een test is eigenlijk een bewering waar je de response controleert tegen over een verwacht antwoord, bv een status code dat dan liefst 200 is.

Een zeer simpele test zou dit kunnen zijn:

Graphical user interface, text, application

Description automatically generated

Dit is een lijkt een zeer simpele test, maar toch kwam ik al snel op een probleem. Er kon geen request worden verzonden.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generatedNa een tijdje zoeken op het internet en zelf te reflecteren wat er mis was, kwam ik tot het besef dat ik zelf was vergeten mijn API te laten runnen. Dit moet elke keer gebeuren voor ik begin met testen anders kan er natuurlijk niets getest worden.

Nu kunnen we de test nog eens proberen te doen. En succesvol, bij de rechtse afbeelding kan je bij ‘Test Results’ zien dat de test is geslaagd en de status code echt 200 is.

Wanneer bijvoorbeeld de request url verkeerd geschereven is en dus niet bestaat zal je natuurlijk een ander resultaat moeten zien. De test zou moeten falen en zoals je kan zien gebeurt dit ook.

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

Maar wat betekend deze test code nu? pm.test is een fuctie voor het schrijven van test specificaties. Tussen de haakjes staat een eerste parameter, dit is een string. Na de comma sraat een tweede parameter, dit is een call back funtie (gebeurt pas na het uitvoeren van de test) dat kijkt op de response status code in dit geval 200 is.

### De tabel Categories simpel testen

#### De data bekijken.

Om deze tests te kunnen doen gaan we eerst is kijken naar de data dat nu al in de database staat. Dit kan gemakkelijk met Visual Studio en de ingebouwde “SQL Server Object Explorer”.

A screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generated

Zoals we kunnen zien is er nu enkel data in de tabel ‘Catagories’. We zullen nu eens testen of we alleen de informatie het eerste categorie kunnen weergeven. Hierin zou dus ‘Soepen’ moeten staan.

#### Eén categorie weergeven

Graphical user interface, text, application

Description automatically generated

Dit is de test, je ziet dat de request URL is aangepast en achter ‘category’ het getal ‘1’ staat. Dit staat dus voor het id van een categorie dat we willen zien.

Graphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, application

Description automatically generated

Zoals je kan zien is deze test geslaagd en hebben we Soepen teruggekregen.

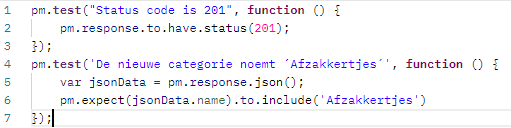
#### Een categorie toevoegen

Een categorie kan makkelijk toegevoegd worden door een beetje json code te schrijven in de body van de request. Het ziet er als volgt uit.

Graphical user interface, text, application, email

Description automatically generated

Hierin kan je zien dat we de catgorie ‘Afzakkertjes’ willen toevoegen. Nu moeten we nog juist de Tests toevoegen.



Eerst gaan we kijken of de POST is gelukt, dit kan via de status code ‘201’. Hierna gaan we kijken of er in de response bij het geelte ‘name’ ‘Afzakkertjes’ staat. Eens we de POST uitvoeren kunnen we de volgende body zien.

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

We kunnen ook al direct zien dat 2/2 tests zijn gelukt. En hier nog een diepere blik in kan ons bevestigen dat beide testen geslaagd zijn.

#### Een categorie verwijderen

Vooraleerst we een categorie gaan verwijderen gaan we eerst een nieuwe categorie toevoegen om te verwijderen met bovenstaande code.

A picture containing chart

Description automatically generated

Nu dit is gelukt kunnen we beginnen aan de DELETE. Deze soort request lijkt heel veel op het opvragen van één categorie. Het enige wat we eigenlijk moeten veranderen is de GET naar DELETE

Graphical user interface, text, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

Als de test geslaagd is zouden we de status code 204 moeten terugkrijgen.

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application

Description automatically generated

Zoals je kan zien is de test gelukt maar hebben we geen body teruggekregen. Dit is wel normaal. Want wat zouden we moeten terug krijgen wanneer we iets verwijderen. Een status code geeft al meer dan genoeg weer, wanneer we weten of het met succes is verwijdert weten we al genoeg.

### De tabel Recipes simpel testen

#### De data bekijken

Zoals de de tabel Categories gaan we de data bekijken via Visual studio. En zoals we kunnen zien is er helemaal nog geen data is de tabel. Dit vermoeilijkt het testen wel omdat er nog geen data is om te testen.

Graphical user interface

Description automatically generated

Graphical user interface, application

Description automatically generatedWe kunnen dit ook nog eens bevestigen via de web interface. Als we naar ‘localhost:**POORT**/api/recipe gaan op een browser, kunnen we eigenlijk ook een request doen. Dit komt omdat ik dit zo heb geprogrammeerd. In de code heb ik de standaard route naar ‘api/recipe’ gezet. En een methode dat alle recipes weergeeft maakt gebruik van deze route zonder enige aanhangsels.

Text

Description automatically generated

Omdat ik zelf de code heb geschreven is dit allemaal gemakkelijk te verstaan voor mij.

#### Data toevoegen

Om onze testen te kunnen beginnen gaan we dus allerlei data toevoegen. We gaan dit eigenlijk doen via een simpele POST request in postman. Om alles mooi gesorteerd te houden binnen postman gaan we eerst beginnen met een folder aan te maken voor alle requests dat met ‘recepi’ te maken heeft. Graphical user interface, application

Description automatically generated

Nu kunenn we beginnen met het schrijven van een eerste post en deze dan uit te voeren.

Graphical user interface, text, application, email

Description automatically generated

Zoals je kan zien loopt er hier iets mis, de request dat ik probeer te maken is nog niet supported door de API.

Graphical user interface, text, application

Description automatically generated

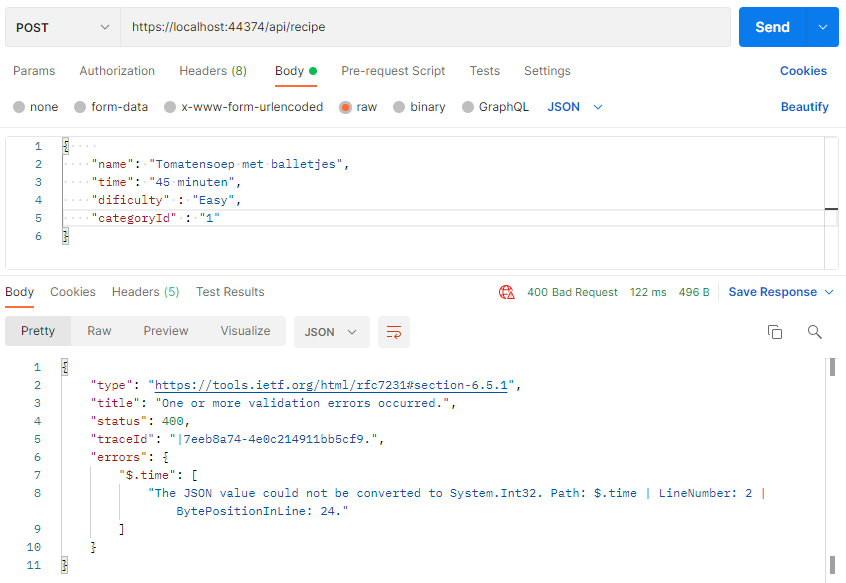
Na enig onderzoek heb ik ondervonden dat er in de code nog geen methode staat voor het toevoegen van een recipe. Dankzij deze zeer simpele test heb ik dit wel kunnen ondervinden en kan ik mijn code aanpassen.

Ik ben dus begonnen met een methode aan te maken waarmee ik een POST request kan versturen. Het ziet er als volgt uit.

Text

Description automatically generated

Gelukkig had ik eerder wel al in de \_recipeService een Create methode gemaakt dus kon ik deze simpel gebruiken en voor de rest moest ik niets meer doen.

Nu gaan we de POST van hiernet nog is proberen uit te voeren. Maar er komt nog een fout op. 

De error code is nu ‘400’, er was een slechte request. Uit het veld ‘errors’ in de body kunnen we aflezen dat we de time niet kunnen converteren naar Int32. Dit betekent dat ik een fout heb gemaakt in de Body. Ik heb een String gebruikt ipv een Intiger. Dit is wel gemakkelijk recht te zetten. Eens we dit hebben gedaan zou er normaal niets meer mogen foutlopen.

Maar zoals we nu al effen pech hebben gehad loopt er toch wel weer iets fout.

Graphical user interface, text, application, email

Description automatically generated

Deze keer is het door de ‘difficulty’. Na een tijdje zoeken heb ik echter gevonden dat je voor difficulty ook getallen moet gebruiken van 1 tot en met 3 in plaats van ‘Easy’, ‘Intermediate’ & ‘Advanced’. Dit omdat deze standaard waarden in een enum zitten.

Text

Description automatically generated

Nu dit allemaal achter de rug is zou het deze keer echt moeten lukken zonder problemen. We passen de ‘difficulty’ aan naar 1 en verzenden ons request nog eens. En zoals je kan zien is de request gelukt, maar we kunnen zien dat de ‘title’ in de response body null is. Dit komt omdat ik in de POST body een fout heb geschreven en in plaats van ‘title’ heb ik ‘name’ gebruikt. Maar dit is ook weer zeer gemakkelijk opgelost.

Graphical user interface, text, application, email

Description automatically generated

En zoals je nu kan zien is de response body wel helemaal in orde.

Graphical user interface, text, application

Description automatically generated

Omdat we wel meer dan één recept nodig hebben zal ik er nu nog enkele toevoegen en ze dan laten zien in een volgende test.

#### Alle recepten weergeven

Nu we verschillende recepten hebben toegevoegd kunnen we deze is bekijken met een simpele test.

Graphical user interface, text, application, email

Description automatically generated

Als we dit uitvoeren kunnen we zien dat we een hele hoop data terugkrijgen (ik heb het in een raw format gezet zodat we alles kunnen bekijken).

Text

Description automatically generated

En daarnaast kan je ook zien dat de test is geslaag.

Graphical user interface, text, application

Description automatically generated

#### Eén recept weergeven

Nu we alle verschillende Id’s hebben van alle recpeten, kunnen we zien dat die zich bevinden tussen de 8 & 25 momenteel. Dit is handig omdat we aan de hand van één van deze Id’s een recept apart kunnen opvragen. Dit kan via de request url dat eindig op ‘api/recipe/**ID**’. We zullen dit nu meteen toepassen binnen PostMan.

Graphical user interface, text, application, email

Description automatically generated

Zoals je kan zien op de screenshot hebben we het recept opgevraagd dat het id = 8 heeft. Wanneer we naar de body kijken kunnen we zien dat de title null is. En dit was dus een foute post. Nu we weten welk ID dit ‘recept’ heeft kunnen we dit in een volgende test verwijderen.

Uit de Test Results kunnen we ook perfect zien dat er niets is fout gelopen en dat we de juiste response code hebben terug gekregen.

Graphical user interface, text, application, chat or text message

Description automatically generated

#### Een recept toevoegen

We hebben nu wel al verschillende recepten toegevoegd, maar we hebben dit nu nog niet direct getest. We gaan hiervoor dus binnen PostMan een aparte request voor maken.

Graphical user interface, text, application, email

Description automatically generated

Zoals je kan zien gaan we proberen Garnaalsoep toe te voegen, met de bijhorende propperties. Nu de body is ingevuld gaan we beginnen met het testing gedeelte.

Graphical user interface, text, application

Description automatically generated

We gaan eerst beginnen met de status code te controlleren. Dit gaan we apart laten weergeven zodat het makkelijk leesbaar is later. Daarna gaan we de titel controleren, hierna kijken we of het recept de juist tijd bevat. En tenslotte gaan we kijken of de laatste propperties voor zowel dificulty als categoryId ook kloppen met wat we versturen.

Nu kunnen we de request doorsturen en het dus ook testen.

Graphical user interface, text, application

Description automatically generated

We kunnen zien dat het id 29 is en zo voort. Op het eerste zicht is deze test dus geslaagd. Maar voor verdere controle gaan we is kijken naar de Test Results.

Graphical user interface, text, application

Description automatically generated

En zoals je wel kan aflezen zijn al onze test results geslaagd, maar nog belangrijker ze zijn ook nog is leesbaar voor de gewone mens of voor iemand dat niet direct iets afweet van wat we hier doen.

#### Een recept verwijderen.

Zoals hiervoor al gezegd geweest is gaan we proberen het recept te verwijderen met Id = 8 omdat deze fout is.

Graphical user interface, text, application, email

Description automatically generated

Omdat we weten dat de response code 204 zou moeten zijn voor een geslaagde delete request, kunnen we dit zo testen. Eens we dit uitvoeren kunnen we het volgende zien.

Graphical user interface, text, application, chat or text message

Description automatically generated

In de body staat niks. Maar dit is dan ook weer normaal, want wat zou je moeten terug krijgen wanneer je iets verwijderd. En status code is dus meer als genoeg.

Graphical user interface, text, application, chat or text message

Description automatically generated

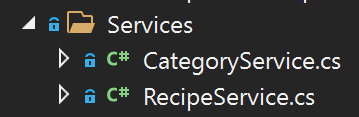
En zoals je nu kan zien kan je uit de Test Results zien dat de status code 204 is en de test dus geslaagd is.

### De tabel ingredients simpel testen

Vooraleer we kunnen beginnen met het testen van de tabel heb ik enige zaken opgemerkt. Zoals je kan zien in de onderstaande screenshots, is er helemaal nog geen data in de tabel. Maar er is ook geen service classe of controller classe, dus zou ik zelfs geen test data kunnen toevoegen. Om deze opdracht zo volledig mogelijk te kunnen maken ga ik dus eerst weer een tijdje in C# programmeren zodat we verder kunnen werken.

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

#### IngredientService maken

Text

Description automatically generated

Text

Description automatically generated

#### IngredientController maken

Text

Description automatically generated

Text

Description automatically generated

#### Test data toevoegen aan de tabel Ingredient

Graphical user interface, text, application, email

Description automatically generated

Wanneer ik test data probeerde toe te voegen via deze POST request. Zoals je kan zien is er iets fout gelopen. Ik heb heel wat tekst teruggekregen dat het probleem probeert uit te leggen. Daarnaast kunnen we ook zien dat het een error code met nummer 500 is. Dit betekent dat er een error is in de Server.

Text

Description automatically generated

Meer over deze error code. De 500 Internal Server-error kan worden veroorzaakt door een fout tijdens de uitvoering van een beleid of door een fout op de doel-/backendserver.

De HTTP-statuscode 500 is een algemene foutreactie. Wat betekent dat de server een onverwachte omstandigheid heeft aangetroffen waardoor deze niet aan het verzoek kon voldoen. Deze fout wordt meestal geretourneerd door de server als er geen andere foutcode geschikt is.

Nu we weten dat het aan de api zelf ligt kunnen we beginnen met troubleshooten. Dit gaan we doen met de volgende stappen:

1. De net geschreven Service classe controlleren
2. De net geschreven Controller classe controlleren
3. Dependencies na kijken in de configuratie.

Aangezien we weten dat andere service classen wel werken kunnen we de service classe van Ingredient controlleren met die van Recipe bijvoorbeeld. Na dit te doen heb ik niks eigenaardigs opgemerkt waardoor deze code niet zou mogen werken.

Nu kunnen we dus gaan kijken naar de controller classe. Deze gaan we op dezelfde manier controlleren als de service classe. En ook na hier lang te controlleren heb ik weer niks gevonden.

Dan gaan we dus is moeten kijken naar alle dependencies in de configuratie wat niet al te evident is want dit zijn er veel. In het begin vond ik eigenlijk niks, tot ik eens aan een collega had gevraagd of hij niets zag. En natuurlijk vond hij deze fout wel.

A screenshot of a computer

Description automatically generated with medium confidence

Zoals je kan zien op de screenshot zijn er hierin twee services gekend. De service classe voor Recipe en voor Category. Maar we hebben er onlangs eentje aangemaakt die van Ingredient. Dus als we gewoon de onderstaande code toevoegen zou dit probleem normaal moeten worden opgelost.

services.AddScoped<IngredientService>();

Nu gaan we onze POST nog is proberen zodat we ook weer test data kunnen toevoegen aan deze tabel. Graphical user interface, text, application

Description automatically generated

En zoals je kan zien is dit gelukt want we krijgen een status code 201 terug. Nu ga ik de tijd nemen om nog meer data toe te voegen zodat we straks goed kunnen testen.

#### Alle ingrediënten weergeven

Graphical user interface, text, application

Description automatically generated

A picture containing background pattern

Description automatically generated

Alle ingrediënten weergeven kan door een simpele get request zonder iets extra aan de url toe te voegen. Wanneer we dit dan ook weer testen zien we dat de test werkt.

#### Eén ingrediënten weergeven

Graphical user interface, text

Description automatically generated

Om eerst één ingrediënt weer te geven gaan we een ingrediënt id nemen uit de lijst van alle ingrediënten.

Graphical user interface, text, application, email

Description automatically generated

Zoals je van uit de screenshot kan afleiden gaan we het ingrediënt met een id:5 proberen weer te geven. Door de get request van alle ingrediënten kunnen we de waarden van dit zien en deze dan in de test gebruiken. Om onze tests uiteindelijk beter te kunnen begrijpen wanneer ze uitgevoerd worden heb ik een if statement in de test code gezet. Zodat als er geen unit is of maateenheid dat de variabele unit dan leeg blijft. En anders wordt de unit in deze variabele gezet samen met een spatie. Dit heb ik dus gedaan zodat we geen standaard spatie voor unit moesten zetten in de pm.test. Anders zou je wel eens een dubbele spatie kunnen hebben wanneer er geen unit is bij een bepaald ingrediënt.

Graphical user interface, text, application

Description automatically generatedZoals je kan zien zijn de 3 testen gelukt en weten we dus dat dit een succesvolle test was.

#### Een ingrediënten toevoegen

Graphical user interface, text, application

Description automatically generated

Om een ingredient toe te voegen moeten we natuurlijk eerst een POST request maken met een bijhorende body. Deze body is geschreven in JSON formaat en is daardoor dus zeer leesbaar.

Graphical user interface, text, application, email

Description automatically generated

Met deze testen gaan we eerst kijken of wez de juiste status code gaan terugkrijgen. Daarna gaan we controleren op de juiste ingrediënt naam. Dan kijken we of het recept id wel overeenkomt met het ingegeven id. En tenslotte gaan we de hoeveelheid en de maateenheid controleren.

Graphical user interface, text, application

Description automatically generatedEn zoals je kan zien uit de test resultaten zijn al deze 4 testen met een succes gelukt.

#### Text Description automatically generatedEen ingrediënten verwijderen

Zoals je kan zien uit deze JSON-data hebben wij twee dezelfde ingrediënten voor hetzelfde recept. Er is dus eentje overtollig en gaan we hiervan dus eentje verwijdereb.

Graphical user interface, text, application, email

Description automatically generated

Omdat we hier dus een DELETE request gaan gebruiken kan deze test heel simpel zijn. Bij een delete request weten we dat onze API-server alleen maar een 204 status code teruggeven.

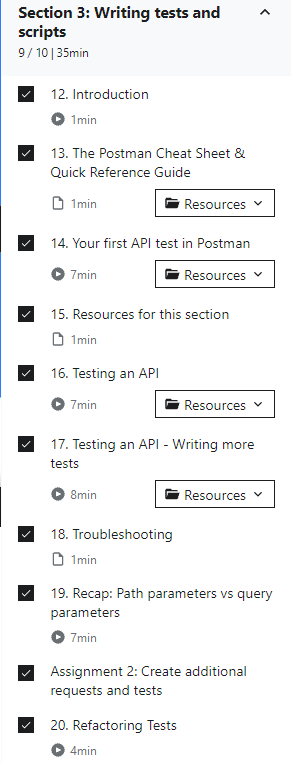
Graphical user interface, text, application, website

Description automatically generatedEn zoals je kan zien is deze test gelukt.

Text

Description automatically generated

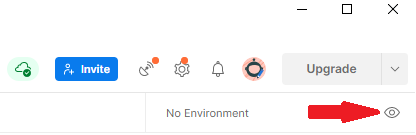
Om dit nog verder te kunnen controleren doen we nog is een GET request voor het zien van alle ingrediënten. EN zoals je wel kan zien is het ingrediënt met id 16 gedeletet.



## Writing tests and scripts using variables

### Globale variabelen

Globale variabelen zijn variabelen dat je over heel postman ga kunnen gebruiken. Om deze aan te kunnen maken moet je zoals op de screenshot hier onder op het ‘oogje’ drukken.



Graphical user interface, text, application, email

Description automatically generated

Eens we hierop hebben gedrukt zien we dit op het scherm verschijnen. Hierop kunnen we ‘Environment’ & ‘Globals’ zien. Wij gaan ons nu focussen op globale en de environment variabelen komen later nog aan de beurt. Nu gaan we dus op ‘Add’ drukken bij Globals. En dan zien we het volgende verschijnen.

Graphical user interface, text, application, email

Description automatically generated

We zien het venster Globals verschijnen waar initieel nog niks instond. Maar ik heb er al een variabele ingezet genaamd ‘url’. Dit omdat het eerste deel van onze request url toch altijd hetzelfde is en dus geen nut heeft om altijd te blijven typen. Dit is dus een goed voorbeeld van een globale variabele. Wat ook nog een goed voorbeeld zou kunnen zijn zijn API’s keys & tokens, maar deze hoef ik niet te gebruiken voor mij api.

Om deze globale variabele te kunnen testen heb ik een simpele GET request van Category gekopieerd.

### Graphical user interface, text, application Description automatically generatedGraphical user interface, text, application, email Description automatically generated

Om onze globale variabele te kunnen gebruiken heb ik dit tussen dubbele accolades moeten schrijven. Wanneer ik dit aan het typen was kwam postman met allerlei andere globale variabelen af dat ik zou kunnen gebruiken. Dit kan je zien in de screenshot. Dit zijn verschillende standaard globale variabelen dat postman u aanbied om zo het werken te kunnen vergemakkelijken.

Zoals je dus uit de bovenstaande test kan afleiden kan je perfect zo met variabelen werken.

Graphical user interface, application

Description automatically generated

Variabelen kunnen ook zeer gemakkelijk gebruikt worden bij de Quary Params.

#### Globale variabelen in scripts.

Om globale variabele in scripts te gebruiken zoals bij test. Moeten we de volgende code schrijven

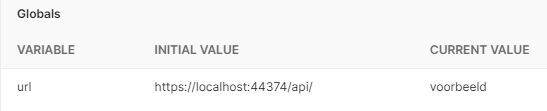


Waarbij ‘url’ de naam van de globale variabele is. Daarnaast kunnen we ook de waarde van een globale variabele veranderen. Dit zal ik aantonen met een test.

Graphical user interface, text, application, email

Description automatically generated

Zoals je kan zien is de url eerst nog de oorspronkelijke url. Maar nadat we hem aanpassen kunnen we zien dat dit veranderd is naar ‘voorbeeld’. Wanneer we nog is op het oogje drukken kunnen we dit ook zien.



We kunnen een globale variabele ook altijd verwijderen door de volgende code:

pm.globals.unset("url")

En we kunnen ook alle globale variabelen verwijderen door deze code:

pm.globals.**clear**();

#### Globale variabelen gebruiken in tests voor latere automatisatie

Hiervoor gaan we enkele al gemaakte testen kopiëren en daarna aanpassen met wat we al van globale variabelen hebben geleerd.

Graphical user interface, text, application, email

Description automatically generatedWe gaan eerst beginnen met een category POST. En na de POST voeren we de standaard testen in maar maken we ook globale variabele aan genaamd ‘CatId’ waar het id gaan instaan & ‘CatName’ waar de naam gaat instaan.

Graphical user interface, text, application, email

Description automatically generated

Hierna gaan we is controlleren met een GET request of er voor de id wel de juiste naam is aangemaakt.

Graphical user interface, text, application, email

Description automatically generated

Daarna gaan we deze categorie verwijderen.

Graphical user interface, text, application

Description automatically generated

En tenslotte gaan we kijken met nog een GET request of deze wel degelijk verwijderd is.

Nu kunnen we dus alle tests één voor één zodat we kunnen kijken of alles werkt.

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Dis is natuurlijk wel omslachtig om zo elke test één voor één te kunnen runnen. Maar door het voorbije gastcollege heb ik iets geleerd dat dit duidelijker kan maken. Ik heb dit nog niet mij mijn cursus geleerd maar ben er wel zeker van dat dit nog gaat komen. Maar ik zal het toch al eens laten zien.

Graphical user interface, application

Description automatically generated

Eerst heb ik alle testen in één map gestoken genaamd ‘Automatisation’ en ook in de juiste volgorde.

Graphical user interface, text, application, email

Description automatically generated

Door op deze map te klikken kunnen we ‘Run’ zien staan. Als je hierop klikt zal je alle testen dat zich in deze map bevinden kunnen uitvoeren en dit ziet er zo uit.

Graphical user interface, application

Description automatically generated

Nu kunnen we nog het aantal itteraties aanpassen, een delay en eventueel nog data toevoegen. Maar dit is niet nodig dus kunnen we op ‘Run Recipe API Software Testing’ drukken.

Graphical user interface, text, application, email

Description automatically generated

En nu dat we dit gedaan hebben kunnen we goed zien dat alle tests na elkaar zijn gerund en dat ze ook allemaal geslaagd zijn.

### Environment variables

Deze soort variabelen zijn vooral handig wanneer je verschillende soorten setups hebt voor een zelfde collectie. Je kan dus selecteren welke environment je zal gebruiken bij bv een localhost of een test omgeving terwijl je toch de zelfde tests zal gebruiken. En zo kan je dus voor verschillende soort testen met dezelfde testen andere variabele gebruiken voor elke test.

Een environment kan je aanmaken via hetzelfde ‘oog’ zoals bij globale variabelen.

Graphical user interface, text, application

Description automatically generated

Ik heb nu snel en eenvoudig door op Add te drukken twee environments aangemaakt met elk één variabele om het verder te kunnen tonen.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Zoals je kan zien heb ik er zeer simpele waarden ingezet zodat ik het snel en gemakkelijk kan aantonen.

Om een environment te kunnen selecteren kan je op het pijltje links van het oog drukken zodat je uit uw aangemaakte environments kan kiezen.

Graphical user interface, text, application, chat or text message

Description automatically generated

Nu heb ik een heel simpele request opgemaakt zelfs zonder request url om het gebruik van environment variabelen aan te tonen.

Graphical user interface, application

Description automatically generated

Nu kan je zien dat ik als environment ‘Test 1’ heb aangeduid en aan de hand van de value van de variabele (als je er over hovert met uw muis) kan je zien dat het idd van test 1 is.

Graphical user interface, text, application

Description automatically generated

Nu wanneer ik ‘Test 2’ aanduid als environment verandert de waarde van deze variabele. En dit is hier dus zo handig aan. Dit kan vooral ook handig zijn als je voor verschillende test situaties verschillende api tokens hebt. En zou zou je die dus gemakkelijk kunnen selecteren. Waardoor je dezelfde request kan gebruiken.

## Advanced assertions

## Automatically running tests

## Running Newman with other CI servers/tools

## Workflows and Scenarios

## Data driven tests: Running a request multiple times with different data sets

## Team Collaboration

## Mock servers

## File uploads (testing, automatic uploads, uploading multiple files)

## Authentication / Authorization

## Advanced topics and user questions

## New features in Postman

## JavaScript fundamentals

## Postman Developer Challenge 2021

## Bonus section

# Timetable

|  |  |  |
| --- | --- | --- |
| **Wat?** | **Tijdsduur** | **Datum** |
| Github in orde zetten | 30min | 4/10/2021 |
| Gepaste API zoeken + verslag | 30min | 5/10/2021 |
| Introduction and first steps in Postman | 55min | 5/10/2021 |
| Intro uitschrijven | 10min | 11/10/2021 |
| Creating with API requests (Requests) | 45min | 13/10/2021 |
| Creating with API requests (Vervolg) | 40min | 14/10/2021 |
| Een eerste test | 65min | 20/10/2021 |
| De tabel Categories simpel testen | 75min | 21/10/2021 |
| Data toevoegen aan de tabel recipe, + GetAll test | 90min | 25/10/2021 |
| De tabel Recipe verder testen | 80min | 27/10/2021 |
| Ingredient Controller & Service classe toevoegen aan C# | 60min | 28/10/2021 |
| Ingredient Controller & Service classe toevoegen aan C# deel 2 | 60min | 6/11/2021 |
| Ingredient Test data & debugging | 200min | 8/11/2021 |
| Ingredient Testen & aanpassen | 60min | 10/11/2021 |
| Ingredient verder testen | 90min | 16/11/2021 |
| Ingredient documentatie afwerken | 30min | 18/11/2021 |
| Globale variabelen | 120min | 18/11/2021 |
| Environments & environment variabelen | 30min | 23/11/2021 |
|  |  |  |
|  |  |  |
|  |  |  |
| **Opgetelde tijdsduur** | **1170min** |  |