**Toegepaste Informatica**

**Opdracht Software Testing**

*API Testing with POSTMAN*

**Nelly Nsengimana Ineza**

(r0699887)

Bachelor – Toegepaste Informatica

**Software Testing**

Frank Serneels

Academiejaar 2021-2022

Inhoud

# Onderzoek naar omgeving

Voor het kiezen van een testomgeving hadden we de keuze uit:

* Selenium WebDriver
* Postman API Testing
* Python testomgeving
* ASP.NET Core
* …

Tijdens de eerste les kregen we uitleg over elke omgeving. Ik twijfelde tussen API tesing met Postman en testen met Selenium. De reden hiervoor is omdat ik met beide omgevingen al in aanraking ben geweest tijdens het vorig academiejaar.

Ik heb dus om het maken van mijn keuze wat onderzocht op het internet om te weten met welke ik me comfortabel voelde, ookal had ik al een idee.

Op basis van mijn voorkennis en vooral interesse koos ik voor API testing met Postman. Omdat programmeren niet mijn sterkte is, stelde ik vast dat dit de beste keuze was.

# Onderzoek naar learning tools

Voor het testen zal ik uiteraard Postman nodig hebben. Zoals eerder vermeldt heb ik tijdens de vorige academiejaren als eens (beperkt) Postman gebruikt. Postman had ik dus al op mijn laptop. Met deze eerste stap was ik dus al klaar.

Voor het leren van het (API) testen met Postman ben ik op zoek gegaan naar verschillende beschikbare tutorials/cursussen. Naast het document verkregen van de docent, heb ik een tal andere links gevonden (zie document “rest api testing postman” op <https://github.com/Nelineza/SoftwareTesting>). Hierbij had ik een aantal online courses, videos en dergelijke. Sommigen hiervan waren gratis, anderen betalend.

Deze links heb ik doorgenomen, alleen en samen met de docent, om te zien welke de beste een meest volledig zijn. Ik besloot dan dat ik volgende bronnen ging gebruiken:

* Document (read-only) van docent
* Course on YouTube: <https://www.youtube.com/watch?v=VywxIQ2ZXw4>
* Course on Coursera: <https://www.coursera.org/projects/start-your-api-testing-journey-with-postman-tool>

Eventueel dacht ik als extra aanvulling volgende bronnen te gebruiken:

* Course on Udemy (Beginners guide, +- 2u): <https://www.udemy.com/course/introduction-to-postman-a-beginners-guide/>
* Course on Udemy (Complete guide, +- 14h): <https://www.udemy.com/course/postman-the-complete-guide/?ranMID=39197&ranEAID=JVFxdTr9V80&ranSiteID=JVFxdTr9V80-1Sk44PfRvJ_DQD5LGpXsdw&utm_source=aff-campaign&utm_medium=udemyads&LSNPUBID=JVFxdTr9V80>

Dit was voornamelijk voor in het geval dat het concept van API Testing en testing automation niet helemaal begrepen werd. Dit zal ook grotendeels afhangen van de time-management. In principe zouden de eerste bronnen voldoende moeten zijn.

***\*\* AL de bronnen zijn in het engels. In mijn documentatie zal ik dus delen in het engels schrijven en delen in het nederlands.\*\****

# Theorie

Om te starten besloot ik om de *course on Youtube* te volgen. Ik denk dat deze video zeker een goede start was. In deze course waren er verschillende hoofdstukken (30 in het totaal). Deze hoofdstukken gingen van een introductie, naar de basics (vb. CRUD operations), testen schrijven, testing automation,… Wat ik ook zeer leuk vond, was dat er tussendoor oefeningen werden gegeven. Dit was een goede manier om te zien of ik de lessen goed had begrepen.

## Les 1-5: Basics

API = an interface to a server that has some data or does some actions

Postman: a tool for interacting with web-based API’s

Before using an API, understand how to interact with that API by looking at the **API documentation**.

Les 6: Collections and variables

Endpoint: an endpoint offers different kind of responses (e.g. “status” is an endpoint)

Collection: a list of multiple requests (typically using the same API)

🡪 je kan dus bij het gebruiken van een API een collection bouwen met verschillende requests van die API als bijvoorbeeld GET order, GET all orders, DELETE order, …

Bij het verzenden van een request, krijg je een bepaald respons met een **status**. Er zijn verschillende statussen. De belangrijkste om te kennen zijn, status 200 (OK), 201 (Created), 400 (Bad request), 404 (Not found), 401 (Unauthorized)

Data wordt in de respons geformatteerd in **JSON**. (Je moet niet noodzakelijk een grote JSON kennis hebben).

Variabelen aanmaken: je kan variabelen aanmaken waarin je bijvoorbeeld de basis url bijhoudt. Vervolgens kan je dan deze variabele gebruiken en koppelen met de verschillende endpoints zonder de url steeds te moeten herschrijven. Deze variabele bewaar je in een bepaalde scope (global, collection,…zelf te kiezen) dus let op! Als je een variabele aanmaakt in een bepaalde collection, ga je deze niet kunnen gebruiken in een query zolang jouw query niet in die collection is opgeslagen.

Les 7: Query parameters

Query parameters = additional data that you can submit with your query.

Deze kunnen optioneel of verplicht zijn afhangend van de API. Je kan meerdere paramters hebben bij een query. Hierbij heb je een key en een value. Hier in mijn voorbeeld is de key “type” van boeken en is de value “fiction”.

A screenshot of a computer screen

Description automatically generated with medium confidence



De bedoeling is dus dat ik in mijn respons enkel fictieve boeken terug krijg.

Graphical user interface, text, application

Description automatically generated



Je kan weten wat je kan gebruiken als query parameters enkel door naar de *documentatie* van de API te kijken. Anders had ik hier in dit voorbeeld niet kunnen weten dat “fiction” een type was, maar had ik misschien “thriller” gebruikt (maar volgens de API documentatie is dit geen type dus had ik een fout gekregen met status 400 Bad request).

Ik had ook bijvoorbeeld een parameter kunnen uitvinden (key=blabla, value=test). Het uitvinden van deze parameter had niets veranderd aan de respons. Postman zou dus in deze situatie de uitgevonden parameter negeren en gewoon de repons tonen zonder rekening te houden met deze parameter (met status code 200 OK).

In mijn voorbeeld zijn er maar 2 optionele parameters beschikbaar, nl. “type” en “limit”.

Opletten: het is case-sensitive! “Limit” gaat dus niet de verwachtte respons teruggeven, maar “limit” wel.

Les 9: Path variables

This endpoint allows to specify a variable which represents one of the books that you’re trying to get detail information from (eg. “key=bookId” en “value=1”), instead of having it in a list with less information. XXX

Bij path variables is het niet de key die wordt verzonden maar de value. Hier maakt het dus niet uit of je als key “bookId” kiest of “id” of iets anders. Wat verzonden zal worden is de “1”.

## Les 10: Post request

AUTHORIZATION

Acces token: soort van een tijdelijk paswoord dat je krijgt en kan gebruiken bij uw request waarbij authorization is. Deze token sla je best op bij current value (niet initial value omdat de kans groot is dat andere mensen dit zien wat je niet wilt).

## Les 17: PATCH:

PATCH gebruik je om te updaten. Met patch kan je een body meegeven, hierin geef je de content mee die je wilt toevoegen.

## Les 18: DELETE

Deze soort request gebruik je uiteraard om iets te verwijderen.

## Les 19: Test automation

Algemeen: test automation is een techniek voor het testen van software om het werkelijke resultaat te testen en te vergelijken met het verwachte resultaat. Om dit te bereiken, schrijven testers testscripts en automatiseren ze die met behulp van een testautomatiseringstool (tests die steeds opnieuw kunnen worden uitgevoerd). U kunt testautomatisering gebruiken om repetitieve tests te automatiseren of tests die moeilijk handmatig uit te voeren zijn. Goede automatisering maakt testen sneller, systematischer, en vermindert menselijke fouten. Het doel van test automation is om de effectiviteit en efficiëntie van testen te verhogen.

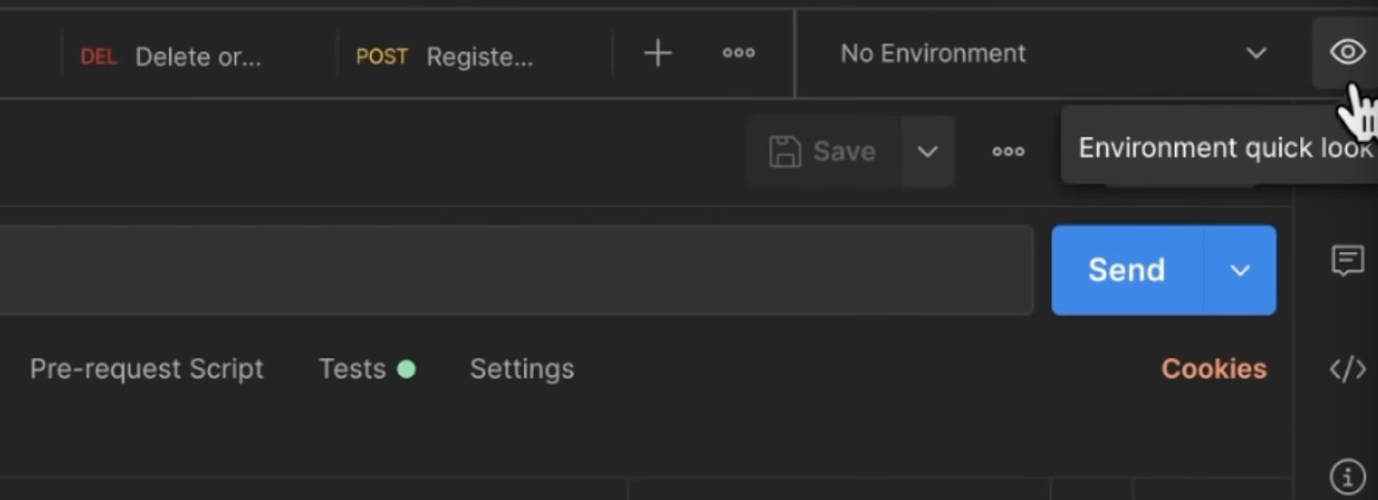
Postman kan worden gebruikt om vele soorten tests te automatiseren, waaronder unit tests, functionele tests, integratietests, end-to-end tests, regressietests, mock tests, enz. De bedoeling is dus om Postman de testen te laten doen, en wij moeten er enkel tussenkomen wanneer er iets fout loopt.

API automatisering laat u de core logica valideren zonder afhankelijk te zijn van de UI (User Interface). API testen zijn eenvoudig te automatiseren binnen dezelfde sprint. Met API automatisering kunt u al vroeg beginnen met het testen van uw applicatie - zelfs zonder een UI. Dit helpt om problemen vroeg in de ontwikkelingscyclus te identificeren en op te lossen die anders duur zouden zijn om op te lossen wanneer ze tijdens het testen van de UI zouden worden geïdentificeerd.

## Les 22: Postman variables

Het steeds moeten kopieëren en plakken van bijvoorbeeld een *order id* in de verschillende requesten is vervelend (zie oefening status code).

Om zo’n situaties te vergemakkelijken kunnen we een Postman variabele gebruiken om hierin de *order id* te zetten. Hiervoor moet je de value (order id) kopieëren en vervolgens hierop drukken:





Je kan dan zo een global variable of een environment variable aanmaken. We gaan nu een global variable aanmaken door op edit te klikken.

Une image contenant texte

Description générée automatiquement



We vullen de tabel aan door de variabele een naam te geven (orderId). Initial value kunnen we leeg laten. In de current value plakken we de id. De variabele is nu aangemaakt.

Une image contenant texte

Description générée automatiquement



Je kan de waarde van dezelfde variabele makkelijk aanpassen door hierop te klikken:

Une image contenant texte, capture d’écran, moniteur, écran

Description générée automatiquement



In het vervolg wanneer je een request wilt sturen waarbij je de *order id* nodig hebt, moet je niet meer die id gaan kopieëren en plakken in het veld “value” maar kan je gewoon dit invullen:

Une image contenant texte, capture d’écran, moniteur, noir

Description générée automatiquement



Voilà! Nu hebben we een manuele taak geautomatiseerd en dus vergemakkelijkt.

## Les 23: Extracting data from the response

Als we kijken naar enkele requests, bijvoorbeeld de *GET List of books* request, zien we dat we nog hard gecodeerde data hebben in de response. Bij aanpassingen is hard gecodeerde data af te raden (zoals bijvoorbeeld als er een boek wordt verwijdert). We moeten dus een manier vinden om deze data dynamischer te maken i.p.v. hard gecodeerd.

We gaan een deel code schrijven voor het plaatsen van die data in een variabele. We gaan hiervoor eerst de repsonse parsen naar een javascript object.

# Oefeningen

## Eerste request met Postman

A screenshot of a computer

Description automatically generated with medium confidence

## Eerst testen schrijven in Postman

Bovenaan kan je een rubriek “tests” vinden. Hier is waar je de code van de testen kan schrijven.

Graphical user interface, application

Description automatically generated

Postman zet ook “snippets” ter beschikking. Dit zijn vooraf-gemaakte typische testjes die je kan gebruiken. Een typische test is bijvoorbeeld nakijken of de status code 200 OK is. Hiervoor moet je deze test dus niet meer schrijven, maar kan je deze simpelweg van de snippets halen.

Text

Description automatically generatedGraphical user interface, text

Description automatically generated with medium confidence



Bij het verzenden van uw request gaat Postman dus nu ook deze test uitvoeren. Het resultaat kan je hier vinden:

Graphical user interface, application

Description automatically generated



Nu ga ik deze test laten falen om te zien hoe dit resultaat eruit ziet:

Graphical user interface, application, website

Description automatically generated

We zien dus dat de verwachtte resultaat “status code is 200” is. Naast dit verwachtte resultaat krijgen we een *AssertionError*. Hierbij staat er dus wat er is gebeurt (waardoor de test is gefaald).

Nu gaan we naar de volgende stap: het is ook belangrijk dat je de response body kan nakijken. Meestal zal deze in JSON zijn. We gaan nu een test schrijven die nakijkt of de status effectief “OK” is (in de body). We gaan hiervoor de response *parsen*. Waarom? Omdat de response body in JSON staat, maar eigenlijk is dit gewoon in ‘tekst’. Wij willen dit als een object in javascript (en dus niet tekst).

We gaan dus een (javascript) variabele definiëren “response”. “pm.repsponse” houdt de respons bij. “json()” is een functie van Postman om de respons van json naar javascript te parsen. Deze zal dus in de variabele “reponse” worden bijgehouden.

Graphical user interface, text

Description automatically generated

Om enkel de waarde terug te krijgen, dus hier enkel de “OK” i.p.v. “status: OK”, kan je “console.log(response.status)” schrijven i.p.v. “console.log(response)”.

Het resultaat in de console lijkt misschien op JSON, maar het is eigenlijk een javascript object.

Text

Description automatically generated

Nu ga ik zelf eens een test schrijven (en geen voorgemaakte test van de snippets). Omdat dit mijn eerste zelfgeschreven test is ga ik het heel simpel houden. Ik ga testen of “1” gelijk is aan “1”.

Graphical user interface, text, application

Description automatically generated

Een test start je altijd met “pm.test” in Postman. Hierbij zijn er 2 parameters.

Als eerste parameter schrijf ik de naam van de test (kies je zelf) en als tweede parameter (na de komma) schrijf ik de function. Het resultaat hiervan is te verwachten, en zou dus “PASS” moeten zijn. Bij het verzenden zie ik dat deze test effectief geslaagd is.

Une image contenant texte

Description générée automatiquement



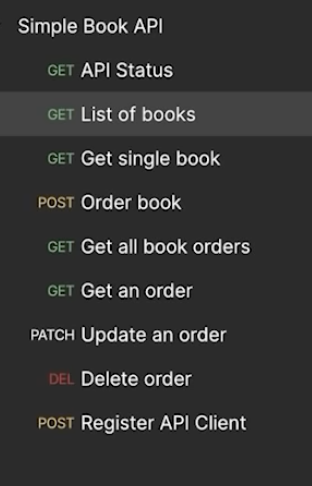
Ik ga nu “eql(1)” eens vervangen door “eql(2)” om uit te testen. 1 is niet gelijk aan 2, dus deze test zou moeten falen. Laten we eens zien:

Une image contenant texte, capture d’écran, moniteur, écran

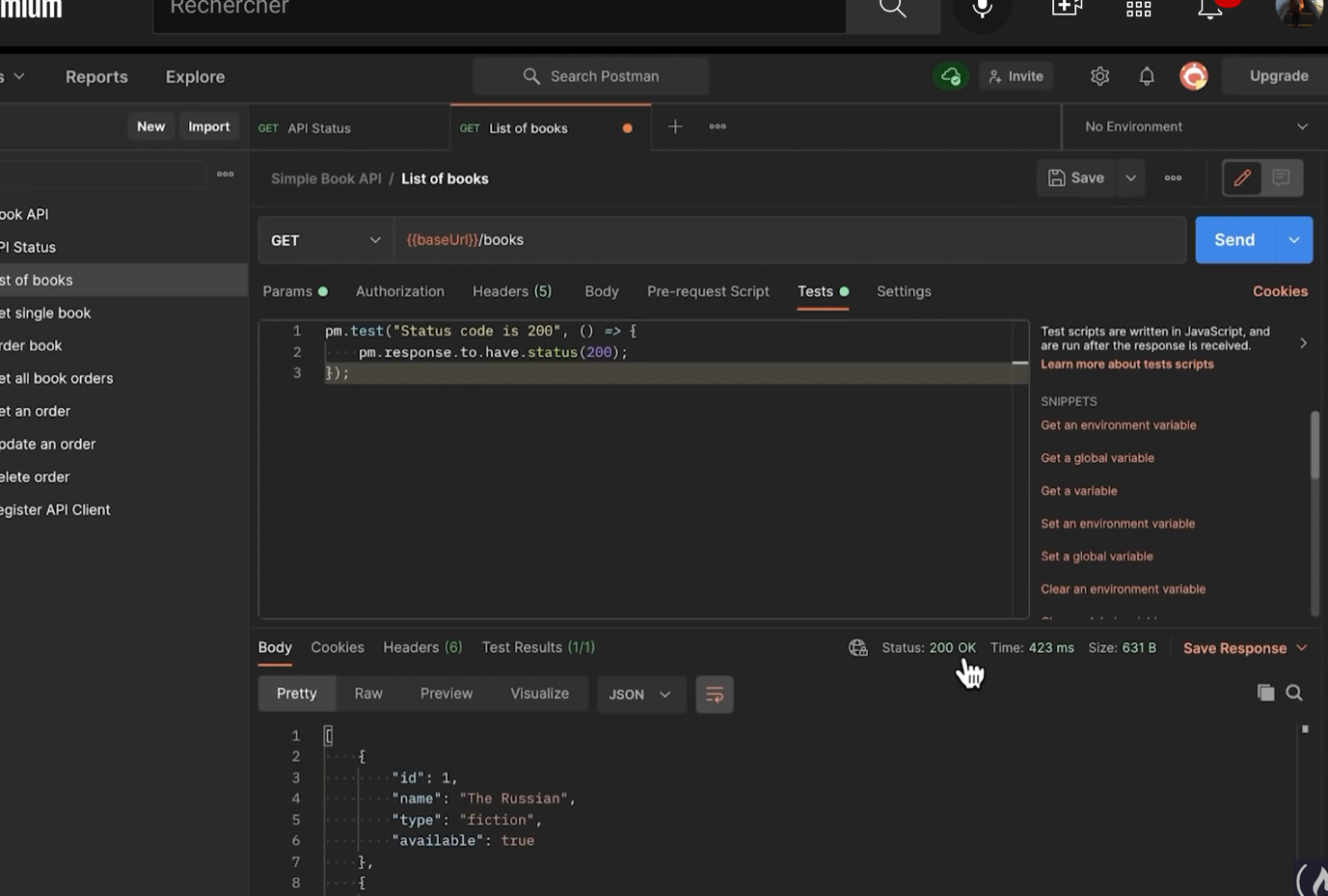
Description générée automatiquement

## Oefening: test de status code

Als oefening ga ik een *status code test* schrijven voor alle requests in de collection. Deze testen moeten ook kunnen falen. Dit zijn de verschillende requests:

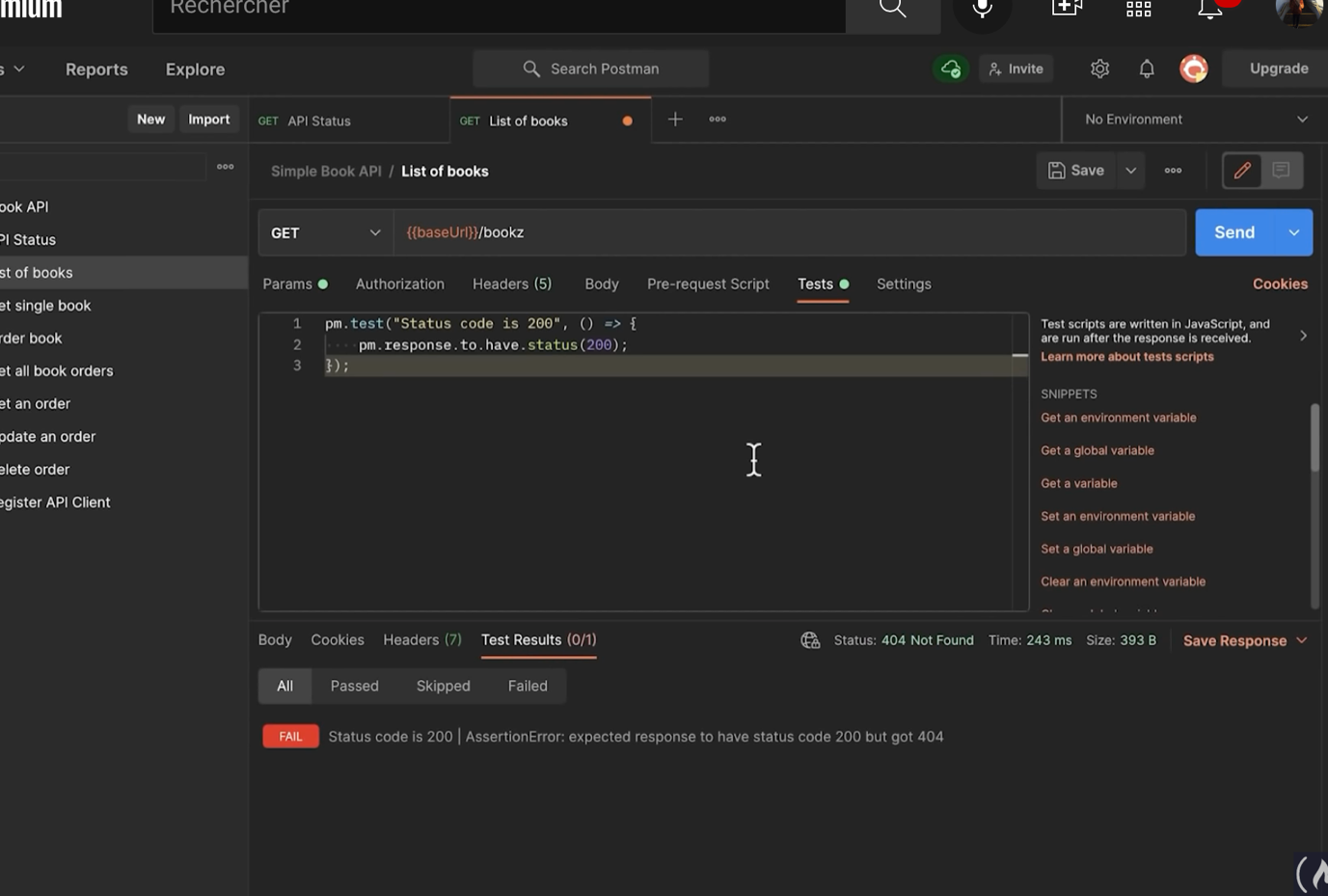


Om te starten vertrek ik van de code die ik al had vanuit de eerste test. Hierbij moet ik maar een kleine aanpassing maken. Ik begin eerst met de GET list of books request. Deze lukte zonder probleem.





Ik heb dan “books” vervangen door “bookz” om te zien of deze test zou falen. De test faalde effectief.





Ik deed vervolgens de test voor de GET single book request. Dit verliep op dezelfde manier als de vorige. Voor de POST order book request was het wat anders. Ik deed eerst hetzelfde als bij de eerste 2 testen. Als resultaat van de test kreeg ik echter een “fail”. Dit komt omdat mijn status code niet “200 OK” was, maar “201 CREATED”. Ik heb dan in mijn test “200” vervangen door “201”. Nu lukte de test wel.

Une image contenant texte, moniteur, capture d’écran, noir

Description générée automatiquement



Bij de PATCH request was het gelijkaardig als de POST. In de plaats van “200” of “201” was het hier dan “204”.

Wat sterk opvallend is bij het maken van deze oefening, is dat ik vaak moest kopieëren en plakken bij het schrijven en uitvoeren van deze testen. De testen hadden allemaal dezelfde principe. Dit was uiteraard wat vervelend. Hier komt testing automation dan!

## Oefening: schrijf een verwachting en test deze uit

We hebben een boek waarvan we weten dat deze beschikbaar is. Bouw de code op van les 23 en schrijf een verwachting waarbij wordt verwacht dat de *type “non-fiction”* (parameter) is. Probeer het ook te laten falen.

Je kan makkelijk beginnen door ten eerste de vorige expectation (vorige lijn) te kopieëren. Nu gaan we moeten kijken naar de type, dus i.p.v. “book.available” schrijven we “book.type”. Als value willen we “non-fiction”, dus dit gaan we invullen i.p.v. “true”. Nu kunnen we de request verzenden (vergeet niet de parameter met key “type” en value “non-fiction” aan te vinken!).

Une image contenant texte

Description générée automatiquement



Nu gaan we eens de parameter uitvinken en de request terug verzenden. De “book found” test zou moeten falen.

Une image contenant texte, capture d’écran, moniteur, noir

Description générée automatiquement



## Oefening: global variable

Gebruik de eerder aangemaakte globale variabele “bookId” in de GET Single book request en de POST Order book request.

# Logboek

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Software Testing: Logboek | | |  | |
|  | | | | |
| Datum | Beschrijving | Duur | | Commentaar |
| 20/09/2021 | Zoeken welke bronnen/courses ik zal gebruiken | 60 min. | | Ik heb verschillende bronnen gevonden die zeer interessant lijken. Ik moet echter nog dieper inzien welke *volledig* zijn voor dit vak. |
| 27/09/2021 | Bronnen/courses uitkiezen | 30 min. | | Na bespreking met de docent heb ik geselecteerd welke bronnen ik effectief zal gebruiken. |
| 27/09/2021 | Installatie Postman | 15 min. | | Postman was al geïnstalleerd op mijn laptop. Ik heb deze nu ook geïnstalleerd op mijn desktop. |
| 29/09/2021 | Youtube course: les 1-8 | 45 min. | | Ik kon volgen en had geen moeilijkheden dankzij de zeer duidelijke uitleg. Ik volgde mee op mijn eigen Postman applicatie. Er was ook een eerste oefening bij (deze lukte). |
| 08/10/2021 | Youtube course: les 9-16 | 50 min. | | Bij dit gedeelte waren er 2 kleine oefeningen (gelukt). Bij les 10 moest ik wat terug gaan en herbekijken omdat ik niet meteen helemaal alles begrijp (API authentication, tokens). |
| 22/10/2021 | Youtube course: les 17-21 | 75 min. | | Bij dit gedeelte leerde ik o.a. mijn eerste test maken. Dit had ik zeker niet alleen gekunt zonder de duidelijke uitleg van deze course. Ook hierbij was er een oefening te maken. |
| 4/11/2021 | Document “API testing and development with Postman” (read-only): Considerations for good API test automation (chapter 4) | 70 min. | | Hieruit heb ik wat elementen bijgeleerd die niet (of kort) aan bod kwamen in de Youtube Course zoals bv. de verschillende type testen, uitgebreide uitleg van de scopes… Dit was eigenlijk een soort van herhaling + aanvulling |
| 15/11/2021 | Document “API testing and development with Postman” (read-only): Understanding authorization options (chapter 5 1ste helft) | 90 min. | | Zoals ik eerder had vermeldt had ik het concept van authorization en tokens niet helemaal door. In dit deel van het document werd het uitgebreider uitgelegd waardoor ik het nu beter begrijp. |
| 18/11/2021 | Document “API testing and development with Postman” (read-only): Understanding authorization options (chapter 5 2e helft) | 80 min. | | Zoals ik eerder had vermeldt had ik het concept van authorization en tokens niet helemaal door. In dit deel van het document werd het uitgebreider uitgelegd waardoor ik het nu beter begrijp. |
| 24/11/2021 | Youtube course: les 22-25 | 60 min. | | Dit deel was iets complexer. Ik zal hoofdstuk 23 nog eens doornemen/herhalen voor een beter begrip. Er waren ook 2 oefeningen waarvan 1 kleine en 1 grotere. De kleine is goed gelukt, maar bij de grotere oefening heb ik wat gesukkelt. |
| - | Youtube course: les 26-30 (laatste deel) |  | |  |
| - | Document |  | |  |
|  |  |  | |  |