Spotitube Integratie Testen

Integratie testen voor de Spotitube applicatie.

Maarten van der Lei (658215)

Klas: ITA-OOSE-A-f

Datum: 27-mei-2022.

Course: DEA herkansing.

Opdrachtgever: Wiebe Rinsma.

HAN-OOSE

DEA-Spotitube

Inhoud

[1. Inleiding 2](#_Toc104496603)

[2. Onderzoeksmethode 3](#_Toc104496604)

[3. Resultaten 5](#_Toc104496605)

[3.1. Opzetten 5](#_Toc104496606)

[3.1.1. SuperTest 5](#_Toc104496607)

[3.1.2. JUnit 5](#_Toc104496608)

[3.1.3. Postman 5](#_Toc104496609)

[3.2. Snelheid end-points 6](#_Toc104496610)

[3.3. Leesbaarheid 8](#_Toc104496611)

[3.3.1. JavaScript SuperTest & Jest 8](#_Toc104496612)

[3.3.2. Java HttpUrlConnection & JUnit 8](#_Toc104496613)

[3.3.3. Postman requests 9](#_Toc104496614)

[3.4. Documentatie 10](#_Toc104496615)

[3.4.1. Postman 10](#_Toc104496616)

[3.4.2. SuperTest & Jest 11](#_Toc104496617)

[3.4.3. HttpUrlConnection & JUnit 11](#_Toc104496618)

[3.5. Integratie testen 12](#_Toc104496619)

[3.5.1. Wat zijn integratie testen? 12](#_Toc104496620)

[3.5.2. Integratie testen binnen een project 12](#_Toc104496621)

[3.5.3. Wat is Node.js? 12](#_Toc104496622)

[3.6. Voordelen integratie testen met Node.js 12](#_Toc104496623)

[3.7. Nadelen integratie testen met Node.js 12](#_Toc104496624)

[4. Discussie 14](#_Toc104496625)

[4.1. Beantwoorden van deelvragen 14](#_Toc104496626)

[4.2. Oordeel deelvragen 16](#_Toc104496627)

[4.3. Beantwoorden hoofdvraag 16](#_Toc104496628)

[5. Conclusie 18](#_Toc104496629)

[Verwijzingen 19](#_Toc104496630)

# Inleiding

Voor de course OOSE-DEA is verzocht om een onderzoek te doen naar een verdieping of uitbreiding van de gemaakte Spotitube REST-API.  
Deze Spotitube applicatie is geschreven in Java en bevat unit tests.

De Spotitube opdracht was een relatief klein project, maar toch vond ik dat ik te lang bezig was om mijn resultaten te testen via de aangereikte website van de HAN.  
Daarom heb ik besloten om een onderzoek te doen naar hoe je integratie testen het beste kan uitvoeren, met de hand of met een script.

Integratie testen zijn bedoeld om de uiteindelijke werking & output van een systeem te testen. Waar e2e bedoeld is om de end-to-end experience te testen van een eindgebruiker.  
In de context van de Spotitube API is daarom niet gekozen voor een onderzoek naar e2e tests van wege het feit dat de developers die de Spotitube API maken niet de front-end gemaakt hebben.

Om de volledigheid van de applicatie te testen kan je handmatig testen, of je kan een script schrijven waarmee je automatisch je volledige applicatie kan testen, dit heet integratie testen.

Het doel van dit onderzoek is om te weten te komen of er winst is, en zo ja, wat de winst is van integratie testen op de Spotitube applicatie met NodeJS.  
Binnen NodeJS zijn er een verschillende integratie test frameworks, in dit onderzoek word alleen gekeken naar SuperTest in combinatie met Jest.  
SuperTest is voor NodeJS populair met ruim 4miljoen downloads per week en kan gebruikt worden zonder NodeJS applicatie om requests te sturen.

Omdat dit project gemaakt is met Java zal er ook gekeken worden naar de verschillen met Java.

Daarvoor heb ik een hoofdvraag opgesteld: “*Wat is de winst van het gebruik van integratie testen voor de Spotitube applicatie die gebaseerd zijn op Node.js(Jest & SuperTest)?”*”

----------------------------------------------

In opeenvolgende hoofdstukken ga je lezen of er winst is, en zo ja, wat de winst is van het integratie testen met node.js op de Spotitube applicatie.

Het doel van dit onderzoek is dan ook om aan te tonen of het winstgevend is, in tijd en gebruiksgemak, om integratie testen te maken voor de spotitube applicatie.

De winst kan uitgedrukt worden in snelheid, voor/nadelen bij gebruik in een soortgelijk java project, gemakkelijkheid van opzetten van een test en het lezen van de integratie testen.

In de Spotitube applicatie zijn al unit-tests gemaakt, maar een unit-test test niet het geheel, maar één unit. Met een Integratie-test test je de samenhang van components of services van elkaar, oftewel: de uiteindelijke resultaten die de webserver binnen zou moeten krijgen.  
*“Integratie testen is vaak de 2e stap in het testen na unit tests zijn geschreven.”* (dzone, 2022).

Er is gekozen om geen onderzoek te doen naar e2e (end-to-end) tests, die keuze is gemaakt omdat de gebruiker wel via de website van Spotitube de API benaderd, maar de developers van de API niet de webserver gebouwd hebben.  
Om die reden testen we dus alleen de gemaakte API met een integratie test.

# Onderzoeksmethode

Om deze de hoofdvraag te beantwoorden heb ik de volgende deelvragen opgesteld:

* Is een integratie test sneller dan handmatig testen?
* Welke test manier heeft de meest geschikte documentatie? Handmatig met Postman of integratie testen met NodeJS?
* Wat is leesbaarder? Het testen via Postman of het schrijven van integratie tests met NodeJS?
* Waarmee ben je in een wat langer project beter af? Handmatig of automatisch?
* Hoe verhoud NodeJS zich tussen de applicatie taal: Java?

Om deze deelvragen te beantwoorden ga ik experimenten uitvoeren en online onderzoek doen.

Dit experiment zal bestaan uit het maken en uitvoeren van verschillende integratie testen op 2 manieren, die verschillende endpoints van de gemaakte Spotitube applicatie aanroepen en de return-waardes checkt, tijdens deze tests worden timers bijgehouden(ingebouwd in de software om menselijke fout tegen te gaan) en zal de documentatie en gemak van opzetten bekeken worden.  
De 2 manieren van testen zijn:

* Met de hand (Postman of browser)
* Met een script:
  + NodeJS (SuperTest & Jest)
  + Java (HTTP & Junit)

Met de resultaten van dit experiment ga ik de deelvragen te beantwoorden om daarna een oordeel te vellen over de hoofdvraag.

Voor dit onderzoek zijn de volgende tools gebruikt:

* NPM
* SuperTest (JavaScript)
* Postman en browser.
* VSCode
* Java

Met behulp van de meest populaire integratie test tool, kunnen er verschillende metingen uitgevoerd worden op de spotitube applicatie.

Vervolgens kan er vergeleken worden met de handmatige http-requests, en hoelang dat duurde.

Ook wordt er gekeken naar de leesbaarheid van een integratie test binnen Node.js, samen met hoe snel je zelf een integratie test kan schrijven.

De **snelheid** zal getest worden door 10x dezelfde test te runnen, dat zal vergeleken worden met 10x dezelfde endpoint te benaderen.

Snelheid kan getest worden door middel van postman, website naar webserver(uitlezen in console) en integratie test.

**Leesbaarheid** wordt bepaald door middel van naamgeving van ingebouwde functies, opbouw van benodigde code en hoeveelheid code nodig voor 1 test.

**Projectbasis**, er zal gekeken worden wat de beste manier is om in een project/bij productie met integratie testen te werken, en wat de winst van Node.js hier dus is.

Code moet voldoen aan de volgende regels:

Code hoeveelheid van 1 test mag niet meer dan 10 regels hebben(afgezien van globale test opzet zoals classes of describes).

Functies ingebouwd in Supertest moeten leesbaar zijn in wat ze doen (geen functienamen zoals: gtstd(), maar wel zoals: getTimeSinceThisDate()).

De opbouw van code moet logisch zijn en voor een gemiddelde programmeur, van boven naar beneden goed te lezen zijn.

# Resultaten

Getest met clean browser: geen cookies, getest met target folder gereset.  
In Postman krijg je net zoals bij je browser in het network tabje een request te zien die word gedaan, daarbij komt de request tijd.  
Browser is getest via de ‘jenkins.aimsites’ Spotitube website.  
Postman is getest met de laatste versie van postman en op localhost.  
Java is getest met JDK 17, ingebouwde packages van java.net en met JUnit die we ook bij de Spotitube applicatie hebben gebruikt.

## Opzetten

Hier onder lees je hoe je de verschillende omgevingen opzet.

### SuperTest

Het opzetten van SuperTest is snel gedaan mits je NodeJS en NPM op je machine hebt staan.  
Waarna je het volgende commando runt: ‘npm install supertest jest –save-dev’.  
Afhankelijk van je internet snelheid is dit meestal binnen **20** **seconden** klaar.  
Waarna je een eerste test aan kan maken in de ‘\_\_tests\_\_’ eindigend op ‘.test.js’.  
Om je Tests je runnen maak je in je ‘package.json’ een script aan die ‘test’ heet, daar in zet je het volgende: ‘jest’. Jest gaat dan op zoek naar de ‘\_\_tests\_\_’ folder en alle bestanden eindigend op ‘.test.js’ worden dan uitgevoerd.

### JUnit

Omdat je met de ingebouwde HttpConnectionUrl, een verzoek kan maken naar een URL, kan je JUnit gebruiken om je test te maken.  
In Java HttpUrlConnection & JUnit kan je zien hoe dit samen met elkaar geïmplementeerd is.  
Door je test toe te voegen aan de test folder van je Spotitube project, zorg je dat JUnit het automatisch detecteert en runt.

### Postman

Om Postman te gebruiken ga je naar de Postman hoofdpagina waarna je direct te zien krijgt of je de desktop app wil downloaden.  
Afhankelijk van je download snelheid, de Postman applicatie is circa 150MB en met een gemiddelde verbinding binnen **10 minuten** gedownload.  
Daarna vul je de gewenste URL in, de gewenste methode en de eventuele toevoegingen zoals query parameters of een request body en klik je op: “send”.  
Waarna je de response uit kan lezen in Postman.

## Snelheid end-points

Hier onder lees je de request tijden van het experiment uitgelegd in de onderzoeksmethode.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| /login wrong creds | Postman | browser | Integratietest NodeJS | Integratietest Java |
| 1 | 97 | 90 | 21 | 95 |
| 2 | 89 | 91 | 47 | 15 |
| 3 | 82 | 90 | 20 | 17 |
| 4 | 89 | 81 | 20 | 15 |
| 5 | 108 | 83 | 21 | 16 |
| 6 | 77 | 86 | 24 | 15 |
| 7 | 86 | 88 | 23 | 15 |
| 8 | 94 | 87 | 19 | 66 |
| 9 | 82 | 83 | 22 | 60 |
| 10 | 88 | 87 | 20 | 13 |
| AVG | 89,2 | 86,6 | 23,7 | 32,7 |

*Snelheid is in milliseconden.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| /login correct creds | Postman | browser | Integratietest NodeJS | Integratietest Java |
| 1 | 102 | 87 | 119 | 142 |
| 2 | 94 | 93 | 120 | 39 |
| 3 | 87 | 84 | 118 | 32 |
| 4 | 107 | 89 | 128 | 166 |
| 5 | 98 | 98 | 125 | 30 |
| 6 | 90 | 105 | 112 | 141 |
| 7 | 88 | 95 | 121 | 29 |
| 8 | 91 | 102 | 122 | 78 |
| 9 | 104 | 99 | 117 | 75 |
| 10 | 105 | 91 | 127 | 25 |
| AVG | 96,6 | 94,3 | 120,9 | 75,7 |

*Snelheid is in milliseconden.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| /playlist correct token | Postman | browser | Integratietest NodeJS | Integratietest Java |
| 1 | 252 | 240 | 183 | 28 |
| 2 | 278 | 232 | 192 | 122 |
| 3 | 282 | 235 | 179 | 128 |
| 4 | 256 | 239 | 193 | 26 |
| 5 | 277 | 253 | 195 | 24 |
| 6 | 273 | 255 | 184 | 25 |
| 7 | 272 | 248 | 183 | 72 |
| 8 | 275 | 286 | 182 | 77 |
| 9 | 270 | 253 | 181 | 80 |
| 10 | 262 | 247 | 179 | 76 |
| AVG | 269,7 | 248,8 | 185,1 | 65,8 |

*Snelheid is in milliseconden.*

*De volgende test is alleen getest met Postman en integratie test in NodeJS, omdat de aanpassingen in de browser hier te lang zouden duren om volledig te testen en er met Java niet volgens de zover geteste manier getest kon worden voor onduidelijke redenen.*

|  |  |  |
| --- | --- | --- |
| /playlist wrong token | Postman | Integratietests NodeJS |
| 1 | 72 | 16 |
| 2 | 69 | 17 |
| 3 | 77 | 20 |
| 4 | 72 | 17 |
| 5 | 80 | 16 |
| 6 | 84 | 17 |
| 7 | 85 | 17 |
| 8 | 75 | 19 |
| 9 | 79 | 16 |
| 10 | 87 | 15 |
| AVG | 78 | 17 |

*Snelheid is in milliseconden.*

## Leesbaarheid

Hier onder vind je de leesbaarheid van de manier van testen.

### JavaScript SuperTest & Jest

1. const supertest = require('supertest');
2. const request = supertest(process.env.API\_URL || 'http://localhost:8010');
3. it('POST /login wrong creds', async () => {
4. const times = [];
5. for(let i = 0; i < 10; i++) {
6. const start = Date.now();
7. // actual test:
8. await request.post(`/login`)
9. .send({
10. user: 'user',
11. password: 'incorrect',
12. })
13. .expect(401)
14. .then((res) => {
15. expect(res.body.token).toBe(undefined);
16. expect(res.body.user).toBe(undefined);
17. });
18. // test end.
19. const end = Date.now();
20. const time = end - start;
21. times.push(time);
22. }
23. console.table(times);
24. });

*Voorbeeld met onjuist wachtwoord/user combinatie + speedtest for-loop direct uit mijn code.*

### Java HttpUrlConnection & JUnit

1. package nl.han.oose.vdlei.spotitube.research;
2. import org.json.JSONObject;
3. import org.junit.jupiter.api.Test;
4. import java.io.BufferedInputStream;
5. import java.io.ByteArrayOutputStream;
6. import java.io.OutputStream;
7. import java.io.OutputStreamWriter;
8. import java.net.HttpURLConnection;
9. import java.net.URL;
10. import java.util.ArrayList;
11. import java.util.Date;
12. import static org.junit.jupiter.api.Assertions.*assertEquals*;
13. public class IntegrationIT {
14. @Test
15. public void CorrectLoginTest(){
16. ArrayList times = new ArrayList<>();
17. JSONObject obj = new JSONObject();
18. obj.put("user","morty");
19. obj.put("password","mortypassword");
20. for (int i = 0; i < 10; i++){
21. var start = new Date().getTime();
22. try {
23. // actual test:
24. URL url = new URL("http://localhost:8003/login");
25. HttpURLConnection con = (HttpURLConnection) url.openConnection();
26. con.setDoOutput(true);
27. con.setRequestMethod("POST");
28. con.setRequestProperty("Content-Type", "application/json");OutputStream os = con.getOutputStream();
29. OutputStreamWriter osw = new OutputStreamWriter(os, "UTF-8");
30. osw.write(obj.toString());
31. osw.flush();
32. osw.close();
33. os.close();
34. con.connect();
35. BufferedInputStream bis = new BufferedInputStream(con.getInputStream());
36. ByteArrayOutputStream buf = new ByteArrayOutputStream();
37. int result2 = bis.read();
38. while(result2 != -1) {
39. buf.write((byte) result2);
40. result2 = bis.read();
41. }
42. String result = buf.toString();
43. System.*out*.println(result);
44. *assertEquals*(con.getResponseCode(), 200);
45. }catch (Exception e) {
46. e.printStackTrace();
47. // test end.
48. } finally {
49. var end = new Date().getTime();
50. var time = end - start;
51. times.add(time);
52. }
53. }
54. System.*out*.println(times);
55. }
56. }

*Voorbeeld van java code met HttpUrlConnection uit eigen test: correct login.*

De login route heeft hier 2 scenario’s: 1 met succesvolle login, nagebootst met Java.  
1 met de foutieve login nagebootst door NodeJS.  
Beide tests zijn in dezelfde taal even veel regels.

### Postman requests

Afbeelding met tekst, monitor, schermafbeelding, zwart

Automatisch gegenereerde beschrijving  
*Postman test methode uit eigen postman omgeving.*

Met Postman kan je een URL opgeven en een Methode, waarna er een grote “send” knop naast staat waarmee je het verzoek verstuurt.  
Postman is gebouwd op chromium en daarmee gebruikt het dezelfde http verzoek methode als de reguliere browser, dit betekend dat je al je response en request waardes kan terug lezen, in de browser kan dit in je ‘network’ tabje, en hier in postman zie je dit op een rijtje.

## Documentatie

Hier onder lees je de verschillende documentatie per test methode.

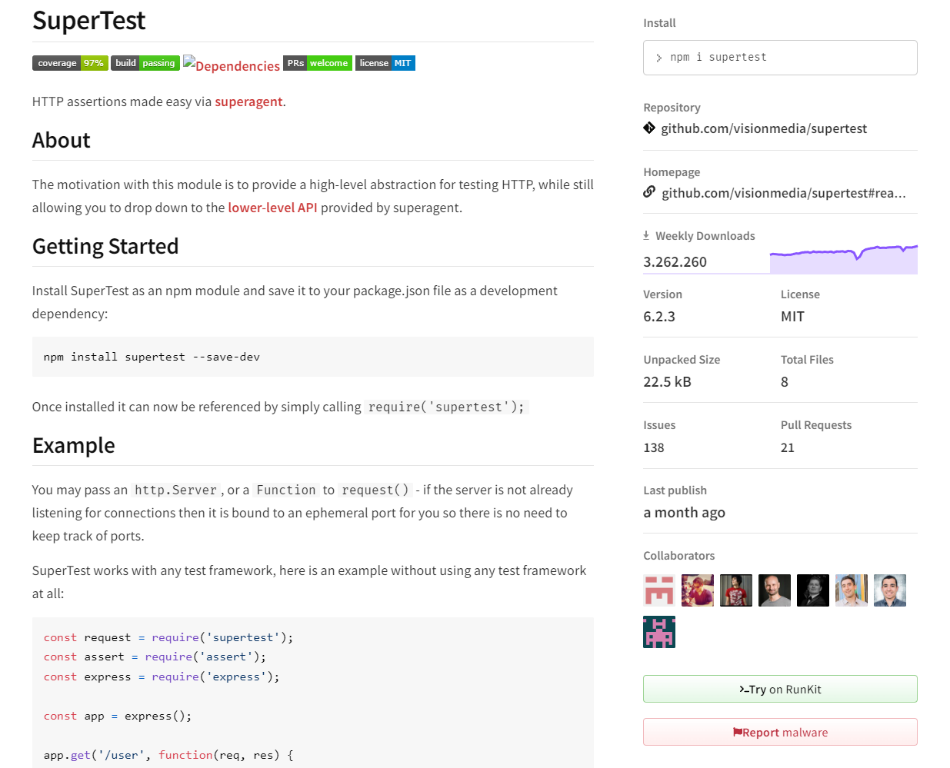
### Postman

Afbeelding met tekst

Automatisch gegenereerde beschrijving  
*Postman documentatie van de* [*https://learning.postman.com/docs/sending-requests/requests/*](https://learning.postman.com/docs/sending-requests/requests/) *site.*

Postman heeft een heleboel documentatie over hoe je een verzoek verstuurt en je kan hier dan ook op alles zoeken en je krijgt wel een soort antwoord, ik heb hier ingetypt: ‘http test’ en dit was het eerste resultaat.

### SuperTest & Jest

  
*SuperTest documentatie van de* [*https://www.npmjs.com/package/supertest*](https://www.npmjs.com/package/supertest) *site.*

SuperTest heeft een hoop voorbeelden die je kan gebruiken binnen je eigen tests.  
Elke functie van SuperTest is binnen deze pagina uitgelegd en hoe je die kan gebruiken.

### HttpUrlConnection & JUnit

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*Documentatie van de HttpUrlConnection van de* [*https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html*](https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html) *site.*

In de Oracle docs staan alle methods uitgelegd met welke parameters ze krijgen en wat ze returnen.

## Integratie testen

In dit hoofdstuk vind je even kort wat informatie over wat integratie testen inhoud, wat het verschil is met e2e testen en wat de invloed is van integratie testen binnen een project om zo de workflow van het project te beïnvloeden.

### Wat zijn integratie testen?

Een integratie test is een test die niet alleen een unit test, maar meerdere units, modules of componenten binnen een applicatie test.

Het doel van een integratie test is dan ook om de connectiviteit en samenhang tussen units te testen (techtarget, 2022)

### Integratie testen binnen een project

Integratie testen binnen een project is belangrijk omdat je vooraf je endpoints kan bedenken, vast stelen, en vervolgens maken naar de specificaties, om ze daarna te testen met de voorafgaand gemaakte integratie testen, ook wel TDD(Test-Driven-Development) genoemd.

*Bron: Dea-course en eerdere projecten*.

**Maar waarom is het dan juist binnen een project belangrijk?**

Omdat je binnen een project meestal vanuit specificatie werkt, en de code schrijft om aan de specificatie te voldoen.  
Met integratie testen kan je elke commit/ronde/sprint, de tests runnen om te kijken welke endpoints al aan de verwachtingen voldaan zijn.  
Ook kan je met een pipeline een “*Continious Integration*” workflow opzetten, een van de manieren is github actions, daarmee kan je de integratie tests runnen bij een commit/push.

### Wat is Node.js?

Node.js is een javascript runtime environment gebouwd op [chrome V8](https://v8.dev/)

Met NodeJS kan je bestanden eindigend op .js uitvoeren op je machine zonder gebruik te maken van een browser.  
Met NodeJS kan je bij de Runtime van je PC, en dus bij je bestanden en CLI omgeving.

Voor meer info: <https://nodejs.org/en/about/> .

//\*\* zijn resultaten van bronnen onderzoek.

## Voordelen integratie testen met Node.js

Voordelen van integratie testen met Node.js voor de Spotitube applicatie:

* Je test van buiten af, en kan dus de gebruikers ervaring beter nabootsen.
* Je schrijft makkelijk te lezen code met javascript in combinatie met async/await of Promises.

## Nadelen integratie testen met Node.js

nadelen van integratie testen met Node.js voor de Spotitube applicatie:

* Omdat je van buiten de Spotitube applicatie test, heb je geen invloed op de database connectie en ben je dus verbonden met de “productie” database.
* Door punt 1 kan je dus in productie geen testen runnen die impact hebben op de data die de eindgebruiker ziet.
* Door punt 1 en 2 sluit je daarmee veel POST, PUT, DELETE requests uit in de Spotitube app.

///\*\* voor en nadelen zijn oordelen, kunnen weg uit resultaten van experiment.

# Discussie

In dit hoofdstuk beantwoord ik de deelvragen die in de [onderzoeksmethode](#_Onderzoeksmethode) zijn opgesteld.

## Beantwoorden van deelvragen

#### Is een integratie test sneller dan handmatig testen?

Het opzetten van de testmethode is voor het integratie testen veruit het snelst, met ruim 10 minuten verschil in opzet tijd ben je met NodeJS sneller uit dan Postman en met Java heb je nog minder opzet tijd: namelijk het aanmaken van het test bestand, omdat je alles al ingebouwd hebt in Java en je Spotitube broncode.

Als ik kijk naar de resultaten van de [snelheden](#_Snelheid_end-points), zie ik dat de script tests sneller zijn dan de handmatige tests via de browser of Postman. In een enkel geval is de browser sneller dan NodeJS, maar dan is Java weer sneller.

Je beste keus voor snelheid is dus met integratie testen via script.

#### Welke test manier heeft de meest geschikte documentatie? Handmatig met Postman of integratie testen met NodeJS?

Wanneer je op de SuperTest pagina van npmjs.com terecht komt, ben je direct op de enige pagina en daarmee kan je snel door scrollen naar de voorbeelden die je direct kan toepassen in je eigen code, wat het heel makkelijk maakt om een integratie test te maken met NodeJS.  
Voor Postman zijn er geen examples te zien als je voor het eerst op de pagina terecht komt.  
Maar de documentatie van Postman is wel een stuk uitgebreider, dus als je een specifiek probleem hebt is dit wel de beste documentatie.

Maar voor Spotitube voldoet SuperTest meer dan voldoende met de voorbeelden en methode uitleg in de documentatie, waar Postman een beetje overweldigend kan overkomen als je niet weet waar je in terecht komt.

#### Wat is leesbaarder? Het testen via Postman of het schrijven van integratie tests met NodeJS?

Het testen via NodeJS is heel makkelijk op te zetten, zeker met de vooraf gegeven voorbeelden van SuperTest. Met maximaal 24 regels voor een simpel test bestand met SuperTest en Jest, is een NodeJS test in 1 oogopslag te zien en vind ik zelf dat je de naamgeving van de code goed kan lezen.  
Zo is het geen magie wat er gebeurd, maar weet je precies wat er met het verzoek meegestuurd word, welke methode er gebruikt word en naar welke URL het verzoek gestuurd word.

Wil je nog minder code? Dan is Postman een goede oplossing, Postman heeft de ruimte voor het aanpassen van je verzoek, en kan zelfs het verzoek omzetten naar broncode in een hele boel talen en manieren die deze talen ondersteunen, zoals JavaScript fetch en JavaScript Axios(een NPM module).

Wat Postman niet voor je doet is het testen, je zal dus zelf je resultaten uit moeten lezen en checken of dit is wat je verwacht met de gegeven parameters.

Om te testen is NodeJS met SuperTest en Jest dus de oplossing, eenvoudig een test opzetten en daarvan de resultaten checken.

#### Waarmee ben je in een wat langer project beter af? Handmatig of met een script testen?

Omdat je in een wat langer project, van vaak ruim 8 weken, veel code aan het schrijven bent, wil je dat je kwaliteit en functionaliteit naar verwachting blijft zodat je geen ongemakkelijke bugs krijgt op een stuk code waar niemand de laatste 2 weken aan heeft gezeten maar niemand heeft getest.

Met NodeJS kan je eenvouding een CI pipeline opzetten, waar dit niet kan met Postman.  
Dit geeft dus de mogelijkheid om je code eenvoudig en volledig te testen.

Om de deelvraag te beanwoorden zijn scripts altijd beter voor de reden bovenstaand genoemd.  
Zo denk ik ook dat je veel betere kwaliteit oplevert als je klaar bent met het project.

Is dit voor de Spotitube applicatie relevant? Ja, want vooraf krijg je een specificatie waaraan je moet voldoen en dan kan je eenvoudig met TDD eerst je tests schrijven en dan op een commit naar je GIT repository eenvoudig zien of je de specificatie hebt behaald.

#### Hoe verhoud NodeJS zich tussen de applicatie taal: Java?

In deze deelvraag beantwoorden we de vraag of NodeJS het wint van Java als integratie test methode.

Daarvoor ga ik elke deelvraag in het kort beantwoorden met NodeJS versus Java.  
Ik heb dit niet meegenomen in de deelvragen zelf, om zo onderscheid te maken tussen handmatig testen en script testen.  
Waar dit verschil tussen 2 script tests methodes is.

* *Is Java testen sneller dan NodeJS testen?*

Volgens de snelheden in de resultaten, is Java sneller dan NodeJS.  
Echter was ik door gebrek aan duidelijke documentatie over integratie testen in Java met JUnit en HttpUrlConnection, langer bezig met het opzetten van 1 test dan NodeJS.  
Voor mij NodeJS de winnaar, omdat het nog steeds dichtbij de snelheden zit van Java, maar veel sneller op te zetten is.

* *Welke test manier heeft de meest geschikte documentatie? Java of NodeJS?*

Zoals ik hierboven ook gezegd heb, is er voor integratie testen in Java niet heel veel documentatie, ik heb uiteindelijk een ver artikel gevonden van bealdung.com(terug te vinden onder: java-http-request in de bronnen).  
Waar NodeJS SuperTest speciaal is voor het integratie testen en met veel voorbeelden, dus wederom is NodeJS mijn winnaar, vanwege de voorbeelden en vindbaarheid.

* *Wat is leesbaarder? Het testen via Java of het schrijven van integratie tests met NodeJS?*

Het schrijven van tests in NodeJS koste mij 24 regels, waar Java daar met meer dan de helft over heen ging en ik al snel niet meer wist waar ik wat moest doen vanwege de verschillende classes die nodig waren om 1 http verzoek te doen en uit te lezen.

Wederom vind ik NodeJS hier de betere methode.

* *Waarmee ben je in een wat langer project beter af? Java of NodeJS?*

Met NodeJS kan je eenvoudig een pipeline opzetten via bijvoorbeeld github actions, echter kan dit ook met Java en maven.  
Waar je met Java in het geval van Spotitube in dezelfde omgeving werkt.

Voor diversiteit is het interessanter om NodeJS te gebruiken binnen je project, maar voor het hebben van 1 codebase met dezelfde taal, is het gebruik van Java voor je integratie tests aan te raden. Daarom kies ik, in het geval van de Spotitube applicatie, Java.

NodeJS is totaal anders dan Java qua opzet werk en qua testen, maar in principe test je de uitkomst van je Spotitube applicatie met beide applicaties op dezelfde manier: je stuurt een http verzoek met parameters, je krijgt antwoord, en je checkt of dit antwoord klopt.

Met Spotitube werk je al in Java en is het dus eenvoudiger om met Java je integratie tests te schrijven, eventueel op een andere manier dan ik heb geprobeerd bij het experiment.  
<https://www.baeldung.com/integration-testing-a-rest-api> Heb ik later gevonden maar niet verder over onderzocht, in eerste opzicht ziet het er gelijkwaardig uit aan de NodeJS variant, maar dan op de ‘Java manier’ met veel functie aanroepen en classes.

Mijn persoonlijke voorkeur gaat uit naar NodeJS vanwege de afwisseling tussen talen en het integratie testen van de applicatie is dan buiten de broncode van de applicatie waarmee je in mijn ogen beter kan testen of je de juiste responses krijgt.

## Oordeel deelvragen

// \*\* TODO

------------------------------------

#### Hoe snel zet je een integratie test op met node.js?

Node.js en NodePackageManager zijn vereist.  
Een integratie test met node.js zet je op met npm.

* Door met npm de packages: Jest en Supertest te installeren met *npm install*.
* Wanneer deze zijn geinstalleerd, maak je een `\_\_tests\_\_` folder aan binnen het project, daar komen alle tests binnen, en jest zal deze testen.
* Vervolgens zet je een test commando in de package.json, dit commando zal gebruikt worden om alle test-suites(gelijk aan de test folder binnen java) in de \_\_tests\_\_ folder te runnen.
* Zet nu eerst aan het begin van je bestand de regels code die de supertest package importeert met de require import methode.
* Zet vervolgens de locatie van de API binnen de supertest(‘API\_URL komt hier’).  
  Stel een constante gelijk aan deze functie, zodat je deze bij de tests kan gebruiken.

Dat gedaan hebbende, kan je beginnen met het schrijven van de 1e test.

* Dat doe je door een ‘describe’ te maken met een naam als string en een functie callback: de daadwerkelijke test.
* Binnen de functie callback van deze describe zet je een ‘it’ met dezelfde eigenschappen, in ons geval maak je de callback een asynchrone functie zodat async/await gebruikt kan worden.
* Roep nu een http methode aan op de zojuist aangemaakte constante van de **supertest(‘API\_URL’)**, en zet **await** hiervoor zodat je wacht totdat er een response terug is.  
  vervolgens kan je allerlei soorten testen halen over de response.

Zie voorbeeld van experiment met supertest [hier](#_Leesbaarheid_&_code).

Om de deelvraag te beantwoorden: een gemiddeld ontwikkelaar kan deze stappen binnen **10minuten** uitvoeren(uitgaande van stabiele internet connectie en kennis van javascript).

// \*\* beantwoord deelvragen in discussie, en vel een oordeel over de deelvragen.  
Stappen plan is geen antwoord maar resultaat.  
In conclusie kan je ingaan op eindoordeel hoofdvraag met korte samenvatting van onderzoek.

#### Hoe snel zet je een handmatige test klaar om een endpoint van spotitube te testen?

Postman of soortgelijke tool is vereist.

* Wanneer je met postman een endpoint wil testen, hoef je alleen maar de API\_URL in te voeren in de duidelijk zichtbare balk.
* Na dat ingevoerd te hebben, voer je eventuele query of path parameters in, en mogelijk geef je een body mee in de vorm van JSON.
* Daarna klik je op de verzend knop, en snel zie je een resultaat van de API.
* Dit resultaat is gemakkelijk te lezen, waarna je zelf kan checken of de waardes kloppen met wat je ingevoerd had.

Om antwoord te geven op de deelvraag: gemiddeld download en installeer je met een stabiele internet connectie, postman in ongeveer 2minuten, waarna nog eens 1minuut voor het typen van de juiste waardes, wat kan uitlopen tot 2minuten maximaal.  
om op een totaal van **+-5minuten** uit te komen.

// \*\* ook weer resultaten, laatste deel is wel oordeel.

#### Hoe snel zijn integratie tests versus handmatig testen?

[Volledige resultaten lijst](#_Snelheid_end-points)

Hier onder samengevat de snelste methode, de responsetijd van postman versus integratie tests.

Deze tests bevatten geen menselijke input, en zijn daarom niet erg relevant, totdat we gaan kijken naar wat de tijd winst is over tijd, op bijvoorbeeld projectbasis.

|  |  |  |
| --- | --- | --- |
|  | Integratie | Handmatig |
| /login correct credentials |  |  |
| /Login wrong credentials |  |  |
| /Playlist correct token |  |  |
| /Playlist wrong token |  |  |

Hier aan zien we dat de integratie test via node.js **75%** van de tests sneller is.

In de resultaten zien we ook het gemiddelde van de tests, waarbij het oploopt van **30 tot 60ms** per endpoint test.

Als je meerdere endpoints in 1 keer wil testen, scheelt dat gemiddeld **40ms** per keer, en bij 10 endpoints is dat al **bijna een halve seconde**.

Run je de integratie tests vaker op een dag, dan het verschil oplopen **tot een paar secondes per dag**, en vergeet niet dat de menselijke factor ook mee telt, als je bijvoorbeeld postman per ongeluk afsluit, ben je zo een paar minuten verder op een dag, terwijl de integratie tests, bij wijze van spreken, in je IDE leven.

In een project van 8 weken is die duur dus veel langer, dan ben je gemiddeld per week een paar minuten sneller, en in 8 weken kan dat oplopen tot een kwartier tot 30minuten.

// \*\* uitleggen waarom relevant? 15minuten op 8 weken is niet zo veel.

Om de deelvraag te beantwoorden: het testen van endpoints met een integratie test versus handmatig, scheelt op dag basis **gemiddeld tot een paar minuten**, meegerekend de menselijke factor van het per ongeluk afsluiten van een applicatie.

#### Integratie testen in Java?

Integratie testen met Java zou in het geval van Spotitube in productie de beste oplossing zijn.

**Waarom?** Bekijk de onderstaande tabel met voor en nadelen in kleur aangegeven van het integratie testen met node.js op de Spotitube applicatie.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Handmatig | Node.js | Java |
| Projectbasis/productie |  | Geen Test DB | Test DB |
| Development |  | geen impact productie DB. | Zelfde omgeving |
| Snelheid |  |  | - |
| Opzet |  |  | - |

(- = niet getest)

Rood = niet geschikt vanwege in resultaten/deelvragen genoemde redenen

Oranje = mogelijk, maar zeker niet de beste in de categorie.

Geel = opzet mogelijk niet nodig/opzet afhankelijk van kennis.

Groen = goed voor de categorie.

Wanneer je niet met een productie omgeving werkt, en je database dus alleen voor ontwikkelaars zichtbaar is, dan is Node.js een goede optie.

Node.js maakt gebruik van een de chrome V8 runtime environment en is daarmee snel voor lokaal gebruik.

// \*\* veel van deze antwoorden kunnen beter in de resultaten.

Uitbreiden met resultaten van JAVA.

## Beantwoorden hoofdvraag

De winst van het integratie testen van de Spotitube applicatie met Node.js is aanwezig op gebied van persoonlijke voorkeur van taal/methode(Java vs Javascript vs handmatig), op gebied van snelheid is node.js sneller dan het handmatig testen van endpoints binnen de spotitube applicatie.

Op gebied van project/productie gericht testen, is Node.js geen goede optie voor de Spotitube applicatie door het feit dat je dan test data in je productie data krijgt wat in het algemeen een slechte gewoonte is binnen ontwikkelaars land.

**Is Integratie testen met Node beter dan handmatig voor Spotitube?** Ja.

**Is Integratie testen met Node beter dan met Java voor Spotitube?** Ja.

#### De Winst van het integratie testen van de Spotitube applicatie met Node.js:

* Je testen zijn **sneller** **dan** als je **handmatig** endpoints test.
* Je testen zijn **makkelijk** op te zetten.
* Je testen zijn met javascript(**persoonlijke voorkeur**) tegenover Java.
* Je test met Node.js van buitenaf de Spotitube applicatie waardoor je de **eindgebruiker** **beter** **kan** **nabootsen**.

// \*\* herschijven naar oordeel deelvragen ^

# Conclusie

Wanneer je werkt met een API die niet wijzigingen doorbrengt binnen een database, dan kan je prima de Node.js integratie tests gebruiken voor een Java applicatie.

Maar omdat we praten over de Spotitube API, en die maakt wel gebruik van database wijzigingen, wil je eigenlijk een test database maken om niet je productie data aan te passen met vervuilde test-data.

Dat kan je niet doen van buiten af de Spotitube applicatie.

Vanuit binnen in kan dit wel door een nieuwe database te maken, en daar in een @BeforeEach mee te verbinden(kan overigens ook bij supertest en express.js, maar dan moeten die wel lokaal samen draaien).

Daarmee komt het uiteindelijk voor de Spotitube applicatie neer op de context van de ontwikkelaar, **is Spotitube in productie?** Dan kies je **Java**, **is Spotitube in ontwikkeling en niet in productie?** Dan ligt de **keus bij de ontwikkelaar** voor welke taal hij/zij wil kiezen en of die graag de integratie tests in hetzelfde project heeft als de applicatie code.

Wat je ook kiest: Integratie testen schrijven is beter dan handmatig testen.

// \*\* schrijf een betere conclusie: geef geen keuze maar maak een keuze.

# Verwijzingen

borg, b. (2022, maart 31). *is-integration-testing-really-necessary*. Opgehaald van onpathtesting: https://www.onpathtesting.com/blog/is-integration-testing-really-necessary

dzone. (2022, maart 30). *integration-testing-what.....* Opgehaald van dzon: https://dzone.com/articles/integration-testing-what-it-is-and-how-to-do-it-ri

geeksforgeeks. (2022, maart 30). *software-engineering-integration-testing*. Opgehaald van geeksforgeeks: https://www.geeksforgeeks.org/software-engineering-integration-testing/

Honig, R. (2022, maart 31). *6-best-practises-integration-testing*. Opgehaald van techbeacon.com: https://techbeacon.com/app-dev-testing/6-best-practices-integration-testing-continuous-integration

jest. (2022, maart 31). *docs/*. Opgehaald van jestjs.io: https://jestjs.io/docs/getting-started

npm. (2022, maart 31). *package/supertest*. Opgehaald van npmjs: https://www.npmjs.com/package/supertest

Paraschiv, E. (2022, maart 31). *integration-testing-a-rest-api*. Opgehaald van bealdung.com: https://www.baeldung.com/integration-testing-a-rest-api

techtarget. (2022, maart 31). *integration-testing*. Opgehaald van techtarget.com: https://www.techtarget.com/searchsoftwarequality/definition/integration-testing

[**https://www.npmjs.com/package/supertest**](https://www.npmjs.com/package/supertest) **\*\* superTest** 2022, 24, 05.

<https://learning.postman.com/docs/getting-started/introduction/> Postman 2022, 25, 05.

<https://www.jrebel.com/blog/how-to-use-java-integration-testing> jrebel blog 2022 , 25, 05

<https://www.baeldung.com/java-http-request> baeldung.com 2022, 25, 05.

<https://www.javatpoint.com/java-json-example> javatpoint.com java-json example 2022, 25, 05