Spotitube Integratie Testen

Integratie testen voor de Spotitube applicatie.

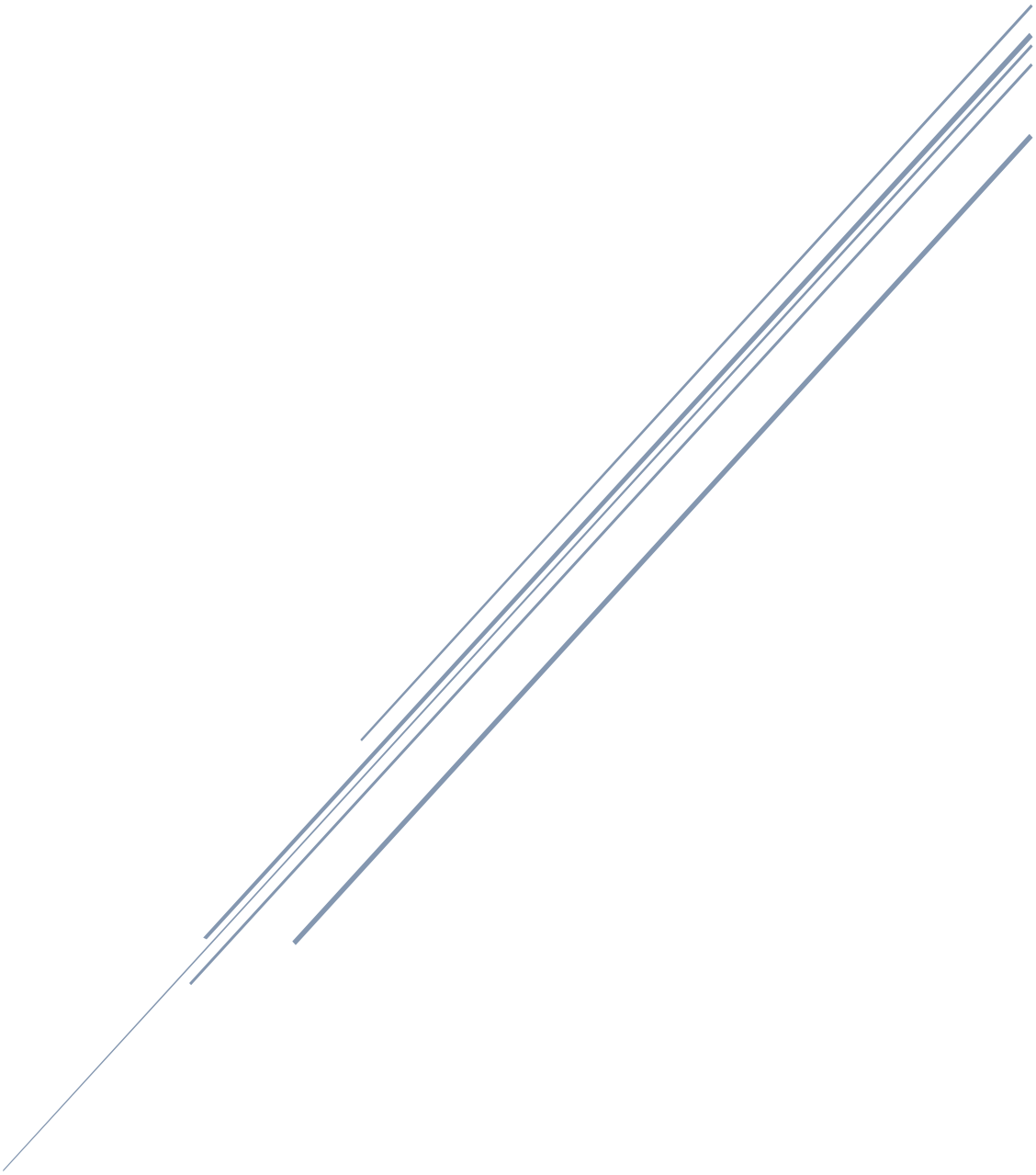
Maarten van der Lei (658215)

Klas: ITA-OOSE-A-f

Datum: 27-mei-2022.

Course: DEA herkansing.

Opdrachtgever: Wiebe Rinsma.



HAN-OOSE
DEA-Spotitube

Inhoud

1. Inleiding

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(end-to-end) 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 vanwege 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 wordt alleen gekeken naar SuperTest in combinatie met Jest omdat SuperTest volgens de NPM(NodePackageManager) de populairste is..
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)?*"

2. Onderzoeksmethode

Om de hoofdvraag, die ik in de inleiding heb opgesteld, 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 verhoudt 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

3. Resultaten

Getest met clean browser: geen cookies, getest met target folder gereset om zo eventuele caching tegen te gaan en een beter testresultaat te krijgen.
In Postman krijg je net zoals bij je browser in het 'network' tabje(in de developer tools van de browser) een request te zien die word gedaan, daarbij komt de request-response 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.

3.1.Opzetten

Hier onder lees je hoe je de verschillende omgevingen opzet om een integratie test te maken.

3.1.1. SuperTest

Het opzetten van SuperTest is snel gedaan mits je NodeJS en NPM op je machine hebt staan.
Waarna je het volgende commando runt in je gekozen folder: 'npm install supertest jest –save-dev'.
Afhankelijk van je internet snelheid is dit meestal binnen **20 seconden** klaar.
Nadat de installatie is afgerond, kan er binnen de map '__tests__' een bestand aangemaakt worden die eindigt op '.test.js'.
Om je tests te runnen maak je in je 'package.json' een script aan die 'test' heet, daar in zet je het volgende: 'jest'.
Dit zit er zo uit: '"test": "jest"'.
Jest gaat dan op zoek naar de '__tests__' folder en alle bestanden eindigend op '.test.js' worden dan uitgevoerd.

3.1.2. 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.

3.1.3. Postman

Om Postman te gebruiken ga je naar de Postman hoofdpagina waarna je direct te zien krijgt of je de desktop app wil downloaden of een account aanmaken om de web versie te gebruiken.
Afhankelijk van je download snelheid, is de Postman applicatie met circa 150MB en een gemiddelde download snelheid van 25MBits/s, binnen **5 minuten** gedownload en geïnstalleerd.
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.

3.2. Snelheid endpoints

Hier onder lees je de request tijden van het experiment uitgelegd in de onderzoeksmethode.
Deze resultaten zijn uitgelezen in de output van de test, opgeslagen in een array, of de response tijden uit de browser, uitgelezen in de developer tools.
Bij Postman is het uitgelezen in de responsetijd in Postman na het doen van een request.

/login wrong creds	Postman	De 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	De 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	De 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.

3.3. Leesbaarheid

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

Voor de code heb ik 1 hele test en opzet gepakt uit mijn gemaakte experiment.

3.3.1. 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.

3.3.2. Java HttpUrlConnection & JUnit

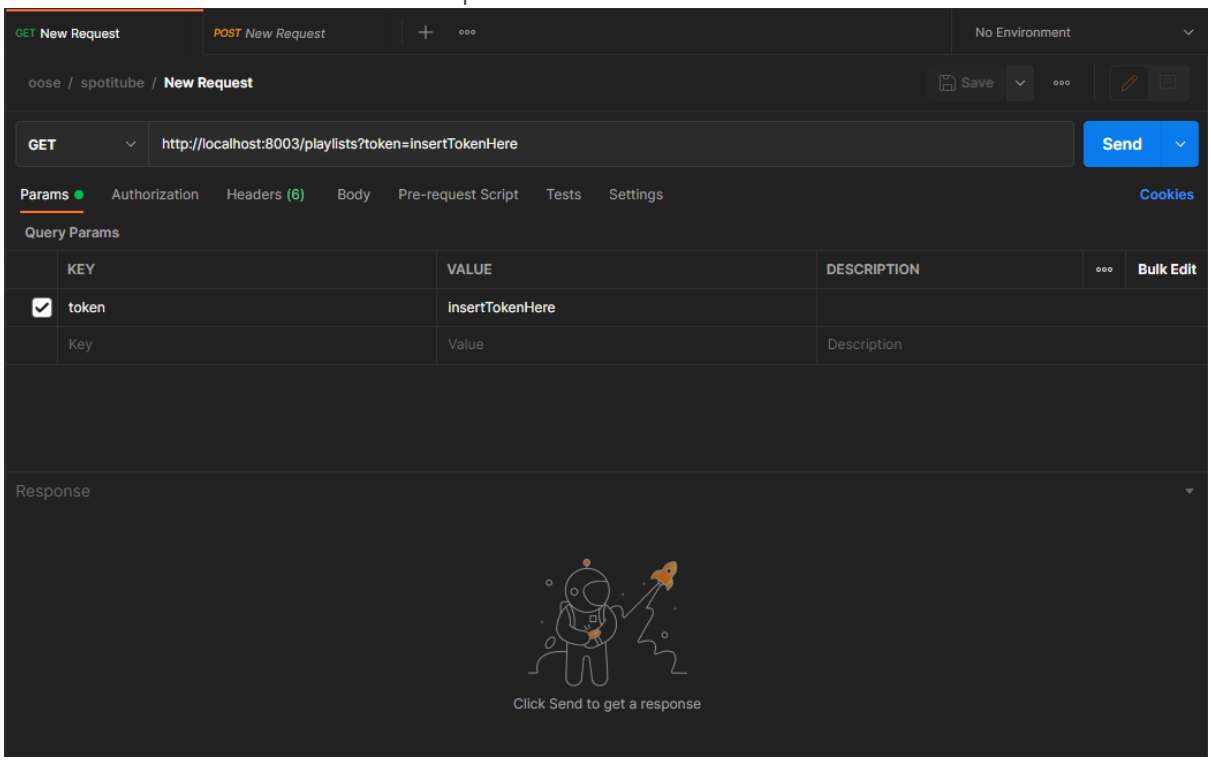
```
1.  package nl.han.oose.vdlei.spotitube.research;
2.  import org.json.JSONObject;
3.  import org.junit.jupiter.api.Test;
4.  import java.io.BufferedInputStream;
5.  import java.io.ByteArrayOutputStream;
6.  import java.io.OutputStream;
7.  import java.io.OutputStreamWriter;
8.  import java.net.HttpURLConnection;
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.

3.3.3. Postman requests



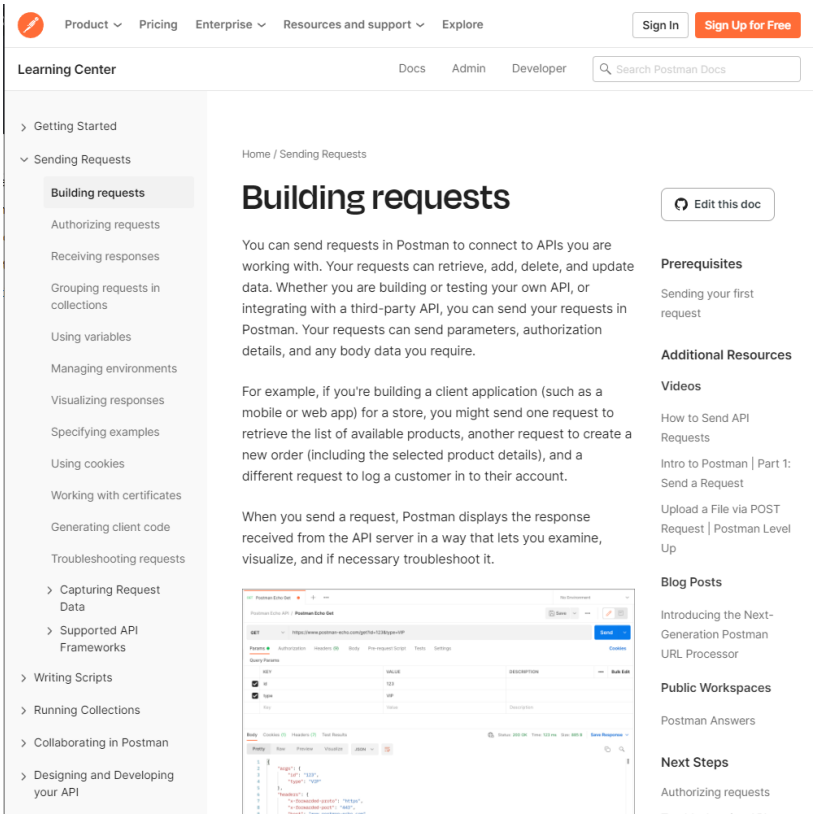
Postman testmethode uit eigen postman omgeving.

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.

3.4. Documentatie

Hier onder lees je de verschillende documentatie per test methode.

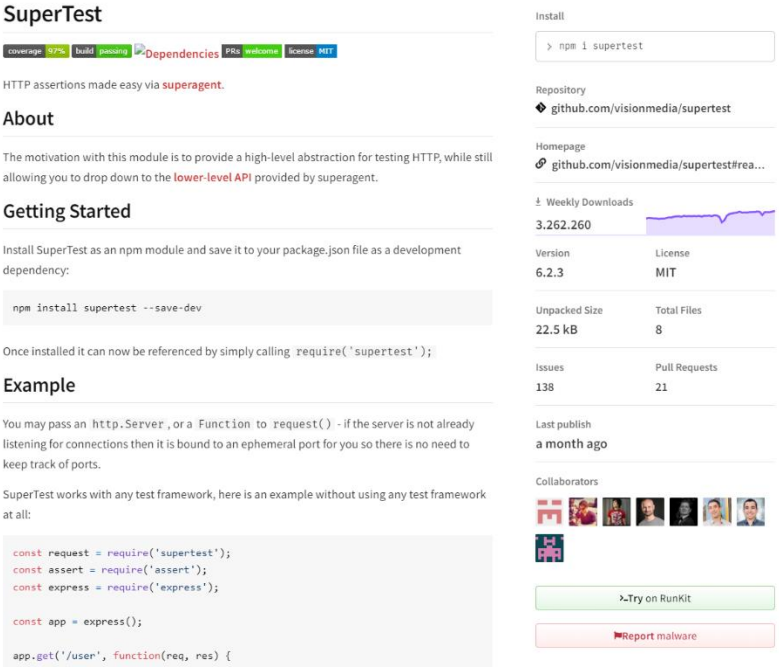
3.4.1. Postman



Postman documentatie van de https://learning.postman.com/docs/sending-requests/requests/ site.

Postman heeft veel documentatie over hoe je een verzoek kan versturen 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.

3.4.2. SuperTest & Jest



SuperTest

coverage 97% | build passing | Dependencies | PRs welcome | license MIT

HTTP assertions made easy via superagent.

About

The motivation with this module is to provide a high-level abstraction for testing HTTP, while still allowing you to drop down to the lower-level API provided by superagent.

Getting Started

Install SuperTest as an npm module and save it to your package.json file as a development dependency:

```
npm install supertest --save-dev
```

Once installed it can now be referenced by simply calling `require('supertest');`

Example

You may pass an `http.Server`, or a `Function` to `request()` - if the server is not already listening for connections then it is bound to an ephemeral port for you so there is no need to keep track of ports.

SuperTest works with any test framework, here is an example without using any test framework at all:

```
const request = require('supertest');
const assert = require('assert');
const express = require('express');

const app = express();

app.get('/user', function(req, res) {
```

Install

```
> npm i supertest
```

Repository
github.com/visionmedia/supertest

Homepage
github.com/visionmedia/supertest#rea...

Weekly Downloads
3.262.260

Version	License
6.2.3	MIT

Unpacked Size	Total Files
22.5 kB	8

Issues	Pull Requests
138	21

Last publish
a month ago

Collaborators

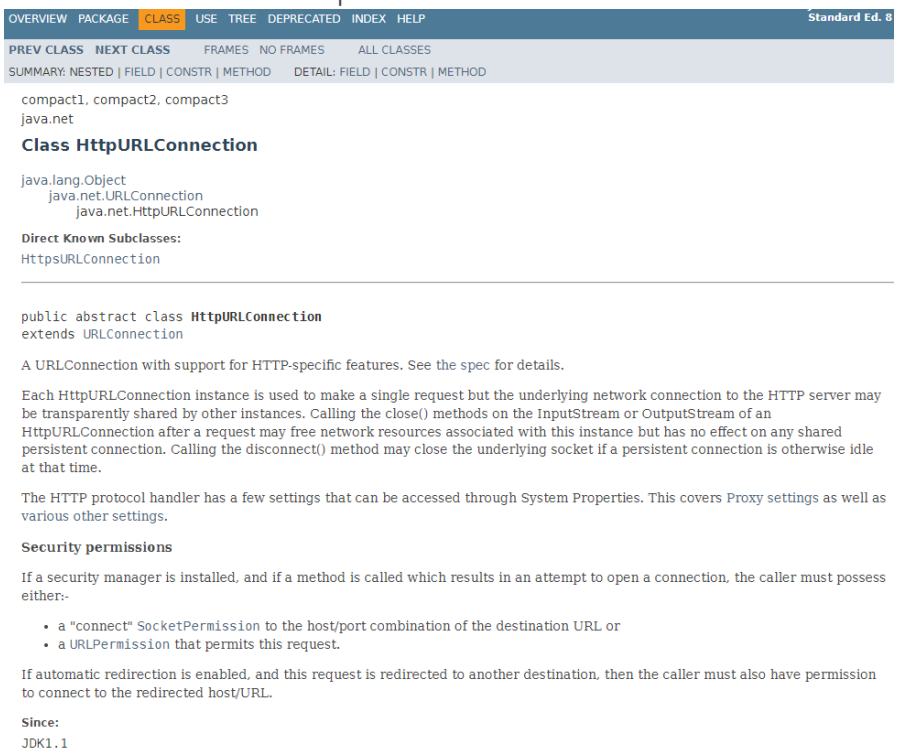
Try on RunKit

Report malware

SuperTest documentatie van de 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.

3.4.3. HttpUrlConnection & JUnit



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.net

Class HttpURLConnection

java.lang.Object
  java.net.URLConnection
    java.net.HttpURLConnection

Direct Known Subclasses:
HttpsURLConnection

```
public abstract class HttpURLConnection
extends URLConnection
```

A URLConnection with support for HTTP-specific features. See the spec for details.

Each HttpURLConnection instance is used to make a single request but the underlying network connection to the HTTP server may be transparently shared by other instances. Calling the close() methods on the InputStream or OutputStream of an HttpURLConnection after a request may free network resources associated with this instance but has no effect on any shared persistent connection. Calling the disconnect() method may close the underlying socket if a persistent connection is otherwise idle at that time.

The HTTP protocol handler has a few settings that can be accessed through System Properties. This covers Proxy settings as well as various other settings.

Security permissions

If a security manager is installed, and if a method is called which results in an attempt to open a connection, the caller must possess either:-

- a "connect" SocketPermission to the host/port combination of the destination URL or
- a URLPermission that permits this request.

If automatic redirection is enabled, and this request is redirected to another destination, then the caller must also have permission to connect to the redirected host/URL.

Since:
JDK1.1

Documentatie van de HttpUrlConnection van de 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.

3.5. 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.

3.5.1. 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)

3.5.2. 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.

3.5.3. Wat is Node.js?

Node.js is een javascript runtime environment gebouwd op chrome V8

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 informatie: https://nodejs.org/en/about/ .

4. Discussie

In dit hoofdstuk beantwoord ik de deelvragen die in de onderzoeksmethode zijn opgesteld.

4.1. 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 5 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, 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 wordt, 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 heleboel 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 werken zodat je geen ongemakkelijke bugs krijgt op een stuk code waar niemand de laatste 2 weken aan heeft gewerkt.

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 verhoudt NodeJS zich tussen de applicatie taal: Java?

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

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 overheen 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.

4.2. Oordeel deelvragen

In eerste opzicht komt NodeJS bij mij naar boven als de beste manier om te integratie testen, maar na het maken van de integratie testen met Java vond ik een artikel over een andere manier waarop Java http verzoeken geschreven kunnen worden, en dat gaf Java meer punten op gebied van leesbaarheid en opzetten van een test.
Na het beantwoorden van de deelvragen en het meenemen van wat ik na het experiment heb geleerd, maken zowel NodeJS als Java goeie kansen bij mij, beide tests zijn sneller dan handmatig testen, waar Java vaak sneller is dan NodeJS.

Ik blijf van mening dat NodeJS beter leesbaarder is, vooral door de opzet van SuperTest waarbij je een assert kan toevoegen om de status te checken en in een '.then' je de rest van het response kan checken met ook weer asserts.
Voor documentatie is NodeJS ook de winnaar van handmatig testen en Java, de Java documentatie is voor mij wat te oud en onoverzichtelijk, je moet vaak de hele tekst doorlezen om kernwoorden te vinden die net een andere tint hebben, om daarop door te klikken voor meer informatie. Daarnaast staan er geen voorbeelden bij en dat maakt het een stuk lastiger om te gebruiken als je niet veel met Java werkt.

Omdat vooraf de Spotitube opdracht al een specificatie klaarlag, kan je daarvoor al testen maken en TDD te werk gaan, met NodeJS is dit makkelijk opgezet en leesbaarder dan Java code.

Ik ben daarom van mening dat de winst van NodeJS als integratie testmethode wel aanwezig is bij gebruik binnen de Spotitube applicatie.

5. Conclusie

Tijdens dit onderzoek heb je kunnen lezen hoe ik een experiment heb uitgevoerd om te kunnen achterhalen wat de winst is van het integratie testen met NodeJS op de Spotitube applicatie.

Ik heb hierbij code geschreven, documentatie doorgelezen en de tests uitgevoerd en vergeleken.
NodeJS heeft alles om volledig je Spotitube applicatie te testen met integratie tests.
Al waren Java en NodeJS in veel gevallen gelijkwaardig, is NodeJS eenvoudiger op te zetten, zijn de voorbeelden uit de documentatie makkelijk te gebruiken in je eigen code en je code ziet er een stuk netter uit met minder regels en goed te lezen structuur met dank aan SuperTest en Jest.
Handmatig testen valt bijna niet meer te vergelijken met NodeJS als we eenmaal de NodeJS met de Java vergelijken, hierom valt handmatig testen via Postman of de browser helemaal af. Je bent hoe dan ook beter af met script testen.
Handmatig testen valt ook af wanneer je TDD werkt.

De winst van NodeJS is dus dat je handigere documentatie hebt, eenvoudiger je test opzet en makkelijk je eigen tests kan lezen en uitvoeren binnen een CI-omgeving.

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 **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