|  |
| --- |
|  |
| William Wong |
| Bachelor Toegepaste Informatica  Afstudeerrichting? |
|  |
|  |
| Software Testing  Frank Serneels |
|  |
| Academiejaar 2021-2022 |

|  |  |
| --- | --- |
| Udemy cursus week 3: toepassing leerstof | |
|  |

# Section 3:

## Response code check:

Een request bestaat meestal uit meerdere testen. Een eerste test die we beter altijd eerst expliciet uitvoeren, is nagaan of we een response code 200 krijgen. Dit is de eerste voorwaarde die voldaan moet worden voordat we gegevens kunnen opvragen of meegeven aan een request. We zullen een test functie uitvoeren die aangeeft of de response code 200 is op een request van de exchange API fixer.io.

Link API: https://data.fixer.io/api/latest?access\_key=13dd7467e4cbb0eeda4e6b607c0af972

We schrijven dus een functie in de tests tabblad:

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

Graphical user interface, application, Word

Description automatically generated

We krijgen als response code 200, dat wilt zeggen dat de request is gelukt:

Graphical user interface, application

Description automatically generated

## API testen:

We zullen nu een API uitproberen van Trello. We zullen proberen om via Postman een request uit te voeren met als doel een nieuwe board te creëren in Trello. We hebben hiervoor een API key en een token nodig. De token dient ervoor om de board op mijn Trello account aan te maken, dus de token zorgt voor de toegang tot je persoonlijke account in Trello. Via de token kan de request dus toegang hebben tot jouw account om daar een nieuwe board te maken.

We zullen een POST request uitvoeren met drie parameters: een token, een API key en een naam voor het board:

Graphical user interface, text, application, email

Description automatically generated

De response code is 200, dus we moeten normaal een nieuwe board hebben op onze account:

A picture containing graphical user interface

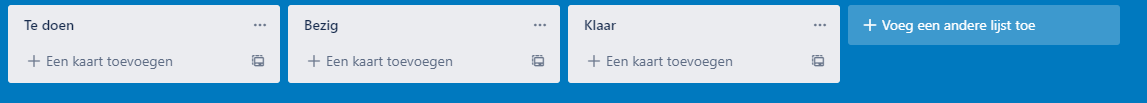
Description automatically generated

Onze board met de naam TestBoard zit ook op mijn persoonlijke account:

Graphical user interface, website

Description automatically generated

Deze board bevat standaard een aantal aantekeningen.



Via de parameters kun je ook aangeven om enkel een leeg bord te creëren. Dit kan door een extra parameter defaultLists met de value false mee te geven:

Graphical user interface, text, application, email

Description automatically generated

Resultaat: een leeg board

Graphical user interface, website

Description automatically generated

A picture containing treemap chart

Description automatically generated

We kunnen ook testen of de naam van het nieuwe board gelijk is aan de value die we meegaven aan de naam parameter. Hiervoor schrijven we de volgende test:

pm.**test**("Status code is 200", **function** () {

pm.**test**("Board is created", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData.name).to.eql("NewTestBoard");

});

    pm.response.to.have.status(200);

});

We zetten de response om in een json object, zodat het doorgegeven kan worden aan een javascriptobject om hiermee verder te werken.

Graphical user interface, text, application, email

Description automatically generated

Resultaat:

A picture containing background pattern

Description automatically generated

Deze test lukt ook, omdat de nieuwe board dezelfde naam heeft als de value die we meegaven aan de name parameter:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

We kunnen meerdere values controleren. We kunnen ook nagaan of de board private is:

pm.**test**("Status code is 200", **function** () {

pm.**test**("Board is created", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData.name).to.eql("NewTestBoard");

});

pm.**test**("Board is private", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData.prefs.permissionLevel).to.eql("private");

});

    pm.response.to.have.status(200);

});

Resultaat:

Graphical user interface, text, application

Description automatically generated

We kunnen deze tests opslaan. Deze kunnen we later dan opnieuw uitvoeren:

Graphical user interface, application

Description automatically generated

We kunnen de tests ook herschrijven door variables te gebruiken. Dit is beter georganiseerd:

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

const response **=** pm.response.json();

pm.**test**("Board is created", **function** () {

    pm.expect(response.name).to.eql("NewTestBoard");

});

pm.**test**("Board is private", **function** () {

    pm.expect(response.prefs.permissionLevel).to.eql("private");

});

Resultaat blijft hetzelfde, maar de code is minder lang door het gebruik van een constante:

Graphical user interface, text, application

Description automatically generated

# Section 4: variables

## Gebruik variables:

We kunnen gebruik maken van variabelen in Postman. Dit zorgt ervoor dat we gegevens kunnen doorgeven van een request naar een andere request, bv:

Me maken een nieuwe Trello board via een request en via een andere request zullen we deze nieuwe board opvullen met een lijst. Hiervoor hebben we de boardID nodig van het eerste request:

Eerste request: een board aanmaken.

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

const response **=** pm.response.json();

pm.**test**("Board is created", **function** () {

    pm.expect(response.name).to.eql("NewTestBoard");

});

pm.**test**("Board is private", **function** () {

    pm.expect(response.prefs.permissionLevel).to.eql("private");

});

pm.globals.**set**("boardID", response.id)

Deze laatste regel zal een globale variabele aanmaken en deze opvullen met de id van de response.

Resultaat:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Deze boardID hebben we nodig om een specifieke board op te vullen met een lijst. Hiervoor voeren we de volgende test in:

API url om een list aan te maken in een board: [https://api.trello.com/1/boards/{{boardID}}/lists?name=todoList&key={{trelloKey}}&token={{trelloToken}}](https://api.trello.com/1/boards/%7b%7bboardID%7d%7d/lists?name=todoList&key=%7b%7btrelloKey%7d%7d&token=%7b%7btrelloToken%7d%7d)

Aangezien we de key en token nog vaak nodig hebben, heb ik hier ook een variabele van gemaakt:

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

const response **=** pm.response.json();

pm.**test**("List is created", **function** () {

    pm.expect(response.name).to.eql("todoList");

});

pm.**test**("List is created in board with boardID", **function** () {

    pm.expect(response.idBoard).to.eql(pm.globals.**get**("boardID"));

});

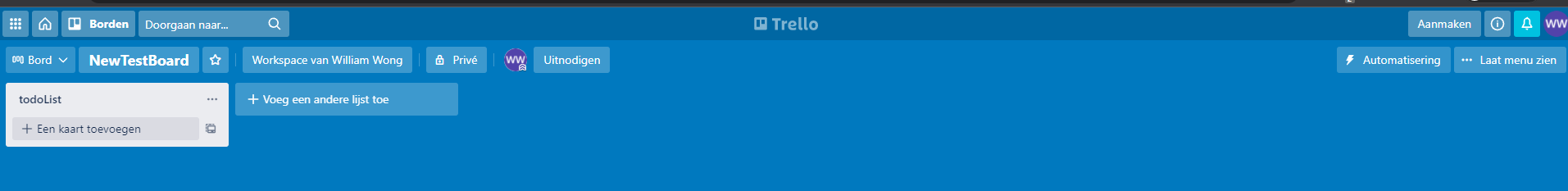
pm.globals.**set**("boardID", response.id)

resultaat:

Graphical user interface, text, application, email

Description automatically generated

De lijst is ook aangemaakt in het aangegeven board:



## Pre-request scripts:

Pre-request scripts zijn handig voor als men bv parameters dynamisch wil doorgeven aan een url. In volgende voorbeeld zal ik een Post request uitvoeren, maar door gebruik te maken van pre-request scripts, zal ik telkens een nieuwe board aanmaken met een andere naam, zonder deze naam telkens door te geven aan de url.

Pre-request script:

let boardName **=** generateRandomBoardName;

pm.globals.**set**("boardName", boardName);

**function** generateRandomBoardName() {

    let boardName **=** "NewTestBoard" **+** **parseInt**(Math.**random**() **\*** 10000);

**return** boardName;

}

Graphical user interface, text, application

Description automatically generated

Opdat deze test lukt, moeten we ook een vorige test aanpassen:

Graphical user interface, text, application, email

Description automatically generated

We moeten de expected value vervangen met de value van de variabele boardname, omdat het een bepaalde waarde verwacht. Deze is namelijk dynamisch dus we moeten de global variabele meegeven opdat deze test altijd zal werken.

Resultaat:

Er wordt een nieuwe board aangemaakt met een willekeurige naam:

Graphical user interface, text, application

Description automatically generated

Note: zowel variables en pre-request scripts zullen ons helpen voor het automatiseren van tests.

# Section 5:

Er was enkel herhalingsoefeningen, die ik niet heb gemaakt. Al de geziene leerstof zal ik namelijk toepassen op een eigen project. Hierdoor heb ik oefeningen van section 5 overgeslagen.

# Section 6:

In Postman kan men gebruik maken van de Chai library. Dit is een assertion library. Hiermee kan ik de response van een request analyseren of nakijken. Zo kan ik bijvoorbeeld nagaan of de response body een bepaalde woord of letter bevat, de response vergelijken met een waarde en meer. Deze assertions zijn scripts die worden uitgevoerd nadat de request is uitgevoerd, in tegenstelling tot de pre-request scripts. Enkele voorbeelden:

pm.**test**("This is correct", **function**(){

    pm.expect(100).to.eql(100);

});

pm.**test**("name includes josh", **function**(){

    pm.expect('josh mcdowel').to.**match**(/**^**josh/);

});

pm.**test**("The array includes 2", **function**(){

    pm.expect(2).to.be.oneOf([1,2,3]);

});

pm.**test**("the object a equals object b", **function**(){

    let a **=** {"name":"joe", "age":"33"};

    let b **=** {"name":"joe", "age":"33"};

    pm.expect(a).to.eql(b);

});

Resultaat:

Graphical user interface, text, application, email

Description automatically generated

We kunnen assertions ook gebruiken voor arrays. De volgende test gaat na of een object in een array al dan niet aan een aantal voorwaarden voldoen.

url: http://www.mocky.io/v2/5ab350d62f00005a00ca3663

Test code:

let jsonData **=** pm.response.json();

let manufacturer **=** jsonData.filters[2];

pm.**test**("manufacturer should not be allowed", **function**(){

    pm.expect(manufacturer.name).to.eql("MANUFACTURER");

    pm.expect(manufacturer.isAllowed).to.be.false;

})

Resultaat:

Graphical user interface, text, application, email

Description automatically generated

Het is echter niet aangeraden om met posities te werken wanneer het op arrays aankomt, omdat sommige responses teveel objecten bevatten of wanneer de posities dynamisch zijn. We moeten dus een andere manier vinden om een bepaald object te vinden in een array. Dit kan door de array te doorlopen en nakijken of elke object een bepaalde key een specifieke value bevat.

Code voor de test:

let jsonData **=** pm.response.json();

let manufacturer;

**for**(let filter **of** jsonData.filters) {

**if**(filter.name **==** "MANUFACTURER") {

        manufacturer **=** filter;

    }

}

pm.**test**("manufacturer should not be allowed", **function**(){

    pm.expect(manufacturer.name).to.eql("MANUFACTURER");

    pm.expect(manufacturer.isAllowed).to.be.false;

})

Resultaat:

Graphical user interface, text, application, email

Description automatically generated

We kunnen ook assertions uitvoeren voor nested responses:

url: http://www.mocky.io/v2/5ab34d8c2f00006600ca3652

Code voor test:

let jsonData **=** pm.response.json();

let commentStatus **=** jsonData.prefs.comments.status;

pm.**test**("Comment should be disabled", **function**(){

    pm.expect(commentStatus).to.eql("disabled");

})

let boardStatus;

**for**(let key **in** jsonData.limits) {

**if**(jsonData.limits[key].hasOwnProperty('boards')) {

        boardStatus **=** jsonData.limits[key].boards.totalPerMember.status;

    }

}

pm.**test**("status must be ok", **function**(){

    pm.expect(boardStatus).to.eql("ok");

})

Resultaat:

Graphical user interface, text, application, email

Description automatically generated