

QA / TESTING - 09/05/2022 - Abel Rios

Vamos a ir probando los test con API. Para ello usamos de ejemplo la página de developer de spotify <https://developer.spotify.com/console/>

CONSOLE		
Albums		
Artists		
Browse		
Episodes		
Follow		
Library		
Markets		
Personalization		
Player		
Playlists		
Get a List of a User's Playlists		
Get a List of Current User's Playlists		
Get a Playlist		
Get a Playlist Cover Image		
Get a Playlist's Items		
Create a Playlist		
Add Items to a Playlist		
Remove Items from a Playlist		
Reorder or replace a Playlist's Items		
Change a Playlist's Details		
Search		
Tracks		

Playlists

METHOD	ENDPOINT	USAGE
DELETE	/v1/playlists/{playlist_id}/tracks	Remove Playlist Items
GET	/v1/me/playlists	Get Current User's Playlists
GET	/v1/playlists/{playlist_id}/images	Get Playlist Cover Image
GET	/v1/playlists/{playlist_id}/tracks	Get Playlist Items
GET	/v1/playlists/{playlist_id}	Get Playlist
GET	/v1/users/{user_id}/playlists	Get User's Playlists
POST	/v1/playlists/{playlist_id}/tracks	Add Items to Playlist
POST	/v1/users/{user_id}/playlists	Create Playlist
PUT	/v1/playlists/{playlist_id}/images	Add Custom Playlist Cover Image
PUT	/v1/playlists/{playlist_id}/tracks	Update Playlist Items
PUT	/v1/playlists/{playlist_id}	Change Playlist Details

Abrimos el postman y creamos una nueva colección (Ejercicio Spotify). Una colección es simplemente una carpeta en la que almacenamos las diferentes requests de un proyecto o una api. Aquí lo vamos a utilizar para guardar todas las requests que haremos en el ejercicio practicando con la API de Spotify.

La primera va a ser de conseguir las playlists. En la página de spotify copiamos el enlace de la segunda request (get playlists). Si enviamos la request falla, pues no hemos enviado el token. En postman, en nuestra request de postman, vamos a "Authorization", elegimos OAuth2.0, nos sale a la derecha una pestaña en la que podemos ingresar el token que hemos adquirido en la página de spotify iniciando nuestra sesión.

Ejercicio Spotify / Get playlist

GET <https://api.spotify.com/v1/me/playlists> Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Type OAuth 2.0

The authorization data will be automatically generated when you send the request. [Learn more about authorization](#)

Add authorization data to Request Hea...

Current Token

This access token is only available to you. Sync the token to let collaborators on this request use it.

Access Token Available Tokens

BQA1SH390yKfc1kGqgp03JUelwq9v ...

Header Prefix

Bearer

body Cookies Headers (18) Test Results

Status: 200 OK Time: 144 ms Size: 29.66 KB Save Response

Pretty Raw Preview Visualize JSON

```
1  "href": "https://api.spotify.com/v1/users/abelxm90/playlists?offset=0&limit=20",
2  "items": [
3    {
4      "collaborative": false,
5      "description": "",
6      "external_urls": {
7        "spotify": "https://open.spotify.com/playlist/4XahA50uzGH1GLS7Aslmai"
8      },
9      "href": "https://api.spotify.com/v1/playlists/4XahA50uzGH1GLS7Aslmai",
10     "id": "4XahA50uzGH1GLS7Aslmai",
11     "images": [
12
```

Ahora vamos a hacer otra request, para conseguir las canciones de una lista:

Ejercicio Spotify / Get songs given a playlist id

GET → https://api.spotify.com/v1/playlists/{playlist_id}/tracks

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Type OAuth 2.0

The authorization data will be automatically generated when you send the request.
[Learn more about authorization](#)

Add authorization data to Request Hea...

Current Token

This access token is only available to you. Sync the token to let collaborators on this request use it.

Access Token

Available Tokens

BQA1SH390yKfc1kGqppo3JUelwq9v ...

Header Prefix

Bearer

Body Cookies Headers (16) Test Results

Status: 404 Not Found Time: 57 ms Size: 850 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": {
3     "status": 404,
4     "message": "Invalid playlist Id"
5   }
6 }
```

Nos vuelve a fallar pues necesita la playlist ID, de la request anterior podemos sacar la ID de cualquier lista, la insertamos en la nueva request y magia, funciona:

Ejercicio Spotify / Get songs given a playlist id

GET → https://api.spotify.com/v1/playlists/4XahA5OuzGHlGLS7Aslmai/tracks

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Type OAuth 2.0

The authorization data will be automatically generated when you send the request.
[Learn more about authorization](#)

Add authorization data to Request Hea...

Current Token

This access token is only available to you. Sync the token to let collaborators on this request use it.

Access Token

Available Tokens

BQA1SH390yKfc1kGqppo3JUelwq9v ...

Header Prefix

Bearer

Body Cookies Headers (17) Test Results

Status: 200 OK Time: 92 ms Size: 67.46 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "href": "https://api.spotify.com/v1/playlists/4XahA5OuzGHlGLS7Aslmai/tracks?offset=0&limit=100",
3   "items": [
4     {
5       "added_at": "2022-05-09T06:15:31Z",
6       "added_by": {
7         "external_urls": {
8           "spotify": "https://open.spotify.com/user/abelm90"
9         },
10        "href": "https://api.spotify.com/v1/users/abelm90",
11        "id": "abelm90",
12        "type": "user",
```

Ahora Fran nos ha pasado una carpeta de colección de requests sobre la que vamos a trabajar (Rick and Morty API).

Postman nos permite crear **entornos**, que son eso, entornos en los que a uno o más parámetros se les da un valor y se queda ese valor guardado. Estos parámetros en postman van entre dos llaves `{{}}`. Se añaden en el icono del ojo (arriba a la derecha).

El último endpoint es un filtrado, en el que en la url está formado en querystring, y se puede rellenar simplemente uno de los valores y eliminar el resto y la api nos devuelve todos los resultados que coinciden en ese parámetro. En postman los test se denominan con **pm**.

Hasta ahora esto son pruebas manuales, vamos a aprender a hacer tests automáticos. Por ejemplo en el primer endpoint, tenemos la pestaña "tests".

Tenemos algunos de ejemplo en la colección de Rick y Morty. Ahora vamos a hacer un par de test nosotros, uno positivo y uno negativo.

Positivo:

Ejercicio Spotify / Get all user's playlist

GET

https://api.spotify.com/v1/me/playlists

Params

Authorization ●

Headers (7)

Body

Pre-request Script

Tests ●

Settings

1

pm.test("Playlist access granted", function() {

2

pm.response.to.have.status(200);

3

}

Body

Cookies

Headers (18)

Test Results (1/1)

All

Passed

Skipped

Failed

PASS

Playlist access granted

Negativo:

Ejercicio Spotify / Prueba test negativo Save

GET → https://api.spotify.com/v1/playlists/

Params Authorization Headers (6) Body Pre-request Script Tests ● Settings

```
1 pm.test("Error por falta de playlist ID", function () {
2   var jsonData = pm.response.json();
3   pm.expect(jsonData.error.status).toEqual(401);
4   pm.expect(jsonData.error.message).toEqual("No token provided");
5 });
```

Test scripts are written and are run after the response is received. [Learn more about test scripts](#)

SNIPPETS

- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable

Body Cookies Headers (15) Test Results (1/1) 🌐 Status: 401 Unauthorized Time: 48 ms Size: 807 B

Pretty Raw Preview Visualize JSON ⌵ 🔍

```
1 {
2   "error": {
3     "status": 401,
4     "message": "No token provided"
5   }
6 }
```

PRUEBAS UNITARIAS

Hemos creado una carpeta nueva para las pruebas unitarias. Usaremos el programa JEST. Hemos creado un archivo js llamado suma, con una función simple de suma, otro de convertir a mayúsculas y otro para hacer el reverse de un string.

```
JS suma.js U X JS mayuscula.js U JS inverso.js U
QA_testing > pruebasUnitarias > JS suma.js > ...
1 function suma(a,b){
2   return a+b;
3 }
4
5 module.export = suma();
```

```
JS suma.js U    JS mayuscula.js U X    JS inverso.js U

QA_testing > pruebasUnitarias > JS mayuscula.js > ...
1  function mayusculas(aux){
2      return aux.toUpperCase()
3  }
4
5  module.export = mayusculas();
```

```
JS suma.js U    JS mayuscula.js U    JS inverso.js U X

QA_testing > pruebasUnitarias > JS inverso.js > ...
1  function inverso(aux){
2      return aux.split("").reverse.join(""); // le damos la vuelta a una palabra
3  }
4
5  module.export = inverso();
```

Ahora instalamos el JEST (npm install jest), y en nuestra carpeta “pruebasUnitarias” creamos una carpeta llamada “__test__” esta es la nomenclatura estándar para las pruebas unitarias con jest. Ahora, dentro de la carpeta __test__ creamos los archivos suma.test.js, mayuscula.test.js, inverso.test.js Aquí los test se llaman **test**.

Debemos modificar el package.json de esto:

```
6  "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8  },
9  "keywords": [],
10 "author": "",
```

A esto:

```
6  "scripts": {
7      "test": "jest"
8  },
9  "keywords": [],
```

Tenemos los archivos de las funciones y las funciones.test en la carpeta de archivos.
Ejemplo de suma.test.js

```
QA_testing > pruebasUnitarias > __test__ > suma.test.js > ...
1  const suma = require("../suma");
2
3  test("Sumar dos números positivos", () => {
4
5      expect(suma(2,2)).toBe(4);
6
7  })
```

La sintaxis es:

```
test("Título del test", ()=> {
    expect(funciónatestear()).toBe(resultadoesperado);
})
```

Para correr los test en consola escribimos el comando: `npm run test`

Métodos comunes de comparación:

`.toBe()` => usa `Object.is` para probar la igualdad exacta.

`.toEqual()` => comprueba recursivamente cada campo de un objeto o array.

`.toBeNull` => solo coincide con nulo

`.toBeUndefined` => coincide solo con `undefined`

`.toBeDefined` => es lo opuesto a `toBeUndefined`

`.toBeTruthy` => coincide con cualquier cosa que una declaración `if` trate como verdadera

`.toBeFalsy` => coincide con cualquier cosa que una declaración `if` trate como falsa

Ahora mismo hemos estado dándole valores a las funciones a lo hardcore, pero vamos a hacer una sólo función que le podamos dar los valores que queramos (DDT- Data Driven Testing).

```
28
29 test.each([[2,2,4],[1,1,2],[3,5,8]])(
30     'i%+i% igual a i%', (a,b,expected) =>{
31         expect(suma(a,b)).toBe(expected)
32     }
33 )
```

Testing de Aceptación de Usuario

Usamos el framework Cucumber. Lo vamos a hacer en la misma carpeta de las pruebas de Cypress, usando ahora el comando `npm install cucumber`

Y para que cypress pueda funcionar con cucumber hacemos:

```
npm install cypress-cucumber-preprocessor
```

E instalamos la extensión de visual studio: Cucumber (Gherkin) Full Support

En el package.json hay que modificar y añadir:

```
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "cucumber": "^6.0.7",
14    "cypress": "^9.6.0",
15    "cypress-cucumber-preprocessor": "^4.3.1"
16  },
17  "cypress-cucumber-preprocessor": {
18    "nonGlobalStepDefinitions": true
19  }
20 }
21
```

Y dentro de la carpeta integration creamos otra carpeta llamada BDD. Y dentro de ésta un archivo llamado: `step_definitions.js`

Y ahora dentro de integration creamos un archivo que se llame `login.feature`

step_definitions: (hecho con el lenguaje Gherkin)

```
GIVEN(' I navigate main page')
WHEN(' I click on Form Authentication')
AND(' I enter username)
AND...
AND...
THEN ('Message Appears')
```

Y ahora en el login.feature hacemos algo tal que así (las descripciones del Given, When, And's y Then deben ser las mismas que el step_definitions):

```
QA_testing > pruebasCypress > cypress > integration > 🌱 login.feature
1   Feature: Test de Login
2
3   Scenario: Login Válido
4
5   Given I navigate main page
6   When I click Form Authentication
7   And I enter username
8   And I enter password
9   And I click the Login button
10  Then A message appears
11
```

```
12  Scenario: Login Inválido
13
14  Given I navigate main page
15  When I click Form Authentication
16  And I enter username
17  And I enter wrong password
18  And I click the Login button
19  Then An error message appears
```


