

¿Qué es Postman aparte de un Cliente REST?

Es un ADE (API Development Environment), una plataforma que soporta y mejora el desarrollo de APIs.

Entre sus herramientas permite:

1. Mocks y Diseño de APIs
2. Monitoreo de APIs
3. Documentación de APIs
4. Testing automático de APIs
5. Debug (Cliente REST, Crear Test)
6. Publicación de APIs

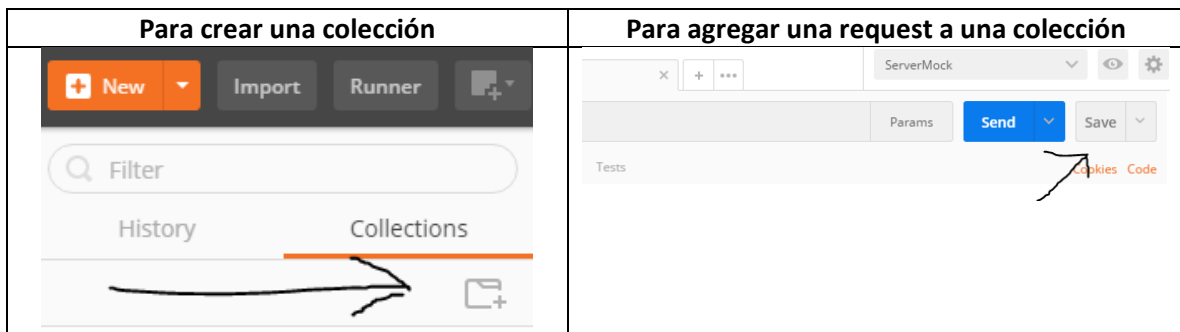
Antes de hablar de testing en Postman introduciremos algunos conceptos:

Colecciones

Una Colección en postman son Request individuales agrupadas en una 'carpeta'.

Para que sirven:

1. Organización
2. Documentación
3. Test suites
4. Conditional workflows



Variables y Entornos

Un Environment es un conjunto de variables, y cada una de estas son un par de key-value.

Trabajando con APIs, usualmente se necesitan diferentes setups, para el servidor en producción o para el servidor en tu maquina local.

Imaginen que tienen una colección con 100 request guardadas y de repente el dominio (mi-dominio.com) de la solución cambia a otro dominio, por lo que se tendrá que cambiar para cada request el dominio. En lugar de ser: mi-dominio.com/endpoint1 pasa a {{domain}}/endpoint1, y la

variable domain se define en el entorno.

GET `{{domain}}/api/endpoint` Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Cookies Code

Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> timestamp	<code>{{timestampHeader}}</code>				
New key	Value	Description			

MANAGE ENVIRONMENTS

Edit Environment

ServerMock

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> domain	<code>https://484de04b-30ae-46e1-887f-77a37bd78c40.mo...</code>	
<input checked="" type="checkbox"/> timestampHeader	<code>2018-03-21T20:42:01.959Z</code>	
New key	Value	

Cancel Update

Postman provee algunas variables útiles:

1. `{{$guid}}` retorna un guid
2. `{{timestamp}}` retorna timestamp actual
3. `{{randomInt}}` retorna un int entre 0 y 1000

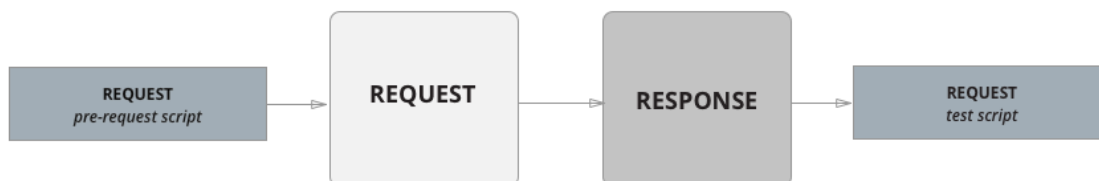
Scripts

A través de estos, Postman puede agregar comportamiento dinámico a las requests y colecciones. Esto nos permite ejecutar test y crear request con parámetros dinámicos. El lenguaje para crear estos scripts es JavaScript.

Hay 2 tipos de scripts:

- pre-request scripts (que se ejecuta antes de que la request sea enviada al servidor)
- test scripts que se ejecuta después de que la response del servidor es recibida.

Orden de ejecución:



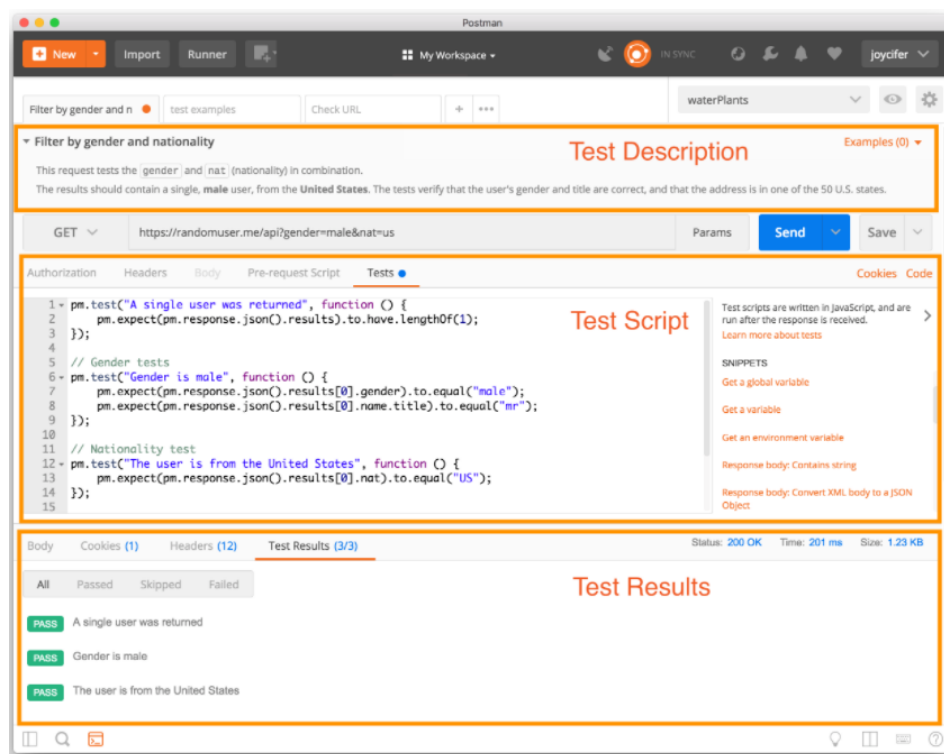
Pre-request scripts

Los pre-request scripts son útiles para incluir headers y modificar o agregar variables antes de una request.

Ejemplo setear la fecha actual en una variable.

```
Authorization Headers Body Pre-request Script Tests
1 //OBTENER UNA VARIABLE DEL ENTORNO GLOBAL
2 pm.globals.get("variable_key");
3
4 //OBTENER UNA VARIABLE DEL ENTORNO
5 pm.variables.get("variable_key");
6
7 //SETEAR UNA VARIABLE DEL ENTORNO GLOBAL
8 pm.environment.set("variable_key", "variable_value");
9
10 //SETEAR UNA VARIABLE DEL ENTORNO
11 pm.globals.set("variable_key", "variable_value");
12
13 //ELIMAR UNA VARIABLE DEL ENTORNO GLOBAL
14 pm.globals.unset("variable_key");
15
16 //ELIMINAR UNA VARIABLE DEL ENTORNO
17 pm.environment.unset("variable_key");
18
19 //ENVIAR UNA REQUEST Y OBTENER UNA RESPONSE
20 pm.sendRequest("URL", function (err, response) {
21     console.log(response.json());
22 });
```

Tests scripts



Los test en postman se realizan invocando al método `pm.test()`.

Este recibe 2 parámetros, el nombre del test y una función que es la que contiene la lógica del test; debiendo retornar un bool o invocar un método **`pm.response.to.*`** o **`pm.expect()`**

Ejemplo: espera que la response tenga el status code 200

```
pm.test("Status code is 200", function () {  
    pm.response.to.have.status(200);  
});
```

`pm.response.to.*`: Provee atajos para los chequeos frecuentes usados en response.

Ejemplo: espera que el header "Content-Type se encuentre presente

```
pm.test("Content-Type is present", function () {  
    pm.response.to.have.header("Content-Type");  
});
```

`pm.expect()`: es una método assert la sintaxis se basa en librería ChaiJS BDD (librería de testing en JS).

Ejemplo: espera que el objeto json retornado el atributo status sea igual a "success"

```
pm.test("Status equal to success", function () {  
    var jsonData = pm.response.json();  
    pm.expect(jsonData.status).to.eql("success");  
});
```

Los test también pueden ser usados para setear variables de entorno que se usaran en otros test:

Ejemplo: Espera que la response tenga status code 200 y recibe a token, si esto ocurre setea la variable token del environment.

```
pm.test("Status code is 200 and set token", function () {  
    var jsonData = pm.response.json();  
    pm.response.to.have.status(200);  
    pm.expect(jsonData).does.not.include("token");  
    if (pm.response.code == 200 && jsonData.token !== null) {  
        pm.environment.set("token", jsonData.token);  
    }  
});
```

Esto podría ser útil ya que el token obtenido podría ser necesario para otros test.

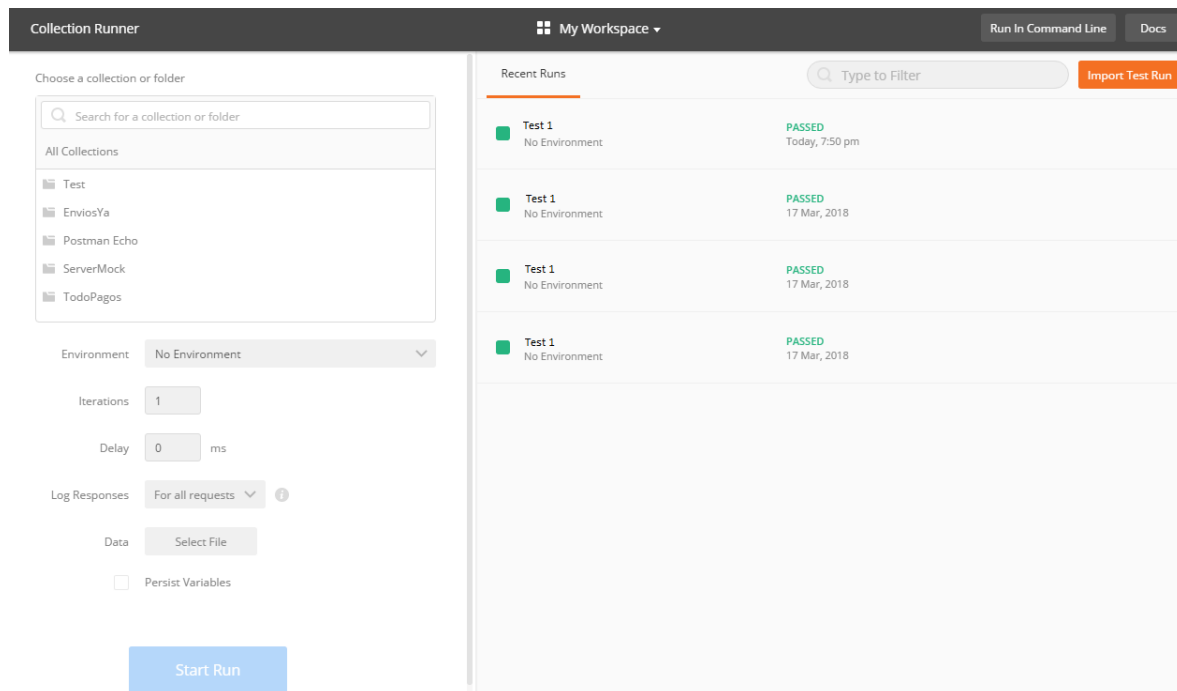
Documentación útil:

1. ChaiJS BDD: <http://www.chaijs.com/api/bdd/>
2. Objeto pm.
https://www.getpostman.com/docs/v6/postman/scripts/postman_sandbox_api_reference
3. Postman SDK: <http://www.postmanlabs.com/postman-collection/index.html>

Testing Automático.

Esto se hace a través de colecciones (grupos de requests) que corren con un enviroment. Usando los test incluidos en las requests.

Para correr las Colecciones ir a Runner.



Collection/Folder: Esta es la colección que va a correr. Las request se ejecutan secuencialmente en orden, esto se puede sobrescribir usando el método `setNextRequest()`.

Environment: Es el enviornment en el cual la colección va a correr.

Iterations: El número de veces que se van a ejecutar las pruebas.

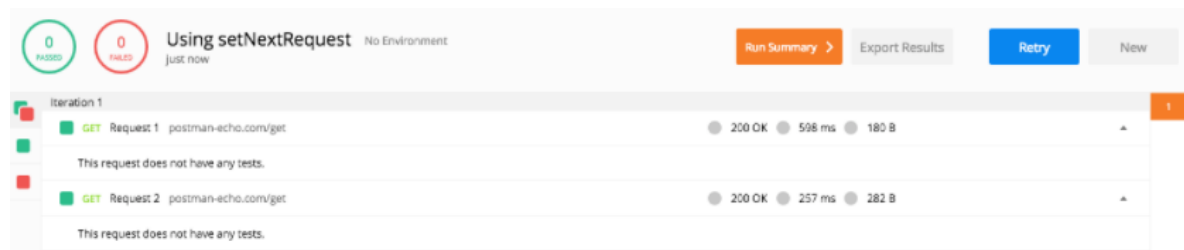
Delay: Intervalo entre cada request (en ms)

Log Responses: Sirve para limitar que responses se van a guardar en el log. Por default, todas las responses son guardadas en el log.

Data: Sirve para proporcionar 'datos' que van a ser usados por la colección que va a correr.

Persist variables: si se quiere que los cambios que se hagan al enviornment se almacenen al terminar los test o no.

Start Run: Empieza a correr los test.



The screenshot displays the Postman test results interface. At the top, there are two circular progress indicators: a green one labeled '0 PASSED' and a red one labeled '0 FAILED'. To the right of these is the text 'Using setNextRequest' and 'No Environment just now'. Further right are buttons for 'Run Summary >', 'Export Results', 'Retry', and 'New'. Below this header, the results are organized into a table. The first section is 'Iteration 1', which contains two rows. The first row is for 'Request 1 postman-echo.com/get', showing a status of '200 OK', a response time of '598 ms', and a response size of '180 B'. Below this row is the text 'This request does not have any tests.' The second row is for 'Request 2 postman-echo.com/get', showing a status of '200 OK', a response time of '257 ms', and a response size of '282 B'. Below this row is also the text 'This request does not have any tests.' On the far right of the table, there is a small orange tab labeled '1'.

Iteration	Request	Status	Response Time	Response Size
Iteration 1	Request 1 postman-echo.com/get	200 OK	598 ms	180 B
	This request does not have any tests.			
Iteration 1	Request 2 postman-echo.com/get	200 OK	257 ms	282 B
	This request does not have any tests.			