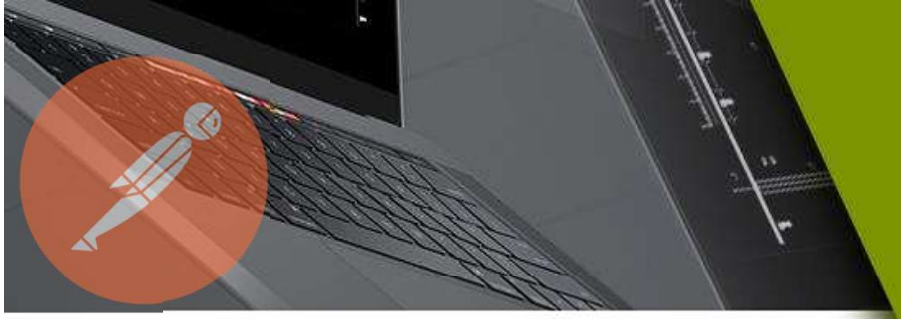
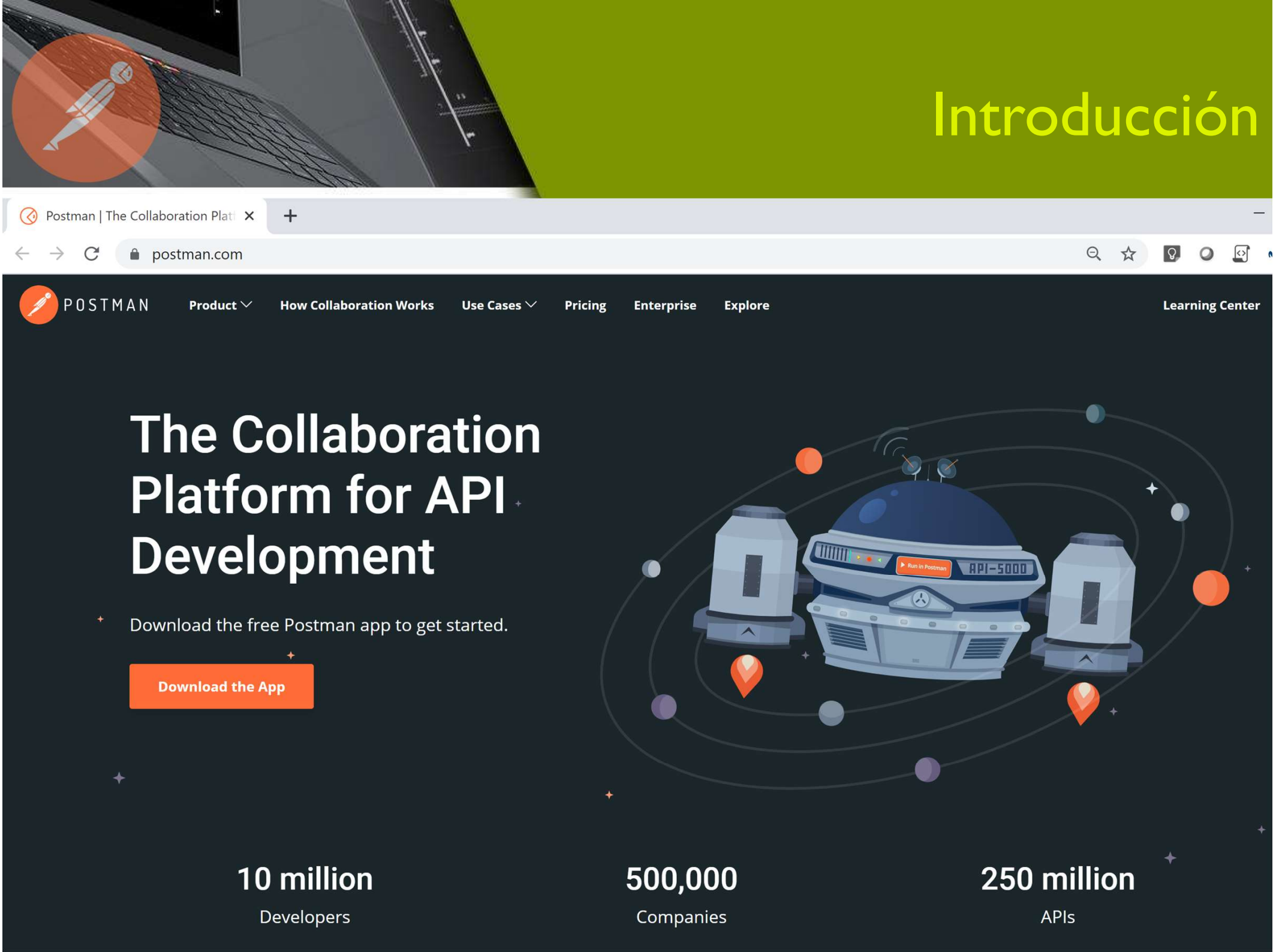


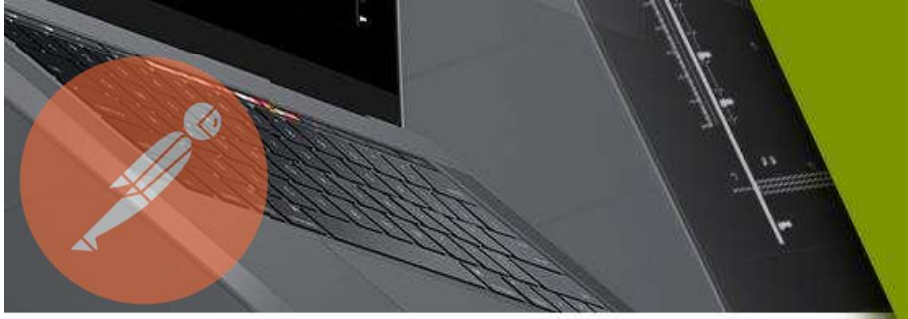
Introducción

POSTMAN

- Postman es la herramienta líder del sector del testing para peticiones de la API
- Gratuita (con versiones de pago)
- Multiplataforma
- Escrita en lenguaje Electron

Introducción

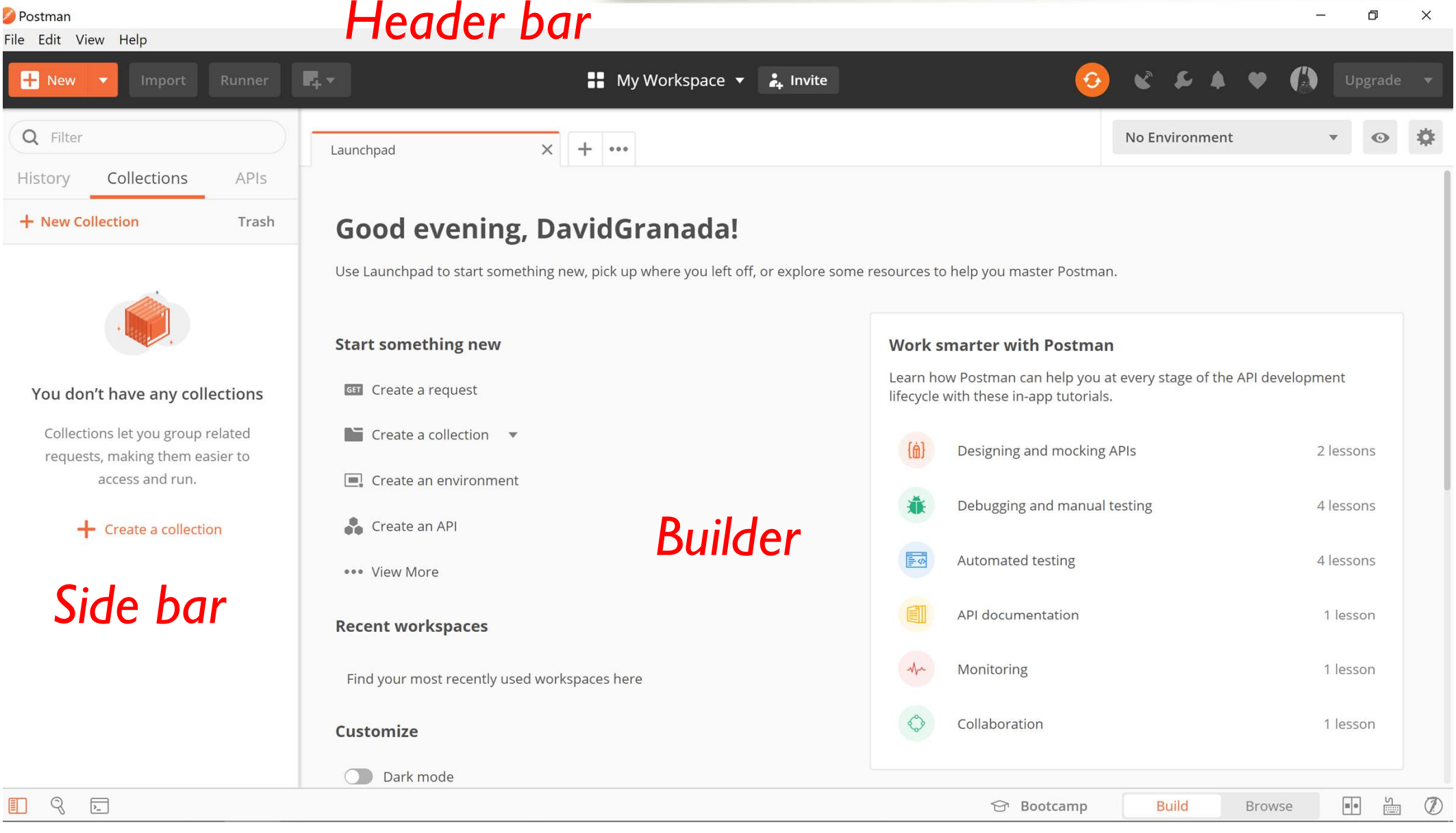




Introducción

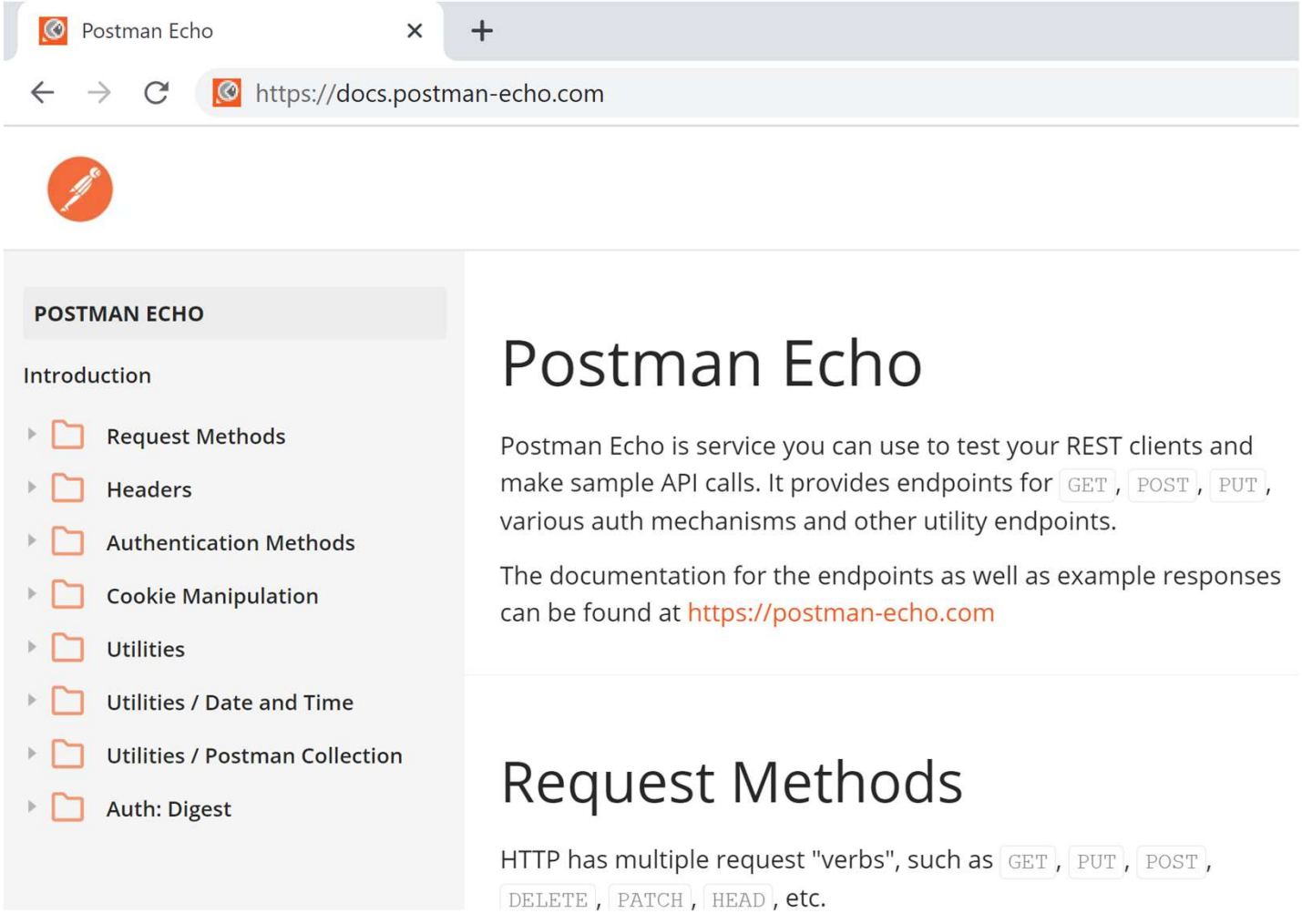
- Existe la posibilidad de instalar Postman como un complemento de Google Chrome, pero esta instalación tendrá muchas limitaciones
- Conviene instalar la versión completa de escritorio para explorar todas sus funcionalidades

Introducción



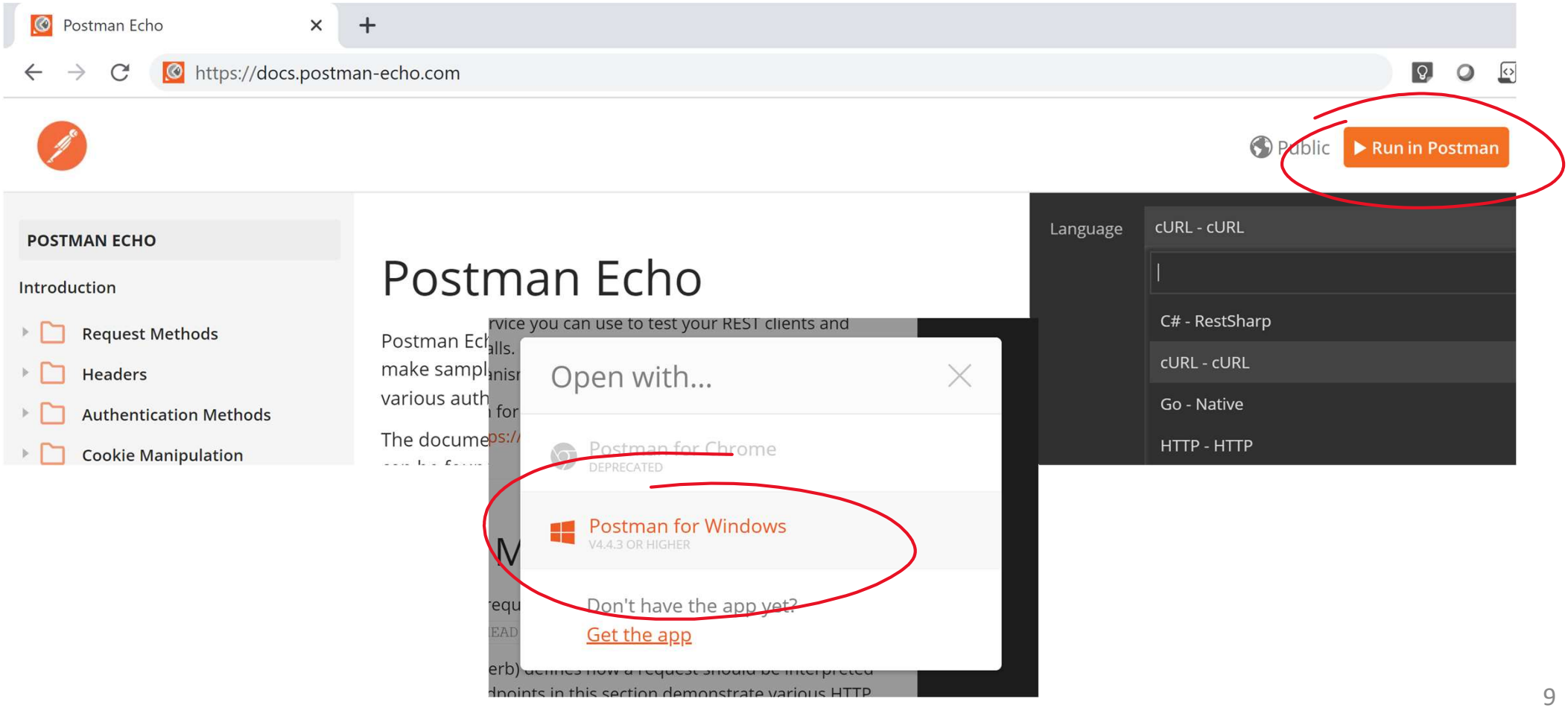
Introducción

- La misma página web de Postman nos proporciona unos servicios web de ejemplo que nos permitirán probar la herramienta:



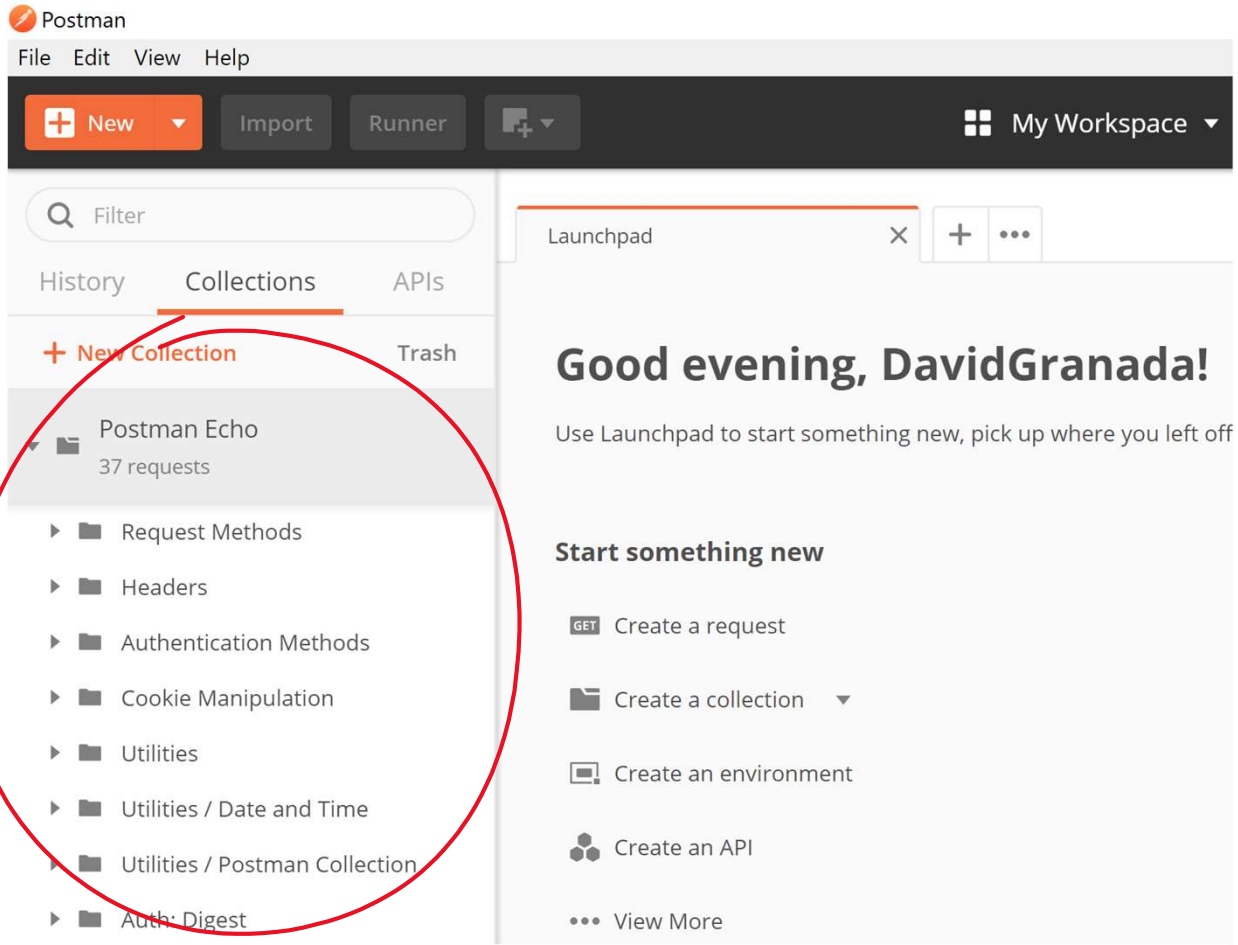
Introducción

- Podemos "llevarnos" estos servicios directamente a la aplicación para probarlos:



Introducción

- Esto nos creará una colección con 37 servicios web de prueba (peticiones web)



Introducción

- Hay listados de aplicaciones públicas desde las que podemos obtener todo tipo de información para crear nuestros servicios web:

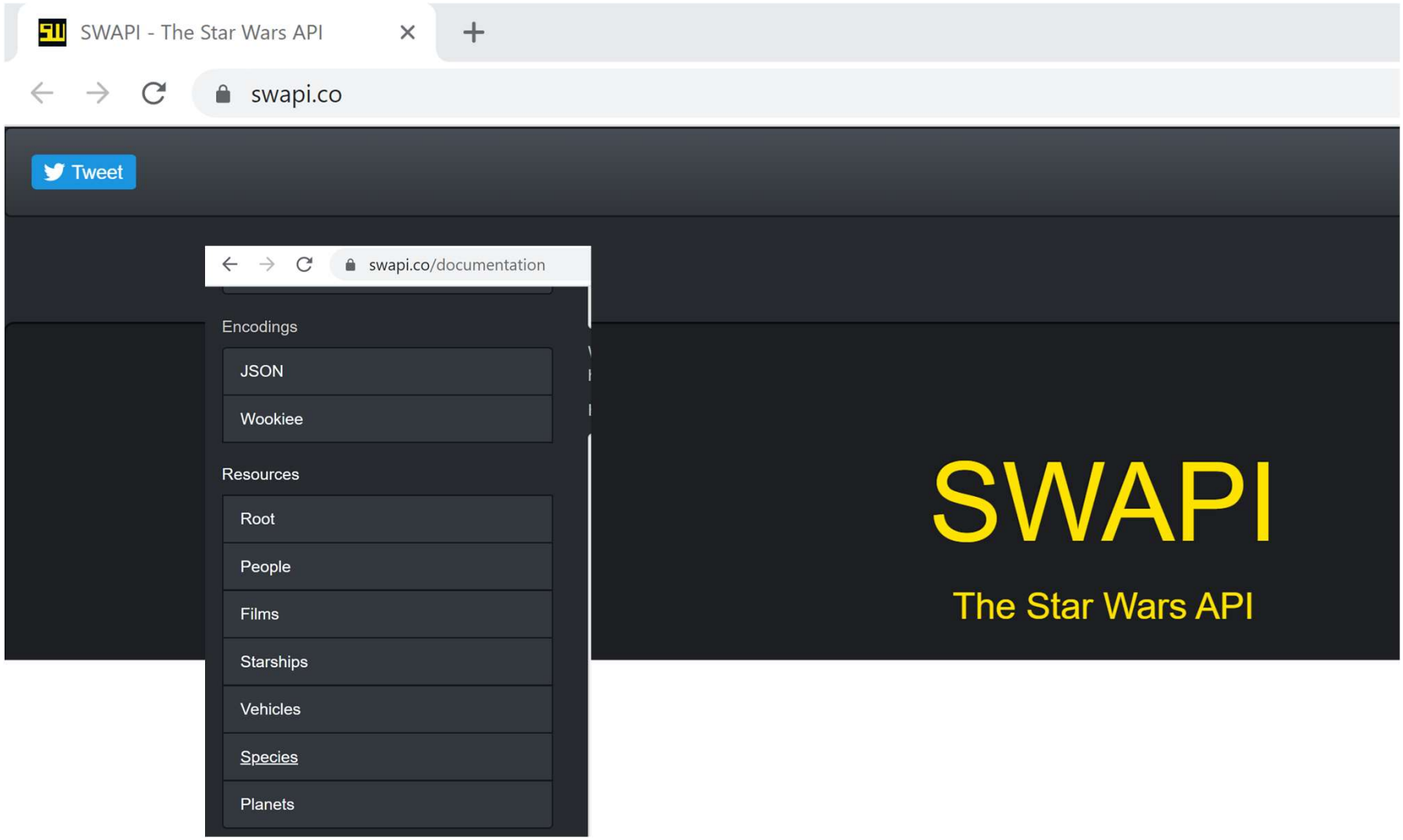
Any API

Documentation and Test Consoles for Over 1400 Public APIs

Powered by LucyBot and APIs Guru

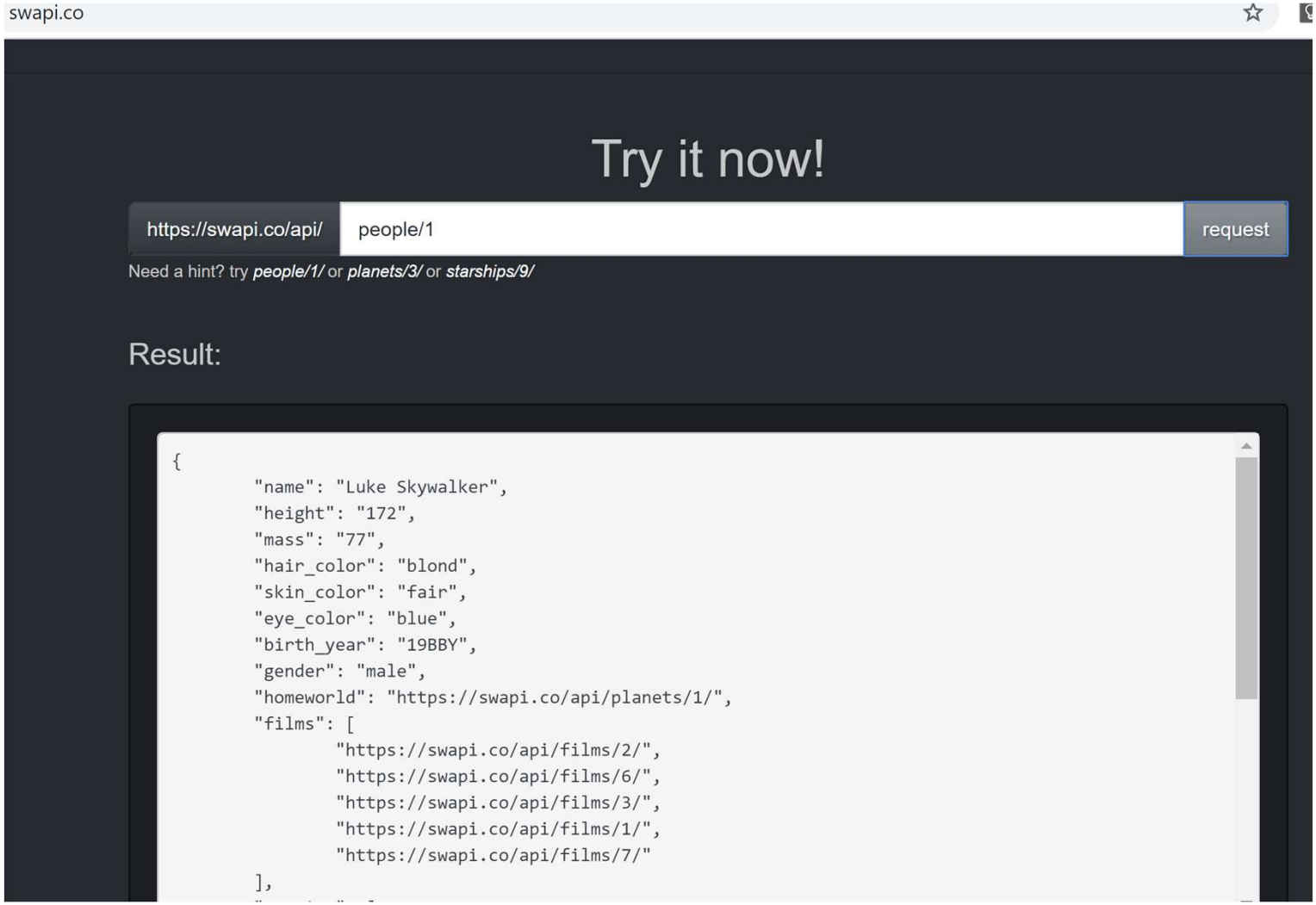
Introducción

- API sobre el universo de Star Wars:



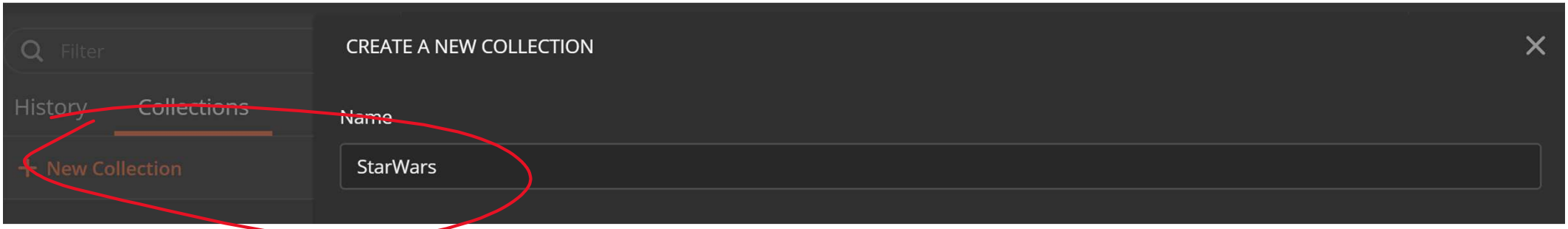
Introducción

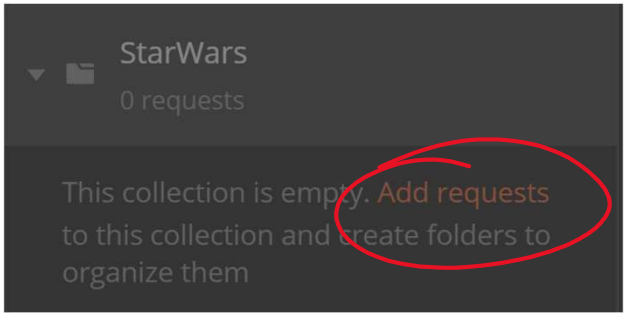
- Esta web nos permite probar peticiones sobre sus datos:

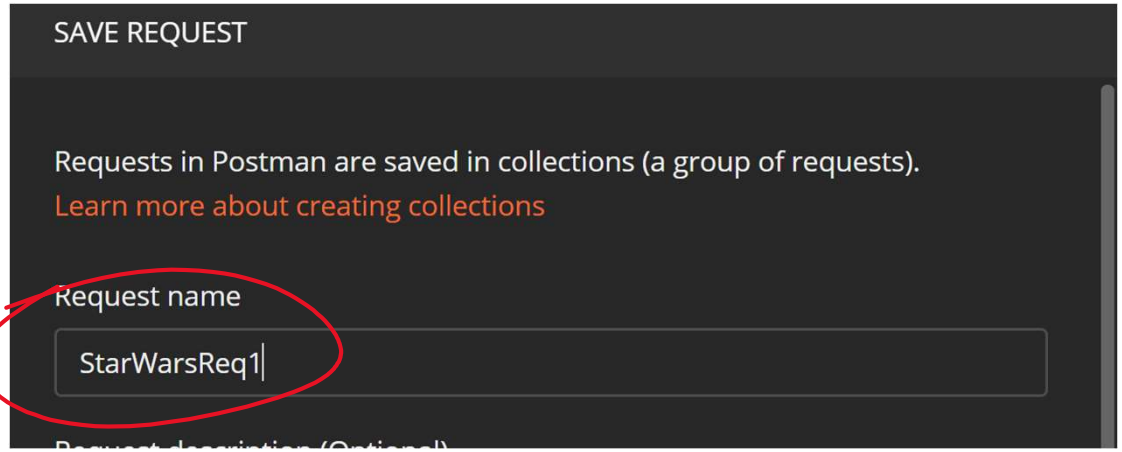


Introducción

- Podemos abrir Postman, crear una colección y realizar peticiones de datos sobre esta API:

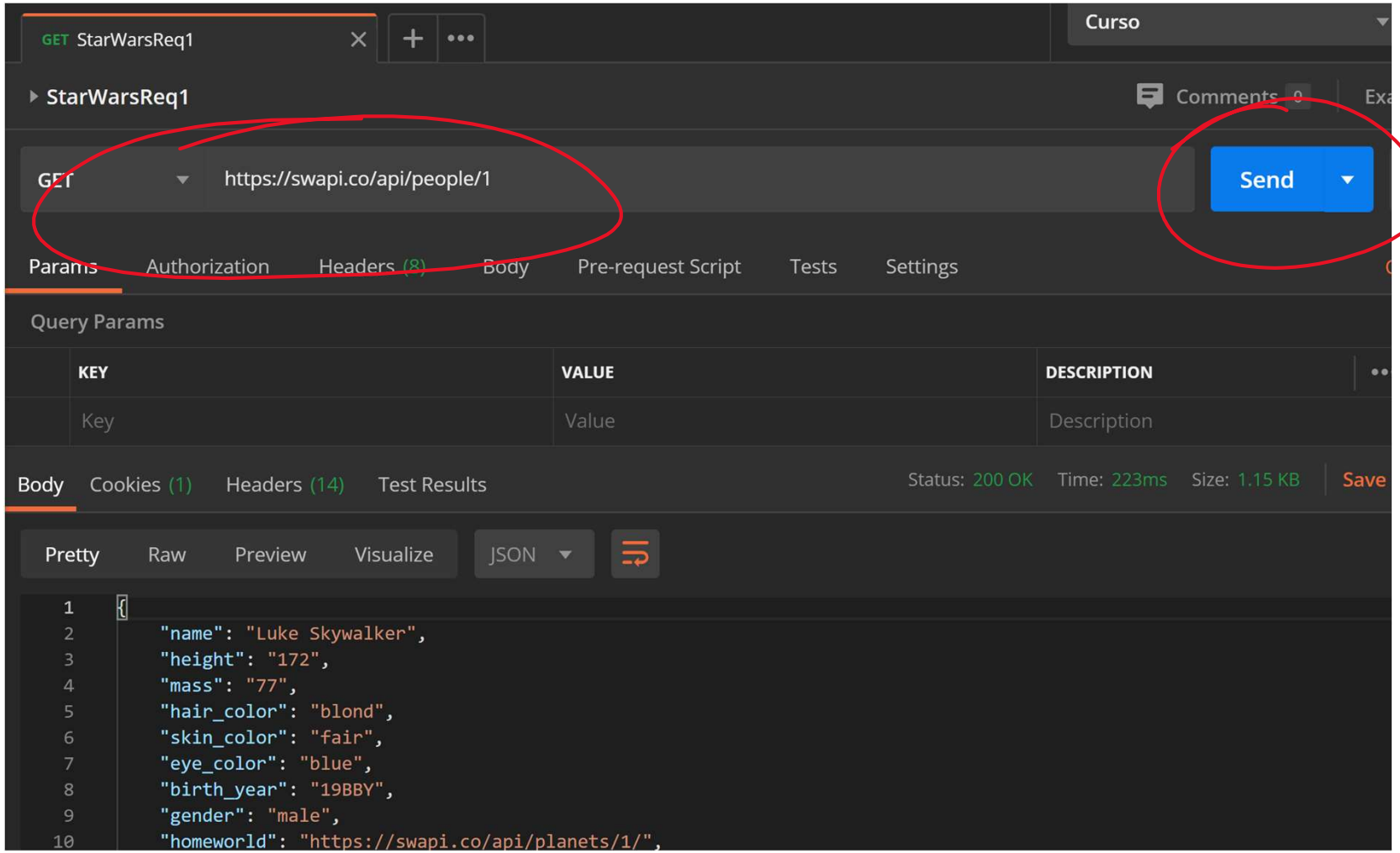






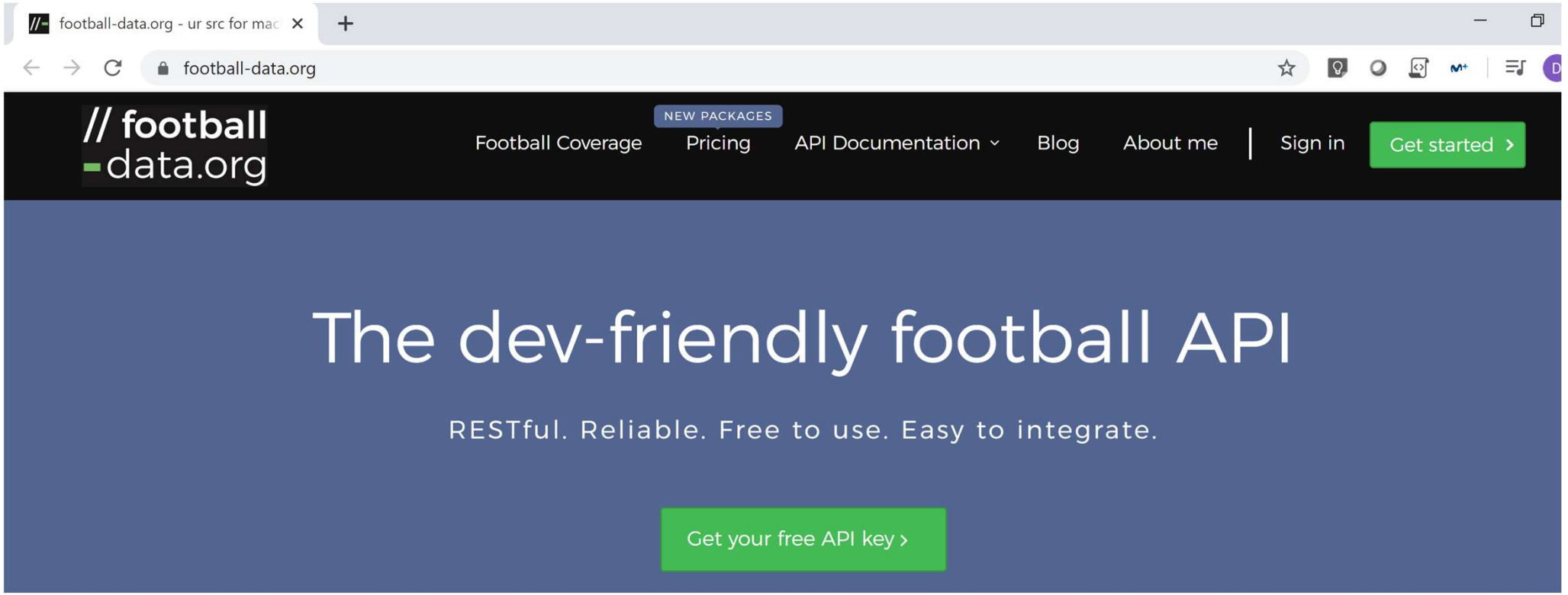
Introducción

- Creamos la petición y probamos otras peticiones:



Introducción

- API de datos sobre fútbol (crear una cuenta):



Introducción

- Hay versiones gratuitas y de pago:

football-data.org/pricing

Free	Free + Three	Standard	Advanced
€0,00 /mo	€25,- /mo	€49,- /mo *	€99,- /mo *
12 competitions	15 competitions	25 competitions	50 competitions
Scores delayed	Choose 3 to your liking	Live scores	Live scores
Fixtures, Schedules	Live scores	Fixtures, Schedules	Fixtures, Schedules
League Tables	Fixtures, Schedules	League Tables	League Tables
—	League Tables	Line-ups & Subs	Line-ups & Subs
—	Line-ups, Squads & Subs	Goal scorers	Goal scorers
—	Goal scorers	Bookings / Cards	Bookings / Cards
—	Bookings / Cards	Squads	Squads
10 calls/minute	30 calls/minute	60 calls/minute	100 calls/minute
Get Started ›	Get Started ›	Get Started ›	Get Started ›

Introducción

- Al registrarnos nos envían un token que nos permitirá obtener los datos de las peticiones a esta API:

Your free API-key for football-

Daniel (football-data.org) <Daniel@football-data

para mí

inglés > español Traducir men

Hi David,

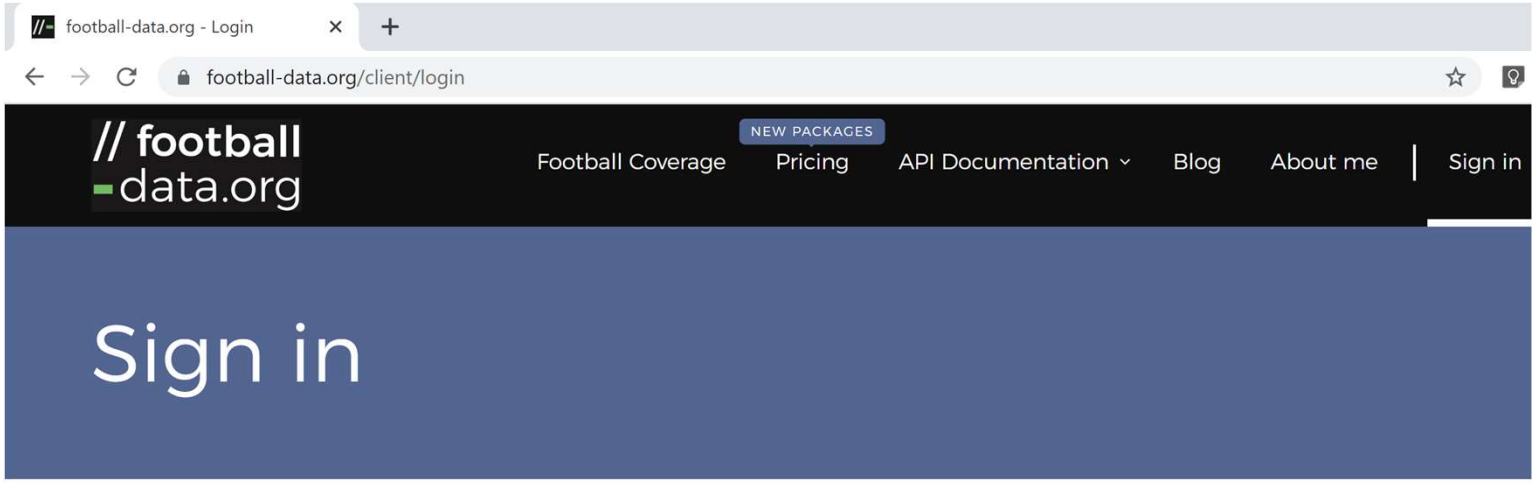
thanks for registering for an API authentication toke
the underneath personal token as value.

Your API email: david.gr4n4d4@gmail.com
Your API token: 1278b5ae3ab74ac199cef459dbea
Your API plan: Free Tier

In order to keep your account active and receive up

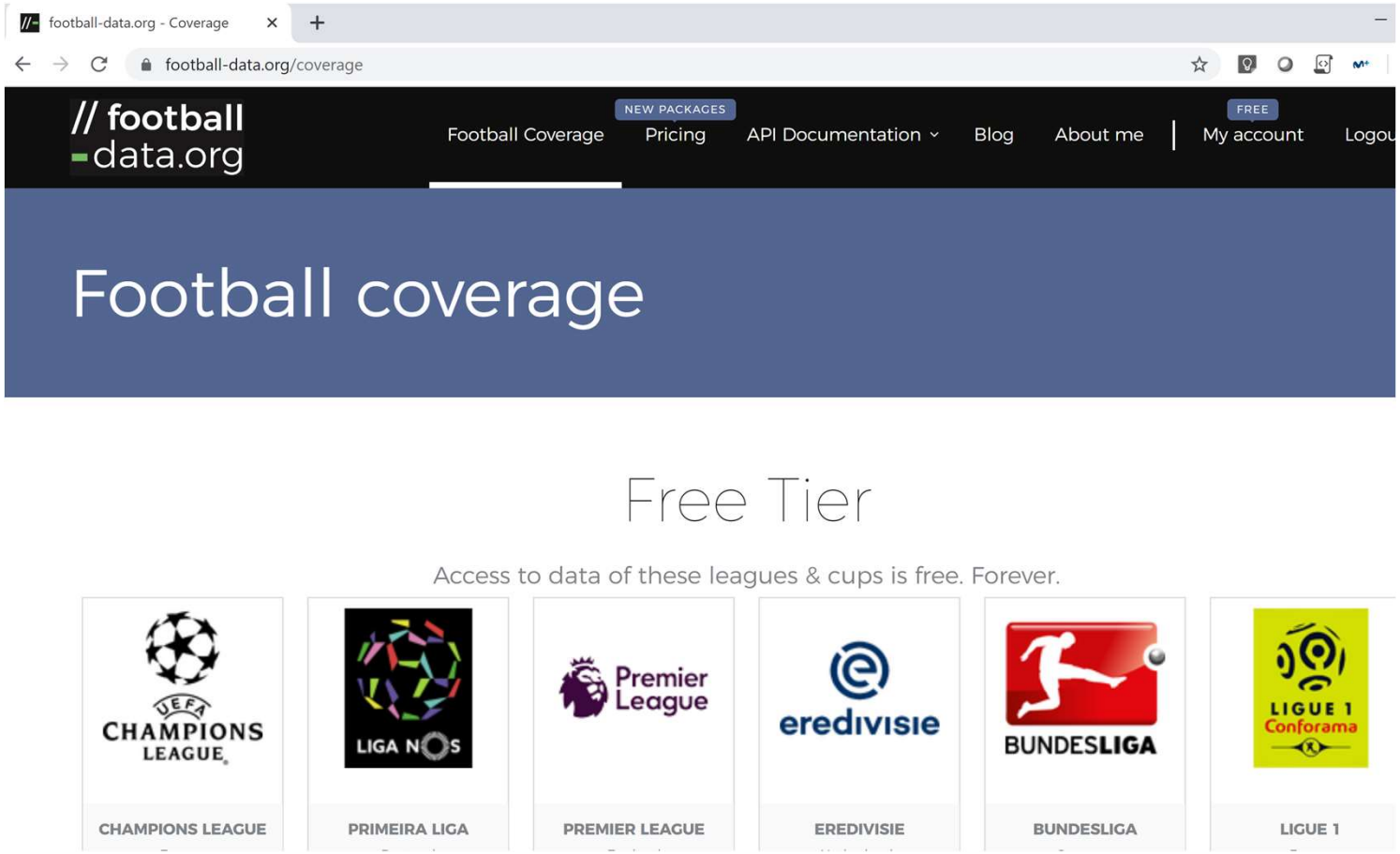
Introducción

- Nos logueamos con el API Token:



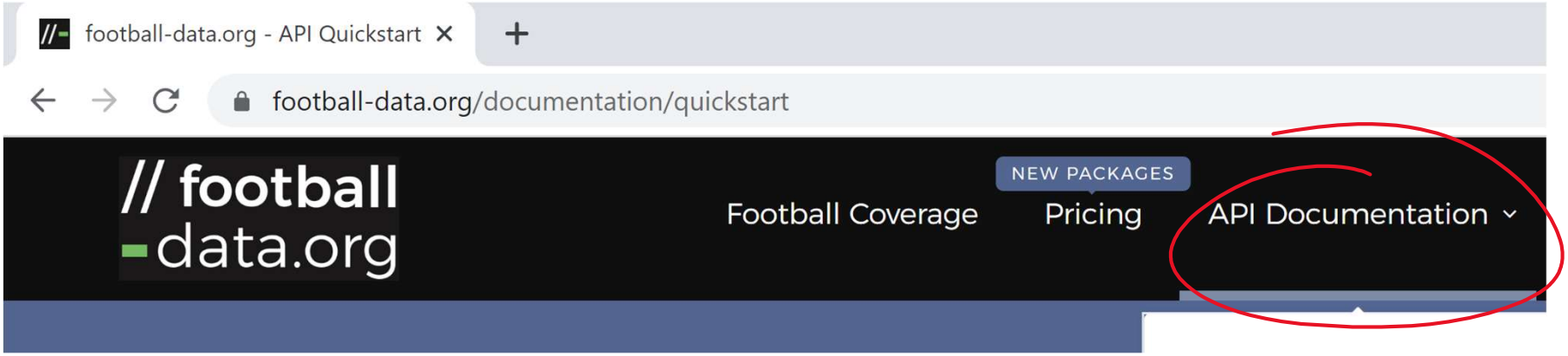
Introducción

- La cuenta gratuita permite obtener datos de 12 competiciones:

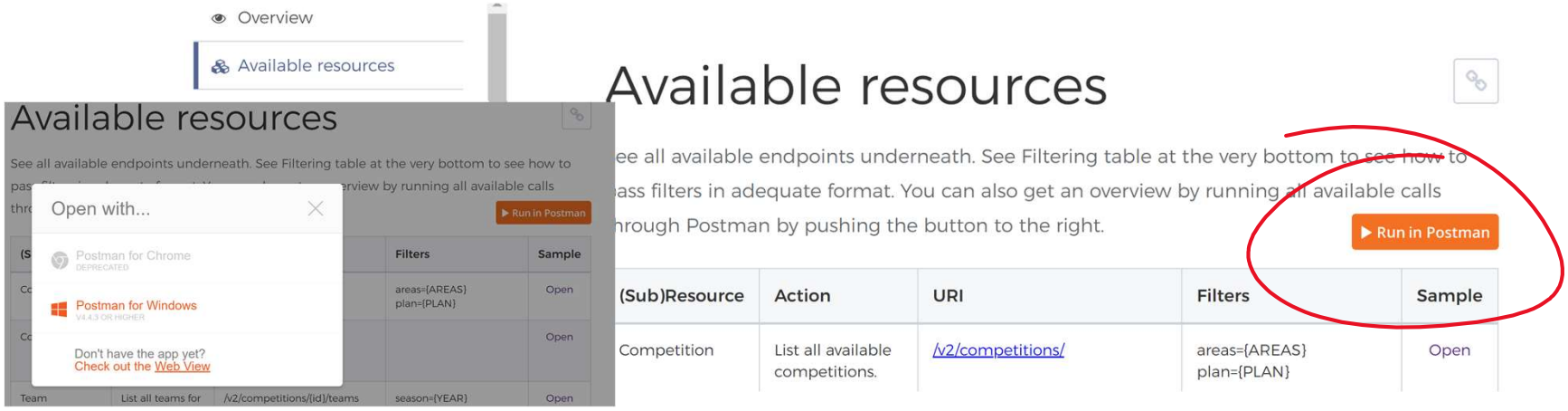


Introducción

- En la documentation tenemos toda la información necesaria para usar esta API:

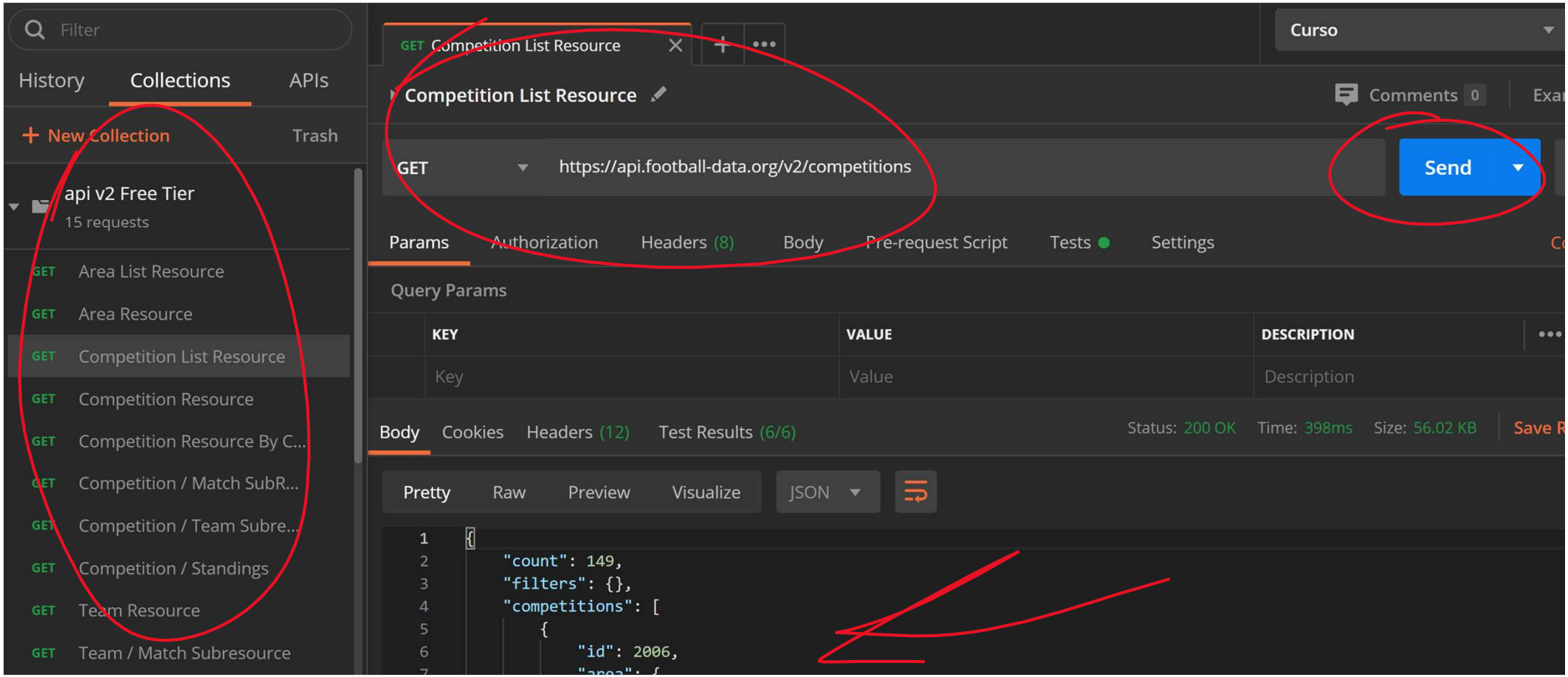


- Vemos que hay un botón de Run in Postman:



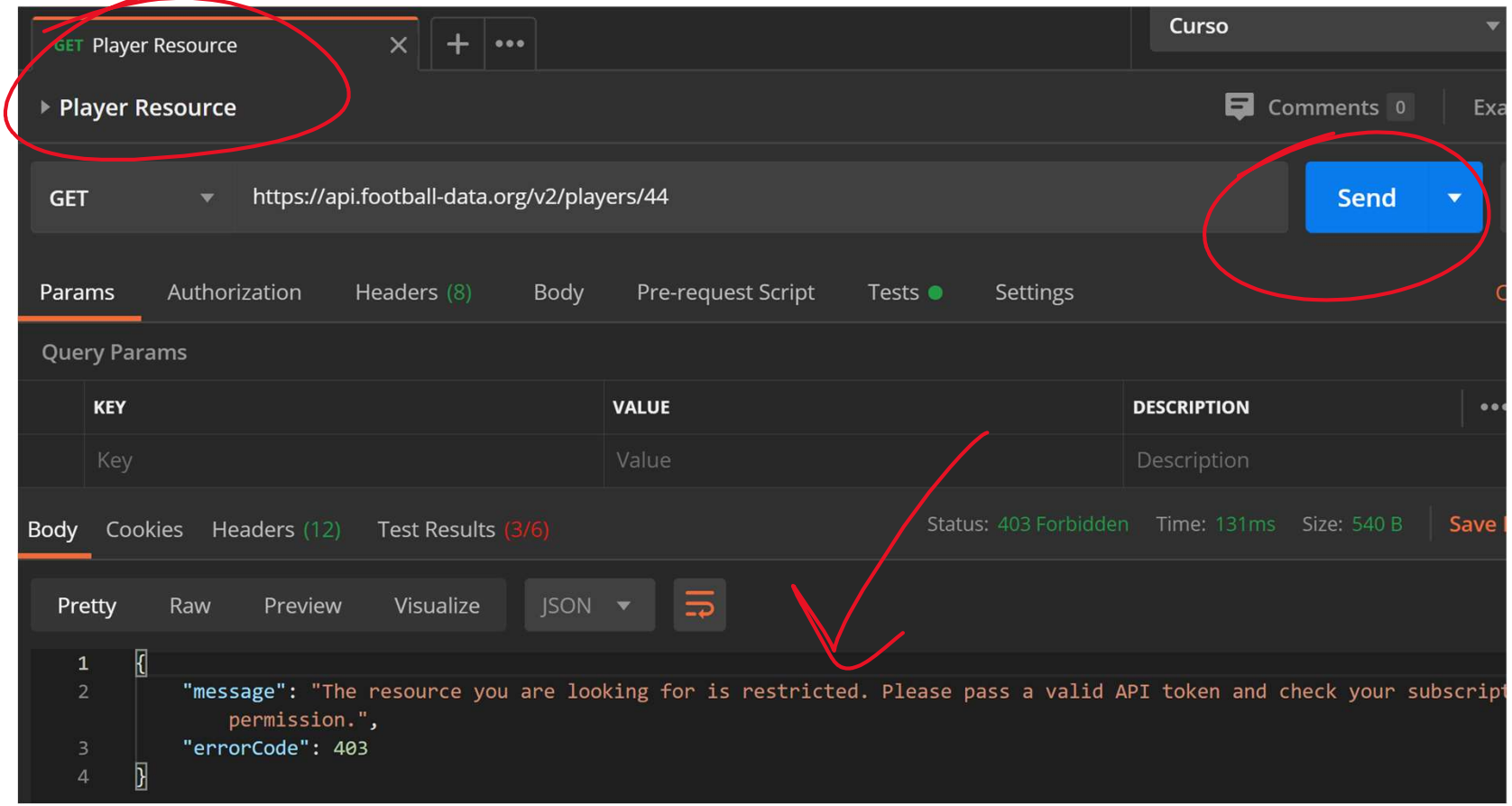
Introducción

- Nos llevará una serie de peticiones de ejemplo al entorno de Postman:



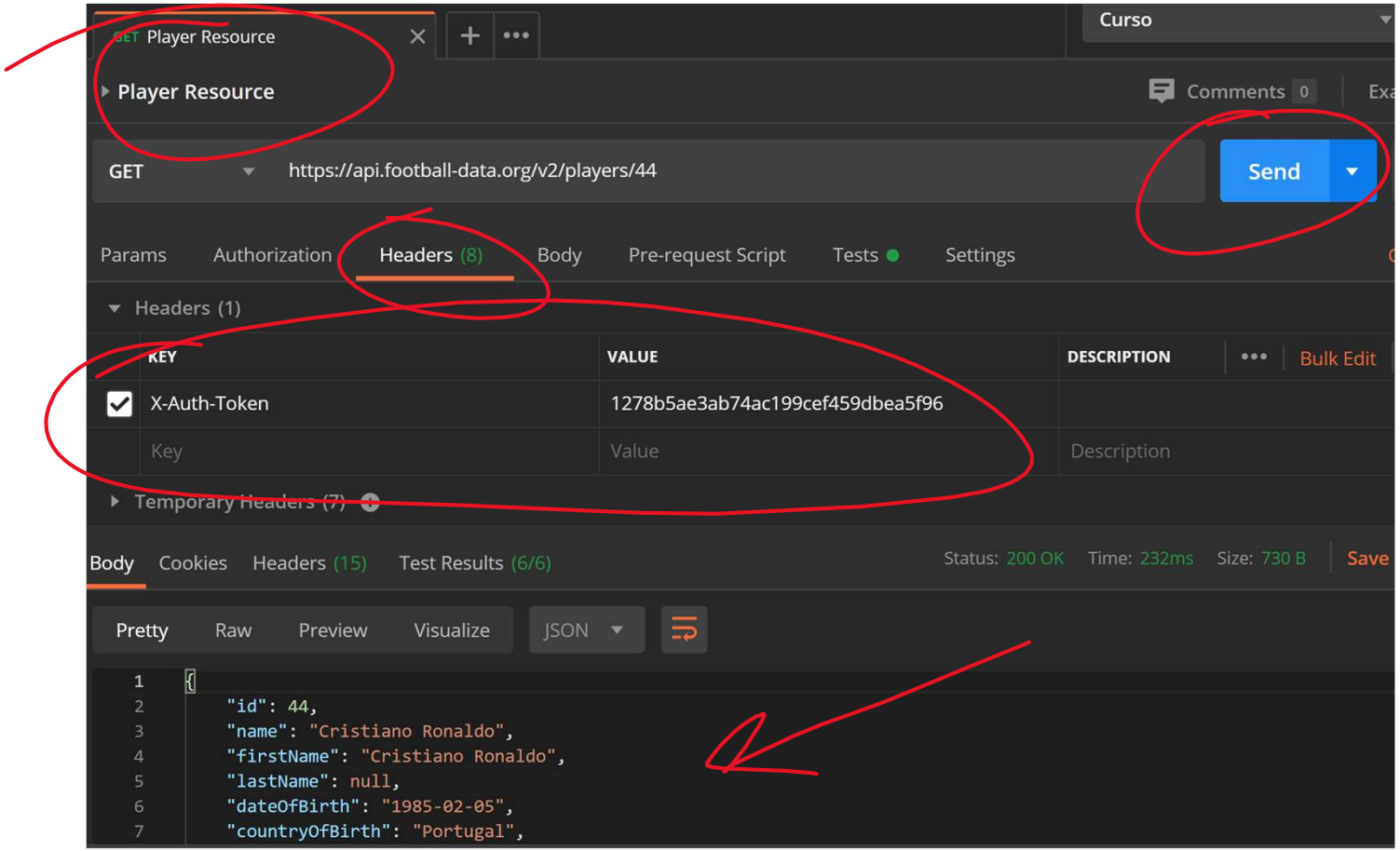
Introducción

- Para algunas peticiones tenemos que usar el Token que nos han proporcionado, si no lo usamos no podríamos obtener los datos:



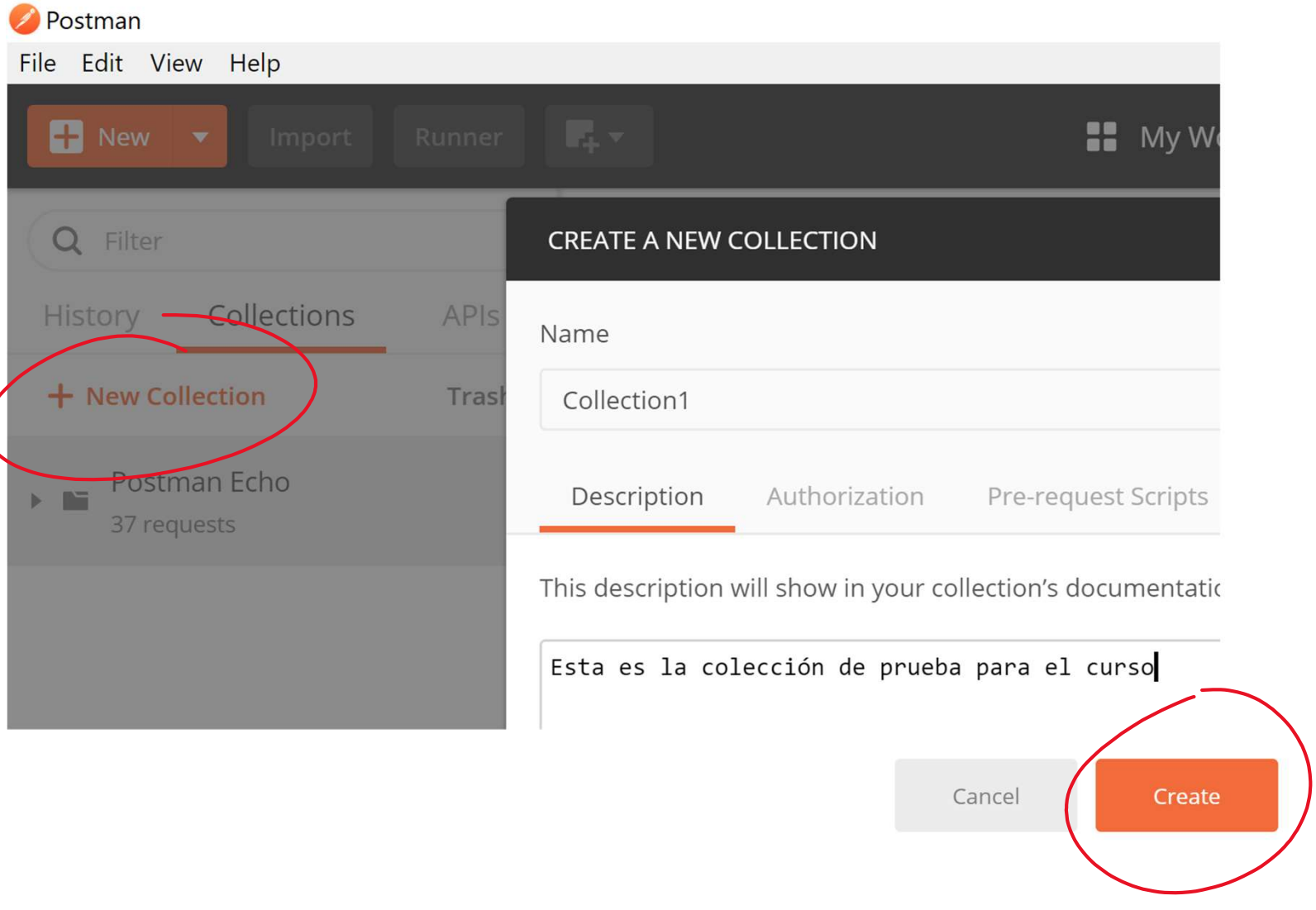
Introducción

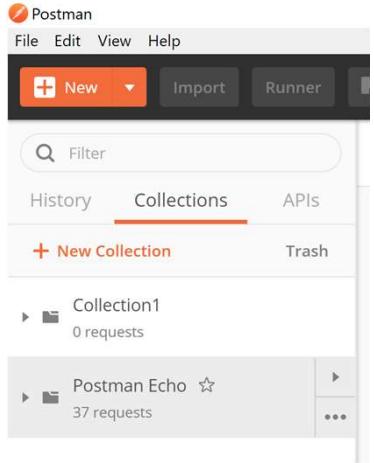
- Insertamos el Token en el Header y al ejecutar obtendremos los datos de la petición:



Introducción

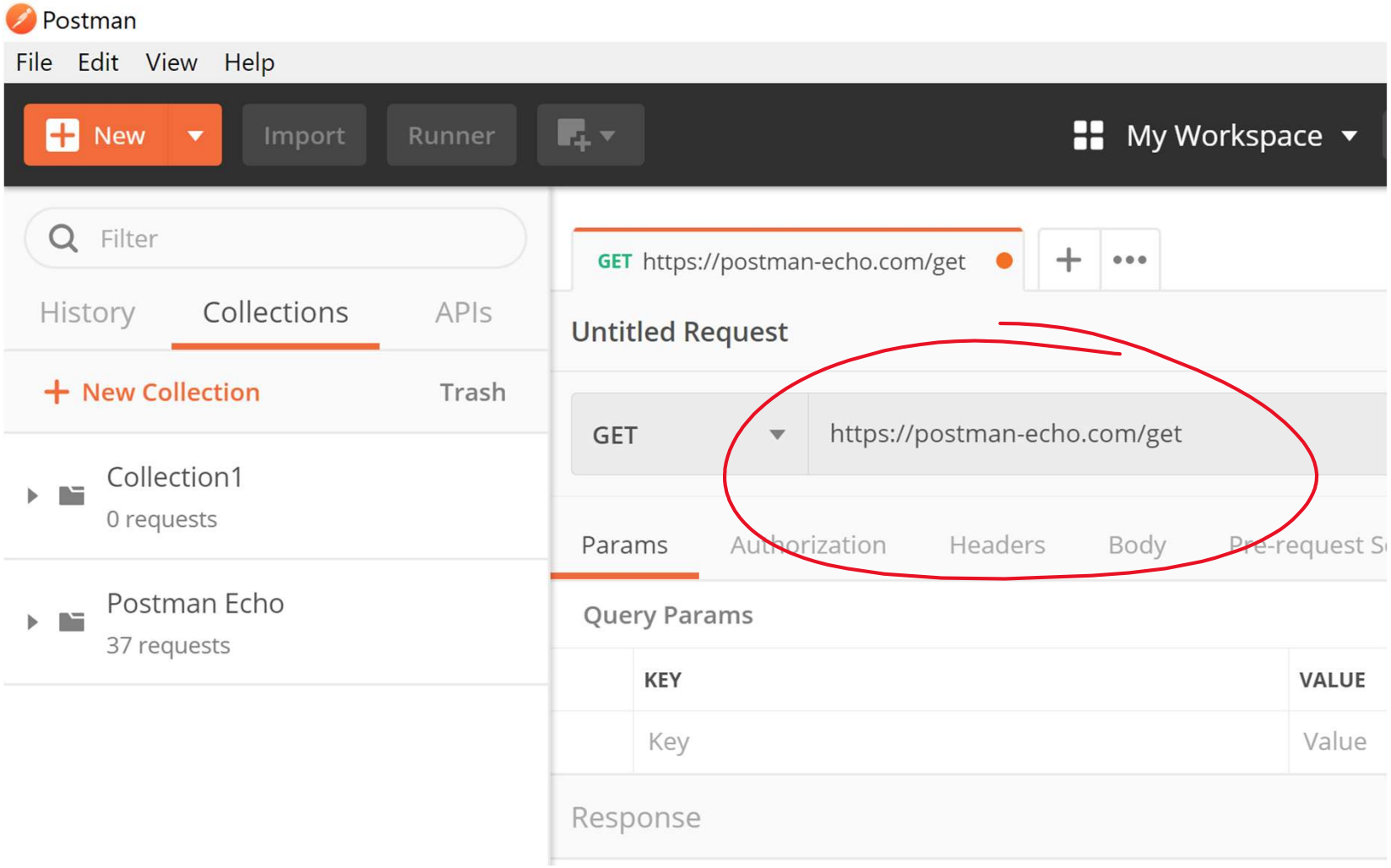
- Vamos a crearnos nuestra primera colección de servicios web:





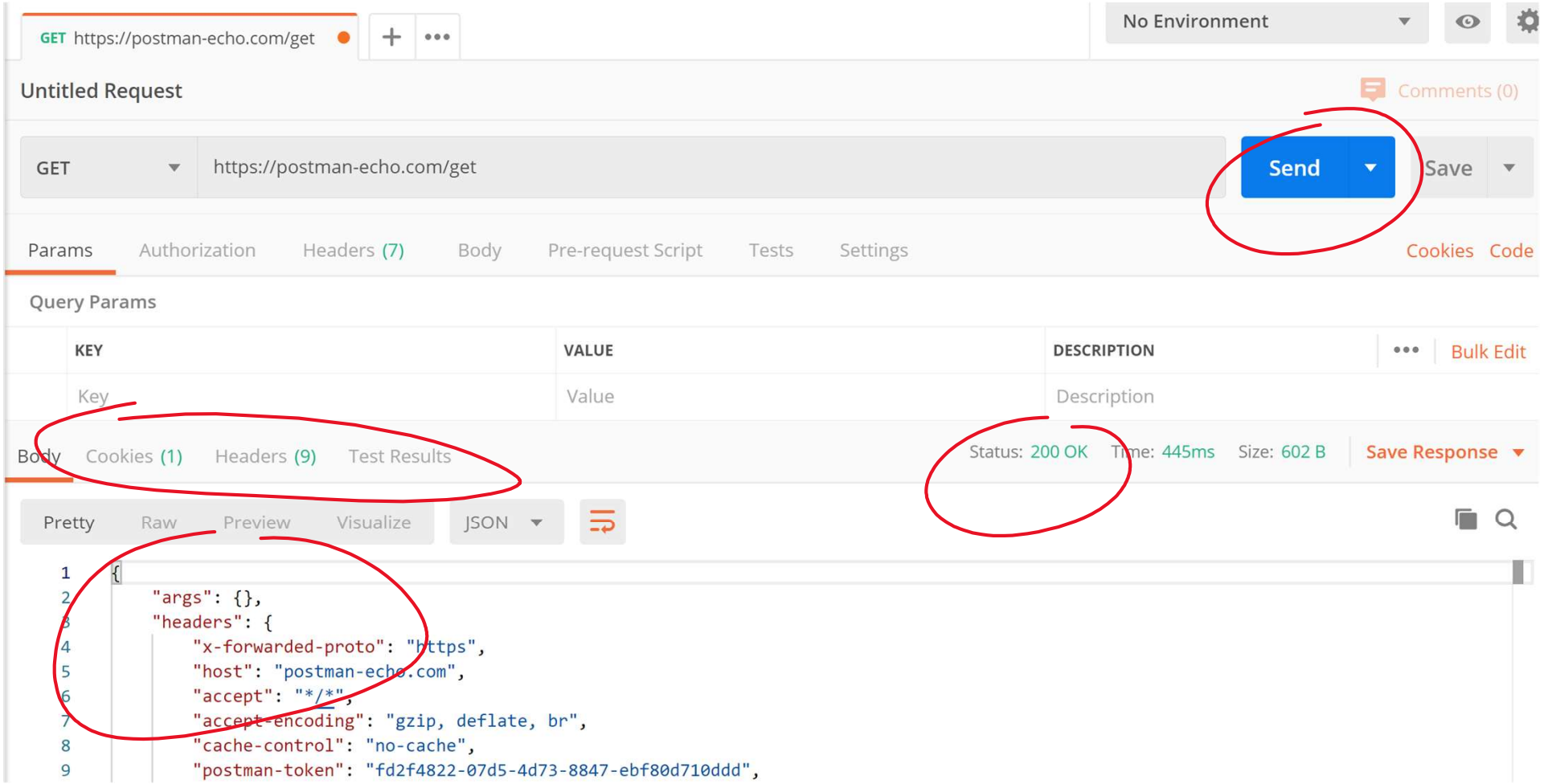
Introducción

- Definimos una primera petición, una llamada a un servicio web:



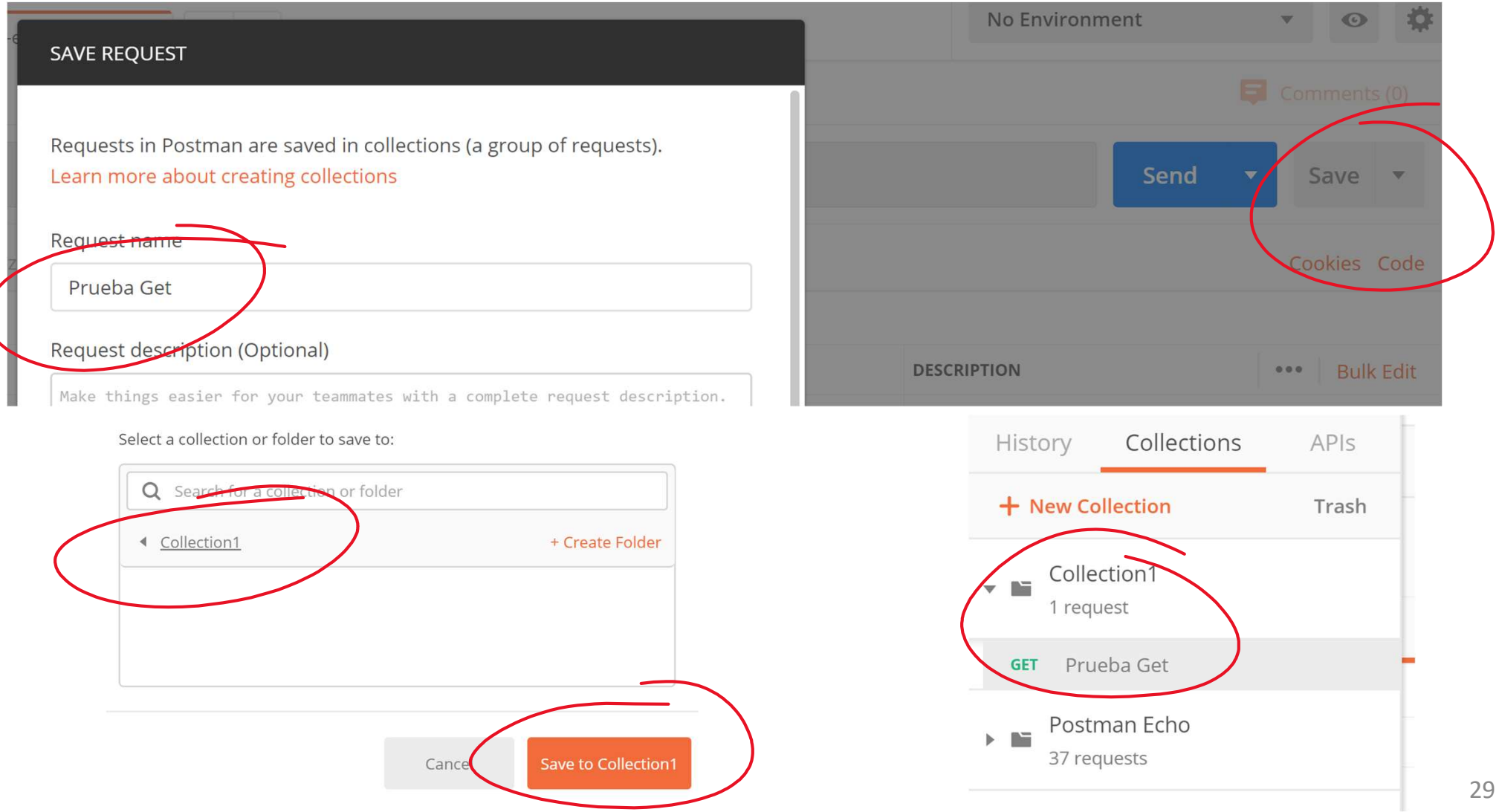
Introducción

- Hacemos clic en Send y vemos que el servicio existe, y nos devuelve información:



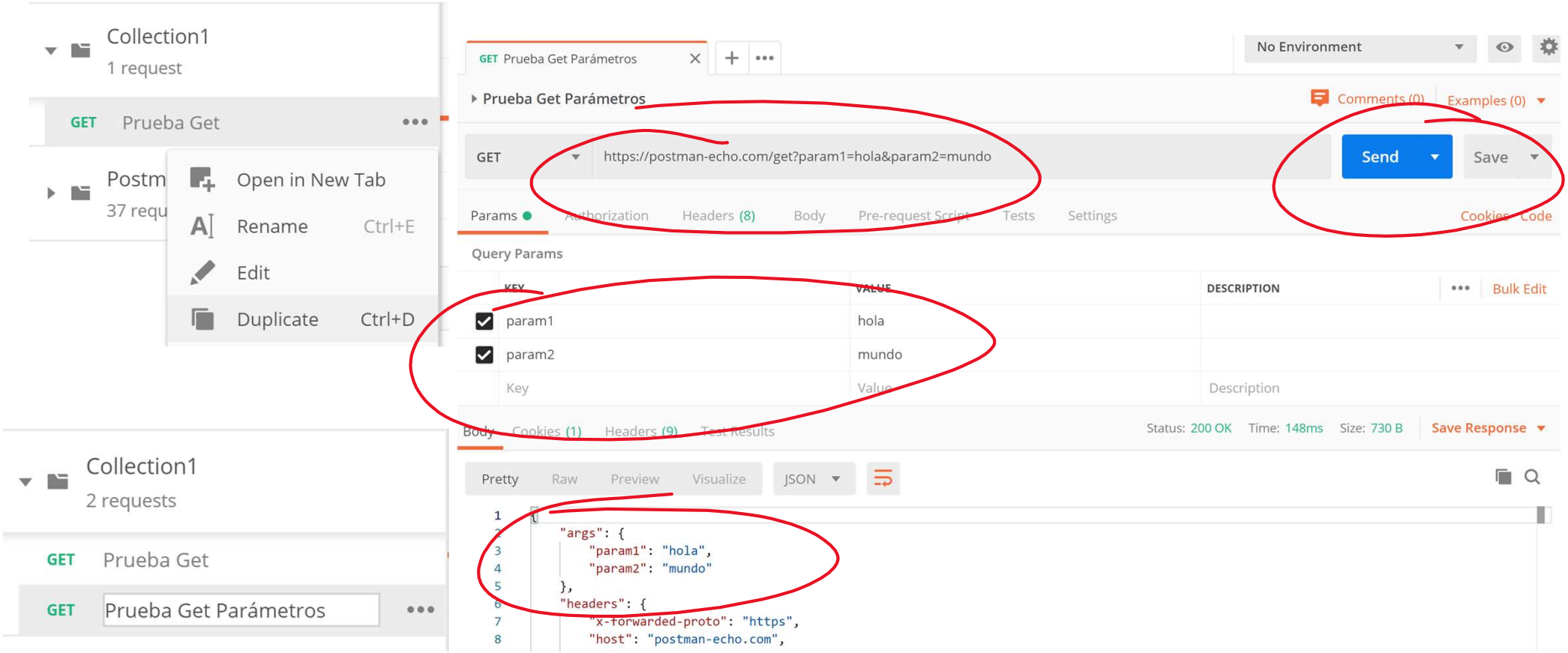
Introducción

- Podemos guardar esta primera petición en nuestra colección:



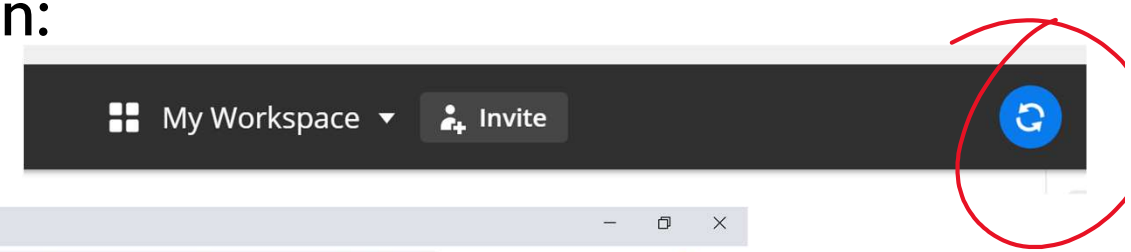
Introducción

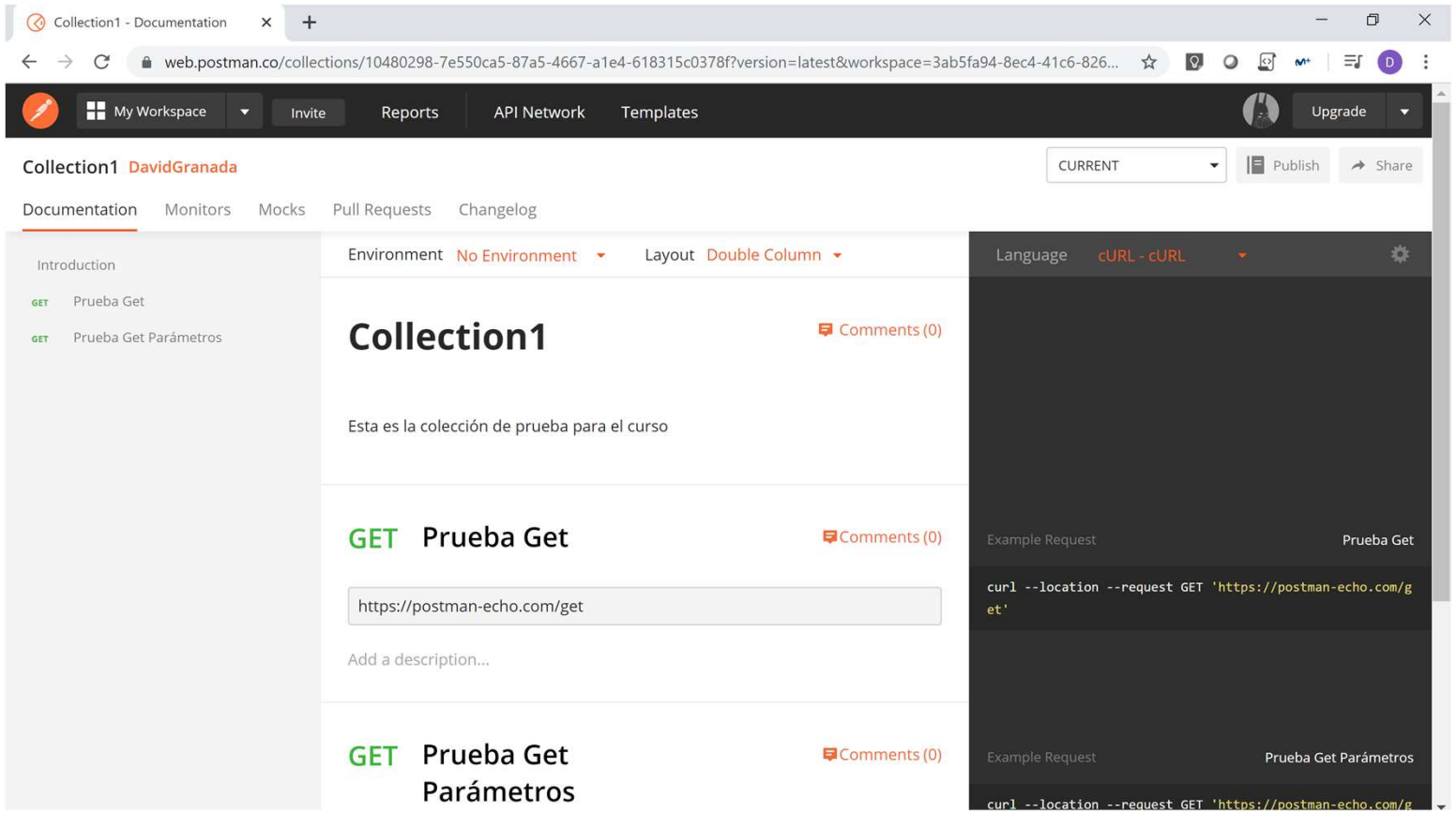
- Duplicamos esa petición para incluir algunos parámetros:



Introducción

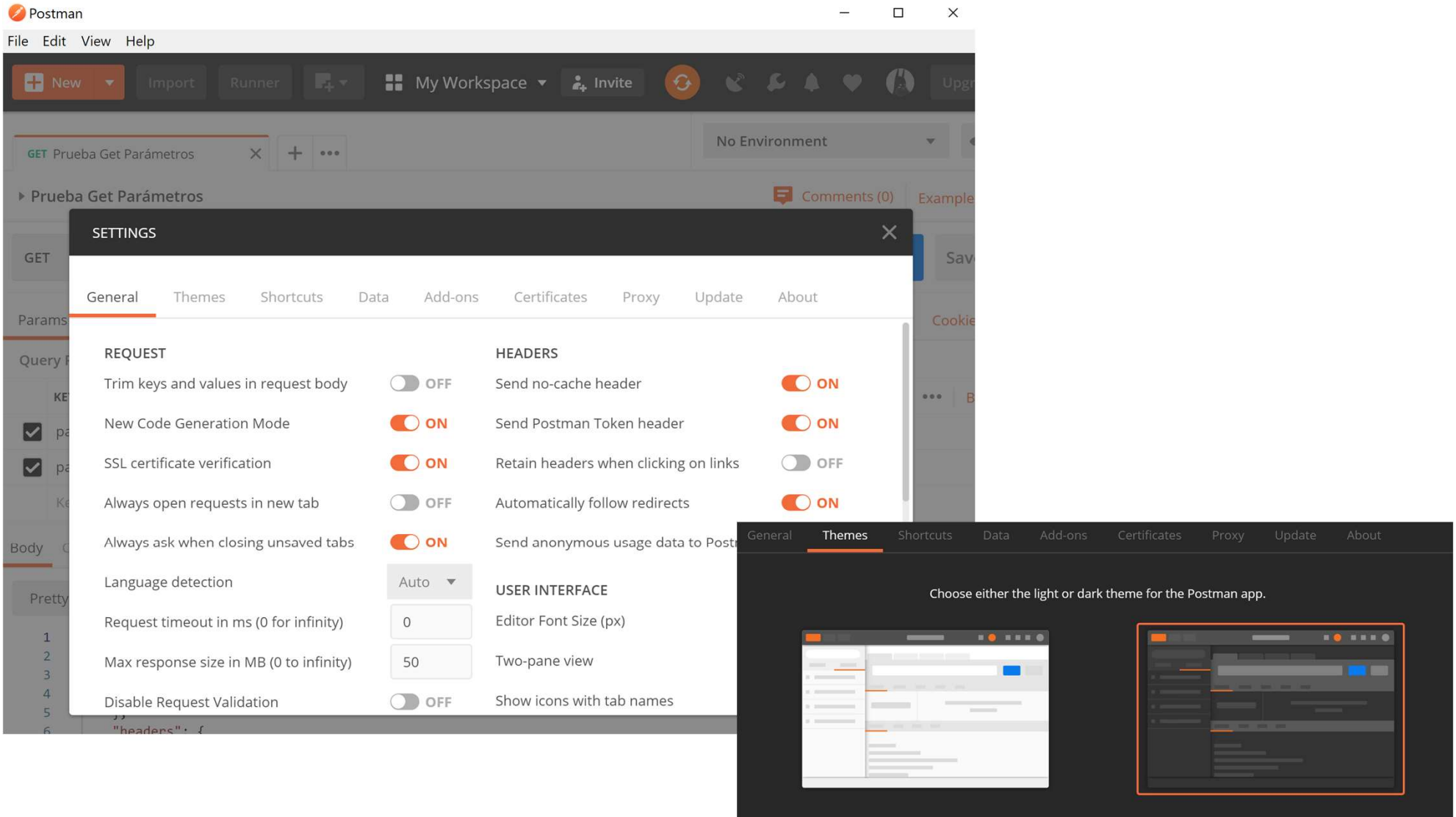
- Podemos sincronizar nuestro trabajo para poder consultarlo en la versión online de Postman:





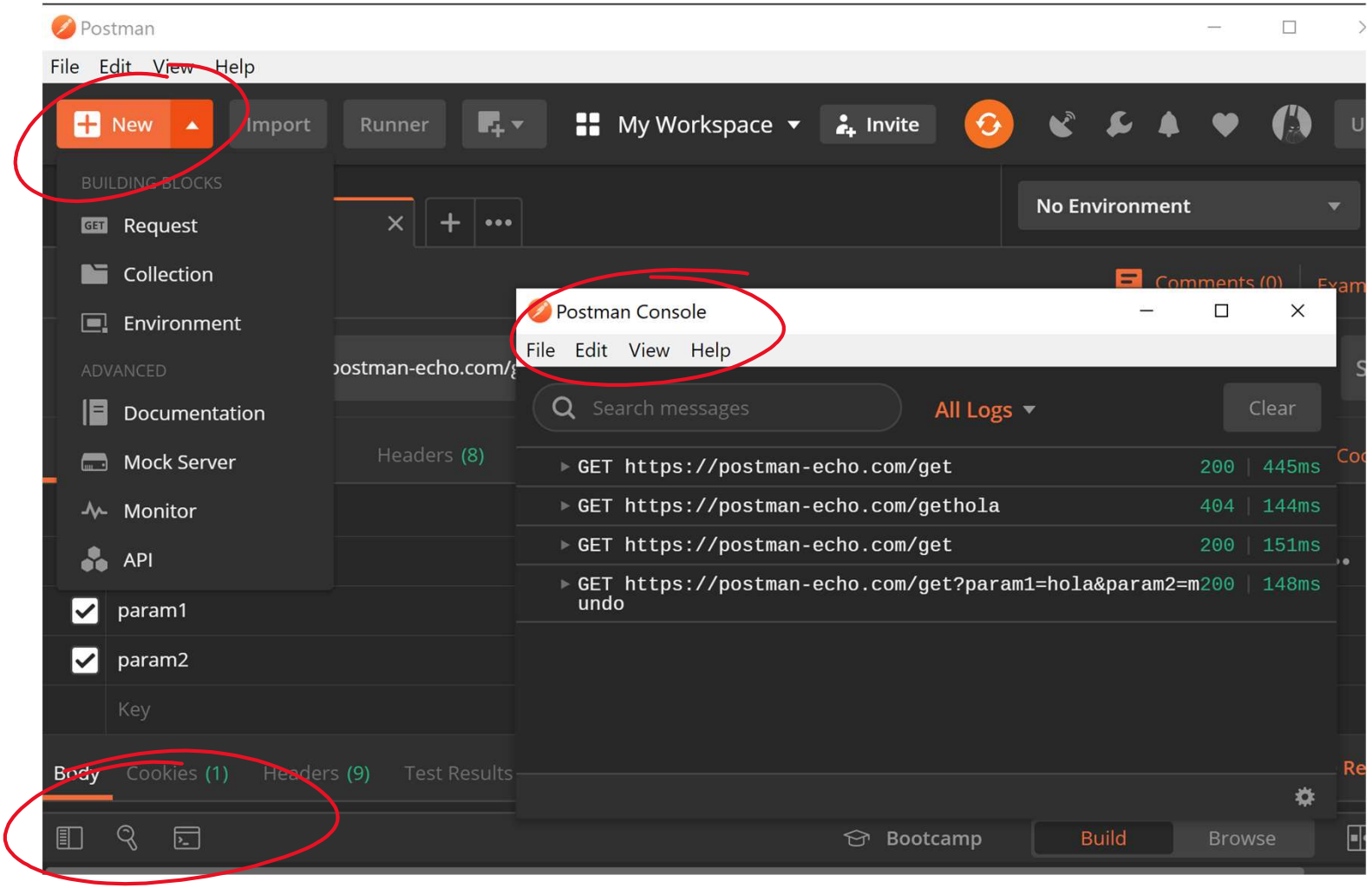
Introducción

- Verificamos los settings de la aplicación (podemos cambiar el theme):



Introducción

- Explorar las opciones del botón New y de los comandos inferiores (find y consola):



Peticiones a la API

API REST

- Una API REST es lo que estamos acostumbrados a hacer a la hora de realizar peticiones a alguna API desarrollada por una empresa.
- Algunos ejemplos de llamadas API REST son las típicas llamadas de GET, POST, PUT y DELETE.
- GET para consultar y leer, POST para crear, PUT para editar y DELETE para eliminar.

API REST

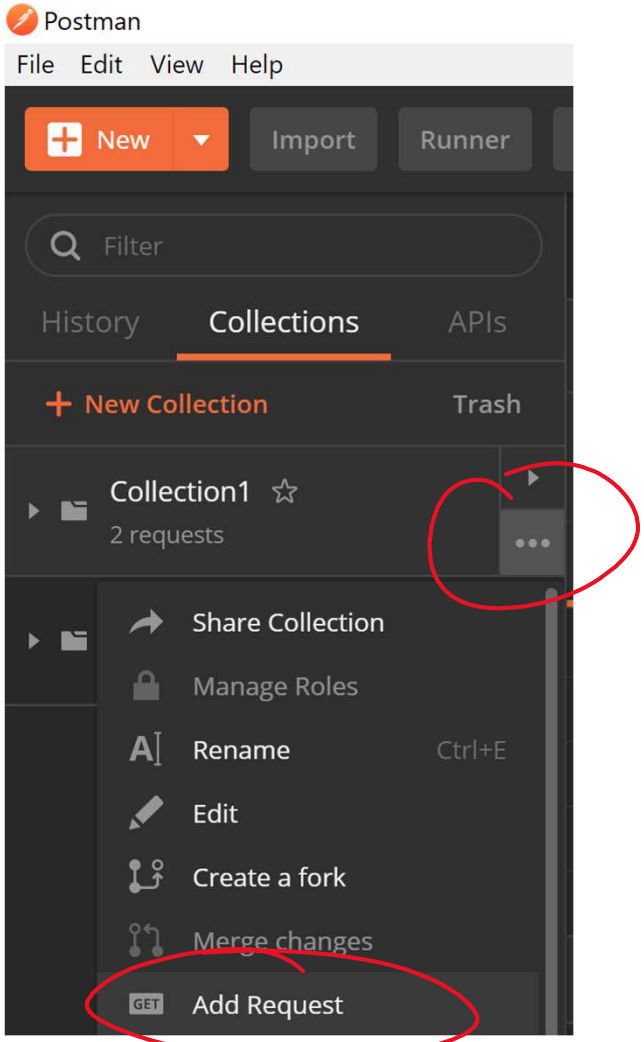
- El término REST (Representational State Transfer) se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP.
- Un servicio REST no es una arquitectura software, sino un conjunto de restricciones con las que podemos crear un estilo de arquitectura software, la cual podremos usar para crear aplicaciones web respetando HTTP.

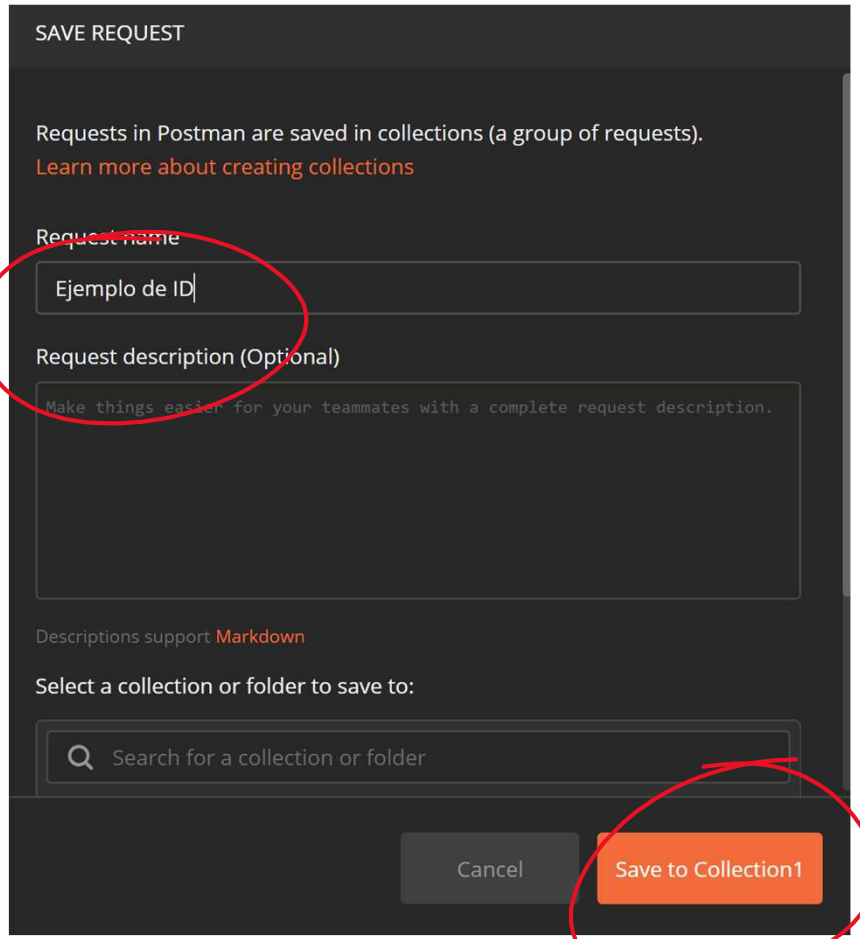
API REST

- Hoy en día la mayoría de las empresas utilizan API REST para crear servicios.
- Esto se debe a que es un estándar lógico y eficiente para la creación de servicios web.
- Por poner algún ejemplo tenemos los sistemas de identificación de Facebook, la autenticación en los servicios de Google (hojas de cálculo, Google Analytics, …).

Ejemplo GET

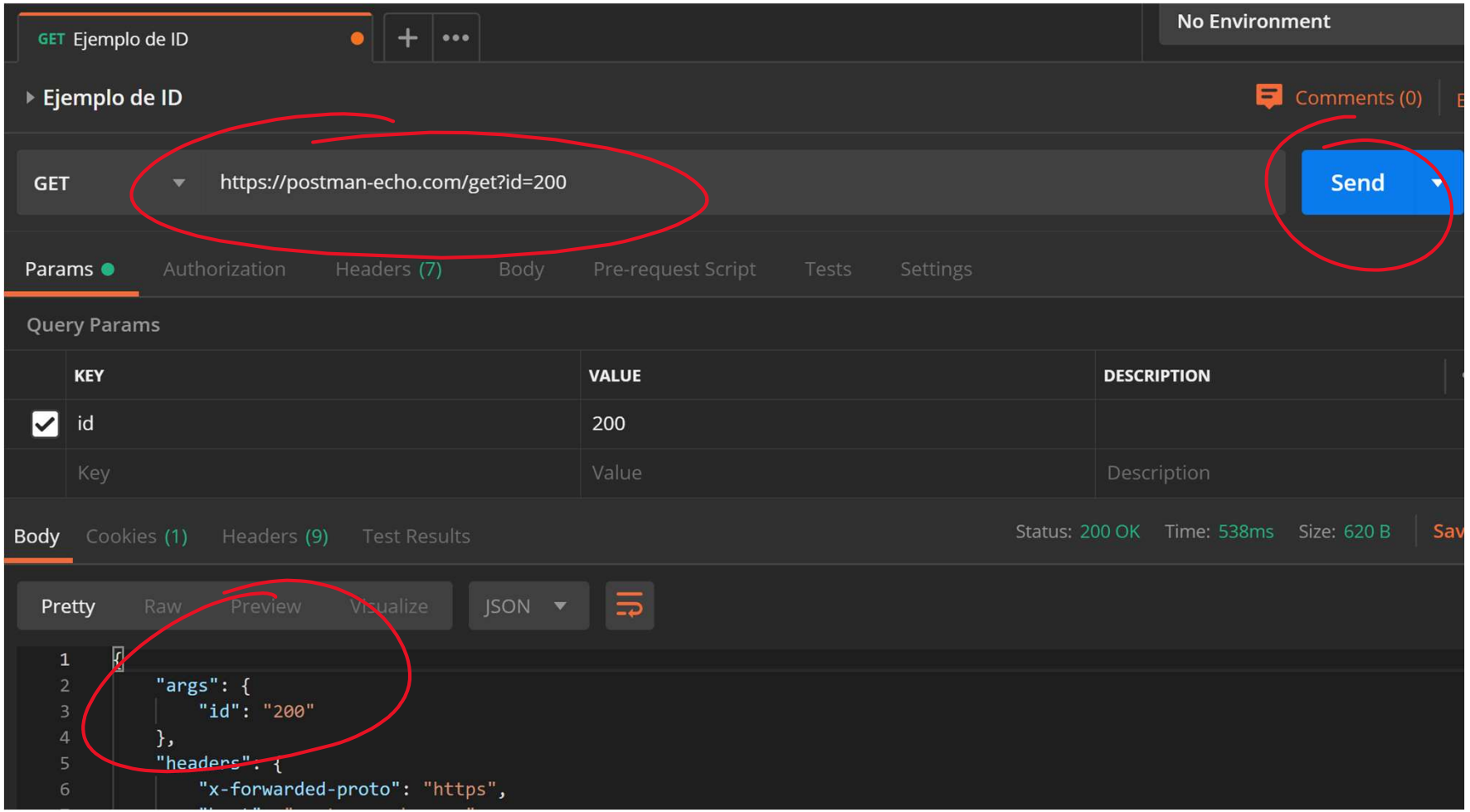
- Vamos a añadir una nueva petición:





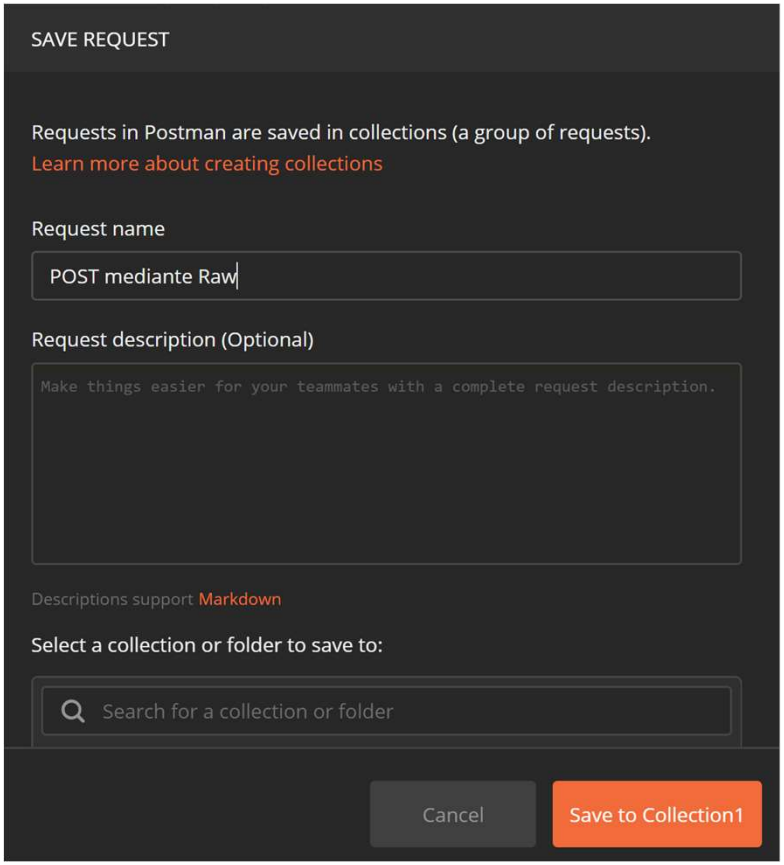
Ejemplo GET

- Las peticiones de tipo GET son para obtener datos, habitualmente van acompañado de parámetros después del símbolo "?"



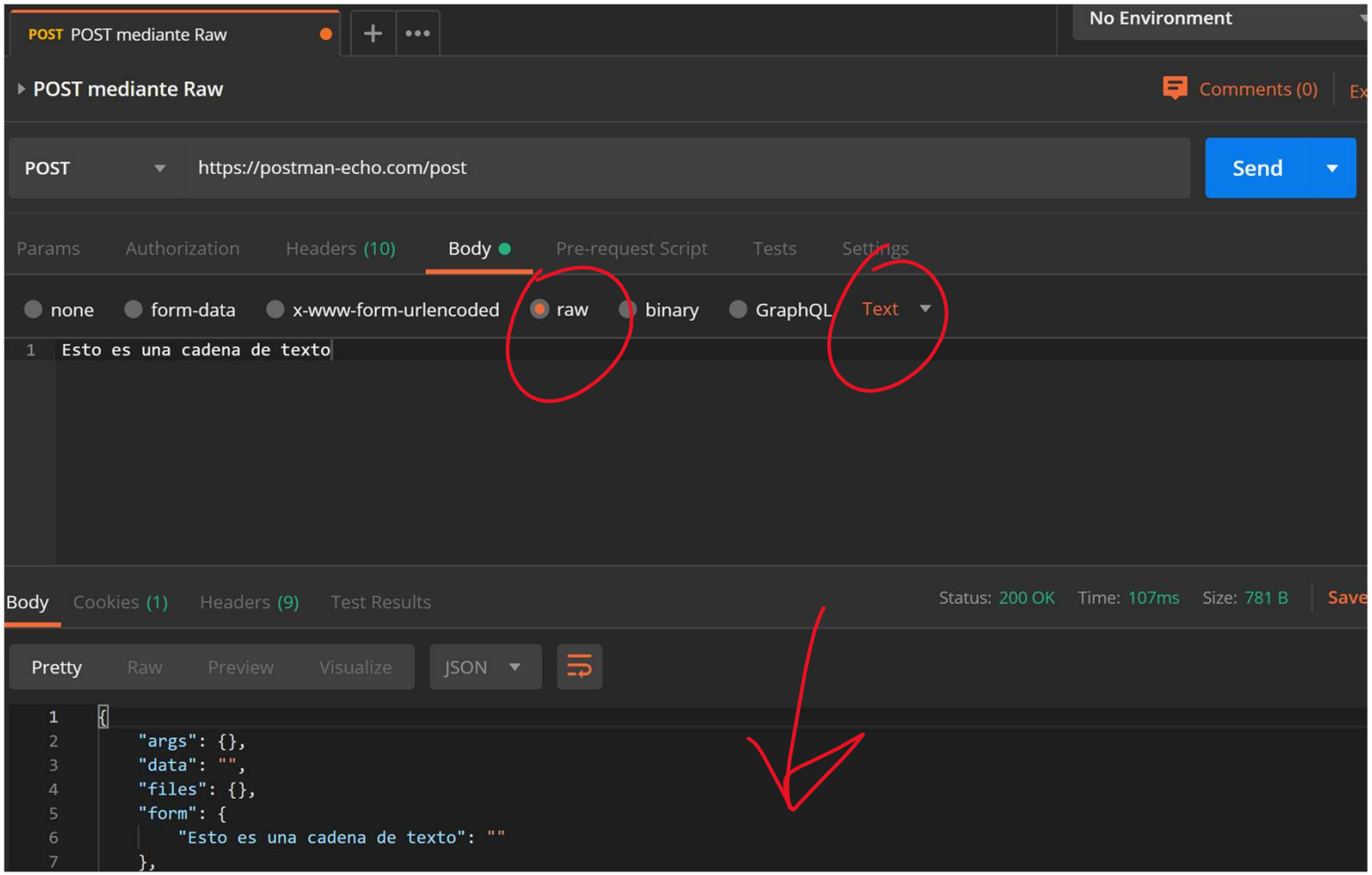
Ejemplo POST

- Para enviar información al servidor y añadir datos a una BBDD usamos POST
- Creamos una nueva petición en la colección:



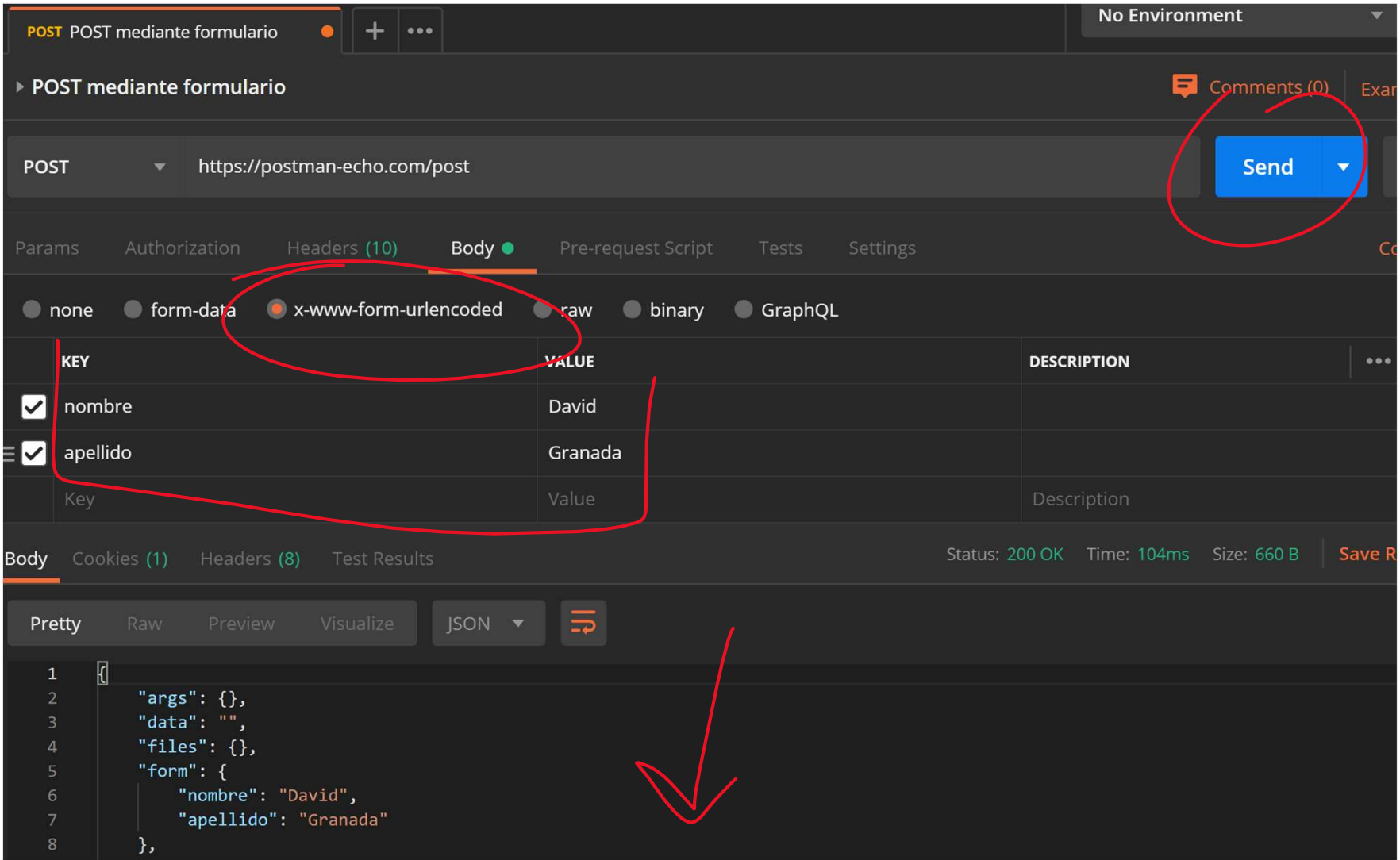
Ejemplo POST

- En las peticiones POST enviamos los datos mediante el Body para que no puedan ser interceptados, en este caso un texto raw:



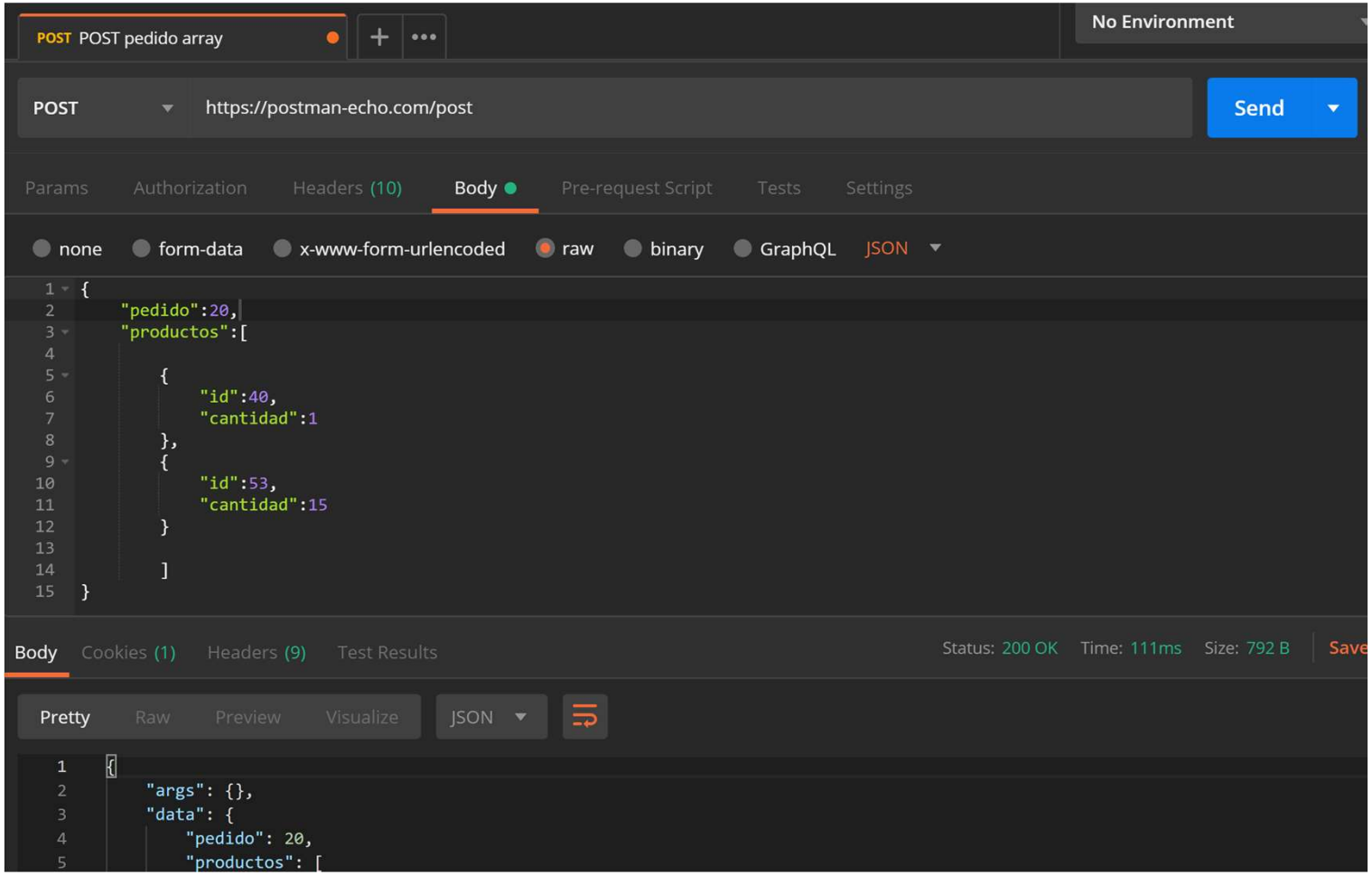
Ejemplo POST

- Podemos enviar datos mediante un formulario:



ARRAY mediante POST

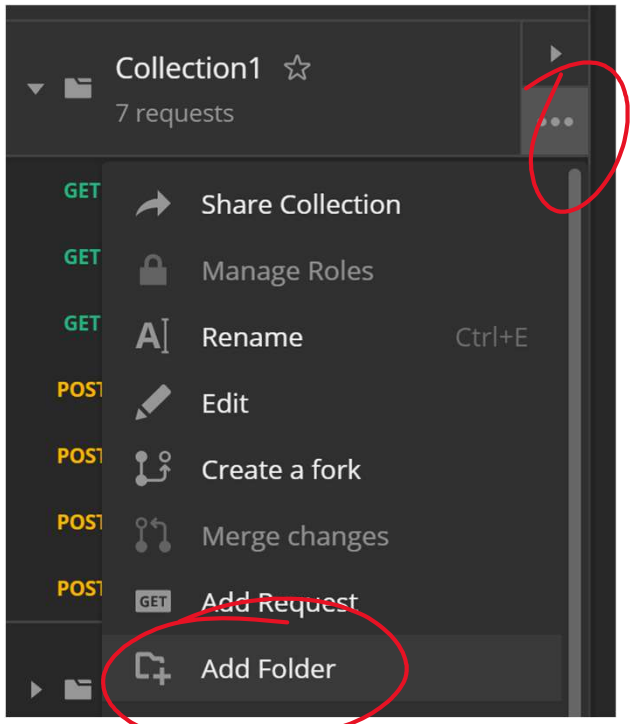
- Cuando se realiza un pedido en una tienda online, habitualmente se envía un array de datos:

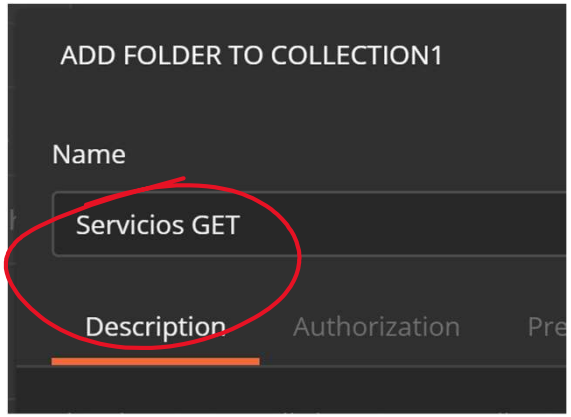


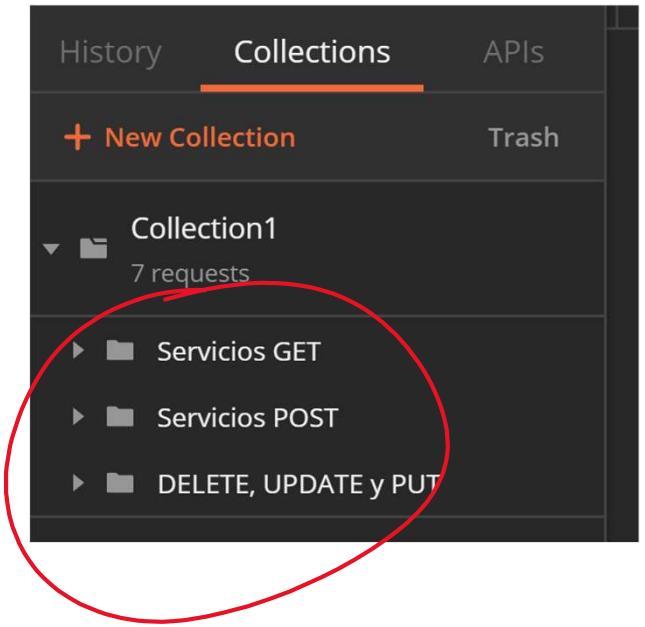
Probemos a quitar alguna coma y a enviar
Mirar el JSON generado

Creación de carpetas

- Dentro de la colección podemos crear carpetas para organizar mejor nuestras peticiones (arrastramos las peticiones a las carpetas):





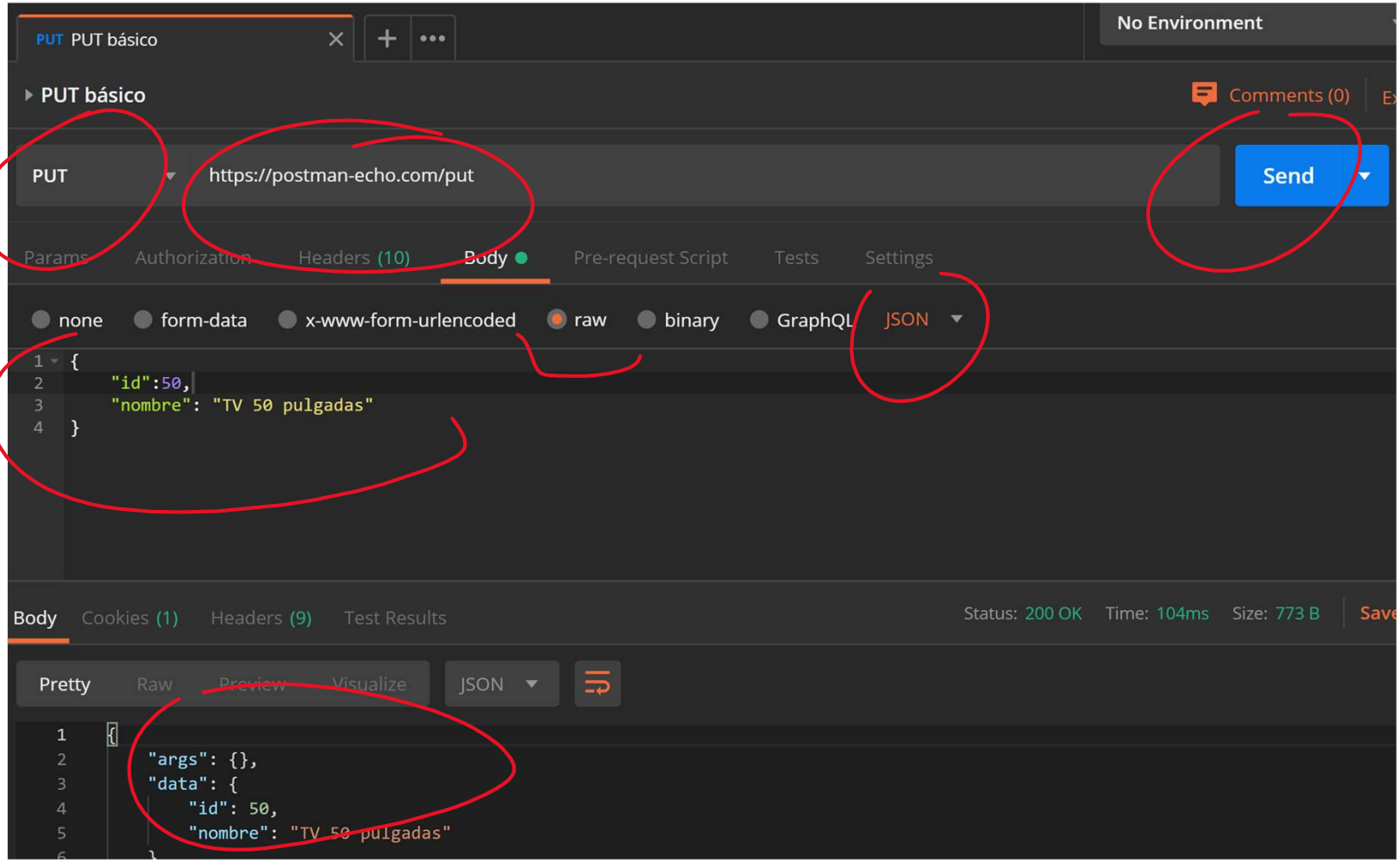


Ejemplo PUT

- Cuando queremos actualizar un registro de la BBDD podemos utilizar las peticiones de tipo PUT
- Al igual que POST, los datos se envían mediante un body para evitar que sean interceptados

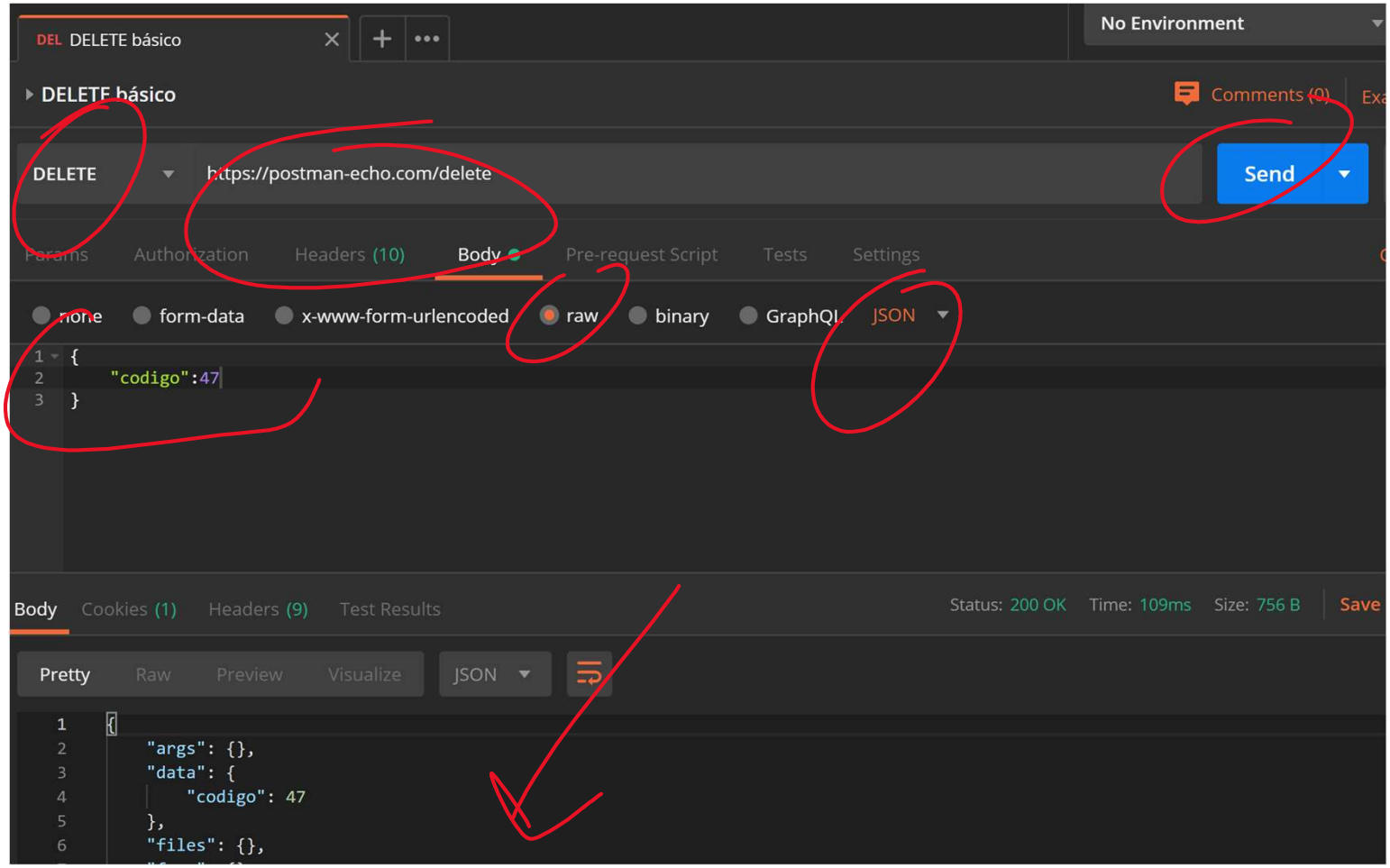
Ejemplo PUT

- Por ejemplo queremos cambiar el nombre de un producto en nuestra base de datos:



Ejemplo DELETE

- Para borrar algún registro de nuestra BBDD podemos usar la petición de tipo DELETE:

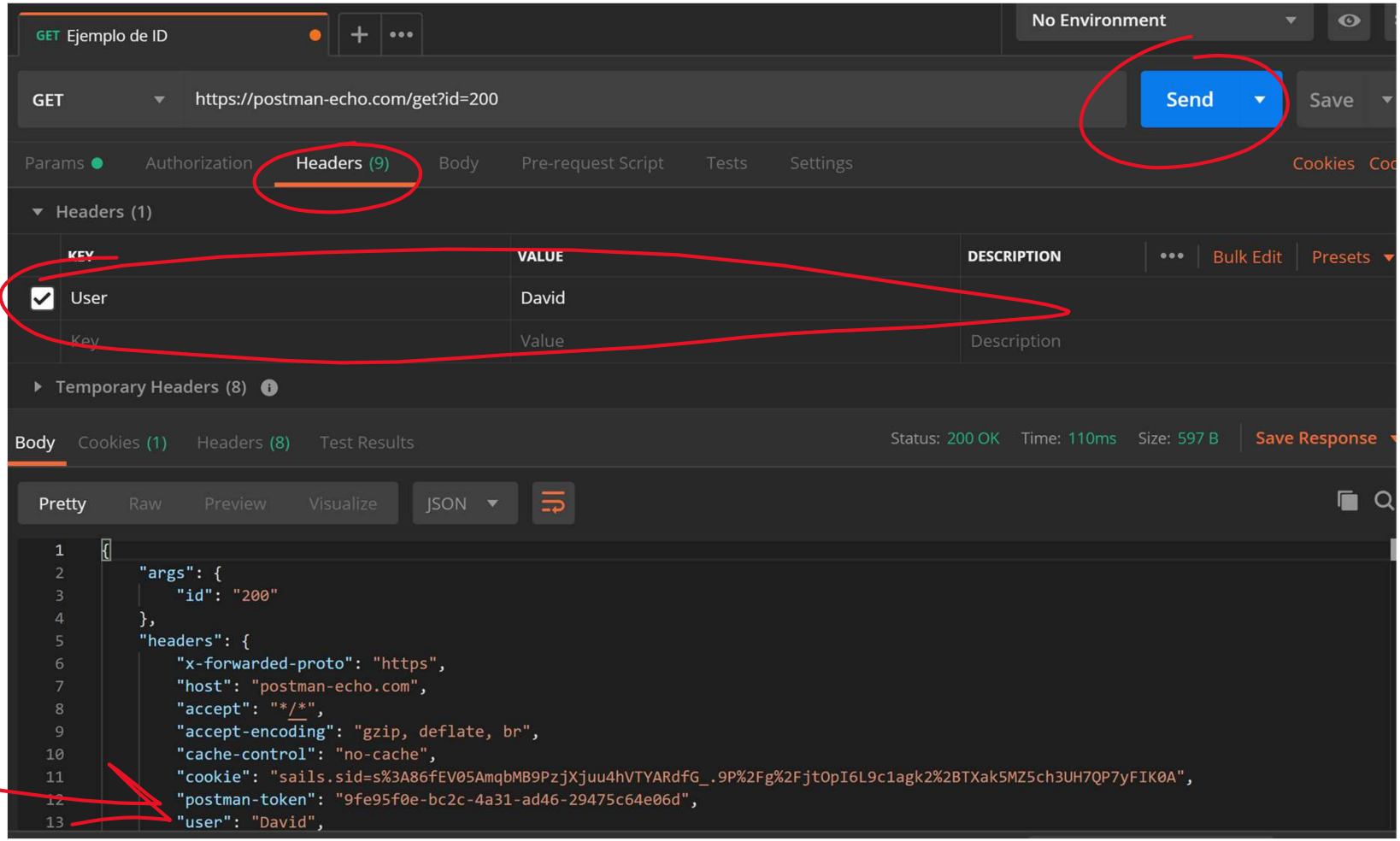


HEADER

- Más allá de enviar datos por la URL o por el Body, podemos enviar datos por la cabecera (HEADER), la cual suele contener información especial relacionada con diferentes aspectos de mi servicios web

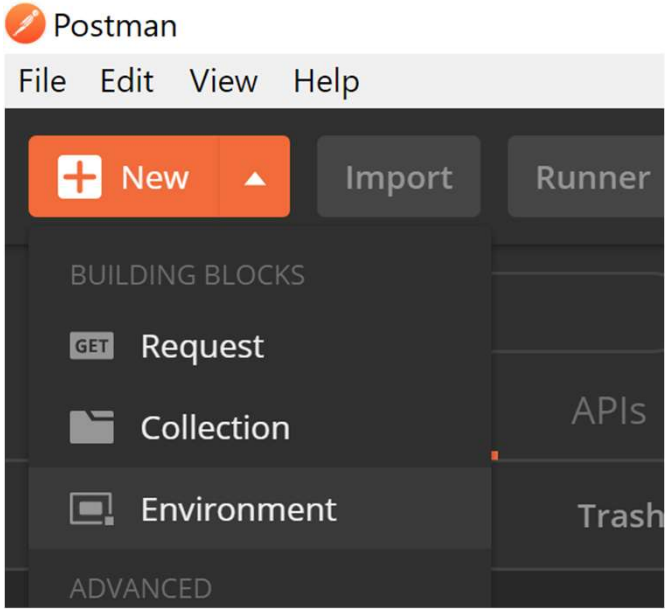
HEADER

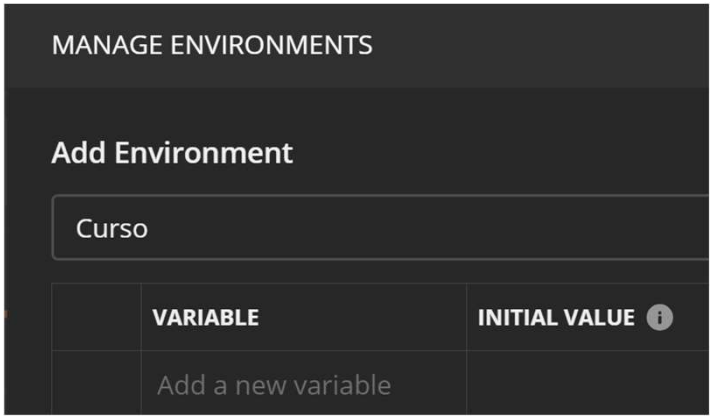
- Abramos la petición GET “Ejemplo de ID” y añadimos un header:

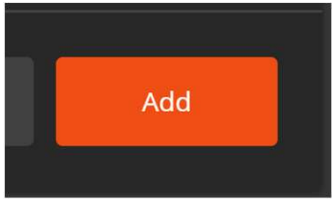


Variables de Entorno

- Podemos crear entornos para organizar nuestro trabajo (sería como una especie de contenedor de nuestras colecciones y de nuestras variables de entorno):

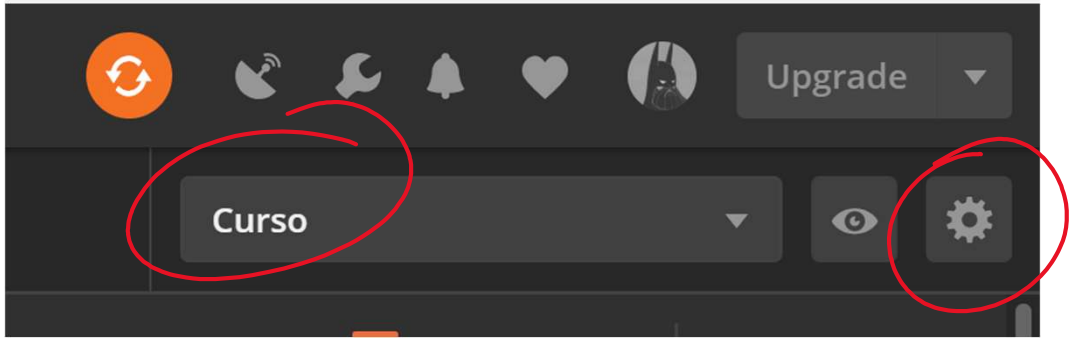


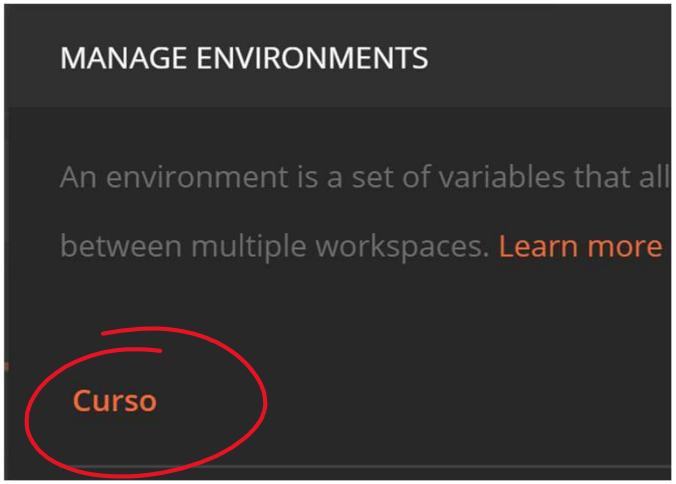


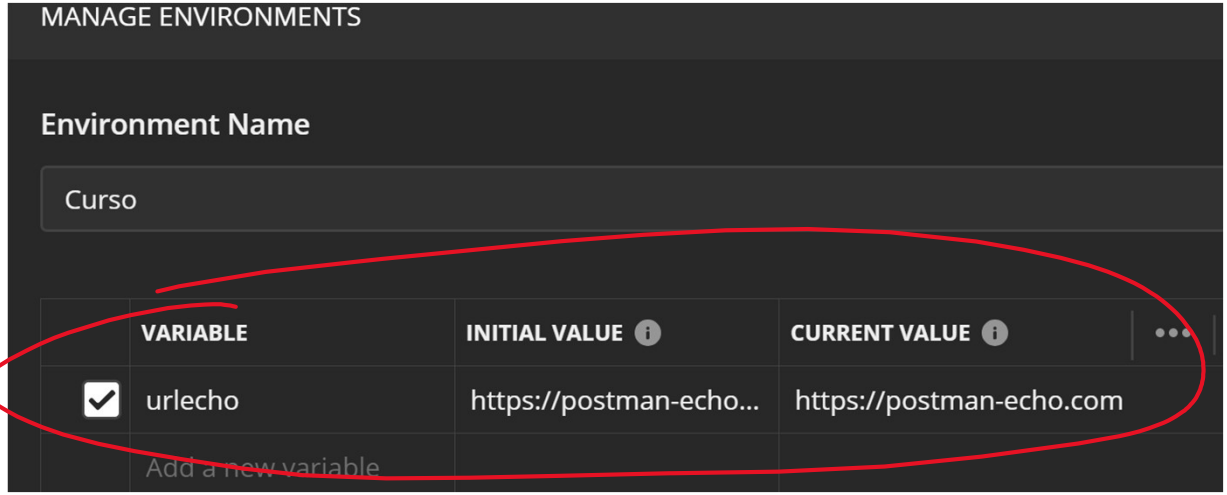


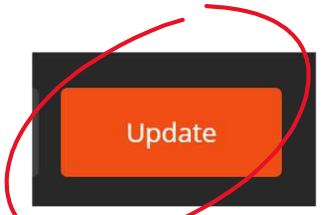
Variables de Entorno

- Una vez creado el entorno vamos a su gestión para crear variables válidas para dicho entorno



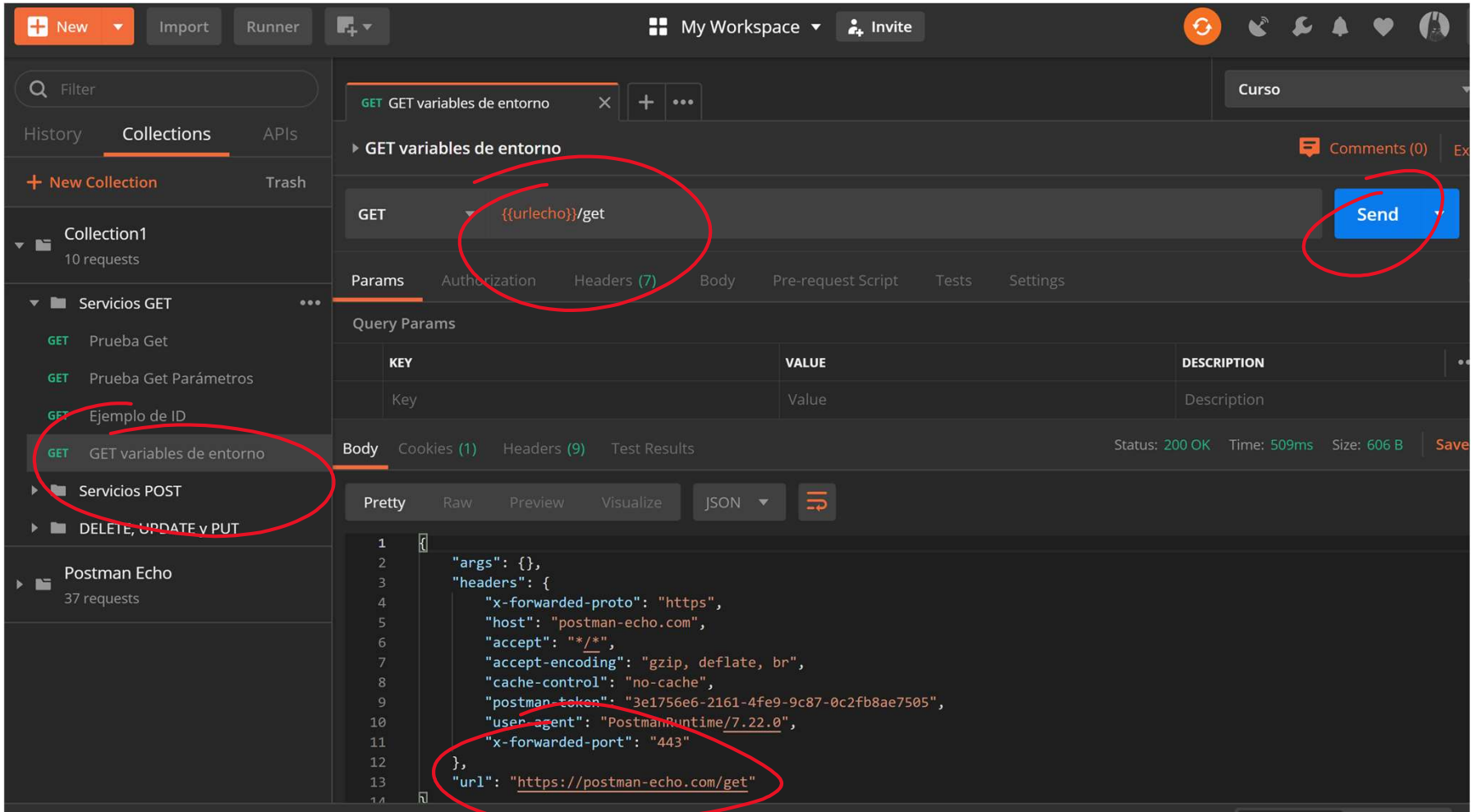


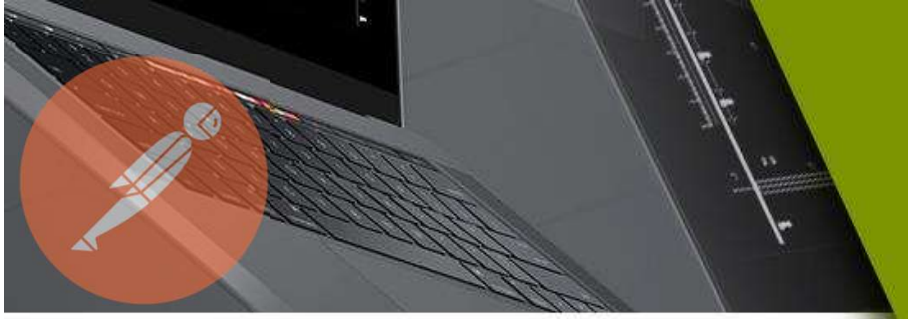




Variables de Entorno

- Ahora podemos utilizar dicha variable para la prueba de nuestros servicios web:



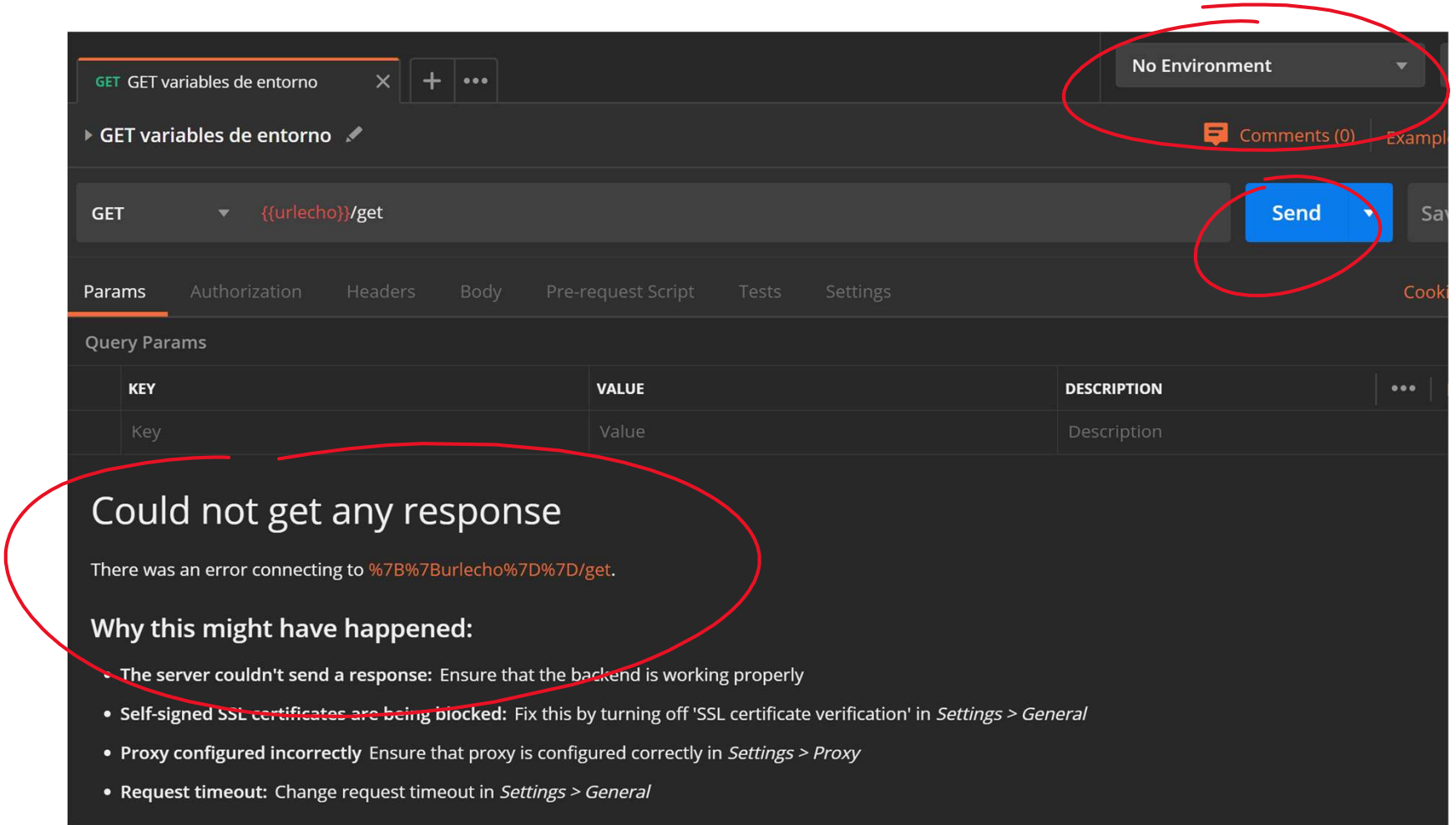


Variables de Entorno

- De esta forma, cuando pasemos a probar con nuestro servidor real, tendremos que cambiar exclusivamente nuestra variable de entorno y no realizar dicho cambio en todas las peticiones

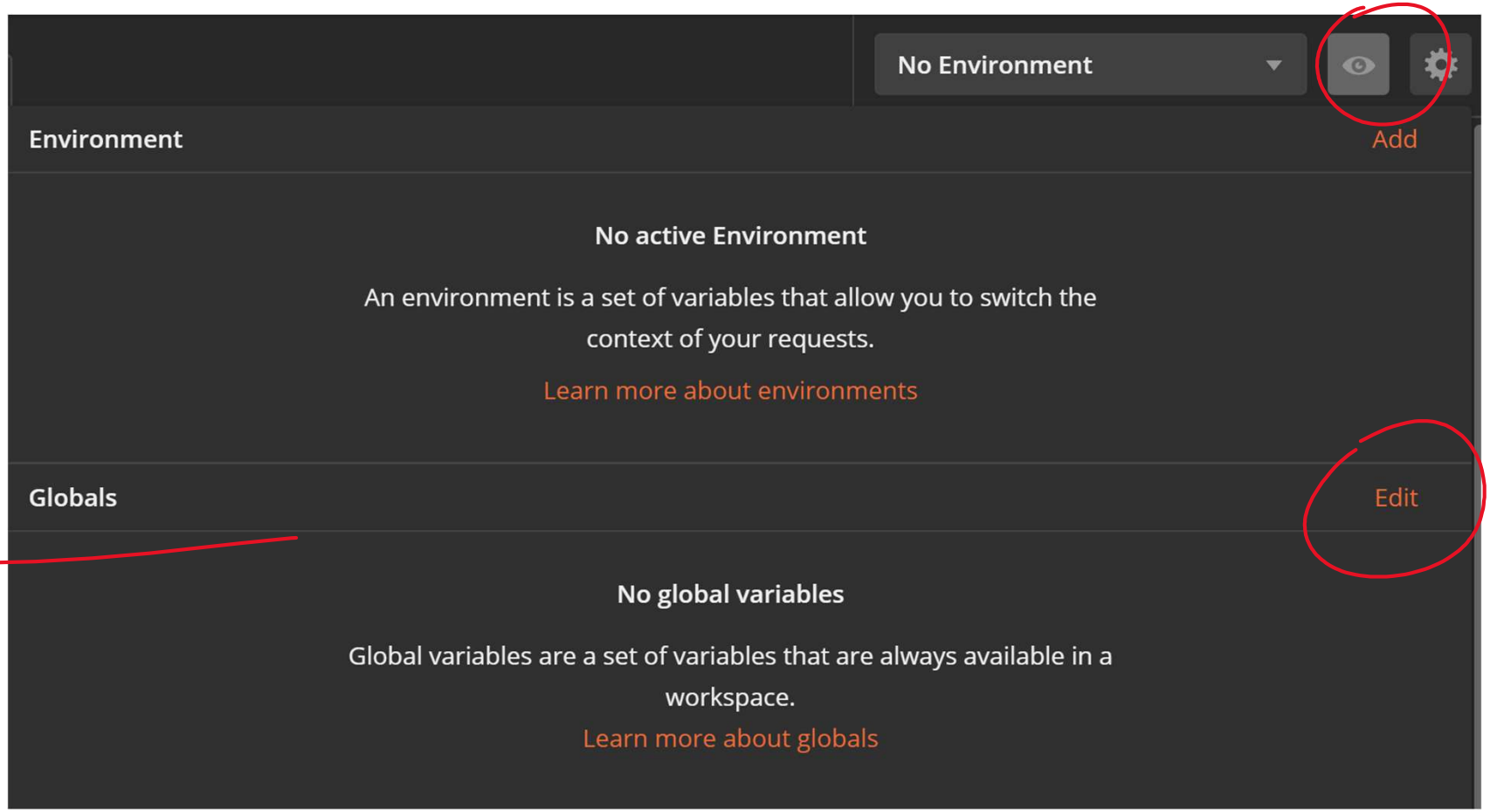
Variables Globales

- Si cambiamos de entorno y probamos la anterior petición, obtendríamos un fallo, al no reconocer dicha variable:



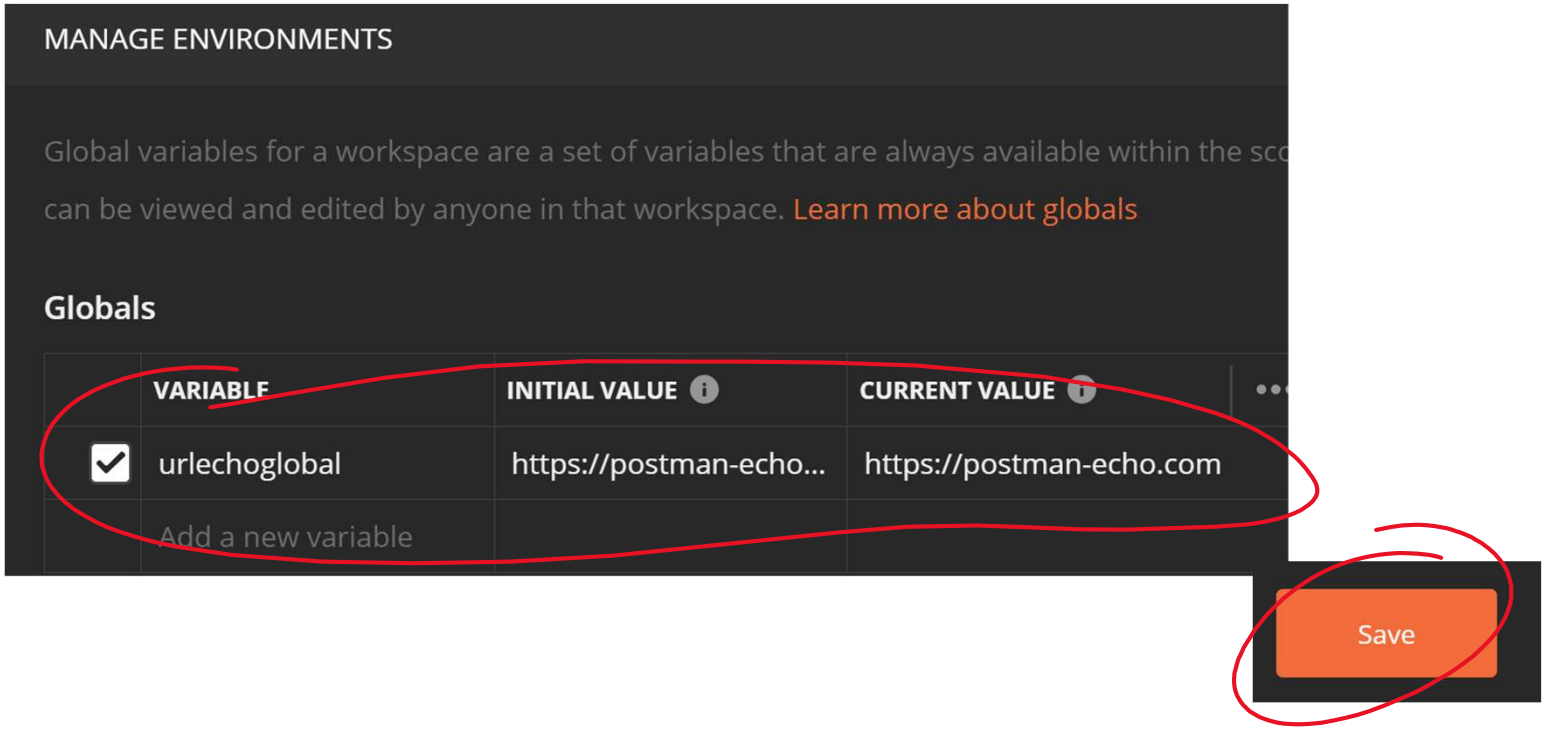
Variables Globales

- Podemos crear variables válidas para todos los entornos (globales):



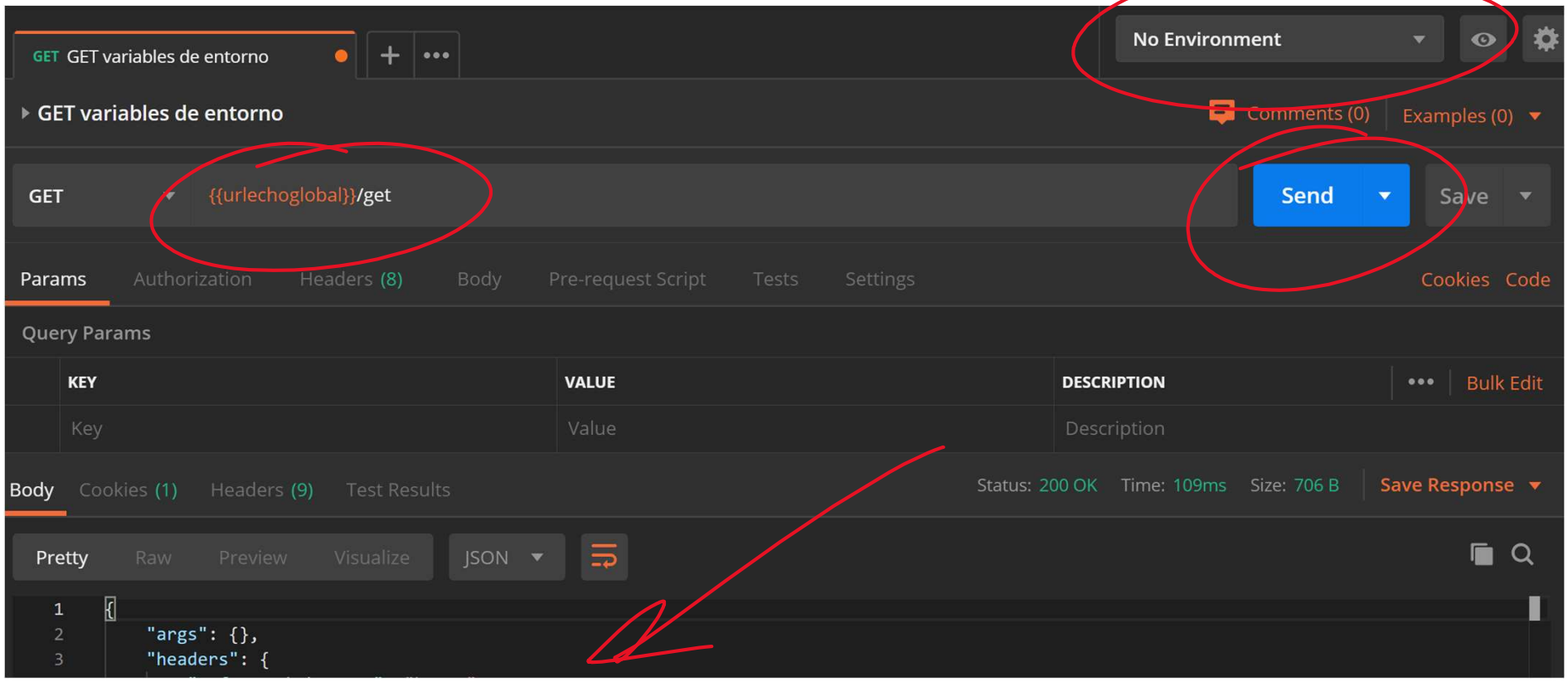
Variables Globales

- Creamos la variable global:



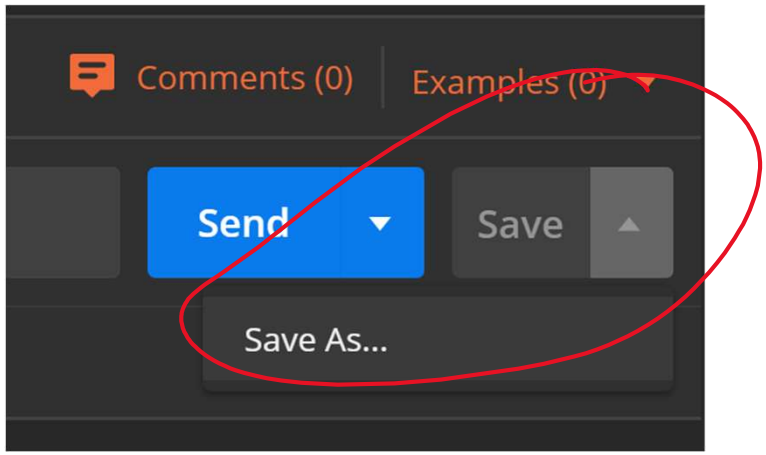
Variables Globales

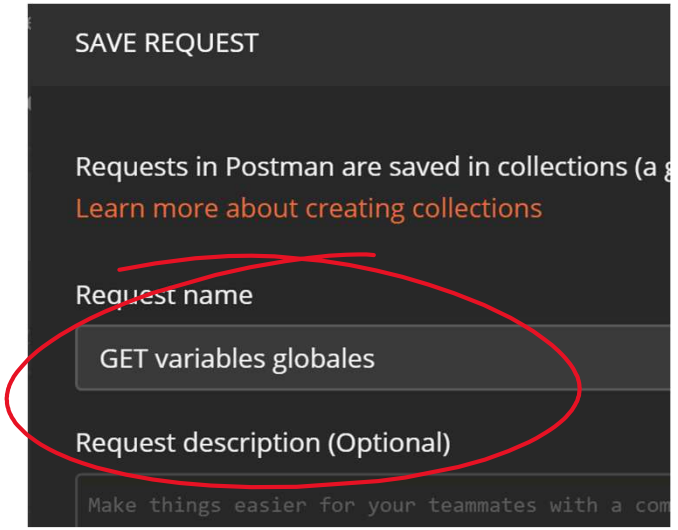
- Ahora podemos probar en ambos entornos y veremos que funciona:

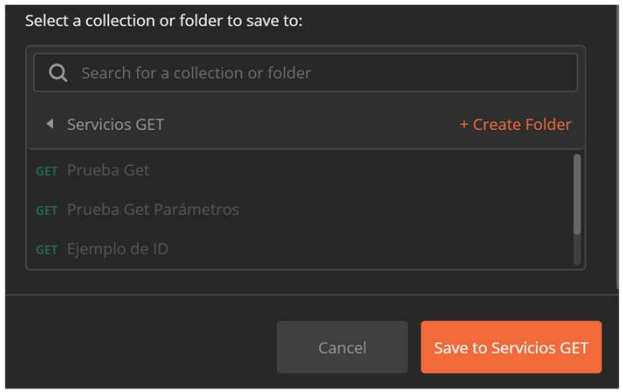


Variables Globales

- Guardamos esa petición en la Collection1 – Servicios GET:

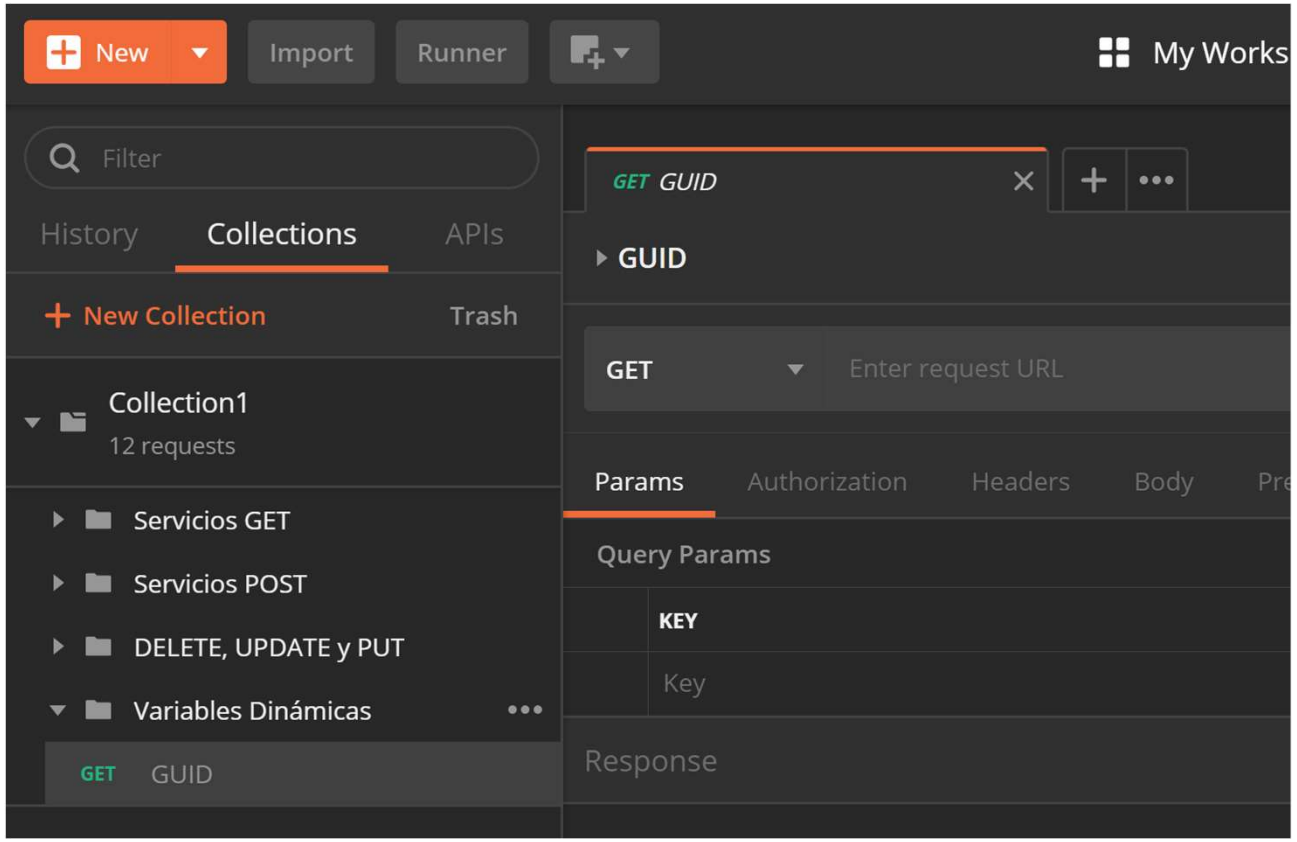






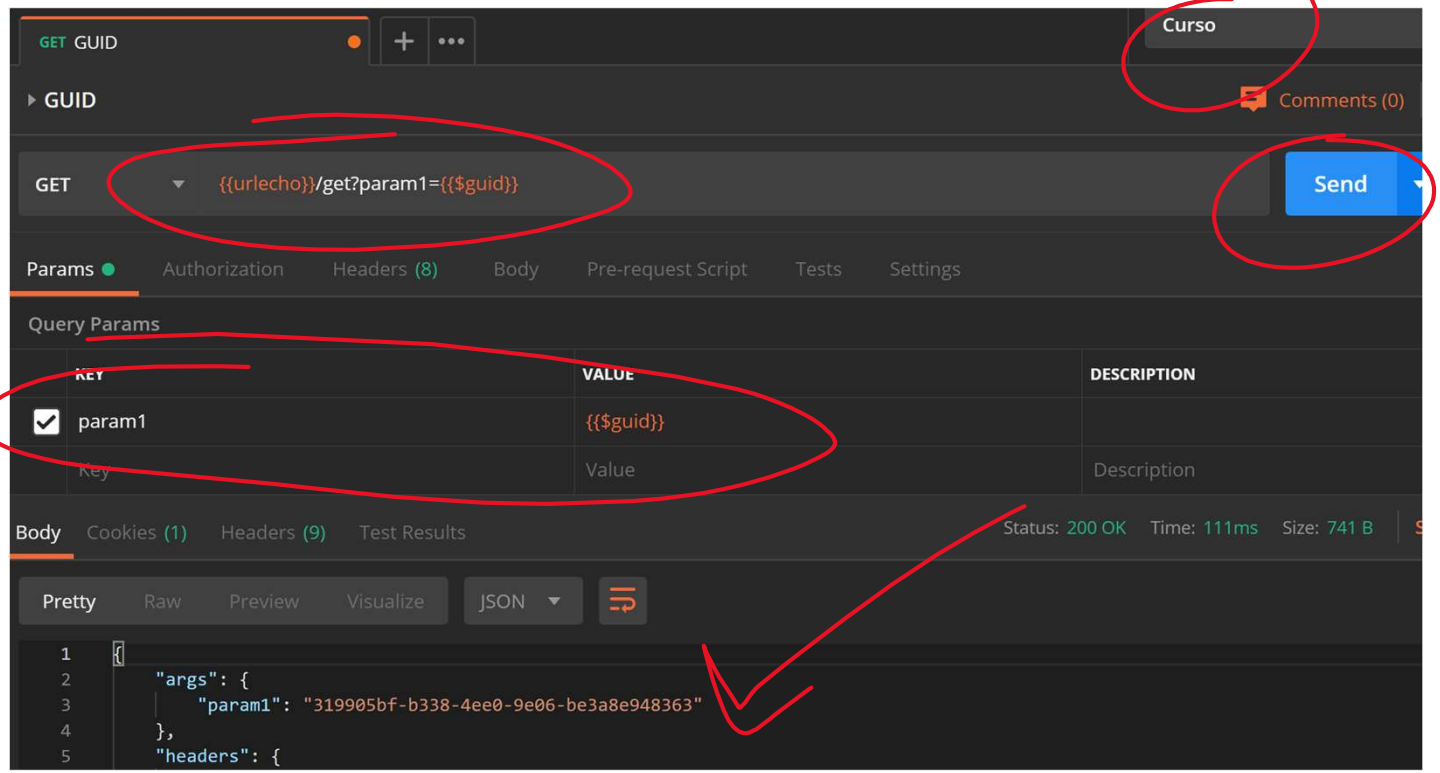
Variables Dinámicas

- Vamos a crear una nueva carpeta llamada Variables Dinámicas y dentro creamos una nueva petición llamada GUID:



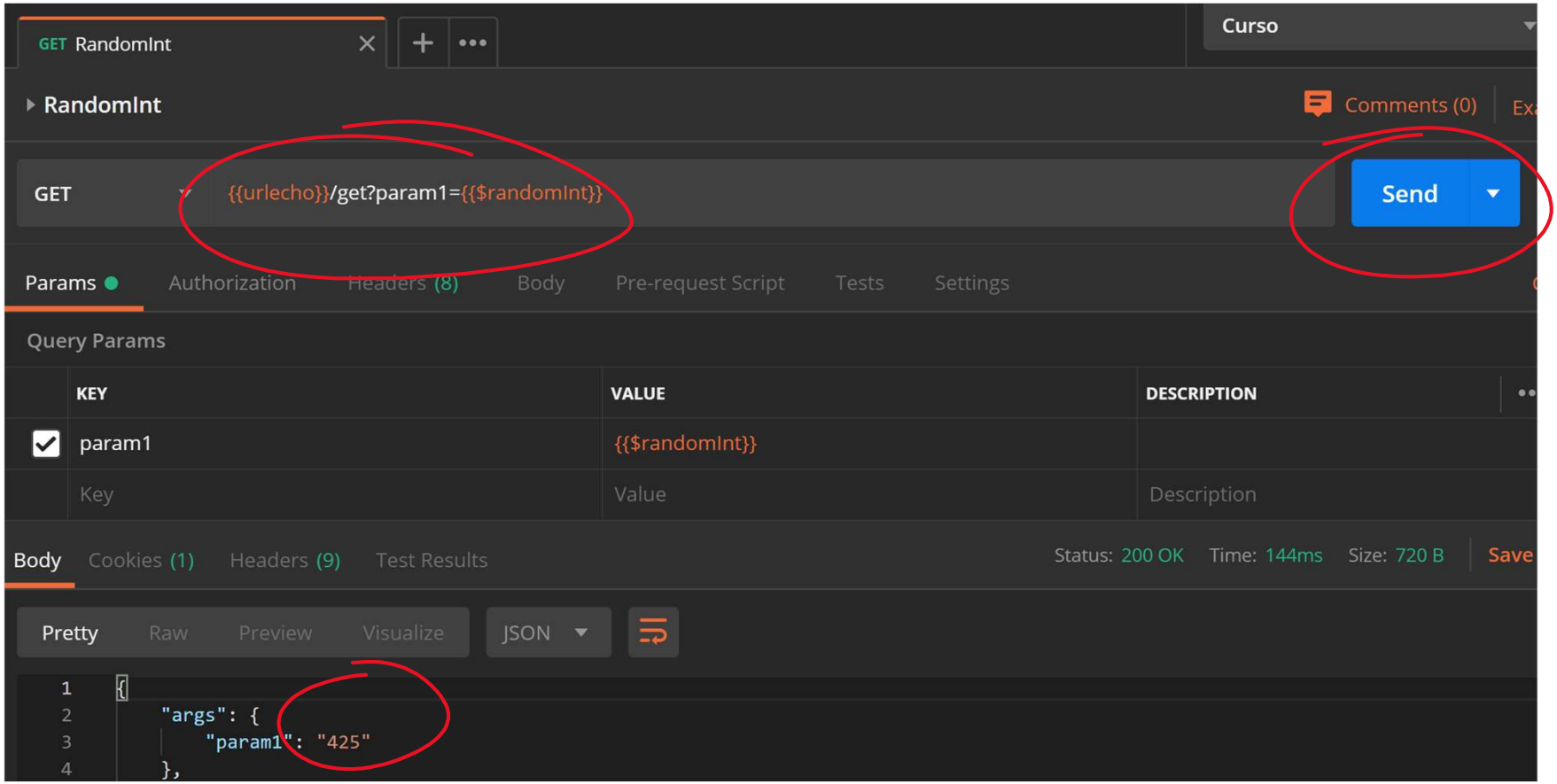
Variables Dinámicas

- Utilizamos una de las variables (dinámicas) de postman para generar una clave única cada vez que enviamos la petición (es útil para simular inserciones únicas en BBDD):



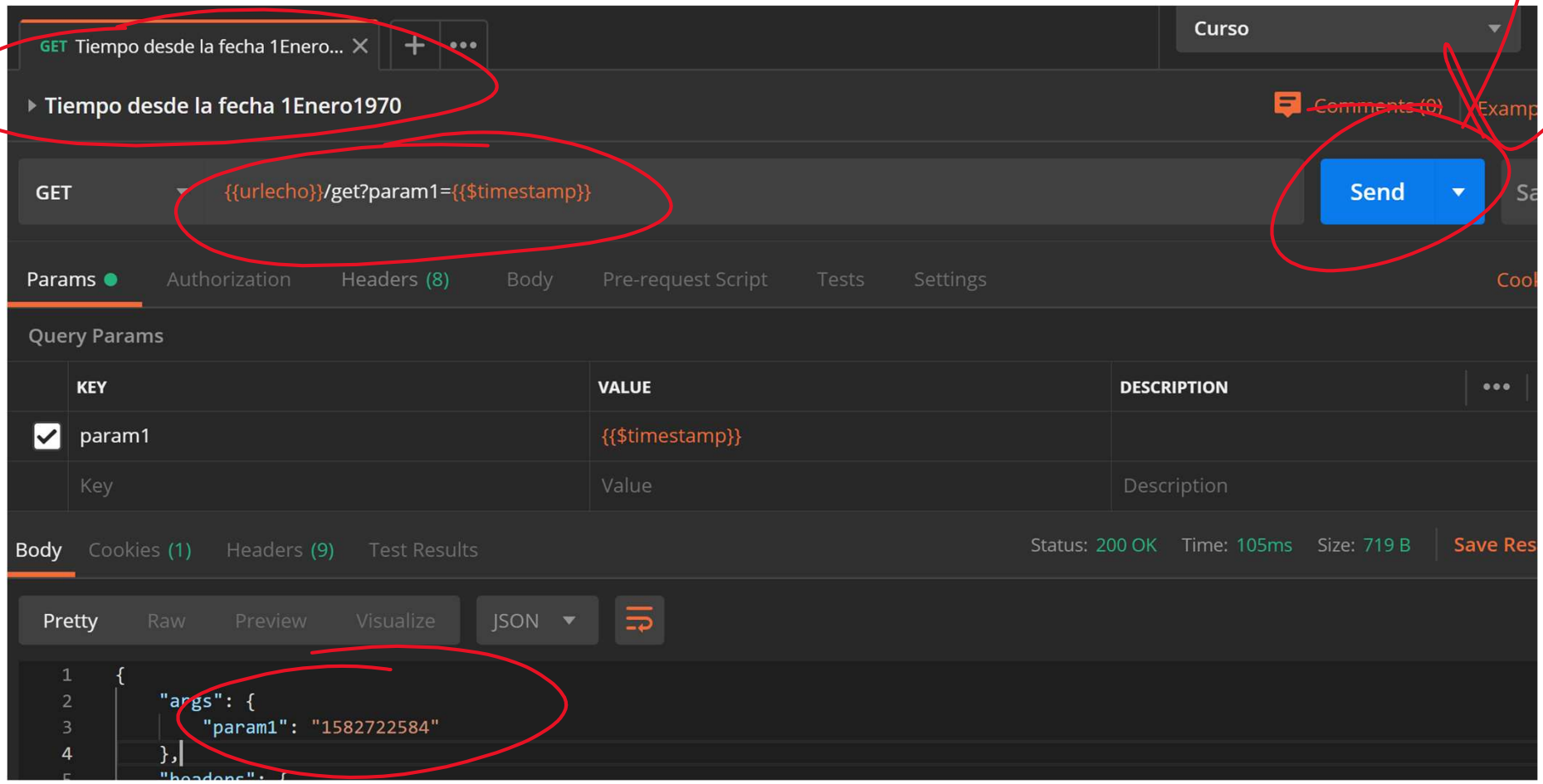
Variables Dinámicas

- Creamos una segunda petición en esta carpeta con otra variable dinámica que me genera un número aleatorio entre 0 y 1000:



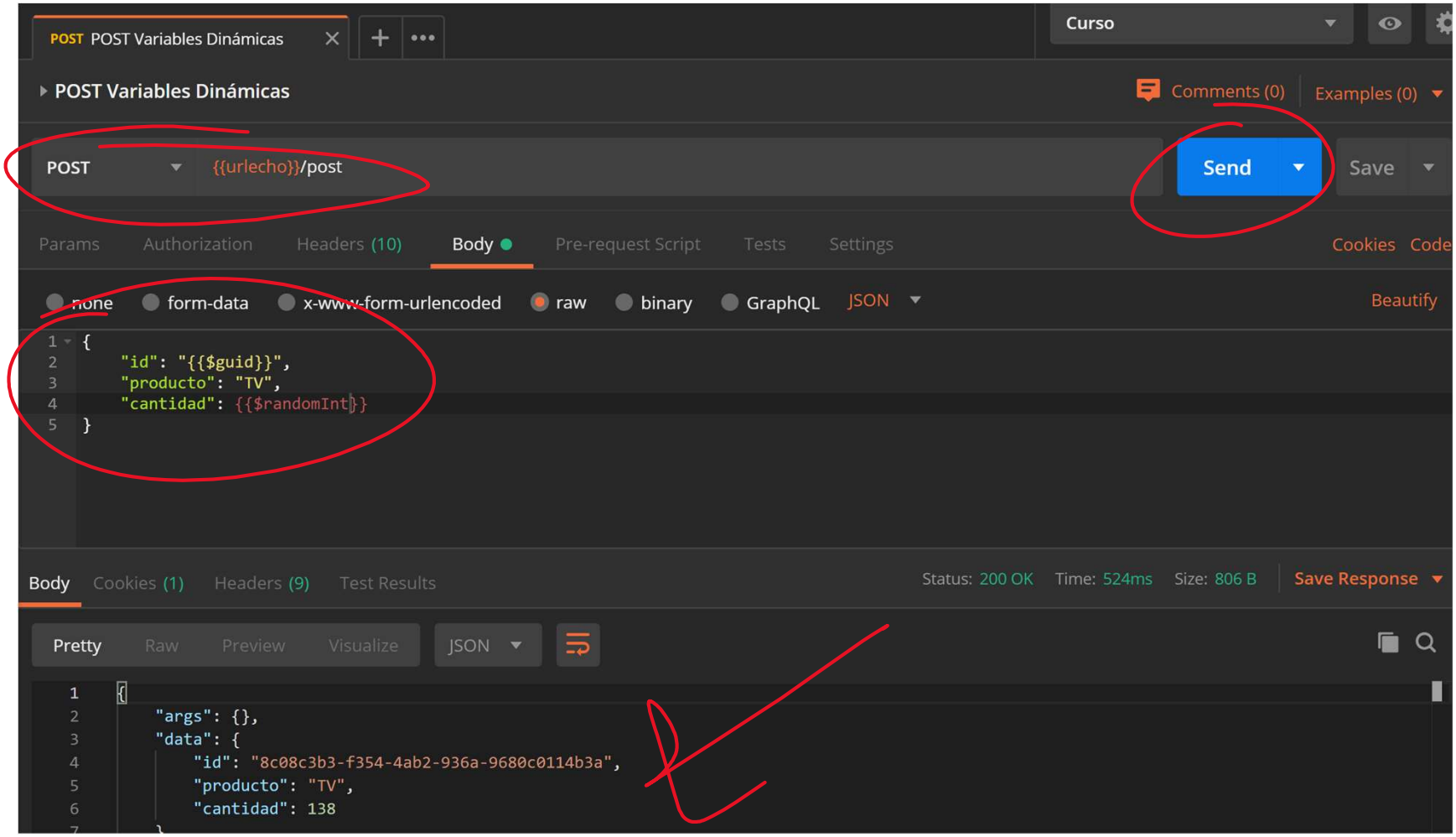
Variables Dinámicas

- Creamos otra petición para calcular tiempos entre llamadas a servicios (segundos):



Variables Dinámicas

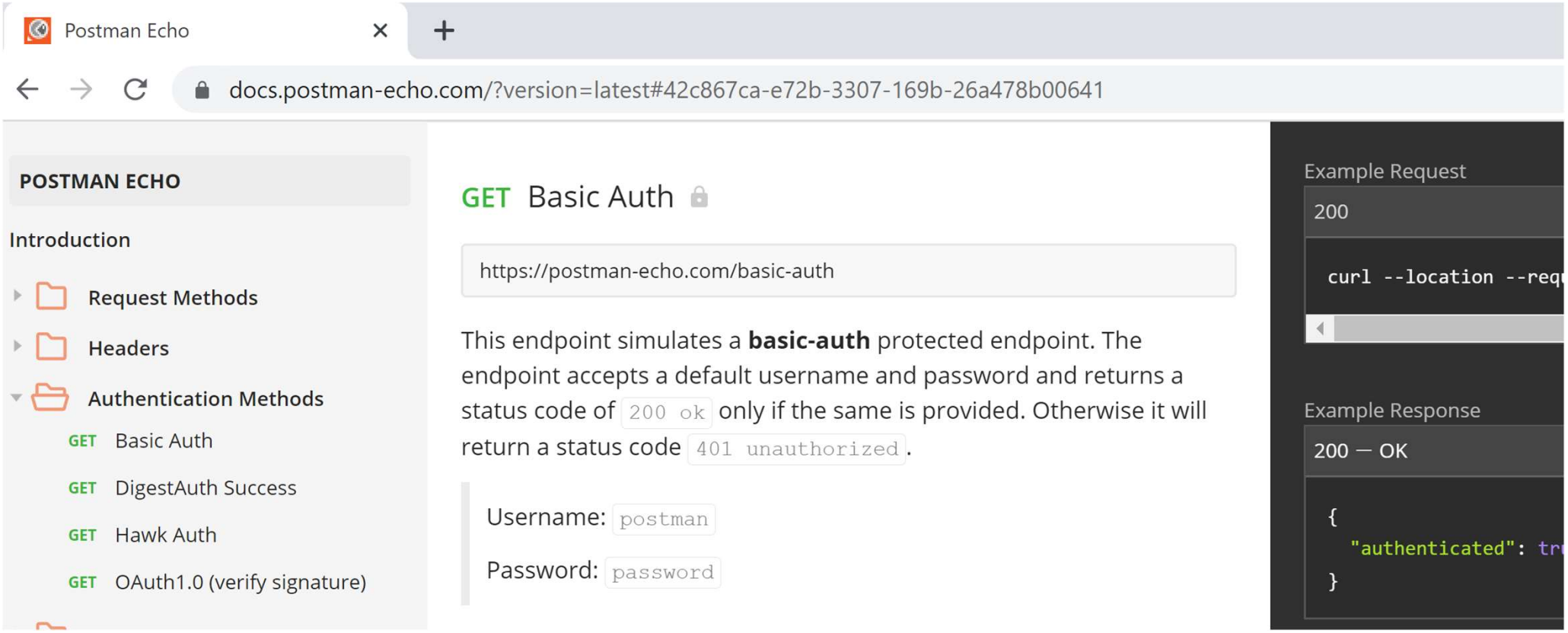
- Creamos otra petición de tipo POST donde podremos ver que podemos usar las variables dinámicas en el body del envío:



Autenticación y Autorización

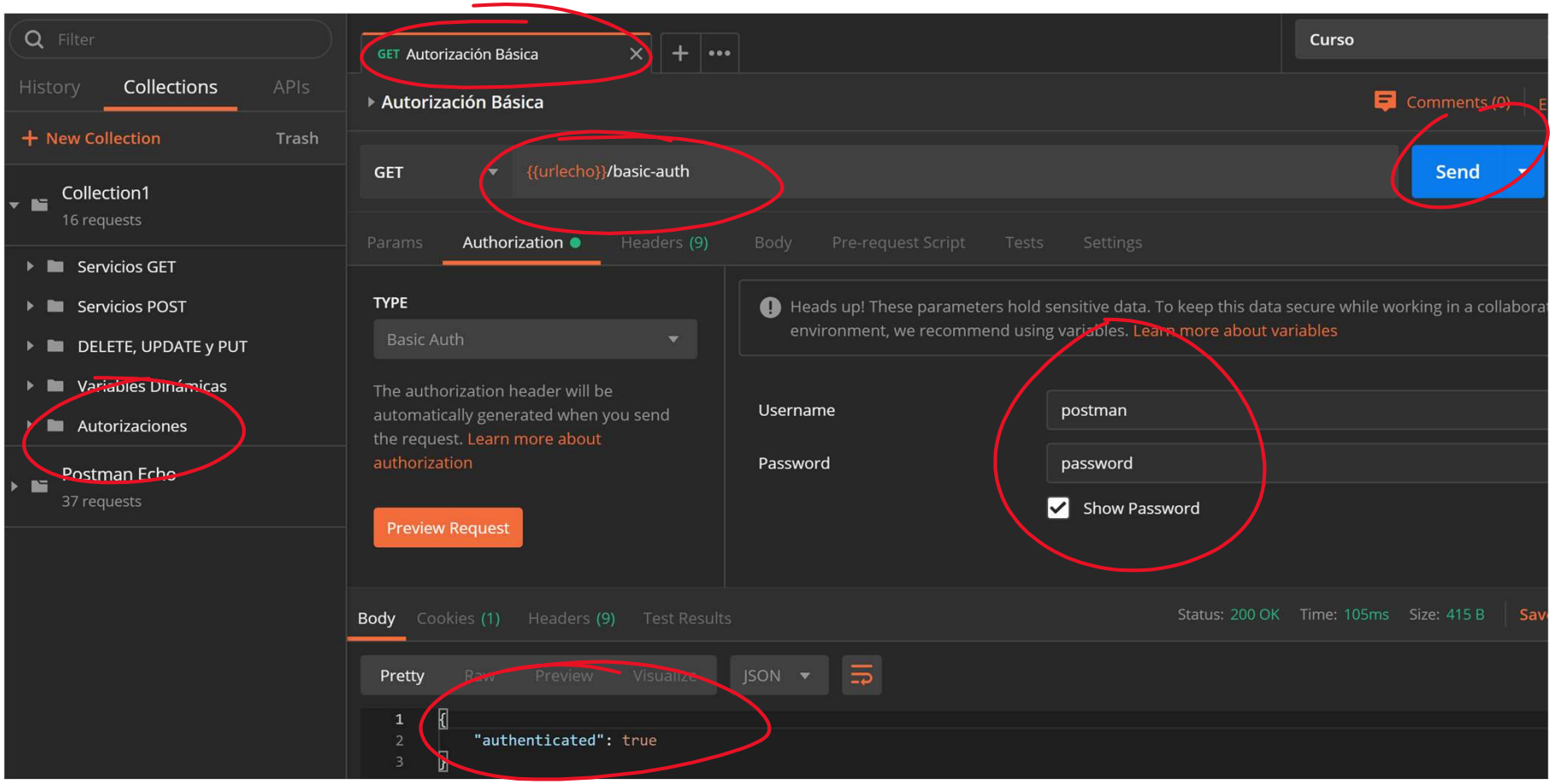
Autenticación básica

- La misma documentación de Postman nos proporciona una guía sobre los diferentes métodos de autenticación de nuestros servicios web:



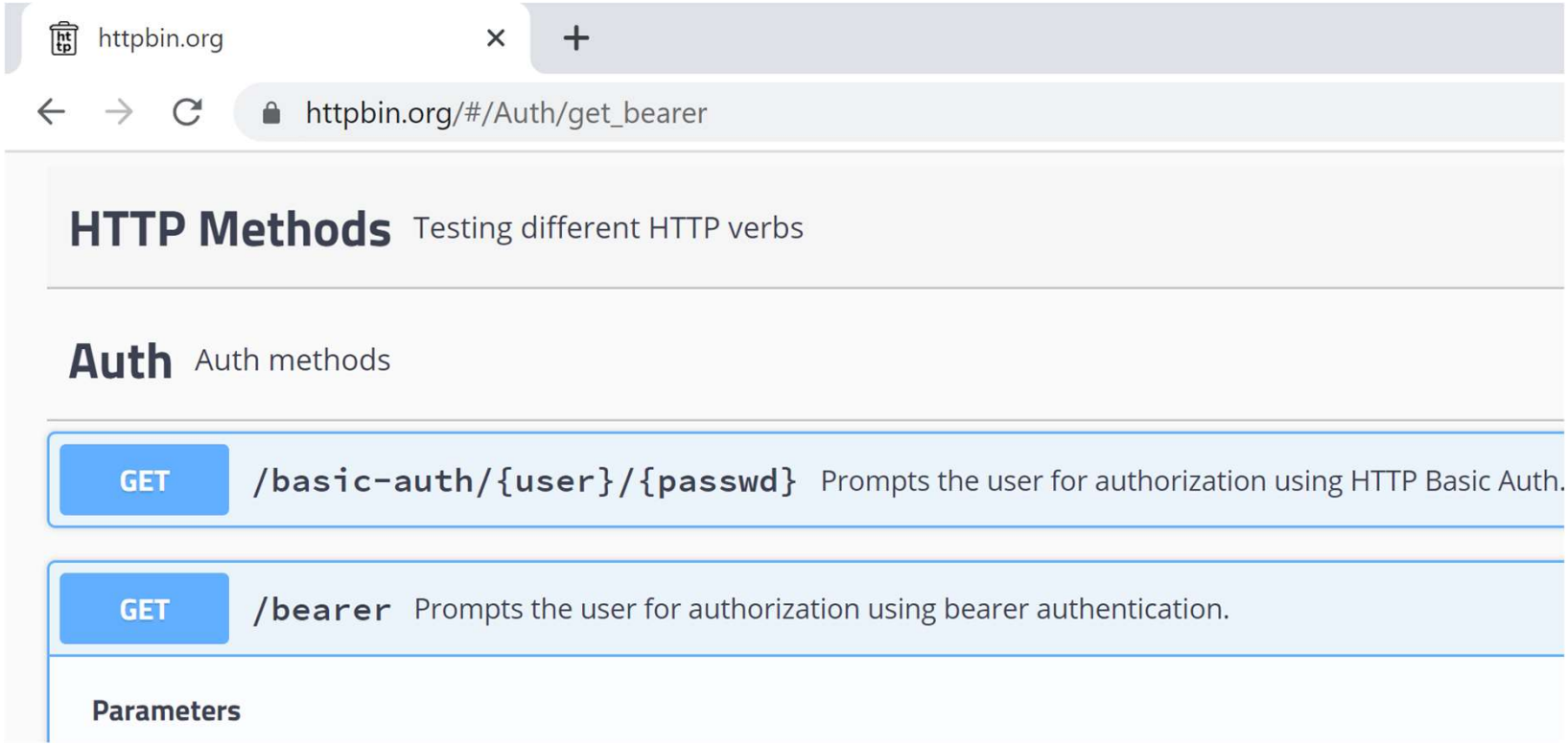
Autenticación básica

- Probamos la autenticación básica en una nueva petición, dentro de una nueva carpeta:



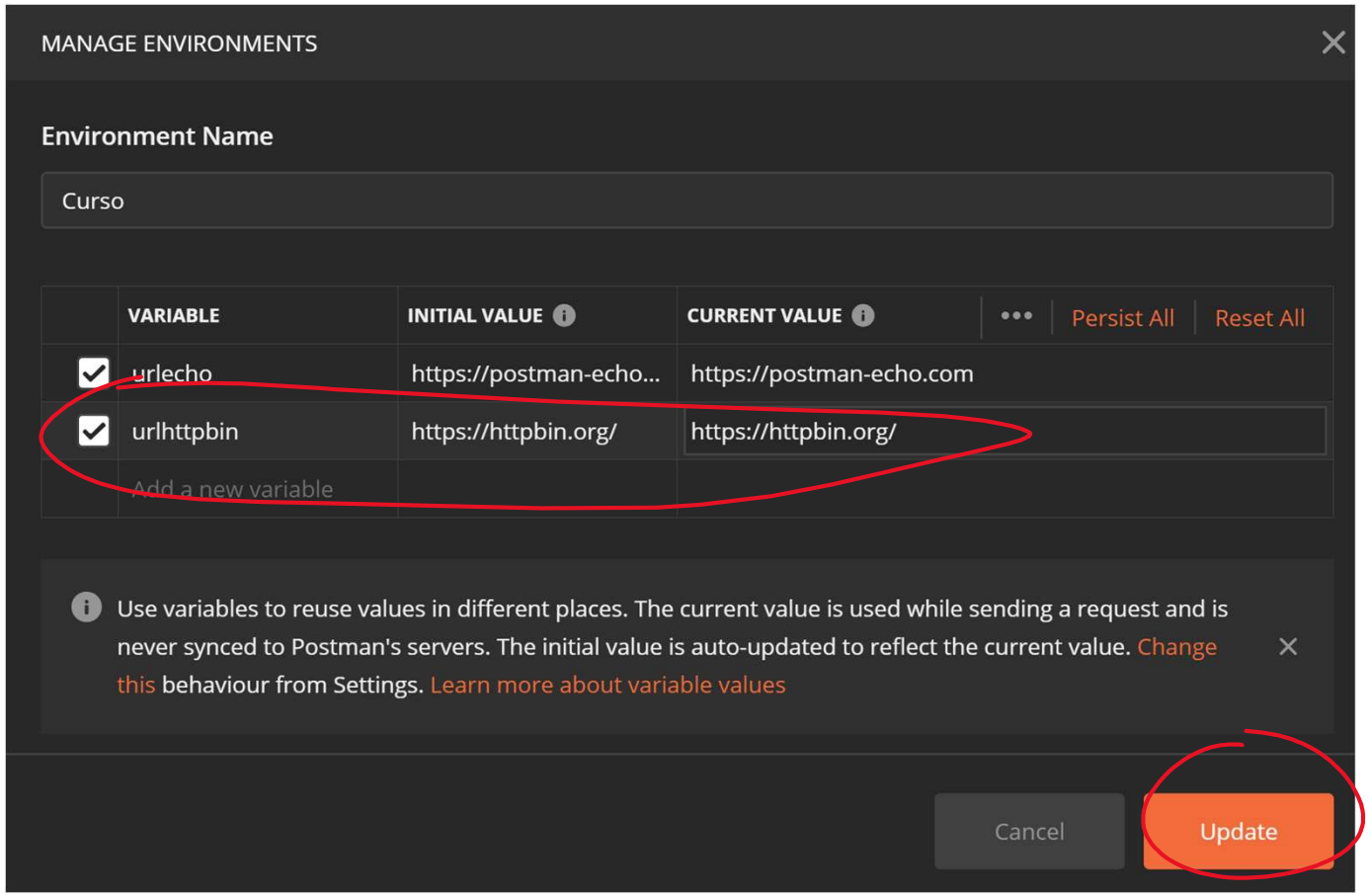
Autenticación por Token

- Existen otros servicios similares a postman-echo como por ejemplo httpbin.org, con el cual podemos probar otro tipo de autenticación mediante tokens:



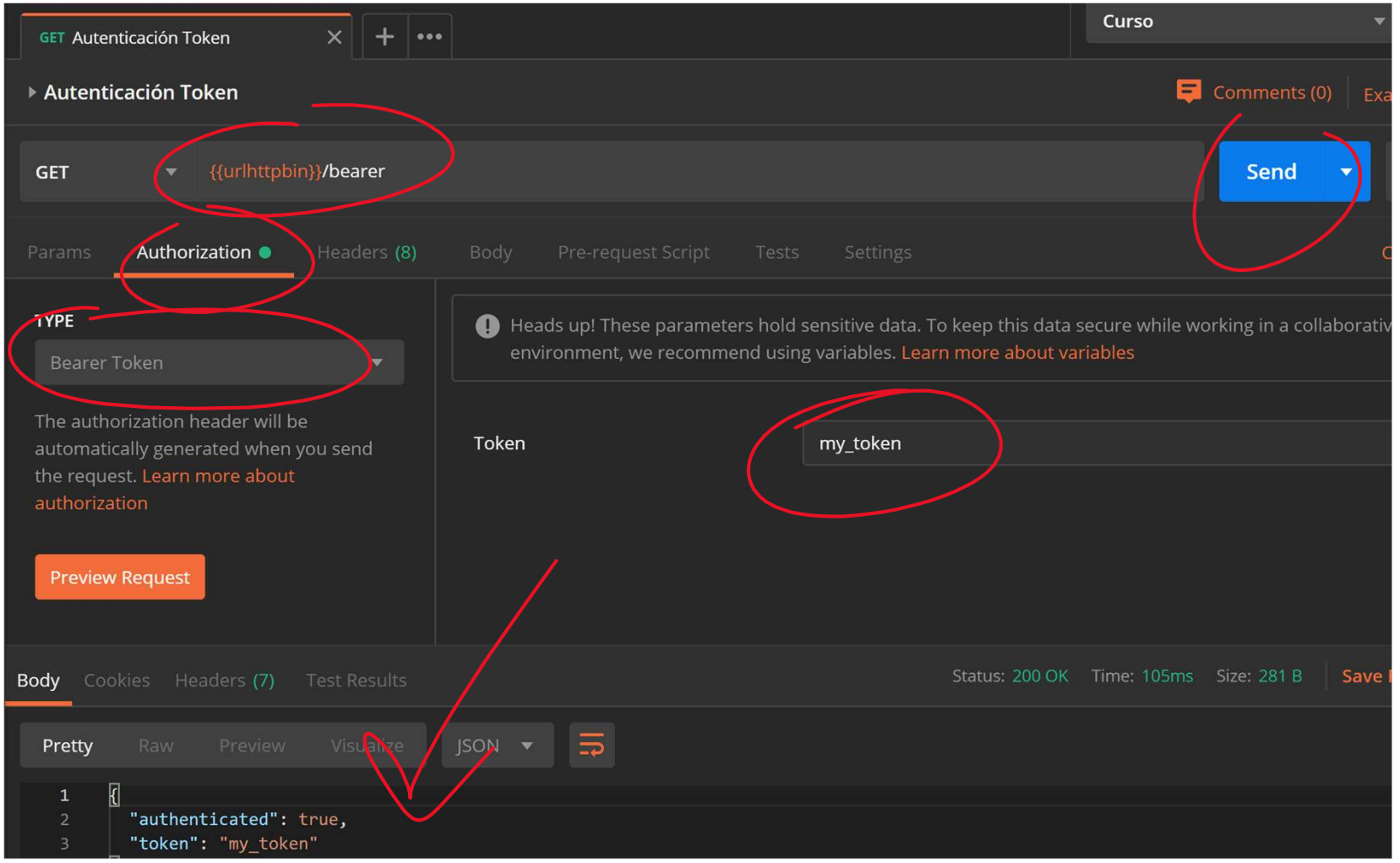
Autenticación por Token

- En primer lugar podemos crear una nueva variable de entorno:



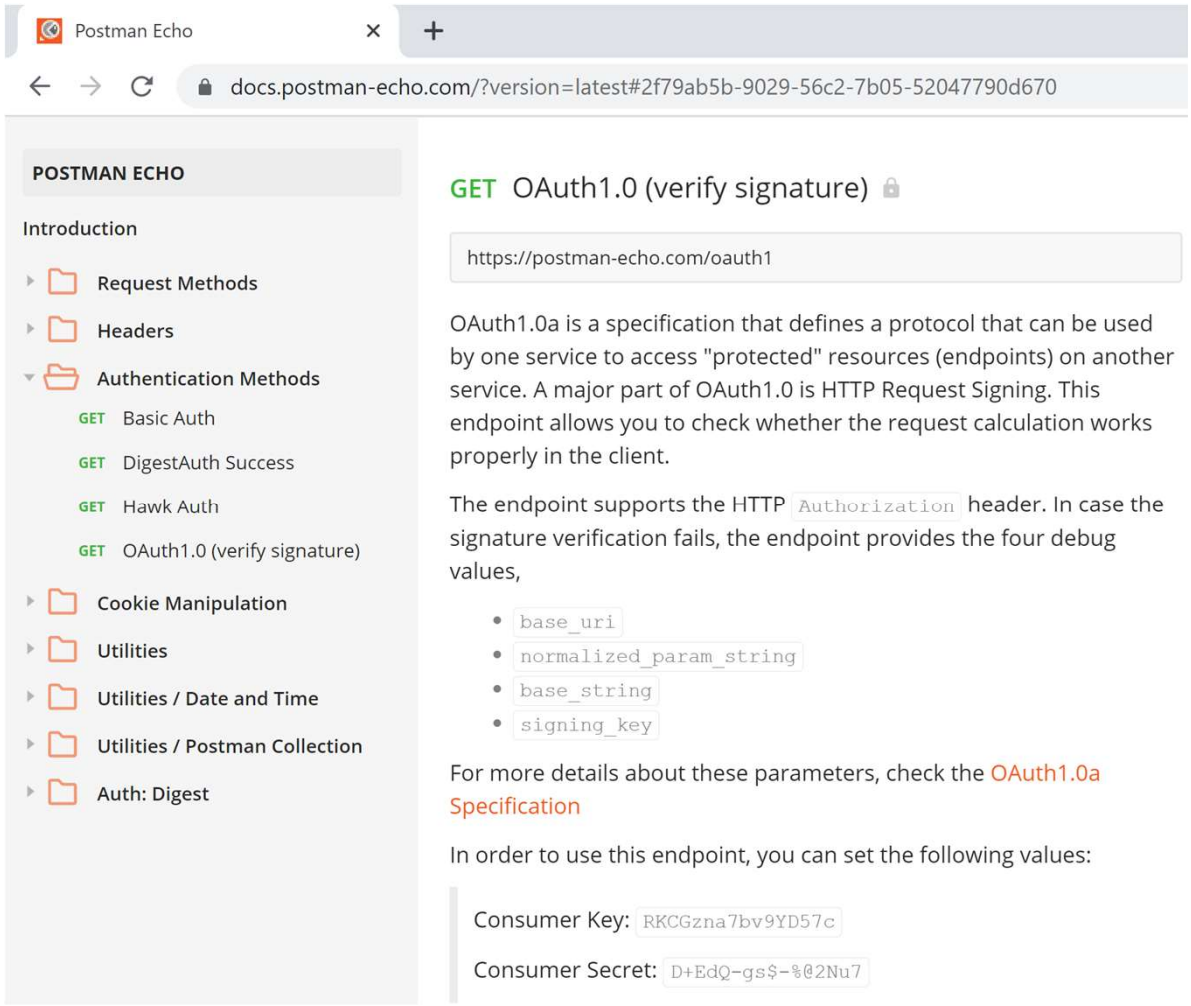
Autenticación por Token

- Creamos la nueva petición:



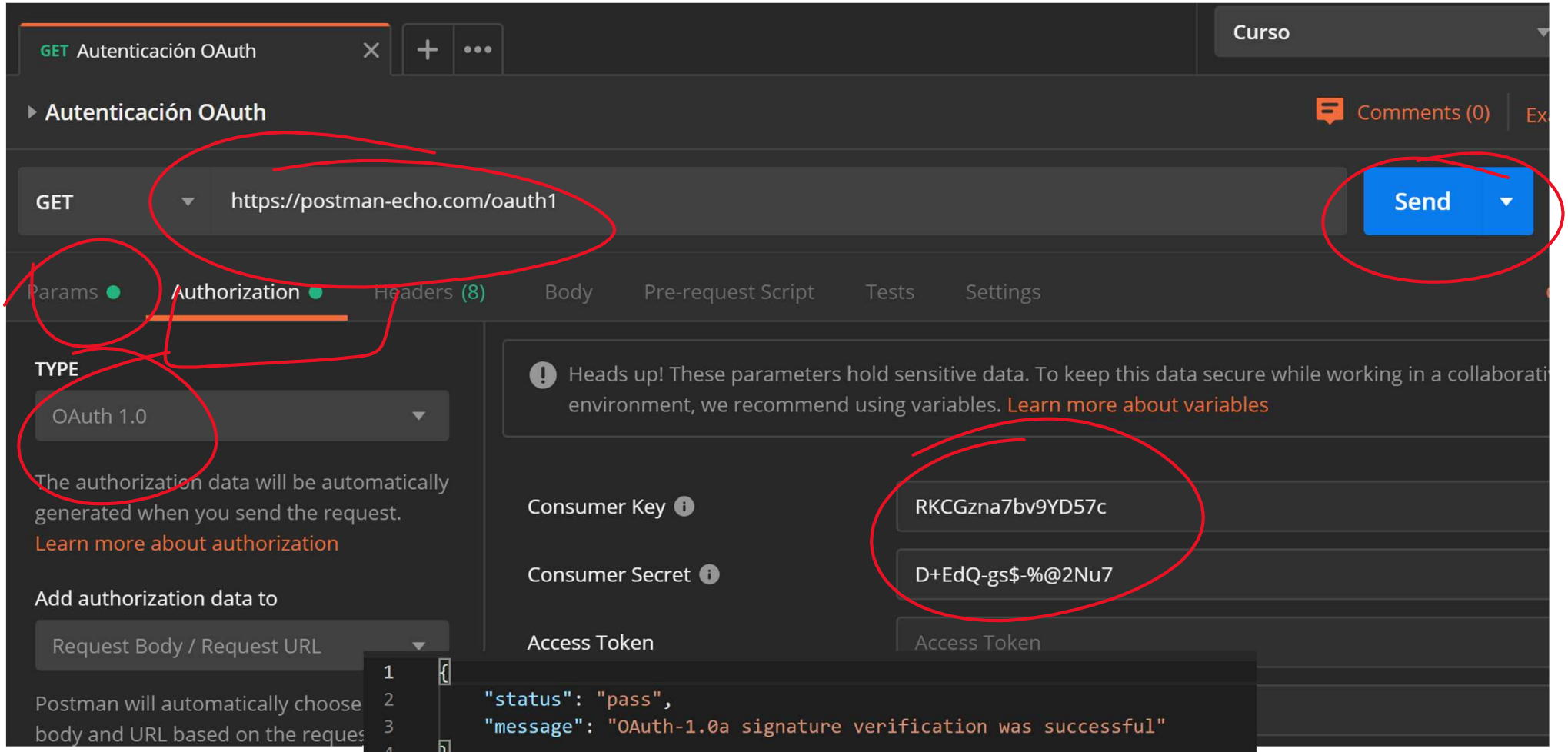
Autenticación OAuth1

- Otro sistema de autenticación bastante usado es el OAuth1, el cual se basa en una combinación de claves mucho más robusto que los dos métodos anteriores:



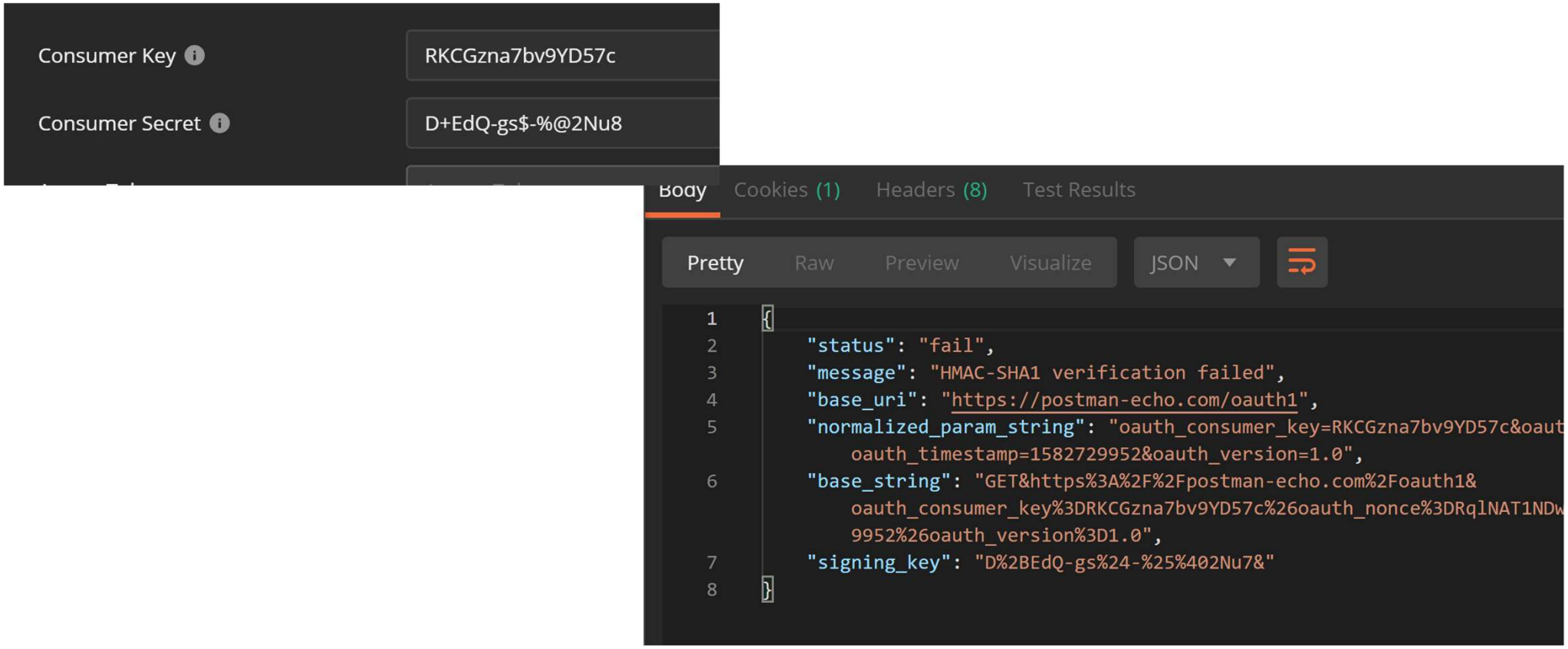
Autenticación OAuth I

- Creamos una nueva petición:



Autenticación OAuth1

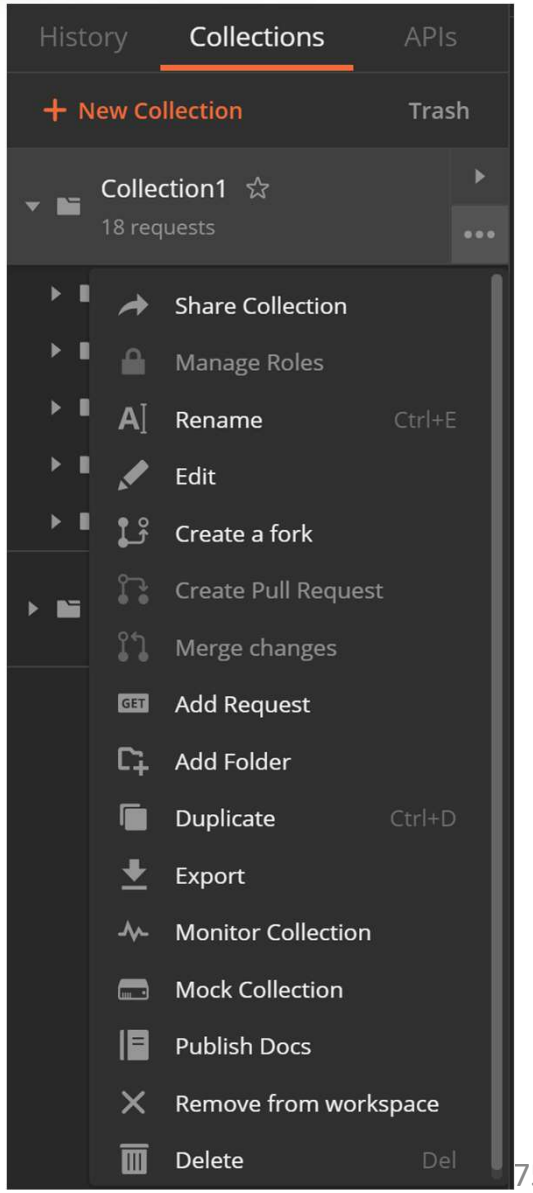
- Si modificamos algún valor del Consumer Secret, la autenticación fallaría:



Colecciones

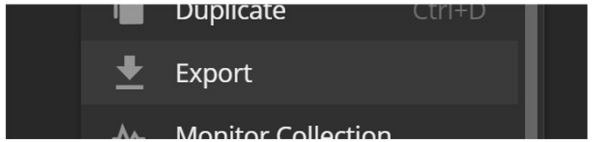
Operaciones sobre colecciones

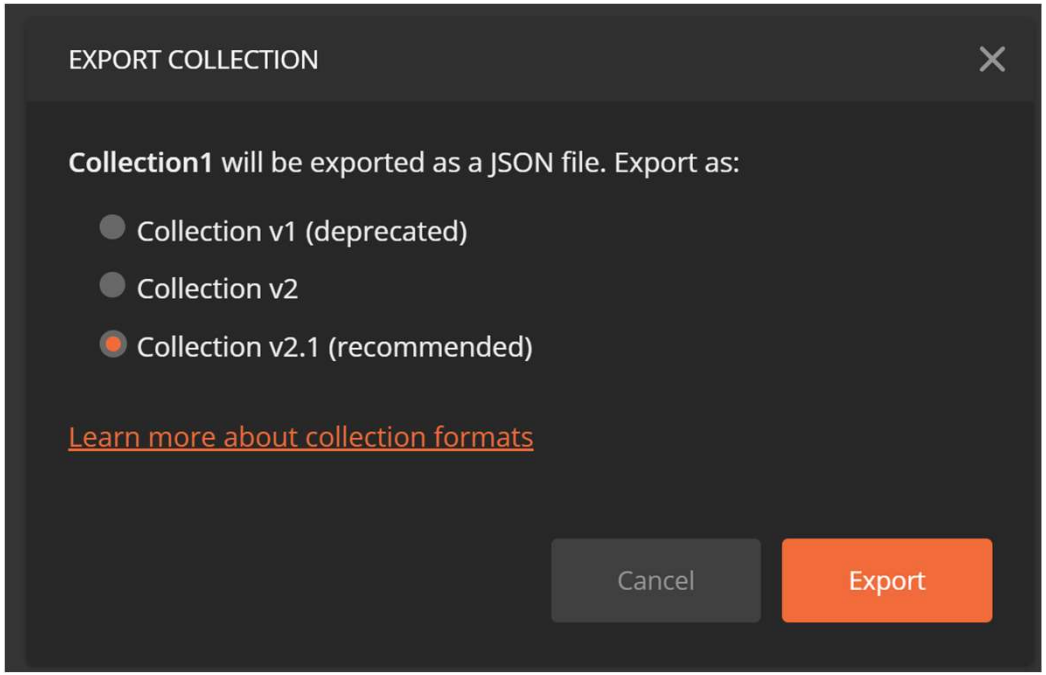
- Podemos realizar diferentes operaciones sobre las colecciones: renombrar, duplicar, exportar, borrar, etc.



Exportar JSON

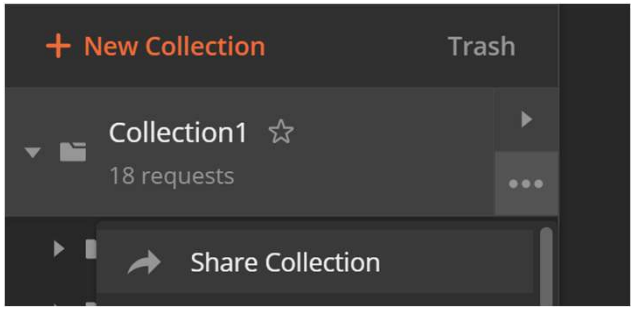
- Podemos exportar una colección como un JSON que posteriormente podríamos importar en otro espacio de trabajo:

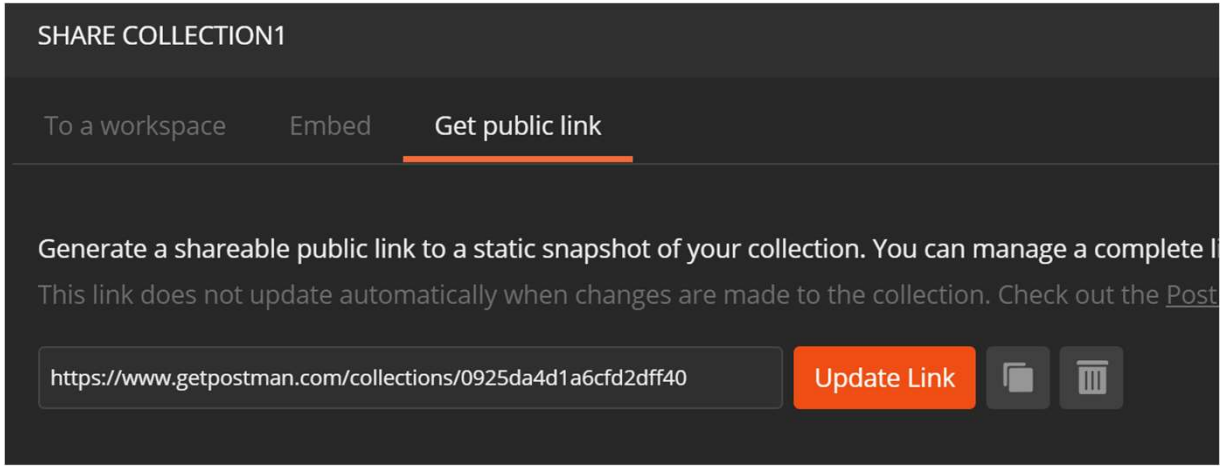




Exportar Link

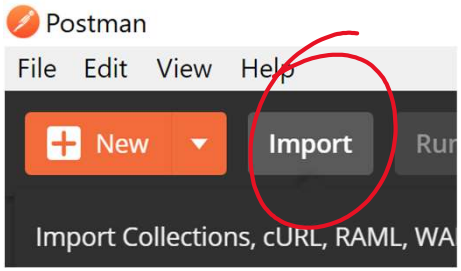
- Es posible compartir mi colección mediante un enlace público:

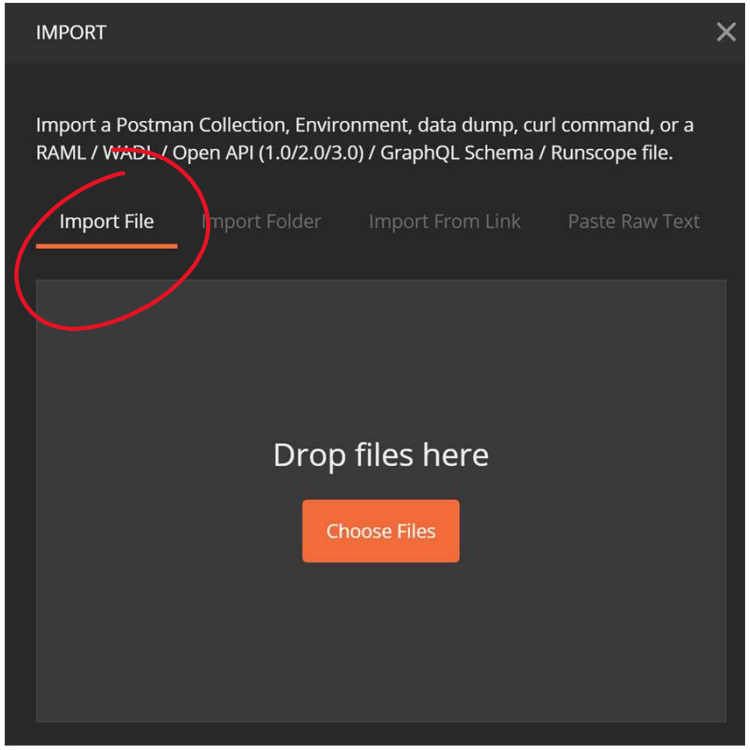


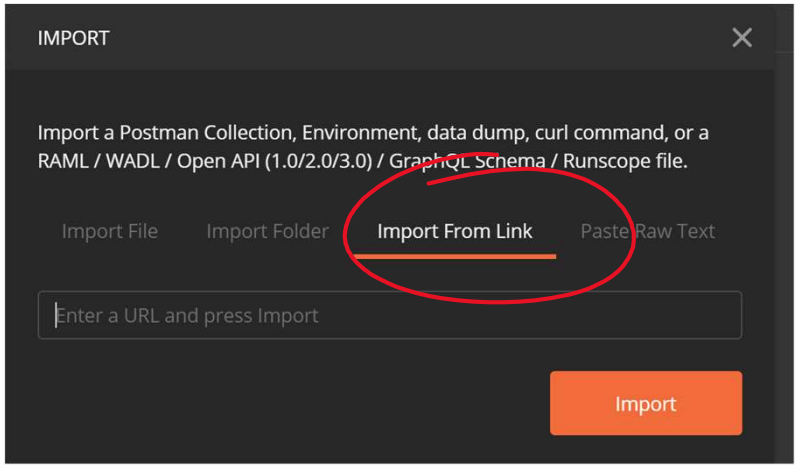


Importar

- Podemos importar el fichero JSON o el link:

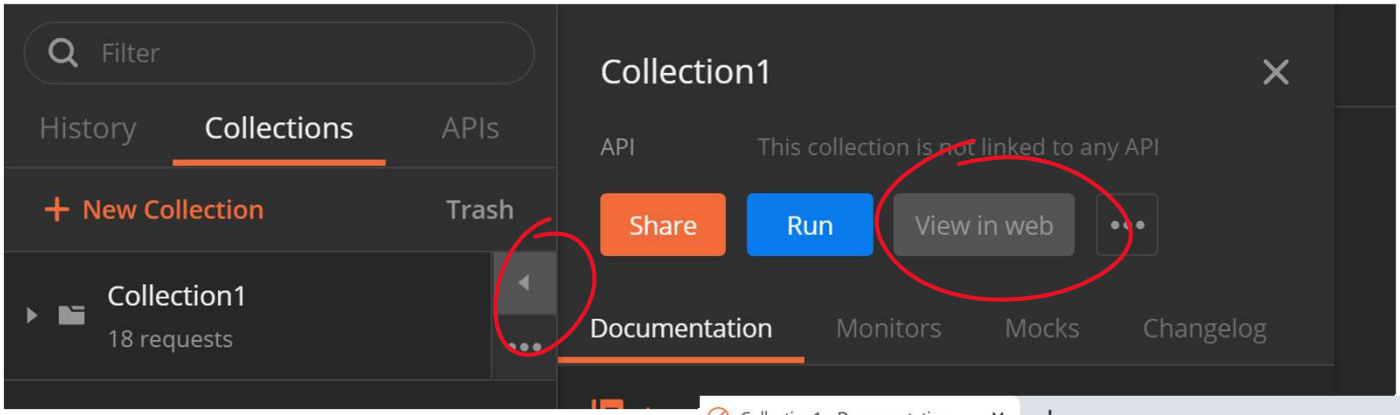


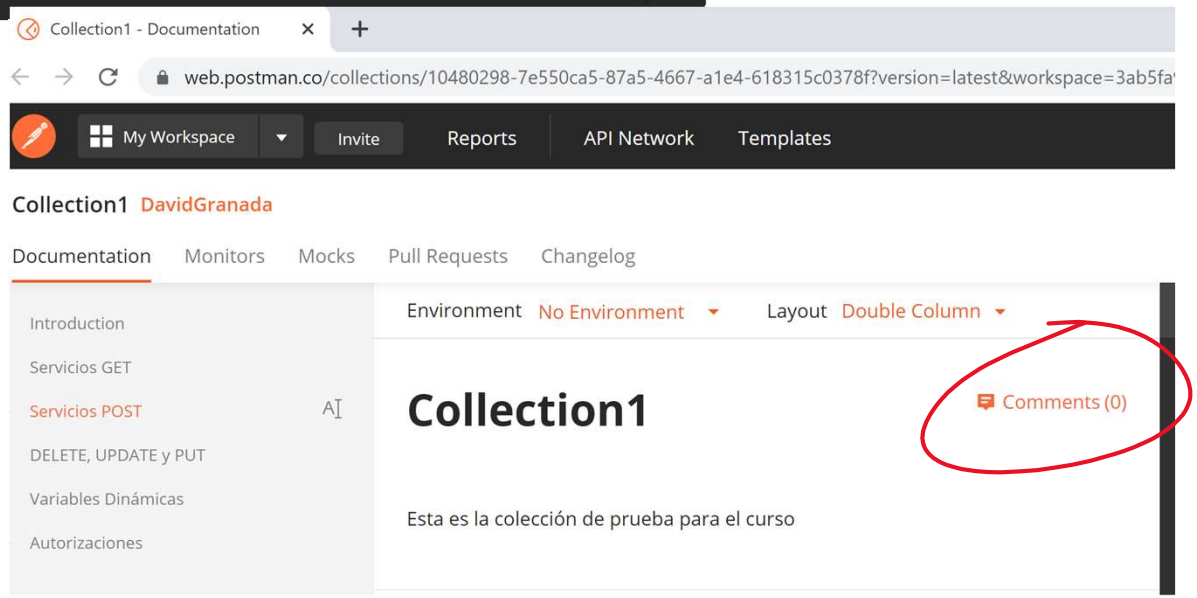




Ver colección en Web

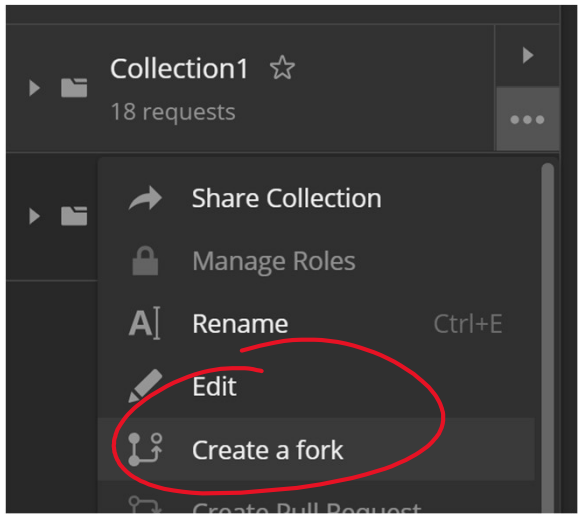
- Podemos ver la web de nuestra colección sobre la que podemos añadir comentarios:

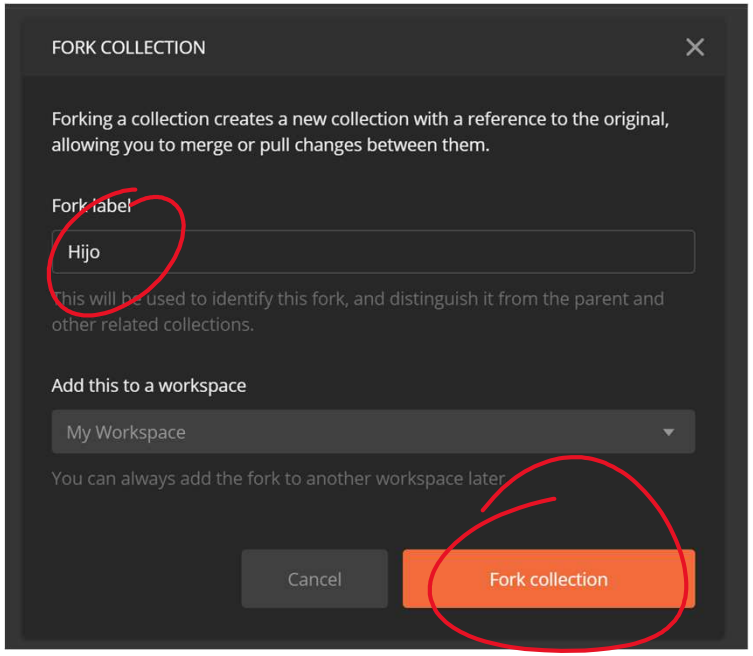


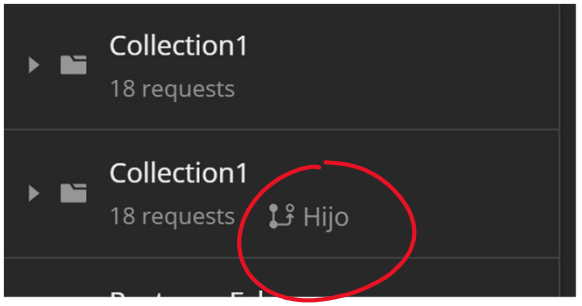


Control de versiones

- Puedo crear forks de las colecciones (habitualmente para que alguien más trabaje independientemente sobre la colección y posteriormente se haga el respectivo merge):

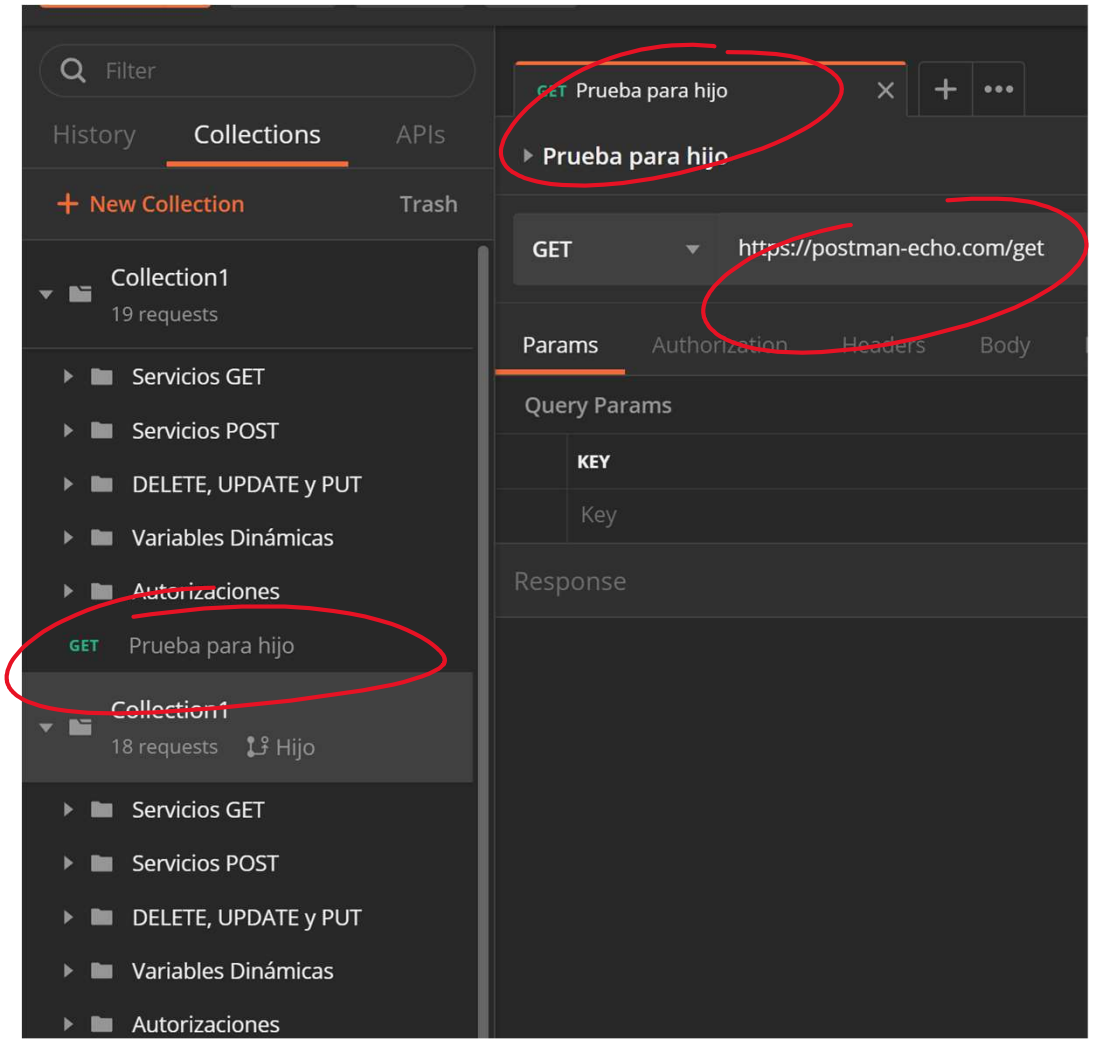






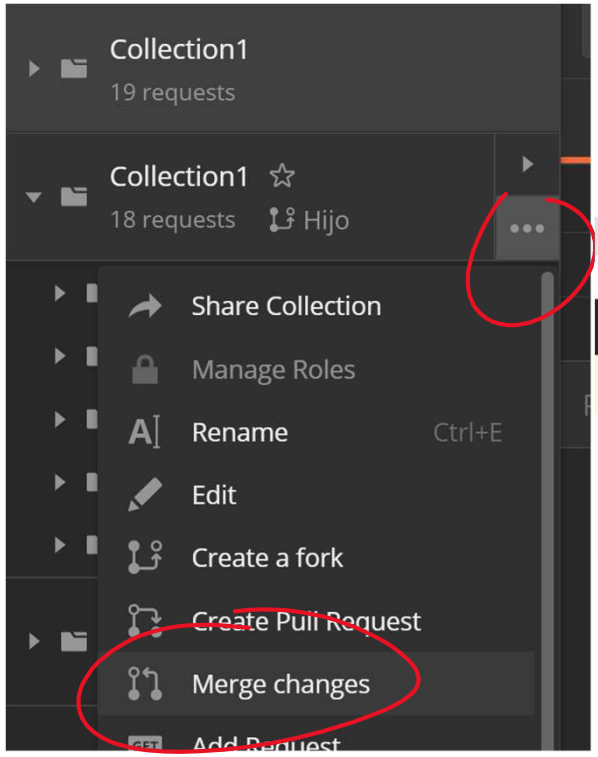
Control de versiones

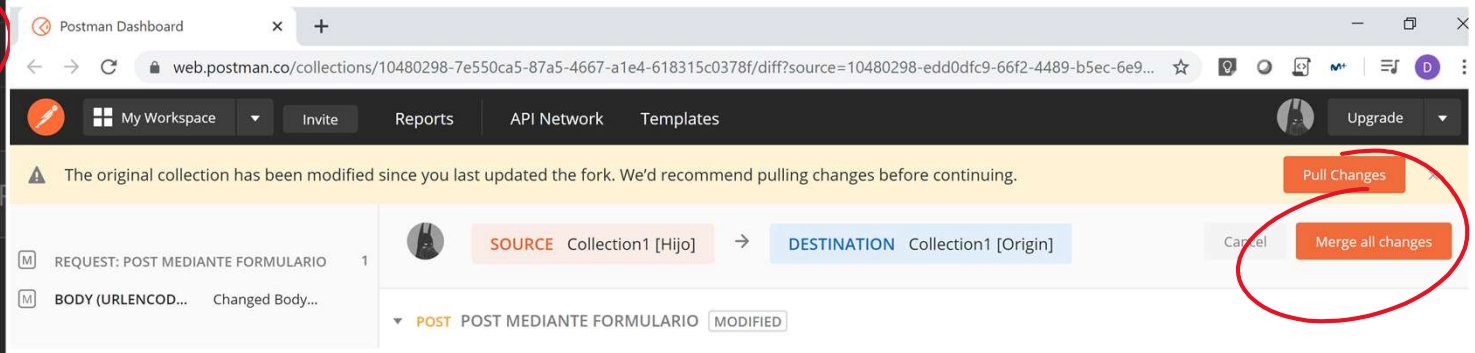
- Creamos una nueva petición (cualquiera) en la colección padre:



Control de versiones

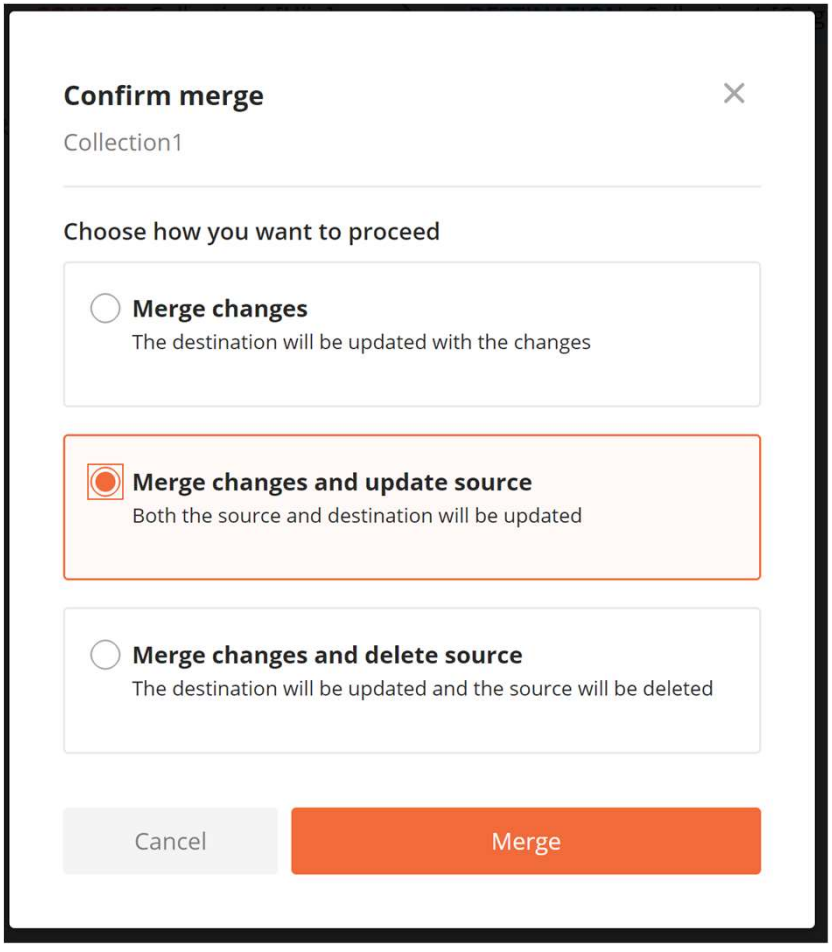
- Desde la colección hijo hacemos un merge changes, se nos abre una página web:

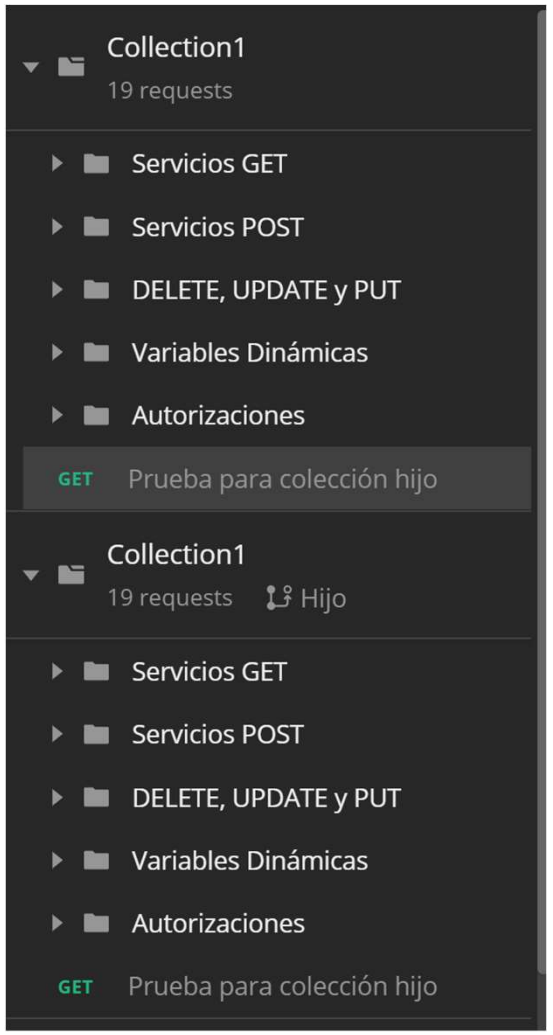




Control de versiones

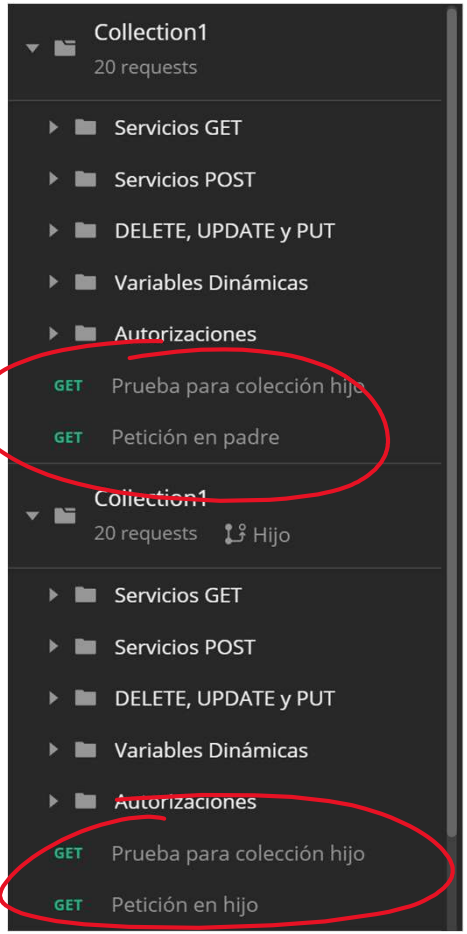
- Podemos elegir el tipo de merge (sólo el destino o los cambios en las dos colecciones):





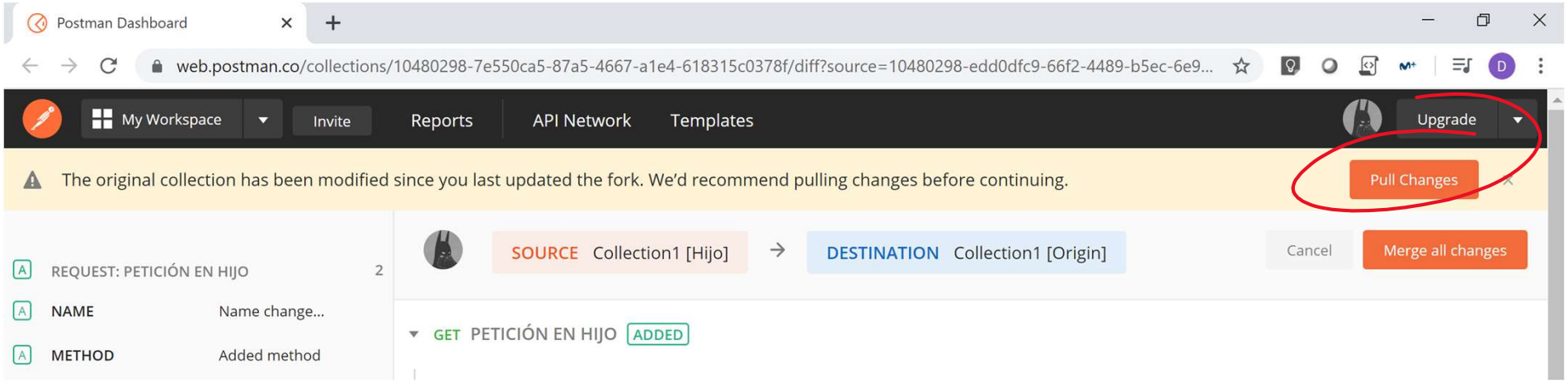
Control de versiones

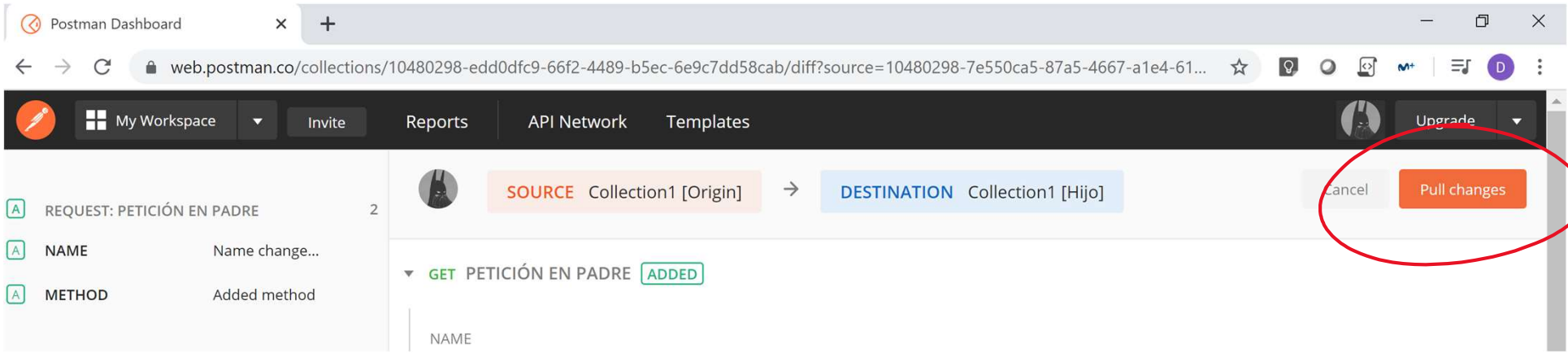
Creamos una petición (vacía) en cada una de las colecciones, simulando el trabajo de dos personas. Hacemos el merge en la colección hijo:



Control de versiones

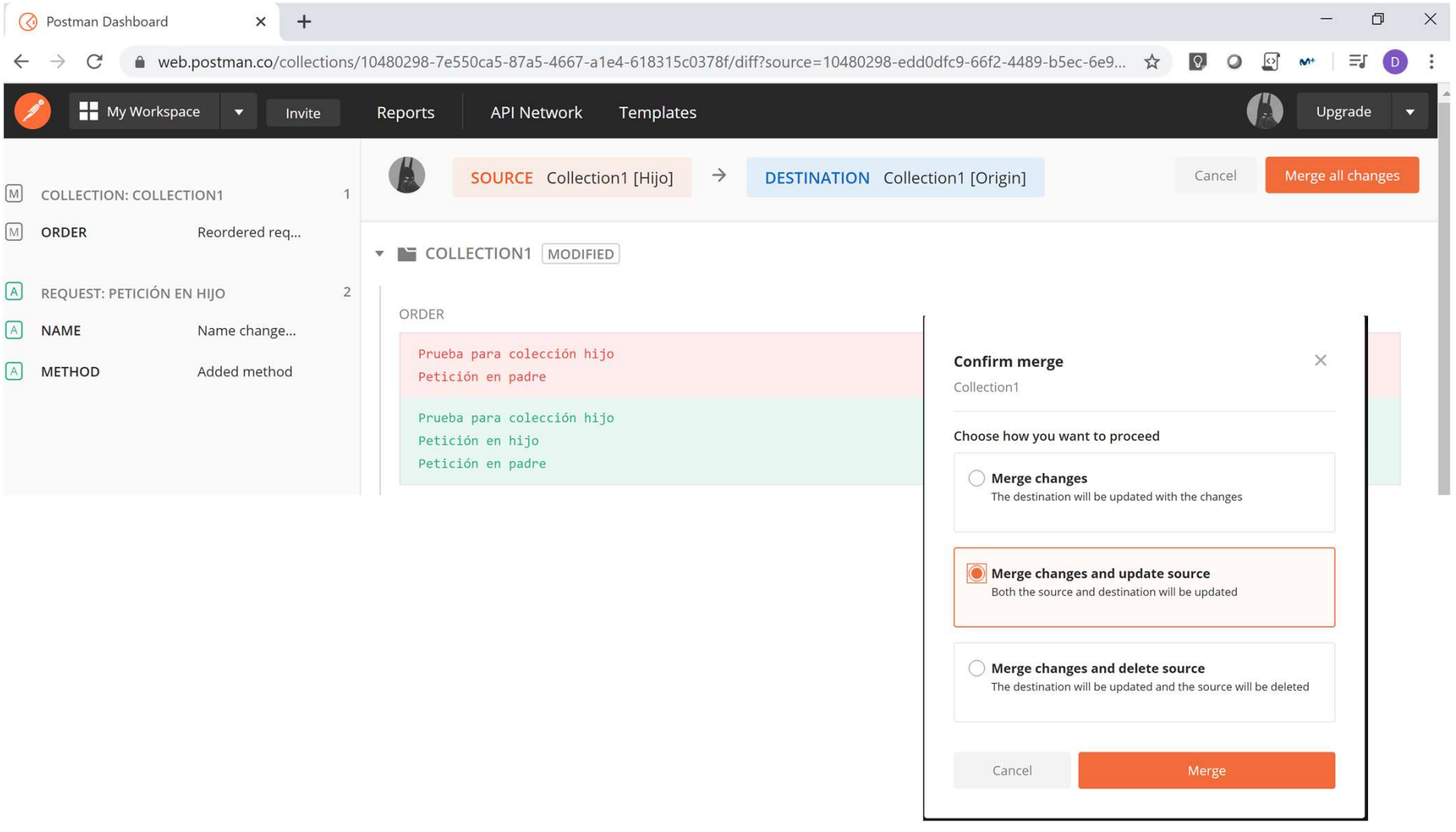
- Hacemos los pull:





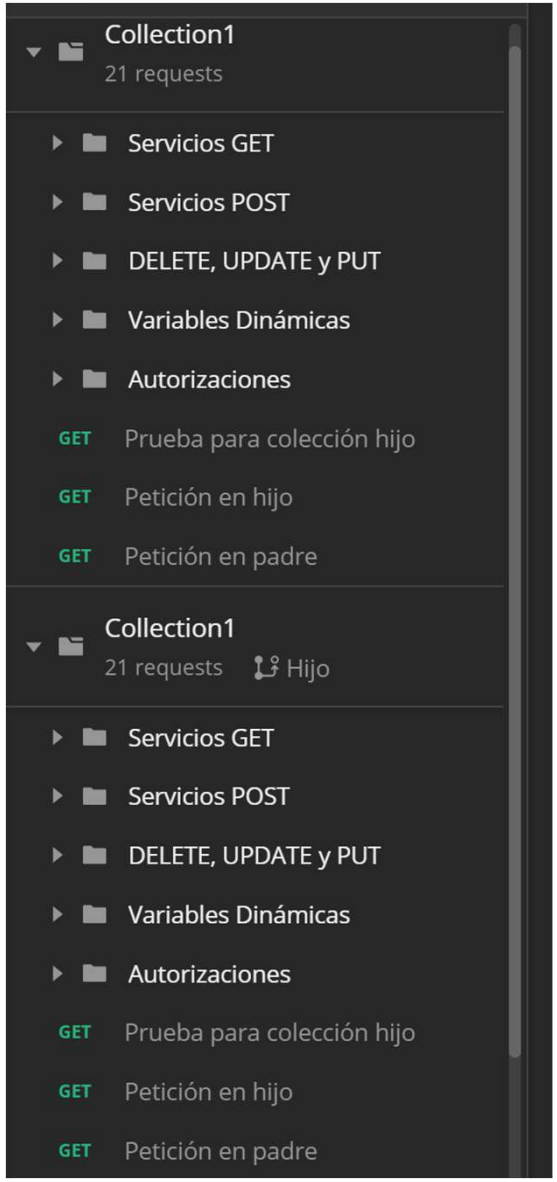
Control de versiones

- Y finalmente el merge:



Control de versiones

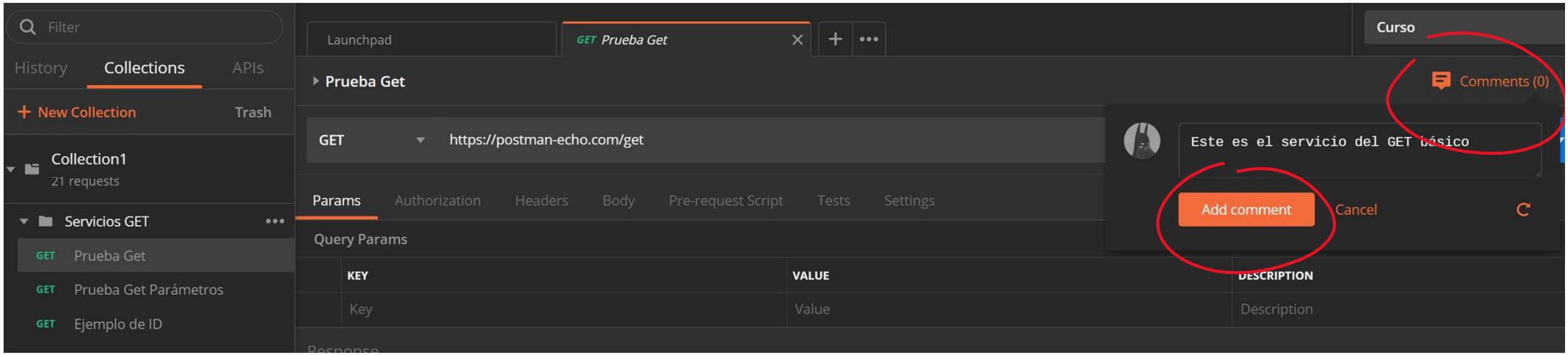
- Al volver a Postman, veremos todas las peticiones en ambas colecciones:



Utilidades

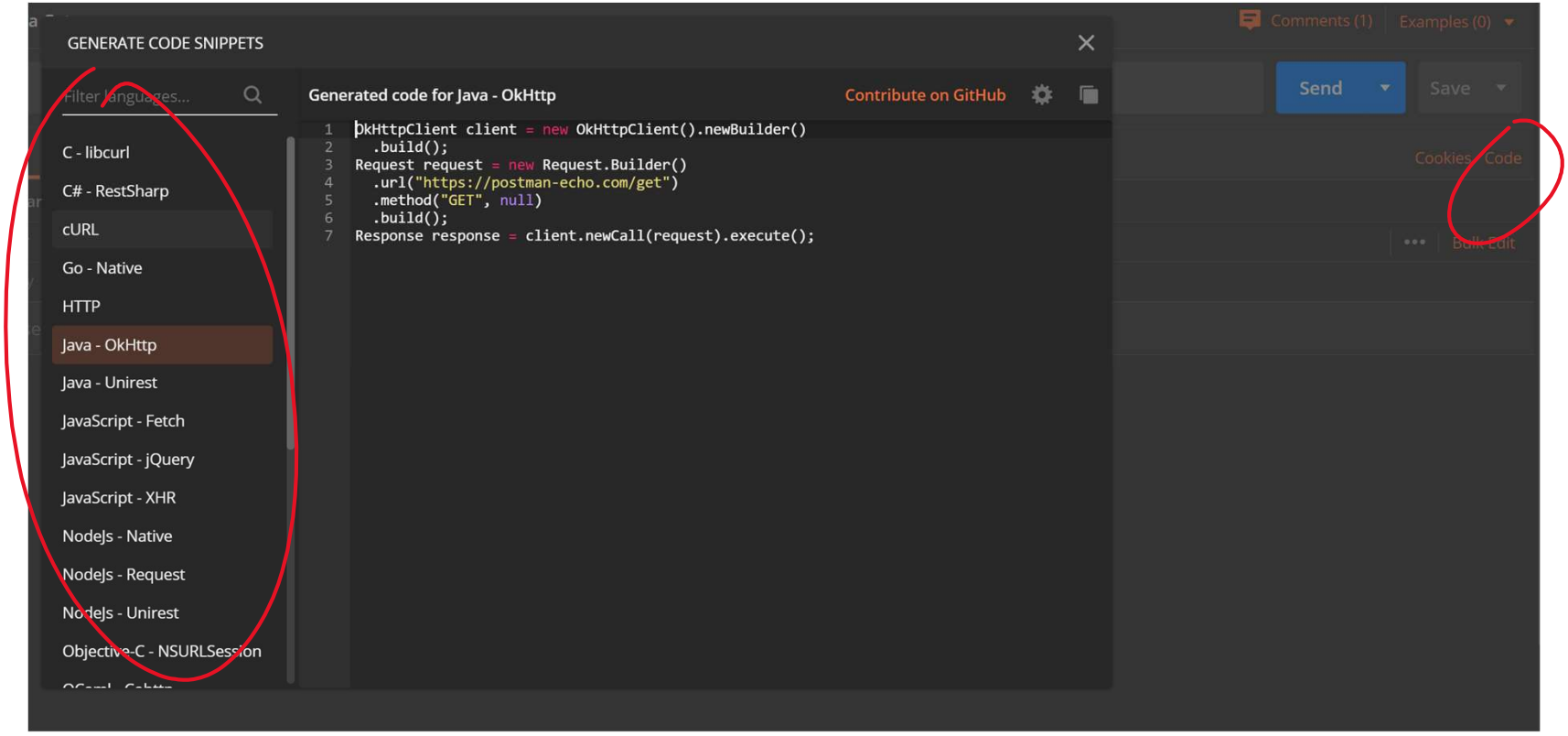
Comentarios

- Podemos ir añadiendo comentarios sobre nuestras peticiones con el fin de generar una mejor documentación:



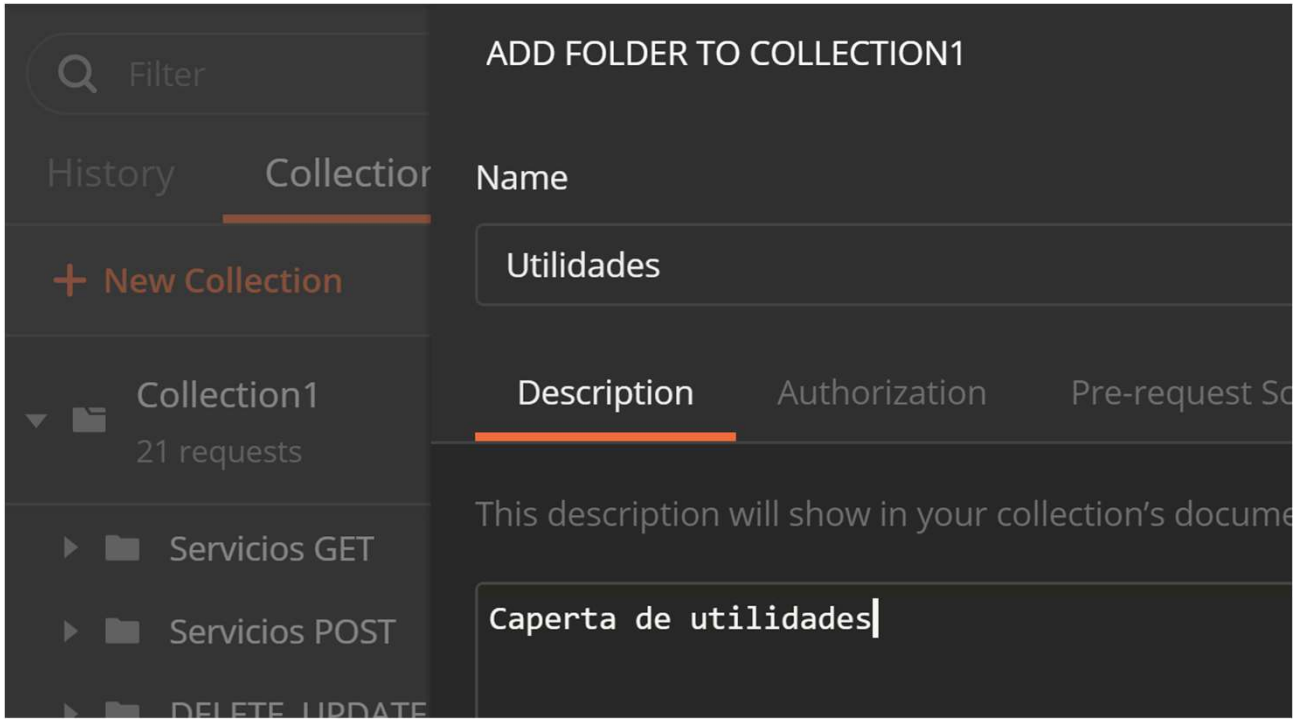
Código

- Postman me genera el código del servicio web de forma automática y para muchos lenguajes de programación:



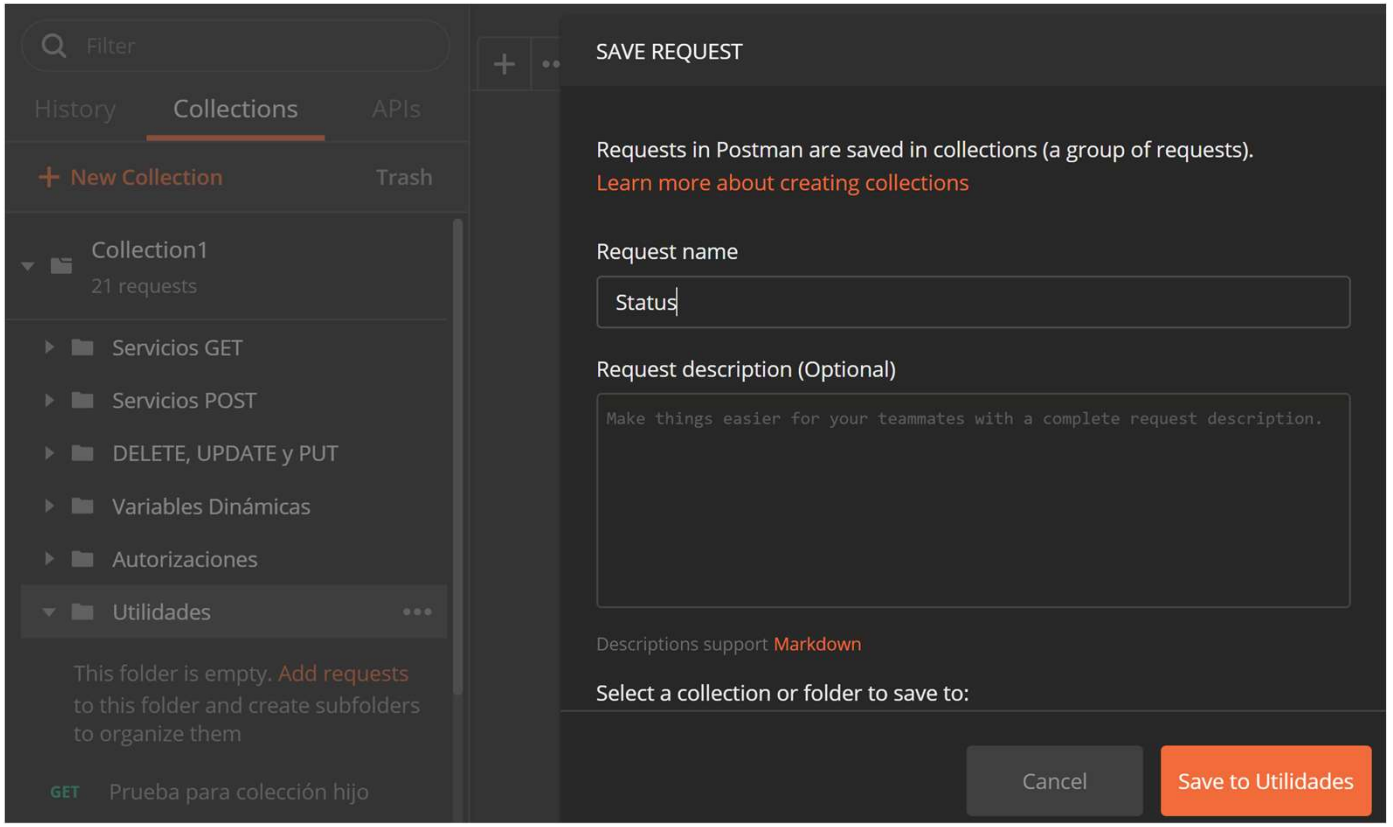
Utilidades

- Vamos a crear una nueva carpeta dentro de la colección:



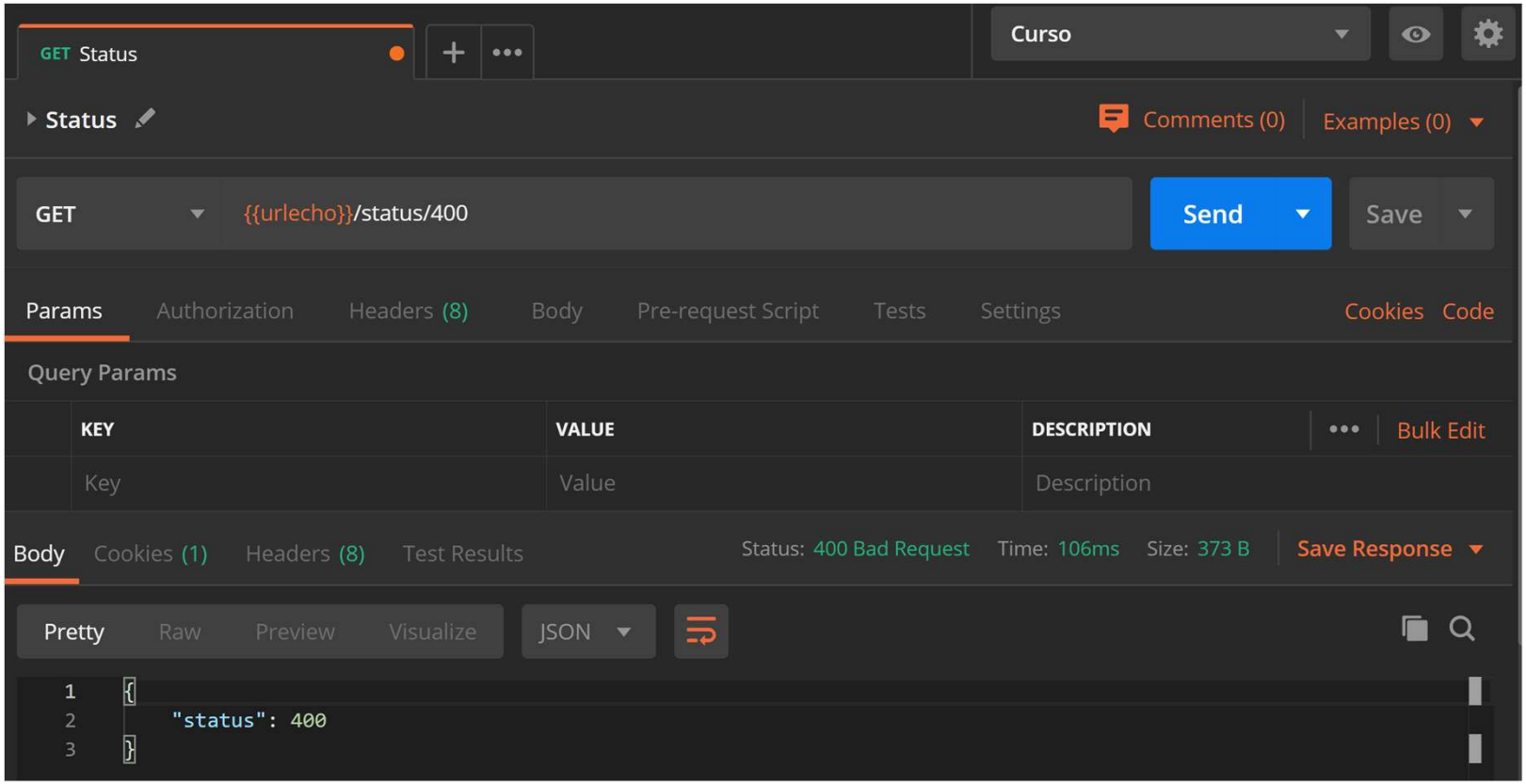
Status

- Dentro de esta carpeta creamos una nueva petición que nos permitirá realizar diferentes pruebas en base al código del estado que nos devuelva un servicio web:



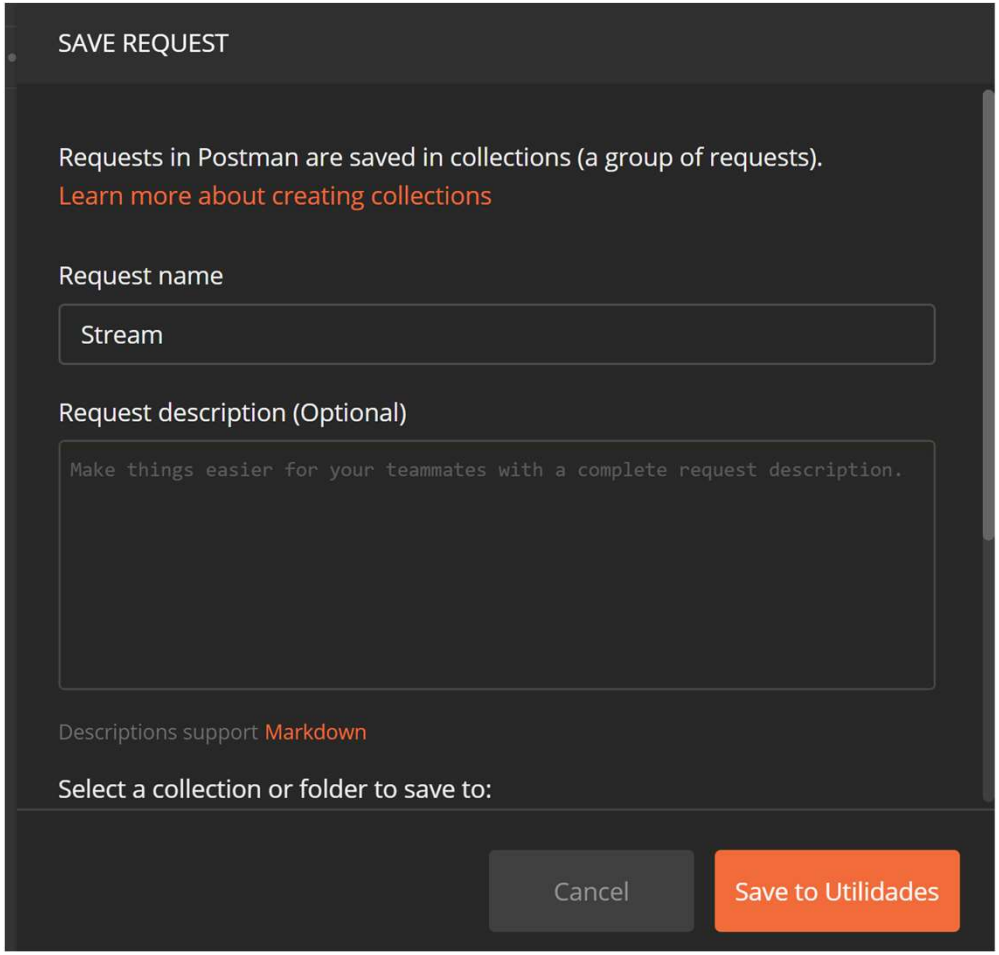
Status

- Podemos utilizar la utilidad status y probar con los diferentes códigos de estado que podría devolver un servicio web: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes



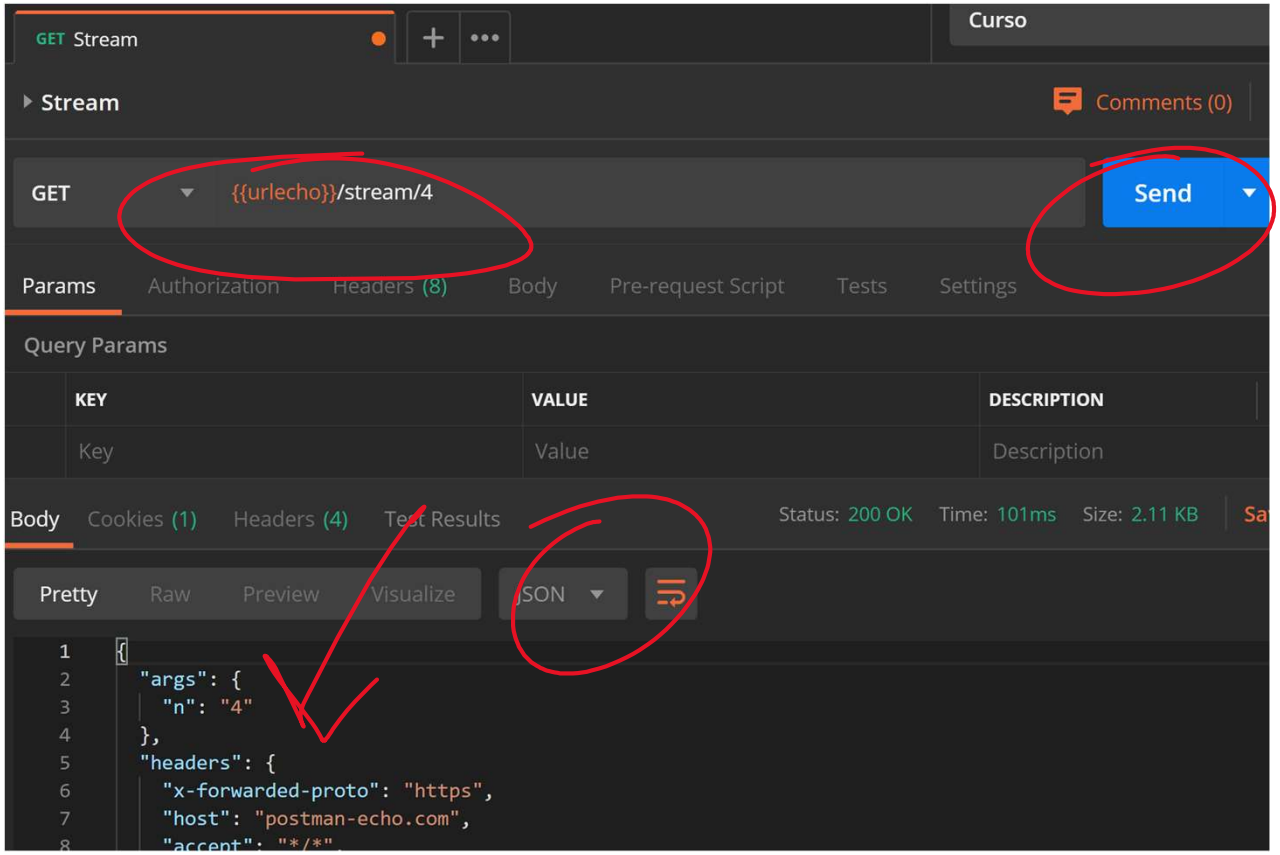
Stream

- Si queremos dividir la respuesta de un servicio web en varias cadenas de texto podemos usar otra utilidad llamada Stream. Nos permite crear aplicaciones que por ejemplo van recogiendo un valor concreto en cada uno de los resultados obtenidos



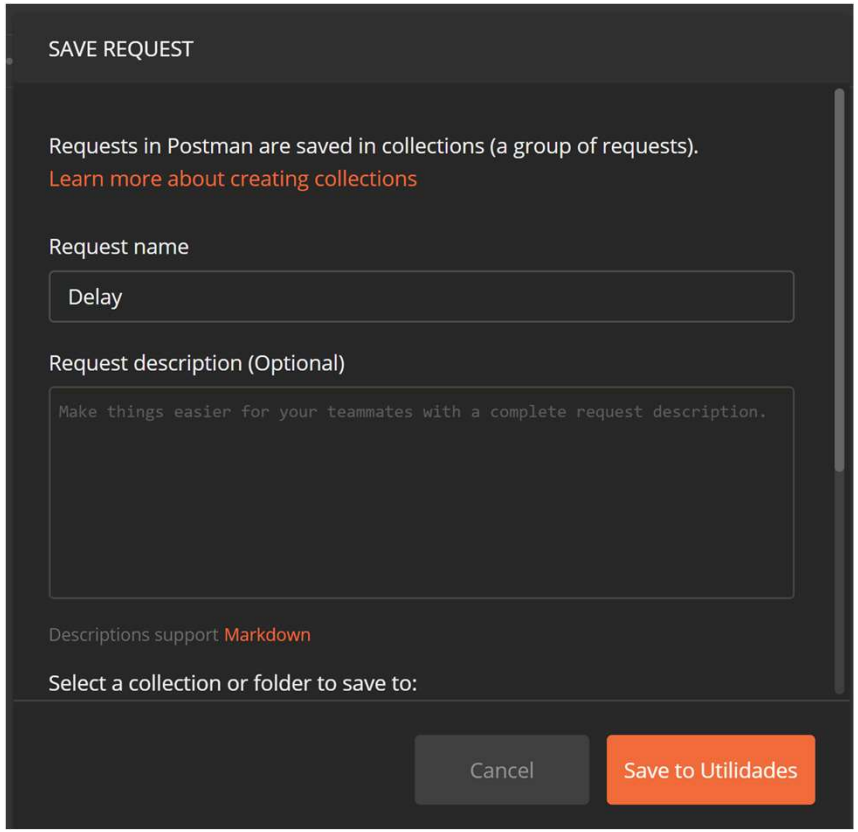
Stream

- Podemos modificar el número de divisiones:



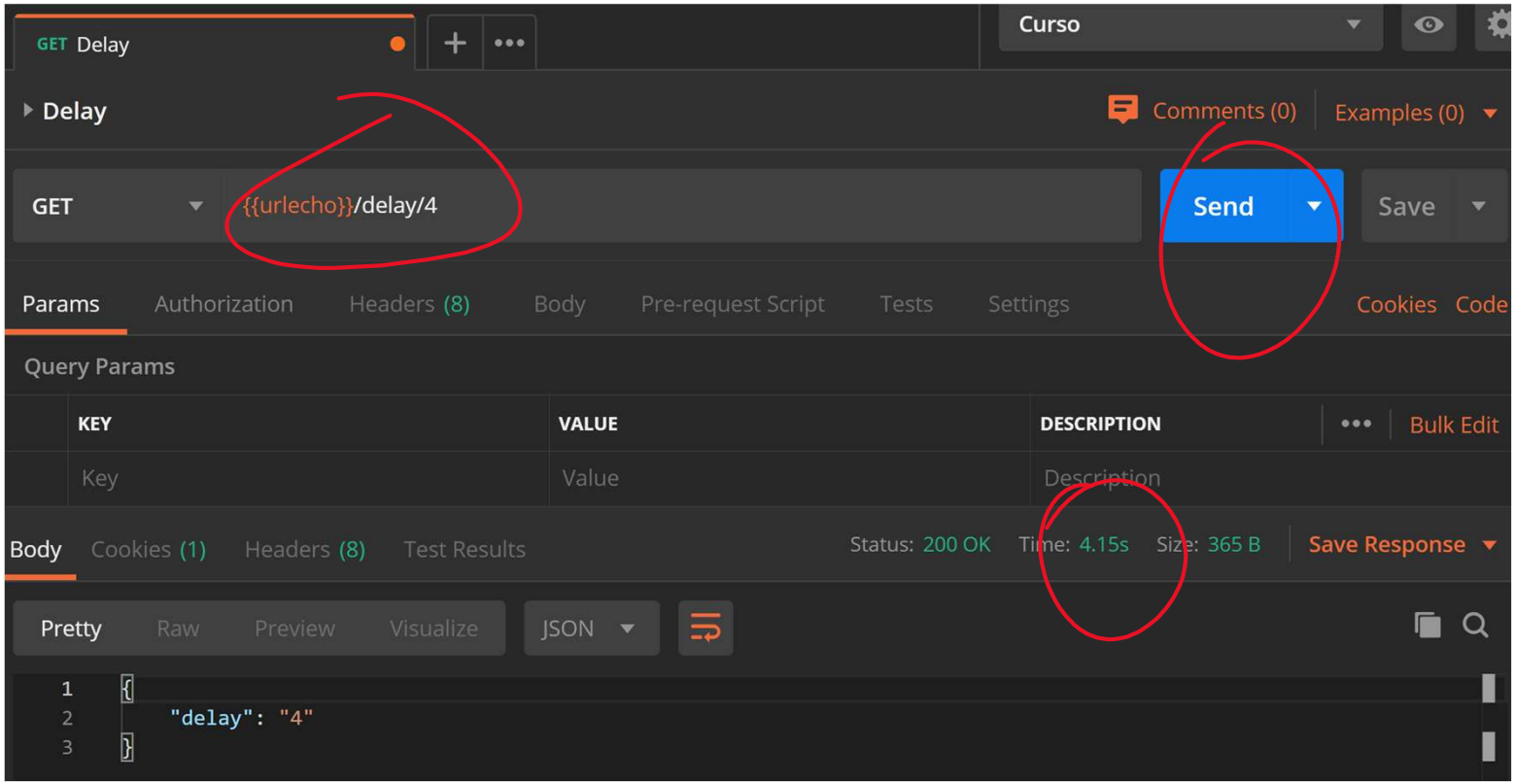
Delay

- Existe otra utilidad que nos permite añadir un retraso al servicio web antes de efectivamente devolver el resultado del servicio
- Creamos otra petición:



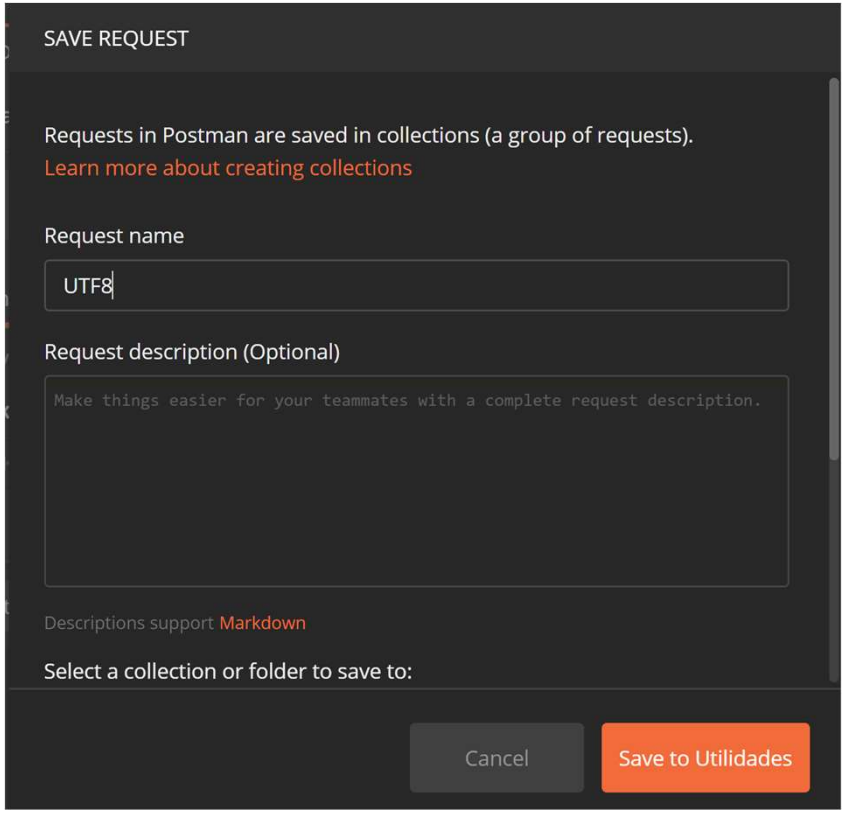
Delay

- Muchos servicios utilizan esta utilidad para mostrar mensajes publicitarios antes de enviar los resultados al usuario:



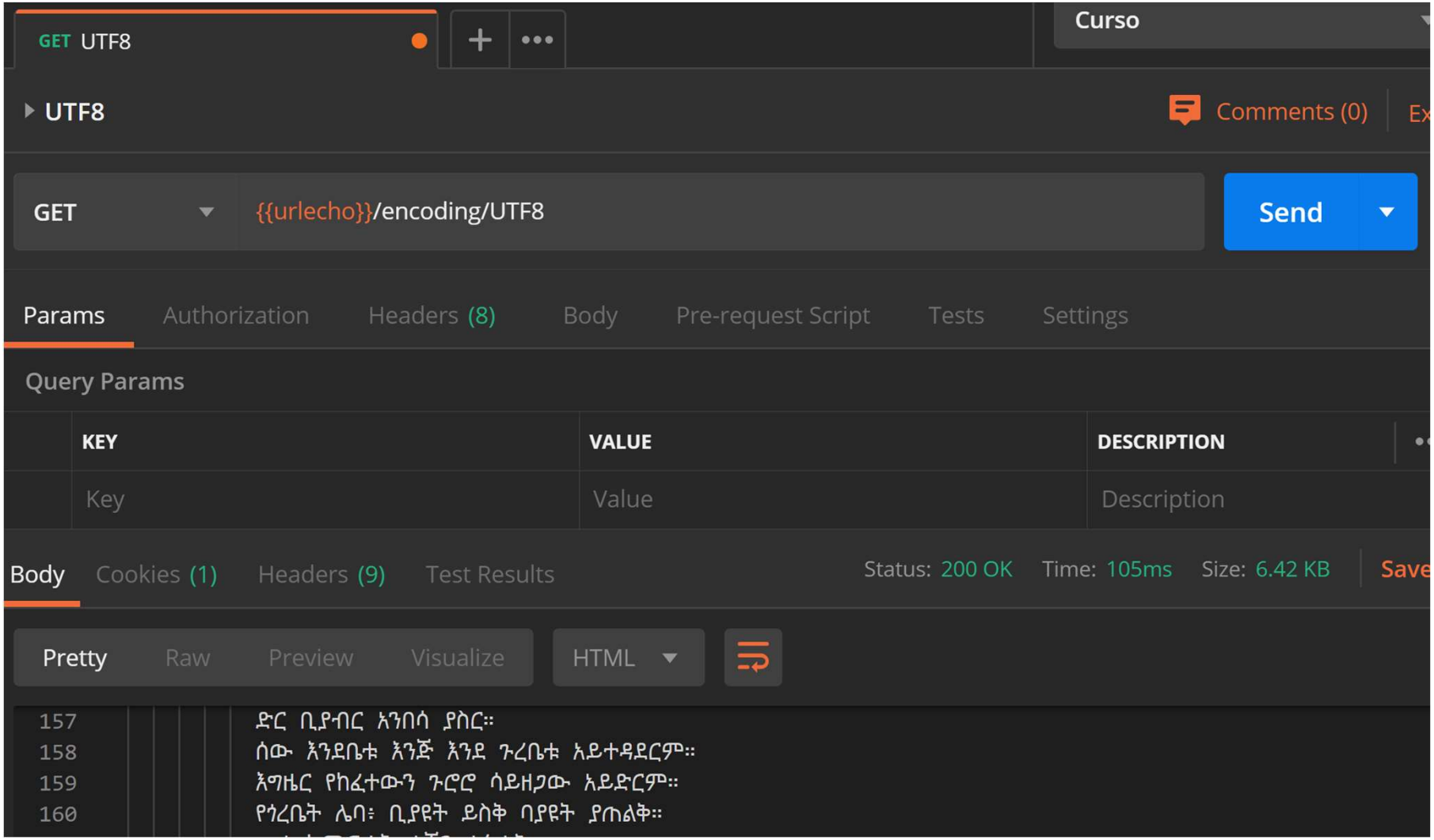
UTF8

- Si creamos una aplicación web multilenguaje podríamos utilizar caracteres diferentes a los utilizados en inglés (tildes, ñ, etc.), con lo cual podría ser útil comprobar si las respuestas de nuestros servicios web se comportan correctamente ante caracteres de este tipo



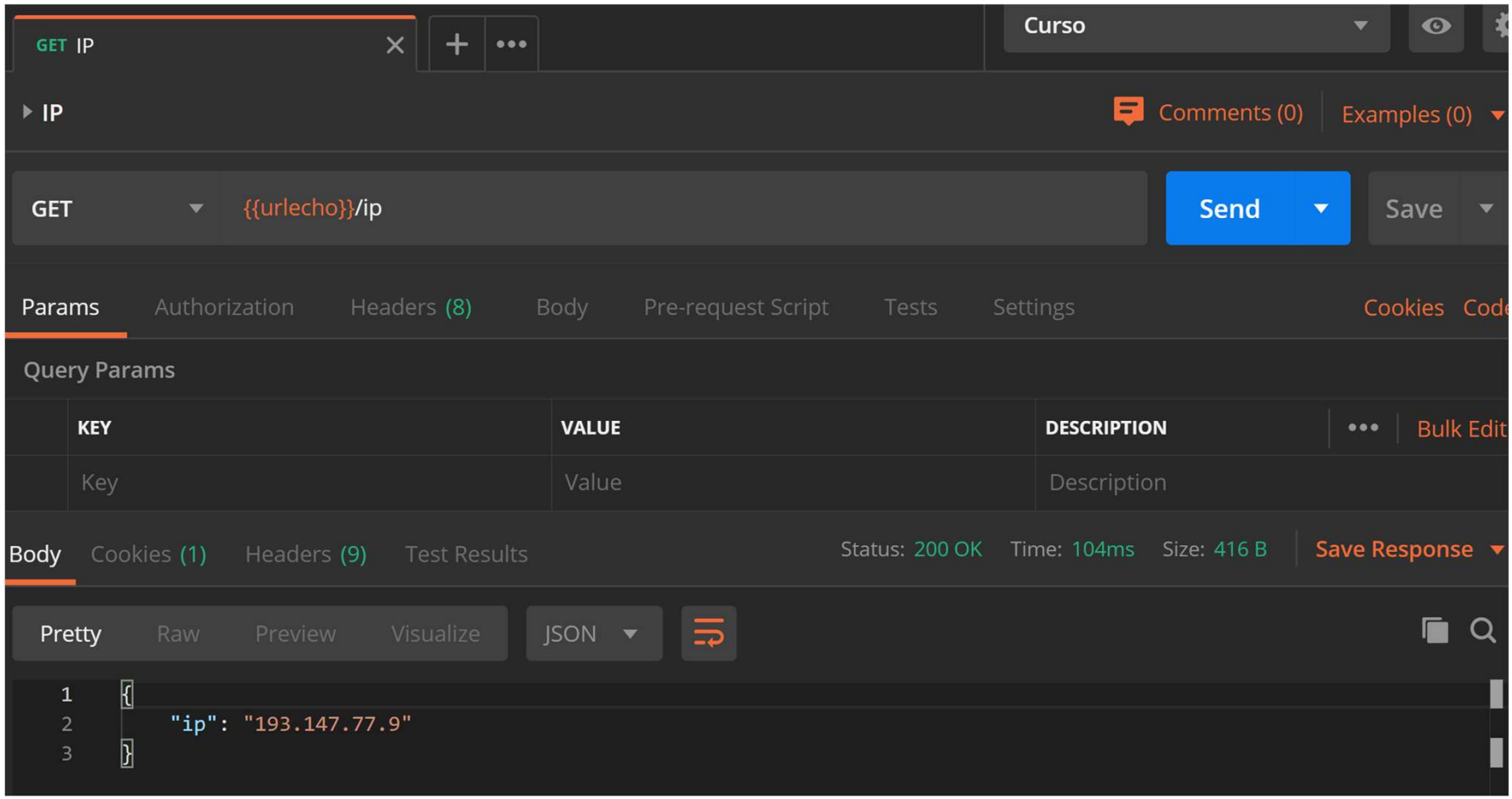
UTF8

- Probamos la utilidad:



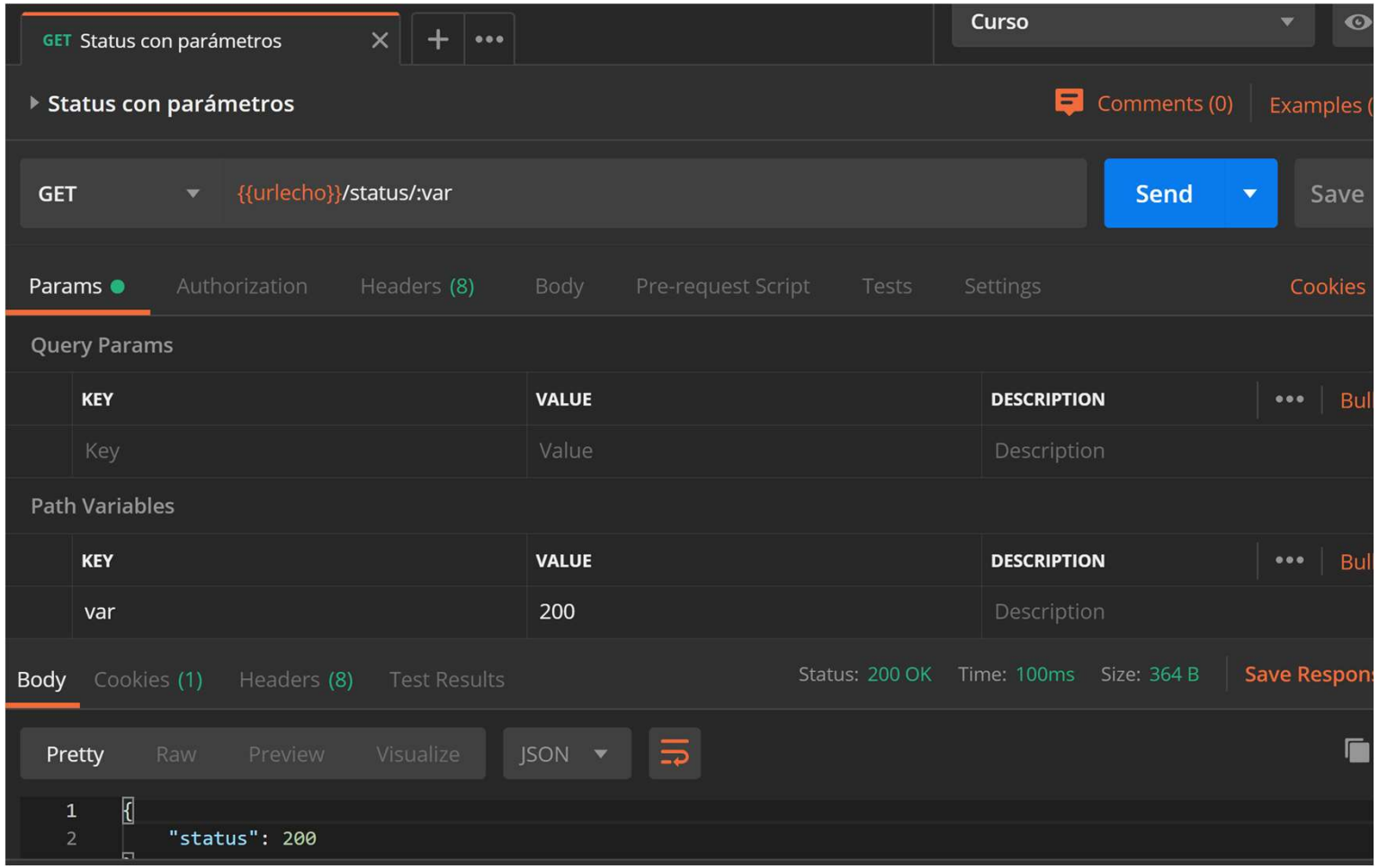
IP

- Si queremos saber la IP de la máquina que está enviando la solicitud podemos probar la utilidad IP:



Parámetros en llamada a servicios

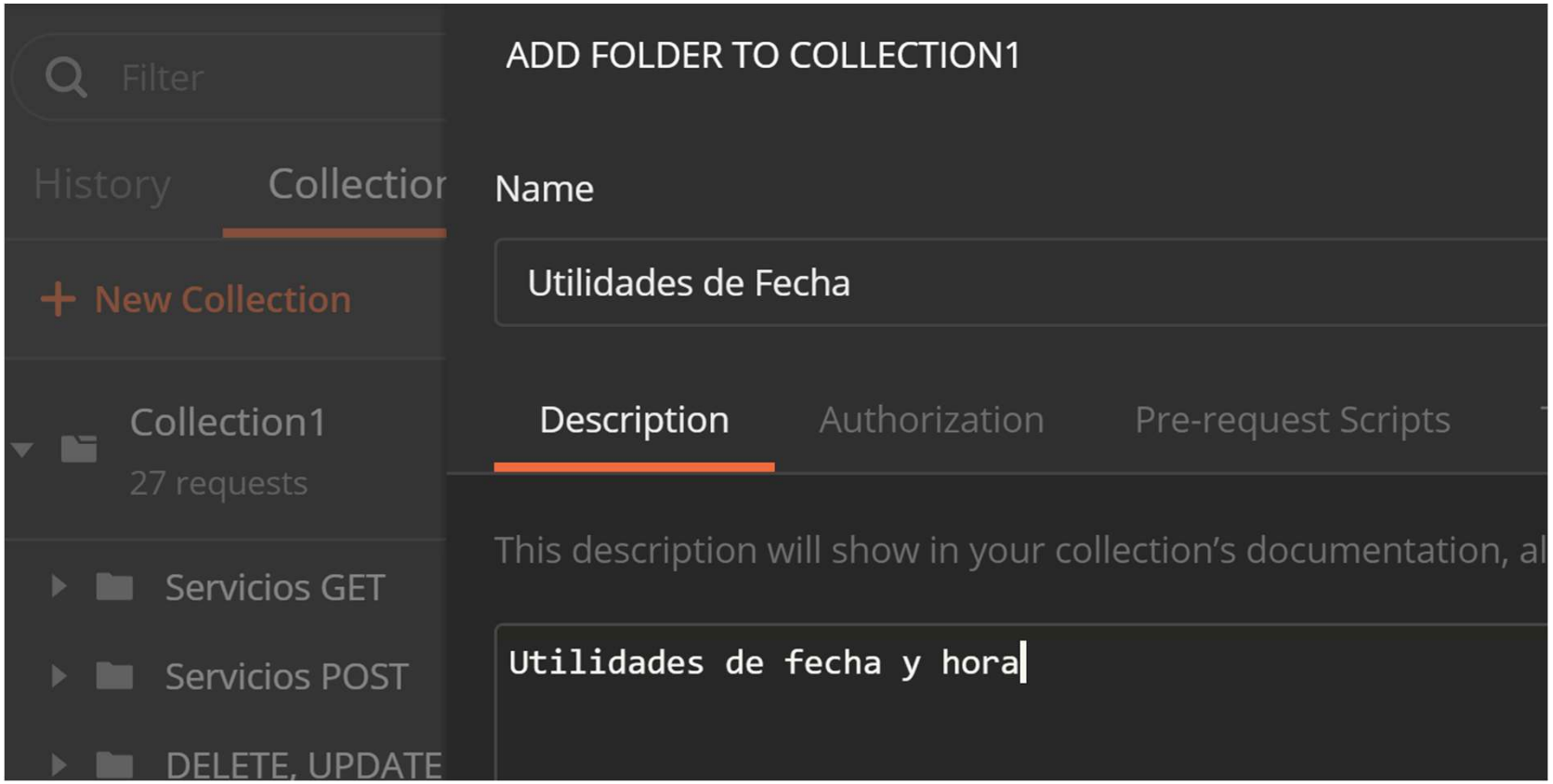
- Podemos utilizar parámetros en las llamadas a los servicios con el uso de los dos puntos (:)



Utilidades de fecha y hora

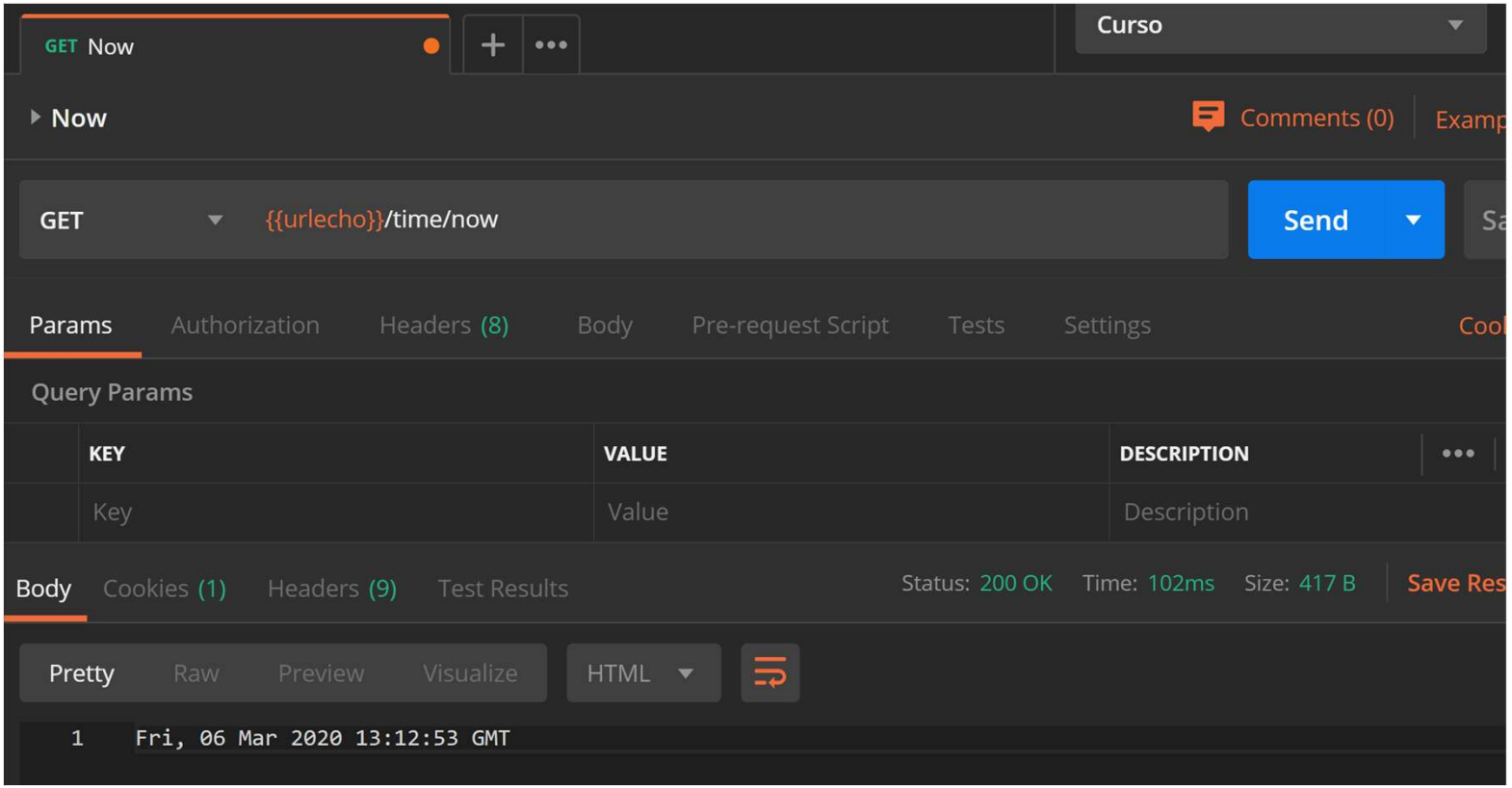
Utilidades fecha y hora

- Vamos a crear una nueva carpeta en nuestra colección:



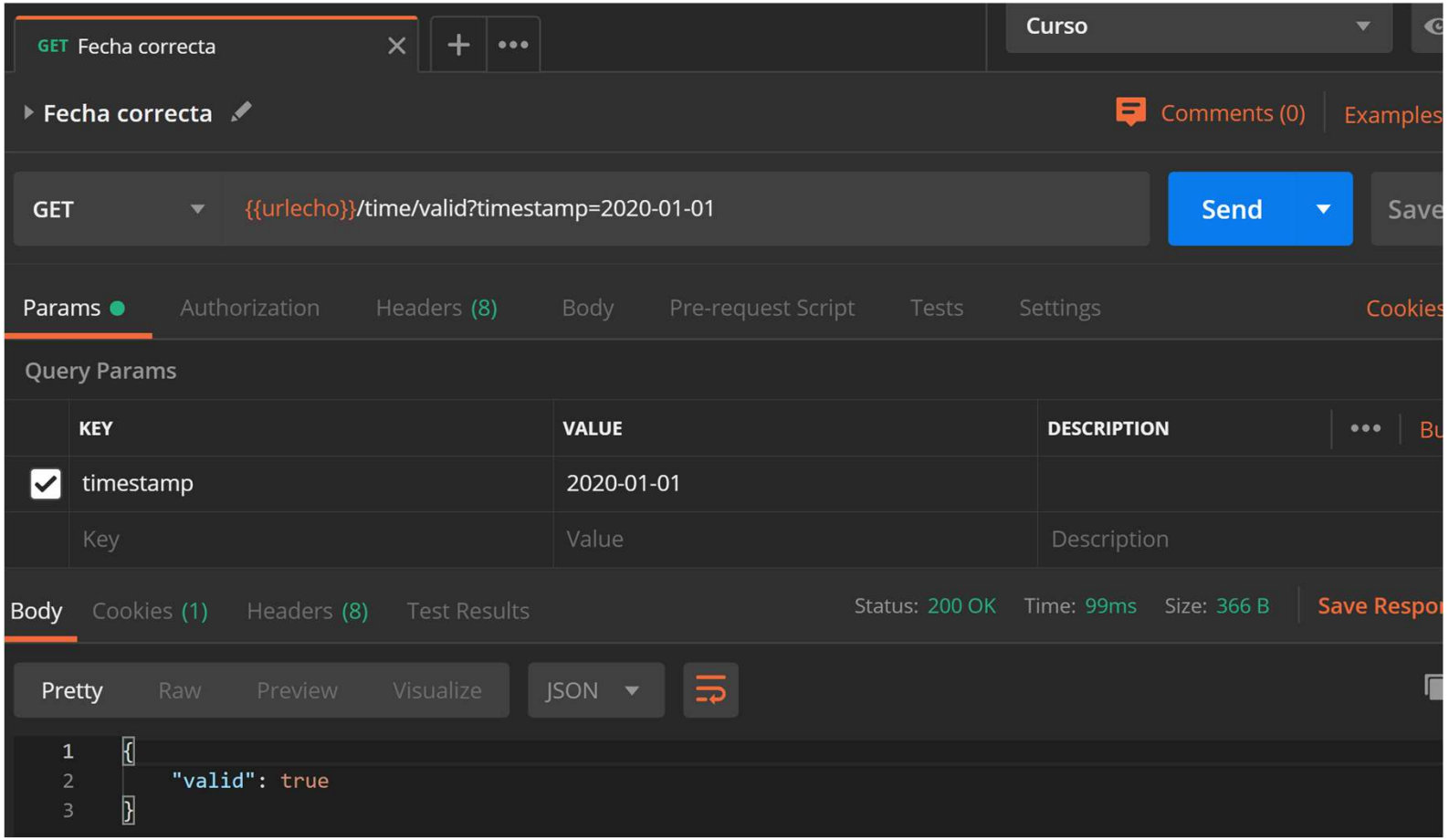
Now

- En muchas servicios web es necesario saber la fecha y hora en la que se ejecuta o solicita el servicio
- Hay una utilidad para obtener esos datos:

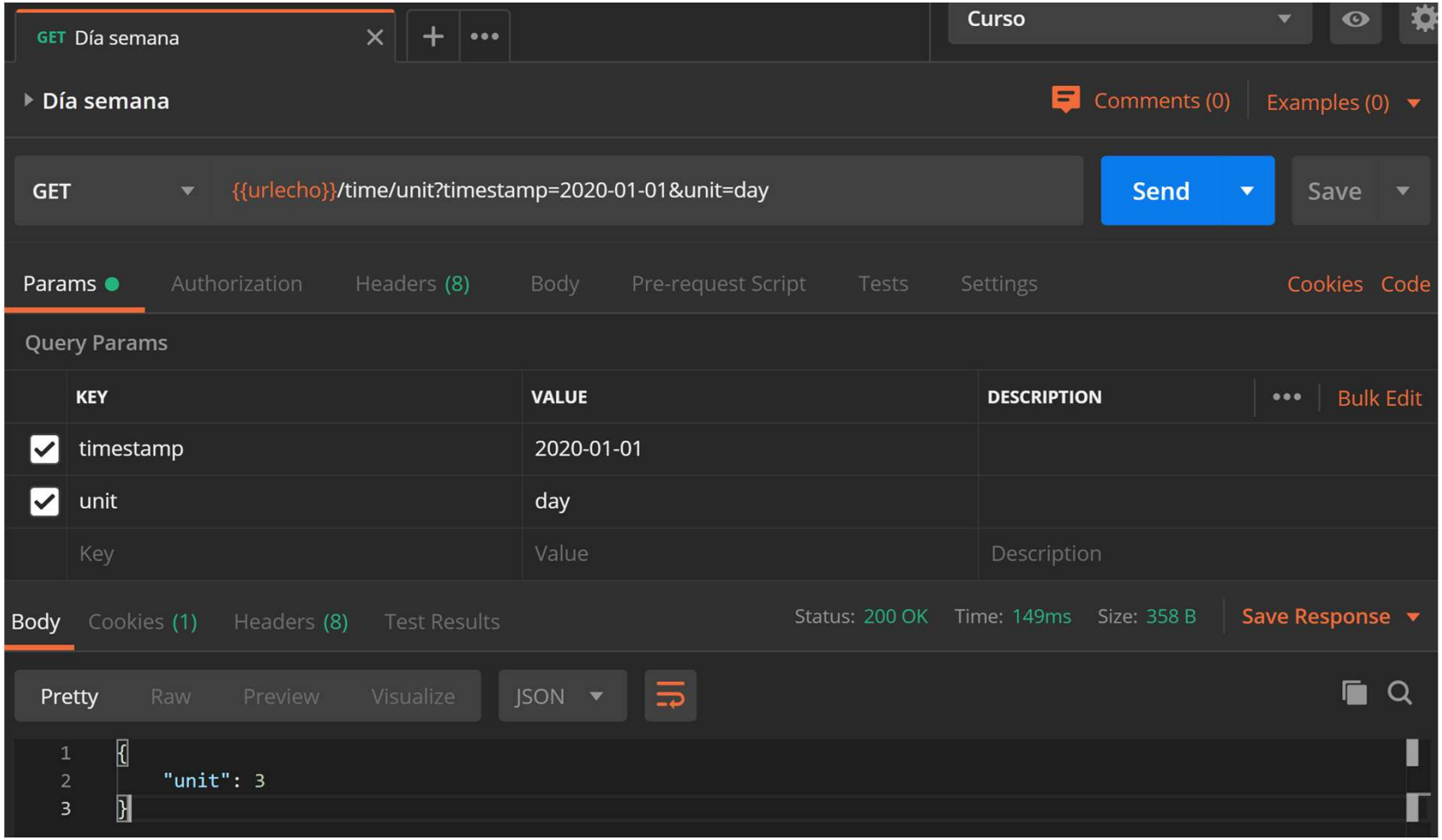


Valid

- La siguiente utilidad nos permite validar si el formato de una fecha es correcto o no (probamos con un día 32):

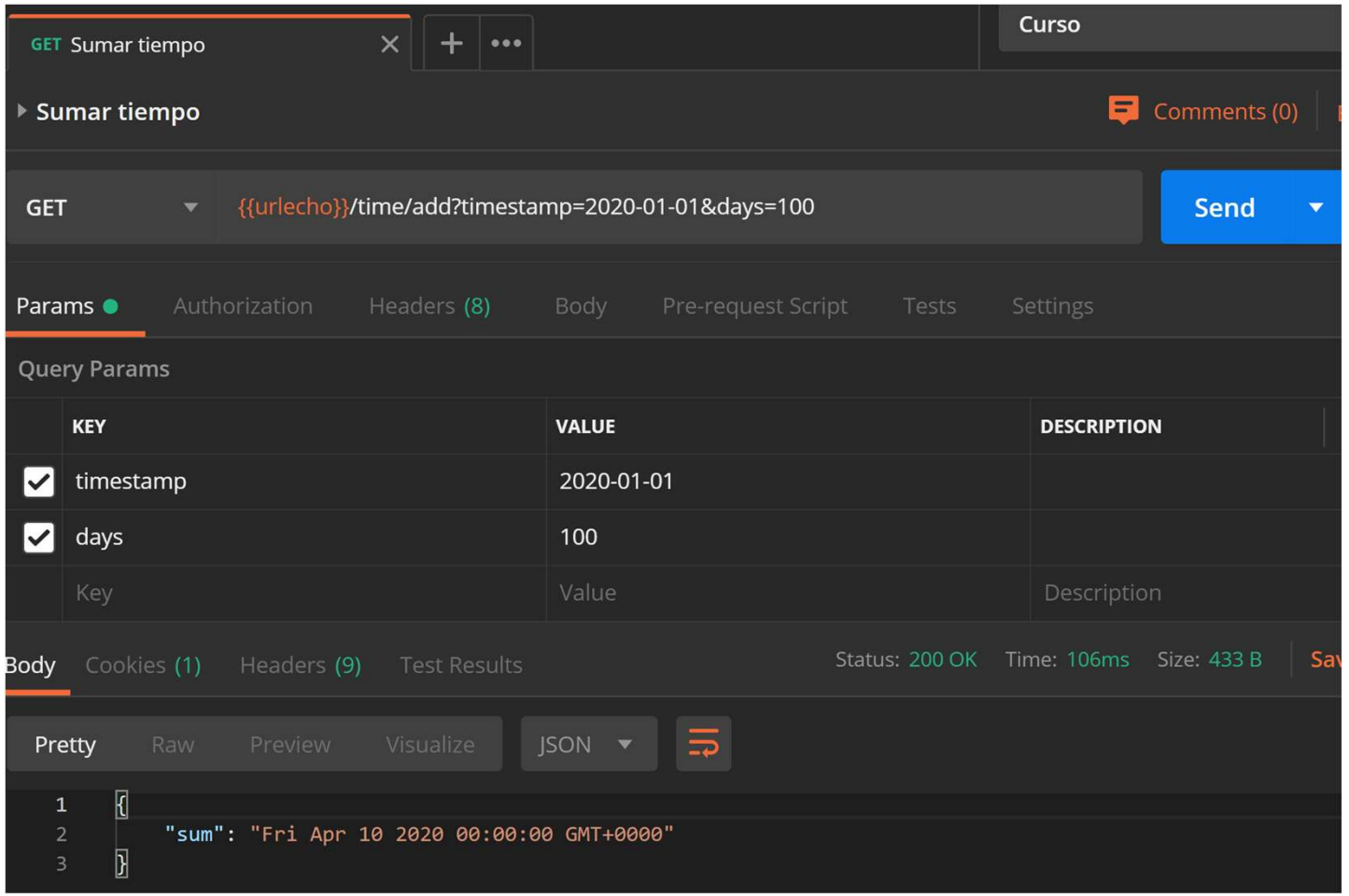


- Utilidad para saber el día de la semana de una fecha en concreto:



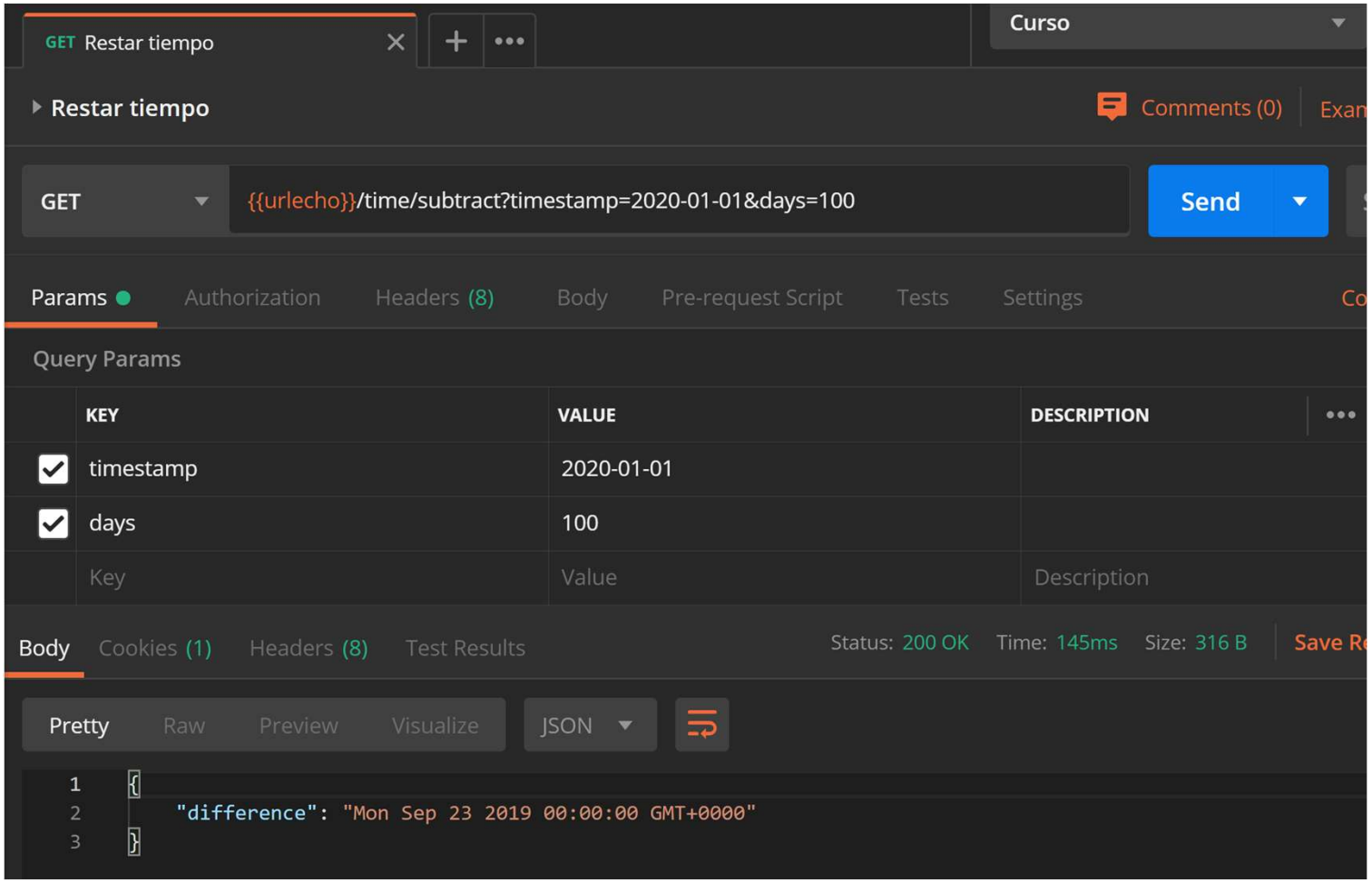
Add

- Utilidad para añadir tiempo a una fecha en concreto (years, months, days, hours, minutes, seconds):



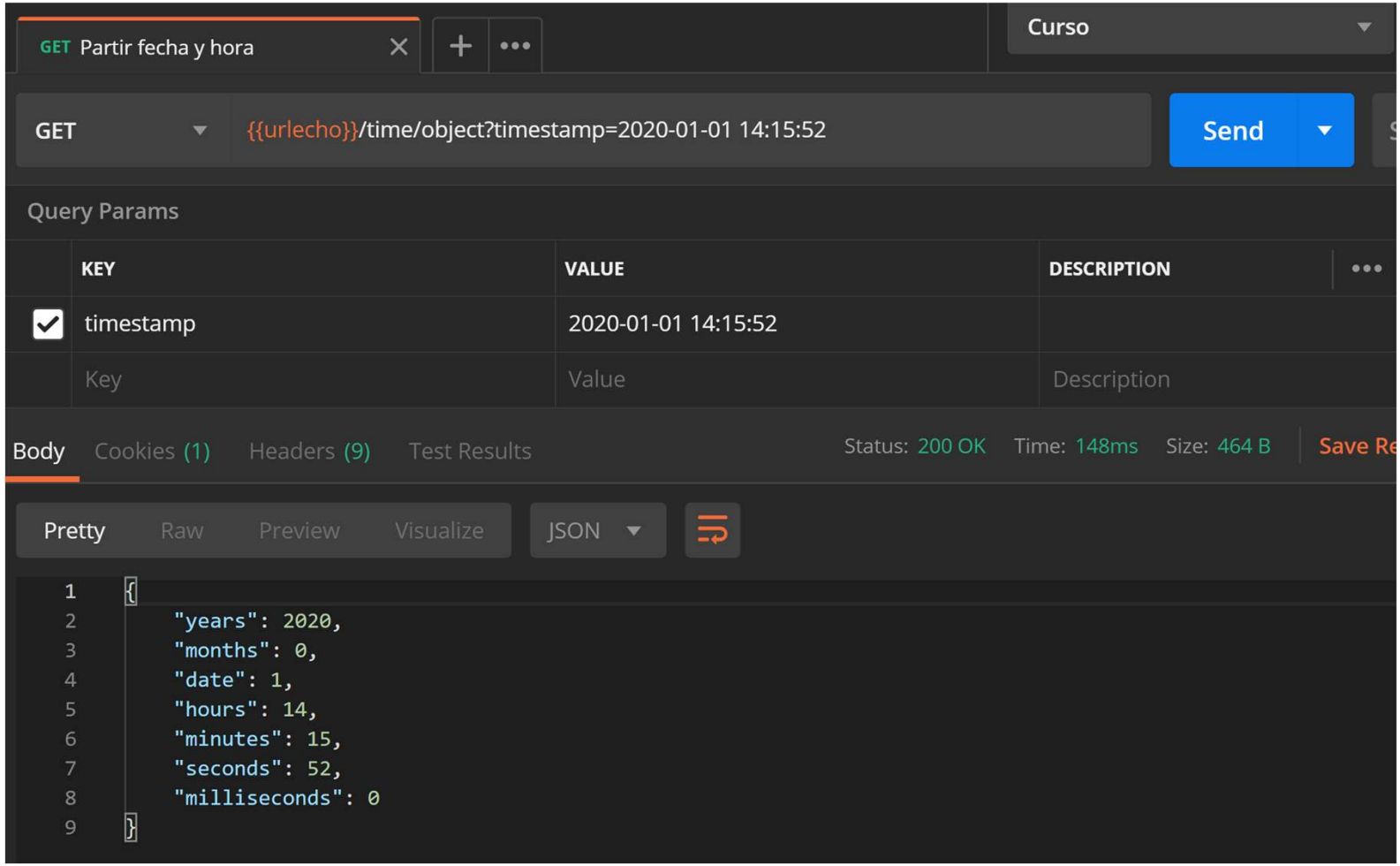
Subtract

- Duplicamos la anterior petición y modificamos el add por el subtract para restar tiempo a una fecha:



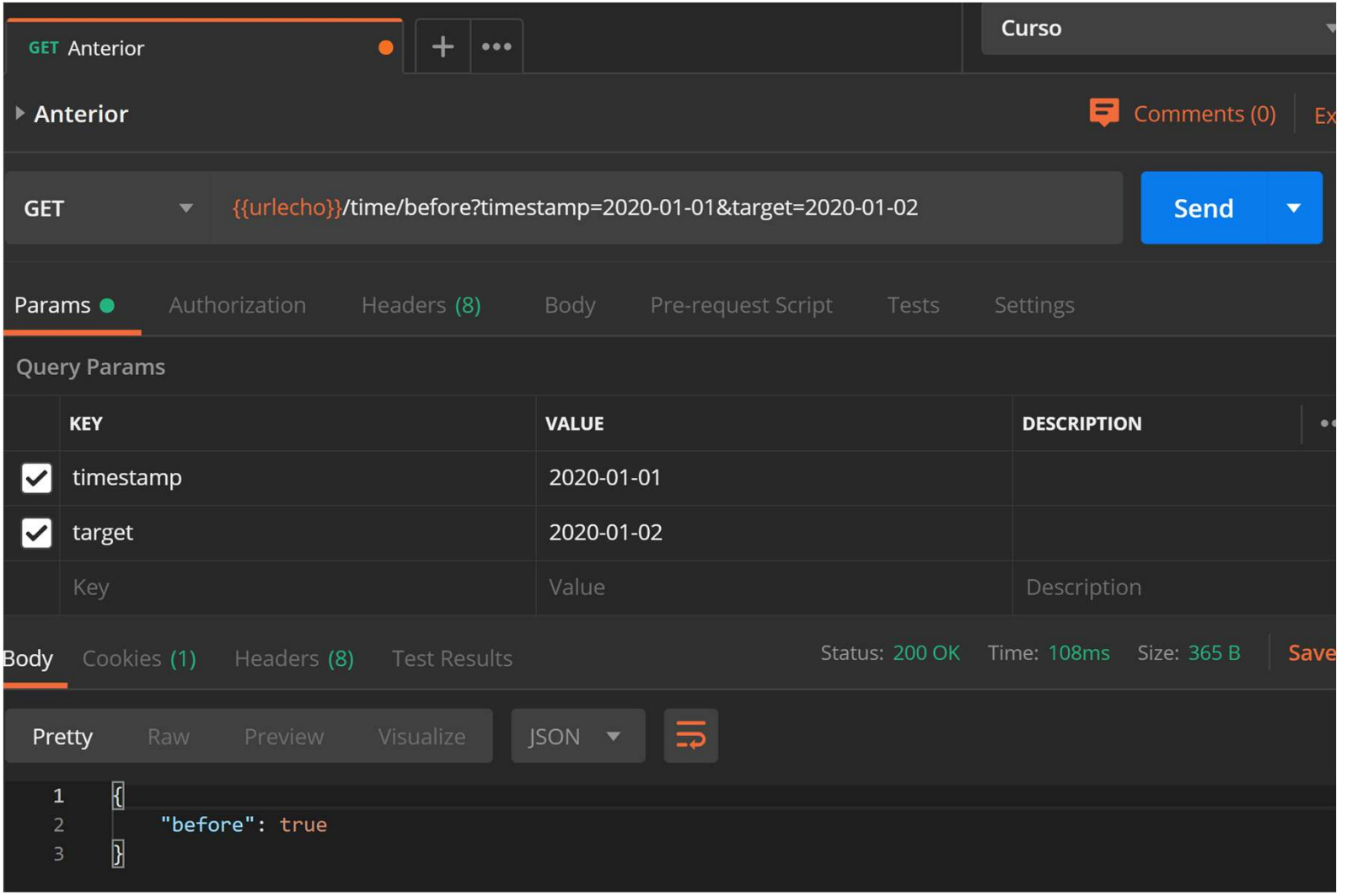
Object

- Utilidad para dividir la fecha y hora en cada una de sus unidades (meses cuentan desde el 0):



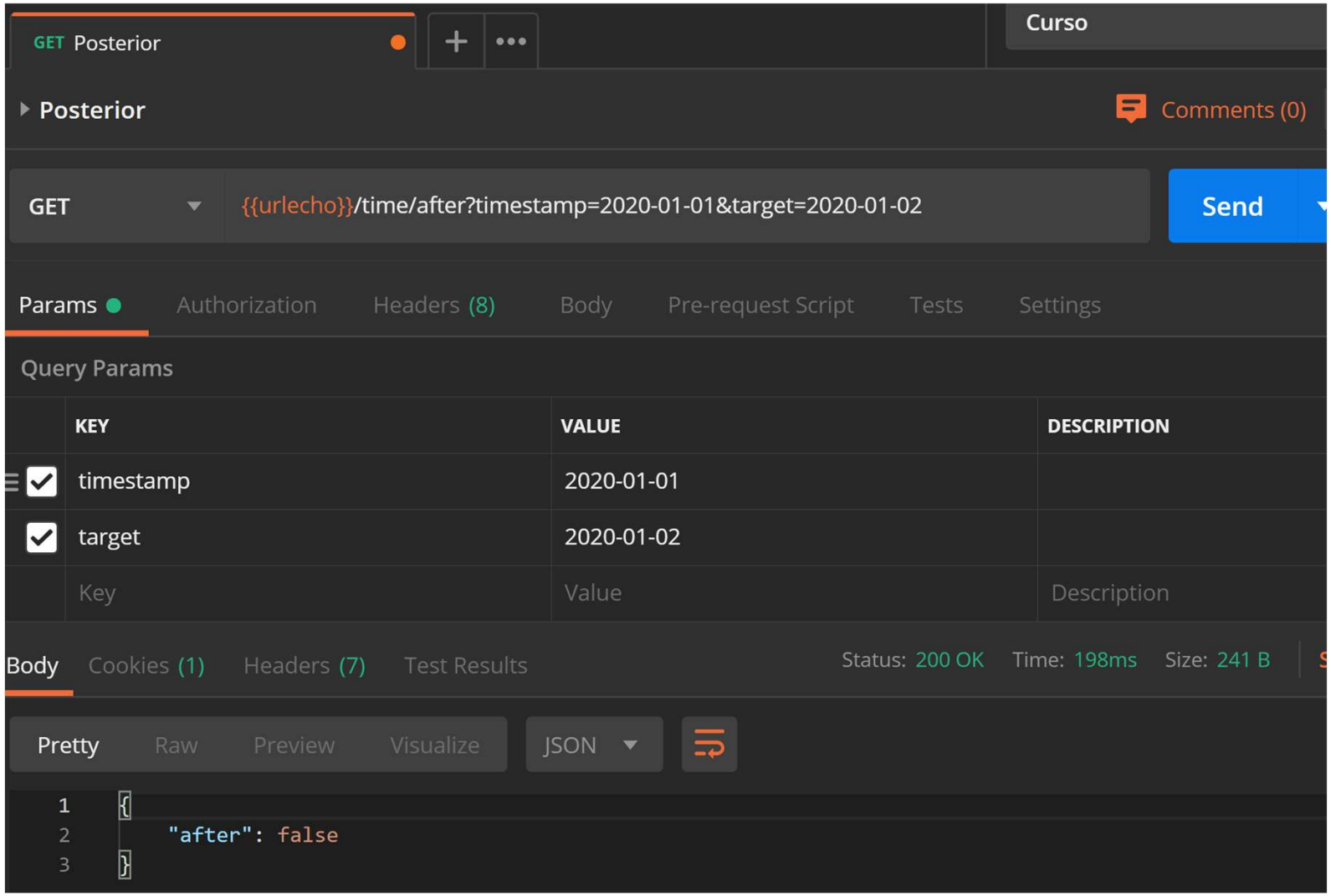
Before

- Utilidad para saber si una fecha es anterior a otra:



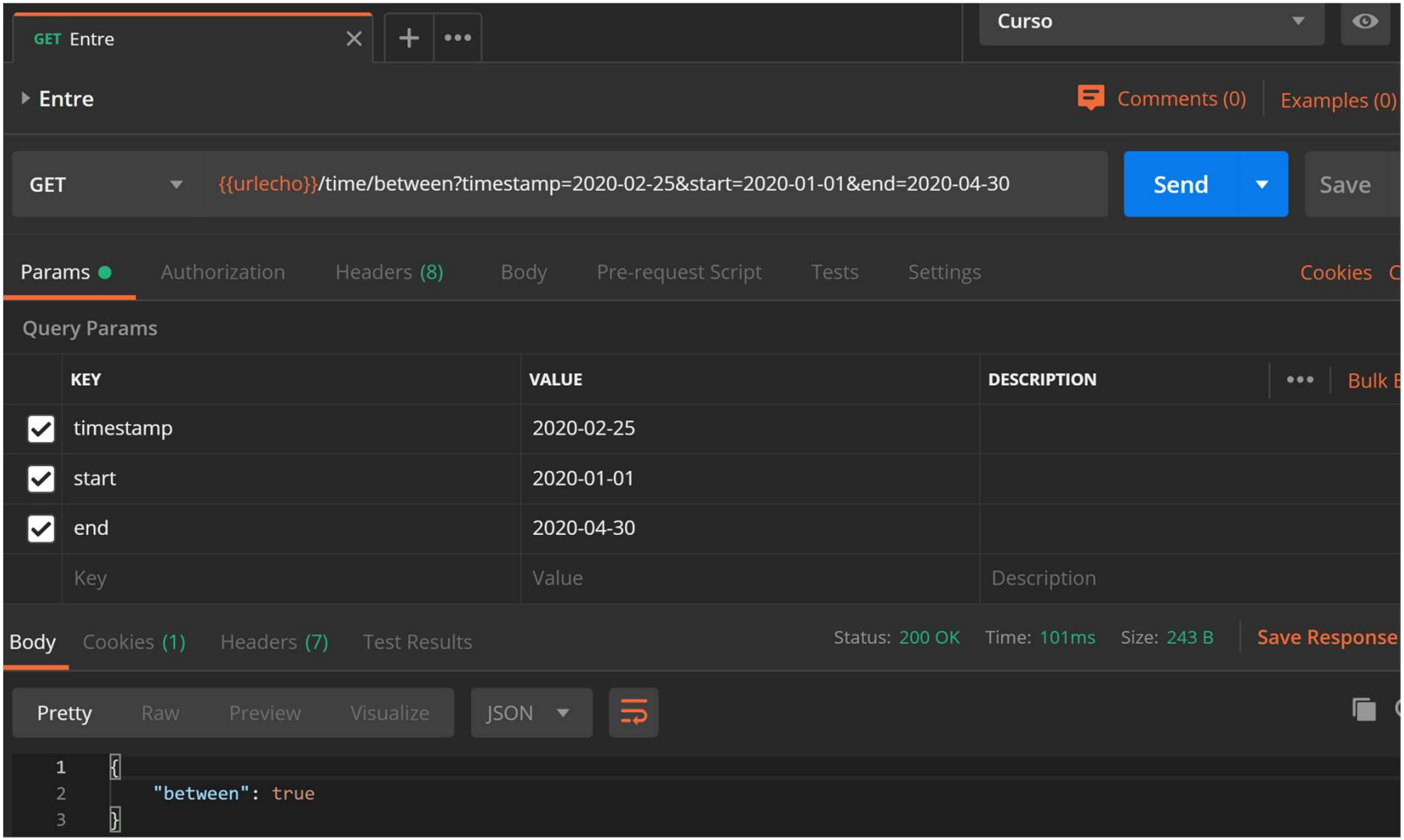
After

- Utilidad para saber si una fecha es posterior a otra:



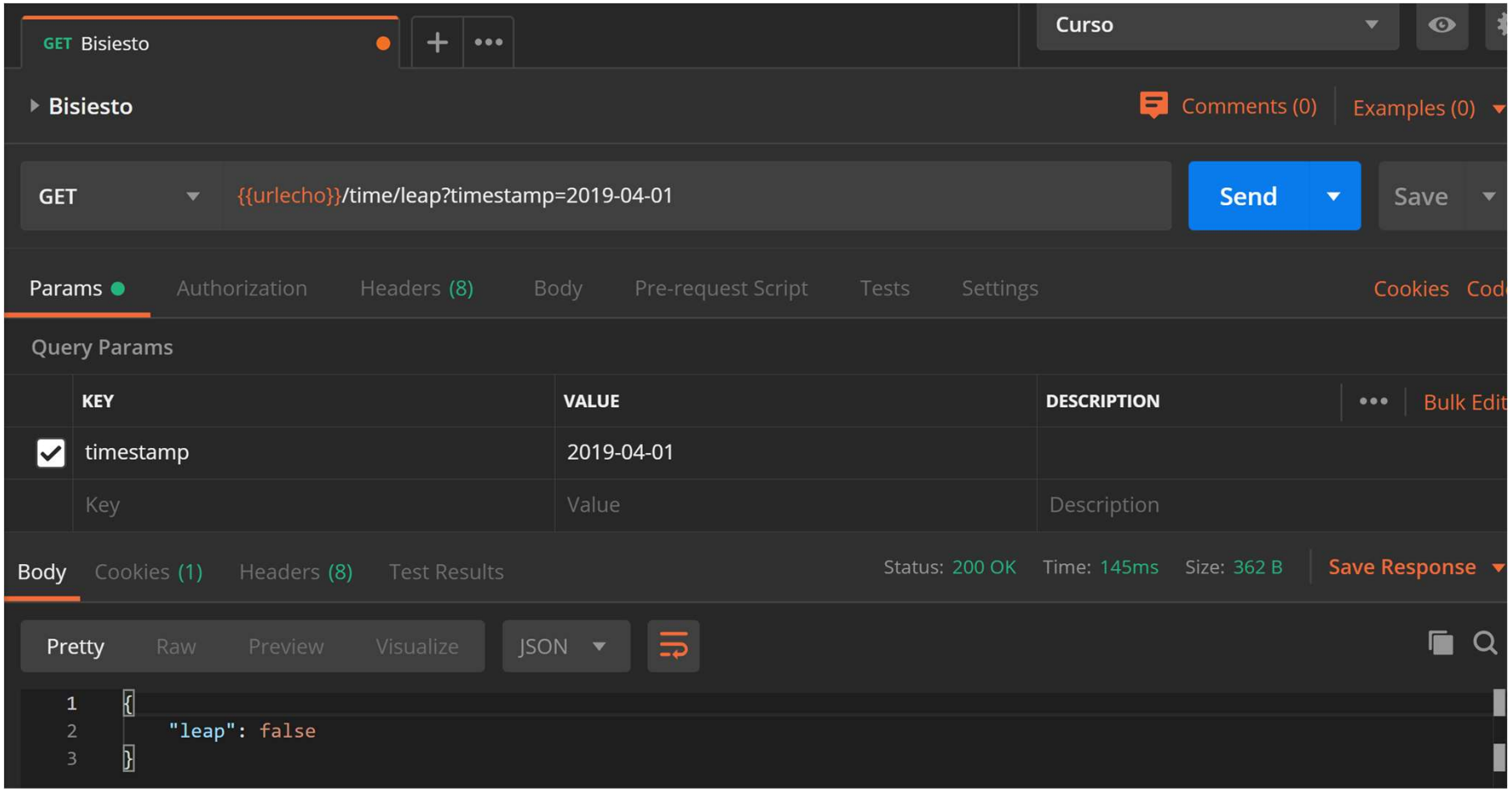
Between

- Utilidad para saber si una fecha está entre dos fechas:



Leap

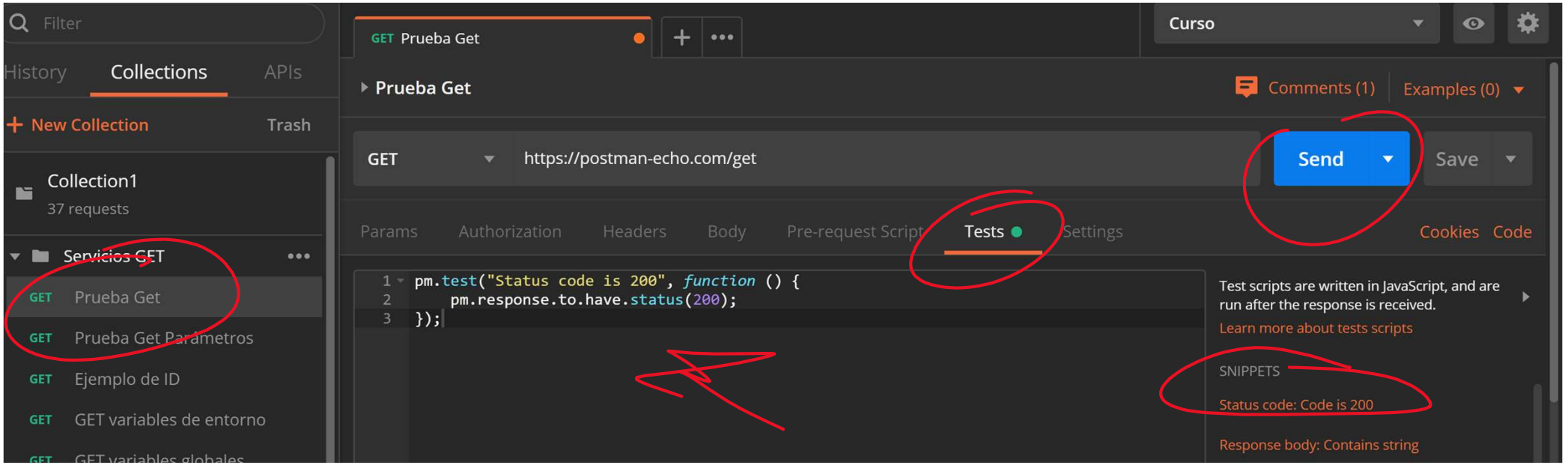
- Utilidad para saber si una fecha pertenece a un año bisiesto:



Tests y Scripts

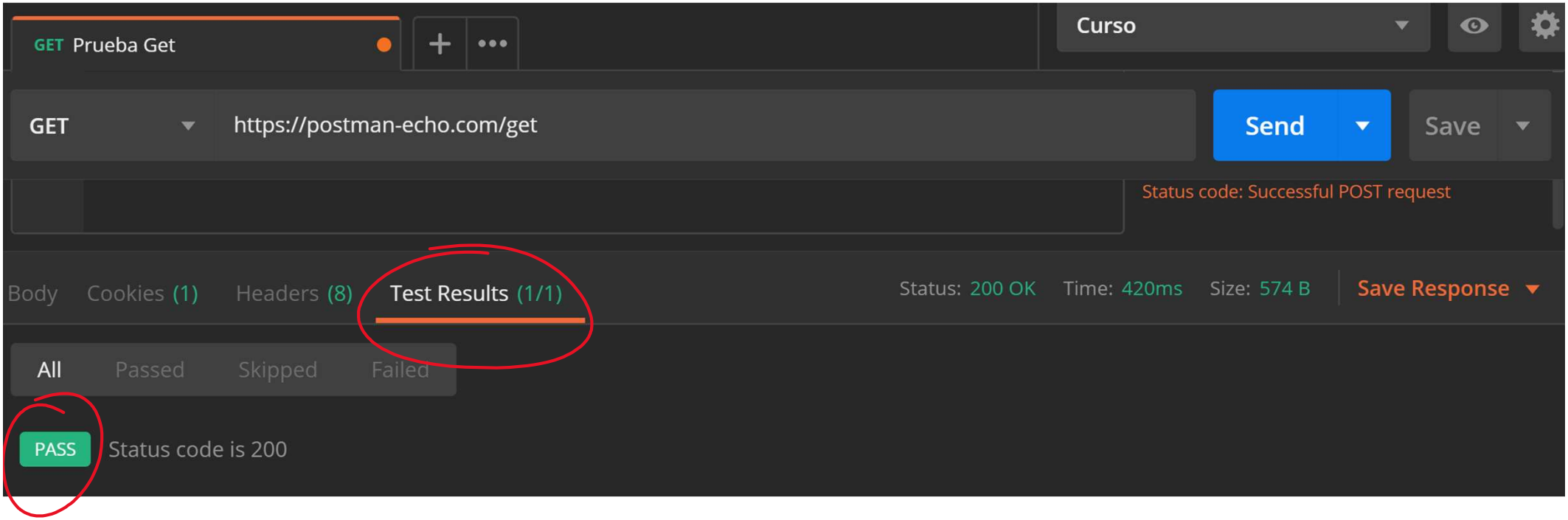
Primer test básico

- Postman está hecho principalmente para realizar test sobre nuestros servicios web (en Javascript):



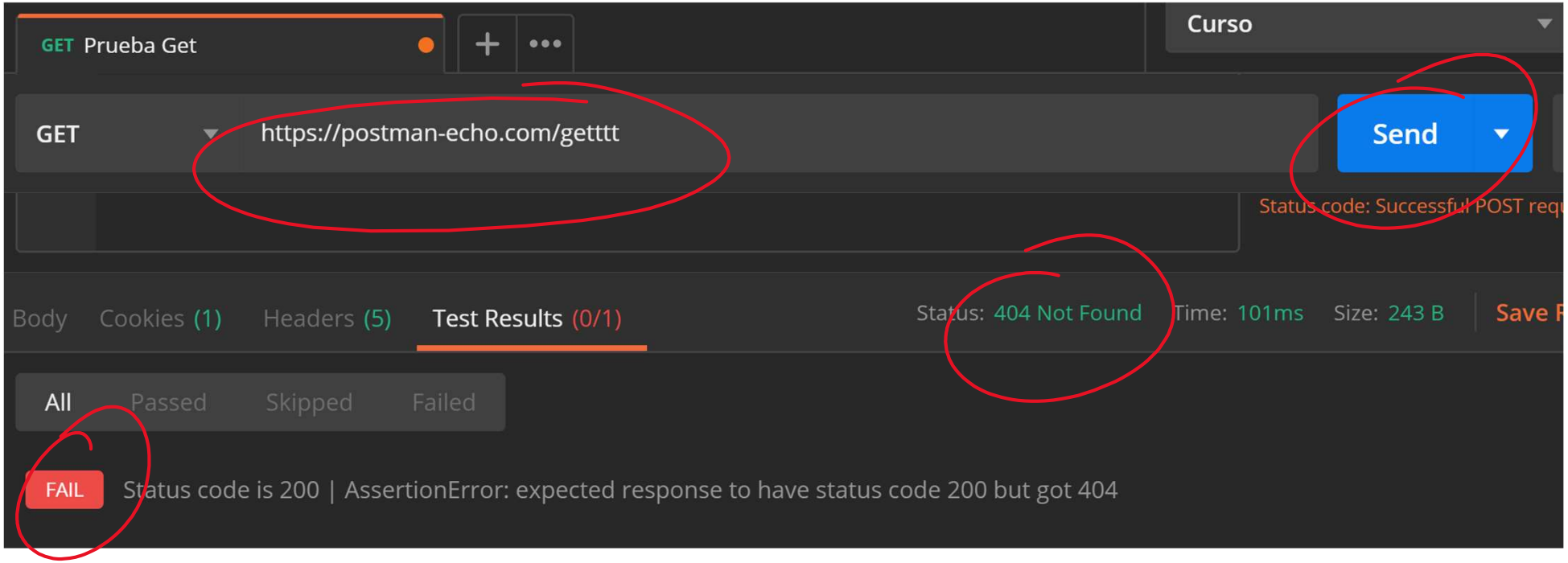
Primer test básico

- Al probar la petición (send) veremos una nueva pestaña que nos muestra si el test se ha realizado con éxito:



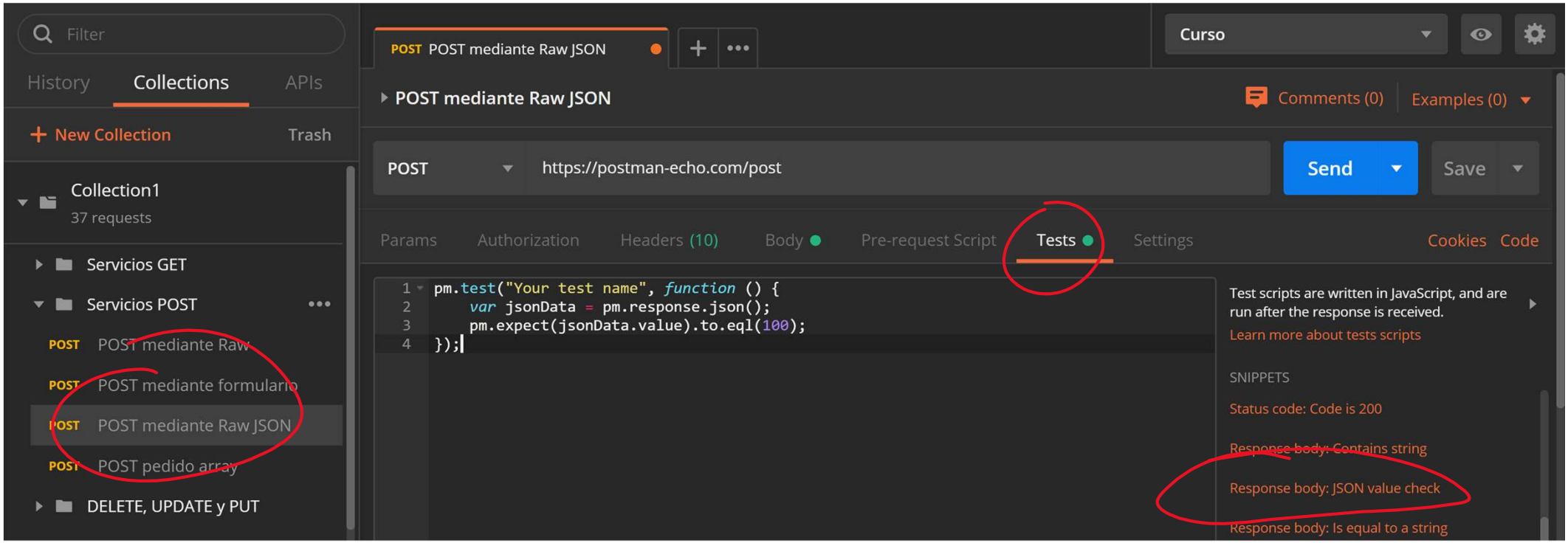
Primer test básico

- Si por ejemplo, escribimos un servicio que no existe, el status no será 200 si no 404 y el test nos indicará el fallo:



Test de JSON

- Podemos realizar más de una prueba en cada petición. Por ejemplo:



Test de JSON

- Modificamos el test:

POST POST mediante Raw JSON

POST mediante Raw JSON

POST https://postman-echo.com/post

Params Authorization Headers (10) Body Pre-request Script Tests Settings

```
pm.test("Resultados del JSON - Nombre", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.data.nombre).to.eql("David");
    pm.expect(jsonData.data.apellido).to.eql("Granada");
});

pm.test("Resultados del JSON - Edad", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.data.edad).to.eql(40);
});
```

Body Cookies (1) Headers (9) Test Results (2/2) Status: 200

Pretty Raw Preview Visualize JSON

```
{
    "args": {},
    "data": {
        "nombre": "David",
        "apellido": "Granada",
        "edad": 40
    },
```

Body Cookies (1) Headers (9) Test Results (2/2)

All Passed Skipped Failed

PASS Resultados del JSON - Nombre

PASS Resultados del JSON - Edad

Test de JSON

- Agregamos alguna otra comprobación:

Params | Authorization | Headers (10) | Body ● | Pre-request Script | Tests ● | Settings

```
pm.test("Resultados del JSON - Nombre", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.data.nombre).to.eql("David");
    pm.expect(jsonData.data.apellido).to.eql("Granada");
});

pm.test("Resultados del JSON - Edad", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.data.edad).to.eql(40);
});

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

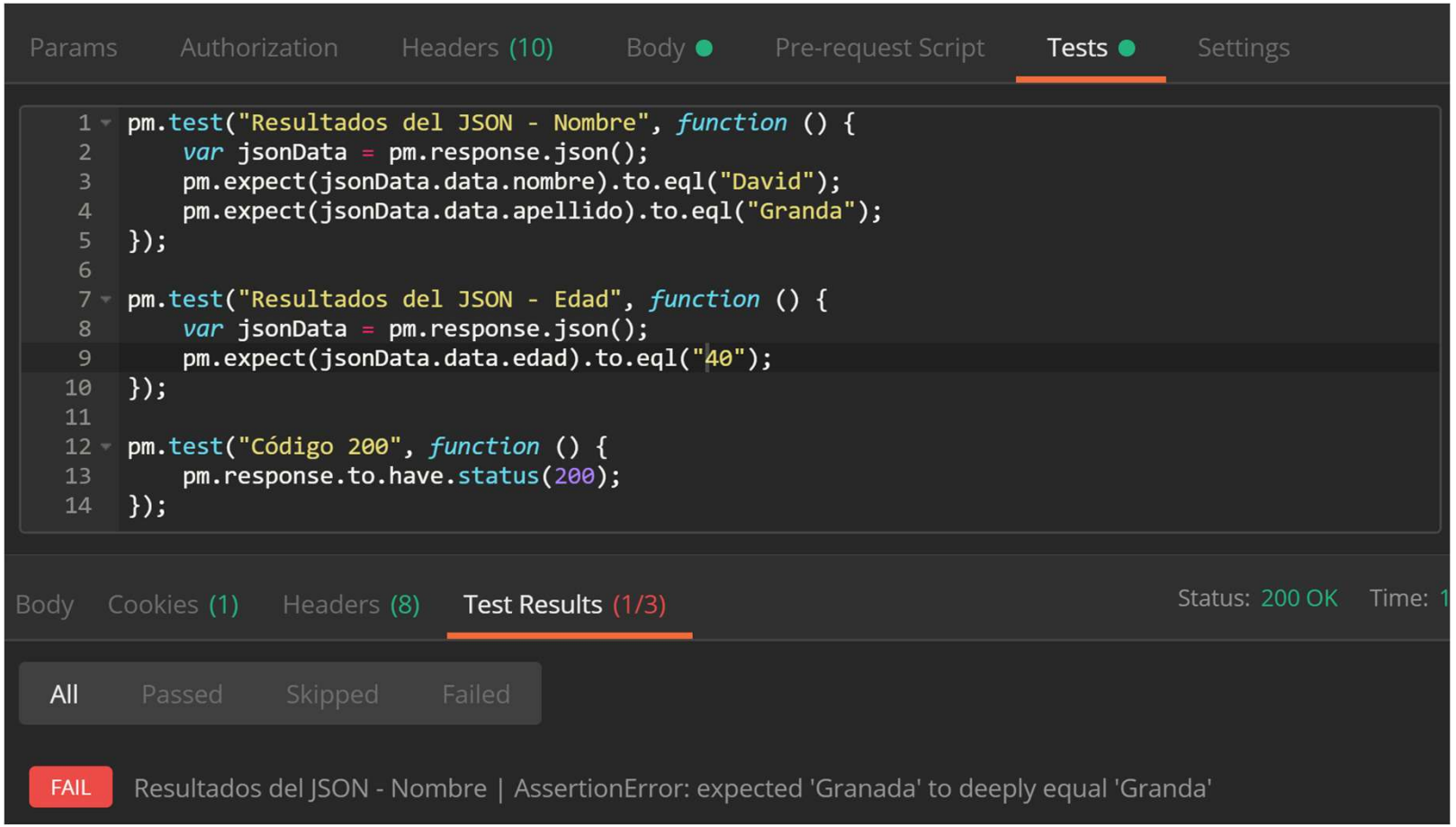
Body | Cookies (1) | Headers (8) | Test Results (3/3) | Status: 200 OK

All | Passed | Skipped | Failed

PASS Resultados del JSON - Nombre

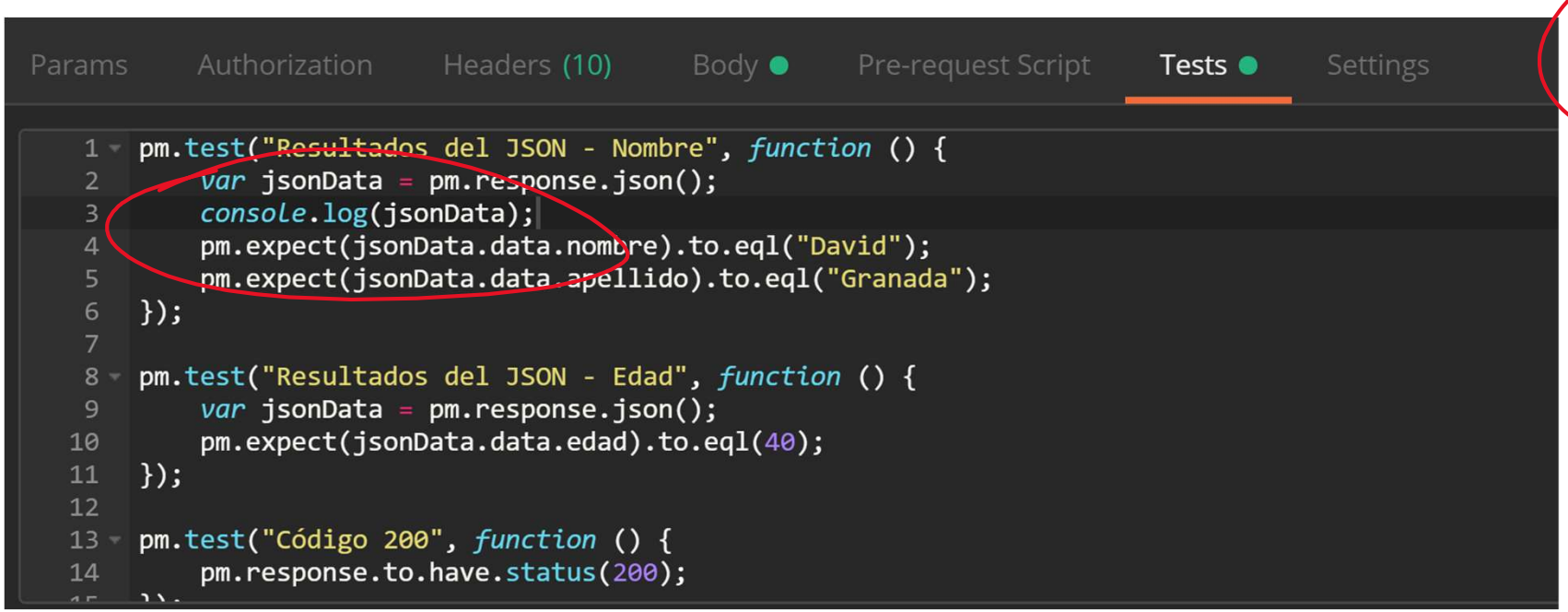
Test de JSON

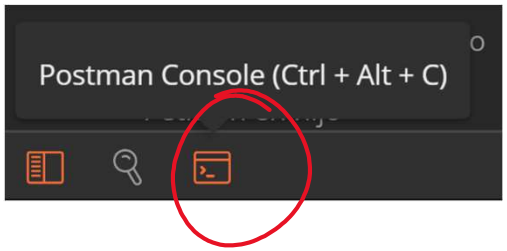
- Podemos probar a cometer errores para ver el resultado:



Test de JSON

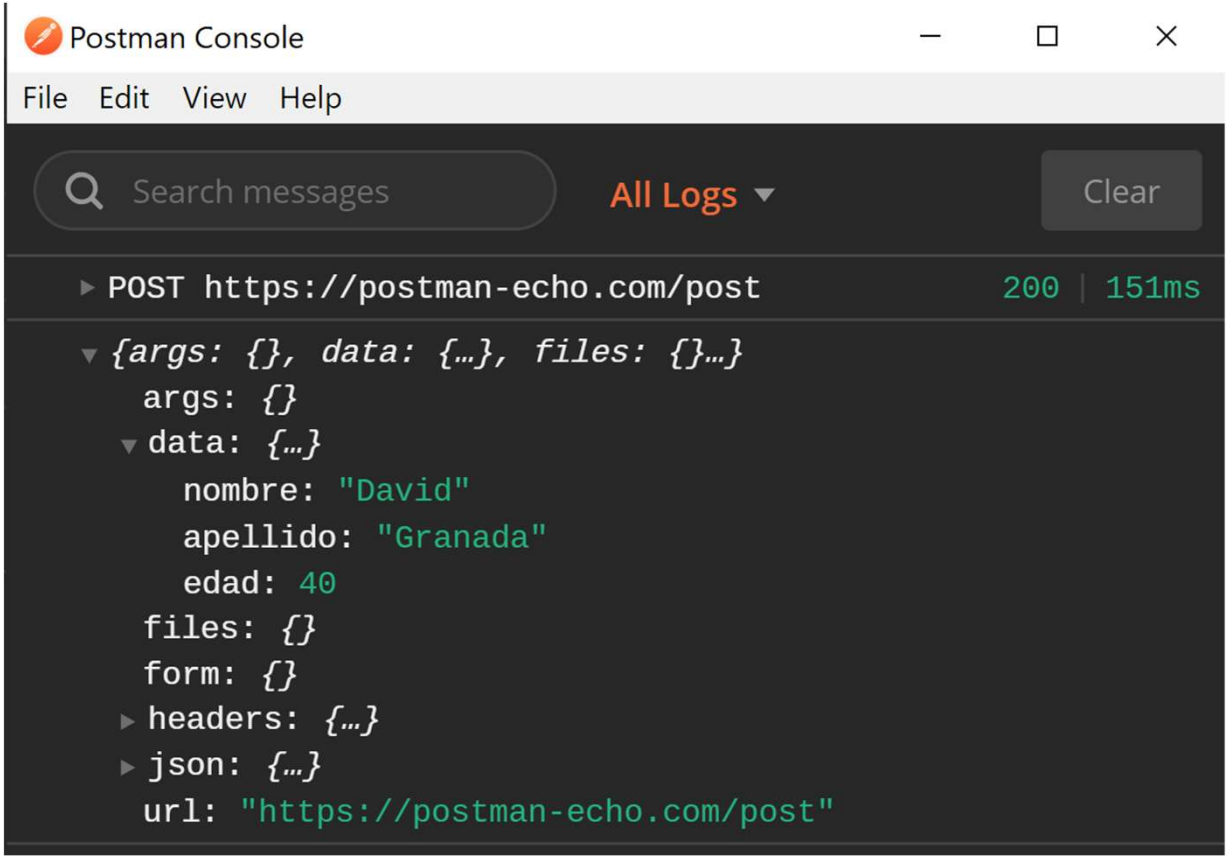
- Es posible imprimir en la consola un log de lo que queramos, por ejemplo:





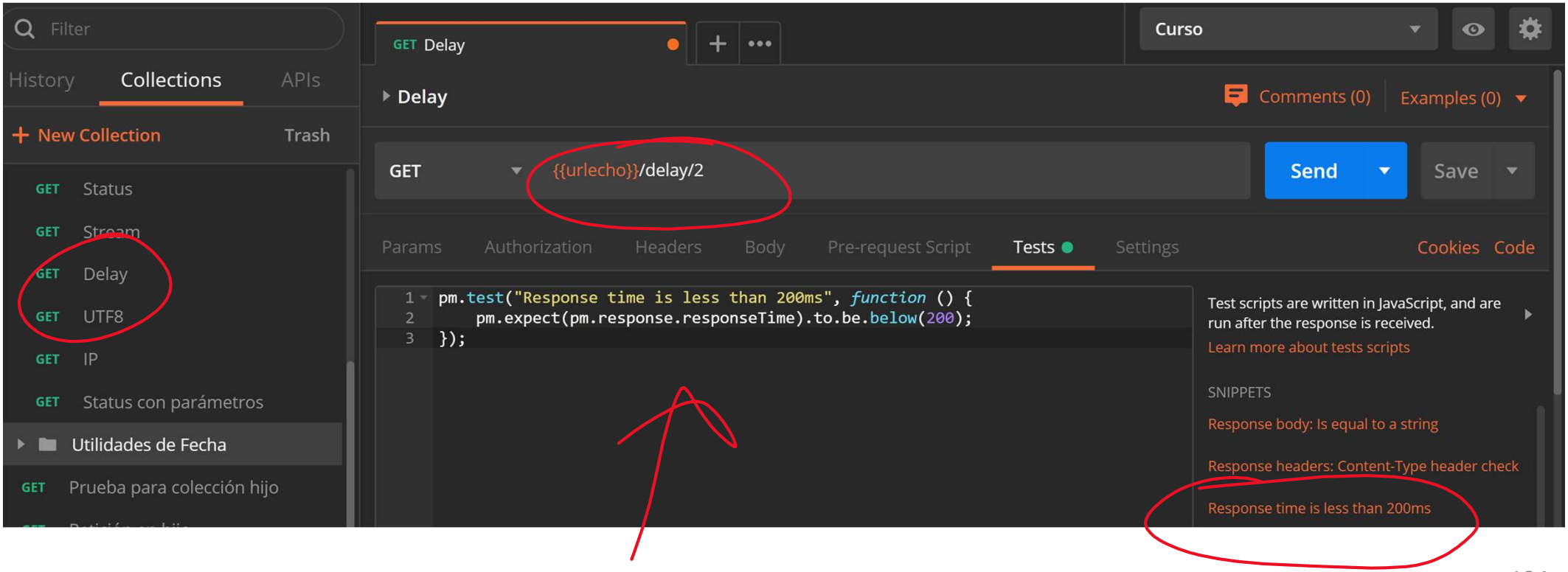
Test de JSON

- Hacemos un clear de la consola, y si ejecutamos de nuevo vemos el contenido de la variable llamada jsonData:



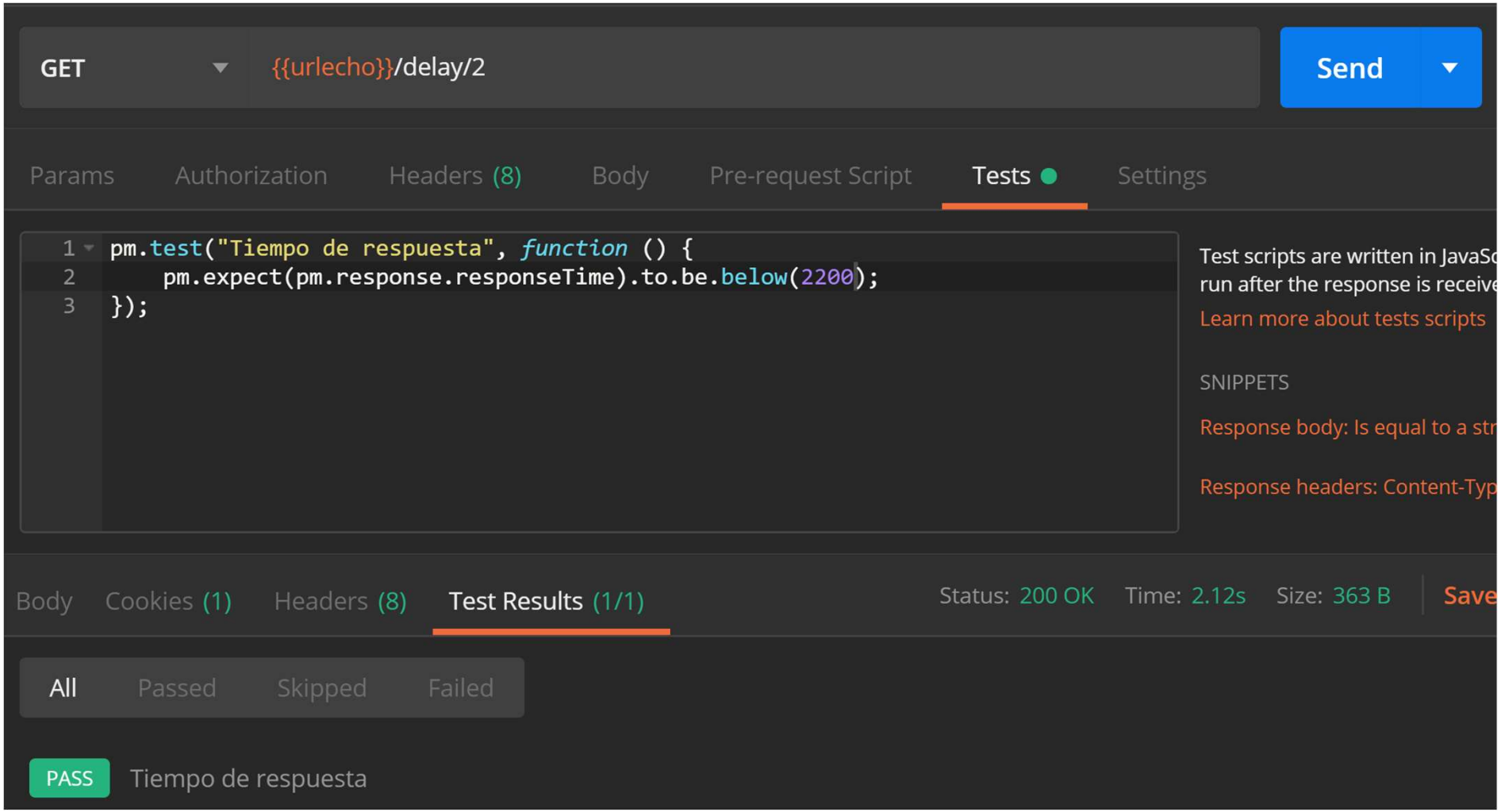
Test de Delay

- Simulemos que queremos verificar que nuestro servicio web no puede tardar más de X milisegundos en devolver lo que haya pedido el usuario:



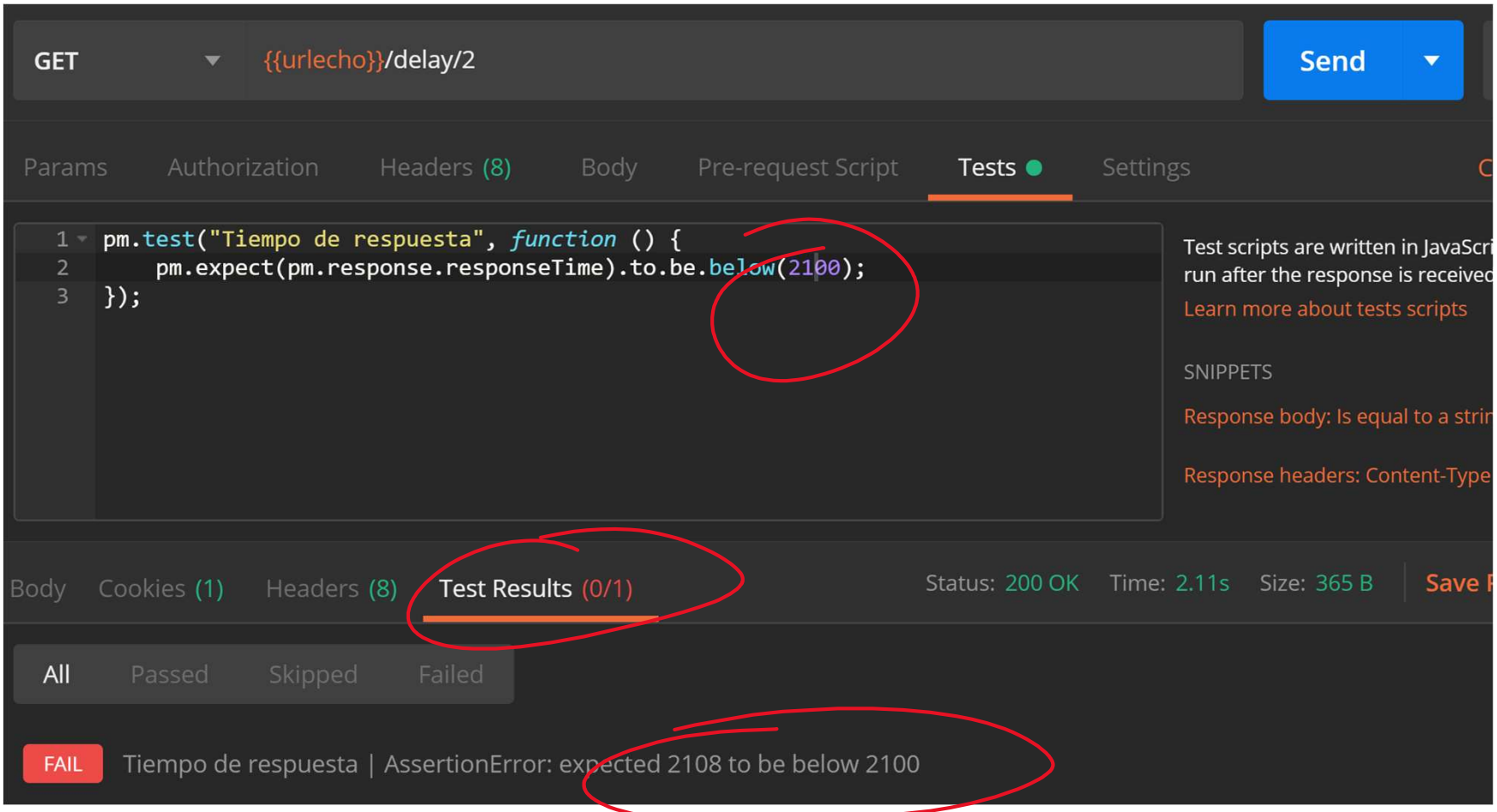
Test de Delay

- Modificamos el código proporcionado por el snippet:



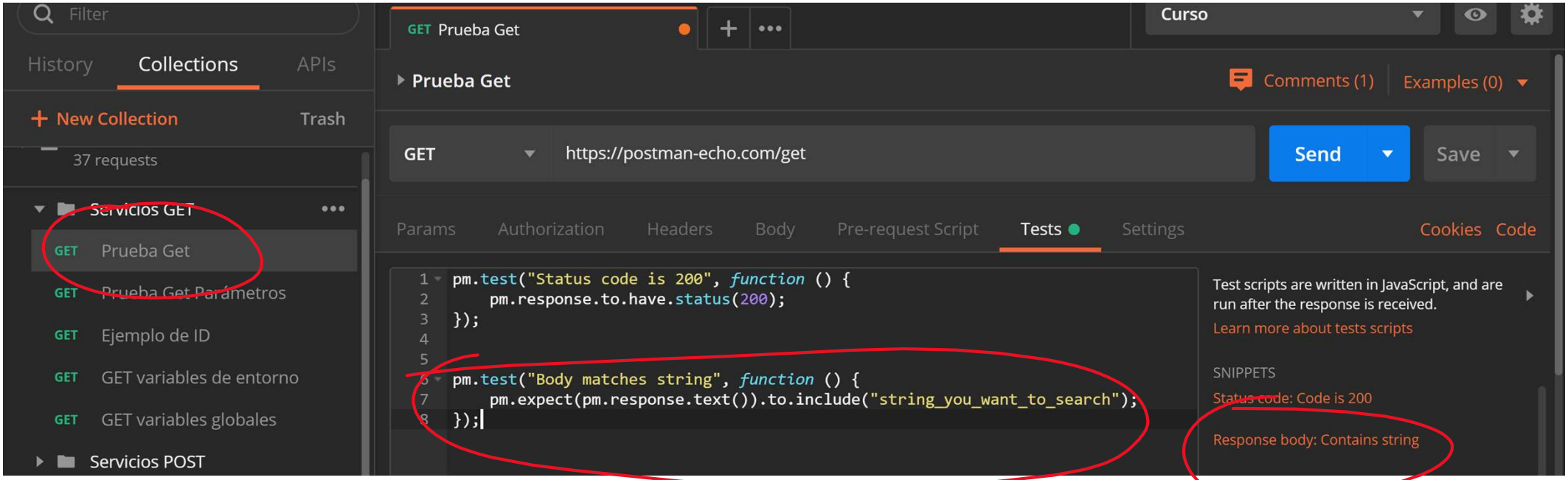
Test de Delay

- Si disminuimos el tiempo puede que no se supere el test:



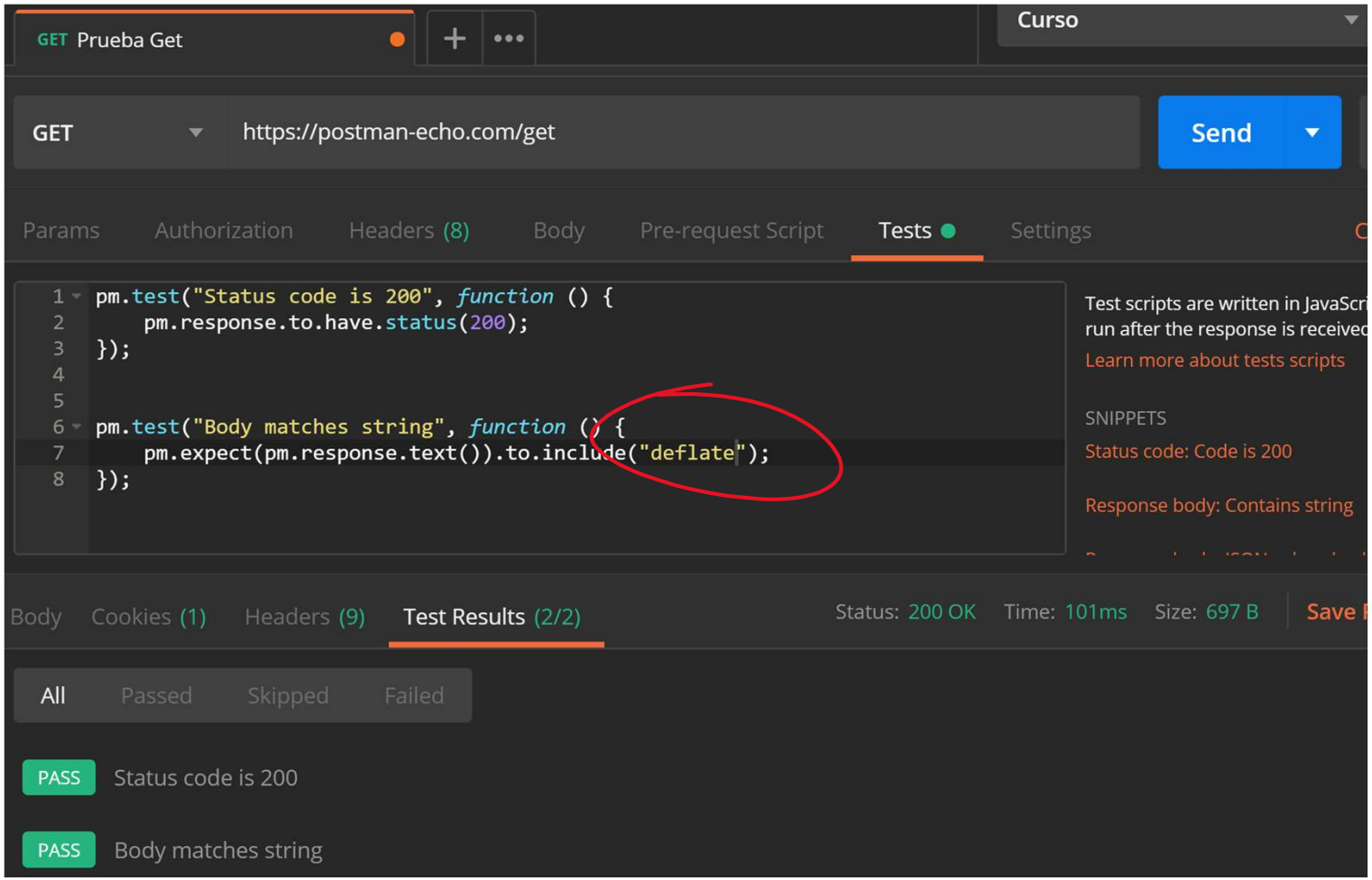
Test de búsqueda

- Otro test podría ser el de comprobar si en la respuesta a la petición, hay una palabra en concreto:



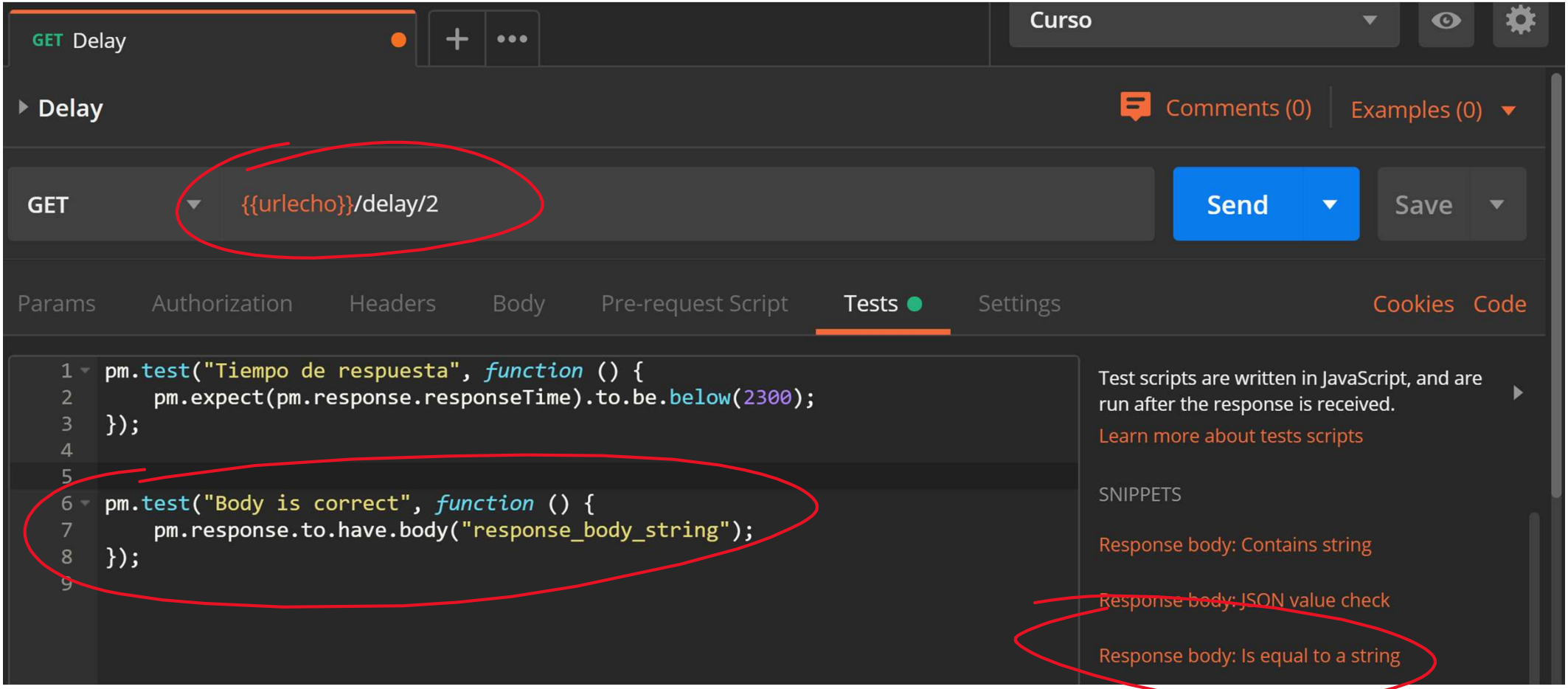
Test de búsqueda

- Modificamos el código para que busque una palabra:



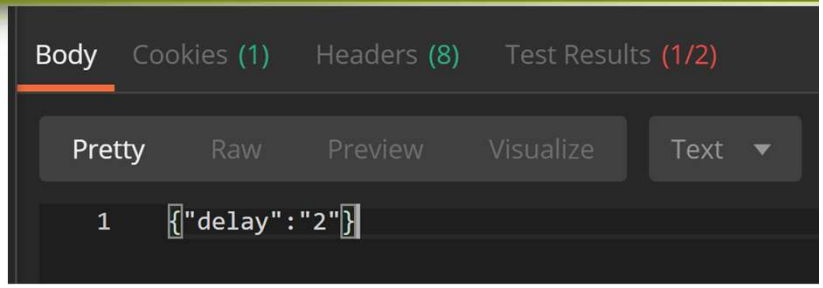
Test de respuesta exacta

- Podemos verificar si se obtiene la respuesta completa de la petición según lo esperado:



Test de respuesta exacta

- La respuesta de esta petición es:
- Modificamos el código del test:



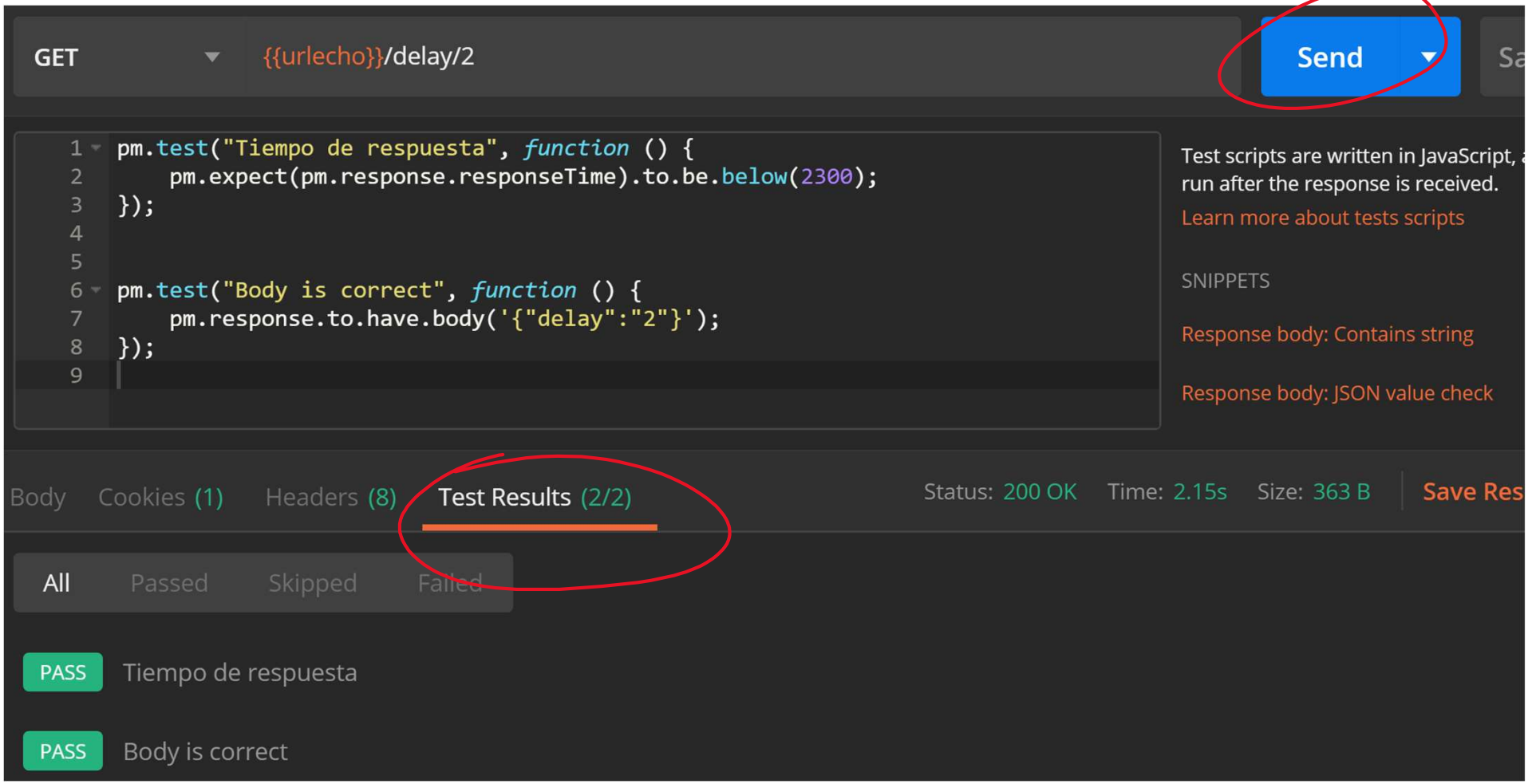

```
pm.test("Tiempo de respuesta", function () {
    pm.expect(pm.response.responseTime).to.be.below(2300);
});


pm.test("Body is correct", function () {
    pm.response.to.have.body('{"delay":"2"}');
});
```

- Ojo con las comillas dobles y simples!

Test de respuesta exacta

- Ejecutamos y vemos el resultado



Otros test

- Podemos verificar prácticamente cualquier valor relacionado con la respuesta del servicio, header, código del status, tiempos, tamaños, etc.

Assertions

Assertions

- Funciones que podemos utilizar en nuestros tests:

Chai

chaijs.com

Chai Assertion Library

Guide API Plugins

Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.

Download Chai for Node

4.2.0 / 2018-09-25

Another platform? Browser Rails

The chai package is available on npm.

$ npm install chai

View Node Guide

Issues | Fork on GitHub | Releases | Google Group | Build Status

Getting Started

Learn how to install and use Chai through a series of guided walkthroughs.

API Documentation

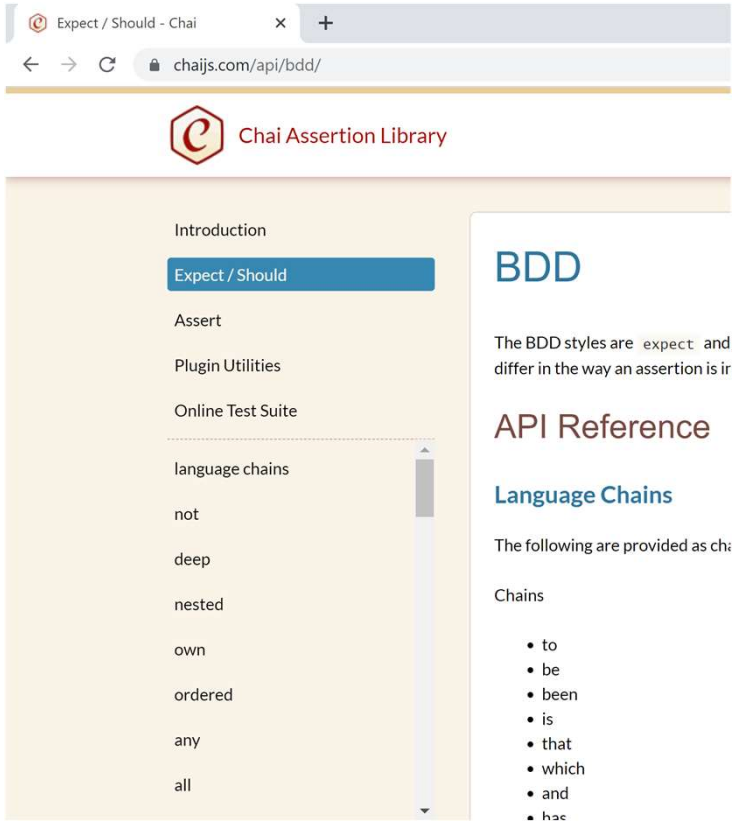
Explore the BDD & TDD language specifications for all available assertions.

Plugin Directory

Extend Chai's with additional assertions and vendor integration.

Assertions

- Por ejemplo, comprobar el tipo de dato de un valor:



.a(type[, msg])

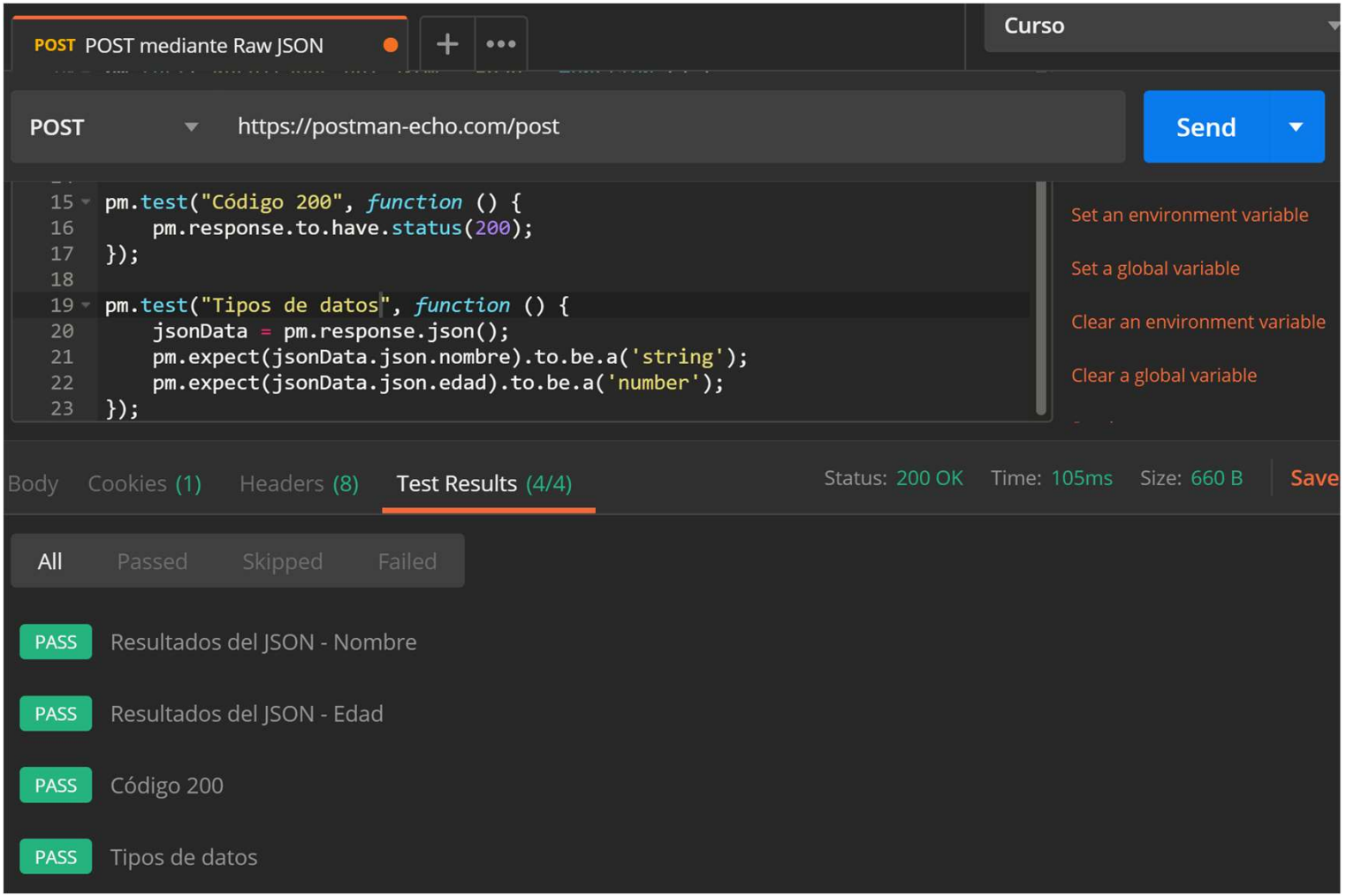
- **@param** *{String}* type
- **@param** *{String}* msg _optional_

Asserts that the target's type is equal to the given string `type`. Types are case insensitive. See the `type-detect` project page for info on the type detection algorithm: https://github.com/chaijs/type-detect.

```
expect('foo').to.be.a('string');
expect({a: 1}).to.be.an('object');
expect(null).to.be.a('null');
expect(undefined).to.be.an('undefined');
expect(new Error).to.be.an('error');
expect(Promise.resolve()).to.be.a('promise');
expect(new Float32Array).to.be.a('float32array');
expect(Symbol()).to.be.a('symbol');
```

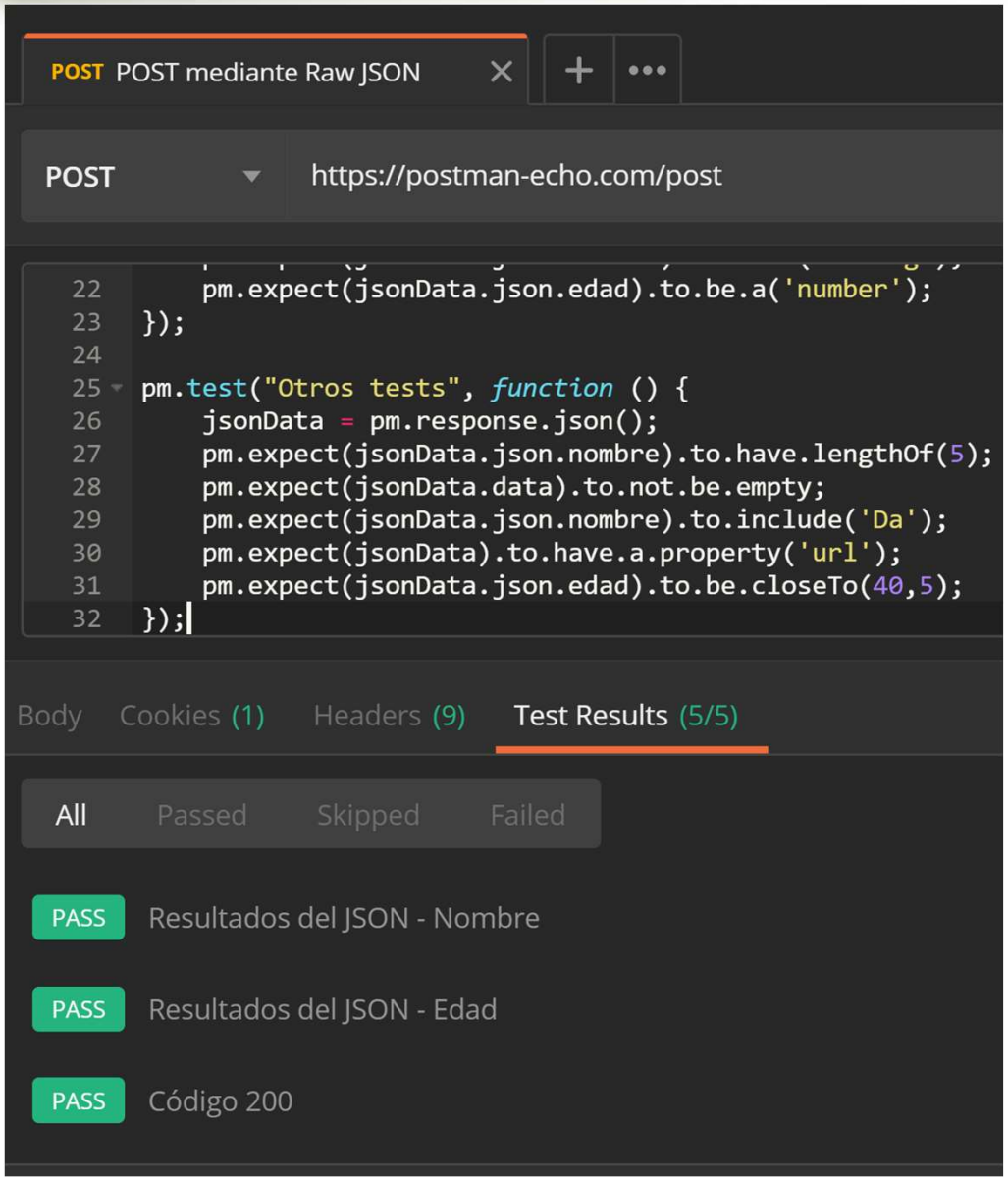
Assertions

- Comprobar si es string o number:



Assertions

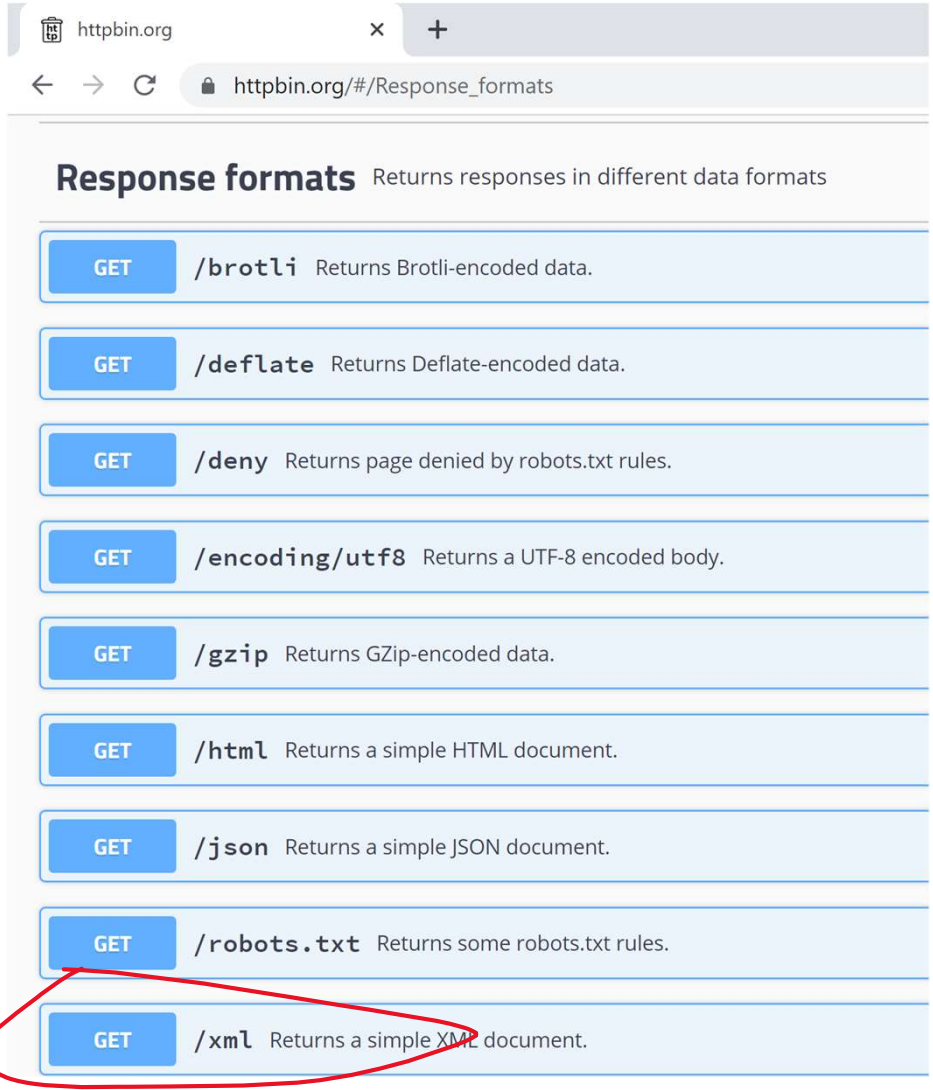
- Otro tipos de test:



Testing XML y texto

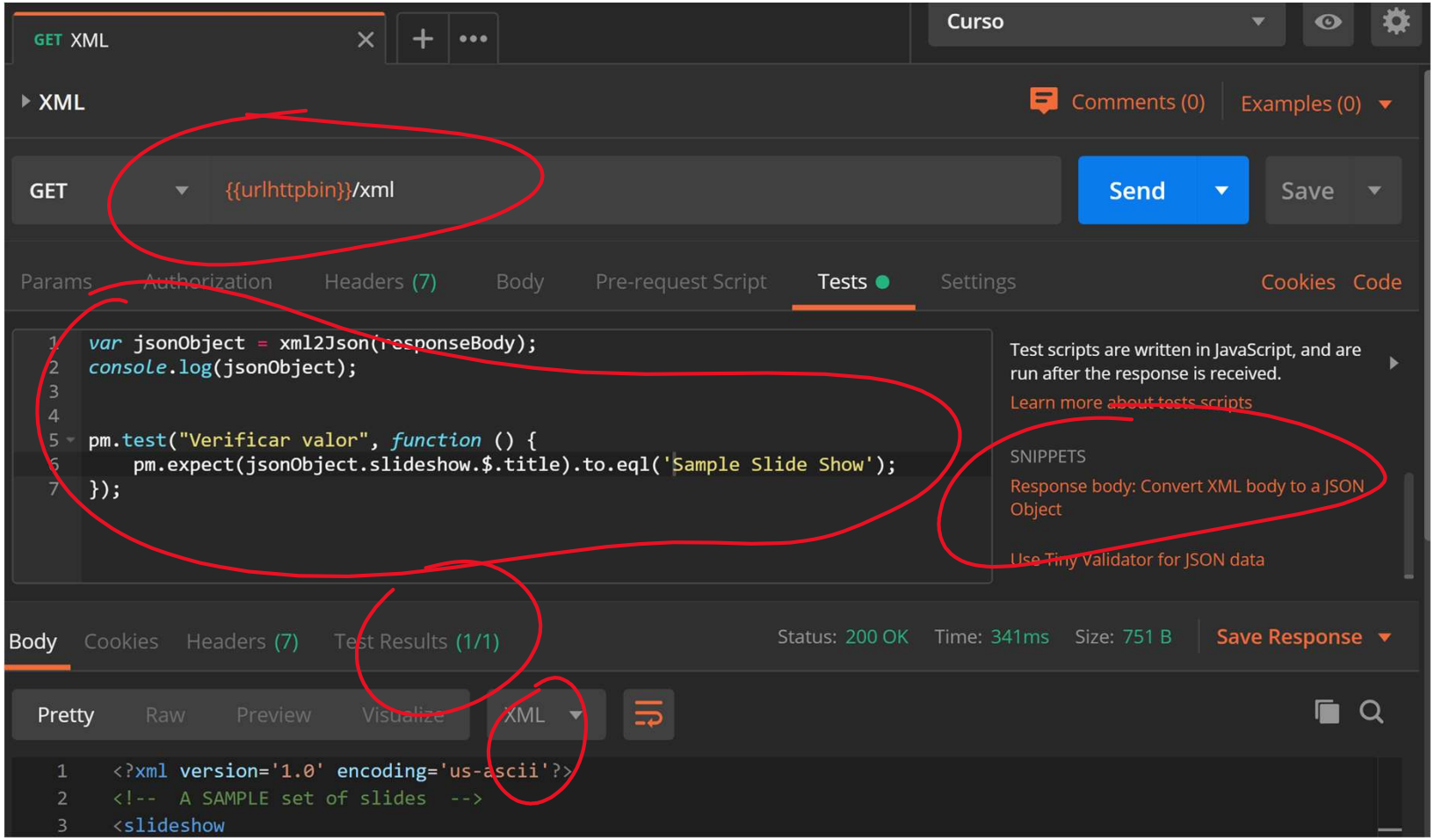
XML

- Podríamos tener respuestas en diferentes formatos:



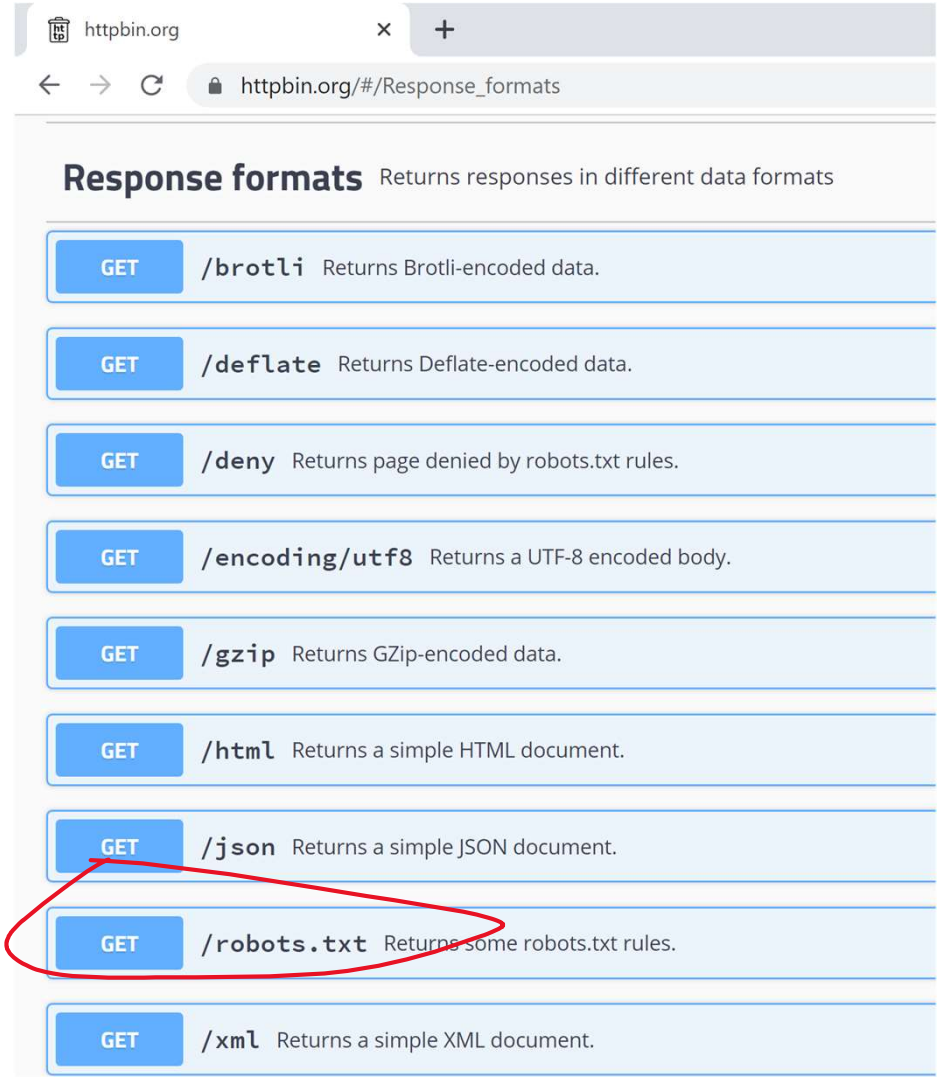
XML

- Si recibimos valores en XML podemos convertirlos a JSON y realizar el tipo de pruebas vistas antes:



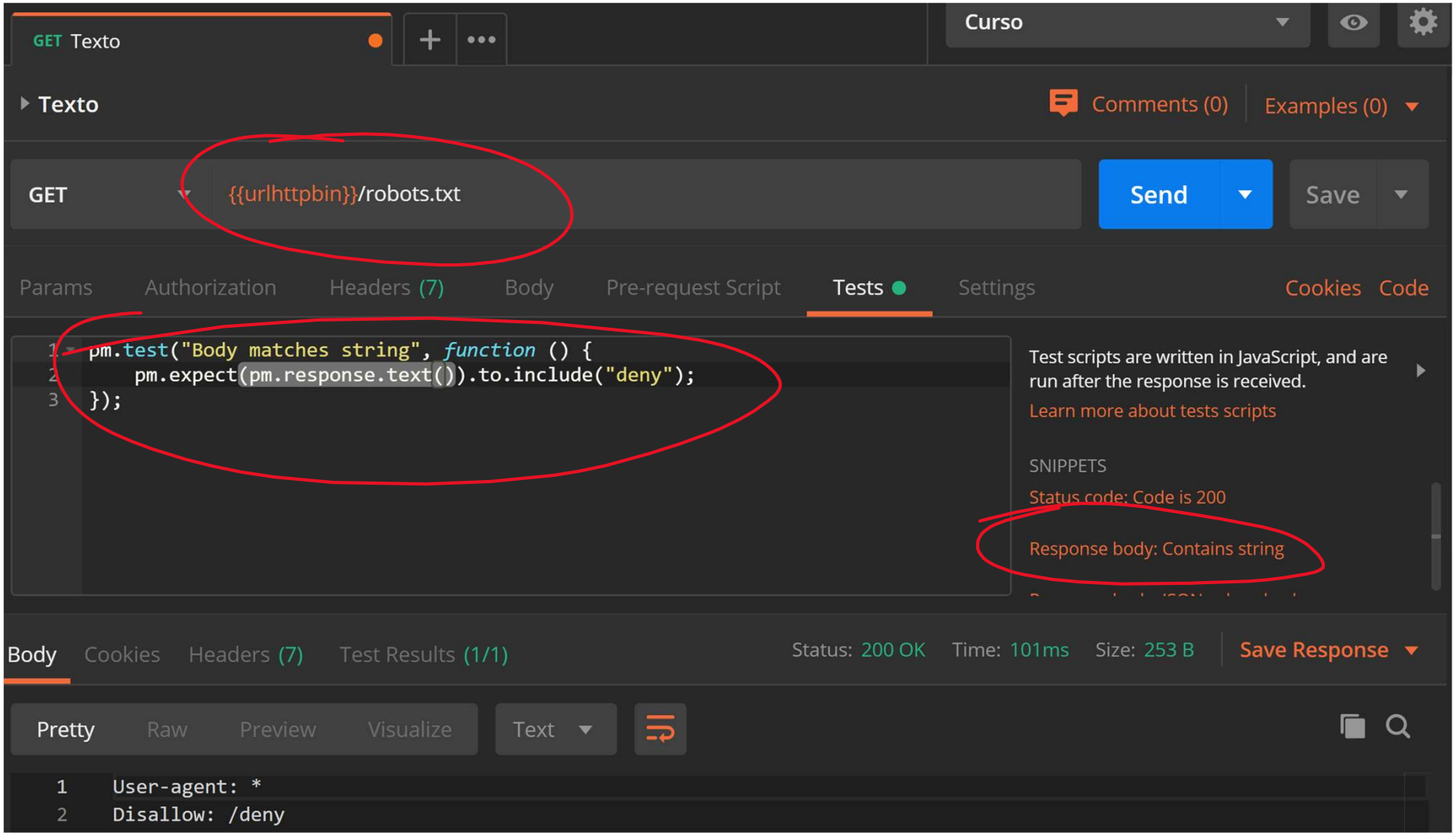
Texto

- Si recibimos texto:



Texto

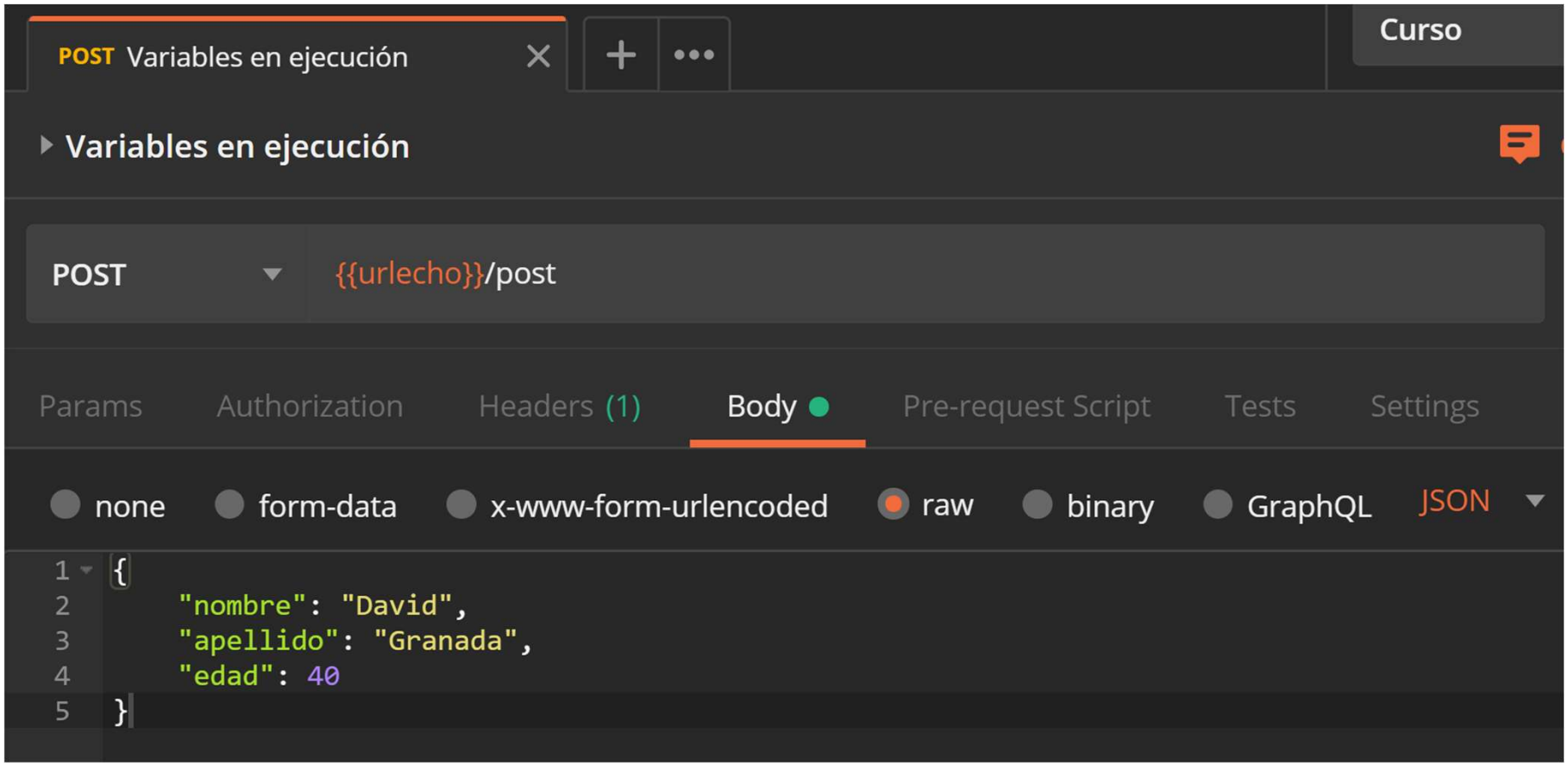
- Si recibimos texto, podemos realizar nuestros test de esta forma:



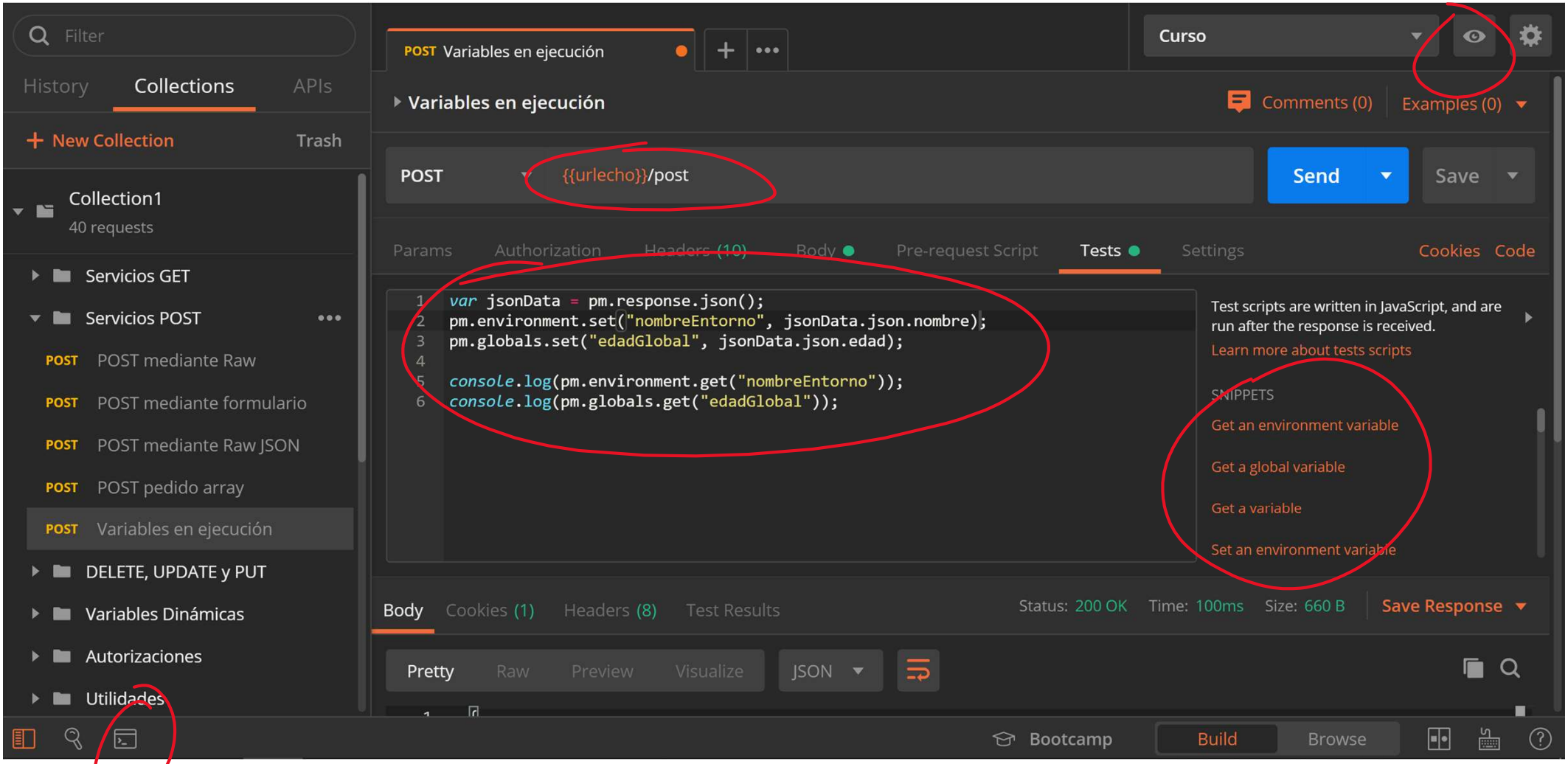
Pre-request Script

Variables en ejecución

- Podemos crear variables globales o de entorno en ejecución. Creamos una nueva petición en POST:

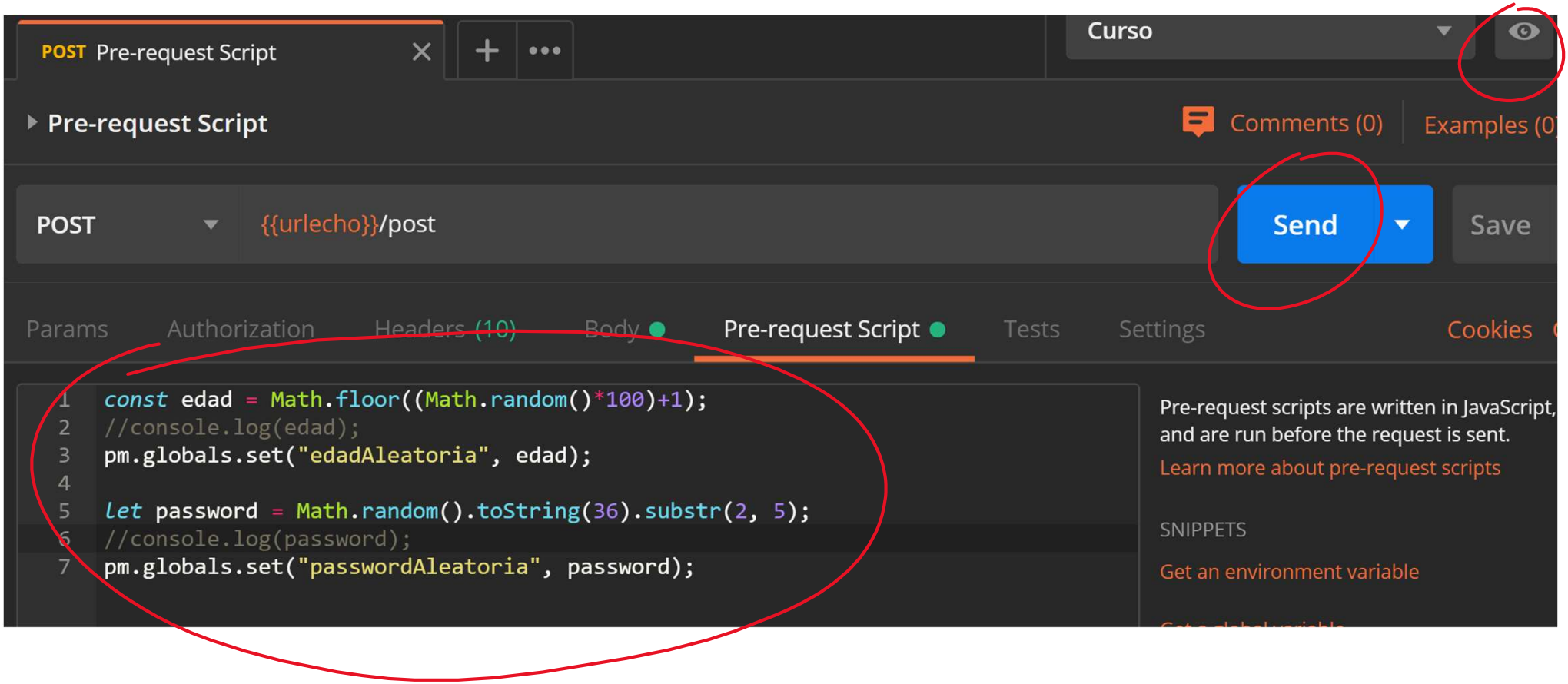


Variables en ejecución



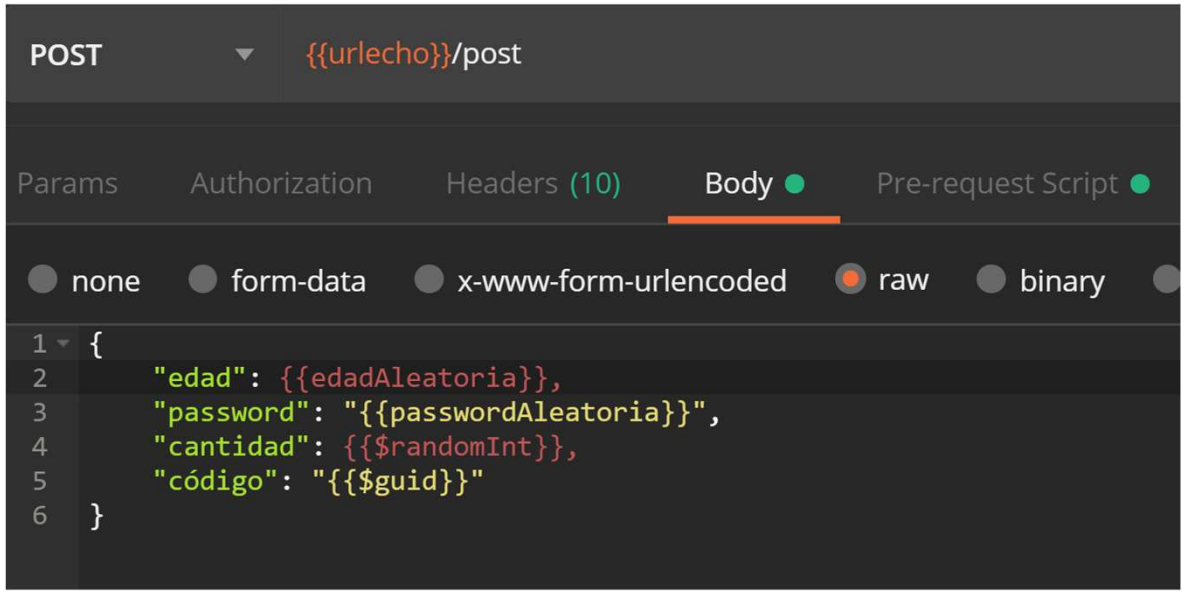
Pre-request Script

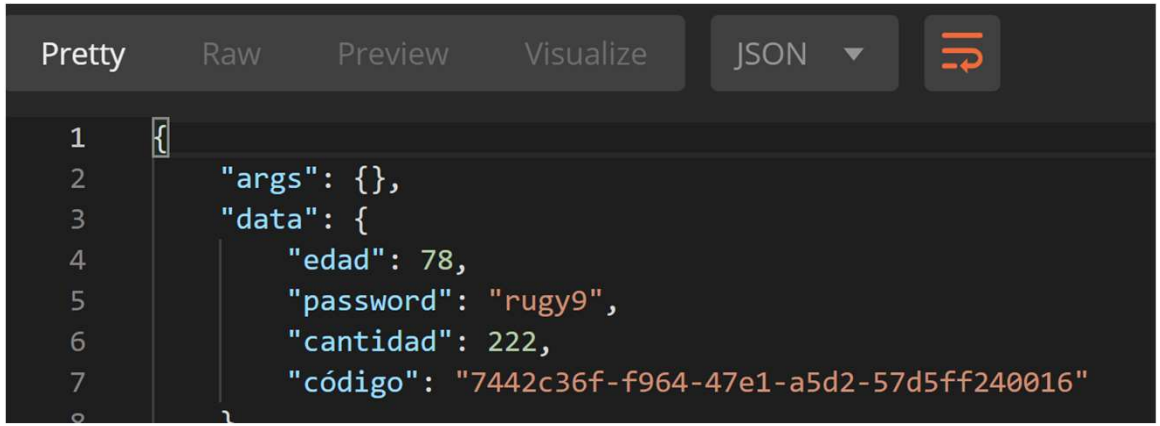
- Podemos crear variables y ejecutar código en general antes de llevar a cabo la ejecución de la petición:



Pre-request Script

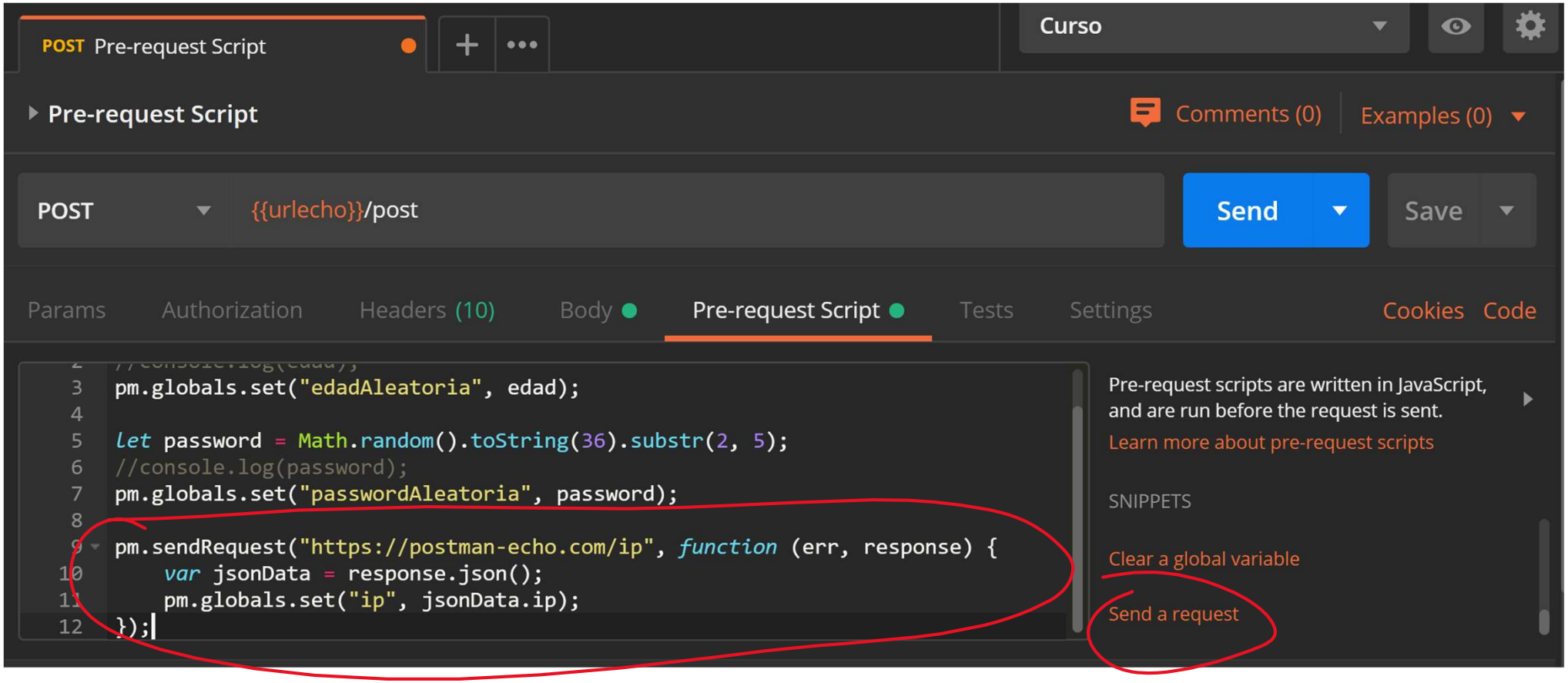
- Estas variables las podemos usar luego en la petición:





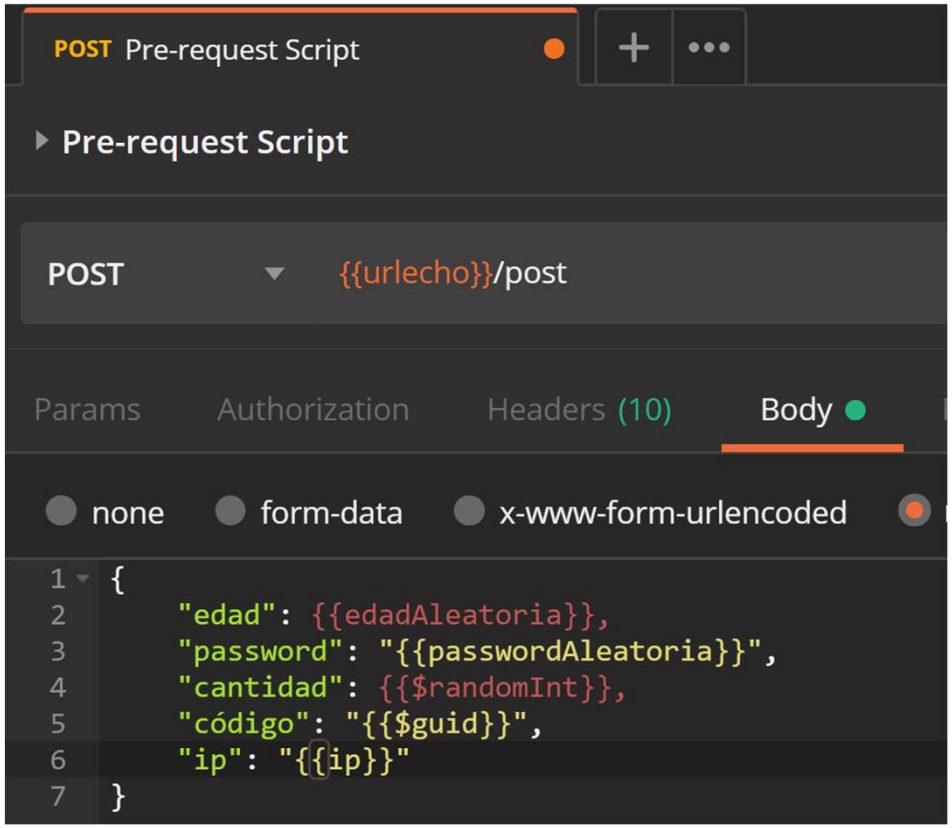
Pre-request Script

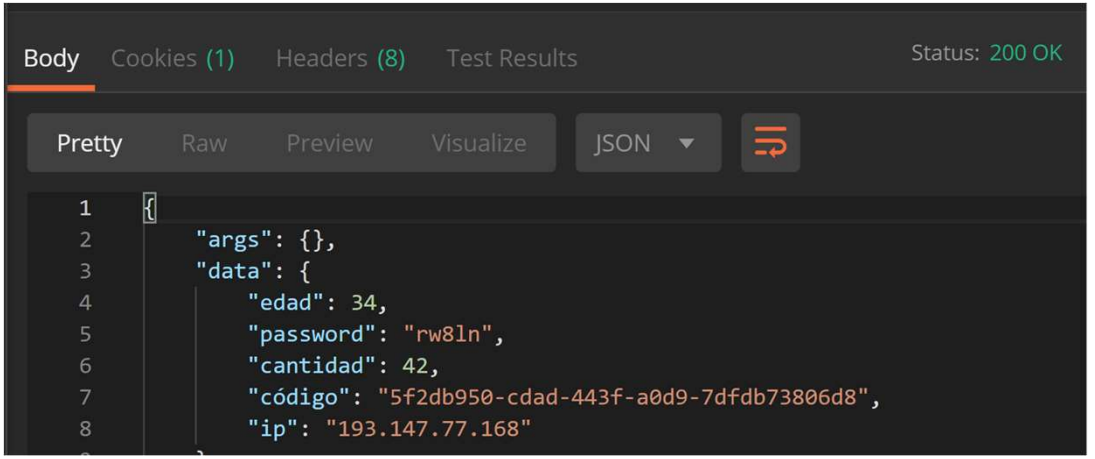
- En el Pre-request podemos también enviar peticiones para recuperar valores de estas peticiones:



Pre-request Script

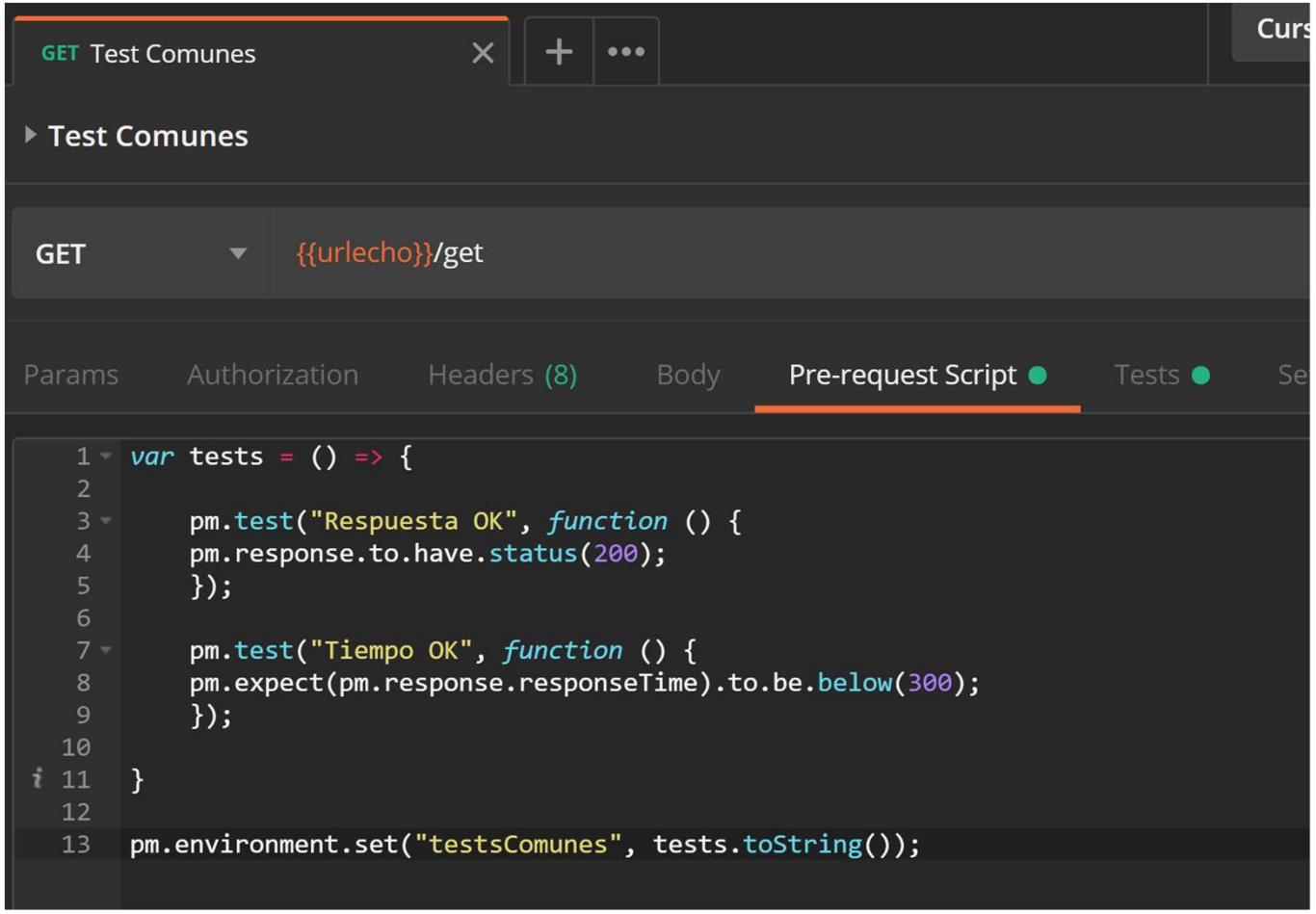
- Dichos valores podemos usarlos en la petición actual:





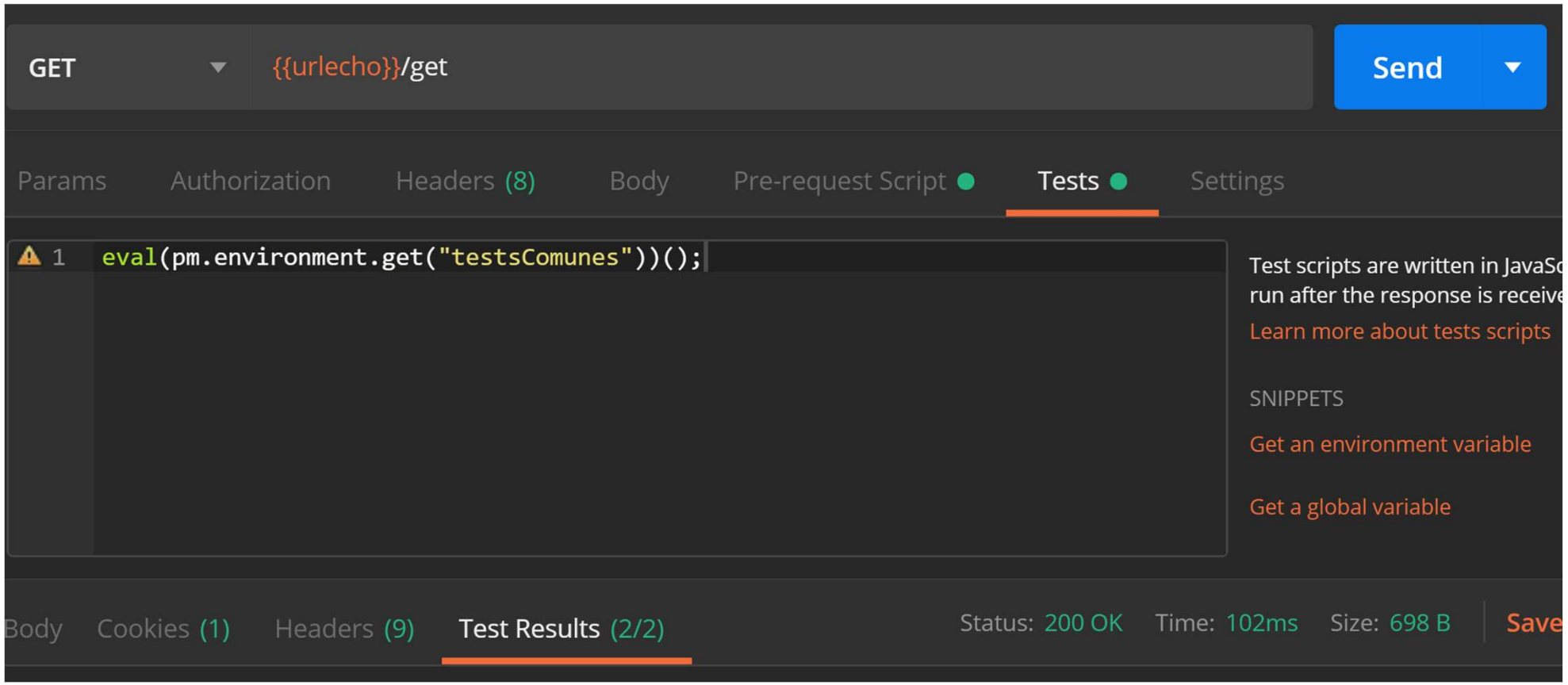
Pre-request Script

- Es habitual tener test comunes en todas las peticiones, como por ejemplo el estado y el tiempo:



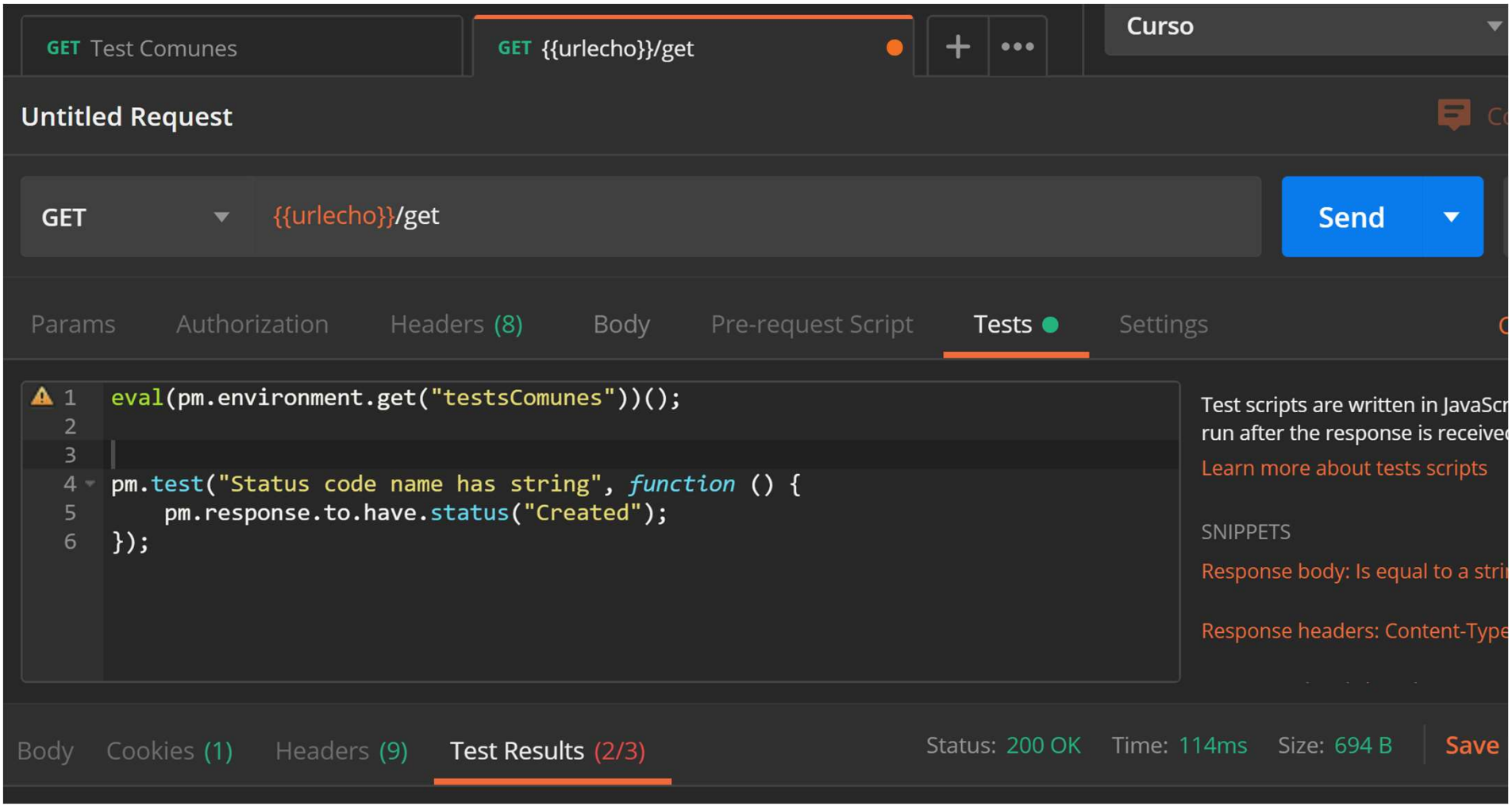
Pre-request Script

- Una vez guardados los tests en una variable, podemos usarlo de esta forma:



Pre-request Script

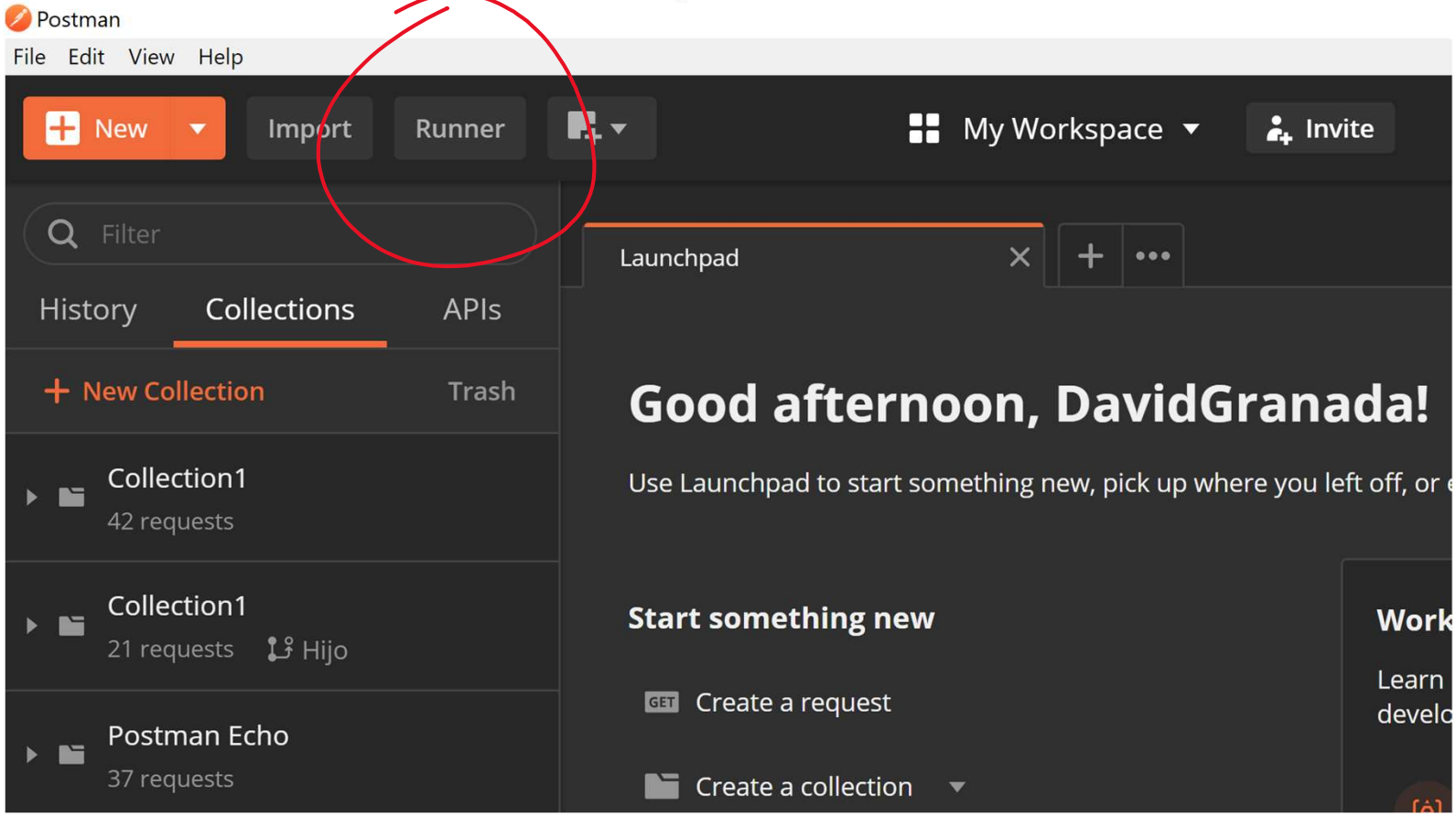
- Y podemos re-utilizarlos en cualquier otra petición:



Runner tool

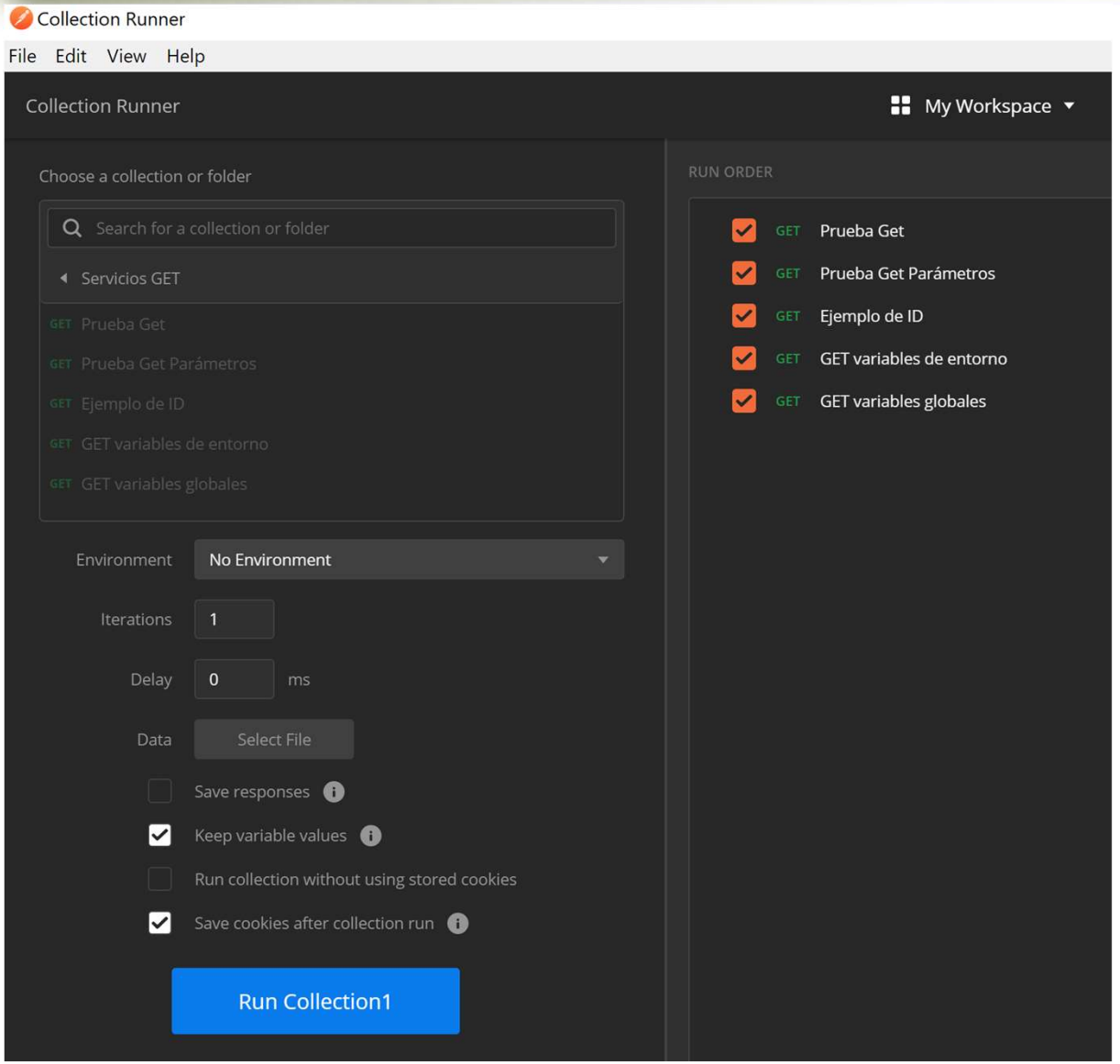
Runner

- Postman tiene una herramienta que nos permite ejecutar todos los tests presentes en una carpeta o en una colección completa:



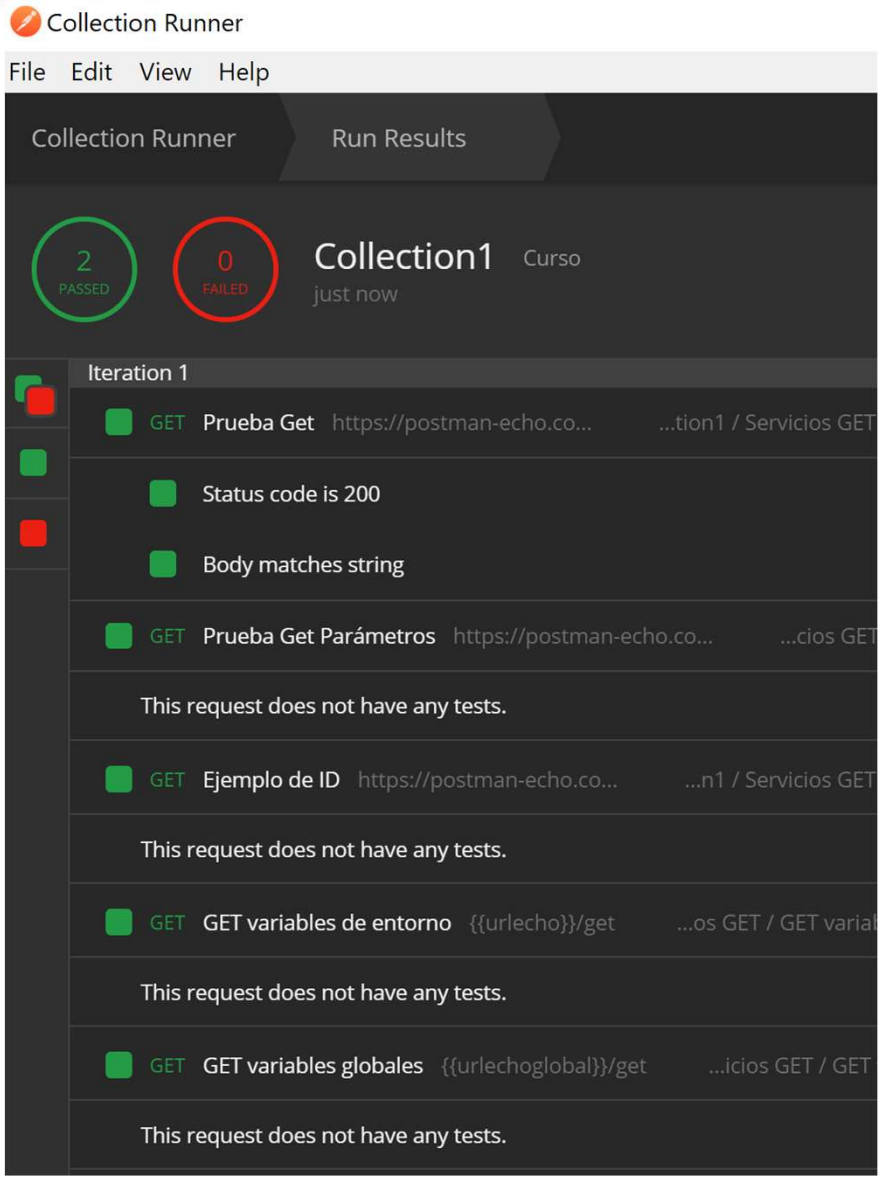
Runner

- Seleccionamos la carpeta y ejecutamos los tests:



Runner

- Vemos el resultado de la ejecución de los test:
- Esto es muy útil para probar todos nuestros servicios web con una sola instrucción.



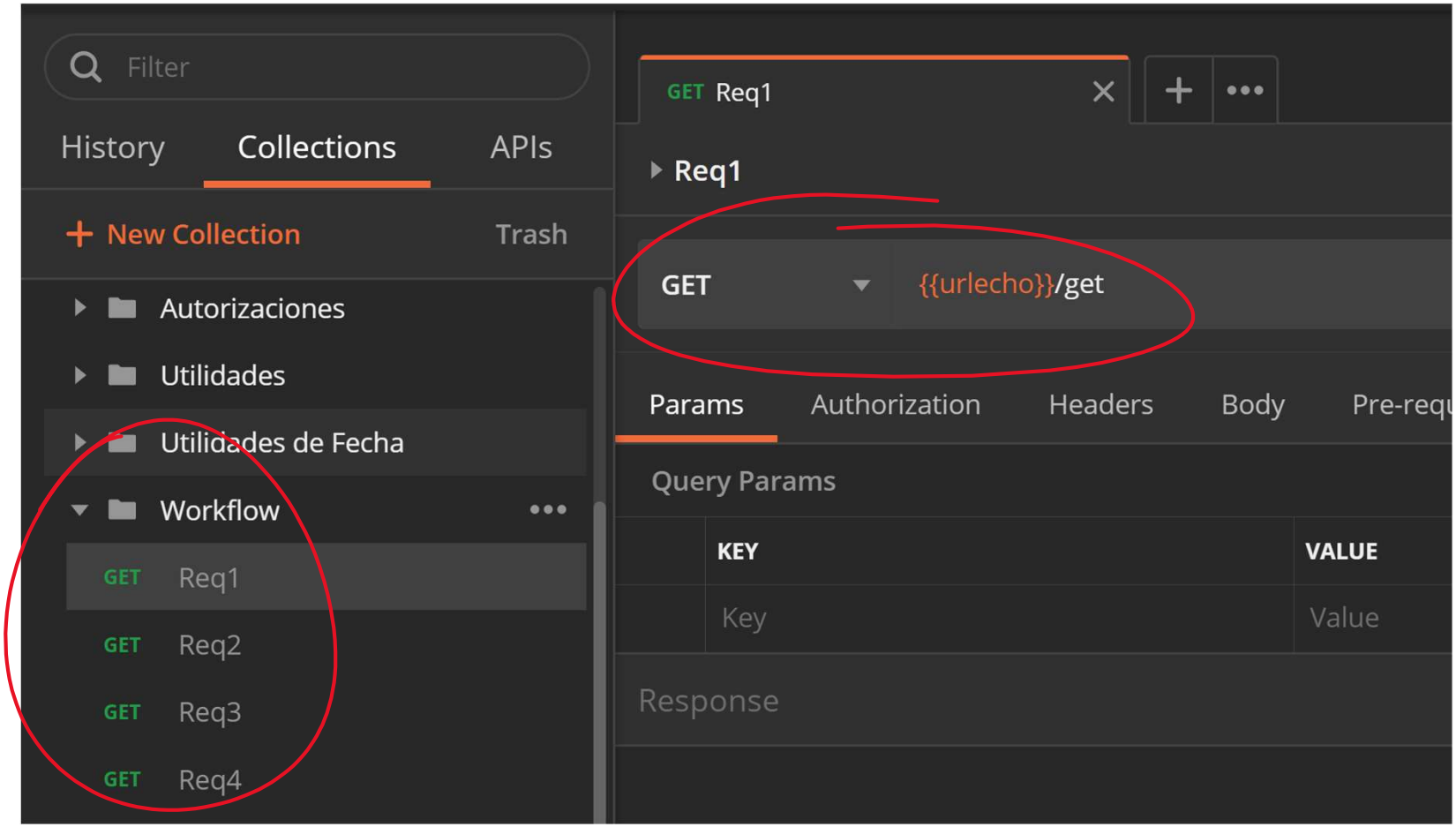
Flujos de trabajo

Flujos de trabajo

- El runner nos permite ejecutar los tests, pero también nos permite simplemente ejecutar las peticiones que tengamos en una carpeta o colección
- Las peticiones se ejecutan según el orden presente en la carpeta, pero podemos modificar ese comportamiento

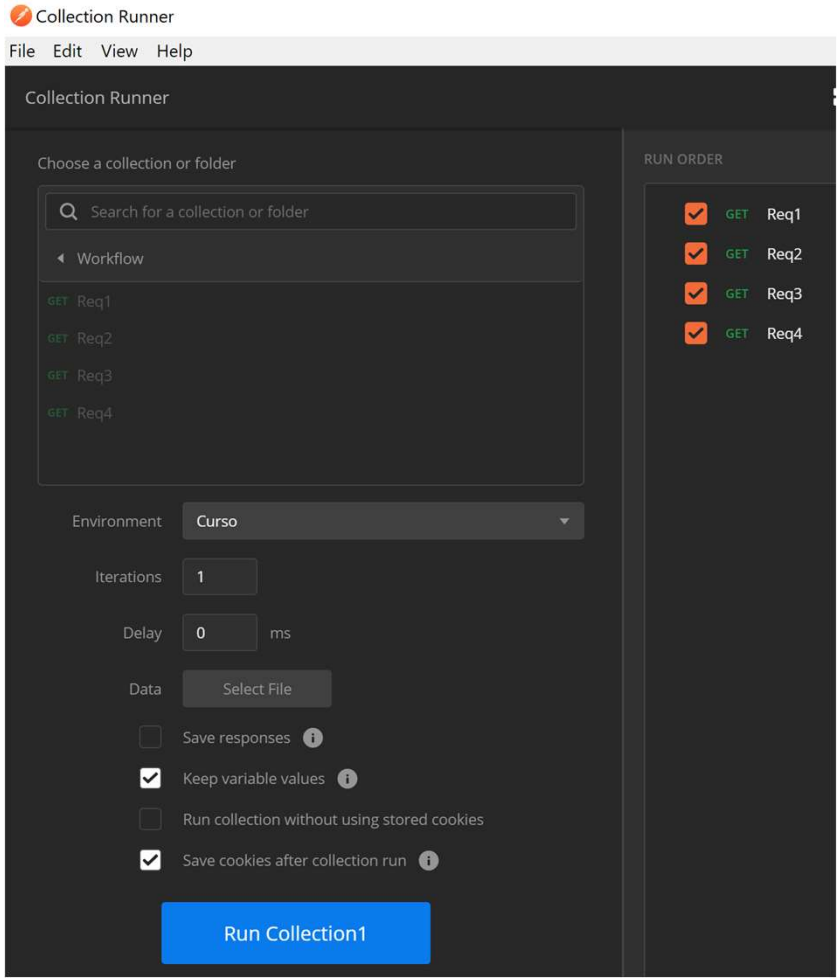
Flujos de trabajo

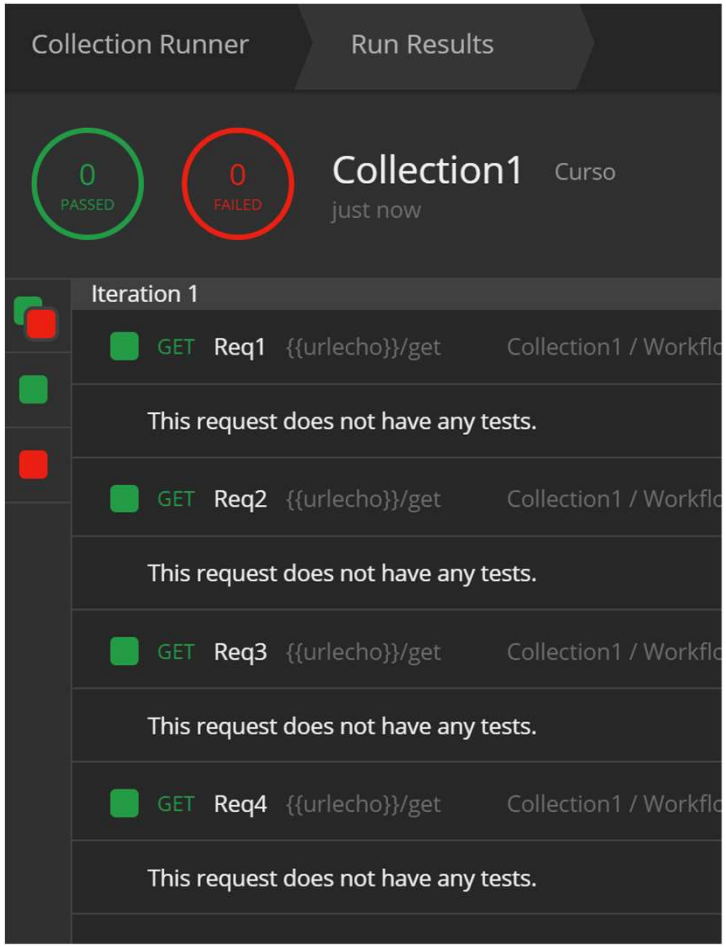
- Vamos a crear una subcarpeta llamada Workflow con 4 peticiones básicas de tipo GET:



Flujos de trabajo

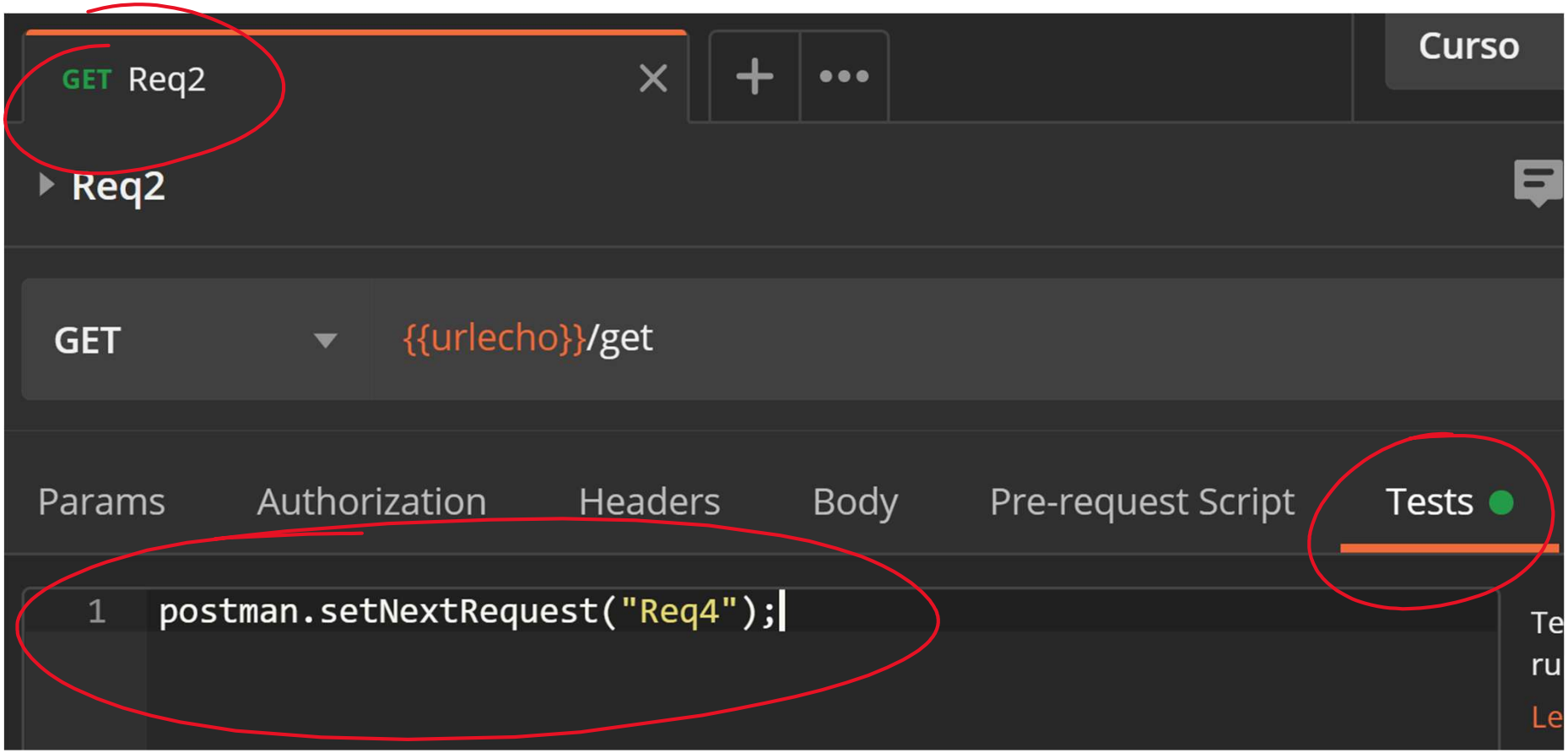
- Al ejecutarlas con el runner, se ejecutan según el orden en el que están en la carpeta:





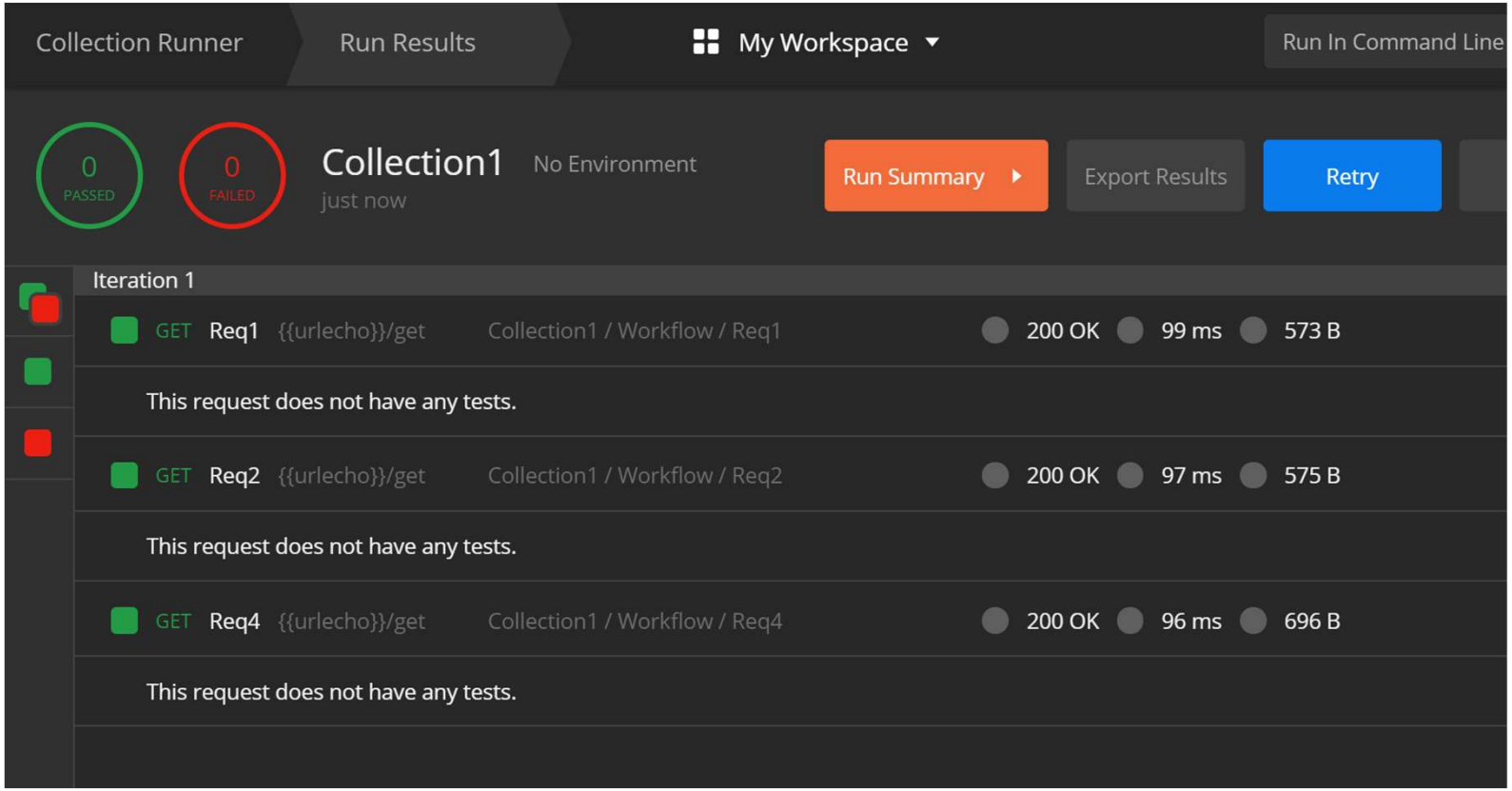
Flujos de trabajo

- Podemos cambiar el orden de esta forma:



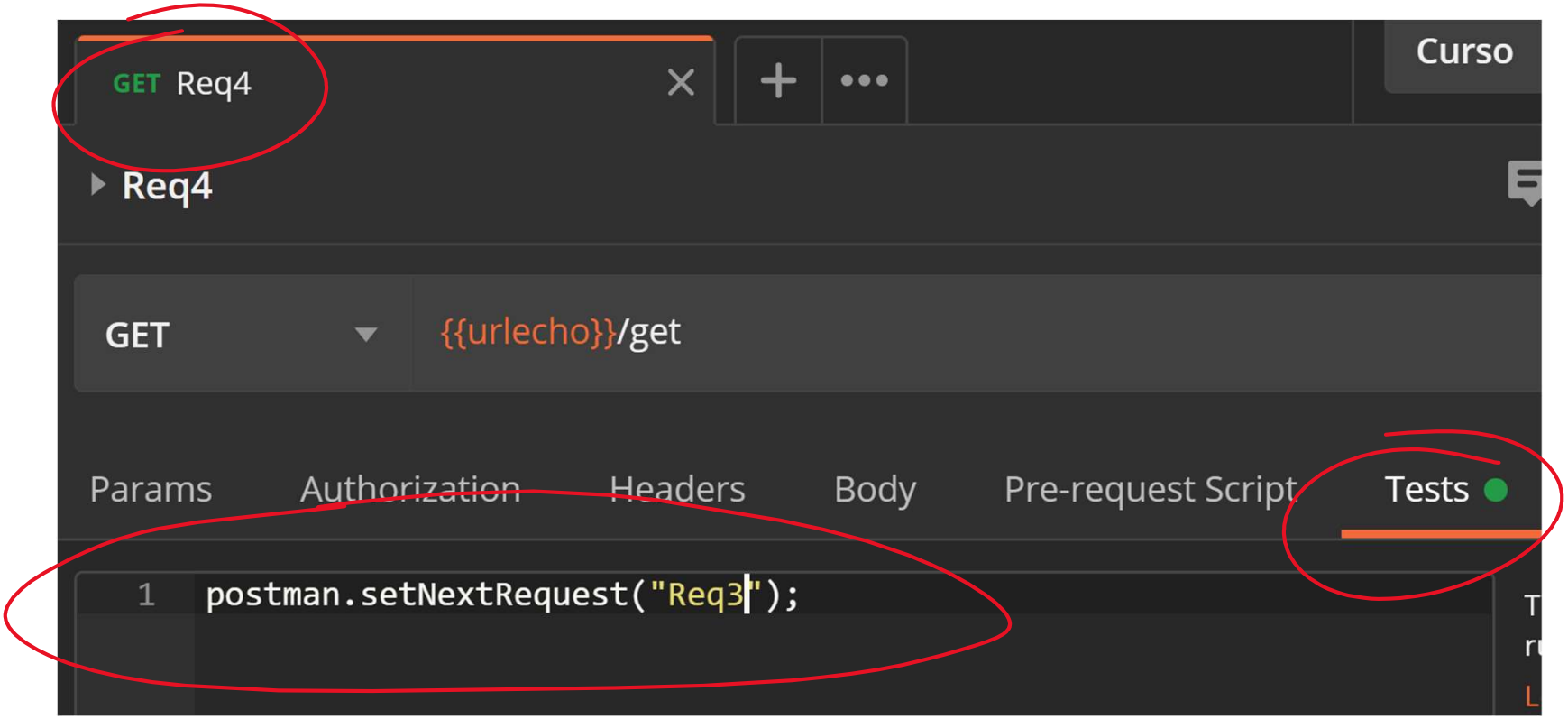
Flujos de trabajo

- Volvemos a ejecutar el runner y vemos que después del Req2 se ejecuta el Req4 (el Req3 no se llega a ejecutar):



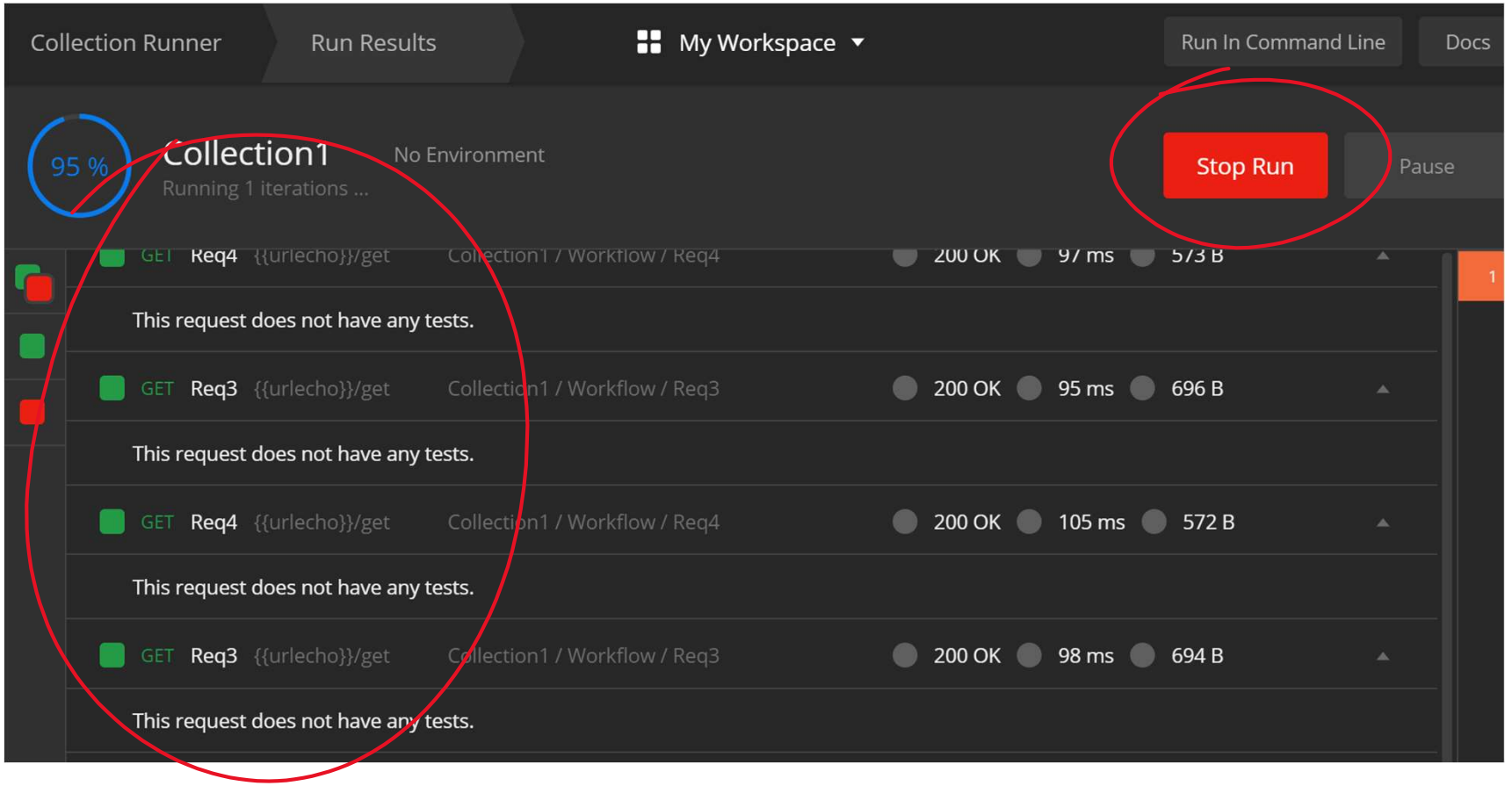
Flujos de trabajo

- Podemos indicar que después del Req 4 se ejecute el Req3:



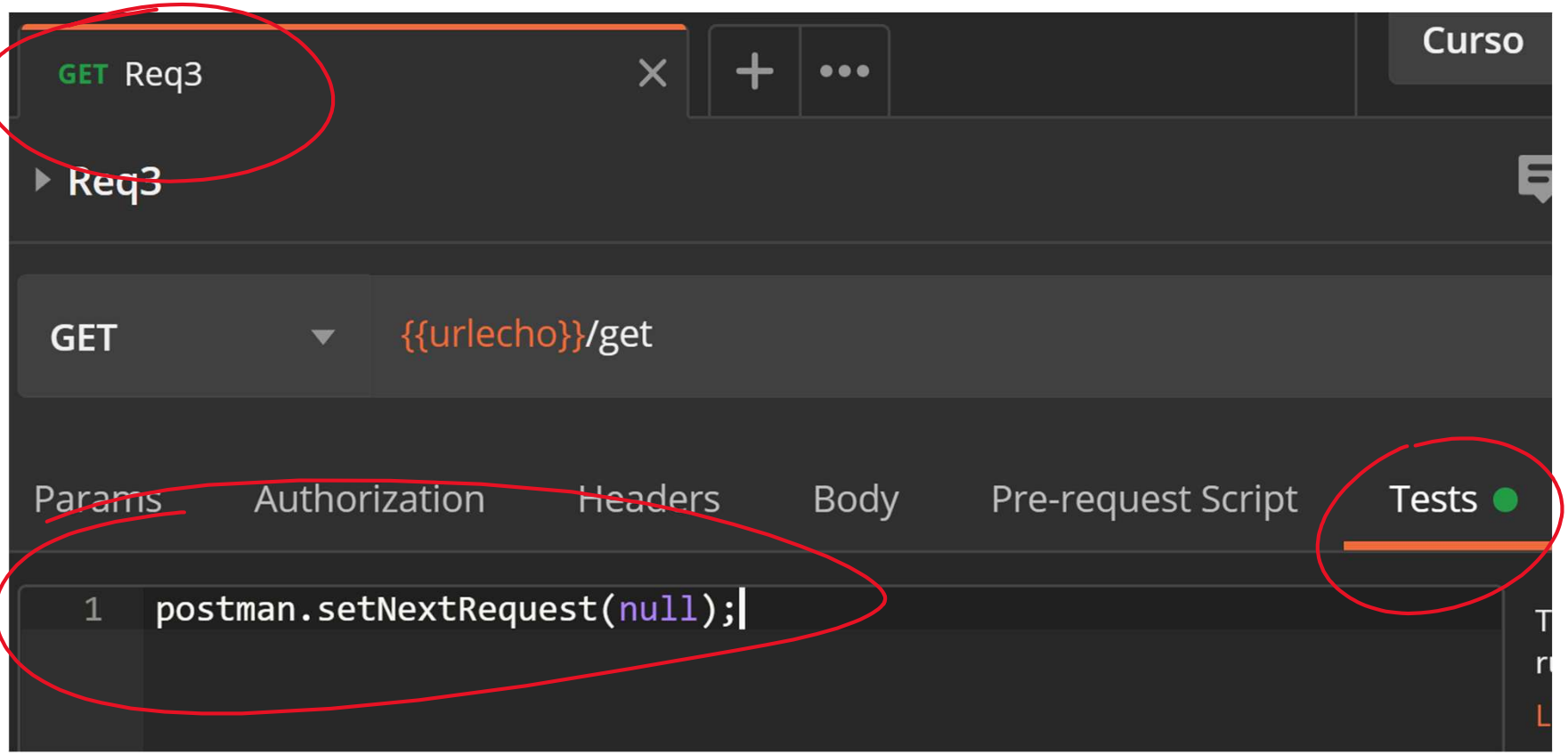
Flujos de trabajo

- Al ejecutar el Runner se crea un bucle infinito:



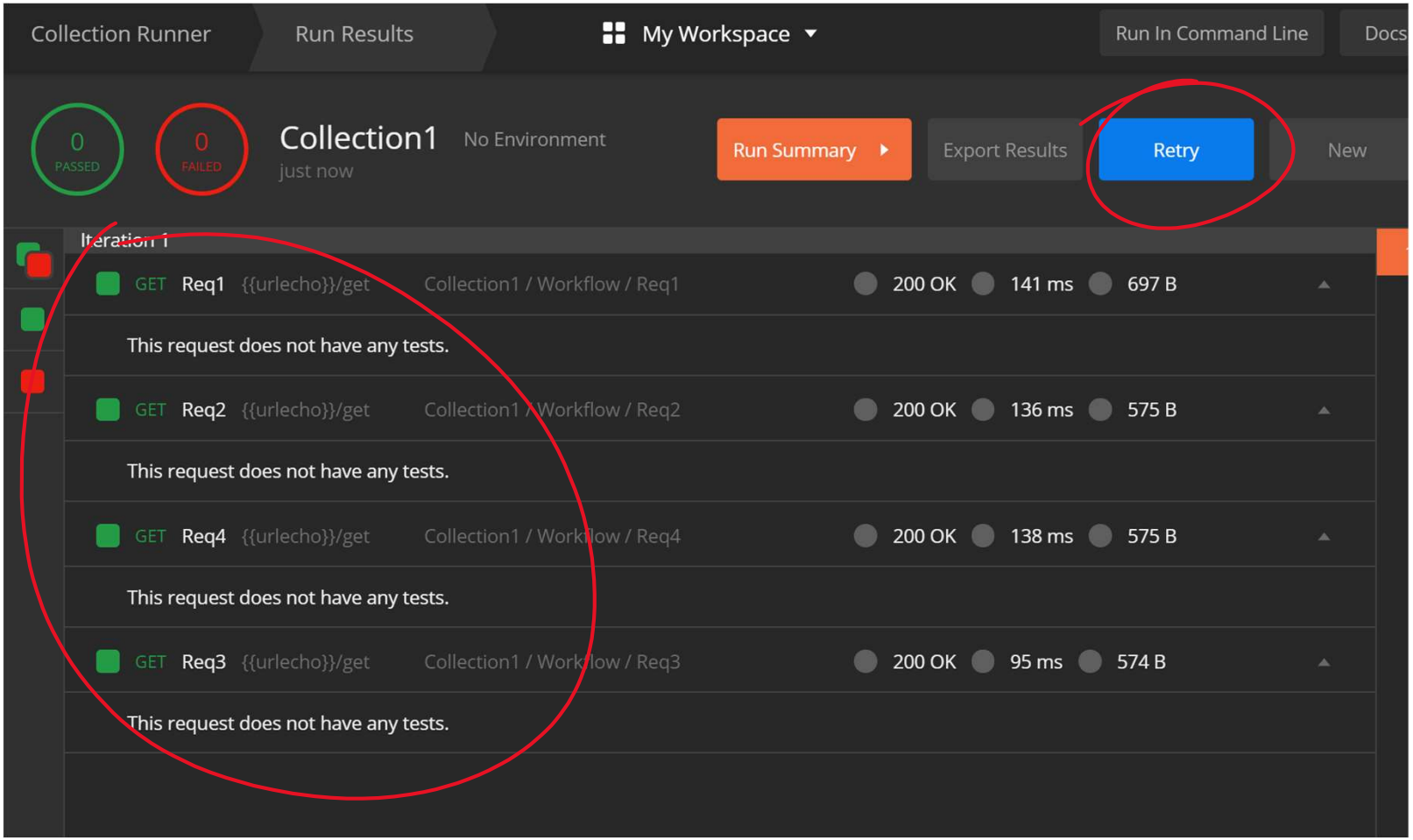
Flujos de trabajo

- Tenemos que indicar que al ejecutar el Req3 se detenga la ejecución global:



Flujos de trabajo

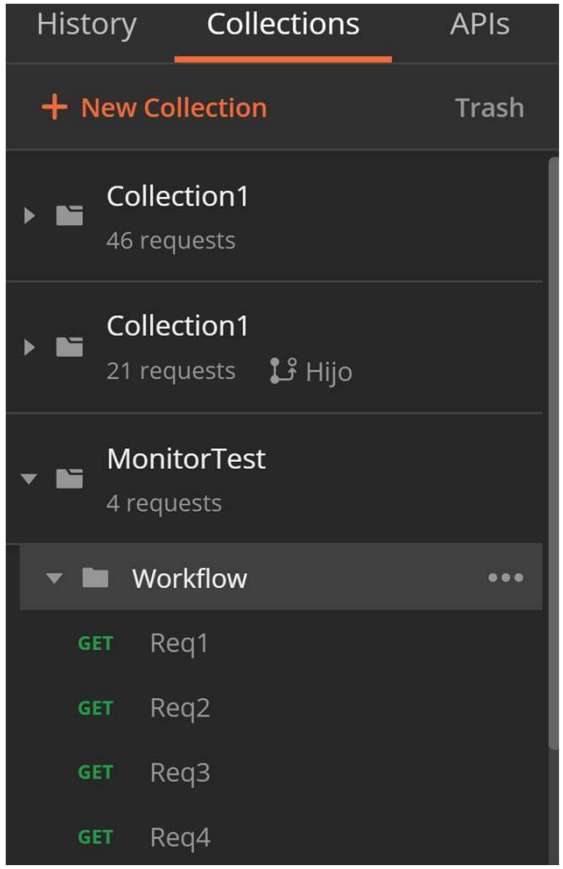
- Al volver a ejecutar el Runner, vemos el flujo que hemos definido:



Monitores

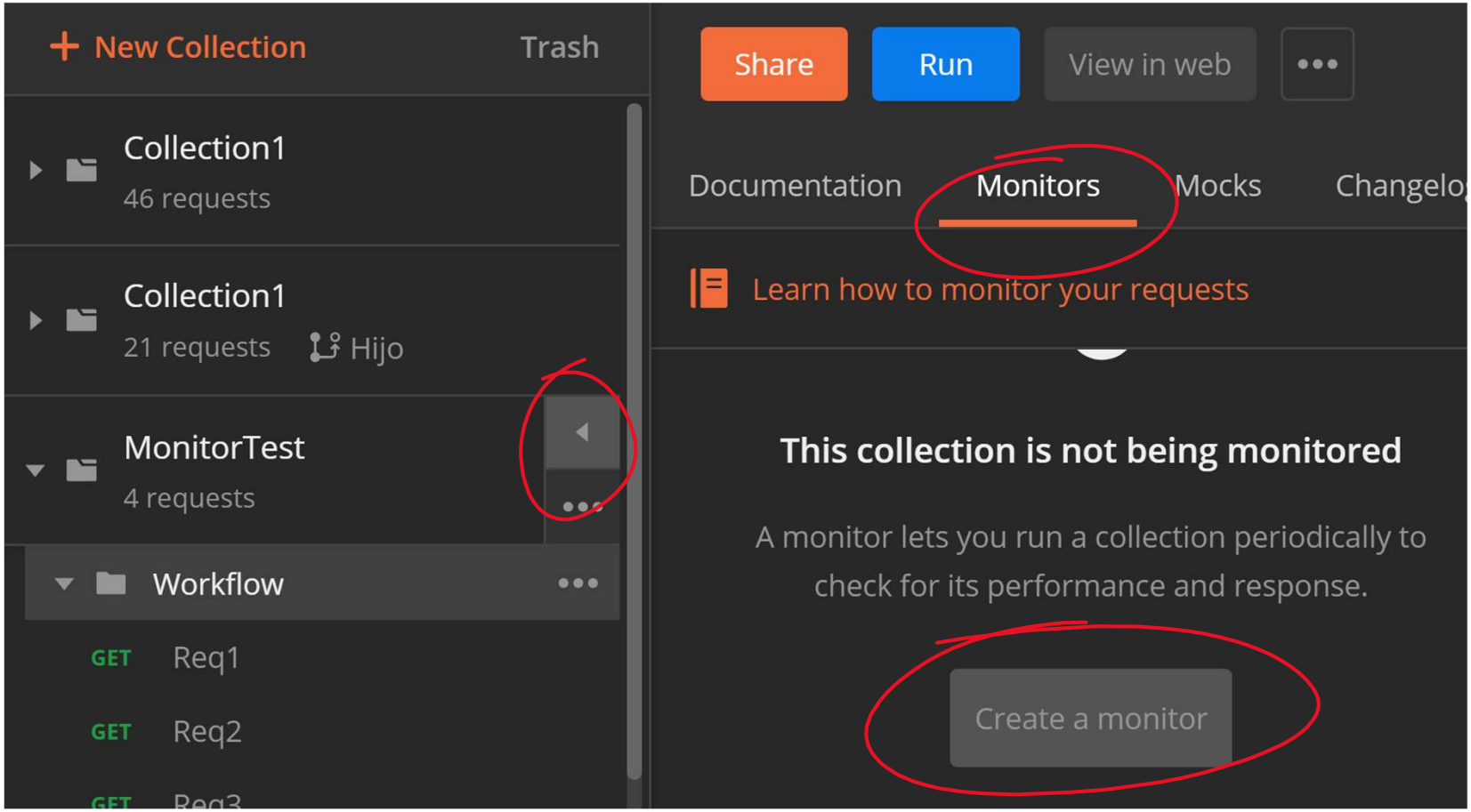
Monitor

- Postman nos permite automatizar periódicamente la ejecución de un conjunto de peticiones mediante su herramienta llamada Monitor (1000 peticiones al mes de forma gratuita)
- Creamos una nueva colección llamada MonitorTest, duplicamos la carpeta Workflow y la arrastramos a la nueva colección:



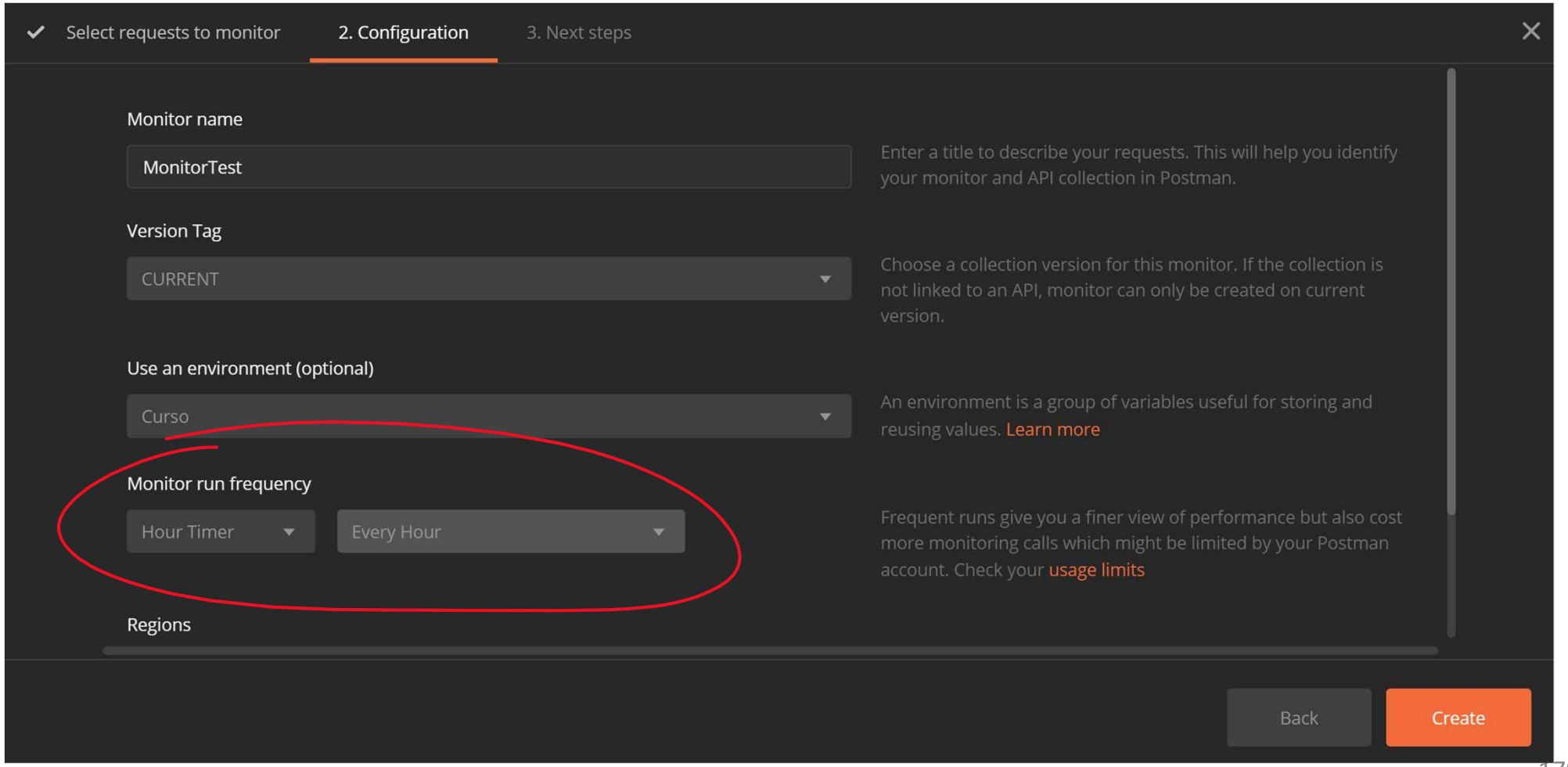
Monitor

- Creamos un nuevo Monitor:



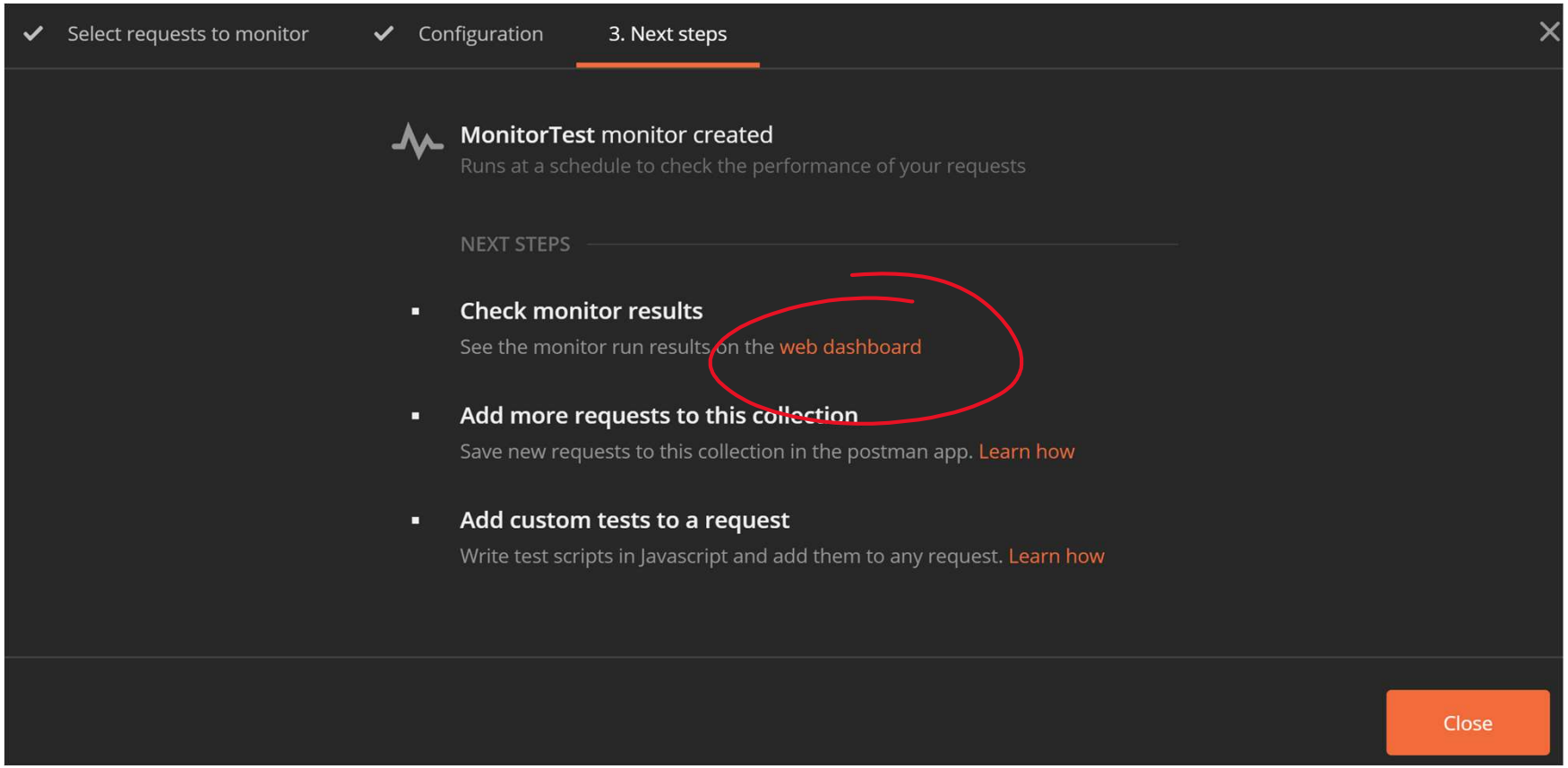
Monitor

- Definimos la configuración y la frecuencia:



Monitor

- Las ejecuciones se llevarán a cabo en los servidores de Postman y podremos consultar los resultados en nuestra Web Dashboard:



Monitor

- Web Dashboard: