

# 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

The screenshot shows the Postman website homepage. At the top left is a circular icon with a stylized orange pen nib. The main title "The Collaboration Platform for API Development" is prominently displayed. Below it, a call-to-action button says "Download the App". To the right, there's a large, stylized illustration of a blue and white robotic head labeled "API-5000" surrounded by concentric circles and small location pins. At the bottom, three large statistics are listed: "10 million Developers", "500,000 Companies", and "250 million APIs". The navigation bar at the top includes links for Product, How Collaboration Works, Use Cases, Pricing, Enterprise, Explore, and Learning Center.

Postman | The Collaboration Plat+

postman.com

POSTMAN    Product ▾    How Collaboration Works    Use Cases ▾    Pricing    Enterprise    Explore    Learning Center

# The Collaboration Platform for API Development

Download the free Postman app to get started.

[Download the App](#)

10 million Developers

500,000 Companies

250 million APIs



# 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



**Header bar**

The header bar of the Postman application includes:

- Postman logo and title bar.
- File, Edit, View, Help menu.
- New, Import, Runner, Create Collection buttons.
- My Workspace, Invite, and environment selection dropdowns.
- Notification icons and Upgrade button.

**Side bar**

The side bar includes:

- Filter search bar.
- History, Collections (selected), APIs tabs.
- + New Collection, Trash buttons.
- A collection icon with a plus sign.
- You don't have any collections message.
- Collections let you group related requests, making them easier to access and run.
- + Create a collection button.
- Start something new section with Create a request, Create a collection (dropdown), Create an environment, Create an API, and View More options.
- Recent workspaces section with a placeholder message: Find your most recently used workspaces here.
- Customize section with Dark mode toggle.

**Builder**

The builder section includes:

- Launchpad tab (selected).
- No Environment dropdown.
- Good evening, DavidGranada! greeting.
- Use Launchpad to start something new, pick up where you left off, or explore some resources to help you master Postman.
- Start something new section with Create a request, Create a collection (dropdown), Create an environment, Create an API, and View More options.
- Work smarter with Postman section with a list of in-app tutorials:
  - Designing and mocking APIs (2 lessons)
  - Debugging and manual testing (4 lessons)
  - Automated testing (4 lessons)
  - API documentation (1 lesson)
  - Monitoring (1 lesson)
  - Collaboration (1 lesson)

Bottom navigation bar:

- Icons for Bootcamp, Build, Browse, and other tools.



# Introducción

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



The screenshot shows a browser window with the following details:

- Title bar: Postman Echo
- Address bar: https://docs.postman-echo.com
- Content area:
  - Header: POSTMAN ECHO
  - Section: Introduction
    - Request Methods
    - Headers
    - Authentication Methods
    - Cookie Manipulation
    - Utilities
    - Utilities / Date and Time
    - Utilities / Postman Collection
    - Auth: Digest
  - Main content:

## Postman Echo

Postman Echo is service you can use to test your REST clients and make sample API calls. It provides endpoints for `GET`, `POST`, `PUT`, various auth mechanisms and other utility endpoints.

The documentation for the endpoints as well as example responses can be found at <https://postman-echo.com>

## Request Methods

HTTP has multiple request "verbs", such as `GET`, `PUT`, `POST`, `DELETE`, `PATCH`, `HEAD`, etc.

# Introducción

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

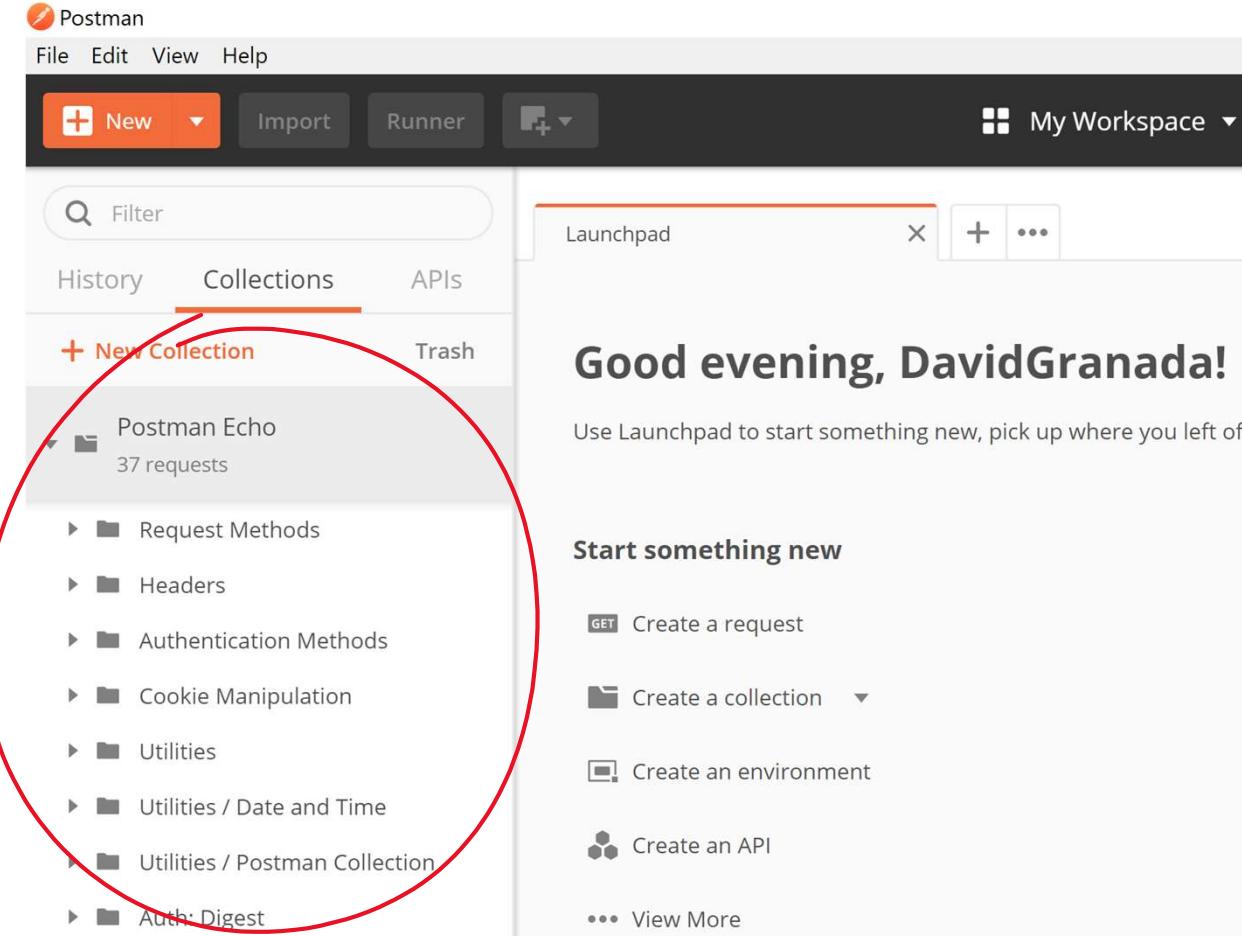
The screenshot shows a web browser window with the following details:

- Title Bar:** Postman Echo
- Address Bar:** https://docs.postman-echo.com
- Content Area:** Displays the "Postman Echo" documentation page. The page includes a sidebar titled "POSTMAN ECHO" with sections like "Introduction", "Request Methods", "Headers", "Authentication Methods", and "Cookie Manipulation".
- Right Panel:** Shows a "Language" dropdown menu with options: "cURL - cURL", "C# - RestSharp", "cURL - cURL", "Go - Native", and "HTTP - HTTP".
- Top Right:** A "Public" button and an orange "Run in Postman" button, which is circled in red.
- Bottom Center:** A modal window titled "Open with..." with two options: "Postman for Chrome DEPRECATED" and "Postman for Windows V4.4.3 OR HIGHER". The "Postman for Windows" option is also circled in red.



# Introducción

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



The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Postman' logo, 'File', 'Edit', 'View', 'Help', and a toolbar with 'New', 'Import', 'Runner', and 'My Workspace'. Below the toolbar is a 'Launchpad' section with a red 'X' button and a '+' button. To the right of the launchpad is a 'Good evening, DavidGranada!' message and a 'Start something new' section with various creation options like 'Create a request', 'Create a collection', 'Create an environment', 'Create an API', and 'View More'. On the left, there's a sidebar with 'History', 'Collections' (which is highlighted with a red underline), and 'APIs'. Under 'Collections', there's a red circle highlighting the '+ New Collection' button. Below it, there's a list of collections: 'Postman Echo' (37 requests) and several sub-folders: 'Request Methods', 'Headers', 'Authentication Methods', 'Cookie Manipulation', 'Utilities', 'Utilities / Date and Time', 'Utilities / Postman Collection', and 'Auth: Digest'.



# Introducción

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



# Any API

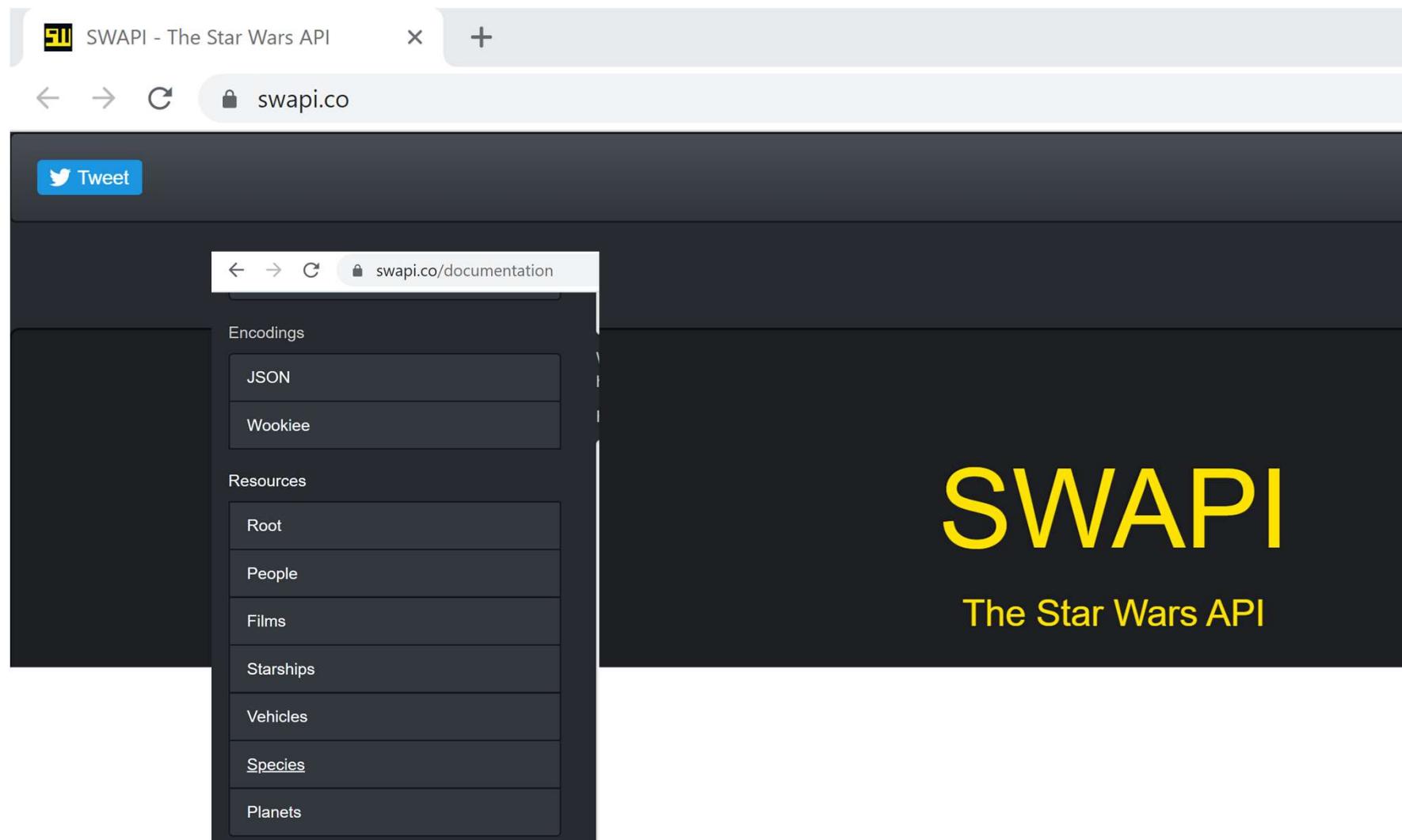
Documentation and Test Consoles for Over 1400 Public APIs

Powered by [LucyBot](#) and [APIs Guru](#)



# Introducción

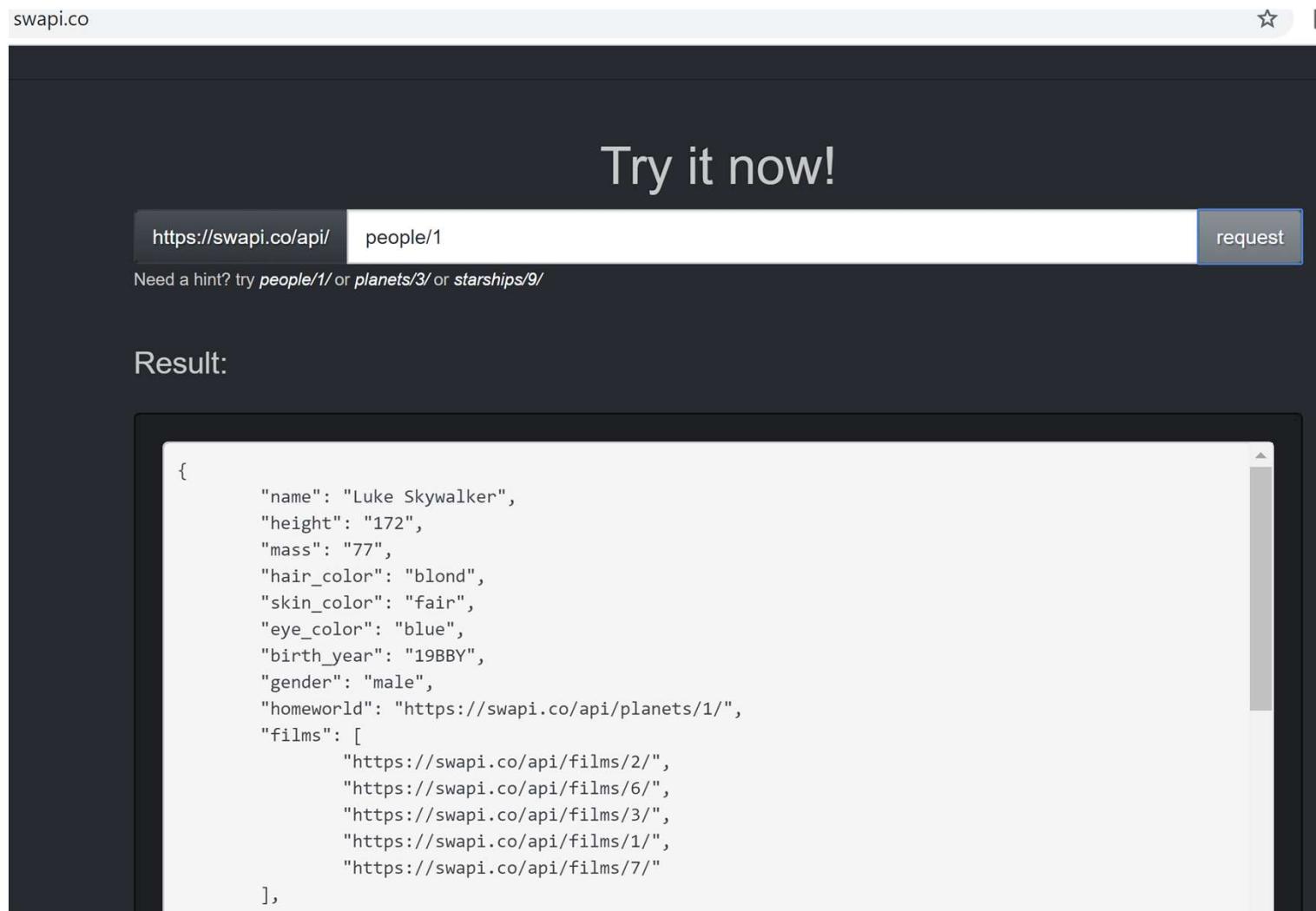
- API sobre el universo de Star Wars:





# Introducción

- Esta web nos permite probar peticiones sobre sus datos:

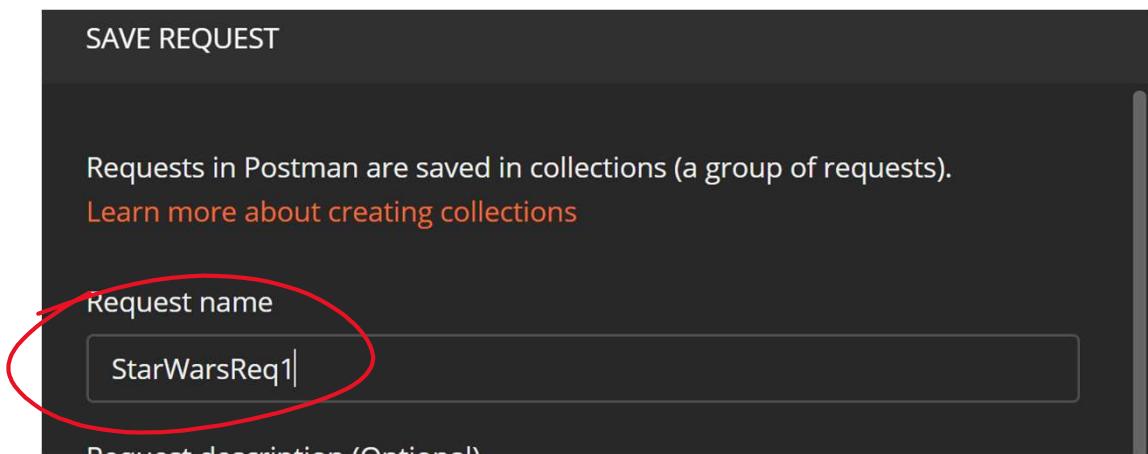
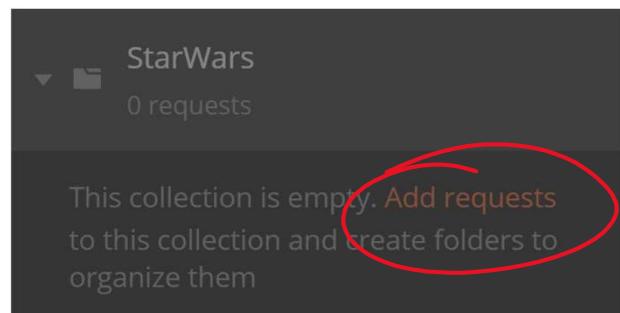


The screenshot shows a web browser window for [swapi.co](https://swapi.co). The URL bar contains `https://swapi.co/api/people/1`. A large button in the center says "Try it now!". Below the URL bar, there is a hint: "Need a hint? try `people/1/` or `planets/3/` or `starships/9/`". To the right of the URL bar is a blue "request" button. The main content area is titled "Result:" and displays the JSON response for Luke Skywalker:

```
{  
  "name": "Luke Skywalker",  
  "height": "172",  
  "mass": "77",  
  "hair_color": "blond",  
  "skin_color": "fair",  
  "eye_color": "blue",  
  "birth_year": "19BBY",  
  "gender": "male",  
  "homeworld": "https://swapi.co/api/planets/1/",  
  "films": [  
    "https://swapi.co/api/films/2/",  
    "https://swapi.co/api/films/6/",  
    "https://swapi.co/api/films/3/",  
    "https://swapi.co/api/films/1/",  
    "https://swapi.co/api/films/7/"  
],  
  "species": [  
    "https://swapi.co/api/species/1/",  
    "https://swapi.co/api/species/12/",  
    "https://swapi.co/api/species/13/",  
    "https://swapi.co/api/species/14/",  
    "https://swapi.co/api/species/15/"  
],  
  "planets": [  
    "https://swapi.co/api/planets/1/",  
    "https://swapi.co/api/planets/8/",  
    "https://swapi.co/api/planets/10/",  
    "https://swapi.co/api/planets/12/",  
    "https://swapi.co/api/planets/16/"  
],  
  "starships": [  
    "https://swapi.co/api/starships/1/",  
    "https://swapi.co/api/starships/13/",  
    "https://swapi.co/api/starships/22/",  
    "https://swapi.co/api/starships/27/",  
    "https://swapi.co/api/starships/31/"  
],  
  "vehicles": [  
    "https://swapi.co/api/vehicles/1/",  
    "https://swapi.co/api/vehicles/14/",  
    "https://swapi.co/api/vehicles/31/",  
    "https://swapi.co/api/vehicles/32/",  
    "https://swapi.co/api/vehicles/33/"  
],  
  "species": [  
    "https://swapi.co/api/species/1/",  
    "https://swapi.co/api/species/12/",  
    "https://swapi.co/api/species/13/",  
    "https://swapi.co/api/species/14/",  
    "https://swapi.co/api/species/15/"  
]  
}
```

# Introducción

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



# Introducción

- Creamos la petición y probamos otras peticiones:

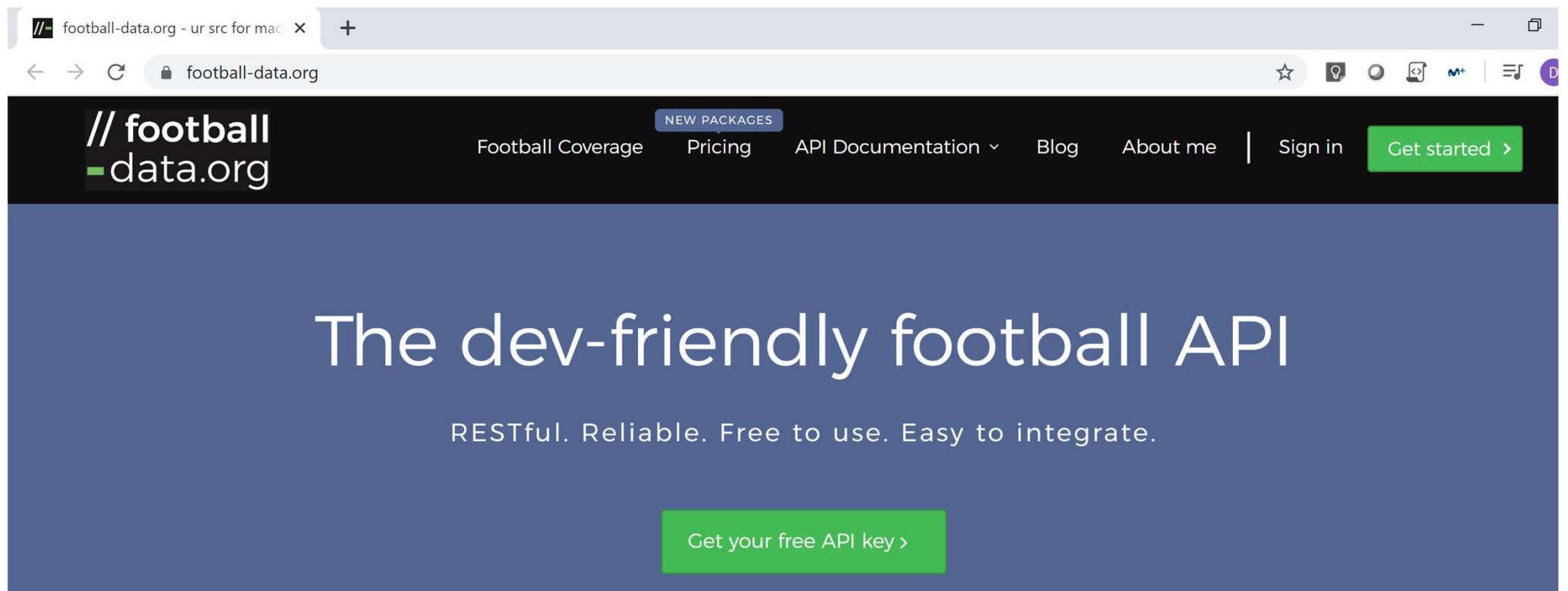
The screenshot shows the Postman application interface. At the top, there's a header bar with a search field, a 'Comments' section, and a 'Send' button. Below the header, a navigation bar includes 'GET StarWarsReq1', a '+' icon, a '...' icon, and a dropdown menu labeled 'Curso'. A red oval highlights the URL 'https://swapi.co/api/people/1' in the main request area. Another red oval highlights the 'Send' button. The main content area displays a 'GET' request to 'https://swapi.co/api/people/1'. Below the request, tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible, with 'Params' being the active tab. A 'Query Params' table is shown with columns for KEY, VALUE, and DESCRIPTION. The 'Body' tab is selected at the bottom, showing a JSON response for Luke Skywalker. The response is displayed in a 'Pretty' format, listing his name, height, mass, hair color, skin color, eye color, birth year, gender, and homeworld. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 223ms', 'Size: 1.15 KB', and a 'Save' button.

```
1 [ {  
2   "name": "Luke Skywalker",  
3   "height": "172",  
4   "mass": "77",  
5   "hair_color": "blond",  
6   "skin_color": "fair",  
7   "eye_color": "blue",  
8   "birth_year": "19BBY",  
9   "gender": "male",  
10  "homeworld": "https://swapi.co/api/planets/1/",  
11 } ]
```



# Introducción

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



football-data.org - ur src for mac

// football  
-data.org

NEW PACKAGES

Football Coverage Pricing API Documentation Blog About me | Sign in Get started >

# The dev-friendly football API

RESTful. Reliable. Free to use. Easy to integrate.

Get your free API key >



# Introducción

- Hay versiones gratuitas y de pago:

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
<a href="#">Get Started &gt;</a>			



# 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 ▾

⇄A inglés ▾ > español ▾ Traducir men

Hi David,

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

Your API email: [david.gr4n4d4@gmail.com](mailto:david.gr4n4d4@gmail.com)

Your API token: 1278b5ae3ab74ac199cef459dbea

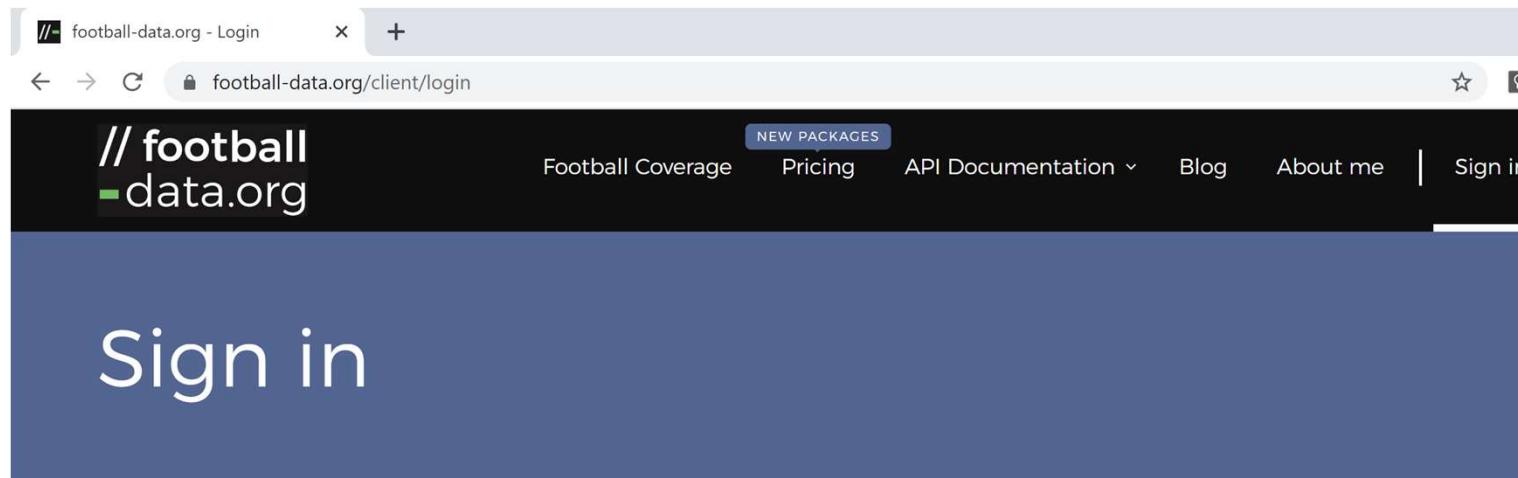
Your API plan: Free Tier

In order to keep your account active and receive up



# Introducción

- Nos logueamos con el API Token:



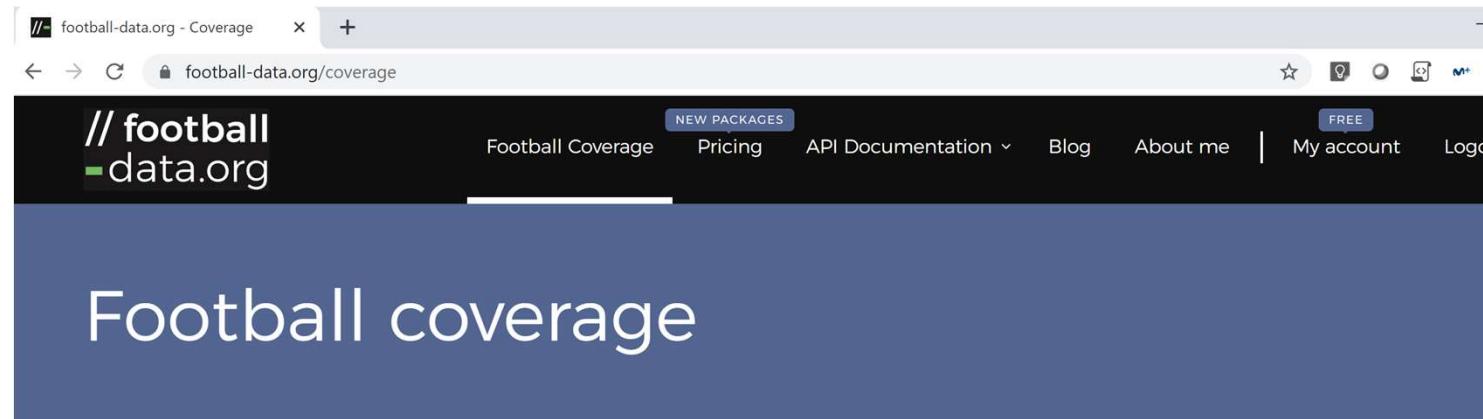
david.gr4n4d4@gmail.com

API Token



# Introducción

- La cuenta gratuita permite obtener datos de 12 competiciones:



## Free Tier

Access to data of these leagues & cups is free. Forever.



CHAMPIONS LEAGUE



PRIMEIRA LIGA



PREMIER LEAGUE



EREDIVISIE



BUNDESLIGA



LIGUE 1

# Introducción

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



- Vemos que hay un botón de Run in Postman:

A screenshot of the 'Available resources' section of the API documentation. The page title is 'Available resources'. Below it, there is a table with columns: '(Sub)Resource', 'Action', 'URI', 'Filters', and 'Sample'. The first row shows 'Competition' with the action 'List all available competitions.', URI '/v2/competitions/', filters 'areas={AREAS} plan={PLAN}', and a 'Sample' button. At the bottom right of the table, there is a large red circle highlighting a 'Run in Postman' button. On the left side of the page, there is a sidebar with tabs for 'Overview' and 'Available resources', and a modal window for 'Available resources' with options like 'Postman for Chrome' and 'Postman for Windows'.

# Introducción

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

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Collections' (highlighted with a red oval), 'History', 'APIs', and 'Trash'. A 'New Collection' button is visible. Below the sidebar, a list of API requests is shown under the 'api v2 Free Tier' collection, including 'Area List Resource', 'Area Resource', 'Competition List Resource' (highlighted with a red oval), 'Competition Resource', 'Competition Resource By C...', 'Competition / Match SubR...', 'Competition / Team Subre...', 'Competition / Standings', 'Team Resource', and 'Team / Match Subresource'. The main workspace shows a 'GET Competition List Resource' request. The request details include the method 'GET', URL 'https://api.football-data.org/v2/competitions', and a 'Params' tab. The response body is displayed in JSON format, showing a list of competitions with one entry: { "id": 2006, "area": { }. The 'Send' button is highlighted with a red oval at the bottom right of the request card.

# Introducción

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

The screenshot shows the Postman application interface. At the top, there's a navigation bar with a collection icon, the word 'CURSO', and a search bar. Below the navigation bar, the 'Player Resource' collection is selected, indicated by a red circle around its name in the left sidebar. In the center, a GET request is being prepared to the URL `https://api.football-data.org/v2/players/44`. The 'Send' button, located on the right side of the request details, is also circled in red. At the bottom of the screen, the response body is displayed in JSON format, showing a 403 Forbidden error message: 

```
1 {  
2   "message": "The resource you are looking for is restricted. Please pass a valid API token and check your subscription permission.",  
3   "errorCode": 403  
4 }
```

# Introducción

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

The screenshot shows the Postman application interface. A red circle highlights the 'Player Resource' item in the left sidebar. Another red circle highlights the 'Send' button in the top right. A large red oval encloses the 'Headers (8)' tab and its contents. A red arrow points from the bottom of this oval to the JSON response body at the bottom of the screen.

GET Player Resource

Player Resource

GET https://api.football-data.org/v2/players/44

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers (1)

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> X-Auth-Token	1278b5ae3ab74ac199cef459dbea5f96	Description
Key	Value	

Temporary Headers (7)

Body Cookies Headers (15) Test Results (6/6) Status: 200 OK Time: 232ms Size: 730 B Save

Pretty Raw Preview Visualize JSON

```
1 [ { "id": 44, "name": "Cristiano Ronaldo", "firstName": "Cristiano Ronaldo", "lastName": null, "dateOfBirth": "1985-02-05", "countryOfBirth": "Portugal", }
```

# Introducción

- Vamos a crearnos nuestra primera colección de servicios web:

The screenshot shows the Postman application interface. On the left, there's a sidebar with tabs for History, Collections (which is highlighted), APIs, and Trash. A red circle highlights the '+ New Collection' button under the Collections tab. The main area is a 'CREATE A NEW COLLECTION' dialog. It has fields for 'Name' (containing 'Collection1'), 'Description' (containing 'Esta es la colección de prueba para el curso'), and 'Authorization', 'Pre-request Scripts'. At the bottom are 'Cancel' and 'Create' buttons, with the 'Create' button circled in red.

The screenshot shows the Postman application interface after the collection was created. The sidebar now lists 'Collection1' under the Collections tab. The main area shows the collection details: 'Collection1' with '0 requests' and a note 'Postman Echo ☆ 37 requests'. The 'Create' button from the previous screen is also circled in red.

# Introducción

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

The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New' (highlighted in orange), 'Import', 'Runner', and 'My Workspace'. The left sidebar features a 'Filter' search bar, 'History', 'Collections' (selected and highlighted in orange), 'APIs', '+ New Collection', and two collections: 'Collection1' (0 requests) and 'Postman Echo' (37 requests). The main workspace displays an 'Untitled Request' for a 'GET' method to 'https://postman-echo.com/get'. A red oval highlights the URL field. Below the method and URL are tabs for 'Params', 'Authorization', 'Headers', 'Body', and 'Pre-request Scripts'. The 'Params' tab is selected. The 'Query Params' table has columns 'KEY' and 'VALUE', with a single row containing 'Key' and 'Value'. The bottom section is labeled 'Response'.

# Introducción

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

The screenshot shows the Postman application interface. At the top, there's a header bar with a 'GET' method, the URL 'https://postman-echo.com/get', and a 'Send' button. Below the header is a toolbar with tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', 'Settings', 'Cookies', and 'Code'. The 'Headers' tab is selected. A red circle highlights the 'Send' button. In the main body, under 'Query Params', there's a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. A red circle highlights the 'Body' tab. At the bottom, there's a preview area showing a JSON response with a red circle highlighting the first few lines. The response status is 'Status: 200 OK'. The page number '27' is in the bottom right corner.

GET https://postman-echo.com/get

No Environment

Comments (0)

Send

Untitled Request

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

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies (1) Headers (9) Test Results

Pretty Raw Preview Visualize JSON

1 {  
2 "args": {},  
3 "headers": {  
4 "x-forwarded-proto": "https",  
5 "host": "postman-echo.com",  
6 "accept": "\*/\*",  
7 "accept-encoding": "gzip, deflate, br",  
8 "cache-control": "no-cache",  
9 "postman-token": "fd2f4822-07d5-4d73-8847-ebf80d710ddd",

Status: 200 OK Time: 445ms Size: 602 B Save Response

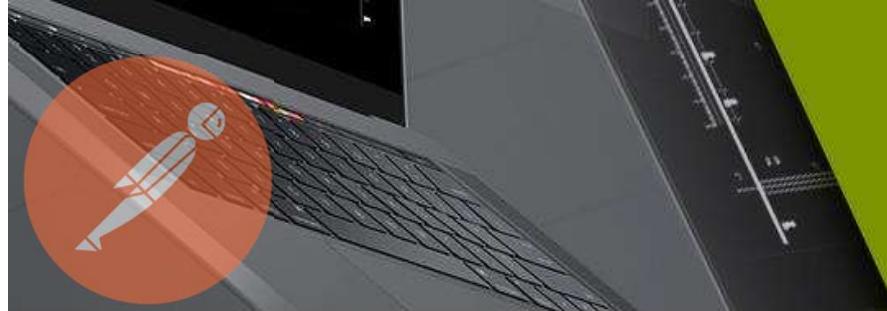
27

# Introducción

- Podemos guardar esta primera petición en nuestra colección:

The screenshot illustrates the workflow for saving a request in Postman. It consists of three main panels:

- SAVE REQUEST Panel (Left):** A modal window titled "SAVE REQUEST". It contains a message about collections, a "Request name" input field containing "Prueba Get", and a "Request description (Optional)" input field. A red circle highlights the "Request name" field. Below it, a dropdown menu shows "Collection1" selected under "Search for a collection or folder". A red circle highlights "Collection1". At the bottom are "Cancel" and "Save to Collection1" buttons, with the latter also highlighted by a red circle.
- Main Interface Panel (Top Right):** Shows the "Send" and "Save" buttons. The "Save" button has a dropdown menu with "Cookies" and "Code" options. A red circle highlights the "Save" button.
- Collections Panel (Bottom Right):** A sidebar with tabs for "History", "Collections" (which is active), and "APIs". Under "Collections", there is a "+ New Collection" button and a list of collections. "Collection1" is expanded, showing it contains "1 request" and listing the "GET Prueba Get" request. A red circle highlights "Collection1". Below it, another collection "Postman Echo" is listed with "37 requests".



# Introducción

- Duplicamos esa petición para incluir algunos parámetros:

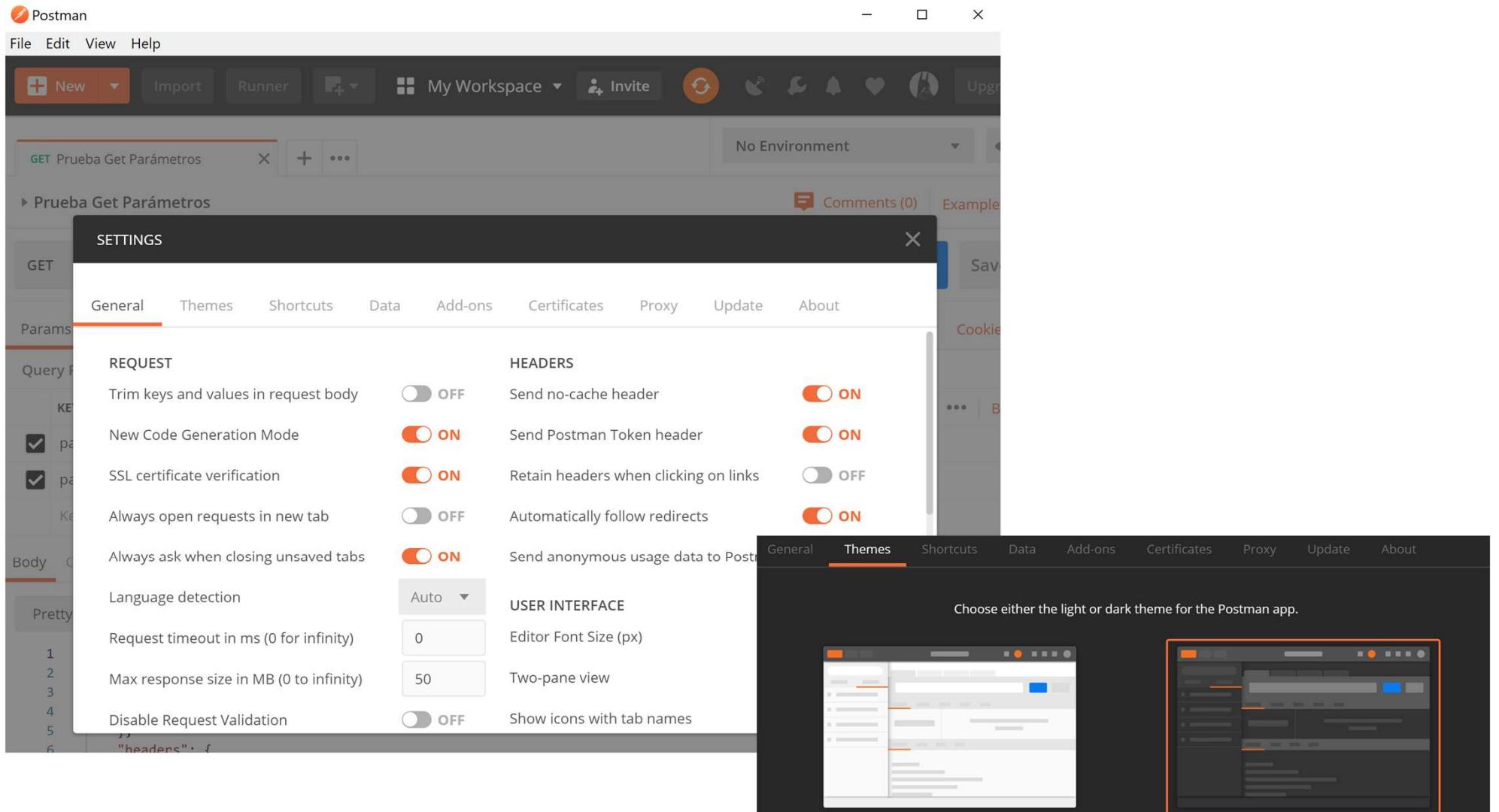
# Introducción

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

The screenshot shows the Postman web application interface. At the top, there's a dark header bar with the text "My Workspace" and "Invite". Below this is a browser-like toolbar with tabs, a refresh button circled in red, and various icons. The main content area is titled "Collection1 - Documentation". It displays a collection named "Collection1" by "DavidGranada". The "Documentation" tab is selected. The collection page contains an introduction section with the text "Esta es la colección de prueba para el curso". Below this are two API endpoints: "Prueba Get" and "Prueba Get Parámetros". Each endpoint has a "GET" method listed, along with a "curl" command example and a "Comments (0)" link. The right side of the screen shows a sidebar with "Language: cURL - cURL" and "CURRENT" selected.

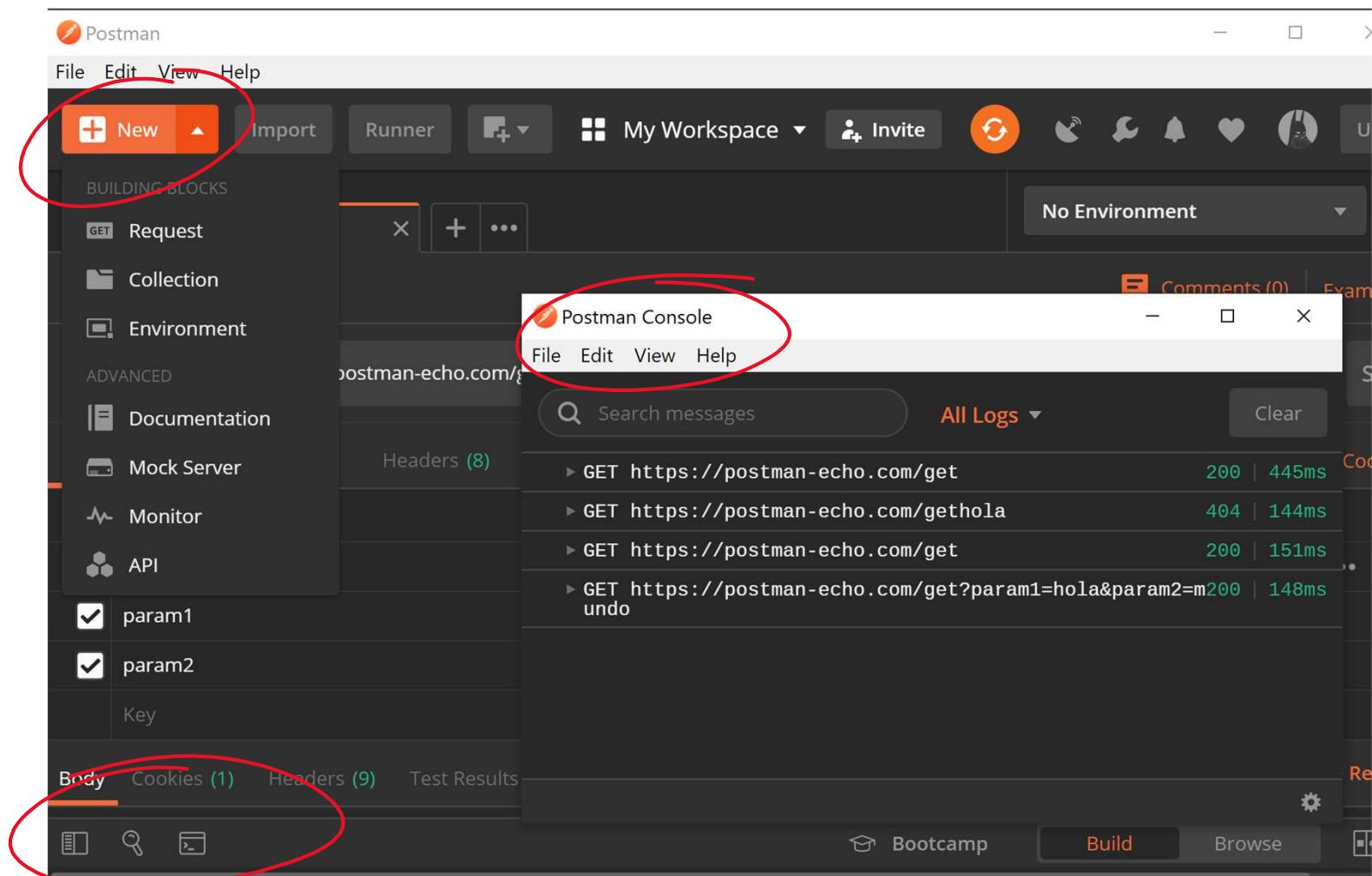
# Introducción

- 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



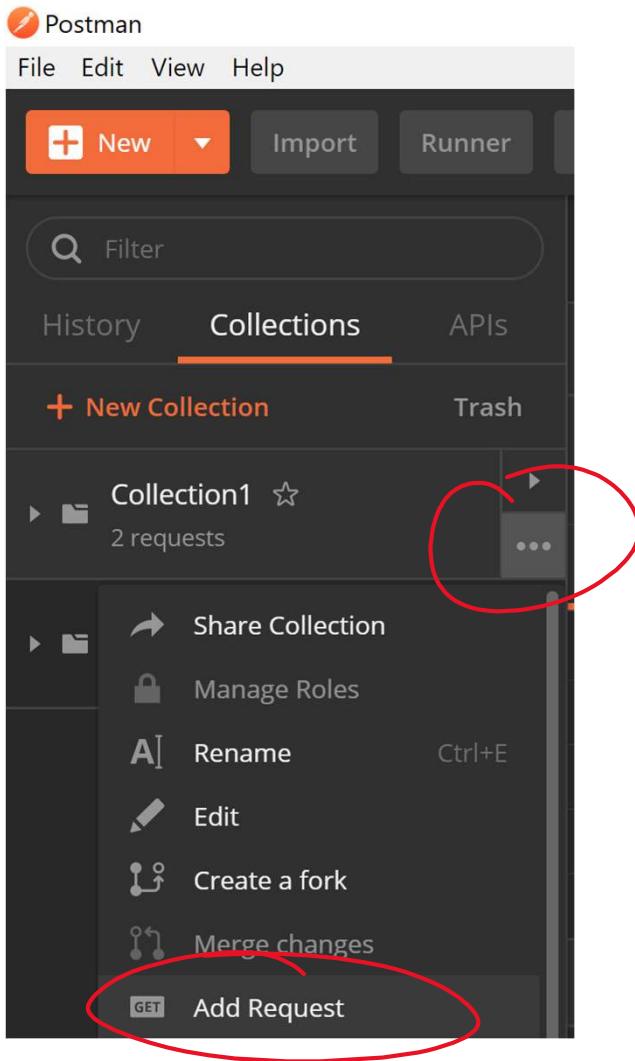
- Una API REST es lo que estamos acostumbrados a hacer a la hora de realizar peticiones a alguna API desarrollada por una empresa.
- Algunos ejemplos de llamadas API REST son las típicas llamadas de GET, POST, PUT y DELETE.
- GET para consultar y leer, POST para crear, PUT para editar y DELETE para eliminar.

- 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.

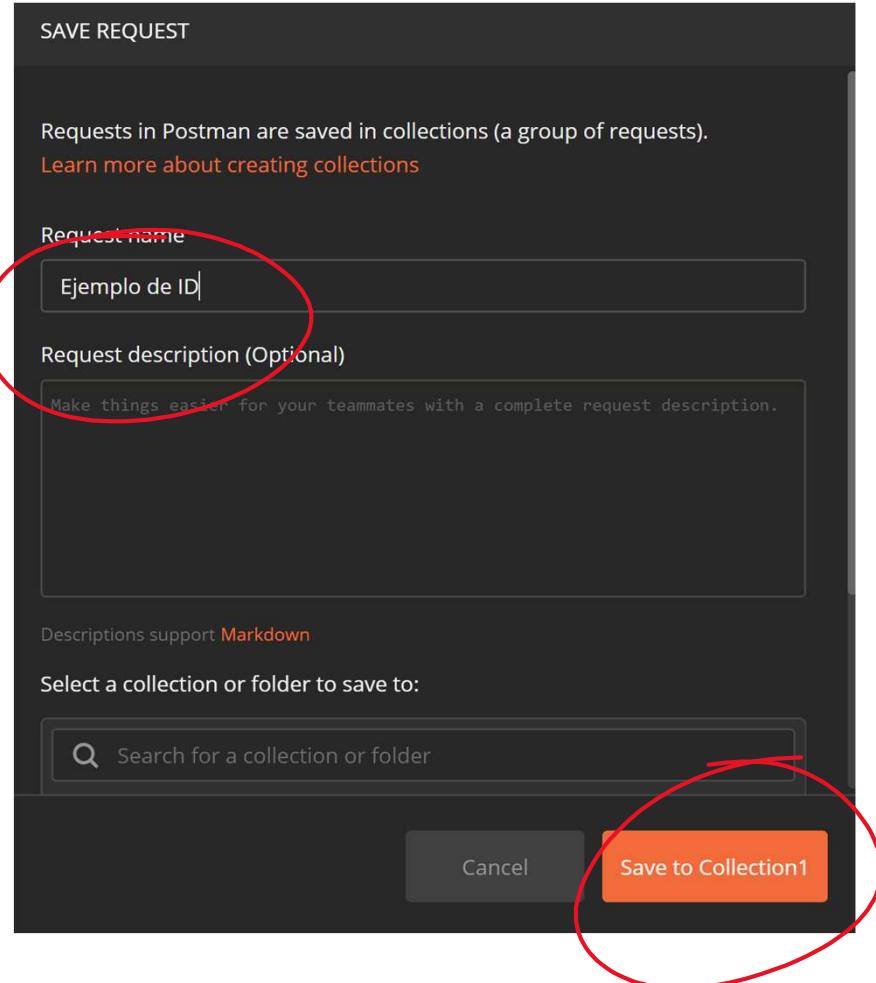
- Hoy en día la mayoría de las empresas utilizan API REST para crear servicios.
- Esto se debe a que es un estándar lógico y eficiente para la creación de servicios web.
- Por poner algún ejemplo tenemos los sistemas de identificación de Facebook, la autenticación en los servicios de Google (hojas de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

# Ejemplo GET

- Vamos a añadir una nueva petición:



The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New', 'Import', and 'Runner'. Below the navigation is a search bar labeled 'Filter'. The main area has tabs for 'History', 'Collections' (which is highlighted with an orange underline), and 'APIs'. A 'New Collection' button is visible. On the left, there's a sidebar with options like 'Share Collection', 'Manage Roles', 'Rename' (with 'Ctrl+E'), 'Edit', 'Create a fork', 'Merge changes', and 'Add Request' (which is circled in red). A collection named 'Collection1' is listed with '2 requests'. A red circle also highlights the three-dot menu icon next to it.



A 'SAVE REQUEST' dialog box is open. It contains instructions: 'Requests in Postman are saved in collections (a group of requests)' and 'Learn more about creating collections'. The 'Request name' field is filled with 'Ejemplo de ID' and is circled in red. The 'Request description (Optional)' field is present but empty. Below it, it says 'Descriptions support Markdown'. A section for selecting a collection to save to is shown with a search bar 'Search for a collection or folder' and a red circle around the 'Save to Collection1' button.

# Ejemplo GET

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

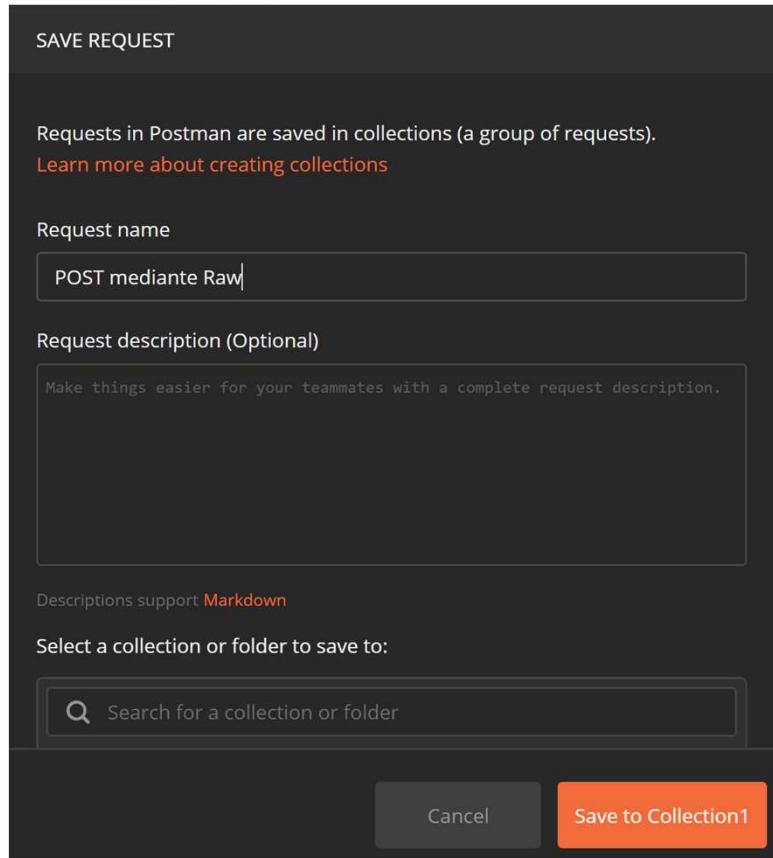
The screenshot shows the Postman application interface. At the top, there's a header bar with a 'GET Ejemplo de ID' button, a '+' icon, and a '...' icon. To the right, it says 'No Environment'. Below the header, the main area has a title 'Ejemplo de ID' with a dropdown arrow. Underneath, a 'GET' method is selected, and the URL 'https://postman-echo.com/get?id=200' is displayed. A large red oval highlights this URL. To the right of the URL is a blue 'Send' button with a dropdown arrow, also highlighted by a red oval. Below the URL, tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible, with 'Params' being the active tab. Under 'Query Params', there's a table with one row: 'id' (checked) with value '200'. The 'Body' tab is selected at the bottom, showing a JSON response:

```
1 [ { "args": { "id": "200" }, "headers": { "x-forwarded-proto": "https", "host": "postman-echo.com" } } ]
```

At the bottom right, status information is shown: 'Status: 200 OK', 'Time: 538ms', 'Size: 620 B', and a 'Save' button.

# 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:

The screenshot shows the Postman interface for a POST request to `https://postman-echo.com/post`. The 'Body' tab is selected, and the 'raw' option is chosen. A red circle highlights the 'raw' button. Another red circle highlights the 'Text' dropdown menu. The request body contains the text: `1 Esto es una cadena de texto`. A red arrow points from the bottom right towards the JSON response area, which displays the received data.

```
1 [
2   "args": {},
3   "data": "",
4   "files": {},
5   "form": {
6     "Esto es una cadena de texto": ""
7 }
```

# Ejemplo POST

- Podemos enviar datos mediante un formulario:

POST POST mediante formulario

POST mediante formulario

No Environment

Comments (0) Examples

Send

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

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

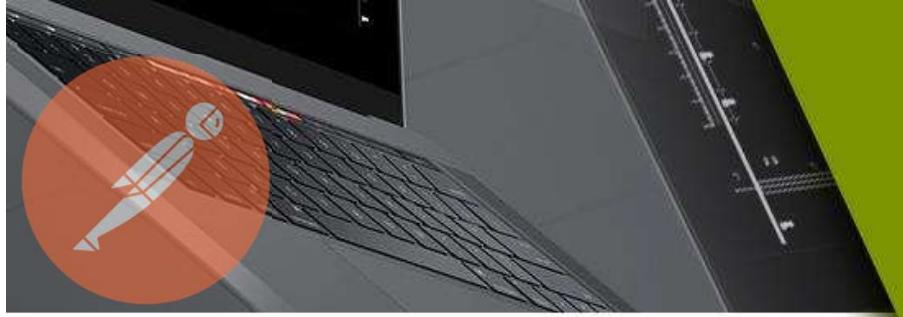
none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION
nombre	David	
apellido	Granada	
Key	Value	Description

Body Cookies (1) Headers (8) Test Results Status: 200 OK Time: 104ms Size: 660 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [ { "args": {}, "data": "", "files": {}, "form": { "nombre": "David", "apellido": "Granada" } }, ]
```



# ARRAY mediante POST

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

The screenshot shows the Postman interface with a POST request to `https://postman-echo.com/post`. The Body tab is active, displaying the following JSON payload:

```
1 {  
2   "pedido":20,  
3   "productos": [  
4     {  
5       "id":40,  
6       "cantidad":1  
7     },  
8     {  
9       "id":53,  
10      "cantidad":15  
11    }  
12  ]  
13 }  
14 }  
15 }
```

The response at the bottom shows:

```
Status: 200 OK Time: 111ms Size: 792 B Save
```

Body Cookies (1) Headers (9) Test Results

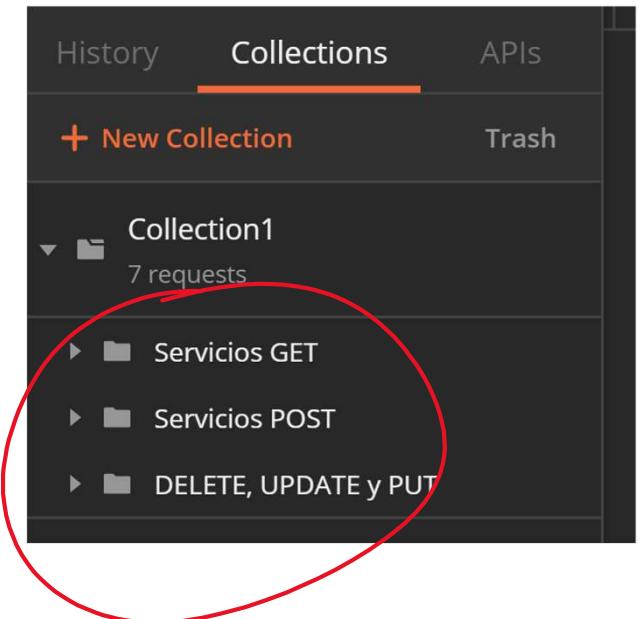
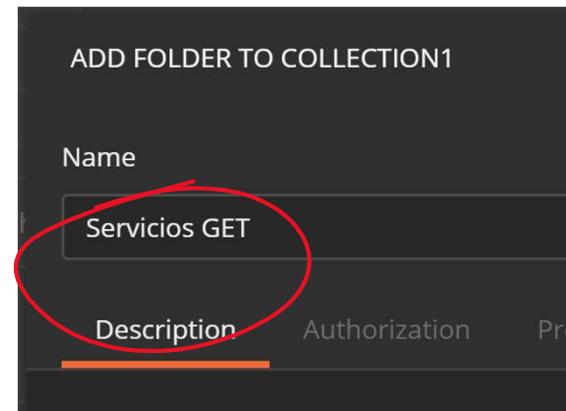
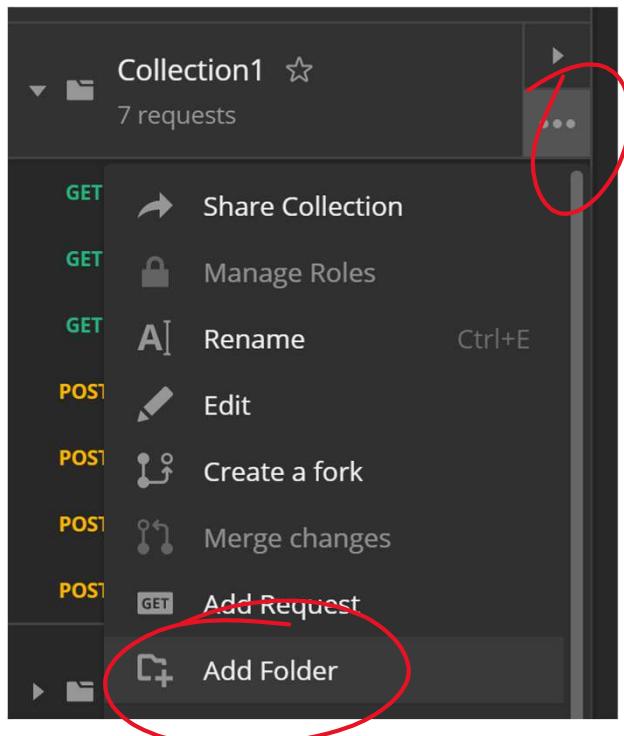
Pretty Raw Preview Visualize JSON

```
1 [{}  
2   "args": {},  
3   "data": {  
4     "pedido": 20,  
5     "productos": [
```

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:

The screenshot shows the Postman interface for a PUT request. The URL is set to `https://postman-echo.com/put`. The request method is highlighted with a red circle. The 'Body' tab is selected, also highlighted with a red circle. The raw JSON body is defined as:

```
1 {
2   "id":50,
3   "nombre": "TV 50 pulgadas"
4 }
```

The 'Send' button is also circled in red. The response status is 200 OK with a size of 773 B.

Below the main request, there is a preview of the response body, which includes additional fields like 'args' and 'data':

```
1 []
2   "args": {},
3   "data": {
4     "id": 50,
5     "nombre": "TV 50 pulgadas"
6   }
```

# Ejemplo DELETE

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

The screenshot shows the Postman interface with a DELETE request setup. The URL is set to `https://postman-echo.com/delete`. The Body tab is selected, showing a raw JSON payload:

```
1 {  
2   "codigo":47  
3 }
```

Red circles highlight several key areas: the method dropdown (DELETE), the URL field, the Body tab, the raw JSON input field, and the Send button. A red arrow points from the bottom JSON response back up to the raw input field, indicating they are connected.

At the bottom, the response status is shown as `Status: 200 OK`, `Time: 109ms`, and `Size: 756 B`.



# 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

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

The screenshot shows the Postman application interface. At the top, there's a header bar with a logo and the word "HEADER". Below it, the main interface has a title "GET Ejemplo de ID" and a URL "https://postman-echo.com/get?id=200". On the right side, there's a "Send" button highlighted with a red circle. In the center, under the "Headers" tab, there's a table with one row: "User" (Key) and "David" (Value). A red circle highlights the "Headers" tab, and another red circle highlights the "Send" button. Below the table, there are tabs for "Temporary Headers" and "Body". The "Body" tab is selected, showing a JSON response with a "user" key. Red arrows point from the "User" header entry in the table to the "user" value in the JSON response. The JSON response also includes other headers like "x-forwarded-proto", "host", "accept", etc.

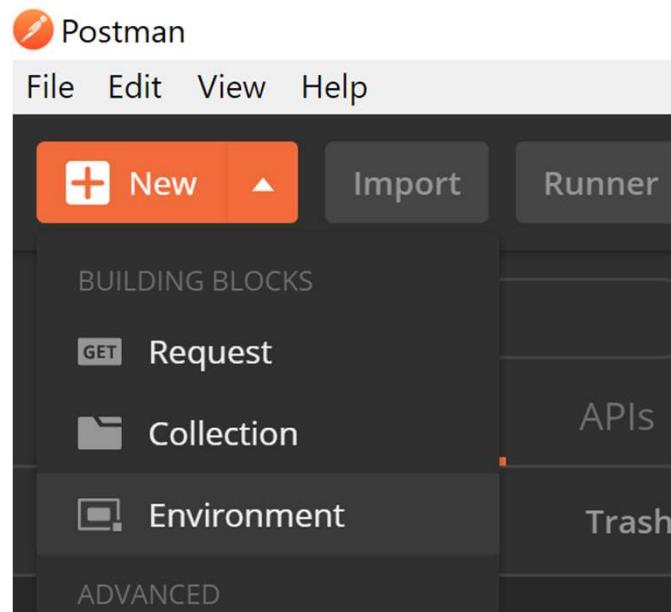
KEY	VALUE	DESCRIPTION
User	David	Description

```
1  {
2    "args": {
3      "id": "200"
4    },
5    "headers": {
6      "x-forwarded-proto": "https",
7      "host": "postman-echo.com",
8      "accept": "*/*",
9      "accept-encoding": "gzip, deflate, br",
10     "cache-control": "no-cache",
11     "cookie": "sails.sid=s%3A86fEV05AmqbMB9PzjXjuu4hVTYARdfG_.9P%2Fg%2Fjt0pI6L9c1agk2%2BTXak5MZ5ch3UH7QP7yFIK0A",
12     "postman-token": "9fe95f0e-bc2c-4a31-ad46-29475c64e06d",
13     "user": "David",
14   }
15 }
```



# 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):

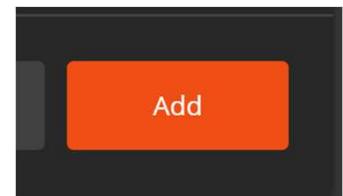


MANAGE ENVIRONMENTS

Add Environment

Curso

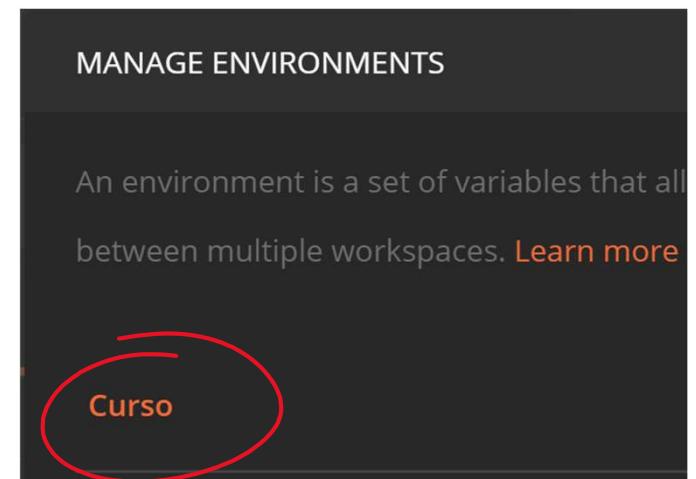
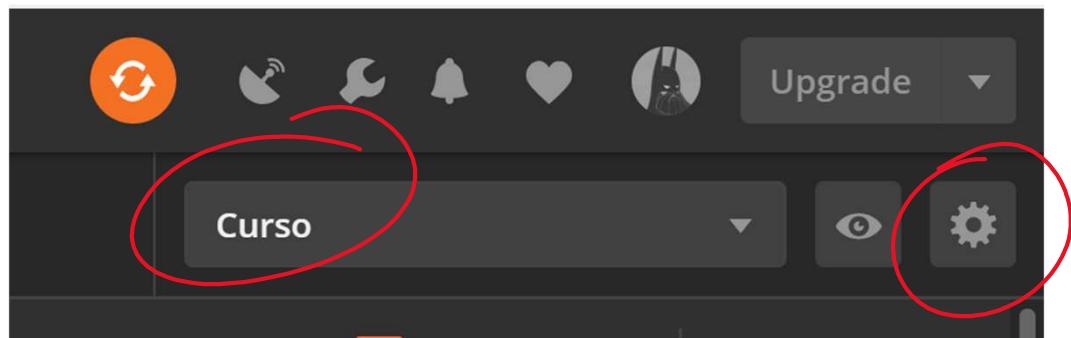
	VARIABLE	INITIAL VALUE ⓘ
	Add a new variable	





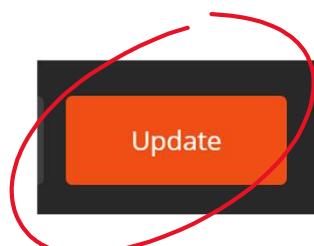
# VARIABLES DE ENTORNO

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



The screenshot shows the 'Environment Name' section with 'Curso' selected. Below it is a table titled 'Environment Variables'. The table has columns: VARIABLE, INITIAL VALUE, CURRENT VALUE, and a three-dot menu. A single row is shown for 'urlecho', with 'https://postman-echo...' in the initial value column and 'https://postman-echo.com' in the current value column. The entire table area is circled with a red oval. In the bottom left corner of the table area, the placeholder text 'Add a new variable' is visible.

VARIABLE	INITIAL VALUE	CURRENT VALUE	...
<input checked="" type="checkbox"/> urlecho	https://postman-echo...	https://postman-echo.com	



# Variables de Entorno

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

The screenshot shows the Postman application interface. On the left, the sidebar lists collections like 'Collections' (selected), 'New Collection', 'Collection1' (with 10 requests), 'Servicios GET' (containing 'Prueba Get', 'Prueba Get Parámetros', 'Ejemplo de ID', and 'GET variables de entorno'), 'Servicios POST', 'DELETE, UPDATE y PUT', and 'Postman Echo' (with 37 requests). A red circle highlights the 'GET variables de entorno' request in the sidebar.

The main workspace shows a 'GET variables de entorno' request. The URL is {{urlecho}}/get. A red circle highlights this URL. To the right, the 'Send' button is also circled in red.

The request details panel shows 'Params' tab selected, with a 'Query Params' table:

KEY	VALUE	DESCRIPTION
Key	Value	Description

The 'Body' tab shows the response in JSON format:

```
1  {
2      "args": {},
3      "headers": {
4          "x-forwarded-proto": "https",
5          "host": "postman-echo.com",
6          "accept": "*/*",
7          "accept-encoding": "gzip, deflate, br",
8          "cache-control": "no-cache",
9          "postman-token": "3e1756e6-2161-4fe9-9c87-0c2fb8ae7505",
10         "user-agent": "PostmanRuntime/7.22.0",
11         "x-forwarded-port": "443"
12     },
13     "url": "https://postman-echo.com/get"
14 }
```

A red circle highlights the 'url' field in the JSON response.



## 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:

The screenshot shows a Postman collection named "GET variables de entorno". A red circle highlights the "No Environment" dropdown in the top right corner. Another red circle highlights the "Send" button. A large red oval encloses the error message at the bottom left: "Could not get any response" followed by "There was an error connecting to %7B%7Burlecho%7D%7D/get.". Below this, a section titled "Why this might have happened:" lists several potential causes.

GET variables de entorno

No Environment

Comments (0)

Send

Could not get any response

There was an error connecting to %7B%7Burlecho%7D%7D/get.

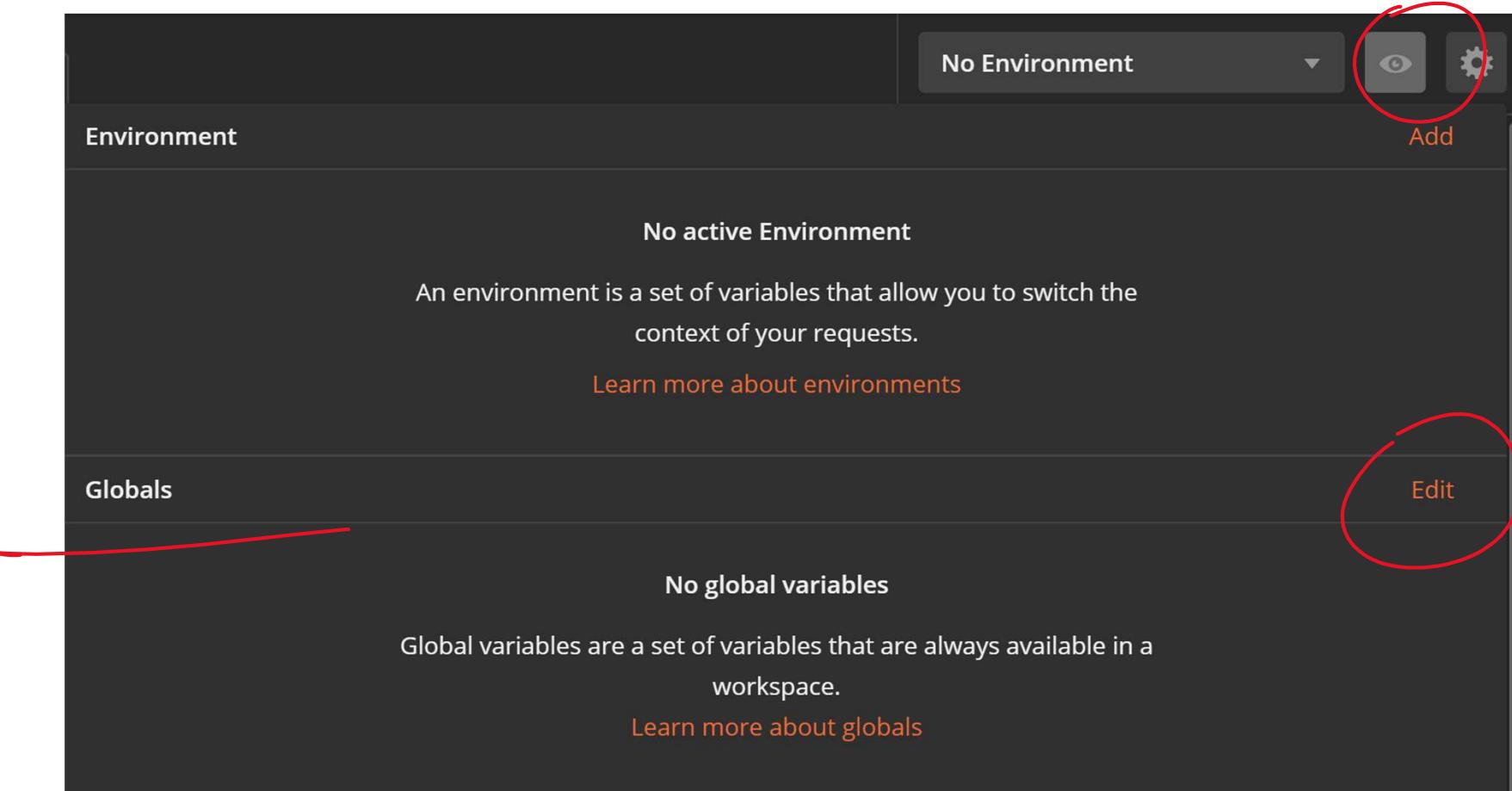
Why this might have happened:

- The server couldn't send a response: Ensure that the backend is working properly
- Self-signed SSL certificates are being blocked: Fix this by turning off 'SSL certificate verification' in *Settings > General*
- Proxy configured incorrectly: Ensure that proxy is configured correctly in *Settings > Proxy*
- Request timeout: Change request timeout in *Settings > General*



# Variables Globales

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



The screenshot shows the Jupyter Notebook interface. At the top, there's a toolbar with a dropdown menu set to "No Environment", an "Add" button, and two icons (eye and gear) circled in red. Below the toolbar, the "Environment" section is visible with the message "No active Environment". It includes a description of what an environment is and a link to learn more. The "Globals" section is also shown with the message "No global variables" and a similar descriptive text with a link. A red line points from the "Edit" button in the Globals section to the "Edit" icon in the toolbar.

No Environment

Add

Environment

No active Environment

An environment is a set of variables that allow you to switch the context of your requests.

Learn more about environments

Globals

No global variables

Global variables are a set of variables that are always available in a workspace.

Learn more about globals

Edit



# Variables Globales

- Creamos la variable global:

MANAGE ENVIRONMENTS

Global variables for a workspace are a set of variables that are always available within the scope of that workspace. They can be viewed and edited by anyone in that workspace. [Learn more about globals](#)

**Globals**

VARIABLE	INITIAL VALUE	CURRENT VALUE	...
<input checked="" type="checkbox"/> urlechoglobal	https://postman-echo...	https://postman-echo.com	
Add a new variable			

Save

X

MANAGE ENVIRONMENTS

57

# Variables Globales

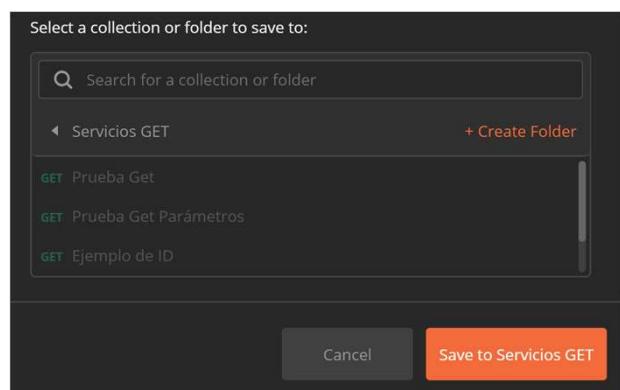
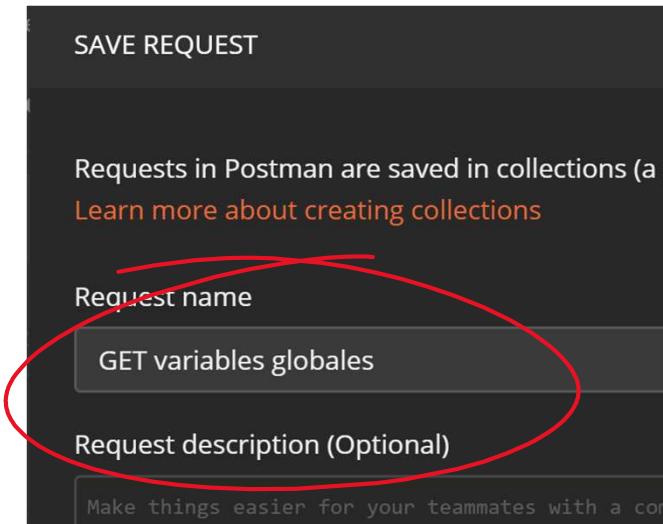
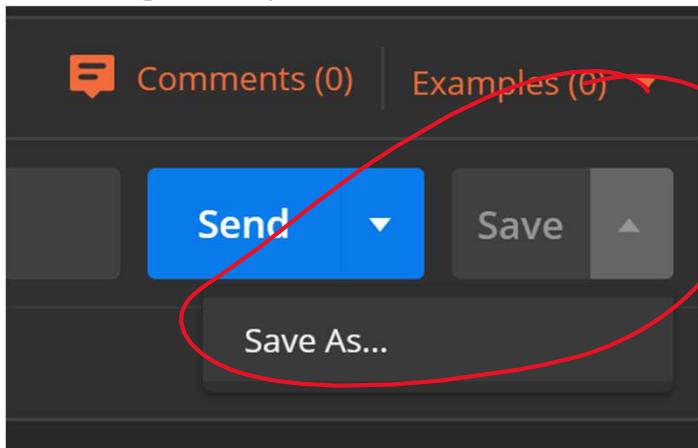
- Ahora podemos probar en ambos entornos y veremos que funciona:

The screenshot shows the Postman application interface. At the top, there's a header bar with a search field containing 'GET variables de entorno'. Below it, a dropdown menu shows 'No Environment' with options to change environment, view comments, and examples. The main workspace shows a GET request for 'variables de entorno'. The URL field contains the placeholder '{{urlechoglobal}}/get'. To the right of the URL is a large red oval. Below the URL, there's a 'Send' button in a blue box, which is also highlighted with a red oval. A red arrow points from the bottom of the 'Send' button towards the JSON response body at the bottom of the screen. The response body is displayed in a table format under the 'Body' tab, showing an empty array. The status bar at the bottom indicates a successful response: 'Status: 200 OK Time: 109ms Size: 706 B'.



# Variables Globales

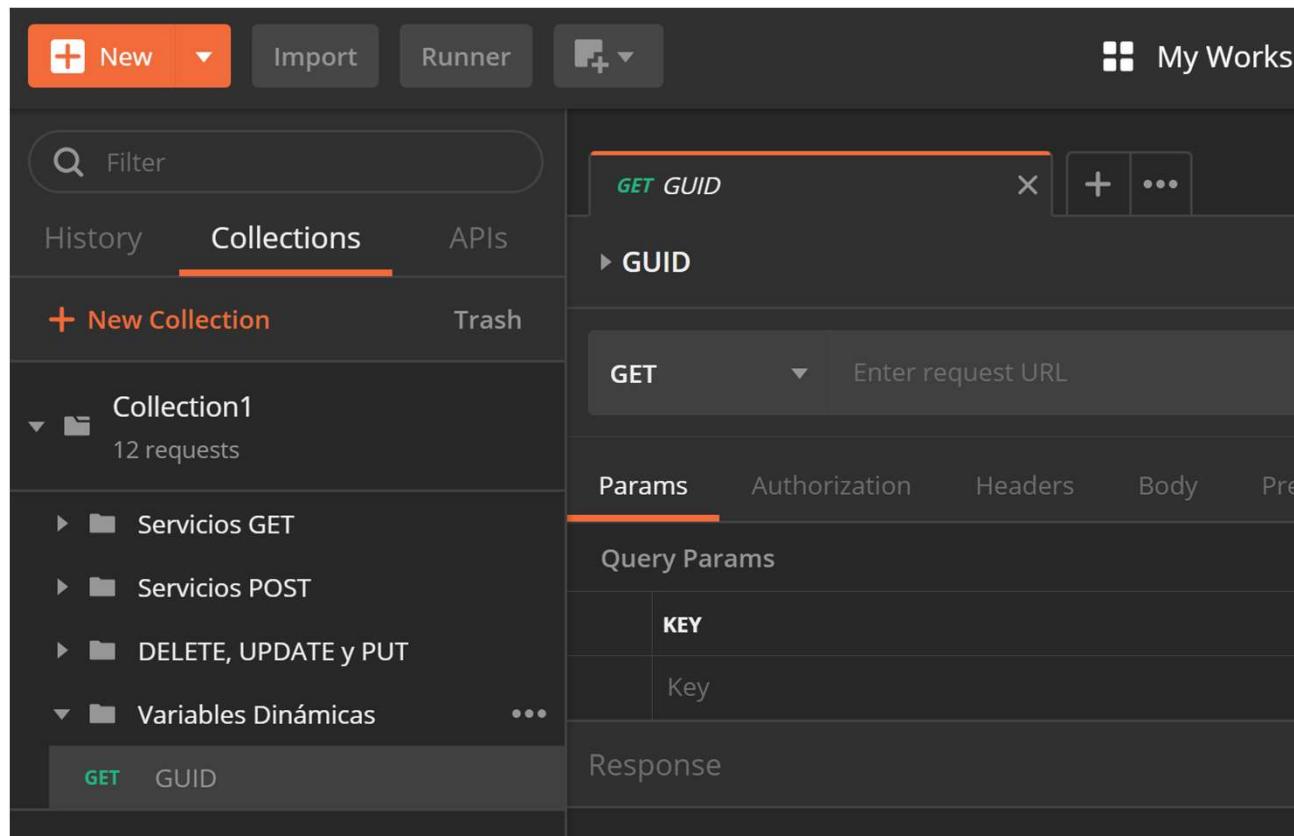
- Guardamos esa petición en la Collection I – Servicios GET:





# Variables Dinámicas

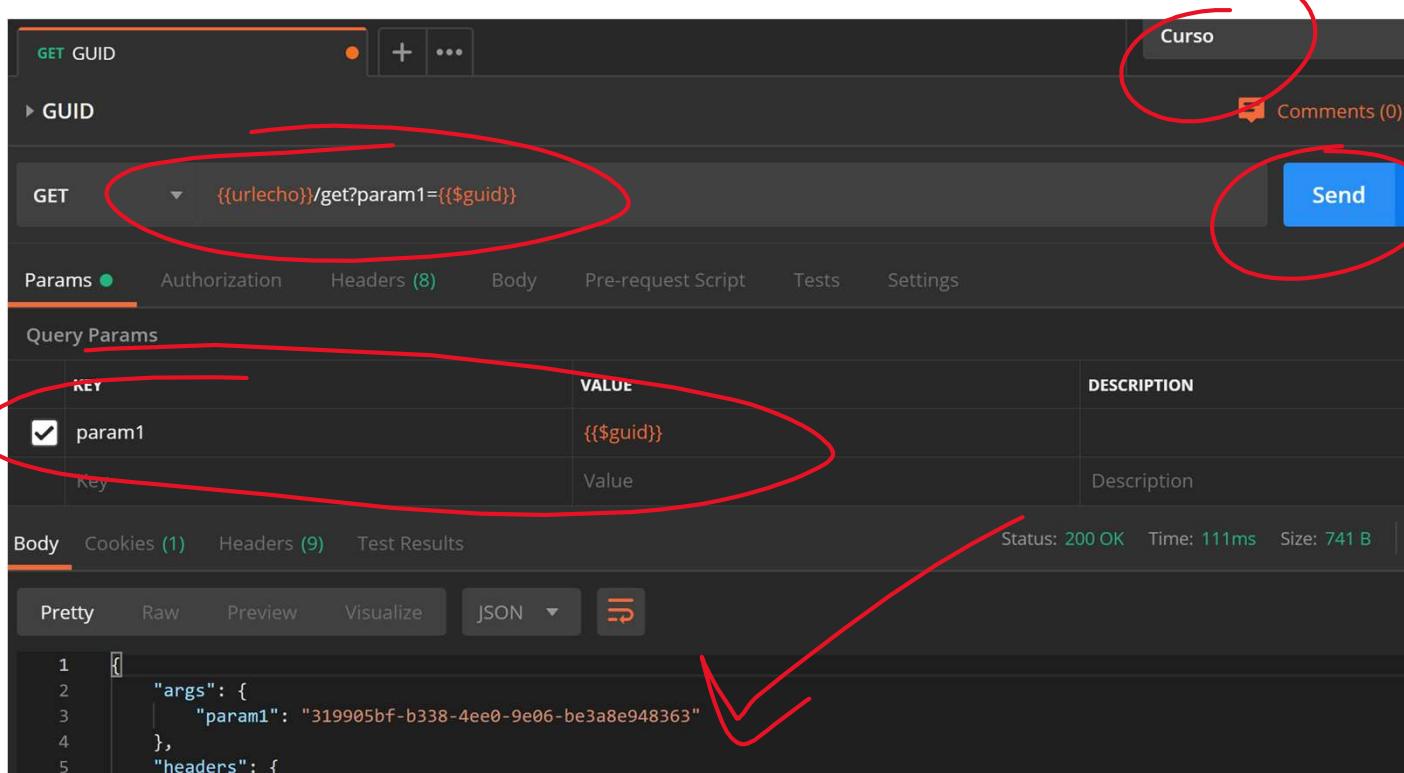
- Vamos a crear una nueva carpeta llamada Variables Dinámicas y dentro creamos una nueva petición llamada GUID:





# 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):



The screenshot shows the Postman interface with the following details:

- Request URL:** {{urlecho}}/get?param1={{\$guid}}
- Send Button:** A blue "Send" button is highlighted with a red oval.
- Body - Params:** A table with one row:

KEY	VALUE	DESCRIPTION
param1	{{\$guid}}	Description
- Response Preview:** The response body is shown in JSON format:

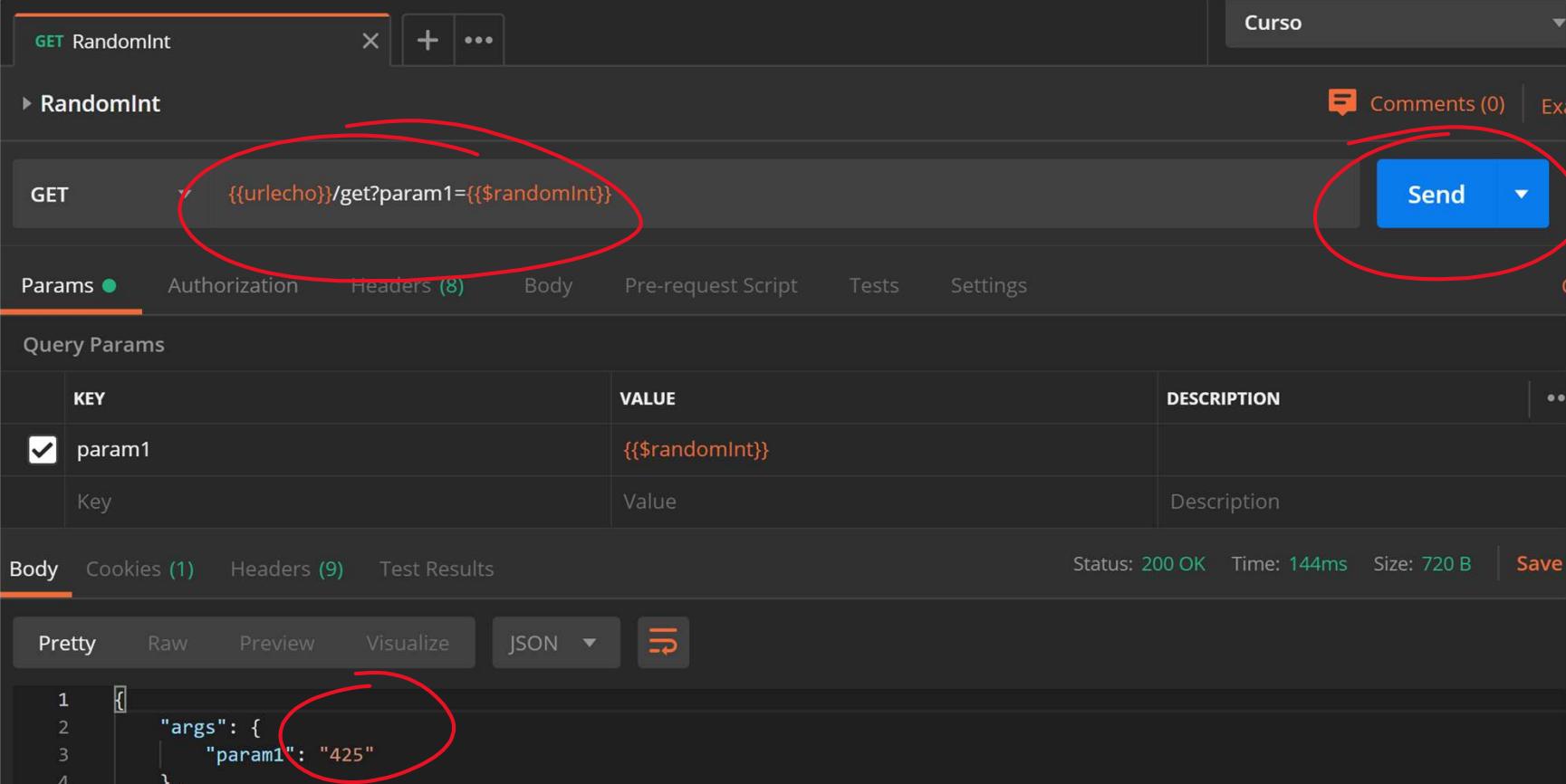
```
1 [ { "args": { "param1": "319905bf-b338-4ee0-9e06-be3a8e948363" }, "headers": { }}
```

A red checkmark points to the "param1" value in the JSON preview.



# Variables Dinámicas

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



The screenshot shows a Postman collection named "RandomInt". A red oval highlights the URL field containing the placeholder `{{urlecho}}/get?param1={{$randomInt}}`. Another red oval highlights the "Send" button. Below the URL, the "Params" tab is selected, showing a table with one row: "param1" with value `{{$randomInt}}`. The "Body" tab at the bottom shows a JSON response with the key "args" and the value "param1": "425". The status bar indicates a successful 200 OK response.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> param1	<code>{{\$randomInt}}</code>	

```
1 {
2   "args": {
3     "param1": "425"
4   },
5 }
```

# Variables Dinámicas

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

The screenshot shows the Postman application interface. A red oval highlights the title bar "GET Tiempo desde la fecha 1Enero1970". Another red oval highlights the URL field containing the placeholder "{{urlecho}}/get?param1={{\$timestamp}}". A third red oval highlights the "Send" button. A fourth red oval highlights the "args" key in the JSON response body, which contains the value "param1": "1582722584".

GET Tiempo desde la fecha 1Enero1970

GET {{urlecho}}/get?param1={{\$timestamp}}

Send

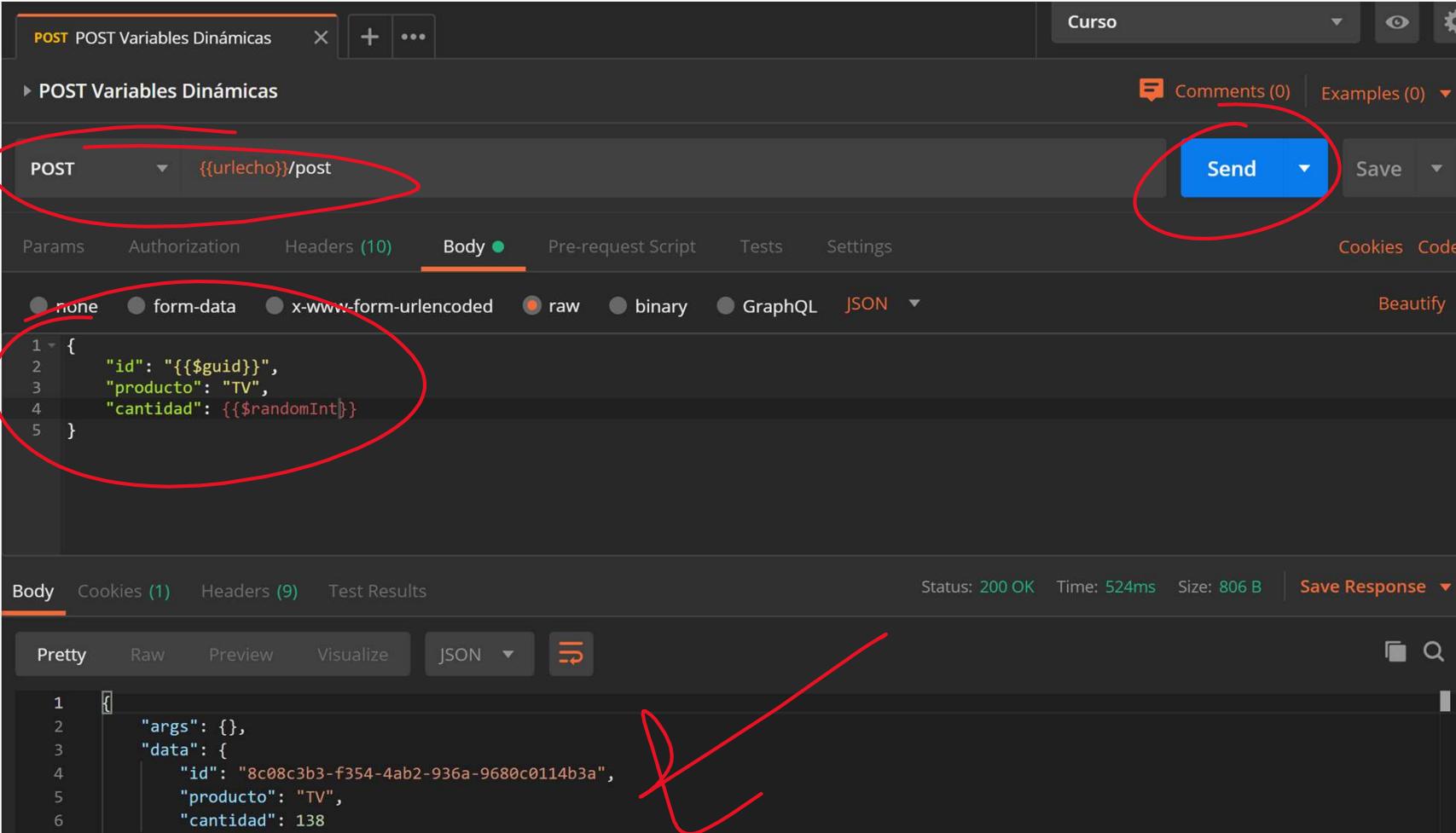
args

param1: 1582722584



# Variables Dinámicas

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



The screenshot shows a POST request in Postman. The URL is {{urlecho}}/post. The Body tab is selected, showing a raw JSON payload:

```
1 {
2   "id": "{$guid}",
3   "producto": "TV",
4   "cantidad": "{$randomInt}"
5 }
```

Red circles highlight the URL field, the 'Send' button, and the JSON body. A red arrow points from the JSON body area down to the response preview, which displays:

```
1 {
2   "args": {},
3   "data": {
4     "id": "8c08c3b3-f354-4ab2-936a-9680c0114b3a",
5     "producto": "TV",
6     "cantidad": 138
7 }
```

# Autenticación y Autorización



# Autenticación básica

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

The screenshot shows the Postman Echo documentation for the 'Basic Auth' endpoint. The left sidebar lists various documentation sections, and the main content area details the 'GET Basic Auth' endpoint.

**POSTMAN ECHO**

Introduction

- Request Methods
- Headers
- Authentication Methods
  - GET Basic Auth
  - GET DigestAuth Success
  - GET Hawk Auth
  - GET OAuth1.0 (verify signature)

**GET Basic Auth**

**https://postman-echo.com/basic-auth**

This endpoint simulates a **basic-auth** protected endpoint. The endpoint accepts a default username and password and returns a status code of `200 ok` only if the same is provided. Otherwise it will return a status code `401 unauthorized`.

Username: `postman`

Password: `password`

**Example Request**

```
200
curl --location --request
```

**Example Response**

```
200 — OK
{
  "authenticated": true
}
```

# Autenticación básica

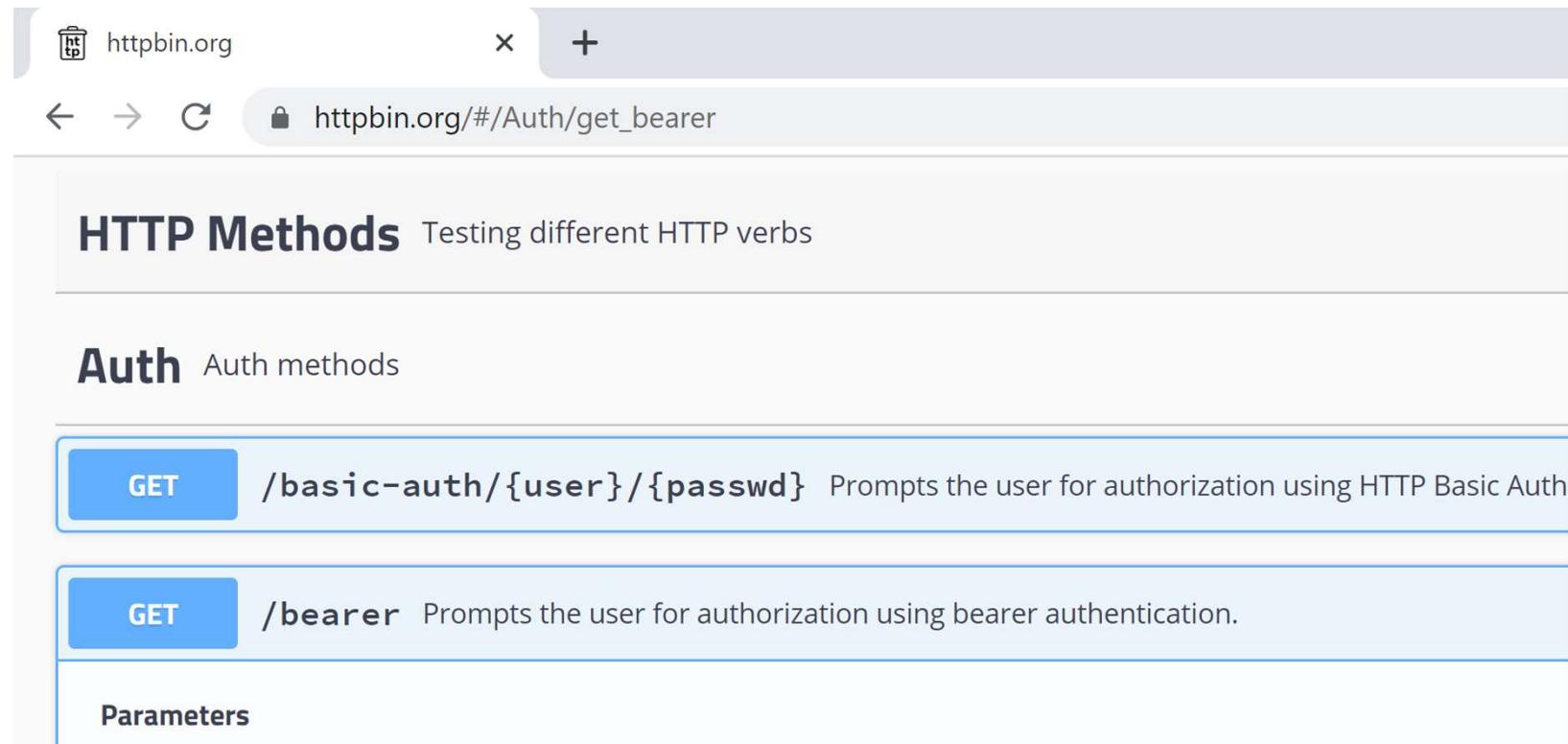
- Probamos la autenticación básica en una nueva petición, dentro de una nueva carpeta:

The screenshot shows the Postman application interface. On the left, the sidebar displays collections like 'Collection1' and 'Postman Echo'. A red circle highlights the 'Collections' tab. In the main area, a request titled 'GET Autorización Básica' is selected. A red circle highlights the 'Authorization' tab under the request type dropdown. Another red circle highlights the 'Send' button. The request URL is {{urlecho}}/basic-auth. The 'Authorization' tab shows 'Basic Auth' selected. A warning message says: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables'. The 'Username' field contains 'postman' and the 'Password' field contains 'password', with a checked 'Show Password' checkbox. The response status is 200 OK. At the bottom, the code editor shows the JSON response: { "authenticated": true }.



# 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:



httpbin.org

httpbin.org/#/Auth/get\_bearer

## HTTP Methods

Testing different HTTP verbs

## Auth

Auth methods

**GET** /basic-auth/{user}/{passwd} Prompts the user for authorization using HTTP Basic Auth.

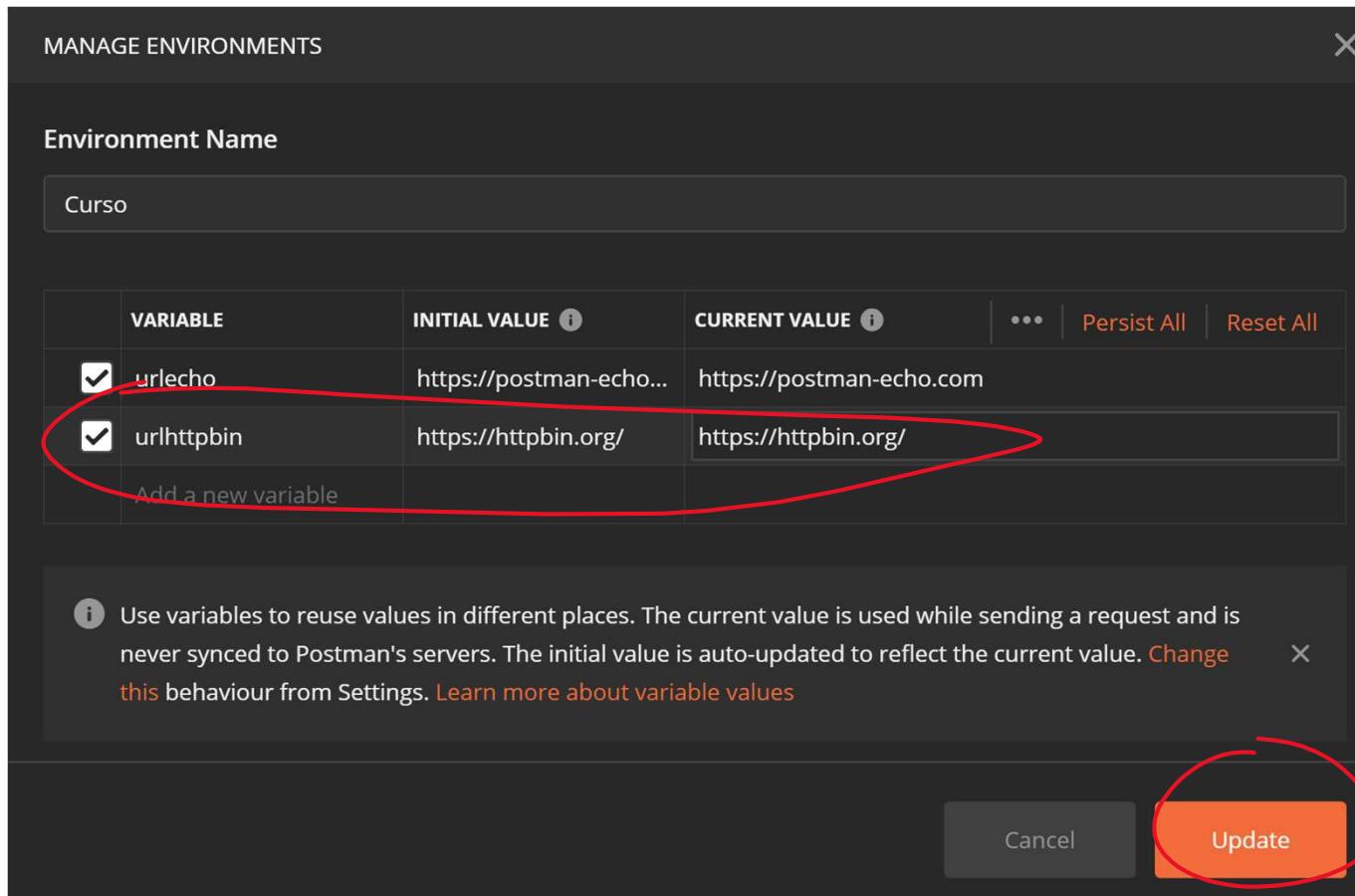
**GET** /bearer Prompts the user for authorization using bearer authentication.

### Parameters



# Autenticación por Token

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



MANAGE ENVIRONMENTS

Environment Name

Curso

VARIABLE	INITIAL VALUE	CURRENT VALUE	...	Persist All	Reset All
<input checked="" type="checkbox"/> urlecho	https://postman-echo...	https://postman-echo.com			
<input checked="" type="checkbox"/> urlhttpbin	https://httpbin.org/	https://httpbin.org/			

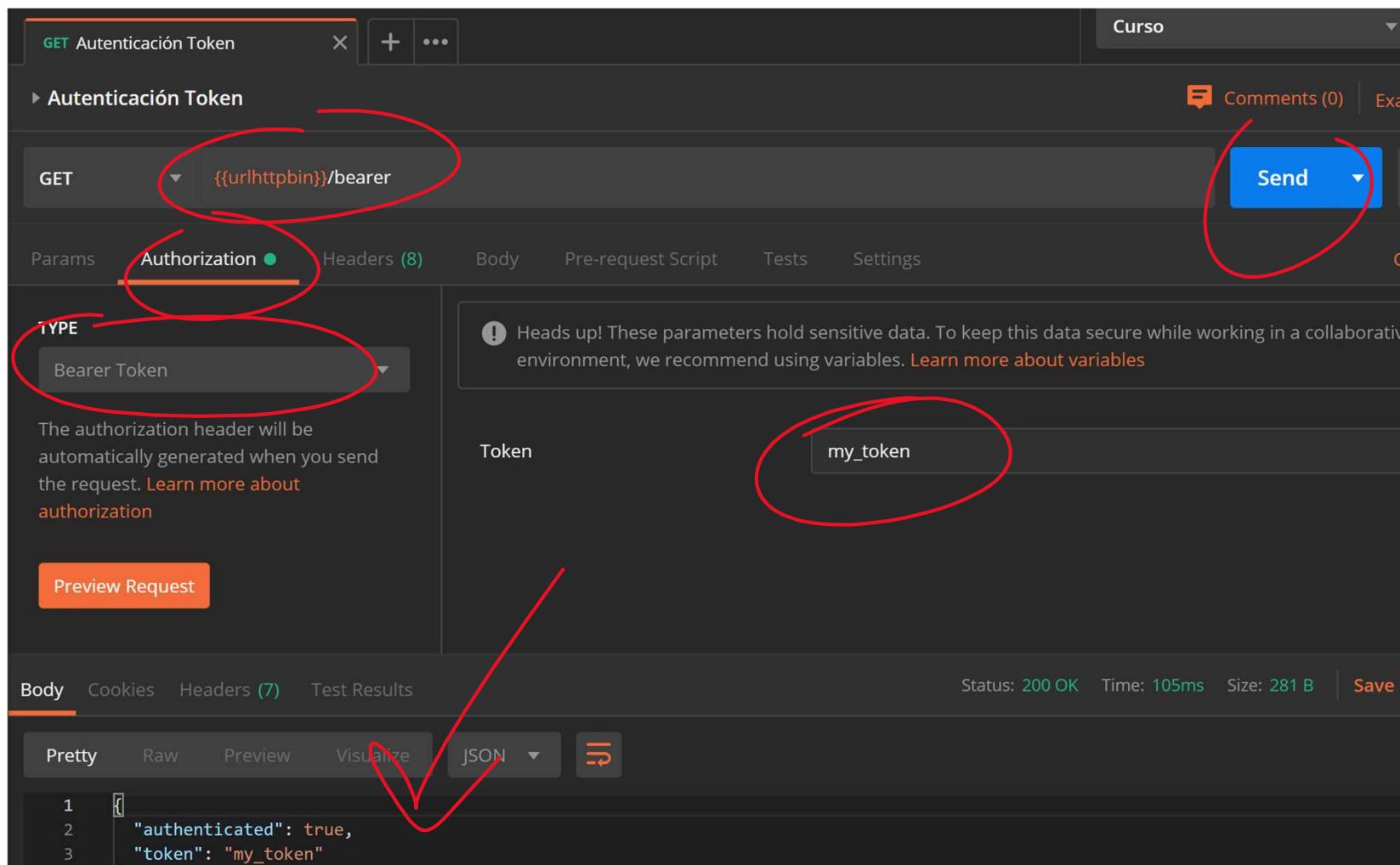
Add a new variable

*i* Use variables to reuse values in different places. The current value is used while sending a request and is never synced to Postman's servers. The initial value is auto-updated to reflect the current value. [Change this behaviour from Settings.](#) [Learn more about variable values](#)

Cancel Update

# Autenticación por Token

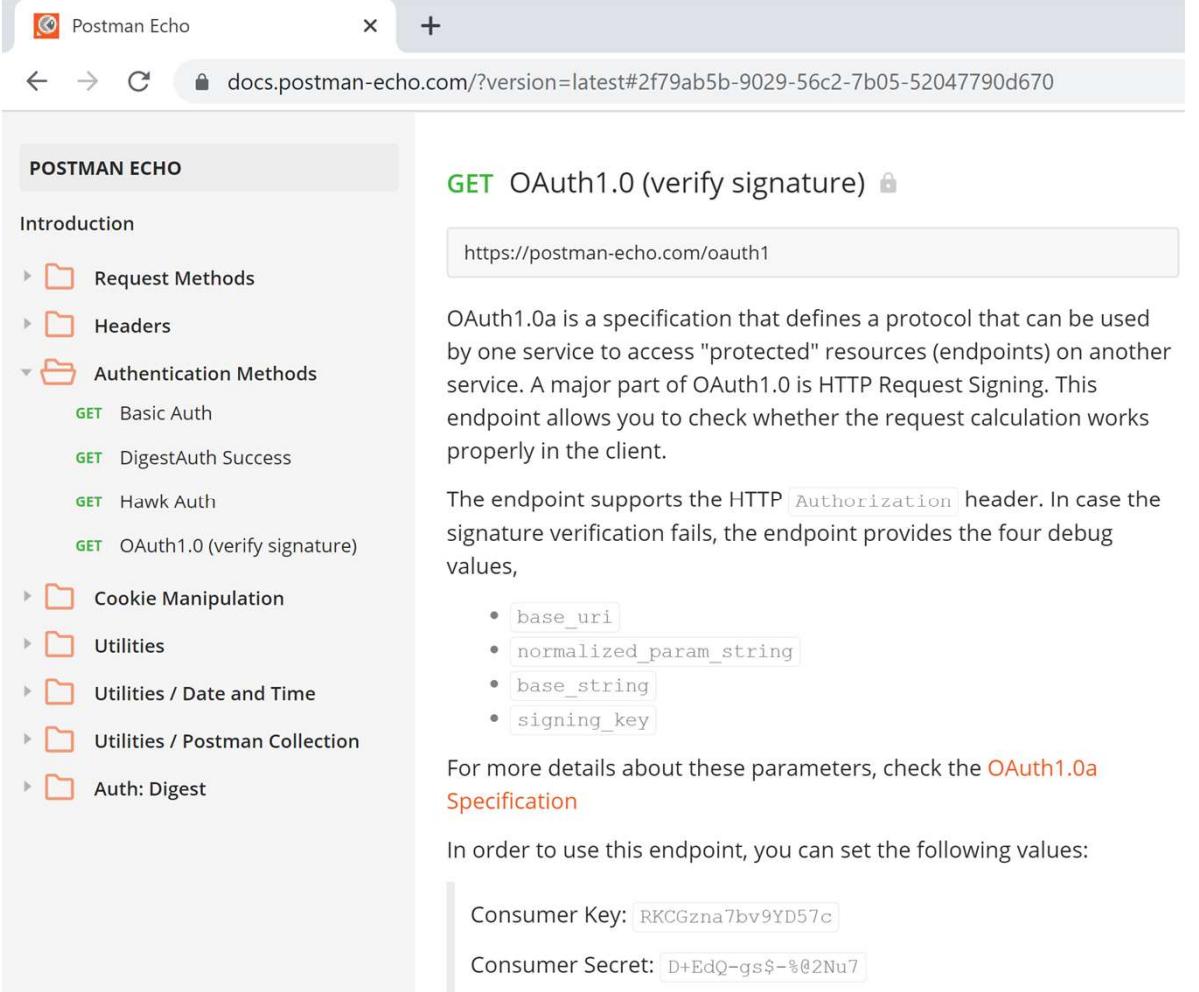
- Creamos la nueva petición:





# Autenticación OAuth I

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



The screenshot shows a browser window for "Postman Echo" at the URL [docs.postman-echo.com/?version=latest#2f79ab5b-9029-56c2-7b05-52047790d670](https://docs.postman-echo.com/?version=latest#2f79ab5b-9029-56c2-7b05-52047790d670). The left sidebar lists various API endpoints under "POSTMAN ECHO". The main content area shows the "GET OAuth1.0 (verify signature)" endpoint. It includes a description of OAuth1.0a as a specification for protocol verification, the supported HTTP header, and four debug values: base\_uri, normalized\_param\_string, base\_string, and signing\_key. It also provides links to the OAuth1.0a Specification and instructions for setting consumer key and secret.

POSTMAN ECHO

Introduction

- Request Methods
- Headers
- Authentication Methods
  - Basic Auth
  - DigestAuth Success
  - Hawk Auth
  - OAuth1.0 (verify signature)
- Cookie Manipulation
- Utilities
- Utilities / Date and Time
- Utilities / Postman Collection
- Auth: Digest

GET OAuth1.0 (verify signature) 🔒

https://postman-echo.com/oauth1

OAuth1.0a is a specification that defines a protocol that can be used by one service to access "protected" resources (endpoints) on another service. A major part of OAuth1.0 is HTTP Request Signing. This endpoint allows you to check whether the request calculation works properly in the client.

The endpoint supports the HTTP `Authorization` header. In case the signature verification fails, the endpoint provides the four debug values,

- base\_uri
- normalized\_param\_string
- base\_string
- signing\_key

For more details about these parameters, check the [OAuth1.0a Specification](#)

In order to use this endpoint, you can set the following values:

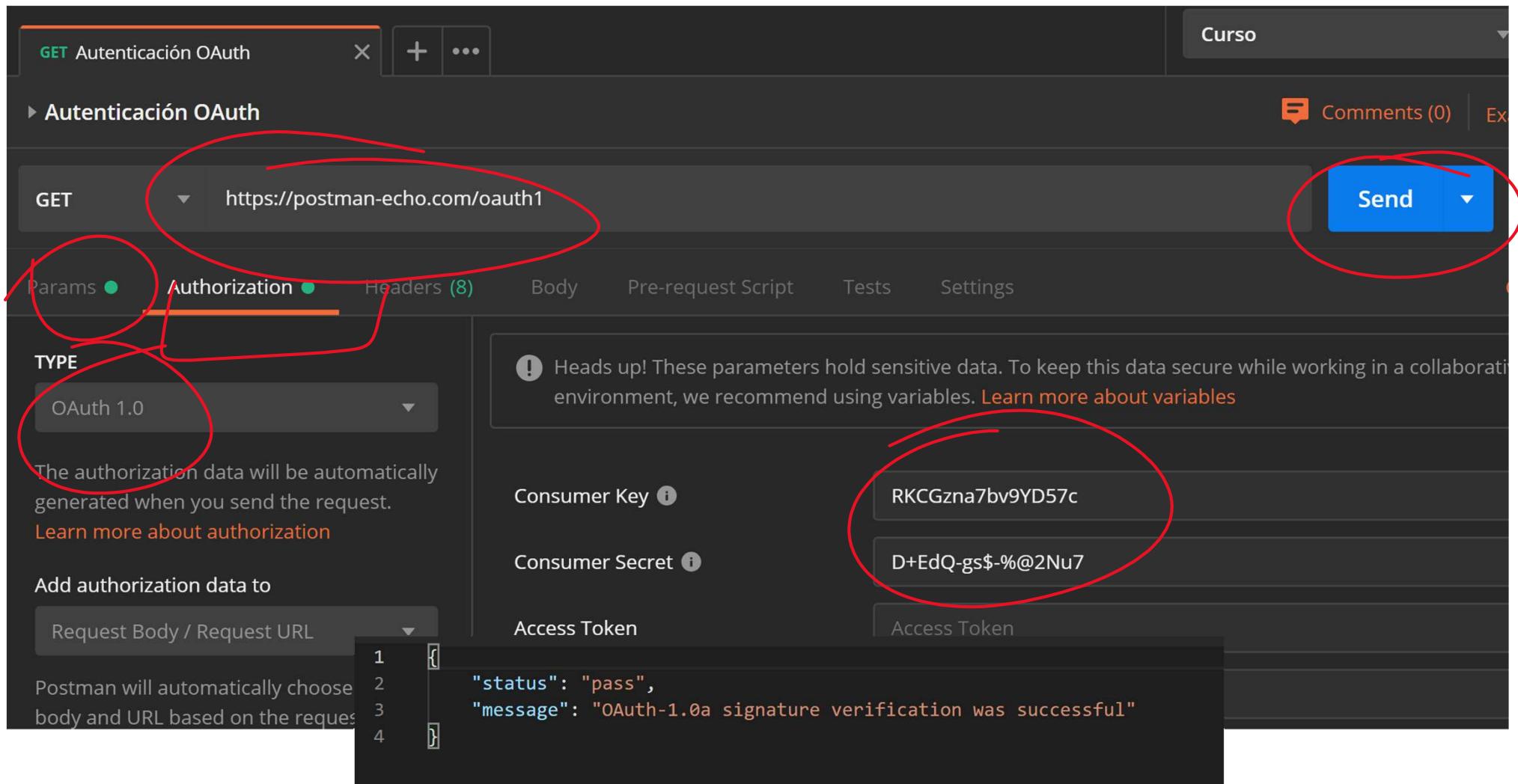
Consumer Key: `RKCGzna7bv9YD57c`

Consumer Secret: `D+EdQ-gs$-%@2Nu7`



# Autenticación OAuth I

- Creamos una nueva petición:



The screenshot shows the Postman application interface with the following details:

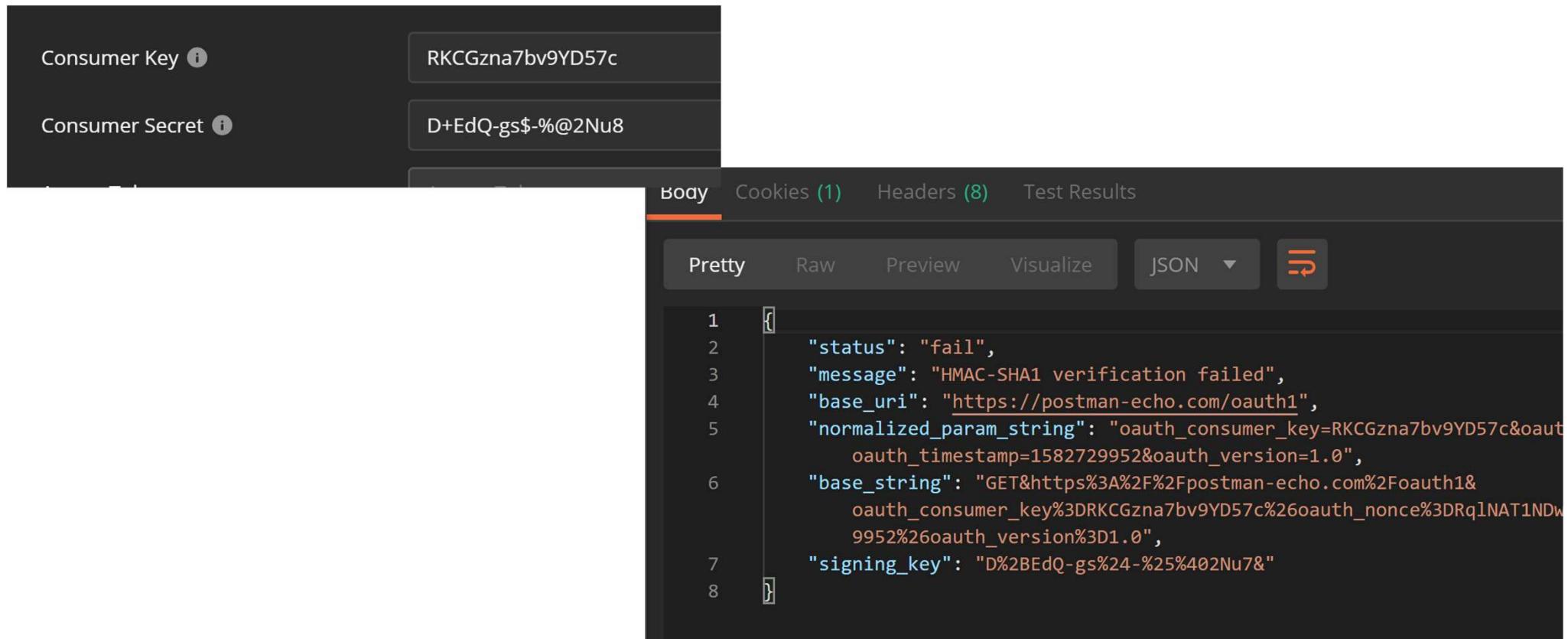
- Request Method:** GET
- URL:** https://postman-echo.com/oauth1
- Authorization Type:** OAuth 1.0
- Params:** A red circle highlights the "Params" section.
- Headers:** A red circle highlights the "Headers (8)" section.
- Send Button:** A large red circle highlights the blue "Send" button in the top right corner.
- OAuth Configuration:** The "Consumer Key" field contains "RKCGzna7bv9YD57c" and the "Consumer Secret" field contains "D+EdQ-gs\$-%@2Nu7". Both fields are highlighted with a red circle.
- Body:** The "Request Body / Request URL" dropdown is set to "Request Body". The body content is:

```
1 {"status": "pass",  
2 "message": "OAuth-1.0a signature verification was successful"}  
3  
4
```
- Comments:** There are 0 comments.
- Tests:** There are no tests.
- Pre-request Script:** There is no pre-request script.
- Settings:** There are no settings.



# Autenticación OAuth I

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



The screenshot shows the Postman application interface. At the top, there are fields for 'Consumer Key' (containing 'RKCGzna7bv9YD57c') and 'Consumer Secret' (containing 'D+EdQ-gs\$-%@2Nu8'). Below these fields is a toolbar with tabs: Body, Cookies (1), Headers (8), and Test Results. The 'Body' tab is selected and contains sub-tabs: Pretty, Raw, Preview, Visualize, and JSON. The JSON tab is also selected. The main content area displays a JSON response object with the following content:

```
1  {
2      "status": "fail",
3      "message": "HMAC-SHA1 verification failed",
4      "base_uri": "https://postman-echo.com/oauth1",
5      "normalized_param_string": "oauth_consumer_key=RKCGzna7bv9YD57c&oauth_timestamp=1582729952&oauth_version=1.0",
6      "base_string": "GET&https%3A%2Fpostman-echo.com%2Foauth1&oauth_consumer_key%3DRKCGzna7bv9YD57c%26oauth_nonce%3DRqlNAT1NDw9952%26oauth_version%3D1.0",
7      "signing_key": "D%2BEDQ-gs%24-%25%402Nu7&"
8  }
```

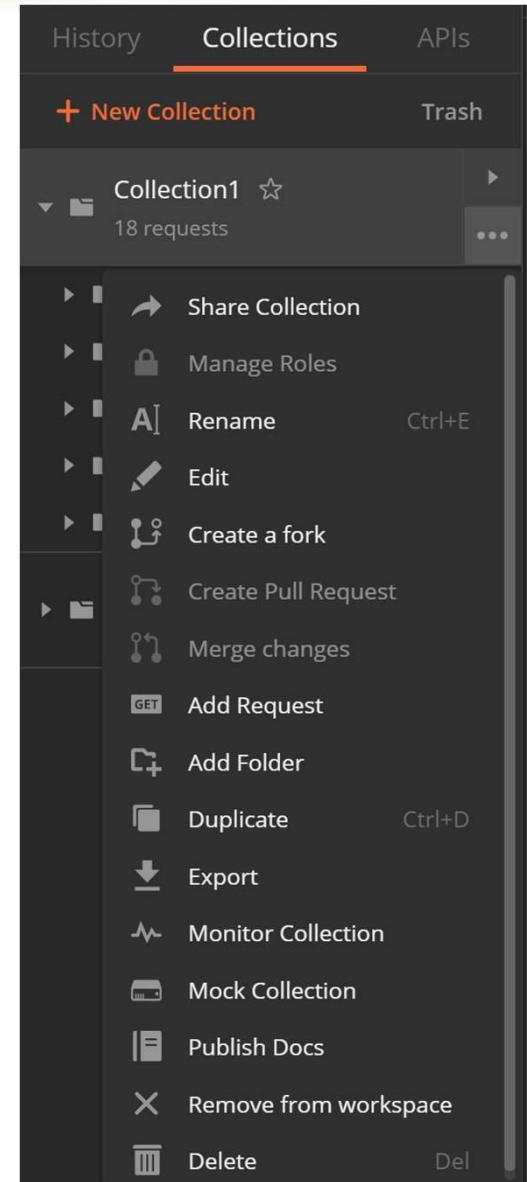
# Colecciones





# Operaciones sobre colecciones

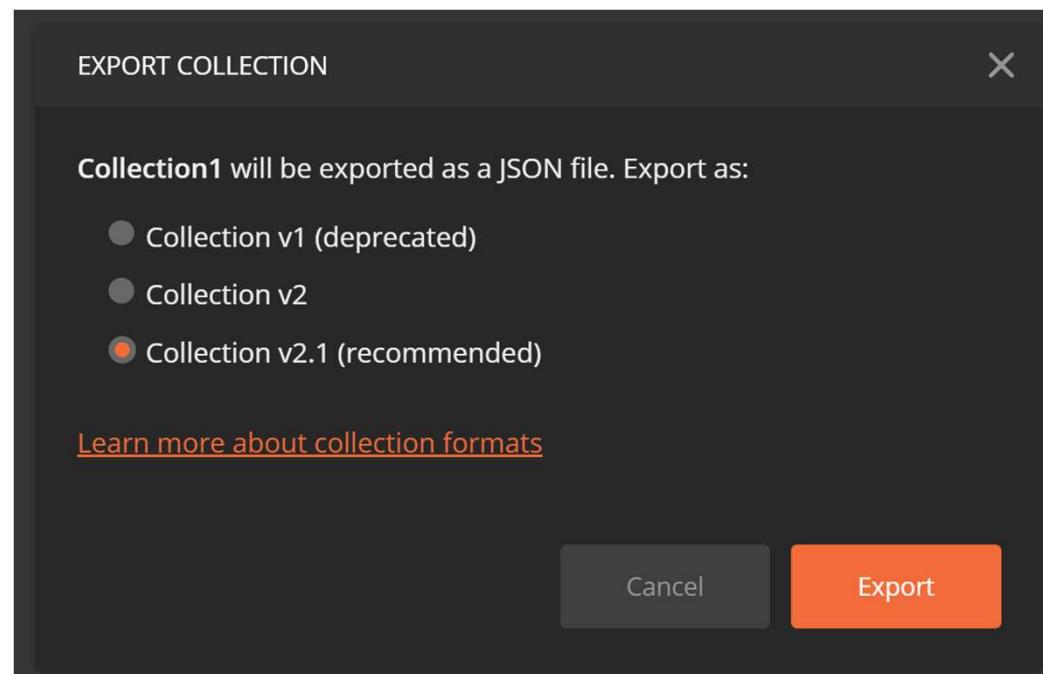
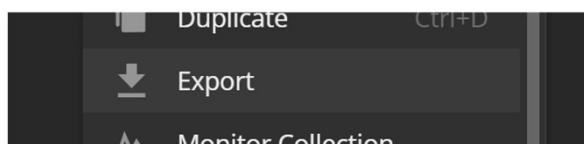
- Podemos realizar diferentes operaciones sobre las colecciones: renombrar, duplicar, exportar, borrar, etc.





# Exportar JSON

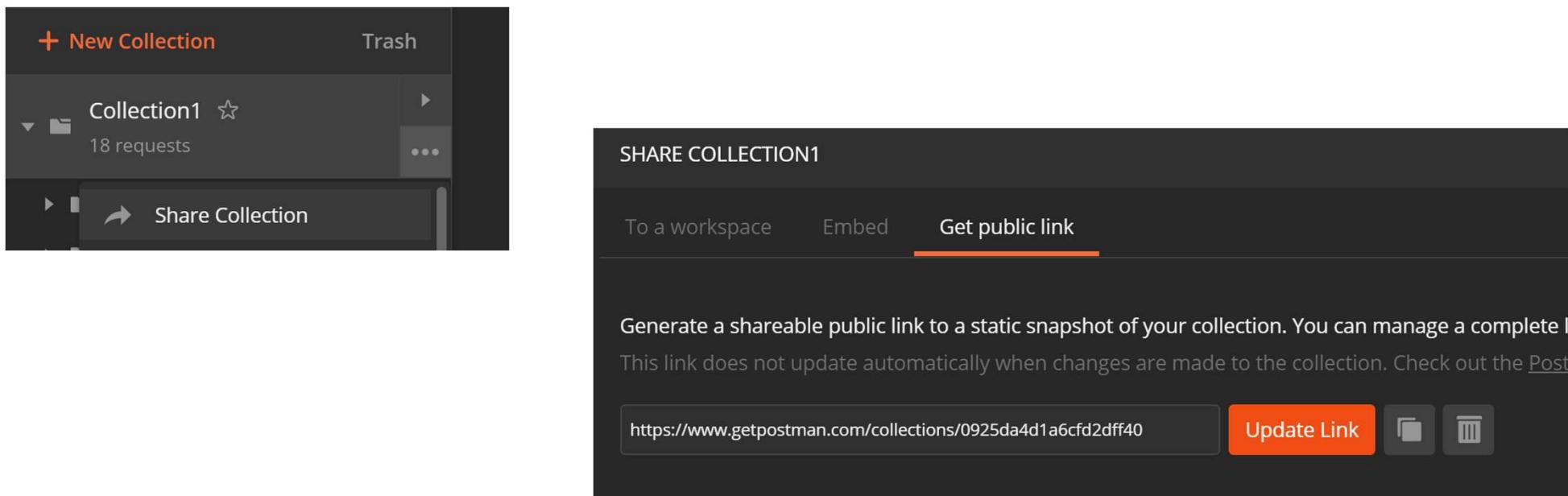
- Podemos exportar una colección como un JSON que posteriormente podríamos importar en otro espacio de trabajo:





# Exportar Link

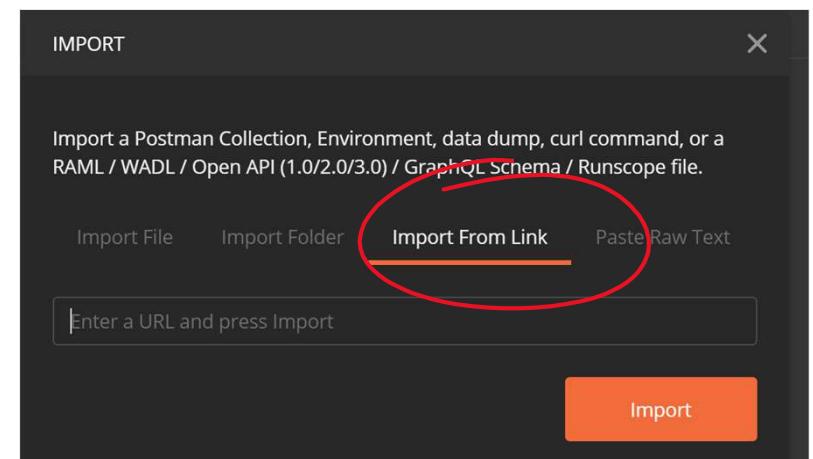
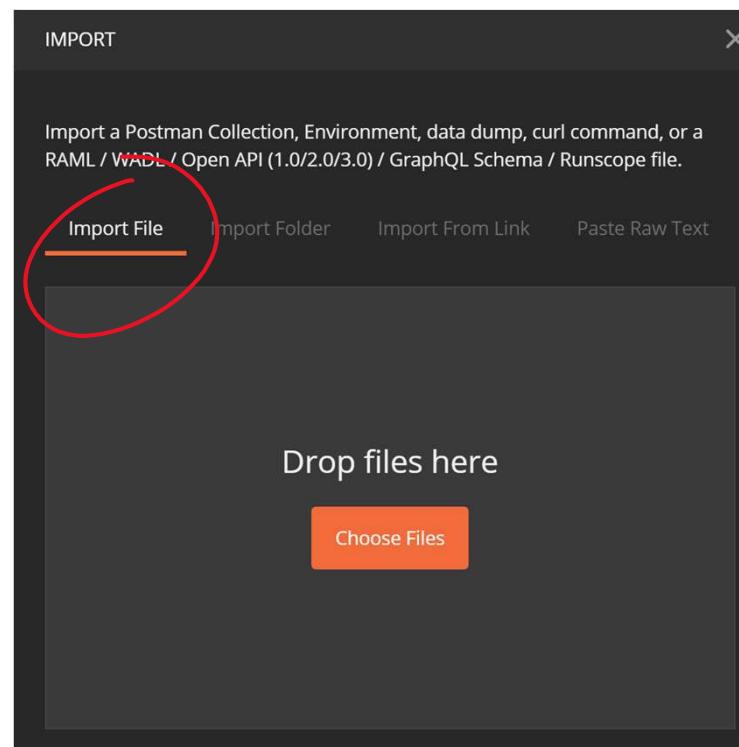
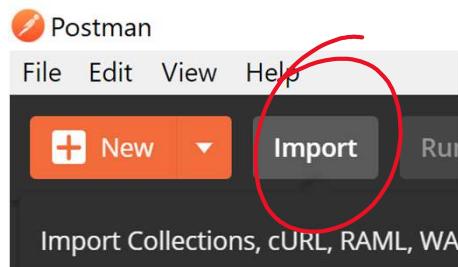
- Es posible compartir mi colección mediante un enlace público:



The image shows two screenshots of the Postman application. The left screenshot displays a list of collections, with 'Collection1' selected. A context menu is open over 'Collection1', showing options like 'Share Collection'. The right screenshot shows the 'SHARE COLLECTION1' dialog. The 'Get public link' tab is selected, highlighted by an orange underline. Below it, there is descriptive text about generating a shareable public link. At the bottom, there is a URL field containing the generated link: <https://www.getpostman.com/collections/0925da4d1a6cf2dff40>, an 'Update Link' button, and two icons for copying and deleting.

# 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:

The screenshot illustrates the process of viewing a collection in the Postman application and then navigating to its documentation page.

**Postman Application Interface:**

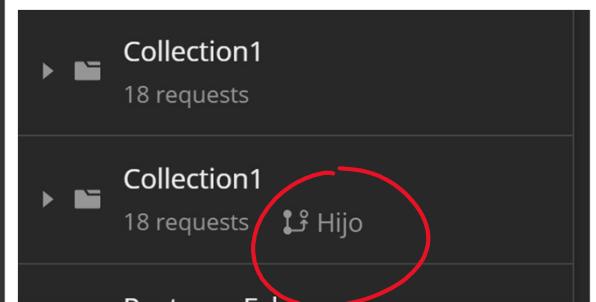
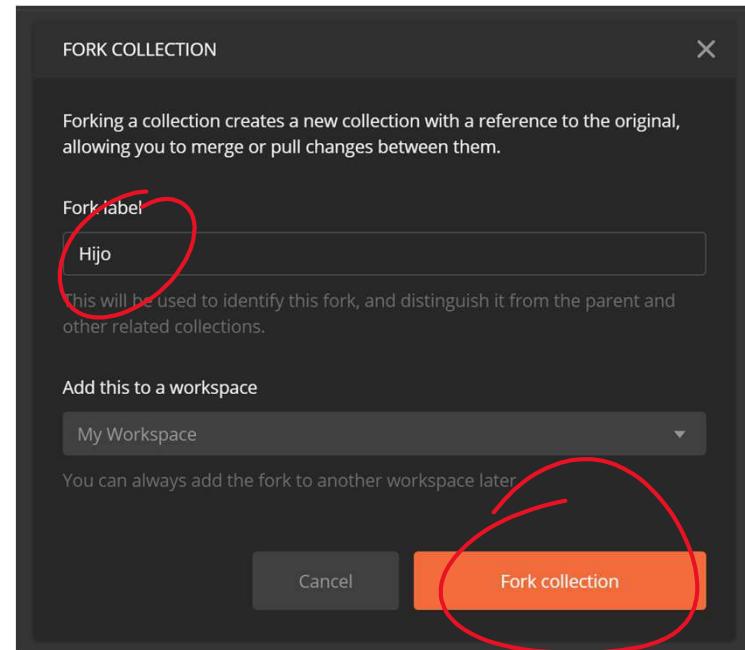
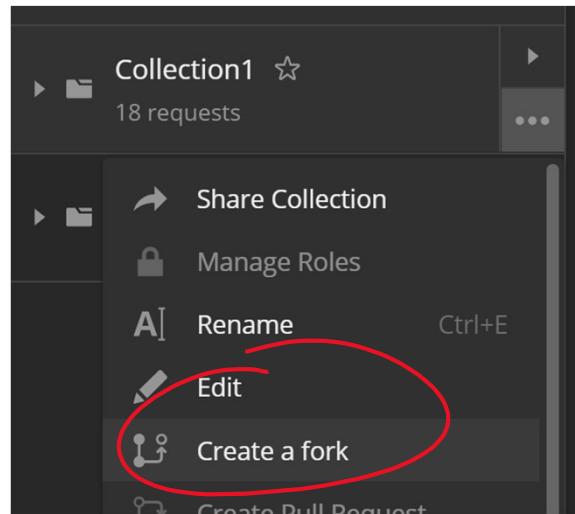
- Left Sidebar:** Shows a search bar labeled "Filter", a "History" tab, a "Collections" tab (which is active and highlighted in orange), a "New Collection" button, and a "Collection1" item with "18 requests".
- Central Area:** A modal window titled "Collection1" displays the collection details. It shows an "API" section stating "This collection is not linked to any API". Below are buttons for "Share", "Run", and "View in web". The "View in web" button is circled in red.
- Bottom Navigation:** Buttons for "Documentation", "Monitors", "Mocks", and "Changelog".

**Browser View:**

- Title Bar:** "Collection1 - Documentation" and the URL "web.postman.co/collections/10480298-7e550ca5-87a5-4667-a1e4-618315c0378f?version=latest&workspace=3ab5fa"
- Header:** Includes a profile icon, "My Workspace", "Invite", "Reports", "API Network", and "Templates".
- Content Area:** The main content area is titled "Collection1 DavidGranada". It features tabs for "Documentation", "Monitors", "Mocks", "Pull Requests", and "Changelog". The "Documentation" tab is active and highlighted in orange.
- Left Sidebar (Documentation Tab):** Lists "Introduction", "Servicios GET", "Servicios POST", "DELETE, UPDATE y PUT", "Variables Dinámicas", and "Autorizaciones".
- Right Sidebar (Documentation Tab):** Includes "Environment" (set to "No Environment"), "Layout" (set to "Double Column"), and a "Comments (0)" button, which is circled in red.
- Text Content:** The main body contains the text "Collection1" and "Esta es la colección de prueba para el curso".

# Control de versiones

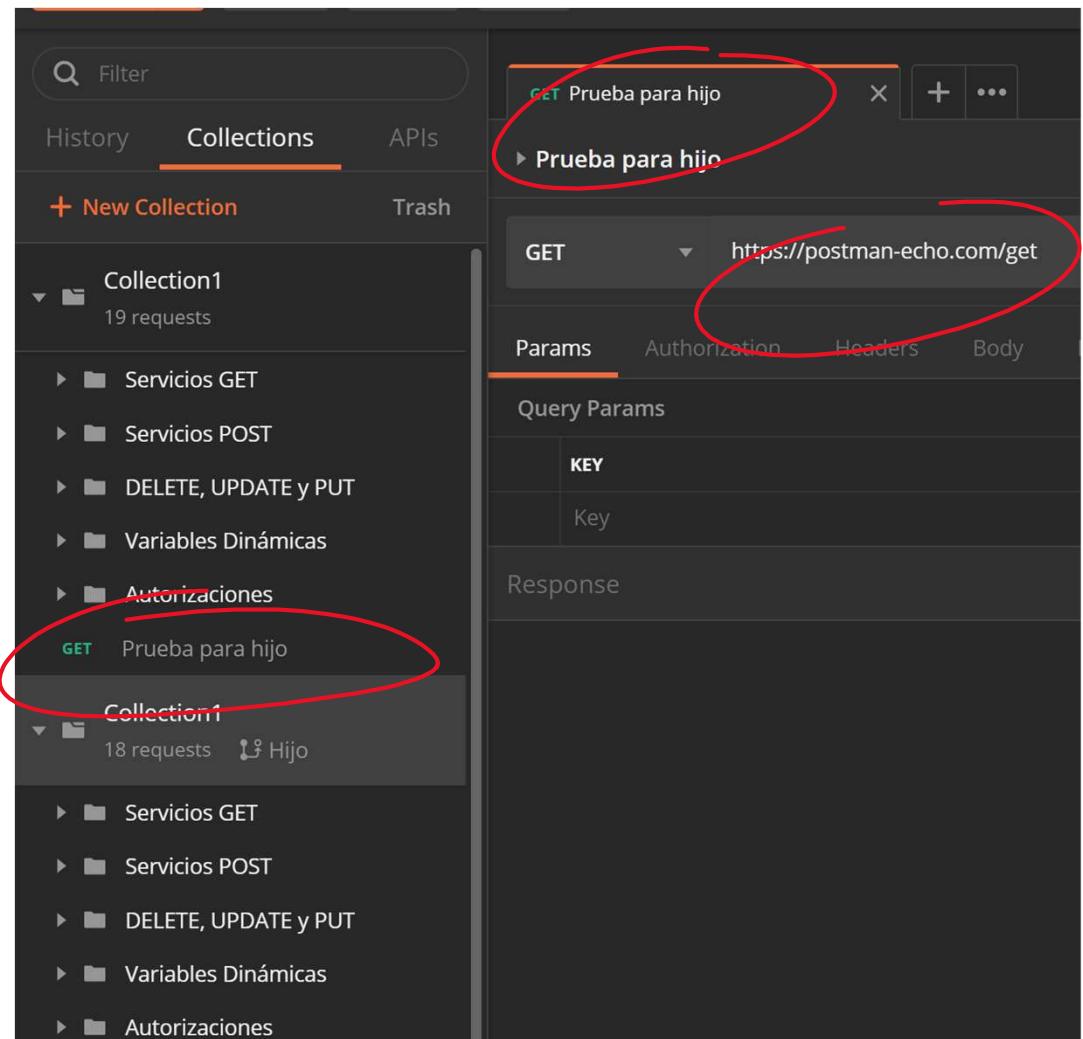
- Puedo crear forks de las colecciones (habitualmente para que alguien más trabaje independientemente sobre la colección y posteriormente se haga el respectivo merge):





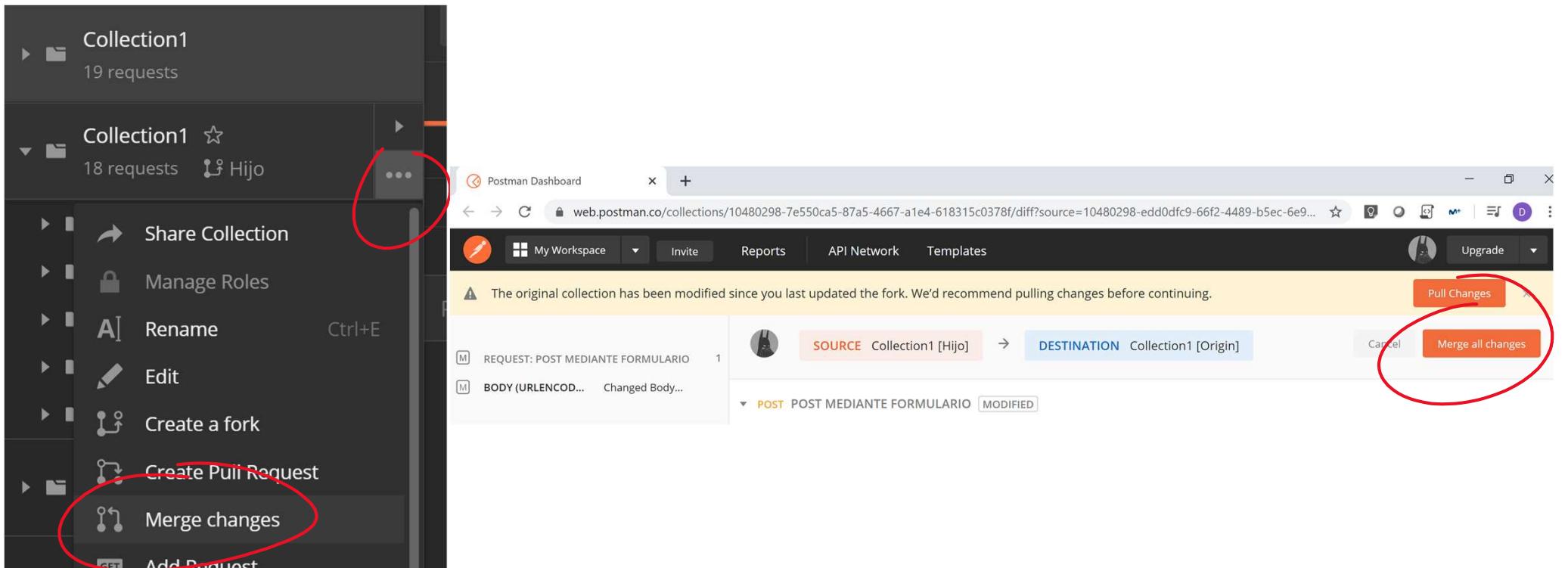
# Control de versiones

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



# Control de versiones

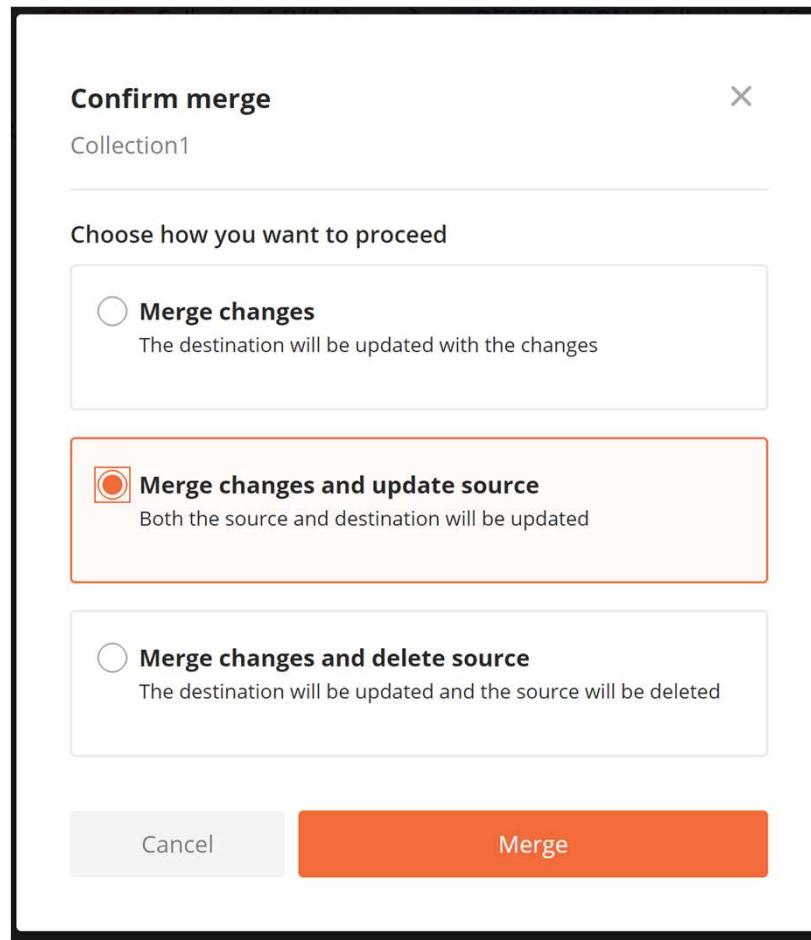
- Desde la colección hijo hacemos un merge changes, se nos abre una página web:





# Control de versiones

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



Collection1  
19 requests

▶ Servicios GET

▶ Servicios POST

▶ DELETE, UPDATE y PUT

▶ Variables Dinámicas

▶ Autorizaciones

GET Prueba para colección hijo

Collection1  
19 requests Hijo

▶ Servicios GET

▶ Servicios POST

▶ DELETE, UPDATE y PUT

▶ Variables Dinámicas

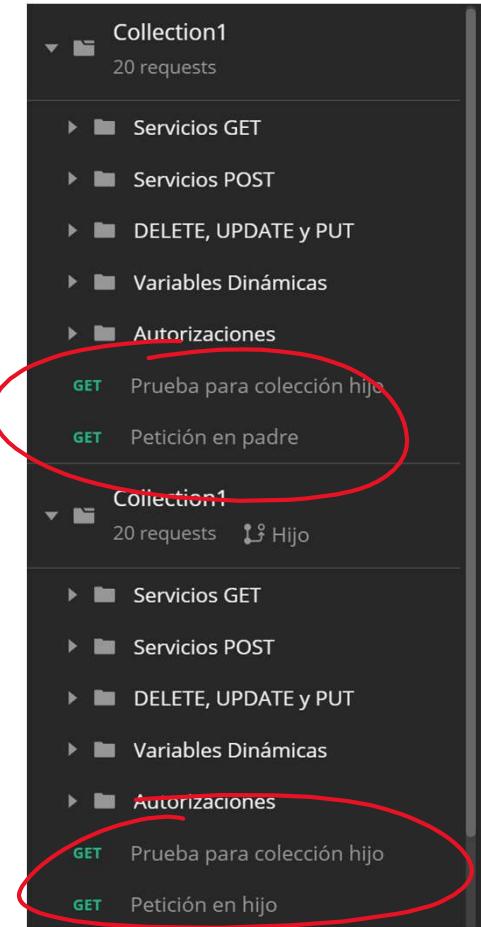
▶ Autorizaciones

GET Prueba para colección hijo



# 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:



The screenshot shows a MongoDB interface with two collections listed on the left:

- Collection1**: Contains 20 requests. It includes sub-folders for Services GET, Services POST, DELETE, UPDATE and PUT, Dynamic Variables, and Authorizations. Under Authorizations, there are two GET requests: "Prueba para colección hijo" and "Petición en padre". The first request is circled in red.
- Collection1\_Hijo**: Contains 20 requests. It includes sub-folders for Services GET, Services POST, DELETE, UPDATE and PUT, Dynamic Variables, and Authorizations. Under Authorizations, there are two GET requests: "Prueba para colección hijo" and "Petición en hijo". The second request is circled in red.

# Control de versiones

- Hacemos los pull:

The image shows two separate instances of the Postman application window side-by-side.

**Top Window (Left):** This window shows a comparison between "Collection1 [Hijo]" (Source) and "Collection1 [Origin]" (Destination). A yellow banner at the top states: "⚠ The original collection has been modified since you last updated the fork. We'd recommend pulling changes before continuing." A red circle highlights the "Pull Changes" button in the top right corner of the dialog.

REQUEST	NAME	METHOD
PETICIÓN EN HIJO	Name change...	Added method

**Bottom Window (Right):** This window shows a comparison between "Collection1 [Origin]" (Source) and "Collection1 [Hijo]" (Destination). A red circle highlights the "Pull changes" button in the top right corner of the dialog.

REQUEST	NAME	METHOD
PETICIÓN EN PADRE	Name change...	Added method

# Control de versiones

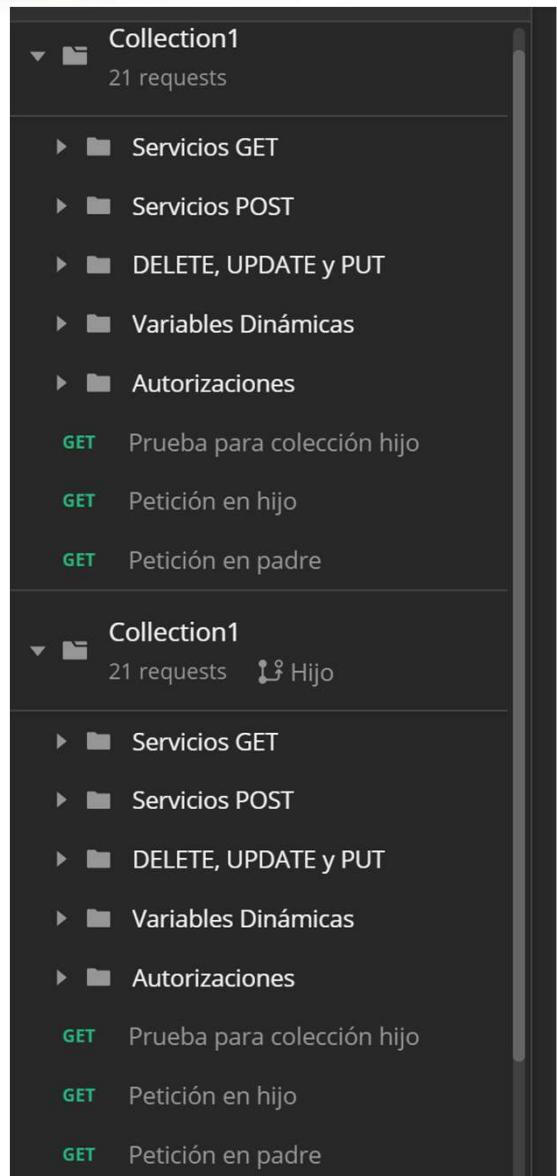
- Y finalmente el merge:

The screenshot shows the Postman Dashboard interface. At the top, there's a navigation bar with 'Postman Dashboard', a search bar containing 'web.postman.co/collections/10480298-7e550ca5-87a5-4667-a1e4-618315c0378f/diff?source=10480298-edd0dfc9-66f2-4489-b5ec-6e9...', and various icons. Below the navigation is a toolbar with 'My Workspace', 'Invite', 'Reports', 'API Network', 'Templates', and 'Upgrade'. The main area displays a 'COLLECTION: COLLECTION1' view with two sections: 'SOURCE Collection1 [Hijo]' and 'DESTINATION Collection1 [Origin]'. A 'Merge all changes' button is visible. On the left, a sidebar lists changes: 'COLLECTION: COLLECTION1' (Reordered req...), 'REQUEST: PETICIÓN EN HIJO' (Name change...), and 'METHOD' (Added method). On the right, a 'Confirm merge' dialog box is open, asking 'Choose how you want to proceed' with three options: 'Merge changes' (destination updated), 'Merge changes and update source' (both source and destination updated, selected), and 'Merge changes and delete source' (destination updated and source deleted). Buttons for 'Cancel' and 'Merge' are at the bottom of the dialog.



# Control de versiones

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



# Utilidades

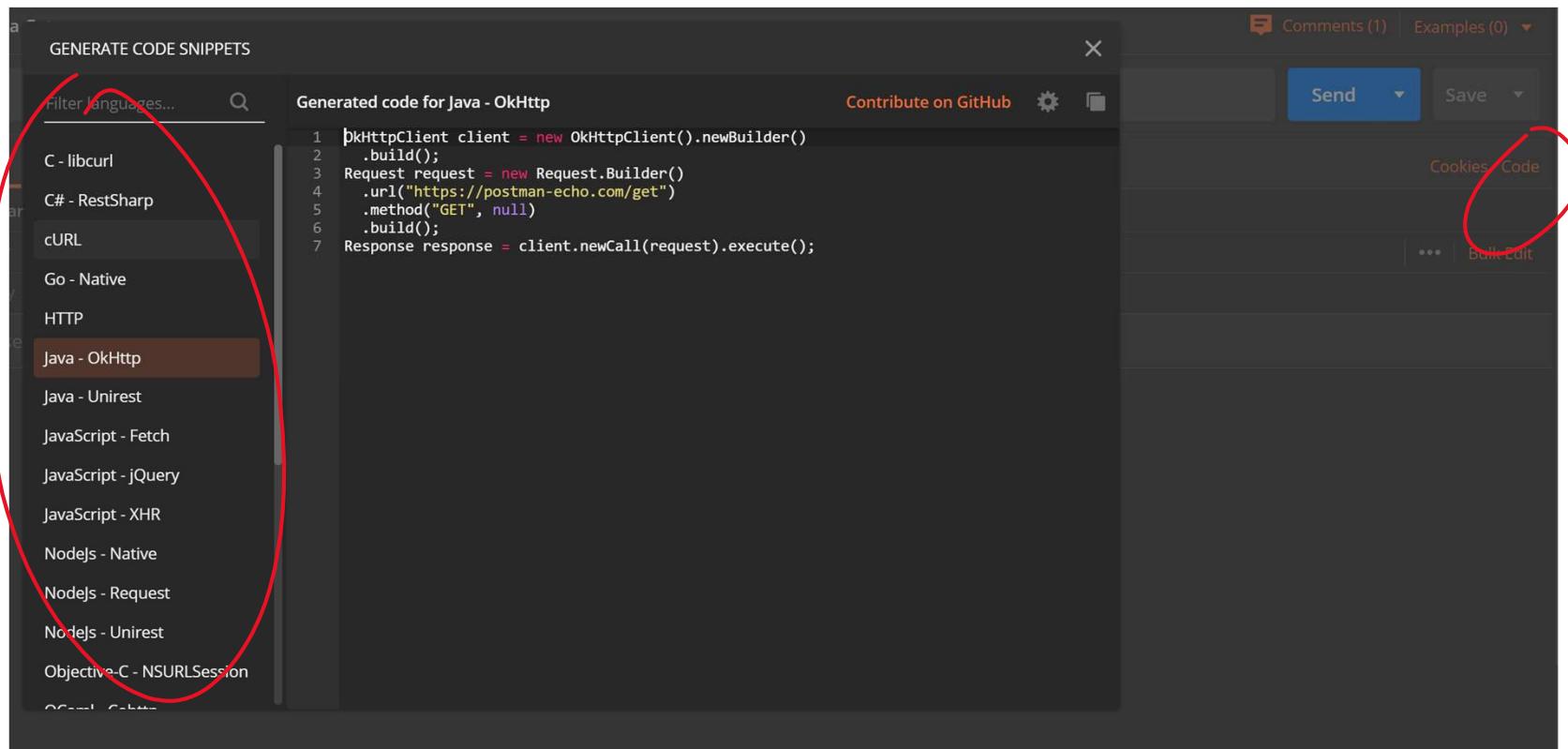


# Comentarios

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

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' selected, showing 'Collection1' and 'Servicios GET' with items like 'Prueba Get', 'Prueba Get Parámetros', and 'Ejemplo de ID'. The main area shows a 'Launchpad' tab and a selected 'GET Prueba Get' request. The request details show 'https://postman-echo.com/get'. Below the request, there are tabs for 'Params', 'Authorization', 'Headers', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. A 'Comments' section is open, containing a message 'Este es el servicio del GET básico' and a red-circled 'Add comment' button. The top right corner has a 'Comments (0)' indicator.

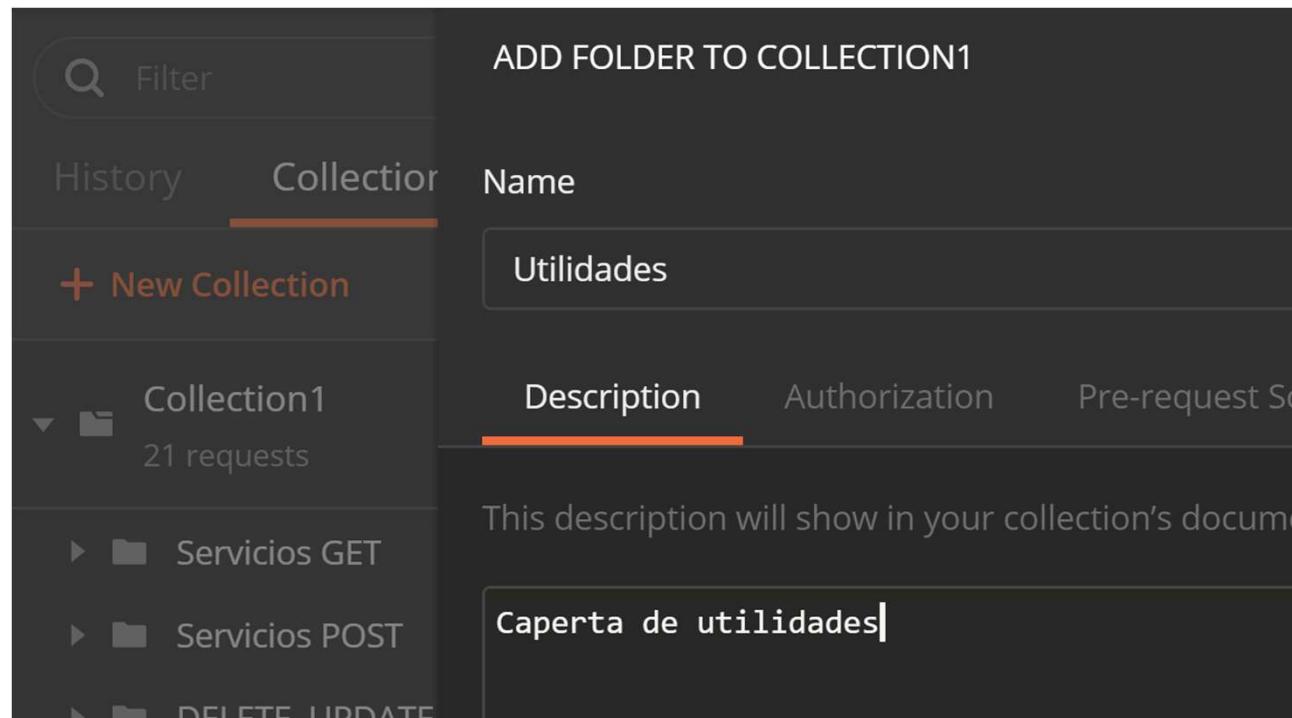
- Postman me genera el código del servicio web de forma automática y para muchos lenguajes de programación:



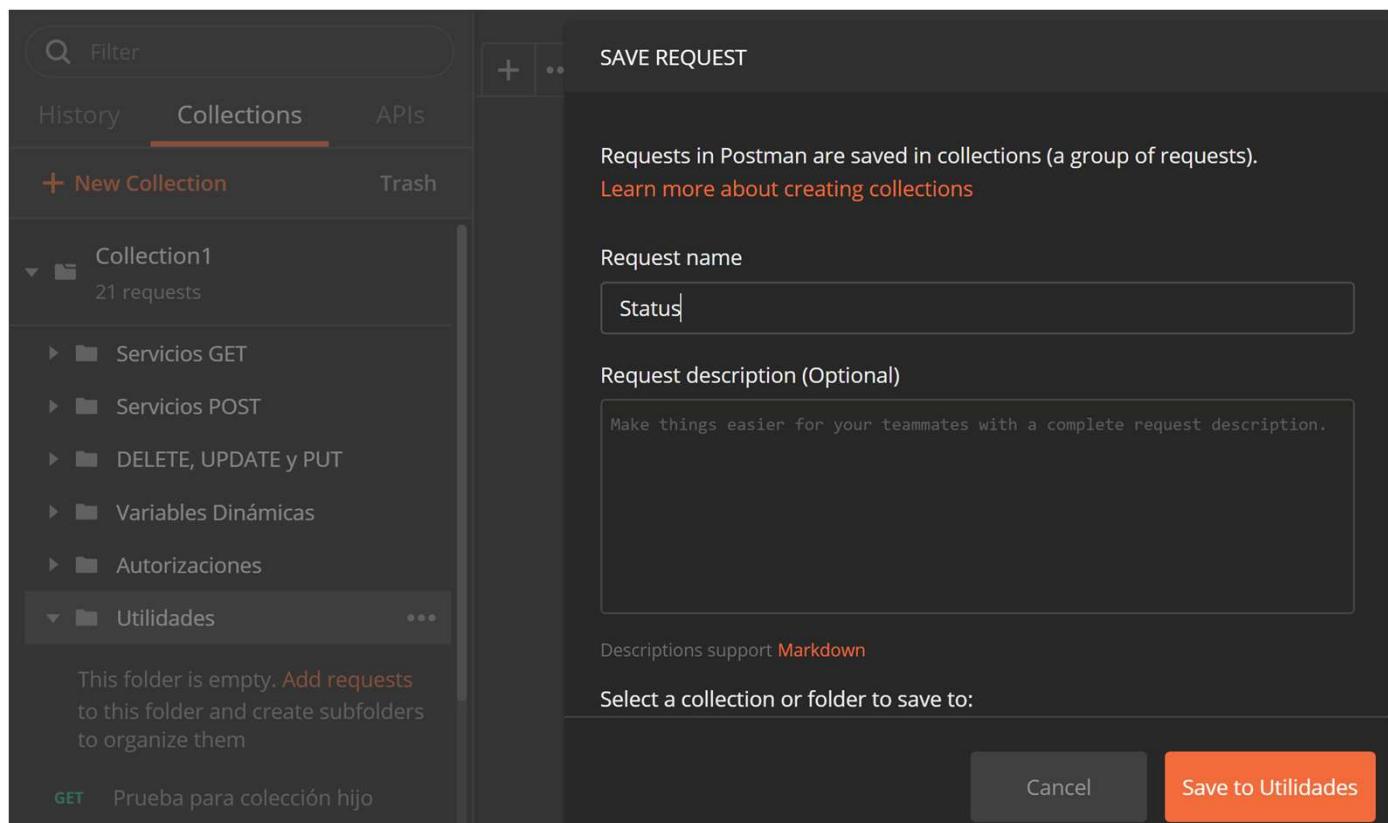


# Utilidades

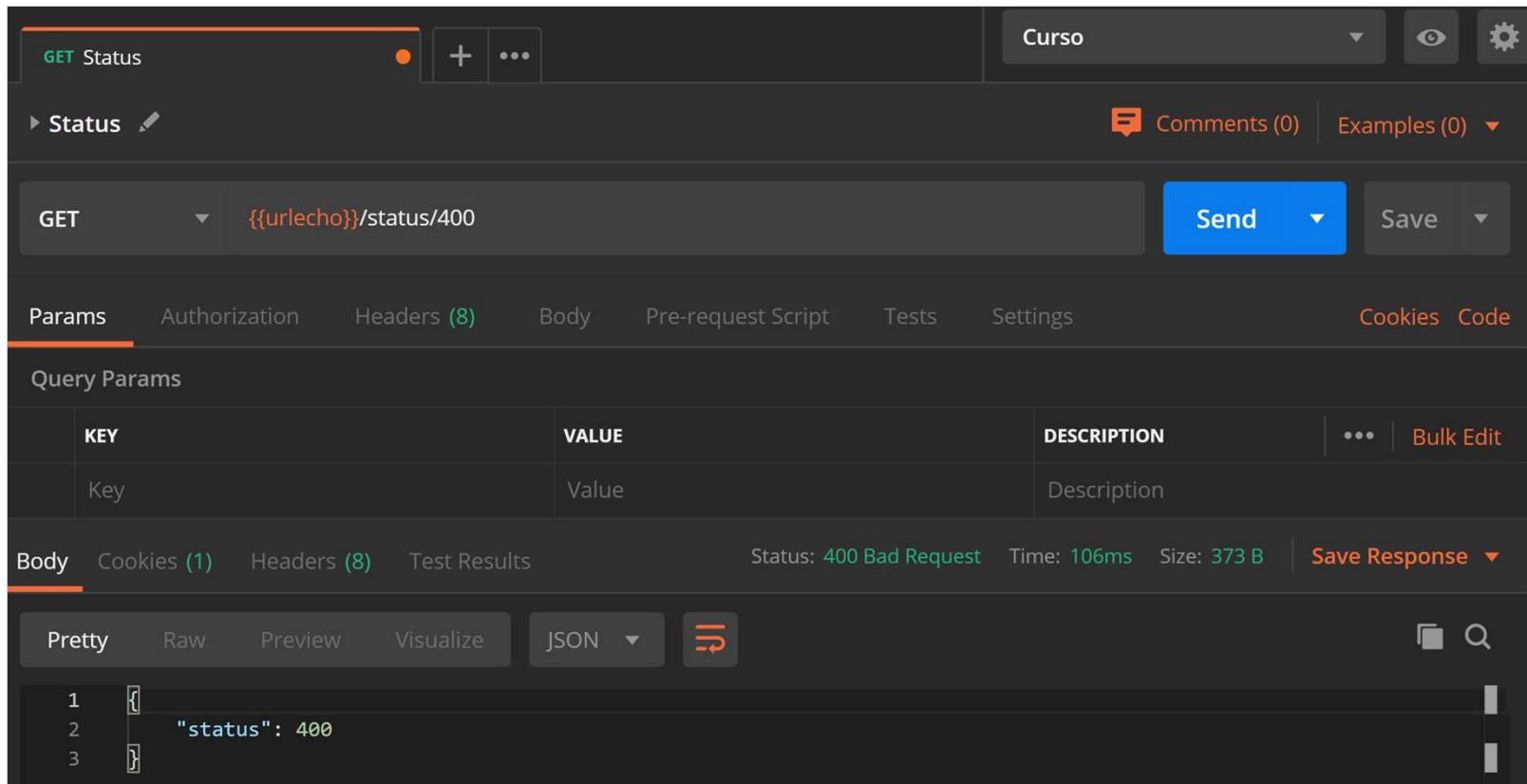
- Vamos a crear una nueva carpeta dentro de la colección:



- 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:



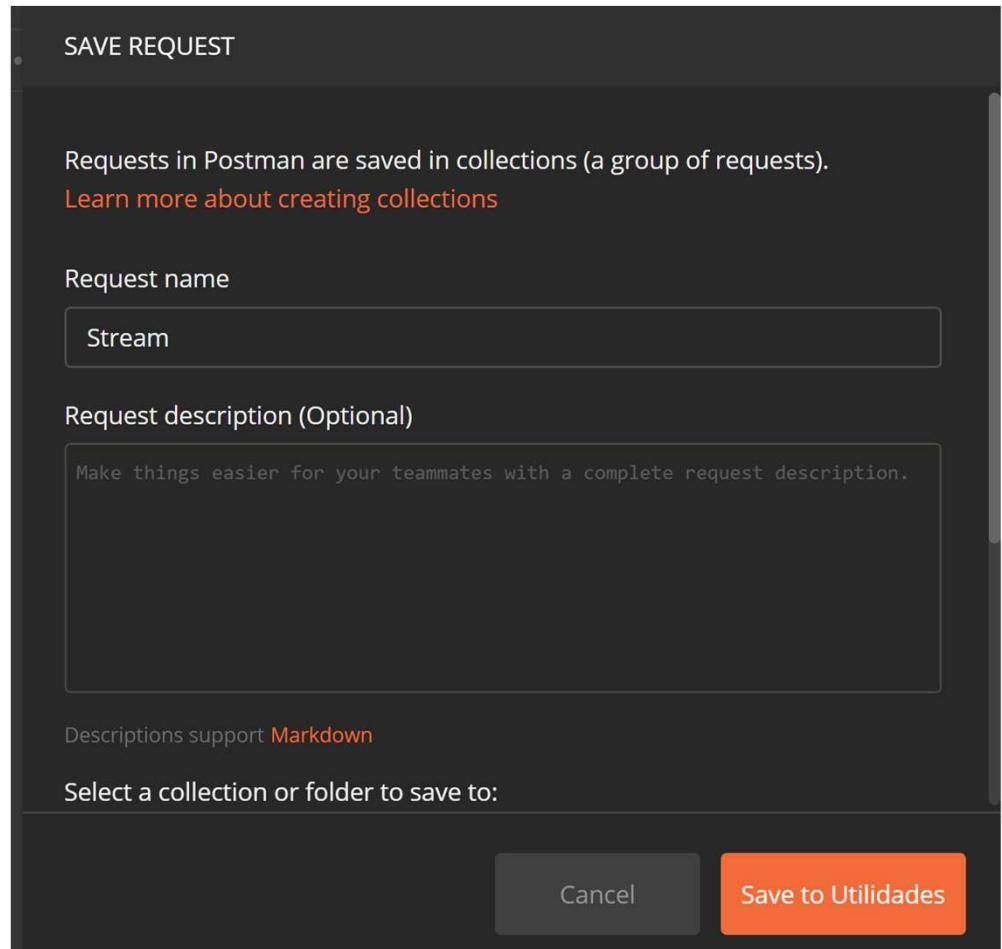
- Podemos utilizar la utilidad status y probar con los diferentes códigos de estado que podría devolver un servicio web: [https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)



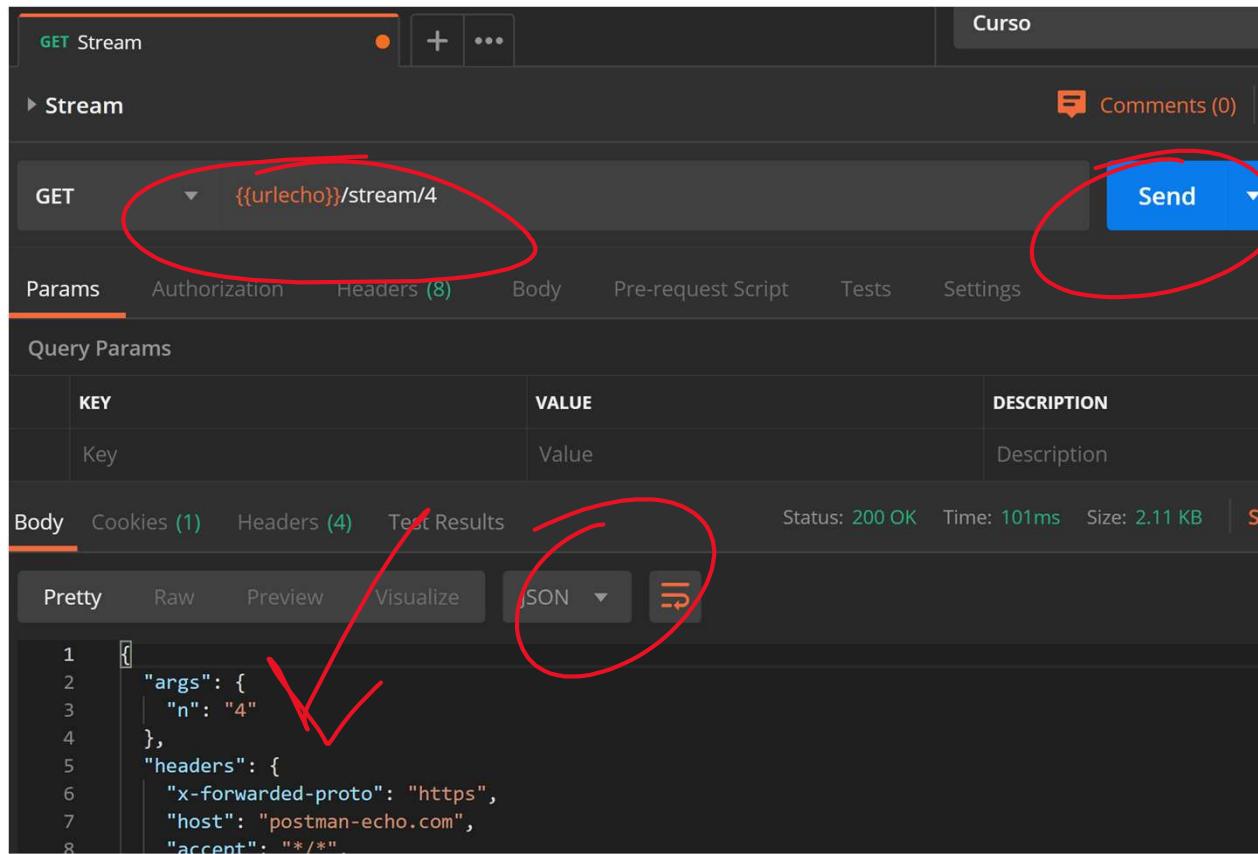
The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'GET Status' selected. Below it, a search bar contains the text 'Curso'. On the right side of the header, there are buttons for 'Comments (0)', 'Examples (0)', 'Send', and 'Save'. The main workspace shows a 'Status' tab selected under a 'Params' tab. The URL field contains 'GET {{urlecho}}/status/400'. The 'Body' tab is selected at the bottom, showing a JSON response with the following content:

```
1 {  
2     "status": 400  
3 }
```

- 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



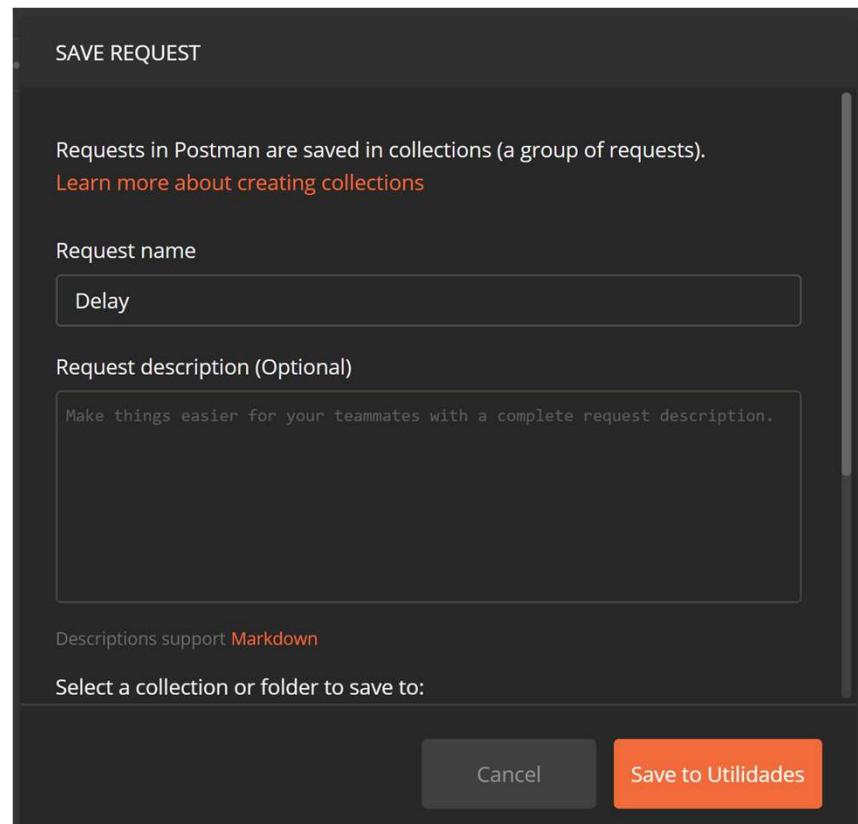
- Podemos modificar el número de divisiones:



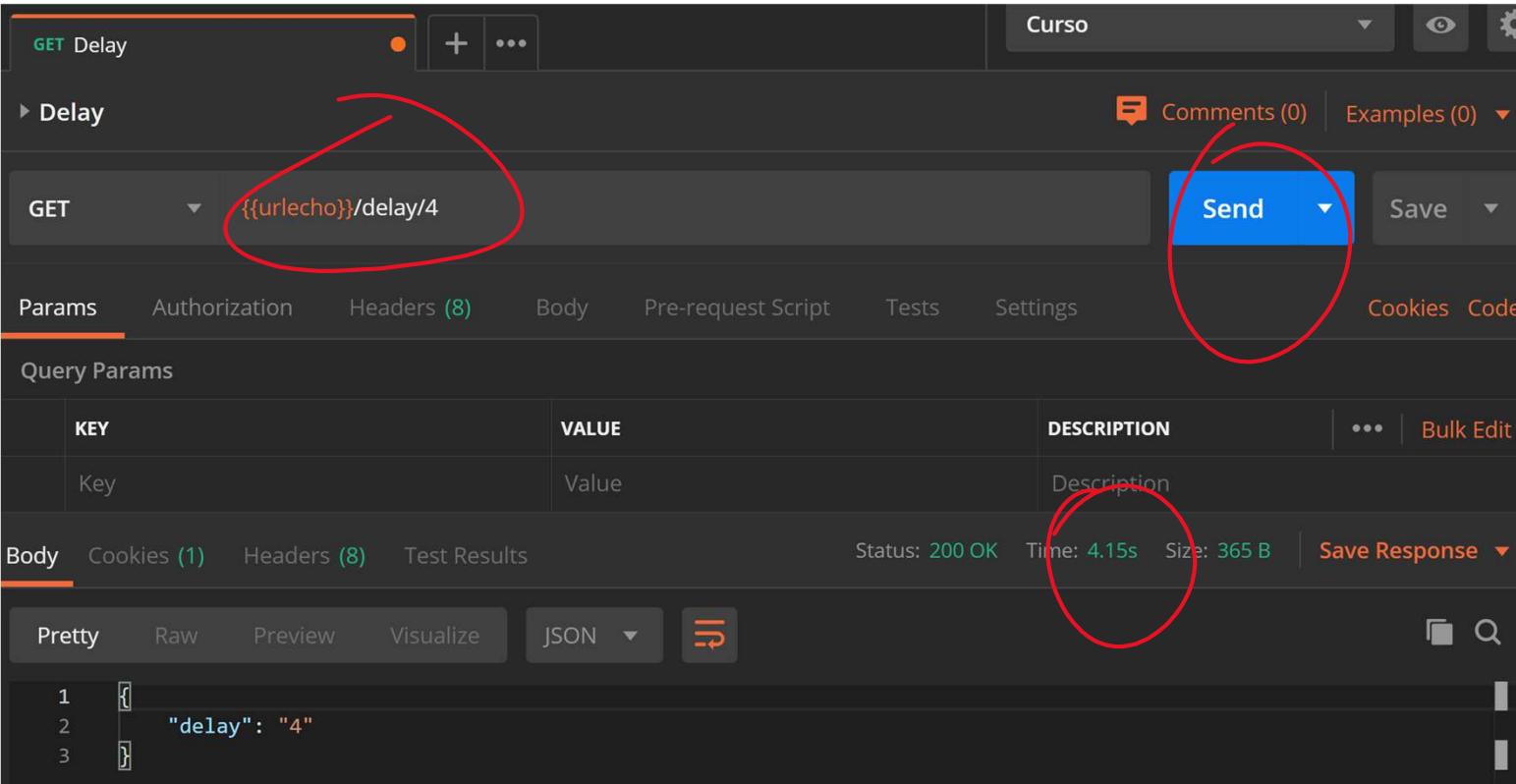
The screenshot shows the Postman application interface. At the top, there's a header with 'GET Stream' and a status indicator. Below it, the 'Stream' collection is selected. A red oval highlights the URL field containing the placeholder `{{urlecho}}/stream/4`. To the right of the URL is a blue 'Send' button, also highlighted with a red oval. Below the URL, tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible, with 'Params' being the active tab. Under 'Query Params', there's a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. The 'Body' tab is selected at the bottom, showing a JSON response. A red arrow points from the 'Send' button towards the JSON response area. The JSON response itself is as follows:

```
1 [ { "args": { "n": "4" }, "headers": { "x-forwarded-proto": "https", "host": "postman-echo.com", "accent": "*/*" } } ]
```

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



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



GET Delay

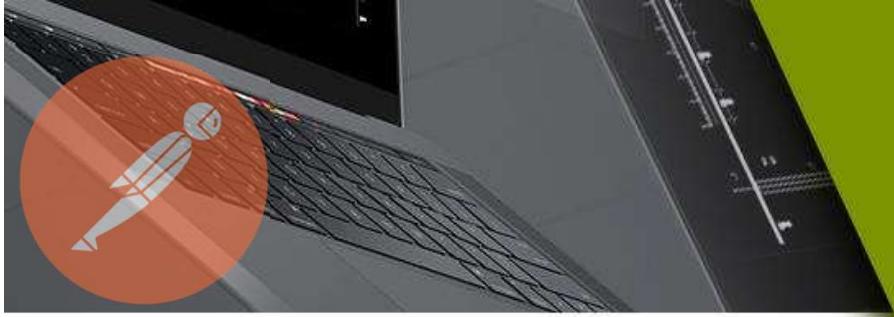
Delay

GET {{urlecho}}/delay/4

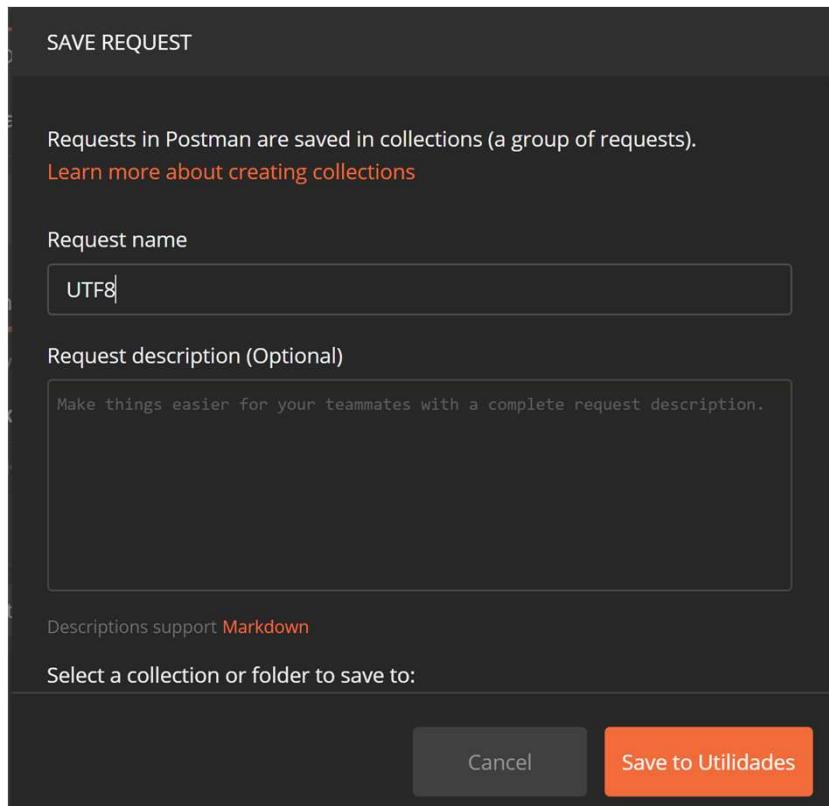
Send

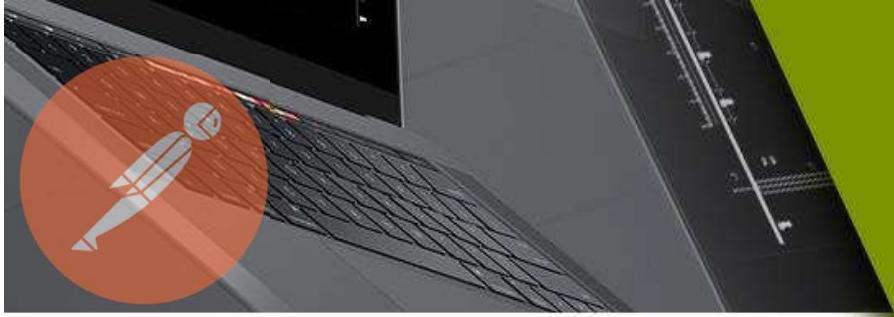
Status: 200 OK Time: 4.15s Size: 365 B

```
1 {  
2   "delay": "4"  
3 }
```



- 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:

GET UTF8

+

...

Curso

▶ UTF8

Comments (0)

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies (1) Headers (9) Test Results Status: 200 OK Time: 105ms Size: 6.42 KB Save

Pretty Raw Preview Visualize HTML

157 የዚህ በያዘው አንበሳ ያስረ::  
158 ስው እንደሆነ እንቂ እንደ ጥሩበቱ አይተካድርም::  
159 እግዚአብ የከፍተማን ጉዳይ ለያዘው አይደርም::  
160 የነገሱት ለበ:: በያዘት ይለቀ በያዘት ያጠልች::

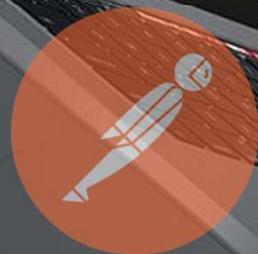


- Si queremos saber la IP de la máquina que está enviando la solicitud podemos probar la utilidad IP:

The screenshot shows the Postman interface with the following details:

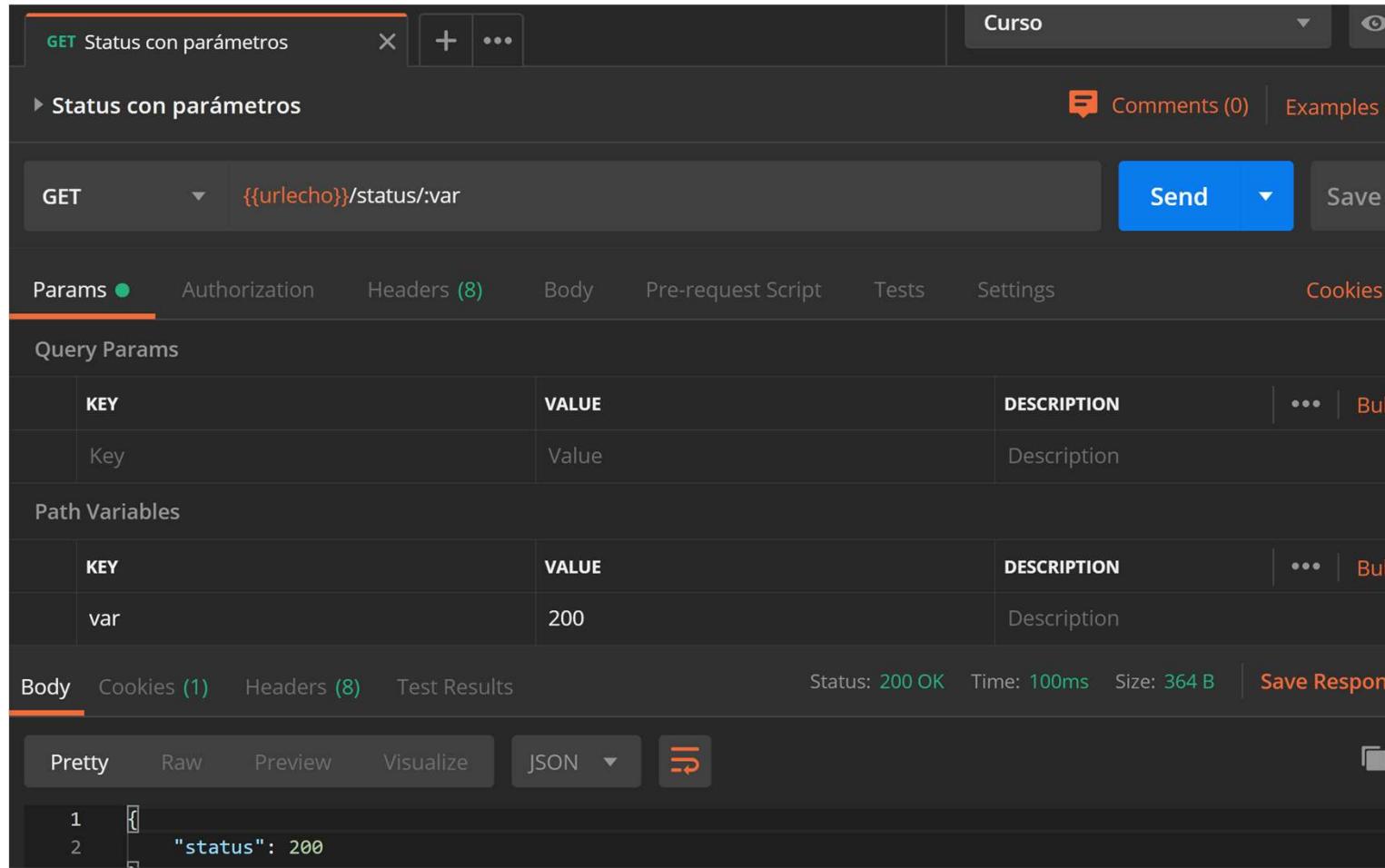
- Request URL:** {{urlecho}}/ip
- Method:** GET
- Headers:** (8)
- Body:** (raw, JSON, pretty, preview, visualize)
- Test Results:** Status: 200 OK, Time: 104ms, Size: 416 B
- Response Body (Pretty JSON):**

```
1 [ { "ip": "193.147.77.9" } ]
```



# Parámetros en llamada a servicios

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



The screenshot shows the Postman application interface. At the top, there's a header bar with the title "GET Status con parámetros" and a status indicator "Curso". Below the header, the main workspace displays a request configuration:

- Method:** GET
- URL:** {{urlecho}}/status/:var
- Send** button (blue)
- Params** tab (selected)
- Query Params** table:

KEY	VALUE	DESCRIPTION	...	Bul
Key	Value	Description		
- Path Variables** table:

KEY	VALUE	DESCRIPTION	...	Bul
var	200	Description		
- Body** tab (selected)
- JSON** dropdown (selected)
- Response Preview**:

```
1 [{}]
2 "status": 200
```

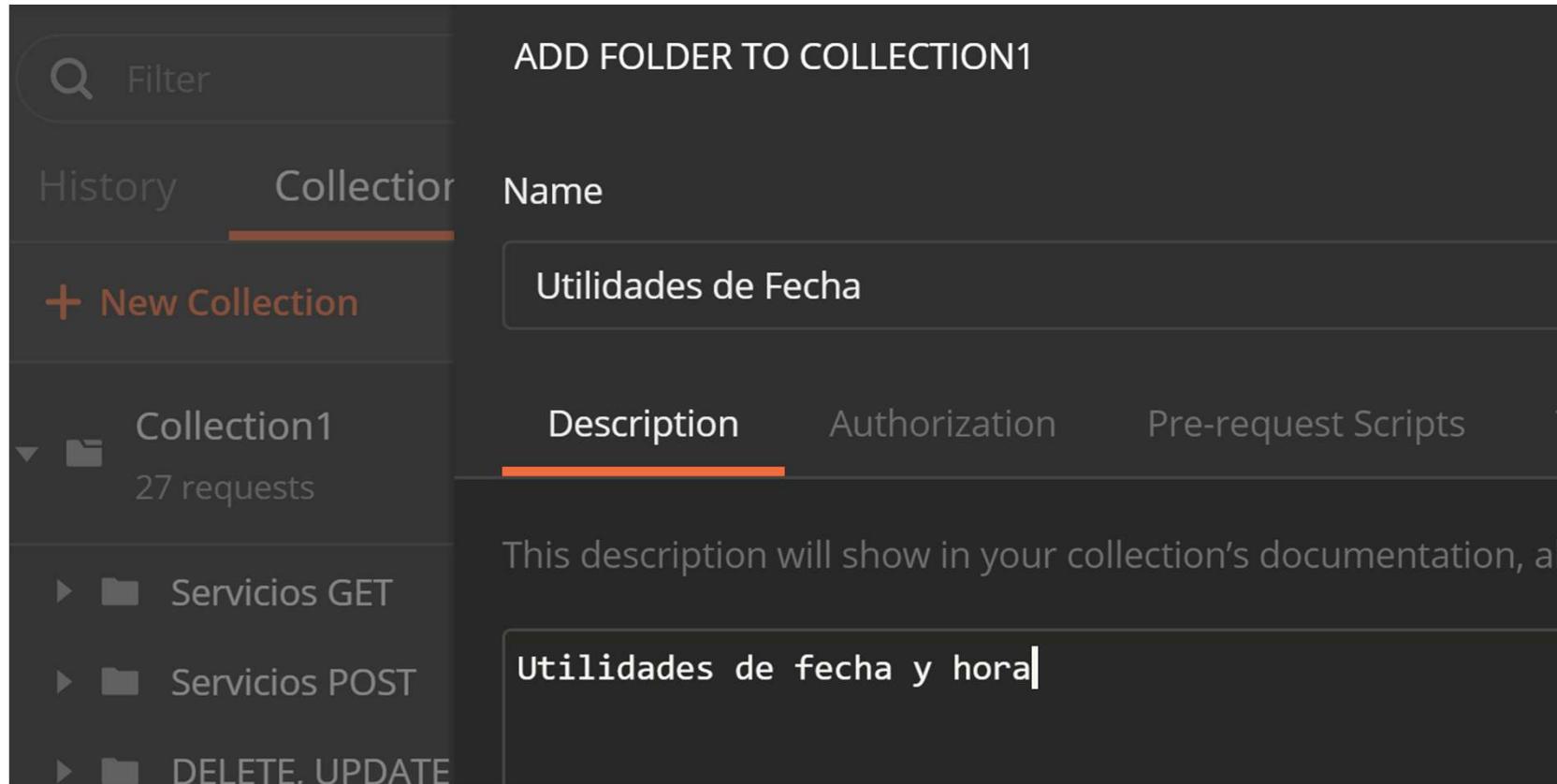
# Utilidades de fecha y hora



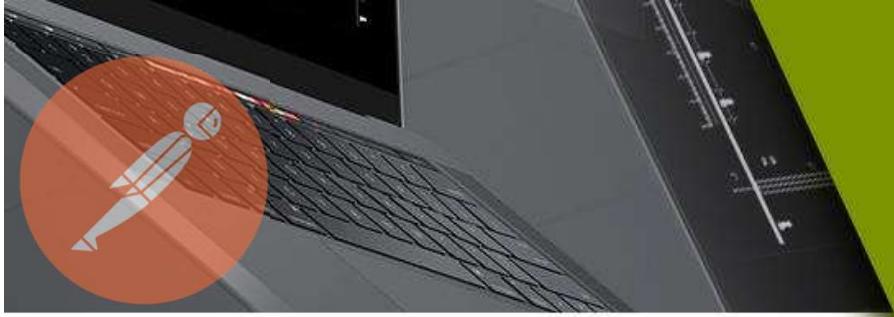


# Utilidades fecha y hora

- Vamos a crear una nueva carpeta en nuestra colección:



The screenshot shows the Postman application interface. On the left, there's a sidebar with tabs for 'History' and 'Collection'. The 'Collection' tab is selected, and a button '+ New Collection' is visible. Below this, there's a list of collections: 'Collection1' which contains '27 requests' and has sub-folders 'Servicios GET', 'Servicios POST', and 'DELETE, UPDATE'. The main area is titled 'ADD FOLDER TO COLLECTION1'. It has fields for 'Name' (containing 'Utilidades de Fecha'), 'Description' (containing 'Utilidades de fecha y hora'), 'Authorization', and 'Pre-request Scripts'. A note says 'This description will show in your collection's documentation, al'. The overall theme of the slide is 'Utilidades fecha y hora'.



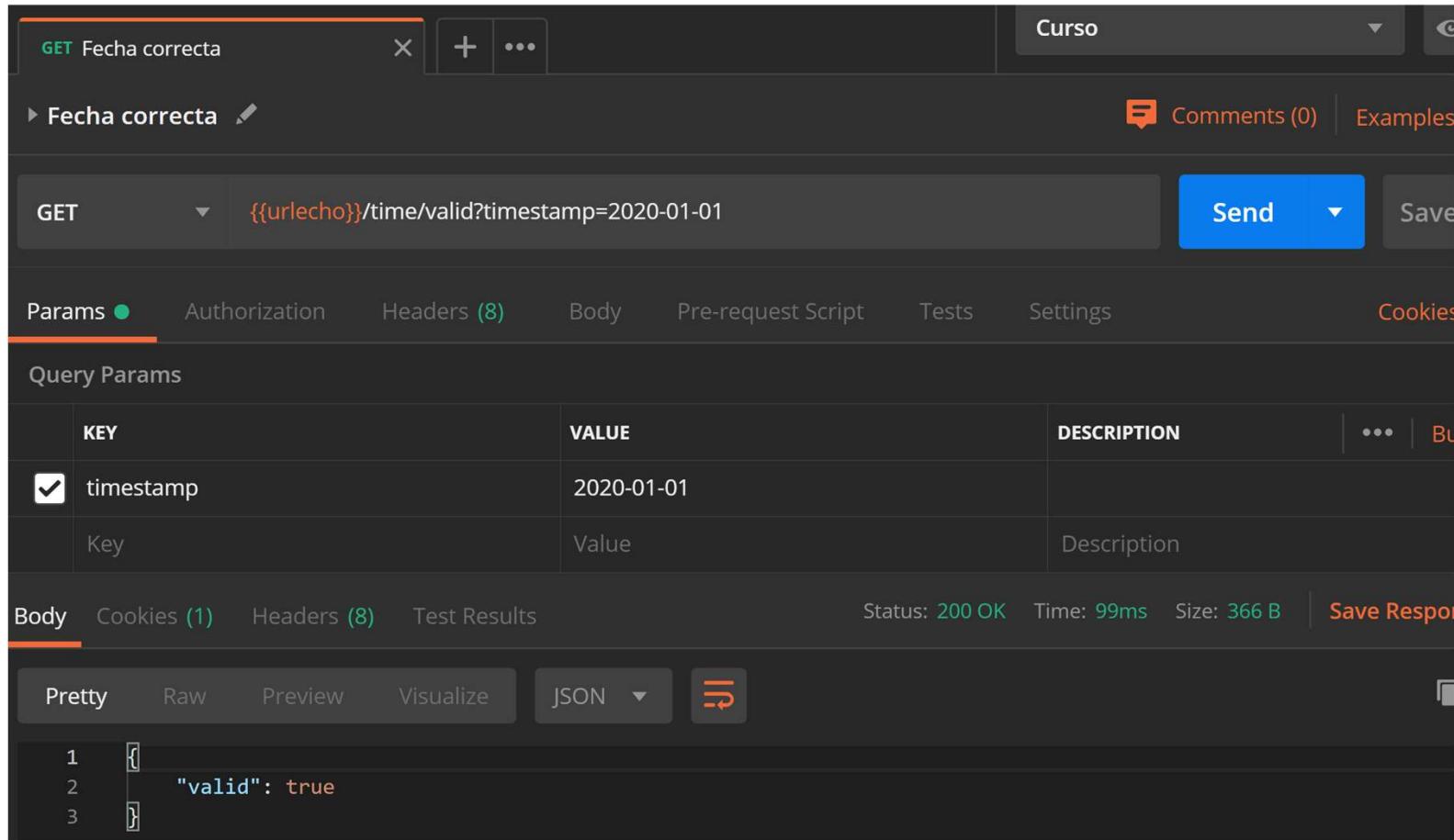
- En muchas servicios web es necesario saber la fecha y hora en la que se ejecuta o solicita el servicio
- Hay una utilidad para obtener esos datos:

The screenshot shows the Postman application interface. At the top, there's a header bar with 'GET Now' and a dropdown menu 'Curso'. Below the header, the main workspace shows a request configuration. The method is set to 'GET' and the URL is '{{urlecho}}/time/now'. To the right of the URL is a 'Send' button. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is currently selected. Under 'Params', there's a table titled 'Query Params' with columns 'KEY', 'VALUE', and 'DESCRIPTION'. A single row is present with 'Key' and 'Value' both empty, and 'Description' also empty. At the bottom of the workspace, there are tabs for 'Body', 'Cookies (1)', 'Headers (9)', and 'Test Results'. The 'Body' tab is selected. On the right side of the bottom bar, status information is displayed: 'Status: 200 OK', 'Time: 102ms', 'Size: 417 B', and a 'Save Response' button. At the very bottom, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'HTML'.

1 Fri, 06 Mar 2020 13:12:53 GMT

 Valid

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



The screenshot shows the Postman interface with the following details:

- Request URL:** GET {{urlecho}}/time/valid?timestamp=2020-01-01
- Method:** GET
- Params Tab (selected):**
  - timestamp: 2020-01-01
- Body Tab:** Status: 200 OK, Time: 99ms, Size: 366 B
- Response Body (Pretty JSON):**

```
1 {  
2     "valid": true  
3 }
```

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

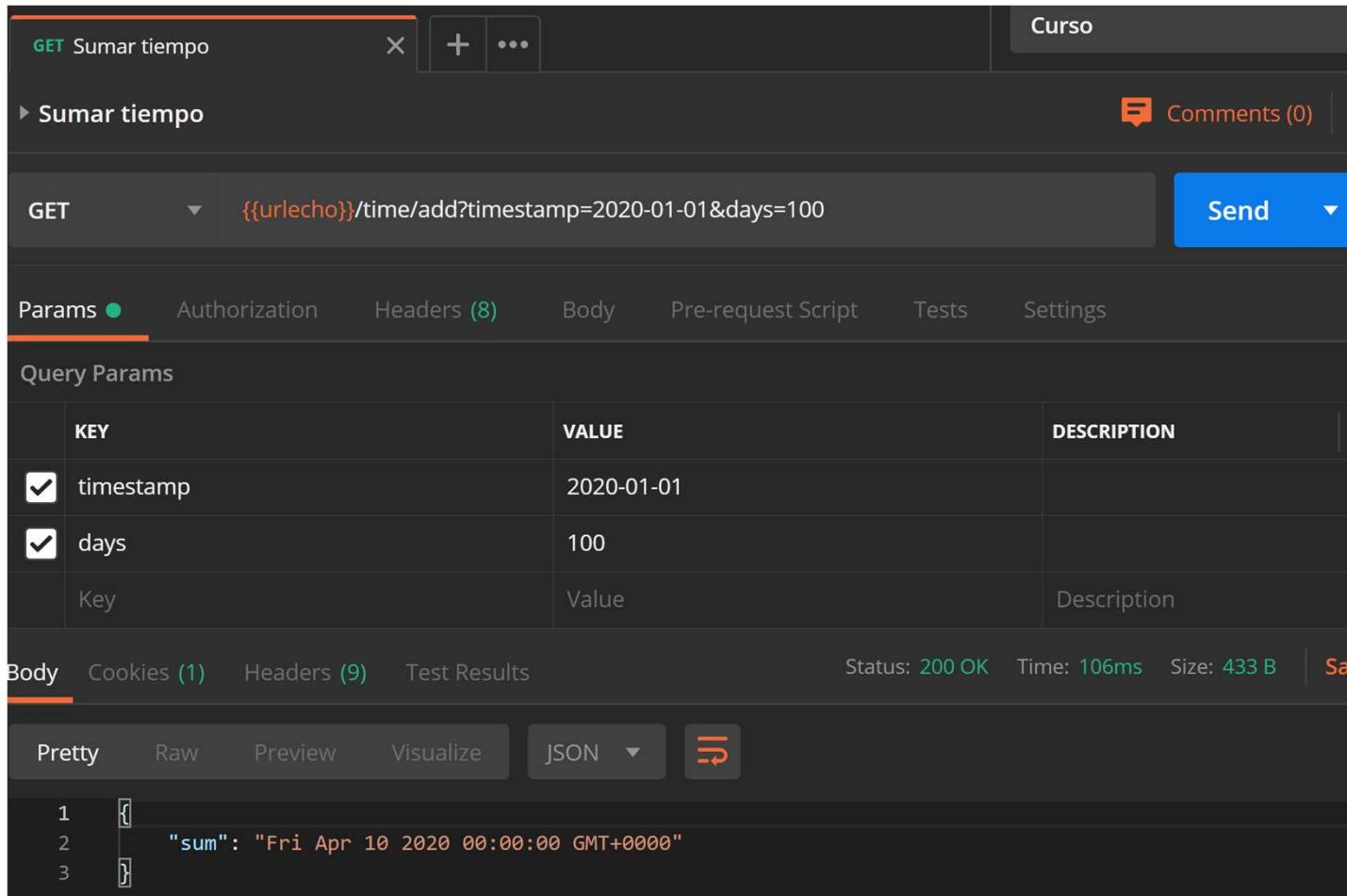
The screenshot shows the Postman interface with the following details:

- Request URL:** {{urlecho}}/time/unit?timestamp=2020-01-01&unit=day
- Method:** GET
- Params:**

KEY	VALUE	DESCRIPTION
timestamp	2020-01-01	
unit	day	
- Body:** {"unit": "3"}
- Status:** 200 OK
- Time:** 149ms
- Size:** 358 B

 Add

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



The screenshot shows the Postman interface with the following details:

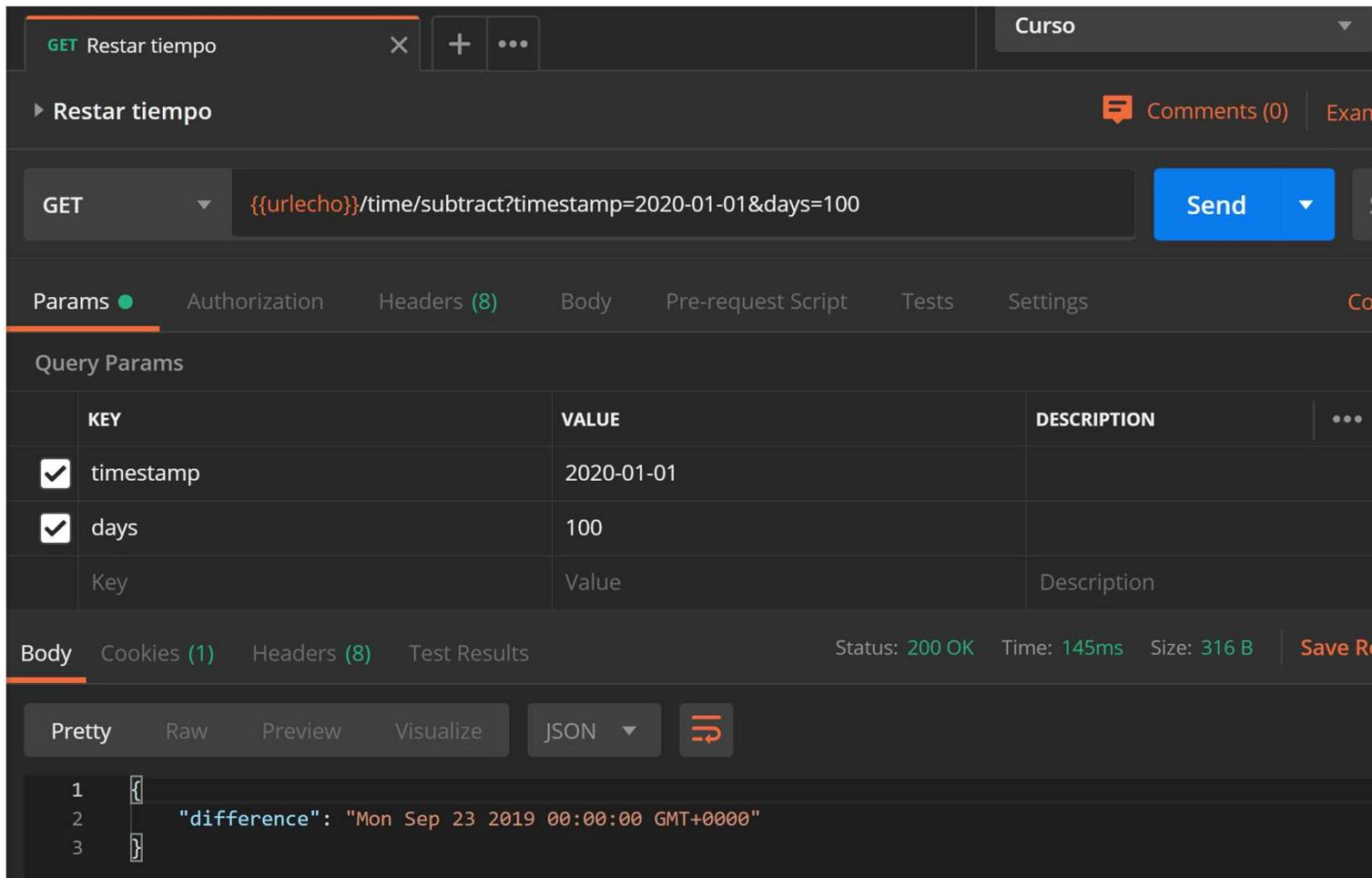
- Request URL:** {{urlecho}}/time/add?timestamp=2020-01-01&days=100
- Method:** GET
- Params:** timestamp: 2020-01-01, days: 100
- Body:** (Pretty, Raw, Preview, Visualize, JSON) - The response body is shown as:

```
1 [ {  
2   "sum": "Fri Apr 10 2020 00:00:00 GMT+0000"  
3 } ]
```



# Subtract

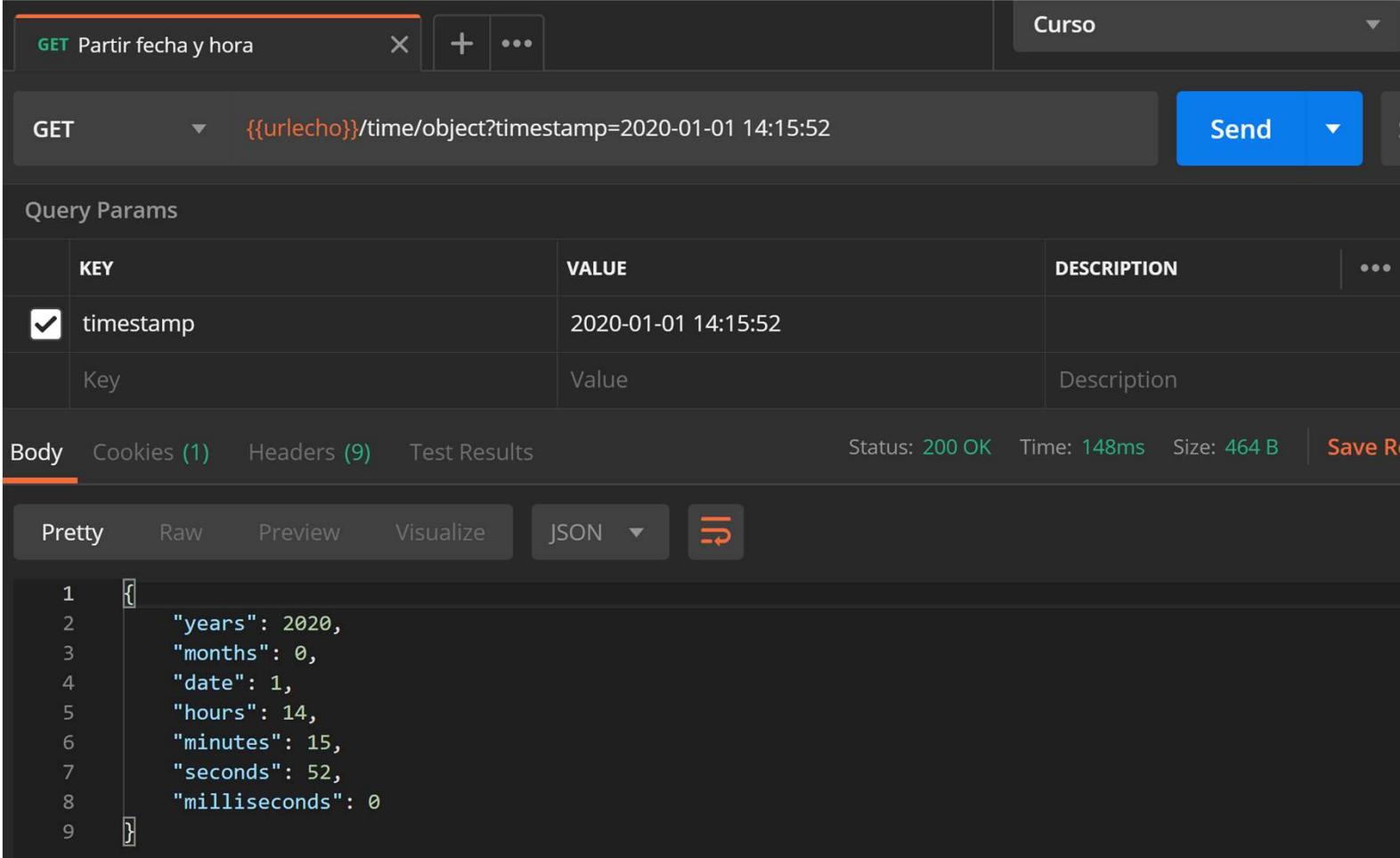
- Duplicamos la anterior petición y modificamos el add por el subtract para restar tiempo a una fecha:



The screenshot shows the Postman application interface. The top bar displays the method as "GET" and the URL as "{{urlecho}}/time/subtract?timestamp=2020-01-01&days=100". The "Params" tab is selected, showing two query parameters: "timestamp" with value "2020-01-01" and "days" with value "100". The "Body" tab is also visible at the bottom. The response status is "200 OK" with a duration of "145ms" and a size of "316 B". The JSON response body is displayed as:

```
1 [ { "difference": "Mon Sep 23 2019 00:00:00 GMT+0000" } ]
```

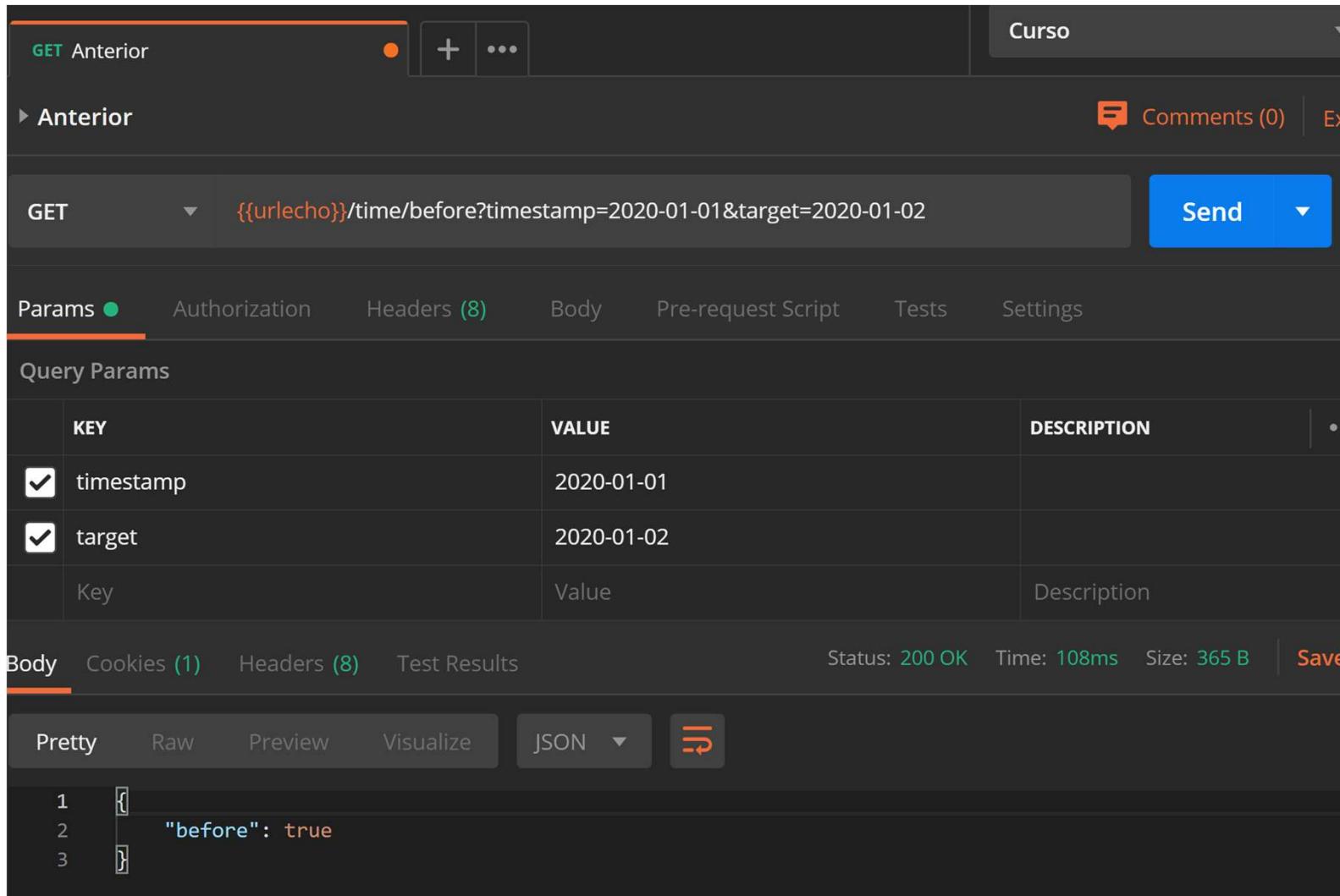
- Utilidad para dividir la fecha y hora en cada una de sus unidades (meses cuentan desde el 0):



The screenshot shows a REST API testing interface. The top bar has a title 'Partir fecha y hora' and a dropdown 'Curso'. The main area shows a GET request to `{{urlecho}}/time/object?timestamp=2020-01-01 14:15:52`. Below it, the 'Query Params' section lists a checked 'timestamp' entry with value '2020-01-01 14:15:52'. The 'Body' tab is selected, displaying a JSON response:

```
1 {
2     "years": 2020,
3     "months": 0,
4     "date": 1,
5     "hours": 14,
6     "minutes": 15,
7     "seconds": 52,
8     "milliseconds": 0
9 }
```

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



The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'GET Anterior' selected, a '+' button, and a 'Curso' dropdown. Below the navigation is a section titled 'Anterior' with 'Comments (0)' and an 'Ex' link. The main area shows a 'GET' request with the URL {{urlecho}}/time/before?timestamp=2020-01-01&target=2020-01-02. A 'Send' button is visible. Below the URL, under 'Params', are two checked query parameters: 'timestamp' with value '2020-01-01' and 'target' with value '2020-01-02'. The 'Body' tab is selected at the bottom, showing a JSON response: { "before": true }. Other tabs like 'Cookies (1)', 'Headers (8)', and 'Test Results' are also present.

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

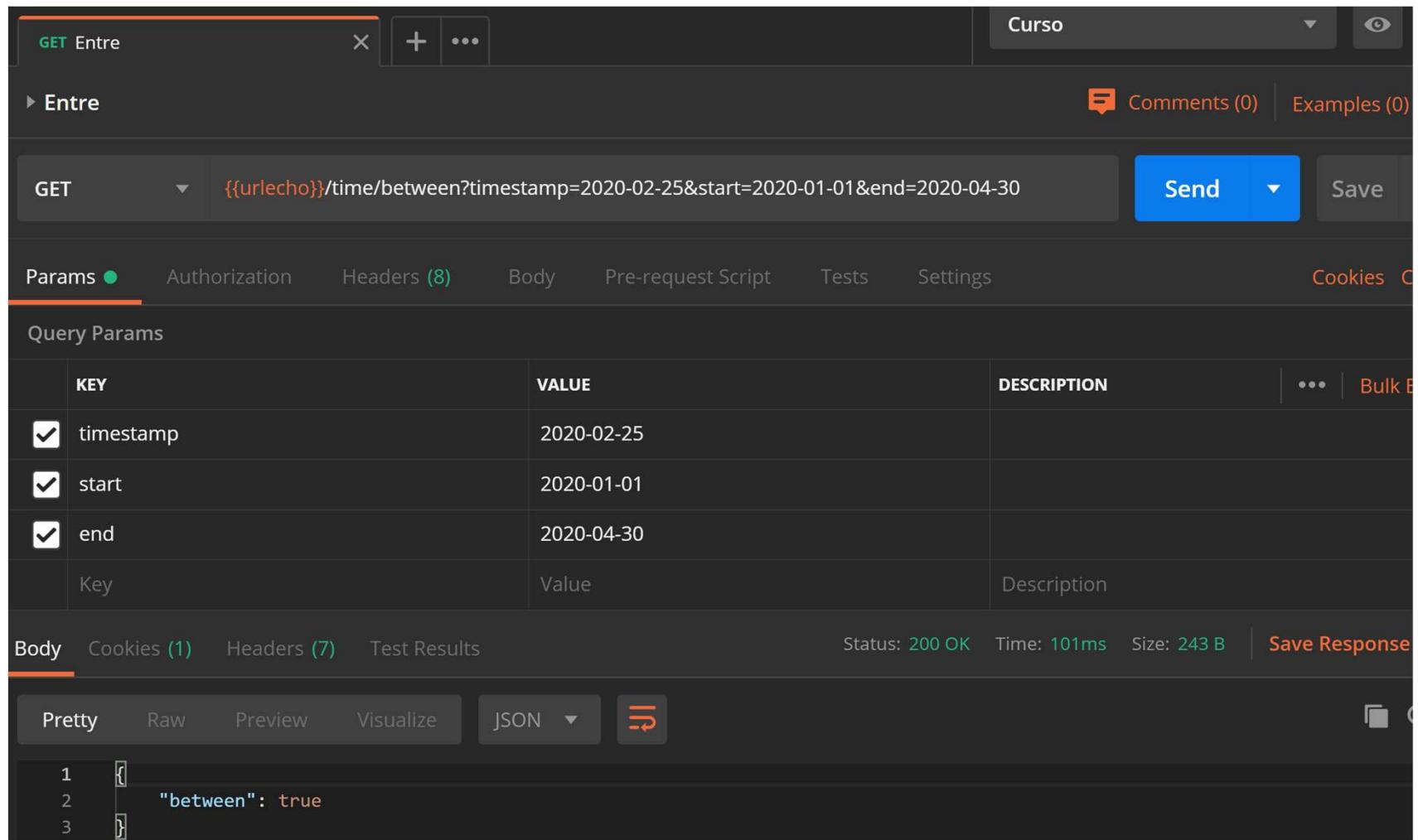
The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'GET Posterior' and a 'Curso' tab. Below the navigation is a search bar with 'Posterior'. On the right side of the header, there are 'Comments (0)' and a 'Send' button. The main area shows a 'GET' request with the URL {{urlecho}}/time/after?timestamp=2020-01-01&target=2020-01-02. The 'Params' tab is selected, showing two query parameters: 'timestamp' (value: 2020-01-01) and 'target' (value: 2020-01-02). The 'Body' tab is also visible at the bottom. The status bar at the bottom right indicates 'Status: 200 OK'.

KEY	VALUE	DESCRIPTION
timestamp	2020-01-01	
target	2020-01-02	

Body

```
1 [{  
2     "after": false  
3 }]
```

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



The screenshot shows the Postman application interface. At the top, there's a header bar with the URL "GET Entre" and a dropdown menu "Curso". Below the header, the main workspace shows a "Params" tab selected, displaying three query parameters: "timestamp" (value: 2020-02-25), "start" (value: 2020-01-01), and "end" (value: 2020-04-30). To the right of the parameters are "Send" and "Save" buttons. Below the parameters, there are tabs for "Body", "Cookies (1)", "Headers (7)", and "Test Results". The "Body" tab is currently active, showing a JSON response: { "between": true }. At the bottom, there are buttons for "Pretty", "Raw", "Preview", "Visualize", and "JSON". The status bar at the bottom indicates "Status: 200 OK", "Time: 101ms", and "Size: 243 B".

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

The screenshot shows the Postman application interface. At the top, there's a header bar with 'GET Bisiesto' and a status indicator. Below it, the collection name 'Bisiesto' is shown with a dropdown for 'Curso'. On the right, there are buttons for 'Comments (0)', 'Examples (0)', 'Send', and 'Save'. The main area shows a 'Params' tab selected, with a table for 'Query Params'. One row has a checked checkbox for 'timestamp' with the value '2019-04-01'. Other tabs include 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', 'Settings', 'Cookies', and 'Code'. At the bottom, there are tabs for 'Body', 'Cookies (1)', 'Headers (8)', and 'Test Results'. The 'Body' tab is active, showing a status of '200 OK', a time of '145ms', and a size of '362 B'. A 'Save Response' button is also present. The JSON response is displayed as follows:

```
1 [{}]
2   "leap": false
3 ]
```

# Tests y Scripts



# Primer test básico

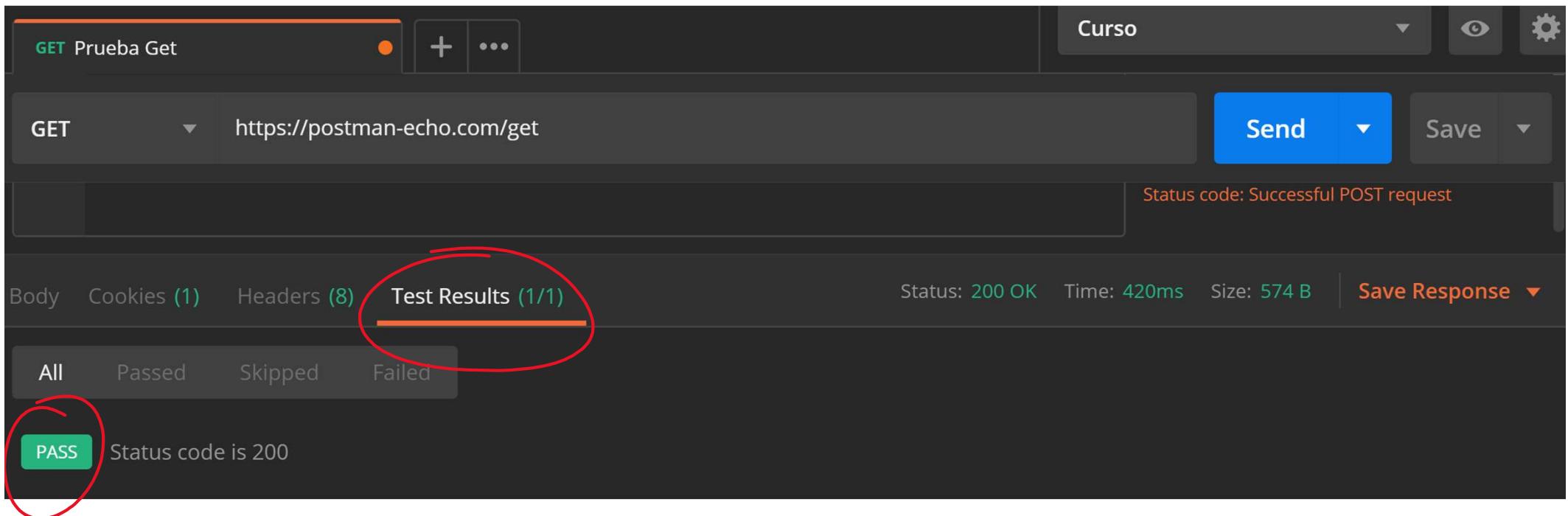
- Postman está hecho principalmente para realizar test sobre nuestros servicios web (en Javascript):

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'History', 'Collections' (which is selected and highlighted in orange), 'APIs', and 'Trash'. Below 'Collections' is a '+ New Collection' button. Under 'Collection1', there's a folder named 'Servicios GET' containing five requests: 'Prueba Get', 'Prueba Get Parámetros', 'Ejemplo de ID', 'GET variables de entorno', and 'GET variables globales'. The 'Prueba Get' request is currently selected. The main workspace shows a 'GET' request to 'https://postman-echo.com/get'. The 'Tests' tab is highlighted with a red circle. A red arrow points from the 'Tests' tab towards the 'Send' button, which is also circled in red. To the right of the 'Send' button, a status message says 'Status code: Code is 200'. At the bottom right, there's a 'SNIPPETS' section with the text 'Status code: Code is 200' and 'Response body: Contains string'.



# Primer test básico

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



The screenshot shows the Postman application interface. At the top, there's a header with the URL "GET Prueba Get" and a status indicator. Below the header, the main area has fields for "Method" (GET), "URL" (https://postman-echo.com/get), and buttons for "Send" and "Save". The "Send" button is blue and has a dropdown arrow. The "Save" button is also blue and has a dropdown arrow. To the right of the URL field, it says "Status code: Successful POST request". Below the URL field, there are tabs for "Body", "Cookies (1)", "Headers (8)", and "Test Results (1/1)". The "Test Results" tab is highlighted with a red oval. At the bottom, there are buttons for "All", "Passed", "Skipped", and "Failed", with "Passed" being the active tab. A green button labeled "PASS" is circled with a red oval. To the right of the "PASS" button, it says "Status code is 200". On the far right, there are buttons for "Status: 200 OK", "Time: 420ms", "Size: 574 B", and "Save Response".

# 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:

The screenshot shows the Postman interface with a failed test result. The URL in the request field is https://postman-echo.com/getttt. The 'Send' button and its status message 'Status code: Successful POST request' are circled in red. The 'Test Results (0/1)' tab is selected, showing a failure message: 'Status: 404 Not Found'. The 'FAIL' button at the bottom left is also circled in red.

# Test de JSON

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

The screenshot shows the Postman application interface. On the left, the sidebar displays a collection named "Collection1" containing several requests, with the "Servicios POST" folder expanded. A red circle highlights the "POST mediante Raw JSON" request under this folder. In the main workspace, a specific POST request is selected, showing its configuration. The "Tests" tab is highlighted with a red circle, indicating where test scripts are written. Below the tabs, a code editor contains a JavaScript test script:

```
1 pm.test("Your test name", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.value).to.eql(100);  
4});|
```

To the right of the code editor, a tooltip provides information about test scripts: "Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts". Further down, another tooltip explains response body checks: "Response body: Contains string", "Response body: JSON value check", and "Response body: Is equal to a string".

# Test de JSON

- Modificamos el test:

The screenshot shows the Postman interface. At the top, it says "POST POST mediante Raw JSON". Below that, under "Tests", there is a code block:

```
1 pm.test("Resultados del JSON - Nombre", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.data.nombre).to.eql("David");  
4     pm.expect(jsonData.data.apellido).to.eql("Granada");  
5 });  
6  
7 pm.test("Resultados del JSON - Edad", function () {  
8     var jsonData = pm.response.json();  
9     pm.expect(jsonData.data.edad).to.eql(40);  
10});
```

At the bottom, the "Body" tab is selected, showing the raw JSON response:

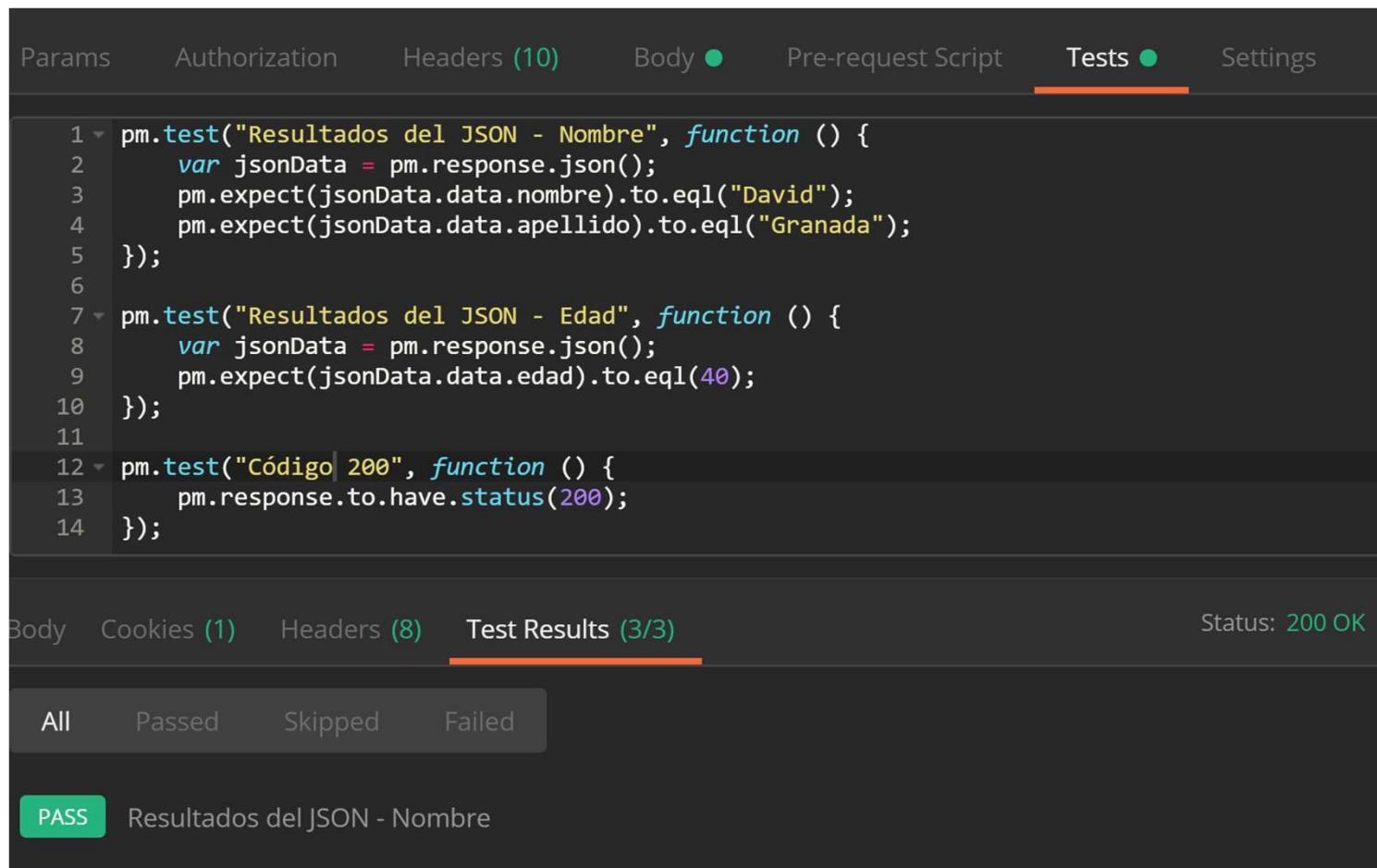
```
1 [ {  
2     "args": {},  
3     "data": {  
4         "nombre": "David",  
5         "apellido": "Granada",  
6         "edad": 40  
7     } } ]
```

All	Passed	Skipped	Failed
2	2	0	0
Body Cookies (1) Headers (9) Test Results (2/2)			
<span>PASS</span> Resultados del JSON - Nombre			
<span>PASS</span> Resultados del JSON - Edad			



# Test de JSON

- Agregamos alguna otra comprobación:



The screenshot shows the Postman interface with the 'Tests' tab selected. The test script contains three assertions:

```
1 pm.test("Resultados del JSON - Nombre", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.data.nombre).to.eql("David");
4     pm.expect(jsonData.data.apellido).to.eql("Granada");
5 });
6
7 pm.test("Resultados del JSON - Edad", function () {
8     var jsonData = pm.response.json();
9     pm.expect(jsonData.data.edad).to.eql(40);
10 });
11
12 pm.test("Código 200", function () {
13     pm.response.to.have.status(200);
14 });
```

The 'Test Results' section shows 3/3 tests passed. The status bar at the bottom indicates 'Status: 200 OK'.

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

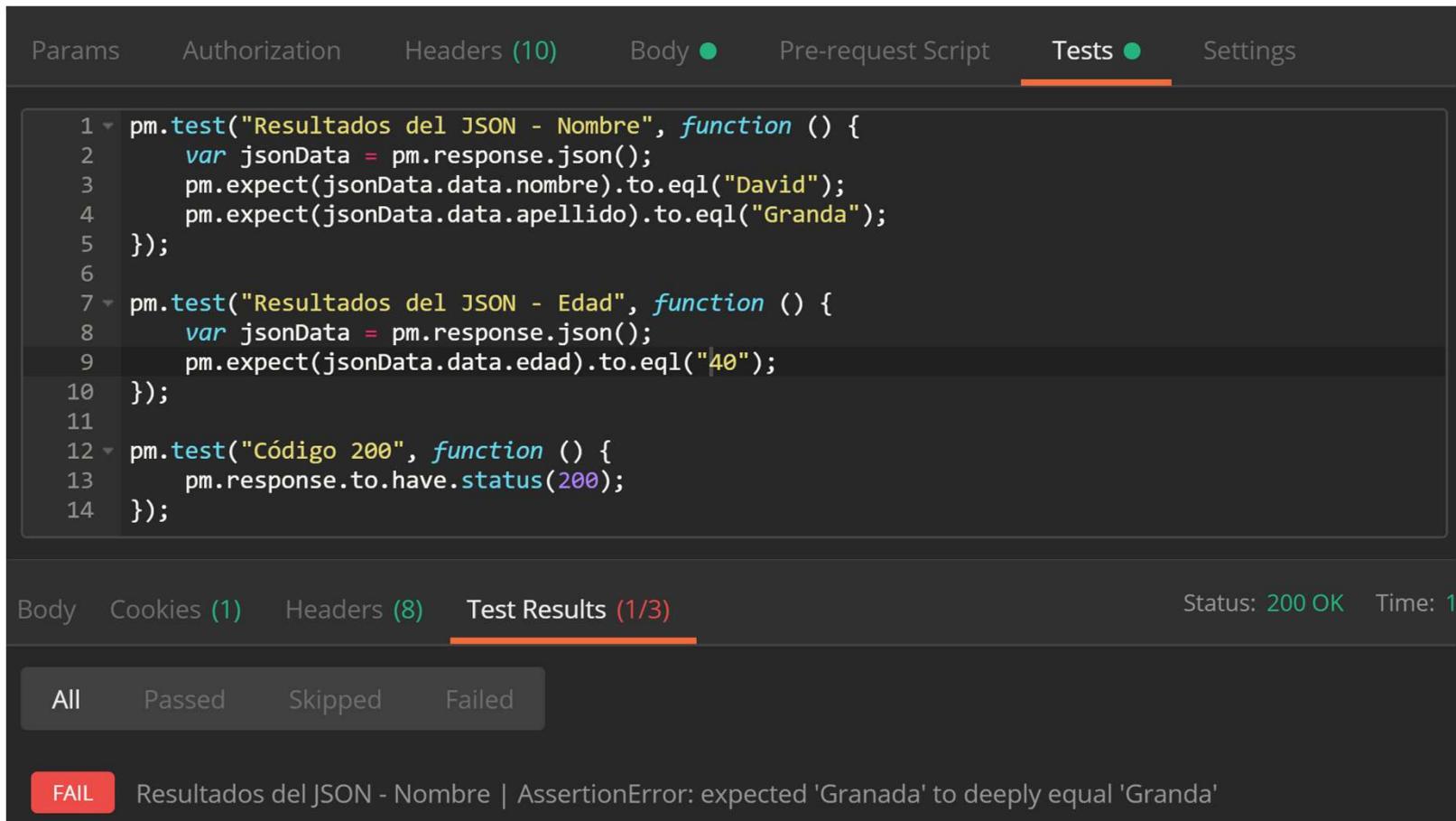
All Passed Skipped Failed

PASS Resultados del JSON - Nombre



# Test de JSON

- Podemos probar a cometer errores para ver el resultado:



The screenshot shows the Postman interface with a JSON test script in the 'Tests' tab and its execution results below.

**Tests Tab Content:**

```
1 pm.test("Resultados del JSON - Nombre", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.data.nombre).to.eql("David");
4     pm.expect(jsonData.data.apellido).to.eql("Granda");
5 });
6
7 pm.test("Resultados del JSON - Edad", function () {
8     var jsonData = pm.response.json();
9     pm.expect(jsonData.data.edad).to.eql("40");
10 });
11
12 pm.test("Código 200", function () {
13     pm.response.to.have.status(200);
14 });
```

**Test Results Tab Content:**

Status: 200 OK Time: 1

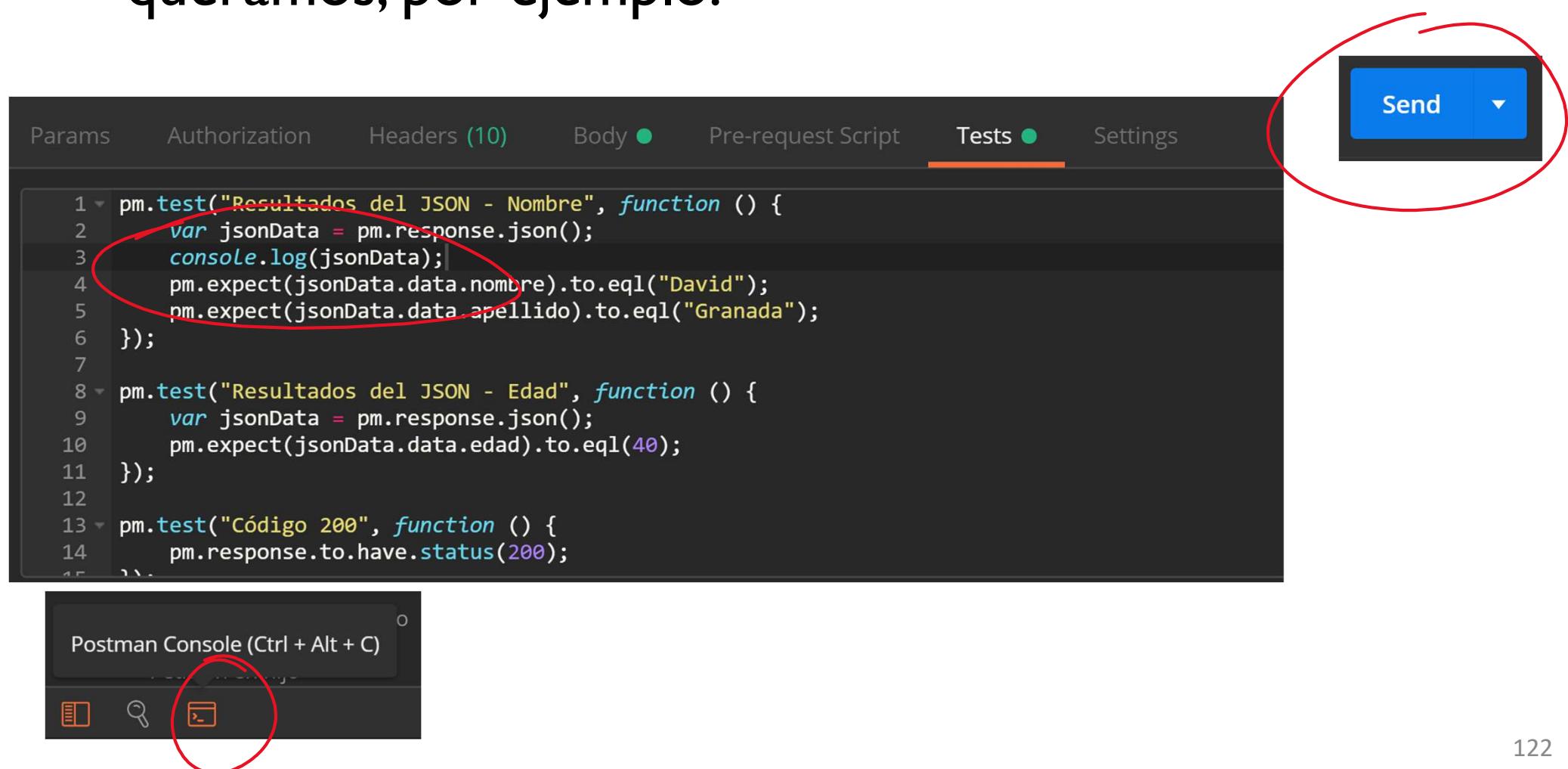
All Passed Skipped Failed

**FAIL** Resultados del JSON - Nombre | Assertion Error: expected 'Granada' to deeply equal 'Granda'



# Test de JSON

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



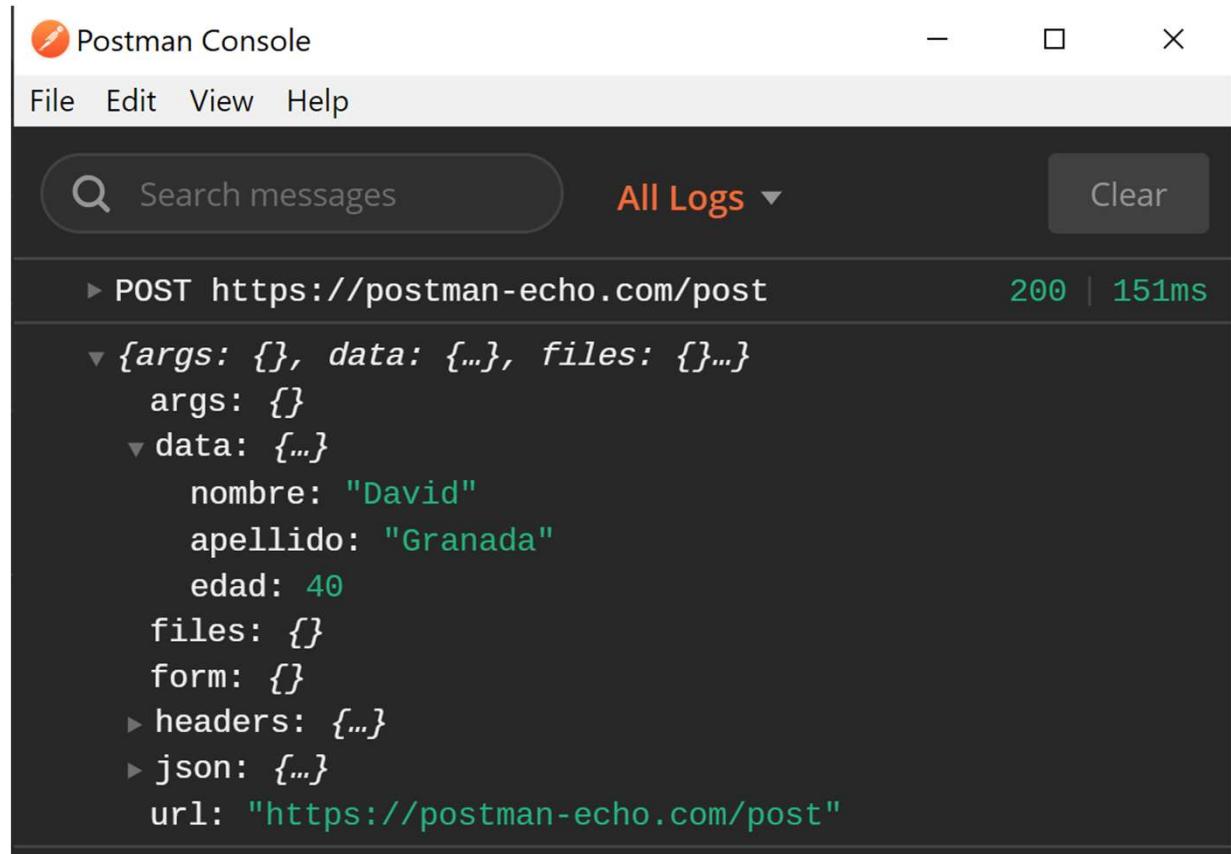
The screenshot shows the Postman interface with the 'Tests' tab selected. A red circle highlights the 'Send' button in the top right corner. Another red circle highlights the 'Console' icon in the bottom left toolbar.

```
1 pm.test("Resultados del JSON - Nombre", function () {
2     var jsonData = pm.response.json();
3     console.log(jsonData);
4     pm.expect(jsonData.data.nombre).to.eql("David");
5     pm.expect(jsonData.data.apellido).to.eql("Granada");
6 });
7
8 pm.test("Resultados del JSON - Edad", function () {
9     var jsonData = pm.response.json();
10    pm.expect(jsonData.data.edad).to.eql(40);
11 });
12
13 pm.test("Código 200", function () {
14     pm.response.to.have.status(200);
15 });
```



# Test de JSON

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



```
Postman Console
File Edit View Help
Search messages All Logs Clear
▶ POST https://postman-echo.com/post 200 | 151ms
▼ {args: {}, data: {...}, files: {}...}
  args: {}
  ▼ data: {...}
    nombre: "David"
    apellido: "Granada"
    edad: 40
    files: {}
    form: {}
  ▶ headers: {...}
  ▶ json: {...}
  url: "https://postman-echo.com/post"
```

# 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:

The screenshot shows the Postman application interface. On the left, there's a sidebar with a collection named 'Delay' circled in red. Below it, other collections like 'Status', 'Stream', 'UTF8', and 'IP' are listed. In the main workspace, a collection named 'Delay' is selected. Inside, a 'GET /delay/2' request is shown. The URL field contains '{{urlecho}}/delay/2'. A red circle highlights this URL. The 'Tests' tab is selected, displaying a JavaScript test script:

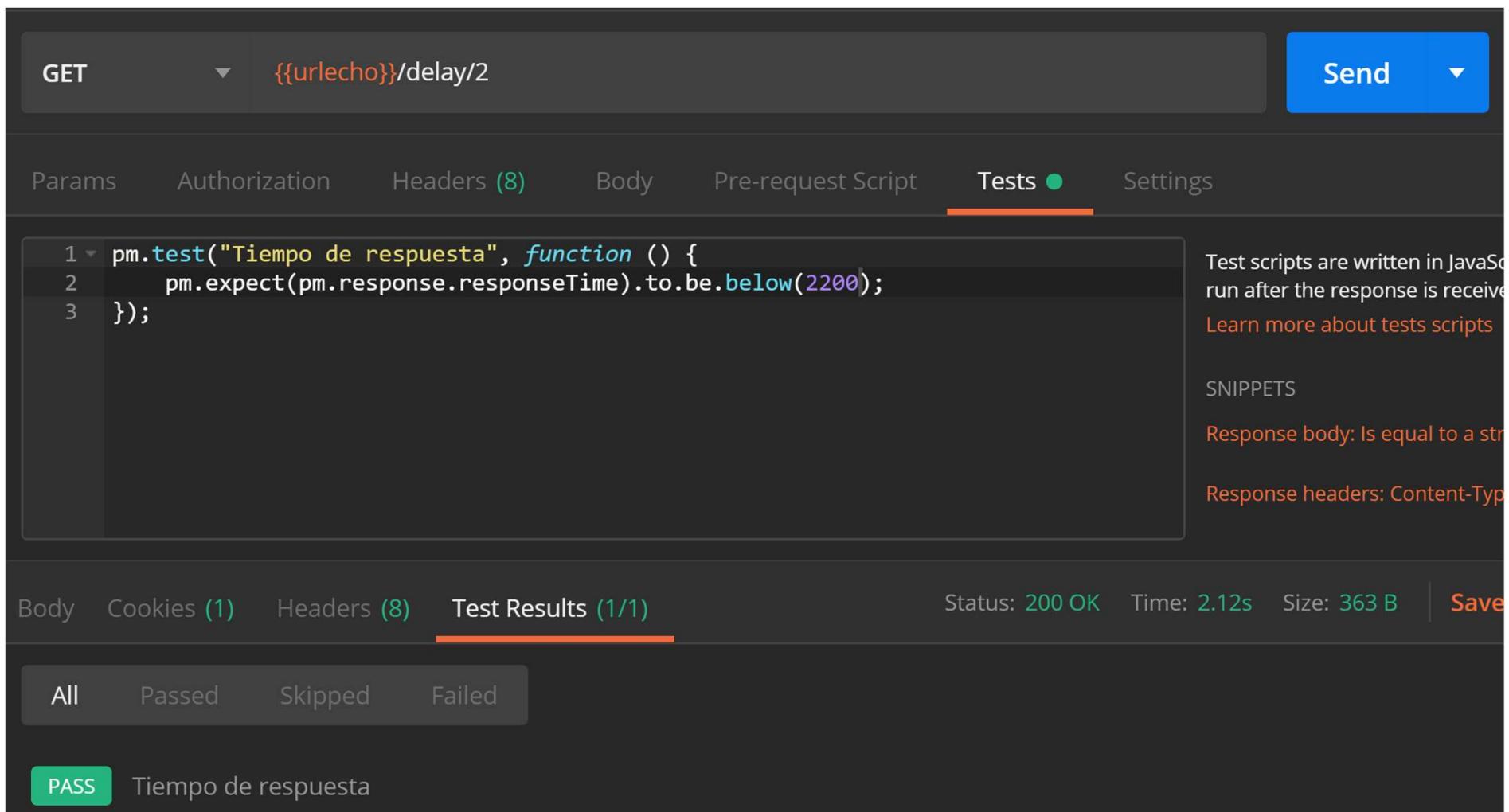
```
1 pm.test("Response time is less than 200ms", function () {  
2     pm.expect(pm.response.responseTime).to.be.below(200);  
3 });
```

A red arrow points from the explanatory text below the script to the 'below(200)' part of the script. To the right of the tests, a snippet of text is also circled in red: 'Response time is less than 200ms'. The top right of the screen shows a dropdown menu set to 'Curso', and the bottom right corner has the number '124'.



# Test de Delay

- Modificamos el código proporcionado por el snippet:



The screenshot shows the Postman interface with a dark theme. At the top, there's a header bar with 'GET' and the URL '{{urlecho}}/delay/2'. To the right is a 'Send' button. Below the header are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', and 'Tests'. The 'Tests' tab is selected, highlighted with a red underline. In the main content area, a code editor displays a snippet of JavaScript:

```
1 pm.test("Tiempo de respuesta", function () {  
2     pm.expect(pm.response.responseTime).to.be.below(2200);  
3 });
```

To the right of the code editor, there's a tooltip with the following text:

Test scripts are written in JavaScript and run after the response is received. Learn more about tests scripts.

Below the code editor, there are tabs for 'Body', 'Cookies (1)', 'Headers (8)', and 'Test Results (1/1)'. The 'Test Results' tab is selected, highlighted with a red underline. At the bottom of the results section, there are buttons for 'All', 'Passed', 'Skipped', and 'Failed'. The status bar at the bottom shows 'Status: 200 OK', 'Time: 2.12s', 'Size: 363 B', and a 'Save' button. At the very bottom, there's a green button labeled 'PASS' and the text 'Tiempo de respuesta'.

# Test de Delay

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

The screenshot shows the Postman interface for a GET request to `{urlecho}/delay/2`. The 'Tests' tab is selected, displaying a JavaScript test script:

```
1 pm.test("Tiempo de respuesta", function () {  
2     pm.expect(pm.response.responseTime).to.be.below(2100);  
3 });
```

A red oval highlights the assertion line `pm.expect(pm.response.responseTime).to.be.below(2100);`. Below the tests section, the 'Test Results (0/1)' button is highlighted with a red oval. At the bottom, a red box labeled 'FAIL' contains the error message: 'Tiempo de respuesta | Assertion Error: expected 2108 to be below 2100'. The status bar at the bottom right shows 'Status: 200 OK'.

# Test de búsqueda

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

The screenshot shows the Postman application interface. On the left, there's a sidebar with a collection named "Servicios GET" containing several requests, one of which is highlighted with a red oval. The main area shows a request titled "Prueba Get" with the URL "https://postman-echo.com/get". The "Tests" tab is selected, displaying a JavaScript code snippet. This code includes two test cases: one for the status code (status 200) and another for the response body containing a specific string ("string\_you\_want\_to\_search"). The results pane on the right shows the test results: "Status code: Code is 200" and "Response body: Contains string". A large red oval encircles the entire "Tests" section, highlighting the search functionality.

```
1 pm.test("Status code is 200", function () {  
2     pm.response.to.have.status(200);  
3 };  
4  
5 pm.test("Body matches string", function () {  
6     pm.expect(pm.response.text()).to.include("string_you_want_to_search");  
7 };|  
8 );|
```

Test scripts are written in JavaScript, and are run after the response is received.  
Learn more about tests scripts

SNIPPETS

Status code: Code is 200

Response body: Contains string

# Test de búsqueda

- Modificamos el código para que busque una palabra:

The screenshot shows the Postman application interface. At the top, there's a navigation bar with a search icon, followed by tabs for 'GET Prueba Get' and 'Curso'. Below the navigation is a toolbar with 'GET', a dropdown for 'https://postman-echo.com/get', and a 'Send' button. The main area has tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests' (which is selected), and 'Settings'. The 'Tests' tab contains the following JavaScript code:

```
1 pm.test("Status code is 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5 pm.test("Body matches string", function () {
6     pm.expect(pm.response.text()).to.include("deflate");
7 });
8 }
```

A red oval highlights the line `pm.expect(pm.response.text()).to.include("deflate");` in the tests script. To the right of the script, there's a sidebar with information about test scripts, snippets, and response body details. At the bottom, there are tabs for 'Body', 'Cookies (1)', 'Headers (9)', and 'Test Results (2/2)'. The 'Test Results' tab is active, showing two green 'PASS' status indicators: 'Status code is 200' and 'Body matches string'. Status details at the bottom include 'Status: 200 OK', 'Time: 101ms', and 'Size: 697 B'.



# Test de respuesta exacta

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

The screenshot shows the Postman interface with a dark theme. At the top, there's a navigation bar with 'GET Delay' selected, a '+' button, and a 'Curso' dropdown. Below the navigation is a section for the 'Delay' collection, showing 'Comments (0)', 'Examples (0)', and a 'Send' button. The main area shows a 'GET' request with the URL {{urlecho}}/delay/2. The 'Tests' tab is active, displaying the following JavaScript code:

```
1 pm.test("Tiempo de respuesta", function () {  
2     pm.expect(pm.response.responseTime).to.be.below(2300);  
3 } );  
4  
5 pm.test("Body is correct", function () {  
6     pm.response.to.have.body("response_body_string");  
7 } );  
8  
9
```

Three parts of the interface are circled with red lines: 1) The URL field containing {{urlecho}}/delay/2. 2) The 'Tests' tab where the script is written. 3) The 'Body is correct' test block in the script.

On the right side of the interface, there are several snippets and descriptions:

- Test scripts are written in JavaScript, and are run after the response is received. [Learn more about tests scripts](#)
- SNIPPETS
  - Response body: Contains string
  - Response body: JSON value check
  - Response body: Is equal to a string



# Test de respuesta exacta

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

The screenshot shows the Postman interface with the 'Body' tab selected. The response body is displayed as a JSON object: `[{"delay": "2"}]`. Below the body, there are tabs for Cookies (1), Headers (8), and Test Results (1/2). At the bottom, there are buttons for Pretty, Raw, Preview, Visualize, and Text.

```
1 pm.test("Tiempo de respuesta", function () {  
2     pm.expect(pm.response.responseTime).to.be.below(2300);  
3 });  
4  
5  
6 pm.test("Body is correct", function () {  
7     pm.response.to.have.body('{"delay": "2"}');  
8 });  
9 |
```

A large red oval highlights the entire code block from line 6 to line 9.

- Ojo con las comillas dobles y simples!



# Test de respuesta exacta

- Ejecutamos y vemos el resultado

The screenshot shows the Postman interface with a dark theme. At the top, there's a header bar with 'GET' and the URL '{{urlecho}}/delay/2'. To the right of the URL is a blue 'Send' button with a dropdown arrow, which is circled in red. Below the header, the test script is displayed:

```
1 pm.test("Tiempo de respuesta", function () {  
2     pm.expect(pm.response.responseTime).to.be.below(2300);  
3 });  
4  
5 pm.test("Body is correct", function () {  
6     pm.response.to.have.body('{"delay":"2"}');  
7 };  
8 );  
9 |
```

To the right of the script, there's a snippet of text: "Test scripts are written in JavaScript, and run after the response is received." with a link "Learn more about tests scripts". Below the script, the results section is shown with tabs for Body, Cookies (1), Headers (8), and Test Results (2/2). The 'Test Results' tab is circled in red. The results table shows two items: 'Tiempo de respuesta' with status PASS and 'Body is correct' with status PASS. At the bottom right, the status is listed as 'Status: 200 OK Time: 2.15s Size: 363 B' and there's a 'Save Res' button.

Test	Status
Tiempo de respuesta	PASS
Body is correct	PASS



## Otros test

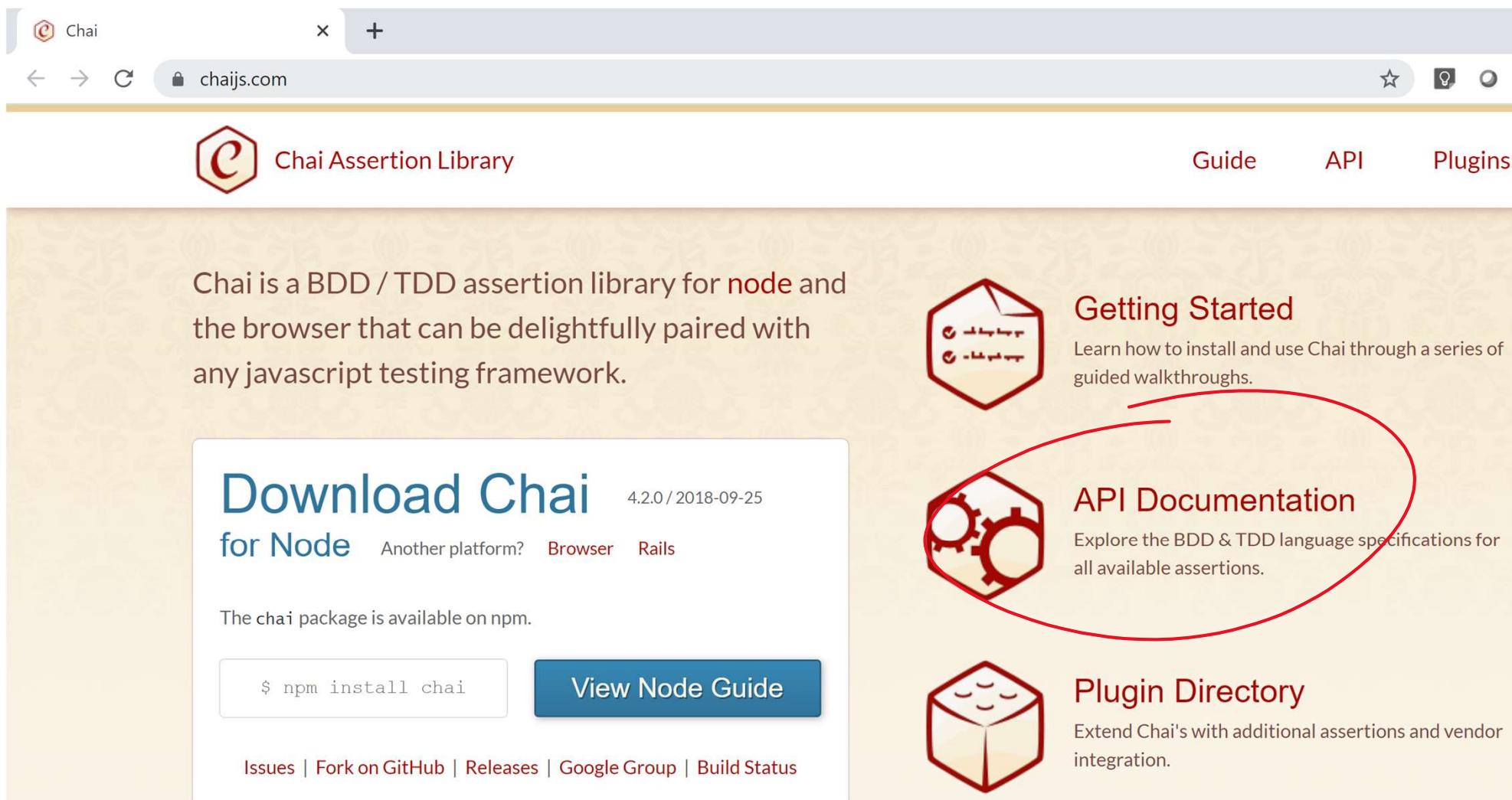
- Podemos verificar prácticamente cualquier valor relacionado con la respuesta del servicio, header, código del status, tiempos, tamaños, etc.

# Assertions



# Assertions

- Funciones que podemos utilizar en nuestros tests:



The screenshot shows the homepage of the Chai Assertion Library. At the top, there's a navigation bar with a logo, a search bar, and links for 'Guide', 'API', and 'Plugins'. Below the header, there's a main content area with a large hexagonal icon and text about Chai being a BDD/TDD assertion library for Node.js and the browser. To the right, there are three sections: 'Getting Started' (with an icon of a document), 'API Documentation' (with an icon of gears), and 'Plugin Directory' (with an icon of a cube). A red oval highlights the 'API Documentation' section. At the bottom left, there's a 'Download Chai' section for Node.js, showing the version 4.2.0 from 2018-09-25, and a link to the Node.js guide. At the very bottom, there are links for Issues, Fork on GitHub, Releases, Google Group, and Build Status.

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.

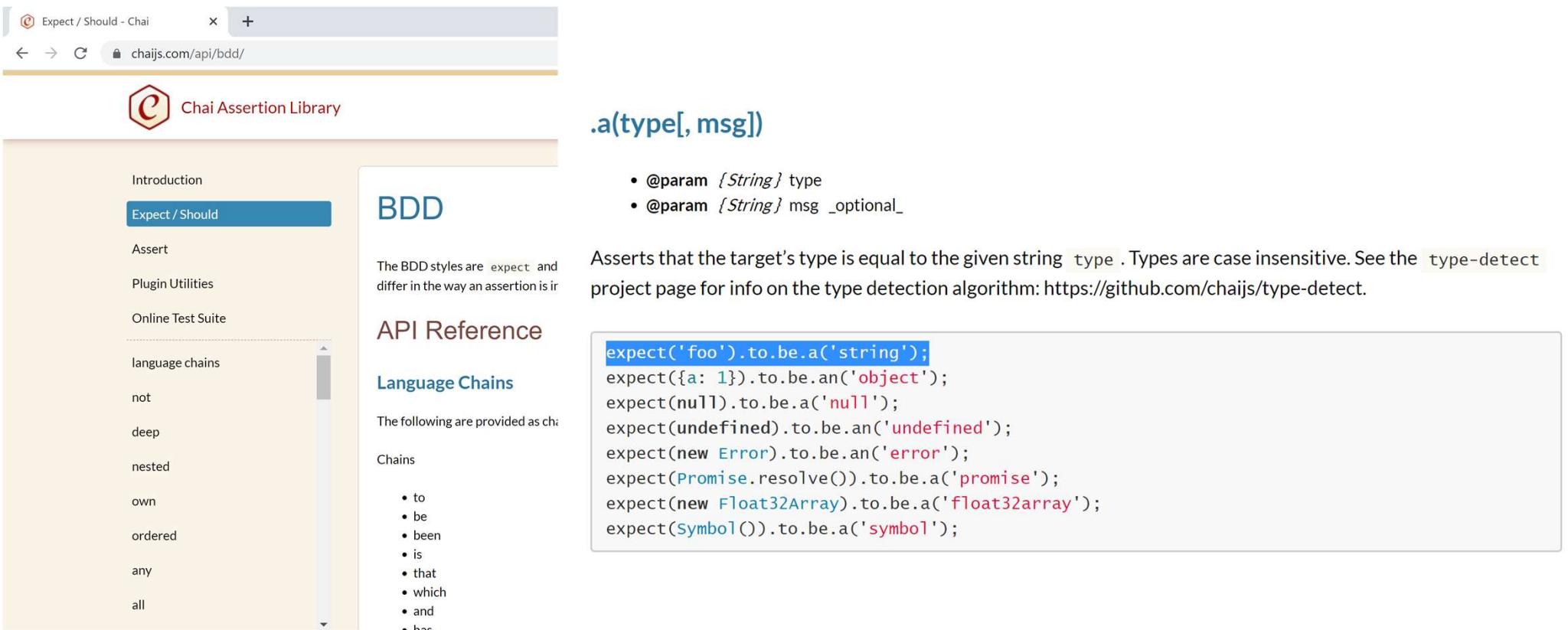
### Plugin Directory

Extend Chai's with additional assertions and vendor integration.



# Assertions

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



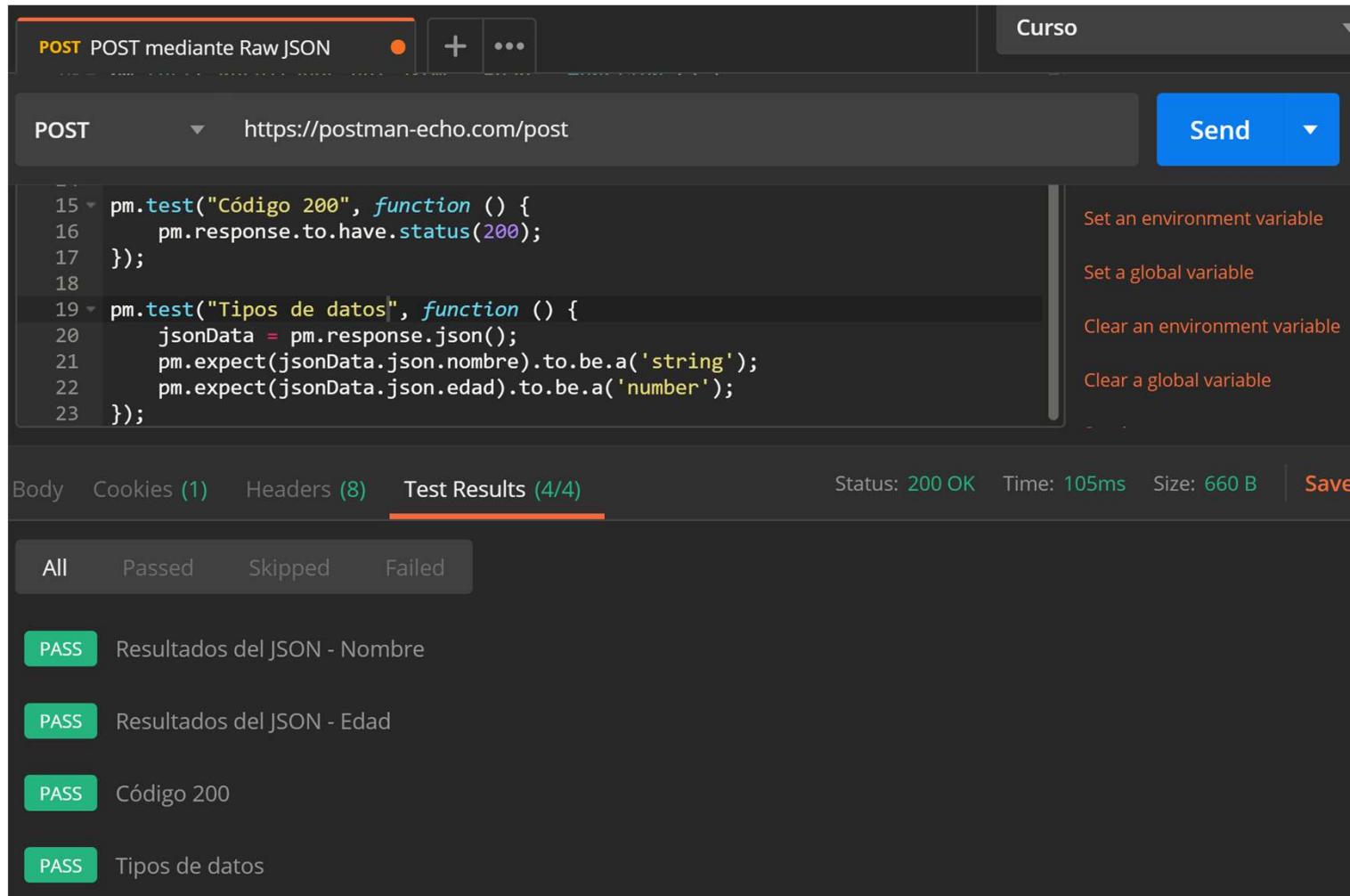
The screenshot shows a browser window displaying the Chai Assertion Library documentation at [chaijs.com/api/bdd/](http://chaijs.com/api/bdd/). The page has a navigation sidebar on the left with links like Introduction, Expect / Should (which is highlighted), Assert, Plugin Utilities, Online Test Suite, language chains, not, deep, nested, own, ordered, any, and all. The main content area has a title "BDD" and a section "API Reference" with a sub-section "Language Chains". The "Language Chains" section contains a list of methods: to, be, been, is, that, which, and, has. The right side of the page is dedicated to the `.a(type[, msg])` method, which is described as asserting that the target's type is equal to the given string `type`. It includes two bullet points for parameters: `@param {String} type` and `@param {String} msg _optional_`. Below this, there is a note about case insensitivity and a link to the `type-detect` project. A code example is provided in a box:

```
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:



The screenshot shows the Postman application interface. At the top, it says "POST POST mediante Raw JSON". The URL is "https://postman-echo.com/post". On the right, there is a "Send" button and a dropdown menu labeled "Curso". The main area contains the following code:

```
15 pm.test("Código 200", function () {
16     pm.response.to.have.status(200);
17 });
18
19 pm.test("Tipos de datos", function () {
20     jsonData = pm.response.json();
21     pm.expect(jsonData.json.nombre).to.be.a('string');
22     pm.expect(jsonData.json.edad).to.be.a('number');
23 });
```

To the right of the code, there is a context menu with four options: "Set an environment variable", "Set a global variable", "Clear an environment variable", and "Clear a global variable".

At the bottom, the "Test Results" tab is selected, showing 4/4 results. The results are:

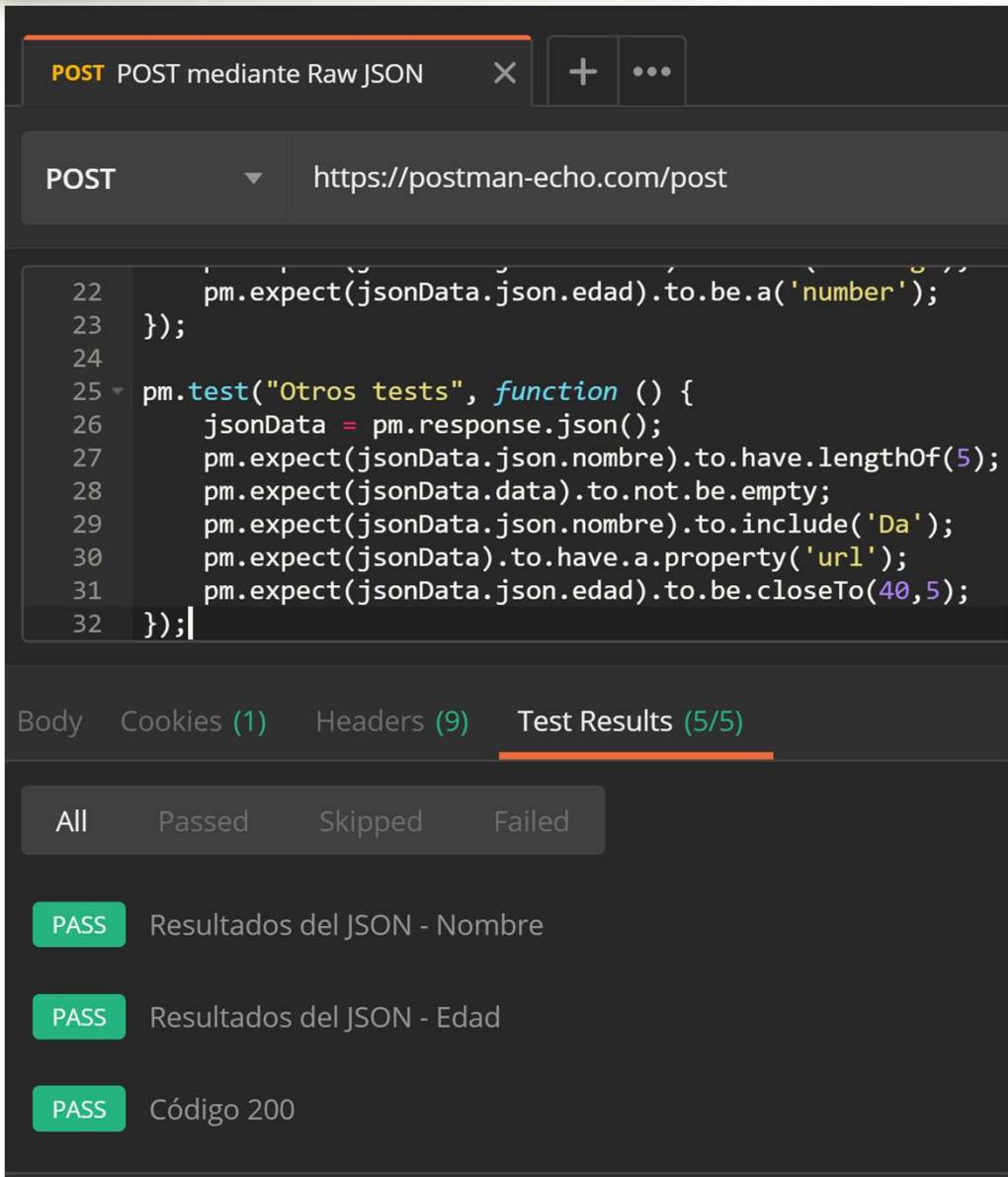
Status	Assertion
PASS	Resultados del JSON - Nombre
PASS	Resultados del JSON - Edad
PASS	Código 200
PASS	Tipos de datos

Below the results, status information is displayed: Status: 200 OK, Time: 105ms, Size: 660 B, and a Save button.



# Assertions

- Otro tipos de test:



The screenshot shows a Postman collection named "POST mediante Raw JSON". The request method is POST and the URL is <https://postman-echo.com/post>. The raw JSON body contains the following assertions:

```
22 pm.expect(jsonData.json.edad).to.be.a('number');
23 });
24
25 pm.test("Otros tests", function () {
26   jsonData = pm.response.json();
27   pm.expect(jsonData.json.nombre).to.have.lengthOf(5);
28   pm.expect(jsonData.data).to.not.be.empty;
29   pm.expect(jsonData.json.nombre).to.include('Da');
30   pm.expect(jsonData).to.have.a.property('url');
31   pm.expect(jsonData.json.edad).to.be.closeTo(40,5);
32 });|
```

Below the code, the "Test Results" tab is selected, showing 5/5 results. The results are:

- PASS Resultados del JSON - Nombre
- PASS Resultados del JSON - Edad
- PASS Código 200

# Testing XML y texto

- Podríamos tener respuestas en diferentes formatos:

httpbin.org

httpbin.org/#/Response\_formats

### Response formats

Returns responses in different data formats

GET	/brotli	Returns Brotli-encoded data.
GET	/deflate	Returns Deflate-encoded data.
GET	/deny	Returns page denied by robots.txt rules.
GET	/encoding/utf8	Returns a UTF-8 encoded body.
GET	/gzip	Returns GZip-encoded data.
GET	/html	Returns a simple HTML document.
GET	/json	Returns a simple JSON document.
GET	/robots.txt	Returns some robots.txt rules.
GET	/xml	Returns a simple XML document.

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

The screenshot shows the Postman application interface. At the top, there is a header bar with the title "GET XML". Below it, the main interface shows a "GET" request to the URL "{{urlhttpbin}}/xml". A red oval highlights this URL. The "Tests" tab is selected, showing the following JavaScript code:

```
1 var jsonObject = xml2json(responseBody);
2 console.log(jsonObject);
3
4
5 pm.test("Verificar valor", function () {
6     pm.expect(jsonObject.slideshow.$.title).to.eql('Sample Slide Show');
7 });
```

A red oval highlights the entire test script area. To the right of the script, a tooltip provides information about test scripts:

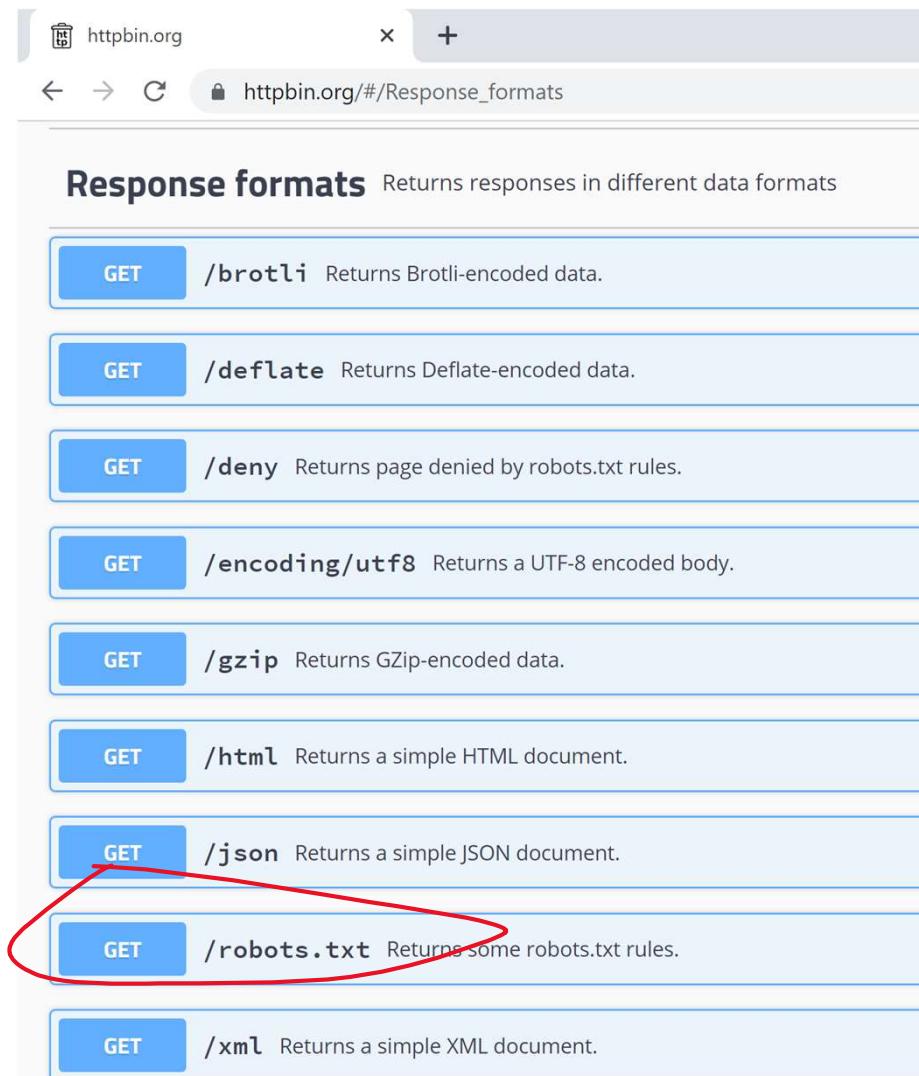
Test scripts are written in JavaScript, and are run after the response is received.  
Learn more [about tests scripts](#)

SNIPPETS  
Response body: Convert XML body to a JSON Object  
Use Tiny Validator for JSON data

At the bottom of the interface, the status bar shows "Status: 200 OK Time: 341ms Size: 751 B". The "Pretty" view is selected, showing the XML response:

```
1 <?xml version='1.0' encoding='us-ascii'?>
2 <!-- A SAMPLE set of slides -->
3 <slideshow>
```

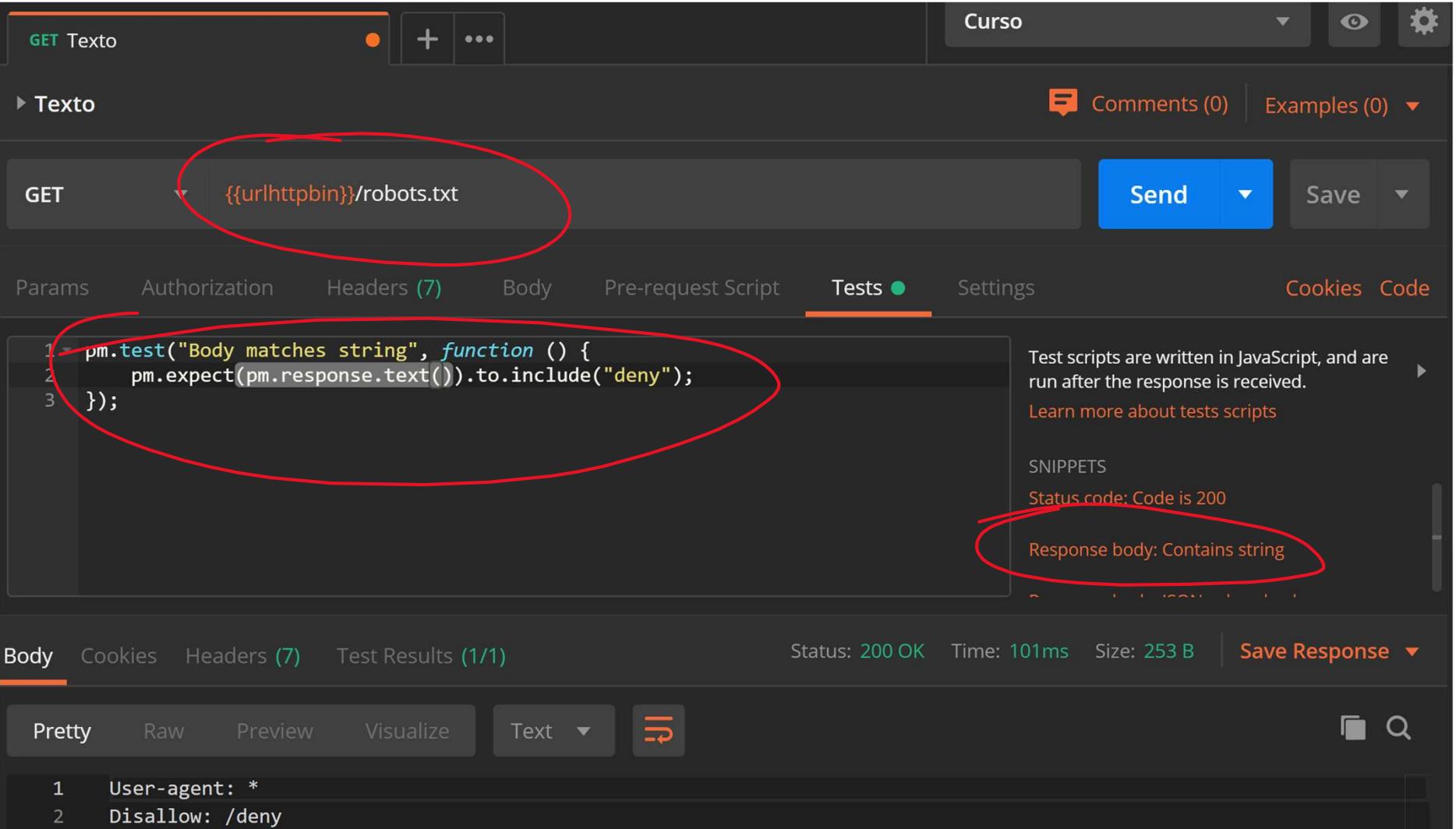
- Si recibimos texto:



The screenshot shows a web browser window with the URL `httpbin.org/#/Response_formats`. The page title is "Response formats" and it says "Returns responses in different data formats". There is a list of endpoints, each with a "GET" method and a description:

- `/brotli` Returns Brotli-encoded data.
- `/deflate` Returns Deflate-encoded data.
- `/deny` Returns page denied by robots.txt rules.
- `/encoding/utf8` Returns a UTF-8 encoded body.
- `/gzip` Returns GZip-encoded data.
- `/html` Returns a simple HTML document.
- `/json` Returns a simple JSON document. (This entry is circled in red.)
- `/robots.txt` Returns some robots.txt rules. (This entry is circled in red.)
- `/xml` Returns a simple XML document.

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



GET Texto

Curso

Comments (0) Examples (0)

Send Save

GET {{urlhttpbin}}/robots.txt

Params Authorization Headers (7) Body Pre-request Script Tests (1) Settings Cookies Code

```
1 pm.test("Body matches string", function () {  
2     pm.expect(pm.response.text()).to.include("deny");  
3 });
```

Test scripts are written in JavaScript, and are run after the response is received.  
Learn more about tests scripts

SNIPPETS  
Status code: Code is 200  
Response body: Contains string

Body Cookies Headers (7) Test Results (1/1) Status: 200 OK Time: 101ms Size: 253 B Save Response

Pretty Raw Preview Visualize Text

```
1 User-agent: *  
2 Disallow: /deny
```

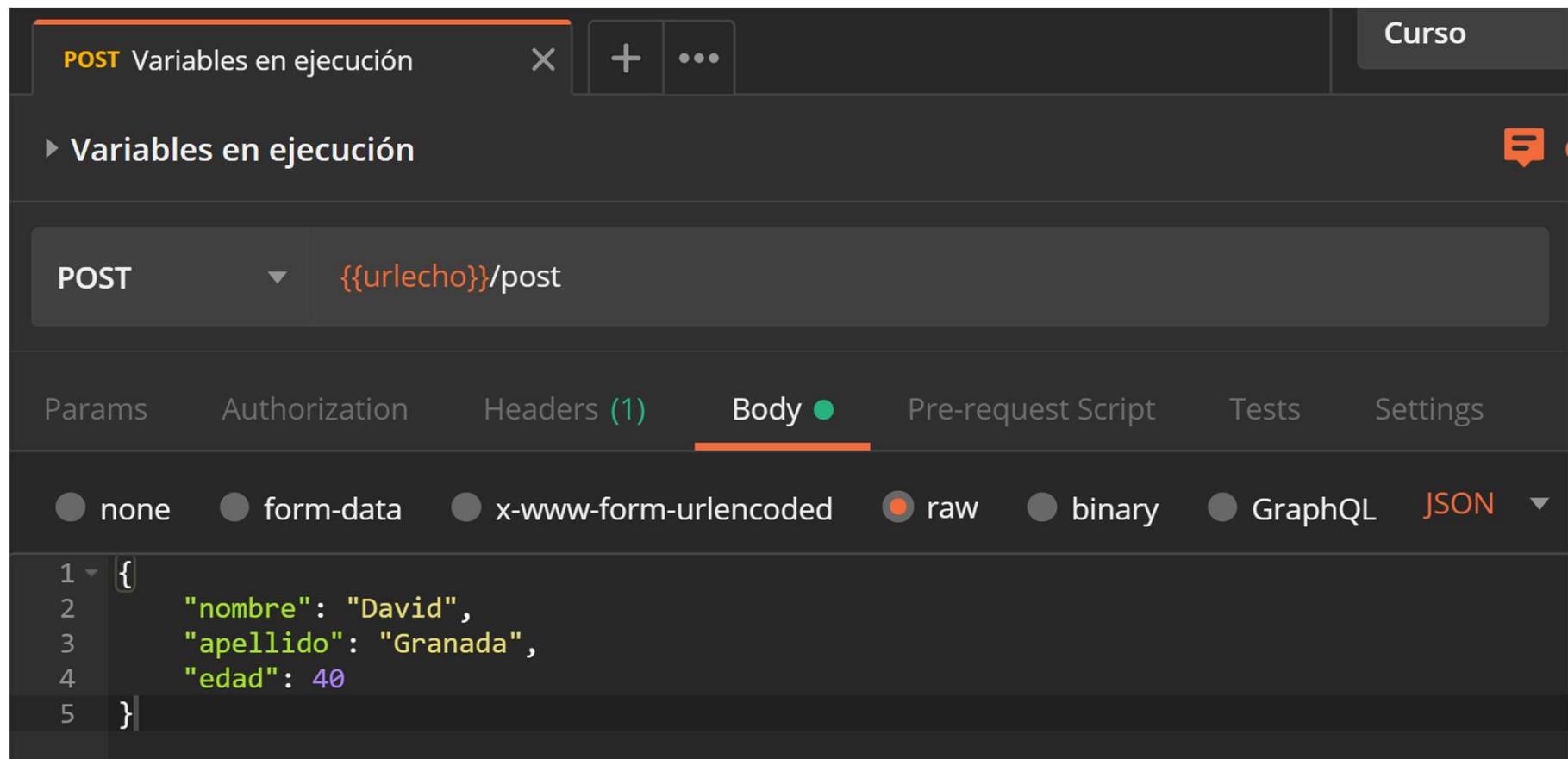
# Pre-request Script





# Variables en ejecución

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



The screenshot shows the Postman application interface. The title bar says "POST Variables en ejecución". The main view shows a single item under "Variables en ejecución". The request details are as follows:

- Method: POST
- URL: {{urlecho}}/post
- Body tab is selected.
- Body type: raw (JSON selected)
- Body content:

```
1 {  
2   "nombre": "David",  
3   "apellido": "Granada",  
4   "edad": 40  
5 }
```

# Variables en ejecución

The screenshot shows the Postman application interface. On the left, the sidebar lists collections: History, Collections (highlighted), APIs, + New Collection, Collection1 (40 requests), Servicios GET, Servicios POST, POST mediante Raw, POST mediante formulario, POST mediante Raw JSON, POST pedido array, POST Variables en ejecución (highlighted), DELETE, UPDATE y PUT, Variables Dinámicas, Autorizaciones, and Utilidades. At the bottom of the sidebar, there are three icons: a document with a magnifying glass, a document with a plus sign, and a question mark.

The main workspace shows a POST request titled "Variables en ejecución". The URL field contains "{{urlecho}}/post", which is circled in red. The "Tests" tab is selected, showing the following JavaScript code:

```
1 var jsonData = pm.response.json();
2 pm.environment.set("nombreEntorno", jsonData.json.nombre);
3 pm.globals.set("edadGlobal", jsonData.json.edad);
4
5 console.log(pm.environment.get("nombreEntorno"));
6 console.log(pm.globals.get("edadGlobal"));
```

The status bar at the bottom indicates: Status: 200 OK, Time: 100ms, Size: 660 B, Save Response ▾, Body, Cookies (1), Headers (8), Test Results, Pretty, Raw, Preview, Visualize, JSON ▾, and a settings icon.

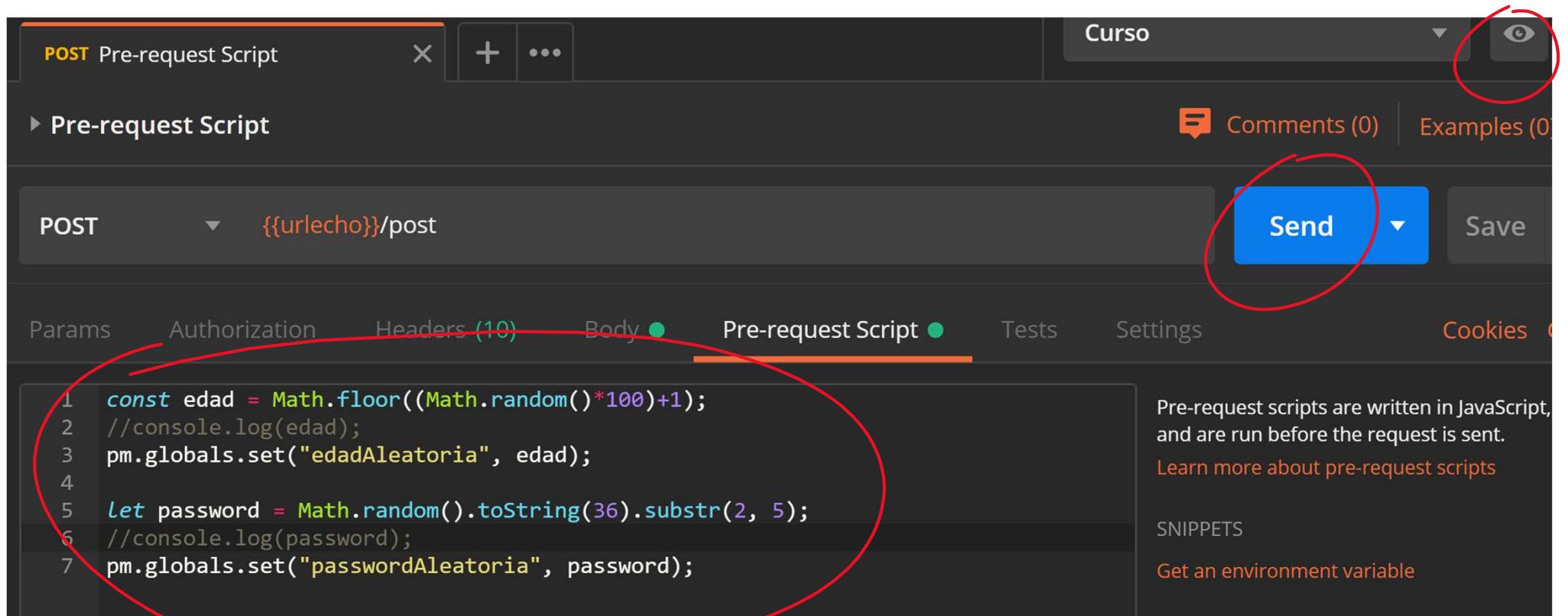
On the right side of the interface, there is a sidebar with the title "Curso" and sections for Comments (0) and Examples (0). A red circle highlights the "Examples (0)" button. Below the sidebar, there is a snippet panel with the following options:

- SNIPPETS
  - Get an environment variable
  - Get a global variable
  - Get a variable
  - Set an environment variable



# Pre-request Script

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



The screenshot shows the Postman interface for a POST request to `{urlecho}/post`. A red circle highlights the 'Send' button in the top right corner. Another red circle highlights the 'Pre-request Script' tab in the navigation bar. A large red oval encloses the script code in the bottom-left panel.

**POST** Pre-request Script

Pre-request Script

Comments (0) Examples (0)

Send Save

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

```
1 const edad = Math.floor((Math.random()*100)+1);
2 //console.log(edad);
3 pm.globals.set("edadAleatoria", edad);
4
5 let password = Math.random().toString(36).substr(2, 5);
6 //console.log(password);
7 pm.globals.set("passwordAleatoria", password);
```

Pre-request scripts are written in JavaScript, and are run before the request is sent.  
Learn more about pre-request scripts

SNIPPETS  
Get an environment variable  
Get a global variable



# Pre-request Script

- Estas variables las podemos usar luego en la petición:

POST {{urlecho}}/post

Params Authorization Headers (10) Body Pre-request Script

none form-data x-www-form-urlencoded raw binary

```
1 {  
2   "edad": "{{edadAleatoria}}",  
3   "password": "{{passwordAleatoria}}",  
4   "cantidad": "{$randomInt}",  
5   "código": "{$guid}"  
6 }
```

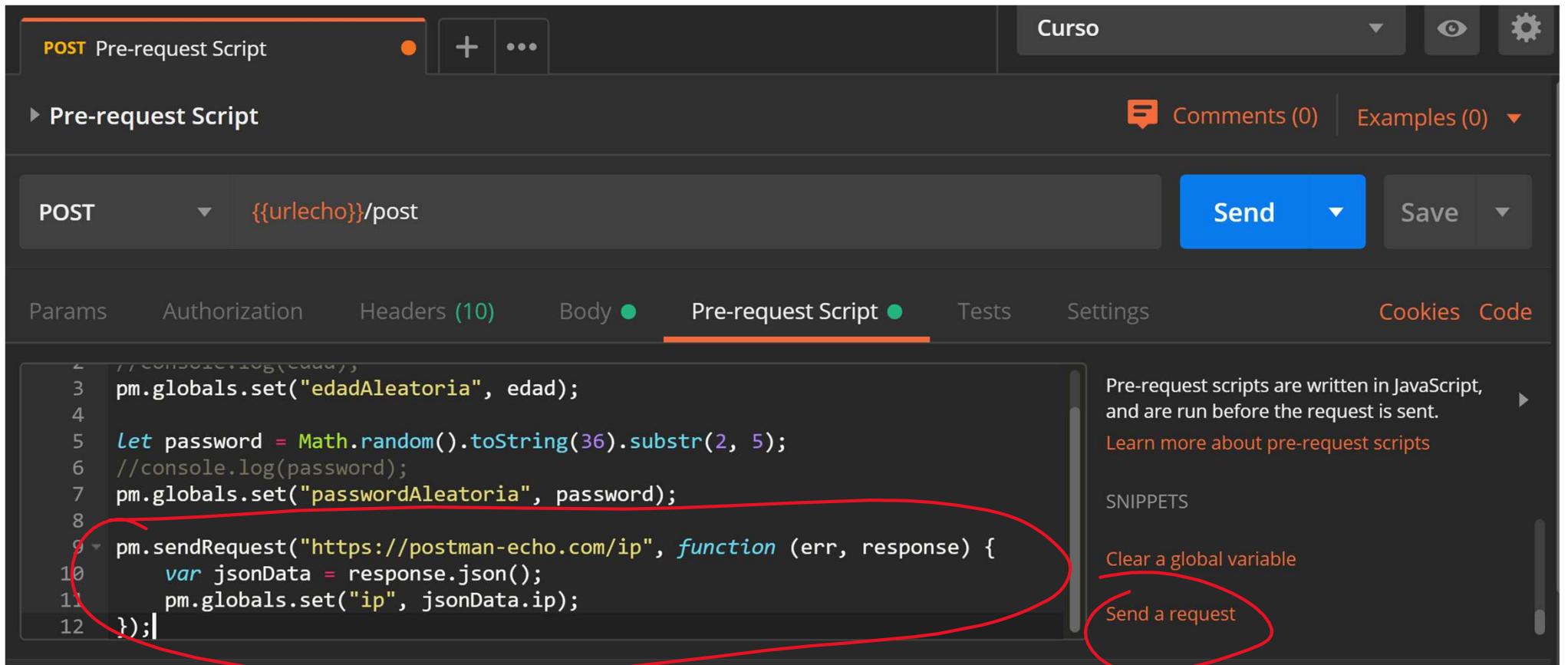
Pretty Raw Preview Visualize JSON 

```
1 [{  
2   "args": {},  
3   "data": {  
4     "edad": 78,  
5     "password": "rugy9",  
6     "cantidad": 222,  
7     "código": "7442c36f-f964-47e1-a5d2-57d5ff240016"  
8   }]
```



# Pre-request Script

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



The screenshot shows the Postman interface with a request configuration. The method is set to POST, the URL is {{urlecho}}/post, and the Pre-request Script tab is selected. The script contains the following code:

```
3 pm.globals.set("edadAleatoria", edad);
4
5 let password = Math.random().toString(36).substr(2, 5);
6 //console.log(password);
7 pm.globals.set("passwordAleatoria", password);
8
9 pm.sendRequest("https://postman-echo.com/ip", function (err, response) {
10     var jsonData = response.json();
11     pm.globals.set("ip", jsonData.ip);
12});
```

A red oval highlights the final part of the script where a request is sent to "https://postman-echo.com/ip". A tooltip on the right side of the interface provides information about pre-request scripts:

Pre-request scripts are written in JavaScript, and are run before the request is sent.  
Learn more about pre-request scripts

SNIPPETS

Clear a global variable  
Send a request



# Pre-request Script

- Dichos valores podemos usarlos en la petición actual:

POST Pre-request Script

▶ Pre-request Script

POST {{urlecho}}/post

Params Authorization Headers (10) Body ●

none form-data x-www-form-urlencoded

```
1 {  
2   "edad": "{{edadAleatoria}}",  
3   "password": "{{passwordAleatoria}}",  
4   "cantidad": "{{$randomInt}}",  
5   "código": "{$guid}}",  
6   "ip": "{{ip}}"  
7 }
```

Status: 200 OK

Body Cookies (1) Headers (8) Test Results

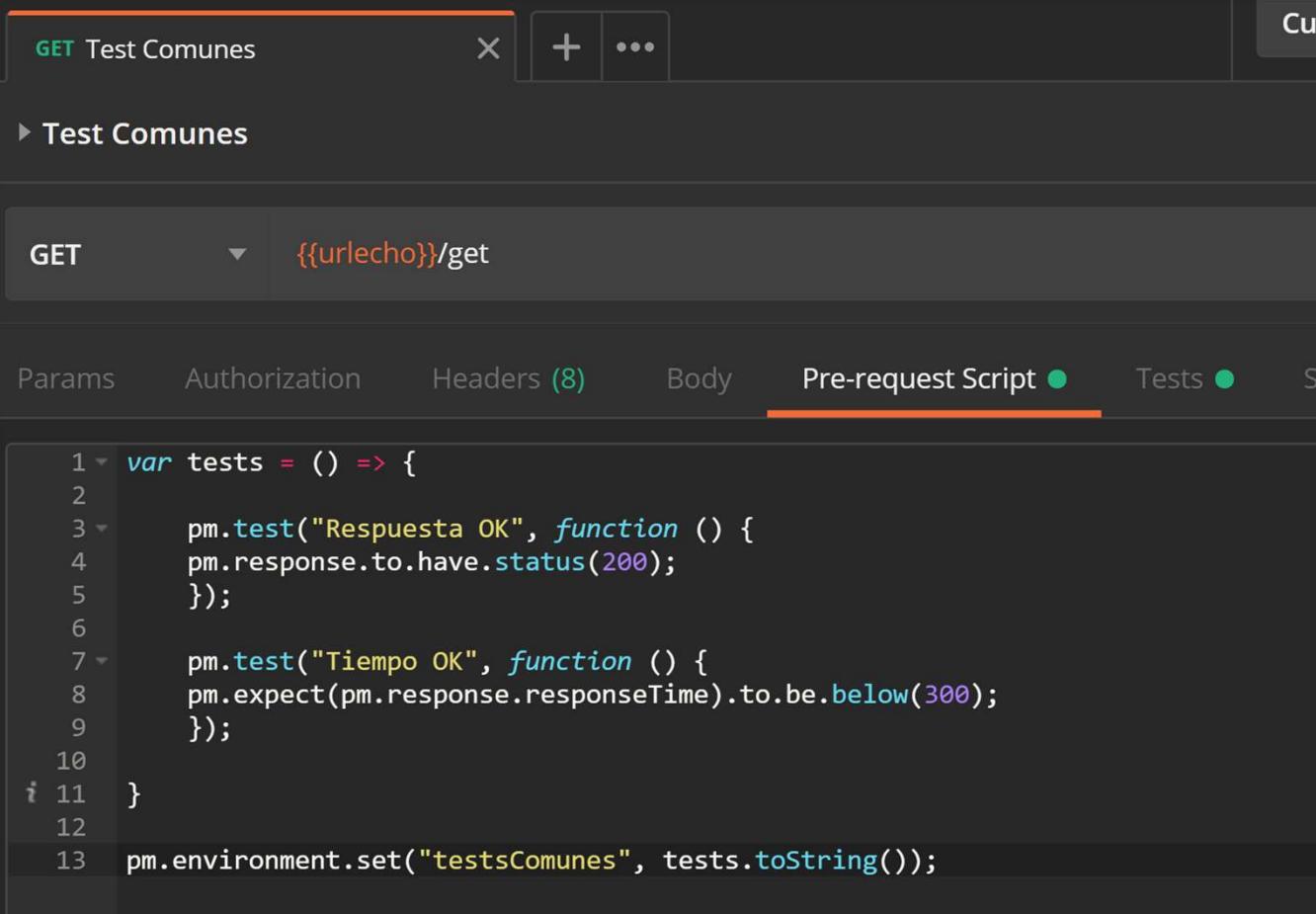
Pretty Raw Preview Visualize JSON

```
1 {  
2   "args": {},  
3   "data": {  
4     "edad": 34,  
5     "password": "rw8ln",  
6     "cantidad": 42,  
7     "código": "5f2db950-cdad-443f-a0d9-7dfdb73806d8",  
8     "ip": "193.147.77.168"  
9   }  
10 }
```



# Pre-request Script

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



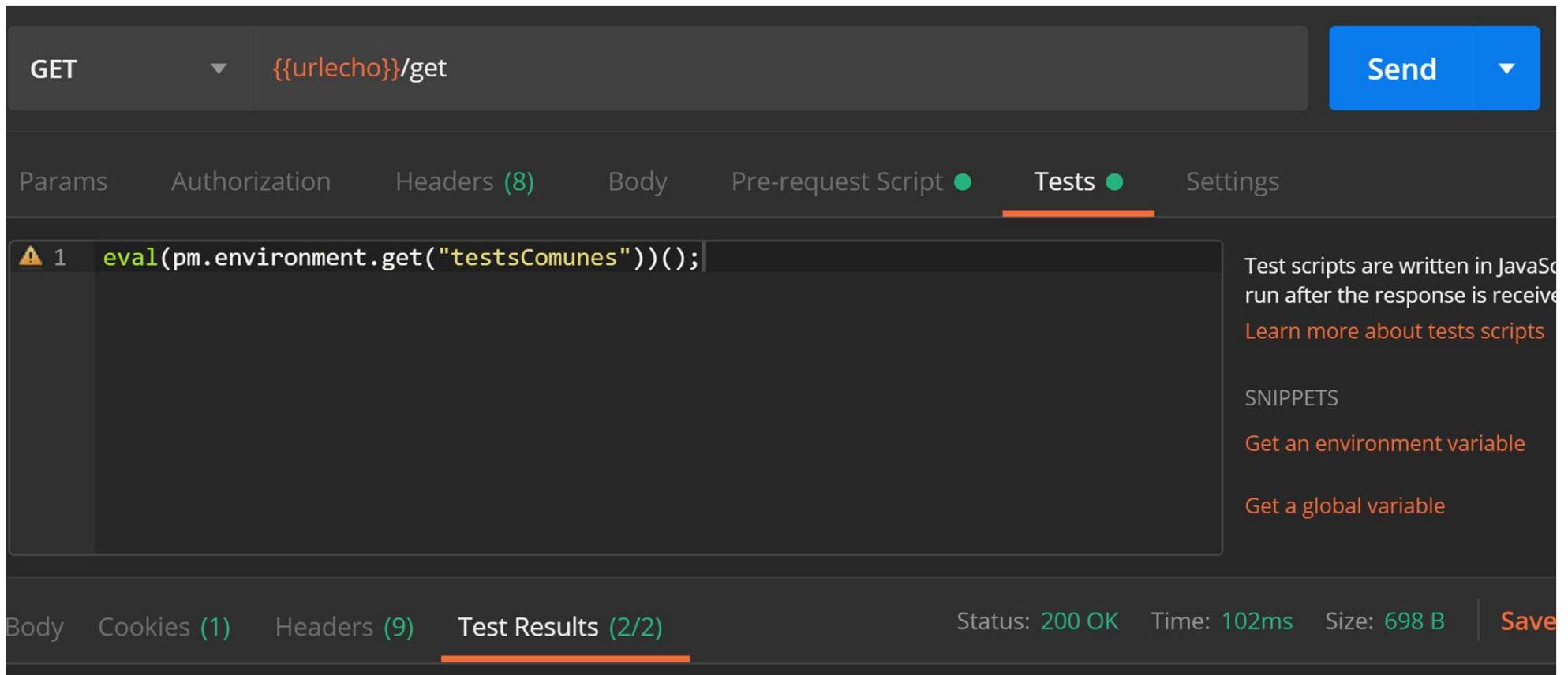
The screenshot shows the Postman interface with a dark theme. At the top, there's a header bar with 'GET Test Comunes' and a '+' button. Below it, a section titled 'Test Comunes' is expanded. Underneath, a 'GET' request is shown with the URL {{urlecho}}/get. The main content area has tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script' (which is highlighted with an orange underline), 'Tests', and 'Se'. The 'Pre-request Script' tab contains the following code:

```
1 var tests = () => {
2
3   pm.test("Respuesta OK", function () {
4     pm.response.to.have.status(200);
5   });
6
7   pm.test("Tiempo OK", function () {
8     pm.expect(pm.response.responseTime).to.be.below(300);
9   });
10
11 }
12
13 pm.environment.set("testsComunes", tests.toString());
```



# Pre-request Script

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



The screenshot shows the Postman interface with a dark theme. At the top, there's a header bar with 'GET' and a URL placeholder {{urlecho}}/get. To the right is a 'Send' button. Below the header are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Tests' tab is currently selected, indicated by an orange underline. In the main content area, under the 'Tests' tab, there is a code editor containing the following JavaScript code:

```
⚠ 1 eval(pm.environment.get("testsComunes"))();
```

To the right of the code editor, there is a tooltip with the following text:

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

Below the code editor, there are three links:

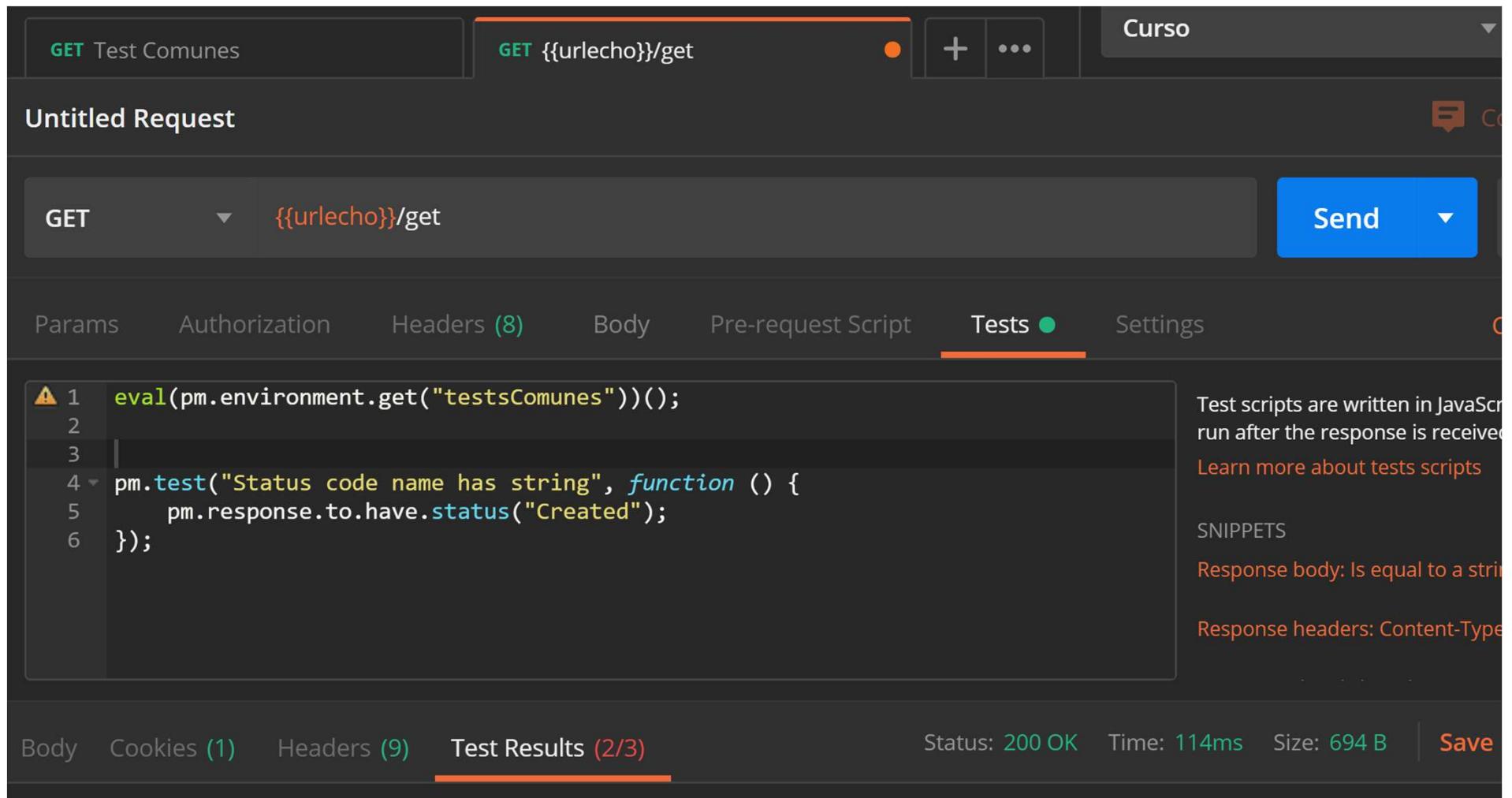
- SNIPPETS
- [Get an environment variable](#)
- [Get a global variable](#)

At the bottom of the interface, there are tabs for 'Body', 'Cookies (1)', 'Headers (9)', and 'Test Results (2/2)'. The 'Test Results' tab is also underlined in orange. On the far right, status information is displayed: 'Status: 200 OK', 'Time: 102ms', 'Size: 698 B', and a 'Save' button.



# Pre-request Script

- Y podemos re-utilizarlos en cualquier otra petición:



The screenshot shows the Postman interface with the following details:

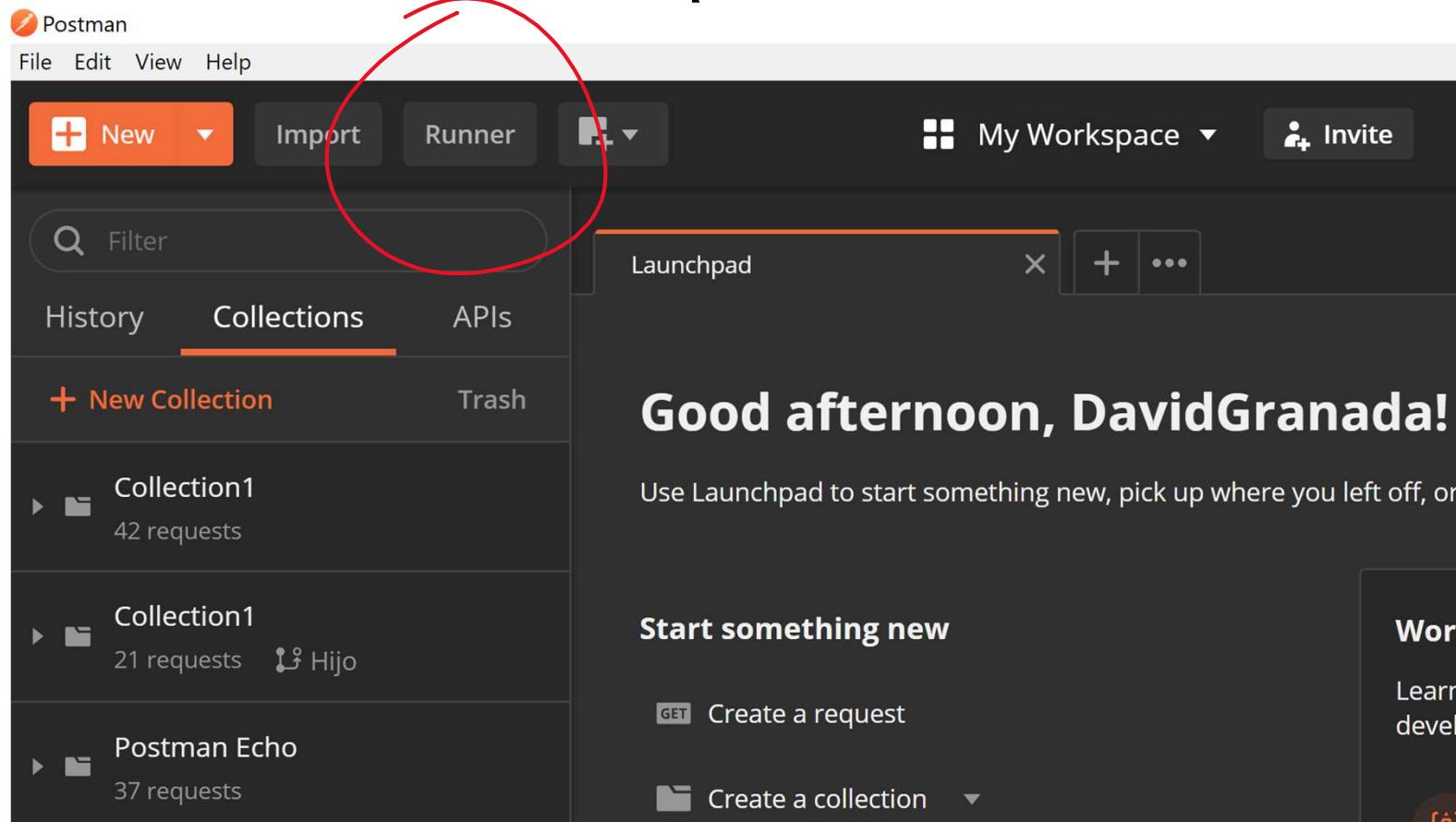
- Request Header:** GET {{urlecho}}/get
- Method:** GET
- URL:** {{urlecho}}/get
- Send Button:** Send
- Test Tab:** The "Tests" tab is selected.
- Test Script Content:**

```
⚠ 1 eval(pm.environment.get("testsComunes"))();  
2  
3  
4 pm.test("Status code name has string", function () {  
5     pm.response.to.have.status("Created");  
6 });
```
- Test Results:** Status: 200 OK, Time: 114ms, Size: 694 B, Save button

# Runner tool



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

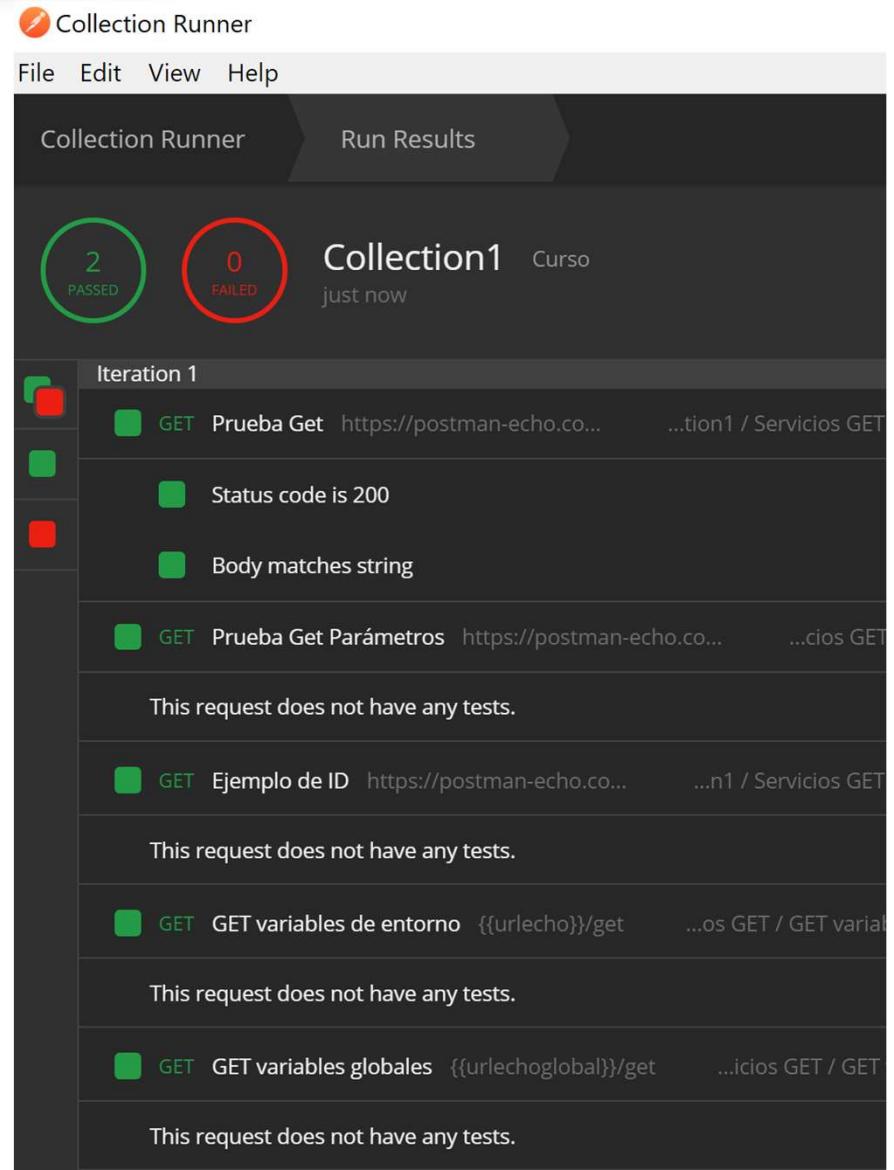


# Runner

- Seleccionamos la carpeta y ejecutamos los tests:

The screenshot shows the 'Collection Runner' interface. At the top, there's a navigation bar with 'File', 'Edit', 'View', and 'Help'. To the right is a dropdown menu labeled 'My Workspace'. The main area is titled 'Collection Runner' and has a sub-header 'Choose a collection or folder'. A search bar says 'Search for a collection or folder'. Below it is a list of items under 'Servicios GET': 'Prueba Get', 'Prueba Get Parámetros', 'Ejemplo de ID', 'GET variables de entorno', and 'GET variables globales'. On the right side, there's a 'RUN ORDER' section listing five tests, each with a checked checkbox and an orange 'GET' icon: 'Prueba Get', 'Prueba Get Parámetros', 'Ejemplo de ID', 'GET variables de entorno', and 'GET variables globales'. Below these sections are configuration fields: 'Environment' set to 'No Environment', 'Iterations' set to '1', 'Delay' set to '0 ms', and a 'Data' button with 'Select File'. At the bottom are several checkboxes: 'Save responses' (unchecked), 'Keep variable values' (checked), 'Run collection without using stored cookies' (unchecked), and 'Save cookies after collection run' (checked). A large blue button at the bottom center says 'Run Collection1'.

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



The screenshot shows the Postman Collection Runner interface. At the top, there's a navigation bar with 'Collection Runner' and 'Run Results'. Below it, a summary for 'Collection1' shows '2 PASSED' and '0 FAILED'. The main area displays 'Iteration 1' with three test cases:

- GET Prueba Get https://postman-echo.co... (Status code is 200, Body matches string) - Passed
- GET Prueba Get Parámetros https://postman-echo.co... (This request does not have any tests.) - Failed
- GET Ejemplo de ID https://postman-echo.co... (This request does not have any tests.) - Failed

Below these, there are two more sections for 'GET variables de entorno' and 'GET variables globales', both of which also state 'This request does not have any tests.'

# Flujos de trabajo





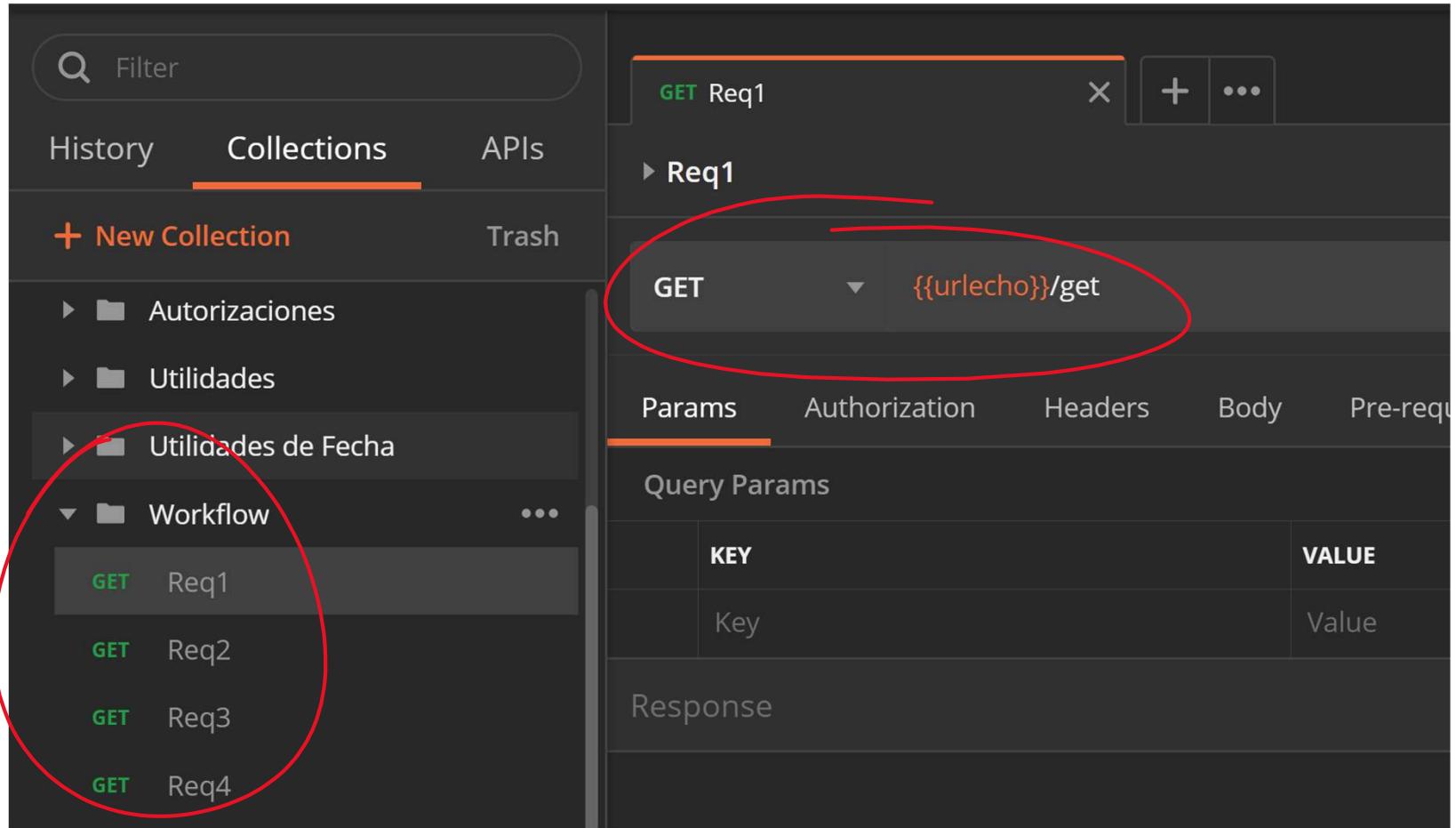
## Flujos de trabajo

- 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:



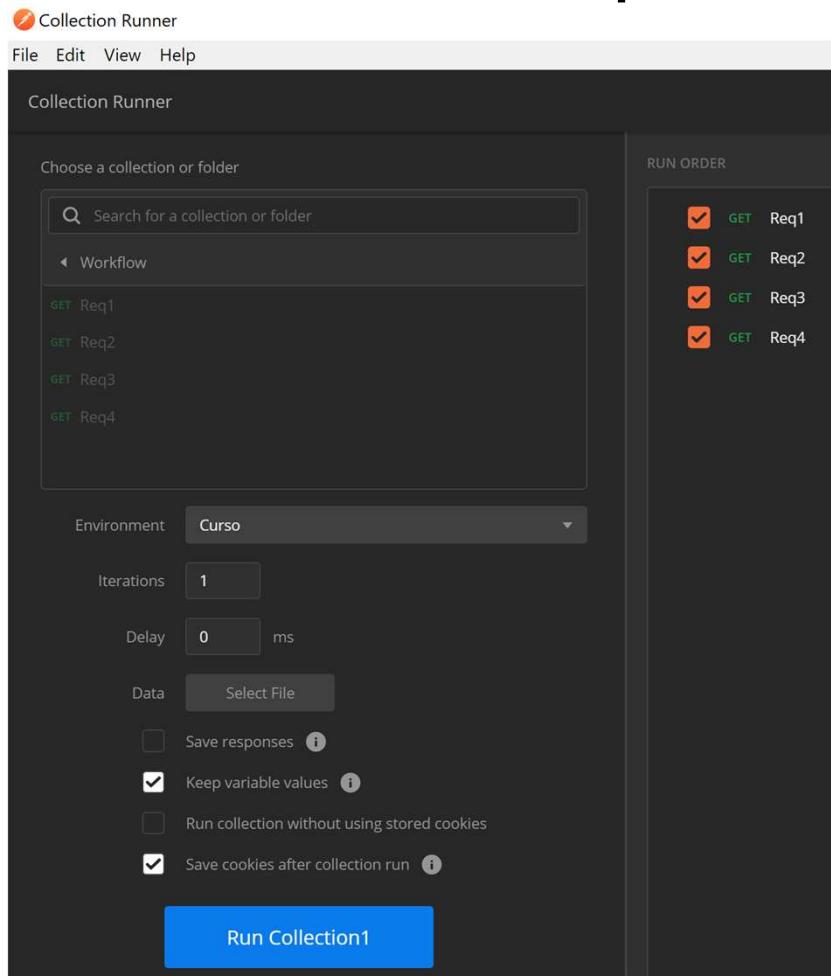
The screenshot shows a user interface for managing API requests. On the left, there's a sidebar with tabs: History, Collections (which is active), APIs, and Trash. Below these are buttons for New Collection and a list of existing collections: Autorizaciones, Utilidades, Utilidades de Fecha, and Workflow. The Workflow folder is expanded, showing four requests: Req1, Req2, Req3, and Req4. Each request is listed with its method (GET) and URL path. To the right of the sidebar, a detailed view of the 'Req1' request is shown. The URL path is highlighted with a red oval and contains the placeholder {{urlecho}}/get. The interface includes tabs for Params, Authorization, Headers, Body, and Pre-request scripts. Under the Params tab, there's a table for Query Params with columns KEY and VALUE, showing a single entry: Key and Value. Below this is a section for Response.

KEY	VALUE
Key	Value



# Flujos de trabajo

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

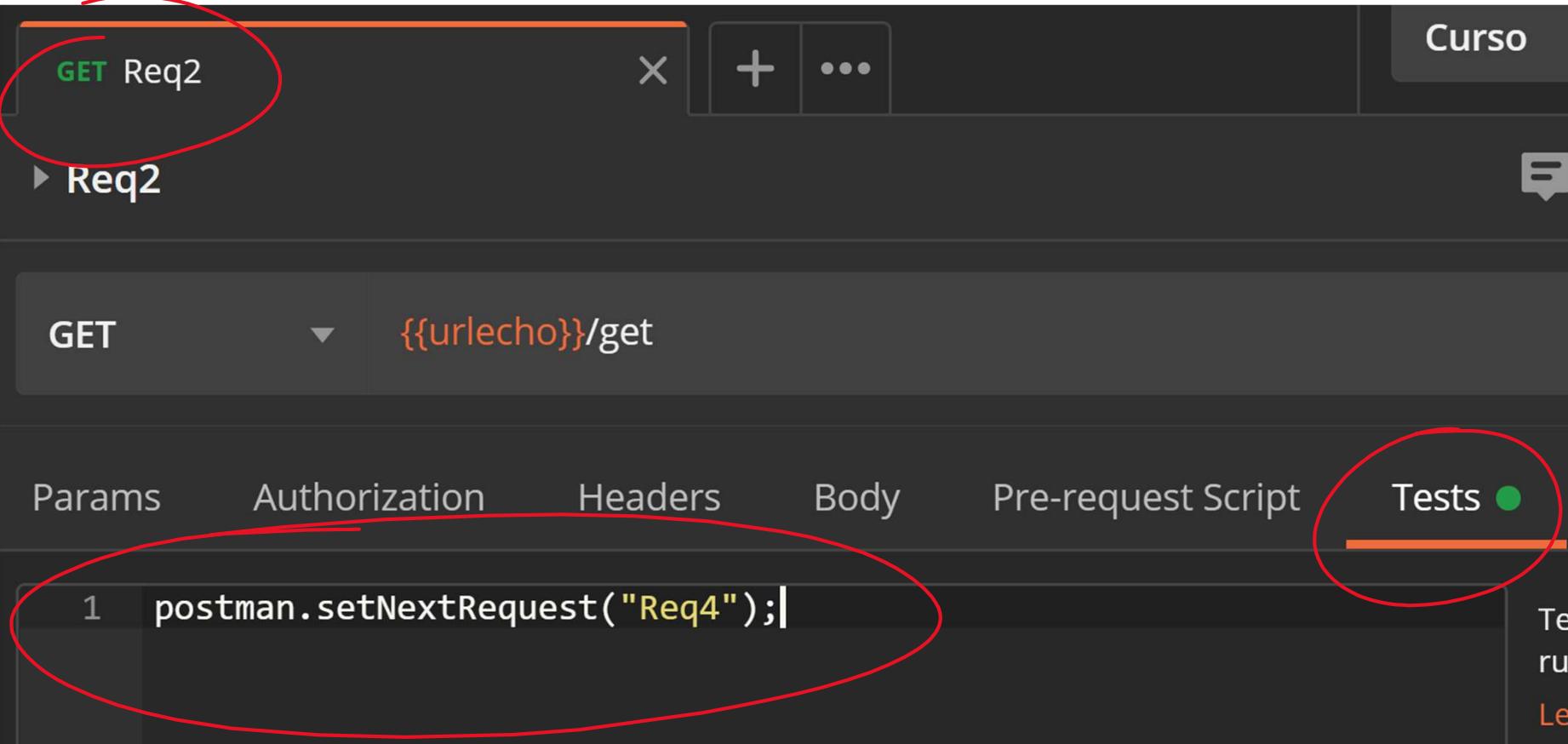


The screenshot shows the 'Run Results' page for 'Collection1' run 'just now'. It displays two summary circles: one green for 'PASSED' (0) and one red for 'FAILED' (0). The main area, 'Iteration 1', lists four requests: 'Req1', 'Req2', 'Req3', and 'Req4'. Each request has a green icon and the text 'This request does not have any tests.' To the right of the requests, it shows they belong to 'Collection1 / Workflow'.



# Flujos de trabajo

- Podemos cambiar el orden de esta forma:



The screenshot shows the Postman interface with a request labeled "Req2" highlighted by a red oval. Below it, another request "Req2" is shown with its details: method "GET" and URL "{{urlecho}}/get". A red arrow points from the "Req2" label to the "Tests" tab, which contains the following code:

```
1 postman.setNextRequest("Req4");|
```

The "Tests" tab is also highlighted by a red oval. The "Params", "Authorization", "Headers", "Body", "Pre-request Script", and "Tests" tabs are visible at the top of the request details panel.

# 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):

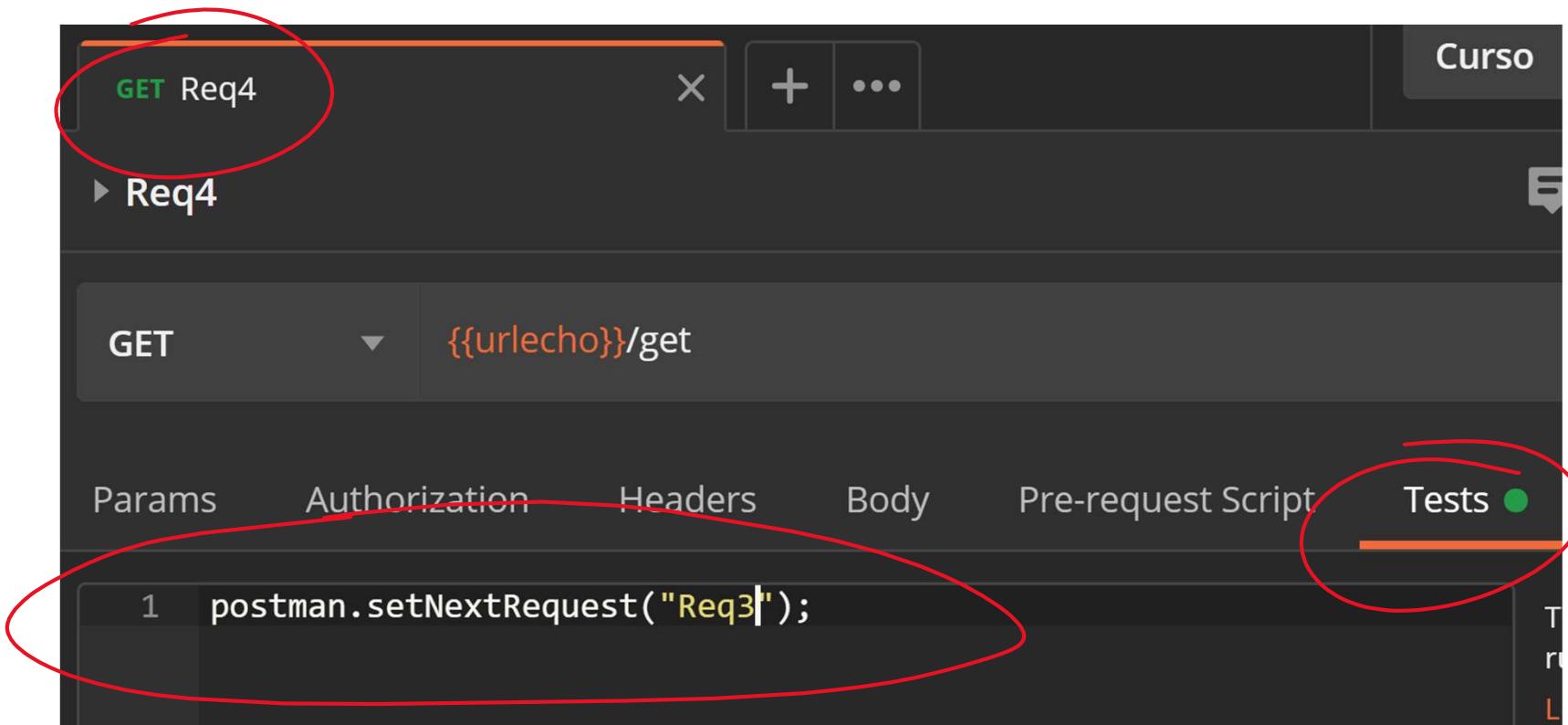
The screenshot shows the Collection Runner interface with the following details:

- Collection Runner** tab is active.
- Run Results** tab is visible.
- My Workspace** dropdown is open.
- Collection1** is selected, with **No Environment**.
- The run was performed **just now**.
- Iteration 1** results:
  - Req1**: GET {{urlecho}}/get, Status: 200 OK, Time: 99 ms, Size: 573 B.
  - Req2**: GET {{urlecho}}/get, Status: 500 Internal Server Error, Time: 97 ms, Size: 575 B. A note says: "This request does not have any tests."
  - Req4**: GET {{urlecho}}/get, Status: 200 OK, Time: 96 ms, Size: 696 B. A note says: "This request does not have any tests."
- Run Summary**, **Export Results**, and **Retry** buttons are present.



# Flujos de trabajo

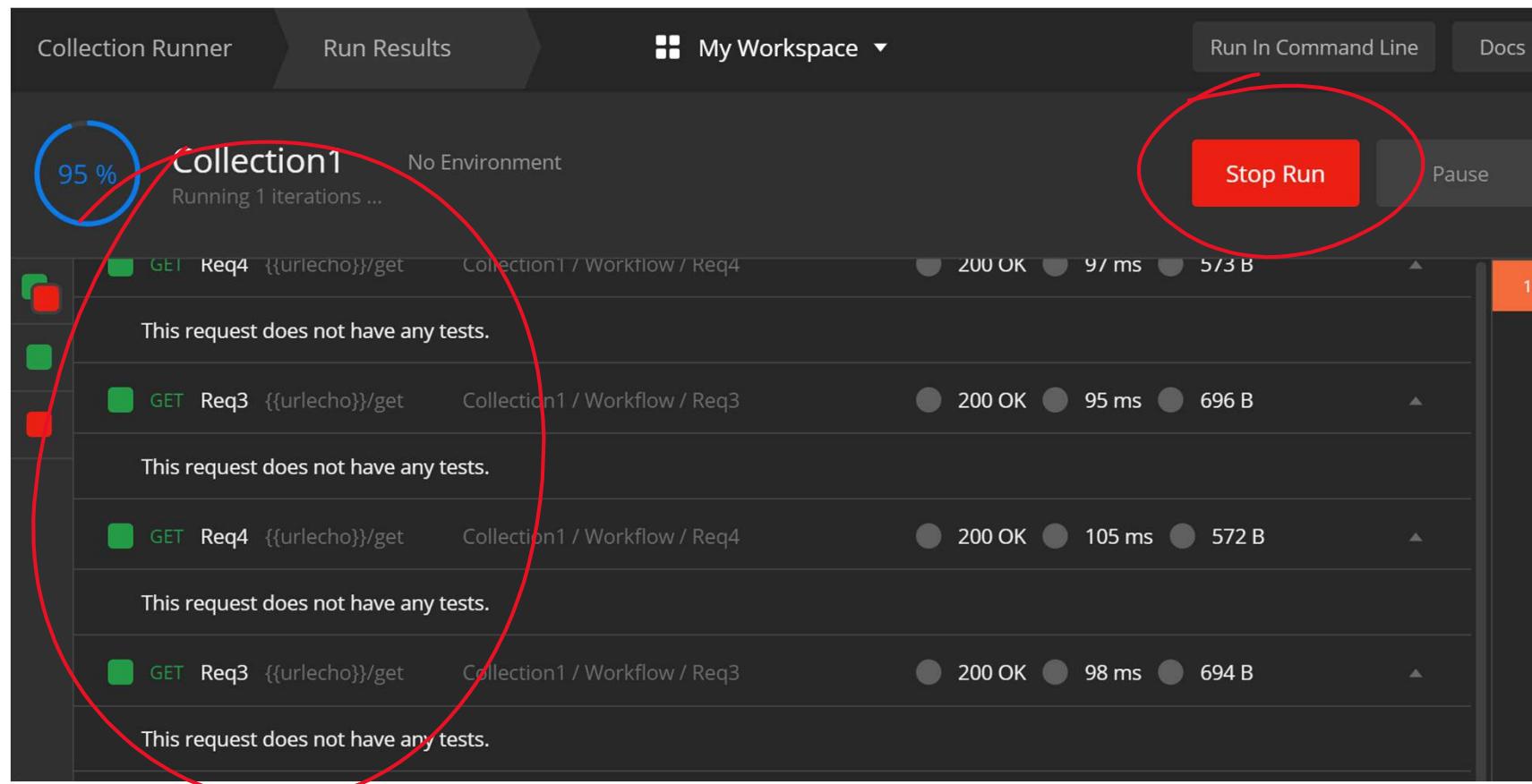
- Podemos indicar que después del Req 4 se ejecute el Req3:





# Flujos de trabajo

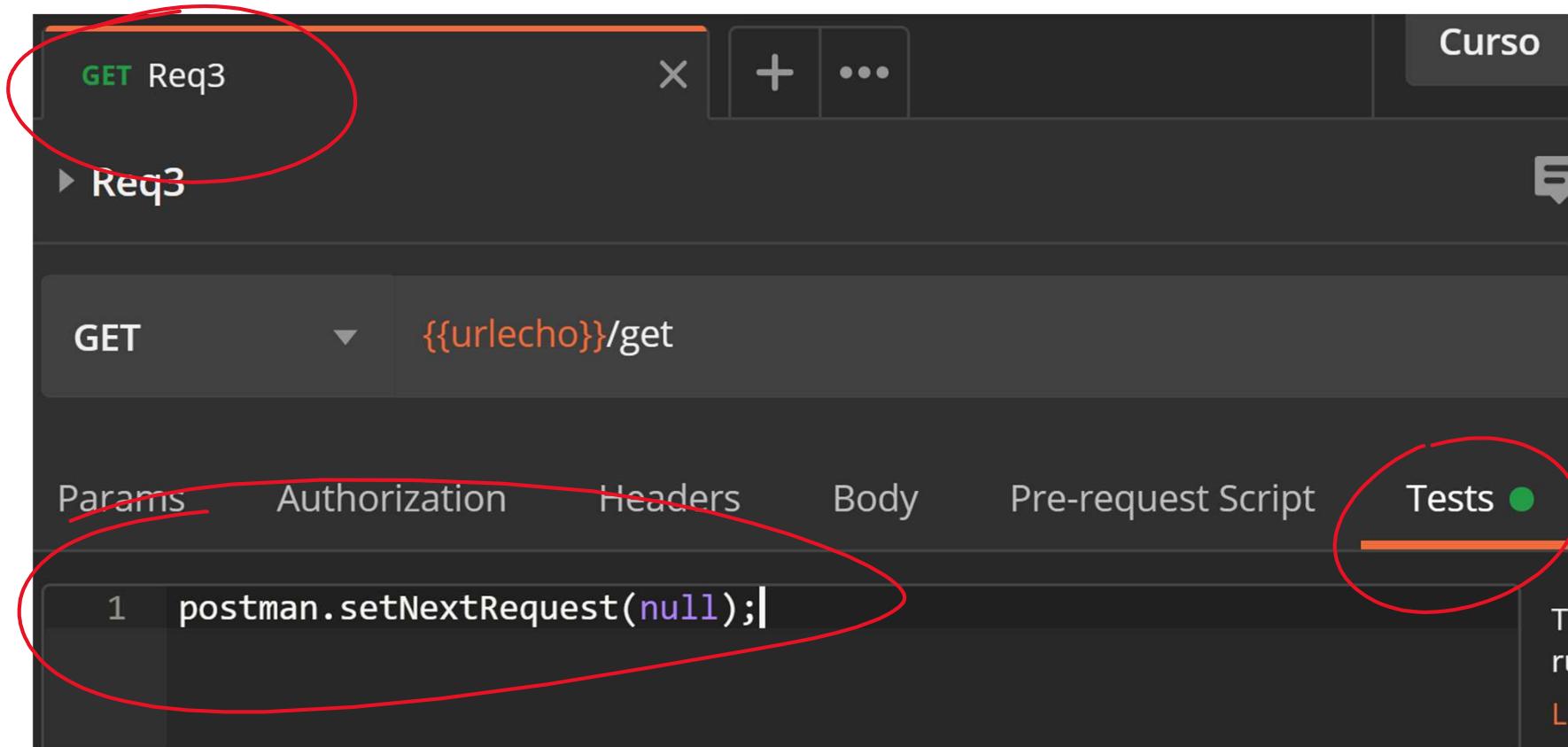
- Al ejecutar el Runner se crea un bucle infinito:





# 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:

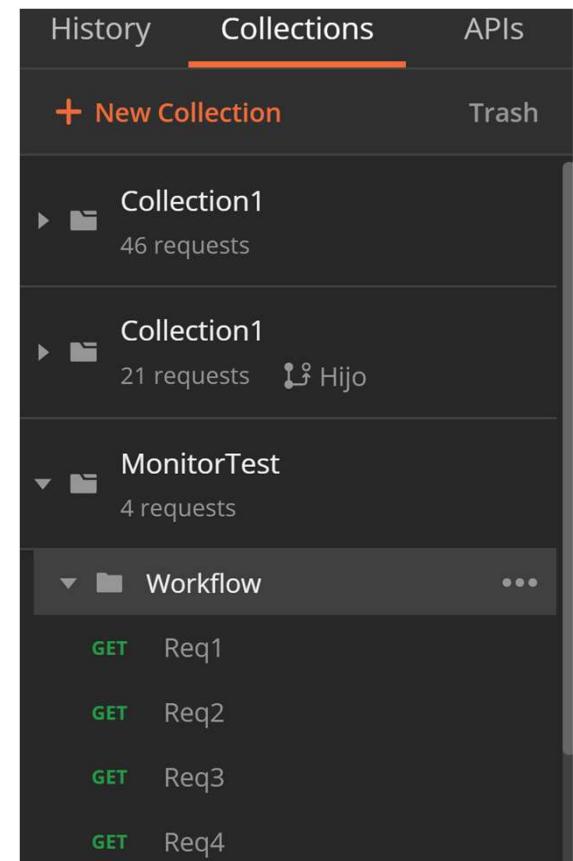
The screenshot shows the Collection Runner interface with the following details:

- Collection Runner** tab is active.
- Run Results** tab is visible.
- My Workspace** dropdown shows "Collection1" selected.
- Collection1** is listed with "No Environment" and "just now".
- Iteration 1** is shown with four requests:
  - Req1**: GET {{urlecho}}/get, Collection1 / Workflow / Req1. Status: 200 OK, 141 ms, 697 B.
  - Req2**: GET {{urlecho}}/get, Collection1 / Workflow / Req2. Status: 200 OK, 136 ms, 575 B.
  - Req4**: GET {{urlecho}}/get, Collection1 / Workflow / Req4. Status: 200 OK, 138 ms, 575 B.
  - Req3**: GET {{urlecho}}/get, Collection1 / Workflow / Req3. Status: 200 OK, 95 ms, 574 B.
- Each request row has a red circle around it, and a large red circle encloses the entire list of requests.
- A blue **Retry** button is highlighted with a red oval.

# Monitores



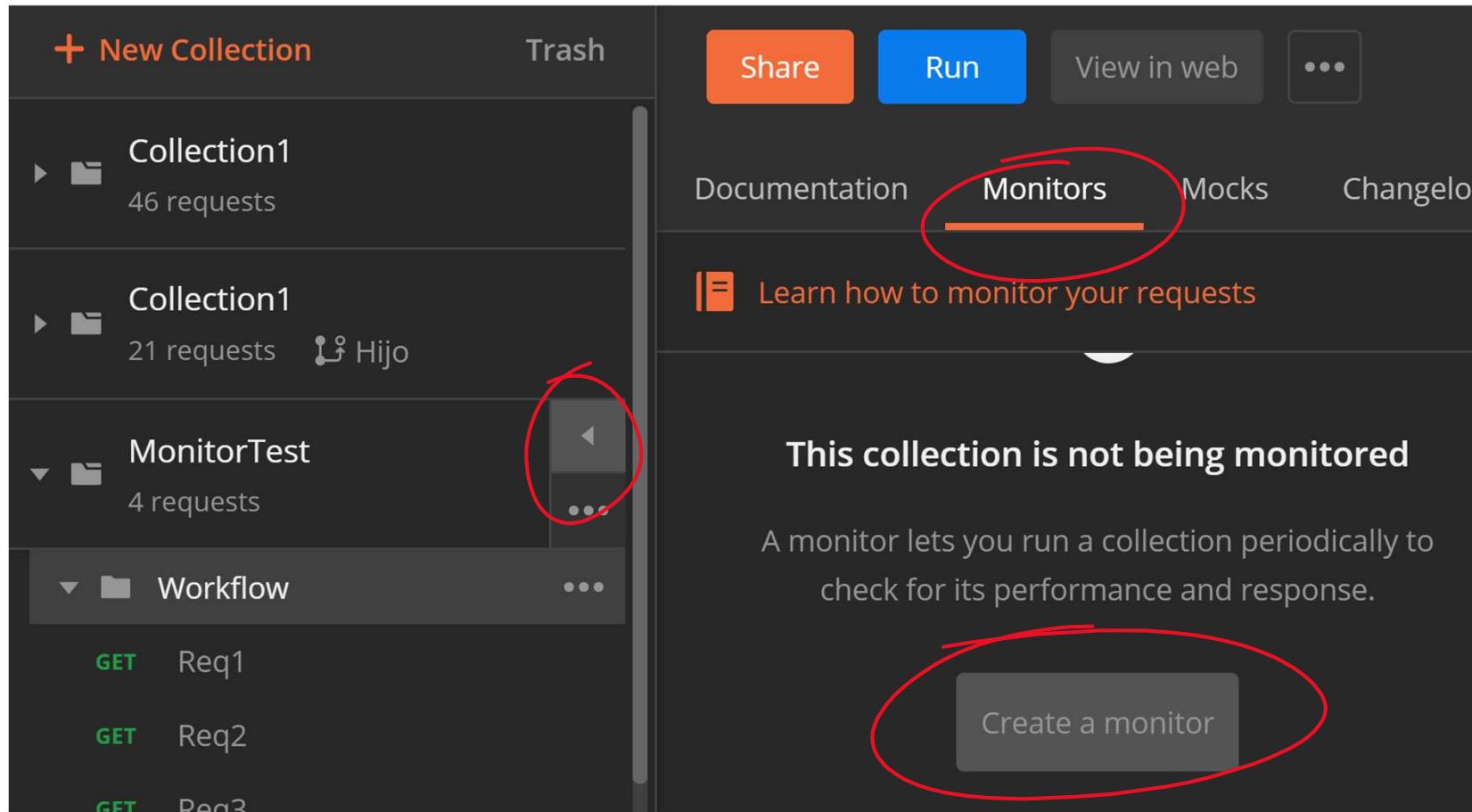
- Postman nos permite automatizar periódicamente la ejecución de un conjunto de peticiones mediante su herramienta llamada Monitor (1000 peticiones al mes de forma gratuita)
- Creamos una nueva colección llamada MonitorTest, duplicamos la carpeta Workflow y la arrastramos a la nueva colección:





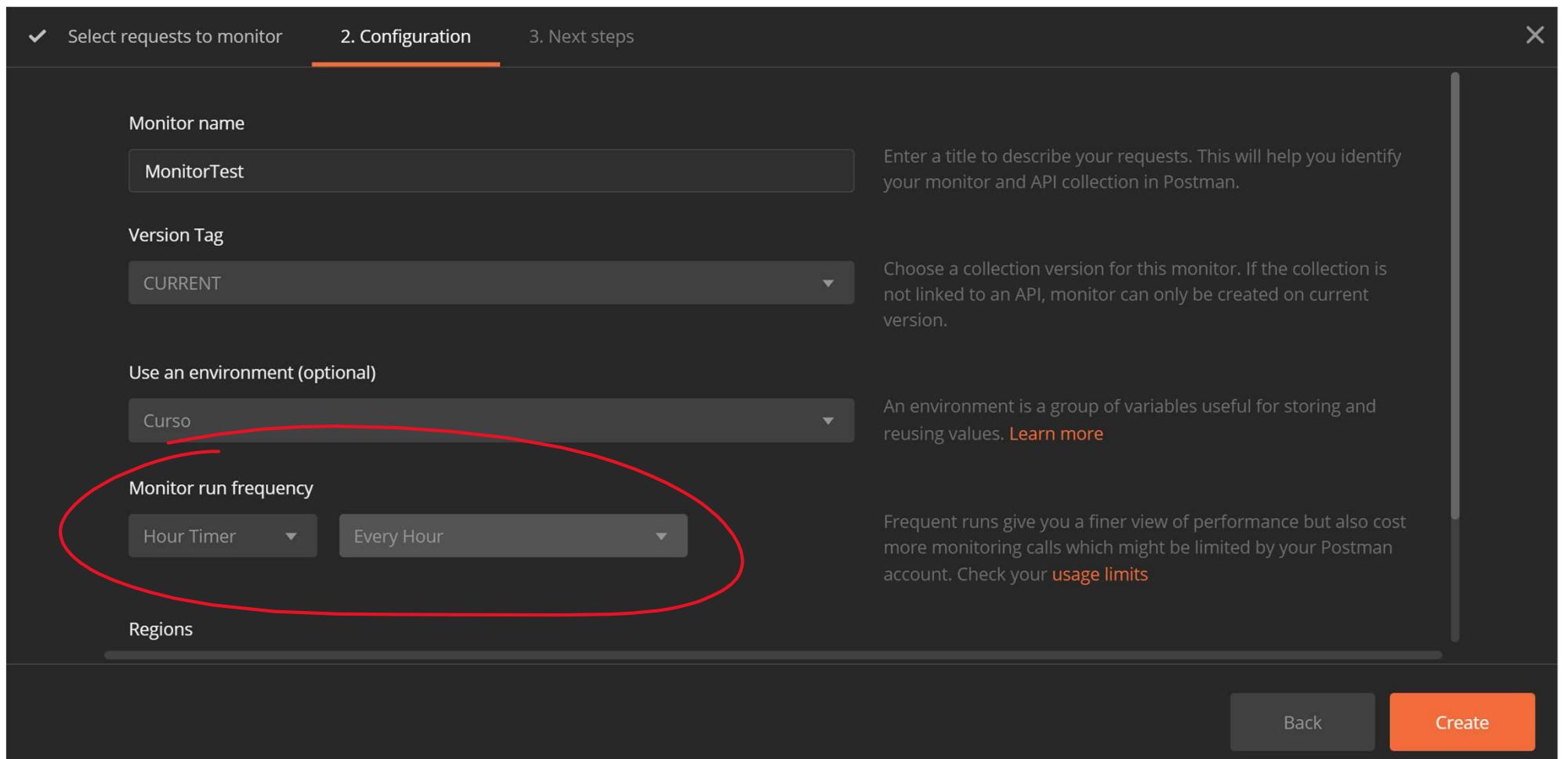
# Monitor

- Creamos un nuevo Monitor:



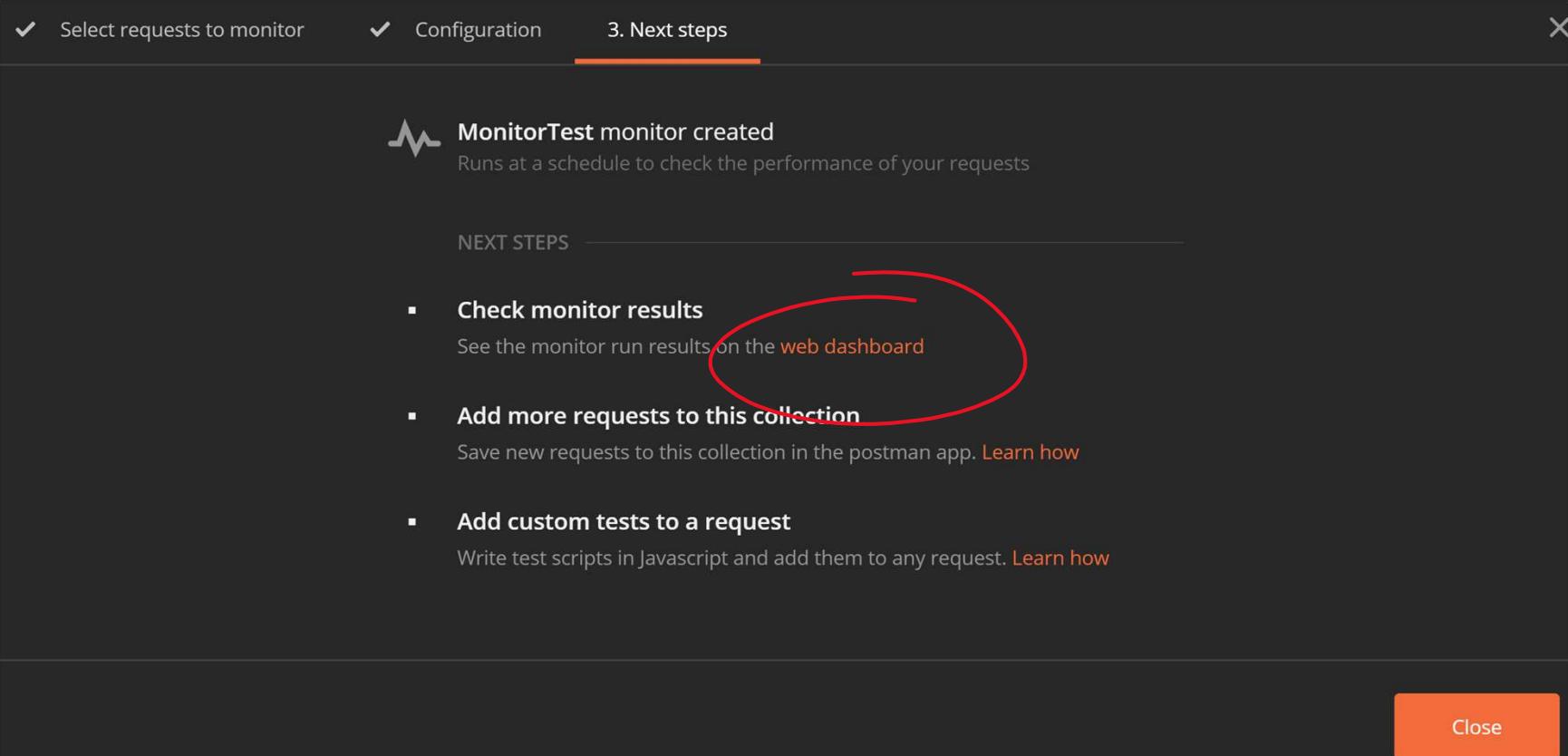
The screenshot shows the Postman application interface. On the left, there's a sidebar with collection names: Collection1 (46 requests), Collection1 (21 requests), MonitorTest (4 requests), and Workflow. The MonitorTest collection is expanded, showing three requests: Req1, Req2, and Req3. A red circle highlights the three-dot menu icon next to Req1. On the right, the main workspace has a toolbar with Share, Run, View in web, and a three-dot menu. Below the toolbar, tabs include Documentation, Monitors (which is highlighted with a red oval), Mocks, and Changelog. A callout from the Documentation tab says "Learn how to monitor your requests". A large text area states "This collection is not being monitored". Below it, a description explains: "A monitor lets you run a collection periodically to check for its performance and response." A prominent red oval encircles a "Create a monitor" button.

- Definimos la configuración y la frecuencia:



The screenshot shows the '2. Configuration' step of a monitor creation process in Postman. The steps are indicated by a progress bar at the top: 'Select requests to monitor' (checkmark), '2. Configuration' (orange bar), and '3. Next steps'.  
**Monitor name:** MonitorTest  
**Version Tag:** CURRENT  
**Use an environment (optional):** Curso  
**Monitor run frequency:** Hour Timer (selected) / Every Hour (highlighted with a red oval)  
**Regions:** (partially visible)  
  
Detailed description of the highlighted area:  
A red oval highlights the 'Monitor run frequency' section, which includes two dropdown menus: 'Hour Timer' and 'Every Hour'. To the right of this section is a descriptive text: 'Frequent runs give you a finer view of performance but also cost more monitoring calls which might be limited by your Postman account. Check your [usage limits](#)'.  
  
At the bottom right are 'Back' and 'Create' buttons.

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



✓ Select requests to monitor   ✓ Configuration   3. Next steps X

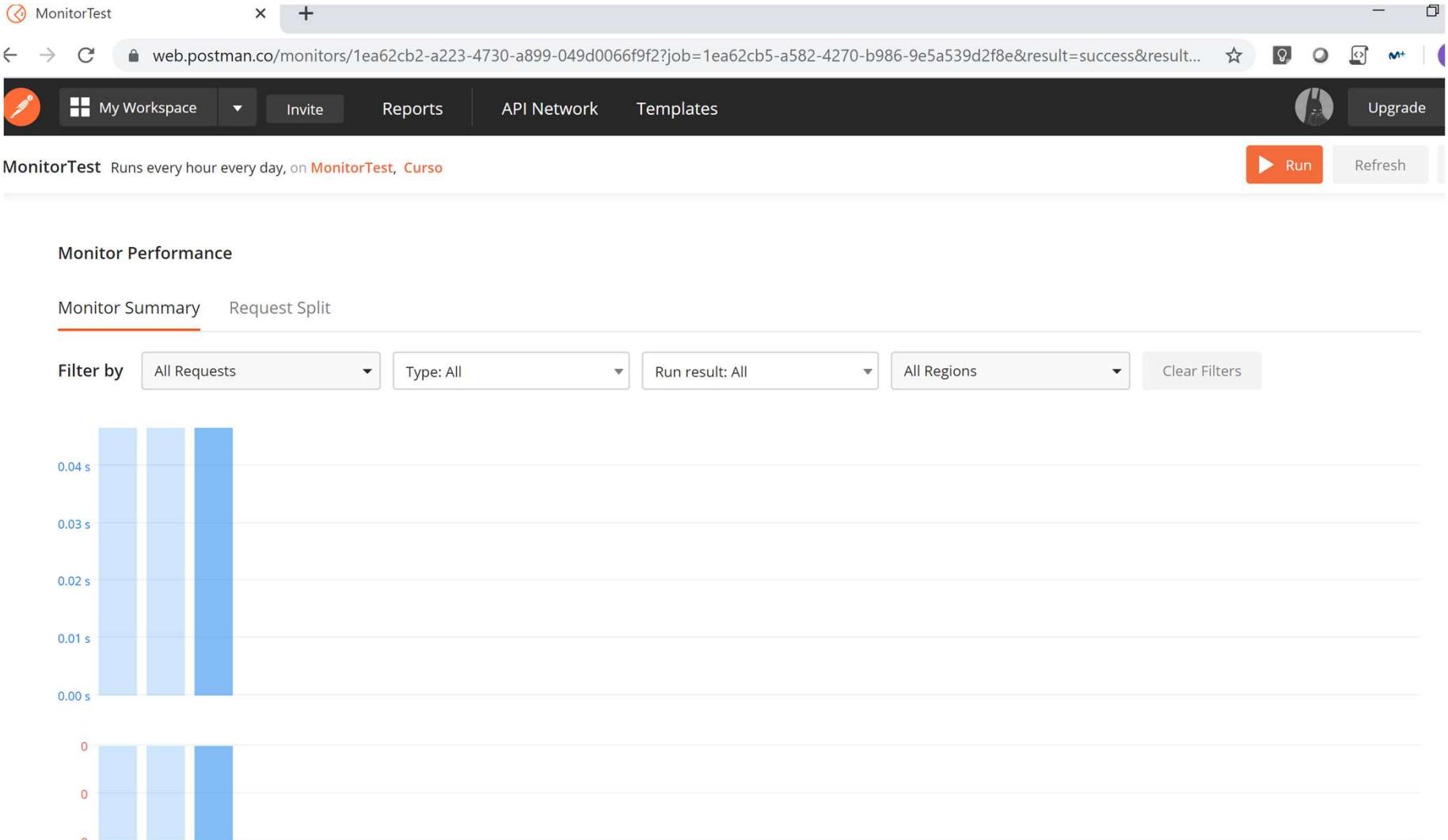
 MonitorTest monitor created  
Runs at a schedule to check the performance of your requests

NEXT STEPS

- **Check monitor results**  
See the monitor run results on the [web dashboard](#)
- **Add more requests to this collection**  
Save new requests to this collection in the postman app. [Learn how](#)
- **Add custom tests to a request**  
Write test scripts in Javascript and add them to any request. [Learn how](#)

Close

- Web Dashboard:



The screenshot shows the Postman Monitor web dashboard for a monitor named "MonitorTest". The monitor runs every hour every day on the "MonitorTest" environment. The dashboard features a "Monitor Performance" section with a "Monitor Summary" tab selected. It displays two bar charts: one for response time (ranging from 0.00 s to 0.04 s) and another for error count (ranging from 0 to 0). The interface includes a toolbar with "Run" and "Refresh" buttons, and a navigation bar with links for "My Workspace", "Reports", "API Network", and "Templates".

Monitor Test

← → ⌛ 🔒 web.postman.co/monitors/1ea62cb2-a223-4730-a899-049d0066f9f2?job=1ea62cb5-a582-4270-b986-9e5a539d2f8e&result=success&result...

My Workspace Reports API Network Templates Upgrade

MonitorTest Runs every hour every day, on MonitorTest, Curso

Run Refresh

Monitor Performance

Monitor Summary Request Split

Filter by All Requests Type: All Run result: All All Regions Clear Filters

Response Time (s)	Count
0.00 s - 0.01 s	3
0.01 s - 0.02 s	0
0.02 s - 0.03 s	0
0.03 s - 0.04 s	3

Error Count	Count
0	3