Iker Ríos Navarro - Práctica de Postman — REST Basics - 2º DAW - 23/09/2025

ÍNDICE

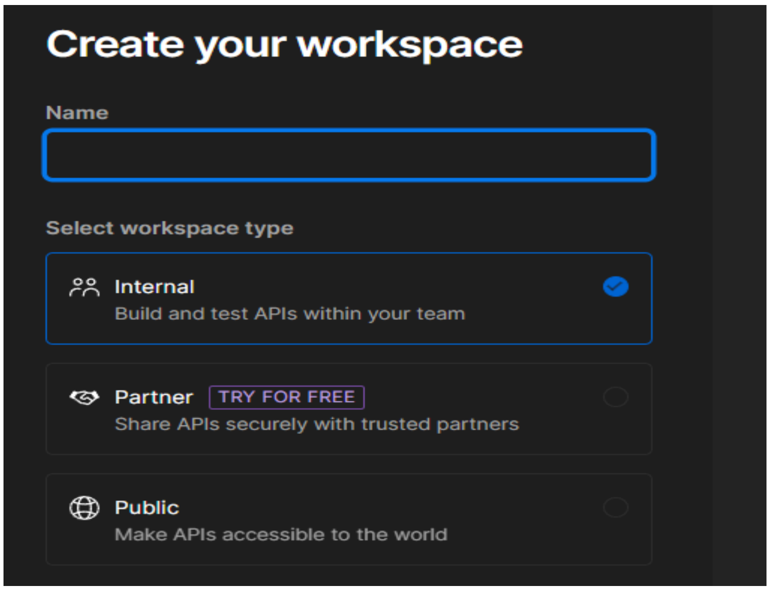
Iker Ríos Navarro - Práctica de Postman — REST Basics - 2º DAW - 23/09/2025
ÍNDICE
Requisitos previos
Preparación del entorno en Postman (5 pasos)
Conceptos clave (mini-resumen)
Parte A — Lectura de recursos (GET)
    A1. Listar recursos (GET collection)
    A2. Obtener un recurso concreto (GET item con path param)
    A3. Filtrado por query params
Parte B — Crear, actualizar y borrar (POST, PUT, PATCH, DELETE)
    B1. Crear (POST)
    B2. Actualizar completo (PUT)
    B3. Actualizar parcial (PATCH)
    B4. Borrar (DELETE)
Parte C — Headers, Auth y "sin estado"
    C1. Header personalizado (User-Agent)
    C2. Autorización tipo Bearer
Parte D — Errores y latencia
    D1. Códigos de error
    D2. Retrasos en la respuesta
Preguntas entregables

Requisitos previos

- Tener Postman instalado.
- Conexión a Internet.
- Usaremos endpoints públicos de prueba:
    - JSONPlaceholder (datos falsos): https://jsonplaceholder.typicode.com
    - httpbin (pruebas de HTTP): https://httpbin.org
    - reqres (opcional autenticación): https://reqres.in

Preparación del entorno en Postman (5 pasos)

1. Crear un WorkSpace: Classroom – Rest Basics.



Create your workspace

Name

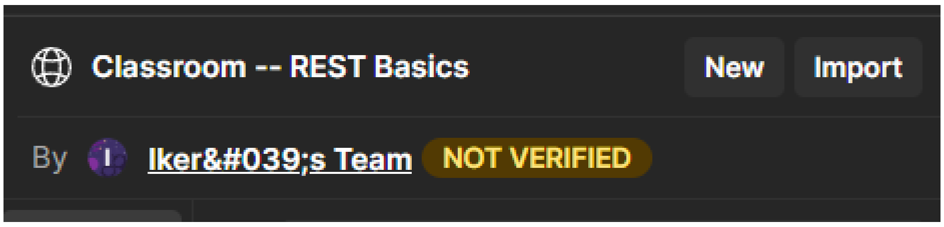
Select workspace type

Internal ✓
Build and test APIs within your team

Partner TRY FOR FREE
Share APIs securely with trusted partners

Public
Make APIs accessible to the world

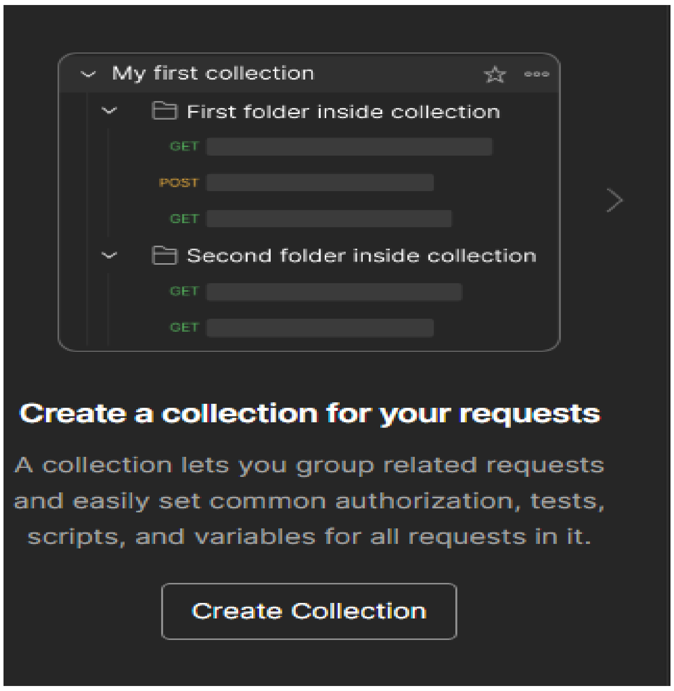
Una vez creado nos saldrá así:



Classroom -- REST Basics

By Iker&#039;s Team NOT VERIFIED

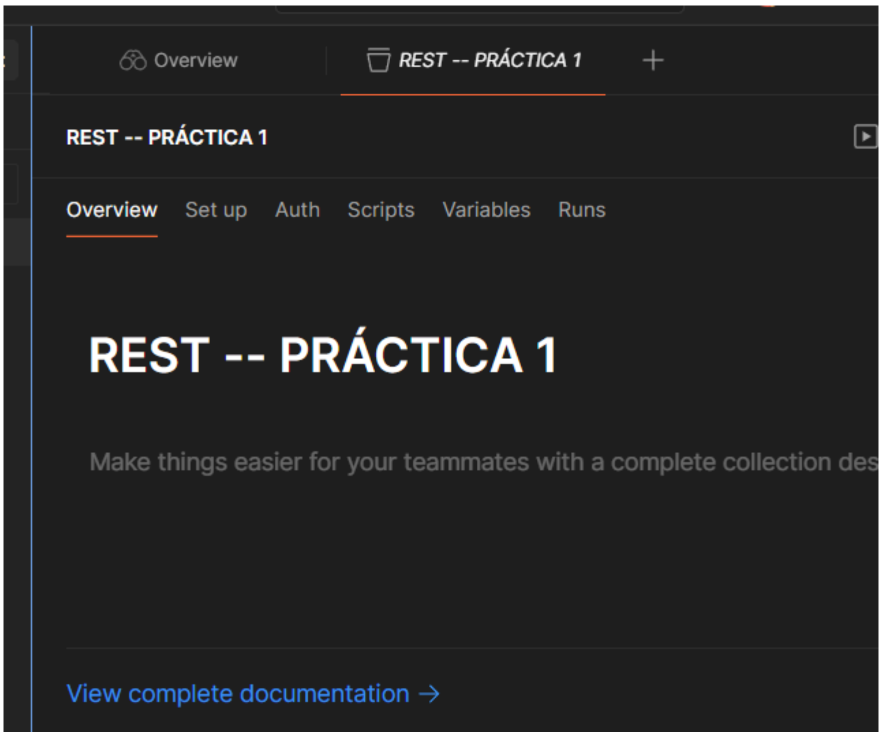
2. Crear una Colección: REST – Práctica 1.


My first collection
First folder inside collection
GET
POST
GET
Second folder inside collection
GET
GET

Create a collection for your requests

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Una vez creado nos saldrá así:



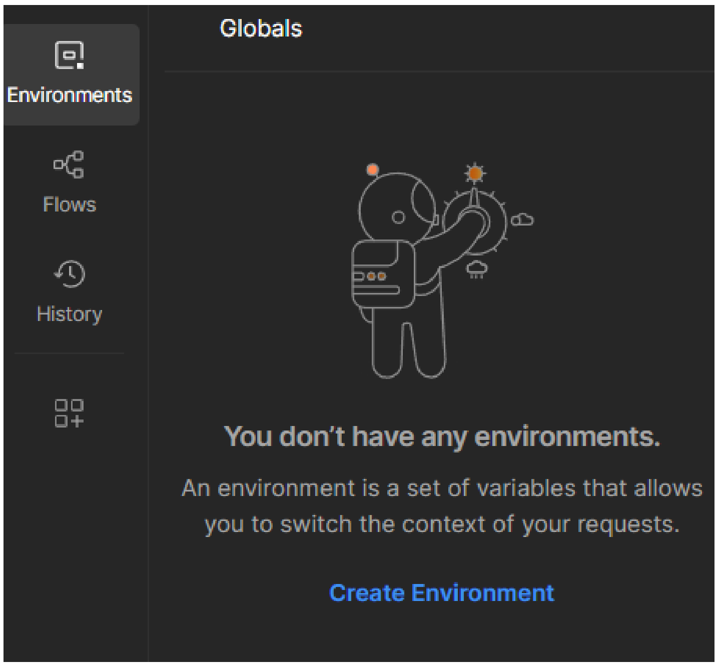
REST -- PRÁCTICA 1

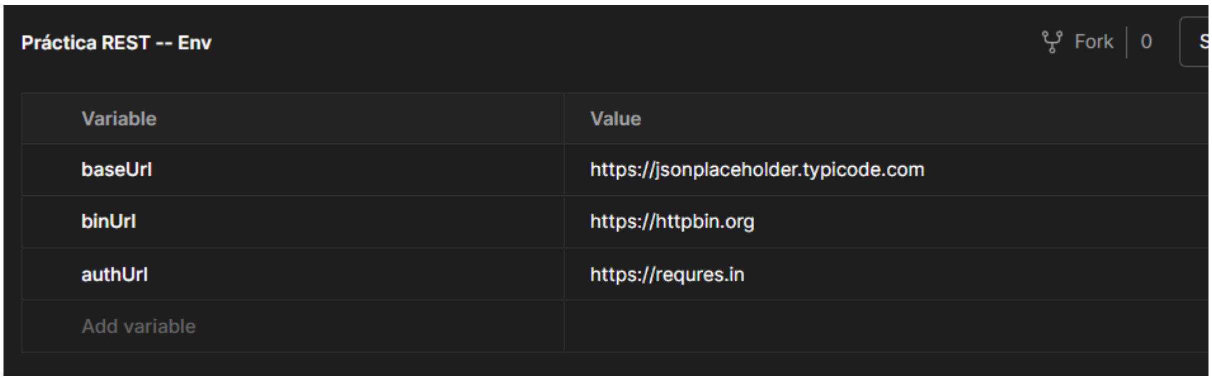
REST -- PRÁCTICA 1

Make things easier for your teammates with a complete collection des

3. Crear un Entorno: Práctica REST – Env. Añade variables:

- baseUrl = https://jsonplaceholder.typicode.com
- binUrl = https://httpbin.org
- authUrl = https://reqres.in (opcional, para el apartado de login)

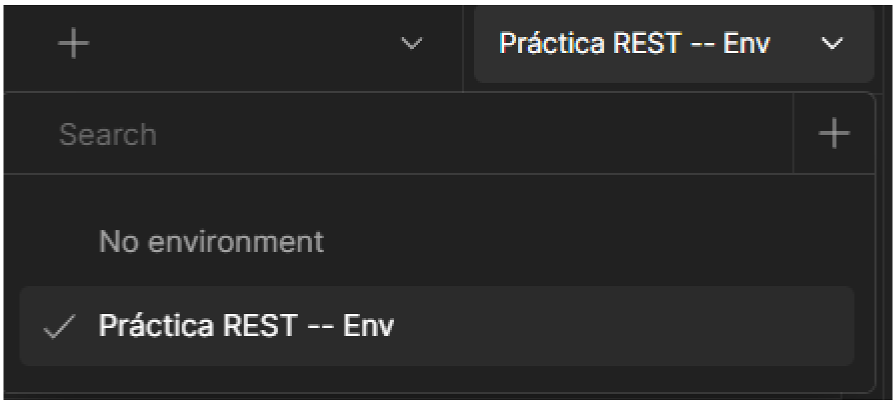

Globals
Environments
Flows
History
You don't have any environments.
An environment is a set of variables that allows you to switch the context of your requests.
Create Environment



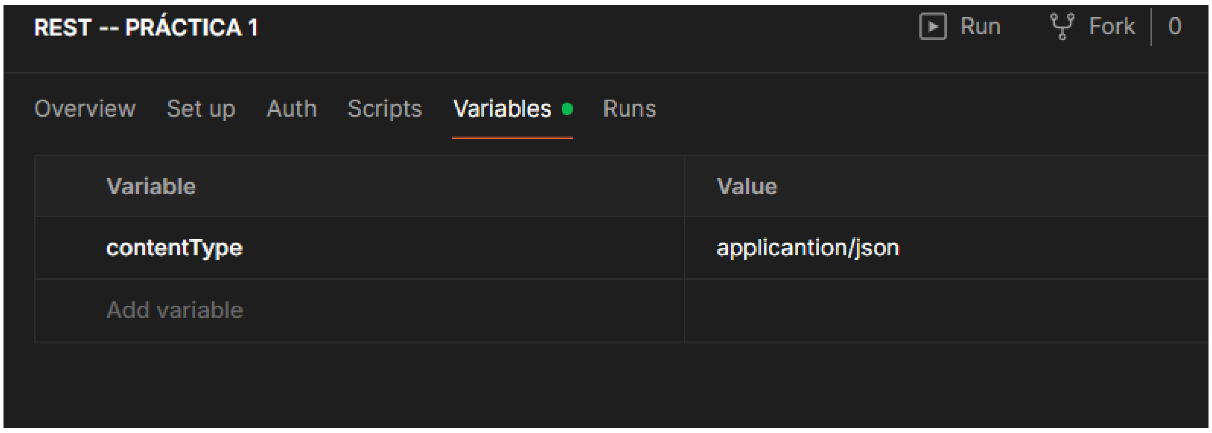
Práctica REST -- Env

Variable	Value
baseUrl	https://jsonplaceholder.typicode.com
binUrl	https://httpbin.org
authUrl	https://requres.in
Add variable	

4. Seleccionar el entorno en la parte superior de Postman:



5. (Opcional) En la Colección, en la pestaña Variables, define: contentType = application/json.


REST -- PRÁCTICA 1

Variable	Value
contentType	applicantion/json
Add variable	

Conceptos clave (mini-resumen)

API: Conjunto de servicios que permiten a un cliente interactuar con un servidor.

REST: Estilo de arquitectura que trabaja con recursos usando métodos HTTP y representaciones (normalmente JSON).

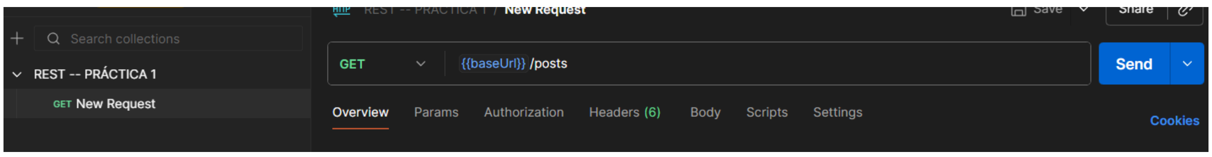
API RESTful: API que aplica correctamente los principios REST (URLs de recursos, verbos HTTP adecuados, sin estado, caché cuando toca, etc.).

Parte A — Lectura de recursos (GET)

A1. Listar recursos (GET collection)

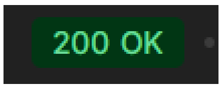
Crea una petición en la colección:

- ***GET {{baseUrl}}/posts***

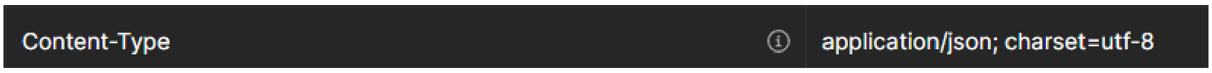

GET {{baseUrl}} /posts

Qué comprobar:

- Código de estado 200.

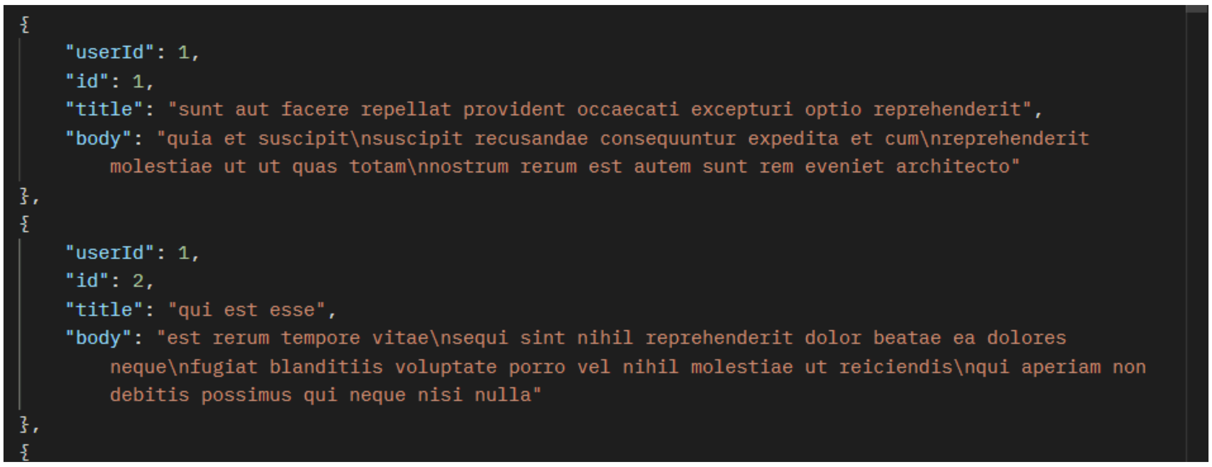


- Header Content-Type: application/json.



Content-Type: application/json; charset=utf-8

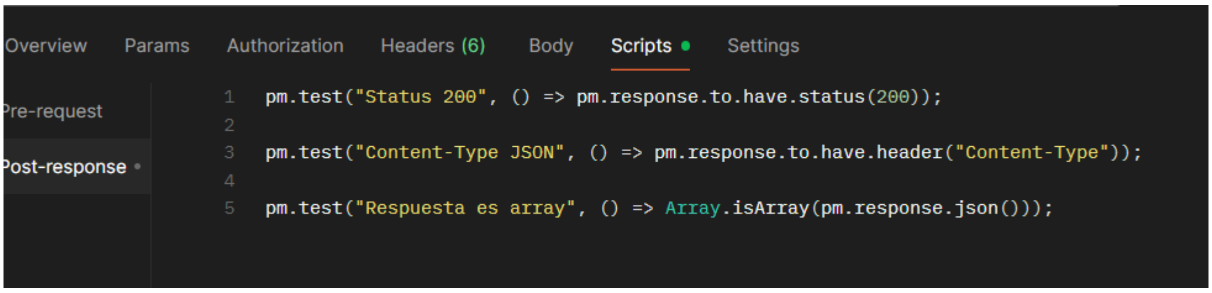
- El cuerpo es un array de posts.



```
{
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit
        molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
},
{
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores
        neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non
        debitis possimus qui neque nisi nulla"
},
{
```

Test:

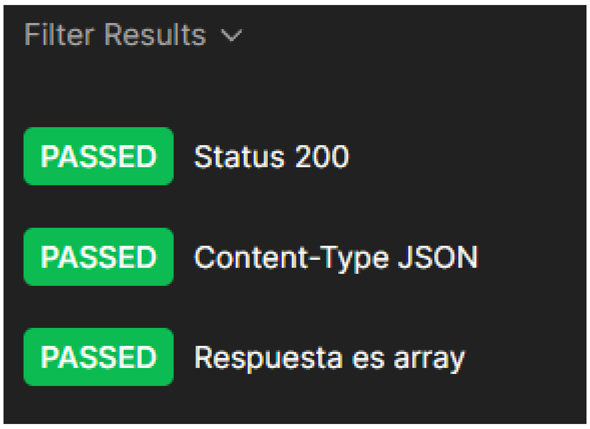
pm.test("Status 200", () => pm.response.to.have.status(200));
pm.test("Content-Type JSON", () => pm.response.to.have.header("Content-Type"));
pm.test("Respuesta es array", () => Array.isArray(pm.response.json()));



```
pm.test("Status 200", () => pm.response.to.have.status(200));

pm.test("Content-Type JSON", () => pm.response.to.have.header("Content-Type"));

pm.test("Respuesta es array", () => Array.isArray(pm.response.json()));
```



PASSED Status 200
PASSED Content-Type JSON
PASSED Respuesta es array

A2. Obtener un recurso concreto (GET item con path param)

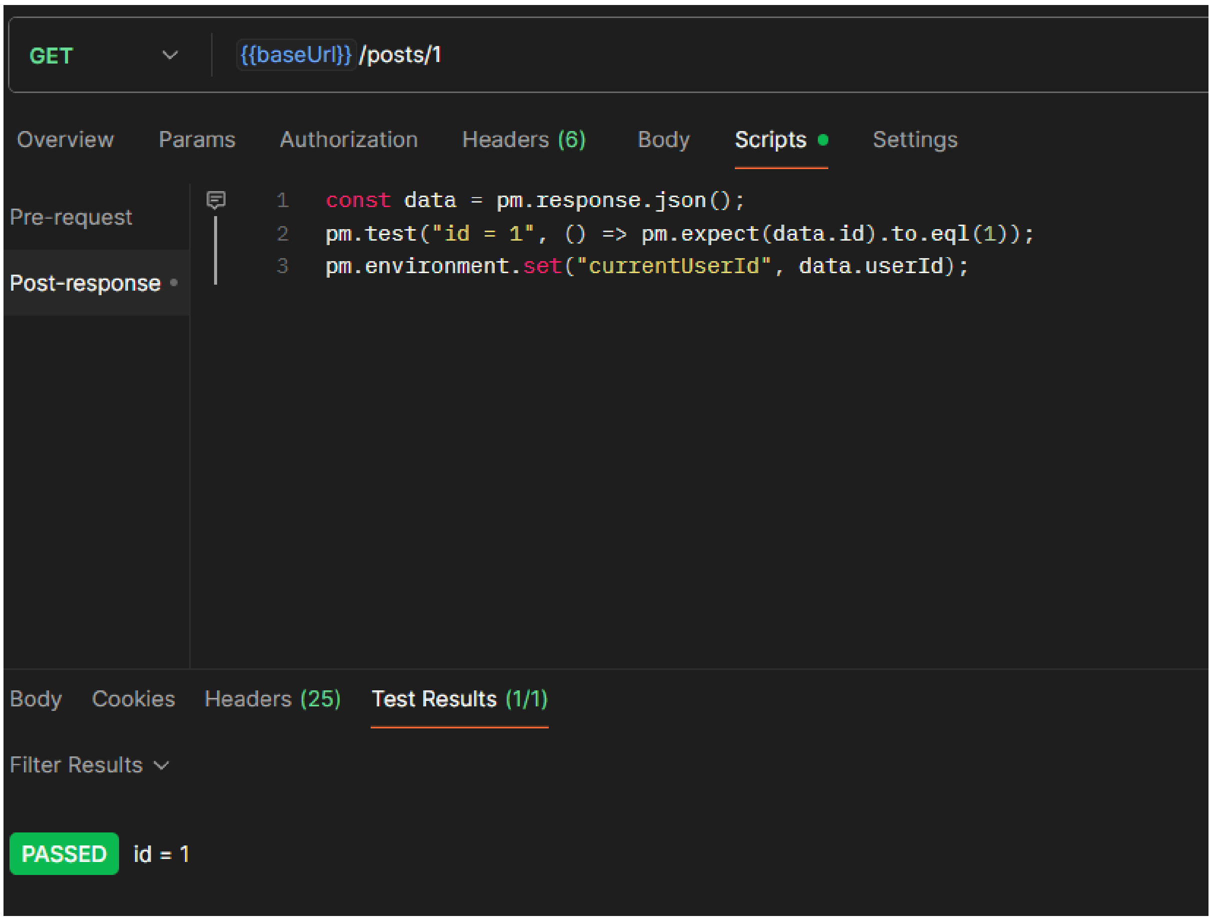
GET {{baseUrl}}/posts/1

Qué comprobar:

- Status 200.
- El cuerpo contiene id: 1.
- Guarda userId en una variable de entorno currentUserId.

Test:

const data = pm.response.json();
pm.test("id = 1", () => pm.expect(data.id).to.eql(1));
pm.environment.set("currentUserId", data.userId);


GET {{baseUrl}} /posts/1

```
const data = pm.response.json();
pm.test("id = 1", () => pm.expect(data.id).to.eql(1));
pm.environment.set("currentUserId", data.userId);
```

```
PASSED id = 1
```

A3. Filtrado por query params

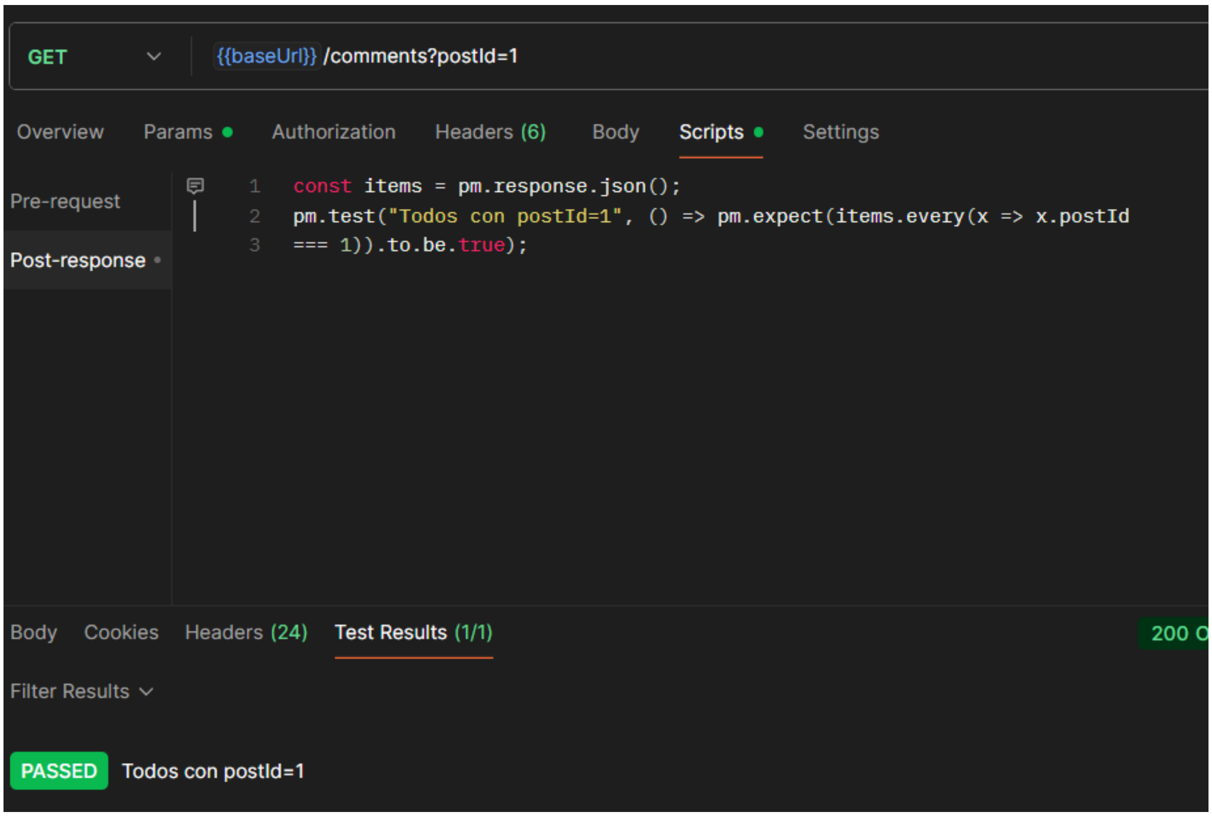
GET {{baseUrl}}/comments?postId=1

Qué comprobar:

- Status 200.
- Todos los comments tienen postId = 1.

Test:

> ***const items = pm.response.json();***
> ***pm.test("Todos con postId=1", () => pm.expect(items.every(x => x.postId === 1)).to.be.true);***



```
const items = pm.response.json();
pm.test("Todos con postId=1", () => pm.expect(items.every(x => x.postId
=== 1)).to.be.true);
```

PASSED Todos con postId=1

Parte B — Crear, actualizar y borrar (POST, PUT, PATCH, DELETE)

B1. Crear (POST)

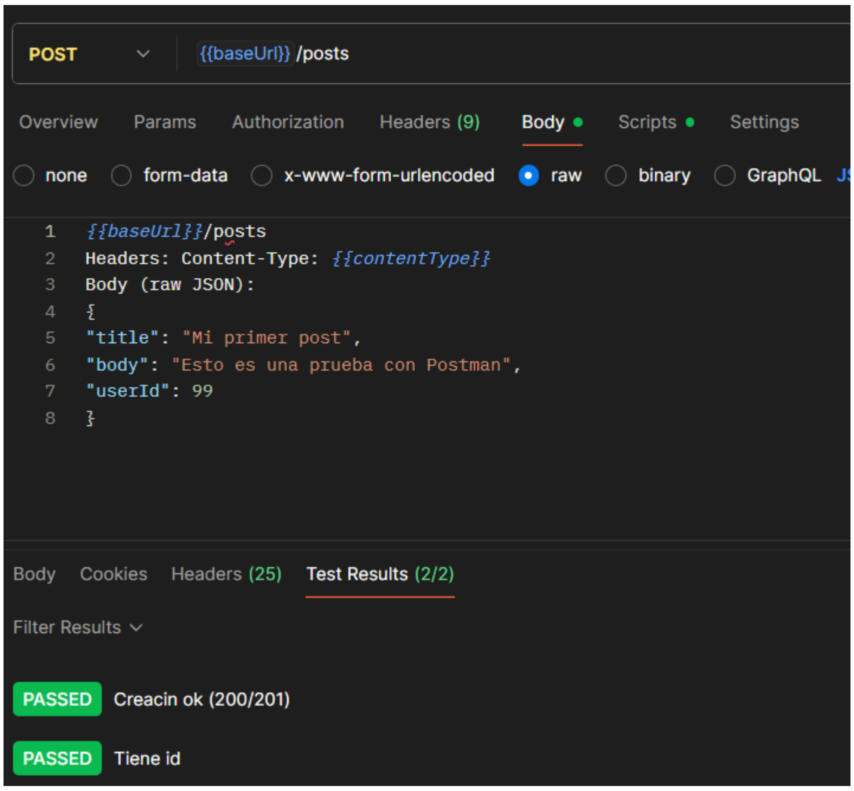
POST {{baseUrl}}/posts
Headers: Content-Type: {{contentType}}
Body (raw JSON):
{
"title": "Mi primer post",
"body": "Esto es una prueba con Postman",
"userId": 99

Qué comprobar:

- Status 201 (o 200 en algunos mocks).
- Respuesta JSON con un id nuevo.
- Guardar id en newPostId.

Test:

pm.test("Creacin ok (200/201)", () => pm.expect([200,201]).to.include(pm.response.code));
const created = pm.response.json();
pm.environment.set("newPostId", created.id);
pm.test("Tiene id", () => pm.expect(created).to.have.property("id"));



```
{{baseUrl}}/posts
Headers: Content-Type: {{contentType}}
Body (raw JSON):
{
"title": "Mi primer post",
"body": "Esto es una prueba con Postman",
"userId": 99
}
```

```
PASSED  Creacin ok (200/201)
PASSED  Tiene id
```

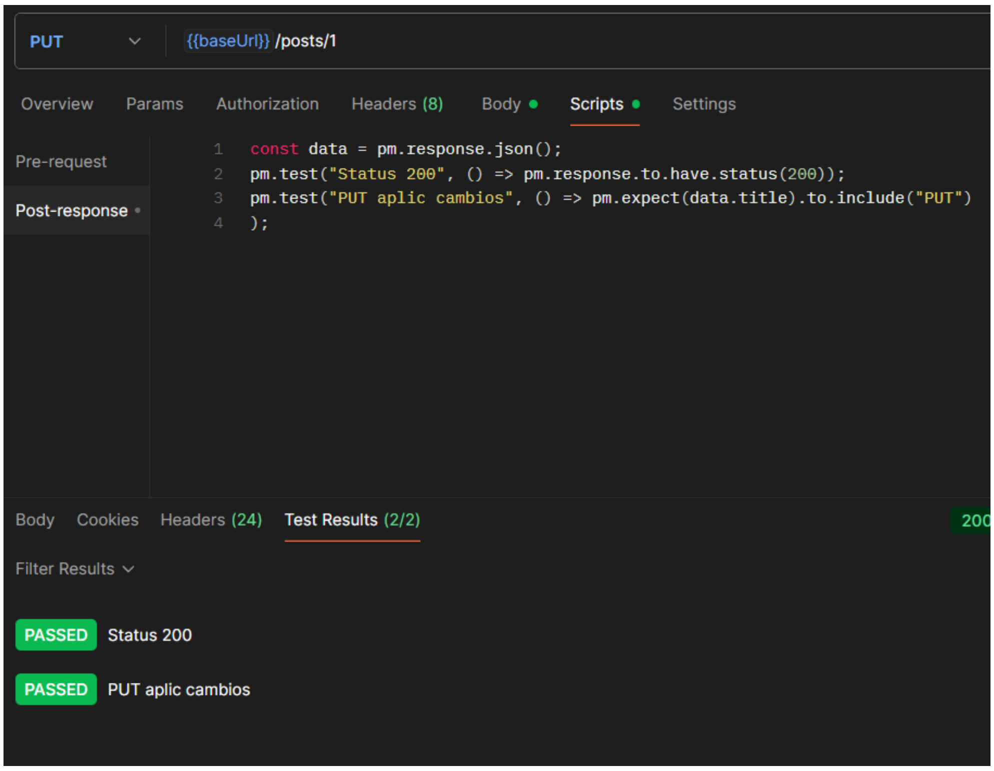
B2. Actualizar completo (PUT)

PUT {{baseUrl}}/posts/1

```
Body:
{
"id": {{newPostId}},
"title": "Ttulo actualizado (PUT)",
"body": "Contenido actualizado por PUT",
"userId": 99
}
```

Test:

```
const data = pm.response.json();
pm.test("Status 200", () => pm.response.to.have.status(200));
pm.test("PUT aplic cambios", () => pm.expect(data.title).to.include("PUT")
);
```



```
const data = pm.response.json();
pm.test("Status 200", () => pm.response.to.have.status(200));
pm.test("PUT aplic cambios", () => pm.expect(data.title).to.include("PUT")
);
```

PASSED Status 200

PASSED PUT aplic cambios

B3. Actualizar parcial (PATCH)

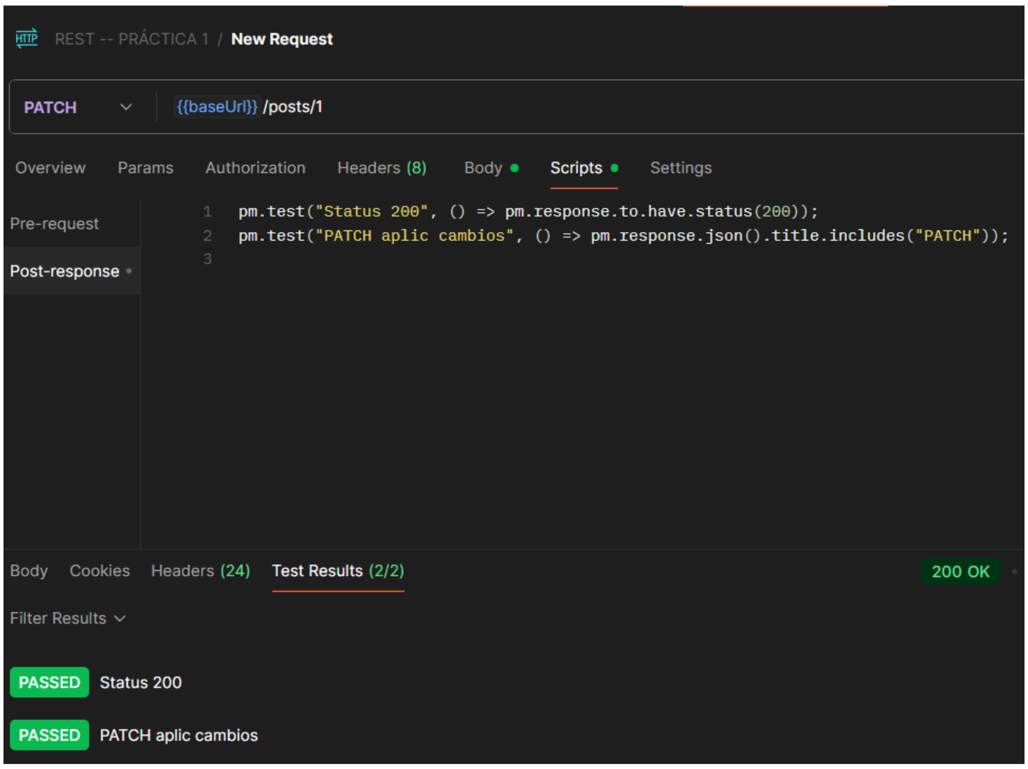
PATCH {{baseUrl}}/posts/1

Body:

```
{
        "title": "Título modificado (PATCH)"
}
```

Test:

```
pm.test("Status 200", () => pm.response.to.have.status(200));
pm.test("PATCH aplic cambios", () => pm.response.json().title.includes("
PATCH"));
```


REST -- PRÁCTICA 1 / **New Request**

PATCH {{baseUrl}} /posts/1

Overview Params Authorization Headers (8) Body ● Scripts ● Settings

Pre-request
Post-response

```
pm.test("Status 200", () => pm.response.to.have.status(200));
pm.test("PATCH aplic cambios", () => pm.response.json().title.includes("PATCH"));
```

Body Cookies Headers (24) Test Results (2/2) 200 OK

Filter Results

PASSED Status 200

PASSED PATCH aplic cambios

B4. Borrar (DELETE)

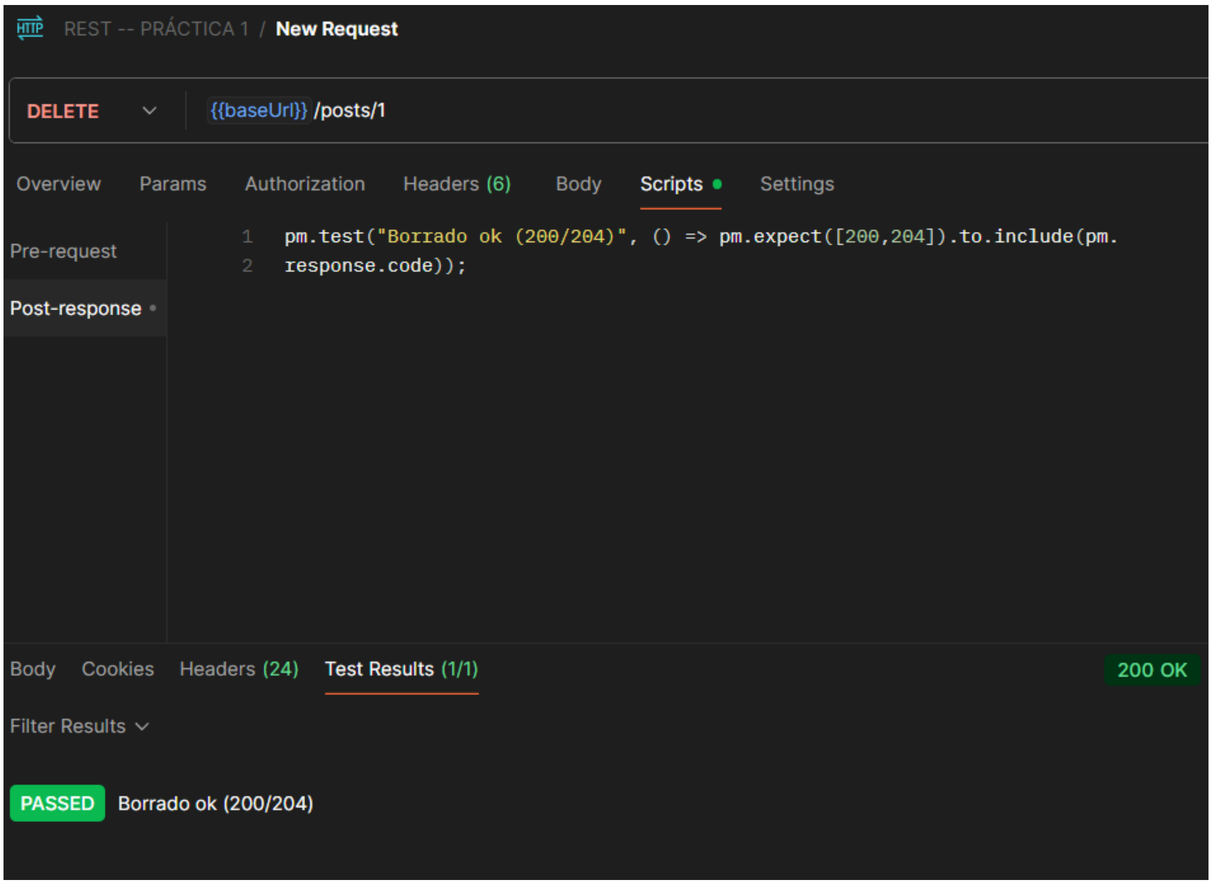
DELETE {{baseUrl}}/posts/1

Test:

```
pm.test("Borrado ok (200/204)", () => pm.expect([200,204]).to.include(pm.
response.code));
```

Idea clave (REST):

- GET, PUT, DELETE son idempotentes.
- POST no es idempotente.



DELETE {{baseUrl}}/posts/1

```
pm.test("Borrado ok (200/204)", () => pm.expect([200,204]).to.include(pm.
response.code));
```

PASSED Borrado ok (200/204)

Parte C — Headers, Auth y “sin estado”

C1. Header personalizado (User-Agent)

Petición GET con un header HTTP a ̃nadido por el cliente.

El servidor refleja los headers recibidos y podemos verificar el cambio.
Finalidad: Practicar la personalización de headers en las peticiones.

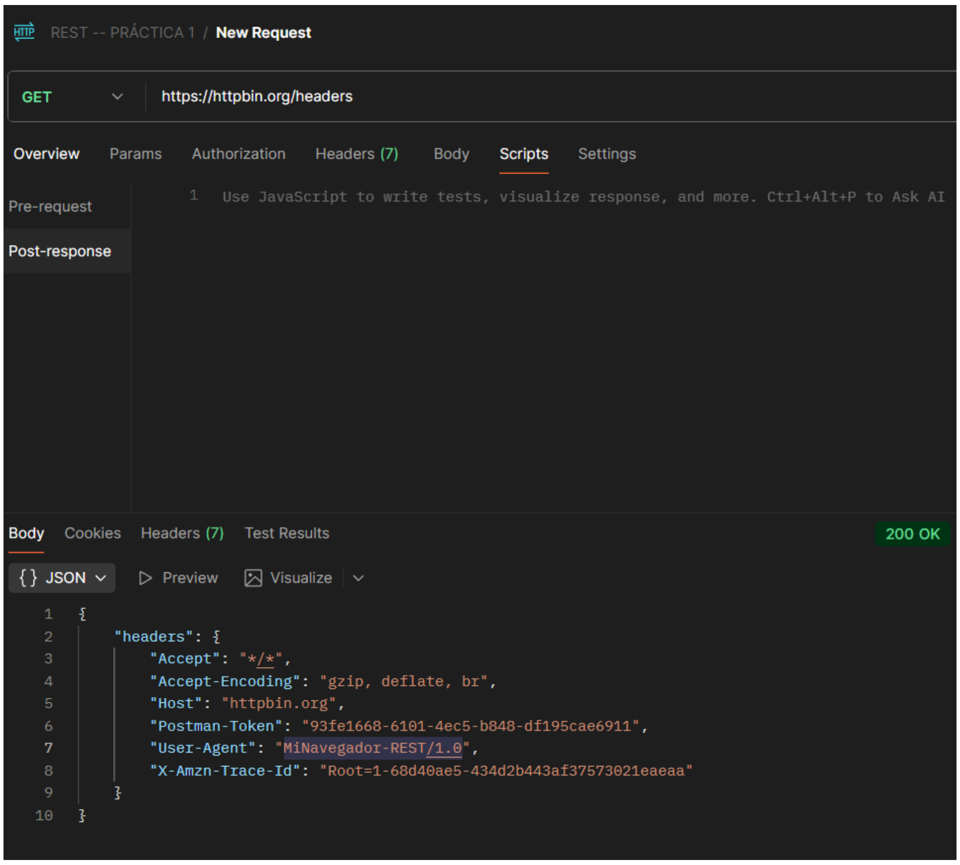
1. Abre una nueva petición: Método: GET.
2. URL: https://httpbin.org/headers
3. Ve a la pestaña Headers.
4. Añade: Key: User-Agent, Value: MiNavegador-REST/1.0
5. Envía la petición.

Respuesta esperada:

```
{
        "headers": {
                "Accept": "*/*",
                "Host": "httpbin.org",
                "User-Agent": "MiNavegador-REST/1.0"
        }
}
```

Test extra:

```
        const j = pm.response.json();
        pm.test("User-Agent modificado", () => {
                pm.expect(j.headers["User-Agent"]).to.eql("MiNavegador-REST/1.0");
});
```



```
{
    "headers": {
        "Accept": "*/*",
        "Accept-Encoding": "gzip, deflate, br",
        "Host": "httpbin.org",
        "Postman-Token": "93fe1668-6101-4ec5-b848-df195cae6911",
        "User-Agent": "MiNavegador-REST/1.0",
        "X-Amzn-Trace-Id": "Root=1-68d40ae5-434d2b443af37573021eaeaa"
    }
}
```

C2. Autorización tipo Bearer

Petición GET con header de autenticación Bearer.
Enviamos el token en Authorization y el servidor valida si es correcto.

Finalidad: Practicar autenticación sin estado (stateless), en donde cada petición es independiente.

1. Crea variable de entorno token = alumno-123.

2. Crea:

GET {{binUrl}}/bearer
Authorization Bearer Token Token: {{token}}

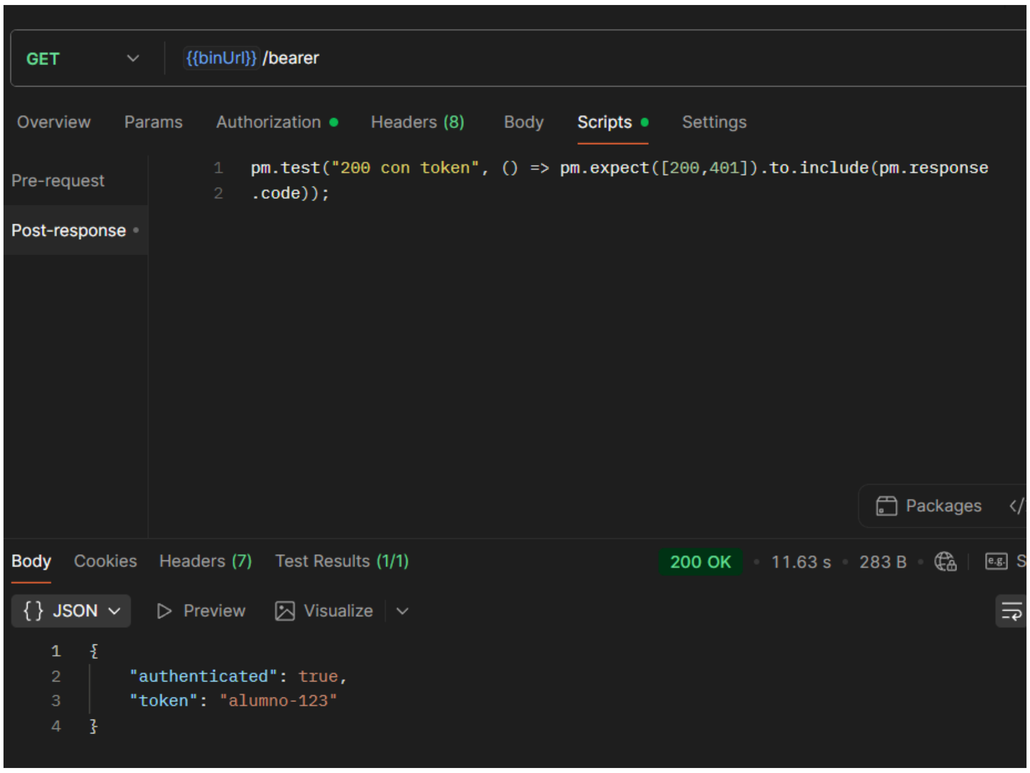
Qué comprobar:

- Status 200 y authenticated: true.
- Sin token debe devolver 401.

Test:

```
pm.test("200 con token", () => pm.expect([200,401]).to.include(pm.response
.code));
```

Principio REST: cada petición lleva lo necesario (sin estado).



{{binUrl}}/bearer

```
pm.test("200 con token", () => pm.expect([200,401]).to.include(pm.response
.code));
```

```
{
    "authenticated": true,
    "token": "alumno-123"
}
```

Parte D — Errores y latencia

D1. Códigos de error

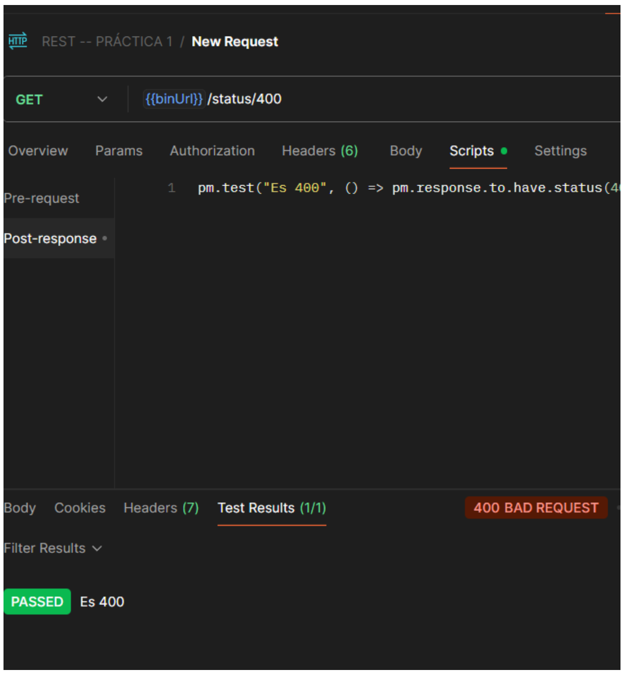
Peticiones diseñadas para provocar errores de forma controlada.

El servidor devuelve el código HTTP correspondiente (404, 400).
Finalidad: Validar el manejo de errores en las pruebas automatizadas.

```
GET {{binUrl}}/status/404
GET {{binUrl}}/status/400
```

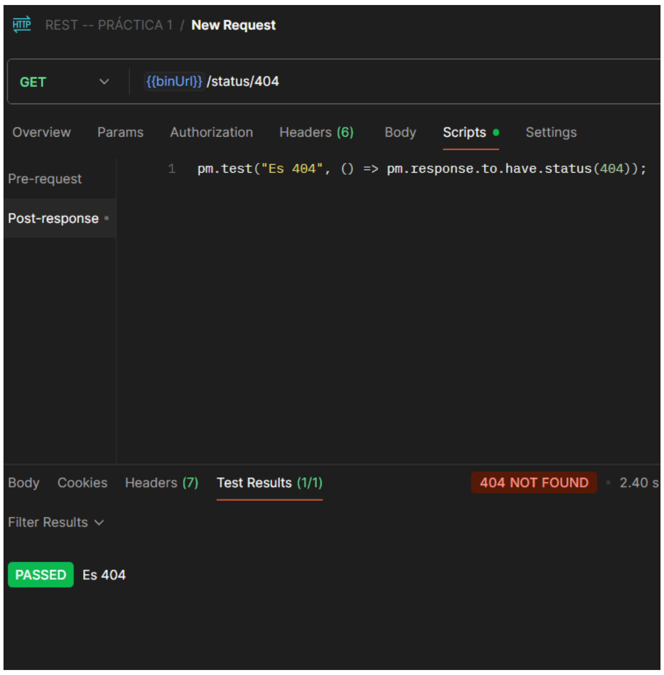
Test:

```
pm.test("Es 404", () => pm.response.to.have.status(404));
```



```
pm.test("Es 400", () => pm.response.to.have.status(4
```

PASSED Es 400



```
pm.test("Es 404", () => pm.response.to.have.status(404));
```

PASSED Es 404

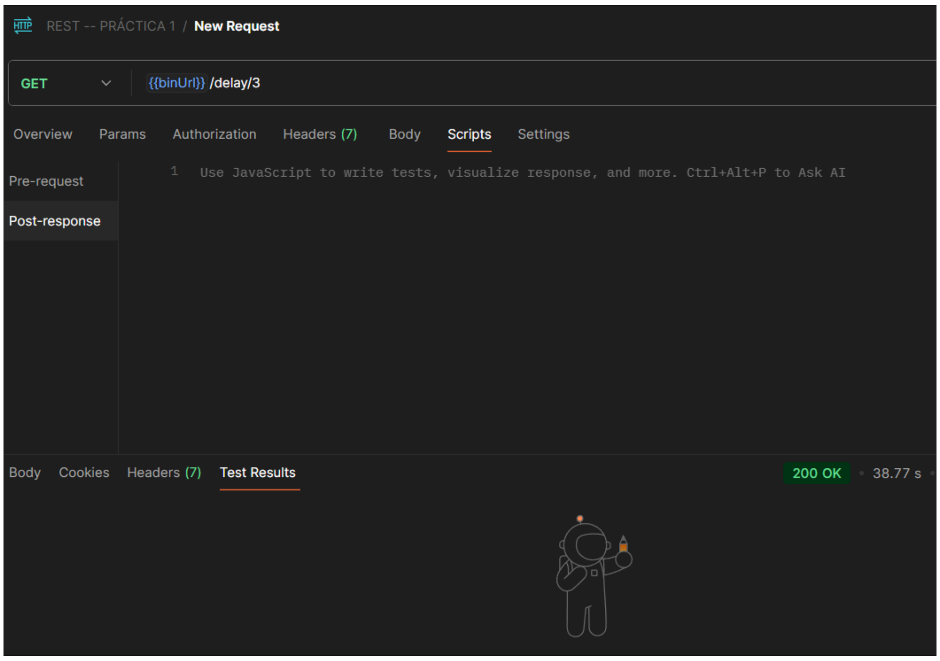
D2. Retrasos en la respuesta

Petición que simula una respuesta lenta.
El servidor espera antes de responder (ej. 3 segundos).

Finalidad: Comprobar configuración de timeout y comportamiento del cliente frente a la latencia.

GET {{binUrl}}/delay/3

Ajusta **Request timeout** si fuese necesario.



```
Use JavaScript to write tests, visualize response, and more. Ctrl+Alt+P to Ask AI
```

Preguntas entregables

Breve respuesta escrita (máx. 10 lıneas):

- Diferencia entre path y query params.

Path lo usamos para buscar algo en especifico y único, mientras que query lo usamos para filtrar y por decirlo así realizar una búsqueda algo más general.

- ¿Por qué POST no es idempotente y PUT sí?

POST: crea cosas nuevas, si lo mandas varias veces, el resultado cambia → **no idempotente**.

PUT: actualiza algo concreto, si lo mandas varias veces, el resultado final es siempre el mismo → **idempotente**.

- ¿Qué significa que una API sea sin estado?

Cada petición que le mandas es independiente, la API no recuerda nada de lo que hiciste antes.