

«Talento Tech»

Testing QA

Clase 11



Clase N°11 | Herramientas de API Testing

Temario:

- Manipulación y evaluación de JSON en testing de API
- Instalación y configuración básica de Postman
- Solicitudes HTTP (GET, POST, PUT, DELETE)
- Creación de colecciones en Postman
- Uso de variables y entornos
- Aserciones y validaciones automáticas en JavaScript
- Collection Runner: ejecución de pruebas por lotes
- Buenas prácticas para organización, documentación y reutilización

Objetivos de clase

En esta clase vamos a aprender a utilizar **Postman**, una de las herramientas más utilizadas en el mundo del testing de APIs. Comenzaremos con solicitudes básicas (GET, POST, PUT y DELETE) y luego avanzaremos paso a paso hacia la creación de colecciones organizadas, el uso de variables y entornos, y la validación automática de respuestas. También veremos cómo ejecutar pruebas en lote con el Collection Runner y cómo documentar nuestras pruebas para que sean claras, reutilizables y compartibles con el equipo. Al finalizar, tendrás tu primera colección de pruebas funcionando.

Manipulación y Evaluación de JSON en Testing de API

El Testing de API implica interactuar con las respuestas JSON para validar que la API devuelva los datos correctos y en el formato esperado.

Postman

Postman es una herramienta popular para desarrollar, probar y automatizar APIs. Permite a los testers, desarrolladores y equipos de QA enviar solicitudes HTTP, analizar respuestas y validar el comportamiento de una API sin necesidad de escribir código desde cero.



¿Para qué se usa Postman?

- Enviar solicitudes HTTP (GET, POST, PUT, DELETE, etc.).
- Probar APIs manualmente y con scripts automatizados.
- Validar respuestas JSON o XML y asegurar que cumplen con los requisitos.
- Automatizar pruebas de API con scripts en JavaScript.
- Realizar pruebas de carga y rendimiento.
- Compartir colecciones de pruebas con equipos de trabajo.

Instalación y Configuración de Postman:

1. Descarga Postman desde <https://www.postman.com/downloads/>
2. Instala y crea una cuenta opcionalmente.
3. Abre la aplicación y accede a la pestaña "**Collections**" para organizar tus pruebas.

Configuración Básica y primera prueba:

URL base de la API:

Cuando vas a probar una API, necesitás conocer su endpoint. En este caso, usaremos:

<https://dummyjson.com/products>

Vamos a hacer una **petición GET** a la API de Dummyjson para obtener un listado de productos.

En Postman:

1. Ingresá la URL en el campo correspondiente.
2. Seleccioná el método **GET**.
3. Hacé clic en **Send** para enviar la solicitud.

Deberías ver una respuesta como esta:

The screenshot shows the Postman interface for a GET request to `https://dummyjson.com/products`. The 'Send' button is visible in the top right. Below the URL bar, the 'Params' tab is active, showing a table for Query Params with columns KEY, VALUE, and DESCRIPTION. The 'Body' tab is selected, displaying the response in a 'Pretty' JSON view. The response is a JSON array of product objects. The status bar at the bottom indicates a successful response with status 200 OK, time 539 ms, and size 44.36 KB.

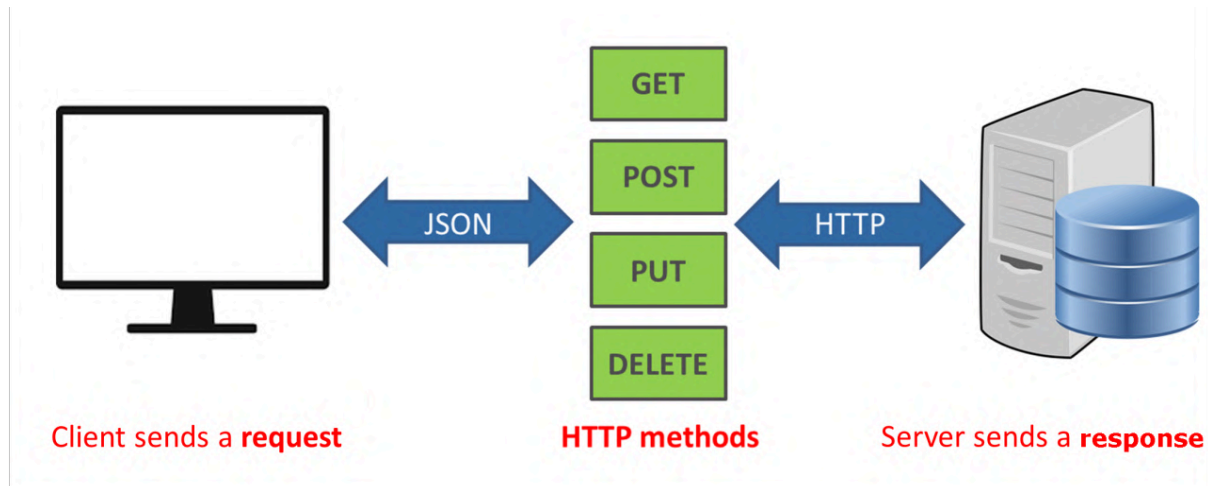
KEY	VALUE	DESCRIPTION
Key	Value	Description

```
1  {
2    "products": [
3      {
4        "id": 1,
5        "title": "Essence Mascara Lash Princess",
6        "description": "The Essence Mascara Lash Princess is a popular mascara known for its volumizing and lengthening effects. Achieve dramatic lashes with this long-lasting and cruelty-free formula.",
7        "category": "beauty",
8        "price": 9.99,
9        "discountPercentage": 7.17,
10       "rating": 4.94,
11       "stock": 5,
12       "tags": [
13         "beauty",
14         "mascara"
15       ],
16       "brand": "Essence",
17       "sku": "RCH45Q1A",
18       "weight": 2,
19       "dimensions": {
20         "width": 23.17,
21         "height": 14.43,
22         "depth": 28.01
23       },
24       "warrantyInformation": "1 month warranty",
```

¿Qué es lo que estamos haciendo?

Estamos realizando una solicitud **GET** a un servidor, que nos responde con un listado completo de productos.

Además, vemos que nos devuelve un **código de estado 200**, lo que indica que la solicitud fue exitosa y procesada correctamente.



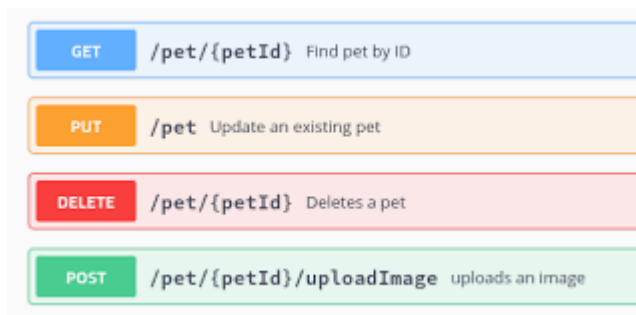
Hacer Peticiones y Recibir JSON en Postman.

Parte 1: Solicitudes HTTP Básicas

¿Qué es una solicitud HTTP?

Es una forma en la que un cliente (como Postman o una app) le **pide algo a un servidor**. En las APIs, esto se hace mediante métodos HTTP como:

- **GET**: para **consultar datos**.
- **POST**: para **crear datos nuevos**.
- **PUT**: para **actualizar datos existentes**.
- **DELETE**: para **eliminar datos**.



GET: Consultar datos

¿Qué hace? Pide información al servidor sin modificar nada.

Ejemplo real: GET <https://jsonplaceholder.typicode.com/users>

¿Qué devuelve? Una lista de usuarios en formato JSON.

Ejemplo de respuesta:

```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "email": "leanne@example.com"
  }
]
```

POST: Crear datos

¿Qué hace? Envía datos nuevos para que el servidor los guarde.

Ejemplo real: POST <https://reqres.in/api/users>

The screenshot shows a REST client interface with a POST request to `https://reqres.in/api/users`. The request body is a JSON object: `{ "name": "Juan", "job": "QA Tester" }`. The response status is 201 Created, with a time of 200 ms and a size of 112 KB. The response body is a JSON object: `{ "name": "Juan", "job": "QA Tester", "id": "818", "createdAt": "2025-04-12T10:29:33.109Z" }`.

POST `https://reqres.in/api/users` Send

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

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

```
1 {
2   "name": "Juan",
3   "job": "QA Tester"
4 }
5
```

Body Cookies Headers (15) Test Results Status: 201 Created Time: 200 ms Size: 112 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "name": "Juan",
3   "job": "QA Tester",
4   "id": "818",
5   "createdAt": "2025-04-12T10:29:33.109Z"
6 }
```


Cuerpo de la solicitud (Body):

```
{  
  "name": "Juan",  
  "job": "QA Tester"  
}
```

¿Qué devuelve? El objeto creado, con su ID y fecha.

Ejemplo de respuesta:

```
{  
  "name": "Juan",  
  "job": "QA Tester",  
  "id": "456",  
  "createdAt": "2025-04-05T14:32:00.000Z"  
}
```

PUT: Actualizar datos existentes

¿Qué hace? Modifica completamente un registro que ya existe.

Ejemplo real: PUT <https://reqres.in/api/users/2>

The screenshot displays a REST client interface for a PUT request to the URL `https://reqres.in/api/users/2`. The request body is a JSON object: `{ "name": "Juan Pérez", "job": "Senior QA" }`. The response status is `200 OK` with a response time of `146 ms` and a size of `1.16 KB`. The response body is a JSON object: `{ "name": "Juan Pérez", "job": "Senior QA", "updatedAt": "2025-04-12T10:30:40.551Z" }`.

PUT `https://reqres.in/api/users/2` Send

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

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

```
1 {  
2   "name": "Juan Pérez",  
3   "job": "Senior QA"  
4 }  
5
```

Body Cookies Headers (17) Test Results Status: 200 OK Time: 146 ms Size: 1.16 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "name": "Juan Pérez",  
3   "job": "Senior QA",  
4   "updatedAt": "2025-04-12T10:30:40.551Z"  
5 }
```

Cuerpo del Body:

```
{  
  "name": "Juan Pérez",  
  "job": "Senior QA"  
}
```

¿Qué devuelve? El objeto actualizado y la fecha de actualización.

Respuesta esperada:

```
{  
  "name": "Juan Pérez",  
  "job": "Senior QA",  
  "updatedAt": "2025-04-05T14:35:00.000Z"  
}
```

DELETE: Eliminar un registro

¿Qué hace? Le pide al servidor que borre un registro.

Ejemplo real:

DELETE <https://reqres.in/api/users/2>

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** https://reqres.in/api/users/2
- Status:** 204 No Content
- Time:** 193 ms
- Size:** 1 KB
- Save Response:** (button)

KEY	VALUE	DESCRIPTION
Key	Value	Description

The interface also shows tabs for Params, Authorization, Headers (8), Body, Pre-request Script, Tests, and Settings. The Body tab is selected, and the response is displayed in a text format.

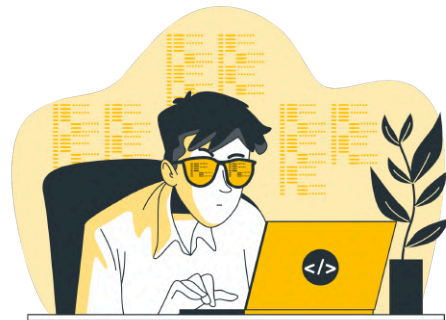
¿Qué devuelve? Generalmente, un **código de estado 204** (sin contenido), que indica que la operación fue exitosa.

Resumen

Método	Acción	Tiene Body	Devuelve Datos
GET	Consultar información	✗	✓ Sí
POST	Crear nuevo recurso	✓ Sí	✓ Sí
PUT	Actualizar recurso	✓ Sí	✓ Sí
DELETE	Eliminar recurso	✗	➖ Generalmente no

Parte 2: Colecciones en Postman

Una **colección** es un conjunto de solicitudes agrupadas de forma ordenada. Es ideal para organizar las pruebas de una API: podés tener una colección por proyecto, por funcionalidad, por microservicio, etc.



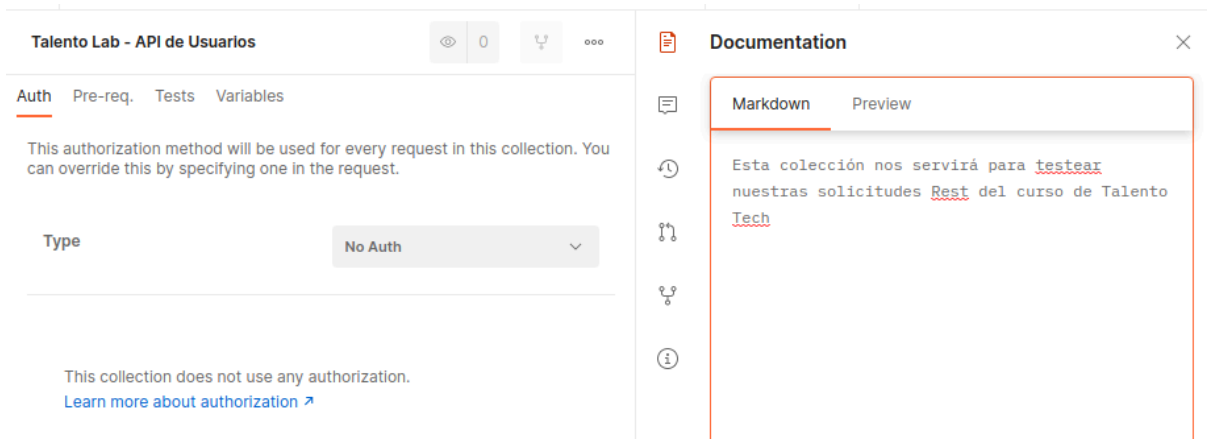
¿Para qué sirve una colección?

- Tener todos los endpoints del proyecto organizados.
- Compartir fácilmente tus pruebas con otros testers o developers.
- Ejecutar muchas pruebas seguidas (con Collection Runner o Newman).
- Documentar tu trabajo de QA de forma clara y reutilizable.

Ejemplo guiado: Crear una colección en Postman

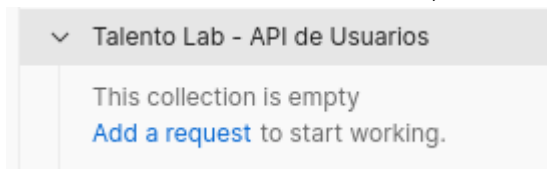
Paso 1: Crear la colección

1. Abrió Postman.
2. Hacer clic en el botón “**Collections**” en la barra lateral izquierda.
3. Presioná “**+ New Collection**”.
4. Poné un nombre, por ejemplo:
Talento Lab - API de Usuarios
5. (Opcional) Agregá una descripción breve para documentarla.



Paso 2: Agregar una solicitud GET

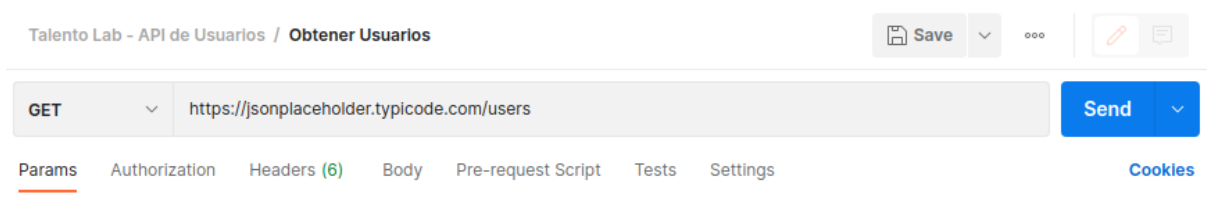
1. Dentro de la colección, hacé clic en **“Add a request”**.



2. Nombrá la solicitud como **Obtener usuarios**.

En el campo URL, escribí: **https://jsonplaceholder.typicode.com/users**

3. Elegí el método **GET**.
4. Hacé clic en **“Save to collection”** y elegí tu colección.



Paso 3: Agregar una solicitud **POST**

1. Repetí el proceso anterior y llamá esta solicitud: **Crear usuario**.

Método: **POST**

URL: <https://regres.in/api/users>

2. Ir a la pestaña “**Body**”, elegir **raw** y **JSON**.

Pegá este JSON:

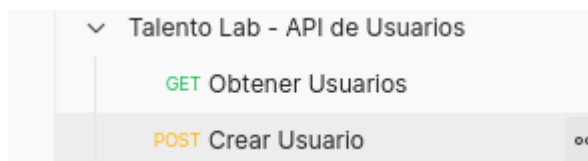
```
{
  "name": "Juan",
  "job": "QA Tester"
}
```

3. Guardá la solicitud en la misma colección.

Paso 4: Revisar tu colección

Ahora tu colección debería tener al menos dos requests:

- **Obtener usuarios** (GET)
- **Crear usuario** (POST)



Podés seguir agregando las otras solicitudes (**PUT**, **DELETE**) siguiendo el mismo patrón.

✓ Buenas prácticas con colecciones

- Agrupá los endpoints por funcionalidad (por ejemplo: usuarios, reservas, login).
- Nombrá las solicitudes claramente.
- Agregá descripciones a cada request para documentar qué prueba realiza.
- Usá variables y entornos (lo veremos en la próxima parte).

Parte 3: Variables y Entornos en Postman

¿Qué es un entorno en Postman?

Un **entorno** es un conjunto de variables que te permite cambiar fácilmente información como URLs, tokens o IDs sin modificar cada solicitud una por una.

¿Para qué sirve?

- Para cambiar entre distintos entornos: **Desarrollo**, **Staging**, **Producción**.
- Para evitar repetir información (como la URL base).
- Para automatizar pruebas y facilitar el mantenimiento de la colección.



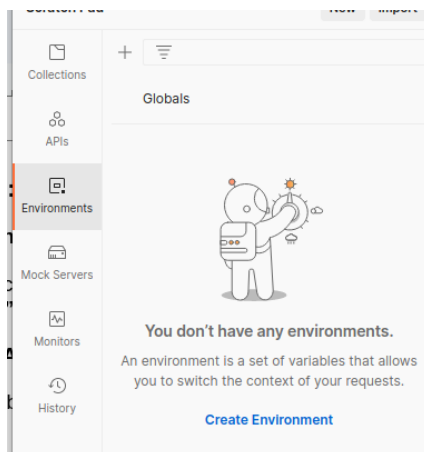
Tipos de variables en Postman

- **Variables de entorno:** específicas para un entorno (ej. URL base, tokens).
- **Variables globales:** se usan en todos los entornos (menos recomendable).
- **Variables locales:** sólo dentro de una solicitud.
- **Variables de colección:** disponibles para todas las requests de una colección.

Ejemplo guiado: Crear un entorno con variables

Paso 1: Crear el entorno

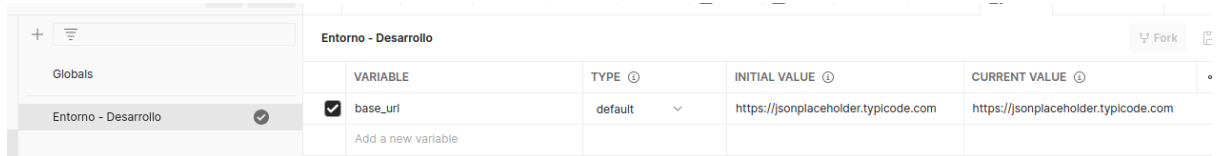
1. En Postman, hacé clic en “**Environments**”.



2. Hacé clic en “**Create Environment**”.
3. Poné como nombre:
Entorno - Desarrollo

Agregá una variable llamada **base_url** y como valor poné:
<https://jsonplaceholder.typicode.com>

4. Guardá el entorno



VARIABLE	TYPE	INITIAL VALUE	CURRENT VALUE
<input checked="" type="checkbox"/> base_url	default	https://jsonplaceholder.typicode.com	https://jsonplaceholder.typicode.com
Add a new variable			

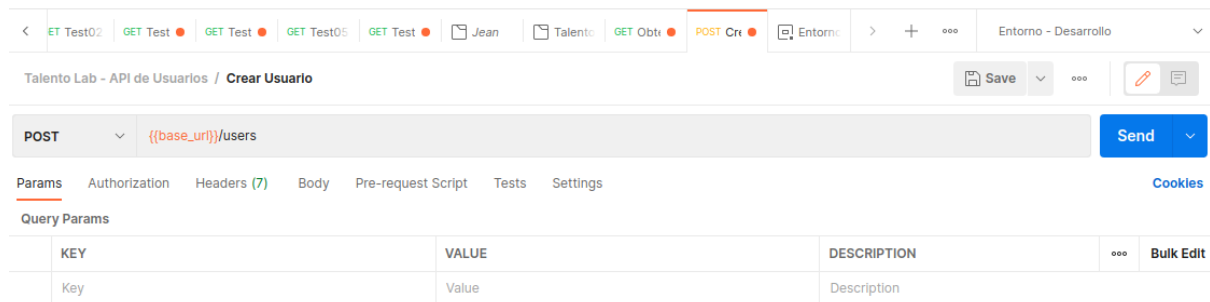
Paso 2: Usar la variable en una solicitud

1. Abrí la solicitud **GET** que hiciste antes (**Obtener usuarios**).

En lugar de usar la URL completa, reemplazala por:

`{{base_url}}/users`

2. En la parte superior derecha de Postman, seleccioná el entorno **Entorno - Desarrollo**
3. Hacé clic en **Send**.



¡La solicitud se ejecuta igual que antes! Pero ahora estás usando una variable, lo que te permite cambiar la URL de todo el proyecto en un solo lugar si es necesario.

Ejemplo real de utilidad

Supongamos que tenés estas 3 URLs:

- Desarrollo: <https://dev.api.talentolab.com>
- Staging: <https://staging.api.talentolab.com>
- Producción: <https://api.talentolab.com>

En lugar de crear nuevas solicitudes, podés crear un entorno para cada una con su propia variable `base_url`, y así reutilizar todas tus pruebas sin duplicar nada.

✓ Buenas prácticas con entornos

- Usá `{{base_url}}` siempre que puedas.
- Guardá tus entornos junto con la colección.
- Si vas a compartir las pruebas, exportá también el entorno.

Parte 4: Aserciones y Validaciones en Postman

¿Qué es una aserción?

Una **aserción** es una afirmación que hacemos sobre la respuesta de una API. Sirve para validar si el comportamiento fue correcto. Si la afirmación no se cumple, el test falla.

En Postman, las aserciones se escriben en JavaScript dentro de la pestaña **Tests**.

¿Para qué se usan?

- Para verificar el código de estado HTTP (200, 201, 404, etc.).
- Para confirmar que el contenido de la respuesta es el esperado.
- Para validar la estructura del JSON (por ejemplo, con JSON Schema).
- Para automatizar pruebas sin intervención manual.

Ejemplo básico de validaciones

Supongamos que hiciste una solicitud **GET** a:

<https://jsonplaceholder.typicode.com/users>

En la pestaña **Tests**, podés agregar este código:

```
pm.test("El código de estado es 200", function () {  
    pm.response.to.have.status(200);  
});  
pm.test("La respuesta es un JSON válido", function () {  
    pm.response.to.be.json;  
});
```

Si la API responde correctamente, estas pruebas pasarán .

Ejemplo de validación de contenido

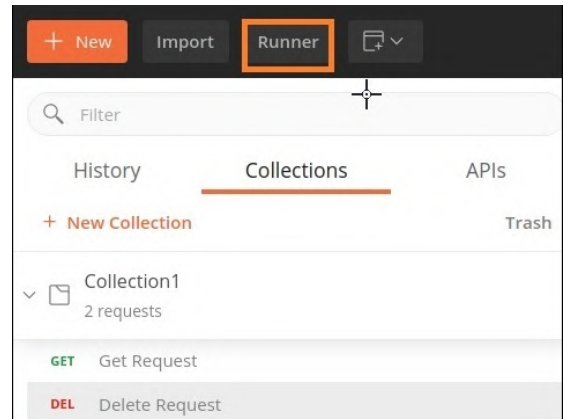
```
const jsonData = pm.response.json();  
  
pm.test("El nombre del primer usuario es Leanne Graham", function () {  
    pm.expect(jsonData[0].name).to.eql("Leanne Graham");  
});
```

Esto comprueba que el primer elemento del array tenga el nombre esperado.

Parte 5: Collection Runner (Ejecutar pruebas por lotes)

¿Qué es el Collection Runner?

Es una herramienta de Postman que te permite **ejecutar todas las solicitudes de una colección**, una tras otra, como un “mini test automático”.



¿Para qué se usa?

- Para validar muchos endpoints seguidos.
- Para probar con múltiples datos (data-driven testing).
- Para simular flujos completos del sistema (por ejemplo: login + creación + consulta + borrado).
- Para ahorrar tiempo y evitar ejecuciones manuales una por una.

Ejemplo guiado: Ejecutar una colección

Paso 1: Armar tu colección

Asegurate de tener al menos dos solicitudes con tests en tu colección, como vimos antes (GET y POST, por ejemplo).

Paso 2: Abrir el Collection Runner

1. Hacé clic en la colección desde la barra lateral.
2. Presioná el botón “Run” (es un ícono de ▶).
3. Se abre una ventana llamada **Collection Runner**.

Paso 3: Ejecutar

1. Seleccioná el entorno que creaste (si tenés uno).
2. (Opcional) Podés cargar un archivo .csv o .json con datos para pruebas múltiples.
3. Hacé clic en **Start Run**.

Postman va a ejecutar todas las requests con sus validaciones y te mostrará los resultados (verde si pasa, rojo si falla).

Resultado esperado

- Vas a ver cuántas pruebas pasaron o fallaron.
- Podés ver los detalles de cada solicitud.
- Si algo falla, podés analizar la respuesta y ajustar tu test.

Buenas prácticas con Collection Runner

- Nombrá bien cada test para entender qué está validando.
- Usá variables si querés repetir pruebas con distintos valores.
- Guardá y exportá los resultados si necesitás documentarlos.

Material Complementario

- [Documentación oficial de Postman](#)
- [Guía práctica de variables y entornos en Postman](#)
- [JSONPlaceholder - API pública para pruebas](#)
- [API Regres para simular operaciones POST, PUT y DELETE](#)
- [Validación de JSON online \(JSONLint\)](#)

Próximos Pasos

En la próxima clase vamos a dar un paso más profundo en el análisis de calidad: incorporaremos el uso de métricas en QA para evaluar el desempeño de nuestras pruebas y detectar oportunidades de mejora.

Lo que veremos:

- ¿Qué son los KPIs en testing?
- ¿Cómo medir defectos, tiempos y costos?
- ¿Qué hacer con los resultados de las métricas?
- Comparación con estándares de la industria.
- ¿Cómo interpretar los datos que generamos desde herramientas como Postman, Jenkins, o Excel?

Tu desafío como QA no solo es encontrar errores, sino entender el **impacto de esos errores en el proyecto**. Para eso, necesitás hablar el lenguaje de los números. En la próxima clase aprenderemos a hacerlo.

¡Otro día en Talento Lab!



Silvia y Matías están entusiasmados con tu avance. Después de haber entendido cómo funcionan las APIs y los métodos HTTP, llegó el momento de profesionalizar el proceso de testing.

Silvia te propone que empieces a documentar todas tus pruebas de APIs usando una herramienta que el equipo ya tiene integrada: **Postman**. Matías,

por su parte, te pide que no sólo pruebes los endpoints, sino que los organices en una **colección clara, con validaciones automáticas**, para que puedan ejecutarse en cada despliegue.



“Si tus pruebas están bien estructuradas y documentadas, cualquier miembro del equipo podrá usarlas y saber qué está fallando, incluso si vos no estás conectado”, te dice Matías.

Además, Silvia menciona que al final del sprint, tus pruebas formarán parte del entregable para el cliente, así que tienen que ser reutilizables, entendibles y fáciles de correr. Tu desafío ahora no es solo probar, sino dejar un rastro claro de **qué se probó, cómo y con qué resultados**.

Ejercicio Práctico

Parte 1: Crear la colección

1. Abrí Postman y creá una colección llamada:
Talento Lab - API de Usuarios
2. Agregá las siguientes solicitudes dentro de esa colección:

a. GET - Listar usuarios

URL: <https://jsonplaceholder.typicode.com/users>

Método: GET

Test:

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

b. POST - Crear usuario

URL: <https://reqres.in/api/users>

Método: POST

Body (raw, JSON):

```
{  
    "name": "María",  
    "job": "QA Tester"  
}
```

Test:

```
pm.test("Status 201", function () {  
    pm.response.to.have.status(201);  
});  
  
pm.test("Nombre correcto", function () {  
    const jsonData = pm.response.json();  
    pm.expect(jsonData.name).to.eql("María");  
});
```

c. PUT - Actualizar usuario

URL: <https://reqres.in/api/users/2>

Método: PUT

Body:

```
{  
    "name": "María",  
    "job": "Senior QA"  
}
```

Test:

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

```
});
```

d. DELETE - Eliminar usuario

URL: <https://reqres.in/api/users/2>

Método: DELETE

Test:

```
pm.test("Status 204", function () {  
    pm.response.to.have.status(204);  
});
```

Parte 2: Documentar

Por cada solicitud, completá en Postman los siguientes campos:

- Nombre claro
- Descripción breve de qué hace y qué valida
- Variables usadas (si aplican)
- Tests incluidos

Parte 3: Ejecutar con Collection Runner

1. Abrí el Collection Runner desde la colección.
2. Ejecutá todas las pruebas.
3. Sacá una captura de los resultados.
4. Revisá si todas las pruebas pasaron.



Buenos Aires
aprende
Agencia de Habilidades para el Futuro

BA Buenos
Aires
Ciudad