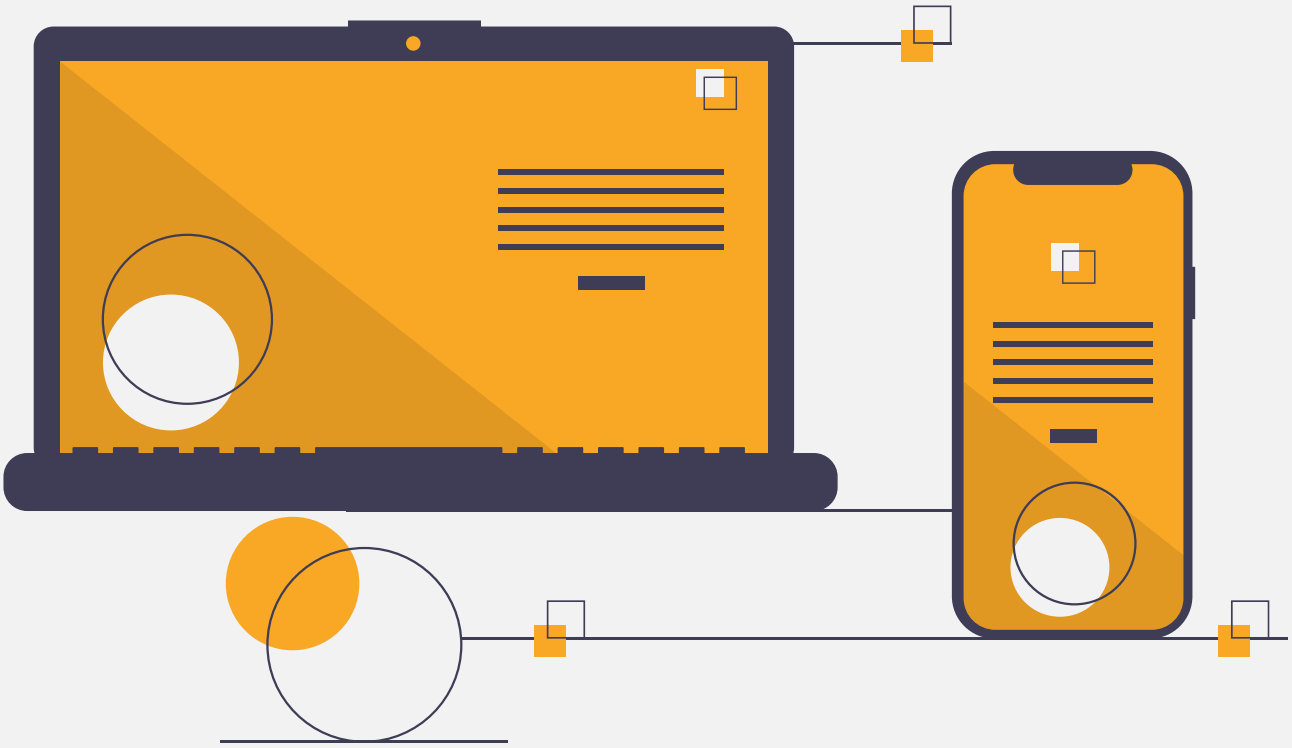
# HTTP y Servidores WEB

## SRI Tema 4

### Ejercicio 1 y 2

Miguel Lillo Romero

2º ASIR



# Índice:

# Introducción

Solución:

## ¿Qué es Postman?

**Postman** es una plataforma con **interfaz gráfica** que ayuda a **crear** y **probar peticiones HTTP**. Es especialmente útil cuando trabajas en proyectos complejos, ya que permite **guardar** y **organizar** diferentes **peticiones**, y **analizar** las respuestas de manera más detallada.

## ¿Qué es cURL?

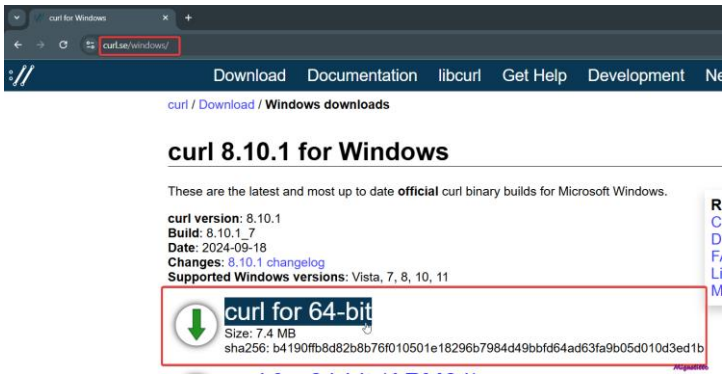
cURL es una herramienta de **línea de comandos** que permite realizar **peticiones HTTP** de forma rápida. Aprender a utilizar cURL es útil porque permite trabajar con **APIs** desde la terminal, lo cual es valioso en entornos de desarrollo y automatización.

## Escenario

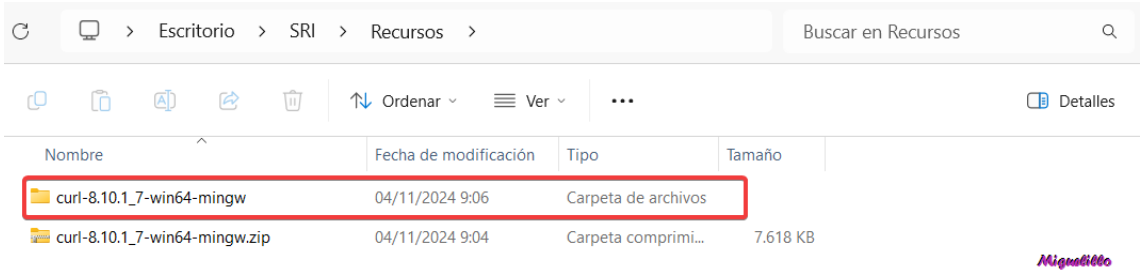
Voy a utilizar mi propio equipo un **Windows 11** de **64bits** con 1 tarjeta de red en modo **DHCP**

## Instalación

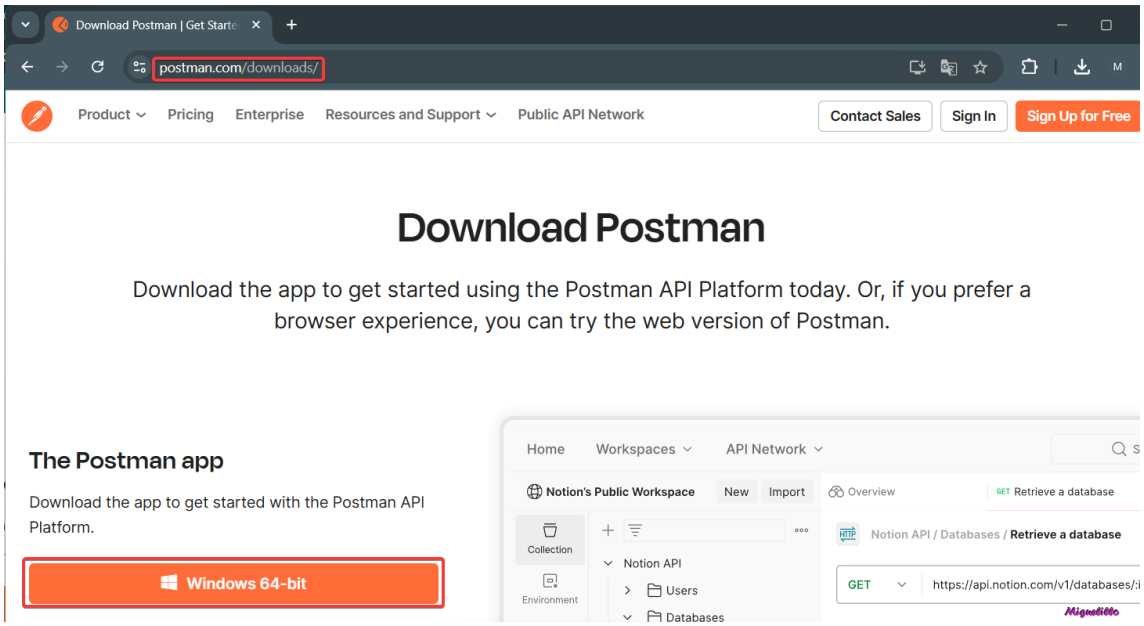
Primero instalaremos **cURL** para ello vamos a su página web y descargamos el archivo comprimido



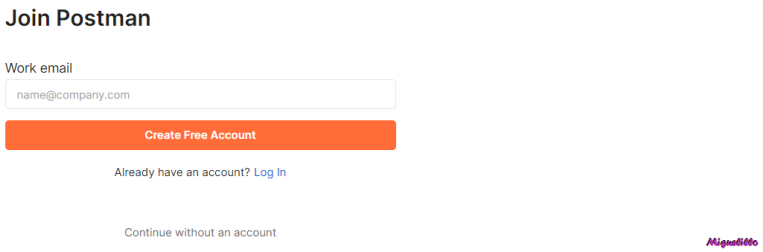

Extraemos el archivo comprimido



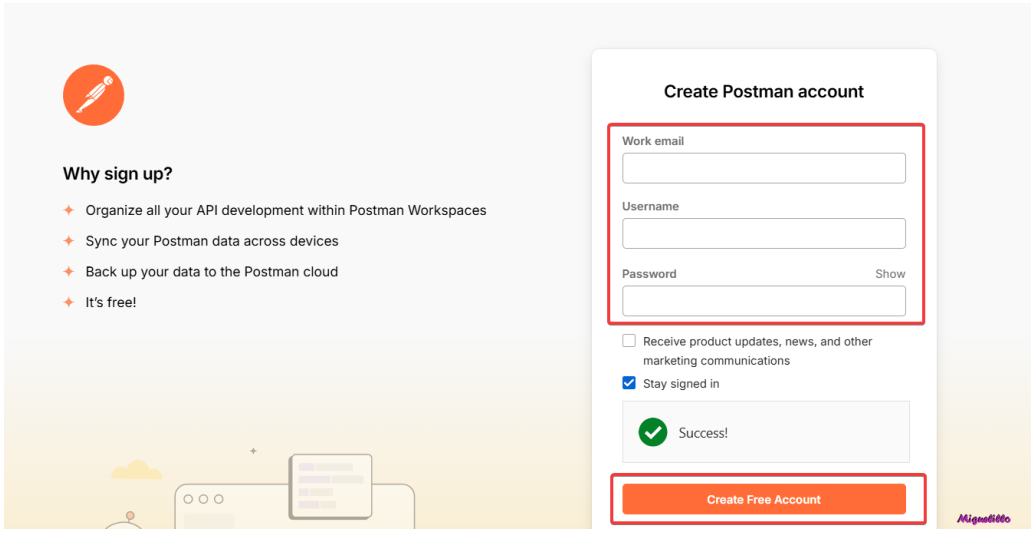
## Descargamos el instalador de **Postman**



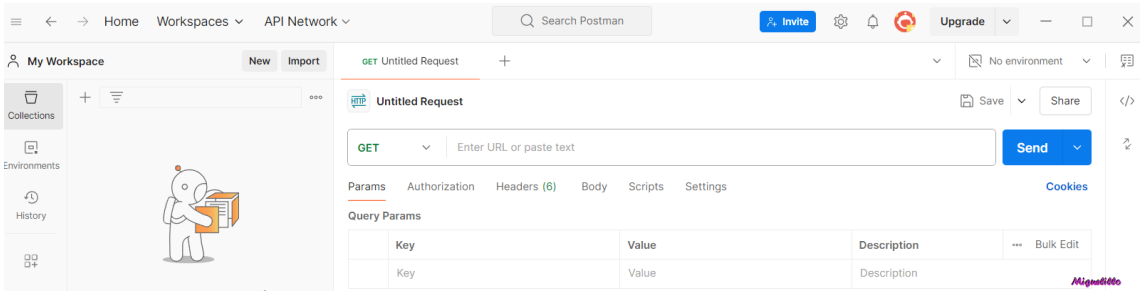

## Ejecutamos él .exe


## Nos creamos una cuenta gratuita



## Ya estaría instalado



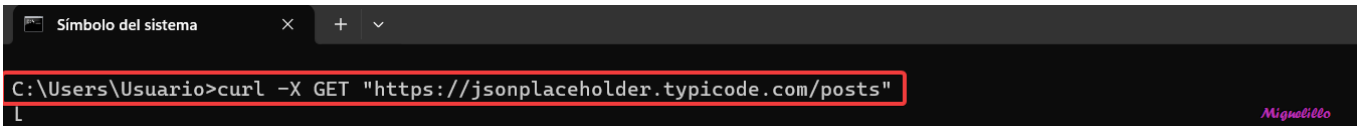
# Ejercicio 1 Peticiones GET Básicas

## a) Obtener todos los posts

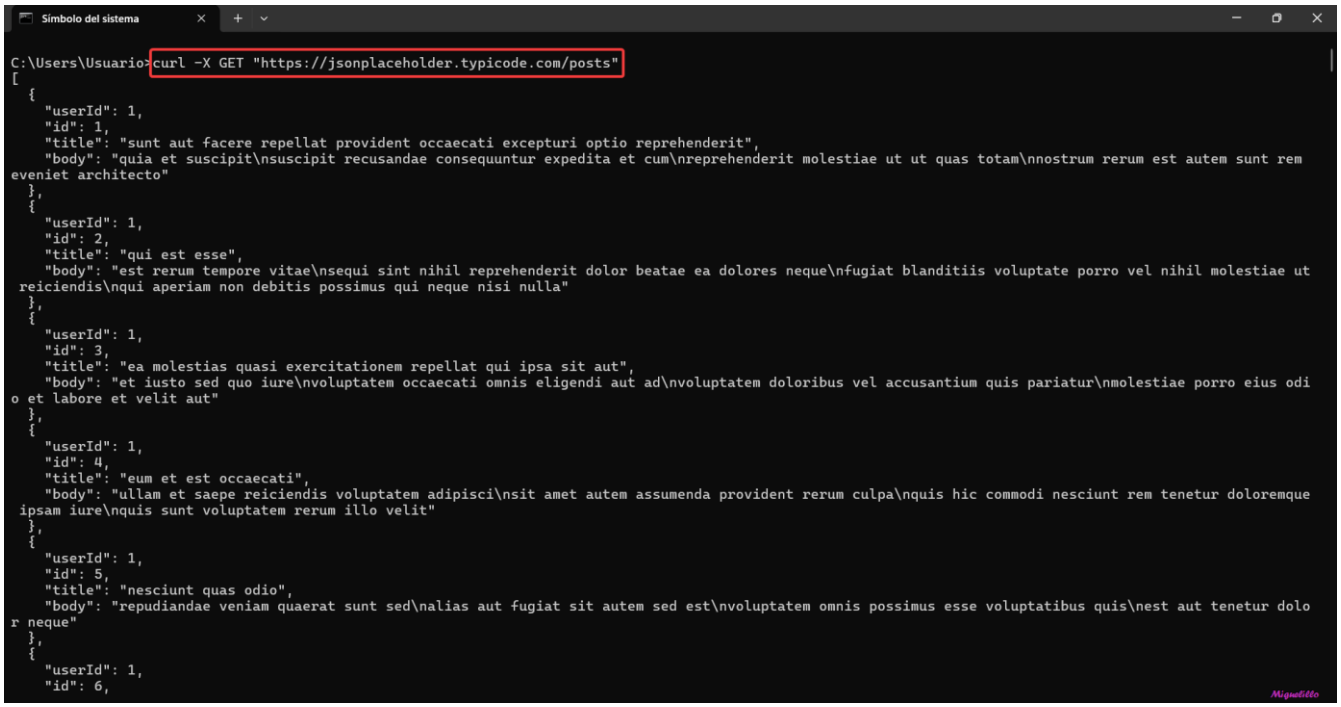
Solución:

Ejecutamos el comando
**curl -X GET https://jsonplaceholder.typicode.com/posts**

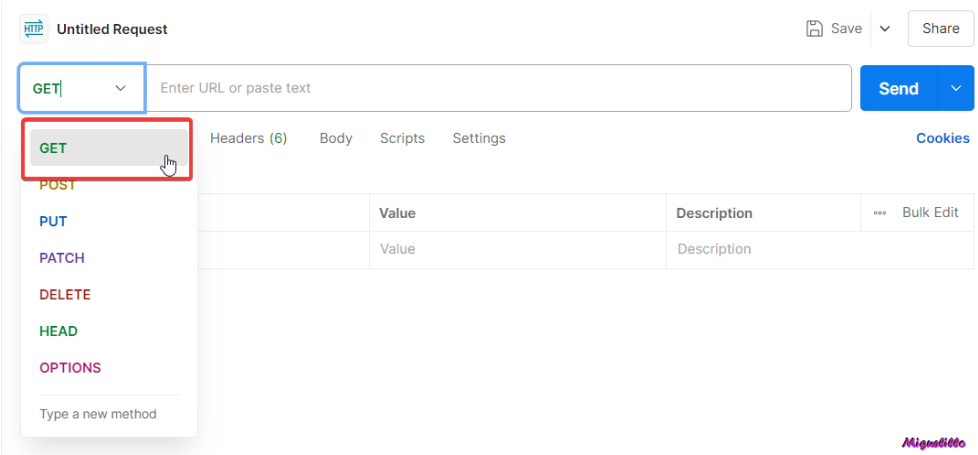
para obtener una lista completa de posts.




Una vez ejecutado vemos una respuesta en formato JSON con una lista de objetos, donde cada objeto representa un "post" con información como userId, id, title, y body.

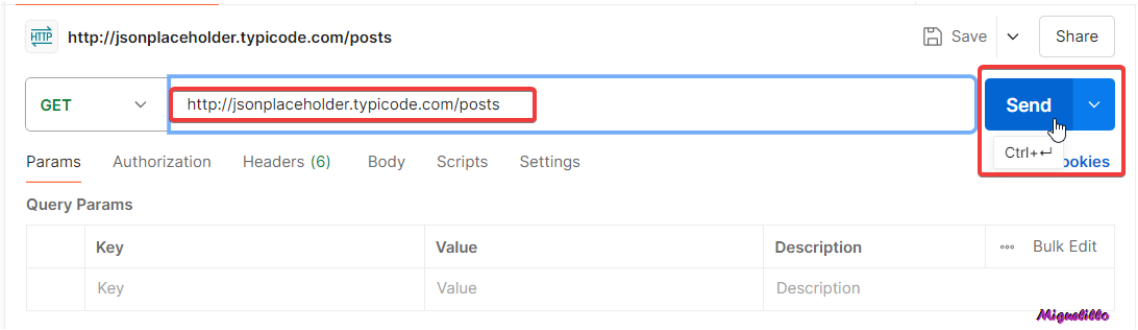



Vamos a postman, Selecciona el método **GET** en el menú desplegable.

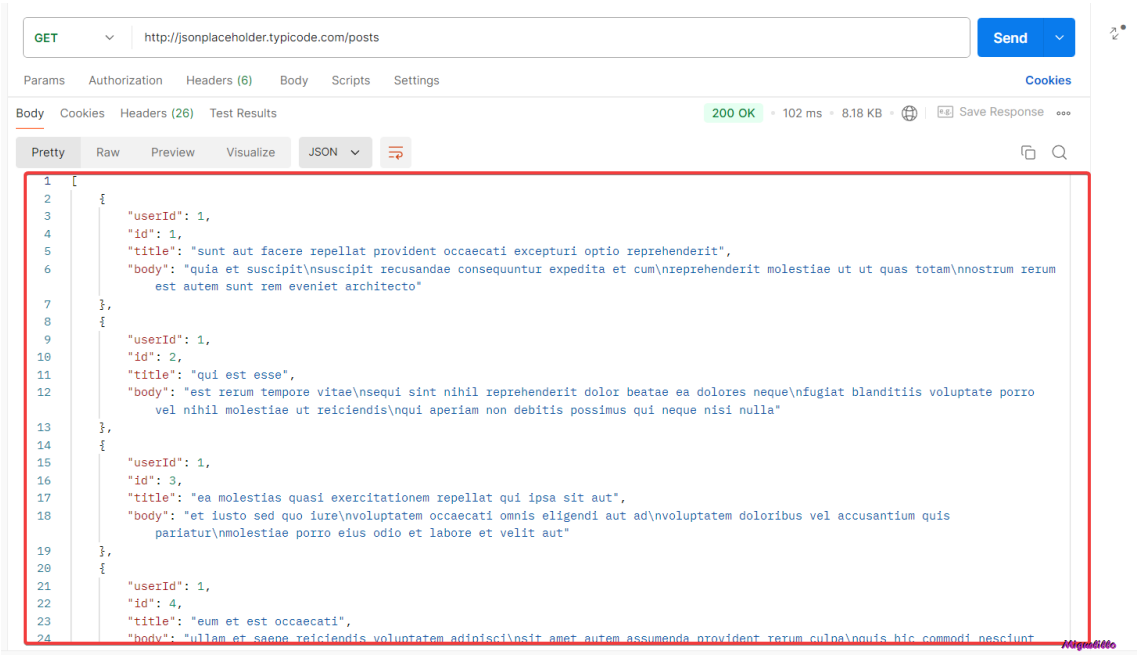


En la barra de **URL**, ingresamos la siguiente dirección **http://jsonplaceholder.typicode.com/posts**

y clicamos en el botón **Send**




En la sección de respuesta de Postman, verás la misma lista de **posts** en formato JSON que obtuviste con cURL.



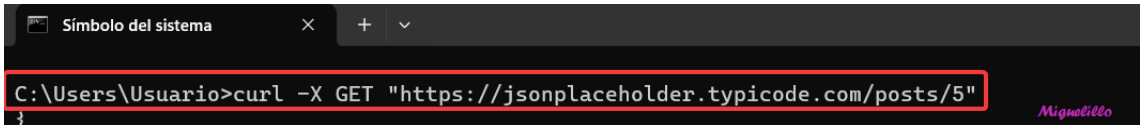
## b) Obtener un post específico (por ejemplo, ID 5)

Solución:

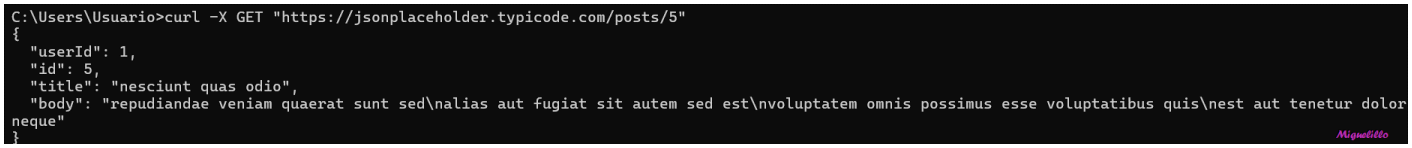
En la terminal, escribimos

**`curl -X GET "https://jsonplaceholder.typicode.com/posts/5"`**

y presionamos **Enter**, este comando realiza una solicitud GET para obtener únicamente el post con el **ID 5**



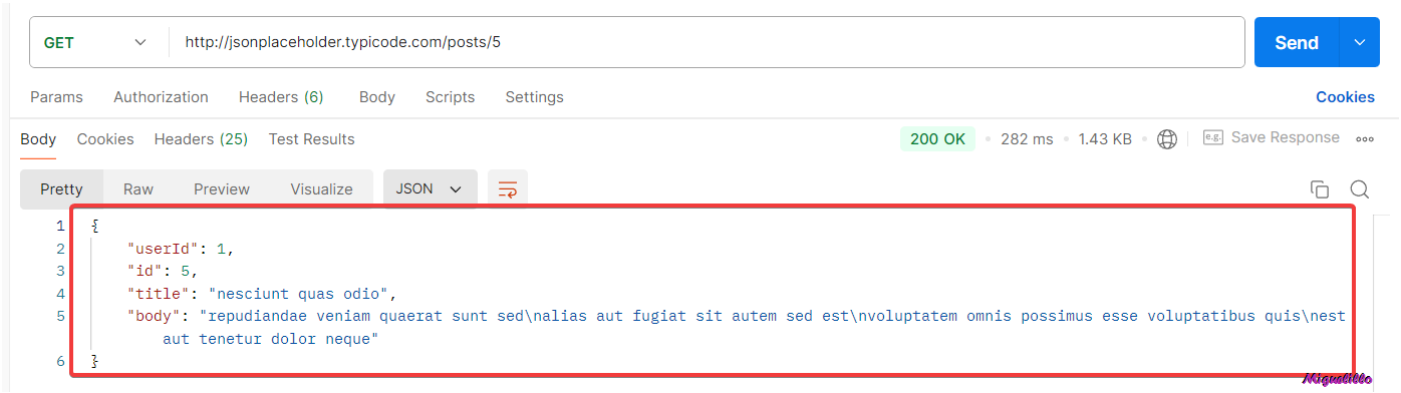

La respuesta mostrará un único objeto JSON con los detalles del post de ID 5, incluyendo userId, id, title, y body




En Postman, cambiamos la URL a

**http://jsonplaceholder.typicode.com/posts/5**


Podemos ver la misma respuesta JSON con los detalles del post con ID 5.

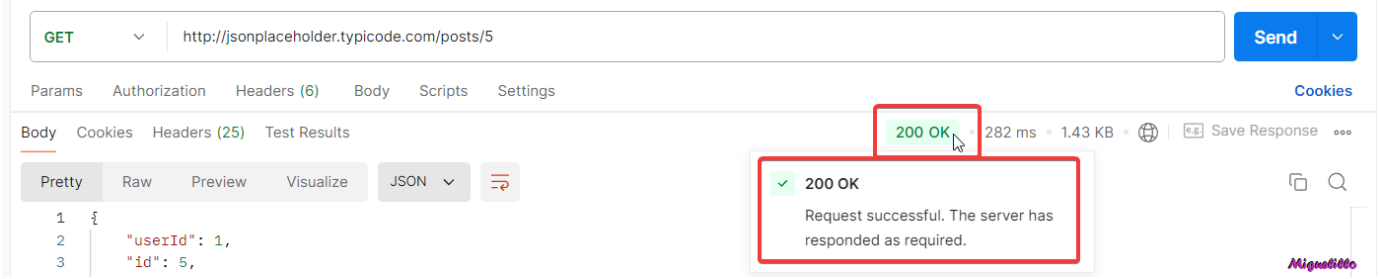


## d) Preguntas

Solución:

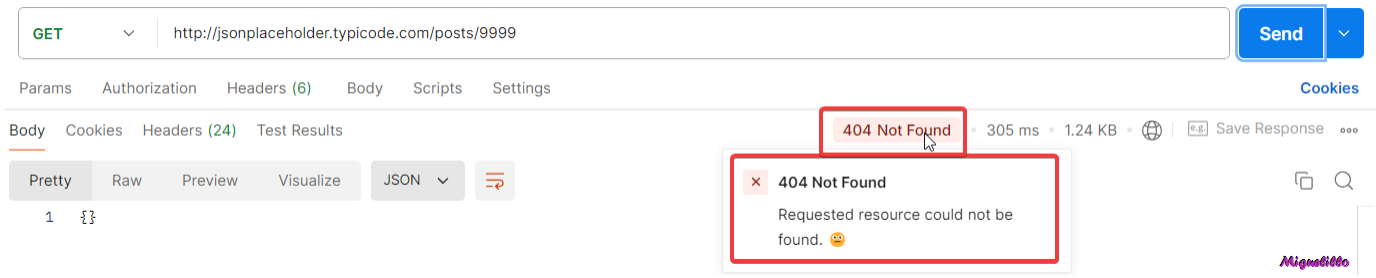
### ¿Qué código de estado devuelve la API cuando accedes a un post específico?

Cuando realizas una solicitud **GET válida** (por ejemplo, para obtener todos los posts o el post con ID 5), la API **devuelve** el código de estado **200 OK**. Este código significa que la solicitud fue exitosa y el servidor devolvió el recurso solicitado.




### ¿Qué sucede si intentas acceder a un post con un ID inexistente?

Si intentas acceder a un post con un ID que **no existe** (por ejemplo, https://jsonplaceholder.typicode.com/posts/9999), el **servidor devolverá** el código de estado **404 Not Found**. Este código **indica** que el recurso solicitado **no** fue **encontrado** en el servidor, y en una aplicación real, se puede utilizar para mostrar un mensaje de error al usuario.



# Ejercicio 2 Uso de Parámetros de Consulta en GET

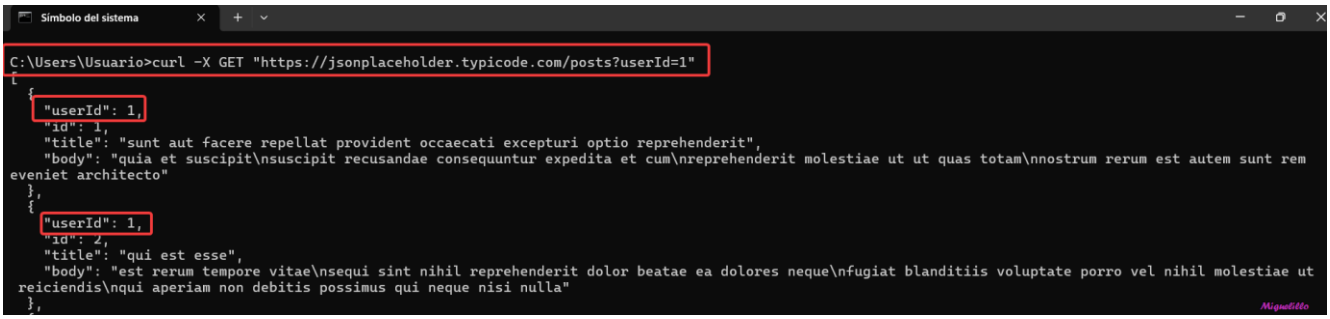
## a) Filtrar posts por usuario (userId=1)

Solución:

Abrimos la terminal y escribimos el siguiente comando

```
curl -X GET "https://jsonplaceholder.typicode.com/posts?userId=1"
```

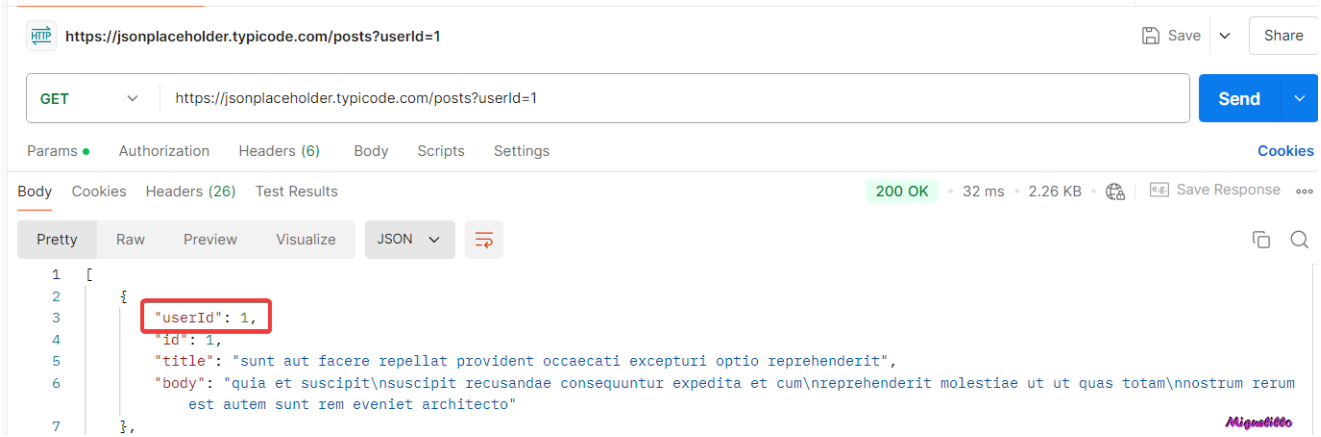
este comando realiza una solicitud GET a la API, añadiendo el parámetro userId=1 para obtener solo los posts del usuario con ID 1




Abrimos **Postman** y seleccionamos el método **GET**.en la barra de URL, ingresamos la siguiente dirección

**`https://jsonplaceholder.typicode.com/posts?userId=1`**

en la respuesta, vemos solo los posts que pertenecen al usuario con userId=1



## b) Preguntas

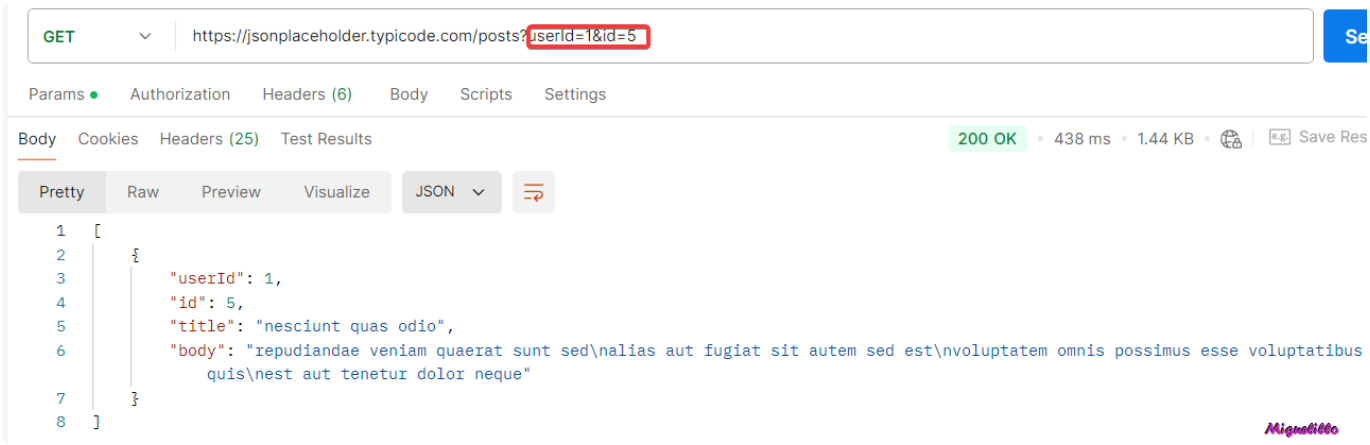
Solución:

### ¿Qué efecto tienen los parámetros de consulta en la respuesta?

Los parámetros de consulta permiten **filtrar** los **datos devueltos** por la API según ciertos criterios. En este caso, el parámetro **userId=1** hace que la API devuelva solo los posts creados por el **usuario** con **ID 1**, en lugar de devolver todos los posts disponibles.

### ¿Qué sucede si se combinan varios parámetros de consulta, por ejemplo, userId=1&id=5?

Cuando se combinan varios parámetros de consulta, como **userId=1&id=5**, la API aplicará **ambos filtros**. Esto significa que solo devolverá el post que **coincida** con **ambos criterios**: que pertenezca al usuario con **userId=1** y tenga un **id=5**. Si no existe ningún post que cumpla con todos los criterios, la respuesta será un **array vacío**.



# Ejercicio 3 Realizar una Petición POST y Validar la Respuesta

## a) Crear un nuevo post

Solución:

Abre la terminal y ejecuta el siguiente comando

```
curl -X POST "https://jsonplaceholder.typicode.com/posts" -H "Content-Type: application/json" -d "{\"title\": \"Nuevo Post\", \"body\": \"Contenido del post\", \"userId\": 1}"
```

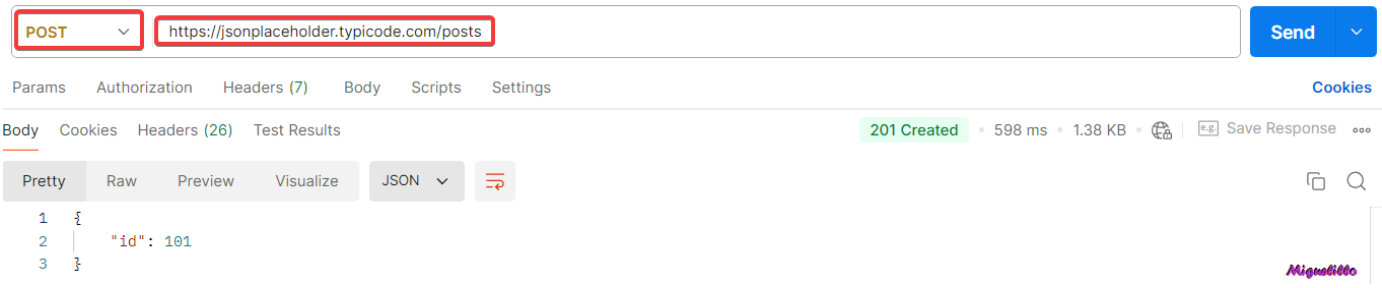
**-X POST** especifica el tipo de solicitud.

**-H "Content-Type: application/json"** define que los datos enviados están en formato JSON.

**-d** permite incluir el cuerpo de la solicitud con el JSON del nuevo post.

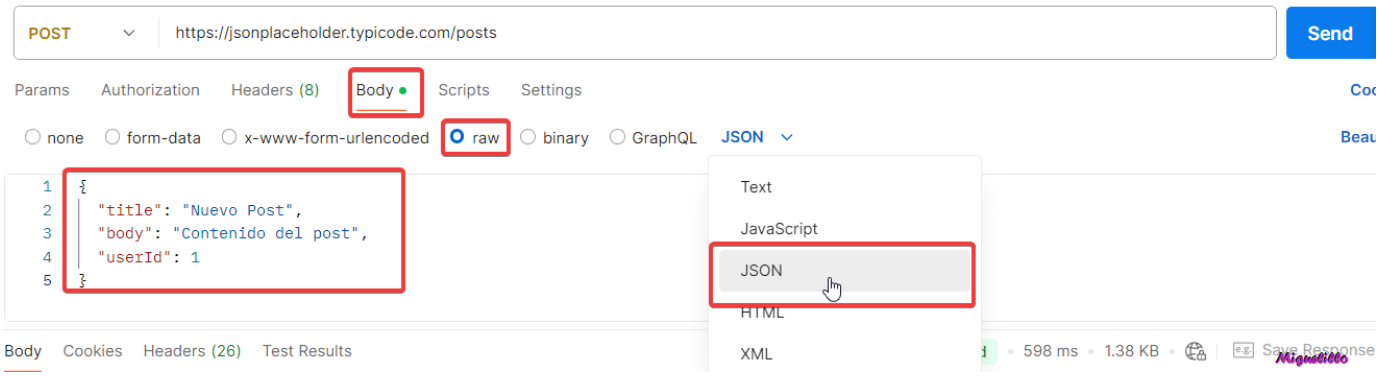

Abre Postman y selecciona el método **POST** en la barra de URL, ingresa la siguiente dirección

**https://jsonplaceholder.typicode.com/posts**



En la pestaña Body, selecciona raw y elige el tipo JSON introduce el siguiente JSON en el cuerpo de la solicitud

```
{
  "title": "Nuevo Post",
  "body": "Contenido del post",
  "userId": 1
}
```

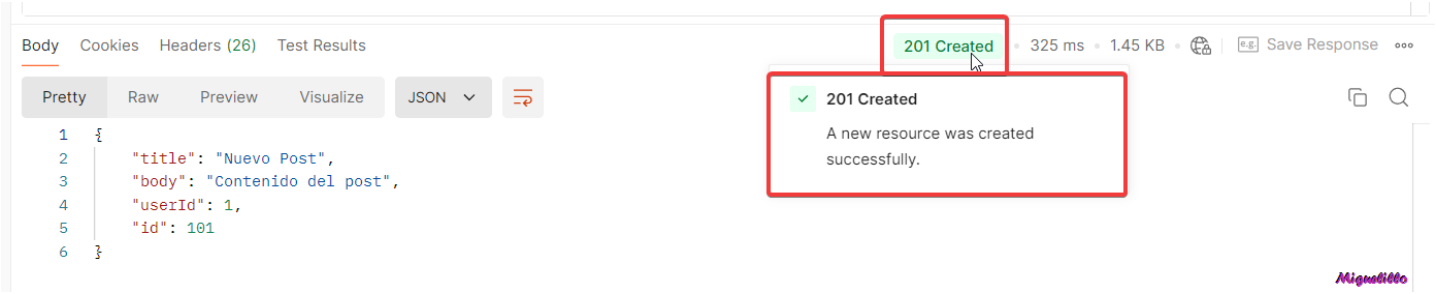


## b) Preguntas

Solución:

### ¿Qué código de estado devuelve la respuesta al crear un nuevo post?

La API devolverá el código de estado 201 Created, que indica que el recurso se ha creado con éxito en el servidor.




### ¿Cuáles son las diferencias en la respuesta de un POST y de un GET?

La respuesta de una solicitud **POST** generalmente incluye el recurso recién creado con su ID asignado, así como el código de estado **201 Created.**

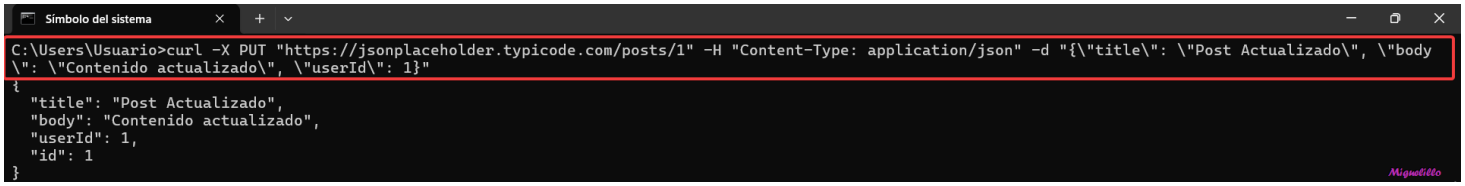
La respuesta de una solicitud **GET** solo proporciona el recurso solicitado o los recursos filtrados sin cambios en el servidor, con el código de estado **200 OK**.

# Ejercicio 4 Actualizar un Post con PUT y PATCH

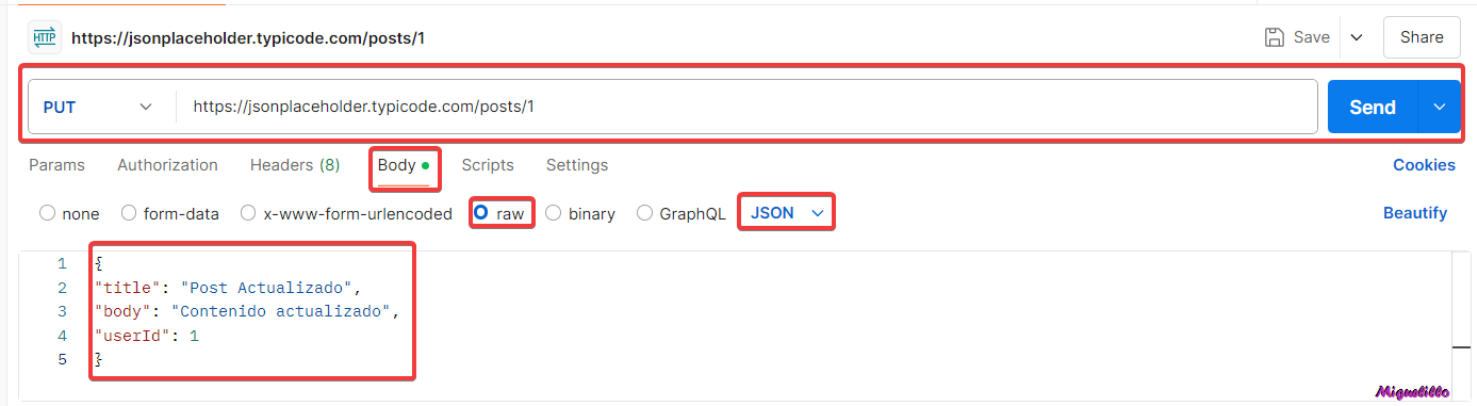
## a) Actualizar un post usando PUT

Solución:

Ejecuta en la consola **`curl -X PUT "https://jsonplaceholder.typicode.com/posts/1" -H "Content-Type: application/json" -d "{\"title\": \"Post Actualizado\", \"body\": \"Contenido actualizado\", \"userId\": 1}"`**




Selecciona **PUT**, ingresa **https://jsonplaceholder.typicode.com/posts/1**, elige **Body → raw → JSON**, y añade el JSON del post actualizado. Haz clic en **Send**.



## b) Actualizar parcialmente un post usando PATCH

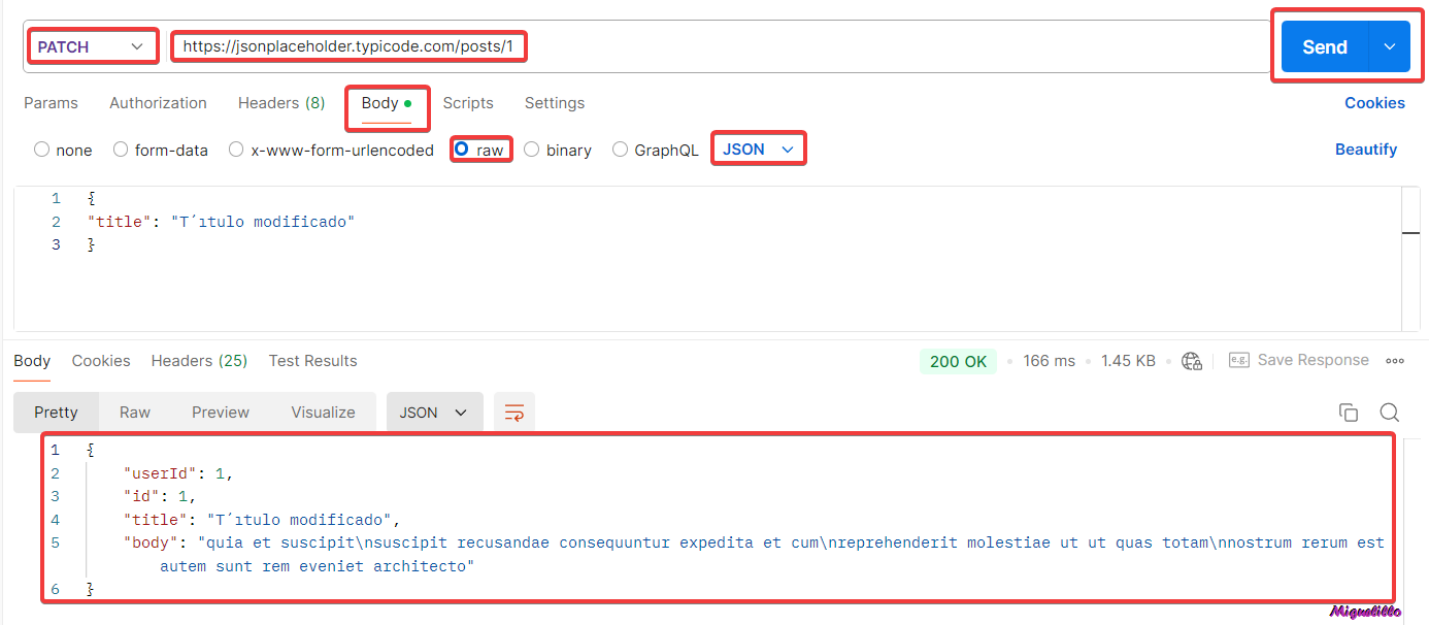
Solución:

Ejecutamos en la consola

```
curl -X PATCH "https://jsonplaceholder.typicode.com/posts/1" -H "Content-Type: application/json" -d "{\"title\": \"Título modificado\"}"
```

```
C:\Users\Usuario>curl -X PUT "https://jsonplaceholder.typicode.com/posts/1" -H "Content-Type: application/json" -d "{\"title\": \"Post Actualizado\", \"body
\": \"Contenido actualizado\", \"userId\": 1}"
{
  "title": "Post Actualizado",
  "body": "Contenido actualizado",
  "userId": 1,
  "id": 1
}
```

Selecciona PATCH, ingresa **https://jsonplaceholder.typicode.com/posts/1** elige **Body > raw > JSON**, y añade solo el campo que deseas modificar. Haz clic en Send



## c) Preguntas

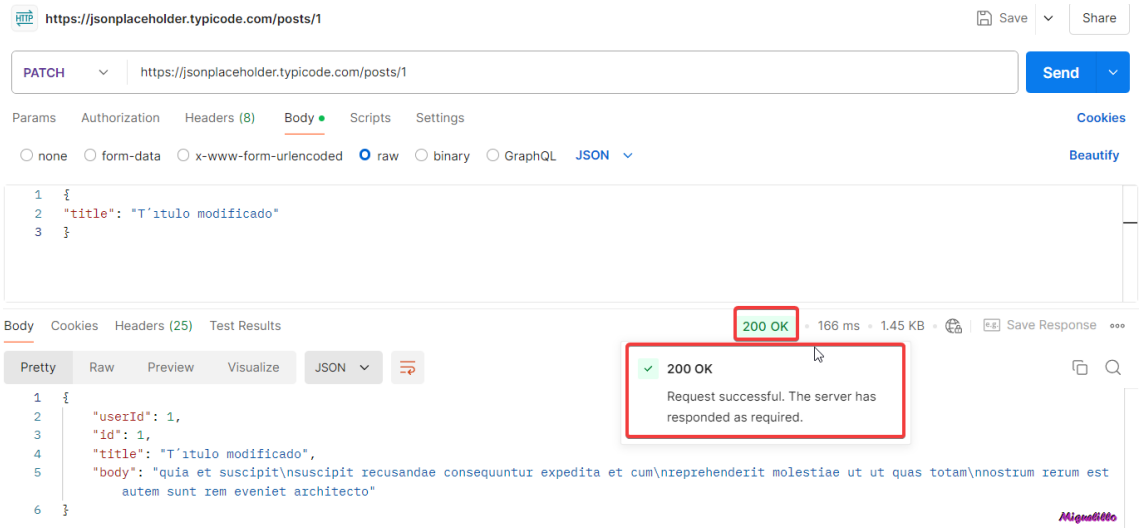
Solución:

### ¿Qué diferencias existen entre las respuestas al usar PUT y PATCH?

**PUT** reemplaza todos los datos del recurso, mientras que **PATCH** actualiza solo los campos especificados

### ¿Qué código de estado se devuelve en cada caso?

Ambos métodos devuelven **200 OK** si la actualización es exitosa.



# Ejercicio 5: Manejo de Errores HTTP

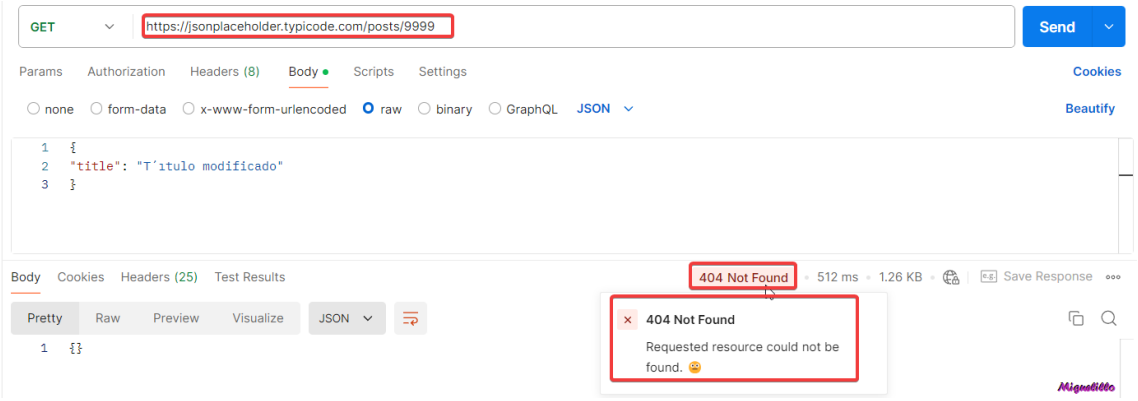
## a) Intenta acceder a un recurso inexistente:

Solución:

Intentamos acceder a un recurso inexistente mediante esta URL

**https://jsonplaceholder.typicode.com/posts/9999**

y nos arroja el error **404 Not Found** ya que el **ID 9999** no existe



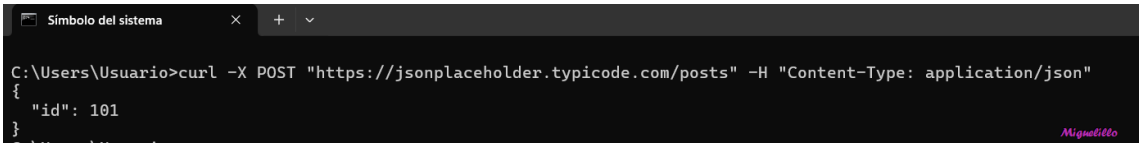
## b) Intenta crear un recurso sin datos:

Solución:

Abrimos la consola

```
curl -X POST "https://jsonplaceholder.typicode.com/posts" -H
"Content-Type: application/json"
```

vemos que no nos arroja un error



## c) Preguntas

Solución:

### ¿Qué código de estado devuelve un recurso inexistente?

Cuando se intenta acceder a un recurso inexistente, el **servidor devuelve** el código de estado **404 Not Found**, indicando que el recurso no se encuentra.

### ¿Qué otros códigos de estado podrían devolver un error y en qué circunstancias?

- **400 Bad Request**: Solicitud mal formada.
- **401 Unauthorized**: Acceso no autorizado.
- **403 Forbidden**: Acceso denegado.
- **500 Internal Server Error**: Error en el servidor.
- **503 Service Unavailable**: Servidor no disponible temporalmente.

# Introducción

Solución:

## ¿Qué es Wireshark?

Wireshark es una herramienta de **análisis de red** de código abierto que permite **capturar** y **examinar** el tráfico que pasa por una red en tiempo real.

## Escenario

Voy a utilizar mi propio equipo un **Windows 11** de **64bits** con 1 tarjeta de red en modo DHCP que ya tiene instalado el **Wireshark**, **cURL** y el **Postman**

## Objetivo

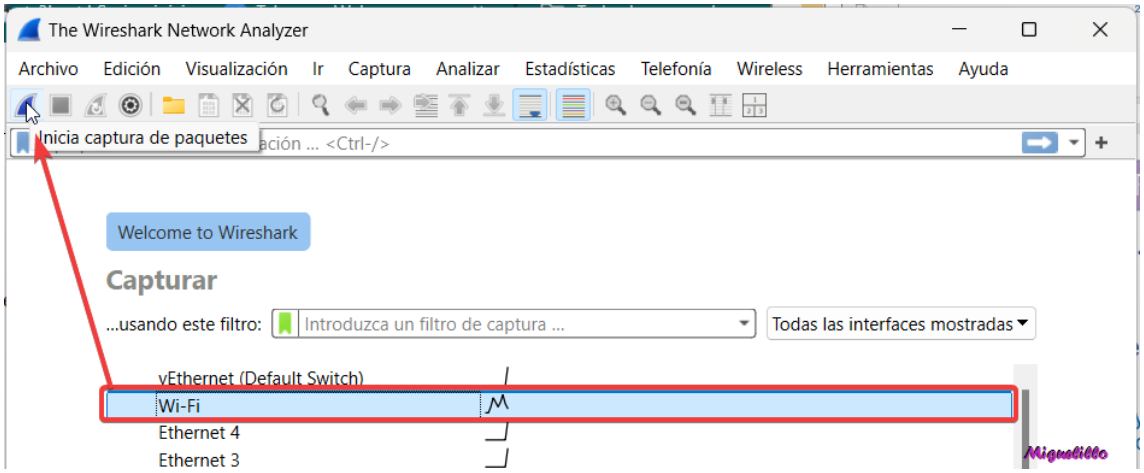
**Inspeccionar** el **tráfico** generado por estas **peticiones** usando **Wireshark**. A través de esta actividad, se busca **familiarizarse** con el **análisis** de las **cabeceras** y el cuerpo de las **solicitudes** y **respuestas** HTTP, así como identificar y **comprender** los **errores HTTP** que puedan surgir.

# Ejercicio 1 Realizar una Petición GET y Capturar el Tráfico en Wireshark

## a) Inicia Wireshark

Solución:

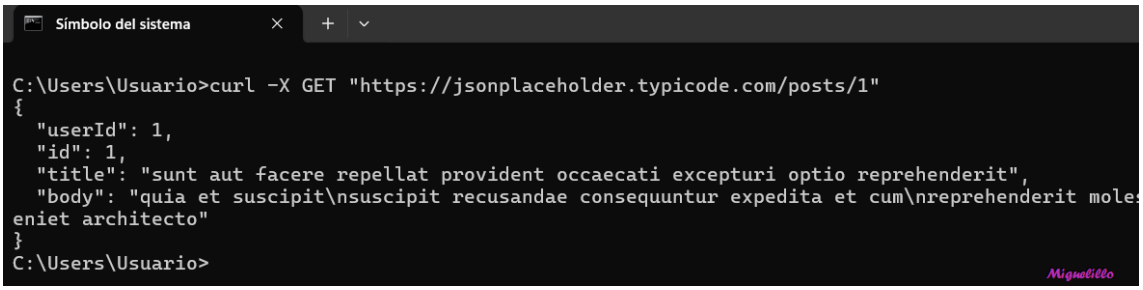
Abre Wireshark y selecciona la **interfaz de red** que **usas** para conectarte a **Internet** y haz clic en el botón **Start**



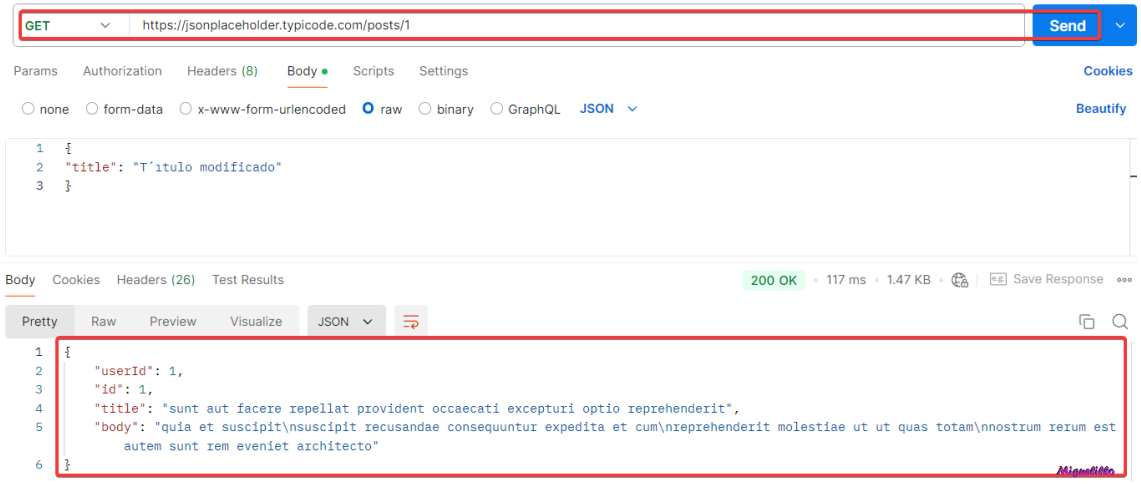
# b) Realiza una petición GET

## Solución:

## Creamos una petición get con cURL



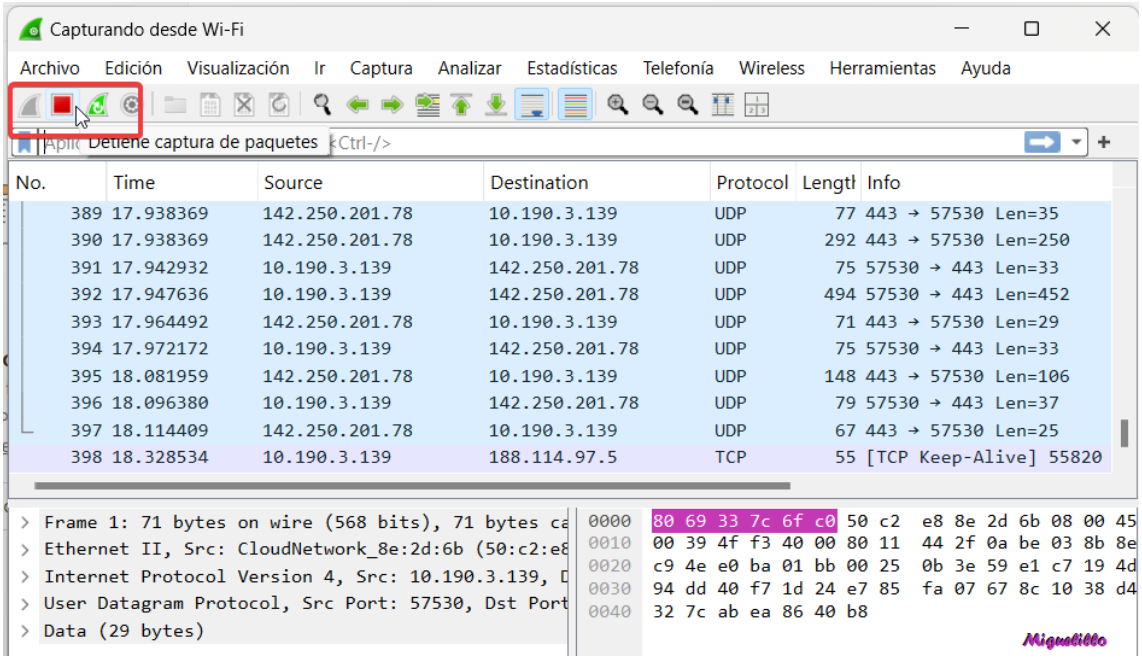

## También se pude realizar en **Postman**



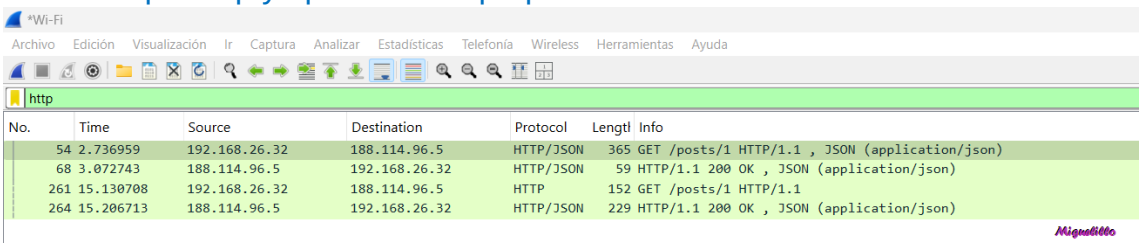
## c) Detén la captura en Wireshark:

### Solución:

En Wireshark, haz clic en el botón **Stop**.




Filtramos por http y aparecen los paquetes



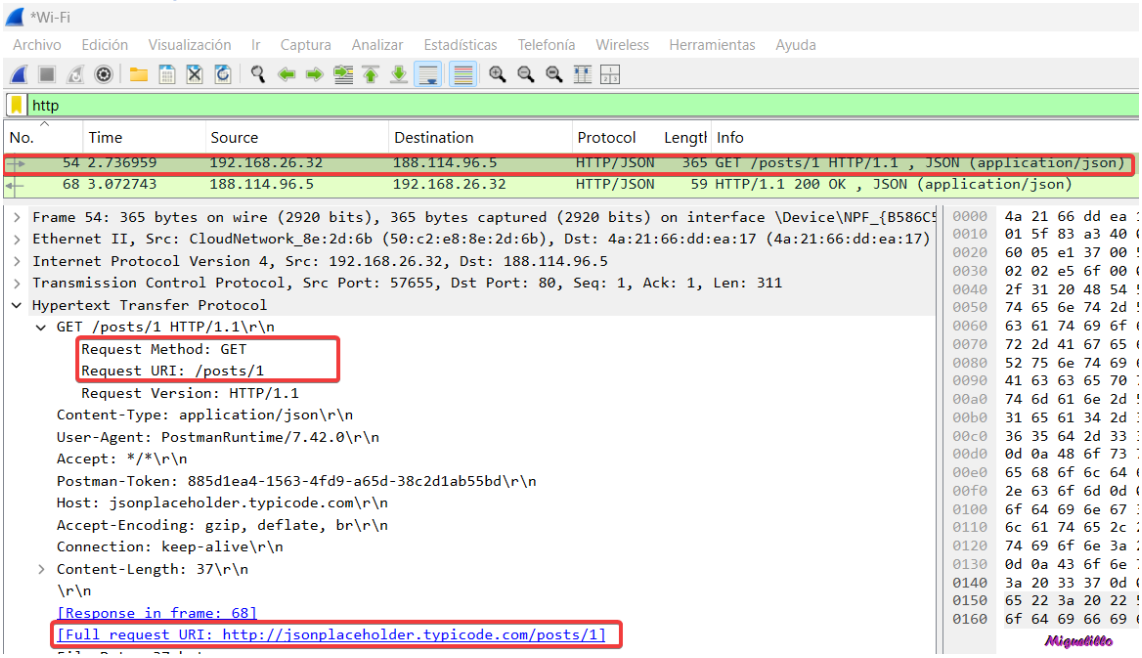

## Incidencia

A la hora de filtrar por paquetes como hemos hecho las peticiones mediante https si filtramos por https no aparecen ya que van encriptados para solucionarlo simplemente haremos el get a http

# Análisis del Paquete HTTP

- ## Cabecera de solicitud

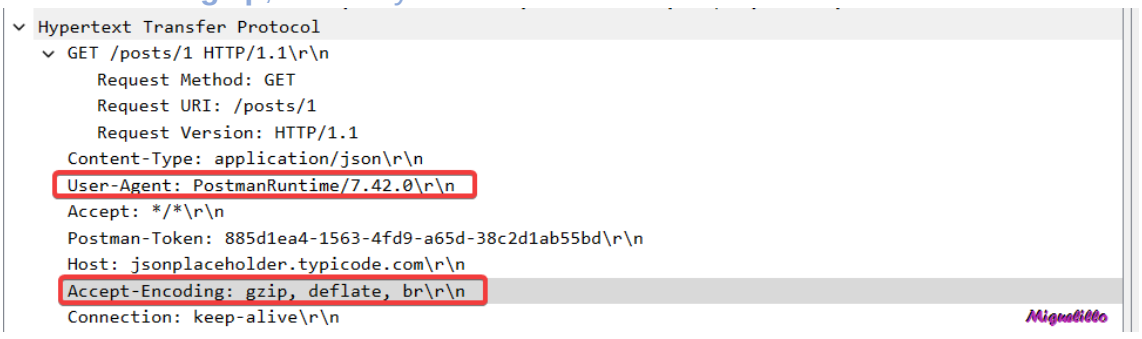
  Vemos que el método que ha utilizado es **GET** y la url solicitada aparece en el full request URI




- ## Cabeceras de cliente

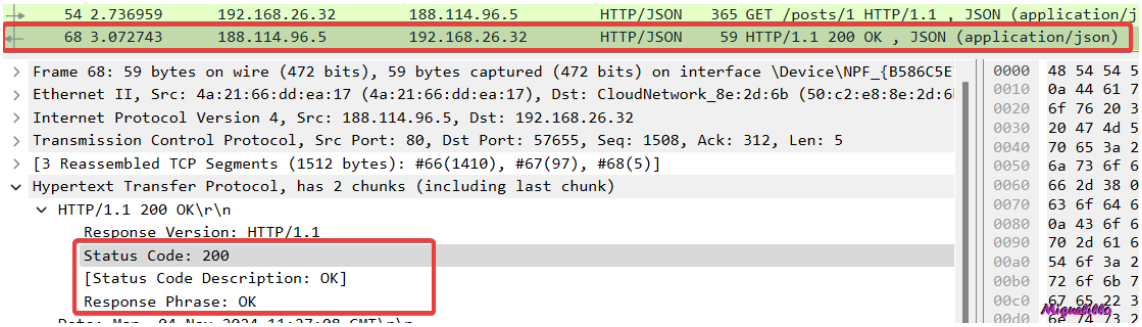
  En el campo **User Agent** indican que la solicitud fue realizada desde **Postman**, con la **versión 7.42.0**

  **Accept-Encoding** muestra que el cliente puede **manejar** compresión en los **formatos gzip**, **deflate** y **Brotli**



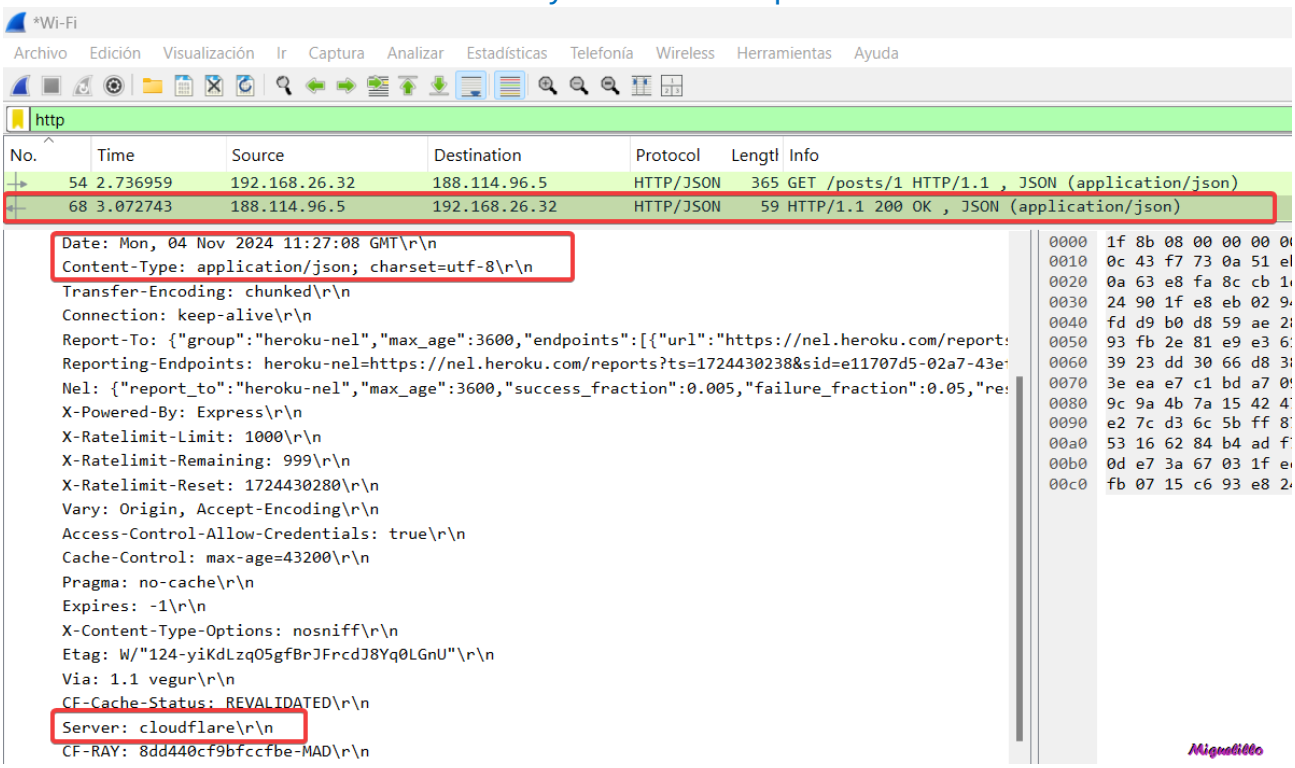
- **Respuesta del servidor:**

Código de estado HTTP: En la sección de **Hypertext Transfer Protocol**, busca **Status Code** que muestra el resultado de la petición.




Cabeceras de respuesta:

- Content-Type: Indica el **tipo** de **contenido** que el servidor está enviando, como application/**json** o text/html.
- Server: Muestra información **sobre** el **servidor** que está respondiendo, si el servidor la proporciona.
- Date: Indica la **fecha** y **hora** de la respuesta del servidor.

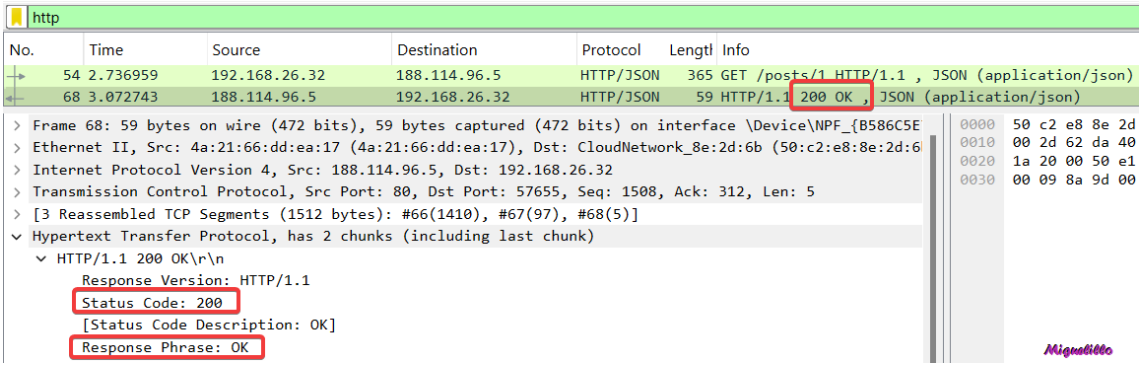


## d) Preguntas

Solución:

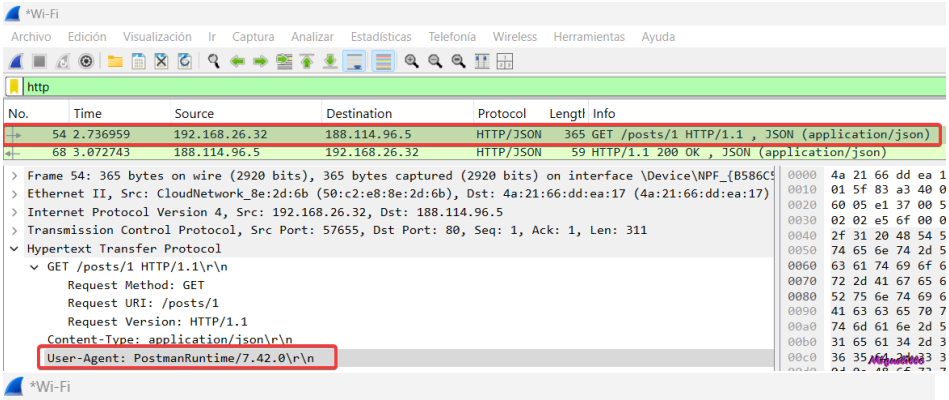
# ¿Qué código de estado aparece en la respuesta?

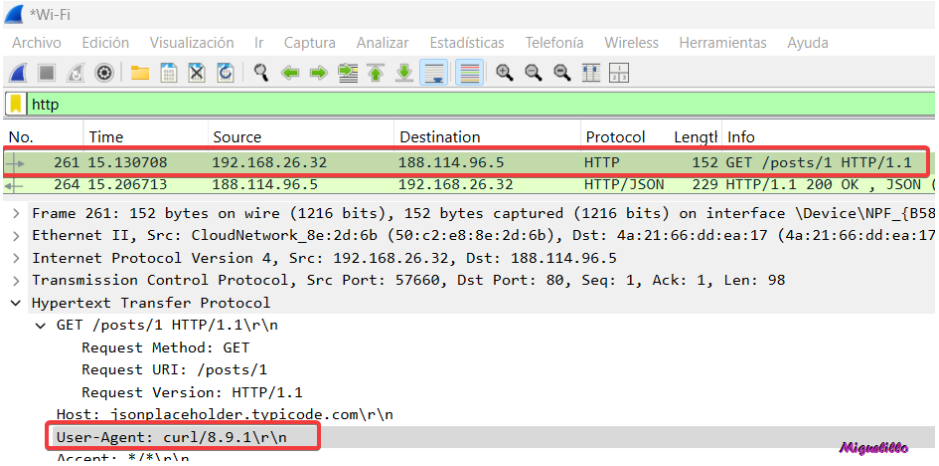
El código de estado en la respuesta de una petición GET exitosa es 200 OK. Este código indica que la solicitud se procesó correctamente y el servidor devolvió la información solicitada.




# ¿Qué información puedes deducir del campo User-Agent?

El campo User-Agent en una cabecera HTTP proporciona detalles sobre el cliente que realiza la solicitud, incluyendo el navegador, sistema operativo y, a veces, la versión del software que realiza la petición. Por ejemplo, al hacer una solicitud desde cURL o Postman, el User-Agent podría indicar algo como curl/7.68.0 o PostmanRuntime/7.29.0.



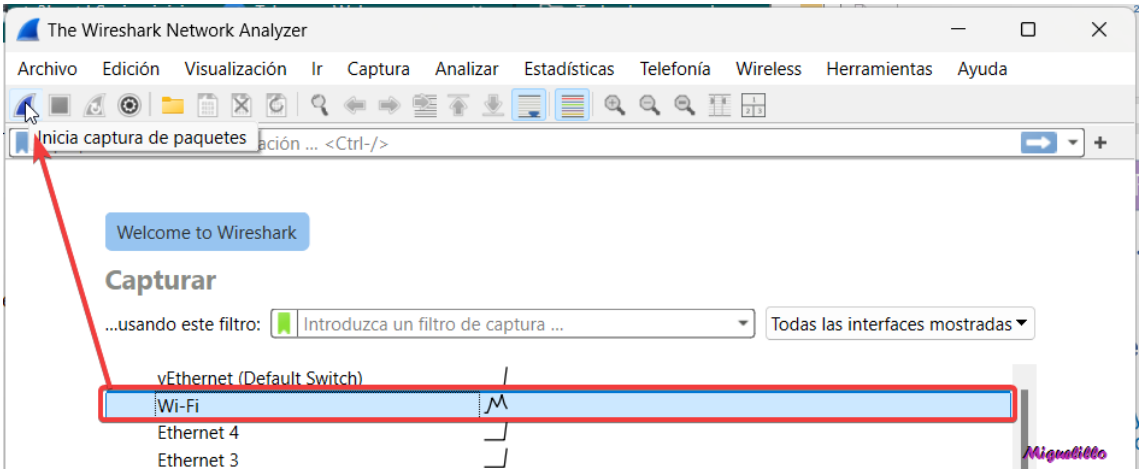



# Ejercicio 2 Inspección de una Petición POST

## a) Comienza una nueva captura en Wireshark.

Solución:

Abre Wireshark y selecciona la **interfaz de red** que **usas** para conectarte a **Internet** y haz clic en el botón **Start**



## b) Realiza una petición POST.

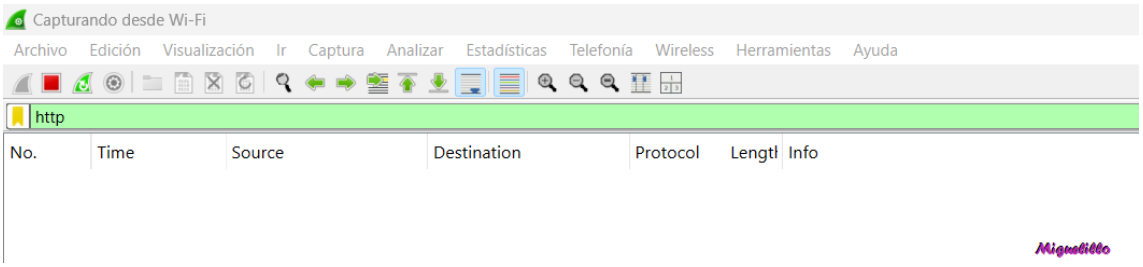
Solución:

Realizamos la petición **POST**

```
curl -X POST "https://jsonplaceholder.typicode.com/posts" -H
"Content-Type: application/json" -d "{\"title\": \"Nuevo Post\",
\"body\": \"Contenido del post\", \"userId\": 1}"
```

## c) Filtra y analiza el tráfico

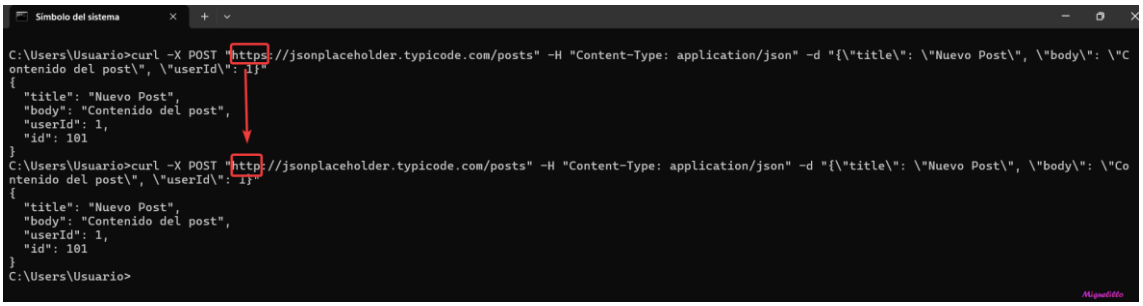
### Solución:



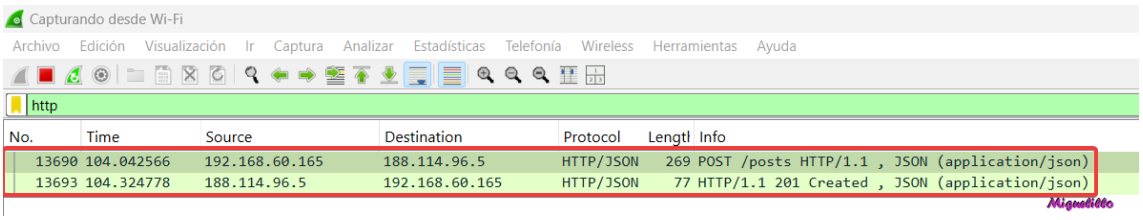

## Incidencia

A la hora de filtrar por paquetes como hemos hecho las peticiones mediante https si filtramos por https no aparecen ya que van encriptados para solucionarlo simplemente haremos el get a http

## Resolución

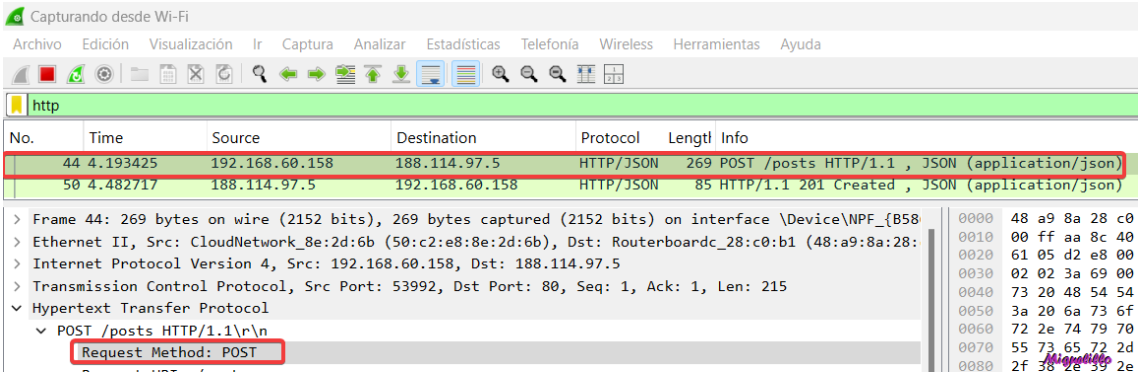



```
curl -X POST "http://jsonplaceholder.typicode.com/posts" -H "Content-Type: application/json" -d "{\"title\": \"Nuevo Post\", \"body\": \"Contenido del post\", \"userId\": 1}"
```

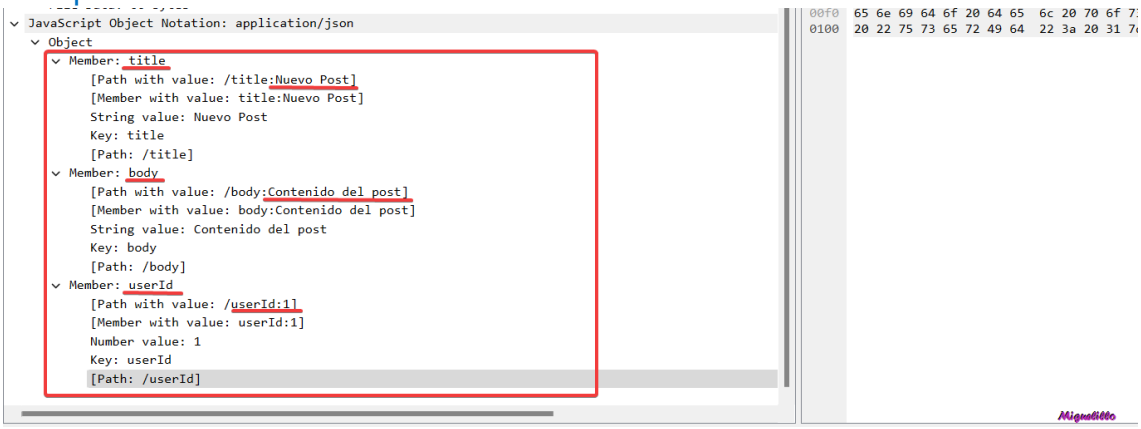


# Análisis del paquete

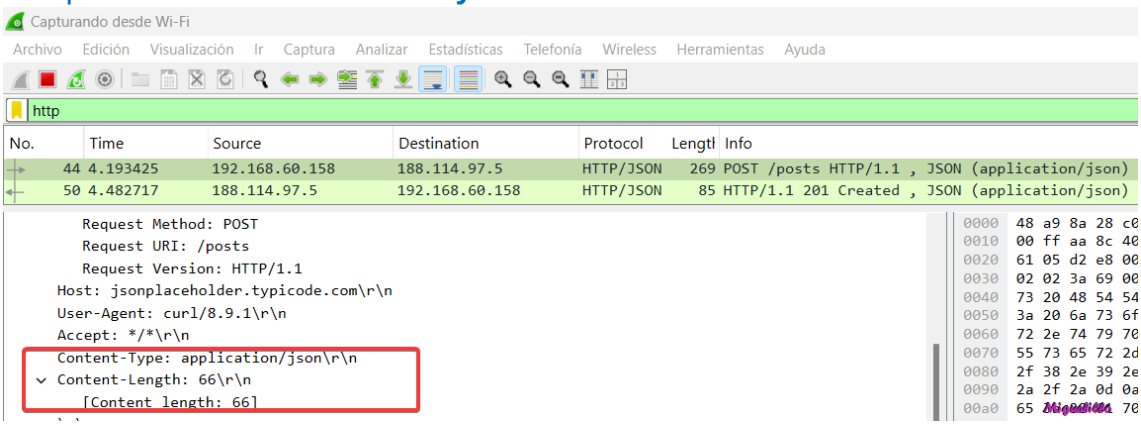
- Método HTTP: Confirma el uso de POST.




- Cuerpo de la solicitud: Visualiza el contenido JSON enviado.




- Cabeceras de solicitud:
  – **Content-Type:** Este campo, que tiene el valor **application/json**, indica que el tipo de datos enviados en el cuerpo de la solicitud es **JSON**.
  – **Content-Length**: Este campo muestra el valor **66**, lo que indica que el cuerpo de la solicitud tiene **66 bytes**.



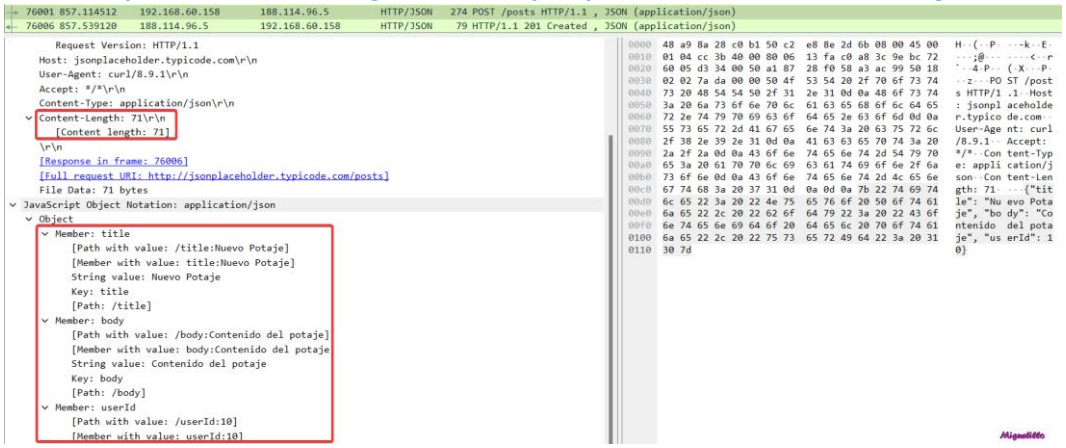
# d) Preguntas

Solución:

## ¿Que observas en el Content-Length cuando cambias el tamaño del cuerpo?

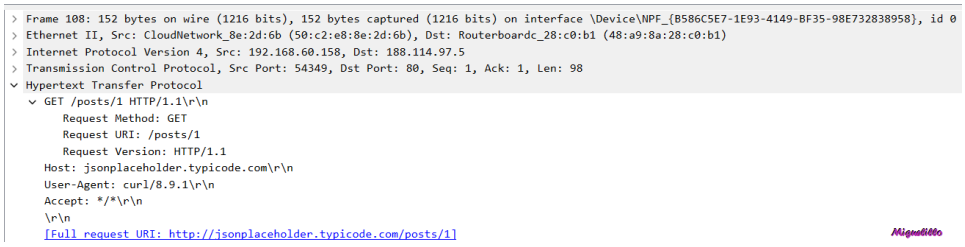
Probamos a cambiar el **POST**


Vemos que el content length cambia ya que el contenido es más grande

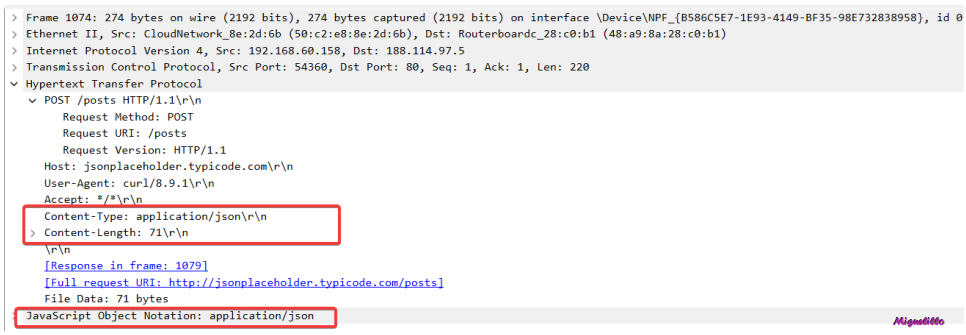



## ¿Qué otros campos se agregan al realizar una petición POST en comparación con GET?

GET




POST



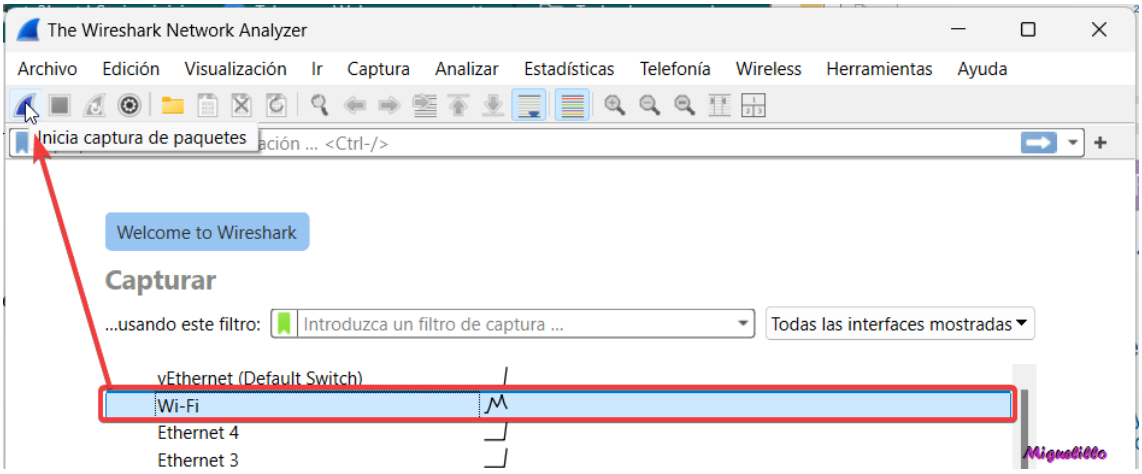

En **POST** podemos ver que se añaden el **Content-Length** y el **Content-Type** y el **JavaScript Objet**

# Ejercicio 3 Análisis de Errores con Wireshark

## a) Iniciar Captura en Wireshark

Solución:

Abre Wireshark y selecciona la **interfaz de red** que **usas** para conectarte a **Internet** y haz clic en el botón **Start**



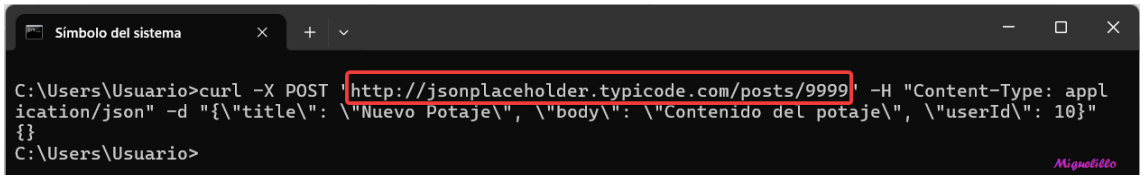
## b) Realizar una Petición a un Recurso Inexistente

Solución:

Realicé una petición a un recurso que no existe, específicamente a la URL **https://jsonplaceholder.typicode.com/posts/9999** Esto generó un error intencional para poder analizar la respuesta.

Hacemos un **GET**

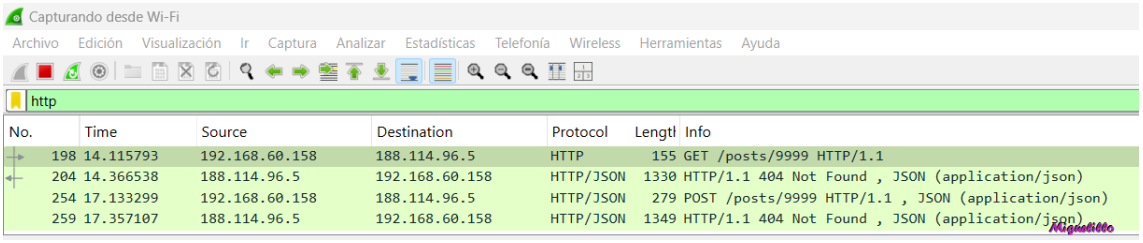

Y también un **POST**



## c) Filtra el tráfico HTTP y analiza la respuesta.

Solución:

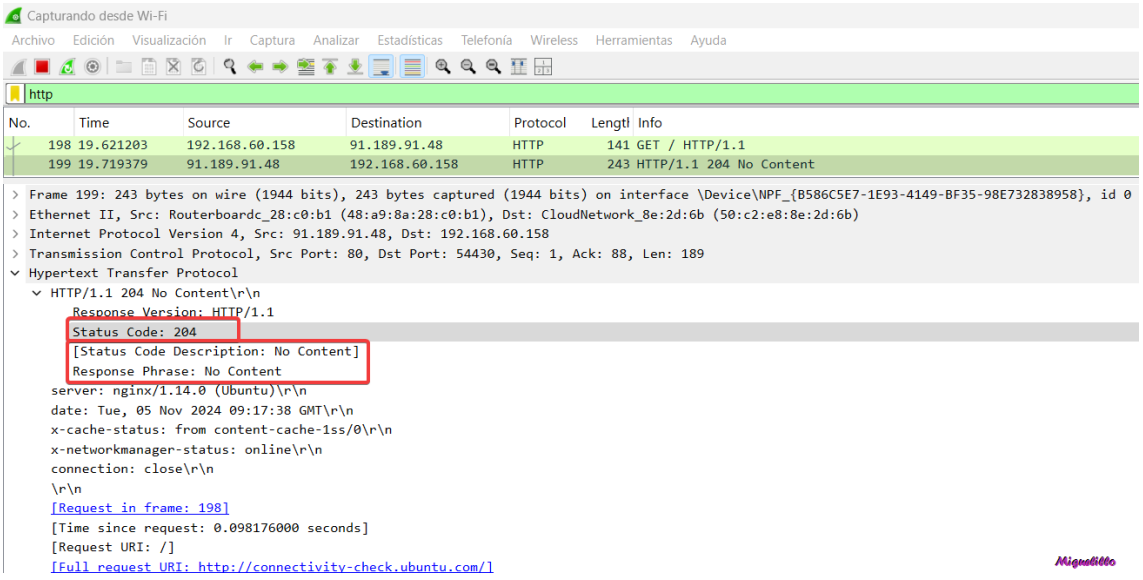
Una vez que se generó el error, volví a Wireshark y apliqué un filtro para visualizar solo el tráfico HTTP. Para ello, escribí http en la barra de filtro



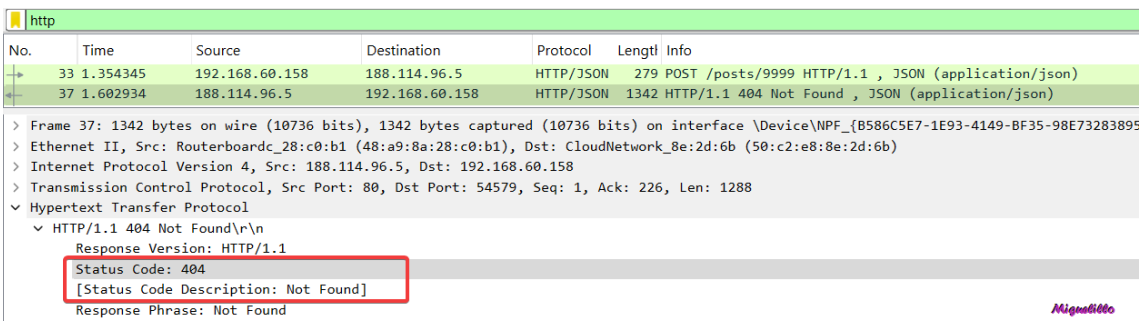

# Análisis del Error

Seleccioné el paquete correspondiente a la respuesta del servidor y revisé los detalles del paquete. Observé los siguientes campos importantes en la respuesta de error

El código de error es el **202 No Content**




Si hacemos un **POST** aparece 404 **Not Found**



## b) Preguntas

Solución:

### ¿Qué diferencias observas en los paquetes que contienen errores frente a los paquetes de éxito?

El código de estado y la respuesta

```
3290 200.099819    188.114.97.5      192.168.60.158    HTTP/JSON   198 HTTP/1.1 200 OK , JSON (application/json)
3410 210.690122    192.168.60.158    188.114.97.5      HTTP/JSON   274 POST /posts HTTP/1.1 , JSON (application/json)
3413 210.928077    188.114.97.5      192.168.60.158    HTTP/JSON    82 HTTP/1.1 201 Created , JSON (application/json)
```

Los paquetes de error: Pueden incluir cabeceras que indican el error o cómo manejarlo. Por ejemplo, cabeceras que indican la necesidad de una autenticación o la presencia de un error temporal.

```
    Response Phrase: Not Found
Date: Tue, 05 Nov 2024 09:42:55 GMT\r\n
Content-Type: application/json; charset=utf-8\r\n
Content-Length: 2\r\n
Connection: keep-alive\r\n
Report-To: {"group":"heroku-nel","max_age":3600,"endpoints":[{"url":"https://nel.heroku.com/reports?ts=1730799775&sid=e11707d5-02a7-43ef-b45e-2cf4d2036f7d&s=P
Reporting-Endpoints: heroku-nel=https://nel.heroku.com/reports?ts=1730799775&sid=e11707d5-02a7-43ef-b45e-2cf4d2036f7d&s=PxHlzGh%2FwMloH14bHzYoebw6ZDTOVEcn%2BP
Nel: {"report_to":"heroku-nel","max_age":3600,"success_fraction":0.005,"failure_fraction":0.05,"response_headers":["Via"]}\r\n
X-Powered-By: Express\r\n
X-Ratelimit-Limit: 1000\r\n
X-Ratelimit-Remaining: 999\r\n
X-Ratelimit-Reset: 1730799817\r\n
Vary: Origin, Accept-Encoding\r\n
Access-Control-Allow-Credentials: true\r\n
Cache-Control: max-age=43200\r\n
Pragma: no-cache\r\n
Expires: -1\r\n
X-Content-Type-Options: nosniff\r\n
Etag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"\r\n
Via: 1.1 vegur\r\n
CF-Cache-Status: EXPIRED\r\n
Server: cloudflare\r\n
CF-RAY: 8ddbe585c82acfb1-MAD\r\n
alt-svc: h3=":443"; ma=86400\r\n
server-timing: cfL4;desc="?proto=TCP&rtt=9752&sent=1&recv=3&lost=0&retrans=0&sent_bytes=0&recv_bytes=101&delivery_rate=0&cwnd=244&unsent_bytes=0&cid=0000000000
\r\n
[Request in frame: 198]
```

Los paquetes de éxito: Las cabeceras suelen incluir información sobre el tipo de contenido, la longitud del contenido y las políticas de caché.

```
> Frame 3410: 274 bytes on wire (2192 bits), 274 bytes captured (2192 bits) on interface \Device\NPF_{B586C5E7-1E93-4149-BF35-98E732838958}, id 0
> Ethernet II, Src: CloudNetwork_8e:2d:6b (50:c2:e8:8e:2d:6b), Dst: Routerboardc_28:c0:b1 (48:a9:8a:28:c0:b1)
> Internet Protocol Version 4, Src: 192.168.60.158, Dst: 188.114.97.5
> Transmission Control Protocol, Src Port: 54671, Dst Port: 80, Seq: 1, Ack: 1, Len: 220
v Hypertext Transfer Protocol
  v POST /posts HTTP/1.1\r\n
      Request Method: POST
      Request URI: /posts
      Request Version: HTTP/1.1
    Host: jsonplaceholder.typicode.com\r\n
    User-Agent: curl/8.9.1\r\n
    Accept: */*\r\n
    Content-Type: application/json\r\n
  > Content-Length: 71\r\n
    \r\n
    [Response in frame: 3413]
    [Full request URI: http://jsonplaceholder.typicode.com/posts]
    File Data: 71 bytes
> JavaScript Object Notation: application/json
```

## ¿Qué indican las cabeceras Retry-After o Warning, si están presentes?

Cabecera **Retry-After:**

Significado: Esta cabecera se utiliza principalmente en respuestas de error (como 503) para indicar al cliente cuánto **tiempo** debe **esperar** antes de volver a **intentar** la **solicitud**.

Ejemplo: Si la respuesta incluye Retry-After: 120, significa que el cliente debe esperar 120 segundos antes de realizar otra solicitud.

Cabecera **Warning**:

Significado: Proporciona **información** adicional sobre el **estado** de la **respuesta**, como **advertencias** sobre posibles **problemas** que no justifican un error, pero que el cliente debería tener en cuenta.

Ejemplo: Puede incluir **advertencias** sobre la **validez** de los datos, problemas **temporales** en el servidor, o que el **contenido** puede estar **desactualizado**.