

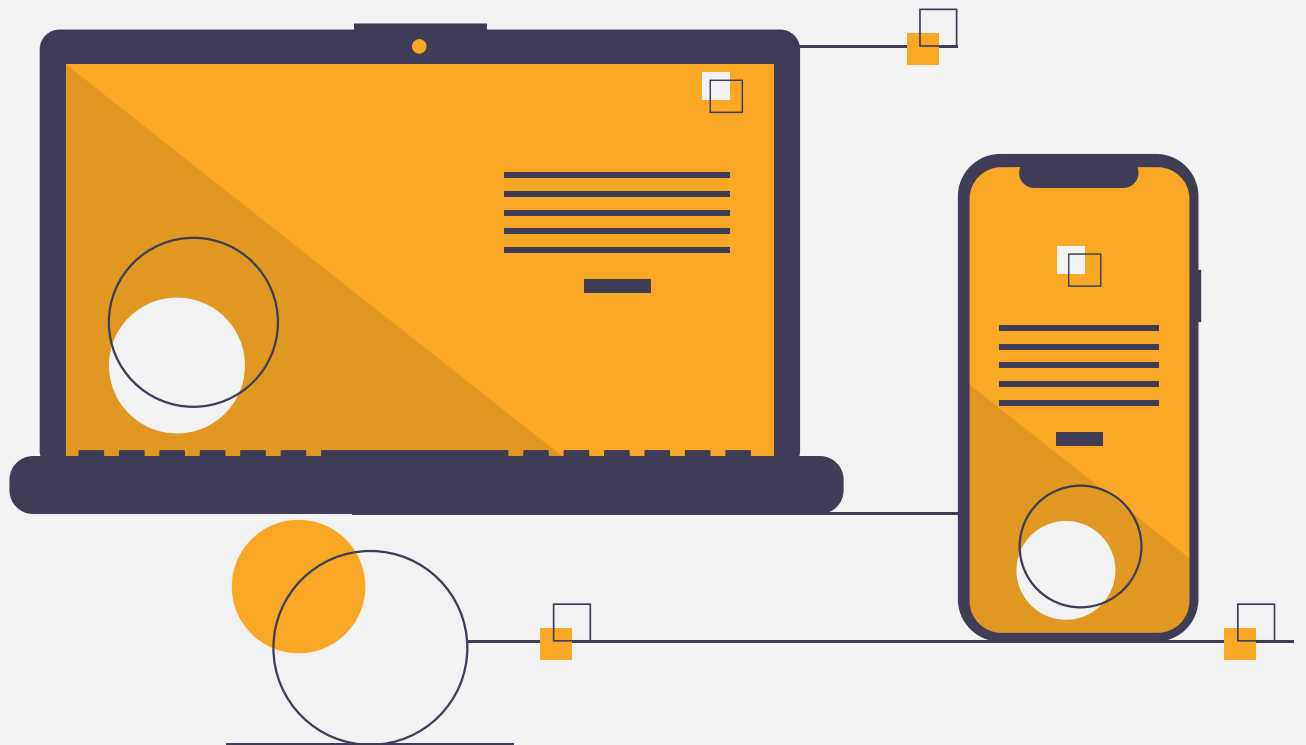
HTTP y Servidores WEB

SRI Tema 4

Ejercicio 1 y 2

Miguel Lillo Romero

2º ASIR



Índice:

Introducción.....	3
Ejercicio 1 Peticiones GET Básicas.....	6
a) Obtener todos los posts.....	6
b) Obtener un post específico (por ejemplo, ID 5).....	8
d) Preguntas.....	9
Ejercicio 2 Uso de Parámetros de Consulta en GET.....	10
a) Filtrar posts por usuario (userId=1).....	10
b) Preguntas.....	11
Ejercicio 3 Realizar una Petición POST y Validar la Respuesta.....	12
a) Crear un nuevo post.....	12
b) Preguntas.....	14
Ejercicio 4 Actualizar un Post con PUT y PATCH.....	15
a) Actualizar un post usando PUT.....	15
b) Actualizar parcialmente un post usando PATCH.....	16
c) Preguntas.....	17
Ejercicio 5: Manejo de Errores HTTP.....	18
a) Intenta acceder a un recurso inexistente:.....	18
b) Intenta crear un recurso sin datos:.....	19
c) Preguntas.....	20
Introducción.....	21
Ejercicio 1 Realizar una Petición GET y Capturar el Tráfico en Wireshark.....	22
a) Inicia Wireshark.....	22
b) Realiza una petición GET.....	23
c) Detén la captura en Wireshark:.....	24
d) Preguntas.....	27
Ejercicio 2 Inspección de una Petición POST.....	28
a) Comienza una nueva captura en Wireshark.....	28
b) Realiza una petición POST.....	29
c) Filtra y analiza el tráfico.....	30
d) Preguntas.....	32

Ejercicio 3 Análisis de Errores con Wireshark	33
a) Iniciar Captura en Wireshark.....	33
b) Realizar una Petición a un Recurso Inexistente	34
c) Filtra el tráfico HTTP y analiza la respuesta.....	35
b) Preguntas	36

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?

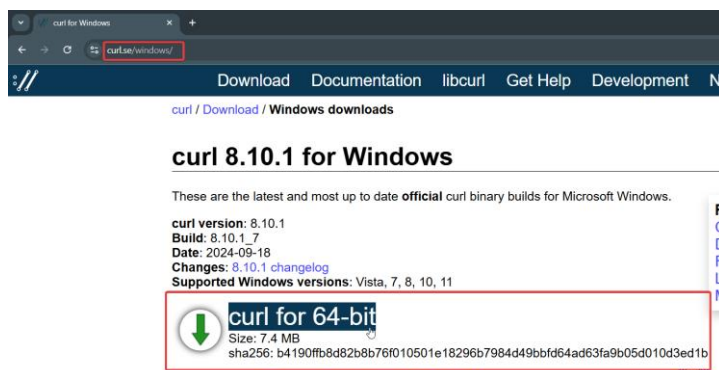
cURL es una herramienta de **línea de comandos** que permite realizar **peticiones HTTP** de forma rápida. Aprender a utilizar cURL es útil porque permite trabajar con **APIs** desde la terminal, lo cual es valioso en entornos de desarrollo y automatización.

Escenario

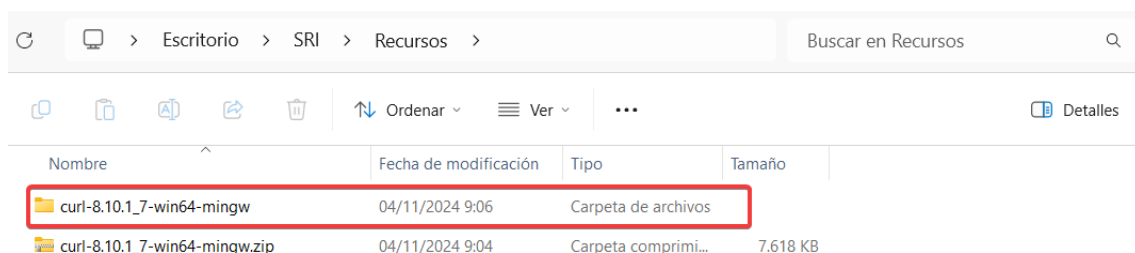
Voy a utilizar mi propio equipo un **Windows 11** de **64bits** con 1 tarjeta de red en modo **DHCP**

Instalación

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



Extraemos el archivo comprimido



Miguelillo

Descargamos el instalador de Postman

The screenshot shows the Postman website's download page. The browser's address bar highlights the URL `postman.com/downloads/`. The page features a navigation bar with links for Product, Pricing, Enterprise, Resources and Support, and Public API Network, along with buttons for Contact Sales, Sign In, and Sign Up for Free. The main heading is "Download Postman", followed by a subheading: "Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman." Below this, a section titled "The Postman app" includes a description and a prominent orange button labeled "Windows 64-bit". To the right, a preview of the Postman application interface is shown, displaying a workspace with a collection named "Notion API" and a request to "Retrieve a database" from "Notion API / Databases".

Ejecutamos el .exe



Join Postman

Work email

[Create Free Account](#)

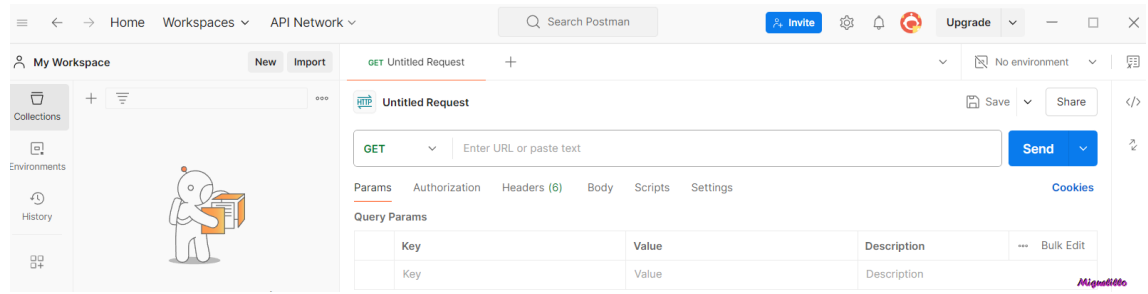
Already have an account? [Log In](#)

[Continue without an account](#)

Nos creamos una cuenta gratuita

The screenshot displays the "Create Postman account" page. On the left, a section titled "Why sign up?" lists four benefits: organizing API development within Postman Workspaces, syncing data across devices, backing up data to the Postman cloud, and the fact that it's free. The main form area contains fields for Work email, Username, and Password, with a "Show" link for the password. Below these fields are checkboxes for "Receive product updates, news, and other marketing communications" (unchecked) and "Stay signed in" (checked). A green checkmark and the text "Success!" are displayed below the form. At the bottom, there is a prominent orange button labeled "Create Free Account".

Ya estaría instalado



Ejercicio 1 Peticiones GET Básicas

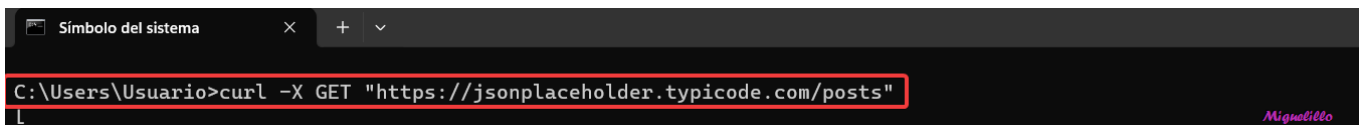
a) Obtener todos los posts

Solución:

Ejecutamos el comando

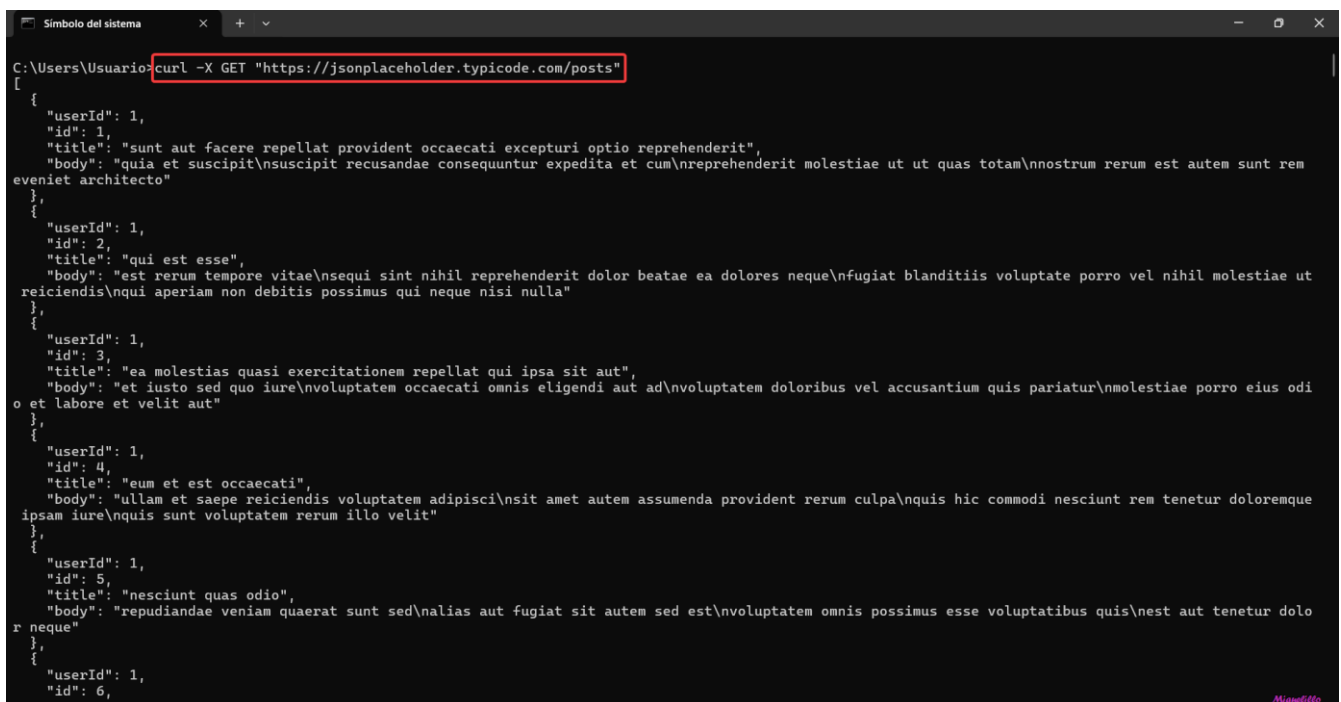
curl -X GET <https://jsonplaceholder.typicode.com/posts>

para obtener una lista completa de posts.



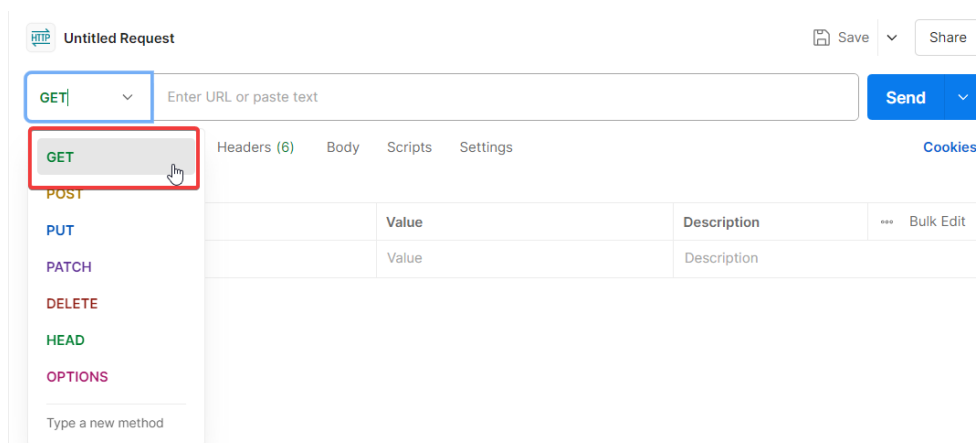
```
C:\Users\Usuario>curl -X GET "https://jsonplaceholder.typicode.com/posts"
```

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.



```
C:\Users\Usuario>curl -X GET "https://jsonplaceholder.typicode.com/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"
  },
  {
    "userId": 1,
    "id": 3,
    "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
    "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut"
  },
  {
    "userId": 1,
    "id": 4,
    "title": "eum et est occaecati",
    "body": "ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit"
  },
  {
    "userId": 1,
    "id": 5,
    "title": "nesciunt quas odio",
    "body": "repudiandae veniam quaerat sunt sed\nalias aut fugiat sit autem sed est\nvoluptatem omnis possimus esse voluptatibus quis\nest aut tenetur dolo"
  },
  {
    "userId": 1,
    "id": 6,
    "title": "facere et omnis",
    "body": "facere et omnis"
  }
]
```

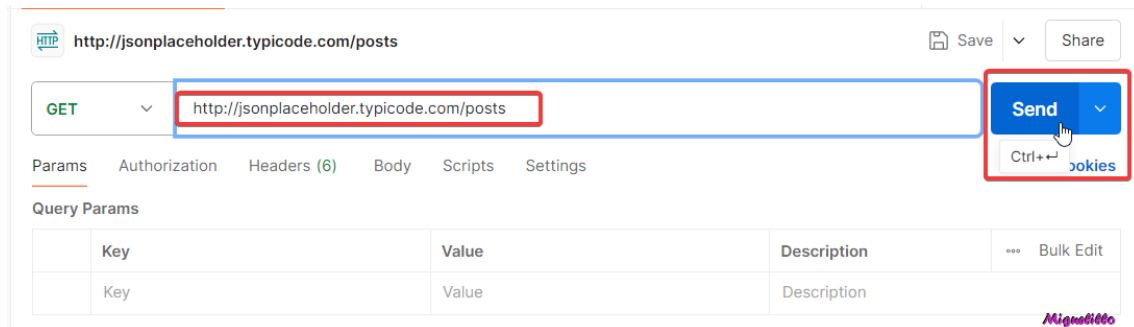
Vamos a postman, Selecciona el método **GET** en el menú desplegable.



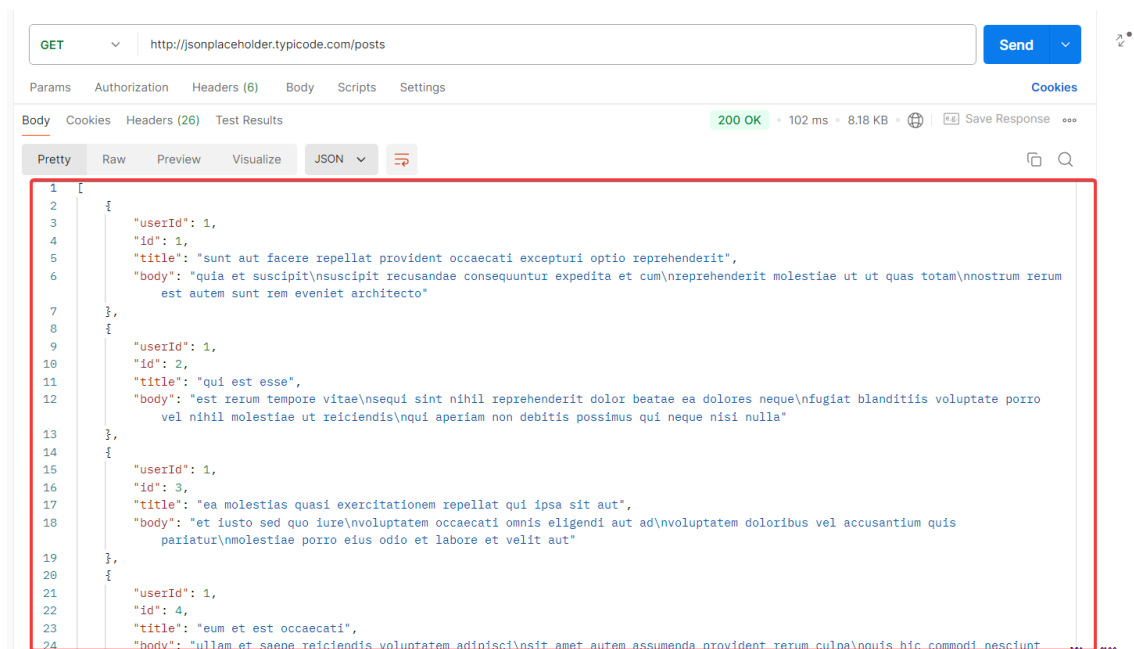
Miguelillo

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.



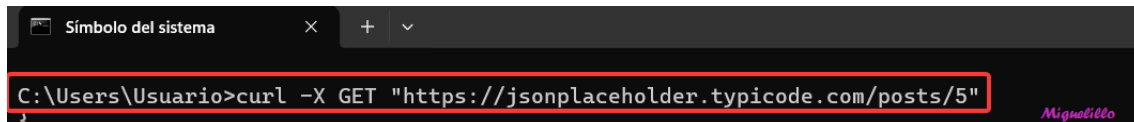
b) Obtener un post específico (por ejemplo, ID 5)

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**



```
Símbolo del sistema
C:\Users\Usuario>curl -X GET "https://jsonplaceholder.typicode.com/posts/5"
```

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



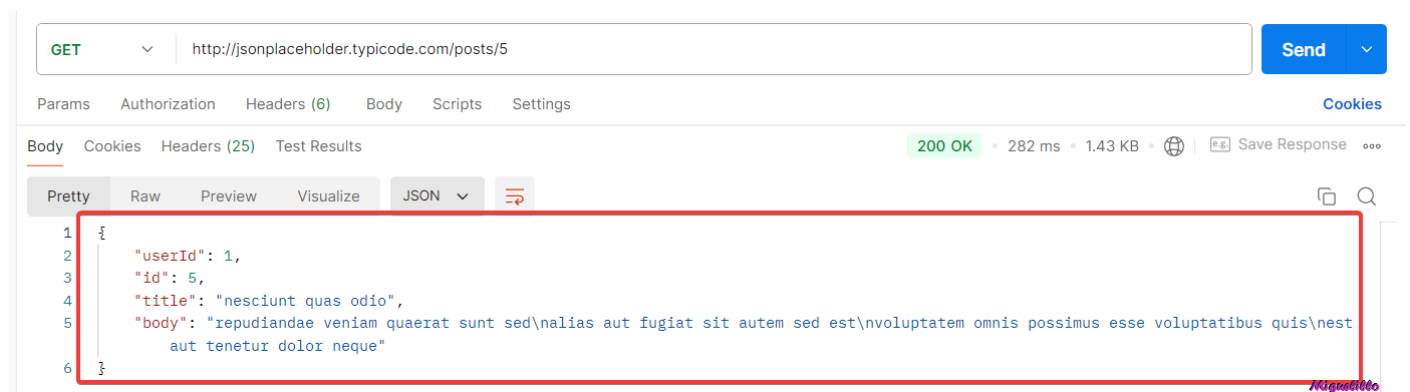
```
C:\Users\Usuario>curl -X GET "https://jsonplaceholder.typicode.com/posts/5"
{"userId": 1,
  "id": 5,
  "title": "nesciunt quas odio",
  "body": "repudiandae veniam quaerat sunt sed\nalias aut fugiat sit autem sed est\nvoluptatem omnis possimus esse voluptatibus quis\nest aut tenetur dolor neque"}
```

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.

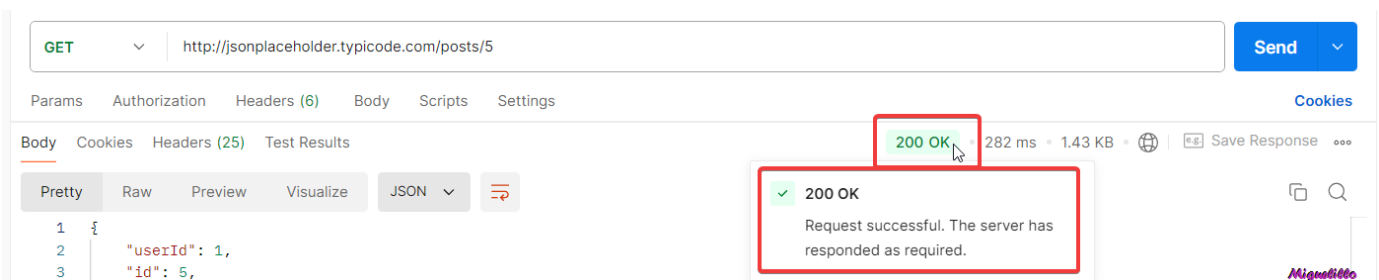


d) Preguntas

Solución:

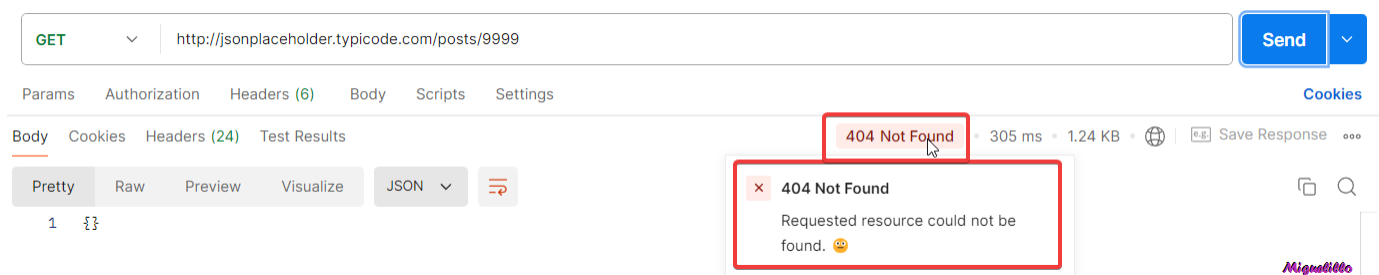
¿Qué código de estado devuelve la API cuando accedes a un post específico?

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

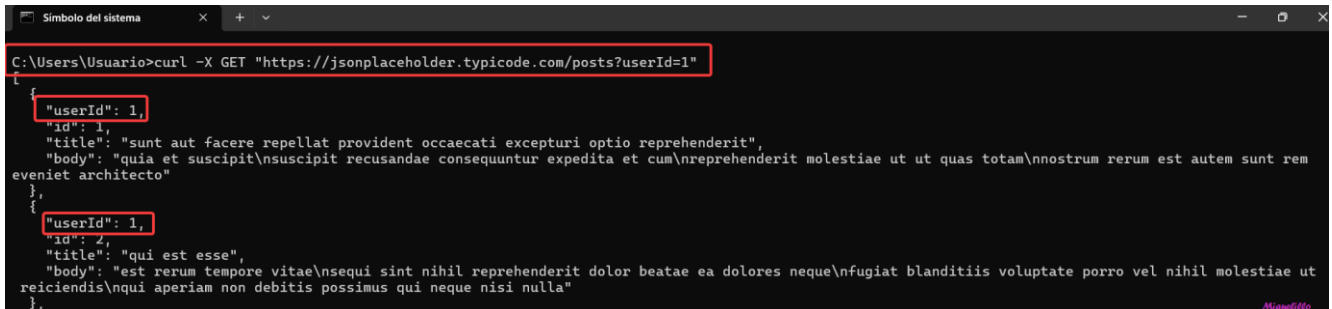
a) Filtrar posts por usuario (userId=1)

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

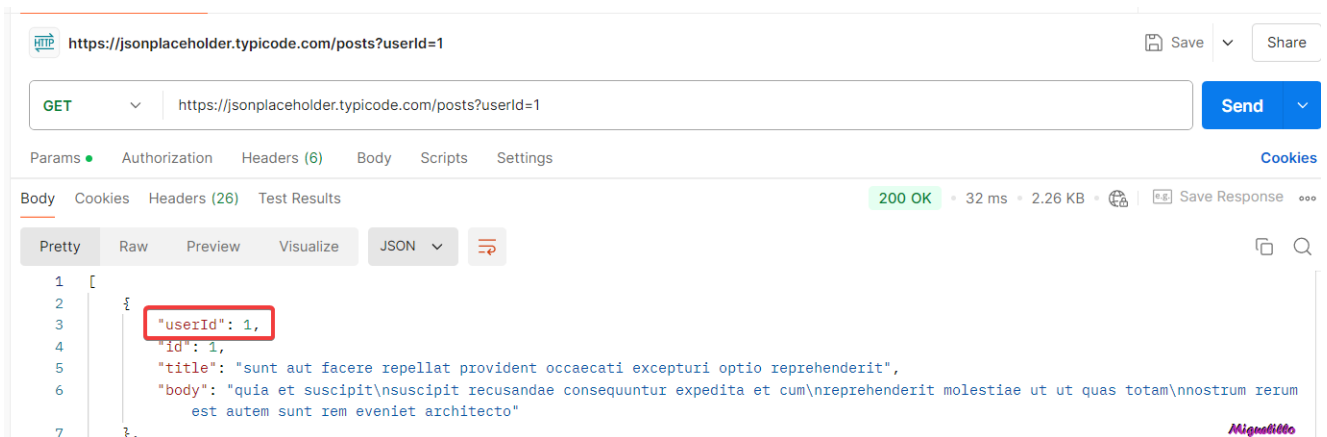


```
Símbolo del sistema
C:\Users\Usuario>curl -X GET "https://jsonplaceholder.typicode.com/posts?userId=1"
[{"userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit 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"}]
```

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`



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

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

Params Authorization Headers (6) Body Scripts Settings

Body Cookies Headers (26) Test Results 200 OK • 32 ms • 2.26 KB • Save Response

Pretty Raw Preview Visualize JSON

```
[
  {
    "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"
  },
  {
    "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"
  }
]
```

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**.

GET https://jsonplaceholder.typicode.com/posts?userId=1&id=5

Params • Authorization Headers (6) Body Scripts Settings

Body Cookies Headers (25) Test Results 200 OK • 438 ms • 1.44 KB • Save Res

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "userId": 1,
4     "id": 5,
5     "title": "nesciunt quas odio",
6     "body": "repudiandae veniam quaerat sunt sed\nalias aut fugiat sit autem sed est\nvoluptatem omnis possimus esse voluptatibus\nquis\nnest aut tenetur dolor neque"
7   }
8 ]
```

Miguelillo

Ejercicio 3 Realizar una Petición POST y Validar la Respuesta

a) Crear un nuevo post

Solución:

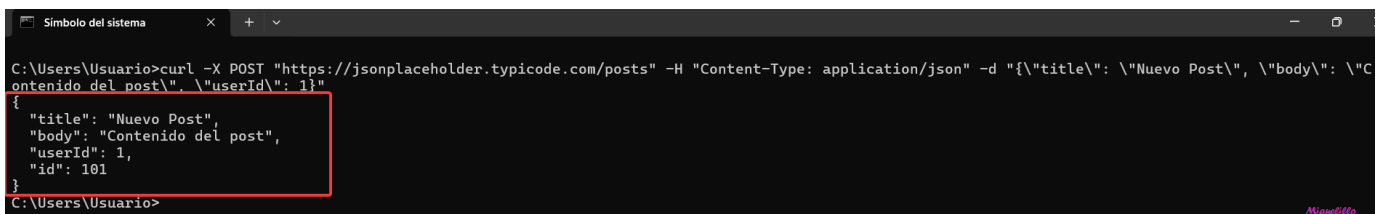
Abre la terminal y ejecuta el siguiente comando

```
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.

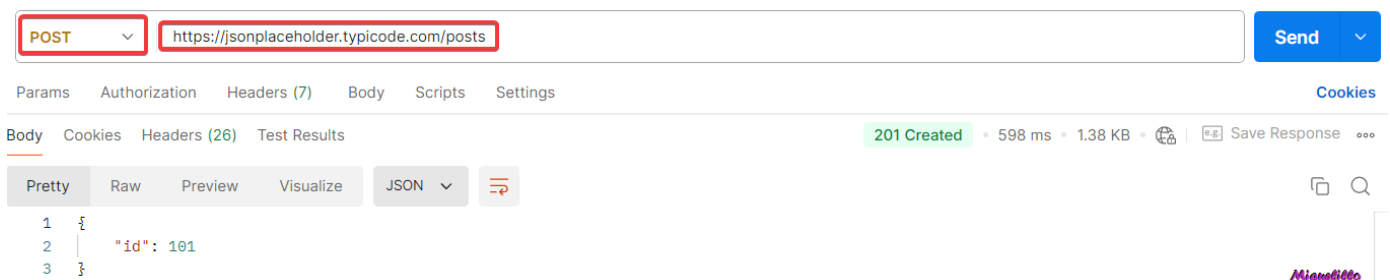
-d permite incluir el cuerpo de la solicitud con el JSON del nuevo post.



```
Simbolo del sistema
C:\Users\Usuario>curl -X POST "https://jsonplaceholder.typicode.com/posts" -H "Content-Type: application/json" -d '{"title": "Nuevo Post", "body": "Contenido del post", "userId": 1}'
{"title": "Nuevo Post",
"body": "Contenido del post",
"userId": 1,
"id": 101}
C:\Users\Usuario>
```

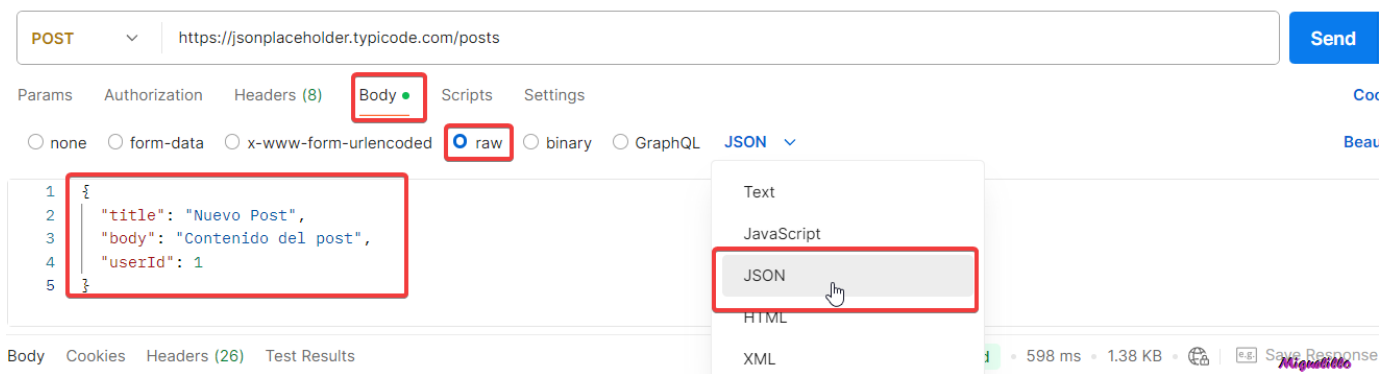
Abre Postman y selecciona el método **POST** en la barra de URL, ingresa la siguiente dirección

<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  
}
```

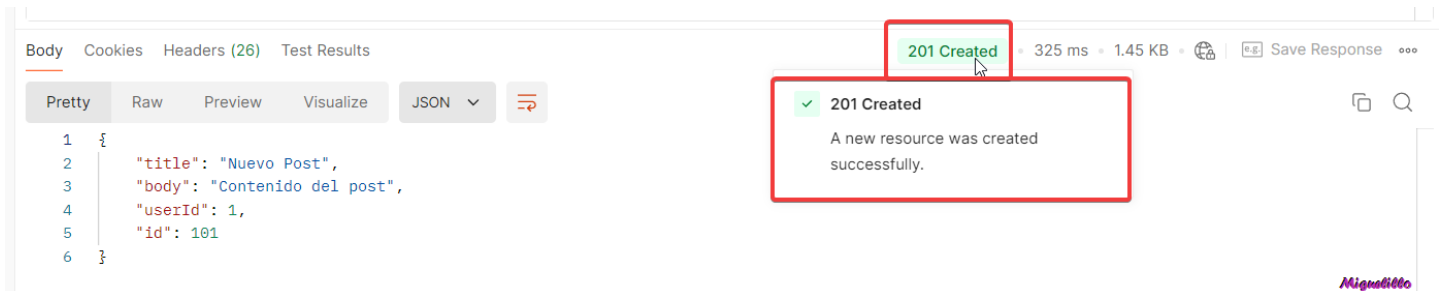


b) Preguntas

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**.

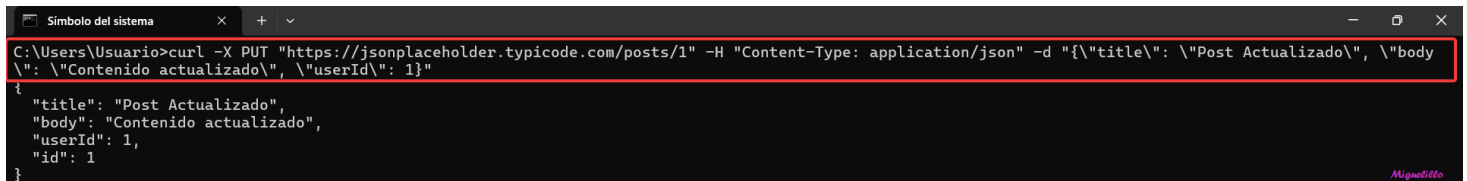
Ejercicio 4 Actualizar un Post con PUT y PATCH

a) Actualizar un post usando PUT

Solución:

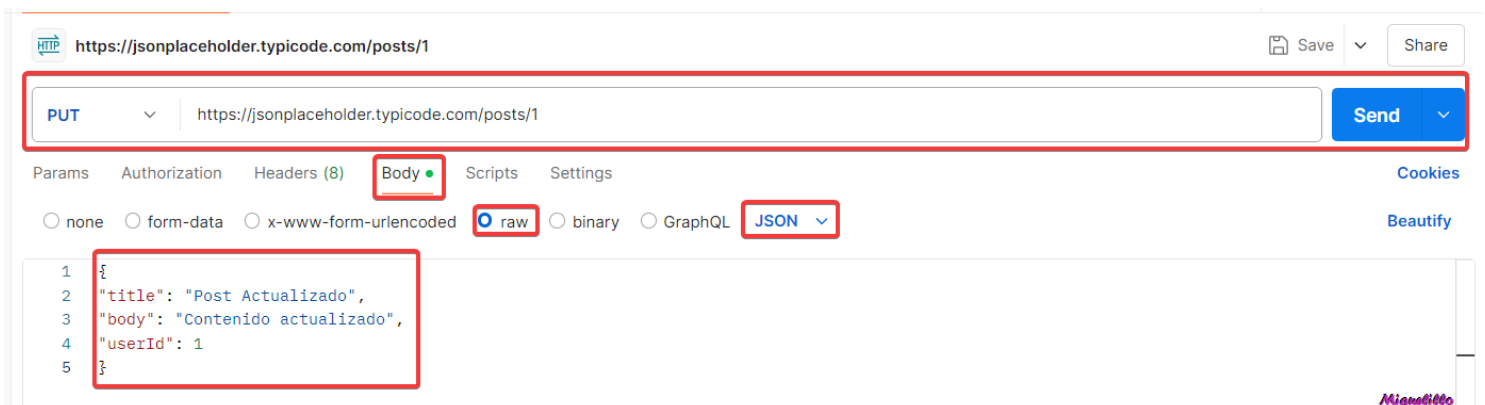
Ejecuta en la consola `curl -X PUT`

```
"https://jsonplaceholder.typicode.com/posts/1" -H "Content-Type: application/json" -d "{\"title\": \"Post Actualizado\", \"body\": \"Contenido actualizado\", \"userId\": 1}"
```



```
Símbolo del sistema
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 **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

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

- Method:** PATCH (selected in a dropdown).
- URL:** <https://jsonplaceholder.typicode.com/posts/1>
- Buttons:** Send (blue button), Cookies, Beautify.
- Tabs:** Params, Authorization, Headers (8), Body (selected), Scripts, Settings.
- Body Type:** raw (selected in a dropdown), JSON (also shown in a dropdown).
- Request Body (raw):**

```
1 {
2   "title": "Título modificado"
3 }
```
- Response:** 200 OK, 166 ms, 1.45 KB. Status bar shows "Save Response".
- Response Body (Pretty):**

```
1 {
2   "userId": 1,
3   "id": 1,
4   "title": "Título modificado",
5   "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
6 }
```

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.

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

- URL:** `https://jsonplaceholder.typicode.com/posts/1`
- Method:** PATCH
- Request Body (JSON):**

```
1 {
2   "title": "Título modificado"
3 }
```
- Response Status:** 200 OK (highlighted with a red box)
- Response Message:** Request successful. The server has responded as required.
- Response Body (JSON):**

```
1 {
2   "userId": 1,
3   "id": 1,
4   "title": "Título modificado",
5   "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
6 }
```

Ejercicio 5: Manejo de Errores HTTP

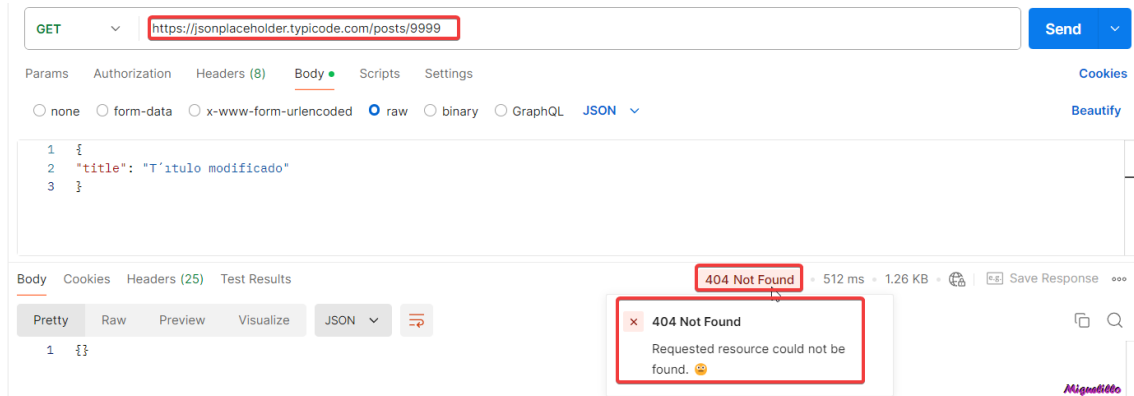
a) Intenta acceder a un recurso inexistente:

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



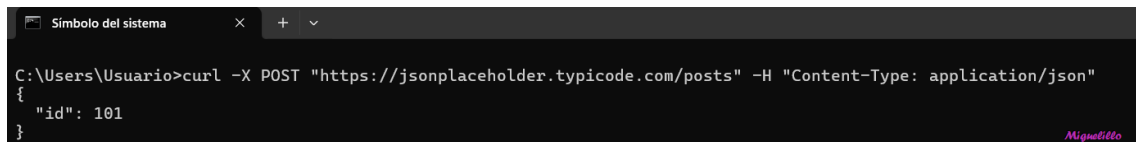
b) Intenta crear un recurso sin datos:

Solución:

Abrimos la consola

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

vemos que no nos arroja un error



The screenshot shows a Windows command prompt window titled 'Símbolo del sistema'. The command entered is `curl -X POST "https://jsonplaceholder.typicode.com/posts" -H "Content-Type: application/json"`. The output is a JSON object: `{ "id": 101 }`. The window has a dark background and a pink 'Miguelillo' watermark in the bottom right corner.

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

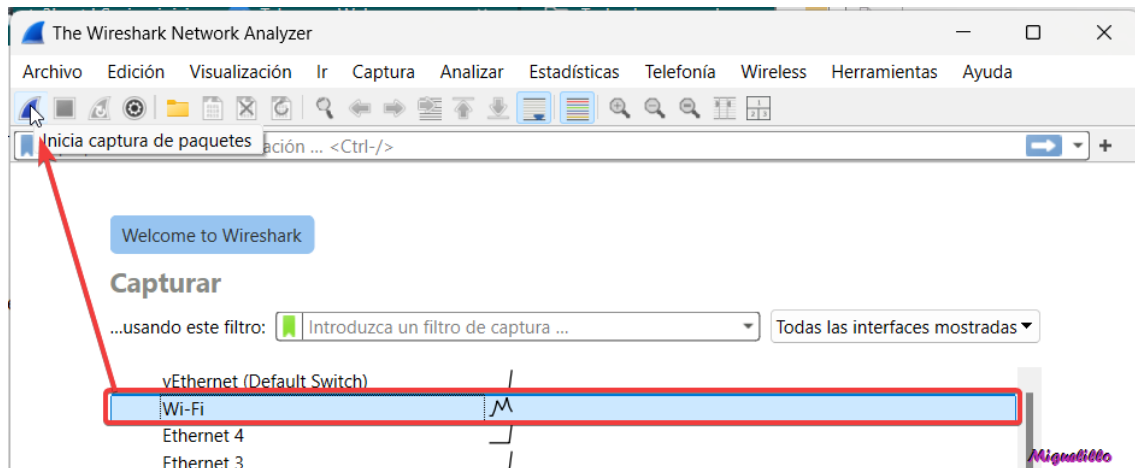
Inspeccionar el **tráfico** generado por estas **peticiones** usando **Wireshark**. A través de esta actividad, se busca **familiarizarse** con el **análisis** de las **cabeceras** y el cuerpo de las **solicitudes** y **respuestas** HTTP, así como identificar y **comprender** los **errores HTTP** que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

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

a) Inicia Wireshark

Solución:

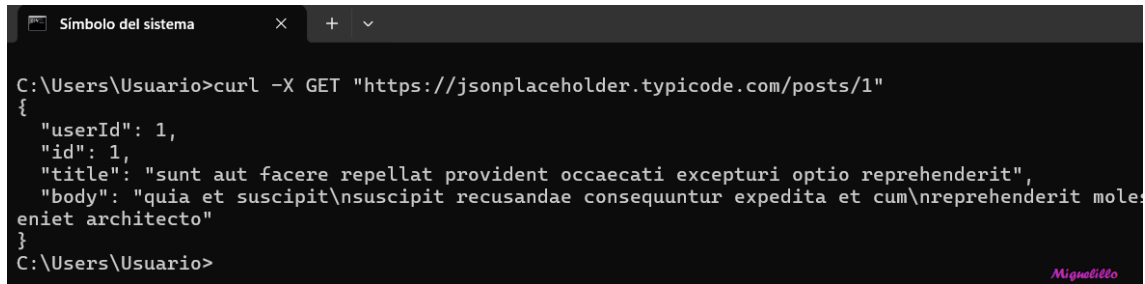
Abre Wireshark y selecciona la **interfaz de red** que **usas** para conectarte a **Internet** y haz clic en el botón **Start**



b) Realiza una petición GET

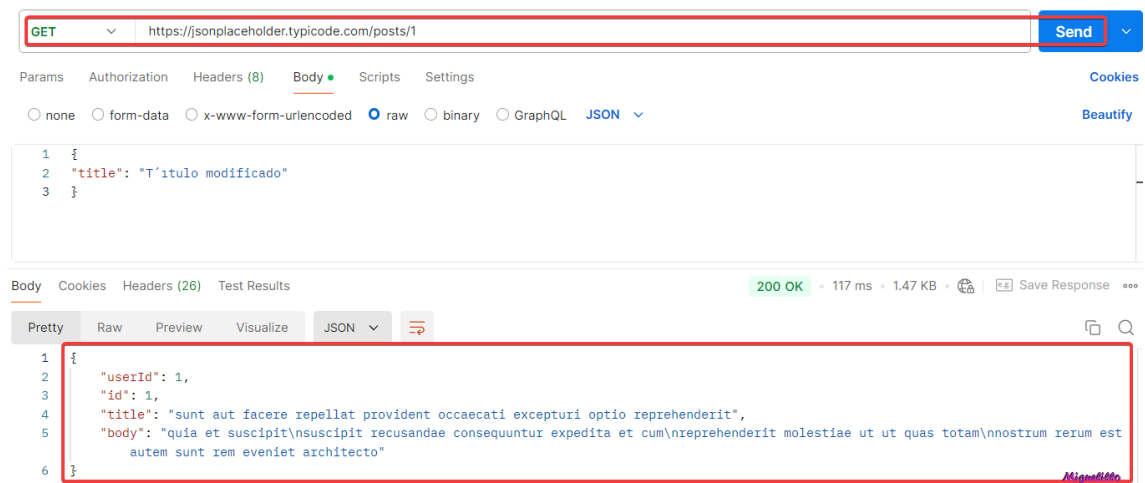
Solución:

Creamos una petición get con cURL



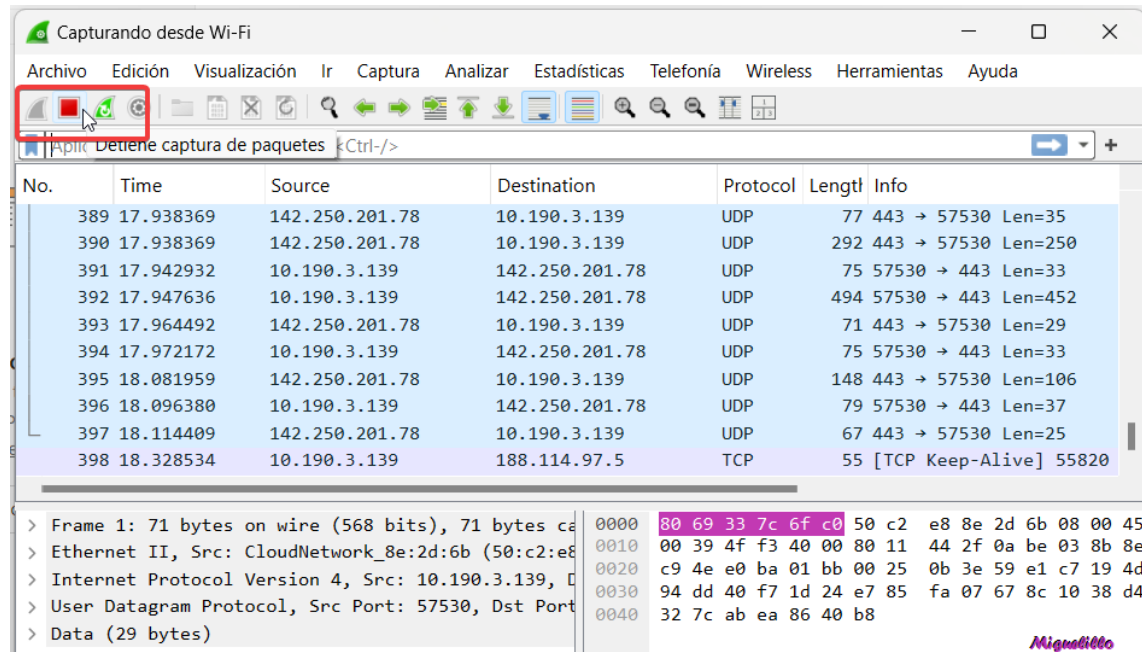
```
C:\Users\Usuario>curl -X GET "https://jsonplaceholder.typicode.com/posts/1"
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
}
```

También se puede realizar en **Postman**

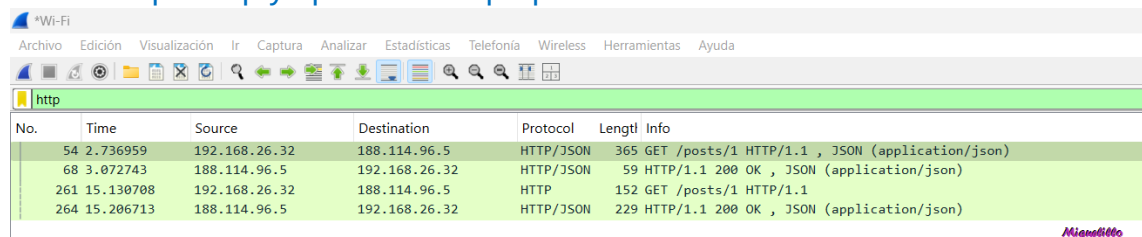


c) Detén la captura en Wireshark:

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

Wireshark packet capture showing an HTTP GET request. The packet list shows frame 54 (2.736959s) from 192.168.26.32 to 188.114.96.5, and frame 68 (3.072743s) as the response. The packet details for frame 54 show the GET request with a full request URI highlighted in red: `http://jsonplaceholder.typicode.com/posts/1`.

■ 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**

Wireshark packet details for the HTTP GET request, showing the client headers. The `User-Agent: PostmanRuntime/7.42.0` and `Accept-Encoding: gzip, deflate, br` fields are highlighted with red boxes.

Respuesta del servidor:

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

```

54 2.736959 192.168.26.32 188.114.96.5 HTTP/JSON 365 GET /posts/1 HTTP/1.1 , JSON (application/json)
68 3.072743 188.114.96.5 192.168.26.32 HTTP/JSON 59 HTTP/1.1 200 OK , JSON (application/json)

> Frame 68: 59 bytes on wire (472 bits), 59 bytes captured (472 bits) on interface \Device\NPF_{B586C5E...
> Ethernet II, Src: 4a:21:66:dd:ea:17 (4a:21:66:dd:ea:17), Dst: CloudNetwork_8e:2d:6b (50:c2:e8:8e:2d:6b)
> Internet Protocol Version 4, Src: 188.114.96.5, Dst: 192.168.26.32
> Transmission Control Protocol, Src Port: 80, Dst Port: 57655, Seq: 1508, Ack: 312, Len: 5
> [3 Reassembled TCP Segments (1512 bytes): #66(1410), #67(97), #68(5)]
v Hypertext Transfer Protocol, has 2 chunks (including last chunk)
  v HTTP/1.1 200 OK\r\n
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  
```

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.

*Wi-Fi

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

http

No.	Time	Source	Destination	Protocol	Length	Info
54	2.736959	192.168.26.32	188.114.96.5	HTTP/JSON	365	GET /posts/1 HTTP/1.1 , JSON (application/json)
68	3.072743	188.114.96.5	192.168.26.32	HTTP/JSON	59	HTTP/1.1 200 OK , JSON (application/json)

```

Date: Mon, 04 Nov 2024 11:27:08 GMT\r\n
Content-Type: application/json; charset=utf-8\r\n
Transfer-Encoding: chunked\r\n
Connection: keep-alive\r\n
Report-To: {"group":"heroku-nel","max_age":3600,"endpoints":[{"url":"https://nel.heroku.com/reports?ts=1724430238&sid=e11707d5-02a7-43e1-9b3d-3923dd3066d8"}]}
Reporting-Endpoints: heroku-nel=https://nel.heroku.com/reports?ts=1724430238&sid=e11707d5-02a7-43e1-9b3d-3923dd3066d8
Nel: {"report_to":"heroku-nel","max_age":3600,"success_fraction":0.005,"failure_fraction":0.05,"retry_wait_seconds":0}
X-Powered-By: Express\r\n
X-Ratelimit-Limit: 1000\r\n
X-Ratelimit-Remaining: 999\r\n
X-Ratelimit-Reset: 1724430280\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/"124-yiKdLzq05gfBrJFrCdJ8Yq0LGnU"\r\n
Via: 1.1 vegur\r\n
CF-Cache-Status: REVALIDATED\r\n
Server: cloudflare\r\n
CF-RAY: 8dd440cf9bfccfbc-MAD\r\n
  
```

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.

The image shows a Wireshark packet capture of an HTTP response. The packet list shows two packets: a GET request (No. 54) and a 200 OK response (No. 68). The response packet is selected, and the packet details pane shows the following information:

- Frame 68: 59 bytes on wire (472 bits), 59 bytes captured (472 bits) on interface \Device\NPF_{B586C5E...}
- Ethernet II, Src: CloudNetwork_8e:2d:6b (50:c2:e8:8e:2d:6b), Dst: 4a:21:66:dd:ea:17 (4a:21:66:dd:ea:17)
- Internet Protocol Version 4, Src: 188.114.96.5, Dst: 192.168.26.32
- Transmission Control Protocol, Src Port: 80, Dst Port: 57655, Seq: 1508, Ack: 312, Len: 5
- [3 Reassembled TCP Segments (1512 bytes): #66(1410), #67(97), #68(5)]
- Hypertext Transfer Protocol, has 2 chunks (including last chunk)
 - HTTP/1.1 200 OK\r\n
 - Response Version: HTTP/1.1
 - Status Code: 200
 - [Status Code Description: OK]
 - Response Phrase: OK

¿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.

The image shows two Wireshark packet captures. The top capture shows a curl request and response. The bottom capture shows a curl request and response.

Top Capture:

- Frame 54: 365 bytes on wire (2920 bits), 365 bytes captured (2920 bits) on interface \Device\NPF_{B586C5E...}
- Ethernet II, Src: CloudNetwork_8e:2d:6b (50:c2:e8:8e:2d:6b), Dst: 4a:21:66:dd:ea:17 (4a:21:66:dd:ea:17)
- Internet Protocol Version 4, Src: 192.168.26.32, Dst: 188.114.96.5
- Transmission Control Protocol, Src Port: 57655, Dst Port: 80, Seq: 1, Ack: 1, Len: 311
- Hypertext Transfer Protocol
 - GET /posts/1 HTTP/1.1\r\n
 - Request Method: GET
 - Request URI: /posts/1
 - Request Version: HTTP/1.1
 - Content-Type: application/json\r\n
 - User-Agent: PostmanRuntime/7.42.0\r\n

Bottom Capture:

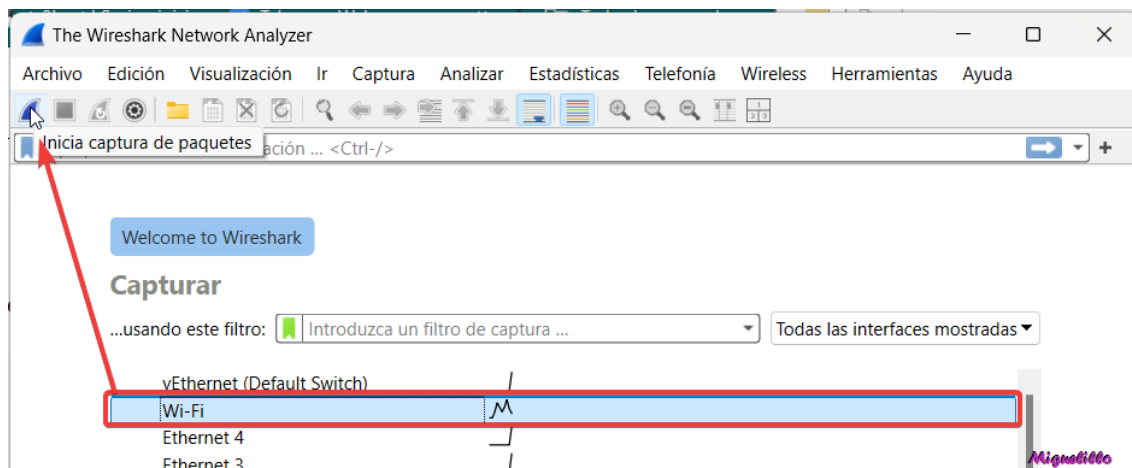
- Frame 261: 152 bytes on wire (1216 bits), 152 bytes captured (1216 bits) on interface \Device\NPF_{B58...}
- Ethernet II, Src: CloudNetwork_8e:2d:6b (50:c2:e8:8e:2d:6b), Dst: 4a:21:66:dd:ea:17 (4a:21:66:dd:ea:17)
- Internet Protocol Version 4, Src: 192.168.26.32, Dst: 188.114.96.5
- Transmission Control Protocol, Src Port: 57660, Dst Port: 80, Seq: 1, Ack: 1, Len: 98
- Hypertext Transfer Protocol
 - GET /posts/1 HTTP/1.1\r\n
 - Request Method: GET
 - Request URI: /posts/1
 - Request Version: HTTP/1.1
 - Host: jsonplaceholder.typicode.com\r\n
 - User-Agent: curl/8.9.1\r\n
 - Accept: */*\r\n

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**

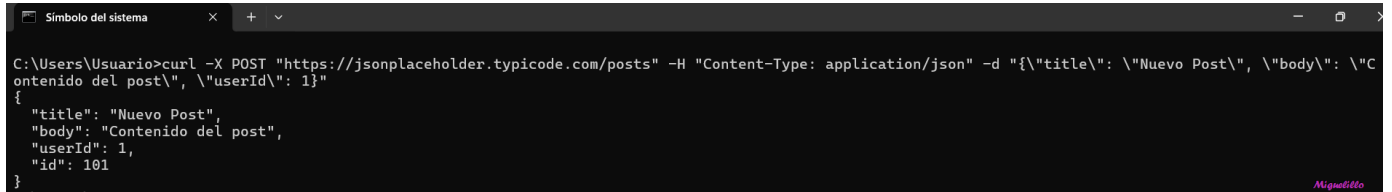


b) Realiza una petición POST.

Solución:

Realizamos la petición **POST**

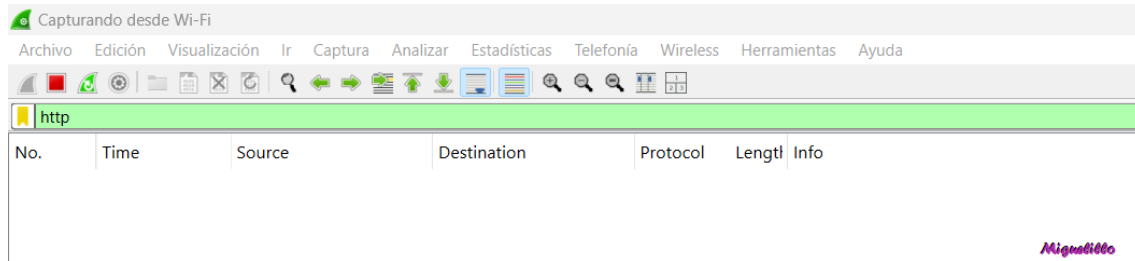
```
curl -X POST "https://jsonplaceholder.typicode.com/posts" -H  
"Content-Type: application/json" -d "{\"title\": \"Nuevo Post\",  
\"body\": \"Contenido del post\", \"userId\": 1}"
```



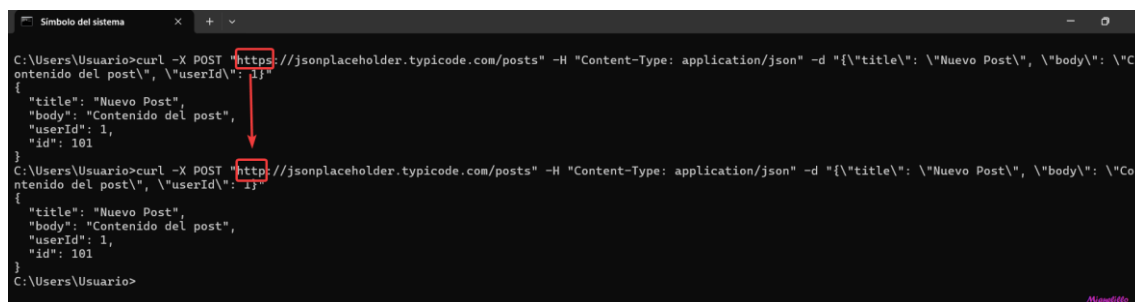
```
Símbolo del sistema
C:\Users\Usuario>curl -X POST "https://jsonplaceholder.typicode.com/posts" -H "Content-Type: application/json" -d "{\"title\": \"Nuevo Post\", \"body\": \"C  
ontenido del post\", \"userId\": 1}"
{"title": "Nuevo Post",  
"body": "Contenido del post",  
"userId": 1,  
"id": 101}
```

c) Filtra y analiza el tráfico

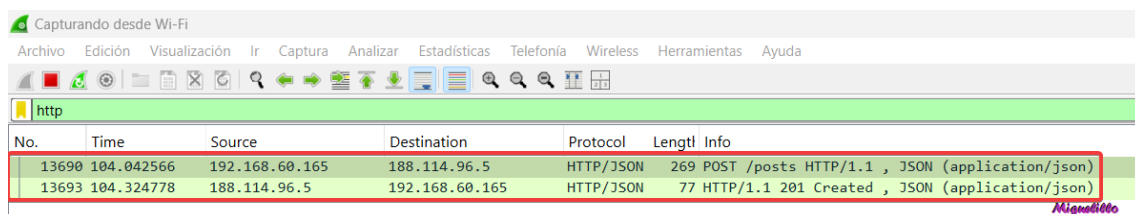
Solución:

**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.

Capturando desde Wi-Fi

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

http

No.	Time	Source	Destination	Protocol	Length	Info
44	4.193425	192.168.60.158	188.114.97.5	HTTP/JSON	269	POST /posts HTTP/1.1 , JSON (application/json)
50	4.482717	188.114.97.5	192.168.60.158	HTTP/JSON	85	HTTP/1.1 201 Created , JSON (application/json)

> Frame 44: 269 bytes on wire (2152 bits), 269 bytes captured (2152 bits) on interface \Device\NPF_{B58...}

> 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: 53992, Dst Port: 80, Seq: 1, Ack: 1, Len: 215

> Hypertext Transfer Protocol

POST /posts HTTP/1.1\r\n

Request Method: POST

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

JavaScript Object Notation: application/json

Object

Member: title

[Path with value: /title:Nuevo Post]

[Member with value: title:Nuevo Post]

String value: Nuevo Post

Key: title

[Path: /title]

Member: body

[Path with value: /body:Contenido del post]

[Member with value: body:Contenido del post]

String value: Contenido del post

Key: body

[Path: /body]

Member: userId

[Path with value: /userId:1]

[Member with value: userId:1]

Number value: 1

Key: userId

[Path: /userId]

- 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**.

Capturando desde Wi-Fi

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

http

No.	Time	Source	Destination	Protocol	Length	Info
44	4.193425	192.168.60.158	188.114.97.5	HTTP/JSON	269	POST /posts HTTP/1.1 , JSON (application/json)
50	4.482717	188.114.97.5	192.168.60.158	HTTP/JSON	85	HTTP/1.1 201 Created , JSON (application/json)

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: 66\r\n

[Content length: 66]

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

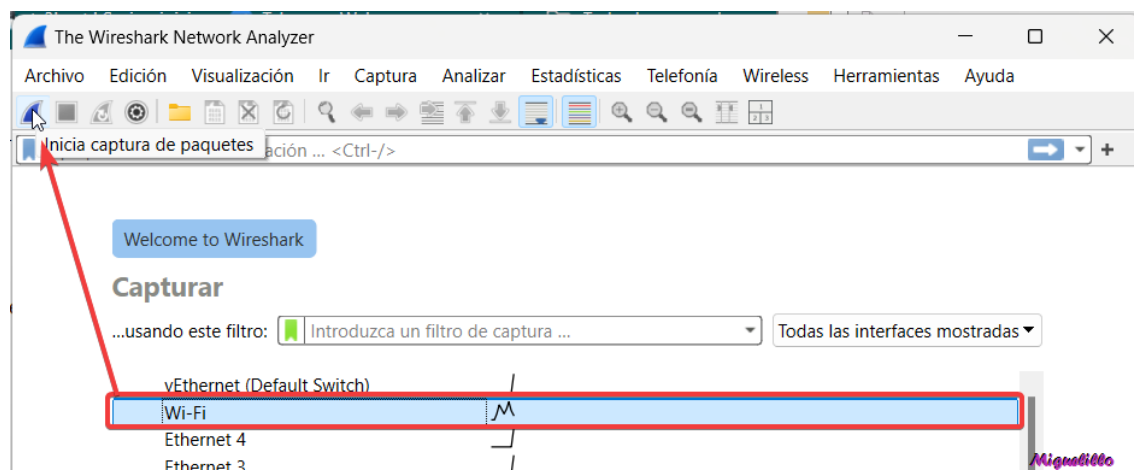
En POST podemos ver que se añaden el Content-Length y el Content-Type y el JavaScript Objet

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**

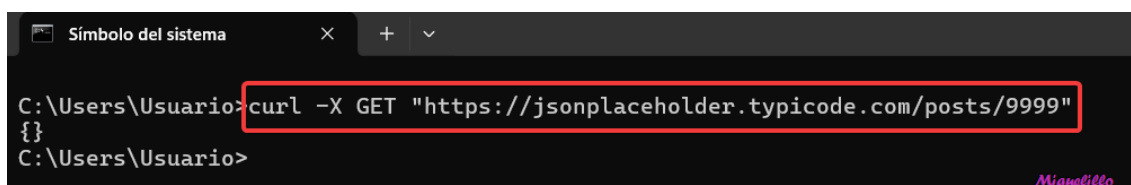


b) Realizar una Petición a un Recurso Inexistente

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.

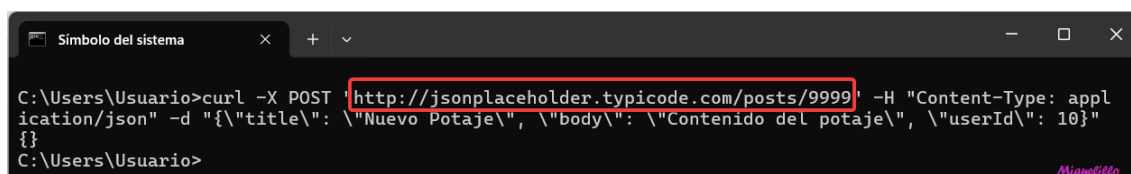
Hacemos un **GET**



```
Símbolo del sistema
C:\Users\Usuario>curl -X GET "https://jsonplaceholder.typicode.com/posts/9999"
{}
C:\Users\Usuario>
```

Miguelillo

Y también un **POST**



```
Símbolo del sistema
C:\Users\Usuario>curl -X POST 'http://jsonplaceholder.typicode.com/posts/9999' -H "Content-Type: application/json" -d '{"title": "Nuevo Potaje", "body": "Contenido del potaje", "userId": 10}'
{}
C:\Users\Usuario>
```

Miguelillo

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

Capturando desde Wi-Fi

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

http

No.	Time	Source	Destination	Protocol	Length	Info
198	14.115793	192.168.60.158	188.114.96.5	HTTP	155	GET /posts/9999 HTTP/1.1
204	14.366538	188.114.96.5	192.168.60.158	HTTP/JSON	1330	HTTP/1.1 404 Not Found , JSON (application/json)
254	17.133299	192.168.60.158	188.114.96.5	HTTP/JSON	279	POST /posts/9999 HTTP/1.1 , JSON (application/json)
259	17.357107	188.114.96.5	192.168.60.158	HTTP/JSON	1349	HTTP/1.1 404 Not Found , JSON (application/json)

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**

Capturando desde Wi-Fi

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

http

No.	Time	Source	Destination	Protocol	Length	Info
198	19.621203	192.168.60.158	91.189.91.48	HTTP	141	GET / HTTP/1.1
199	19.719379	91.189.91.48	192.168.60.158	HTTP	243	HTTP/1.1 204 No Content

> Frame 199: 243 bytes on wire (1944 bits), 243 bytes captured (1944 bits) on interface \Device\NPF_{B586C5E7-1E93-4149-BF35-98E732838958}, id 0

> Ethernet II, Src: Routerboardc_28:c0:b1 (48:a9:8a:28:c0:b1), Dst: CloudNetwork_8e:2d:6b (50:c2:e8:8e:2d:6b)

> Internet Protocol Version 4, Src: 91.189.91.48, Dst: 192.168.60.158

> Transmission Control Protocol, Src Port: 80, Dst Port: 54430, Seq: 1, Ack: 88, Len: 189

> Hypertext Transfer Protocol

HTTP/1.1 204 No Content\r\n

Response Version: HTTP/1.1

Status Code: 204

[Status Code Description: No Content]

Response Phrase: No Content

server: nginx/1.14.0 (Ubuntu)\r\n

date: Tue, 05 Nov 2024 09:17:38 GMT\r\n

x-cache-status: from content-cache-1ss/0\r\n

x-networkmanager-status: online\r\n

connection: close\r\n

\r\n

[Request in frame: 198]

[Time since request: 0.098176000 seconds]

[Request URI: /]

[Full request URI: http://connectivity-check.ubuntu.com/]

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

http

No.	Time	Source	Destination	Protocol	Length	Info
33	1.354345	192.168.60.158	188.114.96.5	HTTP/JSON	279	POST /posts/9999 HTTP/1.1 , JSON (application/json)
37	1.602934	188.114.96.5	192.168.60.158	HTTP/JSON	1342	HTTP/1.1 404 Not Found , JSON (application/json)

> Frame 37: 1342 bytes on wire (10736 bits), 1342 bytes captured (10736 bits) on interface \Device\NPF_{B586C5E7-1E93-4149-BF35-98E732838958}

> Ethernet II, Src: Routerboardc_28:c0:b1 (48:a9:8a:28:c0:b1), Dst: CloudNetwork_8e:2d:6b (50:c2:e8:8e:2d:6b)

> Internet Protocol Version 4, Src: 188.114.96.5, Dst: 192.168.60.158

> Transmission Control Protocol, Src Port: 80, Dst Port: 54579, Seq: 1, Ack: 226, Len: 1288

> Hypertext Transfer Protocol

HTTP/1.1 404 Not Found\r\n

Response Version: HTTP/1.1

Status Code: 404

[Status Code Description: Not Found]

Response Phrase: Not Found

¿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**.