

## 1. Del Cliente al Servidor →

- Explique paso a paso qué ocurre cuando usted escribe `www.youtube.com` en el navegador hasta que el video aparece en pantalla.
  - Debe incluir en su explicación:
    - Qué hace el cliente (navegador).
    - Qué papel cumple el `DNS`.
    - Qué ocurre con la dirección IP.
    - Qué hace el servidor.
    - Cómo entra en juego el protocolo `HTTP/HTTPS`.



Puedes incluir un dibujo o diagrama que muestre cómo viaja la información entre cliente, `DNS` y servidor.

Cuando se escribe la dirección Url de `www.youtube.com` en el buscador que en este caso es el cliente, el cliente manda un request a el servidor del cual se espera recibir un response. El DNS se encarga en transformar esa url (`www.youtube.com`) en la IP pública del servidor de Google. Luego, el servidor DNS no da un response con la ip del servidor de Youtube. Por último, nuestro navegador web hace un request al servidor de Google usando la IP que retornada por el DNS y el servidor de Youtube nos da una o más responses con el contenido de la página web de Youtube. HTTP es el protocolo estándar que utilizan los navegadores y es el que se encarga de enviar requests y recibir responses.

## 2. Frontend y Backend en acción

- Suponga que está construyendo una app web para agendar citas médicas.
  - Indique qué parte del sistema correspondería al `frontend` y cuál al `backend`.
  - Mencione tres tecnologías posibles para cada uno.
  - Explique brevemente cómo el `frontend` se comunicaría con el `backend` (mencione los conceptos de `API`, `HTTP` y `request/response`).

- La parte de la interfaz que ve el usuario final como los botones, apariencia, colores de la app o página web es la parte que corresponde al frontend web. Y la parte que da la respuesta o response da los request del frontend, donde está la lógica, así como las funciones de Post(publicar), Put(actualizar), Patch(actualizar de manera parcial) y Delete(Borrar) es el backend.
- Entre las tecnologías del Frontend están React, Angular, VueJS, Svelte. Y las de Backend son Django, Ruby, Laravel, Express.js, SQL, MySQL, MongoDB, PostgreSQL, Apache, entre otras.
- El Frontend se comunica con el Backend por medio de las API's. El Frontend es la parte visual que hace un request (hace una solicitud al servidor) al Backend y esta devuelve un response (devuelve una respuesta a la solicitud) al request inicial. Las API's son el medio por el que se comunican el Frontend y el backend esto debido a que las API's utilizan los protocolos HTTP para realizar esa comunicación. Un HTTP es un protocolo cliente-servidor de transferencia de hyper texto. Es el protocolo estándar para enviar requests y recibir responses.

### 3. REST vs SOAP vs GraphQL 🔒

Comparar estas opciones según formato de datos, flexibilidad, dificultad de implementación y uso actual, para entender cuándo conviene cada una.

#### Cómo completar la tabla:

- Usa términos concisos (p. ej., "JSON/XML" en formato de datos).
- En "nivel de flexibilidad", piensa: ¿quién controla mejor lo que se envía/recibe, cliente o servidor?
- En "dificultad de implementación", considera curva de aprendizaje, herramientas y mantenimiento.
- En "uso actual", valora su presencia en proyectos modernos (Alta/Media/Baja).

Complete la siguiente tabla:

Tipo de API	Formato de datos usado	Nivel de flexibilidad	Dificultad de implementación	Uso actual (Alta / Media / Baja)
REST				
SOAP				
GraphQL				

#### Luego conteste:

¿Cuál considera más apropiada para una startup moderna que desarrolla un sistema de reservas en línea? ¿Por qué?

Tipo de API	Formato de datos usado	Nivel de flexibilidad	Dificultad de implementación	Uso actual (Alta / Media / Baja)
REST	JSON	Ambos cliente y servidor	Fácil de implementar ya que es la estándar	Alta
SOAP	XML	Controla más el servidor	Son muy complejos de crear e implementar	Baja
GraphQL	JSON	Controla más el cliente	Fácil de implementar pero es nueva y falta más tiempo de uso	Media

Considero que la API más recomendada para una startup moderna que desarrolla un sistema de reservas en línea es REST, esto debido a que es la API estándar hasta el momento y la más usada con formato de datos JSON lo cual también la hace más fácil de utilizar. Lamentablemente SOAP es una API legacy lo que significa que es vieja y ya casi no la usan. Y GraphQL es muy nueva aún y la utilizan más que todo en empresas grandes.

## 4. Explorando APIs con Postman

### Parte 1: Selección de la API

1. Busca una API pública gratuita (sin autenticación obligatoria o con token gratuito).

Por ejemplo podrías utilizar:

- PokéAPI

2. Escribe en tu documento:

- Nombre de la API
- Breve descripción de lo que ofrece.

Nombre de la API: JSONPlaceholder

Es una API pública gratuita que simula datos reales para practicar solicitudes REST. Permite hacer GET, POST, PUT, PATCH y DELETE sin autenticación y es ideal para pruebas y aprendizaje de CRUD.

### Parte 2: Configuración en Postman

1. Crea una nueva colección en Postman con el nombre de la API.

2. Agrega al menos 3 tipos de solicitudes:

- **GET** → obtener información (por ejemplo, lista o detalle de un elemento).
- **POST** → enviar datos (por ejemplo, crear un recurso).
- **PUT/PATCH o DELETE** → actualizar o eliminar un recurso (si la API lo permite).

3. Si la API usa parámetros o tokens, configura las variables necesarias en la pestaña Environment.

Solicitud GET

(ABAJO ↓↓)

## GET

The screenshot shows the Postman interface for a GET request to <https://jsonplaceholder.typicode.com/posts>. The request method is set to GET. The response status is 200 OK, with a response time of 314 ms and a size of 8.02 KB. The response body is displayed as JSON, showing two posts:

```
1 [  
2   {  
3     "userId": 1,  
4     "id": 1,  
5     "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",  
6     "body": "quia et suscipit\\nsuscipit recusandae consequuntur expedita et cum\\nreprehenderit molestiae ut ut quas totam\\nnostrum  
    rerum est autem sunt rem eveniet architecto"  
7   },  
8   {  
9     "userId": 1,  
10    "id": 2,  
11    "title": "qui est esse",  
12  }]
```

## POST

Post está en la parte de abajo

(ABAJO ↓↓)

The screenshot shows the Postman interface with a POST request to <https://jsonplaceholder.typicode.com/posts?id=1&id=2>. The request body is a JSON object:

```
1 {  
2   "userId": 1,  
3   "id": 10,  
4   "title": "This is the new ID for the exercise",  
5   "body": "This is a sample"  
6 }
```

The response status is 201 Created, with a response time of 327 ms and a response size of 1.32 KB. The response body is identical to the request body.

## PATCH

The screenshot shows the Postman interface with a PATCH request to <https://jsonplaceholder.typicode.com/posts/1>. The request body is a JSON object:

```
1 {  
2   "userId": 1,  
3   "id": 1,  
4   "title": "This is the new ID for the exercise",  
5   "body": "This is a sample for PATCH method"  
6 }
```

The response status is 200 OK, with a response time of 290 ms and a response size of 1.31 KB. The response body is a modified version of the request body, indicating changes made via the PATCH method.

The screenshot shows the Postman application interface. At the top, there are two tabs: "PATCH https://jsonplaceholder.typicode.com/posts/1" and "POST https://jsonplaceholder.typicode.com/posts/1". The "PATCH" tab is active. Below the tabs, the URL "https://jsonplaceholder.typicode.com/posts/1" is displayed. To the right of the URL are "Save", "Share", and "Send" buttons. The "Send" button is highlighted in blue. Below the URL, there are tabs for "Docs", "Params", "Authorization", "Headers (8)", "Body", "Scripts", and "Settings". The "Body" tab is selected and has a green dot next to it. Under "Body", there are options: "none", "form-data", "x-www-form-urlencoded", "raw" (which is selected and highlighted in blue), "binary", "GraphQL", and "JSON". The "JSON" option has a dropdown arrow. Below these options is a code editor containing the following JSON:

```
1 {
2   "userId": 1,
3   "id": 1,
4   "title": "This is the new ID for the exercise",
5   "body": "This is a sample for PATCH method"
6 }
```

At the bottom of the interface, there are tabs for "Body", "Cookies", "Headers (24)", and "Test Results". The "Test Results" tab is selected and has a blue outline. To the right of the tabs, the response status is shown as "200 OK" with a green background, followed by "302 ms", "1.19 KB", and other metrics. Below the status, there are buttons for "Copy", "Edit", "Search", and "Close".

## ENVIRONMENT

The screenshot shows the "JSONPlaceholder Environment" section of Postman. At the top, there is a header with "PATCH https://jsonplaceholder.typicode.com/posts/1", "POST https://jsonplaceholder.typicode.com/posts/1", and "JSONPlaceholder Env...". Below the header, there is a "JSONPlaceholder Environment" title. To the right are "Fork" (with 0 forks) and "Share" buttons. The main area is a table titled "Variables" with columns "Variable" and "Value". The variables defined are:

Variable	Value
baseUrl	https://jsonplaceholder.typicode.com
postId	1
userId	1

Below the table, there is a link "Add variable".

Creé un environment llamado JSONPlaceholder Environment con las variables baseUrl, postId y userId.

## Parte 3: Ejecución y análisis 🔎

1. Ejecuta cada solicitud y verifica:

- Código de estado (200, 404, 500, etc.).
- Cuerpo de la respuesta (JSON o texto).
- Headers relevantes (por ejemplo: Content-Type, Authorization).

2. Guarda las respuestas en tu colección.

(Estos son las mismas imágenes de arriba)

The screenshot shows the Postman application interface. At the top, it displays a GET request to the URL <https://jsonplaceholder.typicode.com/posts>. Below the URL, there are tabs for Docs, Params, Authorization, Headers (6), Body, Scripts, and Settings. The Params tab is selected, showing a table for Query Params with one row containing 'Key'. In the main area, under the Body tab, the response is shown as JSON. The JSON output is as follows:

```
1 [  
2 {  
3   "userId": 1,  
4   "id": 1,  
5   "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",  
6   "body": "quia et suscipit\\nrecusandae consequuntur expedita et cum\\nreprehenderit molestiae ut ut quas totam\\nnostrum  
7       rerum est autem sunt rem eveniet architecto"  
8 },  
9 {  
10   "userId": 1,  
11   "id": 2,  
12   "title": "qui est esse",  
13 }]
```

The status bar at the bottom right indicates a 200 OK response with 314 ms duration and 8.02 KB size.

GET https://jsonplaceholder.typicode.com/posts?userId=1&id=2

POST https://jsonplaceholder.typicode.com/posts?id=1&id=2

Body

```
1 {  
2   "userId": 1,  
3   "id": 10,  
4   "title": "This is the new ID for the exercise",  
5   "body": "This is a sample"  
6 }
```

201 Created

{ } JSON

```
1 {  
2   "userId": 1,  
3   "id": 101,  
4   "title": "This is the new ID for the exercise",  
5   "body": "This is a sample"  
6 }
```

PATCH https://jsonplaceholder.typicode.com/posts/1

PATCH https://jsonplaceholder.typicode.com/posts/1

Body

```
1 {  
2   "userId": 1,  
3   "id": 1,  
4   "title": "This is the new ID for the exercise",  
5   "body": "This is a sample for PATCH method"  
6 }
```

200 OK

{ } JSON

```
1 {  
2   "userId": 1,  
3   "id": 1,  
4   "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",  
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 }
```

The screenshot shows the Postman application interface. At the top, the URL `https://jsonplaceholder.typicode.com/posts/1` is entered. Below it, a dropdown menu shows "PATCH". The main area has tabs for "Headers (8)", "Body", "Scripts", and "Settings". The "Headers" tab is currently active. A table lists eight headers with their values: Postman-Token (<calculated when request is sent>), Content-Type (application/json), Content-Length (<calculated when request is sent>), Host (<calculated when request is sent>), User-Agent (PostmanRuntime/7.49.1), Accept (\*/\*), Accept-Encoding (gzip, deflate, br), and Connection (keep-alive). At the bottom right, the response status is shown as "200 OK" with other details like 302 ms and 1.19 KB.

## Parte 4: Explicación técnica 🧐

Crea un archivo `README.md` (o documento de texto) que incluya:

1. Descripción general de la API elegida.
2. Explicación de cada solicitud:
  - Método HTTP usado.
  - Endpoint.
  - Parámetros o cuerpo de la solicitud.
  - Descripción breve de la respuesta.
3. Ejemplo de código o fragmento `JSON` con la respuesta.
4. Qué aprendiste del proceso.

Elegí JSONPlaceholder, una API pública que sirve para hacer pruebas de solicitudes REST. No requiere autenticación y permite usar métodos como GET, POST, PUT, PATCH y DELETE con datos ficticios.

### ◆ GET

Método: GET

Endpoint: GET `/posts/1`

Parámetros: No requiere.

Descripción de la respuesta: Devuelve un post de ejemplo con título, cuerpo y userId.

### ◆ POST

Método: POST

Endpoint: POST /posts

Body (JSON):

```
{  
  "userId": 1,  
  "title": "Nuevo post de prueba",  
  "body": "Contenido creado para el ejercicio"  
}
```

La API responde con código 201 y un objeto con un id generado para simular la creación.

◆ PATCH

Método: PATCH

Endpoint: PATCH /posts/1

Body:

```
{  
  "title": "Título editado"  
}
```

Descripción: La API devuelve el mismo recurso con el campo modificado. Los cambios no se guardan realmente porque es una API de prueba.

◆ DELETE

Método: DELETE

Endpoint: DELETE /posts/1

Parámetros: No requiere.

Descripción: Responde con código 200 o 204 para simular que el recurso fue eliminado.

## Parte 5: Reflexión final



En 1 o 2 párrafos, responde:

- ¿Qué aprendiste sobre el funcionamiento de las APIs ?
- ¿Cómo te ayudó Postman a entender la comunicación entre cliente y servidor?

Aprendí cómo funcionan las APIs y cómo cada solicitud (GET, POST, PATCH, DELETE) tiene un propósito y una respuesta diferente. También entendí la importancia de los códigos de estado y del formato JSON para comunicar datos entre sistemas.

Postman me ayudó a visualizar claramente esa comunicación entre cliente y servidor. Ver las solicitudes y respuestas en tiempo real hizo que el proceso fuera más fácil de entender y mucho más práctico.