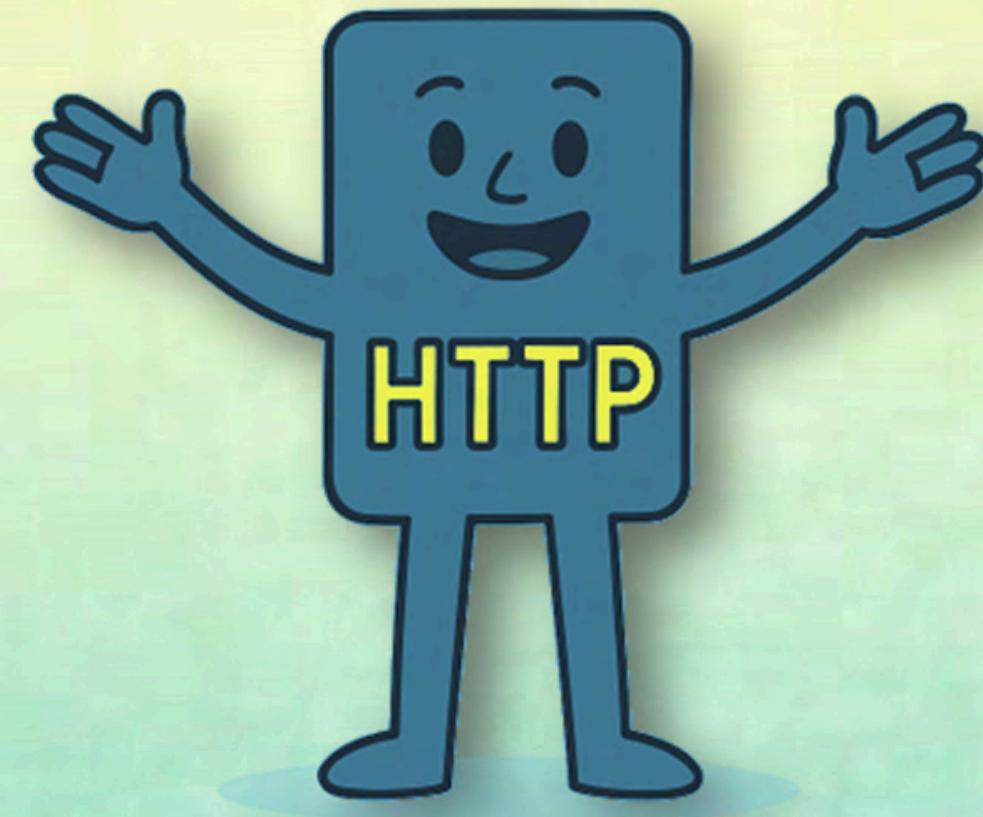


MÉTODOS DE PETICIÓN

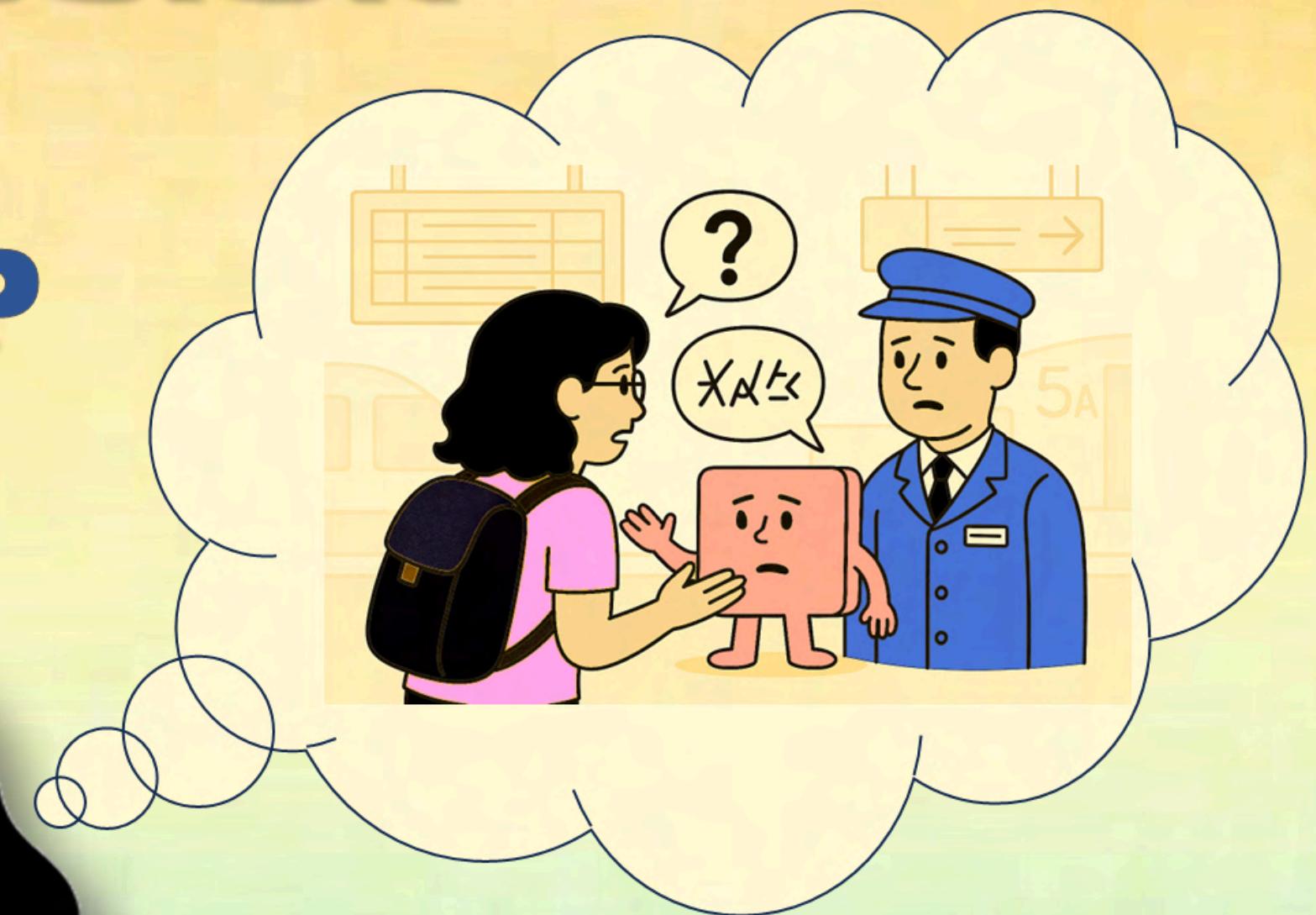
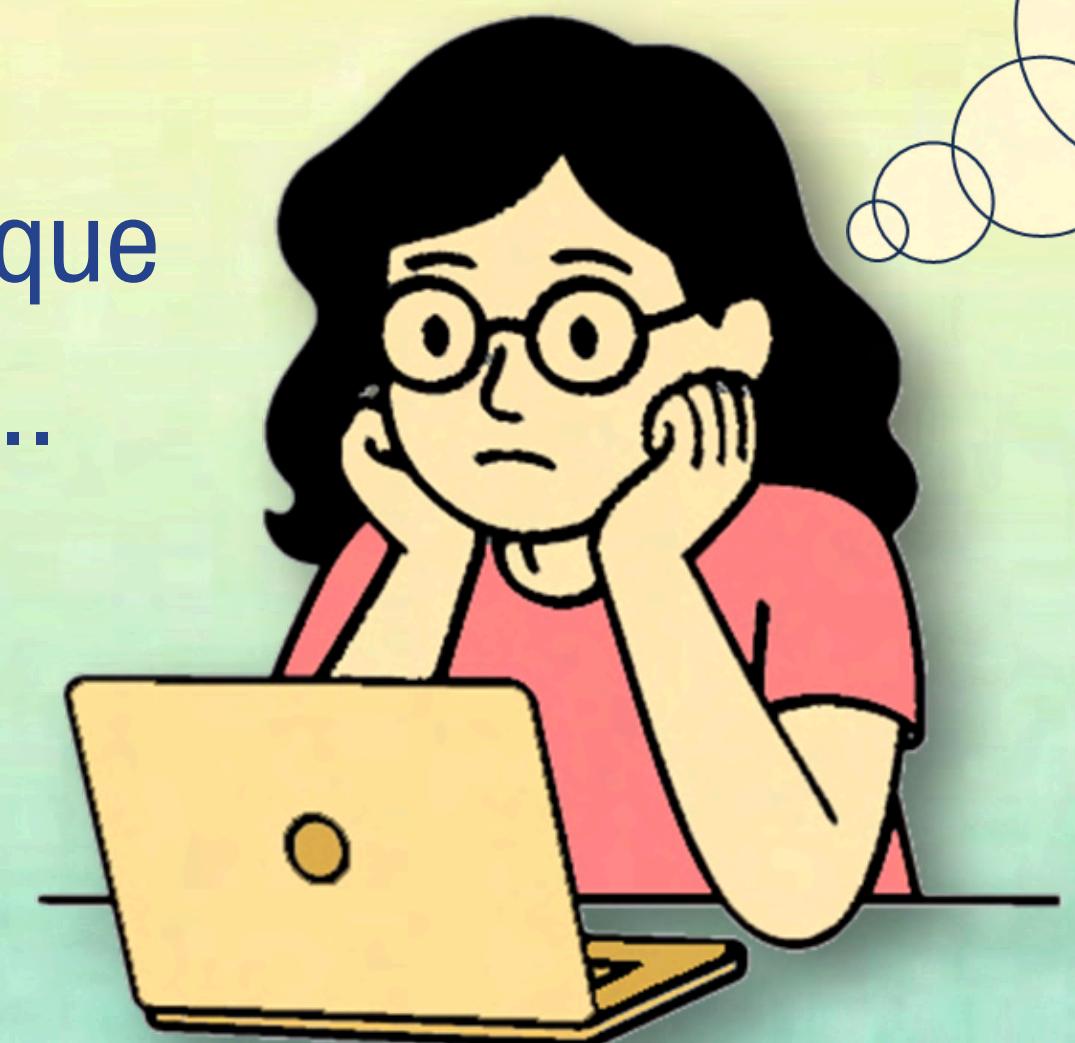
HTTP



INTRODUCCIÓN A HTTP

Imagina que estás,
en otro país, y quieres
comprar un billete.

Pero hay un problema:
no hablas el mismo idioma que
la persona del mostrador...



Entonces... ¿cómo haces para que te entienda?
Usas gestos, señales, dibujos...



¿No sería mejor que ambos conocieran el idioma
para poder comunicarse y entenderse?

Esto mismo pasa cada vez que
tu navegador quiere hablar con un servidor.



IMAGINA QUE....

Tienes una pregunta, pero quien puede ayudarte está lejos...



Entonces, le mandas una nota con un mensajero...



Y cuando la recibe, te responde con otra nota.



¡Así funciona Internet!

¿Y QUÉ PAPEL JUEGA HTTP?

HTTP es un PROTOCOLO.

Un sistema de reglas que siguen los mensajes para comunicarse.

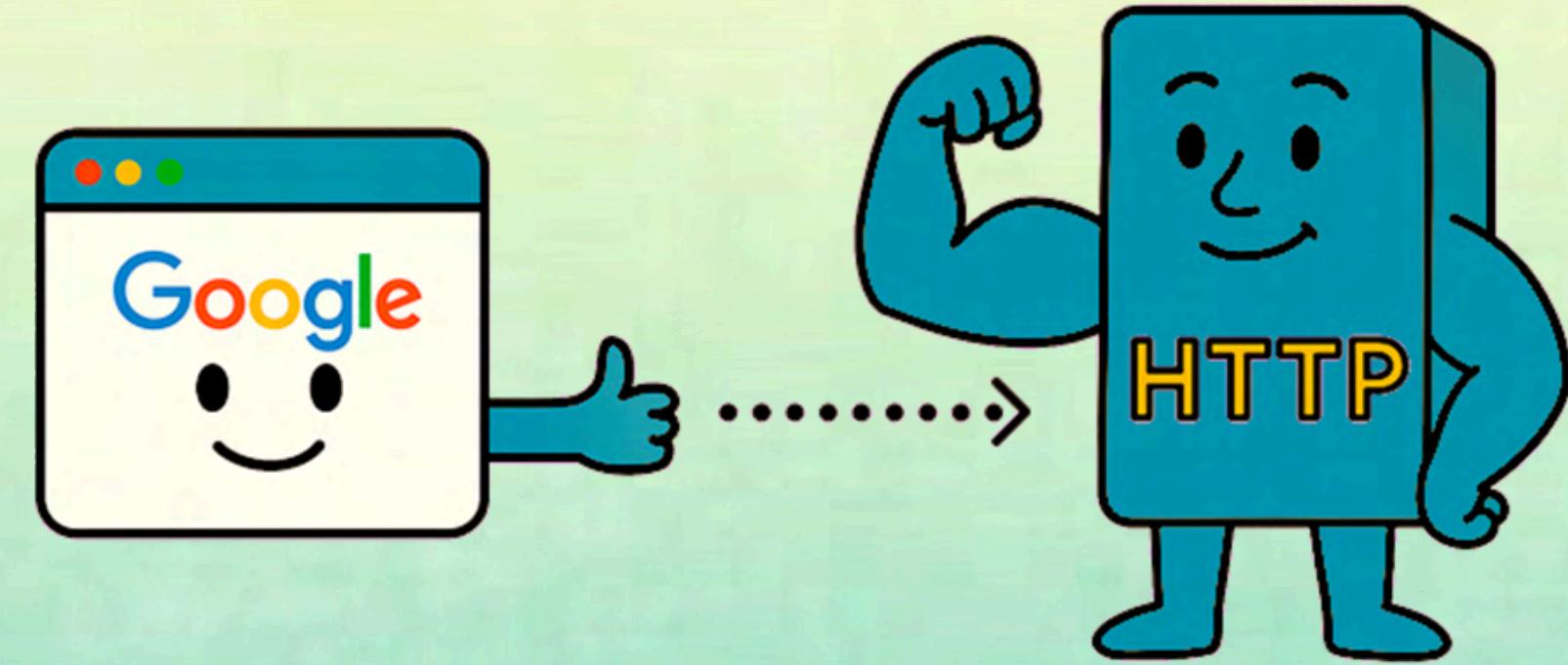
Es el idioma que usan los ordenadores
para hablar entre ellos en Internet .

PROCESO DE COMUNICACIÓN

Cuando buscas una página web, tu navegador pide información a un servidor.



Esa petición viaja usando **HTTP**, y la respuesta también.

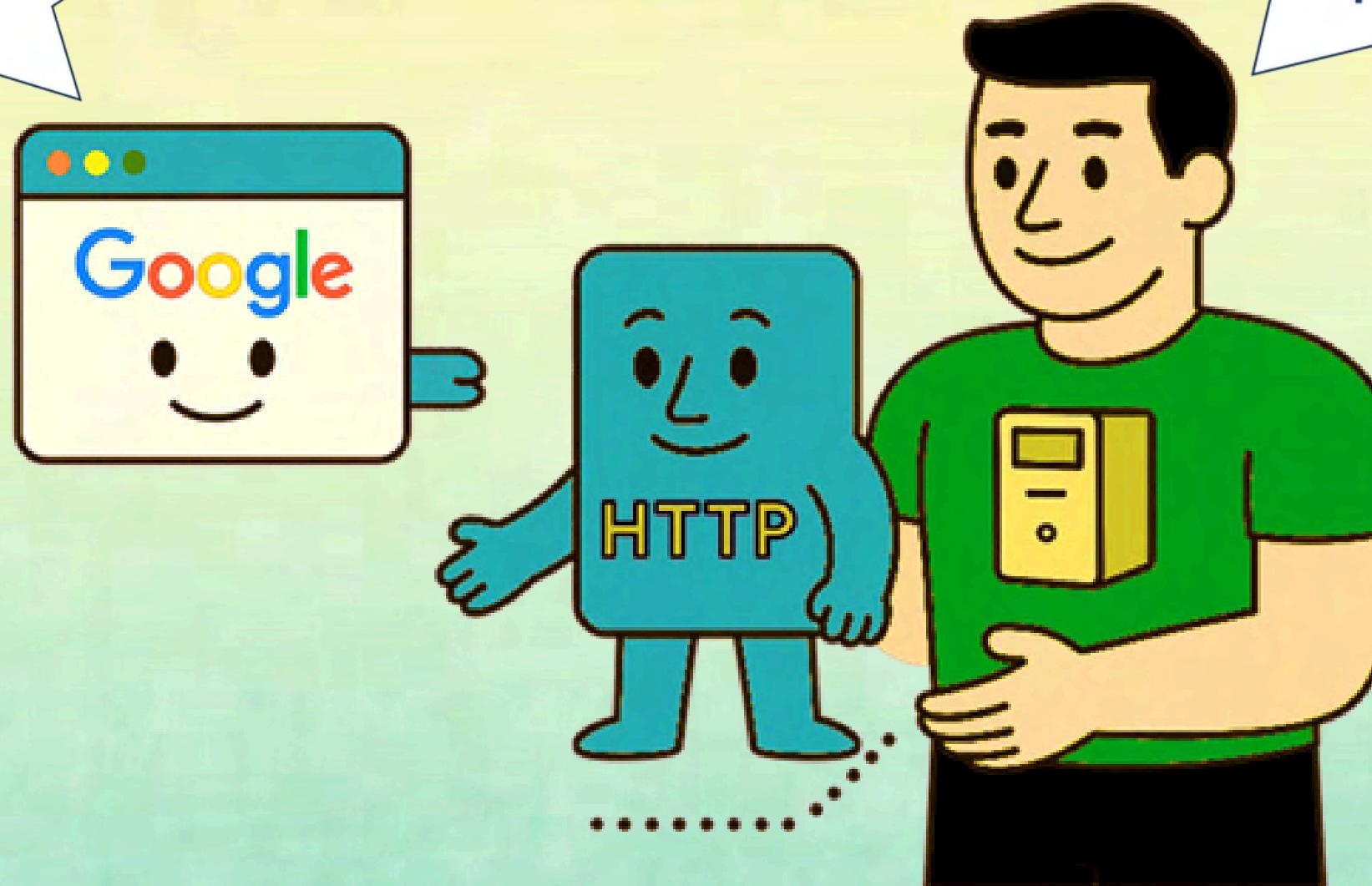


Es como si tu navegador dijera:

“Hola,
¿me puedes dar
la página principal
de factoriaf5.org?”

Y el servidor
le responde:

“Claro, aquí tienes
el archivo
que necesitas”.



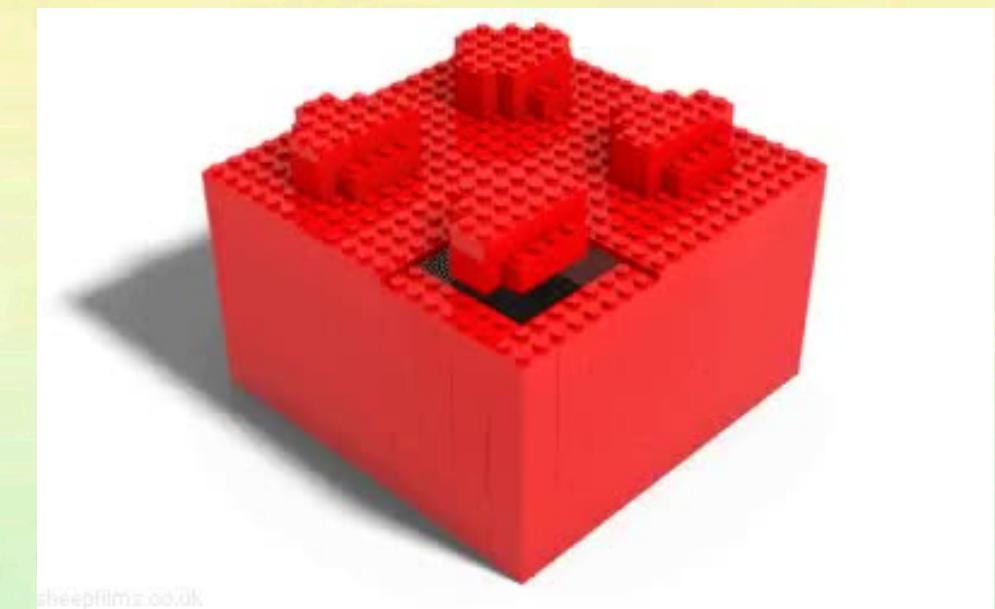
Todo esto sucede gracias a HTTP.



¿QUÉ SON LOS RECURSOS?

Todo lo que ves en una página web: imágenes, textos, estilos, botones...
Son recursos que el navegador pide al servidor.

Cada uno viaja por separado
como si fueran piezas de LEGO
que se ensamblan al llegar.



**Una página web es solo el resultado de muchas
PETICIONES HTTP**

Cada recurso tiene una dirección única llamada
URL (Localizador Uniforme de Recursos).
Es como una dirección postal.



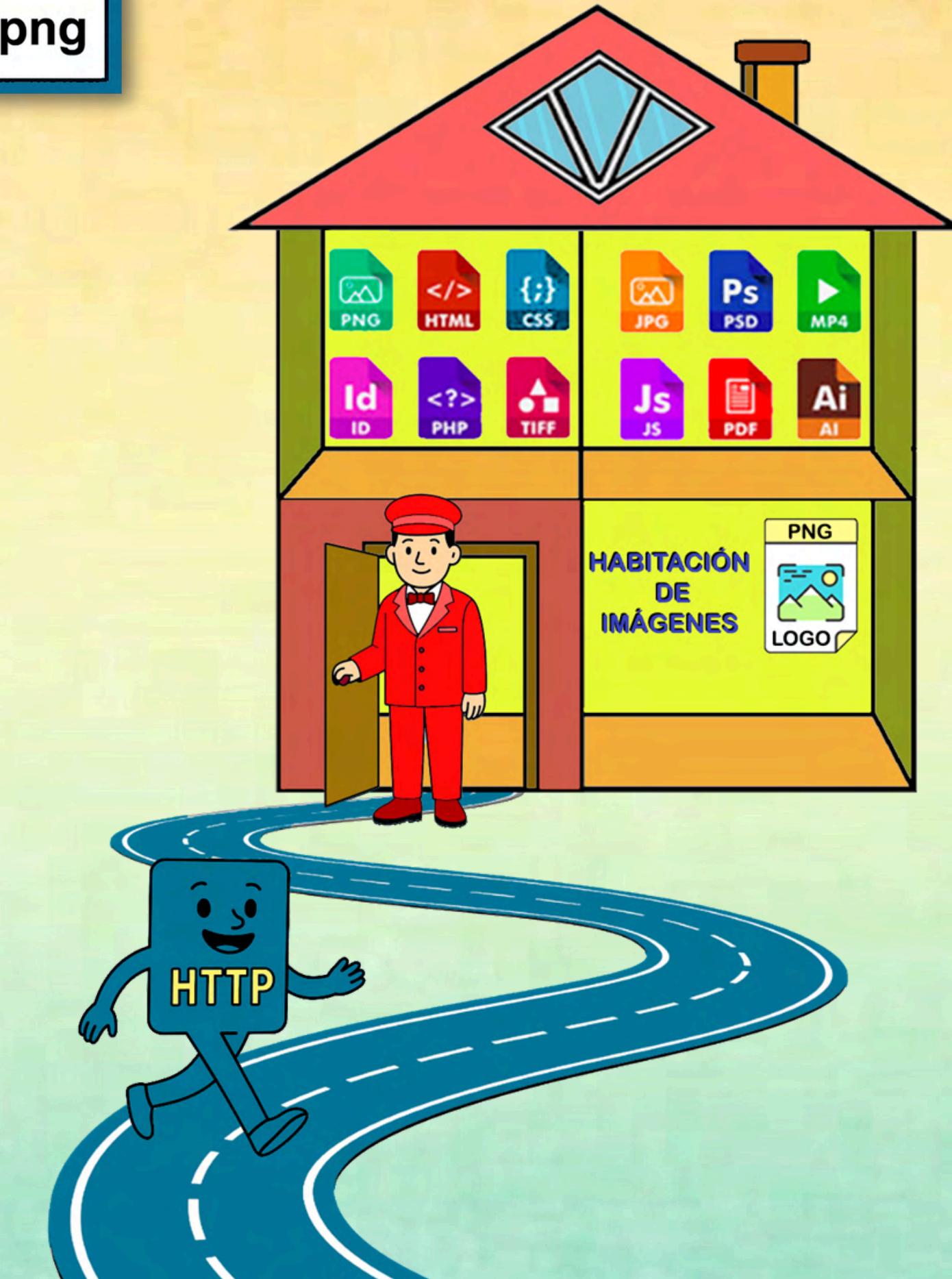
<http://www.factoriaf5.org/imagenes/logo.png>

PROTOCOLO:
es la carretera
HTTP

DOMINIO:
es la casa
factoriaf5.org

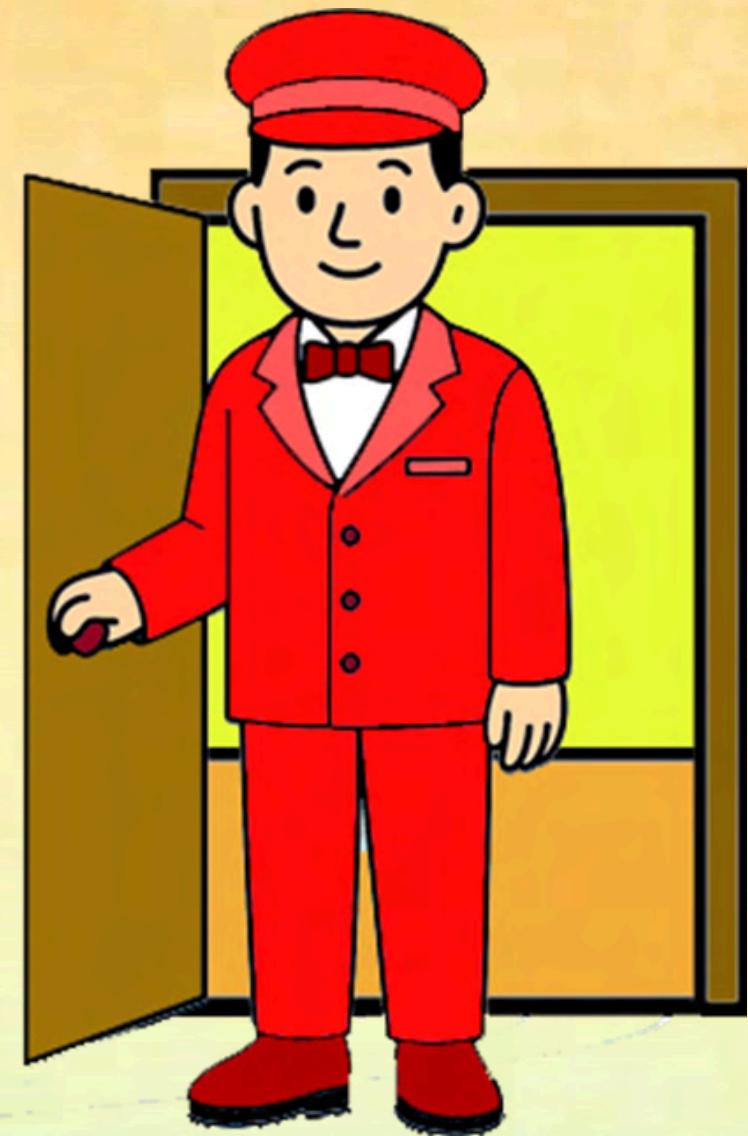
SUBDIRECTORIO:
la habitación
Imágenes

ARCHIVO:
la foto que buscabas
logo.png



SUBDOMINIO:
es el portero que te abre la puerta
www

No tenerlo no cambia el contenido de la web.
Es como decir “puedes entrar por la puerta principal o por la del jardín”.
Al final, llegas a la misma casa”.



Hoy en día no hace falta usar “www” porque los servidores entienden la dirección con o sin el subdominio.

Además, muchas webs redirigen automáticamente a su versión preferida.

Así, es más corto y fácil recordar la dirección sin “www”.

Explicación de cada parte:

PARTE	EJEMPLO	EXPLICACIÓN
Protocolo	http://	Cómo se comunican tu navegador y el sitio web.
Subdominio	www.	La entrada principal al sitio, como un portero.
Dominio	factoriaf5.org	El nombre de la “casa” o sitio web.
Subdirectorio	/imágenes/	Una carpeta dentro del sitio donde están guardados los archivos.
Archivo	logo.png	El archivo específico que quieras ver, como una foto.



YA CONOCEMOS

- 🌐 HTTP: el idioma que usa el navegador para hablar con el servidor.
- chrome icon Cliente: el navegador (nosotros), quien hace las peticiones.
- 💻 Servidor: quien responde a las peticiones con recursos.
- 📦 Recursos: todo lo que forma una web: texto, imágenes, estilos, botones...
- 🔗 URL: la dirección exacta del recurso que queremos.

Pero... ¿cómo sabe el servidor qué tiene que hacer cuando le pedimos algo?

Cuando el navegador hace una petición, no solo quiere una dirección (URL)...

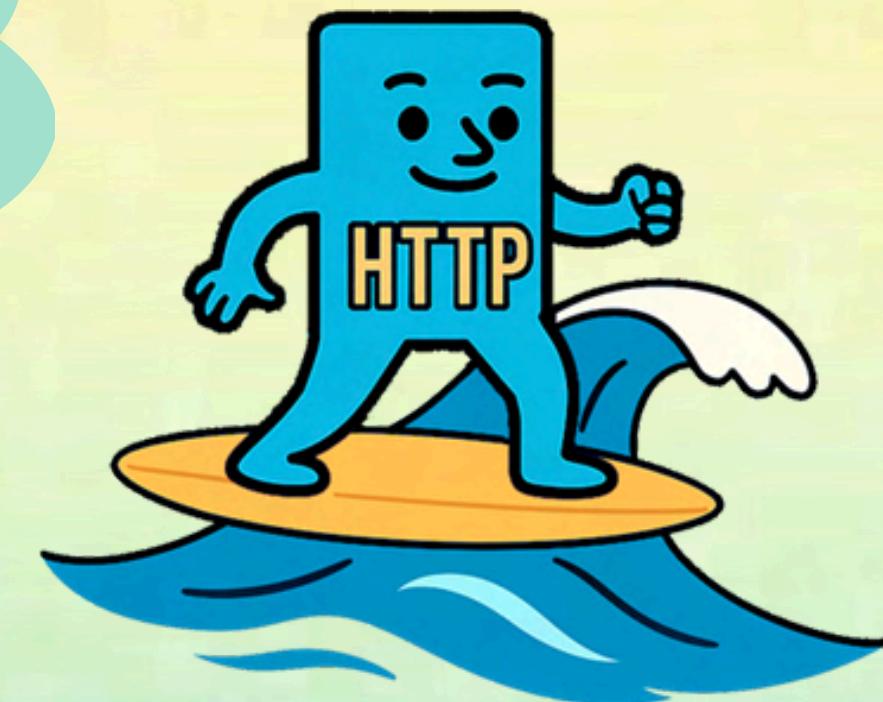
También le dice al servidor qué **acción** quiere realizar.

Y eso se consigue gracias a los **Métodos HTTP**.

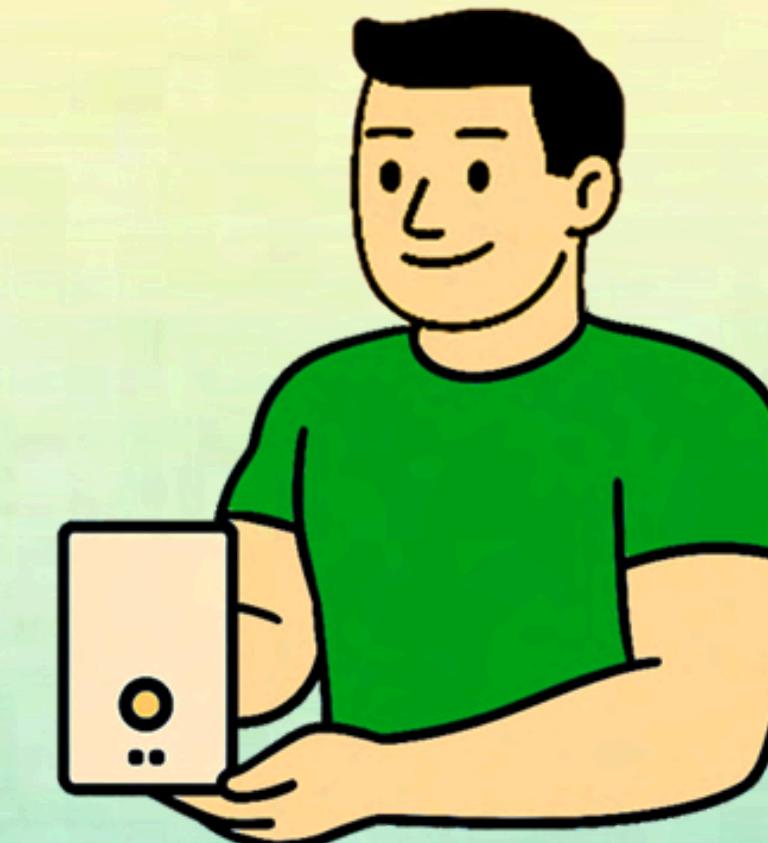
/fotos → "Solo quiero ver las fotos"
/registro → "Toma, los datos del nuevo usuario"
/cuenta/45 → "Borra esta cuenta"



Cliente



ACCIÓN



Servidor

DEFINICIÓN

LOS MÉTODOS HTTP

son instrucciones que el cliente envía al servidor para indicarle qué acción debe realizar sobre un recurso.

Estas acciones están definidas por el protocolo HTTP y permiten, por ejemplo: **consultar información, enviarla, modificarla o eliminarla.**

Es como cuando le pides algo a alguien:
que te enseñe algo, que guarde lo que le das, que cambie algo
que ya tiene o que tire algo que ya no necesitas.

**Eso mismo hace tu navegador cuando se comunica con un servidor:
le dice claramente qué quiere que haga con cierta información.**

VAMOS A CONOCER

A

LOS MÉTODOS

GET La Madrastra

GET pide y obtiene información sin modificar nada.

Cada vez que abres una página web, tu navegador usa **GET** para solicitar el contenido: texto, imágenes, estilos... sin cambiar nada en el servidor.

GET recupera datos, pero no los altera. Es un método seguro y cacheable, ideal para leer recursos.



POST **EI Cartero**

POST envía nueva información del usuario al servidor.

Cuando completas un formulario para registrarte o haces una compra online, **POST** se encarga de enviar esos datos al servidor.

POST transmite datos a un recurso específico y suele usarse para crear nuevos elementos o realizar acciones que modifican el servidor.



PUT El Albañil

PUT reemplaza completamente un recurso viejo con uno nuevo.

Si cambias todo tu perfil (nombre, foto, email...),
PUT borra lo anterior y pone todo de nuevo desde cero.

PUT reemplaza completamente el recurso existente por una nueva versión. Es idempotente, es decir, da el mismo resultado si se repite varias veces.



PATCH

La Pintora

PATCH retoca una parte del recurso sin tocar lo demás.

Si solo quieres cambiar tu número de teléfono en un perfil, **PATCH** lo hace sin reemplazar todo.

PATCH modifica parcialmente un recurso. Es más ligero que **PUT** y no reemplaza todo el contenido.



DELETE

El limpiador

DELETE borra recursos que ya no se necesitan.

Cuando decides eliminar un comentario o una cuenta,
DELETE es quien hace el trabajo.

DELETE elimina un recurso. También es idempotente, es decir, si se hace más de una vez, el resultado no cambia.



OPTIONS

La Reportera

OPTIONS informa sobre las acciones que están permitidas sobre un recurso.

Antes de hacer una petición compleja, **OPTIONS** te dice si puedes usar GET, POST, DELETE, etc.

OPTIONS muestra las opciones disponibles de comunicación. Es útil especialmente en CORS y validaciones previas.

*CORS (Cross-Origin Resource Sharing) es una norma que dice qué sitios web pueden acceder a los datos de otro sitio.



HEAD

La Tarotista

HEAD revisa sin tocar. Mira solo los encabezados de una respuesta, sin ver el contenido completo.

HEAD te permite comprobar si una imagen está disponible, sin tener que descargarla entera. Es como mirar si hay algo en una caja sin abrirla.

HEAD realiza una petición igual a **GET**, pero sin cuerpo de respuesta. Solo devuelve los encabezados.



TRACE

EI Detective

TRACE sirve para saber por dónde ha pasado una solicitud.

Es ideal para diagnosticar fallos en la comunicación.
Muestra el camino exacto que toma una petición para detectar errores. Es útil para debugging.

TRACE devuelve la misma solicitud que fue enviada,
reflejando la ruta y los datos.



CONNECT **El Túnel Digital**

CONNECT establece una conexión directa y segura, como un túnel privado.

Cuando accedes a una web con **HTTPS**, **CONNECT** crea el canal seguro por donde viajan tus datos.

CONNECT crea un túnel entre cliente y servidor. Se usa para canales cifrados y conexiones seguras.



¿Y CÓMO SABEMOS SI LA PETICIÓN FUNCIONÓ?

Cada vez que hacemos una solicitud con un método HTTP (como ver, enviar o borrar datos), el servidor responde con un **CÓDIGO DE ESTADO**

- ✓ Si todo va bien, devuelve un código positivo.
- ⚠ Si hay un problema, devuelve un código de error.

Estos códigos ayudan a entender si la solicitud fue exitosa, si algo salió mal o si hay que corregir algo.

CÓDIGOS DE ESTADO

Os ha pasado alguna vez que habéis querido volver a buscar en una web un vestido que os encantaba en rebajas y os ha aparecido este mensaje...



Cuando visitas una web, el navegador envía una petición al servidor...
Y el servidor responde con un número.
Ese es el **CÓDIGO DE ESTADO DE HTTP**

100
INFORMATIONAL



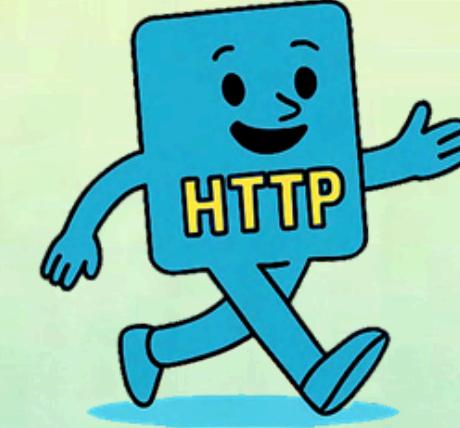
200
SUCCESS



300
REDIRECTION



400
CLIENT ERROR



500
SERVER ERROR



Aunque no siempre los ves, están ahí.
Te dicen si todo salió bien o algo ha fallado.

100-199

Informativos

😔 El servidor recibió la solicitud,...
pero aún no está listo para responder.

📌 Ejemplo:

100 Continue – El cliente puede seguir con su petición.

⌚ Son como un “espera, ya voy” de parte del servidor.
Se usan raramente, pero existen.



200-299

¡Todo bien!

🎉 ¡Éxito total! 🎉

El servidor procesó correctamente la solicitud.

📌 Ejemplos comunes:

- **200 OK** – Todo bien.
- **201 Created** – Se creó un nuevo recurso (como cuando registras una cuenta).
- **204 No Content** – Todo bien, pero sin contenido que mostrar.

⭐ Si no ves errores, ¡es gracias a estos códigos!



300-399

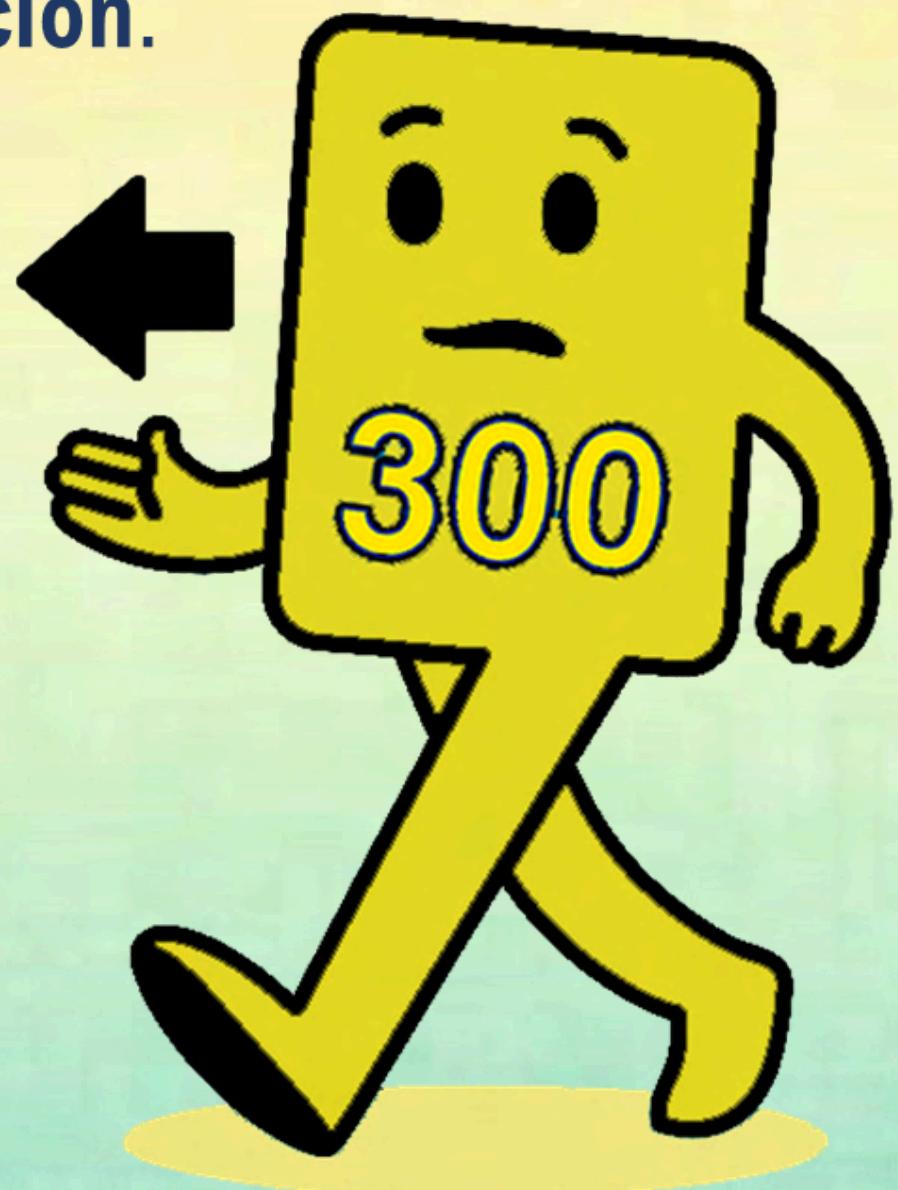
Redirección

📍 El recurso **se movió a otro lugar** o hay que **seguir otra dirección**.

📌 **Ejemplos útiles:**

- **301 Moved Permanently** – El recurso se mudó para siempre.
- **302 Found** – Se encuentra temporalmente en otra dirección.
- **304 Not Modified** – Usa la versión en caché, no hay cambios.

🚧 Es como un GPS que te manda a una nueva ruta.



400-499

Error del Cliente

✗ La solicitud tiene un **problema por parte del usuario**:
URL mal escrita, sin permisos, datos mal enviados...

📌 Errores típicos:

- **400 Bad Request** – Petición mal hecha.
- **401 Unauthorized** – Falta autenticación.
- **403 Forbidden** – Prohibido entrar.
- **404 Not Found** – No se encontró la página.

👤 El problema suele estar “de tu lado”.



500-599

Error del Servidor



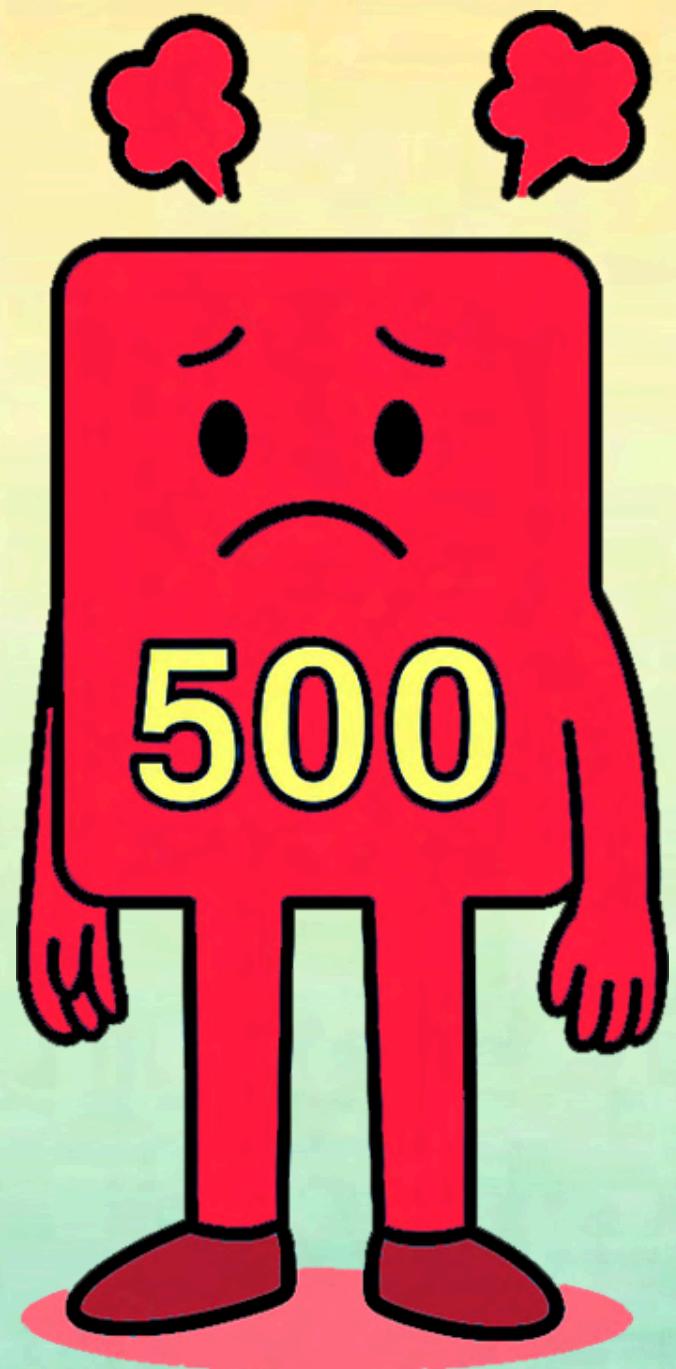
Todo estaba bien... pero el **servidor falló**.
Es como si al tocar el timbre, el portero se desmayara.



Errores comunes:

- **500 Internal Server Error** – Fallo interno del servidor.
- **502 Bad Gateway** – Problema entre servidores.
- **503 Service Unavailable** – El servidor está ocupado o caído.

🔧 Estos errores deben ser solucionados por quien administra el servidor.



¿Cómo usamos los métodos HTTP en el código?

Con fetch()

Es la herramienta de JavaScript que usamos para hacer peticiones al servidor desde el navegador.

¿Qué es fetch?

fetch() es una función de JavaScript que permite al cliente (el navegador) pedir recursos al servidor, enviar información, o hacer acciones como las que vimos con nuestros personajes (GET, POST, PUT, DELETE...).

EJEMPLO

```
fetch("https://api.example.com/usuarios", {  
  method: "GET"  
})  
.then(response => response.json())  
.then(data => console.log(data));
```

- // 1º Le pedimos al servidor este recurso (URL)
- // 2º Indicamos el método HTTP: en este caso, GET para obtener datos
- // 3º Cuando el servidor responde, convertimos esa respuesta a JSON
- // 4º Mostramos en la consola los datos que llegaron

Ejemplo de fetch explicado como una conversación entre el navegador y el servidor

```
1  ↴ fetch("https://api.ejemplo.com/usuarios", {  
2      method: "POST",                      //👉 ¿Qué acción? Enviar algo nuevo (POST)  
3  ↴   headers: {  
4      "Content-Type": "application/json" //👉 ¿Qué tipo de contenido estamos enviando? JSON  
5    },  
6  ↴   body: JSON.stringify({            //👉 ¿Qué datos estamos enviando?  
7      nombre: "Laura",  
8      email: "laura@correo.com"  
9    })  
10 }  
11 .then(response => response.json())    //👉 Cuando el servidor responde, convertimos la respuesta a JSON  
12 .then(data => console.log(data));      //👉 Mostramos lo que responde el servidor  
13
```

¿Qué le decimos con fetch?

Parte del código	¿Qué significa?
• <code>fetch("https://api.ejemplo.com/usuarios")</code>	A dónde ir: dirección del recurso
• <code>method: "POST"</code>	Qué hacer: enviar datos nuevos
• <code>headers</code>	Cómo lo enviamos: avisamos que los datos están en formato JSON
• <code>body</code>	Qué enviamos: los datos del nuevo usuario
• <code>.then(...)</code>	Qué hacemos cuando responde: ver la respuesta del servidor

TRADUCCIÓN

"Hola servidor,
quiero ir a <https://api.ejemplo.com/usuarios>
y usar el método POST
para enviarte estos datos
(nombre y email).
Te aviso que van en formato JSON.
Cuando me respondas,
lo leeré, y mostraré lo que me digas."



HEMOS APRENDIDO



- **HTTP** es el lenguaje de comunicación entre tu navegador y el servidor. Es como enviar una carta con instrucciones claras sobre qué quieres hacer.
- **Los Recursos** son todo lo que forma una web: imágenes, textos, estilos, botones... Tu navegador los pide uno a uno para construir la página.
- **Los Métodos HTTP** son las acciones que puede pedir el navegador al servidor:
 - **GET**: Traer información
 - **POST**: Enviar datos nuevos
 - **PUT**: Reemplazar por completo
 - **PATCH**: Modificar un poco
 - **DELETE**: Eliminar
 - **HEAD**: Comprobar sin descargar
 - **OPTIONS**: Ver qué está permitido
 - **TRACE** y **CONNECT**: Usados para pruebas y túneles de conexión segura



AHORA CONOCEMOS



- **Los Estados HTTP** son la forma en que el servidor responde a una petición, diciendo cómo fue todo:
- 100's – Informativos
 - El servidor recibió la petición y sigue procesándola.
- 200's – Correctos
 - Todo fue bien.
- 300's – Redirecciones
 - El recurso cambió de dirección
- 400's – Errores del cliente
 - El problema fue tuyo (navegador o app)
- 500's – Errores del servidor
 - El problema fue del servidor.



PRACTICAREMOS CON

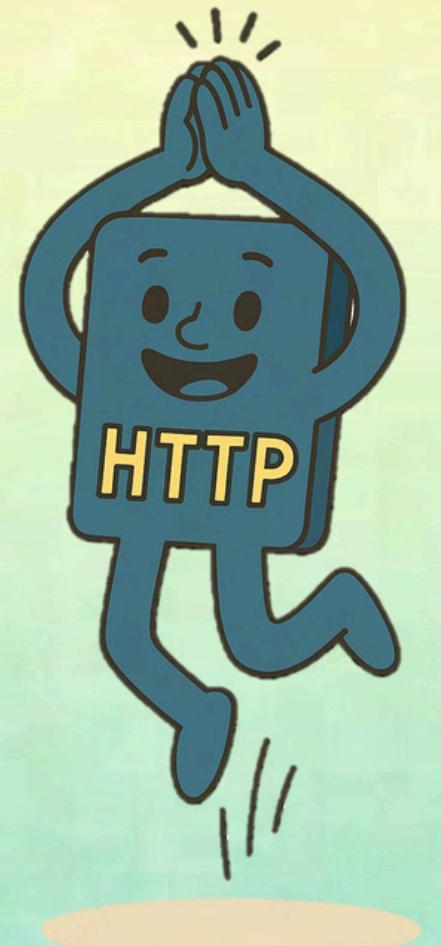


👉 **fetch()** es la herramienta en JavaScript para hacer peticiones.

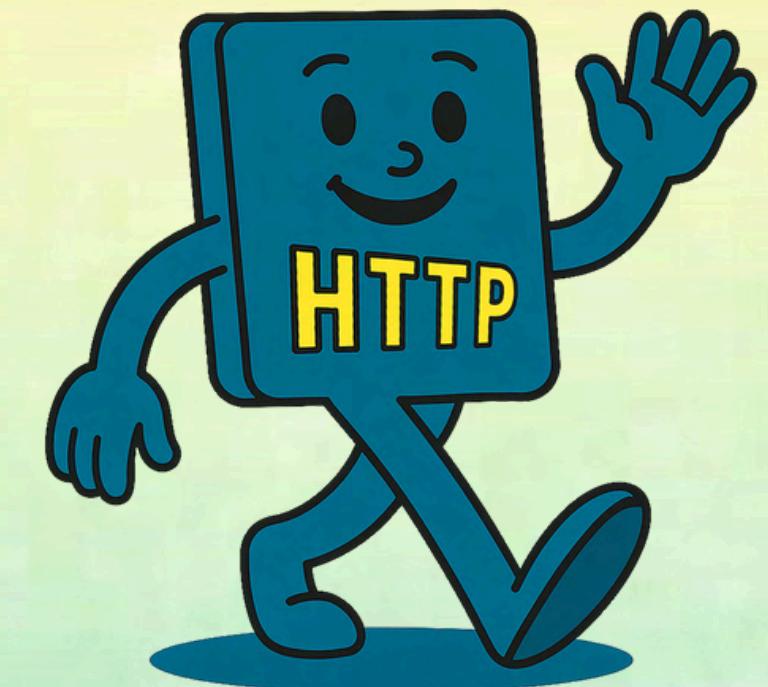
Con ella decimos:

- A dónde ir (URL)
- Qué hacer (method)
- Qué enviamos (body)
- Cómo lo enviamos (headers)
- Y qué hacer con la respuesta (.then())

 **¡Ya sabes cómo
funciona una web
por dentro y cómo
hablar con un servidor
como un@ pro!** 



**¡MUCHAS GRACIAS
POR
VUESTRA ATENCIÓN!**



ADAY