



## Unidad Académica de Economía

Carrera

Licenciatura en Sistemas Computacionales

Grupo y Semestre

Semestre 6

**Nombre estudiante**

Ricardo Matos Vizcarra

**Actividad**

Reto IA 1 – Conversación

**Maestro**

Eligardo Cruz Sánchez

**Materia**

Programación Distribuida del lado Cliente

UNIVERSIDAD AUTÓNOMA DE NAYARIT

Actúa como un tutor socrático experto en protocolos HTTP. Estoy aprendiendo sobre comunicación cliente-servidor y necesito entender profundamente la anatomía de HTTP. Contexto: Soy desarrollador frontend junior trabajando en "EcoMarket", un e-commerce para...

◆ Mostrar razonamiento ▾

¡Hola, colega de EcoMarket! Es un gusto saludarte. No te preocupes por sentir que tu conocimiento es superficial; en el desarrollo frontend, HTTP es el "idioma" que hablamos a diario, pero a veces nos acostumbramos a que las librerías hagan el trabajo sucio por nosotros.

Entender la anatomía de una petición te dará superpoderes para depurar errores y diseñar mejores experiencias para esos productores locales. Vamos a desglosarlo.

### 1. Headers de Request vs. Headers de Response

Imagina que los **Headers** son los metadatos de un envío postal: no son el producto en sí, sino las instrucciones de manejo, el remitente y el destinatario.

#### Headers de Request (Tus peticiones)

Son los que tú, como desarrollador frontend, envías al servidor para decirle quién eres y qué esperas recibir.

- **Ejemplo:** Quieres ver tu lista de productos orgánicos favoritos.
- **Header clave:** `Authorization` (para probar que eres tú) y `Accept-Language` (para que la descripción venga en español).

## 1. Headers de Request vs. Headers de Response

Imagina que los **Headers** son los metadatos de un envío postal: no son el producto en sí, sino las instrucciones de manejo, el remitente y el destinatario.

### Headers de Request (Tus peticiones)

Son los que tú, como desarrollador frontend, envías al servidor para decirle quién eres y qué esperas recibir.

- **Ejemplo:** Quieres ver tu lista de productos orgánicos favoritos.
- **Header clave:** `Authorization` (para probar que eres tú) y `Accept-Language` (para que la descripción venga en español).

HTTP

```
GET /api/v1/favorites HTTP/1.1
Host: api.ecomarket.com
Authorization: Bearer eyJhbGciOiJIUzI1...
Accept-Language: es-MX
```



### Headers de Response (Lo que el servidor te dice)

Son las instrucciones que el servidor de EcoMarket te devuelve para que tu navegador sepa cómo manejar los datos.

- **Ejemplo:** El servidor te entrega la lista de favoritos.
- **Header clave:** `Cache-Control` (para decirte cuánto tiempo puedes guardar estos datos sin volver a preguntar) y `Content-Type` (para decirte qué formato te envió).

### Headers de Response (Lo que el servidor te dice)

Son las instrucciones que el servidor de EcoMarket te devuelve para que tu navegador sepa cómo manejar los datos.

- **Ejemplo:** El servidor te entrega la lista de favoritos.
- **Header clave:** `Cache-Control` (para decirte cuánto tiempo puedes guardar estos datos sin volver a preguntar) y `Content-Type` (para decirte qué formato te envió).

HTTP

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Cache-Control: max-age=3600
X-Powered-By: Express

{
  "status": "success",
  "data": [{ "id": 1, "name": "Miel de Abeja Orgánica" }]
}
```

## 2. Los peligros de elegir mal el método HTTP

En el mundo de las APIs REST, el método (verbo) define la intención. Equivocarse no es solo un error de semántica; tiene consecuencias técnicas graves:

1. **Seguridad (Usar GET en lugar de POST para login):** Si usas `GET /login?user=admin&pass=123`, los datos de tu productor local quedan grabados en el historial del navegador, en los logs del servidor y en los proxies intermedios. **GET envía datos en la URL**, mientras que POST los esconde en el cuerpo (body) de la petición.
2. **Duplicados (Usar POST en lugar de PUT para actualizar stock):** `POST` no es

## 2. Los peligros de elegir mal el método HTTP

En el mundo de las APIs REST, el método (verbo) define la intención. Equivocarse no es solo un error de semántica; tiene consecuencias técnicas graves:

1. **Seguridad (Usar GET en lugar de POST para login):** Si usas `GET /login?`

`user=admin&pass=123`, los datos de tu productor local quedan grabados en el historial del navegador, en los logs del servidor y en los proxies intermedios. **GET envía datos en la URL**, mientras que POST los esconde en el cuerpo (body) de la petición.

2. **Duplicados (Usar POST en lugar de PUT para actualizar stock):** `POST` no es

**idempotente**. Si un productor intenta actualizar el stock de "Tomates" y la conexión falla, al reintentar un POST, podrías terminar creando un nuevo registro de producto por error. `PUT`, al ser idempotente, garantiza que si envías la misma petición 10 veces, el resultado final sea siempre el mismo: el producto con el ID específico actualizado.

3. **Inconsistencia de datos (Usar PUT en lugar de PATCH):** Supongamos que el producto

"Café de Altura" tiene 10 campos (nombre, precio, stock, origen, etc.). Si quieres actualizar **solo el precio** usando `PUT`, el servidor espera que envíes el objeto *completo*. Si solo envías el precio, un servidor mal configurado podría borrar el resto de los campos (dejándolos en `null`), porque `PUT` significa "reemplazar todo". Para cambios parciales, el héroe es `PATCH`.

## 3. El momento de descubrir el Content-Type

Ahora, pasemos a la parte interactiva. En EcoMarket, vas a recibir mucha información de la API. Imagina que el servidor te envía una respuesta que contiene una larga cadena de bytes (ceros y unos).

Para que tú puedas hacer un `data.map()` en React y mostrar tus productos, el navegador necesita interpretar esos bytes correctamente. Aquí es donde entra el header `Content-Type`.

### 3. El momento de descubrir el Content-Type

Ahora, pasemos a la parte interactiva. En EcoMarket, vas a recibir mucha información de la API. Imagina que el servidor te envía una respuesta que contiene una larga cadena de bytes (ceros y unos).

Para que tú puedas hacer un `data.map()` en React y mostrar tus productos, el navegador necesita interpretar esos bytes correctamente. Aquí es donde entra el header `Content-Type`.

**Piénsalo un momento y dime:**

1. Si el servidor te envía una lista de productos, pero en el header `Content-Type` dice `text/plain` en lugar de `application/json`, ¿qué crees que pasaría cuando intentes acceder a `respuesta.data.name` en tu código JavaScript?
2. Considerando que en EcoMarket los productores subirán fotos de sus cosechas, ¿cómo crees que el servidor sabe si lo que le estás enviando en un POST es una imagen `.jpg` o un archivo de configuración `.json`?

¿Qué me dices? Estaré esperando tu respuesta para profundizar en el misterio del parsing.



1.- no lo interpretara correctamente, 2.- Porque directamente le estas diciendo en la cabecera lo que va a ver por "Content-Type"

◆ Mostrar razonamiento ▾

¡Exacto! Vas por muy buen camino. Como bien dices, el **Content-Type** es el "manual de instrucciones" que acompaña a los datos.

Para profundizar un poco más en tu primera respuesta: si el servidor envía `text/plain`, tu navegador o tu cliente de API (como `fetch` o `axios`) tratará toda la información como una simple **cadena de texto (string)**.

Si intentas hacer:

JavaScript

```
console.log(respuesta.data.name);
// Resultado: undefined (¡o incluso un error!),
// porque no puedes pedirle una propiedad a un texto plano.
```

**El reto del "Traductor"**

Ahora que ya sabes que el **Content-Type** le dice al receptor qué es lo que está recibiendo, vamos a complicar un poco las cosas para que domines el flujo en **EcoMarket**.

Imagina este escenario: Un productor local está subiendo un nuevo producto (un frasco de mermelada) mediante un formulario en tu frontend.

### El reto del "Traductor"

Ahora que ya sabes que el `Content-Type` le dice al receptor qué es lo que está recibiendo, vamos a complicar un poco las cosas para que domines el flujo en **EcoMarket**.

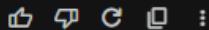
Imagina este escenario: Un productor local está subiendo un nuevo producto (un frasco de mermelada) mediante un formulario en tu frontend.

**Responde a estas dos preguntas para seguir avanzando:**

1. Si tú envías un `POST` con los datos del producto en formato JSON, pero **olvidas** incluir el header `Content-Type: application/json` en tu petición (Request), ¿qué crees que recibirá el programador backend en su variable `request.body`? (Pista: Piensa en qué hace el servidor cuando no sabe qué idioma hablas).
2. Existe otro header muy parecido llamado `Accept`. Si el `Content-Type` sirve para decir "*Esto es lo que te estoy enviando*", ¿para qué crees que sirve el header `Accept` que envía el cliente (tú) al servidor?

**Contexto para la pregunta 2:** Imagina que la API de EcoMarket es muy avanzada y puede devolver la lista de ventas en formato **JSON**, en un archivo **PDF** listo para imprimir, o en un **CSV** para Excel.

¿Qué opinas? Con esto cerramos el círculo de la negociación de contenido.



1.- Probablemente tenga un error ya que lo que se espera recibir no va a ser lo que se quiere almacenar. 2.- No se que haga Accept, en lo que me pudo imaginar suena a que busca algo que esta en el servidor por medio de una verificación.

◆ Mostrar razonamiento ▾

¡Vas muy bien encaminado! Tu instinto sobre el error es totalmente correcto.

### 1. El misterio del Body vacío

Si no envias el `Content-Type`, el servidor de EcoMarket recibe una ráfaga de datos, pero como no tiene la "etiqueta" que diga que es JSON, el programador backend verá su variable `request.body` como un objeto **vacio** o un **undefined**.

Es como si le enviaras al productor una caja cerrada sin ninguna nota: el productor sabe que le llegó algo, pero no sabe si debe abrirlo con cuidado (JSON) o si es basura. Por eso, el servidor suele responder con un error **415 Unsupported Media Type** o un **400 Bad Request**.

### 2. El header `Accept` : El "Menú" del Cliente

Tu sospecha de que "busca algo" tiene sentido. Mientras que el `Content-Type` le dice al servidor: "Oye, te estoy enviando un JSON", el header `Accept` le dice: "Oye, cuando me respondas, por favor hazlo en este formato".

Se llama **Negociación de Contenido**.

**Ejemplo en EcoMarket:** Imagina que quieres descargar el reporte de ventas del mes.

## 2. El header `Accept` : El "Menú" del Cliente

Tu sospecha de que "busca algo" tiene sentido. Mientras que el `Content-Type` le dice al servidor: "Oye, te estoy enviando un JSON", el header `Accept` le dice: "Oye, cuando me respondas, por favor hazlo en este formato".

Se llama **Negociación de Contenido**.

**Ejemplo en EcoMarket:** Imagina que quieres descargar el reporte de ventas del mes.

- Si tu frontend envía: `Accept: application/json`, el servidor te envía datos para que los pintes en una tabla.
- Si tu frontend envía: `Accept: application/pdf`, el servidor te genera un documento listo para descargar.

---

### El último paso: ¿Qué pasó con mi petición?

Ya sabemos cómo hablar (headers) y qué decir (body). Pero en el mundo real, las cosas fallan: el internet se cae, el producto no existe o no tienes permiso para entrar a la zona de administración de EcoMarket.

Para no tener que leer todo el cuerpo del mensaje y descifrar qué pasó, HTTP usa los **Status Codes** (Códigos de estado).

Aquí van mis dos preguntas finales para cerrar tu entrenamiento de hoy:

1. Si intentas borrar un producto que ya no existe en la base de datos de EcoMarket, ¿qué código numérico esperarías recibir del servidor para saber que no se encontró?
2. Imagina que un productor intenta entrar a `/admin/ventas` sin haber iniciado sesión. El servidor le responde con un código **401 Unauthorized**. ¿Qué diferencia crees que hay entre ese **401** y un **403 Forbidden**? (Pista: Piensa en la diferencia entre "No sé quién

### El último paso: ¿Qué pasó con mi petición?

Ya sabemos cómo hablar (headers) y qué decir (body). Pero en el mundo real, las cosas fallan: el internet se cae, el producto no existe o no tienes permiso para entrar a la zona de administración de EcoMarket.

Para no tener que leer todo el cuerpo del mensaje y descifrar qué pasó, HTTP usa los **Status Codes** (Códigos de estado).

Aquí van mis dos preguntas finales para cerrar tu entrenamiento de hoy:

1. Si intentas borrar un producto que **ya no existe** en la base de datos de EcoMarket, ¿qué código numérico esperarías recibir del servidor para saber que no se encontró?
2. Imagina que un productor intenta entrar a `/admin/ventas` sin haber iniciado sesión. El servidor le responde con un código **401 Unauthorized**. ¿Qué diferencia crees que hay entre ese **401** y un **403 Forbidden**? (Pista: Piensa en la diferencia entre "No sé quién eres" y "Sé quién eres, pero no tienes permiso para estar aquí").

¿Cómo los diferenciarías en la interfaz de EcoMarket para ayudar al usuario?



1.- 404 not found. 2.- en 401 no tiene autorización de entrar hasta que se cumpla el haber iniciado sesión, mientras que el 403 esta prohibido entrar porque así fue marcado.

◆ Mostrar razonamiento ▾

¡Distinción perfecta! Has dado en el clavo con uno de los conceptos que más confunde a los desarrolladores cuando empiezan.

Para que nunca se te olvide en el código de **EcoMarket**:

- **401 (Unauthenticated)**: Es como llegar a la puerta del almacén y que el guardia te diga: "*¿Quién eres? No te conozco, enséñame tu identificación*".
  - **Tu acción en el frontend**: Redirigir al usuario al formulario de **Login**.
- **403 (Unauthorized/Forbidden)**: Es como si el guardia viera tu identificación, leyera que eres "Repartidor" y te dijera: "*Sé quién eres, pero solo los 'Administradores' pueden entrar a esta oficina*".
  - **Tu acción en el frontend**: Mostrar un mensaje de "**Acceso denegado**" o una ilustración de "No tienes permisos".

### El Mapa de los Estados (Status Codes)

Como desarrollador en EcoMarket, tu código debe estar listo para reaccionar según la "familia" del código que recibas. Aquí tienes el mapa mental rápido:

| Familia | Significado | Ejemplo en EcoMarket |
|---------|-------------|----------------------|
|---------|-------------|----------------------|

2xx ;Genial! Todo salió bien.

201 Created : El nuevo producto se guardó con éxito

## El Mapa de los Estados (Status Codes)

Como desarrollador en EcoMarket, tu código debe estar listo para reaccionar según la "familia" del código que recibas. Aquí tienes el mapa mental rápido:

| Familia | Significado                             | Ejemplo en EcoMarket   |
|---------|---|--|
| 2xx     | ¡Genial! Todo salió bien.               | 201 Created : El nuevo producto se guardó con éxito.         |
| 3xx     | Vete a otro lado. Redirección.          | 304 Not Modified : El navegador usa la versión en caché.     |
| 4xx     | Metiste la pata. Error del cliente.     | 400 Bad Request : Enviste el precio como texto y era número. |
| 5xx     | El servidor explotó. Error del backend. | 500 Internal Server Error : El código del backend falló.     |

 Exportar a Hojas de cálculo 

## Tu prueba de fuego en el código

Has demostrado que entiendes la teoría detrás de los headers, los métodos y los estados.

Ahora, bajémoslo a la implementación real que harás en **EcoMarket**.

Imagina que estás usando `fetch` para enviar un nuevo producto desde un formulario:

JavaScript 

```
const nuevoProducto = { nombre: "Miel de Abeja", precio: 150 };

fetch('https://api.ecomarket.com/productos', {
```

## Tu prueba de fuego en el código

Has demostrado que entiendes la teoría detrás de los headers, los métodos y los estados.

Ahora, bajémoslo a la implementación real que harás en **EcoMarket**.

Imagina que estás usando `fetch` para enviar un nuevo producto desde un formulario:

JavaScript

```
const nuevoProducto = { nombre: "Miel de Abeja", precio: 150 };

fetch('https://api.ecomarket.com/productos', {
  method: 'POST', // El método que elegimos
  headers: {
    'Content-Type': 'application/json' // Lo que ya descubrimos
  },
  body: JSON.stringify(nuevoProducto)
})
.then(response => {
  // AQUÍ ES DONDE APLICAS LO APRENDIDO
  if (response.status === 201) {
    console.log("¡Producto creado!");
  } else if (response.status === 401) {
    console.log("Sesión expirada, ve a login.");
  }
});
```

Para cerrar este ciclo de tutoría:

¿Te gustaría que ahora practiquemos cómo depurar esto en la pestaña "Network" de tu navegador para que veas estos headers y códigos "en vivo" con una API real? Es la herramienta número uno de un desarrollador frontend.

