



UNIVERSIDAD AUTONOMA DE NAYARIT



## Unidad Académica de Economía

### Carrera

*Licenciatura en Sistemas Computacionales*

### Grupo y Semestre

*Semestre 6*

### Nombre estudiante

*Ricardo Matos Vizcarra*

### Actividad

*Reto IA 2 – Conversación*

### Maestro

*Eligardo Cruz Sánchez*

### Materia

*Programación Distribuida del lado Cliente*

UNIVERSIDAD AUTÓNOMA DE NAYARIT

Actúa como un analista experto en respuestas HTTP. Tu tarea es ayudarme a entender qué código de estado usar en diferentes escenarios. Escenario base: API de EcoMarket para gestión de productos orgánicos. Para cada uno de estos casos, explícame:...

◆ Mostrar razonamiento ▼

¡Hola! Como analista de APIs para **EcoMarket**, me encanta este ejercicio. Elegir el código de estado (status code) correcto no es solo una cuestión técnica; es el lenguaje con el que el servidor le "habla" al frontend para que la experiencia de usuario sea fluida.

Aquí tienes el desglose técnico para tus escenarios:

#### 1. El usuario intenta crear un producto pero el JSON está mal formado

**CÓDIGO:** `400 Bad Request` **RAZÓN:** Se utiliza cuando el servidor no puede procesar la solicitud debido a algo que es percibido como un error del cliente (syntax de solicitud mal formada). No usamos `422 Unprocessable Entity` aquí porque el 422 se reserva para cuando el JSON es sintácticamente correcto pero lógicamente inválido (ej. un precio negativo). Si el JSON tiene una coma de más o le falta una llave, es un **400**. **ACCIÓN CLIENTE:** El frontend debe revisar la estructura de la petición. No tiene sentido reintentar la misma petición sin cambios; el desarrollador debe corregir el código que genera el JSON.

#### 2. El usuario intenta ver un producto que fue eliminado ayer

**CÓDIGO:** `410 Gone` **RAZÓN:** Aunque el famoso `404 Not Found` funcionaría, el **410** es

## 2. El usuario intenta ver un producto que fue eliminado ayer

**CÓDIGO:** `410 Gone` **RAZÓN:** Aunque el famoso `404 Not Found` funcionaría, el **410** es mucho más preciso para un analista experto. Indica que el recurso existía pero ya no está disponible y, lo más importante, que esta condición es **permanente**. Esto ayuda a que los motores de búsqueda dejen de indexar esa URL de producto de EcoMarket. **ACCIÓN CLIENTE:** El frontend debe eliminar el producto de sus cachés locales, mostrar un mensaje informativo ("Este producto ya no forma parte de nuestro catálogo") y redirigir al usuario a la lista general de productos orgánicos.

---

## 3. El usuario intenta actualizar el precio de un producto pero su sesión expiró

**CÓDIGO:** `401 Unauthorized` **RAZÓN:** A pesar de su nombre, semánticamente significa "Unauthenticated" (no autenticado). El servidor le dice al cliente: "No sé quién eres, así que no puedo dejarte cambiar precios". No usamos `403 Forbidden` porque el 403 implica que el servidor sí sabe quién eres, pero que específicamente tú no tienes permiso para esa acción. **ACCIÓN CLIENTE:** El frontend debe capturar este error, guardar el estado actual del formulario (para no perder el nuevo precio que el usuario escribió) y redirigir inmediatamente a la pantalla de **Login**.

---

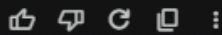
## 4. El servidor de base de datos está sobrecargado temporalmente

**CÓDIGO:** `503 Service Unavailable` **RAZÓN:** Indica que el servidor no está listo para manejar la solicitud. Es mejor que un `500 Internal Server Error` porque el 503 comunica que es una **condición temporal**. A menudo incluye un encabezado `Retry-After` para decirle al cliente cuánto esperar. **ACCIÓN CLIENTE:** El frontend debe mostrar una pantalla de "Estamos experimentando mucho tráfico, reintentando en unos segundos..." y programar un reintento automático (idealmente con un algoritmo de *exponential backoff*) en lugar de dejar al usuario abandonado.

#### 5. El usuario intenta crear un producto con un SKU que ya existe

**CÓDIGO:** `409 Conflict` **RAZÓN:** Este es el código ideal para violaciones de integridad o conflictos con el estado actual del servidor. Un SKU debe ser único en EcoMarket; intentar crear un duplicado crea un **conflicto de recursos**. Algunos usan `422`, pero el `409` es más específico para errores de duplicidad en bases de datos. **ACCIÓN CLIENTE:** El frontend debe resaltar el campo del SKU en rojo y mostrar un mensaje de error específico: "Este SKU ya está registrado para otro producto. Por favor, verifica el código".

¿Te gustaría que diseñemos juntos el cuerpo del mensaje (JSON de error) que debería acompañar a estos códigos para que tu frontend sea aún más robusto?



Te voy a pasar mis predicciones de los 5 casos, en los cuales quiero realizar una tabla sobre los casos en los que me equivoqué para usar como referencia futura. Estos fueron mis predicciones: 1.- El usuario intenta crear...

◆ **Mostrar razonamiento** ▼

¡Oye, para no ser (aún) un analista de protocolos, tu intuición lógica es de **10/10**! Tienes muy claro el *qué* debe pasar, lo cual es el 80% del trabajo. Mi labor ahora es ponerle la "etiqueta" técnica (el número) a esa lógica para que el navegador y el servidor se entiendan sin ambigüedades.


Aquí tienes la tabla comparativa para tu referencia futura. He marcado en qué punto tu lógica fue acertada y dónde el código HTTP añade una capa extra de precisión.

**Tabla de Referencia: Intuición vs. Estándar Técnico**

| Caso                         | Tu Predicción (Lógica)               | Código Técnico         | ¿Dónde estuvo el ajuste?   |
|------------------------------|--------------------------------------|------------------------|--|
| <b>1. JSON Malformado</b>    | Validar errores y avisar al usuario. | <b>400 Bad Request</b> | Tu lógica es correcta. El ajuste es que el <b>400</b> detiene la petición antes de intentar validar datos; es un error de "idioma" (sintaxis). |
| <b>2. Producto Eliminado</b> | Mostrar que no se encuentra.         | <b>410 Gone</b>        | Acertaste en que "no está". El ajuste es usar <b>410</b> en vez de <b>404</b> para decirle al cliente: "No me busques más, esto no volverá".   |

### Tabla de Referencia: Intuición vs. Estándar Técnico

| Caso                         | Tu Predicción (Lógica)                    | Código Técnico           | ¿Dónde estuvo el ajuste?   |
|------------------------------|---|--------------------------|--|
| <b>1. JSON Malformado</b>    | Validar errores y avisar al usuario.      | <b>400 Bad Request</b>   | Tu lógica es correcta. El ajuste es que el <b>400</b> detiene la petición antes de intentar validar datos; es un error de "idioma" (sintaxis). |
| <b>2. Producto Eliminado</b> | Mostrar que no se encuentra.              | <b>410 Gone</b>          | Acertaste en que "no está". El ajuste es usar <b>410</b> en vez de <b>404</b> para decirle al cliente: "No me busques más, esto no volverá".   |
| <b>3. Sesión Expirada</b>    | Re-iniciar sesión y validar identidad.    | <b>401 Unauthorized</b>  | ¡Exacto! Tu flujo de redirigir al login es la implementación perfecta del código <b>401</b> .  |
| <b>4. Sobrecarga DB</b>      | Avisar que intente más tarde.             | <b>503 Service Unav.</b> | Diste en el clavo. El <b>503</b> es precisamente el código que permite enviar un tiempo de espera ( <code>Retry-After</code> ).                |
| <b>5. SKU Duplicado</b>      | Avisar que ya existe y evitar duplicados. | <b>409 Conflict</b>      | ¡Perfecto! Tu enfoque en la "integridad de datos" es exactamente lo que define a un conflicto <b>409</b> .                                     |

 Exportar a Hojas de cálculo



### Análisis de tus aciertos

- **En los casos 3, 4 y 5:** Tu lógica fue impecable. Entiendes que el error no es "culpa" del programador, sino un estado del sistema (sesión, carga o datos existentes) que el usuario debe conocer.
- **En el caso 1:** Identificaste bien la necesidad de validación. Como analista, solo recuerda que si el JSON está "roto" (falta una llave `{ }`), el servidor ni siquiera puede empezar a

### Análisis de tus aciertos

- **En los casos 3, 4 y 5:** Tu lógica fue impecable. Entiendes que el error no es "culpa" del programador, sino un estado del sistema (sesión, carga o datos existentes) que el usuario debe conocer.
- **En el caso 1:** Identificaste bien la necesidad de validación. Como analista, solo recuerda que si el JSON está "roto" (falta una llave `{`), el servidor ni siquiera puede empezar a validar los campos, por eso lanzamos el **400** de inmediato.
- **En el caso 2:** Tu respuesta fue la estándar ( `404` ). El paso a experto es usar el **410**, que es como un "404 con memoria": sabemos que estuvo ahí, pero ya se fue para siempre.

### Un consejo de "pro":

Cuando documentes estos errores para el equipo de Frontend, siempre recomienda enviar un **cuerpo de respuesta** junto al código, algo como:

JSON



```
{  
  "error": "conflict_sku",  
  "message": "El SKU 'ORG-123' ya pertenece a otro producto."  
}
```

¿Te gustaría que te ayude a redactar los mensajes de error (JSON) específicos para cada uno de estos casos y así completar tu guía de referencia?

