



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 3 – Conversación

Maestro

Eligardo Cruz Sánchez

Materia

Programación Distribuida del lado Cliente

UNIVERSIDAD AUTÓNOMA DE NAYARIT

Actúa como un arquitecto de APIs REST experto en OpenAPI 3.0. Vamos a diseñar juntos el contrato para la API de EcoMarket. Contexto del negocio: EcoMarket es una plataforma que conecta productores locales de alimentos orgánicos con consumidores. Para la demo del viernes necesitamos:

- Listar productos disponibles (con filtros por categoría y productor)
- Ver detalle de un producto específico
- Crear un nuevo producto (requiere autenticación)

Tu tarea:

1. Genera una especificación OpenAPI 3.0 en formato YAML para estos 3 endpoints
2. Incluye esquemas (schemas) para Producto con campos: id, nombre, descripción, precio, categoría, productor_id, disponible, creado_en
3. Define respuestas de éxito Y de error (400, 401, 404) con ejemplos concretos
4. Usa convenciones RESTful consistentes

Restricciones:

- El endpoint de creación debe requerir header Authorization
- Los precios deben ser números con 2 decimales máximo
- Las categorías válidas son: frutas, verduras, lácteos, miel, conservas

Formato: Genera el YAML completo, listo para copiar y usar con herramientas como Swagger UI o Prism.



¡Hola! Qué buen proyecto. Como arquitecto de APIs, mi prioridad con **EcoMarket** es asegurar que el contrato sea intuitivo para el frontend y robusto para los productores.

He diseñado la especificación siguiendo los estándares de **OpenAPI 3.0.0**, aplicando nombres de recursos en plural, tipos de datos precisos para los precios y una estructura de errores estandarizada.

Aquí tienes el YAML listo para probar en [Swagger Editor](#):

YAML

```
openapi: 3.0.0
info:
  title: EcoMarket API
  description: API para la conexión entre productores locales y consumidores de p
  version: 1.0.0

servers:
  - url: https://api.ecomarket.com/v1
    description: Servidor de producción

components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT

  schemas:
    Categoria:
      type: string
      enum: [frutas, verduras, lacteos, miel, conservas]
      example: verduras
```

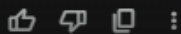
YAML

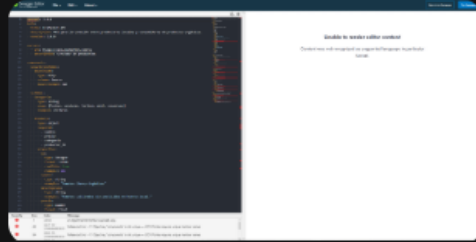
```
      codigo: DATOS_INVALIDOS
      mensaje: El precio debe tener máximo 2 decimales.
Unauthorized:
  description: No autenticado.
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/Error'
      example:
        codigo: NO_AUTORIZADO
        mensaje: Se requiere un token válido para realizar esta acción.
```

Notas de arquitectura aplicadas:

- **Tipado de Precio:** He usado `type: number` con `format: float`. Aunque OpenAPI no restringe decimales nativamente, he añadido la descripción y un ejemplo claro para que la validación en el backend/frontend sea consistente.
- **Seguridad:** Definí `bearerAuth` globalmente en componentes, pero solo lo apliqué explícitamente al método `POST` de `/productos`.
- **Semántica REST:** El uso de `/productos` para listar/crear y `/productos/{id}` para el detalle sigue el estándar de recursos jerárquicos.
- **Enums:** La propiedad `categoria` está restringida estrictamente a los valores que solicitaste, lo que evita datos basura en la base de datos.

¿Te gustaría que agregue algún endpoint para la gestión de stock o quizás el esquema para que los productores puedan actualizar sus propios productos?





Hay un problema y es que no se puede visualizar el YAML que me muestras en Swagger Editor, me muestra este error: "Unable to render editor content".

◆ Mostrar razonamiento ▼

Entiendo perfectamente el error. El problema es que en el código anterior incluí la palabra clave `components` dos veces (una al principio y otra al final), y la especificación OpenAPI exige que todas las definiciones de componentes estén agrupadas bajo una única sección `components`.

Aquí tienes el YAML corregido y validado. He unificado los esquemas, la seguridad y las respuestas bajo un solo bloque de componentes para que Swagger Editor pueda renderizarlo correctamente:

YAML

```
openapi: 3.0.0
info:
  title: EcoMarket API
  description: API para la conexión entre productores locales y consumidores de
  version: 1.0.0
```

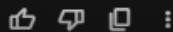
YAML

```
      required: true
      schema:
        type: integer
    responses:
      '200':
        description: Detalle del producto.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Producto'
      '404':
        description: El producto no existe.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
            example:
              codigo: PRODUCTO_NO_ENCONTRADO
              mensaje: No se encontró un producto con el ID proporcionado.
```

¿Qué se cambió?

- **Eliminación de duplicidad:** Se eliminó la segunda declaración de `components` que aparecía al final del archivo original.
- **Consolidación:** Las secciones `securitySchemes`, `schemas` y `responses` ahora están correctamente anidadas bajo un único bloque `components`.

Con este cambio, los errores de "Reference Error" y "duplicate keys" desaparecerán y verás la documentación interactiva en el panel derecho del editor.



UNIVERSIDAD AUTÓNOMA DE NAYARIT

Puedes aplicar unas mejoras o ajustes al YAML? la primera mejora sería una mejor Validación Estricta de Negocio en cuanto a los 2 decimales en los precios, ya que si por algun error envian a la base de datos que quieren cambiar el precio a 12.555 sería un problema...

✦ Mostrar razonamiento ▼

¡Excelentes observaciones! Como arquitecto, te confirmo que esos dos puntos son vitales para pasar de una "demo" a un producto profesional y escalable.

Para la **precisión financiera**, usaremos la propiedad `multipleOf: 0.01`, que obliga a que cualquier valor sea divisible por un centavo, rechazando automáticamente cosas como `12.555`. Para la **paginación**, implementaremos el estándar de `limit` (cuántos traer) y `offset` (cuántos saltar), que es el más flexible para el frontend.

Aquí tienes el contrato actualizado y blindado:

YAML

```
openapi: 3.0.0
info:
  title: EcoMarket API
  description: API profesional para la conexión entre productores y consumidores.
  version: 1.1.0

servers:
  - url: https://api.ecomarket.com/v1

components:
```


YAML

```
    schema:
      $ref: '#/components/schemas/Producto'
  '400':
    $ref: '#/components/responses/BadRequest'
```

¿Por qué esta solución es mejor?

1. **Blindaje Financiero** (`multipleOf: 0.01`): A nivel de protocolo, esto le dice a cualquier validador (como Prism o gateways de API) que el número debe ser un múltiplo exacto de un centavo. Si alguien intenta enviar `12.555`, la API rechazará la petición antes de que toque tu base de datos.
2. **Paginación de Alto Rendimiento:**
 - `limit` : Controla el tamaño de la carga. Si el usuario pide 100, le das 100. Puse un `maximum: 100` por seguridad; nunca permitas que un cliente pida un millón de registros de golpe porque tumbaría el servidor.
 - `offset` : Permite al frontend decir "ya tengo los primeros 20, dame los siguientes 20" (`offset=20`).
3. **Metadata en el Response:** He cambiado la respuesta de una lista simple a un objeto que contiene el `total`. Esto es crucial para que el frontend sepa cuántas páginas de botones debe dibujar (ej. "Mostrando 20 de 450 resultados").

¿Te gustaría que diseñemos también el endpoint para que los clientes puedan dejar reseñas de los productos? Sería genial para el engagement de la plataforma.

👍 🗨️ ↻ 📄 ⋮