# Documentación Técnica – Prototipo de Vista de Detalle de Producto estilo Mercado Libre

**Autor:** Diego Chinchilla  
**Tecnologías usadas:**

* Backend: **FastAPI (Python)**
* Frontend: **HTML, CSS, JavaScript**
* Persistencia: **Archivo local JSON**

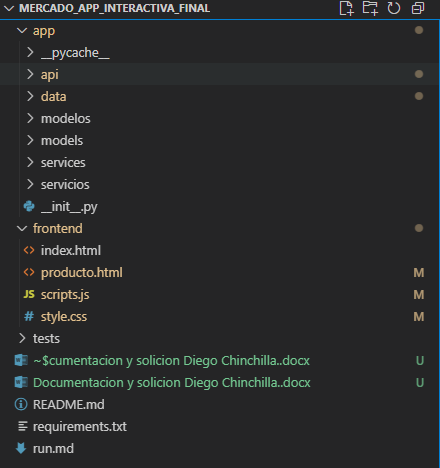
## Opciones de diseño adoptadas

### Arquitectura basada en capas y orientación a objetos

Se optó por separar responsabilidades claramente:

* Item: clase que representa cada producto con sus atributos.
* ItemService: clase encargada de cargar productos desde un archivo local (items.json) y facilitar su búsqueda.
* main.py: punto de entrada con rutas limpias y RESTful.

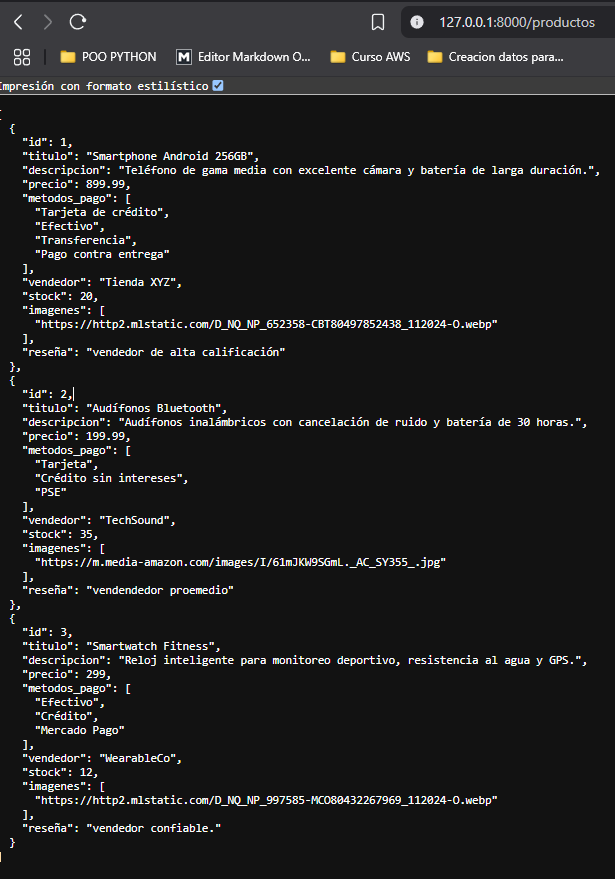
Este diseño facilita el mantenimiento, la legibilidad del código y una posible extensión futura (ej. migrar a una BD real).



### API RESTful modular

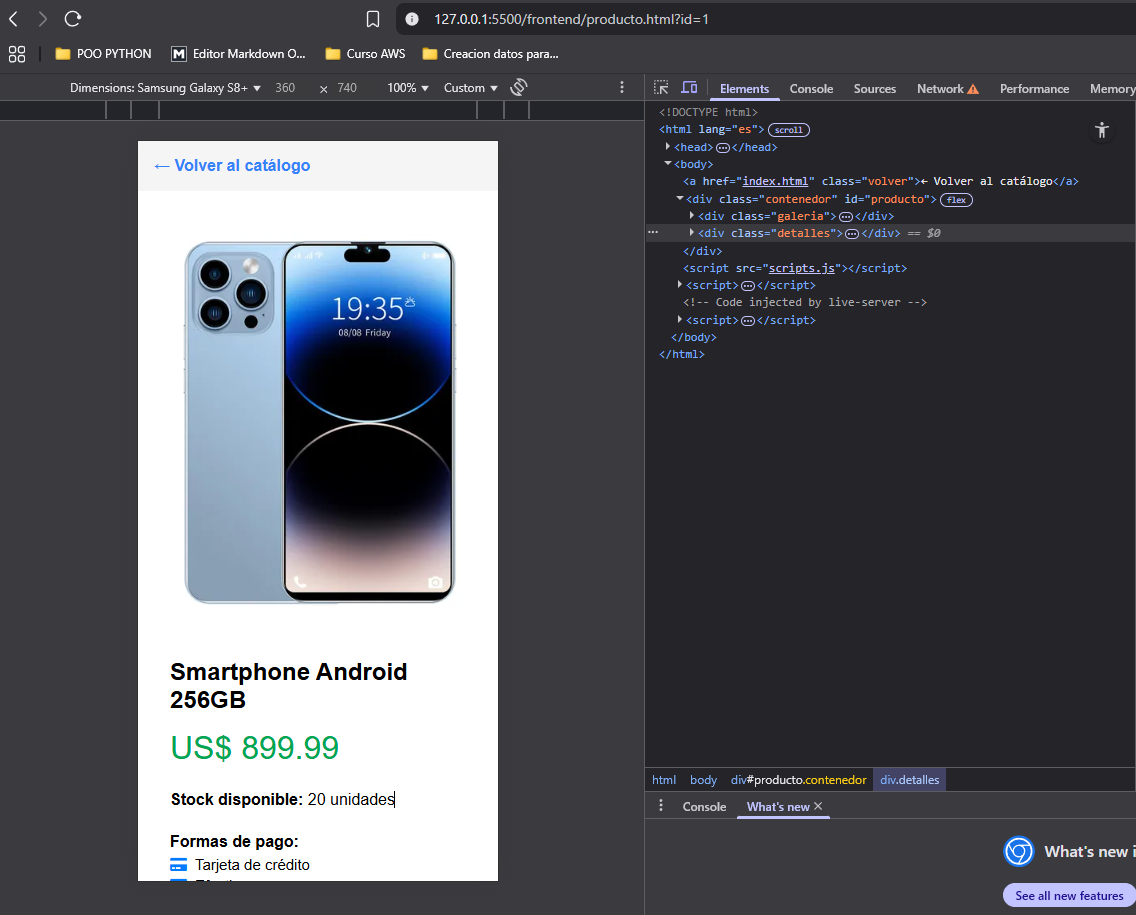
* El endpoint principal /productos/{id} permite consultar los detalles completos de un producto por su ID.
* El endpoint /productos lista todos los productos disponibles en el catálogo.

Ambos endpoints responden en formato JSON, permitiendo su consumo simple desde el frontend.



## **Frontend desacoplado, responsivo y directo**

* La interfaz de usuario fue construida desde cero usando HTML, CSS y JavaScript puro, con estructura clara y sin dependencias externas.
* Se organizó un index.html para el catálogo y un producto.html para el detalle del producto.
* Se implementó una galería vertical de imágenes, detalles de producto y un botón interactivo de "me gusta", imitando la experiencia de usuario de MercadoLibre.
* Se cargan dinámicamente los datos desde la API usando fetch.



# Diseños encontrados y como los resolví

## Sincronización entre datos JSON y estructura de clase

Inicialmente, el campo reseña no se estaba mapeando correctamente en el método desde\_diccionario(), generando errores. Solución: corregí la asignación para que el campo reseña apunte correctamente al dato esperado (data["reseña"] en lugar de data["imagenes"]). Error que se generó por duplicidad de código.

## Estilos y diseño dinámico con JavaScript

Uno de los retos fue lograr un diseño moderno sin frameworks como React o Bootstrap. Solución: estructuré cuidadosamente el HTML y utilicé template literals en JS para generar los elementos dinámicamente, aplicando lógica condicional (por ejemplo, para el botón de me gusta).

## Interacción dinámica y carga de datos asincrónica

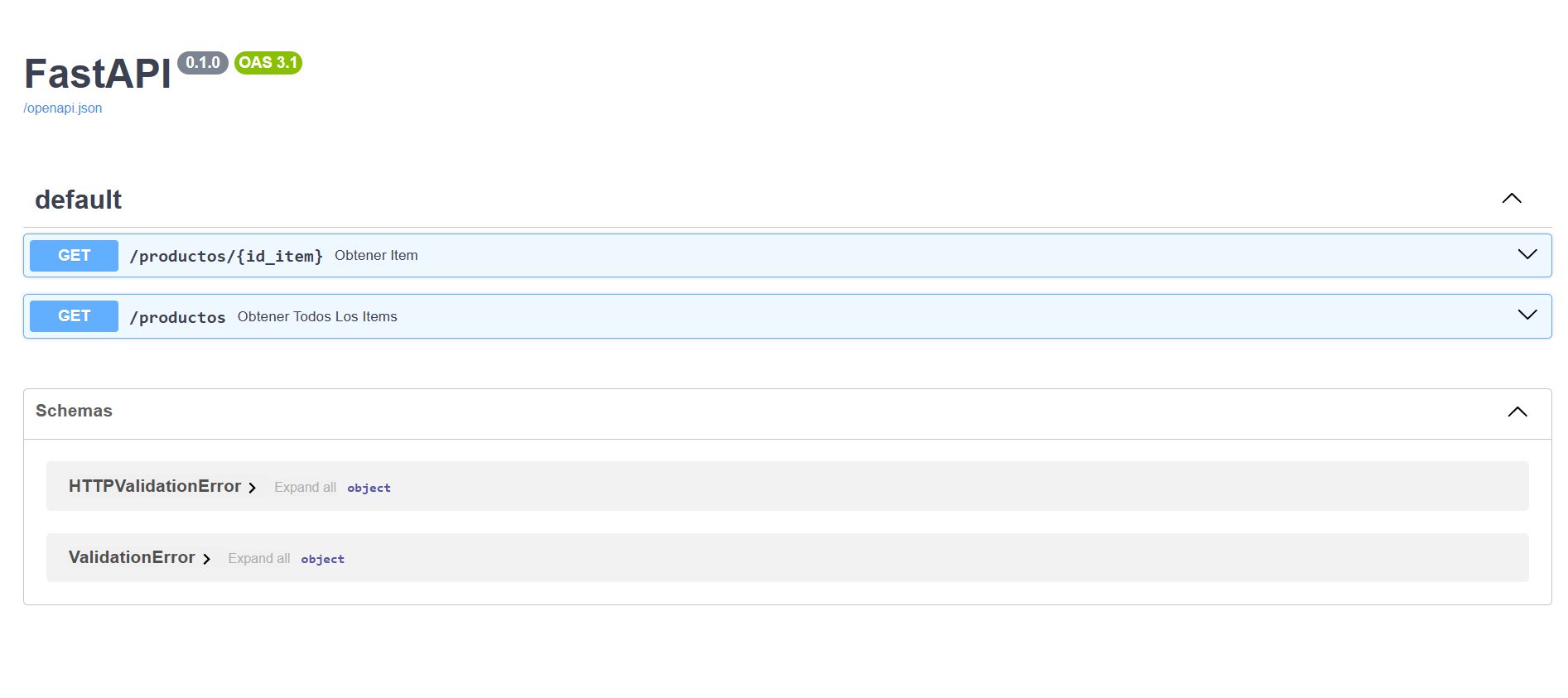
El botón "me gusta" inicialmente no respondía porque el evento no se estaba adjuntando después del innerHTML. Solución: reubiqué el addEventListener justo después de inyectar el HTML, asegurándome de que el DOM estuviera disponible.

## Visualización en lista de métodos de pago

Inicialmente el método de pago aparecía como texto plano. Solución: transformé metodos\_pago en una lista <ul>, generada con map() para mayor semántica y estilo visual.

# Conclusiones.

El proyecto fue abordado con enfoque modular, buenas prácticas de desarrollo web y una estructura clara tanto en frontend como en backend. Se cumplió con cada uno de los requisitos funcionales solicitados, manteniendo un diseño limpio, interactivo y fácil de extender. Y generando mas enfoque en la creación del backend para el consumo de los datos.



## Test de ejecución Productos

