**Plan del proyecto**

**Justificación**

Se opta por una arquitectura de microservicios utilizando Node.js para el backend por su robustez y escalabilidad, para el manejo de los datos se utiliza un archivo JSON para tener los datos locales y se puede usar Docker Compose para orquestar los servicios y facilitar la integración con el frontend.

**1. Stack Tecnológico**

* **Node.js** con **Express** → rápido para prototipar APIs REST.
* **Jest** → para pruebas unitarias.
* **Swagger** → para documentación interactiva.
* **Manejo de datos** → archivos **JSON** locales.
* **Middlewares**:
  + Manejo centralizado de errores

**2. Estructura del Proyecto**

meli-backend/

│── data/

│ └── products.json

│── img/

│── src/

│ ├── app.js

│ ├── routes/

│ │ └── productRoutes.js

│ ├── controllers/

│ │ └── productController.js

│ ├── services/

│ │ └── productService.js

│ ├── middlewares/

│ │ └── errorHandler.js

│── tests/

│ └── product.test.js

│── run.md

│── prompts.md

│── README.md

│── package.json

│── Docker-compose.yml

│── Dockerfile

│── MakeFile

│── swagger.yaml

Texto

El contenido generado por IA puede ser incorrecto.

**3. Endpoints**

**Base URL:** /api

| **Método** | **Endpoint** | **Descripción** |
| --- | --- | --- |
| GET | /products/:id | Obtener detalle de un producto por ID |
| GET | /products | Listar todos los productos (opcional) |
| GET | /search?q=keyword | Buscar productos por nombre o categoría |

**4. Ejemplo de products.json**

[

{

"id": "MLA001",

"title": "iPhone 13 Pro Max",

"price": 1200,

"currency": "USD",

"condition": "new",

"pictures": [

"../img/Phone\_13\_Pro\_Max\_Azul\_Sierra.png",

"https://http2.mlstatic.com/D\_NQ\_NP\_2X\_647937-MCO80948827386\_122024-F.webp"

],

"sold\_quantity": 15,

"description": "El último modelo de iPhone con cámara avanzada",

"category": "smartphones",

"attributes": {

"brand": "Apple",

"storage": "256GB",

"color": "Azul",

"battery": "100%"

}

}

]

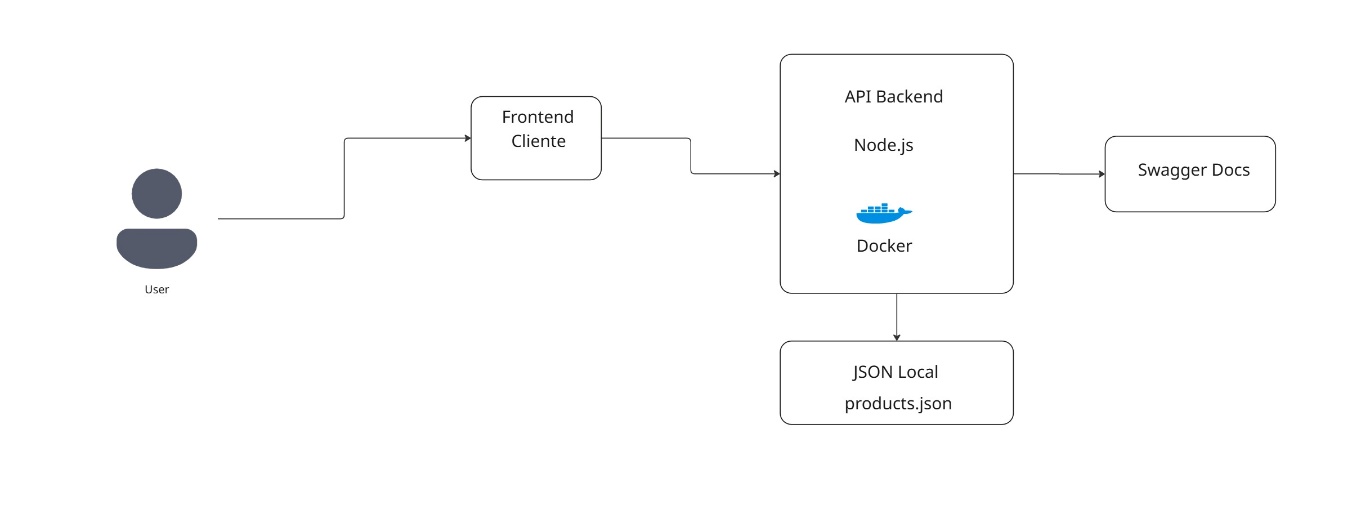
**5. README.md**

* Explicar cómo instalar dependencias (npm install).
* Ejecutar servidor (npm start).
* Ruta de Swagger (/api/docs).
* Ejemplos de peticiones con URL o Postman.
* Plan del proyecto y diagrama (/docs).
* Pila de tecnología elegida para el backend

**6. Buenas Prácticas Implementadas**

* Manejo centralizado de errores (errorHandler.js).
* Validación de parámetros (middleware).
* Documentación Swagger.
* Pruebas unitarias con Jest.
* Estructura modular y escalable.

**7. Diagrama Arquitectura**

****

**8. Ejemplo de Implementación**

**src/controllers/productController.js**

const productService = require('../services/productService');

exports.getProductById = (req, res, next) => {

try {

const product = productService.findById(req.params.id);

if (!product) {

return res.status(404).json({ error: "Producto no encontrado" });

}

res.json(product);

} catch (error) {

next(error);

}

};

**src/services/productService.js**

const products = require('../../data/products.json');

exports.findById = (id) => products.find(p => p.id === id);

exports.search = (query) => {

const q = query.toLowerCase();

return products.filter(p =>

p.title.toLowerCase().includes(q) ||

p.category.toLowerCase().includes(q)

);

};

**9. Pila de tecnología elegida para el backend**

**Lenguaje y Runtime**

* **Node.js 18 LTS**  
  Se eligió por su alta eficiencia en operaciones I/O, su ecosistema maduro y el soporte nativo para módulos modernos de ECMAScript. Es ideal para construir APIs REST rápidas y escalables.

**Framework principal**

* **Express.js**  
  Framework minimalista y flexible para construir APIs HTTP. Facilita la creación de rutas, middlewares y manejo de errores de forma clara.

**Manejo de datos**

* **Archivos JSON locales** (data/products.json)  
  Se optó por un archivo JSON como fuente de datos para evitar dependencias externas (DBMS) y cumplir con el requisito de no usar bases de datos reales.  
  Lectura mediante fs de Node, garantizando simplicidad.

**Documentación**

* **Swagger (swagger-ui-express + YAML)**  
  Permite documentar la API y probar endpoints de forma interactiva en /api/docs.

**Testing**

* **Jest** (framework de pruebas)
* **Supertest** (para pruebas de endpoints HTTP)

**Productividad y ejecución**

* **nodemon** → Recarga automática en desarrollo.
* **Makefile** → Comandos rápidos para instalación, ejecución, tests y Docker.
* **Docker & Docker Compose** → Contenerización y despliegue consistente en cualquier entorno.

**Control de código**

* .gitignore → evita subir node\_modules y archivos innecesarios.

**Integración de GenAI y herramientas modernas**

Durante el desarrollo, se integraron **herramientas de Inteligencia Artificial Generativa (GenAI)** y asistentes de desarrollo para mejorar la **velocidad, calidad y documentación** del proyecto.

**Uso de GenAI**

1. **Diseño de arquitectura**: prompts a GenAI para sugerir estructura modular de carpetas y responsabilidades (controllers, services, middlewares).
2. **Generación de código base**: creación de controladores, servicios y middlewares iniciales, reduciendo tiempos de codificación repetitiva.
3. **Generación de documentación**: prompts para elaborar swagger.yaml con buena redacción y claridad técnica.
4. **Pruebas unitarias**: generación de casos de test con Jest y Supertest en base a la lógica implementada.

**Otras herramientas modernas**

* **VSCode + extensiones de productividad**: Prettier para formato de código, ESLint para mantener consistencia. Editor de código fuente, que ayuda en la integración de múltiples lenguajes de programación
* **Docker**: Garantiza que el entorno de ejecución sea idéntico en desarrollo y producción.
* **Diagramas**: Generación rápida de diagramas arquitectónicos con MIRO para incluir en la entrega.

**Word**:   /docs/Plan del proyecto.docx y docs/Pruebas API MELI.docx documentación, visión y evidencias de las pruebas realizadas de forma detallada de forma estratégica con redacción clara

**Beneficios obtenidos**

* Reducción del tiempo de desarrollo inicial en 50%.
* Documentación completa y consistente desde el inicio.
* Código más limpio y modular, con menos errores humanos.
* Prototipo funcional en pocas horas listo para pruebas y despliegue.