Tecsup
TECNOLOGÍA
CON SENTIDO

DESARROLLO DE APLICACIONES WEB AVANZADO

LABORATORIO N° 5

IMPLEMENTA UNA APLICACIÓN USANDO API RESTFUL

Alumnos	Nota

Grupo	
Fecha de Entrega	
Docente	**Renato Usnayo Cáceres**

DISEÑO Y DESARROLLO DE SOFTWARE
PROGRAMA DE FORMACIÓN REGULAR

OBJETIVOS:

- Diseña y desarrolla APIs RESTful

SEGURIDAD:

	**Advertencia:** **En este laboratorio está prohibida la manipulación del hardware, conexiones eléctricas o de red; así como la ingestión de alimentos o bebidas.**

FUNDAMENTO TEÓRICO:

- Revisar el texto guía que está en el campus Virtual.

NORMAS EMPLEADAS:

- No aplica

RECURSOS:

- En este laboratorio cada alumno trabajará con un equipo con Windows 11.

METODOLOGÍA PARA EL DESARROLLO DE LA TAREA:

- El desarrollo del laboratorio es individual

PROCEDIMIENTO:

Nota:

Las secciones en azul y cursiva brindan una explicación teórica o del código

Procedimiento:

Configuración del Proyecto

Crear la carpeta del proyecto y configurarlo

```
mkdir api-restful
cd api-restful
npm init -y
```

Esto generará un archivo package.json con la configuración del proyecto.

Instalar las dependencias necesarias

```
npm install express cors nodemon
```

express: Framework para crear la API.

cors: Middleware para permitir solicitudes desde otros dominios.

nodemon: Recarga automática del servidor en desarrollo.

Configurar el script en package.json

Abre package.json y agrega en "scripts":

```
"start": "node index.js",
"dev": "nodemon index.js"
```

npm run dev iniciará el servidor con recarga automática.

Crear el Servidor con Express

Crear el archivo index.js y configurar Express

```
const express = require("express");
const cors = require("cors");

const app = express();
const PORT = 3000;

// Middleware
app.use(express.json()); // Para leer JSON en las solicitudes
app.use(cors()); // Permitir solicitudes de otros dominios

// Mensaje de prueba en la raíz
app.get("/", (req, res) => {
  res.send("¡Bienvenido a la API RESTful!");
});

// Iniciar el servidor
app.listen(PORT, () => {
  console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```

express.json() permite recibir datos en formato JSON.

cors() habilita el acceso a la API desde otros servidores.

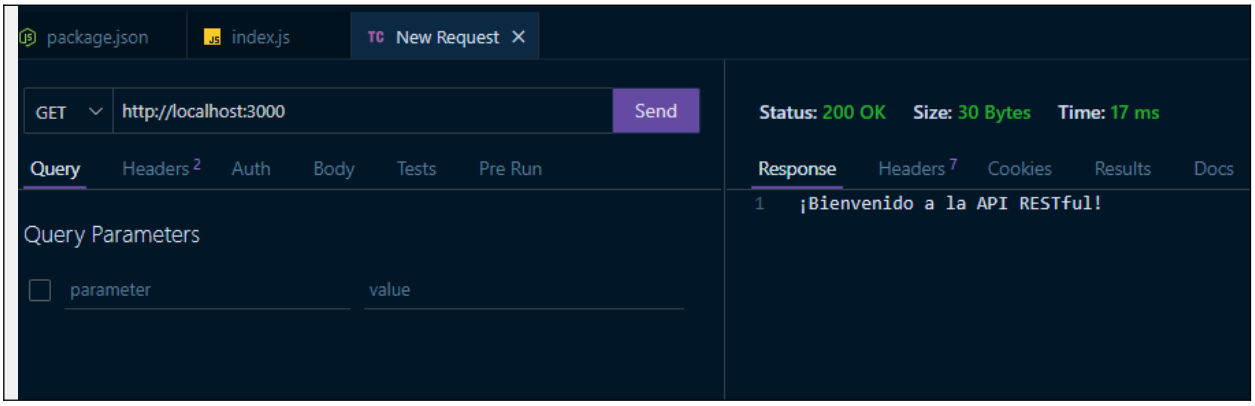
app.get("/") devuelve un mensaje de bienvenida.

Iniciar el servidor

```
npm run dev
```

Probar en el navegador o con Postman

Abre en el navegador: http://localhost:3000



Crear Datos Simulados

Crear una carpeta data/ y el archivo data.json

```
[
  { "id": 1, "nombre": "Producto A", "precio": 10.99 },
  { "id": 2, "nombre": "Producto B", "precio": 20.49 },
  { "id": 3, "nombre": "Producto C", "precio": 30.99 }
]
```

Simulamos una base de datos con un array de productos en JSON.

Implementar Endpoints RESTful

Crear un archivo routes/products.js

```
const express = require("express");
const fs = require("fs");
const path = require("path");

const router = express.Router();
const dataPath = path.join(__dirname, "../data/data.json");

// Obtener todos los productos (GET)
```

```
router.get("/", (req, res) => {
  const productos = JSON.parse(fs.readFileSync(dataPath, "utf8"));
  res.json(productos);
});

// Obtener un producto por ID (GET)
router.get("/:id", (req, res) => {
  const productos = JSON.parse(fs.readFileSync(dataPath, "utf8"));
  const producto = productos.find(p => p.id === parseInt(req.params.id));
  producto ? res.json(producto) : res.status(404).json({ mensaje: "Producto no
encontrado" });
});

// Crear un nuevo producto (POST)
router.post("/", (req, res) => {
  const productos = JSON.parse(fs.readFileSync(dataPath, "utf8"));
  const nuevoProducto = { id: productos.length + 1, ...req.body };
  productos.push(nuevoProducto);
  fs.writeFileSync(dataPath, JSON.stringify(productos, null, 2));
  res.status(201).json(nuevoProducto);
});

// Actualizar un producto (PUT)
router.put("/:id", (req, res) => {
  const productos = JSON.parse(fs.readFileSync(dataPath, "utf8"));
  const index = productos.findIndex(p => p.id === parseInt(req.params.id));

  if (index !== -1) {
    productos[index] = { ...productos[index], ...req.body };
    fs.writeFileSync(dataPath, JSON.stringify(productos, null, 2));
    res.json(productos[index]);
  } else {
    res.status(404).json({ mensaje: "Producto no encontrado" });
  }
});

// Eliminar un producto (DELETE)
router.delete("/:id", (req, res) => {
  let productos = JSON.parse(fs.readFileSync(dataPath, "utf8"));
  const productosFiltrados = productos.filter(p => p.id !== parseInt(req.params.id));

  if (productos.length !== productosFiltrados.length) {
    fs.writeFileSync(dataPath, JSON.stringify(productosFiltrados, null, 2));
    res.json({ mensaje: "Producto eliminado" });
  } else {
```

```
    res.status(404).json({ mensaje: "Producto no encontrado" });
  }
});

module.exports = router;
```

GET /productos → Devuelve todos los productos.

GET /productos/:id → Devuelve un producto por ID.

POST /productos → Crea un nuevo producto.

PUT /productos/:id → Actualiza un producto.

DELETE /productos/:id → Elimina un producto.

Integrar las rutas en index.js

```
const productRoutes = require("./routes/products");
app.use("/productos", productRoutes);
```

Ahora, todas las rutas de productos están disponibles en /productos.

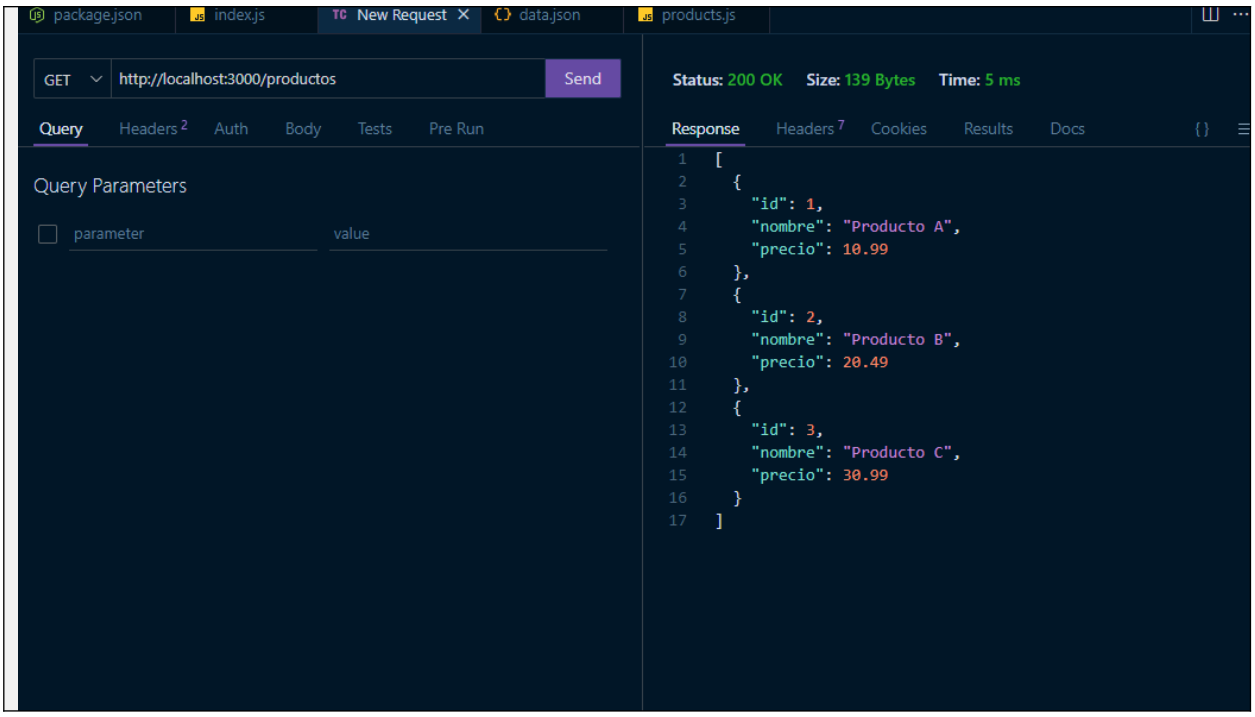
Probar la API RESTful

Usar Postman o cURL para probar la API

Obtener todos los productos:

```
curl -X GET http://localhost:3000/productos
```

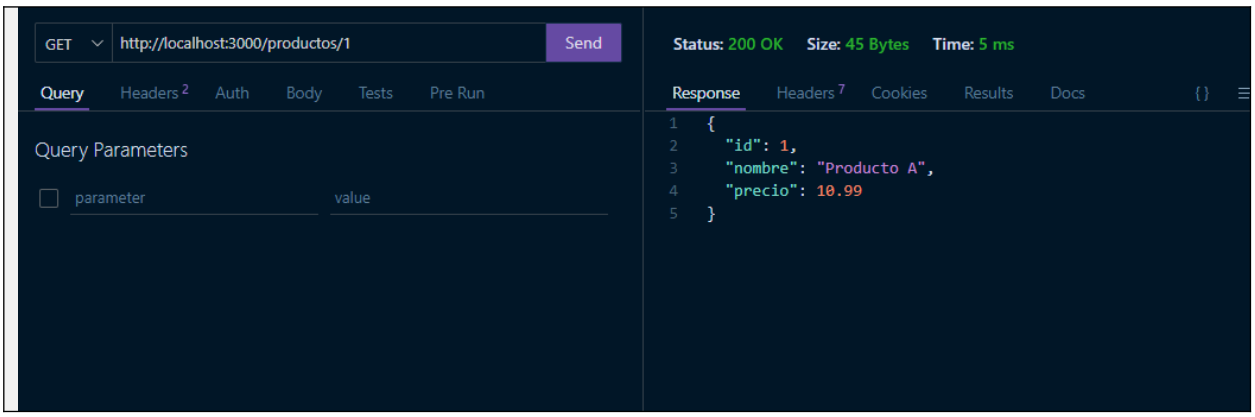
Captura del resultado



Obtener un producto por ID:

```
curl -X GET http://localhost:3000/productos/1
```

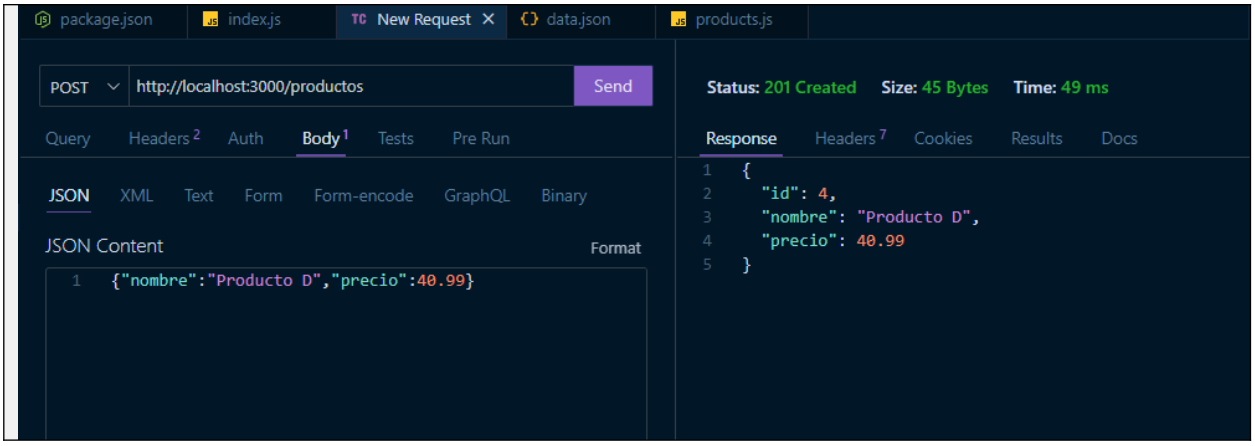
Captura del resultado



Crear un producto:

```
curl -X POST http://localhost:3000/productos -H "Content-Type: application/json" -d '{"nombre":"Producto D","precio":40.99}'
```

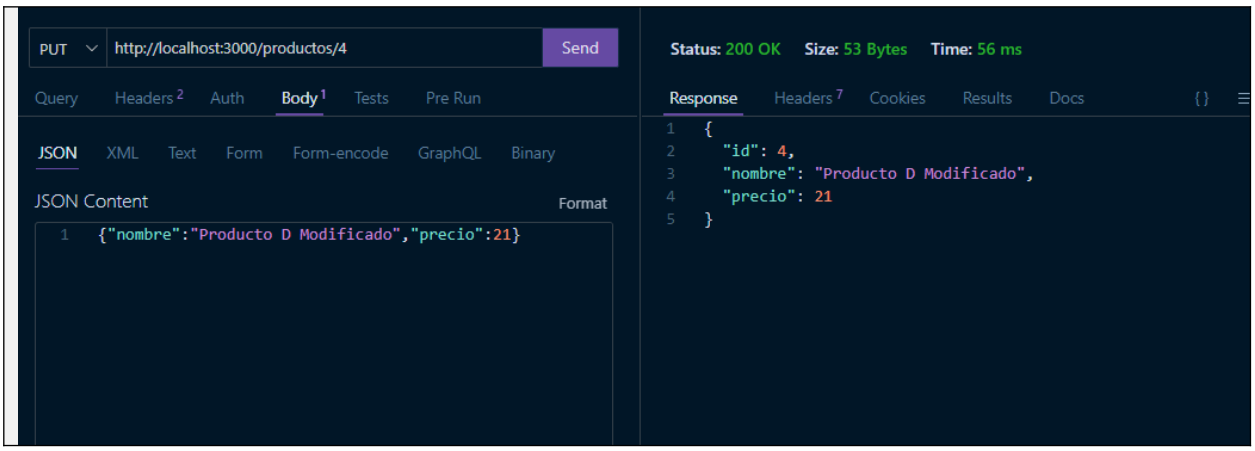
Captura del resultado



Actualizar un producto:

```
curl -X PUT http://localhost:3000/productos/1 -H "Content-Type: application/json" -d '{"precio":15.99}'
```

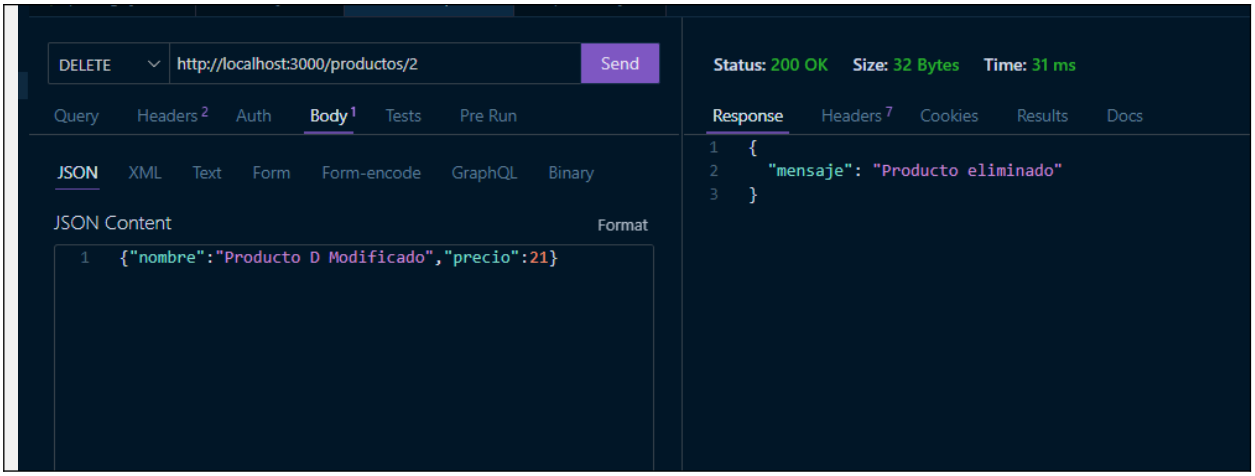
Captura del resultado



Eliminar un producto:

```
curl -X DELETE http://localhost:3000/productos/2
```

Captura del resultado


DELETE
http://localhost:3000/productos/2
Send
Status: 200 OK
Size: 32 Bytes
Time: 31 ms
Query
Headers 2
Auth
Body 1
Tests
Pre Run
Response
Headers 7
Cookies
Results
Docs
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{"nombre":"Producto D Modificado","precio":21}
{
"mensaje": "Producto eliminado"
}

TAREA

Agregar un nuevo recurso a la API

Objetivo: Extender la API RESTful con un nuevo conjunto de datos.

Instrucciones:

- Agrega un nuevo recurso a la API, por ejemplo, categorías de productos.
- Crea un nuevo archivo data/categories.json con algunas categorías.
- Implementa endpoints CRUD en routes/categories.js.
- Prueba los endpoints con Postman o cURL.

Implementar paginación en los productos

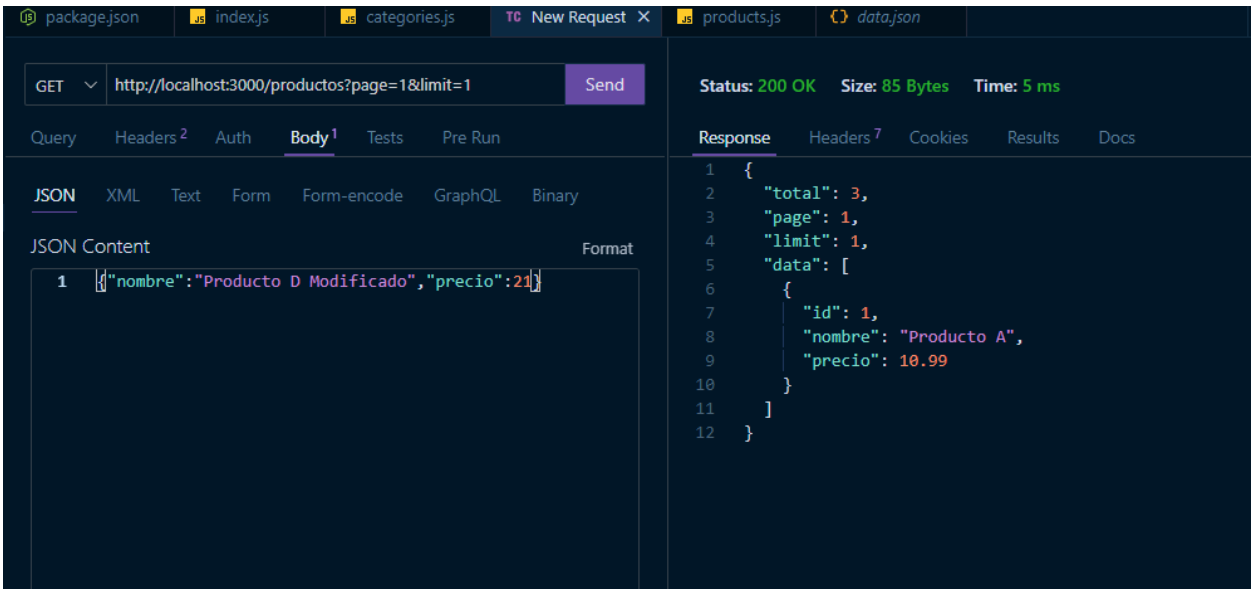
Objetivo: Optimizar el manejo de datos cuando la lista de productos crezca.

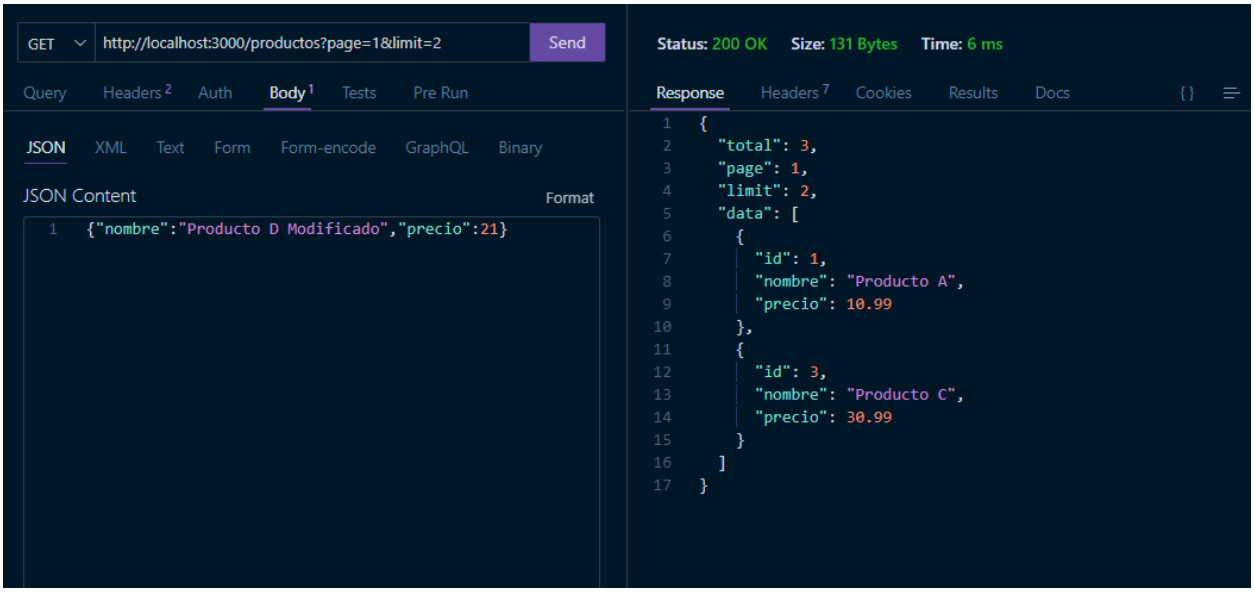
Instrucciones:

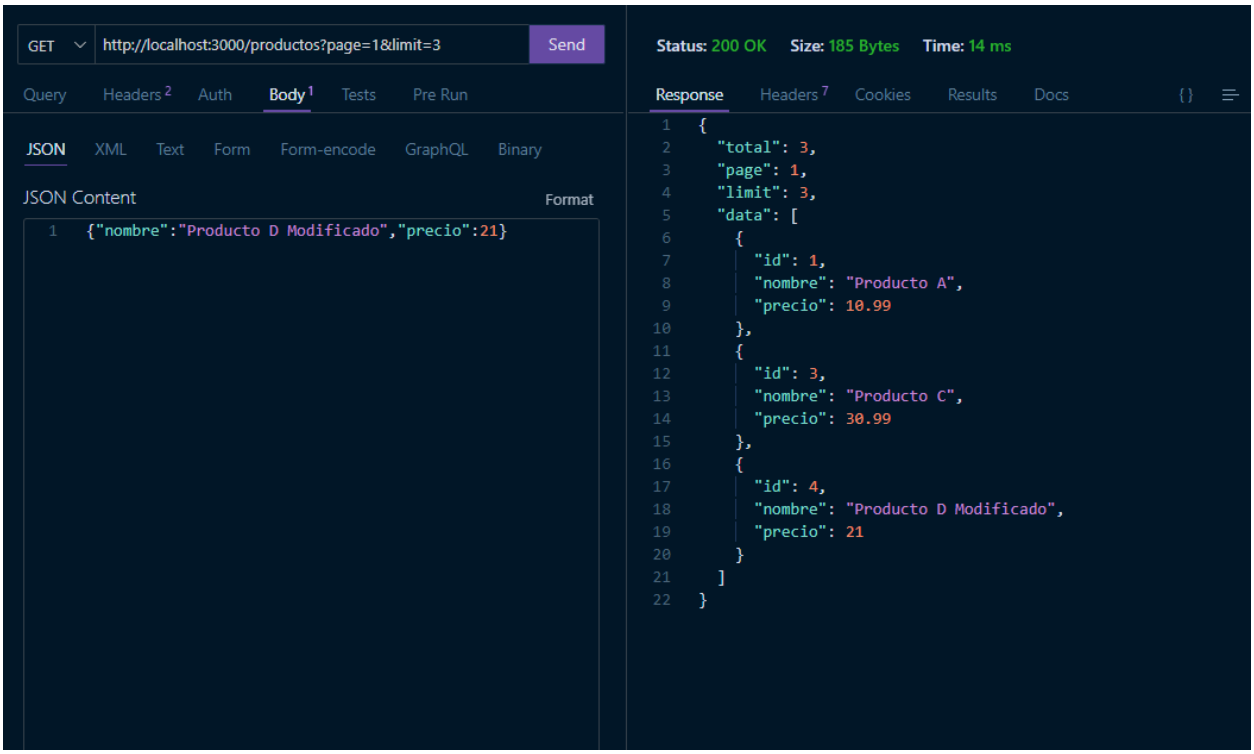
- Modifica el endpoint GET /productos para permitir paginación con ?page=1&limit=5.
- Usa query params (req.query) para obtener los valores de page y limit.

Colocar capturas del código y capturas de ejecución

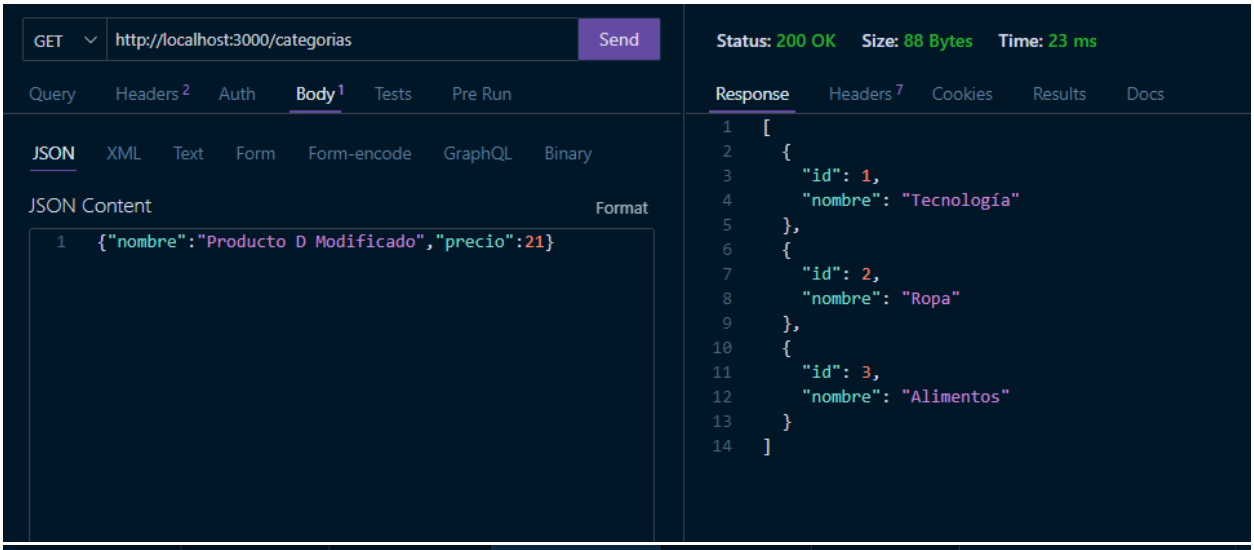
Grabar explicando el código y ejecución de este en máximo 4 minutos

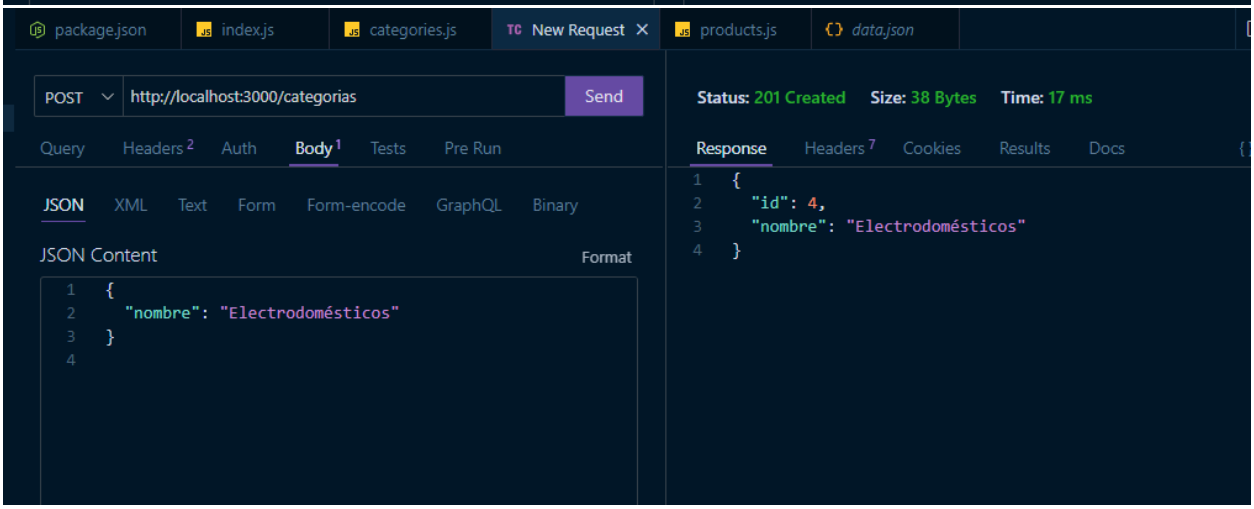


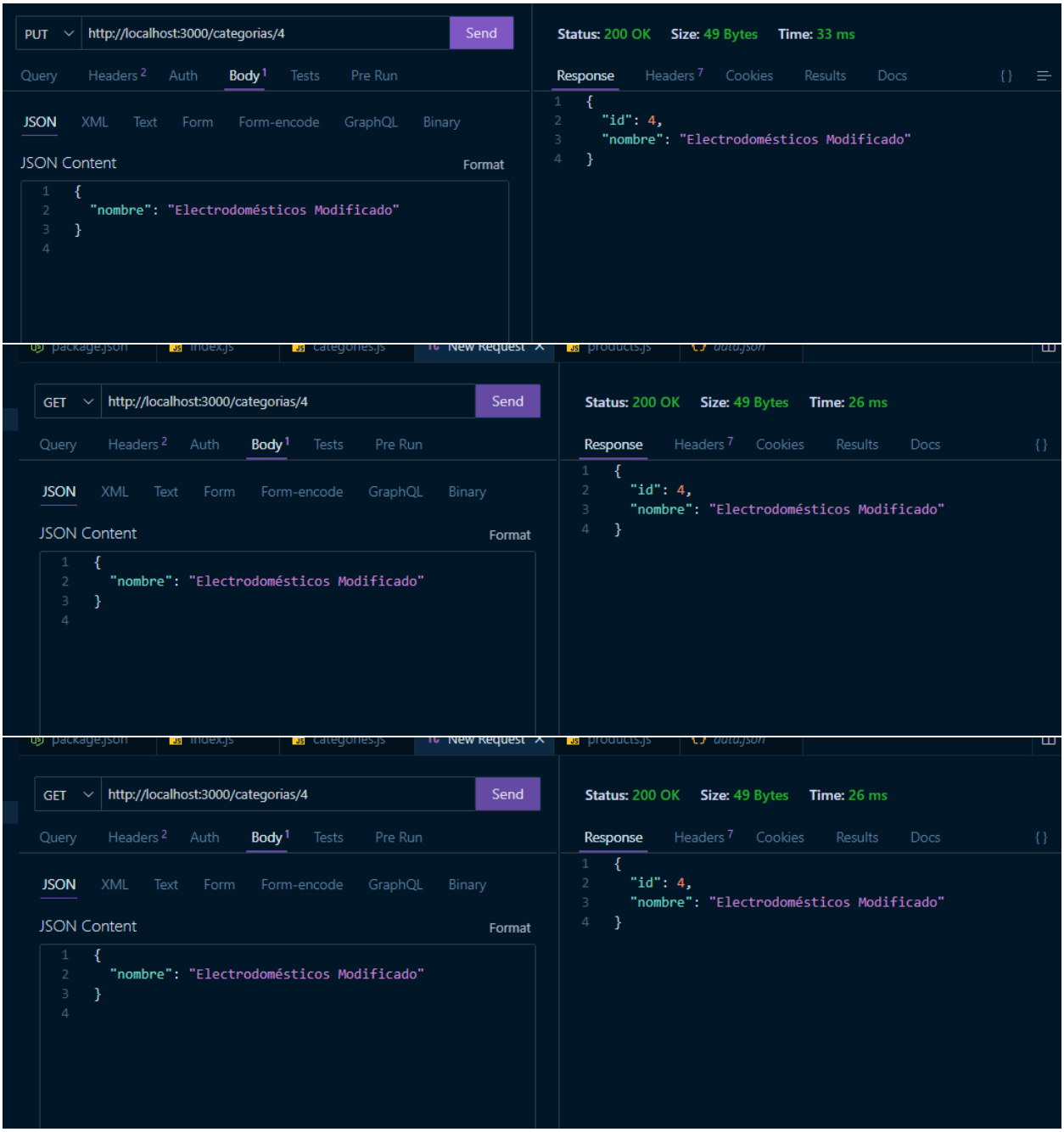

GET
http://localhost:3000/productos?page=1&limit=2
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{"nombre":"Producto D Modificado","precio":21}
Status: 200 OK
Size: 131 Bytes
Time: 6 ms
Response
Headers 7
Cookies
Results
Docs
{
"total": 3,
"page": 1,
"limit": 2,
"data": [
{
"id": 1,
"nombre": "Producto A",
"precio": 10.99
},
{
"id": 3,
"nombre": "Producto C",
"precio": 30.99
}
]
}


GET
http://localhost:3000/productos?page=1&limit=3
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{"nombre":"Producto D Modificado","precio":21}
Status: 200 OK
Size: 185 Bytes
Time: 14 ms
Response
Headers 7
Cookies
Results
Docs
{
"total": 3,
"page": 1,
"limit": 3,
"data": [
{
"id": 1,
"nombre": "Producto A",
"precio": 10.99
},
{
"id": 3,
"nombre": "Producto C",
"precio": 30.99
},
{
"id": 4,
"nombre": "Producto D Modificado",
"precio": 21
}
]
}

- Categorias


GET
http://localhost:3000/categorias
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{"nombre":"Producto D Modificado","precio":21}
Status: 200 OK
Size: 88 Bytes
Time: 23 ms
Response
Headers 7
Cookies
Results
Docs
[
{
"id": 1,
"nombre": "Tecnología"
},
{
"id": 2,
"nombre": "Ropa"
},
{
"id": 3,
"nombre": "Alimentos"
}
]


package.json
index.js
categories.js
New Request
products.js
data.json
POST
http://localhost:3000/categorias
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{
"nombre": "Electrodomésticos"
}
Status: 201 Created
Size: 38 Bytes
Time: 17 ms
Response
Headers 7
Cookies
Results
Docs
{
"id": 4,
"nombre": "Electrodomésticos"
}



OBSERVACIONES: *(Las observaciones son las notas aclaratorias, objeciones y problemas que se pudo presentar en el desarrollo del laboratorio)*

Se trabajó de forma modular, separando las rutas y los datos en archivos distintos

Se utilizó fs.readFileSync y fs.writeFileSync para simular operaciones de una base de datos.

No se añadieron validaciones para campos requeridos o tipos de datos.

Se logró implementar la paginación mediante req.query.page y req.query.limit, permitiendo la consulta de productos de manera más controlada.

CONCLUSIONES: *(Las conclusiones son una opinión sobre tu trabajo, explicar cómo resolviste las dudas o problemas presentados en el laboratorio. Además de aportar una opinión critica de lo realizado)*

- Se logró construir una API RESTful básica utilizando Node.js y Express, aplicando los métodos HTTP
- Separar las rutas y los datos mejora el orden del código, facilita su mantenimiento y permite extender fácilmente la API con nuevos recursos.
- Aunque no se utilizó una base de datos real, se comprendió el proceso de lectura, escritura y actualización de datos usando archivos JSON como fuente persistente.
- Se utilizaron herramientas como Thunder Client y cURL para verificar el funcionamiento de la API, lo cual permitió detectar errores rápidamente y validar los resultados esperados.