César Cázares

CRUD USERS | CATEGORY | BRANDS | PRODUCTS

Universidad La Salle Bajío
Facultad de Ingenierías y Tecnologías

índice

Tabla de contenido

Introducción y Objetivo

Este documento presenta el desarrollo de un proyecto universitario sobre la creación de una API REST modular utilizando Node.js, Express.js y otras tecnologías. El proyecto implementa un sistema de gestión para una tienda en línea con entidades modulares.

Objetivo Principal: Crear endpoints modulados para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en las siguientes entidades:

- **USERS**: Gestión de usuarios
- **CATEGORIES**: Gestión de categorías de productos
- **BRANDS**: Gestión de marcas
- **PRODUCTS**: Gestión de productos
- **MOVIES**: Gestión de películas (módulo adicional)

Requerimientos Previos

Para el desarrollo de este proyecto se requirió:

- **Editor de código**: Visual Studio Code
- **Tecnologías**: Node.js, Express.js, Faker.js
- **Conocimientos**: JavaScript, manejo de terminal, conceptos de APIs REST
- **Dependencias**: express, faker, nodemon, eslint, prettier

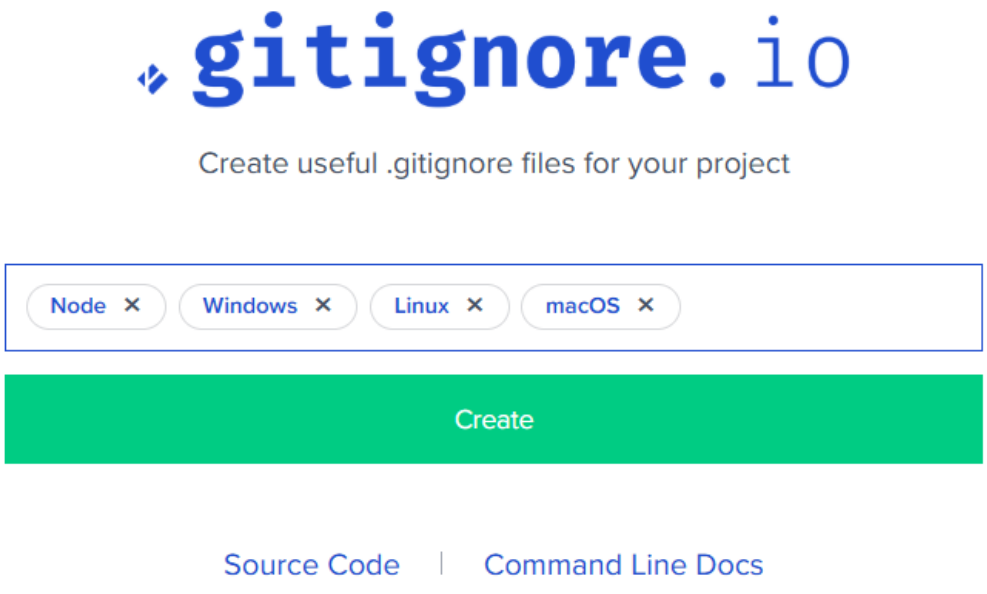
Desarrollo

Configuración del Entorno

Se inició el proyecto con los siguientes comandos:

```
npm install express
npm install faker
```

Se configuró un archivo .gitignore utilizando gitignore.io para excluir archivos innecesarios del control de versiones.



Estructura del Proyecto

```
mystore312/
├── index.js
├── package.json
├── .eslintrc.json
├── .gitignore
└── routes/
    ├── rutas.js
    ├── productsRouter.js
    ├── UsersRouter.js
    ├── CategoriesRouter.js
    ├── BrandsRouter.js
    └── moviesRouter.js
```

Implementación del Servidor Principal (index.js)

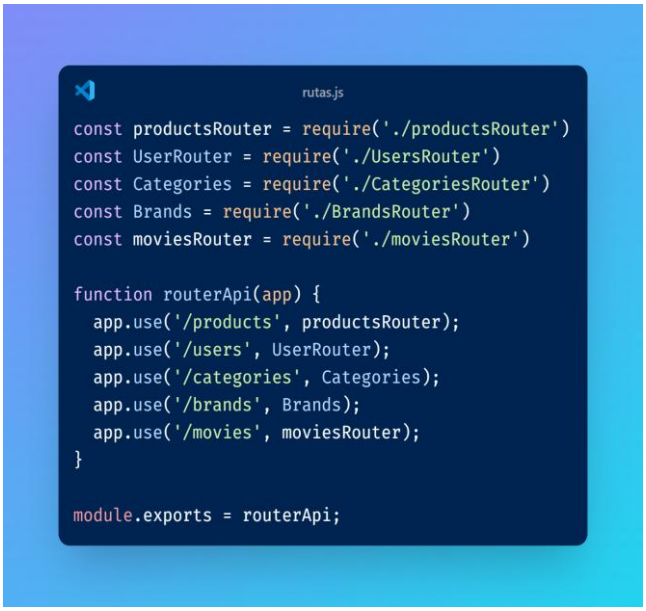
El archivo principal configura el servidor Express:

index.js

```
const express = require('express');
const faker = require('faker');
const routerApi = require('./routes/rutas')
const app = express();
const port = 4000;

//Middleware to parse JSON bodies
app.use(express.json());

app.get("/", (req, res) => {
  res.send("Hola mi server en express")
})

app.get("/nuevaruta", (req, res) => {
  res.send("Hola soy una nueva ruta")
})

app.listen(port, () => {
  console.log("Mi port is working on: " + port)
  console.log("http://localhost:" + port)
})

routerApi(app);
```

Sistema de Enrutamiento Modular (rutas.js)

Archivo centralizador que organiza todas las rutas:



Módulos Implementados

1. ProductsRouter.js

- **Endpoints**:
    - GET /products - Lista todos los productos
    - GET /products/:id - Obtiene producto por ID
    - GET /products/category/:categoryId - Filtra por categoría
    - GET /products/brand/:brandId - Filtra por marca
    - POST /products - Crea nuevo producto
    - PATCH /products/:id - Actualiza producto
    - DELETE /products/:id - Elimina producto

2. UsersRouter.js

- **Endpoints**:
    - GET /users - Lista todos los usuarios

    - GET /users/:id - Obtiene usuario por ID
    - POST /users - Crea nuevo usuario
    - PATCH /users/:id - Actualiza usuario
    - DELETE /users/:id - Elimina usuario

3. CategoriesRouter.js

- **Endpoints**:
    - GET /categories - Lista todas las categorías
    - GET /categories/:id - Obtiene categoría por ID
    - POST /categories - Crea nueva categoría
    - PATCH /categories/:id - Actualiza categoría
    - DELETE /categories/:id - Elimina categoría

4. BrandsRouter.js

- **Endpoints**:
    - GET /brands - Lista todas las marcas
    - GET /brands/:id - Obtiene marca por ID
    - POST /brands - Crea nueva marca
    - PATCH /brands/:id - Actualiza marca
    - DELETE /brands/:id - Elimina marca

5. MoviesRouter.js (Módulo Adicional)

- **Endpoints**:
    - GET /movies - Lista todas las películas
    - GET /movies/:id - Obtiene película por ID
    - POST /movies - Crea nueva película
    - PATCH /movies/:id - Actualiza película
    - DELETE /movies/:id - Elimina película

Características Técnicas

- **Generación de Datos**: Uso de Faker.js para datos de prueba
- **Validaciones**: Manejo de errores 404 para recursos no encontrados
- **Métodos HTTP**: Implementación completa de GET, POST, PATCH, DELETE
- **Estructura Modular**: Código organizado y mantenible
- **Middleware**: Parseo automático de JSON

Ejecución del Proyecto

Para ejecutar el proyecto:

```
npm run dev
```

El servidor se inicia en: http://localhost:4000

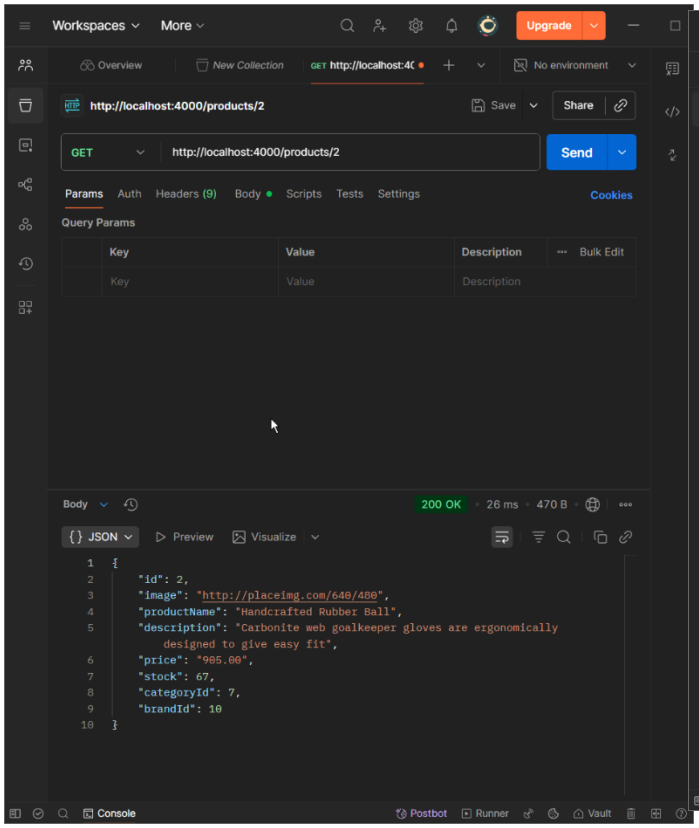
Resultados Obtenidos

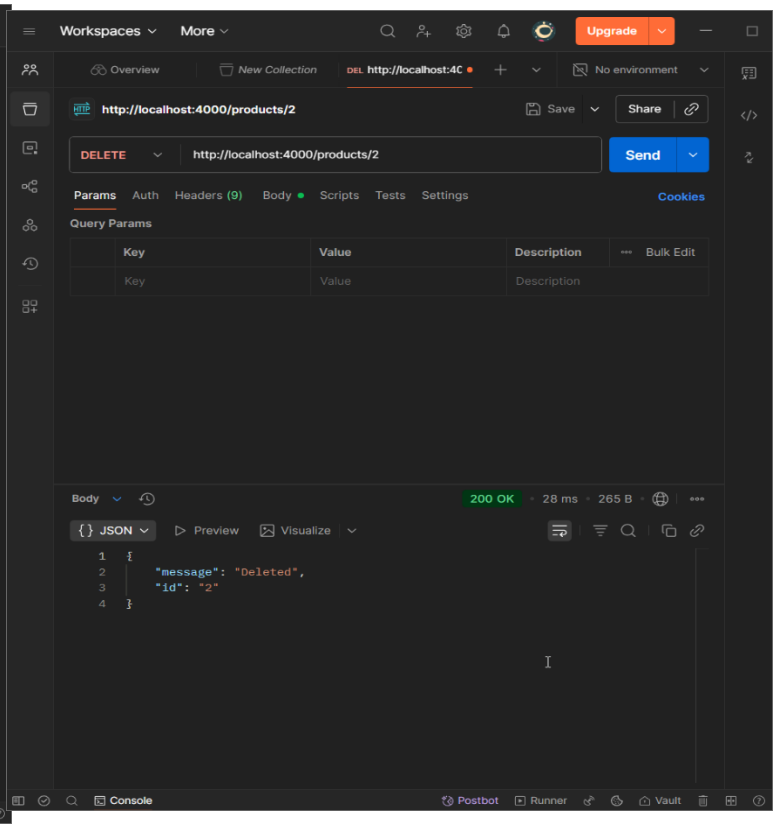
Endpoints Principales

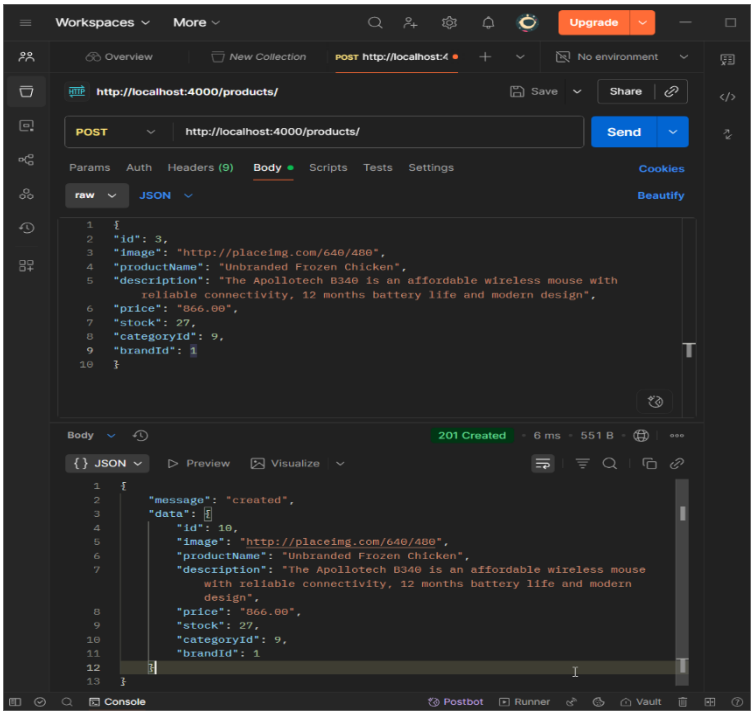
1. **Endpoint Raíz**: GET / - Mensaje de bienvenida
2. **Products**: Operaciones CRUD completas con filtros por categoría y marca
3. **Users**: Gestión completa de usuarios
4. **Categories**: Administración de categorías
5. **Brands**: Administración de marcas
6. **Movies**: Módulo adicional para gestión de películas

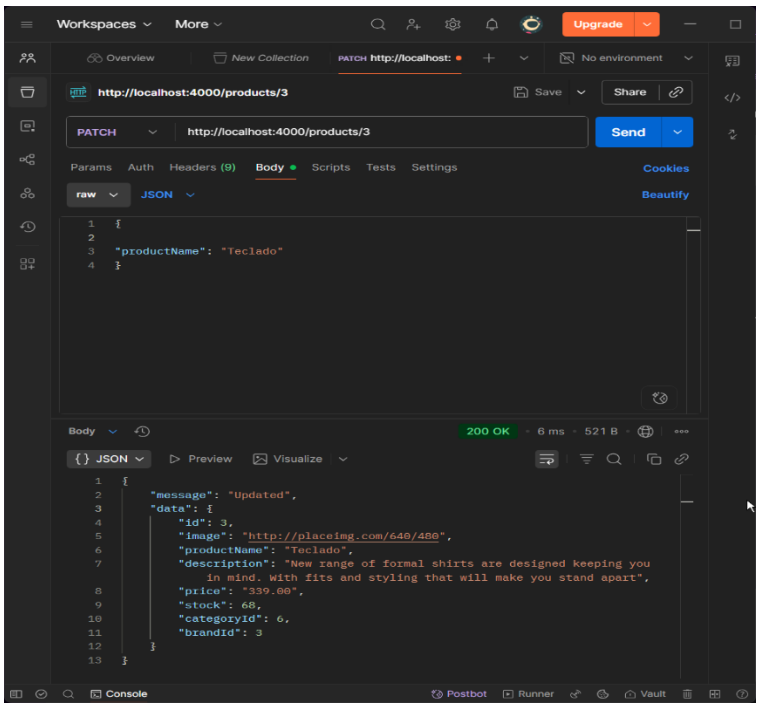
Resultados 1 - Vista principal de la documentación automática

Resultados 2 – vista CRUD para products

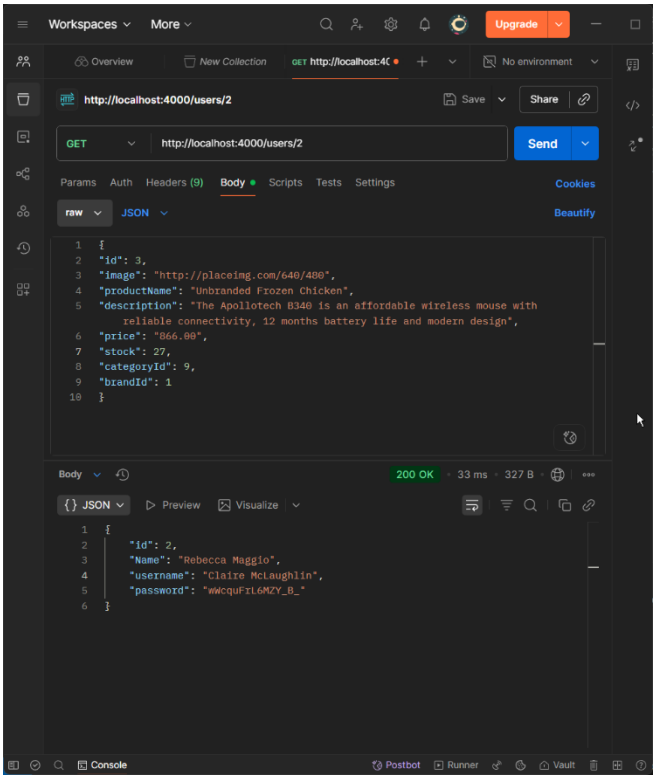


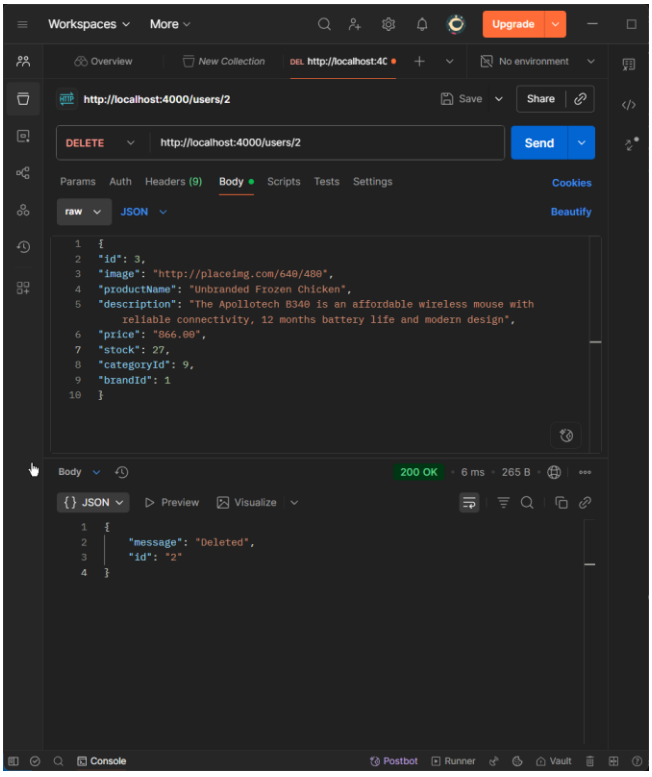




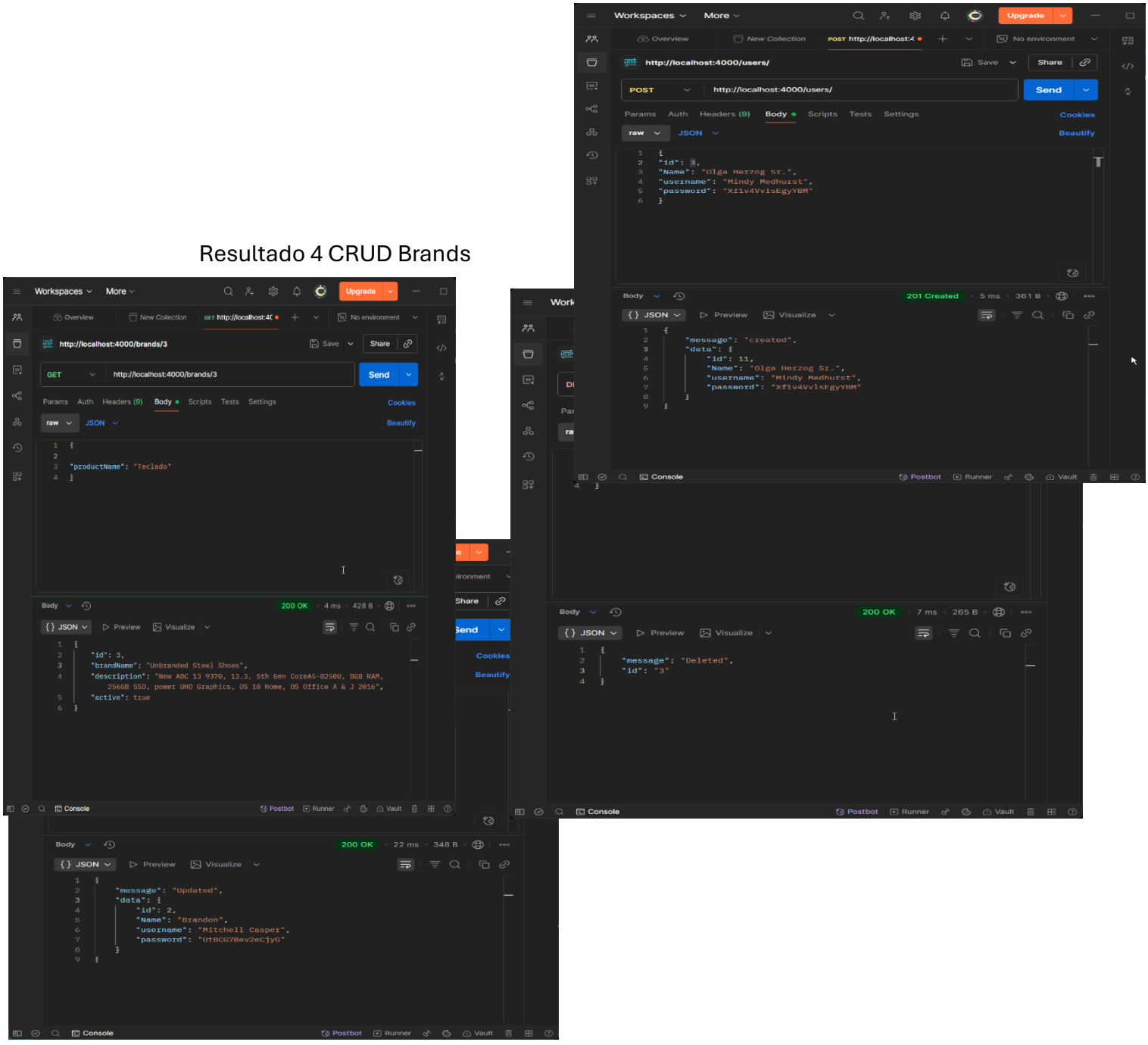


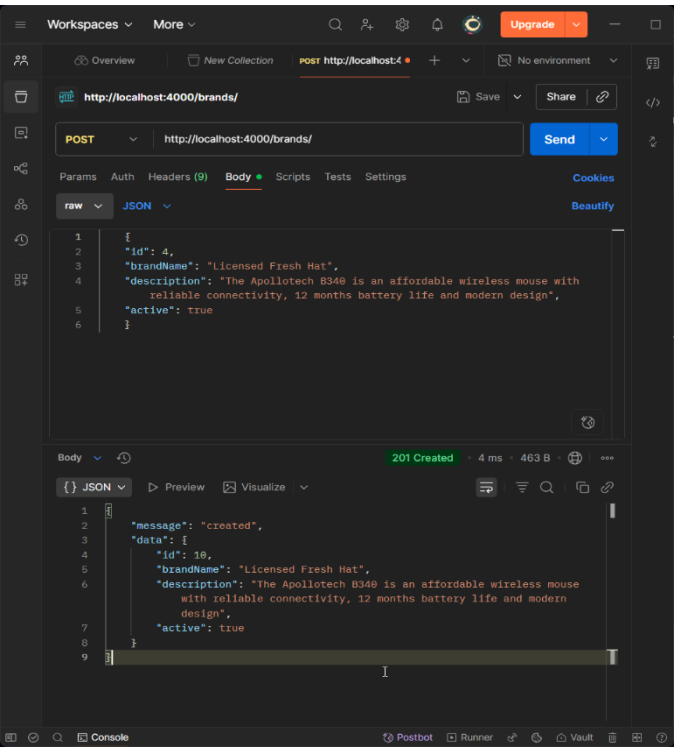
Resultado 3 CRUD users

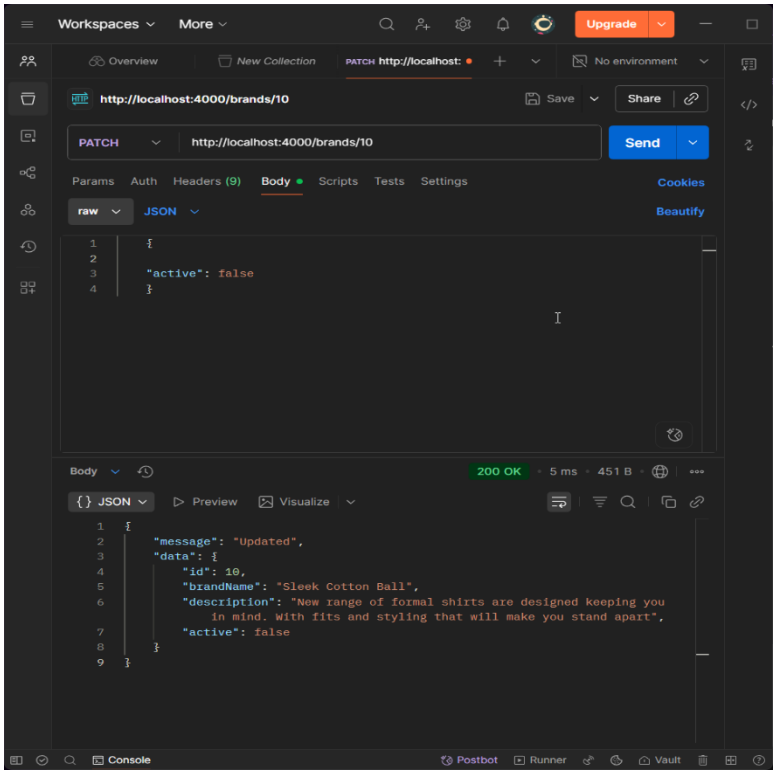




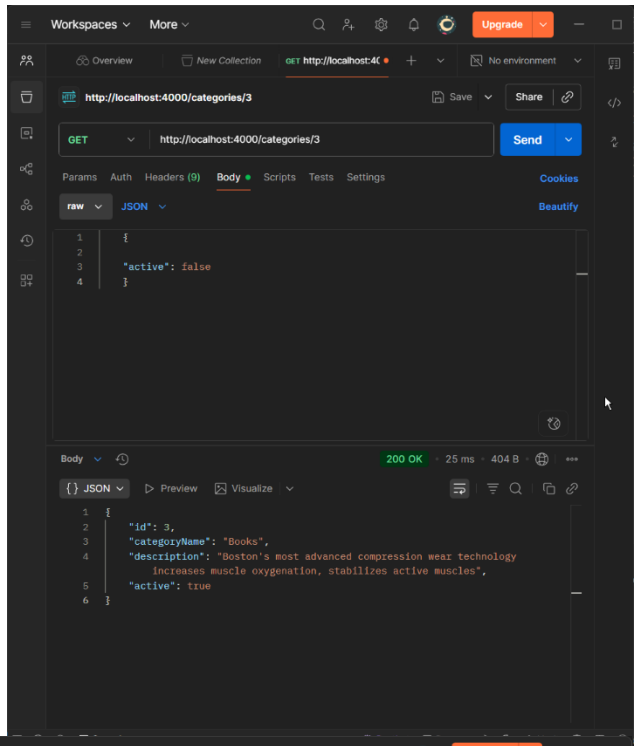
Resultado 4 CRUD Brands

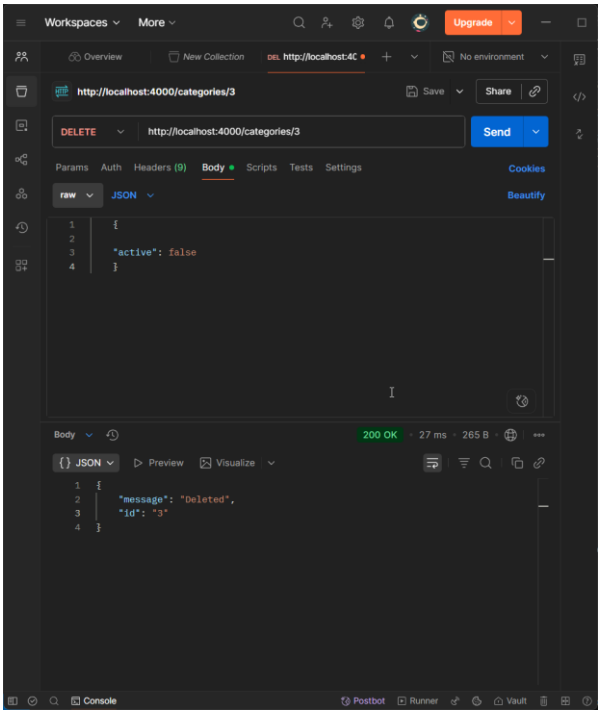


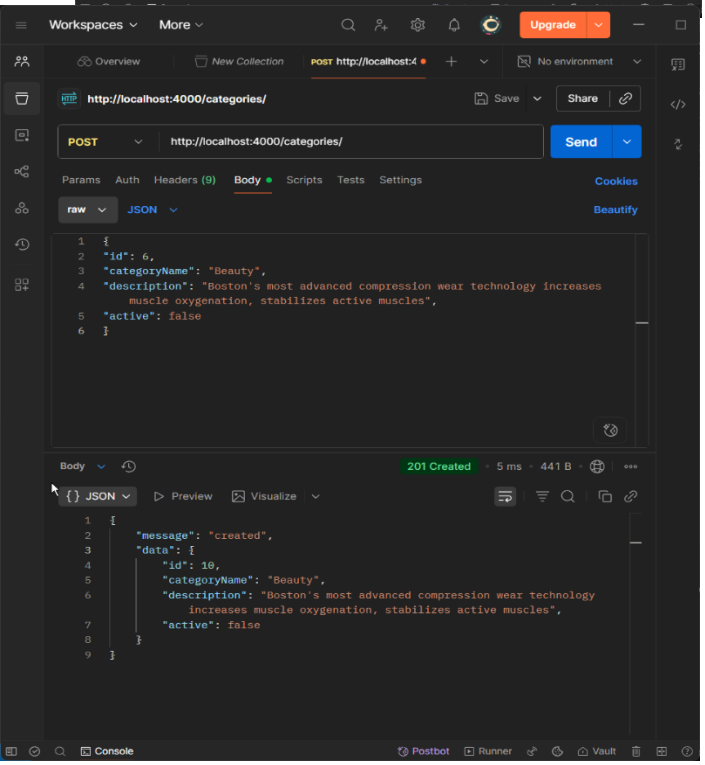


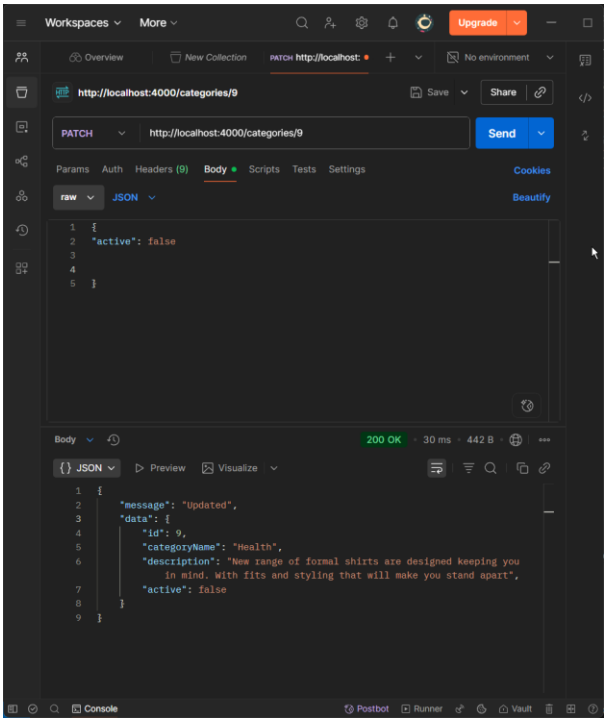


Resultado 4 CRUD category


Workspaces
More
Upgrade
Overview
New Collection
GET http://localhost:4(
No environment
http://localhost:4000/categories/3
Save
Share
GET
http://localhost:4000/categories/3
Send
Params
Auth
Headers (9)
Body
Scripts
Tests
Settings
Cookies
raw
JSON
Beautify
{
"active": false
}
Body
200 OK
25 ms
404 B
{ } JSON
Preview
Visualize
{
"id": 3,
"categoryName": "Books",
"description": "Boston's most advanced compression wear technology increases muscle oxygenation, stabilizes active muscles",
"active": true
}


Workspaces
More
Upgrade
Overview
New Collection
DEL http://localhost:4(
No environment
http://localhost:4000/categories/3
Save
Share
DELETE
http://localhost:4000/categories/3
Send
Params
Auth
Headers (9)
Body
Scripts
Tests
Settings
Cookies
raw
JSON
Beautify
{
"active": false
}
Body
200 OK
27 ms
265 B
{ } JSON
Preview
Visualize
{
"message": "Deleted",
"id": "3"
}
Console
Postbot
Runner
Vault


Workspaces
More
Upgrade
Overview
New Collection
POST http://localhost:4
No environment
http://localhost:4000/categories/
Save
Share
POST
http://localhost:4000/categories/
Send
Params
Auth
Headers (9)
Body
Scripts
Tests
Settings
Cookies
raw
JSON
Beautify
{
"id": 6,
"categoryName": "Beauty",
"description": "Boston's most advanced compression wear technology increases muscle oxygenation, stabilizes active muscles",
"active": false
}
Body
201 Created
5 ms
441 B
{ } JSON
Preview
Visualize
{
"message": "created",
"data": [
{
"id": 10,
"categoryName": "Beauty",
"description": "Boston's most advanced compression wear technology increases muscle oxygenation, stabilizes active muscles",
"active": false
}
]
Console
Postbot
Runner
Vault


Workspaces
More
Upgrade
Overview
New Collection
PATCH http://localhost:
No environment
http://localhost:4000/categories/9
Save
Share
PATCH
http://localhost:4000/categories/9
Send
Params
Auth
Headers (9)
Body
Scripts
Tests
Settings
Cookies
raw
JSON
Beautify
{
"active": false
}
Body
200 OK
30 ms
442 B
{ } JSON
Preview
Visualize
{
"message": "Updated",
"data": [
{
"id": 9,
"categoryName": "Health",
"description": "New range of formal shirts are designed keeping you in mind. With fits and styling that will make you stand apart",
"active": false
}
]
Console
Postbot
Runner
Vault

Conclusiones

Este proyecto se desarrolló lo siguiente para el correcto funcionamiento del proyecto y también para el aprendizaje de una ApiRest:

1. **Aprendizaje Práctico**: Desarrollo manual desde cero de un servidor Express
2. **Modularización**: Organización efectiva del código en routers especializados
3. **API REST Completa**: Implementación de todos los verbos HTTP y operaciones CRUD
4. **Manejo de Dependencias**: Uso correcto de npm para gestión de paquetes
5. **Calidad de Código**: Configuración de herramientas como ESLint y Prettier
6. **Documentación**: Creación de reportes detallados del progreso