

DISEÑO Y DESARROLLO DE SERVICIOS WEB - PROYECTO

APRENDIZ

BRAYAN STIVEN PEÑA QUINAYAS

ADSO

SENA

2023

INTRODUCCION

En el contexto del proyecto formativo, nos proponemos diseñar y desarrollar APIs para nuestra tienda de videojuegos online. Se utilizará JavaScript como lenguaje de programación principal, aprovechando su versatilidad y amplio uso en el desarrollo web moderno. El objetivo es crear servicios web robustos y escalables que permitan gestionar productos, usuarios, pedidos y otras funcionalidades esenciales de una tienda online.

HERRAMIENTAS:

- Entorno de desarrollo: Visual Studio Code
- Herramienta de Versionamiento: Git, Github
 - Documentación: Postman

TECNOLOGIAS Y BIBLIOTECAS:

- Node.js y Express para backend.
- MongoDB para almacenamiento de datos.

1. DEFINICION DE REQUISITOS Y CARACTERISTICAS DEL SOFTWARE

Análisis de requisitos:

Para una tienda online, los requisitos básicos serian:

- **Gestión de Productos:** Permitir añadir, modificar, eliminar y listar productos.
- **Gestión de Usuarios:** Registro, inicio de sesión, edición de perfil y gestión de pedidos.
- **Carrito de Compras:** Añadir productos al carito, modificar cantidades y finalizar compras.
- **Procesamiento de Pagos:** integración con pasarelas de pago para procesar transacciones.
- **Seguridad:** Autenticación y autorización de usuarios, encriptación de datos sensibles.

Selección de Tecnologías:

Dado que se utilizará JavaScript podremos optar por las siguientes tecnologías:

- **Backend:** Node.js con express para crear el servidor y gestionar las APIs.
- **Base de datos:** MongoDB como base de datos para almacenar productos usuarios y pedidos.
 - **Frontend:** React
 - **Pasarela de pago:** PayPal o Stripe.

2. DISEÑO DE LAS APIs

Definición de Endpoints:

Basándonos en los requisitos identificados, podemos definir los siguientes endpoints para nuestra tienda online:

PRODUCTOS:

- ``GET / productos``: Obtener la lista de todos los productos.
- ``GET / productos/:id``: Actualizar un producto existente por ID.
 - ``POST /productos``: Añadir un nuevo producto.
- ``PUT / productos/:id``: Actualizar un producto existente por ID.
 - ``DELETE /productos/id``: Eliminar un producto ID.

USUARIOS:

- ``POST /usuarios/registro``: Registrar un nuevo usuario.
 - ``POST /usuarios/login``: Iniciar sesión de un usuario.
- ``GET /usuarios/id``: Obtener información de un usuario por ID.
- ``PUT /usuarios/id``: Actualizar información de un usuario por ID.

PEDIDOS:

- ``POST /pedidos``: Crear un nuevo pedido.
- ``GET /pedidos/id``: Obtener detalles de un pedido por ID.
- ``PUT /pedidos/:id``: Actualizar estado de un pedido por ID.

Modelado de datos:

Para el modelado de datos, consideramos los siguientes esquemas en formato JSON:

PRODUCTO:

```
VS Iniciar JS app.js {} modelado.json X
{} modelado.json > ...
1  {
2    "id": "string",
3    "nombre": "string",
4    "descripcion": "string",
5    "precio": "Number",
6    "imagen": "string"
7  }
8
9
```

USUARIO:

```
VS Iniciar JS app.js {} modelado.json X
{} modelado.json > ...
1  {
2    "id": "string",
3    "nombre": "string",
4    "email": "string",
5    "password": "string(hash)",
6    "direccion": "string",
7    "telefono": "number"
8  }
9
```

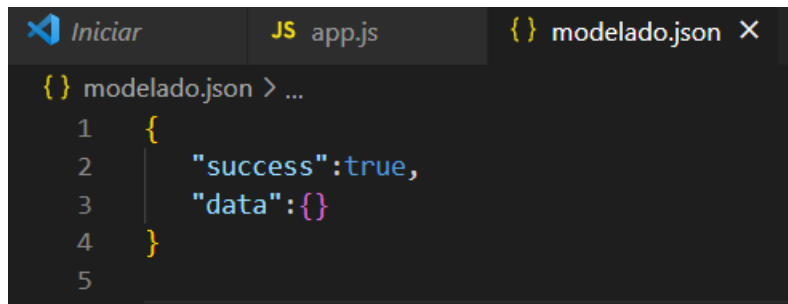
PEDIDO:

```
VS Iniciar JS app.js {} modelado.json X
{} modelado.json > ...
1  {
2    "id": "string",
3    "usuarioID": "string",
4    "productos": [
5      {
6        "productos": "string",
7        "cantidad": "Number"
8      }
9    ],
10   "total": "number",
11   "estado": "string"
12 }
13
14
15
```

Esquema de Respuesta:

Respuesta de cada endpoint:

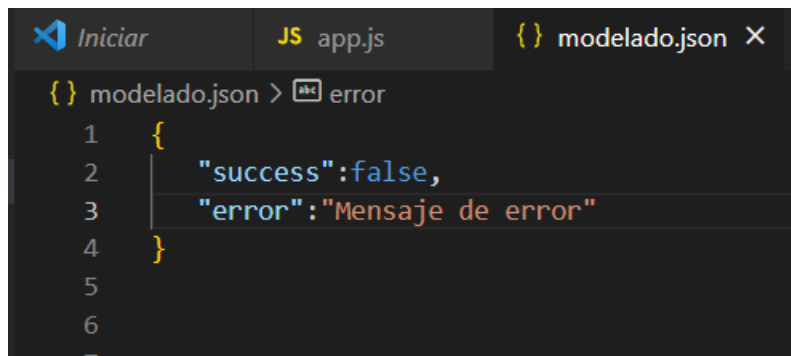
Respuesta exitosa:



The screenshot shows a Visual Studio Code editor with three tabs: 'Iniciar', 'JS app.js', and 'modelado.json'. The 'modelado.json' tab is active, displaying a JSON object with the following structure:

```
{  
  "success": true,  
  "data": {}  
}
```

Respuesta de error:



The screenshot shows the same Visual Studio Code editor with the 'modelado.json' tab active. The JSON object now represents an error response:

```
{  
  "success": false,  
  "error": "Mensaje de error"  
}
```

3. CODIFICACION DE LAS APIs

Configuración del entorno:

- Primero debemos tener instalado Node.js
- Creamos el nuevo proyecto de Node.js:

```
C:\Users\LENOVO\Desktop\endpoints> npm init -y
Wrote to C:\Users\LENOVO\Desktop\endpoints\package.json:

{
  "name": "endpoints",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

- Instalamos las respectivas dependencias:

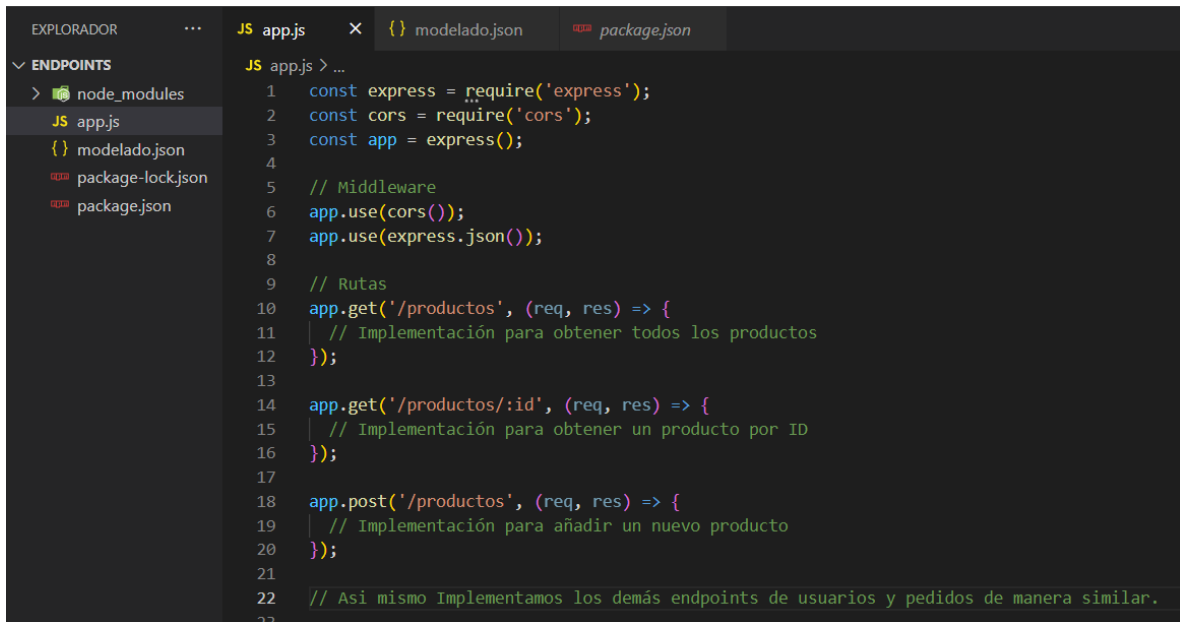
```
C:\Users\LENOVO\Desktop\endpoints> npm install express mongoose cors
added 86 packages, and audited 87 packages in 44s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 10.1.0 -> 10.6.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.6.0
npm notice Run npm install -g npm@10.6.0 to update!
npm notice
```

Desarrollo de Endpoints

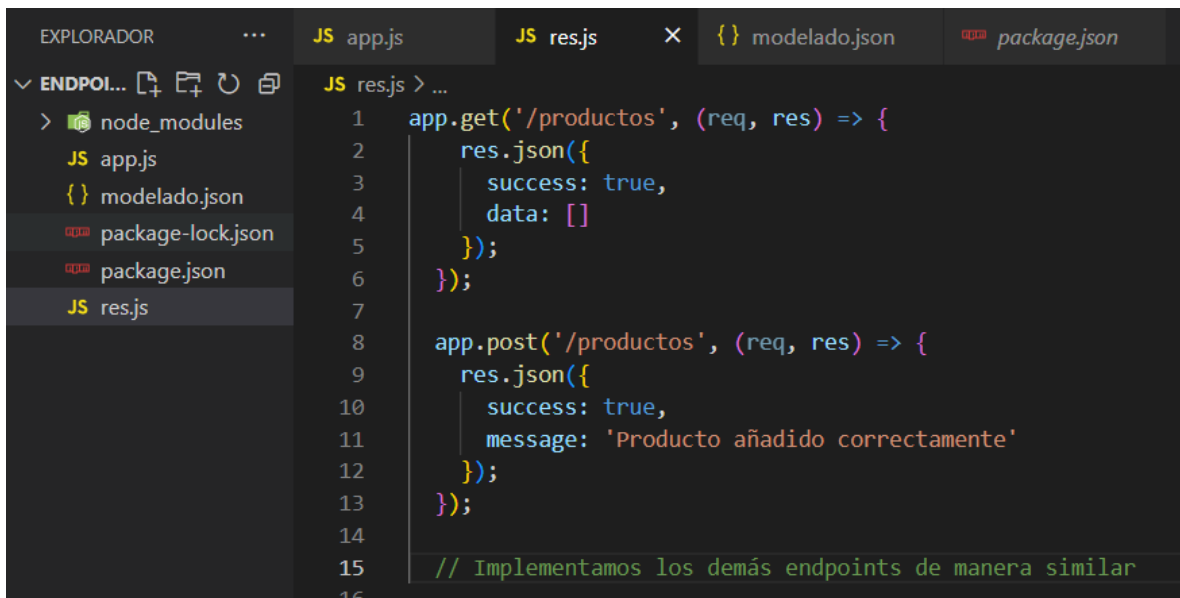
Creamos un archivo js para empezar a definir los endpoints utilizando Express:



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files: node_modules, app.js, modelado.json, package-lock.json, and package.json. The code editor shows the content of app.js, which includes imports for express and cors, middleware setup, and initial route definitions for GET /productos and GET /productos/:id.

```
1 const express = require('express');
2 const cors = require('cors');
3 const app = express();
4
5 // Middleware
6 app.use(cors());
7 app.use(express.json());
8
9 // Rutas
10 app.get('/productos', (req, res) => {
11   // Implementación para obtener todos los productos
12 });
13
14 app.get('/productos/:id', (req, res) => {
15   // Implementación para obtener un producto por ID
16 });
17
18 app.post('/productos', (req, res) => {
19   // Implementación para añadir un nuevo producto
20 });
21
22 // Así mismo Implementamos los demás endpoints de usuarios y pedidos de manera similar.
23
```

Implementamos los Endpoints:



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files: node_modules, app.js, modelado.json, package-lock.json, package.json, and res.js. The code editor shows the content of res.js, which includes route definitions for GET /productos and POST /productos, both using res.json() to return data.

```
1 app.get('/productos', (req, res) => {
2   res.json({
3     success: true,
4     data: []
5   });
6 });
7
8 app.post('/productos', (req, res) => {
9   res.json({
10    success: true,
11    message: 'Producto añadido correctamente'
12  });
13 });
14
15 // Implementamos los demás endpoints de manera similar
16
```


Pruebas locales:

Ejecución del servidor:

```
25
26 app.listen(3000, () => {
27   console.log('Servidor corriendo en puerto 3000');
28 });
29
30
```

4. DOCUMENTACION DE ENDPOINTS

- Endpoint / productos:

Descripción:

Este endpoint permite gestionar los productos de la tienda.

Propósito:

Administrar la creación, recuperación, actualización y eliminación de productos.

Métodos permitidos:

GET: obtener lista de productos

POST: Añadir un nuevo producto

Parámetros de entrada:

GET: no requiere parámetros

POST:

```
{ } modelado.json > ...
1  {
2    "nombre": "String",
3    "descripcion": "String",
4    "precio": "Number",
5    "imagen": "String"
6  }
7
```

Formato de respuesta:

```
{ } modelado.json > ...
1  {
2    "success": "Boolean",
3    "data": "Array de Productos o Mensaje de Éxito/Error"
4  }
5
6
7
```

- Endpoint / Usuarios

Descripción:

Este endpoint permite gestionar los usuarios de la tienda.

Propósito:

Administrar el registro, inicio de sesión, obtención y actualización de información de usuarios.

Métodos Permitidos:

GET: Registrar un nuevo usuario

GET: Obtener información de un usuario por ID.

PUT: Actualizar información de un usuario por ID.

Parámetros de entrada:

POST:

```
{ } modelado.json > ...
1  {
2    "nombre": "String",
3    "email": "String",
4    "password": "String",
5    "direccion": "String",
6    "telefono": "String"
7  }
8
```

GET y PUT no requieren parámetros adicionales.

Formato de respuesta:

```
{ } modelado.json > ...
1  {
2    "success": "Boolean",
3    "data": "Información del Usuario o Mensaje de Éxito/Error"
4  }
5
```

- Endpoint / pedidos

Descripción:

Este endpoint permite gestionar los pedidos realizados en la tienda.

Propósito:

Administrar la creación, obtención y actualización de pedidos.

Métodos permitidos:

POST: Crear un nuevo pedido.

GET: Obtener detalles de un pedido por ID.

PUT: Actualizar estado de un pedido por ID.

Parámetros de entrada

POST:

```
{ } modelado.json > ...
1  {
2    "usuarioId": "String",
3    "productos": [
4      {
5        "productoId": "String",
6        "cantidad": "Number"
7      }
8    ],
9    "total": "Number"
10 }
11
```

GET y PUT: No requieren parámetros adicionales.

Formato de respuesta:

```
{ } modelado.json > ...  
1  {  
2    "success": "Boolean",  
3    "data": "Información del Pedido o Mensaje de Éxito/Error"  
4  }  
5
```

5. VERSIONAMIENTO DEL PROYECTO.

Inicialización del repositorio GIT

Creamos un nuevo repositorio y subimos el proyecto.

Link del repositorio:

<https://github.com/stivenrpzx/Endpoints.git>

CONCLUSION

En este taller hemos aprendido a diseñar y desarrollar APIs para una tienda online utilizando JavaScript como lenguaje de programación principal. Hemos abordado cada paso, desde la definición de requisitos hasta el versionamiento y organización.

Se han diseñado endpoints claros y funcionales para gestionar productos, usuarios y pedidos, y los hemos implementado utilizando Express.js y MongoDD. Además, hemos documentado detalladamente cada uno de los endpoints.