**Tutorial: Crear una API REST con MongoDB y Fetch API**

**1. Introducción**

En este tutorial aprenderás a crear una API REST con **Node.js, Express y MongoDB** y a conectarla con un **frontend en HTML/JavaScript** mediante **Fetch API**. Explicaremos todos los conceptos teóricos paso a paso para que puedas comprender cómo funciona el flujo de datos entre el frontend y el backend.

**2. ¿Qué es una API REST?**

**2.1. Definición de API**

Una **API (Interfaz de Programación de Aplicaciones)** permite que diferentes aplicaciones se comuniquen entre sí. En nuestro caso, el frontend (HTML/JavaScript) podrá hacer peticiones al backend (Node.js + MongoDB) para gestionar datos.

**2.2. ¿Qué significa REST?**

**REST (Representational State Transfer)** es un conjunto de principios que define cómo deben estructurarse las APIs. Una API RESTful sigue estas reglas:

* **Usa métodos HTTP**: GET, POST, PUT, DELETE.
* **Responde en formato JSON**.
* **Es stateless**: cada petición es independiente y el servidor no guarda estado de conexión.

**3. Instalación y configuración del entorno**

**3.1. Requisitos previos**

Antes de comenzar, necesitas instalar:

* **Node.js** (Descargar de <https://nodejs.org/>)
* **MongoDB** (Descargar de <https://www.mongodb.com/try/download/community>)

Verifica que están instalados:

node -v # Verificar versión de Node.js

npm -v # Verificar versión de npm

mongod --version # Verificar versión de MongoDB

**3.2. Crear el proyecto**

Abre la terminal y ejecuta:

mkdir gestion-almacen

cd gestion-almacen

npm init -y

Esto creará un archivo package.json que administrará las dependencias del proyecto.

**3.3. Instalar dependencias**

Ejecuta el siguiente comando para instalar las herramientas necesarias:

npm install express mongoose cors dotenv

Explicación:

* express: Framework para crear el servidor web.
* mongoose: Interactuar con MongoDB.
* cors: Permitir conexiones desde el frontend.
* dotenv: Manejo de variables de entorno.

**4. Crear el servidor con Express y conectar MongoDB**

**4.1. Configurar MongoDB**

Ejecuta MongoDB en una nueva terminal:

Mongod

**4.2. Crear el archivo server.js**

Crea el archivo server.js en la carpeta raíz del proyecto:

require('dotenv').config();

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const app = express();

app.use(cors());

app.use(express.json());

// Conexión a MongoDB

mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

}).then(() => console.log("Conectado a MongoDB"))

.catch(err => console.error(err));

app.get('/', (req, res) => {

res.send('¡Bienvenido al sistema de gestión de almacén!');

});

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Servidor corriendo en http://localhost:${PORT}`);

});

**4.3. Configurar variables de entorno**

Crea el archivo .env en la raíz del proyecto:

MONGO\_URI=mongodb://localhost:27017/almacen

PORT=3000

**5. Crear el modelo de datos (MongoDB)**

Crea una carpeta models y dentro un archivo Producto.js:

const mongoose = require('mongoose');

const ProductoSchema = new mongoose.Schema({

nombre: { type: String, required: true },

precio: { type: Number, required: true },

stock: { type: Number, required: true }

});

module.exports = mongoose.model('Producto', ProductoSchema);

**6. Crear las rutas de la API**

Crea una carpeta routes y dentro un archivo productos.js:

const express = require('express');

const Producto = require('../models/Producto');

const router = express.Router();

// Obtener todos los productos

router.get('/', async (req, res) => {

const productos = await Producto.find();

res.json(productos);

});

// Agregar un producto

router.post('/', async (req, res) => {

const nuevoProducto = new Producto(req.body);

await nuevoProducto.save();

res.json({ mensaje: 'Producto agregado', producto: nuevoProducto });

});

// Eliminar un producto

router.delete('/:id', async (req, res) => {

await Producto.findByIdAndDelete(req.params.id);

res.json({ mensaje: 'Producto eliminado' });

});

module.exports = router;

En **server.js**, importa y usa las rutas:

const productosRoutes = require('./routes/productos');

app.use('/productos', productosRoutes);

**7. Crear el frontend con Fetch API**

Crea un archivo index.html:

<!DOCTYPE html>

<html>

<head>

<title>Gestión de Almacén</title>

</head>

<body>

<h1>Lista de Productos</h1>

<button onclick="cargarProductos()">Cargar Productos</button>

<ul id="lista-productos"></ul>

<h2>Agregar Producto</h2>

<input type="text" id="nombre" placeholder="Nombre">

<input type="number" id="precio" placeholder="Precio">

<input type="number" id="stock" placeholder="Stock">

<button onclick="agregarProducto()">Agregar</button>

<script>

async function cargarProductos() {

const res = await fetch('http://localhost:3000/productos');

const productos = await res.json();

document.getElementById('lista-productos').innerHTML = productos.map(p =>

`<li>${p.nombre} - ${p.precio}€ - ${p.stock} unidades</li>`

).join('');

}

async function agregarProducto() {

const nombre = document.getElementById('nombre').value;

const precio = document.getElementById('precio').value;

const stock = document.getElementById('stock').value;

await fetch('http://localhost:3000/productos', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ nombre, precio, stock })

});

cargarProductos();

}

</script>

</body>

</html>

**8. Conclusión**

¡Felicidades! Has creado una aplicación web completa con una API REST y MongoDB. Puedes mejorarla agregando autenticación, interfaces más avanzadas o desplegándola en la nube.