


CRUD / Express y MongoDB - Parte 3

En esta tercera parte terminamos de agregar las funciones de **mostrarUnCliente**, **eliminarClientes** y **modificarClientes** para nuestro archivo **controllers/ClienteController.js**

controller/ClienteController.js

 Copiar código

```
// Exportamos nuestro modelo
const Cliente = require('../models/Cliente');

// Creamos una función para agregar clientes
exports.agregarClientes = async(req, res) => {
  try {
    let clientes = new Cliente(req.body)
    await clientes.save();
    res.send(clientes);

  } catch (error) {
    console.log(error)
    res.status(500).send('hubo un error al agregar un cliente 😞')
  }
}

// Creamos la función para mostrar clientes
exports.mostrarClientes = async (req, res) => {
  try {
    const clientes = await Cliente.find();
    res.json(clientes);

  } catch (error) {
    console.log(error)
    res.status(500).send('hubo un error al mostrar un cliente 😞')
  }
}

//Session27
// Creamos la función para mostrar un solo cliente
exports.mostrarUnCliente = async(req, res) =>{
  try {
    let clientes = await Cliente.findById(req.params.id);
    if(!clientes){
      res.status(404).json({msg: "No se encuentra cliente con ese ID"})
    }

    res.send(clientes);
  } catch (error) {
    console.log(error)
    res.status(500).send('hubo un error al buscar un cliente 😞');
  }
}
```

```

    }
  }
  // Creamos la función para eliminar clientes
  exports.eliminarClientes = async(req, res) =>{
    try {
      let clientes = await Cliente.findById(req.params.id);

      if(!clientes){
        res.status(404).json({msg: "El cliente no existe"});
        return
      }

      await Cliente.findOneAndDelete({_id: req.params.id});
      res.json({msg: "El cliente fue eliminado"});


    } catch (error) {
      console.log(error)
      res.status(500).send('hubo un error al eliminar un cliente 😞')
    }
  }
  // Creamos la función para modificar un cliente
  exports.modificarClientes = async(req, res) =>{
    try {
      let cliente = await Cliente.findByIdAndUpdate(req.params.id, req.body, {new: true});

      if(!cliente){
        return res.status(404).send("Cliente no encontrado");
      }
      res.json(cliente)
    } catch (error) {
      console.log(error)
      res.status(500).send('hubo un error al modificar un cliente 😞');
    }
  }
}

```

Nuestro archivo **routes/RouterCliente.js** con las nuevas rutas quedaría configurado de esta manera:

routes/RouterCliente.js

 Copiar código

```

const express = require('express');
const router = express.Router();
const ClienteController = require('../controllers/ClienteController');

router.post('/', ClienteController.agregarClientes);
router.get('/', ClienteController.mostrarClientes);

// Sesión 27
router.get('/:id', ClienteController.mostrarUnCliente);
router.delete('/:id', ClienteController.eliminarClientes);
router.put('/:id', ClienteController.modificarClientes);

module.exports = router;

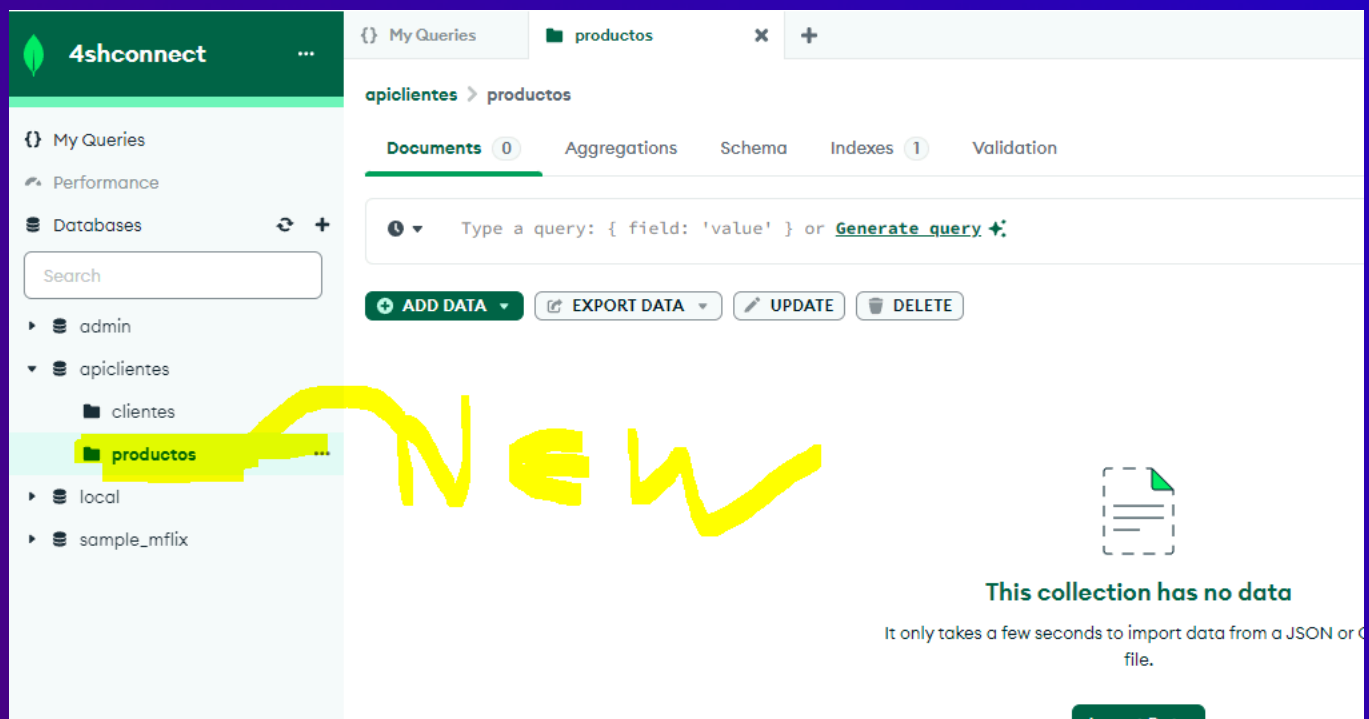
```

Y de esa manera queda nuestro backend para esta entrega a continuación se integra un nuevo modulo llamado **Productos** donde pondremos a prueba todo lo anterior visto usando **Postam**

Actividad Añadiendo nuevo modulo


Crearemos un nuevo modulo llamado 'Productos' donde se usaran las mismas funciones para ingresar, mostrar, eliminar y modificar datos de nuestra base de datos.

Primero creamos una colección de datos en nuestro Compass llamada 'Productos'



Ahora en nuestro proyecto de node lo único que vamos a necesitar es crear tres archivos nuevos:


js

 Copiar código ^{M2}

```
↓ config
  db.js
↓ controllers
  ClienteController.js
  ProductosController.js // New
↓ models
  Cliente.js
  Productos.js // New
→ node_modules
↓ routes
  RouterCliente.js
  RouterProductos.js // New
→ src
  .env
  package-lock.json
  package.json
```

La configuración de los archivos sería igual a los anteriores usados para 'Clientes', solo se modifican algunas rutas y nombres de variables

controller/ProductosController.js

 Copiar código

```
// Exportamos nuestro modelo
const Productos = require('../models/Productos');

// Creamos una función para agregar productos
exports.agregarProductos = async(req, res) => {
  try {
    let productos = new Productos(req.body)
    await productos.save();
    res.send(productos);

  } catch (error) {
    console.log(error)
    res.status(500).send('hubo un error al agregar un producto 😞')
  }
}

// Creamos la función para mostrar clientes
exports.mostrarProductos = async (req, res) => {
  try {
    const productos = await Productos.find();
    res.json(productos);

  } catch (error) {
    console.log(error)
    res.status(500).send('hubo un error al mostrar un cliente 😞')
  }
}
```

```
// Creamos la función para mostrar un solo producto
exports.mostrarUnProducto = async(req, res) =>{
  try {
    let productos = await Productos.findById(req.params.id);
    if(!productos){
      res.status(404).send("No se encuentra el producto con ese ID")
    }
    res.send(productos);

  } catch (error) {
    console.log(error)
    res.status(500).send('hubo un error al buscar un producto 😞');
  }
}

// Creamos la función para eliminar productos
exports.eliminarProductos = async(req, res) =>{
  try {
    let productos = await Productos.findById(req.params.id);

    if(!productos){
      res.status(404).json({msg: "El producto no existe"});
      return
    }

    await Productos.findOneAndDelete({_id: req.params.id});
    res.json({msg: "El producto fue eliminado"});

  } catch (error) {
    console.log(error)
    res.status(500).send('hubo un error al eliminar un producto 😞')
  }
}

// Creamos la función para modificar un producto
exports.modificarProductos = async(req, res) =>{
  try {
    let productos = await Productos.findByIdAndUpdate(req.params.id, req.body, {new: true});

    if(!productos){
      return res.status(404).send("Producto no encontrado");
    }
    res.json(productos)
  } catch (error) {
    console.log(error)
    res.status(500).send('hubo un error al modificar un producto 😞');
  }
}
```

NOTA

Este código define funciones para agregar, mostrar, buscar, eliminar y modificar productos en una base de datos MongoDB con Express. Utiliza el modelo Productos para realizar operaciones como agregar nuevos productos con `save()`, mostrar todos con `find()`, buscar por ID con `findById()`, eliminar con `findOneAndDelete()`, y modificar con `findByIdAndUpdate()`. Cada función maneja errores, enviando respuestas adecuadas como mensajes 404 o 500 según corresponda.

```
const mongoose = require ('mongoose');

const productosSchema = mongoose.Schema({

  marca: {
    type: String,
    required: true
  },
  categoria: {
    type: String,
    required: true
  },
  proveedor: {
    type: String,
    required: true
  },
  referencia: {
    type: String,
    required: true
  },
  precio: {
    type: Number,
    required: true
  },
}, {versionkey: false});

module.exports = mongoose.model('Productos', productosSchema);
```

NOTA

El código define un esquema de mongoose para el modelo Productos. Este esquema especifica las propiedades de los productos, incluyendo marca, categoría, proveedor, referencia y precio, con sus respectivos tipos y requisitos. Al final, el esquema se exporta como un modelo de mongoose llamado Productos, listo para ser utilizado en la aplicación para interactuar con la base de datos MongoDB.

```
const express = require('express');
const router = express.Router();
const ProductosController = require('../controllers/ProductosController');

router.post('/', ProductosController.agregarProductos);
router.get('/', ProductosController.mostrarProductos);
router.get('/:id', ProductosController.mostrarUnProducto);
router.delete('/:id', ProductosController.eliminarProductos);
router.put('/:id', ProductosController.modificarProductos);


module.exports = router;
```

NOTA

En este ultimo fragmento se configura el enrutador en Express para que maneje diferentes rutas relacionadas con los productos. Utiliza el controlador ProductosController para gestionar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Las rutas definidas incluyen agregar un producto, mostrar todos los productos, mostrar un producto específico por su ID, eliminar un producto por su ID y modificar un producto existente. Al exportar este enrutador, se facilita su uso y modularidad en la aplicación principal.

Finalmente agrega la ruta de el modulo **Productos** al index

js

 Copiar código

```
// Aquí iniciaremos express
const express = require('express');

//new sesión 26
const conectarBD = require('../config/db');
const cors = require('cors');

// Configurando express y puerto
const app = express();
const port = 5000;

//new sesión 26
app.use(express.json());
// Aca van las rutas de los módulos
app.use('/api/clientes', require('../routes/RoutersCliente'));

// New sesión 27
// Creando nuevo modulo para productos
app.use('/api/productos', require('../routes/RoutersProductos'));
// New sesión 27

//enlazamos la conexión de la BD
conectarBD();
app.use(cors());

// Puerto donde se lanza el servidor
app.listen(port, () => console.log('Nuestro servidor se encuentra conectado 😊 http://localhost:5000'));

app.get('/', (req, res) =>{
  res.send('Bienvenido, nuestro servidor esta configurado');
});
```

Probando el nuevo modulo 'Productos'

17.04.2024 S27-M2

Vamos a usar **Postam** para probar cada una de la funciones, tanto de ingresar, mostrar eliminar etc.

.agregarProductos

The screenshot shows the Postman interface for a POST request to `http://localhost:5000/api/productos`. The request body is a JSON object with the following fields:

```
{
  "marca": "Totto",
  "categoria": "Gagas",
  "proveedor": "Tottosas",
  "referencia": "T0-345",
  "precio": "160000"
}
```

The response body is a JSON object with the following fields:

```
{
  "marca": "Totto",
  "categoria": "Gagas",
  "proveedor": "Tottosas",
  "referencia": "T0-345",
  "precio": 160000,
  "_id": "662096f6d7b4f0135e482dd3",
  "__v": 0
}
```

The status of the request is 200 OK, with a response time of 126 ms and a body size of 375 B. A yellow checkmark is drawn next to the response body.

.mostrarProductos

GET http://localhost:5000/api/... No environment 17.04.2024 S27-M2

http://localhost:5000/api/productos Save

GET http://localhost:5000/api/productos Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

1

Body Cookies Headers (7) Test Results 200 OK 105 ms 377 B Save as example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "_id": "662096f6d7b4f0135e482dd3",
4     "marca": "Totto",
5     "categoria": "Gagas",
6     "proveedor": "Tottosas",
7     "referencia": "T0-345",
8     "precio": 160000,
9     "__v": 0
10  }
11 ]
```

Live

.mostrarUnProducto

GET http://localhost:5000/api

No environment

http://localhost:5000/api/productos/662096f6d7b4f0135e482dd3

Save

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

Beautify

1

Body Cookies Headers (7) Test Results

200 OK 107 ms 375 B Save as example

Pretty

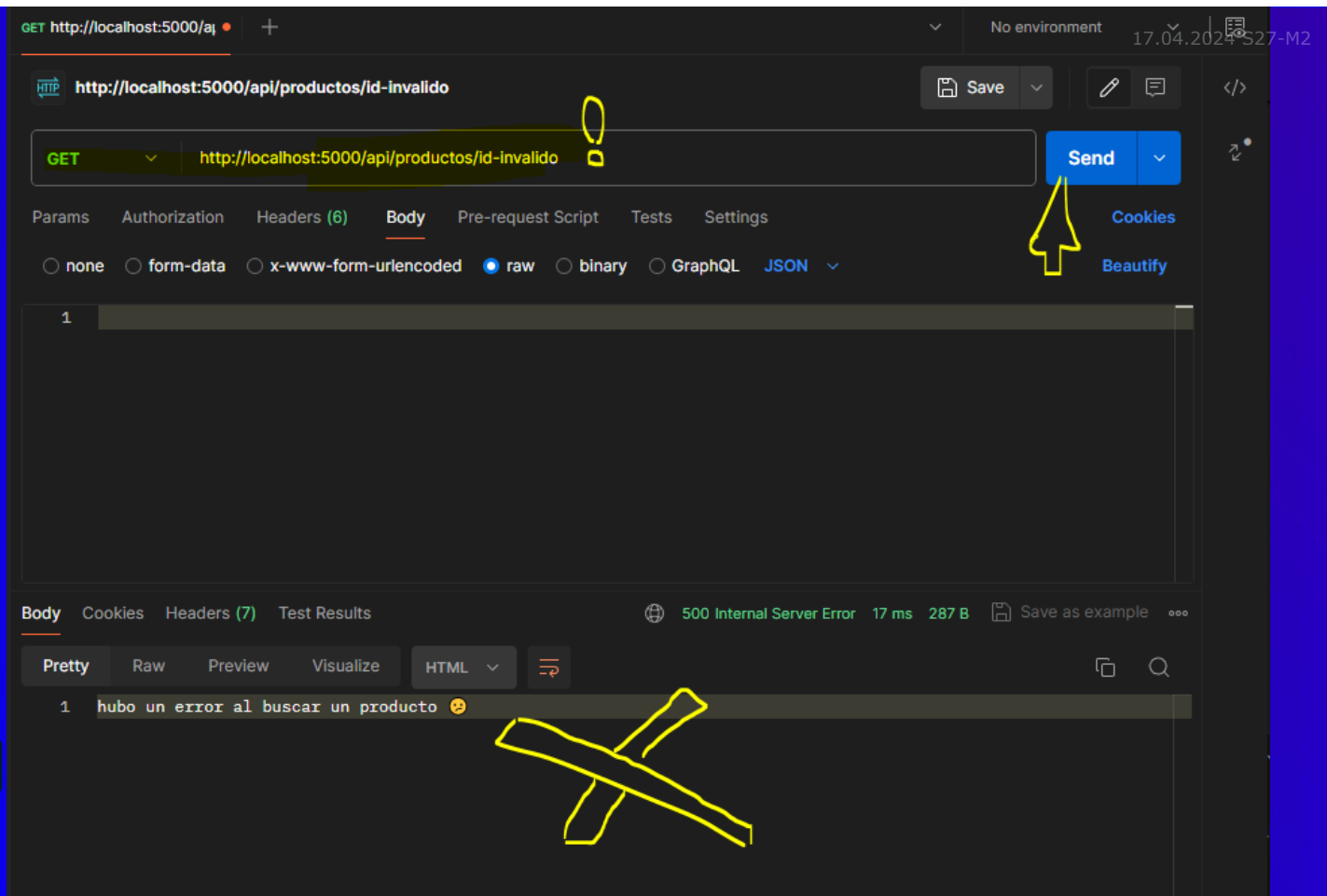
Raw

Preview

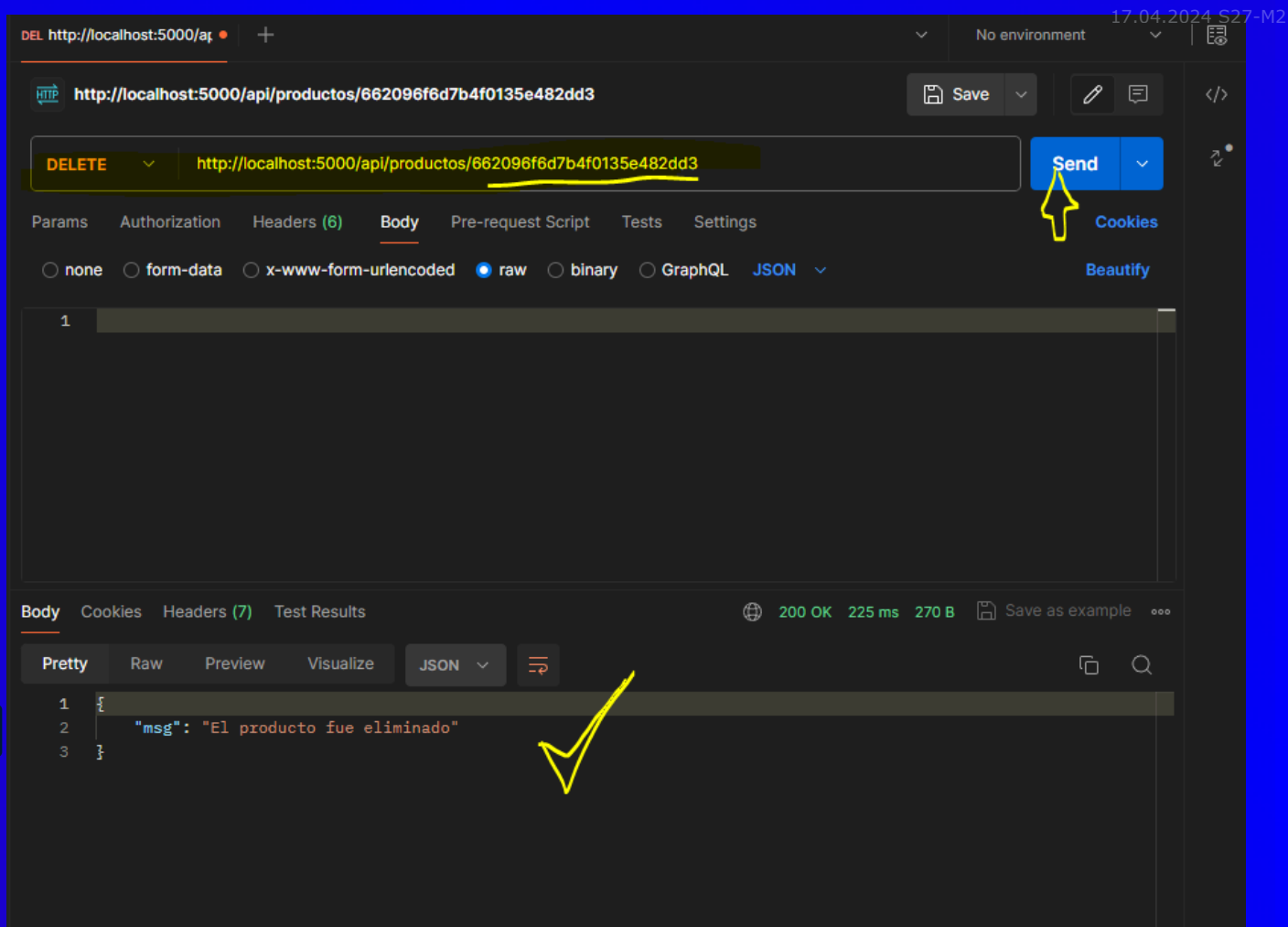
Visualize

JSON

```
1 {
2   "_id": "662096f6d7b4f0135e482dd3",
3   "marca": "Totto",
4   "categoria": "Gagas",
5   "proveedor": "Tottosas",
6   "referencia": "T0-345",
7   "precio": 160000,
8   "__v": 0
9 }
```



.eliminarProductos



.modificarProductos

PUT http://localhost:5000/api/... 17.04.2024 S27-M2

http://localhost:5000/api/productos/66209a73eb930c9d94e60baf

PUT http://localhost:5000/api/productos/66209a73eb930c9d94e60baf Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "marca": "Chloe",
3   "categoria": "Gafa",
4   "proveedor": "New2024",
5   "referencia": "chaslk",
6   "precio": "230000"
7 }
```

Body Cookies Headers (7) Test Results 200 OK 117 ms 373 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "66209a73eb930c9d94e60baf",
3   "marca": "Chloe",
4   "categoria": "Gafa",
5   "proveedor": "New2024",
6   "referencia": "chaslk",
7   "precio": 230000,
8   "__v": 0
9 }
```



Mira el repositorio en Github