

# Manual Paso a Paso - Proyecto Backend con NestJS + MongoDB

## Introducción

Este documento describe de forma detallada y técnica cómo se desarrolló un sistema backend usando **NestJS** y **MongoDB**, incluyendo la creación de un CRUD completo. La entidad de ejemplo que usaremos es **Productos**, pero el patrón aplica para todas las demás.

## Instalar el CLI de NestJS

### ¿Qué es el CLI de NestJS?

El CLI (*Command Line Interface*) de NestJS es una herramienta oficial que facilita la creación y gestión de proyectos NestJS. Con ella podrás generar módulos, controladores, servicios y mucho más de forma automática, rápida y ordenada.

**Comando:** `npm i -g @nestjs/cli`

### ¿Qué hace este comando?

- `npm`: el gestor de paquetes de Node.js.
- `i`: es la abreviación de `install`, es decir, instalar.
- `-g`: indica que se instalará **globalmente** en tu sistema (no solo en un proyecto).
- `@nestjs/cli`: es el paquete del CLI oficial de NestJS.

Con este comando estás instalando la herramienta `nest`, que podrás usar en cualquier parte de tu sistema para crear y administrar proyectos NestJS.

## Creación del Proyecto

**Comando:** `nest new proyecto-tienda`

### Qué hace:

- Crea la estructura base del proyecto con NestJS.
- Incluye carpetas como `src`, `test`, y archivos como `main.ts`, `app.module.ts`, etc.

## Creación de Recursos (Entidades CRUD)

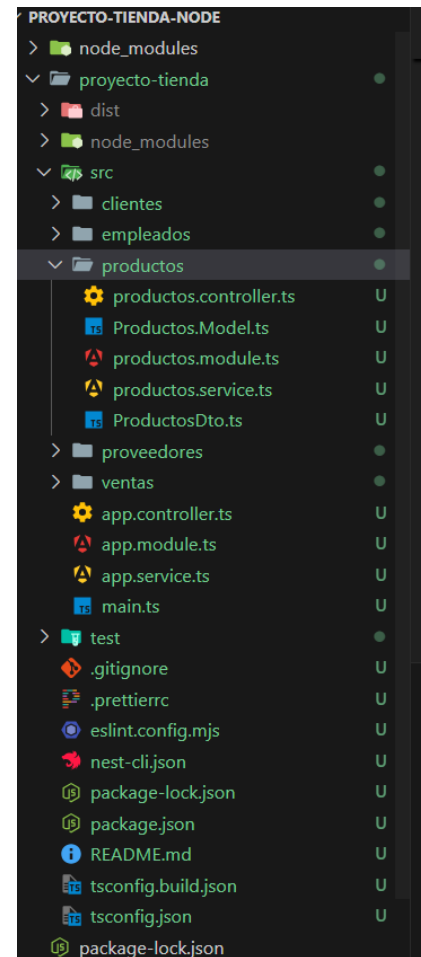
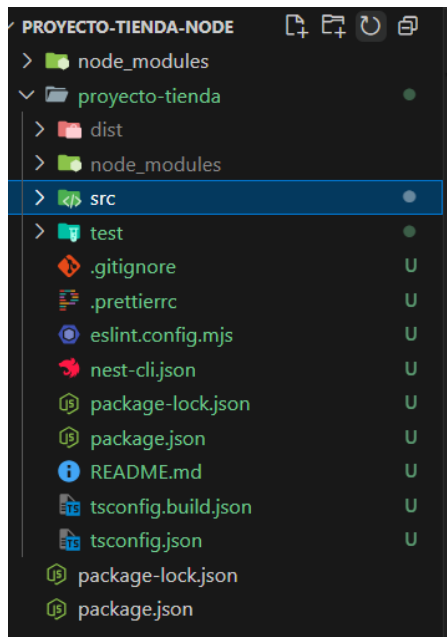
**Comandos usados:**

- nest generate resource productos
- nest generate resource clientes
- nest generate resource ventas
- nest generate resource empleados
- nest generate resource proveedores

### Qué hace:

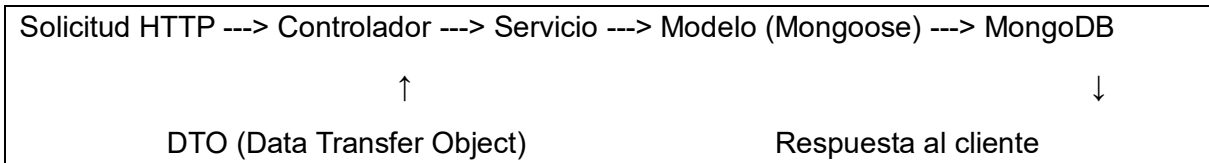
Crea para cada entidad:

- Controlador
- Servicio
- DTO
- Módulo
- Archivo de pruebas (opcional)



## Explicación General del CRUD

### Arquitectura General



## Componentes

Componente	Qué hace
DTO	Define la forma del objeto que se recibe del frontend.
Controlador	Recibe la solicitud HTTP y llama al servicio.
Servicio	Contiene la lógica para crear/leer/actualizar/eliminar.
Modelo	Define la estructura de los datos en MongoDB.
Módulo	Registra controlador, servicio y modelo dentro del ecosistema NestJS.

## Conexión a MongoDB

- Código en app.module.ts, esto permite que todos los recursos puedan inyectar modelos Mongoose y trabajar con la base de datos.

```

import { Module } from '@nestjs/common';
import { AppController } from '../app.controller';
import { AppService } from '../app.service';
import { ProductosModule } from '../productos/productos.module';
import { ProveedoresModule } from '../proveedores/proveedores.module';
import { VentasModule } from '../ventas/ventas.module';
import { ClientesModule } from '../clientes/clientes.module';
import { MongooseModule } from '@nestjs/mongoose';
import { EmpleadosModule } from '../empleados/empleados.module';

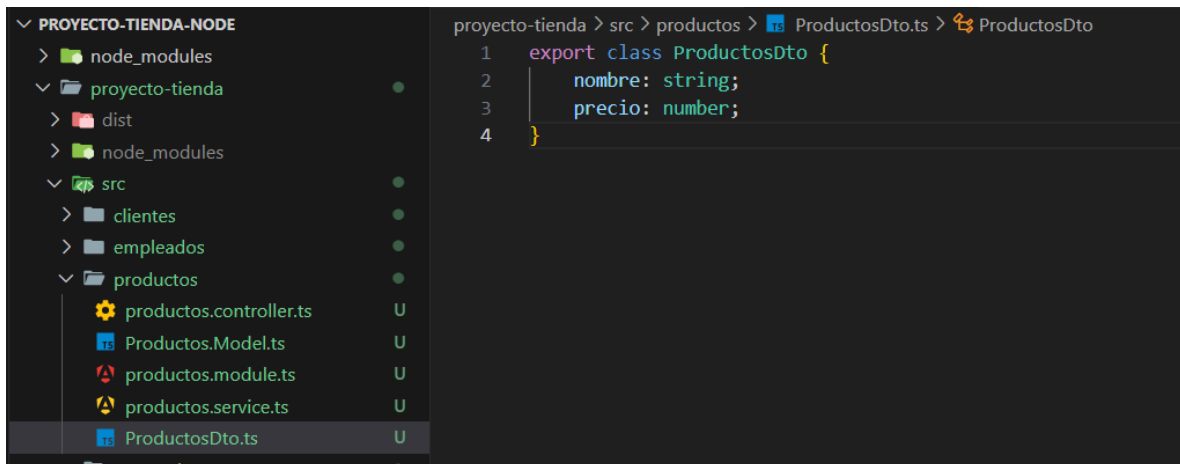
@Module({
  imports: [ProductosModule, ProveedoresModule, VentasModule, ClientesModule, MongooseModule.forRoot("mongodb+srv://Admin:Adso2932015@tiendanestjs.ujoytab.mongodb.net/"), EmpleadosModule],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}

```

## CRUD Completo de Productos (Entidad elegida)

### ProductosDto.ts

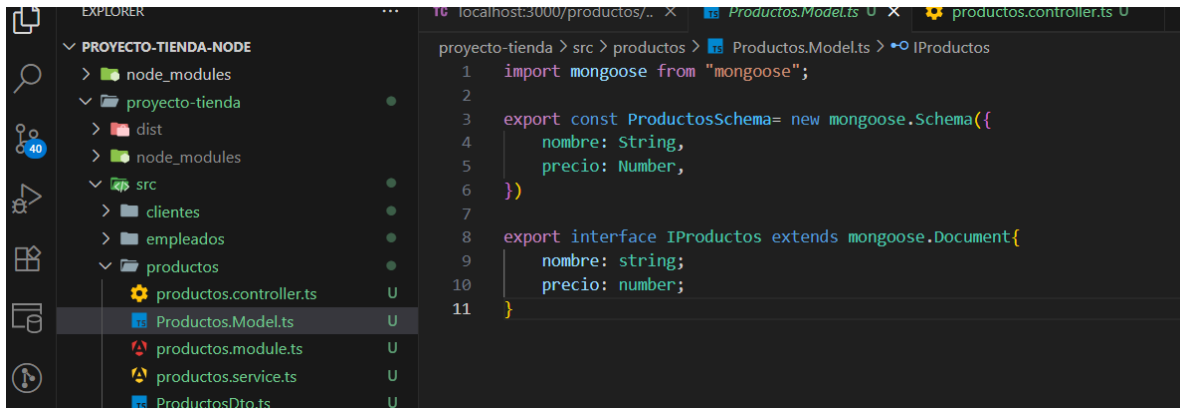
Define los datos que se pueden recibir al crear o actualizar un producto.



**ProductosDto:** Clase que representa los datos que espera recibir el backend. NestJS lo usará con `@Body()` para validar la entrada.

### Productos.Model.ts

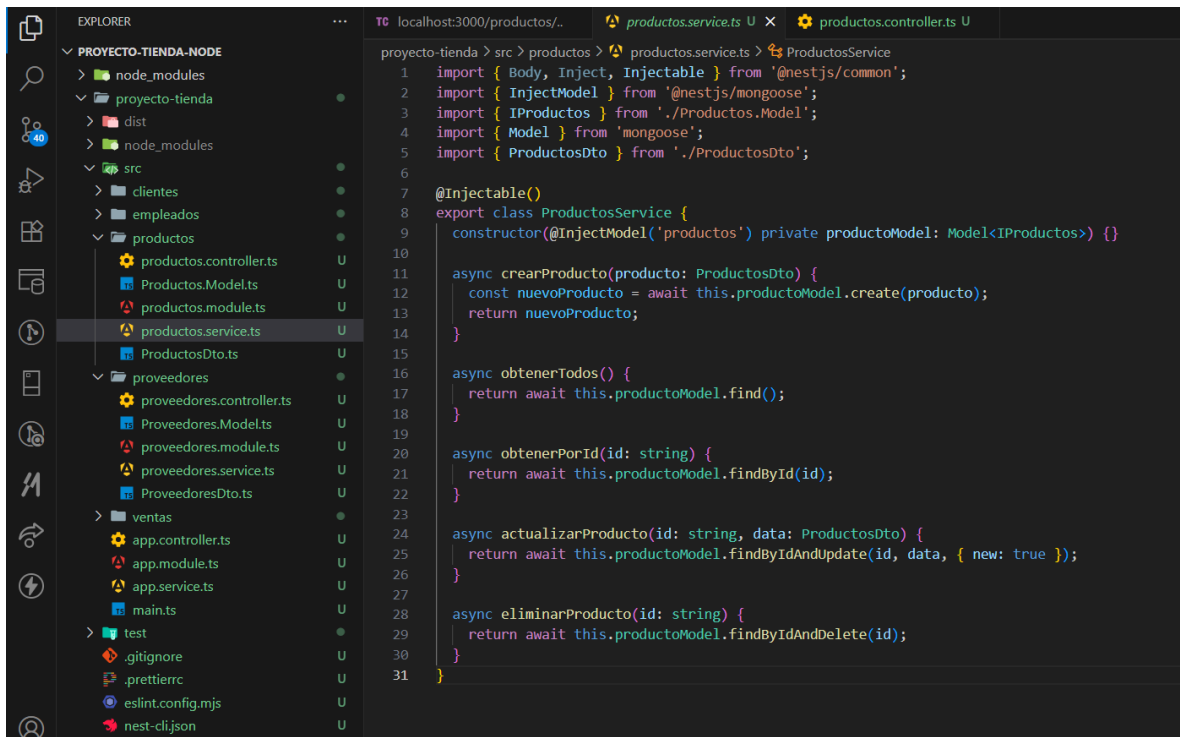
Define el esquema y la interfaz del modelo de MongoDB.



- ProductosSchema: Define cómo se almacena el documento en MongoDB.
- IProductos: Interfaz que extiende Document de Mongoose, usada para tipar los datos en NestJS.

## productos.service.ts

Maneja la lógica de negocio para productos.

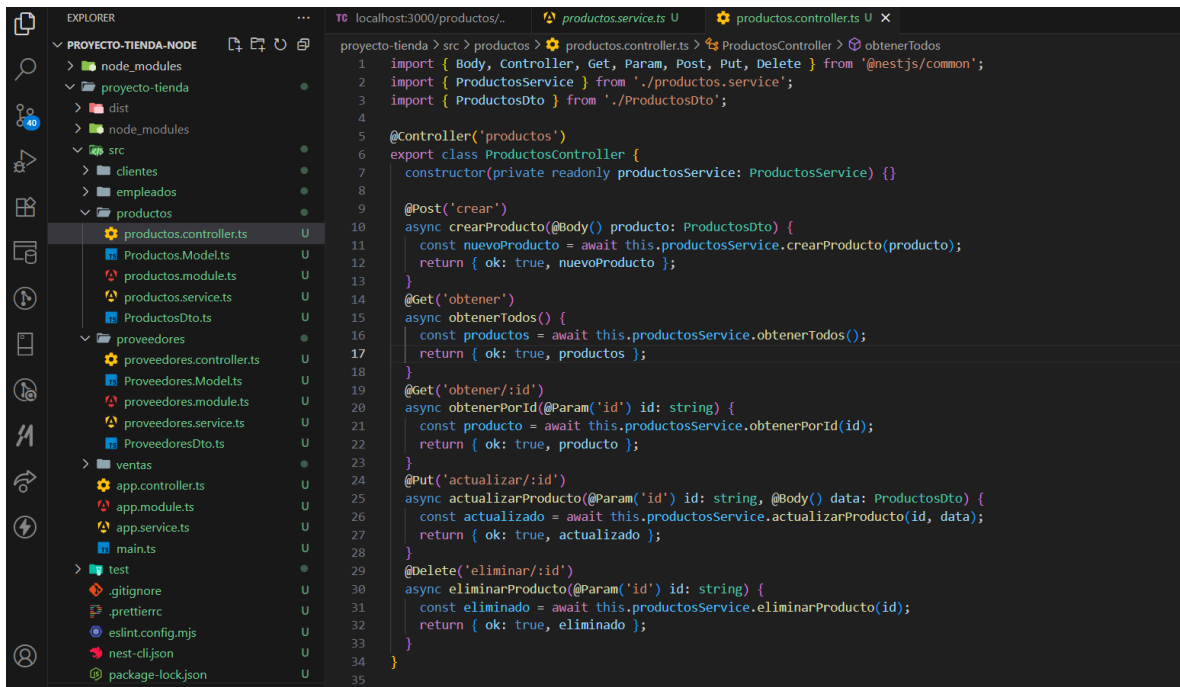


- @Injectable() indica que este servicio se puede inyectar.
- @InjectModel('productos') permite inyectar el modelo de Mongoose asociado.

- crearProducto() guarda un producto nuevo.

## productos.controller.ts

Recibe las peticiones HTTP y las pasa al servicio.



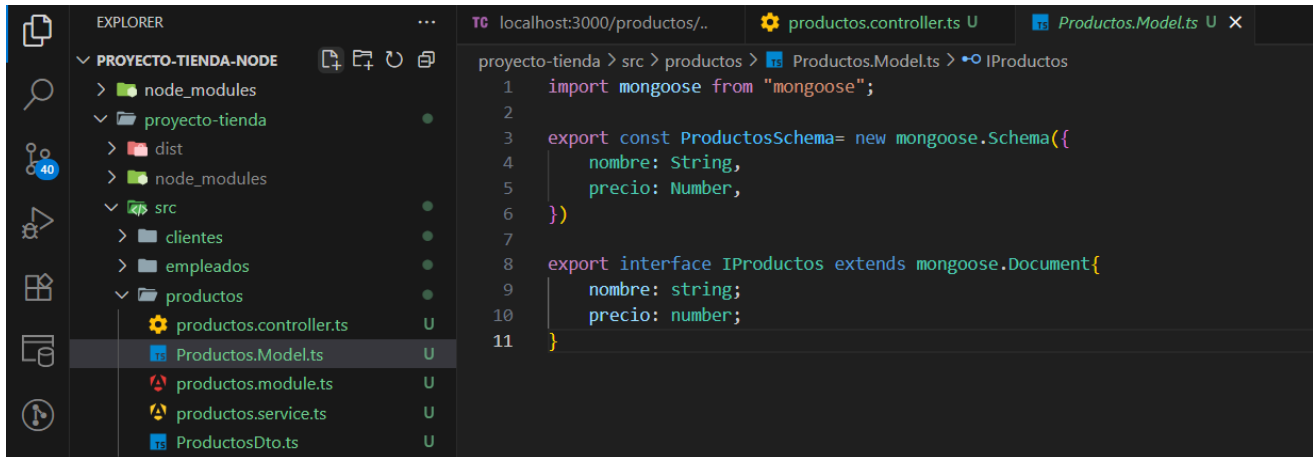
```

1 import { Body, Controller, Get, Param, Post, Put, Delete } from '@nestjs/common';
2 import { ProductosService } from '../productos.service';
3 import { ProductosDto } from '../ProductosDto';
4
5 @Controller('productos')
6 export class ProductosController {
7   constructor(private readonly productosService: ProductosService) {}
8
9   @Post('crear')
10  async crearProducto(@Body() producto: ProductosDto) {
11    const nuevoProducto = await this.productosService.crearProducto(producto);
12    return { ok: true, nuevoProducto };
13  }
14
15  @Get('obtener')
16  async obtenerTodos() {
17    const productos = await this.productosService.obtenerTodos();
18    return { ok: true, productos };
19  }
20
21  @Get('obtener/:id')
22  async obtenerPorId(@Param('id') id: string) {
23    const producto = await this.productosService.obtenerPorId(id);
24    return { ok: true, producto };
25  }
26
27  @Put('actualizar/:id')
28  async actualizarProducto(@Param('id') id: string, @Body() data: ProductosDto) {
29    const actualizado = await this.productosService.actualizarProducto(id, data);
30    return { ok: true, actualizado };
31  }
32
33  @Delete('eliminar/:id')
34  async eliminarProducto(@Param('id') id: string) {
35    const eliminado = await this.productosService.eliminarProducto(id);
36    return { ok: true, eliminado };
37  }
38 }
  
```

- @Controller('productos'): Define la ruta base /productos.
- @Post('crear'): Ruta POST /productos/crear.
- @Body(): Extrae el body de la petición.

## productos.module.ts

Agrupar todo lo relacionado a productos.



**MongooseModule.forFeature** asocia el nombre 'productos' con el esquema ProductosSchema.

## Flujo Técnico del CRUD

1. **Petición HTTP** llega al **@Controller** → se enruta según el verbo y la ruta (@Post('crear')).
2. El **Controlador** recibe los datos (en el body) y llama al **Servicio**.
3. El **Servicio** usa **@InjectModel()** para acceder a MongoDB a través de Mongoose.
4. El documento se guarda y se retorna al controlador.
5. El **Controlador** responde al cliente con un JSON que contiene el resultado.

## Registro producto en la base de datos

The screenshot shows a REST client interface in VS Code. The request is a POST to `localhost:3000/productos/crear` with a body encoded as form data. The response is a 201 Created status with a JSON body indicating successful creation of a product.

**Request:**

- Method: POST
- URL: `localhost:3000/productos/crear`
- Body (Form-encoded):
  - ☒ nombre: Leche
  - ☒ precio: 3000
  - ☐ name: value

**Response:**

```
1 {
2   "ok": true,
3   "nuevoProducto": {
4     "nombre": "Leche",
5     "precio": 3000,
6     "_id": "6852e00a9c108b2a7ea2f699",
7     "__v": 0
8   }
9 }
```

The screenshot shows the MongoDB Compass interface. The left sidebar lists collections: `sample_mflix`, `test`, `clientes`, `empleados`, `productos` (selected), `proveedores`, and `ventas`. The main panel shows the query results for the `productos` collection, displaying two documents.

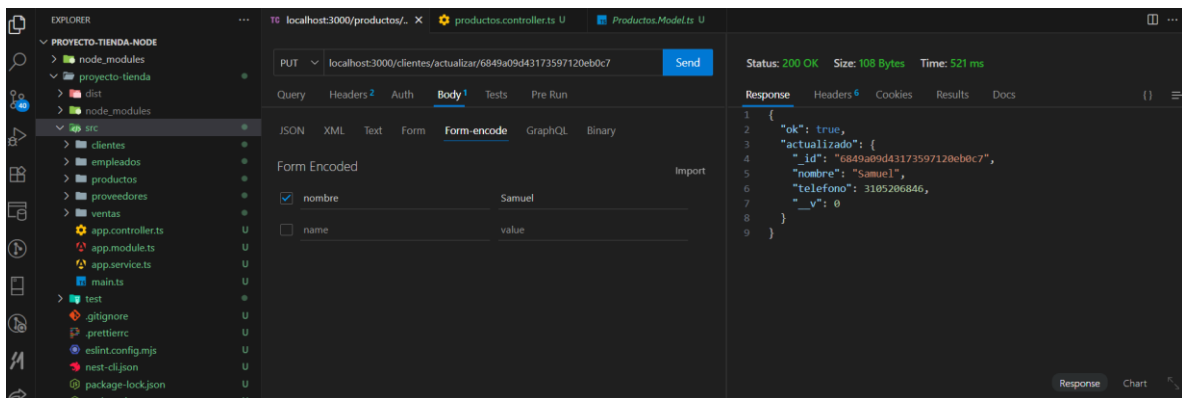
**QUERY RESULTS: 1-2 OF 2**

```
1 {
2   "_id": ObjectId('684997b868c36d8948b10072'),
3   "nombre": "arroz huila",
4   "precio": 5000,
5   "__v": 0
6 }
7
8 {
9   "_id": ObjectId('6852e00a9c108b2a7ea2f699'),
10  "nombre": "Leche",
11  "precio": 3000,
12  "__v": 0
13 }
```

System Status: All Good



## Actualizar clientes en la base de datos



VS Code interface showing a REST client PUT request to `localhost:3000/clientes/actualizar/6849a09d43173597120eb0c7`. The request body is form-encoded with fields `nombre` (Samuel) and `telefono` (3105206846). The response is a 200 OK status with a JSON body indicating the update was successful.

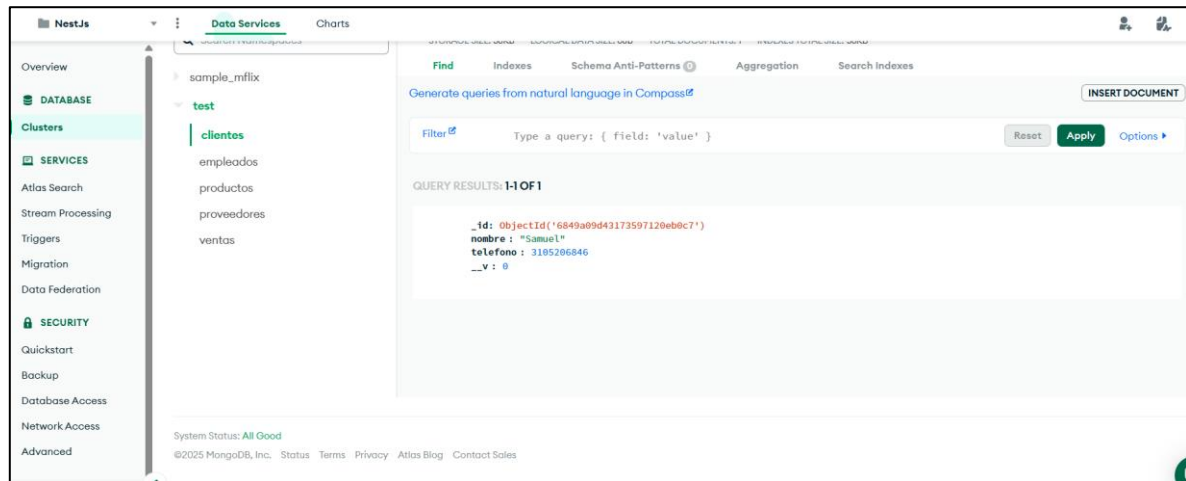
```
PUT localhost:3000/clientes/actualizar/6849a09d43173597120eb0c7
```

Form Encoded

<input checked="" type="checkbox"/>	nombre	Samuel
<input type="checkbox"/>	telefono	3105206846

Response

```
{
  "ok": true,
  "actualizado": {
    "_id": "6849a09d43173597120eb0c7",
    "nombre": "Samuel",
    "telefono": 3105206846,
    "__v": 0
  }
}
```



MongoDB Compass interface showing the 'clientes' collection in the 'test' database. The query results show a single document with fields `_id`, `nombre`, `telefono`, and `__v`.

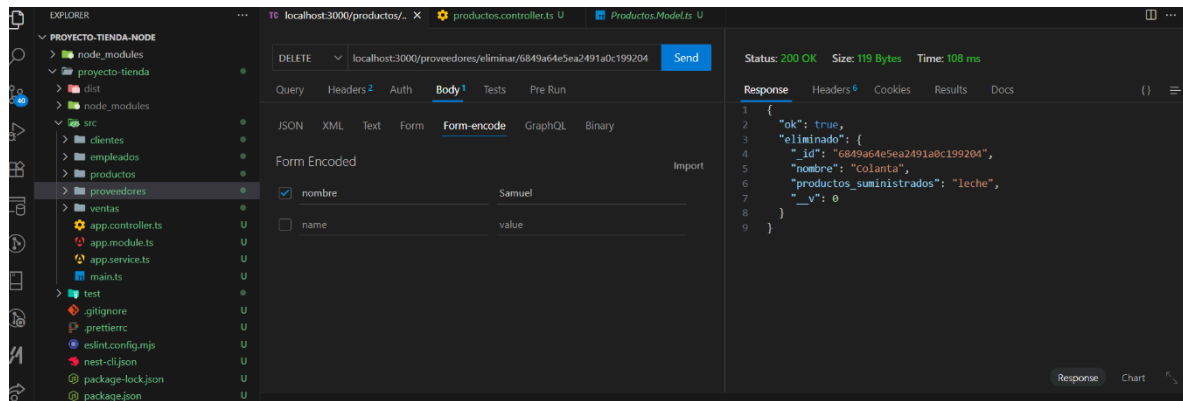
Find

Filter: Type a query: { field: 'value' }

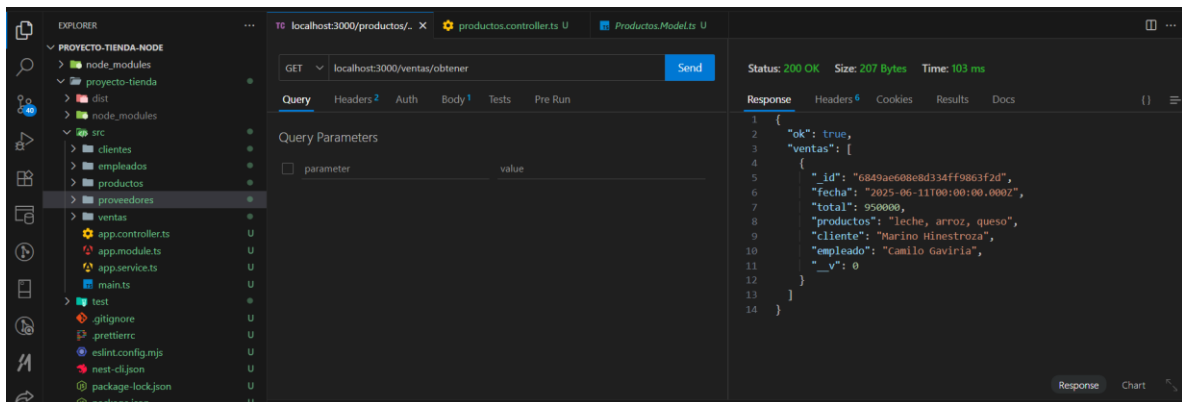
QUERY RESULTS: 1-1 OF 1

```
{
  "_id": ObjectId("6849a09d43173597120eb0c7"),
  "nombre": "Samuel",
  "telefono": 3105206846,
  "__v": 0
}
```

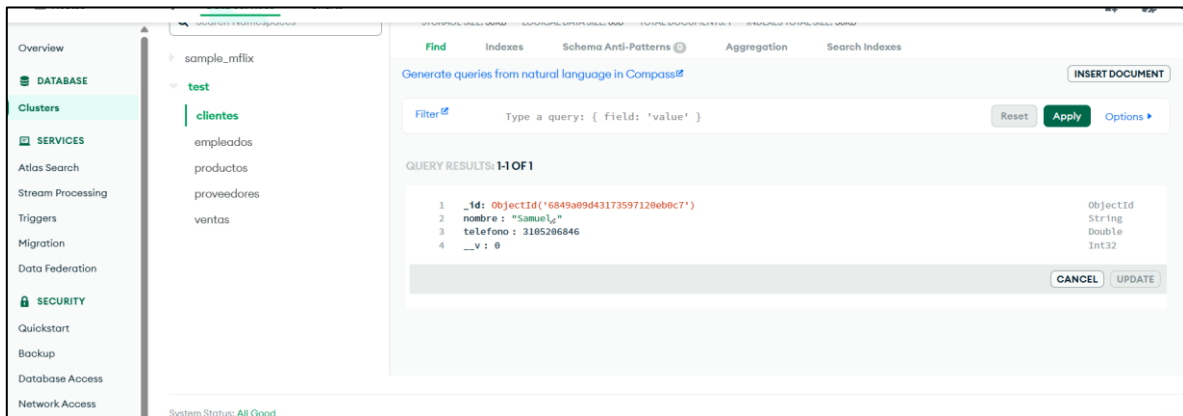
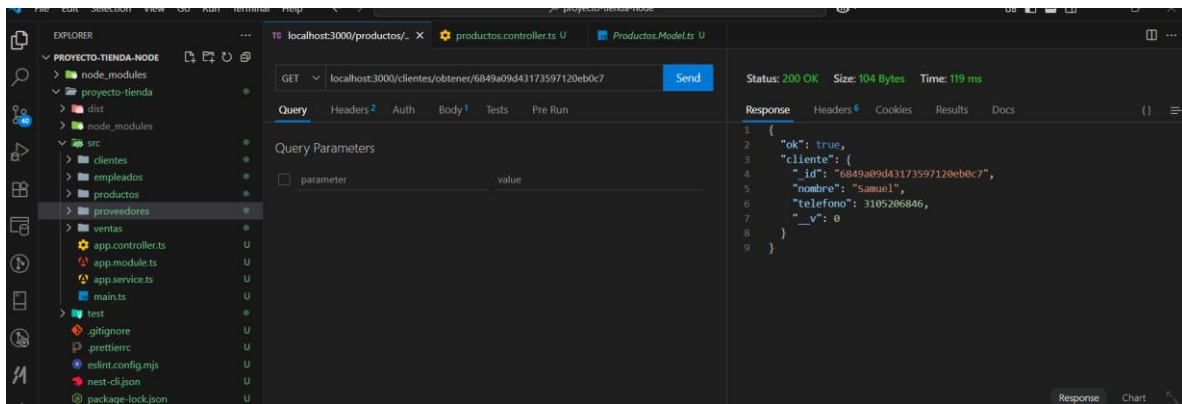
## Eliminar Proveedores de la base de datos



## Obtener Ventas de la base de datos



## Obtener Clientes por ID de la base datos



## Conclusión

Con esta guía se puede crear un backend modular y escalable usando NestJS y MongoDB. El patrón de dividir lógica en controladores, servicios y módulos facilita la mantenibilidad del código. Con una sola estructura clara se pueden escalar fácilmente otras entidades como categorías, pedidos, etc.