

Pruebas del Backend en Postman

Este documento contiene las pruebas realizadas en Postman para verificar el correcto funcionamiento del backend.

Se han ejecutado distintas operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre las entidades Usuarios y Productos,

además de las pruebas para los contadores de registros y el monitoreo de operaciones realizadas.

Cada prueba incluye:

- ✓ El método HTTP utilizado (GET, POST, PUT, DELETE).
- ✓ La URL de la petición.
- ✓ El cuerpo de la petición (si llega a ser necesario).
- ✓ El resultado esperado.
- ✓ Una captura de pantalla de la prueba exitosa.

Estas pruebas validan que el backend funciona correctamente y cumple con los requerimientos del proyecto.

Prueba: Obtener todos los usuarios (GET /usuarios)

- ◆ Método: `GET`
- ◆ URL: `http://localhost:5000/usuarios`

Resultado esperado:

- Devuelve un array con todos los usuarios en la base de datos.

The screenshot shows a Postman request and its response. The request is a GET to `http://localhost:5000/usuarios`. The response is a 200 OK status with 8 ms latency and 445 B size. The response body is a JSON array containing two user objects:

```
1 [  
2 {  
3   "_id": "67c7cdd0a8e2dbe84d1039a1",  
4   "nombre": "Alejandro",  
5   "edad": 27,  
6   "ocupacion": "Ingeniero"  
7 },  
8 {  
9   "_id": "67c7cdd0a8e2dbe84d1039a2",  
10  "nombre": "Ana",  
11  "edad": 25,  
12  "ocupacion": "Diseñadora"  
13 }]  
14 ]
```

Prueba: Crear un usuario (POST /usuarios)

- ◆ Método: `POST`
- ◆ URL: `http://localhost:5000/usuarios`
- ◆ raw
- ◆ Body (JSON):

```
{  
  "nombre": "Chelo",  
  "edad": 19,  
  "ocupacion": "Ingeniero"  
}
```

Resultado esperado:

- Devuelve el usuario creado con su `_id` generado por MongoDB.
- Comprobamos con un GET de usuarios como en el 1er paso.

The screenshot shows the Postman interface with the following details:

- Request URL:** `http://localhost:5000/usuarios`
- Method:** POST
- Body Content:**

```
1 {  
2   "nombre": "Chelo",  
3   "edad": 19,  
4   "ocupacion": "Ingeniero"  
5 }  
6
```
- Response Status:** 201 Created
- Response Headers:** 163 ms, 365 B
- Response Body:**

```
1 {  
2   "nombre": "Chelo",  
3   "edad": 19,  
4   "ocupacion": "Ingeniero",  
5   "_id": "67d24d4b962247acc20b78e6",  
6   "__v": 0  
7 }
```

The screenshot shows a Postman interface with the following details:

- Request URL:** GET <http://localhost:5000/usuarios>
- Response Headers:** No environment
- Response Status:** 200 OK
- Response Time:** 6 ms
- Response Size:** 540 B
- Response Body:**

```

1  [
2    {
3      "_id": "67c7cdd0a8e2dbe84d1039a1",
4      "nombre": "Alejandro",
5      "edad": 27,
6      "ocupacion": "Ingeniero"
7    },
8    {
9      "_id": "67c7cdd0a8e2dbe84d1039a2",
10     "nombre": "Ana",
11     "edad": 25,
12     "ocupacion": "Diseñadora"
13   },
14   {
15     "_id": "67d24d4b962247acc20b78e6",
16     "nombre": "Chelo",
17     "edad": 19,
18     "ocupacion": "Ingeniero",
19     "__v": 0
20   }
21 ]

```

Prueba: Actualizar un usuario (PUT /usuarios/:id)

- ◆ Método: PUT
- ◆ URL: http://localhost:5000/usuarios/{ID_DEL_USUARIO}
(Reemplaza {ID_DEL_USUARIO} con un _id válido de la base de datos.)
- ◆ Body (JSON):

```
{
  "nombre": "Chelo ",
  "edad": 20
}
```

Resultado esperado:

- Devuelve el usuario actualizado con los nuevos datos.
- Comprobamos con un GET de usuarios como en el 1er paso.

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `PUT http://localhost:5000/u: ●`, a plus icon for creating new requests, and a dropdown for 'No environment'. Below the header, the main interface has a toolbar with 'HTTP' and the URL `http://localhost:5000/usuarios/67d24d4b962247acc20b78e6`. To the right of the URL are 'Save' and 'Share' buttons. The main workspace contains a 'PUT' method selector, the target URL, and a 'Send' button. Below these are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Scripts', and 'Settings'. The 'Body' tab is selected and shows the raw JSON payload:

```
1 {  
2   "nombre": "Chelo ",  
3   "edad": 20  
4 }  
5
```

Below the body, there are tabs for 'Cookies', 'Beautify', and 'Test Results'. The 'Test Results' tab is currently active and displays the response details: `200 OK`, `21 ms`, `361 B`, and a progress bar indicating the response content. The response body is shown in JSON format:

```
1 {  
2   "_id": "67d24d4b962247acc20b78e6",  
3   "nombre": "Chelo ",  
4   "edad": 20,  
5   "ocupacion": "Ingeniero",  
6   "__v": 0  
7 }
```

The screenshot shows the Postman interface with a successful HTTP request. The URL is `http://localhost:5000/usuarios`. The response status is `200 OK` with a response time of `5 ms` and a size of `541 B`. The response body is a JSON array containing three user documents:

```
1 [  
2 {  
3   "_id": "67c7cdd0a8e2dbe84d1039a1",  
4   "nombre": "Alejandro",  
5   "edad": 27,  
6   "ocupacion": "Ingeniero"  
7 },  
8 {  
9   "_id": "67c7cdd0a8e2dbe84d1039a2",  
10  "nombre": "Ana",  
11  "edad": 25,  
12  "ocupacion": "Diseñadora"  
13 },  
14 {  
15  "_id": "67d24d4b962247acc20b78e6",  
16  "nombre": "Chelo ",  
17  "edad": 20,  
18  "ocupacion": "Ingeniero",  
19  "__v": 0  
20 }  
21 ]
```

Prueba: Eliminar un usuario (DELETE /usuarios/:id)

- ◆ Método: DELETE
- ◆ URL: `http://localhost:5000/usuarios/{ID_DEL_USUARIO}`
(Reemplaza `{ID_DEL_USUARIO}` con un `_id` válido de la base de datos.)

Resultado esperado:

- Devuelve un mensaje de confirmación indicando que el usuario fue eliminado.
- Comprobamos con un GET de usuarios como en el 1er paso.

DELETE http://localhost:5000/usuarios/67d24d4b962247acc20b78e6

Body (6) Headers Body Scripts Settings Cookies

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

This request does not have a body

Body Cookies Headers (8) Test Results

200 OK 5 ms 312 B

```
{ } JSON > Preview ⚡ Visualize
```

```
1 {
2   "message": "Usuario eliminado correctamente"
3 }
```

GET http://localhost:5000/usuarios

GET http://localhost:5000/usuarios

Params Authorization Headers (6) Body Scripts Settings Cookies

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

This request does not have a body

Body Cookies Headers (8) Test Results

200 OK 5 ms 445 B

```
{ } JSON > Preview ⚡ Visualize
```

```
1 [
2   {
3     "_id": "67c7cdd0a8e2dbe84d1039a1",
4     "nombre": "Alejandro",
5     "edad": 27,
6     "ocupacion": "Ingeniero"
7   },
8   {
9     "_id": "67c7cdd0a8e2dbe84d1039a2",
10    "nombre": "Ana",
11    "edad": 25,
12    "ocupacion": "Diseñadora"
13  }
14 ]
```

Prueba: Obtener todos los productos (GET /productos)

- ◆ Método: GET
- ◆ URL: http://localhost:5000/productos

Resultado esperado:

- Devuelve un array con todos los productos en la base de datos.

The screenshot shows the Postman interface with a successful GET request to `http://localhost:5000/productos`. The response status is `200 OK` with a response time of `10 ms` and a size of `449 B`. The response body is displayed as JSON:

```

1  [
2   {
3     "_id": "67c7ce0ca8e2dbe84d1039a3",
4     "nombre": "Laptop",
5     "precio": 1200,
6     "categoria": "Tecnología"
7   },
8   {
9     "_id": "67c7ce0ca8e2dbe84d1039a4",
10    "nombre": "Silla",
11    "precio": 150,
12    "categoria": "Muebles"
13 }
14 ]

```

Prueba: Crear un producto (POST /productos)

- ◆ **Método:** POST
- ◆ **URL:** <http://localhost:5000/productos>
- ◆ **Body (JSON):**

```
{
  "nombre": "Pc",
  "precio": 1000,
  "categoria": "Electronica"
}
```

Resultado esperado:

- Devuelve el producto creado con su `_id` generado por MongoDB.
- Comprobamos con un GET de usuarios como en el 1er paso.

POST http://localhost:5000/ +

HTTP http://localhost:5000/productos

Save Share

POST http://localhost:5000/productos

Send

Params Authorization Headers (8) Body Scripts Settings Cookies

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

```
1 {  
2   "nombre": "Pc",  
3   "precio": 1000,  
4   "categoria": "Electronica"  
5 }  
6
```

Body Cookies Headers (8) Test Results

201 Created 5 ms 368 B

{ } JSON Preview Visualize

```
1 {  
2   "nombre": "Pc",  
3   "precio": 1000,  
4   "categoria": "Electronica",  
5   "_id": "67d24ff9962247acc20b78f0",  
6   "__v": 0  
7 }
```

GET http://localhost:5000/ +

HTTP http://localhost:5000/productos

Save Share

GET http://localhost:5000/productos

Send

Params Authorization Headers (6) Body Scripts Settings Cookies

Body none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (8) Test Results

200 OK 10 ms 547 B

{ } JSON Preview Visualize

```
1 [  
2   {  
3     "_id": "67c7ce0ca8e2dbe84d1039a3",  
4     "nombre": "Laptop",  
5     "precio": 1200,  
6     "categoria": "Tecnologia"  
7   },  
8   {  
9     "_id": "67c7ce0ca8e2dbe84d1039a4",  
10    "nombre": "Silla",  
11    "precio": 150,  
12    "categoria": "Muebles"  
13  },  
14  {  
15    "_id": "67d24ff9962247acc20b78f0",  
16    "nombre": "Pc",  
17    "precio": 1000,  
18    "categoria": "Electronica",  
19    "__v": 0  
20  }  
21 ]
```

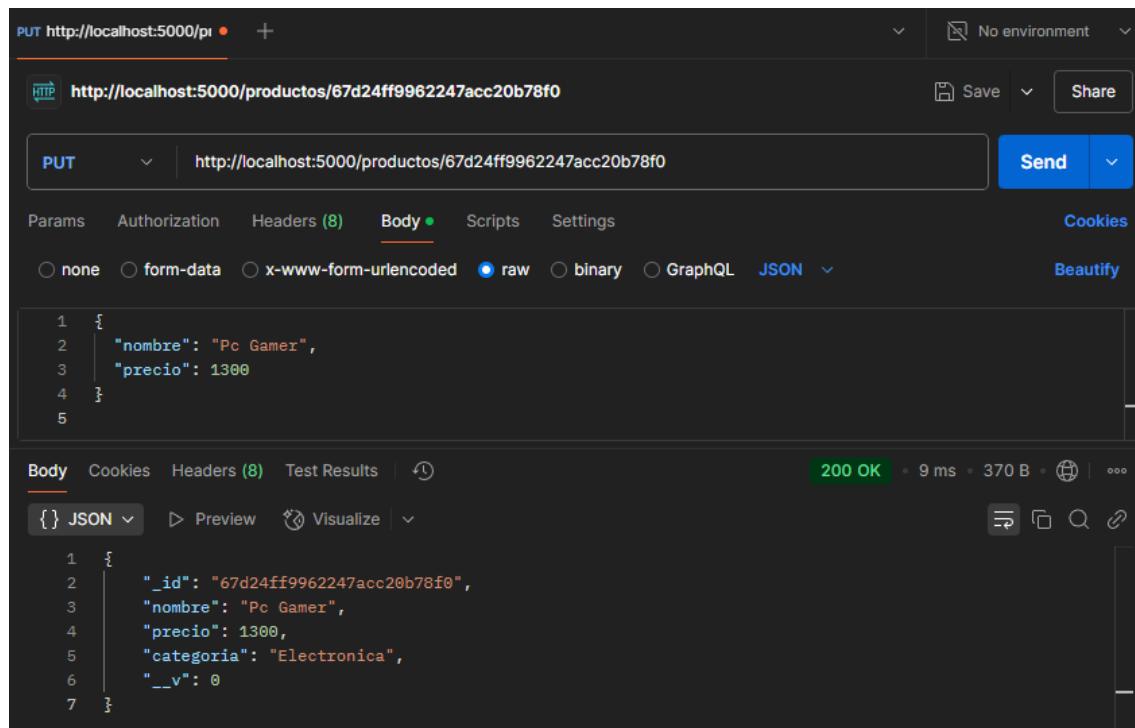
Prueba: Actualizar un producto (PUT /productos/:id)

- ◆ Método: PUT
- ◆ URL: http://localhost:5000/productos/{ID_DEL_PRODUCTO}
(Reemplaza {ID_DEL_PRODUCTO} con un _id válido de la base de datos.)
- ◆ Body (JSON):

```
{  
    "nombre": "Pc Gamer",  
    "precio": 1300  
}
```

Resultado esperado:

- Devuelve el producto actualizado con los nuevos datos.
- Comprobamos con un GET de usuarios como en el 1er paso.

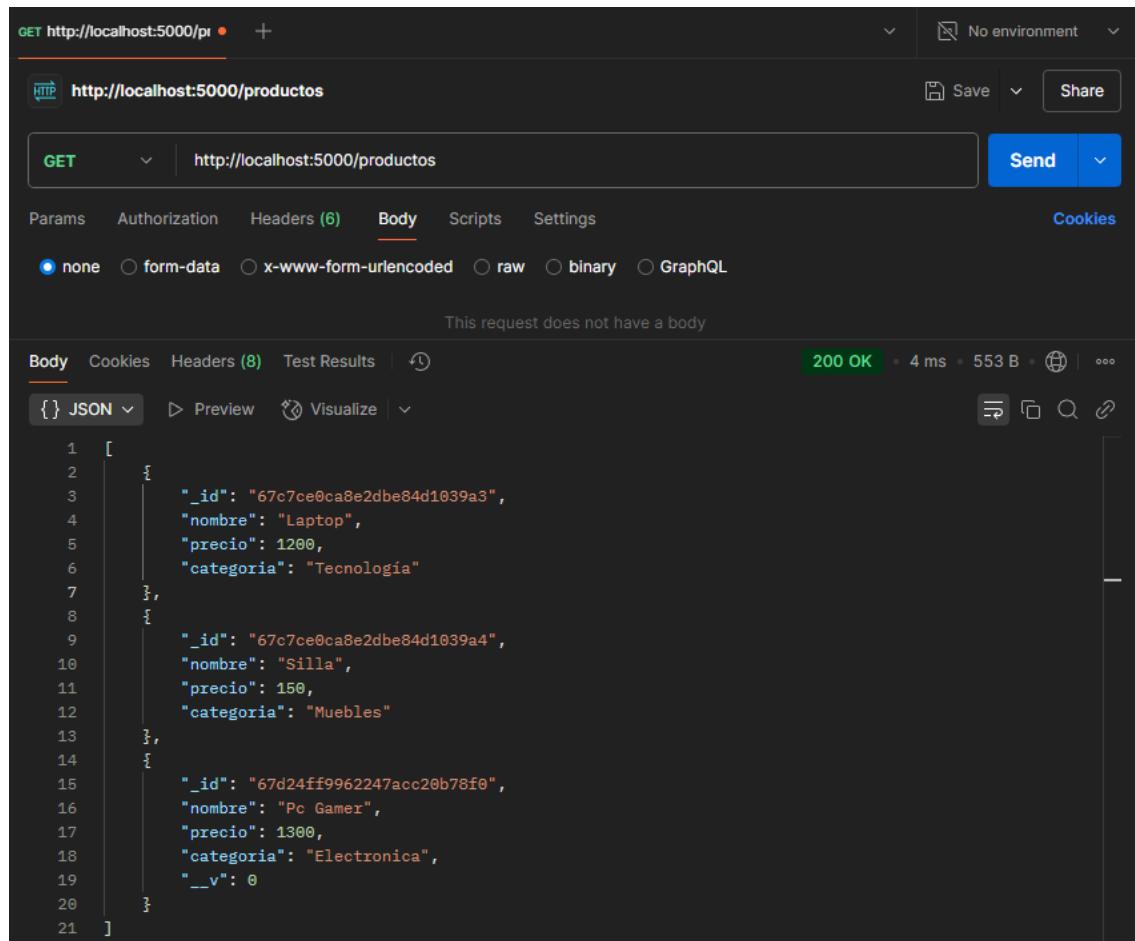


The screenshot shows the Postman application interface. The top bar indicates a PUT request to <http://localhost:5000/productos/67d24ff9962247acc20b78f0>. The 'Body' tab is selected, showing the JSON payload:

```
1  {  
2      "nombre": "Pc Gamer",  
3      "precio": 1300  
4  }
```

The response at the bottom shows a 200 OK status with 9 ms response time and 370 B size. The response body is identical to the request body:

```
1  {  
2      "_id": "67d24ff9962247acc20b78f0",  
3      "nombre": "Pc Gamer",  
4      "precio": 1300,  
5      "categoria": "Electronica",  
6      "__v": 0  
7  }
```



The screenshot shows a Postman collection with one item. The item has the URL `http://localhost:5000/productos`. The method is set to `GET`. The response status is `200 OK` with a response time of 4 ms and a size of 553 B. The response body is displayed in JSON format:

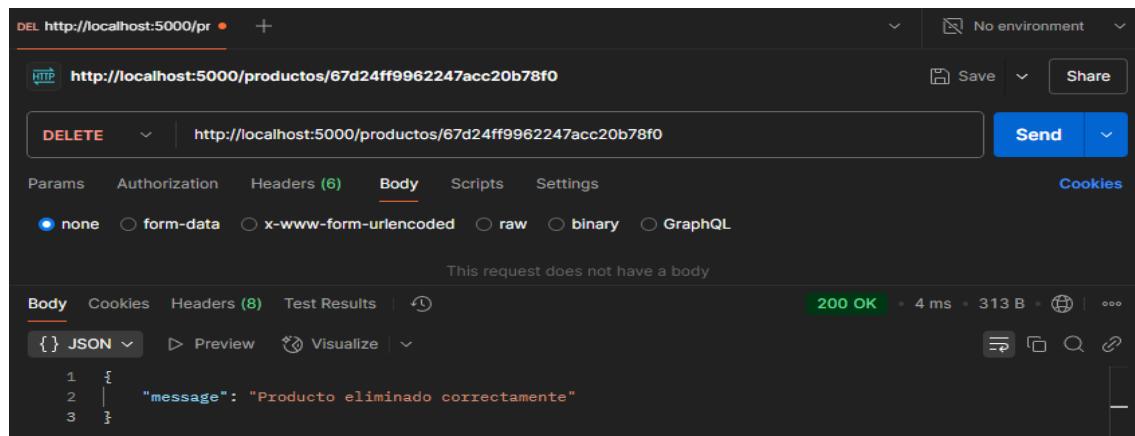
```
[{"_id": "67c7ce0ca8e2dbe84d1039a3", "nombre": "Laptop", "precio": 1200, "categoria": "Tecnología"}, {"_id": "67c7ce0ca8e2dbe84d1039a4", "nombre": "Silla", "precio": 150, "categoria": "Muebles"}, {"_id": "67d24ff9962247acc20b78f0", "nombre": "Pc Gamer", "precio": 1300, "categoria": "Electronica", "__v": 0}]
```

Eliminar un producto (DELETE /productos/:id)

- ◆ Método: DELETE
- ◆ URL: `http://localhost:5000/productos/{ID_DEL_PRODUCTO}`
(Reemplaza `{ID_DEL_PRODUCTO}` con un `_id` válido de la base de datos.)

Resultado esperado:

- Devuelve un mensaje de confirmación indicando que el producto fue eliminado.
- Comprobamos con un GET de usuarios como en el 1er paso.



The screenshot shows a Postman collection with one item. The item has the URL `http://localhost:5000/productos/67d24ff9962247acc20b78f0`. The method is set to `DELETE`. The response status is `200 OK` with a response time of 4 ms and a size of 313 B. The response body is displayed in JSON format:

```
{"message": "Producto eliminado correctamente"}
```

The screenshot shows a Postman interface with the following details:

- Request URL:** `http://localhost:5000/productos`
- Method:** GET
- Headers:** (6) (selected), Authorization, Headers, Body, Scripts, Settings, Cookies
- Body Content Type:** none
- Response Status:** 200 OK
- Response Time:** 5 ms
- Response Size:** 449 B
- Response Body (JSON):**

```
1 [  
2 {  
3     "_id": "67c7ce0ca8e2dbe84d1039a3",  
4     "nombre": "Laptop",  
5     "precio": 1200,  
6     "categoria": "Tecnología"  
7 },  
8 {  
9     "_id": "67c7ce0ca8e2dbe84d1039a4",  
10    "nombre": "Silla",  
11    "precio": 150,  
12    "categoria": "Muebles"  
13 }  
14 ]
```

Prueba: Contadores de registros (GET /contadores)

- ◆ Método: GET
- ◆ URL: `http://localhost:5000/contadores`

Resultado esperado:

- Devuelve un JSON con el total de usuarios y productos en la base de datos.

```
{  
  "totalUsuarios": 2,  
  "totalProductos": 2  
}
```

GET http://localhost:5000/contadores

http://localhost:5000/contadores

Send

Params Authorization Headers (6) Body Scripts Settings Cookies

None form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

200 OK • 33 ms • 305 B

Body Cookies Headers (8) Test Results

{ } JSON Preview Visualize

```
1 {  
2   "totalUsuarios": 2,  
3   "totalProductos": 2  
4 }
```

Prueba: Contador de operaciones realizadas (GET /operaciones)

- ◆ Método: GET
- ◆ URL: http://localhost:5000/operaciones

Resultado esperado:

- Devuelve un JSON con la cantidad total de operaciones realizadas en el backend.

```
{  
  "totalOperaciones": 20  
}
```

GET http://localhost:5000/operaciones

http://localhost:5000/operaciones

Send

Params Authorization Headers (6) Body Scripts Settings Cookies

None form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

200 OK • 3 ms • 290 B

Body Cookies Headers (8) Test Results

{ } JSON Preview Visualize

```
1 {  
2   "totalOperaciones": 20  
3 }
```

Tras ejecutar todas las pruebas en Postman, se ha verificado que el backend desarrollado con Node.js, Express.js y MongoDB funciona correctamente y cumple con los requerimientos del proyecto.

Las pruebas demostraron que:

- ✓ Las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) en Usuarios y Productos funcionan sin errores.
- ✓ Los contadores de registros reflejan el número exacto de documentos en la base de datos.
- ✓ El middleware de conteo de operaciones registra correctamente cada petición realizada.

Este backend está listo para ser utilizado en un proyecto real o ser ampliado con nuevas funcionalidades.

Link de GitHub: <https://github.com/JoanRiv/BackendExpress>