**Pablo Andres Cely**

**Javier Adrián Martínez**

**Yerson Tumay**

# Guía Flask

### Paso 1:

### Crea una carpeta

1. **Guarda el código** como **Get.py** dentro de esa carpeta.  
     
   from flask import Flask, request, jsonify
2. app = Flask(\_\_name\_\_)
3. # "Base de datos" en memoria. Es una lista de diccionarios.
4. # Comenzamos con algunos datos de ejemplo más "llenos".
5. items\_db = [
6. {"id": 1, "name": "Laptop Modelo X", "description": "Potente laptop para trabajo y juegos, 16GB RAM, SSD 512GB."},
7. {"id": 2, "name": "Mouse Inalámbrico Ergonómico", "description": "Mouse con diseño ergonómico para mayor comodidad, conexión Bluetooth."},
8. {"id": 3, "name": "Teclado Mecánico RGB", "description": "Teclado con switches mecánicos, retroiluminación RGB personalizable."},
9. {"id": 4, "name": "Monitor Curvo 27 pulgadas", "description": "Monitor Full HD curvo para una experiencia visual inmersiva."},
10. {"id": 5, "name": "ADRIAN ES GAY", "description": "CHUPA MONDA POR DINERO"}
11. ]
12. # Para generar IDs únicos para nuevos items.
13. # Se ajusta al último ID de la lista de ejemplo + 1.
14. next\_item\_id = 5
15. @app.route('/', methods=['GET'])
16. def home():
17. return jsonify({"message": "Bienvenido a la API de Items. Accede a /items para ver los items."})
18. # Rutas para el recurso "items"
19. @app.route('/items', methods=['GET', 'POST'])
20. def handle\_items():
21. global next\_item\_id # Necesitamos indicar que vamos a modificar la variable global
22. if request.method == 'GET':
23. # Devolver todos los items de nuestra "base de datos"
24. return jsonify(items\_db)
26. elif request.method == 'POST':
27. # Crear un nuevo item
28. new\_item\_data = request.json # Los datos vienen en el cuerpo de la solicitud como JSON
30. # Validación simple: el 'name' es requerido
31. if not new\_item\_data or 'name' not in new\_item\_data or not new\_item\_data['name'].strip():
32. return jsonify({"error": "Faltan datos o el campo 'name' es requerido y no puede estar vacío"}), 400 # Bad Request
33. new\_item = {
34. "id": next\_item\_id, # Asignamos el ID automáticamente
35. "name": new\_item\_data.get("name"),
36. "description": new\_item\_data.get("description", "") # Descripción es opcional, por defecto cadena vacía
37. }
38. items\_db.append(new\_item) # Agregamos el nuevo item a nuestra lista
39. next\_item\_id += 1 # Incrementamos el contador para el próximo ID
40. return jsonify({"message": "Item creado exitosamente", "item": new\_item}), 201 # 201 Created
41. @app.route('/items/<int:item\_id>', methods=['GET', 'PUT', 'DELETE'])
42. def handle\_specific\_item(item\_id):
43. # Buscar el item en nuestra "base de datos" por su ID
44. item\_found = None
45. for item in items\_db:
46. if item["id"] == item\_id:
47. item\_found = item
48. break
49. # Si el item no se encuentra, devolver un error 404
50. if not item\_found:
51. return jsonify({"error": f"Item con ID {item\_id} no encontrado"}), 404 # Not Found
52. if request.method == 'GET':
53. # Devolver los detalles del item encontrado
54. return jsonify(item\_found)
56. elif request.method == 'PUT':
57. # Actualizar un item existente
58. updated\_data = request.json # Los datos para actualizar vienen en el cuerpo JSON
59. if not updated\_data:
60. return jsonify({"error": "No se proporcionaron datos para actualizar"}), 400 # Bad Request
61. # Actualizar los campos permitidos. Si 'name' viene vacío, se podría añadir validación.
62. if 'name' in updated\_data and updated\_data['name'].strip(): # Asegura que el nombre no sea solo espacios
63. item\_found["name"] = updated\_data["name"]
64. if 'description' in updated\_data: # La descripción puede ser una cadena vacía
65. item\_found["description"] = updated\_data["description"]
67. return jsonify({"message": f"Item {item\_id} actualizado exitosamente", "item": item\_found})
69. elif request.method == 'DELETE':
70. # Eliminar un item existente
71. items\_db.remove(item\_found) # Eliminamos el item de la lista
72. return jsonify({"message": f"Item {item\_id} eliminado exitosamente"})
73. if \_\_name\_\_ == '\_\_main\_\_':
74. # El servidor se ejecutará en http://127.0.0.1:5000/ por defecto
75. # debug=True es útil para desarrollo:
76. # - Recarga el servidor automáticamente con los cambios en el código.
77. # - Muestra errores detallados en el navegador.
78. # ¡No uses debug=True en un entorno de producción!
79. app.run(debug=True)
80. **TODOS DEBEMOS TENER INSTALADO FLASK**

### Paso 2:

* **Importaciones e Inicialización de Flask:**

Python  
from flask import Flask, request, jsonify  
app = Flask(\_\_name\_\_)

* Importa las herramientas necesarias de Flask (Flask para la app, request para manejar datos de entrada, jsonify para respuestas JSON). Luego, crea la aplicación Flask.
* **"Base de Datos" y Contador de ID:**

Python  
items\_db = [ ... ] # Lista de items de ejemplo  
next\_item\_id = 5

items\_db es una lista que simula una base de datos con algunos items iniciales. next\_item\_id se usa para asignar IDs a nuevos items (¡cuidado con el valor inicial si tus datos de ejemplo ya usan ese ID!).

* **Ruta Principal (/):**

Python  
@app.route('/', methods=['GET'])  
def home():  
 # …

Define qué sucede cuando alguien visita la URL raíz (ej. http://127.0.0.1:5000/). Devuelve un mensaje de bienvenida.

* **Ruta /items**

Python  
@app.route('/items', methods=['GET', 'POST'])  
def handle\_items():  
 # ...  
  
Esta función maneja dos situaciones para la URL /items

* **Ruta /items/<int:item\_id> (GET, PUT, DELETE):**

Python  
@app.route('/items/<int:item\_id>', methods=['GET', 'PUT', 'DELETE'])  
def handle\_specific\_item(item\_id):  
 # ...  
  
Esta función maneja operaciones para un item individual, identificado por su item\_id en la URL

.

* **Ejecución de la Aplicación**

Python  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

Este bloque estándar de Python hace que, cuando ejecutas python app.py, se inicie el servidor de desarrollo de Flask.

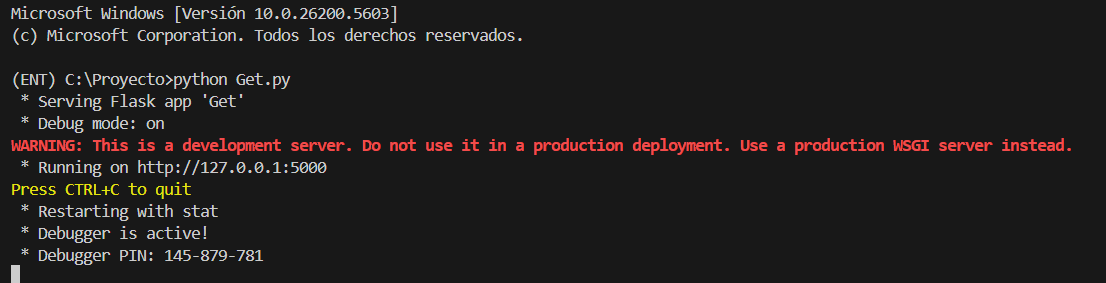
### Paso 3: Ejecutar la API

### 1.

### 

En la terminal de VS Code , ejecuta:  
python Get[.py](http://app.py)



1. Verás un mensaje indicando que el servidor está corriendo en http://127.0.0.1:5000/  
   ****

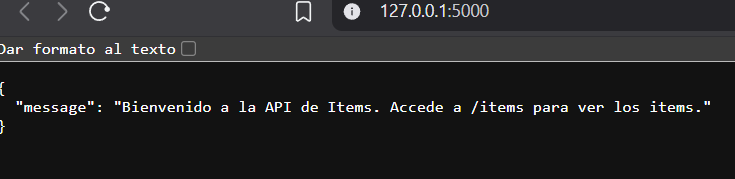
### Paso 4: Probar la API

#### Con Navegador (para GET):

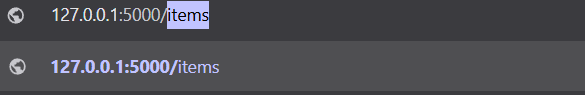
1. **Bienvenida:** Abre http://127.0.0.1:5000/ en tu navegador.

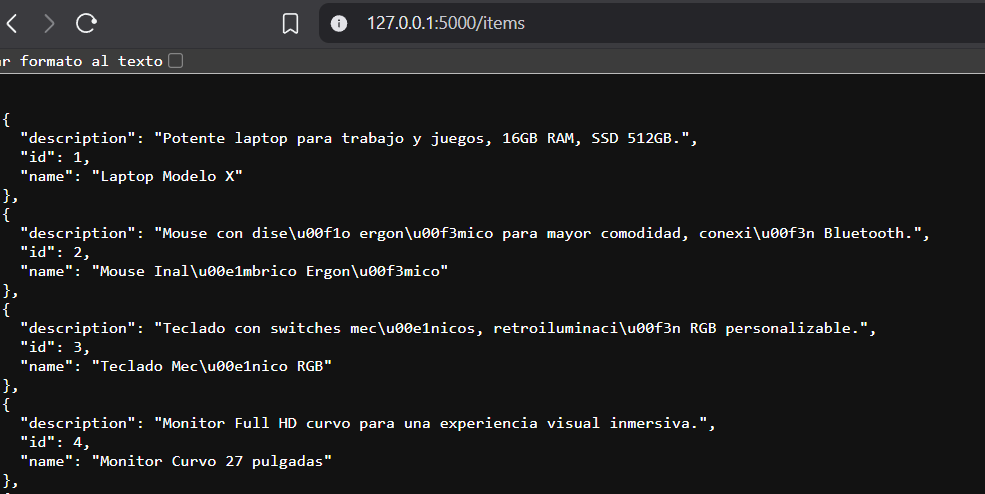


1. **Todos los Items:** Abre <http://127.0.0.1:5000/items>



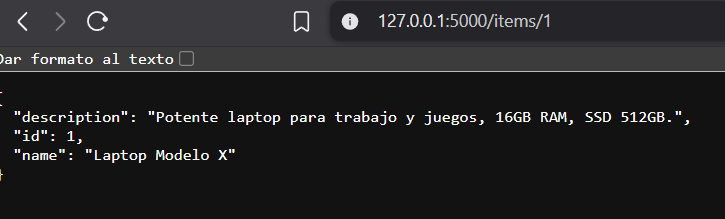
3. Búsqueda Items.



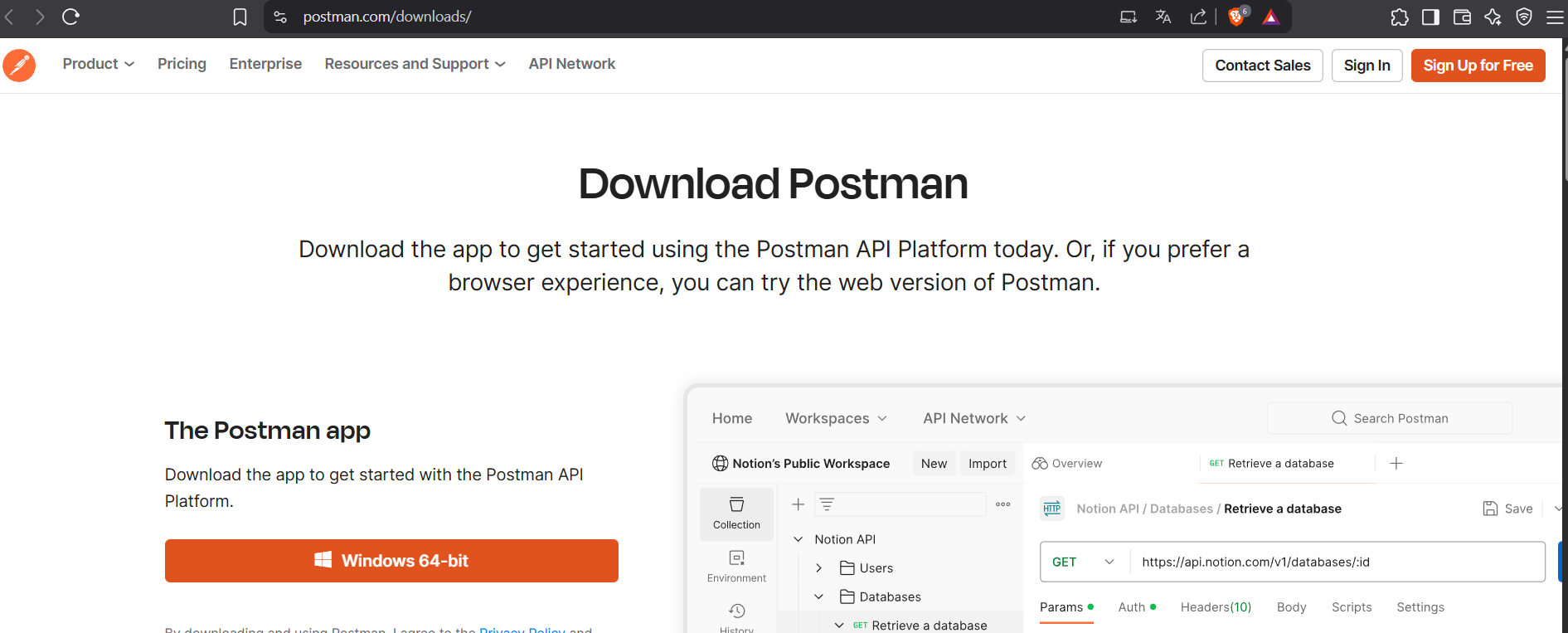


1. **Item Específico:** Abre http://127.0.0.1:5000/items/1

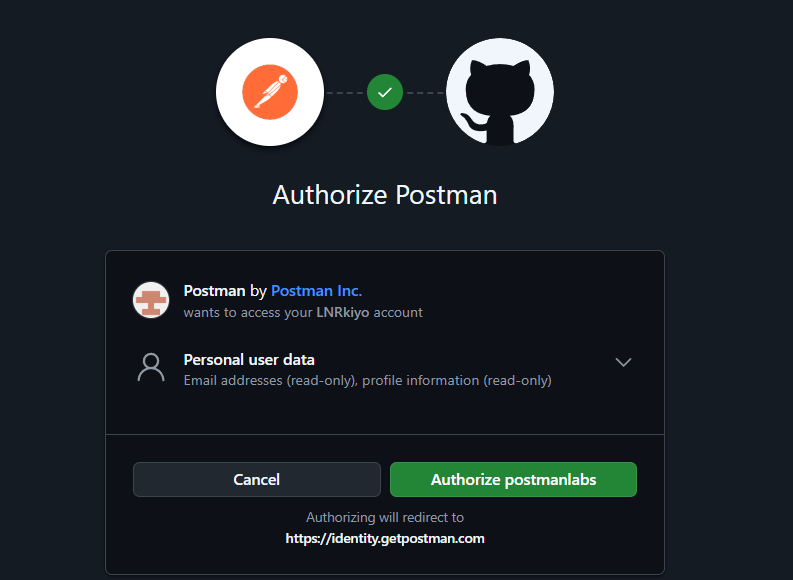
****

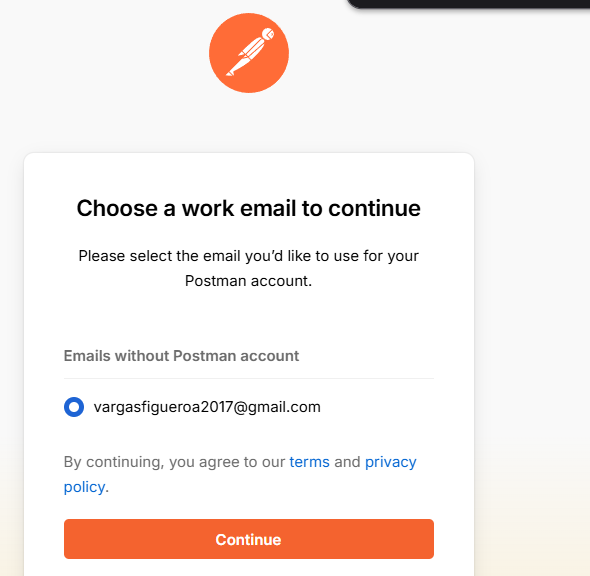
****

1. **Con Herramienta API( POST, PUT, DELETE):**
2. **Crear Item (POST):**
   * + **Descargar Post**

****

**Ingresamos con Github**

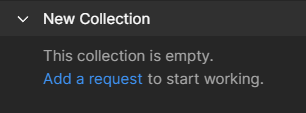
****

****

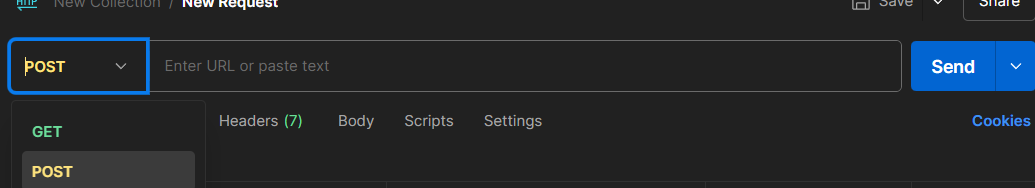
**para correr POST**

****

**Damos en la opcion ADD A REQUESTS**

****

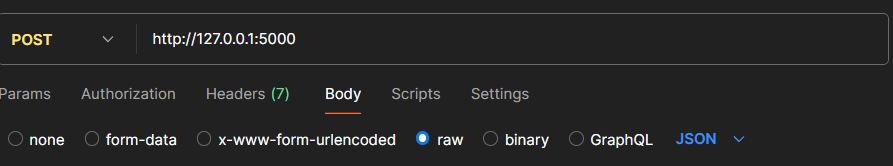
**Seleccionamos POST**

****

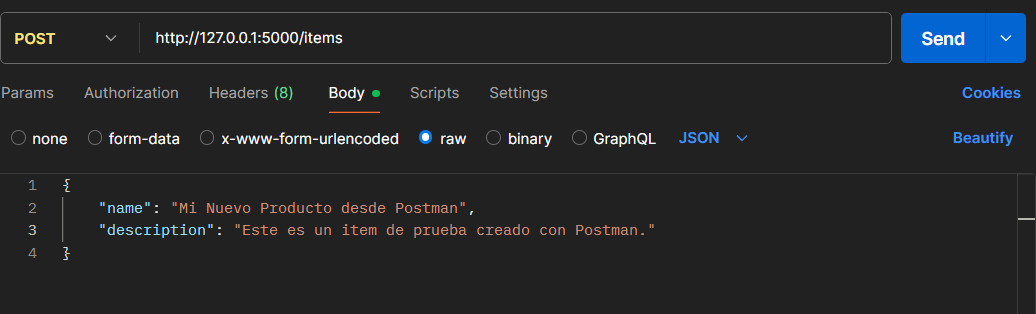
**Pegamos nuestro URL**

****

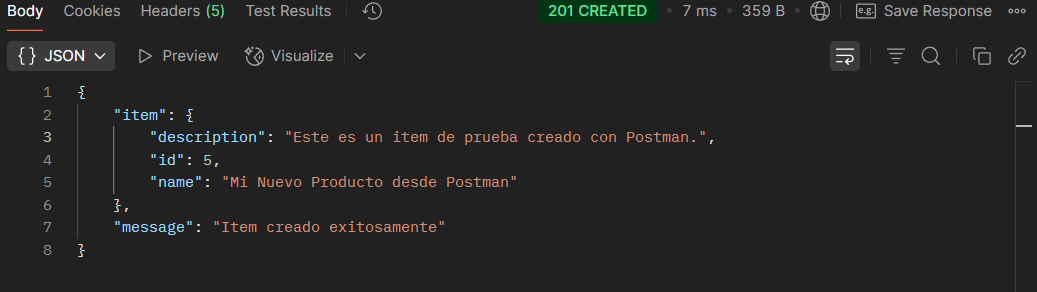
**Seleccionamos RAW y JSON**

****

**Hacemos el siguente codigo en la parte de abajo y le damos en SEND**

****

**Funciono**

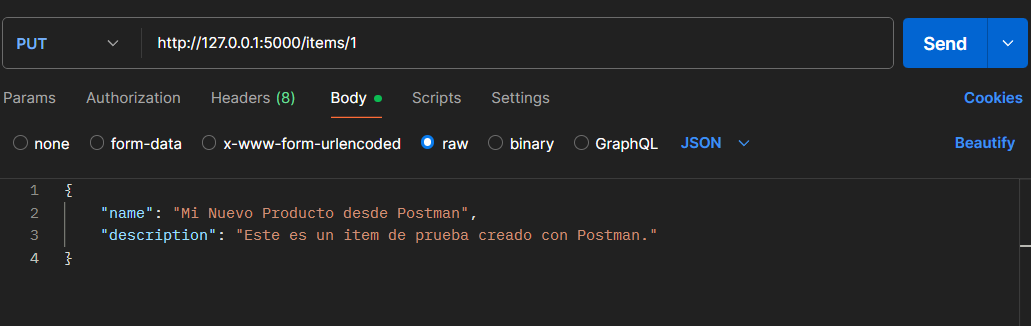
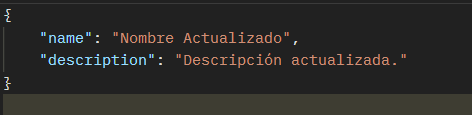
****

**Resultados**

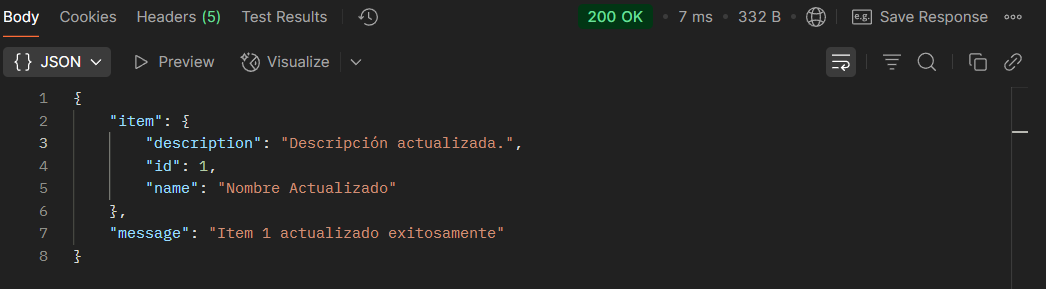
****

**Código**  
 {  
 "name": "Producto Nuevo",  
 "description": "Descripción del producto nuevo."  
 }

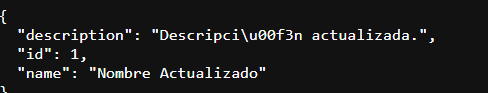
1. **Actualizar Item (PUT):**

  
**Código**  
{  
 "name": "Nombre Actualizado",  
 "description": "Descripción actualizada."  
}  


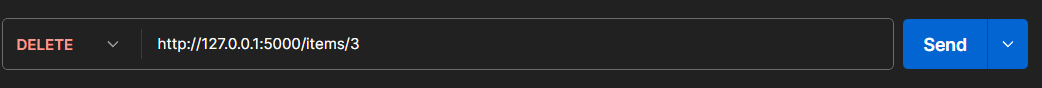
Deberías recibir un código 200 OK y el item actualizado



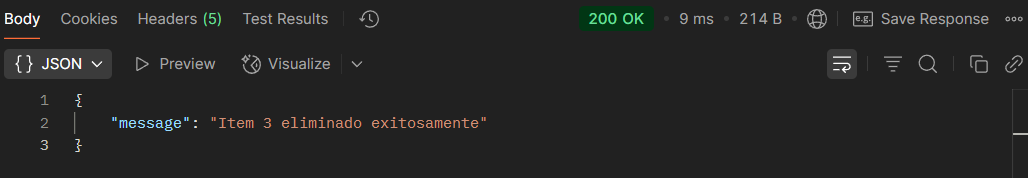
**Resultados PÁGINA**

****

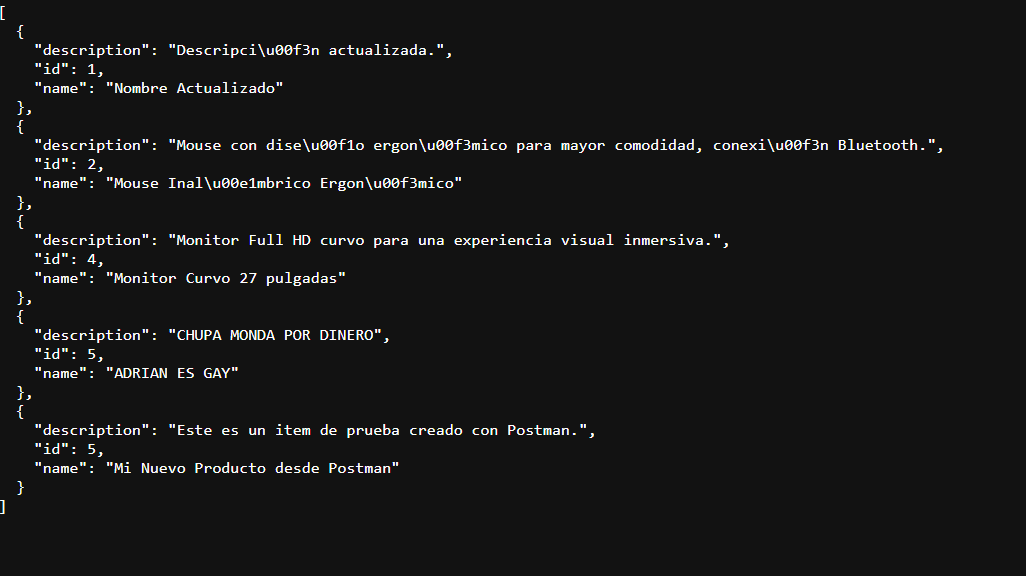
1. **Eliminar Item (DELETE)**

****

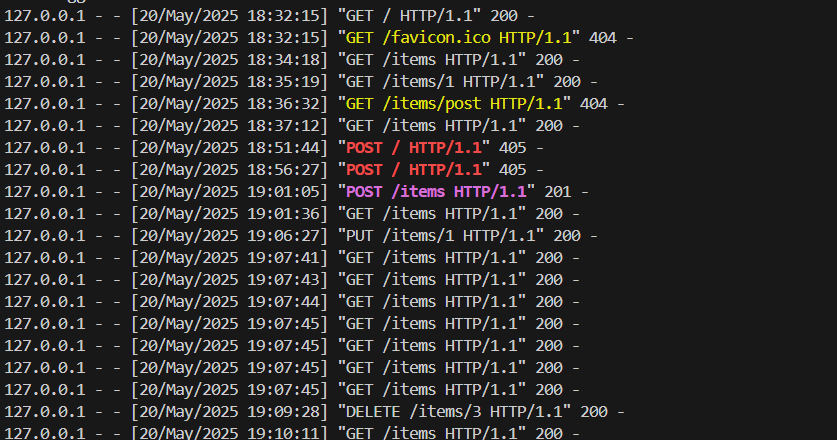
**Exito**

****

**No Exite El Item borrado**

****

**Todo lo hecho queda en el VS**

****