

Integrantes:

- Leidy Daniela Londoño Candelo - A00392917
- Isabella Huila Cerón - A00394751

Nombre del proyecto: Burger House

Descripción

Burger House es un restaurante especializado en hamburguesas que busca optimizar la gestión de pedidos y mejorar la experiencia del cliente. Actualmente, la falta de un sistema integrado genera ineficiencias como retrasos en la preparación, errores en la comunicación y una experiencia de compra poco fluida. Este sistema resolverá estos problemas al centralizar la gestión de pedidos en una plataforma intuitiva y eficiente, permitiendo a los clientes explorar el menú, personalizar sus órdenes, realizar pedidos rápidamente y rastrear su estado en tiempo real.

Funcionalidades implementadas en la API

Modulo Producto:

El módulo Product gestiona la creación, consulta, actualización y eliminación de productos dentro del sistema. Cada producto contiene atributos como nombre, descripción, precio y categoría (por ejemplo: burgers, Accompaniments, drinks). La información se almacena en una base de datos mediante TypeORM.

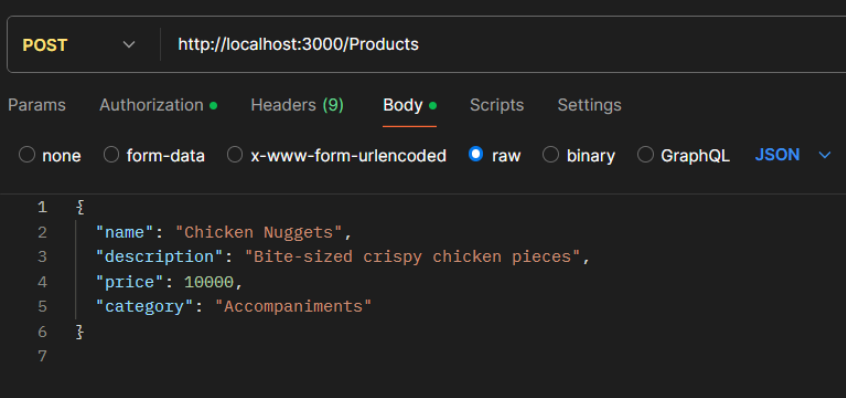
Endpoints Expuestos

- **POST /products**

Descripción: Crea un nuevo producto.

Autenticación: Requiere token JWT con rol de admin.

Body:



```
POST http://localhost:3000/Products

{
  "name": "Chicken Nuggets",
  "description": "Bite-sized crispy chicken pieces",
  "price": 10000,
  "category": "Accompaniments"
}
```

Respuestas:

- **201 Created:** Producto creado correctamente.
- **400 Bad Request:** Datos inválidos o conflicto en BD.
- **403 Forbidden:** Acceso no autorizado.

- GET /products

Descripción: Obtiene todos los productos activos.

Parámetros (Query):

limit (opcional): Número máximo de productos a devolver (default: 10).

offset (opcional): Paginación (default: 0).

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:3000/Products`
- Method:** `GET`
- Status:** `200 OK` (27 ms, 2.04 KB)
- Response Type:** `JSON`
- Response Body:** A JSON array containing three burger objects.

```
1 [
2   {
3     "id": "e085a66e-9fdd-48b9-b14f-d2b12400ab66",
4     "name": "Classic Burger",
5     "description": "Delicious beef burger with lettuce, tomato, and special sauce",
6     "price": "20000.00",
7     "isActive": true,
8     "category": "burgers"
9   },
10  {
11    "id": "03041bd0-df64-460a-ad36-b323f10aeb58",
12    "name": "Cheese Burger",
13    "description": "Burger with cheddar cheese and caramelized onions",
14    "price": "22000.00",
15    "isActive": true,
16    "category": "burgers"
17  },
18  {
19    "id": "1d7971c3-2876-4f93-8412-d760b8cf3524",
20    "name": "Chicken Burger",
```

Respuestas:

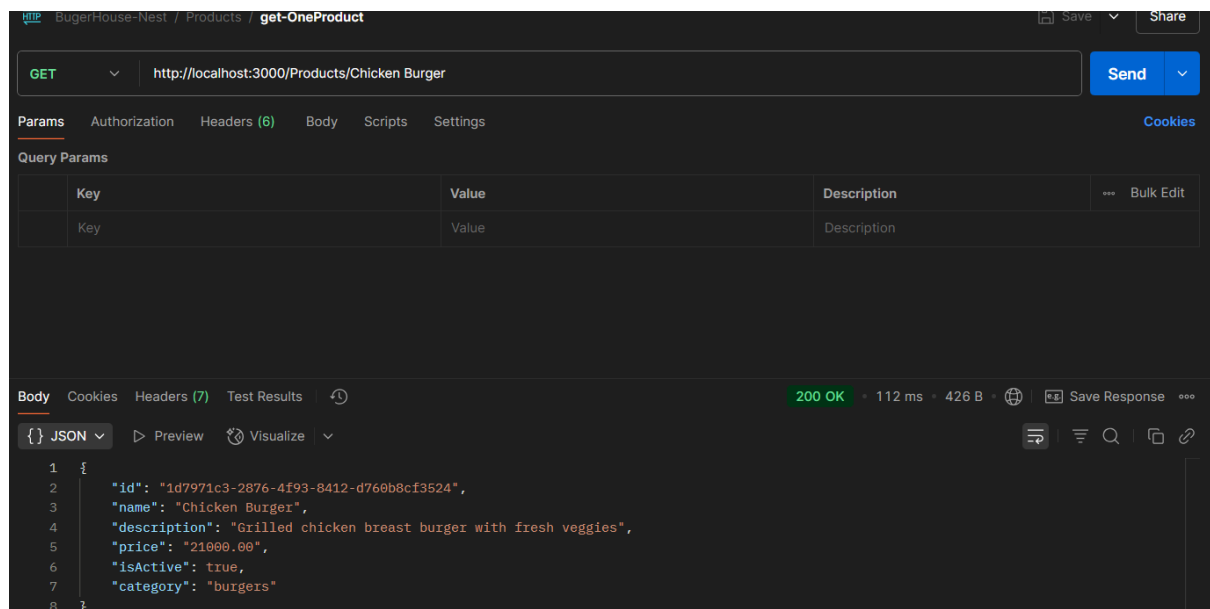
- **200 OK:** Lista de productos activos.

- **GET /products/:name**

Descripción: Obtiene los detalles de un producto por su nombre.

Parámetros (URL):

name: Nombre del producto.



Respuestas:

- **200 OK:** Datos del producto.
- **404 Not Found:** Producto no encontrado o inactivo.

- **PATCH /products/:name**

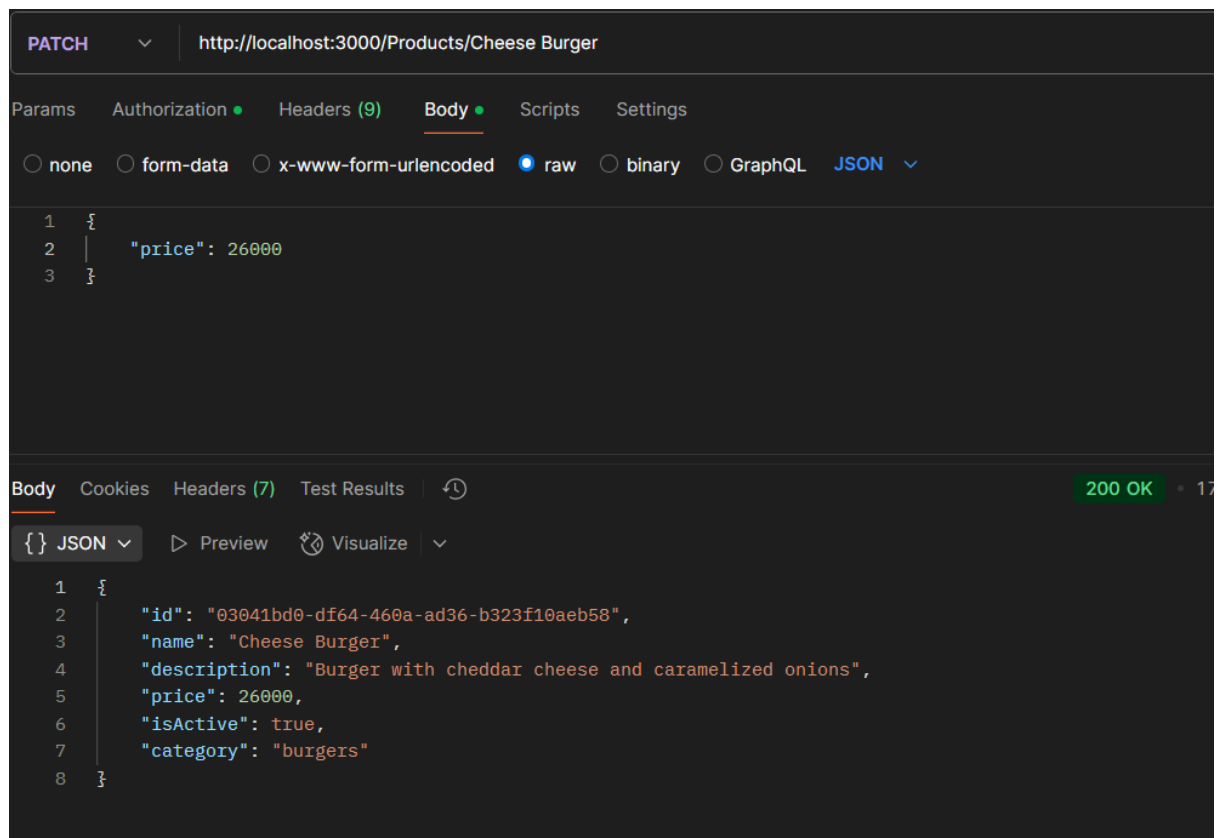
Descripción: Actualiza los datos de un producto.

Autenticación: Requiere token JWT con rol de admin.

Parámetros (URL):

name: Nombre del producto a actualizar.

Body:



Respuestas:

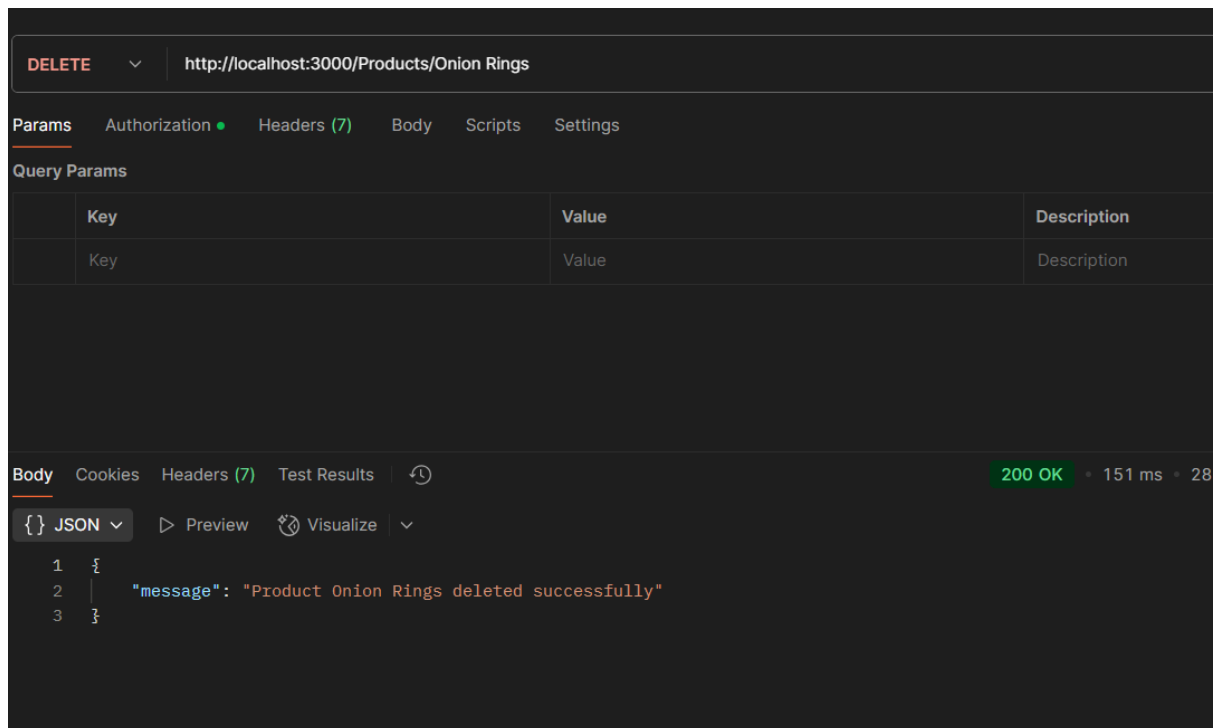
- **200 OK:** Producto actualizado correctamente.
- **400 Bad Request:** Datos inválidos.
- **404 Not Found:** Producto no encontrado.
- **DELETE /products/:name**

Descripción: Realiza una eliminación lógica (soft delete) de un producto.

Autenticación: Requiere token JWT con rol de admin.

Parámetros (URL):

name: Nombre del producto a eliminar.



Respuestas:

- **200 OK:** Producto desactivado exitosamente.
- **404 Not Found:** Producto no encontrado

ModuloTopping:

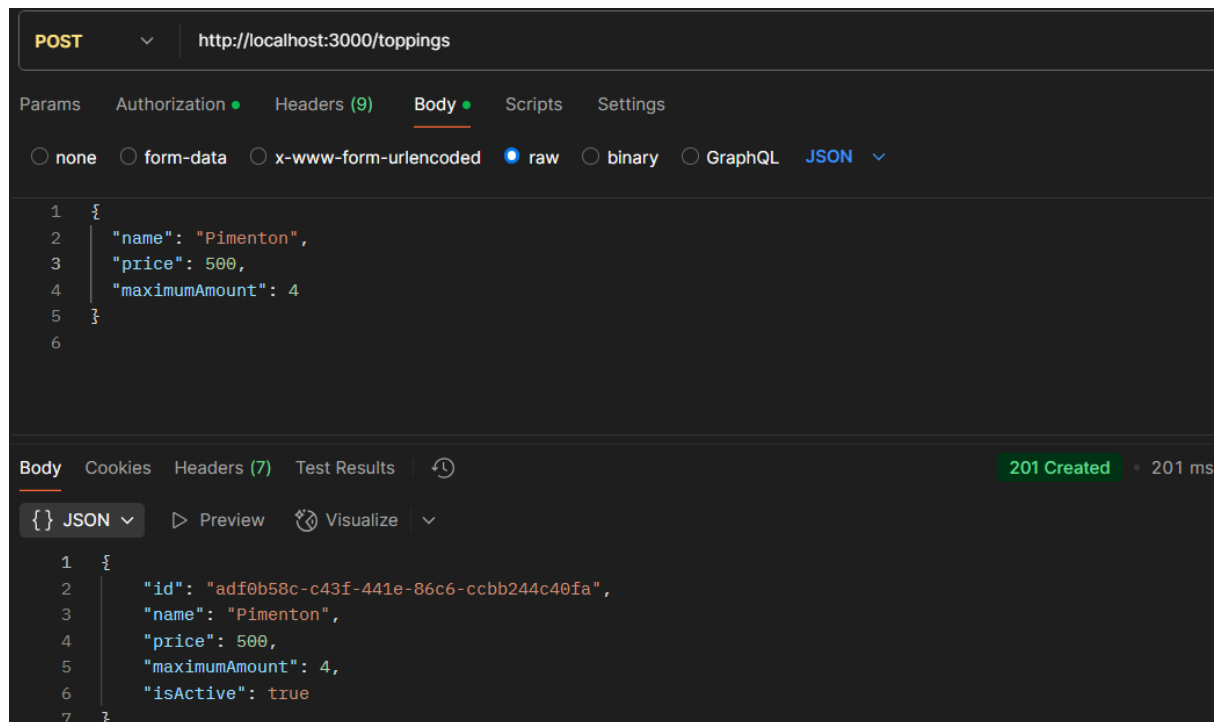
El módulo Topping administra los toppings disponibles para productos, incluyendo su creación, edición, desactivación (soft delete) y su asignación a productos mediante una relación intermedia (ProductTopping). Se asegura que los toppings asignados respeten restricciones como el máximo permitido por tipo.

- **POST /toppings**

Descripción: Crea un nuevo topping.

Autenticación: Requiere token JWT con rol de admin.

Body:



Respuestas:

- **201 Created:** Topping creado exitosamente.
- **400 Bad Request:** Error de validación o duplicado.
- **GET /toppings**

Descripción: Obtiene todos los toppings activos con paginación.

Autenticación: No requiere.

Query Params (opcional):

limit (number): Cantidad de resultados por página.

offset (number): Desplazamiento de resultados.

GET http://localhost:3000/toppings

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (7) Test Results 200 OK • 176 ms • 943 B

{ } JSON Preview Visualize

```
1  [
2    {
3      "id": "c55ba50c-2ccc-471e-9920-38d7736a4745",
4      "name": "Extra Cheese",
5      "price": "2000.00",
6      "maximumAmount": 5,
7      "isActive": true
8    },
9    {
10     "id": "bf80fd38-4d01-456f-9e39-2de6eed61ce1",
11     "name": "Bacon Bits",
12     "price": "2500.00",
13     "maximumAmount": 6,
14     "isActive": true
15   },
16 ]
```

Respuestas:

- **200 OK:** Lista de toppings activos.
- **GET /toppings/:name**

Descripción: Obtiene un topping por su nombre.

Autenticación: No requiere.

Parámetros (URL):

name (string): Nombre del topping a buscar.

GET http://localhost:3000/toppings/Extra Cheese

Params Authorization Headers (6) Body Scripts Settings

Query Params

	Key	Value	Descrip
	Key	Value	Descrip

Body Cookies Headers (7) Test Results 200 OK

{ } JSON Preview Visualize

```
1 {
2   "id": "c55ba50c-2ccc-471e-9920-38d7736a4745",
3   "name": "Extra Cheese",
4   "price": "2000.00",
5   "maximumAmount": 5,
6   "isActive": true
7 }
```

Respuestas:

- **200 OK:** Topping encontrado.
- **404 Not Found:** Topping no encontrado o inactivo.
- **PATCH /toppings/:name**

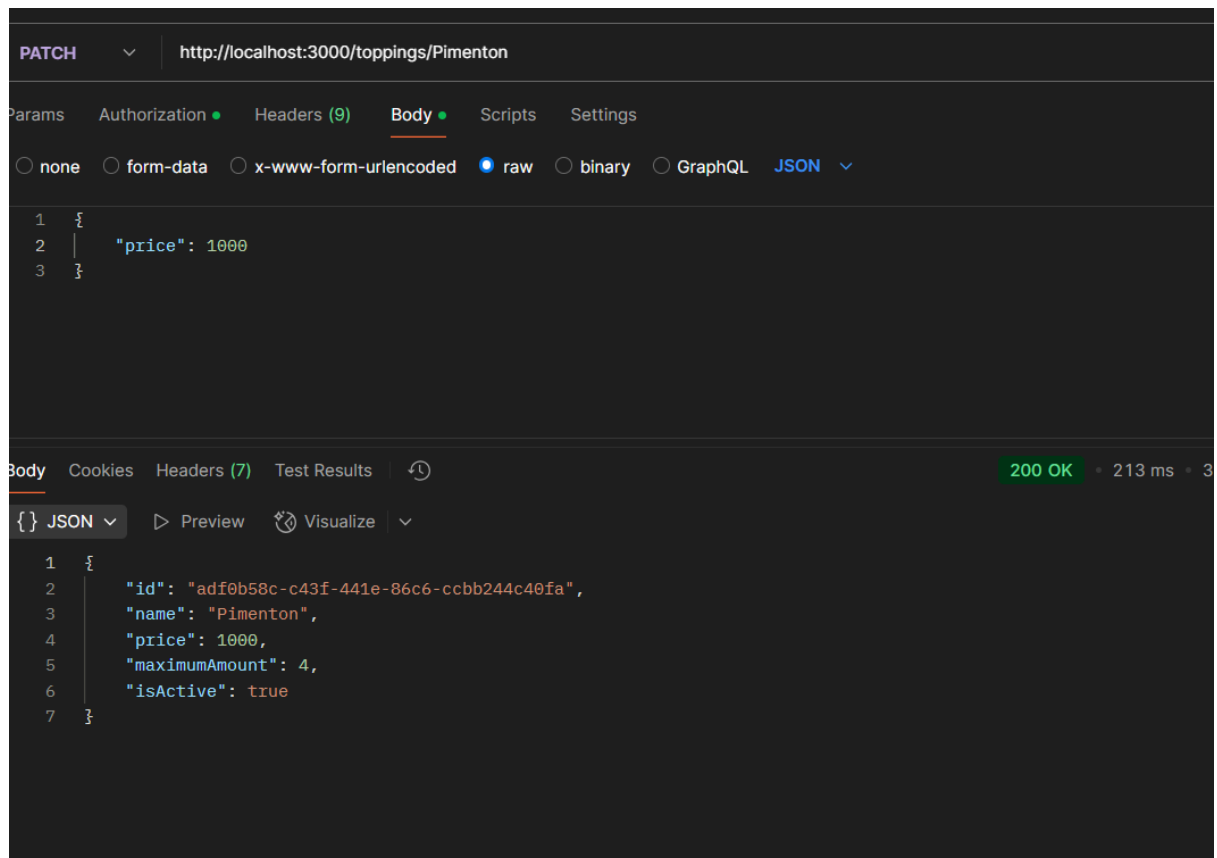
Descripción: Actualiza la información de un topping.

Autenticación: Requiere token JWT con rol de admin.

Parámetros (URL):

name (string): Nombre del topping a actualizar.

Body (opcional):



Respuestas:

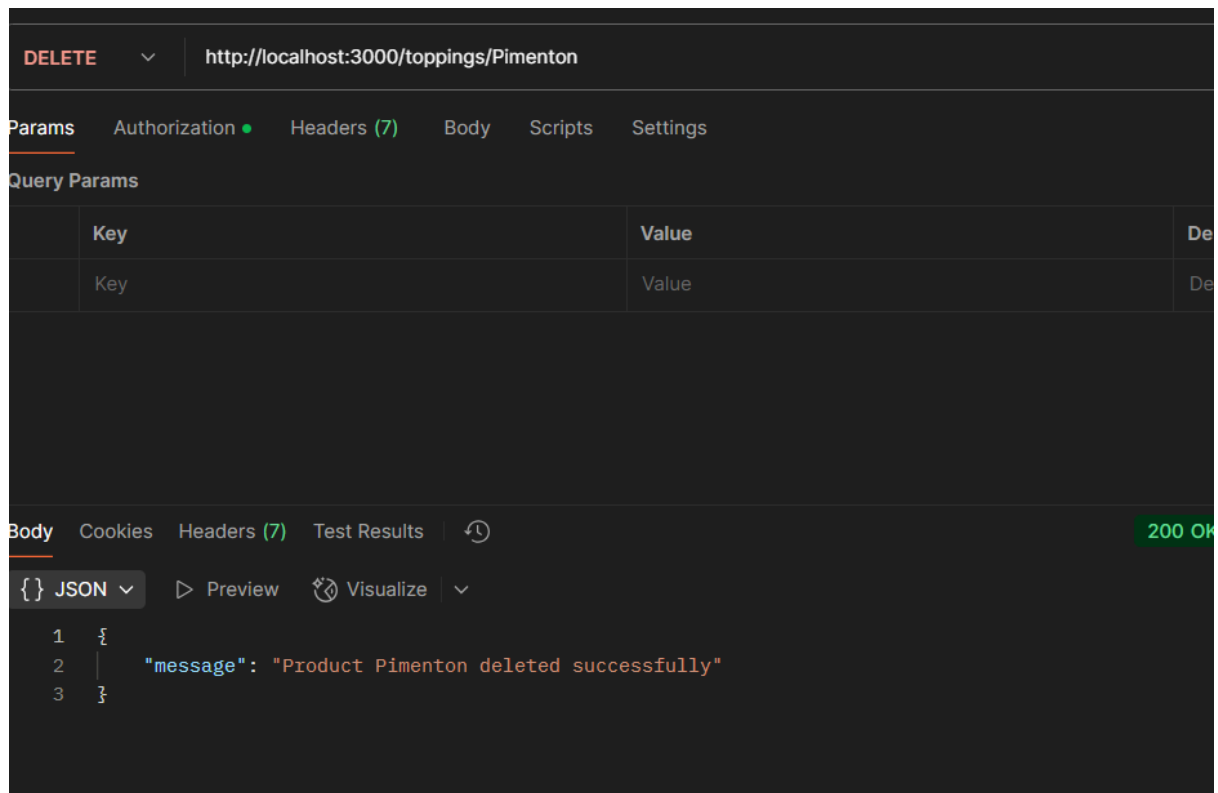
- **200 OK:** Topping actualizado exitosamente.
- **400 Bad Request:** Error de validación o conflicto.
- **404 Not Found:** Topping no encontrado.
- **DELETE /toppings/:name**

Descripción: Realiza una eliminación lógica (soft delete) de un topping.

Autenticación: Requiere token JWT con rol de admin.

Parámetros (URL):

name (string): Nombre del topping a eliminar.



Respuestas:

- **200 OK:** Topping desactivado exitosamente.
- **404 Not Found:** Topping no encontrado.
- **POST /toppings/add-topping**

Descripción: Asigna un topping a un producto.

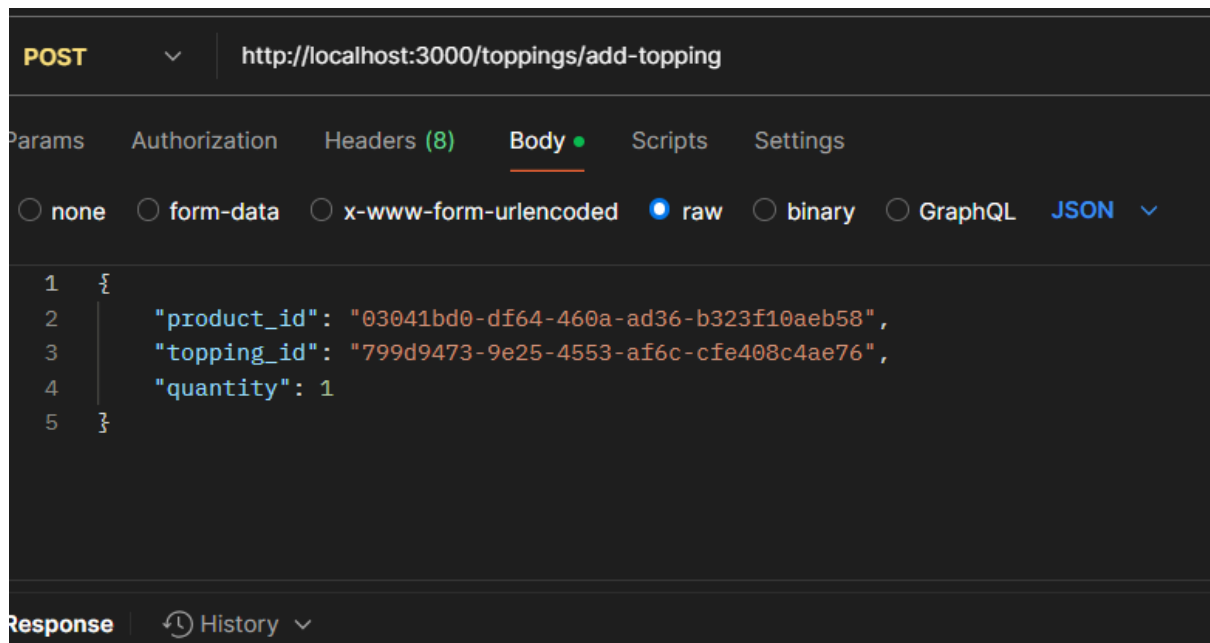
Autenticación: No requiere.

Body:

product_id (UUID): ID del producto.

topping_id (UUID): ID del topping.

quantity (number): Cantidad del topping.



Validaciones:

- La cantidad no debe superar el maximumAmount definido en el topping.
- No se permite asignar un mismo topping dos veces al mismo producto.

Respuestas:

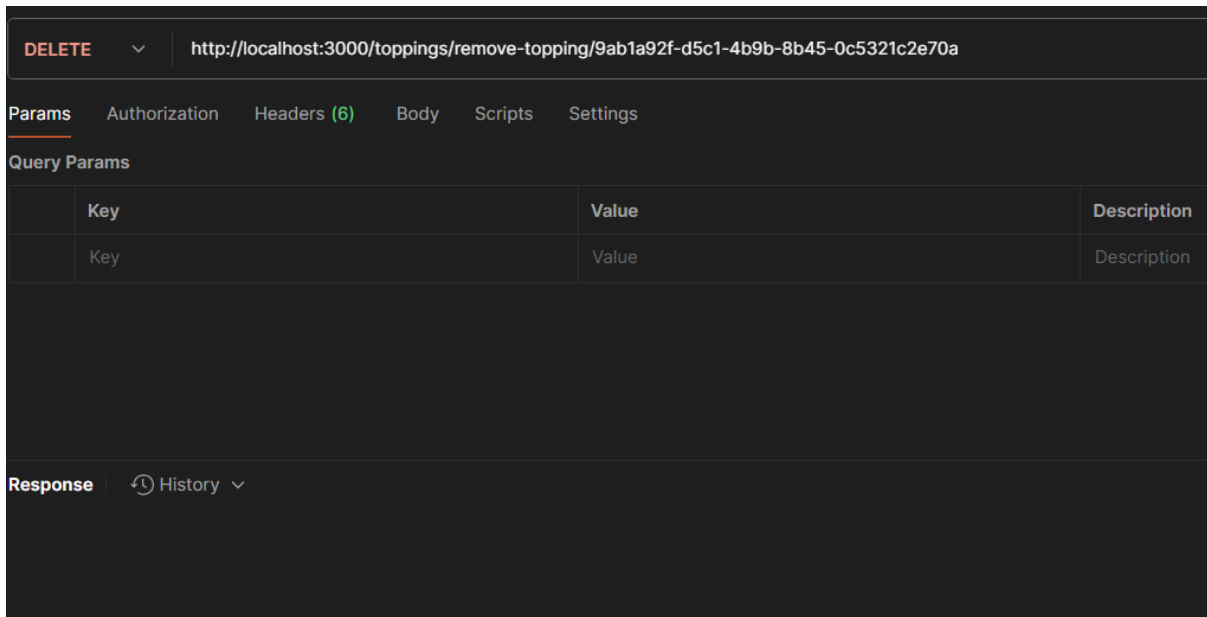
- **201 Created:** Topping asignado al producto.
- **400 Bad Request:** Ya existe la relación o cantidad inválida.
- **404 Not Found:** Producto o topping no encontrados.
- **DELETE /toppings/remove-topping/:id**

Descripción: Elimina un topping de un producto.

Autenticación: No requiere.

Parámetros (URL):

id (UUID): ID de la relación ProductTopping.



Respuestas:

- **200 OK:** Relación eliminada exitosamente.
- **404 Not Found:** Relación no encontrada.

Modulo de Ordenes:

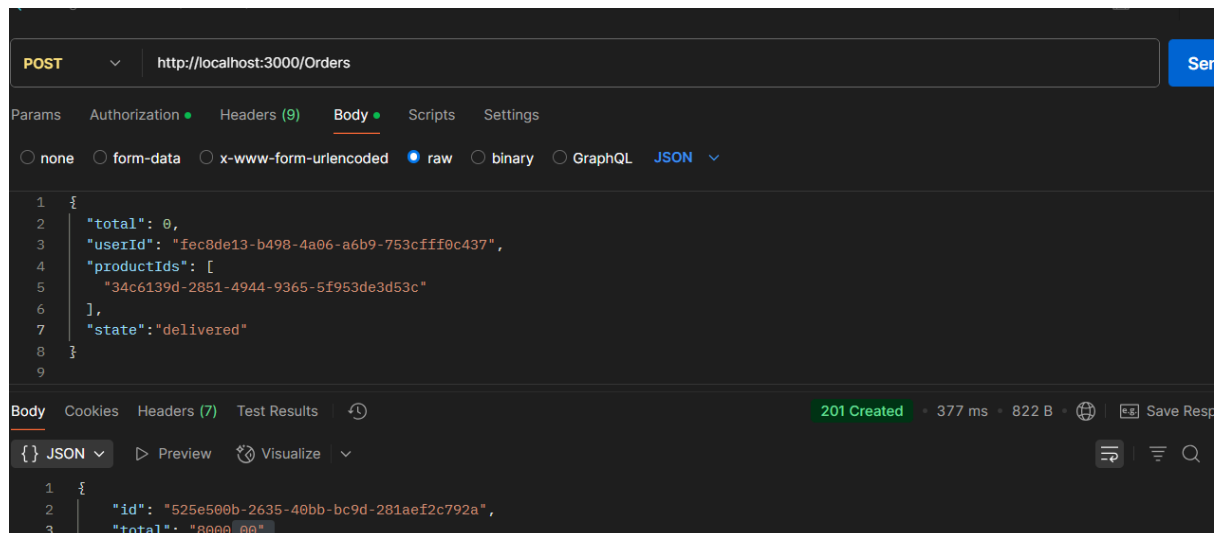
Administra la creación, consulta, actualización y cancelación de órdenes realizadas por los usuarios. Incluye control de acceso según roles (customer, admin, delivery) y maneja el estado de la orden durante su ciclo de vida (Pending, Preparing, OnTheWay, Delivered, Cancelled). Permite asociar productos a cada orden y consultar por usuario o por rango de fechas.

- **POST /orders**

Descripción: Crea una nueva orden. Solo los usuarios con rol customer pueden crear órdenes.

Autenticación: Requiere token JWT con rol de customer y admin.

Body:

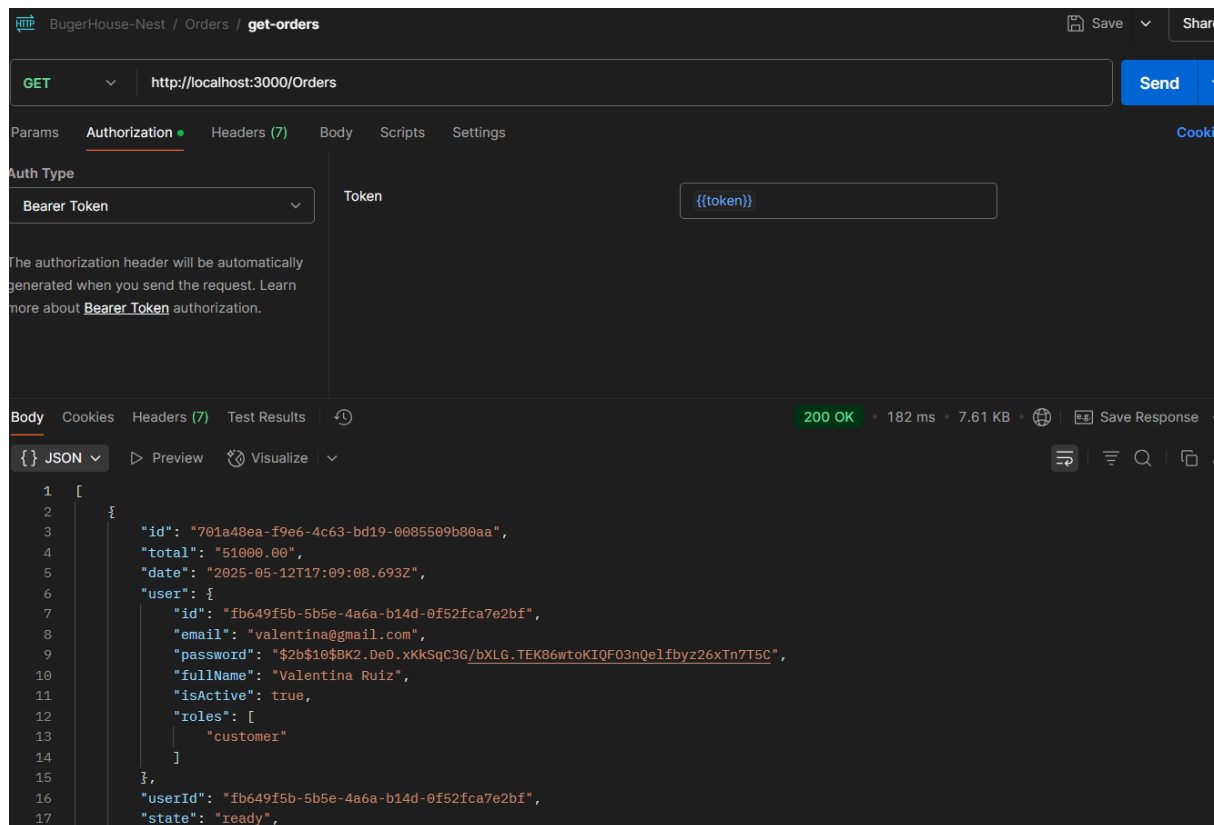


Respuestas:

- **201 Created:** Orden creada exitosamente.
- **400 Bad Request:** Datos inválidos.
- **403 Forbidden:** Solo usuarios con rol customer pueden crear órdenes.
- **GET /orders**

Descripción: Obtiene todas las órdenes.

Autenticación: Requiere token JWT con rol admin o delivery.

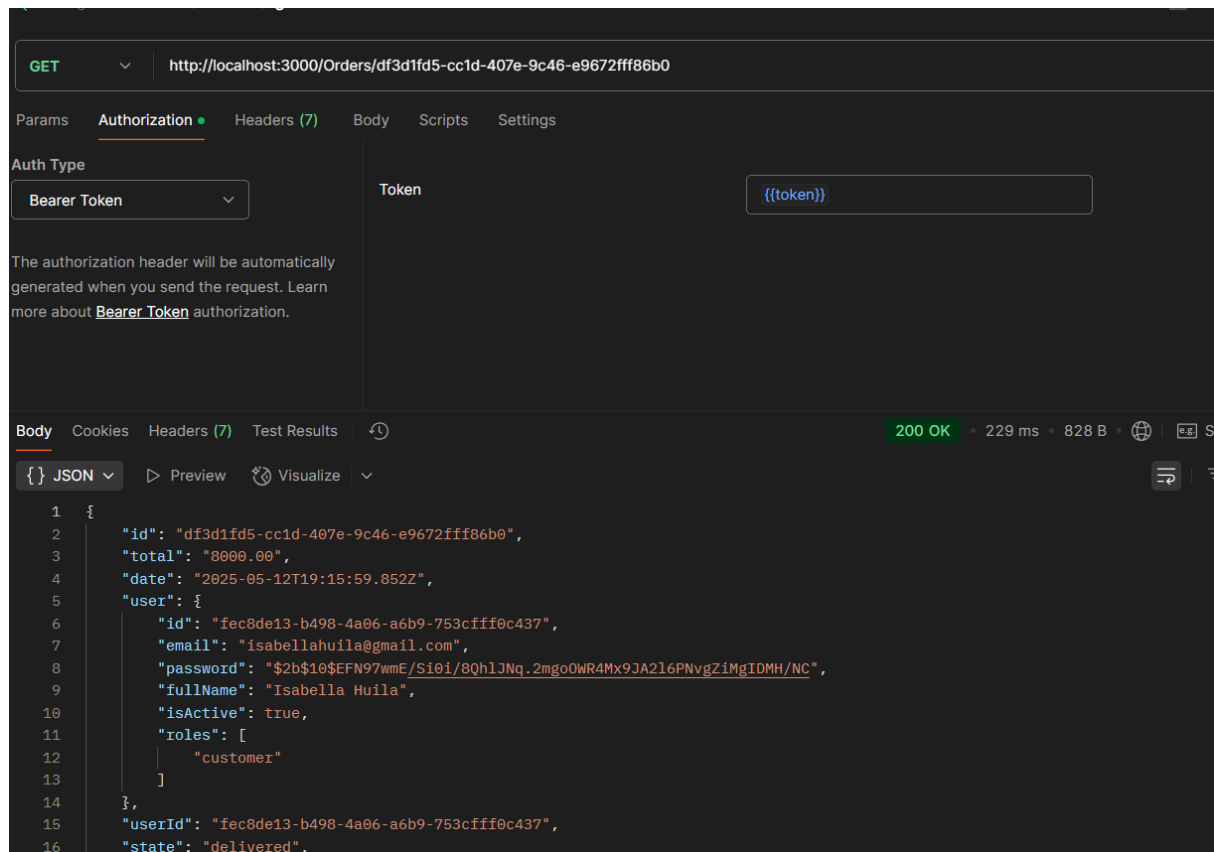


Respuestas:

- **200 OK:** Lista de órdenes.
- **403 Forbidden:** No autorizado.
- **GET /orders**

Descripción: Obtiene las órdenes del usuario autenticado.

Autenticación: Requiere token JWT con rol customer o admin.



Respuestas:

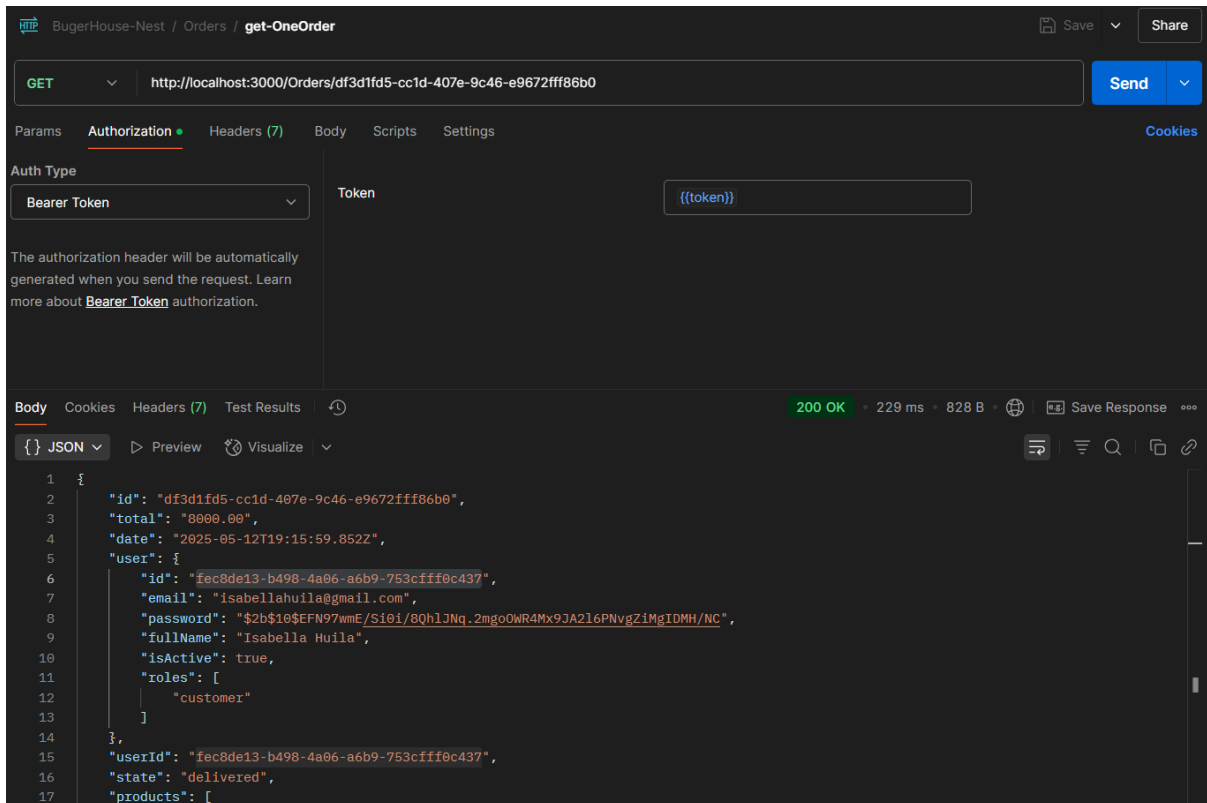
- **200 OK:** Lista de órdenes del usuario autenticado.
- **GET /orders/:id**

Descripción: Obtiene una orden por ID.

Autenticación: Requiere token JWT con rol admin, delivery o el dueño de la orden.

Parámetros (URL):

id (UUID): ID de la orden.



Respuestas:

- **200 OK:** Orden encontrada.
- **403 Forbidden:** Acceso denegado.
- **404 Not Found:** Orden no encontrada.
- **PATCH /orders/:id**

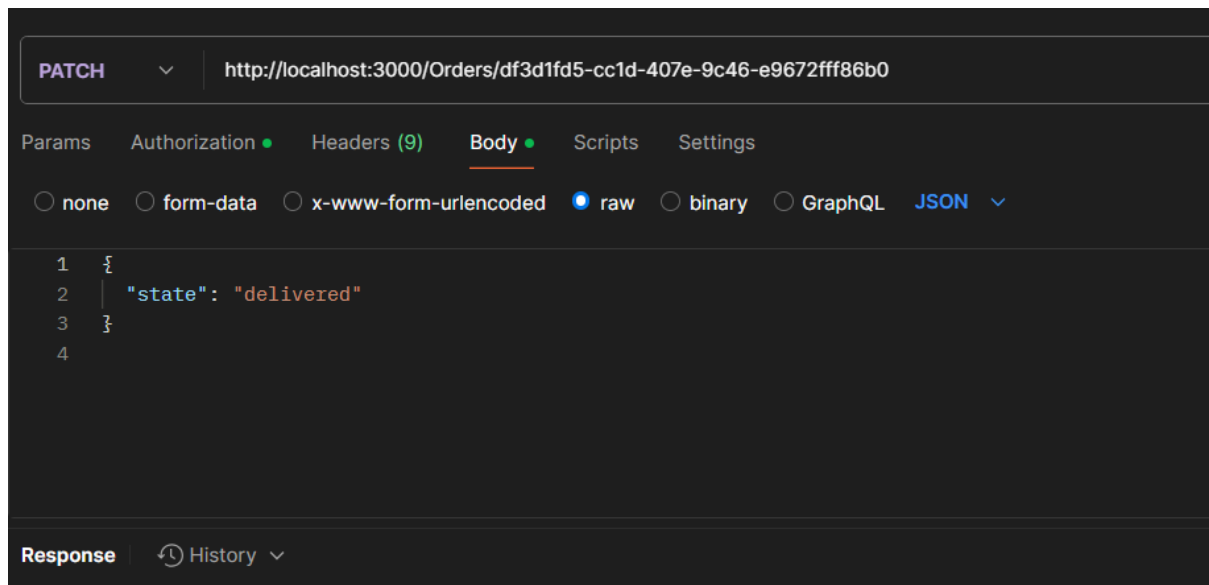
Autenticación: Requiere token JWT.

Parámetros (URL):

id (UUID): ID de la orden.

Body:

Los usuarios con rol delivery solo pueden cambiar el state.



Respuestas:

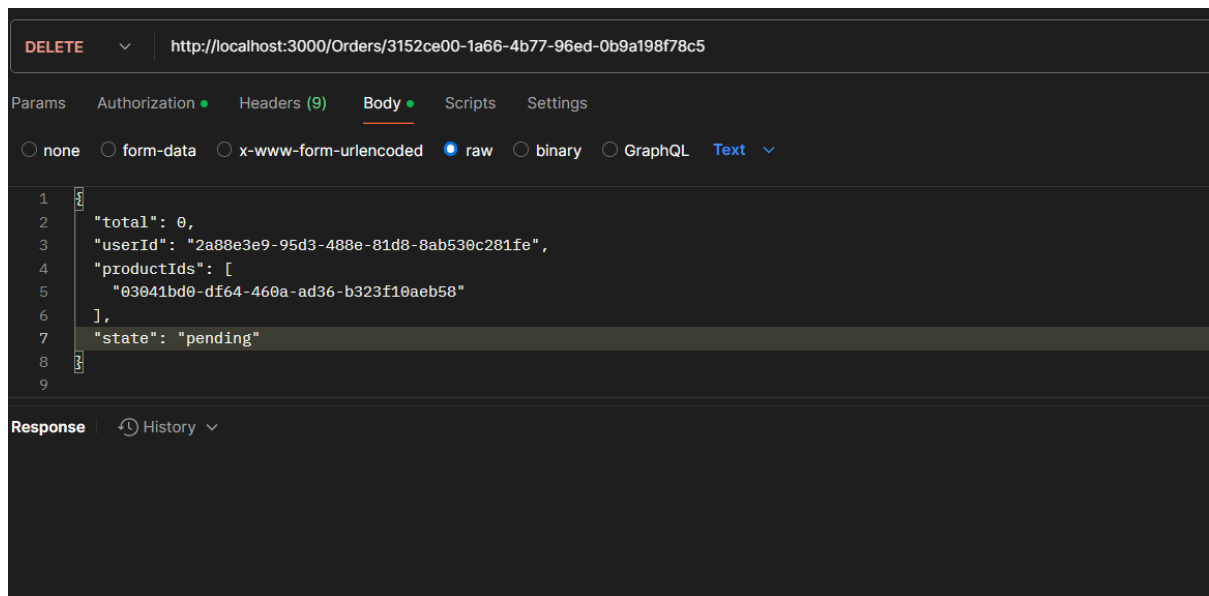
- **200 OK:** Orden actualizada.
- **400 Bad Request:** Datos inválidos.
- **403 Forbidden:** No autorizado.
- **404 Not Found:** Orden no encontrada.
- **DELETE /orders/:id**

Descripción: Cancela una orden (soft delete por cambio de estado).

Autenticación: Requiere token JWT con rol admin, delivery o el dueño de la orden.

Parámetros (URL):

id (UUID): ID de la orden.



Respuestas:

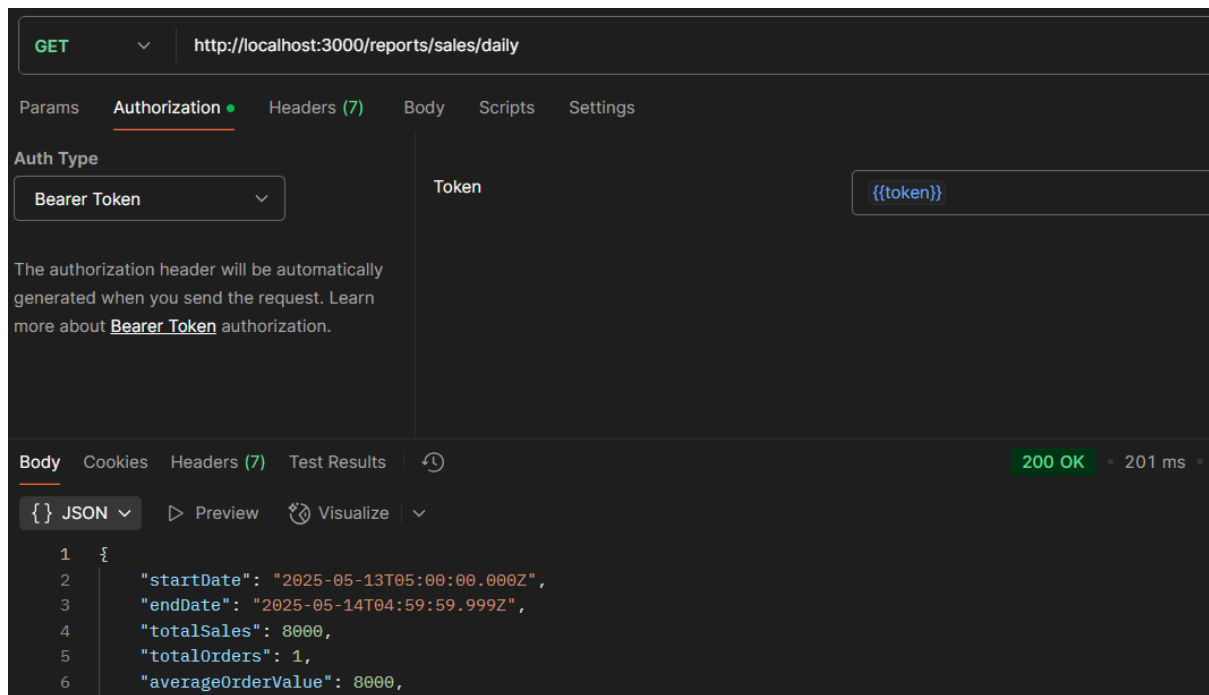
- **200 OK:** Orden cancelada exitosamente.
- **400 Bad Request:** No se puede cancelar una orden entregada o en camino.
- **403 Forbidden:** No autorizado.
- **404 Not Found:** Orden no encontrada.

Módulo Reporte:

El módulo Reports proporciona diversos endpoints para generar reportes analíticos sobre las ventas realizadas y los productos más vendidos. Los reportes pueden obtenerse por día, semana o mes. Está restringido a usuarios con el rol admin.

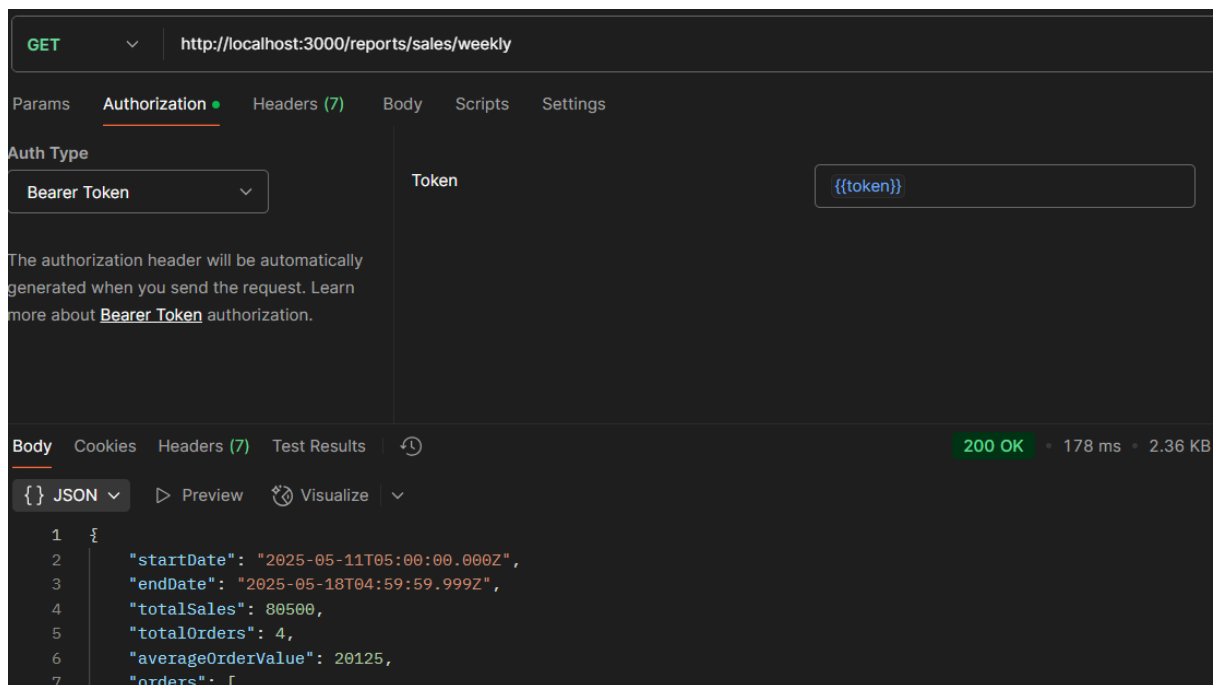
- GET /reports/sales/daily

Descripción: Retorna el reporte de ventas del día especificado (o del día actual si no se indica fecha).



- GET /reports/sales/weekly

Descripción: Retorna el reporte de ventas del mes correspondiente a la fecha proporcionada (o el mes actual).



- GET /reports/products/top-selling/daily

Descripción: Retorna los productos más vendidos del día especificado.

GET <http://localhost:3000/reports/products/top-selling/daily>

Params **Authorization** Headers (7) Body Scripts Settings

Auth Type

Bearer Token

Token `{{token}}`

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

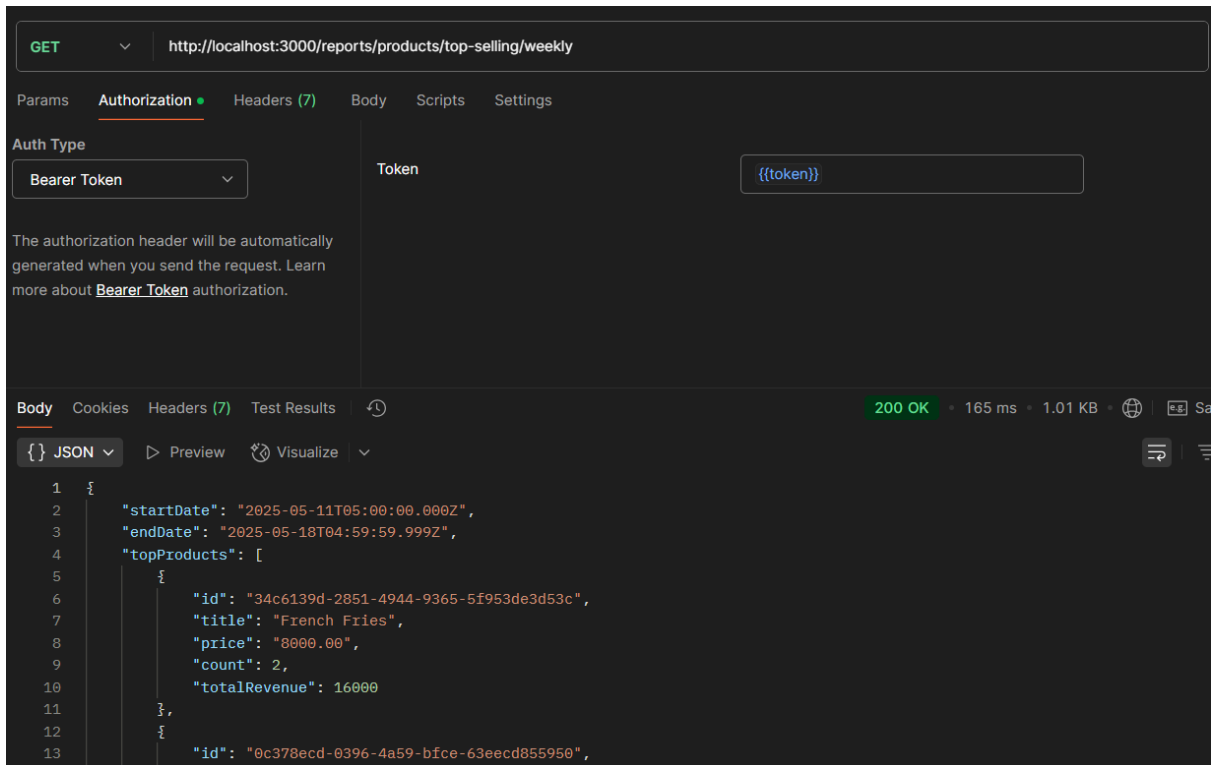
Body Cookies Headers (7) Test Results **200 OK** • 136

{ } JSON Preview Visualize

```
1 {
2   "startDate": "2025-05-13T05:00:00.000Z",
3   "endDate": "2025-05-14T04:59:59.999Z",
4   "topProducts": [
5     {
6       "id": "34c6139d-2851-4944-9365-5f953de3d53c",
7       "title": "French Fries",
8       "price": "8000.00",
9       "count": 1,
10      "totalRevenue": 8000
11    }
12  ]
13 }
```

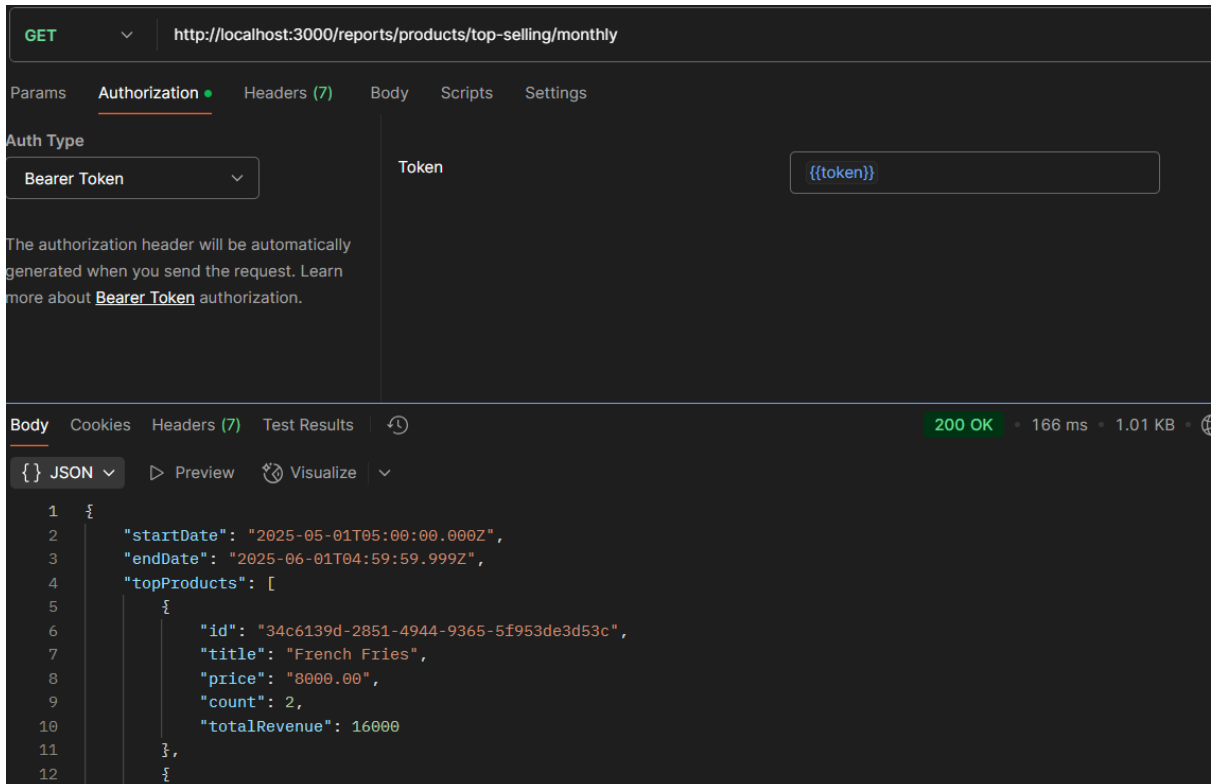
- **GET /reports/products/top-selling/weekly**

Descripción: Retorna los productos más vendidos durante la semana especificada.



- GET /reports/products/top-selling/monthly

Descripción: Retorna los productos más vendidos durante el mes especificado.



Características de autenticación, autorización y persistencia en la base de datos.

Autenticación

El sistema utiliza JSON Web Tokens (JWT) para la autenticación de usuarios:

Registro de usuarios:

Los usuarios se registran mediante el endpoint POST /users/register

Se validan los datos del usuario mediante DTOs como CreateUserDto

Las contraseñas se encriptan con bcrypt antes de almacenarlas en la base de datos

Al registrarse, se genera un token JWT

Inicio de sesión:

Se utiliza el endpoint POST /users/login con LoginUserDto

Se verifica el email del usuario en la base de datos

Se comparan las contraseñas usando bcrypt

Si las credenciales son correctas, se genera un token JWT

Estrategia JWT:

Se implementa mediante la clase JwtStrategy que extiende PassportStrategy

Extrae el token del encabezado de autorización (Bearer token)

Valida el token usando el secreto almacenado en variables de entorno

Verifica que el usuario exista y esté activo

Autorización

El sistema implementa un enfoque basado en roles para la autorización:

Roles de usuario:

Se definen en el enum ValidRoles: admin, customer y delivery

Por defecto, los usuarios nuevos tienen el rol "customer"

Guardias de roles:

Se utiliza UserRoleGuard para proteger rutas específicas

El guard verifica que el usuario tenga los roles requeridos para la operación

Si el usuario no tiene los permisos, se lanza una excepción ForbiddenException

Decoradores personalizados:

@Auth() para proteger rutas que requieren autenticación

@Auth(ValidRoles.admin) para rutas que requieren rol específico

@GetUser() para obtener el usuario autenticado en los controladores

Control de acceso:

Solo administradores pueden listar todos los usuarios (GET /users)

Los usuarios solo pueden editar su propio perfil, a menos que sean administradores

Solo administradores pueden eliminar usuarios

Persistencia en la Base de Datos

La persistencia de datos se implementa utilizando un enfoque de mapeo objeto-relacional (ORM) que facilita la interacción con la base de datos:

Entidades y relaciones: El sistema define entidades como usuarios, con relaciones claramente establecidas y restricciones de integridad.

Operaciones CRUD: Se implementan operaciones de creación, lectura, actualización y eliminación para gestionar los datos de forma eficiente.

Validación: Los datos se validan antes de ser almacenados para garantizar su integridad y seguridad.