API REST - SISTEMA DE GESTIÓN COMERCIAL

Documentación Técnica Completa
Endpoints Implementados y Especificación de Consultas

Commercial Management System Team
(Daniel Alonso Chavarro Chipatecua)

Noviembre 2025

Índice

1. Introducción

1.1. Propósito del Documento

Este documento presenta la especificación completa de la API REST del Sistema de Gestión Comercial, incluyendo todos los endpoints implementados, consultas a base de datos, y ejemplos de uso práctico.

1.2. Información del Sistema

Tecnologías Implementadas:

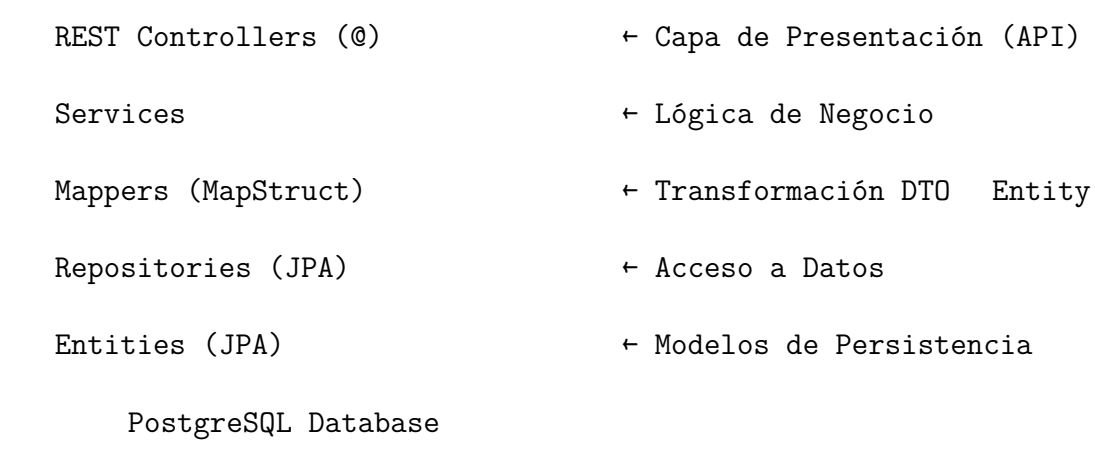
- **Framework**: Spring Boot 3.5.7
- **Lenguaje**: Java 21
- **ORM**: Spring Data JPA + Hibernate
- **Base de Datos**: PostgreSQL
- **Mapeo**: MapStruct 1.6.3
- **Build Tool**: Maven 3.x

URL Base de la API:

```
http://localhost:8080/api/v1
```

1.3. Arquitectura del Sistema

El sistema sigue una arquitectura multicapa:



1.4. Resumen de Endpoints

Cuadro 1: Total de Endpoints Implementados por Controlador

Controlador	Endpoint Base	Total Endpoints
DepartmentController	/api/v1/departments	5
CityController	/api/v1/cities	5
UserRoleController	/api/v1/user-roles	5
CategoryController	/api/v1/categories	6
UserController	/api/v1/users	9
StoreController	/api/v1/stores	7
ProductController	/api/v1/products	13
StoreProductController	/api/v1/store-products	5
SaleController	/api/v1/sales	9
SaleProductController	/api/v1/sale-products	5
TOTAL		**69**

2. Guía de Tipos de Parámetros

2.1. Formas de Pasar Información a la API

La API acepta información de tres formas diferentes según el tipo de operación:

2.1.1. 1. Path Parameters (Parámetros de Ruta)

¿Qué son? Valores que forman parte de la URL del endpoint.
¿Cuándo se usan? Para identificar un recurso específico (por ID).
Ejemplo:

```
GET /api/v1/products/550e8400-e29b-41d4-a716-446655440000
                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                     Path Parameter: id del producto
```

Endpoints que usan Path Parameters:

- GET /resource/{id} - Obtener por ID
- PUT /resource/{id} - Actualizar por ID
- DELETE /resource/{id} - Eliminar por ID

2.1.2. 2. Query Parameters (Parámetros de Consulta)

¿Qué son? Pares clave-valor que van después del símbolo ? en la URL.
¿Cuándo se usan? Para filtrar, buscar o paginar resultados.
Ejemplo:

```
GET /api/v1/products/search/by-price-range?minPrice=100&maxPrice=500
                                          ^^^^^^^^^^^^^^^^^^^^^^^^^
                                          Query Parameters
```

Sintaxis:

- Primer parámetro: `?nombre=valor`
- Parámetros adicionales: `&nombre=valor`

Endpoints que usan Query Parameters:

- GET /resource/search/* - Búsquedas y filtros
- GET /resource/analytics/* - Consultas analíticas

2.1.3. 3. Request Body (Cuerpo JSON)

¿Qué es? Datos en formato JSON enviados en el cuerpo de la petición HTTP.
¿Cuándo se usa? Para crear o actualizar recursos con información compleja.
Ejemplo:

```
POST /api/v1/products
Content-Type: application/json

{
  "productName": "Laptop Gaming Pro",
  "productDescription": "High-performance gaming laptop",
  "price": 1299.99
}
```

Endpoints que usan Request Body:

- POST /resource - Crear nuevo recurso
- PUT /resource/{id} - Actualizar recurso existente

2.2. Tabla Resumen: Método HTTP vs Tipo de Parámetro

Cuadro 2: Relación entre Método HTTP y Tipo de Parámetro

Método	Path Param	Query Param	Body JSON
GET (todos)			
GET (por ID)	(id)		
GET (búsqueda)		(filtros)	
POST			(datos)
PUT	(id)		(datos)
DELETE	(id)		

2.3. Tipos de Datos Comunes

Cuadro 3: Tipos de Datos por Campo

Tipo	Ejemplo	Descripción
Long	1, 2, 100	Números enteros (IDs secuenciales)
UUID	550e8400-e29b-41d4-a716-446655440000	Identificador único universal
String	"Laptop Gaming Pro"	Cadenas de texto
Double	1299.99	Números decimales (precios)
Integer	2, 5, 10	Números enteros (cantidades)
LocalDateTime	2025-11-22T10:30:00	Fecha y hora (formato ISO 8601)
Boolean	true, false	Valores lógicos

2.4. Formato de Fechas

Todas las fechas deben enviarse en formato ISO 8601:
Formato: `YYYY-MM-DDTHH:mm:ss`
Ejemplos válidos:

- `2025-11-22T10:30:00` - Con hora específica
- `2025-11-22T00:00:00` - Inicio del día
- `2025-11-22T23:59:59` - Fin del día

2.5. Headers Requeridos

Para peticiones POST y PUT (con body JSON):

```
Content-Type: application/json
```

3. 1. Departments API

3.1. Descripción

Gestiona las divisiones administrativas geográficas (departamentos/estados/provincias).

3.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/departments	Obtener todos los departamentos
GET	/departments/{id}	Obtener departamento por ID
POST	/departments	Crear nuevo departamento
PUT	/departments/{id}	Actualizar departamento
DELETE	/departments/{id}	Eliminar departamento

Cuadro 4: Endpoints del Departamento Controller

3.3. Especificación de Parámetros

Cuadro 5: Tipo de Datos por Endpoint - Departments

Endpoint	Path Params	Query Params	Body (JSON)
GET /departments	-	-	-
GET /departments/{id}	id (Long)	-	-
POST /departments	-	-	departmentName
PUT /departments/{id}	id (Long)	-	departmentName
DELETE /departments/{id}	id (Long)	-	-

3.4. Ejemplos de Peticiones

3.4.1. GET - Obtener Todos los Departamentos

Request:

```
GET http://localhost:8080/api/v1/departments
```

Response (200 OK):

```
[
  {
    "departmentId": 1,
    "departmentName": "Antioquia"
  },
  {
    "departmentId": 2,
    "departmentName": "Cundinamarca"
  }
]
```

3.4.2. POST - Crear Departamento

Tipo de Entrada: Body (JSON)
Estructura del Body:

- `departmentName` (String, obligatorio, max 64 caracteres)

Request:

```
POST http://localhost:8080/api/v1/departments
Content-Type: application/json

{
  "departmentName": "Risaralda"
}
```

Response (201 Created):

```
{
  "departmentId": 9,
  "departmentName": "Risaralda"
}
```

3.4.3. PUT - Actualizar Departamento

Tipo de Entrada: Path Parameter + Body (JSON)
Path Parameter:

- `id` (Long) - ID del departamento a actualizar

Estructura del Body:

- `departmentName` (String, obligatorio, max 64 caracteres)

Request:

```
PUT http://localhost:8080/api/v1/departments/9
Content-Type: application/json

{
  "departmentName": "Risaralda Actualizado"
}
```

Response (200 OK):

```
{
  "departmentId": 9,
  "departmentName": "Risaralda Actualizado"
}
```

3.4.4. DELETE - Eliminar Departamento

Tipo de Entrada: Path Parameter
Path Parameter:

- `id` (Long) - ID del departamento a eliminar

Request:

```
DELETE http://localhost:8080/api/v1/departments/9
```

Response (204 No Content):

```
(Sin contenido)
```

4. 2. Cities API

4.1. Descripción

Gestiona ciudades asociadas a departamentos.

4.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/cities	Obtener todas las ciudades
GET	/cities/{id}	Obtener ciudad por ID
POST	/cities	Crear nueva ciudad

Método	Endpoint	Descripción
PUT	/cities/{id}	Actualizar ciudad
DELETE	/cities/{id}	Eliminar ciudad

Cuadro 6: Endpoints del City Controller

4.3. Especificación de Parámetros

Cuadro 7: Tipo de Datos por Endpoint - Cities

Endpoint	Path Params	Query Params	Body (JSON)
GET /cities	-	-	-
GET /cities/{id}	id (Long)	-	-
POST /cities	-	-	cityName, department
PUT /cities/{id}	id (Long)	-	cityName, department
DELETE /cities/{id}	id (Long)	-	-

4.4. Ejemplos de Peticiones

4.4.1. POST - Crear Ciudad

Tipo de Entrada: Body (JSON)
Estructura del Body:

- `cityName` (String, obligatorio, max 64 caracteres)
- `department` (Object, obligatorio)
  - `departmentId` (Long, obligatorio)
  - `departmentName` (String, opcional - se ignora)

Request:

```
POST http://localhost:8080/api/v1/cities
Content-Type: application/json

{
  "cityName": "Pereira",
  "department": {
    "departmentId": 1,
    "departmentName": "Antioquia"
  }
}
```

Response (201 Created):

```
{
  "cityId": 16,
  "cityName": "Pereira",
  "department": {
    "departmentId": 1,
    "departmentName": "Antioquia"
  }
}
```

5. 3. User Roles API

5.1. Descripción

Gestiona los roles de usuario del sistema (ADMIN, USER, MANAGER).

5.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/user-roles	Obtener todos los roles
GET	/user-roles/{id}	Obtener rol por ID
POST	/user-roles	Crear nuevo rol
PUT	/user-roles/{id}	Actualizar rol
DELETE	/user-roles/{id}	Eliminar rol

Cuadro 8: Endpoints del User Role Controller

5.3. Ejemplos de Peticiones

5.3.1. POST - Crear Rol de Usuario

Request:

```
POST http://localhost:8080/api/v1/user-roles
Content-Type: application/json

{
  "role": "ADMIN"
}
```

Response (201 Created):

```
{
  "userRoleId": 1,
  "role": "ADMIN"
}
```

Valores Permitidos:

- ADMIN - Administrador del sistema
- USER - Usuario regular
- MANAGER - Gerente de tienda

6. 4. Categories API

6.1. Descripción

Gestiona las categorías de productos para clasificación y búsqueda.

6.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/categories	Obtener todas las categorías
GET	/categories/{id}	Obtener categoría por ID
POST	/categories	Crear nueva categoría
PUT	/categories/{id}	Actualizar categoría
DELETE	/categories/{id}	Eliminar categoría
GET	/categories/search/by-name?categoryName=	Buscar categoría por nombre

Cuadro 9: Endpoints del Category Controller

6.3. Consultas de Repositorio

```
// Busqueda por nombre exacto
Optional<CategoryEntity> findByCategoryName(String categoryName);
```

Listing 1: CategoryRepository - Query Methods

6.4. Especificación de Parámetros

Cuadro 10: Tipo de Datos por Endpoint - Categories

Endpoint	Path Params	Query Params	Body (JSON)
GET /categories	-	-	-
GET /categories/{id}	id (Long)	-	-
POST /categories	-	-	categoryName
PUT /categories/{id}	id (Long)	-	categoryName
DELETE /categories/{id}	id (Long)	-	-
GET .../by-name	-	categoryName (String)	-

6.5. Ejemplos de Peticiones

6.5.1. POST - Crear Categoría

Request:

```
POST http://localhost:8080/api/v1/categories
Content-Type: application/json

{
  "categoryName": "Electronics"
}
```

Response (201 Created):

```
{
  "categoryId": 1,
  "categoryName": "Electronics"
}
```

6.5.2. GET - Buscar por Nombre

Request:

```
GET http://localhost:8080/api/v1/categories/search/by-name?categoryName=
    Electronics
```

Response (200 OK):

```
{
  "categoryId": 1,
  "categoryName": "Electronics"
}
```

7. 5. Users API

7.1. Descripción

Gestiona usuarios del sistema con capacidades avanzadas de búsqueda geográfica.

7.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/users	Obtener todos los usuarios
GET	/users/{id}	Obtener usuario por ID
POST	/users	Crear nuevo usuario
PUT	/users/{id}	Actualizar usuario
DELETE	/users/{id}	Eliminar usuario
GET	/users/search/by-lastname?lastName=	Buscar por apellido
GET	/users/search/by-city?cityId=	Buscar por ID de ciudad
GET	/users/search/by-city-name?cityName=	Buscar por nombre de ciudad
GET	/users/search/by-department?departmentName=	Buscar por departamento

Cuadro 11: Endpoints del User Controller

7.3. Consultas de Repositorio

```
// Busqueda por apellido (case-insensitive)
List<UserEntity> findByLastNameIgnoreCase(String lastName);

// Busquedas geograficas
List<UserEntity> findByCity_CityId(Long cityCityId);
List<UserEntity> findByCity_CityName(String cityCityName);
List<UserEntity> findByCity_Department_DepartmentName(String departmentName);

// Busqueda con patrones (LIKE)
List<UserEntity> findByFirstNameLikeIgnoreCase(String firstName);
```

Listing 2: UserRepository - Query Methods

7.4. Especificación de Parámetros

Cuadro 12: Tipo de Datos por Endpoint - Users

Endpoint	Path Params	Query Params	Body (JSON)
GET /users	-	-	-
GET /users/{id}	id (UUID)	-	-
POST /users	-	-	firstName, lastName, username, email, password, phone, city, role
PUT /users/{id}	id (UUID)	-	firstName, lastName, username, email, password, phone, city, role
DELETE /users/{id}	id (UUID)	-	-
GET .../by-lastname	-	lastName (String)	-
GET .../by-city	-	cityId (Long)	-
GET .../by-city-name	-	cityName (String)	-
GET .../by-department	-	departmentName (String)	-

7.5. Ejemplos de Peticiones

7.5.1. POST - Crear Usuario

Tipo de Entrada: Body (JSON)
Estructura del Body:

- `firstName` (String, obligatorio, max 32 caracteres)
- `lastName` (String, obligatorio, max 32 caracteres)
- `username` (String, obligatorio, único)
- `email` (String, obligatorio, único)
- `password` (String, obligatorio - debe estar encriptado)
- `phone` (String, obligatorio, max 10 caracteres)
- `city` (Object, obligatorio)
  - `cityId` (Long, obligatorio)
- `role` (Object, obligatorio)
  - `userRoleId` (Long, obligatorio)

Request:

```
POST http://localhost:8080/api/v1/users
Content-Type: application/json

{
  "firstName": "Juan",
  "lastName": "Perez",
  "username": "jperez",
  "email": "juan.perez@example.com",
  "password": "$2a$10$encrypted.password",
  "phone": "3001234567",
  "city": {
    "cityId": 1
  },
  "role": {
    "userRoleId": 2
  }
}
```

Response (201 Created):

```
{
  "userId": "123e4567-e89b-12d3-a456-426614174000",
  "firstName": "Juan",
  "lastName": "Perez",
  "username": "jperez",
  "email": "juan.perez@example.com",
  "phone": "3001234567",
  "createdAt": "2025-11-22T10:30:00",
  "city": {
    "cityId": 1,
    "cityName": "Medellin"
  },
  "role": {
    "userRoleId": 2,
    "role": "USER"
  }
}
```

7.5.2. GET - Buscar por Ciudad

Request:

```
GET http://localhost:8080/api/v1/users/search/by-city-name?cityName=Medellin
```

Response (200 OK):

```
[
  {
    "userId": "123e4567-e89b-12d3-a456-426614174000",
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@example.com",
    "city": {
      "cityName": "Medellin"
    }
  }
]
```

8. 6. Products API

8.1. Descripción

Gestiona el catálogo de productos con búsquedas avanzadas y análisis de ventas.

8.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/products	Obtener todos los productos
GET	/products/{id}	Obtener producto por ID
POST	/products	Crear nuevo producto
PUT	/products/{id}	Actualizar producto
DELETE	/products/{id}	Eliminar producto
GET	/products/search/by-price-range?minPrice=&maxPrice=	Filtrar por rango de precio
GET	/products/search/sorted-by-price-asc	Ordenar por precio ascendente
GET	/products/search/sorted-by-price-desc	Ordenar por precio descendente
GET	/products/search/recent?date=	Productos creados después de fecha
GET	/products/search/by-store-id?storeId=	Productos por ID de tienda
GET	/products/search/by-store-name?storeName=	Productos por nombre de tienda
GET	/products/analytics/best-sellers	Productos más vendidos
GET	/products/analytics/top-best-sellers?limit=	Top N productos más vendidos

Cuadro 13: Endpoints del Product Controller

8.3. Consultas de Repositorio

```
// Filtrado por precio
List<ProductEntity> findByPriceBetween(Double minPrice, Double maxPrice);
List<ProductEntity> findByOrderByPriceAsc();
List<ProductEntity> findByOrderByPriceDesc();

// Filtrado por fecha
List<ProductEntity> findByCreatedAtAfter(LocalDateTime createdAtAfter);

// Busqueda por categoria (SQL nativo)
@Query(value = "SELECT p.* FROM product p " +
        "INNER JOIN product_category pc ON p.product_id = pc.product_id_fk " +
        "WHERE pc.category_id_fk = :categoryId", nativeQuery = true)
List<ProductEntity> findByCategoryId(@Param("categoryId") Long categoryId);

@Query(value = "SELECT p.* FROM product p " +
        "INNER JOIN product_category pc ON p.product_id = pc.product_id_fk " +
        "INNER JOIN category c ON pc.category_id_fk = c.category_id " +
        "WHERE c.category_name = :categoryName", nativeQuery = true)
```

```
19 List<ProductEntity> findByCategoryName(@Param("categoryName") String
       categoryName);
20
21 // Busqueda por tienda (JPQL)
22 @Query("SELECT sp.product FROM StoreProductEntity sp WHERE sp.store.storeId =
       :storeId")
23 List<ProductEntity> findByStoreId(@Param("storeId") UUID storeId);
24
25 @Query("SELECT sp.product FROM StoreProductEntity sp WHERE sp.store.storeName
       = :storeName")
26 List<ProductEntity> findByStoreName(@Param("storeName") String storeName);
```

Listing 3: ProductRepository - Query Methods

8.4. Especificación de Parámetros

Cuadro 14: Tipo de Datos por Endpoint - Products (Parte 1)

Endpoint	Path Params	Query Params	Body (JSON)
GET /products	-	-	-
GET /products/{id}	id (UUID)	-	-
POST /products	-	-	productName, productDescription, price
PUT /products/{id}	id (UUID)	-	productName, productDescription, price
DELETE /products/{id}	id (UUID)	-	-

Cuadro 15: Tipo de Datos por Endpoint - Products (Parte 2 - Búsquedas)

Endpoint	Path Params	Query Params	Body
GET .../by-price-range	-	minPrice (Double), maxPrice (Double)	-
GET .../sorted-by-price-asc	-	-	-
GET .../sorted-by-price-desc	-	-	-
GET .../recent	-	date (LocalDateTime, formato ISO)	-
GET .../by-store-id	-	storeId (UUID)	-
GET .../by-store-name	-	storeName (String)	-
GET .../best-sellers	-	-	-
GET .../top-best-sellers	-	limit (Integer)	-

8.5. Ejemplos de Peticiones

8.5.1. POST - Crear Producto

Tipo de Entrada: Body (JSON)
Estructura del Body:

- `productName` (String, obligatorio, max 128 caracteres)
- `productDescription` (String, obligatorio)
- `price` (Double, obligatorio, valor positivo)

Request:

```
POST http://localhost:8080/api/v1/products
Content-Type: application/json

{
  "productName": "Laptop Gaming Pro",
  "productDescription": "High-performance gaming laptop",
  "price": 1299.99
}
```

Response (201 Created):

```
{
  "productId": "550e8400-e29b-41d4-a716-446655440000",
  "productName": "Laptop Gaming Pro",
  "productDescription": "High-performance gaming laptop",
  "price": 1299.99,
  "createdAt": "2025-11-22T10:30:00",
  "updatedAt": "2025-11-22T10:30:00"
}
```

8.5.2. GET - Filtrar por Rango de Precio

Tipo de Entrada: Query Parameters
Query Parameters:

- `minPrice` (Double, obligatorio) - Precio mínimo
- `maxPrice` (Double, obligatorio) - Precio máximo

Request:

```
GET http://localhost:8080/api/v1/products/search/by-price-range?minPrice=100&
    maxPrice=500
```

Response (200 OK):

```
[
  {
    "productId": "123e4567-e89b-12d3-a456-426614174001",
    "productName": "Wireless Mouse",
    "price": 29.99
  },
  {
    "productId": "123e4567-e89b-12d3-a456-426614174002",
    "productName": "Mechanical Keyboard",
    "price": 149.99
  }
]
```

8.5.3. GET - Productos Más Vendidos

Request:

```
GET http://localhost:8080/api/v1/products/analytics/top-best-sellers?limit=5
```

Response (200 OK):

```
[
  {
    "productId": "123e4567-e89b-12d3-a456-426614174000",
    "productName": "Laptop Gaming Pro",
    "price": 1299.99,
    "totalSold": 85
  },
  {
    "productId": "123e4567-e89b-12d3-a456-426614174001",
    "productName": "Wireless Mouse",
    "price": 29.99,
    "totalSold": 120
  }
]
```

9. 7. Stores API

9.1. Descripción

Gestiona ubicaciones físicas de tiendas y su relación con productos.

9.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/stores	Obtener todas las tiendas
GET	/stores/{id}	Obtener tienda por ID
POST	/stores	Crear nueva tienda
PUT	/stores/{id}	Actualizar tienda
DELETE	/stores/{id}	Eliminar tienda
GET	/stores/search/by-city-id?cityId=	Buscar tiendas por ID de ciudad
GET	/stores/search/by-city-name?cityName=	Buscar tiendas por nombre de ciudad

Cuadro 16: Endpoints del Store Controller

9.3. Consultas de Repositorio

```
// Busquedas geograficas
List<StoreEntity> findByCity_CityId(Long cityCityId);
List<StoreEntity> findByCity_CityName(String cityCityName);
```

Listing 4: StoreRepository - Query Methods

9.4. Especificación de Parámetros

Cuadro 17: Tipo de Datos por Endpoint - Stores

Endpoint	Path Params	Query Params	Body (JSON)
GET /stores	-	-	-
GET /stores/{id}	id (UUID)	-	-
POST /stores	-	-	storeName, city
PUT /stores/{id}	id (UUID)	-	storeName, city
DELETE /stores/{id}	id (UUID)	-	-
GET .../by-city-id	-	cityId (Long)	-
GET .../by-city-name	-	cityName (String)	-

9.5. Ejemplos de Peticiones

9.5.1. POST - Crear Tienda

Tipo de Entrada: Body (JSON)
Estructura del Body:

- `storeName` (String, obligatorio, max 128 caracteres)
- `city` (Object, obligatorio)
  - `cityId` (Long, obligatorio)

Request:

```
POST http://localhost:8080/api/v1/stores
Content-Type: application/json

{
  "storeName": "Tech Store Centro",
  "city": {
    "cityId": 1
  }
}
```

Response (201 Created):

```
{
  "storeId": "550e8400-e29b-41d4-a716-446655440000",
  "storeName": "Tech Store Centro",
  "city": {
    "cityId": 1,
    "cityName": "Medellin"
  }
}
```

10. 8. Store Products API

10.1. Descripción

Gestiona el inventario de productos en cada tienda, incluyendo stock y ubicación física.

10.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/store-products	Obtener todos los registros de inventario
GET	/store-products/{id}	Obtener registro por ID
POST	/store-products	Crear nuevo registro de inventario
PUT	/store-products/{id}	Actualizar registro de inventario
DELETE	/store-products/{id}	Eliminar registro de inventario

Cuadro 18: Endpoints del Store Product Controller

10.3. Especificación de Parámetros

Cuadro 19: Tipo de Datos por Endpoint - Store Products

Endpoint	Path Params	Query Params	Body (JSON)
GET /store-products	-	-	-
GET /store-products/{id}	id (Long)	-	-
POST /store-products	-	-	stock, address, store, product
PUT /store-products/{id}	id (Long)	-	stock, address, store, product
DELETE /store-products/{id}	id (Long)	-	-

10.4. Ejemplos de Peticiones

10.4.1. POST - Agregar Producto a Tienda

Tipo de Entrada: Body (JSON)
Estructura del Body:

- `stock` (Long, obligatorio, valor positivo o cero)
- `address` (String, obligatorio) - Ubicación física en la tienda
- `store` (Object, obligatorio)
  - `storeId` (UUID, obligatorio)
- `product` (Object, obligatorio)
  - `productId` (UUID, obligatorio)

Request:

```
POST http://localhost:8080/api/v1/store-products
Content-Type: application/json

{
  "stock": 50,
```

```
  "address": "Pasillo A, Estante 3",
  "store": {
    "storeId": "550e8400-e29b-41d4-a716-446655440000"
  },
  "product": {
    "productId": "123e4567-e89b-12d3-a456-426614174000"
  }
}
```

Response (201 Created):

```
{
  "storeProductId": 1,
  "stock": 50,
  "address": "Pasillo A, Estante 3",
  "store": {
    "storeId": "550e8400-e29b-41d4-a716-446655440000",
    "storeName": "Tech Store Centro"
  },
  "product": {
    "productId": "123e4567-e89b-12d3-a456-426614174000",
    "productName": "Laptop Gaming Pro"
  }
}
```

11. 9. Sales API

11.1. Descripción

Gestiona transacciones de venta con análisis y reportes.

11.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/sales	Obtener todas las ventas
GET	/sales/{id}	Obtener venta por ID
POST	/sales	Crear nueva venta
PUT	/sales/{id}	Actualizar venta
DELETE	/sales/{id}	Eliminar venta
GET	/sales/search/by-user-id?userId=	Ventas por ID de usuario
GET	/sales/search/by-user-firstname?firstName=	Ventas por nombre de usuario
GET	/sales/search/by-min-total-amount?amount=	Ventas con monto mínimo
GET	/sales/analytics/total-by-date?date=	Total de ventas por fecha

Cuadro 20: Endpoints del Sale Controller

11.3. Consultas de Repositorio

```
// Busquedas por usuario
List<SaleEntity> findByUser_UserId(UUID userUserId);
List<SaleEntity> findByUser_FirstName(String userFirstName);
```

```
4
5  // Filtrado por monto
6  List<SaleEntity> findByTotalAmountAfter(Long totalAmountAfter);
7
8  // Agregacion por fecha (JPQL)
9  @Query("SELECT SUM(s.totalAmount) FROM SaleEntity s WHERE s.saleDate = ?1")
10 Long sumTotalAmountBySaleDate(LocalDateTime saleDate);
```

Listing 5: SaleRepository - Query Methods

11.4. Especificación de Parámetros

Cuadro 21: Tipo de Datos por Endpoint - Sales (Parte 1)

Endpoint	**Path Params**	**Query Params**	**Body (JSON)**
GET /sales	-	-	-
GET /sales/{id}	id (UUID)	-	-
POST /sales	-	-	totalAmount, user
PUT /sales/{id}	id (UUID)	-	totalAmount, user
DELETE /sales/{id}	id (UUID)	-	-

Cuadro 22: Tipo de Datos por Endpoint - Sales (Parte 2 - Búsquedas)

Endpoint	**Path**	**Query Params**	**Body**
GET .../by-user-id	-	userId (UUID)	-
GET .../by-user-firstname	-	firstName (String)	-
GET .../by-min-total-amount	-	amount (Long, en centavos)	-
GET .../total-by-date	-	date (LocalDateTime, formato ISO)	-

11.5. Ejemplos de Peticiones

11.5.1. POST - Crear Venta

Tipo de Entrada: Body (JSON)
Estructura del Body:

- `totalAmount` (Long, obligatorio) - Monto total en centavos
- `user` (Object, obligatorio)
  - `userId` (UUID, obligatorio)

Nota: El campo `totalAmount` debe ser un número entero que representa el monto en centavos. Por ejemplo: $1,299.99 = 129999 centavos.
Request:

```
POST http://localhost:8080/api/v1/sales
Content-Type: application/json

{
  "totalAmount": 129999,
  "user": {
    "userId": "123e4567-e89b-12d3-a456-426614174000"
  }
}
```

Response (201 Created):

```
{
  "saleId": "987e6543-e21b-45d6-b789-123456789abc",
  "saleDate": "2025-11-22T10:30:00",
  "totalAmount": 129999,
  "user": {
    "userId": "123e4567-e89b-12d3-a456-426614174000",
    "firstName": "Juan",
    "lastName": "Perez"
  }
}
```

11.5.2. GET - Analítica de Ventas Diarias

Request:

```
GET http://localhost:8080/api/v1/sales/analytics/total-by-date?date=2025-11-22
    T00:00:00
```

Response (200 OK):

```
{
  "date": "2025-11-22",
  "totalAmount": 5849970,
  "totalAmountFormatted": "$58,499.70"
}
```

12. 10. Sale Products API

12.1. Descripción

Gestiona la relación muchos-a-muchos entre ventas y productos (líneas de venta).

12.2. Endpoints Implementados

Método	Endpoint	Descripción
GET	/sale-products	Obtener todas las líneas de venta
GET	/sale-products/{id}	Obtener línea de venta por ID
POST	/sale-products	Crear nueva línea de venta
PUT	/sale-products/{id}	Actualizar línea de venta
DELETE	/sale-products/{id}	Eliminar línea de venta

Cuadro 23: Endpoints del Sale Product Controller

12.3. Consultas de Repositorio

```
// Productos mas vendidos (con GROUP BY)
@Query("SELECT sp.product FROM SaleProductEntity sp " +
        "GROUP BY sp.product " +
        "ORDER BY SUM(sp.quantity) DESC")
List<ProductEntity> findBestSellingProducts();

// Top N productos mas vendidos
@Query("SELECT sp.product FROM SaleProductEntity sp " +
        "GROUP BY sp.product " +
        "ORDER BY SUM(sp.quantity) DESC " +
        "LIMIT :limit")
List<ProductEntity> findTopBestSellingProducts(@Param("limit") int limit);
```

Listing 6: SaleProductRepository - Query Methods

12.4. Especificación de Parámetros

Cuadro 24: Tipo de Datos por Endpoint - Sale Products

Endpoint	Path Params	Query Params	Body (JSON)
GET /sale-products	-	-	-
GET /sale-products/{id}	id (Long)	-	-
POST /sale-products	-	-	quantity, unitPrice, sale, product
PUT /sale-products/{id}	id (Long)	-	quantity, unitPrice, sale, product
DELETE /sale-products/{id}	id (Long)	-	-

12.5. Ejemplos de Peticiones

12.5.1. POST - Agregar Producto a Venta

Tipo de Entrada: Body (JSON)
Estructura del Body:

- `quantity` (Integer, obligatorio, valor positivo) - Cantidad de unidades
- `unitPrice` (Double, obligatorio, valor positivo) - Precio unitario
- `sale` (Object, obligatorio)
  - `saleId` (UUID, obligatorio)
- `product` (Object, obligatorio)
  - `productId` (UUID, obligatorio)

Request:

```
POST http://localhost:8080/api/v1/sale-products
Content-Type: application/json

{
  "quantity": 2,
  "unitPrice": 1299.99,
  "sale": {
    "saleId": "987e6543-e21b-45d6-b789-123456789abc"
  },
  "product": {
    "productId": "123e4567-e89b-12d3-a456-426614174000"
  }
}
```

Response (201 Created):

```
{
  "saleProductId": 1,
  "quantity": 2,
  "unitPrice": 1299.99,
  "sale": {
    "saleId": "987e6543-e21b-45d6-b789-123456789abc"
  },
  "product": {
    "productId": "123e4567-e89b-12d3-a456-426614174000",
    "productName": "Laptop Gaming Pro"
  }
}
```

13. Códigos de Respuesta HTTP

13.1. Códigos de Éxito

Cuadro 25: Códigos de Respuesta Exitosos

Código	Significado	Uso
200	OK	Solicitud GET, PUT exitosa
201	Created	Recurso creado con POST
204	No Content	DELETE exitoso

13.2. Códigos de Error

Cuadro 26: Códigos de Error Comunes

Código	Significado	Causa
400	Bad Request	Datos inválidos en el body
404	Not Found	Recurso no encontrado
409	Conflict	Violación de constraint único
500	Internal Server Error	Error no controlado del servidor

14. Seguridad y Buenas Prácticas

14.1. Validaciones Implementadas

A Nivel de Entidad:

- Campos `nullable = false` para datos obligatorios
- Restricciones de longitud (`length = 128`)
- Constraints únicos (`unique = true`)
- Timestamps automáticos con `@CreationTimestamp` y `@UpdateTimestamp`

14.2. Recomendaciones de Seguridad

Para Implementación Futura:

- Autenticación JWT para proteger endpoints
- Autorización basada en roles (ADMIN, USER, MANAGER)
- Validación de entrada con Bean Validation (@Valid, @NotNull, @Min, etc.)
- Rate limiting para prevenir abuso
- HTTPS en producción
- Sanitización de inputs para prevenir SQL Injection

15. Ejemplos Prácticos Completos

15.1. Flujo Completo: Crear una Venta

A continuación se presenta un ejemplo completo de cómo crear una venta desde cero, incluyendo todos los pasos necesarios:

15.1.1. Paso 1: Verificar Usuario (GET con Path Parameter)

Objetivo: Obtener información del usuario que realizará la compra.

```
GET /api/v1/users/123e4567-e89b-12d3-a456-426614174000
    ^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    Endpoint    Path Parameter (userId)
```

Tipo de parámetro: Path Parameter
Response:

```
{
  "userId": "123e4567-e89b-12d3-a456-426614174000",
  "firstName": "Juan",
  "lastName": "Perez"
}
```

15.1.2. Paso 2: Buscar Productos por Precio (GET con Query Parameters)

Objetivo: Encontrar productos disponibles en un rango de precio.

```
GET /api/v1/products/search/by-price-range?minPrice=100&maxPrice=500
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^             ^^^^^^^^^^^^^^^^^^^^^^^^^^
    Endpoint de búsqueda                      Query Parameters
```

Tipo de parámetros: Query Parameters (`minPrice`, `maxPrice`)
Response:

```
[
  {
    "productId": "550e8400-e29b-41d4-a716-446655440000",
    "productName": "Mechanical Keyboard",
    "price": 149.99
  }
]
```

15.1.3. Paso 3: Crear la Venta (POST con Body JSON)

Objetivo: Registrar la transacción de venta.

```
POST /api/v1/sales
Content-Type: application/json

{
  "totalAmount": 14999,
  "user": {
    "userId": "123e4567-e89b-12d3-a456-426614174000"
  }
}
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Request Body (JSON)
```

Tipo de parámetro: Body JSON
Nota: El totalAmount es 14999 centavos = $149.99
Response:

```
{
  "saleId": "987e6543-e21b-45d6-b789-123456789abc",
  "saleDate": "2025-11-22T10:30:00",
  "totalAmount": 14999
}
```

15.1.4. Paso 4: Agregar Producto a la Venta (POST con Body JSON)

Objetivo: Registrar el detalle de productos en la venta.

```
POST /api/v1/sale-products
Content-Type: application/json

{
  "quantity": 1,
  "unitPrice": 149.99,
  "sale": {
    "saleId": "987e6543-e21b-45d6-b789-123456789abc"
  },
  "product": {
```

```
    "productId": "550e8400-e29b-41d4-a716-446655440000"
  }
}
```

Tipo de parámetro: Body JSON con objetos anidados

15.2. Comparación de Métodos de Envío

15.2.1. Ejemplo 1: Path Parameter vs Query Parameter

Path Parameter (Recurso específico):

```
GET /api/v1/users/123e4567-e89b-12d3-a456-426614174000
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                  El ID es parte de la ruta
```

Query Parameter (Búsqueda/Filtro):

```
GET /api/v1/users/search/by-city-name?cityName=Medellin
                                      ^^^^^^^^^^^^^^^^^
                                      El filtro va después de ?
```

¿Cuál usar?

- Path Parameter: Cuando necesitas UN recurso específico por ID
- Query Parameter: Cuando buscas MÚLTIPLES recursos con filtros

15.2.2. Ejemplo 2: Query Parameters Múltiples

Un solo parámetro:

```
GET /api/v1/users/search/by-lastname?lastName=Perez
```

Múltiples parámetros (separados por &):

```
GET /api/v1/products/search/by-price-range?minPrice=100&maxPrice=500
                                           ^           ^
                                           Primer      Segundo
                                           parámetro   parámetro (con &)
```

15.2.3. Ejemplo 3: Body JSON Simple vs Anidado

Body Simple (solo campos primitivos):

```
POST /api/v1/categories
Content-Type: application/json

{
  "categoryName": "Electronics"
}
```

Body Anidado (con objetos relacionados):

```
POST /api/v1/users
Content-Type: application/json

{
  "firstName": "Juan",
  "lastName": "Perez",
  "username": "jperez",
```

```
  "email": "juan@example.com",
  "password": "$2a$10$encrypted",
  "phone": "3001234567",
  "city": {                    <- Objeto anidado
    "cityId": 1
  },
  "role": {                    <- Objeto anidado
    "userRoleId": 2
  }
}
```

15.3. Errores Comunes y Soluciones

15.3.1. Error 1: Mezclar Path y Query Parameters

INCORRECTO:

```
GET /api/v1/products?id=550e8400-e29b-41d4-a716-446655440000
```

CORRECTO:

```
GET /api/v1/products/550e8400-e29b-41d4-a716-446655440000
```

Explicación: Los IDs van en la ruta, no como query parameter.

15.3.2. Error 2: Olvidar Content-Type en POST/PUT

INCORRECTO:

```
POST /api/v1/products
{
  "productName": "Laptop"
}
```

CORRECTO:

```
POST /api/v1/products
Content-Type: application/json

{
  "productName": "Laptop"
}
```

15.3.3. Error 3: Formato Incorrecto de UUID

INCORRECTO:

```
{
  "userId": "123456"
}
```

CORRECTO:

```
{
  "userId": "123e4567-e89b-12d3-a456-426614174000"
}
```

Formato UUID: 8-4-4-4-12 caracteres hexadecimales separados por guiones.

15.3.4. Error 4: Formato Incorrecto de Fecha

INCORRECTO:

```
GET /api/v1/products/search/recent?date=22/11/2025
```

CORRECTO:

```
GET /api/v1/products/search/recent?date=2025-11-22T00:00:00
```

15.4. Checklist de Validación

Antes de hacer una petición, verifica:

Cuadro 27: Checklist de Validación de Peticiones

Verificación	
¿El método HTTP es correcto? (GET, POST, PUT, DELETE)	
¿Los Path Parameters están en la URL?	
¿Los Query Parameters empiezan con ? y se separan con &?	
¿El Body JSON está bien formateado?	
¿Incluí Content-Type: application/json para POST/PUT?	
¿Los UUIDs tienen el formato correcto?	
¿Las fechas están en formato ISO 8601?	
¿Los campos obligatorios están presentes?	
¿Los tipos de datos son correctos? (Long, String, Double)	

16. Conclusiones

16.1. Estado Actual del Sistema

El Sistema de Gestión Comercial cuenta con una API REST completa que incluye:

- **69 Endpoints Implementados** distribuidos en 10 controladores
- **CRUD Completo** para todas las entidades del sistema
- **Búsquedas Avanzadas** con filtros múltiples
- **Analítica** de ventas y productos más vendidos
- **Soporte Geográfico** para operaciones multi-ubicación

16.2. Valor para el Negocio

El sistema proporciona una base sólida para:

- **Eficiencia Operativa**: Gestión centralizada de inventario y ventas

- **Toma de Decisiones**: Analítica de ventas y productos
- **Escalabilidad**: Arquitectura preparada para crecimiento
- **Integración**: API REST estándar para múltiples clientes
- **Mantenibilidad**: Código limpio y bien documentado

17. Apéndices

17.1. A. Tabla Resumen: Todos los Endpoints y sus Parámetros

HTTP	Endpoint	Path Params	Query Params	Body JSON
DEPARTMENTS				
GET	/departments	-	-	-
GET	/departments/{id}	id (Long)	-	-
POST	/departments	-	-	departmentName
PUT	/departments/{id}	id (Long)	-	departmentName
DELETE	/departments/{id}	id (Long)	-	-
CITIES				
GET	/cities	-	-	-
GET	/cities/{id}	id (Long)	-	-
POST	/cities	-	-	cityName, department
PUT	/cities/{id}	id (Long)	-	cityName, department
DELETE	/cities/{id}	id (Long)	-	-
USER ROLES				
GET	/user-roles	-	-	-
GET	/user-roles/{id}	id (Long)	-	-
POST	/user-roles	-	-	role
PUT	/user-roles/{id}	id (Long)	-	role
DELETE	/user-roles/{id}	id (Long)	-	-
CATEGORIES				
GET	/categories	-	-	-
GET	/categories/{id}	id (Long)	-	-
POST	/categories	-	-	categoryName
PUT	/categories/{id}	id (Long)	-	categoryName
DELETE	/categories/{id}	id (Long)	-	-
GET	/categories/search/by-name	-	categoryName	-
USERS				
GET	/users	-	-	-
GET	/users/{id}	id (UUID)	-	-

HTTP	Endpoint	Path Params	Query Params	Body JSON
POST	/users	-	-	firstName, lastName, username, email, password, phone, city, role
PUT	/users/{id}	id (UUID)	-	firstName, lastName, username, email, password, phone, city, role
DELETE	/users/{id}	id (UUID)	-	-
GET	/users/search/by-lastname	-	lastName	-
GET	/users/search/by-city	-	cityId	-
GET	/users/search/by-city-name	-	cityName	-
GET	/users/search/by-department	-	departmentName	-
PRODUCTS				
GET	/products	-	-	-
GET	/products/{id}	id (UUID)	-	-
POST	/products	-	-	productName, productDescription, price
PUT	/products/{id}	id (UUID)	-	productName, productDescription, price
DELETE	/products/{id}	id (UUID)	-	-
GET	/products/search/by-price-range	-	minPrice, maxPrice	-
GET	/products/search/sorted-by-price-asc	-	-	-
GET	/products/search/sorted-by-price-desc	-	-	-
GET	/products/search/recent	-	date	-

HTTP	Endpoint	Path Params	Query Params	Body JSON
GET	/products/search/by-store-id	-	storeId	-
GET	/products/search/by-store-name	-	storeName	-
GET	/products/analytics/best-sellers	-	-	-
GET	/products/analytics/top-best-sellers	-	limit	-
STORES				
GET	/stores	-	-	-
GET	/stores/{id}	id (UUID)	-	-
POST	/stores	-	-	storeName, city
PUT	/stores/{id}	id (UUID)	-	storeName, city
DELETE	/stores/{id}	id (UUID)	-	-
GET	/stores/search/by-city-id	-	cityId	-
GET	/stores/search/by-city-name	-	cityName	-
STORE PRODUCTS				
GET	/store-products	-	-	-
GET	/store-products/{id}	id (Long)	-	-
POST	/store-products	-	-	stock, address, store, product
PUT	/store-products/{id}	id (Long)	-	stock, address, store, product
DELETE	/store-products/{id}	id (Long)	-	-
SALES				
GET	/sales	-	-	-
GET	/sales/{id}	id (UUID)	-	-
POST	/sales	-	-	totalAmount, user
PUT	/sales/{id}	id (UUID)	-	totalAmount, user

HTTP	Endpoint	Path Params	Query Params	Body JSON
DELETE	/sales/{id}	id (UUID)	-	-
GET	/sales/search/by-user-id	-	userId	-
GET	/sales/search/by-user-firstname	-	firstName	-
GET	/sales/search/by-min-total-amount	-	amount	-
GET	/sales/analytics/total-by-date	-	date	-
SALE PRODUCTS				
GET	/sale-products	-	-	-
GET	/sale-products/{id}	id (Long)	-	-
POST	/sale-products	-	-	quantity, unitPrice, sale, product
PUT	/sale-products/{id}	id (Long)	-	quantity, unitPrice, sale, product
DELETE	/sale-products/{id}	id (Long)	-	-

Cuadro 28: Resumen Completo de Endpoints y Tipos de Parámetros

17.2. B. Variables de Entorno

Para usar la colección de Postman, configure las siguientes variables:

Cuadro 29: Variables de Entorno para Postman

Variable	Valor de Ejemplo
base_url	http://localhost:8080/api/v1
department_id	1
city_id	1
user_id	123e4567-e89b-12d3-a456-426614174000
product_id	550e8400-e29b-41d4-a716-446655440000
store_id	987e6543-e21b-45d6-b789-123456789abc
sale_id	abc123de-f456-7890-ghij-klmnopqrst12

Nota: puede que las UUID (aquellas id alfanuméricos) se deban remplazar por las generadas.

17.3. C. Configuración de Base de Datos

application.properties:

```
# Database Configuration
spring.datasource.url=jdbc:postgresql://localhost:5432/commercial_db
spring.datasource.username=postgres
spring.datasource.password=your_password
spring.datasource.driver-class-name=org.postgresql.Driver

# JPA Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
   PostgreSQLDialect

# Server Configuration
server.port=8080
server.servlet.context-path=/
```