

CURSO DRF & DJANGO ORM — FASE I

ISS-12: Consulta y Anulación de Ventas

Lectura de Ventas, Preservación de Precio Histórico y Anulación Idempotente con Reposición de Inventario

¿Qué construiremos en este ISS?

Objetivo Técnico Principal

- Consultar ventas registradas con listado (GET /api/sales/) y detalle completo (GET /api/sales/<id>/).
- Garantizar el principio de Inmutabilidad del Precio Histórico: las líneas conservan unit_price independientemente de cambios futuros en Product.price.
- Implementar la Anulación Blanda (DELETE /api/sales/<id>/) mediante la invocación del método sale.void().
- Asegurar la Idempotencia en la devolución de stock: anular 2 o más veces mantiene el inventario exacto sin reponer múltiples veces.

Reglas de Dominio y Seguridad

- Cero Borrado Físico: prohibido el uso de QuerySet.delete() sobre la tabla sales o product_sales.
- Acceso OPEN (Fase I): endpoints en permiso AllowAny, reutilizando la infraestructura establecida en ISS-11.
- Anidación Automática: las líneas de venta (ProductSale) no poseen endpoint directo; se consumen anidadas dentro de Sale.
- Optimización SQL: uso de select_related('client') y prefetch_related('lines__product') para evitar el problema N+1.

Punto de Partida y Elección de SaleReadViewSet

Mezcla de Mixins en SaleReadViewSet

SaleReadViewSet no hereda de ModelViewSet ni de ReadOnlyModelViewSet, sino que combina exactamente los mixins requeridos:

- ✓ **ListModelMixin:** Proporciona la acción `.list()` para responder a **GET /api/sales/**
- ✓ **RetrieveModelMixin:** Proporciona la acción `.retrieve()` para responder a **GET /api/sales/<id>/**
- ✓ **DestroyModelMixin:** Proporciona la acción `.destroy()` para responder a **DELETE /api/sales/<id>/**
- ✓ **GenericViewSet:** Aporta la infraestructura base de DRF sin imponer handlers de creación ni actualización

Por qué NO se incluyeron UpdateModelMixin ni CreateModelMixin:

1. Los totales de una venta NO se editan vía CRUD (requieren registro transaccional en SaleQuerySet.register).
2. El alta de la venta ya está resuelta mediante SaleRegisterAPIView (POST /api/sales/).

Implementación de Vistas — SaleReadViewSet

Código de apps/sale/views.py (Extensión para Lectura y Anulación)

```
from rest_framework import mixins, viewsets
from rest_framework.permissions import AllowAny
from apps.sale.models import Sale
from apps.sale.serializers import SaleSerializer

class SaleReadViewSet(
    mixins.ListModelMixin,
    mixins.RetrieveModelMixin,
    mixins.DestroyModelMixin,
    viewsets.GenericViewSet,
):
    permission_classes = [AllowAny]
    serializer_class = SaleSerializer
    lookup_value_converter = "int"

    def get_queryset(self):
        return Sale.objects.select_related("client").prefetch_related("lines__product")

    def perform_destroy(self, instance):
        instance.void()
```

Enrutamiento Manual con `as_view()` en `apps/sale/urls.py`

Parche en `apps/sale/urls.py`

```
from django.urls import path
from apps.sale.views import (
    SaleCollectionAPIView,
    SaleReadViewSet,
)

urlpatterns = [
    path(
        "sales/",
        SaleCollectionAPIView.as_view(),
        name="sale-collection",
    ),
    path(
        "sales/<int:pk>/",
        SaleReadViewSet.as_view({
            "get": "retrieve",
            "delete": "destroy",
        }),
        name="sale-detail",
    ),
]
```

Análisis del Mapeo Muestral

- Diccionario de Acciones: `as_view({'get': 'retrieve', 'delete': 'destroy'})` mapea explícitamente las operaciones HTTP permitidas.
- Ausencia de Router: No se usa `SimpleRouter` para evitar la autogeneración no deseada de métodos PUT/PATCH o POST.
- Reutilización de GET `/api/sales/`: `SaleCollectionAPIView` implementa handler GET delegando al queryset de `SaleReadViewSet`.
- Convertidor Path: `<int:pk>` coincide estrictamente con `lookup_value_converter = 'int'`.

Lógica Interna del Método `sale.void()`

Flujo de Anulación Idempotente (`apps/sale/models.py`)

```
def void(self):
    with transaction.atomic():
        sale = Sale.objects.select_for_update().get(pk=self.pk)
        if sale.status == RecordStatus.INACTIVE:
            return sale # IDEMPOTENCIA: Si ya está anulada, no altera stock de nuevo

        lines = sale.lines.select_related("product").select_for_update().order_by("product_id")
        for line in lines:
            if line.status != RecordStatus.ACTIVE:
                continue
            product = Product.objects.select_for_update().get(pk=line.product_id)
            product.stock += line.quantity
            product.save(update_fields=["stock", "updated_at"])
            line.status = RecordStatus.INACTIVE
            line.save(update_fields=["status", "updated_at"])

        sale.status = RecordStatus.INACTIVE
        sale.save(update_fields=["status", "updated_at"])
        return sale
```

Verificación con SaleTransactionTests

Métodos de Prueba en Acceso OPEN

```
def test_detail_keeps_historical_price(self):
    response = self.client.post("/api/sales/", self._payload([{"productId": self.water.id, "quantity": 1}], format="json")
    Product.objects.filter(pk=self.water.pk).update(price=Decimal("99.00")) # Cambio precio vivo
    detail = self.client.get(f"/api/sales/{response.data['id']}/")
    self.assertEqual(detail.data["lines"][0]["unit_price"], "10.00") # Mantiene 10.00!

def test_void_restores_stock_once(self):
    created = self.client.post("/api/sales/", self._payload([{"productId": self.water.id, "quantity": 2}], format="json")
    sale_id = created.data["id"]
    first = self.client.delete(f"/api/sales/{sale_id}/") # HTTP 204
    second = self.client.delete(f"/api/sales/{sale_id}/") # HTTP 204
    self.assertEqual(first.status_code, 204)
    self.assertEqual(second.status_code, 204)
    self.water.refresh_from_db()
    self.assertEqual(self.water.stock, 5) # Restaura a 5 exactamente una vez!
```

Criterios de Aceptación, Evidencias y Compuerta GATE

Criterio (AC)	Verificación Técnica	Evidencia Generada	Resultado
AC-12-01	GET detalle incluye líneas y unit_price congelado	test_detail_keeps_historical_price PASS	PASS
AC-12-02	DELETE responde HTTP 204 No Content	test_void_restores_stock_once PASS	PASS
AC-12-03	Fila sales permanece conservada en inactive	EVI-12-01 / Sale.objects.get status INACTIVE	PASS
AC-12-04	Idempotencia: anular N veces repone stock 1 sola vez	test_void_restores_stock_once stock == 5	PASS

Preguntas para la Defensa Oral del ISS-12

Preguntas de Comprensión Arquitectónica

Q: ¿Por qué no se usó ModelAndView ni ReadonlyModelViewSet en la consulta de ventas?

A: Porque ModelAndView permitiría la modificación arbitraria con PUT/PATCH (violando la inmutabilidad contable) y ReadonlyModelViewSet bloquearía la acción de anulación por DELETE.

Q: ¿Qué asegura que el stock no se incremente dos veces si un cliente presiona anular repetidamente?

A: La verificación temprana en `sale.void()`: ``if sale.status == RecordStatus.INACTIVE: return sale``. Al estar el bloque protegido con `select_for_update()`, la segunda llamada detecta el estado inactivo y retorna inmediatamente.

Q: ¿Por qué las líneas de detalle (ProductSale) no tienen su propia URL directa en la API?

A: Porque ProductSale es una entidad dependiente de la agregación Sale. No posee existencia comercial aislada y se consulta/anula siempre a través de la raíz del agregado.