

CURSO DE INGENIERÍA DE SOFTWARE / DJANGO REST FRAMEWORK

ISS-10 — Disponibles con ListAPIView

Publicación de Lecturas Especializadas y Filtradas mediante GenericAPIView + ListModelMixin

¿Qué construiremos en el ISS-10?

Objetivo Técnico Central

- **Colección Filtrada Especializada:** Publicar el endpoint GET /api/products/available/ para consultar únicamente productos que puedan ser vendidos.
- **Justificación de GenericAPIView:** Utilizar ListAPIView (GenericAPIView + ListModelMixin) en lugar de ModelViewSet para evitar la exposición indeseada de verbos de mutación (POST, PUT, DELETE).
- **Triple Filtro de Negocio:** Garantizar que solo retornen productos con status='active', stock > 0, y cuyo tipo de producto asociado esté también activo (product_type__status='active').
- **Optimización SQL con select_related:** Prevenir consultas N+1 al traer los datos del ProductType en un solo JOIN SQL.

Diferenciación de Vistas DRF

1. ModelViewSet (ISS-09)

Expone CRUD completo (6 acciones). Excelente para administración de entidades homogéneas.

1. ListAPIView (ISS-10)

Expone únicamente la lectura GET de una lista. Sin mutaciones, altamente optimizada.

1. APIView (ISS-11)

Control total manual para operaciones transaccionales complejas (e.g. registro de ventas).

Punto de Partida y Enrutamiento en URLs

Punto de Partida (ISS-09 Superado)

- Modelos de Catálogo Listos: ProductType y Product funcionando con relaciones FK y constraints.
- ViewSets Registrados: ProductTypeViewSet y ProductViewSet publicando /api/product-types/ y /api/products/ via SimpleRouter.
- Requisito del ISS-10: Incorporar el endpoint /api/products/available/ sin alterar ni contaminar el ModelViewSet del catálogo.

Parche en apps/product/urls.py (Precedencia de Rutas)

```
urlpatterns = [  
    path(  
        "products/available/",  
        AvailableProductListAPIView.as_view(),  
        name="product-available",  
    ),  
]  
urlpatterns += router.urls
```

Construcción Progresiva: AvailableProductListView

Implementación en apps/product/views.py

```
from rest_framework.generics import ListAPIView
from rest_framework.permissions import AllowAny

class AvailableProductListView(ListAPIView):
    permission_classes = [AllowAny]
    serializer_class = ProductSerializer

    def get_queryset(self):
        return Product.objects.select_related("product_type").filter(
            status=RecordStatus.ACTIVE,
            stock__gt=0,
            product_type__status=RecordStatus.ACTIVE,
        )
```

Explicación por Bloques Semánticos:

- `permission_classes = [AllowAny]`: Permiso abierto durante la Fase I (OPEN). Se asegura con JWT/RBAC en ISS-22.
- `serializer_class = ProductSerializer`: Reutiliza el serializer con soporte camelCase (`productTypeId`, `productName`).
- `get_queryset()` con `select_related`: Efectúa un JOIN SQL con `ProductType` y aplica los 3 filtros de disponibilidad obligatorios.

Flujo de Ejecución HTTP y Filtro de Disponibilidad

1. HTTP Request: GET /api/products/available/

2. URL Dispatcher: Coincidencia con 'products/available/' (precedencia sobre router)

3. ListAPIView: Ejecuta get_queryset() e invoca ProductSerializer(many=True)

4. Consulta ORM: SELECT ... FROM products JOIN product_types WHERE status='active' AND stock > 0 AND type_status='active'

5. HTTP Response: 200 OK con arreglo JSON camelCase

Verificación HTTP con APITestCase (apps/product/tests.py)

Test de Integración: test_product_crud_and_available_filter

```
def test_product_crud_and_available_filter(self):
    # 1. Crear producto activo con stock = 4
    created = self.client.post("/api/products/", {
        "productTypeId": self.product_type.id,
        "name": "Agua", "price": "2.50", "stock": 4, "status": "active"
    }, format="json")
    self.assertEqual(created.status_code, 201)

    # 2. Verificar presencia en disponibles (len == 1)
    available = self.client.get("/api/products/available/")
    self.assertEqual(available.status_code, 200)
    self.assertEqual(len(available.data), 1)

    # 3. Cambiar stock a 0 via ORM y verificar desaparición de la lista (len == 0)
    Product.objects.filter(pk=created.data["id"]).update(stock=0)
    empty = self.client.get("/api/products/available/")
    self.assertEqual(empty.data, [])
```

Criterios de Aceptación, Evidencias y GATE

Criterios de Aceptación (AC) y Evidencias

- AC-10-01: Un producto activo con stock > 0 y tipo activo aparece en GET /api/products/available/.
- AC-10-02: Al quedar el stock en 0 (o inactivarse el tipo), desaparece de la lista pero sigue existiendo en la BD.
- EVI-10-01: Prueba de integración pasa de [item] a [] tras agotar stock sin borrar la fila física.

Compuerta de Decisión (GATE)

Criterio	Verificación	Evidencia Documentada	Resultado
AC-10-01	GET /api/products/available/	EVI-10-01 (ProductApiTests PASS)	PASS
AC-10-02	Stock = 0 oculta sin eliminar	EVI-10-01 (ProductApiTests PASS)	PASS

Preguntas de Defensa Oral para el ISS-10

Cuestionario de Defensa Técnica

1. ¿Por qué utilizar ListAPIView en lugar de agregar una acción al ModelViewSet?

R: ModelViewSet expone operaciones de mutación CRUD. ListAPIView de DRF restringe por diseño la vista a lectura de solo lista de manera idiomática y limpia.

2. ¿Por qué es crítico incluir `product_type__status=RecordStatus.ACTIVE` en el filtro?

R: Si un tipo de producto es inhabilitado comercialmente, ninguno de sus productos asociados debe ofrecerse para venta, aunque su stock sea mayor a cero.

3. ¿Cuál es el impacto del uso de `select_related('product_type')` en esta vista?

R: Realiza un INNER JOIN en la consulta SQL principal, reduciendo de N+1 consultas a exactamente 1 consulta a la base de datos al serializar productName.

4. ¿Por qué la ruta literal `'products/available/'` debe definirse antes de `router.urls`?

R: Garantiza la precedencia explícita en el enrutador de Django y previene colisiones o ambigüedades con convertidores de parámetros en los endpoints del router.