

ISS-07 — Serializers y JSON camelCase

Traducción de modelos Django ORM a JSON sin renombrar campos Python

StoreLab Architecture Series • Material Pedagógico Universitario

¿Qué construiremos en el ISS-07?

Objetivos Principales

- Construcción de la capa de API con Django REST Framework (DRF).
- Traducción automática de atributos Python (snake_case) a JSON en el cable (camelCase).
- Implementación de serializers para Client, ProductType, Product, ProductSale y Sale.
- Habilitación de CORS para consumo seguro desde clientes frontend web/móvil.
- Limpieza y normalización de entradas de usuario (lower/strip en emails).

Regla Fundamental de Borde

- El código Python y la base de datos CONSERVAN snake_case (created_at).
- El payload JSON transmitido por HTTP usa camelCase (createdAt).
- La conversión es realizada de forma transparente por renderers y parsers de DRF.
- NO se renombra ningún campo en los modelos Django para simular otros frameworks.
- Mantiene la convención nativa de Python en backend y de JavaScript en frontend.

Punto de partida y estado previo

Estado Alcanzado en ISS-06

- Base de datos relacional modelada con Custom User (security.User), Client, ProductType, Product, Sale y ProductSale.
- Modelos abstractos TimeStampedModel y StoreLabModel con status y validaciones de estado.
- Reglas de negocio transaccionales e inmutabilidad de precio histórico (unit_price) implementados.
- Anulación atómica de ventas con restitución de inventario vía Sale.void().

Necesidad Arquitectónica en ISS-07

- Hasta el ISS-06 el proyecto consistía únicamente en modelos Django ORM y base de datos.
- No existían endpoints HTTP ni capacidad de serialización JSON.
- El ISS-07 instala las librerías fundamentales de API REST y establece el contrato de interfaz pública.

Paquetes CLI e Instalaciones Requeridas

django-rest-framework ==3.17.2

- Aporta la infraestructura base de API.
- Proporciona serializers (ModelSerializer), APIView, ModelViewSet y APIClient de pruebas.
- Base obligatoria para todo el ecosistema REST del proyecto.

django-rest-framework -camel-case==1.4.2

- Renderiza JSON saliente en camelCase.
- Parsea JSON entrante de camelCase a snake_case.
- Transforma created_at a createdAt automáticamente sin modificar Python.

django-cors-headers ==4.9.0

- Inserta CorsMiddleware en el ciclo HTTP.
- Evita bloqueos de origen cruzado en navegadores.
- Configura CORS_ALLOW_ALL_ORIGINS para el laboratorio académico.

Configuración en config/settings/__init__.py

INSTALLED_APPS y MIDDLEWARE

- INSTALLED_APPS += ['rest_framework', 'corsheaders']
- MIDDLEWARE inserta 'corsheaders.middleware.CorsMiddleware' antes de SecurityMiddleware.
- CORS_ALLOW_ALL_ORIGINS lee variable de entorno (default True).

Diccionario REST_FRAMEWORK

- DEFAULT_PERMISSION_CLASSES:
['rest_framework.permissions.AllowAny'] (Fase I abierta).
- DEFAULT_RENDERER_CLASSES:
 - CamelCaseJSONRenderer
 - CamelCaseBrowsableAPIRenderer
- DEFAULT_PARSER_CLASSES:
 - CamelCaseJSONParser
 - CamelCaseFormParser / CamelCaseMultiPartParser

ClientSerializer — Normalización y Limpieza

Estructura en apps/client/serializers.py

- model = Client
- fields = ['id', 'name', 'address', 'phone', 'email', 'status', 'created_at', 'updated_at']
- read_only_fields = ['id', 'created_at', 'updated_at']
- extra_kwargs define min_length=2, max_length=150 en name.
- email, address y phone se marcan como required=False y allow_null=True.

Hooks de Validación Personalizados

- validate_email(self, value):
Normaliza el correo a minúsculas y remueve espacios (strip().lower()).
Si viene vacío o espacios, retorna None.
- validate(self, attrs):
Itera sobre ('address', 'phone'). Si el valor recibido es "", lo convierte en None.
Evita violaciones de unicidad o cadenas vacías inconsistentes en BD.

ProductTypeSerializer y ProductSerializer

ProductTypeSerializer

- `model = ProductType`
- `fields = ['id', 'name', 'description', 'status', 'created_at', 'updated_at']`
- `read_only_fields = ['id', 'created_at', 'updated_at']`
- `validate_description(self, value)`: convierte cadenas vacías en `None`.

ProductSerializer con Desacoplamiento

- `product_type_id = PrimaryKeyRelatedField(source='product_type', queryset=ProductType.objects.all())`: clave escriturable.
- `product_type_name = CharField(source='product_type.name', read_only=True)`: nombre descriptivo de solo lectura.
- Informa el nombre del tipo sin requerir un query HTTP adicional.
- En el cable HTTP se serializan como `productTypeId` y `productTypeName`.

SaleSerializer y ProductSaleSerializer

ProductSaleSerializer (Detalle)

- `product_name = CharField(source='product.name', read_only=True)`
- `fields = ['id', 'product', 'product_name', 'quantity', 'unit_price', 'line_total', 'status', 'created_at', 'updated_at']`
- `read_only_fields = fields` (en ISS-07 es completamente de lectura).

SaleSerializer (Cabecera Anidada)

- `client_name = CharField(source='client.name', read_only=True)`
- `lines = ProductSaleSerializer(many=True, read_only=True)`
- Permite obtener la venta completa con sus detalles anidados en una sola petición GET.
- El alta transaccional de ventas se implementa posteriormente en ISS-11.

Flujo de Traducción en el Borde HTTP

Ciclo Bidireccional de Datos

- 1. Solicitud Entrante (POST/PUT):
Payload JSON en el cable HTTP: {"createdAt": "2026-10-01...", "productTypeId": 1}
CamelCaseJSONParser traduce las claves a snake_case en Python: {'created_at': ..., 'product_type_id': 1}.
 -
- 2. Procesamiento Backend & ORM:
Python opera nativamente con snake_case en modelos y BD.
 -
- 3. Respuesta Saliente (GET/201 Created):
Serializer pasa el objeto Python con atributos snake_case.
CamelCaseJSONRenderer convierte las claves a camelCase para el cliente HTTP: {"createdAt": ..., "productName": "Lácteos"}.

Verificación del Contrato API: `response.content` vs `response.data`

`response.data` (Estructura Interna)

- Retornado por el APIClient de DRF.
- El APIClient de pruebas parsea automáticamente el cuerpo de respuesta usando el parser configurado.
- Por lo tanto, en `response.data` los atributos vuelven a aparecer en snake_case (`created_at`).
- Es útil para aserciones de tipos de datos y valores lógicos.

`response.content` (Cuerpo HTTP Real)

- Contiene el string de bytes crudo enviado a través de la red HTTP.
- Es el que recibe un cliente externo (React, Flutter, cURL).
- Aserción de contrato requerida en pruebas:
`self.assertIn(b'createdAt', response.content)`
`self.assertNotIn(b'created_at', response.content)`

Criterios de Aceptación y Evidencias

Matriz AC y Evidencia

- • AC-07-01: El cuerpo HTTP de un cliente contiene 'createdAt' y NO 'created_at'.
Evidencia EVI-07-01: Aserción sobre response.content en tests pasa exitosamente.
-
- • AC-07-02: 'Maria@Example.com' se guarda en base de datos como 'maria@example.com'.
Verificación: Test de cliente valida normalización en validate_email.
-
- • AC-07-03: El producto informa 'productName' sin requerir un segundo query al cliente.
Verificación: Test de producto valida presencia de productName en payload de respuesta.

Compuerta GATE y Preguntas de Defensa Oral

Tabla GATE (ISS-07)

- AC-07-01 | JSON camelCase | EVI-07-01 | PASS
- AC-07-02 | email normalizado | test cliente | PASS
- AC-07-03 | nombre del tipo | test producto | PASS
-
- Estado del GATE: PASS (Completamente superado)

Preguntas de Defensa Oral

- 1. ¿Por qué se prefiere traducir a camelCase en el renderer en lugar de modificar los nombres de campos en los modelos Django?
- 2. ¿Por qué response.content muestra 'createdAt' pero response.data en APIClient muestra 'created_at'?
- 3. ¿Qué beneficio aporta product_type_name con source='product_type.name' en el frontend?