Sistema Inteligente de Gestión de Inventario

Documentación (v0.2)

23 de noviembre de 2025

Índice

1. Arquitectura General del Sistema

El proyecto **Smart Inventory** está diseñado como un sistema modular de gestión de inventario que combina:

- Un **frontend simple** basado en HTML, TailwindCSS y JavaScript.
- Una **API REST robusta** implementada con Django + Django REST Framework.
- Un **modelo de datos** estable y orientado a trazabilidad mediante lotes.
- Un módulo futuro de **aprendizaje automático** para predicciones de uso.

El flujo de trabajo completo es:

Frontend (QR/Formulario) ⟶ API REST (/api/scan/) ⟶ Base de Datos (SQLite/PostgreSQL) ⟶ ML (futuro)

2. Frontend

2.1. Interfaz de escaneo

La interfaz principal del sistema está en `scan.html`. Permite ejecutar 4 tipos de movimientos:

- Entrada de stock (IN)
- Salida de stock (OUT)
- Auditoría parcial (AUD)
- Auditoría total (AUDTOTAL)

El frontend envía un JSON de este estilo:

```
{
  "payload": "PRD:<uuid>",          // opcional en IN; obligatorio en OUT
  "movement_type": "IN",
  "quantity": 2,
  "location": "<uuid_de_location>",
  "new_product": {
    "name": "Leche entera",
    "unit": "l",
    "category": "L cteos",
    "expiration_date": "2025-11-01"
  }
}
```

La ubicación enviada en el JSON es siempre el **UUID público** de la Location, no la ruta completa.

3. Modelo de Datos Relevante

Los modelos implicados en la lógica del endpoint `/api/scan/` son:

3.1. Product

- `id` (UUID interno)
- `name`, `category`, `unit`
- `location`: ubicación principal del producto
- `min_stock`
- `qr_payload`: identificador permanente `PRD:<uuid>`

3.2. Batch (Lote)

- `product` (FK)
- `quantity` (stock del lote)
- `expiration_date`
- `entry_date`
- `opened_units` (0 o 1)
- `opened_at`
- `open_expires_at`
- `is_depleted`
- `depleted_at`

3.3. Location

Ubicación jerárquica con un método `full_path()` que devuelve rutas tipo:

```
Cocina > Armario 1 > Balda 2
```

3.4. Movement

Historial estructurado de operaciones:

- `movement_type` {IN, OUT, AUD, AUDTOTAL}
- `location`
- `quantity` (+ para entradas, – para salidas)
- `metadata` (JSON: lotes consumidos, lote abierto, etc.)

4. Lógica del Endpoint /api/scan/

El endpoint procesa todos los movimientos del sistema mediante un único punto de entrada. La lógica se divide en cuatro handlers internos:

- **_handle_in**: Entradas de stock
- **_handle_out**: Salidas de stock (incluye apertura)
- **_handle_aud**: Auditoría de ubicación
- **_handle_audtotal**: Auditoría global

Antes de delegar en un handler, la API ejecuta una fase de validación común:

- Validación de `movement_type`
- Validación de `quantity >0` (IN/OUT)
- Validación de `payload` y PRD
- Validación de `location`
- Validación de `mark_open` y `open_days`

5. Movimiento IN (Entrada)

El movimiento `IN` crea o amplía stock mediante lotes nuevos. Existen dos rutas de ejecución según el contenido de `payload`.

5.1. Caso A: IN con PRD válido

1. El `payload = "PRD:<uuid>"` se parsea y se busca el producto.
2. Si no existe, error `product_not_found`.
3. La ubicación se resuelve así:
   - si el request incluye `location`, se usa ésta,
   - si no, se usa la del producto.
4. Se crea un **nuevo lote** (`Batch`) con:
   - `quantity = abs(quantity)`
   - `expiration_date`
   - `notes`
5. Se registra un `Movement IN`.
6. Se devuelve el `payload` QR del producto.

5.2. Caso B: IN sin PRD (alta o reutilización)

1. Se requiere obligatoriamente:
   - `name`
   - `unit`
   - `location` válida
2. Se busca un `Product` que coincida en:
   - nombre normalizado,
   - ubicación,
   - tenant.
3. Si existe, se reutiliza; si no existe, se crea.
4. Se crea un lote nuevo asociado al producto.
5. Se registra un `Movement IN`.
6. Se devuelve o asigna el `PRD:<uuid>` permanente.

6. Movimiento OUT (Salida)

El movimiento `OUT` es el más complejo del sistema porque incluye:

1. Consumo de lotes abiertos.
2. Marcaje de apertura de envase (`mark_open`).
3. Consumo FIFO estándar.

Antes de procesar, la API valida:

- **payload obligatorio**: si no existe PRD, error `missing_prd`.
- **UUID válido**: si no es `PRD:<uuid>` correcto → `invalid_payload`.
- **quantity >0**: si no → `invalid_quantity`.

6.1. 1. Prioridad: consumo de unidad abierta

Si existe un lote con `opened_units = 1`, ésta es la operación primaria:

1. Si `mark_open = true`, error: ya hay un lote abierto.
2. Se descuenta una unidad del lote.
3. Se resetean los campos de apertura.
4. Se marca `is_depleted` si corresponde.
5. Se registra `Movement OUT`.

6.2. 2. Marcaje de apertura (`mark_open = true`)

Cuando el objetivo es abrir un envase:

1. Requiere `quantity = 1`; si no, error `invalid_mark_open`.
2. `open_days` debe ser nulo o un entero positivo.
3. Se elige el lote más próximo a caducar.
4. Si ya está abierto, se rechaza.
5. Se establece:
   - `opened_units = 1`
   - `opened_at = now()`
   - `open_expires_at = now() + open_days`

6.3. 3. Consumo FIFO estándar

Si no se cumple ninguna de las dos ramas anteriores:

1. Se calcula el stock total disponible.
2. Si el stock es insuficiente → `insufficient_stock`.
3. Se abre una transacción y se aplican descuentos FIFO:
   - primero por `expiration_date`,
   - luego por `entry_date`,
   - por último por `id`.
4. Se construye una lista de trazabilidad `consumed_batches`.
5. Si hay carrera (`remaining >0`) → `concurrency_race`.
6. Se registra `Movement OUT`.

6.4. Rango de caducidades tras la salida

Tras descontar stock, la API devuelve:

- `nearest_expiration`: lote más próximo a caducar.
- `farthest_expiration`: última fecha de caducidad con stock.

Esto permite al frontend mostrar un resumen claro:

- stock total restante,
- caducidad mínima,
- caducidad máxima.

7. Auditoría Parcial (AUD)

`AUD` devuelve el estado del inventario en una ubicación concreta.

1. Se valida el `location`.
2. Se listan los `Product` asociados.
3. Para cada producto:
   - lotes activos,
   - total de unidades,
   - caducidad más próxima.

8. Auditoría Total (AUDTOTAL)

Recorre todas las ubicaciones del tenant y devuelve una estructura:

- ubicación
- productos dentro
- totales de unidades
- caducidad mínima por producto

9. Reglas de Consistencia y Errores

- **Stock no negativo**: nunca se descuenta por debajo de 0.
- **Un solo lote abierto por producto**: enforced en OUT.
- **FIFO estricto**: orden por caducidad y fecha de entrada.
- **Transacciones atómicas**: para evitar estados intermedios corruptos.
- **Errores estructurados**:
  - `missing_prd`
  - `invalid_payload`
  - `invalid_quantity`
  - `invalid_mark_open`
  - `invalid_open_days`
  - `insufficient_stock`
  - `concurrency_race`
  - `location_required`

Estas reglas aseguran que todas las operaciones de escaneo se reflejen de manera consistente en la base de datos y que sea posible auditar todo el historial del sistema de forma fiable.