

# Respuesta Técnica (Auditoría TI): Seguridad, JSON, Almacenamiento, Pricing y Flujo de Datos

Proyecto EMMA / EMMO

23 de enero de 2026

## 1. Contexto

Este documento responde, con terminología de ingeniería y auditoría TI, a los comentarios sobre:

- estructura del proyecto, ciberseguridad y gestión de `.env`;
- organización del JSON para OCR (multi-tipo de factura);
- gestión y retención de PDFs por trimestre;
- logging, observabilidad y política de fallos;
- pricing con reglas matemáticas sin IA;
- migrabilidad SQLite → Postgres;
- nomenclatura del modelo de datos y trazabilidad;
- cómo pasa `ocr_info_clothes` a “Importación Artículos Montcau”.

## 2. Estructura del proyecto, ciberseguridad y `.env`

### 2.1. Estructura y responsabilidades

El repositorio separa responsabilidades:

- `backend/`: API de negocio (FastAPI), modelos SQLAlchemy, servicios (OCR, storage, pricing), tests.
- `docs/`: documentación técnica ( $\text{\LaTeX}$ ) y material de requisitos.
- `automate_reading_accountant.js`: prototipo/legado para pruebas iniciales; **no** se ejecuta en producción ni es dependencia del backend.

### 2.2. Gestión de secretos y configuración (`.env`)

Se utiliza `.env` sólo para desarrollo local. En producción la configuración debe provenir del entorno (CI/CD, contenedor, secret manager).

Variables relevantes (prefijo `EMMO_`):

- `EMMO_DATABASE_URL`: string de conexión (SQLite por defecto; Postgres en prod).
- `EMMO_API_KEY`: secreto estático para autenticar clientes.
- `EMMO_REQUIRE_API_KEY_FOR_WRITE`: exige API key para escritura (POST/PUT).
- `EMMO_REQUIRE_API_KEY_FOR_READ`: (opcional) exige API key para lectura (GET) cuando la API se expone.

## 2.3. Ciberseguridad: cómo se protegen las APIs (MVP + ruta a prod)

MVP (ya implementado):

- Autenticación por API key en cabecera X-API-Key.
- Comparación en tiempo constante (mitiga timing attacks en verificación de secretos).
- Security headers (hardening del navegador): X-Content-Type-Options, X-Frame-Options, Referrer-Policy, Permissions-Policy.
- Request correlation: se propaga X-Request-ID y se inyecta en logs para trazabilidad.
- Rate limiting básico (in-memory) configurable por minuto; no multi-instancia.
- Trusted hosts (opcional) para mitigar host-header attacks si se expone directamente.

Producción (recomendación de auditoría):

- Terminar TLS (HTTPS) en reverse proxy/API gateway.
- WAF / rate limit centralizado (Redis/gateway) si se escala horizontalmente.
- Secret manager (no .env en disco) y rotación del API key.
- Segmentación de red (allowlist) y CORS estricto.

## 3. JSON: organización, multi-tipo de factura y consistencia

### 3.1. Diseño recomendado

Para no “confundir modelos” ni forzar claves vacías:

1. Separar **cabecera** y **líneas**.
2. Declarar **tipo de factura** en `invoice.invoice_type`.
3. Guardar campos variables de negocio en `invoice.optional_fields` (diccionario libre).

### 3.2. Contrato de OCR (payload normalizado)

Ejemplo de shape canónico:

```
{  
    "invoice": {  
        "cif_supplier": "B123...",  
        "name_supplier": "Proveedor SL",  
        "num_invoice": "F-2026-001",  
        "date": "2026-01-16",  
        "total_invoice_amount": 123.45,  
        "invoice_type": "textil",  
        "optional_fields": {"campana": "rebajas"},  
        "raw_text": "..."  
    },  
    "lines": [  
        {  
            "reference_code": "ABC123",  
            "description": "CAMISETA",  
            "quantity": 10,  
            "price": 5.5,  
            "total_no_iva": 55.0  
        }  
    ]  
}
```

### 3.3. Referencia: evitar colisiones y claves vacías

Para manejar facturas con referencias incompletas y evitar colisiones:

- `reference_code_raw`: lo que vino del OCR/manual.
- `reference_code`: formato canónico `SUP_____ + código` (prefijo 3 letras del proveedor).
- Si no hay referencia: se habilita un fallback determinista `SUP_<hash64>` o `human-in-the-loop` para completarla.

## 4. Invoices: gestión y retención de PDFs por trimestre

### 4.1. Decisión de auditoría

Sí, se guardan los PDFs/imagenes para trazabilidad, auditoría y presentación trimestral.

### 4.2. Estrategia de almacenamiento

Al subir una factura, el backend persiste el fichero en:

`<storage_root>/invoices/<year>/Q<quarter>/<uuid>.<ext>`

Metadatos en BBDD:

- ruta relativa, nombre, MIME;
- `sha256` del contenido (integridad / no repudio interno);
- tamaño en bytes.

## 5. Tests: logging, seguimiento y política de fallos

### 5.1. Logs y observabilidad

Los logs se emiten a `stdout` (patrón cloud-native) y se pueden serializar como JSON.

Seguimiento recomendado:

- Agregador (ELK/Datadog/Grafana Loki) capturando `stdout`.
- Correlación por `X-Request-ID`.
- Alarmas por ratio de `4xx/5xx` y por `last_error_code`.

### 5.2. Fallo blando vs fallo duro (no parar el negocio)

**Fallo duro (bloquea request / para el flujo):**

- autenticación (401),
- MIME no permitido (415),
- exceder tamaño (413),
- conflictos de integridad (409) donde el dato sería inconsistente.

**Fallo blando (no para sistema; se registra para corregir):**

- fallo del OCR externo: se crea factura con `status=needs_review` y se rellenan `last_error_*`.
- líneas sin `reference_code`: se ingieren y se corrigen vía `human-in-the-loop`.

## 6. Pricing: corrección de precios demasiado bajos (sin IA)

### 6.1. Criterios y fuente de verdad

Dado que las descripciones son ruidosas y no hay SKU, el anclaje debe ser `ref_code`.

Criterios (reglas matemáticas):

1. Si existe maestro de artículos con `coste_unitario`: validar que `price ≥ coste_unitario × min_ratio`.
2. Si no existe coste maestro: usar un histórico por `reference_code` (mediana) y validar contra `median × min_ratio`.

### 6.2. «Necesitas una BBDD de antiguos precios?»

Sí: para el segundo caso. Se mantiene una tabla de observaciones (precio, referencia, timestamp). Esto permite reglas por negocio (p.ej. ventanas trimestrales, percentiles, etc.) sin IA.

## 7. Arquitectura de BBDD: SQLite hoy, Postgres mañana

Se usa SQLite para iterar rápido; el acceso está abstraído por SQLAlchemy y se parametriza por `EMMO_DATABASE_URL`. En producción se recomienda Postgres por concurrencia, locking y operación.

## 8. 5. Modelo de datos (nomenclatura, PK/FK, trazabilidad)

### 8.1. 5.1 Cabecera (proveedor/factura)

Decisiones:

- `name_supplier`: string.
- `tel_number_supplier`: string (internacionalización, prefijos, formatos).
- `email_supplier`: string.
- `num_invoice`: string.
- `total_invoice_amount`: evita ambigüedad con SQL (más claro que `total_supplier`).
- Trazabilidad de mensajes: `source_channel`, `source_thread_id`, `source_message_id`.

### 8.2. 5.2 PK y FK (glosario)

- **PK (Primary Key)**: identificador único de una fila (ej. `data_ocr_invoice.id`).
- **FK (Foreign Key)**: referencia a otra tabla (ej. `ocr_info_clothes.invoice_id` apunta a la PK de factura).

### 8.3. `ref_code`: múltiples estrategias (OCR + hash64 + human-in-the-loop)

Estrategia gradual:

1. OCR intenta extraer `reference_code`.
2. Si viene, se normaliza a `SUP___ + ref_code` (prefijo 3 letras + “\_” + código).
3. Si no viene, se genera `SUP_<hash64>` (determinista) o se solicita al operador (human-in-the-loop).

## 9. OCR: Tesseract

Tesseract es una buena opción para un primer escalón local (imagen) y como fallback. En PDFs complejos suele requerir pipeline extra (render PDF → imagen → OCR) o un proveedor especializado.

## 10. Paso de `ocr_info_clothes` a Importación Artículos Montcau

### 10.1. Modelo de integración

Se implementa una exportación en JSON (preparado para transformación a CSV o API externa):

- Se seleccionan líneas de la factura.
- Se filtran las que tienen `reference_code` (las demás quedan para human-in-the-loop).
- Se proyectan campos a “fila Montcau”: `reference_code`, `descripcion`, `cantidad`, `coste_unitario`.

### 10.2. Endpoint de exportación (backend)

```
GET /invoices/{invoice_id}/export/importacion-montcau
```

### 10.3. Integración con PrestaShop (siguiente paso)

Dado que esta semana hay reuniones con el equipo de integración, el enfoque recomendado es:

- acordar el **contrato** (campos obligatorios, encoding, unidad de medida, reglas de merge),
- decidir si la integración es por:
  - (a) import CSV, (b) API interna Montcau, (c) API PrestaShop (webservice) + mapeo a productos,
- mantener el backend como **fuente de trazabilidad** (auditable) y la integración como capa de entrega.

## 11. Comandos de compilación (terminal)

Recomendado (latexmk):

```
cd docs  
latexmk -pdf respuesta_auditoria_ti.tex
```

Alternativa (sin latexmk):

```
cd docs  
pdflatex respuesta_auditoria_ti.tex  
pdflatex respuesta_auditoria_ti.tex
```