Sistema EMMA: OCR de Facturas & Automatización de Artículos

Documentación técnica

16 de enero de 2026

1. Resumen

Este documento describe el sistema EMMA para automatizar la lectura de facturas (foto/PDF), extraer datos mediante OCR y poblar una base de datos con:

- Cabecera de factura (proveedor, número, total, etc.).
- Líneas de artículos detectadas en la factura.
- Maestro de artículos para importación/gestión (enriquecible con catálogo).

El sistema se implementa con dos APIs:

1. **Backend principal (FastAPI)**: persiste datos y expone endpoints de negocio.
2. **OCR Provider (FastAPI)**: API separada que recibe el archivo y devuelve un JSON normalizado (`invoice` + `lines`).

2. Estructura del proyecto

El repositorio contiene:

- `backend/`: API principal + OCR Provider + tests.
- `docs/`: documentación (este LaTeX) y material de requisitos (p.ej. PDF).
- `automate_reading_accountant.js`: script legado/prototipo (no modificado por el backend).

Dentro de `backend/` (resumen):

- `app/main.py`: crea la app FastAPI y registra startup.
- `app/api/routes.py`: endpoints REST (procesado, CRUD, ingestas).
- `app/services/ocr.py`: fachada OCR (modo stub o HTTP hacia OCR Provider).
- `app/ocr_provider_main.py`: OCR Provider API (stub de ejemplo y contrato).
- `app/db/*`: modelos SQLAlchemy, sesión/engine e inicialización de tablas.
- `tests/`: pruebas automáticas (pytest) de los flujos principales.

3. Alcance y objetivos

3.1. Objetivos

- Subir una factura (imagen o PDF) y procesarla end-to-end.
- Guardar la cabecera en `data_ocr_invoice`.
- Guardar líneas en `ocr_info_clothes`.
- Crear/actualizar artículos en `importacion_articulos_montcau` a partir de las líneas.

3.2. No incluido (por ahora)

- Integración con proveedor OCR concreto (Azure Document Intelligence, Tesseract, etc.) *(se deja preparado).*
- Reglas de pricing avanzadas (p.ej. corrección por PVP demasiado bajo).
- UI/web dashboard.

4. Arquitectura

4.1. Componentes

1. **Cliente** (móvil/web/script): toma una foto o adjunta PDF y llama al backend.
2. **Backend principal** (FastAPI):
   - Expone endpoints REST.
   - Orquesta el OCR llamando a `EMMA_OCR_API_URL` (opcional).
   - Persiste la información en SQLite (o DB futura).
3. **OCR Provider** (FastAPI):
   - Recibe el archivo y devuelve JSON normalizado.
   - En producción, encapsula el proveedor OCR real.
4. **Base de datos** (SQLite por defecto): almacena cabeceras, líneas y artículos.

4.2. Flujo principal

1. El cliente envía una imagen/PDF a `POST /process/invoice`.
2. El backend llama al OCR Provider (si está configurado) o usa un stub.
3. El backend guarda cabecera y líneas.
4. El backend hace upsert de artículos por `reference_code`.

4.3. Flujo de reprocesado (actualización)

Cuando se dispone de una nueva foto/PDF para una factura ya creada:

1. El cliente envía un nuevo archivo a `POST /invoices/{id}/process`.
2. El backend ejecuta OCR (HTTP o stub) y actualiza cabecera/líneas asociadas.
3. Se recalcula el upsert de artículos según las líneas resultantes.

5. Modelo de datos

5.1. Tabla: data_ocr_invoice

Campo	Tipo	Descripción
id	int	PK
cif_supplier	string	Identificador fiscal del proveedor
name_supplier	string?	Nombre del proveedor

Campo	Tipo	Descripción
tel_number_supplier	string?	Teléfono
email_supplier	string?	Email
num_invoice	string?	Número de factura
total_supplier	float?	Total de la factura (si se detecta)
raw_text	text?	Texto OCR completo (opcional)
source_channel	string?	Canal de origen (p.ej. `whatsapp`, `telegram`)
source_thread_id	string?	Identificador de hilo/chat para correlación
source_message_id	string?	Identificador del mensaje origen (trazabilidad)
status	string	Estado de la factura (por defecto `draft`)
created_at	datetime	Fecha de inserción

5.2. Tabla: ocr_info_clothes

Campo	Tipo	Descripción
id	int	PK
invoice_id	int	FK a `data_ocr_invoice.id`
cif_supplier	string	Copia del CIF del proveedor
name_supplier	string?	Nombre del proveedor
num_invoice	string?	Número de factura
date	date?	Fecha (si el OCR la devuelve)
reference_code	string?	Código de referencia (puede ser `null` si el OCR no lo trae)
description	string?	Descripción de la línea
quantity	int?	Cantidad
price	float?	Precio unitario (interpretación depende del proveedor)
total_no_iva	float?	Total sin IVA (si se detecta)

Nota: existe una restricción de unicidad a nivel de base de datos para evitar duplicados por `(invoice_id, reference_code)` cuando la referencia está informada.

5.3. Tabla: importacion_articulos_montcau

Esta tabla actúa como maestro de artículos. Para la primera versión el backend hace upsert mínimo con:

- `reference_code` (clave de negocio)
- `descripcion` (desde la línea OCR)
- `cantidad` (desde la línea OCR)
- `coste_unitario` (desde la línea OCR)

El resto de campos (EAN, familia, marca, fotos, etc.) se pueden enriquecer en fases posteriores.

6. APIs

6.1. Backend principal (puerto 8000)

- `GET /health`: healthcheck.

- `POST /process/invoice`: procesa factura nueva (archivo) y crea registros.
- `POST /ingest/invoice`: ingesta de OCR ya extraído (p.ej. desde WhatsApp/Telegram).
- `POST /ingest/line`: ingesta de OCR de una prenda/línea para una factura existente.
- `POST /invoices/{id}/lines/{line_id}/set-reference`: completar referencia en una línea.
- `POST /invoices`: crea cabecera manual.
- `GET /invoices/{id}`: consulta cabecera.
- `POST /invoices/{id}/lines`: añade línea manual.
- `GET /invoices/{id}/lines`: lista líneas.
- `POST /invoices/{id}/process`: reprocesa factura existente con nueva foto/archivo.
- `PUT /articles`: upsert de artículo por `reference_code`.
- `GET /articles/{reference_code}`: consulta artículo.

6.2. Notas de diseño de API

- Los endpoints que aceptan fichero aplican validación de tamaño y allowlist de tipo MIME.
- La autenticación por `X-API-Key` es opcional, activable por variable de entorno.
- La ingesta por JSON (WhatsApp/Telegram) permite persistir líneas aun sin `reference_code`.

7. Ingestión desde WhatsApp y Telegram

En este escenario, el OCR (de factura y/o prenda) llega como mensajes por WhatsApp o Telegram.

7.1. Patrones recomendados

1. **Webhooks**: un servicio receptor valida la autenticidad del proveedor (firma/token), extrae texto/media y llama al backend.
2. **Normalización**: el texto OCR se transforma a JSON estable (cabecera + líneas) y se envía a `/ingest/invoice` o `/ingest/line`.
3. **Correlación**: usar un identificador de hilo (chat id) y/o un código interno de factura para asociar mensajes al `invoice` correcto.

7.2. Referencias (`reference_code`)

- Si el OCR trae `reference_code`, se guarda directamente en `ocr_info_clothes` y se hace upsert del artículo.
- Si **no** hay `reference_code`, lo **óptimo** es guardar la línea igualmente (con `reference_code = null`) y completar la referencia después.

7.3. Óptimo cuando falta la referencia

Recomendación como auditoría TI:

1. **Persistir sin bloquear**: no rechazar la línea; guardar descripción, cantidad, precio y el texto OCR.
2. **Estado y trazabilidad**: marcar factura como `draft` y registrar origen (canal, hilo, message id).
3. **Resolución posterior**: completar la referencia vía endpoint (manual) o mediante matching automático por:
   - EAN (si existe),
   - similitud de descripción (fuzzy),
   - catálogo del proveedor,
   - histórico de compras.

7.4. Seguridad (muy importante)

- Verificar firma/token de webhooks (WhatsApp/Telegram) y no aceptar peticiones anónimas.
- Evitar guardar media completa si no es necesario; aplicar retención limitada.
- No loggear PII en texto plano; redactar o minimizar.
- Rate limiting y protección ante spam.

7.5. OCR Provider (puerto 8001)

- `GET /health`: healthcheck.
- `POST /ocr/invoice`: recibe archivo en `multipart/form-data` con campo `file`.

7.6. Contrato del OCR Provider

Recomendado:

```
{
  "invoice": {
    "cif_supplier": "B123...",
    "name_supplier": "Proveedor SL",
    "num_invoice": "F-2026-001",
    "date": "2026-01-16",
    "total_supplier": 123.45,
    "raw_text": "texto completo OCR (opcional)"
  },
  "lines": [
    {
      "reference_code": "ABC123",
      "description": "CAMISETA",
      "quantity": 10,
      "price": 5.5,
      "total_no_iva": 55.0
    }
  ]
}
```

El backend es tolerante: si el payload no coincide exactamente, intenta extraer lo posible y guarda el resto en `raw_text`.

8. Configuración

Variables de entorno (en `backend/.env`):

- `EMMA_DATABASE_URL`: por defecto `sqlite:///./emma.db`.
- `EMMA_API_KEY`: si se define, exige cabecera `X-API-Key` en endpoints protegidos.
- `EMMA_CORS_ORIGINS`: lista separada por comas (si está vacía, no se habilita CORS).
- `EMMA_MAX_UPLOAD_BYTES`: límite de tamaño para subidas (protección DoS básica).
- `EMMA_ALLOWED_UPLOAD_MIME_TYPES`: allowlist de MIME (p.ej. `application/pdf,image/jpeg,image/png`).
- `EMMA_OCR_API_URL`: endpoint del OCR Provider, p.ej. `http://localhost:8001/ocr/invoice`.
- `EMMA_OCR_API_KEY`: opcional, se envía como bearer token.
- `EMMA_OCR_API_TIMEOUT_S`: timeout.

Para facilitar el arranque se incluye `backend/.env.example`.

9. Ejecución local

9.1. Backend

```
cd backend
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8000
```

9.2. OCR Provider (stub)

```
cd backend
source .venv/bin/activate
uvicorn app.ocr_provider_main:app --reload --port 8001
```

10. Consideraciones de seguridad

- Validar tamaño de archivo (límites) y tipo MIME.
- Autenticación (API key/JWT) si se expone a Internet (en esta versión: `X-API-Key` opcional).
- Registro (logging) sin almacenar PII innecesaria.
- Control de errores del OCR (timeouts, reintentos, circuit breaker).

11. Despliegue (Docker)

El backend incluye un `Dockerfile` para ejecutar la API principal en contenedor. En esta versión se usa SQLite por defecto (válido para desarrollo y baja concurrencia). Para producción, se recomienda Postgres.

12. Pruebas automáticas

Se han añadido pruebas de integración con `pytest` usando `TestClient` de FastAPI:

- Healthcheck.
- Ingesta de factura por JSON con línea sin referencia.
- Completar referencia y verificar upsert de artículo.
- Validación de seguridad (API key requerida cuando está configurada).
- Rechazo de MIME no permitido.

Ejecución:

```
cd backend
source .venv/bin/activate
pip install -r requirements-dev.txt
pytest -q
```

Nota técnica importante: se corrigió la inicialización de base de datos para tests, asegurando que `init_db()` crea las tablas sobre el engine activo (DB temporal) y no sobre una referencia antigua.

13. Cambios implementados (resumen)

Esta versión del proyecto incluye, además del flujo base del diagrama, los siguientes cambios y ampliaciones:

- **Integración OCR por HTTP (opcional)**: si se define `EMMA_OCR_API_URL`, el backend envía el fichero a un servicio OCR externo.
- **OCR Provider API (servicio separado)**: FastAPI adicional con `POST /ocr/invoice` que devuelve un contrato JSON estable.
- **Reprocesado de facturas**: endpoint `POST /invoices/{id}/process` para actualizar una factura existente con una nueva foto/PDF.
- **Ingesta WhatsApp/Telegram**: endpoints `/ingest/invoice` y `/ingest/line` para recibir OCR ya extraído en JSON.
- **Referencia opcional en líneas**: `reference_code` puede ser `null` y completarse después con `/set-reference`.
- **Trazabilidad de origen**: columnas `source_channel`, `source_thread_id`, `source_message_id` y `status` en la cabecera.
- **Endurecimiento básico**: API key opcional, CORS configurable, validación de tamaño y MIME de uploads.
- **Pruebas automáticas**: suite pytest con DB temporal por test.

14. Plan de trabajo (por fases)

14.1. Fase 0: Base funcional (ya implementada)

- BD con 3 tablas.
- Backend con endpoints y flujo end-to-end.
- OCR Provider stub y conexión por HTTP.

14.2. Fase 1: OCR real

1. Elegir proveedor OCR (Azure DI recomendado para facturas).
2. Implementar en OCR Provider:
   - Extracción de cabecera (CIF, número, fecha, total).
   - Extracción de líneas (reference_code, cantidad, precio, total sin IVA).
3. Definir normalizaciones:
   - Formatos numéricos, separador decimal.
   - Limpieza de `reference_code` (espacios, guiones, etc.).

14.3. Fase 2: Reglas de negocio y calidad de datos

1. Detección de duplicados (factura por proveedor + num_invoice).
2. Validaciones: cantidades negativas, precios anómalos.
3. Reglas de pricing (p.ej. corregir PVP muy bajo) con trazabilidad.

14.4. Fase 3: Integración con catálogo / ERP

1. Sincronizar EAN, familias, marca, fotos.
2. Enriquecer artículos por reference_code.
3. Exportaciones (CSV/Excel) según el formato requerido.

14.5. Fase 4: Operación y despliegue

1. Migrar SQLite a Postgres si hay concurrencia.
2. Contenerizar (Docker) y despliegue.
3. Observabilidad: logs estructurados, métricas, trazas.

15. Preguntas abiertas (para cerrar requisitos)

- **Formato de entrada**: % imagen/PDF, resolución típica.
- **Campos obligatorios**: CIF, num_invoice, fecha, etc.
- **Regla de pricing**: cuál es el umbral para considerar PVP "muy bajo".
- **Identidad del artículo**: reference_code de 3 letras + resto (según el diagrama).