Intelligent Document Information Extractor

Automated Processing using GenAI, OCR, and Vector Database

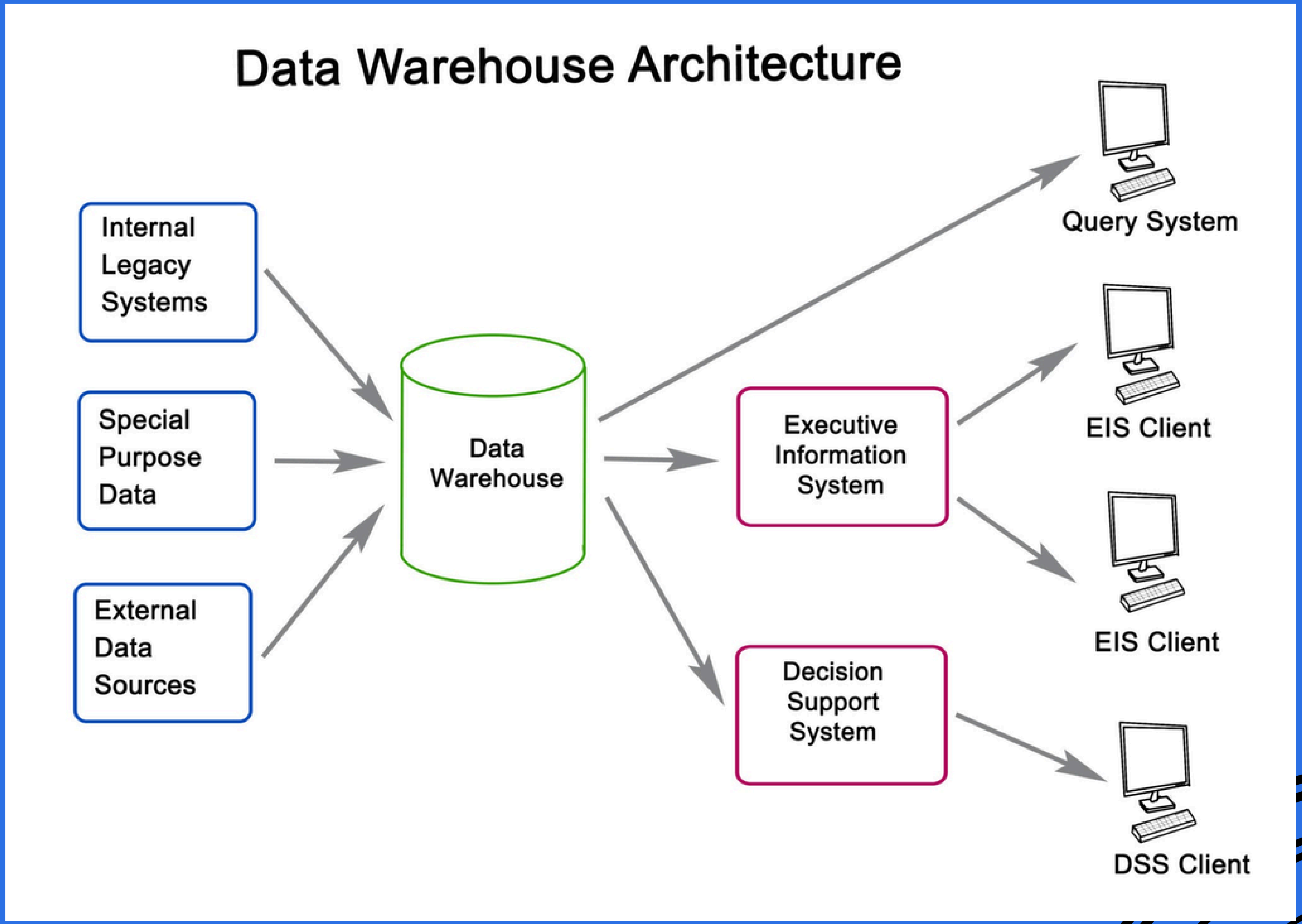
Project Overview & Objective

- The Problem: Manual data entry from invoices, purchase orders, and approvals is time-consuming and error-prone.
- The Solution: An end-to-end pipeline that automates text extraction, document classification, and structured data storage.
- Key Capabilities:
- Handles multiple formats (PDF, PNG, JPG).
- Automatically detects document type.
- Extracts structured JSON data.
- Prevents duplicate entries.

High-Level Architecture

01

- Frontend: Streamlit (User Interface).
- Processing Layer:
  - OCR: Tesseract & PDFPlumber (Text Extraction).
  - LLM: Google Gemini 2.5 Flash (Classification & Extraction).
  - Embeddings: Sentence-Transformers (Vectorization).
- Storage Layer: MongoDB (NoSQL Database for Metadata + Vectors).



Technology Stack

- Language: Python 3.x
- AI Models:
  - Generative: Google Gemini 2.5 Flash (via LangChain).
  - Embedding: paraphrase-multilingual-MiniLM-L12-v2 (Hugging Face).
- Libraries:
  - pytesseract & pdfplumber: Optical Character Recognition.
  - pymongo: Database connectivity.
  - streamlit: Web application framework.
- Environment: managed via .env (API Keys).

STEP 1 - INGESTION & OCR LOGIC

- Hybrid Extraction Strategy:
  - PDFs: Uses pdfplumber to extract native text directly (faster, higher accuracy).
  - Images: Uses pytesseract (OCR) to convert pixel data to text.
- Code Highlight:
  - if file.endswith(".pdf"):
  -     pdfplumber.open(file)...
  - else:
  -     pytesseract.image_to_string(image)...

STEP 2 - INTELLIGENT CLASSIFICATION

- Concept: Before extraction, the system identifies what the document is.
- Mechanism: Sends the first 3000 characters to Gemini with a specific classification prompt.
- Supported Categories:
  - Invoice
  - Purchase Order
  - Approval
- Logic: Dynamic routing to specific database collections based on the AI's response.

STEP 3 - STRUCTURED EXTRACTION (THE "BRAIN")

- Model: gemini-2.5-flash with Temperature 0.1 (Low creativity, high precision).
- The Prompt: "You are a [Doc Type] information extractor... convert text into structured JSON."
- Output: Clean JSON objects containing key-value pairs (e.g., total_amount, vendor_name, date).
- Visual: A split screen showing a messy receipt on the left and clean JSON code on the right.

STEP 4 - VECTOR EMBEDDINGS

- Why Embeddings? To enable future semantic search (RAG) capabilities.
- Model: sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2.
- Process: Converts the raw extracted text into a high-dimensional vector list.
- Usage: Stored alongside the document metadata in MongoDB.

DATABASE DESIGN & DUPLICATE HANDLING

- Database: MongoDB (invoice_reader_db).
- Collections: Dynamically separated (invoices, purchase_orders, approvals).
- Duplicate Detection Logic:
  - Queries DB before insertion.
  - Checks for matching file_name OR identical extracted_text.
  - Prevents data redundancy.

Conclusion

- SUMMARY: SUCCESSFULLY BUILT A SCALABLE, AI-POWERED DOCUMENT PROCESSOR.
- IMPACT: REDUCES MANUAL EFFORT AND DIGITIZES PHYSICAL WORKFLOWS.
- FUTURE ENHANCEMENTS:
  - CHAT WITH DOCUMENTS: USE THE STORED EMBEDDINGS FOR A RAG Q&A BOT.
  - BATCH PROCESSING: ALLOW UPLOADING MULTIPLE FILES AT ONCE.
  - ANALYTICS DASHBOARD: VISUALIZE SPENDING BASED ON EXTRACTED JSON DATA.

https://huggingface.co/spaces/ahmed-ayman/automated_document_processing

https://huggingface.co/spaces/ahmed-ayman/erp-chatbot

Thank you