CASE STUDY DOCUMENTATION

By Noor Alarab

Contents

Global Architecture Document (GAD)

This document describes the architecture of the Financial Document Reader Proof of Concept (PoC), focusing on Named Entity Recognition (NER) for financial documents. The reader extracts key entities from DOCX and chat files and provides them via API and optional UI.

1. Overview

The system is composed of the following components:

1. **User Interface (UI)**
    - Web-based front-end (optional)
    - Allows users to upload documents (.docx, .txt)
    - Displays extracted entities
2. **API Layer**
    - Implemented in FastAPI
    - Provides endpoints for:
        - /upload/docx → DOCX parser
        - /upload/chat → ML-based NER
    - Accepts file uploads via HTTP POST
    - Returns structured JSON response
3. **Processing Pipelines**

Document Type	**Parser Type**	**Technology**	**Output**
DOCX	Rule-based parser	Python (python-docx)	JSON with financial entities
Chat	NER model	Python (spaCy)	JSON with mapped financial entities
PDF	LLM-based pipeline (future)	GPT / RAG	JSON (methodology only)

4. **Storage Layer**
    - Temporary storage of uploaded files (tempfile)

2. Data Flow

Step 1: Upload Document

- User uploads DOCX or chat file via UI or API.
- File is saved temporarily on the server.

Step 2: Process Document

- For DOCX:
    - Rule-based parser reads tables and fields.
    - Extracted entities returned as JSON.
- For chat:
    - ML-based NER model (spaCy) extracts entities.
    - Post-processing maps generic labels (ORG, DATE, CARDINAL) to financial entities.

Step 3: Return Results

- JSON response sent to UI or API client.
- Example JSON structure:

```
{
 "Counterparty": "BANK ABC",
 "Notional": "200 mio",
 "ISIN": "FR001400QV82",
 "Underlying": "AVMAFC FLOAT",
 "Maturity": "2Y",
 "Bid": "estr+45bps",
 "Offer": "offer 2Y EVG estr+45bps",
 "PaymentFrequency": "Quarterly"
}
```

3. Component Interactions

Source / Component	Description / Flow
User / Client	Uploads document via UI or API
UI / API (FastAPI)	Receives file, routes to the correct parser
DOCX Parser (Rule-based)	Extracts entities from structured DOCX tables
Chat NER Parser (ML)	Uses spaCy to extract financial entities from chat
PDF Parser / LLM (Future)	Processes unstructured PDFs with LLM + RAG
JSON Response	All pipelines return structured entities as JSON

Flow:

1. User uploads document → UI/API receives it
2. API routes to the correct parser (DOCX / Chat / PDF)
3. Parser extracts entities → returns JSON to API
4. API sends JSON response back to user/client

4. Technology Stack

Layer	Technology
Backend	Python, FastAPI
NLP	spaCy, python-docx
UI (optional)	HTML/JS, React, or simple forms
Deployment	Local venv / Docker / Cloud VM

5. Post-Processing Rules

- ISIN Detection: Regex pattern [A-Z]{2}[0-9A-Z]{10}
- Notional Amounts: Look for <number> mio or <number> USD/EUR
- Maturity: Detect patterns like 1Y, 2Y, 6M

- Bid / Offer: Regex for estr+<number>bps and lines starting with offer
- Underlying: Extract text near ISIN identifiers

6. Limitations & Future Enhancements

- General-purpose NER may mislabel entities (e.g., “ABC” alone)
- Underlying extraction may require context-aware parsing
- Can integrate a fine-tuned financial NER model for higher accuracy
- Future integration with LLM pipeline for PDF and complex chat documents

7. Summary

The proposed methodology combines:

1. Open-source general-purpose NER (spaCy)
2. Domain-specific post-processing to map to financial fields
3. Optional fine-tuning for better performance on financial chats

Global Methodology Document (GMD) – Chat NER

This document explains the methodology used to extract financial entities from chat documents using a general-purpose Named Entity Recognition (NER) model. It also outlines how the model can be fine-tuned for financial entities.

1. Model Selection

- **NER Model:** spaCy (en_core_web_sm) – open-source, lightweight, general-purpose NER.
- **Reason:**
    - Easy to integrate with Python and FastAPI
    - Pre-trained on generic entities like ORG, DATE, CARDINAL
    - Can be post-processed or fine-tuned for financial use cases

2. Pipeline Overview

1. **Input:** Chat text file (.txt, .chat, .log)
2. **NER Processing:**
    - Load the spaCy model
    - Run nlp(text) to detect generic entities
    - Extract labels: ORG, CARDINAL, DATE, etc.
3. **Post-processing / Mapping:**
    - Map generic entities to financial fields:

spaCy Label	Financial Entity
ORG	Counterparty
CARDINAL	Notional / Maturity
DATE	PaymentFrequency

    - Use regex for ISIN, Underlying, Bid, Offer
4. **Output:** JSON with financial entities:

```
{
  "Counterparty": "BANK ABC",
  "Notional": "200 mio",
  "ISIN": "FR001400QV82",
  "Underlying": "AVMAFC FLOAT",
  "Maturity": "2Y",
  "Bid": "estr+45bps",
  "Offer": "offer 2Y EVG estr+45bps",
  "PaymentFrequency": "Quarterly"
}
```

Global Methodology Document (GMD) – PDF / LLM Pipeline

This section explains the methodology to extract financial entities from unstructured and verbose PDF documents using Large Language Models (LLMs). PDFs often contain tables, free text, and multi-line structures, making rule-based parsing insufficient.

1. Pipeline Overview

Step 1: Document Ingestion

- Input: PDF files
- Extract text using libraries like pdfplumber, PyPDF2, or pdfminer.six

Step 2: Preprocessing

- Normalize whitespace and line breaks
- Optionally extract tables separately (using camelot or tabula-py)
- Split text into manageable chunks for LLM processing

Step 3: LLM-Based Entity Extraction

- Use an LLM (e.g., OpenAI GPT, LLaMA, or Hugging Face models)
- Apply prompting strategies to extract entities, for example:

  Extract the following financial entities from the text:

  Counterparty, Notional, ISIN, Underlying, Maturity, Bid, Offer, Payment Frequency.

  Provide output as JSON.

- Optionally combined with Retrieval-Augmented Generation (RAG):
  - Use embeddings to retrieve relevant sections of the PDF
  - Feed only the relevant chunks to the LLM to improve accuracy

Step 4: Post-processing

- Validate entity formats (ISIN regex, Notional patterns, date formats)
- Handle multi-line or nested values (e.g., Barrier, Coupon)
- Return structured JSON

2. Prompting Strategy

1. **Direct Extraction Prompt**
   - Simple prompt asking the LLM to extract fields
   - Works for short PDFs
2. **Chunked + RAG Prompt**
   - Split document into sections (e.g., paragraphs or table rows)
   - Embed sections and retrieve relevant chunks
   - Feed to LLM for entity extraction
3. **JSON Schema Enforcement**
   - Include instructions to output strictly in JSON
   - Ensures downstream API can consume data without parsing errors

4. Technology Stack

Layer	**Technology**
PDF Parsing	pdfplumber / PyPDF2 / tabula-py
LLM	OpenAI GPT, LLaMA, Hugging Face Transformers
Vector Store (RAG)	FAISS, Weaviate, or Pinecone
API / Backend	Python, FastAPI
Post-processing	Regex, Python

5. Limitations & Considerations

- LLM costs / latency for large PDFs
- Accuracy depends on prompt quality and chunking
- Confidentiality: sensitive PDFs should be handled securely
- Complex tables or images may require additional parsing logic

6. Summary

This methodology allows the system to process unstructured PDF documents that cannot be reliably parsed with rule-based or standard NER approaches. By combining LLMs, prompting, and optional RAG, we can extract financial entities in a structured JSON format for use with the API.