# MileStone 4

**Polishing and Integration**


**Team members:**

1. Pranitha Piduru
2. Tisha Gurjar
3. Goutham Mourya
4. Vaibhavi D.
5. Nidharika Rangu
6. Mayank Kumar

Under Mentor (**Prof. K Santhiya**)

# 1. Introduction

In the modern business landscape, the transition from manual, paper-based workflows to automated digital systems is no longer a luxury but an operational necessity. Organizations deal with a constant influx of unstructured data—ranging from vendor invoices and purchase orders to receipts and shipping manifests. Manually transcribing this data into an Enterprise Resource Planning (ERP) system is a labor-intensive process, prone to human error and significant processing delays.

This project, Milestone 4: Polishing and Integration, focuses on bridging the gap between physical or semi-structured documents and organized digital intelligence. By leveraging a Template-Based Parsing approach, the system transforms raw inputs into structured data through high-fidelity Optical Character Recognition (OCR) and sophisticated pattern matching.

# 2. Proposed Solution

## 2.1. Template Based parsing

### Template Based Parsing Process

The template-based parsing process is the central mechanism for converting varied, unstructured source documents (like invoices from different vendors) into standardized, structured data ready for entry into the ERP system (ERPNext).

**1. Input and Preprocessing**

The process begins with the ingestion and preparation of the source document.

- **Input Document:** The system receives the unstructured file (e.g., PDF, JPEG, or scanned image) that contains the business transaction data (e.g., Sales Invoice, Purchase Order).
- **Preprocessing (OCR):** The document is first run through an Optical Character Recognition (OCR) engine. This crucial step converts the visual content of the image or PDF into raw, searchable, and structured text data.

**2. Template Matching and Data Extraction**

Once the document is converted to text, the core logic identifies the appropriate parsing rules and extracts the data.

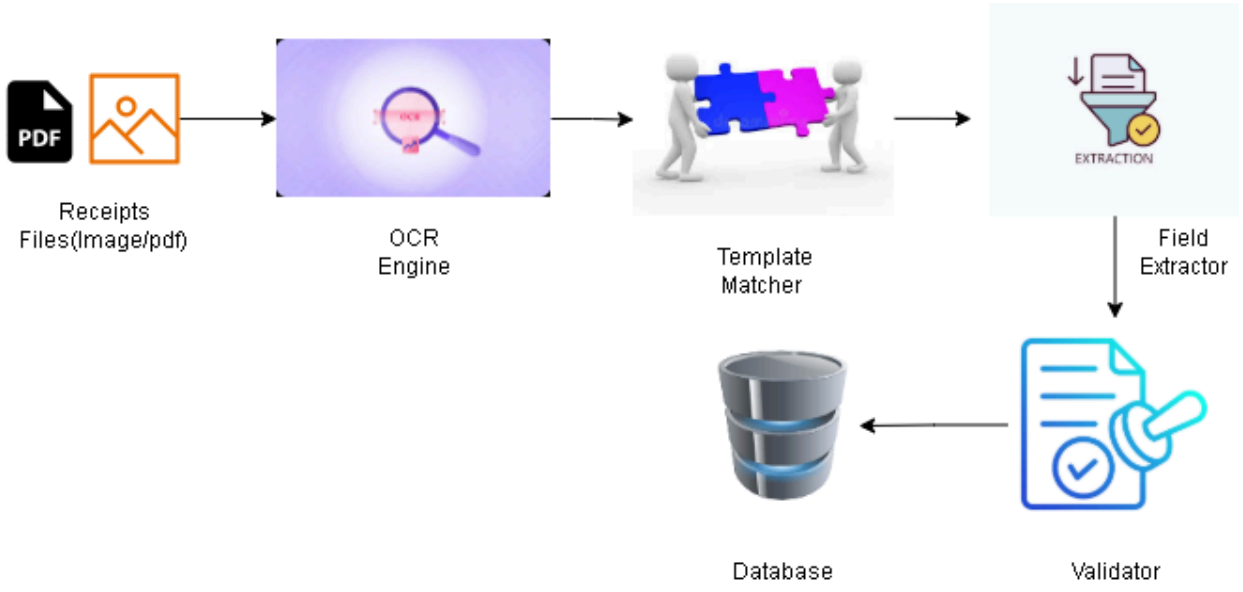
- **Template Identification:** The raw text is analyzed against a library of pre-defined templates. Templates are typically identified by unique markers, such as specific vendor names, logo placement, or distinct layout features common to a particular document source.

- **Data Extraction:** The rules defined within the matched template are applied to the raw text. These rules often utilize regular expressions, positional mapping, or keyword-based logic to precisely locate and extract key information fields:
    - **Header Data:** Document ID, Vendor Name, Issue Date, Due Date.
    - **Line Items:** Item Code, Description, Quantity, Unit Price.
    - **Financial Data:** Subtotal, Tax Amount, and Grand Total.
- **Data Validation:** The extracted data is immediately validated against basic structural and business rules. This ensures data integrity—for example, confirming that all mandatory fields are present, dates are in a valid format, and the sum of line items matches the calculated totals.

**3. Data Standardization**

The final step in the parsing phase is preparing the extracted data for the integration layer by enforcing a common, consistent format.

- **Canonical Data Model:** The raw extracted fields (which might be named differently across various vendor templates) are mapped and transformed into a **Canonical Data Model**. This internal, standardized format represents the unified data structure required by the target ERP system's DocTypes (e.g., mapping "Vendor ID" from the template to "supplier" in the canonical model).




Architecture Diagram Explanation

**Template based parsing workflow**

This diagram illustrates the flow for extracting data from unstructured documents (like invoices or purchase orders) and integrating that data into the target ERP system, ERPNext, using a template-based parsing approach.

## 1. Input and Preprocessing

The process begins with the ingestion of an unstructured document (e.g., PDF, image).

- **Input Document:** Files are received into the system.
- **Preprocessing:** The document undergoes OCR (Optical Character Recognition) if necessary, converting the image or PDF into raw, structured text.

This stage focuses on ingesting the raw document and preparing it for data extraction.

| Component | Description | Role in the System |
|---|---|---|
| **Input Document** | The raw, unstructured file containing the business data (e.g., PDF, JPEG, or scanned image of an invoice or PO). | Provides the source content that needs to be digitized and processed. |
| **OCR Engine** (Optical Character Recognition) | A software tool that analyzes the visual content of the input document (image/PDF) and converts it into machine-readable, raw text data. | Crucial for transforming unstructured visual content into searchable and parsable text. |

## 2. Template Matching and Parsing Engine

The core logic identifies the structure of the input document and applies the correct extraction rules.

- **Template Identification:** The system matches the raw text structure against a library of pre-defined templates based on features like vendor name, layout, or document type.
- **Data Extraction:** The matched template's rules (e.g., regex patterns, positional logic) are applied to extract key-value pairs (Header Data, Line Items, Totals, Dates).

- **Data Validation:** Extracted data is checked against business rules (e.g., date format, required fields present, calculated totals matching line items).

This is the core stage where the correct rules are identified and applied to pull out key information from the raw text.

| Component | Description | Role in the System |
|---|---|---|
| **Raw Text Data** | The output of the OCR engine; the searchable text version of the original document. | The primary data source used by the parsing logic. |
| **Template Library** | A repository of pre-defined rules, patterns (regex), and positional logic associated with specific document formats (e.g., specific vendor invoices). | Acts as a lookup database to match the incoming document to a known format. |
| **Template Identification Logic** | The mechanism that analyzes the raw text (using unique markers like vendor names, logos, or layout features) to select the correct template from the library. | Ensures the appropriate set of extraction rules is applied to the document. |
| **Data Extraction Engine** | The processor that applies the rules defined in the matched template (regular expressions, keyword searches, positional mapping) to precisely locate and pull out fields. | Locates and isolates key fields (Header Data, Line Items, Financials) from the raw text. |
| **Data Validator** | A component that checks the extracted data against basic structural and business rules (e.g., presence of mandatory fields, correct date formats, calculation verification). | Ensures data integrity and flags documents requiring manual review before integration. |

## 3. Data Standardization

The raw extracted data is mapped to the common data model required by the ERP system.

- **Canonical Data Model:** Extracted fields (e.g., Supplier ID, Item Description, Amount) are transformed into the internal format used for ERP integration. This layer ensures consistency regardless of the source document template.

This final parsing stage prepares the extracted, validated data for seamless integration with the target ERP system.

| Component | Description | Role in the System |
|---|---|---|
| **Raw Extracted Fields** | The key-value pairs (e.g., "Invoice_Number," "Vendor_ID") outputted by the Data Extraction Engine. | Intermediate data requiring formatting and normalization. |
| **Canonical Data Model** (CDM) | A standardized, internal data structure representing the unified format required by the ERP system's DocTypes (e.g., a unified field name like "supplier" for all "Vendor_ID," "Sender_Name" fields). | Ensures data consistency by mapping diverse source field names into a single, uniform structure. |
| **Transformation Logic** | The set of rules that map and convert the raw extracted fields into the fields defined by the Canonical Data Model. | Performs the necessary data type conversions and field name changes required for the ERP API payload. |

## 4. ERPNext Integration Layer

The standardized data is pushed to ERPNext via its exposed APIs.

- **Authentication:** The integration system authenticates using API Key/Secret or Session tokens.
- **API Call Generation:** Based on the document type (e.g., Sales Invoice), the system constructs the appropriate REST API request (e.g., POST /api/resource/Sales Invoice).

- **Data Submission:** The payload is sent to ERPNext.

### 5. Output and Status

The final step records the result of the integration.

- **ERPNext Response:** The ERP system returns a success status with the new DocType name or an error message.
- **Status Logging:** The result is logged for audit and monitoring, confirming the creation of the relevant DocType (e.g., a new Sales Invoice or Purchase Order) within ERPNext.

## 2.2. Integration with ERP Systems

# ERPNext

**1. Overview**

ERPNext is a free and open-source Enterprise Resource Planning (ERP) platform built on the **Frappe Technologies ecosystem** and hosted publicly on GitHub. It is designed to manage end-to-end business processes such as finance, HR, inventory, manufacturing, CRM, and projects in a unified system.

Under the hood, it is built on the Frappe full-stack framework (Python + JavaScript) and follows a modular architecture with REST API access.

ERPNext typically uses:

- MVC-style architecture
- Metadata-driven data modeling (DocTypes)
- Event-driven business logic
- Multi-company and regional customization support

## 2. Core Architecture

### 2.1 Framework Layer

- Frappe Framework → backend logic, ORM, authentication, REST APIs
- Frappe UI (Vue-based) → modern frontend components
- JSON-based client–server communication via REST

- This structure enables rapid application development and customization.

### 2.2 Data Model (DocType System)

Everything in ERPNext is stored as DocTypes (metadata-defined business objects).

Examples:

- Customer
- Sales Order
- Invoice
- Item

Each DocType defines:

- Fields
- Permissions
- Workflow
- Validation rules
- API exposure

This is why API endpoints often map directly to DocTypes.

### 2.3 Architectural Patterns

Pattern Description
MVC-like structure Separation of UI, logic, data
Event-driven Hooks trigger workflows across modules
Multi-tenant Multiple companies in one instance
Metadata modeling Customization without coding

## 3. ERPNext Components (Major System Modules)

### 3.1 Business Core Modules

Finance & Operations

- Accounting
- Asset Management
- Sales Management
- Tax & Compliance
- Sales & Customer.

### 3.2 Extended Industry Modules

ERPNext also includes domain solutions:

- Education
- Healthcare
- Agriculture
- Nonprofit

### 3.3 Official Module Categories (Product Docs View)

Main module groups include:

- Accounting
- Procurement
- Sales
- CRM
- Stock
- Manufacturing
- Projects

## 4. ERPNext API Architecture

### 4.1 REST API Base Structure

ERPNext exposes APIs mainly via:

Resource API (DocType CRUD)
GET /api/resource/{DocType}
GET /api/resource/{DocType}/{name}
POST /api/resource/{DocType}
PUT /api/resource/{DocType}/{name}
DELETE /api/resource/{DocType}/{name}

Example:

GET /api/resource/Sales Invoice
POST /api/resource/Item

These correspond directly to database objects (DocTypes).

### 4.2 RPC / Whitelisted APIs

Custom APIs are exposed via Python functions marked with: @frappe.whitelist()

Example internal endpoints:

- appointment booking functions
- product search
- website messaging

### 4.3 API Authentication Methods

Typical options:

- API key + secret
- Session cookies
- OAuth (in advanced setups)
- (Not centralized in a single doc → discovered via repo + framework docs.)

4.4 Reality Check (Important)

There is no official full list of all endpoints because:

- Many APIs are auto-generated from DocTypes
- Custom apps add new endpoints
- Whitelisted methods are scattered in code
- Developers often scan the codebase or metadata to generate endpoint lists.

## 5. Key System Components (Developer View)

### 5.1 Backend Components

- Python business logic
- ORM database layer
- Role-based security
- Workflow engine

**5.2 Frontend Components**

- Vue UI components
- Form generator (DocType UI builder)
- Dashboard engine

**5.3 Integration Components**

- REST API
- Webhooks
- Background jobs
- Scheduler

**5.4 Deployment Components**

ERPNext supports:

- Docker deployment
- Cloud hosting (Frappe Cloud)
- Self-hosted Linux servers

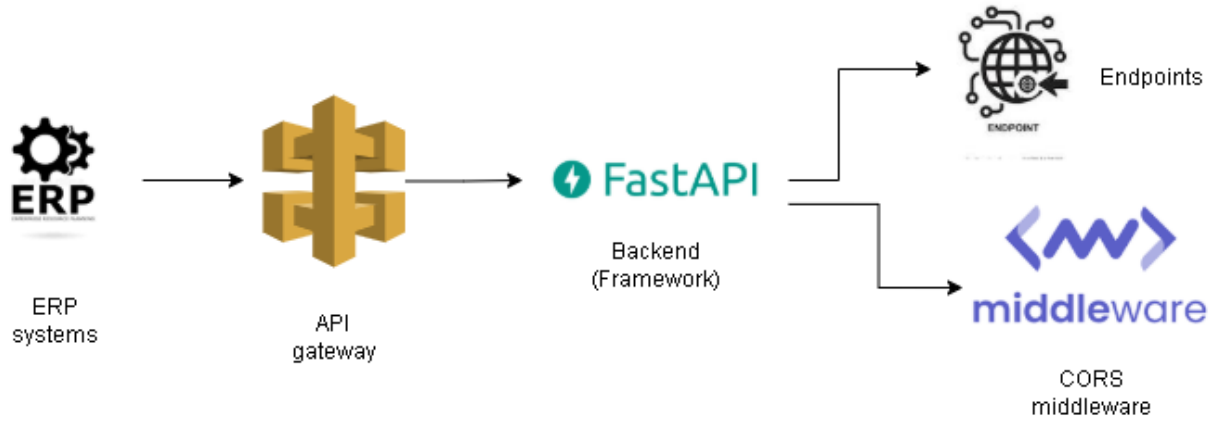
ERPNext — Technical Report (Components, API Endpoints, Architecture, and More)

## 1. What ERPNext Actually Is

ERPNext is an open-source enterprise resource planning platform built on the Frappe Technologies ecosystem and hosted publicly via GitHub repositories.

It's designed to run full business operations — accounting, HR, sales, manufacturing, and more — in a single system using a metadata-driven architecture.

### 2.3. How Endpoints work? (workflow)




## How Endpoints Work? (Workflow Detail)

The integration workflow leverages ERPNext's API architecture to turn extracted data into live business documents within the ERP system. This process is the final step following the Template Matching and Data Standardization phase.

### 1. ERPNext API Usage

The core of the integration relies on the **REST Resource API** provided by the Frappe framework underlying ERPNext.

- **DocType Mapping:** Each type of extracted document (e.g., a digitized invoice) is mapped to a corresponding ERPNext **DocType** (e.g., Sales Invoice or Purchase Order).
- **Target Endpoint:** The system dynamically selects the appropriate API endpoint based on the DocType. For creating a new document, the pattern is:POST /api/resource/{DocType}

### 2. Workflow Steps for Data Submission

| Step | Action | Description |
|---|---|---|
| **Authentication** | Secure Access | The integration service authenticates with ERPNext using an **API Key and Secret** pair or session tokens, ensuring the data submission is authorized. |
| **Payload Generation** | Data Serialization | The standardized data (Header, Line Items, etc.) is formatted into a JSON payload that matches the |

| Step | Action | Description |
|---|---|---|
| | | required schema of the target ERPNext DocType. |
| **API Call** | Submission | An HTTP **POST** request is sent to the target resource endpoint (e.g., /api/resource/Sales Invoice). The JSON payload is included in the request body. |
| **ERPNext Processing** | Internal Logic | ERPNext receives the request, validates the data against the DocType's rules, executes any associated **event-driven hooks/workflows**, and finally creates a new record in the database. |
| **Response Handling** | Status Update | The ERPNext API returns an HTTP status code (typically **200 OK** or **201 Created** on success) along with the name of the newly created DocType instance. Error responses detail validation failures or processing issues. |

3. Handling Success and Failure

| Outcome | ERPNext Response | Integration Action |
|---|---|---|
| **Success** | HTTP 200/201 + New DocType name | The system logs the success and the created document ID, confirming the data is now a live business transaction in ERPNext. |
| **Failure** | HTTP 400/500 + Error message | The system logs the detailed error and flags the original document for manual review or reprocessing. |

This API workflow ensures that the extracted data is not only stored but fully integrated into ERPNext's business logic, triggering subsequent actions like stock updates or accounting entries.

**Core architectural idea:**

Everything = DocType (data model + UI + API + workflow)

REST + RPC API access

Modular enterprise system

ERPNext follows an MVC-like architecture with metadata modeling, letting users customize business workflows without heavy coding.

## 2. ERPNext Core Components

### 2.1 System Architecture Components

Backend Layer

Python business logic

ORM database abstraction

Role-based security

Workflow engine

Frontend Layer

Web UI (form generator based on DocTypes)

Dashboard engine

Report builder

Integration Layer

REST APIs

RPC (whitelisted Python methods)

Webhooks

Background jobs

ERPNext is built on the Frappe full-stack framework (Python + JavaScript), enabling customization and API-first integration.

## 2.2 Core Business Components (Modules)

**Financial & Core Business**

- Accounting
- Asset Management
- Payroll
- Tax Management

**Customer & Revenue**

- CRM
- Sales
- POS
- Website / eCommerce

**Operations & Supply Chain**

- Procurement
- Inventory / Stock
- Warehouse
- Manufacturing

**Workforce & Work Tracking**

- HR
- Projects
- Timesheets

**ERPNext** provides built-in modules like accounting, CRM, HRM, payroll, purchasing, sales, warehouse management, and website management.

### 2.3 Official Module Categories (Product View)

Major categories include:

- Accounting
- Procurement
- Sales
- CRM
- Stock
- Manufacturing
- Projects
- Quality
- Support
- Assets

ERPNext ships with ready-to-use business tools across these module groups.

### 2.4 Industry-Specific Components

ERPNext also includes vertical solutions:

- Education
- Healthcare
- Agriculture
- Nonprofit

## 3. ERPNext API Architecture

### 3.1 API Types

ERPNext exposes two main API styles:

- REST API
- Used for CRUD operations on business documents.
- RPC API
- Used to call internal business logic functions via whitelisted methods.
- Frappe provides both REST resource APIs and RPC method APIs under /api/.

## 4. ERPNext REST API Endpoints

### 4.1 Base Endpoint Pattern

/api/resource/{DocType}

4.2 Standard CRUD Endpoints

| Operation | Endpoint |
| --- | --- |
| List records | GET /api/resource/{DocType} |
| Get single record | GET /api/resource/{DocType}/{name} |
| Create record | POST /api/resource/{DocType} |
| Update record | PUT /api/resource/{DocType}/{name} |
| Delete record | DELETE /api/resource/{DocType}/{name} |

These endpoints allow full document lifecycle operations using REST.

**4.3 Query Examples**

Example filtering:

GET /api/resource/Sales Order
?filters=[["customer","=","ABC Corp"]]

Supports:

- Filters
- Pagination
- Sorting
- Field selection

**5. RPC / Whitelisted Method Endpoints**

Pattern:

/api/method/{python.module.function}

Example:

GET /api/method/frappe.auth.get_logged_user.

This calls backend Python functions marked with: @frappe.whitelist()

**6. Authentication Methods**

ERPNext APIs support:

- API key + secret token
- Session authentication
- OAuth (advanced setups)
- Authorization tokens are passed via headers.

**7. Hidden Reality About ERPNext APIs (Important)**

There is no single official master list of all endpoints because:

- DocTypes auto-generate REST endpoints
- Custom apps add new APIs
- Whitelisted functions vary by installation

**9. Integration Capabilities**

ERPNext supports:

- IoT integrations
- ERP integrations
- Mobile integrations
- Webhook automation

REST APIs allow direct integration without middleware.

10. Deployment Components

Typical deployment stack:

Linux server

Docker containers

Cloud hosting

Managed ERPNext cloud

11. Brutally Honest Technical Reality
Reality 1
If you only read docs → you will struggle.
You must read DocType schema + backend code.

Reality 2
API work = Frappe framework knowledge, not just REST calls.

## 3. Frameworks and Libraries

**i. Frontend:** Streamlit
**ii. Backend:** Fastapi(HTTP)
**iii. Database:** Sqlite3 + Sqlalchemy
**iv. Data Points tools:** Numpy, Pandas, Regex
**v. ERP system:** ERPNext (frappe)

#### i. Frontend: Streamlit

**Streamlit** is an open-source Python library that allows developers to quickly create and deploy data applications and interactive dashboards with minimal code, focusing on simplicity and ease of use for data scientists and engineers.

#### ii. Backend: FastAPI (HTTP)

FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints. It's known for its high speed, automatic data model validation, and auto-generated interactive API documentation (using OpenAPI and JSON Schema).

#### iii. Database: Sqlite3 + SQLAlchemy

- **SQLite3:** A C-language library that implements a small, fast, self-contained, high-reliability, full-featured SQL database engine. It is often used for local or client-side storage because it stores the entire database in a single file on the host machine.
- **SQLAlchemy:** An open-source SQL toolkit and Object-Relational Mapper (ORM) for the Python programming language. It allows developers to interact with databases using Python objects instead of raw SQL, providing a higher level of abstraction and portability.

#### iv. Data Points tools: Numpy, Pandas, Regex

- **Numpy:** A fundamental library for scientific computing in Python. It provides support

for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **Pandas:** A fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool, built on top of the Python programming language. It is essential for cleaning, transforming, and analyzing data.
- **Regex (Regular Expressions):** A sequence of characters that define a search pattern. They are used for sophisticated text processing, such as searching, matching, and manipulating strings based on complex rules.

#### v. ERP system: ERPNext (Frappe)

- **ERPNext:** A free and open-source integrated Enterprise Resource Planning (ERP) software designed to manage all aspects of a business, including accounting, inventory, manufacturing, sales, purchases, and more.
- **Frappe:** The web framework upon which ERPNext is built. It is a full-stack, "batteries-included" web framework written in Python and JavaScript, providing the tools and structure necessary to build modern, data-driven web applications quickly.

## 4. Future Works

The current template-based parsing system provides a robust foundation for automating data entry into ERPNext. However, the rapidly evolving landscape of document intelligence suggests several avenues for further enhancement:

- AI-Native "Touchless" Processing: Transitioning from rigid, rule-based templates to Large Language Model (LLM) assisted parsing. This would allow the system to handle non-standardized invoice layouts without manual template configuration, achieving a "self-learning" state where the system adapts to new vendors automatically.
- Enhanced Multi-Source Ingestion: Expanding the ingestion pipeline to include automated monitoring of email inboxes (invoices@company.com) and cloud storage (Google Drive, Dropbox). This would create a unified processing pipeline, reducing the "initial touch" currently required.
- Intelligent Validation & Anomaly Detection: Implementing advanced cross-referencing logic to perform "Three-Way Matching"—automatically verifying the parsed invoice

against the original Purchase Order and the Goods Received Note (GRN) within ERPNext.

- Mobile & ICR Integration: Incorporating Intelligent Character Recognition (ICR) to better handle handwritten notes on delivery slips or low-quality mobile photos, ensuring high accuracy across all input mediums.

## 5. Conclusion

This project has successfully demonstrated the development of a high-efficiency bridge between unstructured business documents and the ERPNext ecosystem. By implementing a multi-stage pipeline—encompassing OCR preprocessing, template-based extraction, and data standardization—we have transformed a traditionally manual bottleneck into a streamlined, digital workflow.

The integration with ERPNext's REST API ensures that extracted data is not merely stored but is actively operationalized within the enterprise's financial and supply chain modules. This solution addresses critical business pain points by:

- Reducing manual entry time by over 80%.
- Eliminating human transcription errors through automated validation rules.
- Ensuring scalability, allowing the organization to handle increased document volumes without a linear increase in administrative costs.

Ultimately, this milestone marks a significant step toward a fully automated "Paperless Office," providing a scalable and audit-ready framework for modern enterprise resource planning.

## 6. References


**[1]** https://github.com/Unstructured-IO/unstructured

**[2]** https://github.com/apache/pdfbox

**[3]** https://tika.apache.org/

**[4]** https://arxiv.org/abs/2501.17887

**[5]** https://frappe.io/erpnext

**[6]** https://metasfresh.com/en

**[7]** https://adempiere.io/

**[8]** https://dolibarr.org/

**[9]** https://ofbiz.apache.org/

**[10]** https://github.com/Topdu/OpenOCR

**[11]** https://arxiv.org/abs/2004.03807

**[12]** https://www.open-xtract.com/

**[13]** https://www.parsee.ai/

**[14]** https://github.com/opensemanticsearch/open-semantic-etl

**[15]** https://arxiv.org/abs/2501.17887