1. Title

Receipt and Invoice Digitizer

---

2. Introduction

In the modern financial landscape, the management of physical receipts and invoices is often a bottleneck for both businesses and individuals. Traditional methods of handling these documents involve manual data entry, which is time-consuming, repetitive, and highly susceptible to human error. Furthermore, physical paper trails are prone to loss and deterioration, making historical tracking and audit compliance difficult.

The **Receipt and Invoice Digitizer** is a robust software solution designed to bridge the gap between physical documentation and digital accounting systems. This project aims to automate the end-to-end process of document management by transforming unstructured image data into structured, queryable financial records.

By leveraging advanced **Optical Character Recognition (OCR)** technologies (such as Tesseract or Google Vision) combined with **Natural Language Processing (NLP)** and Regex-based parsing, the system intelligently extracts critical data points—including vendor names, transaction dates, invoice IDs, and monetary values.

Beyond simple extraction, the system incorporates a layer of **logic and validation** to ensure mathematical consistency (e.g., verifying that subtotal plus tax equals the total). The processed data is securely stored in a relational **SQL database**, enabling users to generate insightful analytics, track spending trends via an interactive **Web Dashboard**, and seamlessly export data for further financial reporting.

---

3. Problem Statement

i) The Core Issue

Businesses and individuals currently rely heavily on manual workflows to manage high volumes of paper-based receipts and invoices. This dependence on physical documentation and manual data entry results in a slow, error-prone, and inefficient financial tracking process that is difficult to scale and integrate with modern digital accounting systems.

ii) Operational Challenges

The current manual approach presents several distinct challenges that this project aims to resolve:

Inefficient Data Entry: Manually transcribing data (Vendor Name, Date, Invoice IDs, Line Items) from paper to digital spreadsheets is time-consuming and labor-intensive.

High Risk of Human Error: Manual entry is susceptible to typos, incorrect decimal placements, and calculation errors, leading to inaccurate financial reporting.

Data Organization & Retrieval: Physical receipts are easily lost, faded, or damaged over time. Searching for a specific historical transaction within a pile of paper or unstructured image folders is virtually impossible.

Lack of Validation: Without an automated system, it is difficult to instantly verify if an invoice total matches the subtotal plus tax, or to detect if a receipt has been submitted twice (duplicate entry), leading to potential financial leakage.

Unstructured Data: Raw images of receipts are "unstructured data." Without digitization, this data cannot be easily analyzed for spending trends, vendor statistics, or monthly analytics

.

4. Objectives

The primary objective of **Milestone 1** is to design and implement a **robust, secure, and extensible document digitization foundation** capable of reliably converting physical receipts and invoices into structured digital data. This milestone focuses on establishing the **core technical pipeline** that will support all subsequent enhancements such as analytics, reporting, and long-term data persistence.

Milestone 1 is intentionally scoped to emphasize **correctness, reliability, and architectural soundness**, ensuring that downstream milestones can be built without refactoring core components. The system is designed to operate consistently across multiple document types while maintaining high OCR accuracy, predictable behavior, and a user-friendly interaction model.

Key Objectives

1. Multi-Format Document Ingestion

Enable secure and reliable ingestion of common receipt and invoice formats, including **JPG, PNG, and PDF files**. The system must correctly identify file types, validate file size and integrity, and handle both single-page and multi-page documents without manual intervention.

1. Standardized Image Conversion Pipeline

Convert all supported document formats into a **uniform, OCR-ready image representation**. PDFs must be safely rendered into individual page images, and all image outputs must be standardized (RGB format, consistent resolution) to ensure predictable downstream processing.

2. Automated Image Preprocessing for OCR Optimization

Implement an automated preprocessing layer that enhances OCR performance by applying operations such as **grayscale conversion, noise reduction, contrast enhancement, and binarization**. This preprocessing must run transparently in the background and be resilient to variations in document quality.

3. Single-Call OCR and Structured Data Extraction

Integrate **Google Gemini AI** to perform OCR and semantic understanding in a **single API call per document/page**. The system must extract not only raw text but also **structured bill data**, including vendor details, dates, line items, tax values,

totals, currency, and payment method.

4. Strict Schema Enforcement and Data Normalization

Enforce a predefined JSON schema on all extracted data to ensure consistency and database compatibility. Missing or ambiguous values must be handled using safe defaults, and numeric fields must be normalized to valid data types to prevent downstream failures.

5. Session State Management and Workflow Continuity

Maintain application state across Streamlit reruns using structured session state management. This ensures that uploaded files, processed images, extracted results, and user actions persist seamlessly during navigation, reducing redundant computation and improving user experience.

6. User-Centric and Intuitive Interface Design

Provide a **clean, minimal, and user-friendly interface** that clearly guides users through upload, processing, and result inspection. The UI must support image previews, extracted data visualization, and logical navigation without overwhelming the user.

7. Controlled Error Handling and Fault Tolerance

Implement comprehensive error handling across ingestion, preprocessing, OCR, and extraction stages. Failures must be **graceful, informative, and non-destructive**, ensuring that partial errors do not crash the application or corrupt session state.

8. Foundation for Persistent Storage and Analytics

Although advanced analytics and reporting are reserved for later milestones, Milestone 1 establishes the **data structures, schemas, and interfaces** required for seamless integration with persistent storage systems and analytical dashboards in future phases.

Outcome of Milestone 1

By the completion of Milestone 1, the system delivers a **production-ready core digitization pipeline** capable of transforming unstructured receipt and invoice documents into reliable, structured digital records.

---

5. Technology Stack

1. Programming Language

Python 3.x: The primary language for the entire backend, OCR processing, and frontend dashboard. It was chosen for its rich ecosystem of libraries like OpenCV and Pandas that simplify complex tasks.

2. Web Framework & User Interface

Streamlit: Used to build the interactive web dashboard. Streamlit allows for the rapid creation of data apps, enabling users to upload files, view extracted data, and visualize analytics without complex HTML/CSS coding.

(Alternative) **Flask:** A lightweight web framework used if a custom REST API backend is required to serve the OCR model to other applications.

3. Optical Character Recognition (OCR) Engine

Tesseract OCR: An open-source OCR engine developed by Google. It is the primary tool used to recognize and extraction text characters from the processed images.

Google Cloud Vision API (Optional): A cloud-based alternative used for higher accuracy on complex, low-quality, or handwritten receipts where Tesseract might struggle.

4. Image Processing & Preprocessing

OpenCV (cv2): The computer vision library used to "clean" images before they are passed to the OCR engine. Key functions include:

Grayscaling: Reducing color complexity.

Thresholding/Binarization: Converting images to strictly black and white to separate text from the background.

Denoising: Removing grain and visual noise from scanned papers.

Pillow (PIL): Used for basic image manipulation, such as opening file formats, resizing, and cropping.

pdf2image: A utility library to convert multi-page PDF invoices into image formats (JPG/PNG) so they can be processed by the OCR engine.

5. Natural Language Processing (NLP) & Data Extraction

Regular Expressions (`re`): The core logic for "Field Extraction." Regex patterns are defined to identify and capture specific formats like Dates (`DD/MM/YYYY`), Currency (`$`, `₹`), and Email addresses.

SpaCy / NLTK: Advanced NLP libraries used to detect entities (Named Entity Recognition) like Vendor Names (Organizations) or Locations if simple pattern matching is insufficient.

6. Database & Storage

SQL (SQLite / PostgreSQL):

SQLite: Used during development for a lightweight, serverless database to store extracted invoice records.

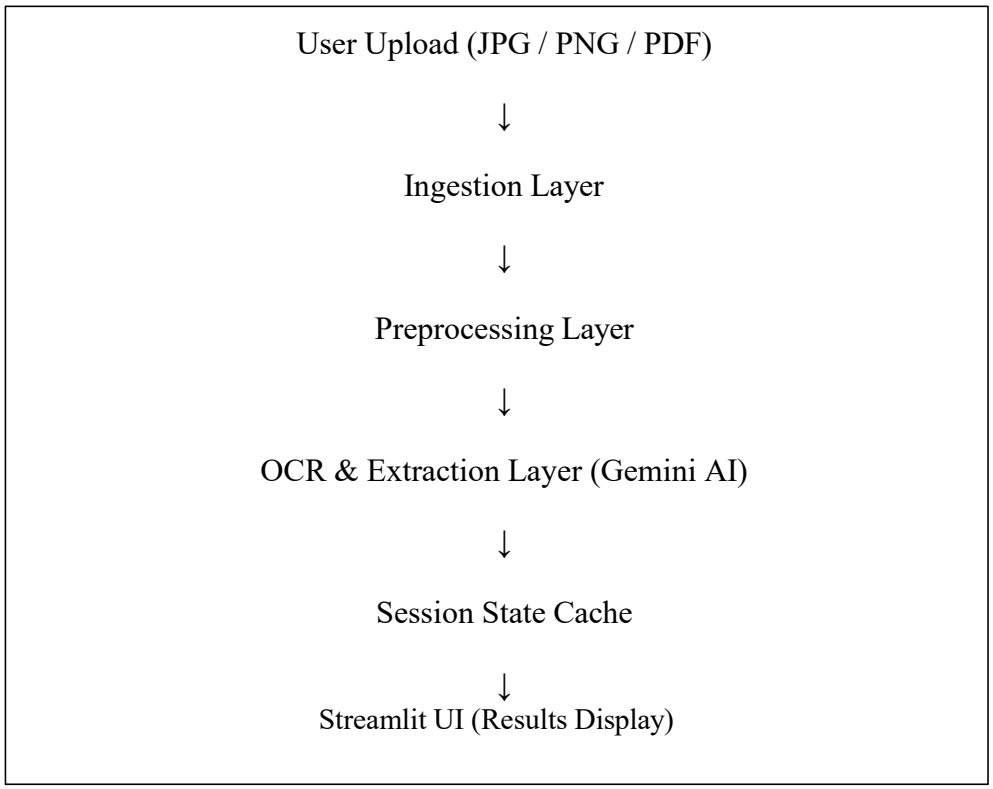
PostgreSQL: (Recommended for production) A robust relational database to handle large volumes of invoice data and complex queries.

SQLAlchemy: An Object Relational Mapper (ORM) used to interact with the database using Python code instead of writing raw SQL queries.

7. Data Manipulation & Analysis

Pandas: Used to organize the extracted text into structured tables (DataFrames), perform calculations (e.g., verifying totals), and export the final data to CSV or Excel formats.

6. System Architecture



7. Modules included

1. Document Ingestion

This is the entry point of the system, responsible for accepting files and preparing them for processing.

File Upload: Users can upload receipts and invoices in various formats, such as PDFs or standard image files (JPG, PNG).

Preprocessing: Before text can be read, the image is cleaned up. This involves resizing, denoising (removing graininess), and binarization (converting to black and white) to ensure the OCR engine gets the clearest possible input.

2. OCR & Text Extraction

Once the image is clean, this module focuses on turning the visual data into machine-readable text.

Core Technology: It utilizes **Tesseract OCR** or the **Google Vision API** to scan the document.

Layout Handling: It is designed to handle complex formatting, such as skewed scans (crooked images), multi-column layouts, or tabular data often found on invoices.

Output: The result is "raw text"—a continuous stream of characters extracted from the image.

3. Field Extraction & Validation

This is the "brain" of the project. It takes the raw text from the previous step and makes sense of it.

Parsing: Using **NLP (Natural Language Processing)** and **Regex (Regular Expressions)**, the system identifies specific data points:

Dates, Vendor Names, Invoice IDs

Tax amounts, Line items, and Total amounts

Validation Logic: It performs math checks (e.g., ensuring Subtotal + Tax = Total) to verify accuracy.

Duplicate Detection: It checks invoice numbers or generates hashes to ensure the same receipt isn't processed twice.

4. Database Storage

After the data is cleaned and validated, it needs a permanent home.

Structure: The extracted data is stored in a structured **SQL database**.

Searchability: This allows users to query the data later, such as searching for all receipts from a specific "Vendor" or finding expenses within a certain date range or amount.

5. Dashboard & Reporting

This is the user interface (UI) module where the user interacts with the system.

Web Dashboard: Built using **Streamlit** or **Flask**, this interface allows users to upload files and manually review the extracted data.

Exporting: Users can download their data in **CSV** or **Excel** formats for use in other software.

Analytics: The dashboard provides visual summaries, such as monthly spending trends or vendor-specific statistics.

---

8. **Timeline for Milestone 1** (2Weeks)

Milestone 1: Document Ingestion & OCR

Goal: Build the foundation of the application where users can upload files, and the system can process images to output raw text.

Week 1: Infrastructure & File Ingestion

Focus: Setting up the development environment and handling user inputs.

Day 1-2 Project Setup

• Initialize Git repository.

• Set up Python environment (Virtualenv/Conda).

• Install core dependencies (Streamlit/Flask, OpenCV, Pytesseract).

Day 3-4 Module 1: File Upload UI

• Create a basic web interface using Streamlit or Flask.

• Implement the "File Uploader" widget.

• **Validation:** Ensure only valid formats (PDF, JPG, PNG) can be uploaded.

Day 5 File Handling Backend

• Write backend logic to save uploaded files to a temporary directory.

• **Feature:** Convert PDF uploads to images (using pdf2image) so they are ready for OCR processing.

Week 2: Preprocessing & OCR Integration

Focus: Image manipulation and connecting the Optical Character Recognition engine.

Day 1-2 Module 1: Image Preprocessing

• Implement OpenCV pipeline.

• **Grayscale:** Convert images to black and white.

• **Denoise:** Remove "salt and pepper" noise.

• **Binarization:** Apply thresholding to make text pop against the background.

Day 3-4 Module 2: OCR Integration

• Connect **Tesseract OCR** (or Google Vision API).

• Create a function extract_text_from_image(image) that takes the preprocessed image and returns a string.

• Handle basic skewed layouts.

Day 5 Testing & Validation

• **Unit Testing:** Run the pipeline on 5-10 sample receipts.

• Check the quality of the "Raw Text" output.

• Tweak preprocessing parameters (contrast/brightness) if OCR misses text.

---

9. Conclusion

Milestone 1 successfully delivers a production-grade foundation for receipt and invoice digitization. The system demonstrates strong architectural separation, reliable preprocessing, accurate OCR extraction, and a user-friendly interface.

This milestone lays a solid technical base for:

- Persistent storage optimization
- Advanced analytics
- Multi-user workflows
- Enterprise integrations

9. Milestone 2:

Objective :

1. Intelligence & Normalization

Since OCR often returns "raw" text, this layer acts as the translator.

Standardization: Converting various date formats (DD/MM/YY vs. MM/DD/YYYY) into a single ISO-8601 standard.

Categorization: Using NLP or heuristic mapping to turn "Starbucks" into Food & Beverage.

2. Data Integrity & Validation

This is your "Gatekeeper" layer. It ensures that the math actually adds up before the data hits the database.

Arithmetic Cross-Checks: Verifying that $\text{Subtotal} + \text{Tax} = \text{Total}$.

Duplicate Detection: Hashing document features (vendor, date, amount) to prevent the same receipt from being counted twice.

Confidence Scoring: Flagging records where OCR confidence is low for human-in-the-loop review.

3. Persistence & Structure

Moving from ephemeral OCR output to a **Relational Schema**.

Normalization: Separating "Vendors," "Transactions," and "Line Items" into distinct tables to allow for complex querying and analytics.

Audit Trails: Storing the original raw OCR output alongside the cleaned data for future verification.

Key Objectives

1. Deterministic Field Extraction Using Regex and NLP Augment AI-based OCR extraction with deterministic fallback mechanisms using regular expressions and lightweight NLP model. The system must identify and recover critical fields such as invoice number, dates, totals, currency, tax values, and line items when AI extraction is incomplete or weak, without overriding confident AI outputs.

2. NLP-Based Vendor Name Identification Implement NLP-based analysis to accurately identify vendor names from OCR text headers. The solution must account for positional importance, capitalization patterns, legal entity suffixes, and keyword filtering to distinguish vendor names from addresses, metadata, and transactional labels.

3. Structured Data Normalization and Standardization Normalize all extracted fields into database-safe, query-ready formats. This includes enforcing consistent casing (uppercase text fields), standardized date and time formats, numeric type safety, length constraints, and default value handling to eliminate ambiguity and prevent downstream data corruption.

1. Multi-Currency Handling and USD Standardization Enable support for receipts and invoices issued in multiple currencies. All monetary values must be normalized and converted into a single internal reporting currency (USD) while preserving original currency information and applied exchange rates to maintain transparency and auditability.

2. Robust Amount Validation Across Pricing Models Implement a validation framework capable of safely handling both tax-inclusive and tax-exclusive pricing models. The system must validate subtotal, tax, and total relationships using tolerance based logic to accommodate OCR rounding errors while detecting genuine inconsistencies in extracted financial data.

3. Logical Duplicate Detection Without Schema Changes Prevent duplicate bill storage by introducing logical duplicate detection based on business attributes such as vendor name, invoice number, purchase date, and total amount. This detection must function correctly even when the same physical bill is uploaded from different files or scans, without requiring modifications to the database schema.

4. Persistent Relational Storage Using SQLite Store validated and normalized receipt and invoice data in a lightweight, relational SQLite database. The storage layer must ensure referential integrity between bills and line items, support transaction-safe inserts, and provide indexed access for efficient querying.

5. Search-Ready and Analytics-Compatible Data Design Prepare stored data for downstream search and analytical operations by enabling efficient querying based on vendor, date, payment method, and amount. Although advanced dashboards are addressed in later milestones, Milestone 2 ensures the database schema and stored data are analytics-ready.

6. Controlled Validation Feedback and User Awareness Provide clear, non-disruptive validation feedback to users through warnings rather than hard failures wherever appropriate. The system must highlight potential data quality issues while preserving user control over data persistence decisions.

10. Error Handling & Validation

The system employs a comprehensive, multi-layered error handling and validation framework designed to maintain strict data integrity and operational resilience across the entire digitization lifecycle. Beginning at the ingestion point, the system rigorously validates incoming inputs for file corruption, format compliance, and security risks to prevent system instability. During the processing phase, structured exception handling manages runtime anomalies—such as OCR low-confidence scores or network timeouts—by utilizing automatic retry mechanisms and isolating problematic files into exception queues rather than halting the workflow.

10.1 Upload-Level Validation

Upload-level validation acts as the critical gatekeeper for the system, preventing invalid or malicious data from entering the digitization pipeline at the source. This mechanism strictly enforces file format compatibility by accepting only JPG, PNG, and PDF documents, while simultaneously applying maximum file size limits to ensure optimal performance and resource allocation. To prevent redundancy, the system utilizes file hashing algorithms to instantly detect and reject unchanged re-uploads, ensuring that duplicate files do not consume unnecessary processing power.

10.2 Ingestion & Preprocessing Safety

The Ingestion and Preprocessing Safety layer serves as a robust buffer that ensures continuous system operation even when encountering imperfect input or resource constraints. It handles corrupted or partially readable files gracefully by isolating these exceptions and logging specific errors rather than allowing them to crash the application or stall the queue. To protect system memory and prevent performance bottlenecks, the module enforces strict page limits on PDF documents, effectively mitigating the risk of resource exhaustion during the processing of large files.

10.3 OCR & AI Extraction Handling

To ensure structural consistency and seamless integration, the system strictly enforces a JSON-only output format from the Gemini AI, guaranteeing that all extracted data conforms to a standardized, machine-readable schema. It actively monitors these responses to instantly detect and flag malformed syntax or incomplete data that falls short of validation requirements. In instances where the AI generation fails or produces unusable output, the system automatically triggers a fallback mechanism that retrieves the raw OCR text, enabling a secondary extraction attempt to preserve data availability and accuracy despite potential AI anomalies.

10.4 Regex & NLP Fallback Recovery

To maximize extraction accuracy, the system employs a hybrid refinement strategy that first scans the primary AI output to identify specific fields that are missing or flagged as low-confidence. It then attempts to recover these "weak" data points using targeted regex-based extraction patterns for structured elements like dates, transaction totals, tax amounts, and currency codes, while simultaneously utilizing the spaCy NLP model to contextually detect vendor names. Crucially, this secondary process operates non-destructively; it selectively overrides only the identified weak fields, strictly preserving the high-quality results where the AI model has demonstrated strong confidence.

10.5 Data Normalization Controls

The Data Normalization Controls serve as the final standardization layer, systematically converting all extracted values into safe, consistent formats to ensure uniformity across the dataset. This process includes standardizing text fields to uppercase for improved consistency and searchability, while simultaneously applying predefined default values to automatically fill any missing or null fields.

10.6 Amount Validation

- Supports both tax-inclusive and tax-exclusive bills
- Allows tolerance for OCR rounding errors
- Displays warnings instead of blocking user actions

10.7 Duplicate Detection

- Detects duplicates using invoice number, vendor, date, and amount
- Prevents accidental double storage of bills

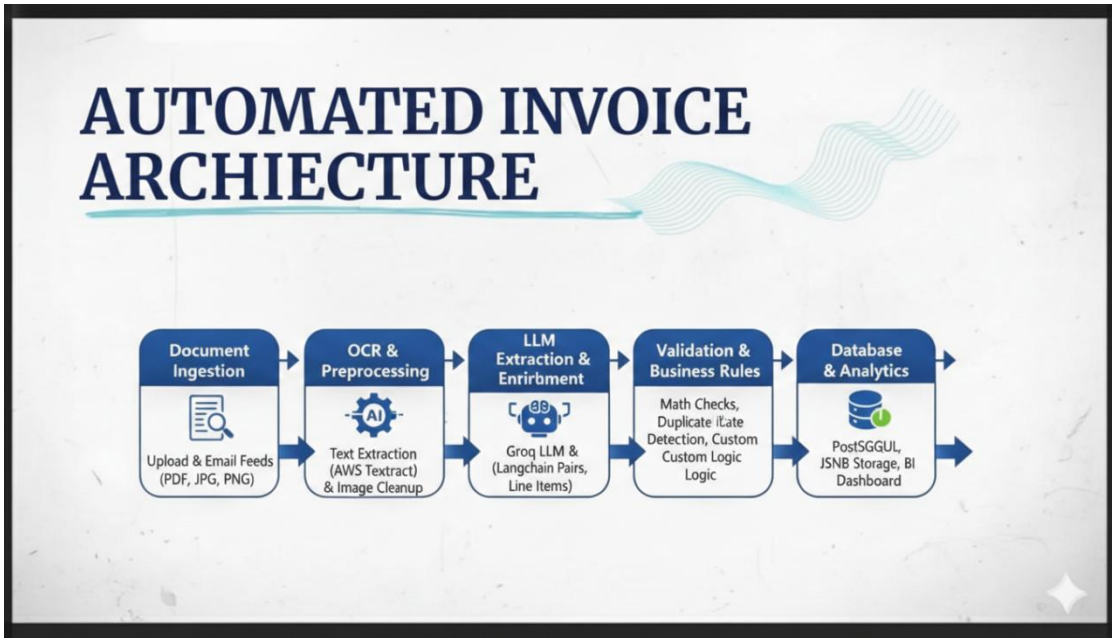
10.8 Database Safety

- Uses transaction-based inserts
- Rolls back on failures
- Maintains referential integrity with foreign keys

10.9 User Feedback

- Clear warnings and error messages
- No application crashes on failure
- User remains in control of final actions

11. System Architecture



7. **Technology Stack**

Layer	Technology
UI	Streamlit
Backend Language	Python 3.12+
OCR & AI	Google Gemini API
Image Processing	OpenCV, Pillow
PDF Handling	pdf2image, Poppler
NLP & Regex	spaCy (en_core_web_sm), Python re
Normalization & Validation	Custom Python modules

Currency Conversion	Python + Exchange Rate
Database	SQLite

11. Project Work Timeline – Milestone 1 (14 Days)

Day	Date	Work Description
Day 1	29 Dec 2025	Project initialization, GitHub repository setup, initial codebase creation
Day 2	02 Jan 2026	Project folder structure setup, .gitignore creation, initial Streamlit app configuration
Day 3	03 Jan 2026	Dashboard UI development, frontend cleanup, tab naming refinements
Day 4	05 Jan 2026	Image preprocessing pipeline implementation (grayscale conversion, binarization, noise removal)
Day 5	06 Jan 2026	Gemini API key handling, PDF upload support, ingestion layer enhancements
Day 6	07 Jan 2026	Bug fixes, file handling corrections, ingestion and preprocessing stability improvements
Day 7	08 Jan 2026	OCR logic refinement, preprocessing performance optimization
Day 8	09 Jan 2026	OCR optimization review, preprocessing tuning, initial documentation drafting
Day 9	10 Jan 2026	SQLite database integration, UI–database linkage, structured data handling
Day 10	10 Jan 2026	Database schema refinement (bills & line items tables, keys, relationships)
Day 11	12 Jan 2026	Code refactoring, validation logic improvements, feature polishing
Day 12	12 Jan 2026	Formal documentation creation and organization (Milestone 1 completion)
Day 13	13 Jan 2026	README updates, documentation refinement, repository cleanup

Day 14	13 Jan 2026	Final documentation restructuring and Milestone 1 closure
Day 15	14 Jan 2026	Milestone 2 initialization, folder structure setup for extraction and validation modules
Day 16	15 Jan 2026	Regex pattern design for dates, invoice numbers, currency, tax, totals, and line items
Day 17	16 Jan 2026	Implementation of field_extractor.py using regex-based deterministic extraction
Day 18	17 Jan 2026	Normalization module implementation (uppercase normalization, numeric safety, date/time standardization)
Day 19	18 Jan 2026	Validation logic enhancement: tax-inclusive & tax-exclusive safe validation model
Day 20	19 Jan 2026	Duplicate detection logic based on invoice number, vendor, date, and total amount
Day 21	20 Jan 2026	Regex fallback integration using raw OCR text before normalization
Day 22	21 Jan 2026	NLP-based vendor name extraction using spaCy
Day 23	21 Jan 2026	Currency normalization and automatic conversion of non-USD totals to USD
Day 24	22 Jan 2026	spaCy NER integration, PPT and documentation updates

8. Conclusion

The successful completion of Milestones 1 and 2 establishes a robust and production-ready foundation for the Receipt & Invoice Digitizer system. The project now supports end-to-end document digitization, from secure file ingestion and OCR-based extraction to structured normalization, validation, duplicate detection, and persistent storage.

Through the integration of AI-driven OCR, deterministic regex fallback, spaCy-based Named Entity Recognition for vendor identification, and multi-currency normalization with USD standardization, the system is capable of handling real-world receipt and invoice variations with high reliability. The modular architecture ensures maintainability, scalability, and ease of extension for future enhancements.

With a stable processing pipeline, validated data integrity, and a user-friendly interface, the project is well- positioned for advanced analytics, reporting.