## 1. Title

# Receipt and Invoice Digitizer

---

## 2. Introduction

In modern business environments, receipts and invoices are generated at high volumes across retail, logistics, services, and enterprise operations. These documents often exist in unstructured physical or semi-structured digital formats such as scanned images or PDFs. Manual handling of such documents leads to inefficiencies, human errors, loss of records, delayed reimbursements, and limited financial visibility.

The **Receipt & Invoice Digitizer** project aims to solve this problem by providing an automated, AI-assisted system that converts unstructured receipt and invoice documents into machine-readable digital data. The application is implemented as a **multi-page Streamlit web application**, integrating advanced image preprocessing techniques with **Google Gemini AI** for Optical Character Recognition (OCR) and structured information extraction.

This document describes **Milestone 1**, which focuses on building a **robust digitization foundation**—from document ingestion to OCR extraction and UI visualization—without yet emphasizing long-term analytics optimization or enterprise integrations.

---

## 3. Problem Statement

Organizations and individuals routinely process large volumes of receipts and invoices. The traditional workflow typically involves:

- Manual data entry into spreadsheets or accounting systems
- Storing physical copies for compliance
- High probability of transcription errors
- Inconsistent formatting across vendors

- Difficulty in tracking expenses over time

These challenges result in:

- Increased operational cost
- Reduced accuracy in financial records
- Delays in expense reconciliation
- Poor audit readiness

There is a strong need for an **automated, intelligent, and scalable solution** that can:

- Accept receipts and invoices in common formats
- Extract text accurately
- Convert unstructured content into structured data
- Present results clearly to users

---

## 4. Milestone 1 Objective

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.
2. **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.
3. **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.

4. **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.
5. **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.
6. **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.
7. **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.
8. **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.
9. **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. This milestone ensures technical stability, modularity, and scalability, forming a solid base for advanced features in subsequent milestones.

## 5. Milestone 2 Objective

The primary objective of **Milestone 2** is to extend the foundational digitization pipeline established in Milestone 1 into a **reliable, intelligent, and validation-driven data processing system** capable of producing high-quality, normalized, and persistent financial records from OCR-extracted content.

While Milestone 1 focuses on accurate document ingestion and OCR-based extraction, Milestone 2 emphasizes **data integrity, consistency, validation, and long-term usability**. This milestone ensures that extracted receipt and invoice data is not only readable, but also **trustworthy, comparable, and suitable for analytics, querying, and financial tracking**.

Milestone 2 is intentionally scoped to address real-world challenges such as inconsistent document formats, OCR inaccuracies, multi-currency transactions, duplicate uploads, and ambiguous tax structures, without requiring changes to the core ingestion or OCR architecture.

**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.

4. **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.

5. **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.

6. **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.

7. **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.

8. **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.

9. **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.

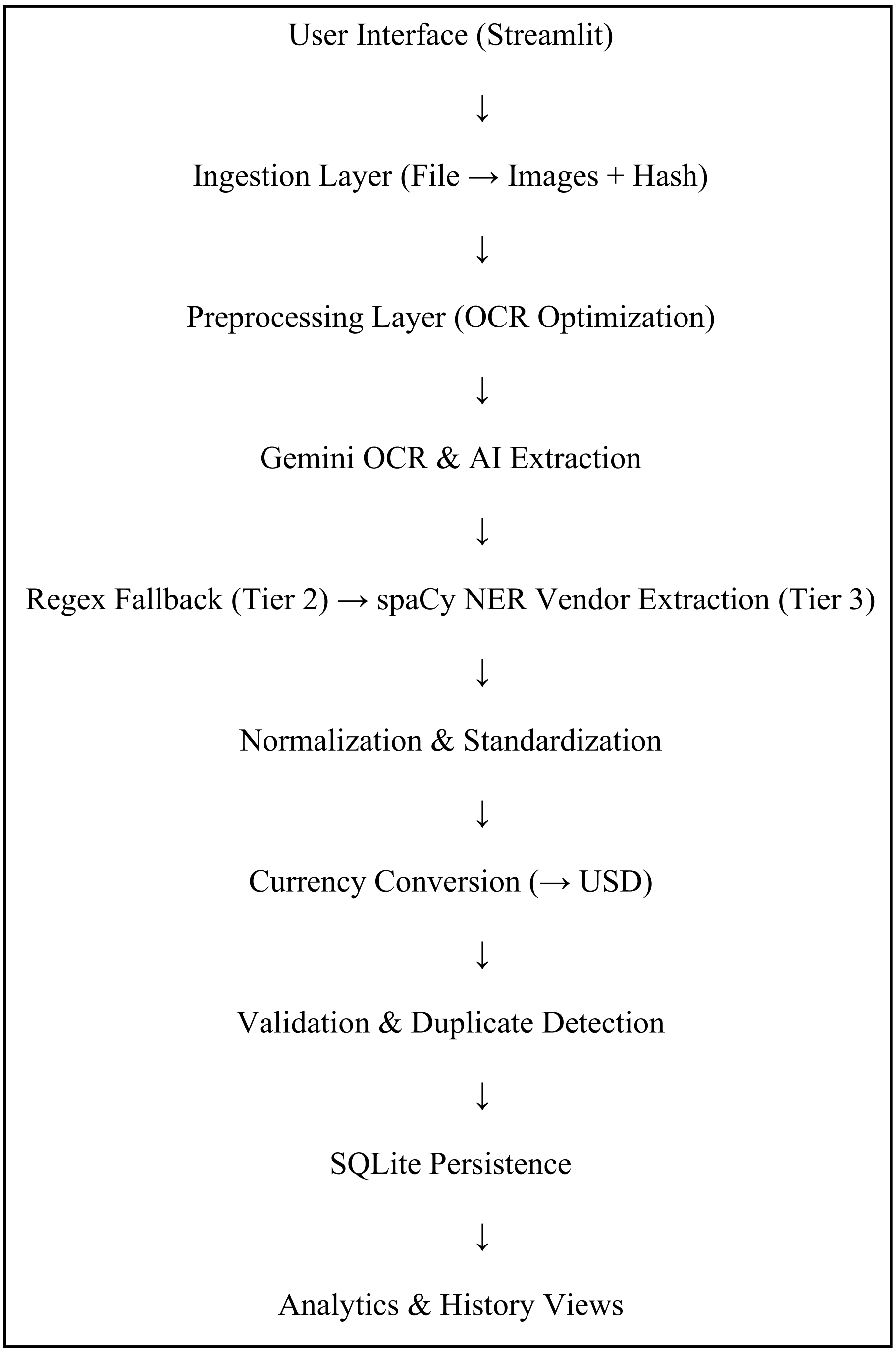
**Outcome of Milestone 2**

By the completion of Milestone 2, the system evolves from a document digitization pipeline into a reliable financial data processing platform. Extracted receipt and invoice data is deterministically recovered, normalized, validated, de-duplicated, and persistently stored in a structured database.

This milestone ensures that all stored records are accurate, consistent, and analytics-ready, establishing a solid foundation for advanced reporting, dashboards, and intelligent financial insights in subsequent milestones.

---

## 6. High-Level Architecture


User Interface (Streamlit)
↓
Ingestion Layer (File → Images + Hash)
↓
Preprocessing Layer (OCR Optimization)
↓
Gemini OCR & AI Extraction
↓
Regex Fallback (Tier 2) → spaCy NER Vendor Extraction (Tier 3)
↓
Normalization & Standardization
↓
Currency Conversion (→ USD)
↓
Validation & Duplicate Detection
↓
SQLite Persistence
↓
Analytics & History Views

---

## 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 |
| Analytics | Pandas, Plotly |
| Version Control | Git, GitHub |

---

## 8. Modules Implemented

### 8.1 Ingestion Layer (ingestion.py)

**Purpose:**

The ingestion layer acts as the **gateway** between user uploads and internal processing. Its responsibility is to **safely load documents**, normalize them, and produce a consistent internal representation.

**Key Responsibilities**

- Detect file type (image vs PDF)
- Validate file integrity
- Convert PDFs into page-wise images
- Apply security limits
- Generate file hash for change detection

**Supported Inputs**

- Local file paths
- Streamlit UploadedFile objects
- In-memory byte streams (BytesIO)

**Security Controls**

- Maximum PDF pages (prevents OOM attacks)
- Maximum image pixel threshold (prevents decompression bombs)
- SHA-256 file hashing

**Outputs**

- List[PIL.Image]
- Metadata dictionary containing:

  - Filename
  - File type
  - Number of pages
  - File hash
  - Truncation status

**Design Principle**

The ingestion layer never performs preprocessing or OCR. It only prepares data.

## 8.2 Preprocessing Layer (preprocessing.py)

**Purpose:**

Raw images captured from cameras or scanners often contain noise, skew, low contrast, or transparency. The preprocessing layer ensures images are **OCR-ready**.

**Processing Pipeline:**

1. Safe image loading with EXIF correction
2. Transparency handling (RGBA → RGB with white background)
3. Grayscale conversion
4. Contrast enhancement
5. Otsu thresholding (binarization)
6. Noise removal (median filtering)
7. Optional resizing for performance optimization

**Output:**

Clean, binary PIL image optimized for OCR

**Design Principle:**

Preprocessing is **purely visual** and **content-agnostic**.

## 8.3 OCR & Extraction Layer (ocr.py)

**Purpose:**

This module performs OCR and semantic understanding using Google Gemini AI, extracting structured receipt and invoice data in a single API call.

**Key Responsibilities**

- Send preprocessed images to Gemini Vision API
- Enforce strict JSON-only structured output

- Extract both raw OCR text and structured bill data
- Handle AI failures and malformed responses gracefully

**Extracted Data**

- Raw OCR text (for fallback and auditing)
- Vendor name
- Invoice number
- Purchase date and time
- Currency
- Line items (description, quantity, pricing)
- Tax amount
- Total amount
- Payment method

**Design Principle**
Single-call extraction minimizes latency and cost while ensuring consistent structured output.

## 8.4 Regex & NLP Fallback Extraction Layer (regex_patterns.py, field_extractor.py, vendor_extractor_spacy.py)

**Purpose**

This layer provides deterministic and NLP-assisted fallback extraction when AI-generated structured output is incomplete, ambiguous, or unreliable. It ensures critical business fields are recovered before normalization and validation.

**Key Responsibilities**

- Detect weak, missing, or unreliable fields in AI extraction
- Extract structured fields using deterministic regex patterns
- Perform vendor name recovery using spaCy Named Entity Recognition (NER)
- Apply fallback logic selectively without overwriting strong AI-derived values

**Techniques Used**

- Regex-based pattern matching for deterministic field recovery:
    - Dates
    - Invoice / receipt numbers
    - Subtotal, tax, and total amounts

    - Currency detection
    - Payment methods
- **spaCy Named Entity Recognition (NER)** for vendor name extraction:
    - Uses ORG (Organization) entity detection
    - Applied only when vendor name remains weak after regex fallback
    - Operates on raw OCR text returned by Gemini
    - If the spaCy model is unavailable, the system degrades gracefully without blocking extraction.

**Fallback Confirmation Logic**

- A field is considered *weak* if it is:
    - None, empty, or whitespace
    - Zero-value for monetary fields
    - Invalid placeholder values (null, undefined)
- Regex and spaCy fallbacks are triggered only for weak fields
- Strong AI-extracted fields are preserved and never overwritten

**Inputs**

Raw OCR text extracted by Gemini AI

**Outputs**

Partially or fully recovered structured bill fields suitable for normalization

**Design Principle**

Fallback extraction is conservative, tiered, and non-destructive:

- Tier 1: Gemini AI structured output
- Tier 2: Regex-based deterministic recovery
- Tier 3: spaCy NER–based vendor identification

Fallback logic improves robustness without introducing hallucinations or uncontrolled overrides.

## 8.5 Data Normalization Layer (normalizer.py)

**Purpose:**
The normalization layer standardizes extracted data to ensure consistency, database compatibility, and reliable querying.

**Key Responsibilities**

- Normalize date and time formats (ISO-compliant)
- Convert all textual fields to uppercase
- Enforce length constraints for database fields
- Safely convert numeric values
- Apply default values for missing fields
- Normalize line item structures

**Normalized Fields**

- Vendor name
- Invoice number
- Currency code
- Payment method
- Line item descriptions
- Monetary values

**Design Principle**

Normalization guarantees that all downstream systems receive clean, predictable, and validated data formats.

## 8.6 Currency Conversion Layer (currency_converter.py)

**Purpose:**

This module converts all extracted monetary values into USD to enable unified analytics while preserving original currency information.

**Key Responsibilities**

- Detect non-USD currencies
- Apply exchange rates for conversion
- Convert subtotal, tax, total, and line item values
- Preserve original currency, original amounts, and exchange rate metadata

**Outputs**

- USD-normalized monetary values
- Original currency audit fields

**Design Principle**

Currency conversion enhances analytics uniformity without losing financial transparency.

## 8.7 Validation Layer (validation.py)

**Purpose**

The validation layer ensures numerical consistency and financial correctness of extracted bill data before it is persisted. It focuses strictly on amount validation.

**Key Responsibilities**

- Validate financial totals for logical consistency
- Handle OCR-induced rounding and precision errors using tolerance thresholds
- Detect inconsistent or suspicious monetary values
- Provide structured validation results for downstream decision-making

**Validation Logic**

The system evaluates extracted amounts using two accepted accounting models:

- Tax-Inclusive Model

  sum(items) ≈ total_amount

- Tax-Exclusive Model

  sum(items) + tax_amount ≈ total_amount

A bill is considered valid if either model passes within a configurable tolerance (±0.02).

**Tolerance Handling**

- Small numeric deviations caused by OCR errors are tolerated
- Prevents false negatives due to rounding, formatting, or currency conversion noise

Outputs

```
{       "is_valid": bool,
        "items_sum": float,
        "tax_amount": float,
        "total_amount": float,
        "errors": [...],
        "warnings": [...]          }
```

**Design Principle**

Validation is corrective and warning-driven, not destructive.

## 8.8 Logical Duplicate Detection Layer (duplicate.py)\

**Purpose**

This layer ensures that duplicate bills are not stored multiple times by applying logical matching rules after normalization and currency conversion.

**Key Responsibilities**

- Detect hard duplicates that must block saving
- Detect soft duplicates that require user awareness
- Provide clear reasoning for duplicate detection outcomes
- Operate independently of extraction logic

**Duplicate Detection Strategy**

- Hard Duplicate Detection (Blocking):

  Matches on:

  - Invoice number
  - Vendor name (case-insensitive)
  - Purchase date
  - Total amount (within tolerance ±0.02)

- Soft Duplicate Detection (Warning):

  Applied when invoice number is missing

  Matches on:

  - Vendor name
  - Purchase date
  - Total amount (within tolerance)

**Inputs**

- Fully normalized bill data
- User ID (for future multi-user isolation)

**Outputs**

```
{ "duplicate": bool,
  "soft_duplicate": bool,
```

"reason": "Human-readable explanation" }

**Design Principle**

Duplicate detection is logical, deterministic, and isolated:

## 8.9 Database Persistence Layer (database.py)

**Purpose:**

Provides reliable, persistent storage of bills and line items using a relational database model.

**Key Responsibilities**

- Initialize database schema
- Insert bills and associated line items
- Enforce foreign key relationships
- Support cascade deletion
- Enable retrieval for history and analytics

**Storage Model**

- bills table (header-level data)
- lineitems table (item-level data)

**Design Principle**

Database operations are atomic, transactional, and integrity-safe.

**Table: bills**

Stores high-level information about each receipt or invoice.

| Column Name | Data Type | Description |
|---|---|---|
| bill_id | INTEGER (Primary Key, Auto Increment) | Unique identifier for each bill |
| user_id | INTEGER (Default: 1) | User identifier (future multi-user support) |
| invoice_number | VARCHAR(100) | Invoice or receipt number |
| vendor_name | VARCHAR(255) NOT NULL | Name of the merchant or vendor |
| purchase_date | DATE NOT NULL | Date of transaction |
| purchase_time | TIME | Time of transaction (if available) |
| subtotal | DECIMAL(10,2) | Amount before tax |
| tax_amount | DECIMAL(10,2) | Tax applied to the bill |
| total_amount | DECIMAL(10,2) | Final payable amount |
| currency | VARCHAR(10) | Stored currency (USD after conversion) |

| | | |
|---|---|---|
| original_currency | VARCHAR(10) | Currency detected from receipt |
| original_total_amount | DECIMAL(10,2) | Total amount in original currency |
| exchange_rate | DECIMAL(10,6) | Exchange rate used for conversion |
| payment_method | VARCHAR(50) | Payment mode (Cash, Card, UPI, etc.) |
| created_at | TIMESTAMP | Record creation timestamp |

**Table: lineitems**

Stores individual item-level details linked to a bill.

| Column Name | Data Type | Description |
|---|---|---|
| item_id | INTEGER (Primary Key, Auto Increment) | Unique identifier for each line item |
| bill_id | INTEGER NOT NULL | Reference to parent bill |
| description | TEXT | Item name or description |
| quantity | INTEGER | Quantity purchased |
| unit_price | DECIMAL(10,2) | Price per unit |
| total_price | DECIMAL(10,2) | Total price for the item |

## 8.10 User Interface & Workflow Layer (app.py)

**Purpose:**

This module orchestrates the complete user workflow and presents extracted data through an intuitive interface.

**Key Responsibilities**

- Manage multi-page navigation
- Maintain session state across reruns
- Provide upload, processing, and save controls
- Display extracted data and validation feedback
- Support history browsing and analytics access

**Design Principle**

The UI layer is user-centric, stateful, and resilient to reruns and partial failures.

---

# 9 Streamlit Application Design

### 9.1 Session State Management

Streamlit reruns the entire script on every interaction.

Session state is used to preserve:

- Uploaded file context
- API key
- Preprocessed images
- OCR results
- Navigation state
- Save status

**This prevents:**

- Redundant OCR calls
- Data loss on reruns
- UI inconsistency

### 9.2 Sidebar Navigation

The sidebar manages:

- **Gemini API key input**
- **Page navigation**
- **Application metadata**

The API key is stored only in session memory and never logged or persisted.

### 9.3 Upload & Process Workflow

**Supported Scenarios:**

- Single image receipts
- Single-page PDFs
- Multi-page PDFs (page-wise processing)

User Interface (Streamlit Web Application)

↓

File Upload Layer

(JPG / PNG / PDF)

↓

Ingestion Layer

(File Type Detection, Hash Generation, PDF-to-Image Conversion)

↓

Image Preprocessing Layer

(Grayscale, Noise Reduction, Binarization, Contrast Enhancement)

↓

User Action Trigger

("Save My Bill")

↓

OCR & Structured Extraction Layer

(Google Gemini Vision API – Single Call)

↓

Regex & NLP Fallback Layer

(Weak Field Detection, Regex Parsing, Vendor NLP spay)

↓

Normalization Layer

(Date/Time Formatting, Uppercase Standardization, Type Safety)

↓

Currency Conversion Layer

(Multi-Currency → USD with Original Value Preservation)

↓

Validation & Duplicate Detection Layer

(Amount Consistency Checks, Logical Duplicate Detection)

↓

Persistent Storage Layer

(SQLite Database – Bills & Line Items)

↓

Presentation & Analytics Layer

(Streamlit Tables, History View, Dashboard Metrics)

**9.4 Results Display**

Results are presented using:

- Uploaded image
- Preprocessed image previews
- Structured bill data
- File metadata

Tabbed layout improves readability and user experience.

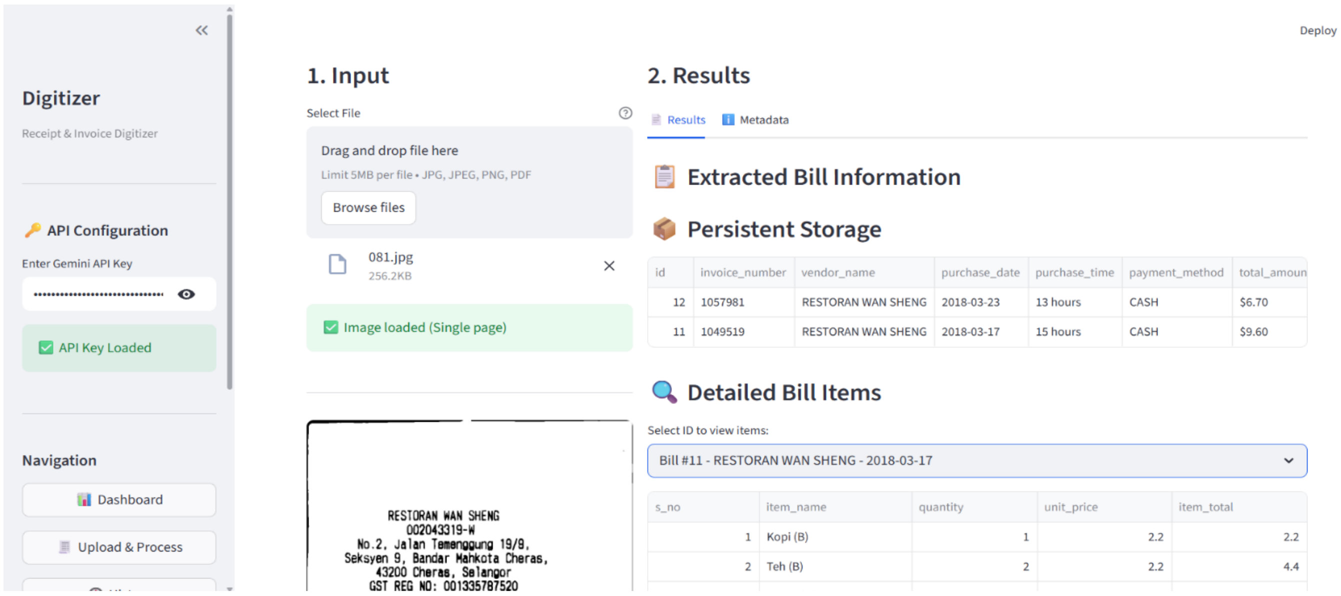


Image loaded (Single page)

RESTORAN WAN SHENG
002043319-W
No.2, Jalan Temenggung 19/9,
Seksyen 9, Bandar Mahkota Cheras,
43200 Cheras, Selangor
GST REG NO: 001335787520

Extracted Bill Information

Persistent Storage

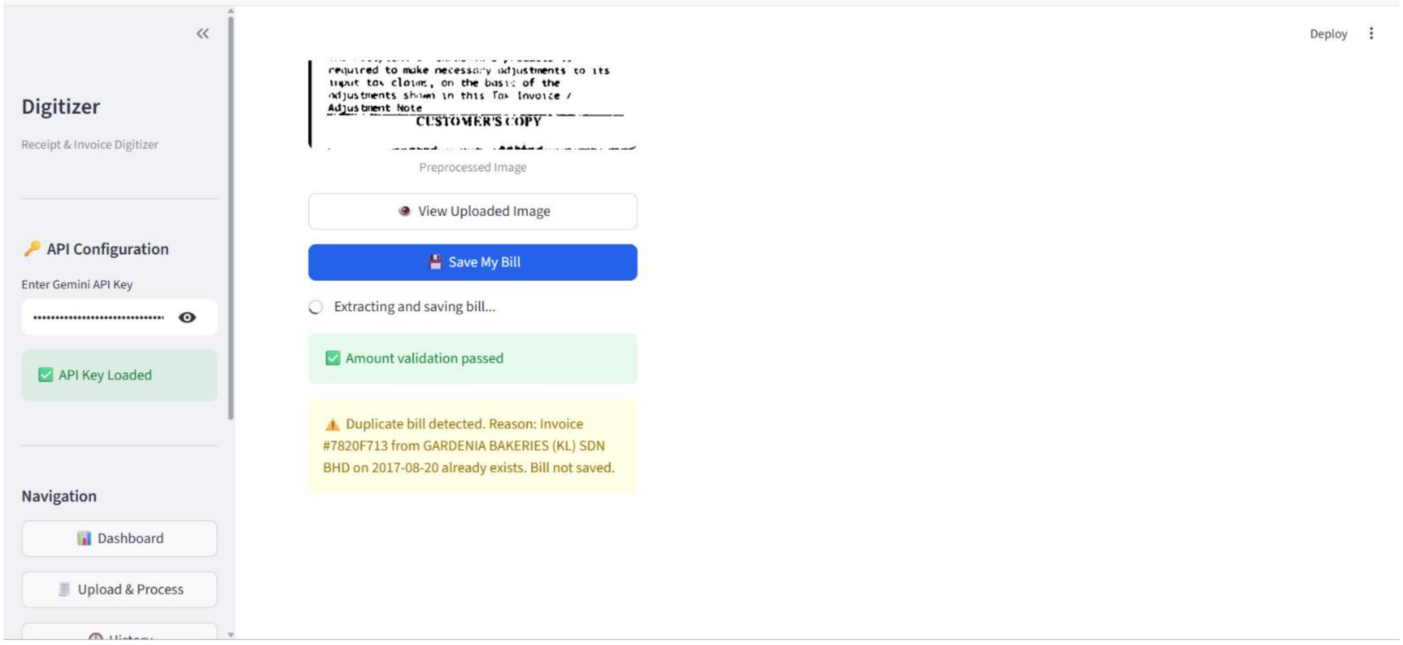
| id | invoice_number | vendor_name | purchase_date | purchase_time | payment_method | total_amoun |
|---|---|---|---|---|---|---|
| 12 | 1057981 | RESTORAN WAN SHENG | 2018-03-23 | 13 hours | CASH | $6.70 |
| 11 | 1049519 | RESTORAN WAN SHENG | 2018-03-17 | 15 hours | CASH | $9.60 |

Detailed Bill Items

Select ID to view items:

Bill #11 - RESTORAN WAN SHENG - 2018-03-17

| s_no | item_name | quantity | unit_price | item_total |
|---|---|---|---|---|
| 1 | Kopi (B) | 1 | 2.2 | 2.2 |
| 2 | Teh (B) | 2 | 2.2 | 4.4 |



required to make necessary adjustments to its input tax claims, on the basis of the adjustments shown in this Tax Invoice / Adjustment Note

CUSTOMER'S COPY

Extracting and saving bill...

Amount validation passed

Duplicate bill detected. Reason: Invoice #7820F713 from GARDENIA BAKERIES (KL) SDN BHD on 2017-08-20 already exists. Bill not saved.

## 10 Error Handling & Validation

The system implements structured error handling and validation to ensure data reliability, system stability, and user trust throughout the digitization workflow.

### 10.1 Upload-Level Validation

- Supports only JPG, PNG, and PDF formats
- Enforces maximum file size limits
- Detects unchanged re-uploads using file hashing
- Prevents invalid files from entering the pipeline

### 10.2 Ingestion & Preprocessing Safety

- Handles corrupted or partially readable files gracefully
- Limits PDF pages to prevent memory issues
- Falls back to original images if preprocessing fails

### 10.3 OCR & AI Extraction Handling

- Enforces JSON-only output from Gemini AI
- Detects malformed or incomplete AI responses
- Requests raw OCR text for fallback extraction

### 10.4 Regex & NLP Fallback Recovery

- Identifies weak or missing fields
- Applies regex-based extraction for dates, totals, tax, currency
- Uses NLP model spacy for vendor name detection
- Overrides only weak fields, preserving strong AI results

### 10.5 Data Normalization Controls

- Converts all values to safe, consistent formats
- Standardizes text to uppercase
- Applies default values for missing fields
- Prevents type and schema errors before storage

### 10.6 Currency Conversion Validation

- Converts non-USD currencies safely
- Preserves original currency and exchange rate
- Skips conversion if unsupported without failing

### 10.7 Amount Validation

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

### 10.8 Duplicate Detection

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

### 10.9 Database Safety

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

### 10.10 User Feedback

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

---

## 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 |

## 12 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, and multi-user support in subsequent milestones.