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. Milestone 3 Objective

The primary objective of Milestone 3 is to transform the structured, validated, and persistently stored financial data generated in Milestones 1 and 2 into meaningful analytical insights, interactive visualizations, and intelligent summaries.

While previous milestones focused on ingestion, extraction, validation, and storage, Milestone 3 introduces a full analytics layer capable of:

• Providing real-time financial visibility
• Enabling interactive dashboard-based exploration
• Generating automated statistical insights
• Delivering AI-generated natural language summaries
• Supporting export and reporting capabilities

This milestone completes the data lifecycle by enabling actionable intelligence derived from stored receipt and invoice data.

Key Objectives

1. **Interactive Analytics Dashboard**

   Develop a fully interactive dashboard using Streamlit that enables users to visualize and explore financial data through:

   • Key performance indicators (KPIs)
   • Time-series spending trends
   • Vendor-level spending breakdowns
   • Payment method analysis
   • Transaction frequency analysis
   • Item-level spending analysis

   The dashboard must update dynamically based on user-selected filters such as date range and payment method.

2. **Structured Data Analytics Engine**

   Implement a modular analytics engine capable of computing statistical aggregations and derived datasets required for visualization and reporting.

   The analytics engine must support:

   • Monthly spending aggregation
   • Transaction frequency analysis
   • Vendor spending rankings
   • Payment method distribution
   • Tax and subtotal breakdowns
   • Item-level aggregation

   All analytics computations must operate efficiently using in-memory Pandas dataframes.

3. **Advanced Data Visualization Layer**

   Implement a professional visualization layer using Plotly to render interactive charts, including:

   • Line charts for spending trends
   • Bar charts for transaction counts and vendor rankings
   • Pie charts for vendor and payment distribution
   • Histogram charts for transaction value analysis

• Cumulative spending curves
• Year-over-year comparison charts

Charts must support:

• Hover interactions
• Tooltips
• Dynamic filtering
• Responsive layout

4. **Automated Statistical Insight Generation**

Implement a statistical insight engine that converts numerical analytics into human-readable summaries.

This engine must automatically identify:

• Spending increases or decreases
• Highest spending vendors
• Most frequently purchased items
• Dominant payment methods
• Spending peaks and patterns

Insights must be generated dynamically based on filtered data.

5. **AI-Generated Financial Insights using Gemini**

Integrate Google Gemini AI to generate intelligent financial summaries and interpretations based on structured analytics data.

The AI insights engine must:

• Generate structured financial analysis
• Highlight anomalies and trends
• Identify behavioral patterns
• Provide business-level financial interpretation

AI generation must be deterministic, structured, and triggered only on user request.

6. **Efficient Data Retrieval and Caching**

   Implement optimized data retrieval mechanisms to ensure fast dashboard performance.

   This includes:

   • Cached database queries
   • Efficient dataframe transformations
   • Lazy computation of expensive analytics
   • Minimal redundant computation

   Caching improves responsiveness and scalability.

7. **Export and Reporting Capability**

   Enable export of financial data into standard reporting formats including:

   • CSV export
   • Excel export
   • PDF export
   • Detailed line-item export

   Exports must support both summary and detailed views.

8. **Modular and Scalable Dashboard Architecture**

   Ensure the dashboard architecture is modular, allowing independent development and extension of:

   • Analytics logic
   • Chart rendering
   • Insight generation
   • AI integration
   • Export functionality

   This design ensures long-term maintainability and scalability.

Outcome of Milestone 3

By the completion of Milestone 3, the system evolves from a data extraction and storage platform into a full-featured financial analytics system.

The system now provides:

• Interactive dashboard visualization
• Automated statistical insights
• AI-generated financial summaries
• Vendor and payment analysis
• Spending trend analysis
• Exportable financial reports

This milestone completes the intelligent analytics layer, enabling users to derive meaningful insights, track financial behavior, and make informed decisions based on receipt and invoice data.

7. Milestone 4 Objective

The primary objective of Milestone 4 is to enhance extraction accuracy, improve dashboard usability, and optimize database performance by introducing template-based parsing, advanced search and filtering capabilities, and efficient query handling mechanisms.

While Milestones 1 and 2 established a reliable extraction, validation, and storage pipeline, and Milestone 3 introduced analytics and visualization, Milestone 4 focuses on improving precision, performance, and usability through deterministic parsing and optimized data access. This milestone ensures that extracted financial data is not only accurate and structured, but also easily searchable, efficiently retrievable, and highly reliable for analytical and reporting purposes.

Milestone 4 is intentionally scoped to address real-world challenges such as vendor-specific receipt layouts, extraction inconsistencies caused by OCR variations, slow query performance with growing datasets, and limited dashboard interactivity. These enhancements are implemented without modifying the core ingestion or OCR architecture, ensuring system stability while improving accuracy and performance.

Key Objectives

1. **Template-Based Parsing for Vendor-Specific Extraction**

   Introduce a deterministic template-based parsing system to improve extraction accuracy for known vendor formats. The system must identify vendor-specific templates using keyword matching and apply structured parsing rules to extract critical fields such as invoice number, purchase date, subtotal, tax, total amount, and line items. Template parsing must operate as a supplementary extraction layer that fills missing or weak fields without overriding reliable AI-extracted values.

2. **Template Integration with Existing Extraction Pipeline**

   Integrate the template parser into the existing extraction workflow to ensure seamless operation with AI and regex-based extraction layers. The template parser must execute after vendor detection and fallback extraction, allowing it to enhance structured output while maintaining compatibility with existing normalization, validation, and persistence mechanisms.

3. **Vendor Template Registry and Extensible Parsing Framework**

   Implement a modular template registry system that enables easy addition and management of vendor-specific templates. Templates must define vendor aliases, extraction patterns, and field rules in a structured format, allowing new vendors to be supported without modifying core extraction logic. This ensures long-term extensibility and scalability of the extraction system.

4. **Advanced Search and Filtering Capability in Dashboard**

   Enhance the dashboard by implementing advanced search and filtering functionality that enables efficient exploration of stored financial records. The system must support filtering based on vendor name, date range, transaction amount, and payment method. Filtering must be performed at the database level to ensure efficient query execution and consistent results across dashboard analytics and reports.

5. **Optimized Database Query Execution and Data Retrieval**

   Improve database query efficiency by implementing optimized query patterns and reducing unnecessary in-memory data processing. The system must retrieve only required records and fields based on user-selected filters, ensuring fast dashboard loading and scalable performance as the volume of stored records increases.

6. **Improved Dashboard Performance and Responsiveness**

   Enhance dashboard responsiveness by minimizing redundant database calls, optimizing analytics queries, and using caching mechanisms where appropriate. The system must ensure fast loading of charts, metrics, and insights while maintaining accuracy and consistency of displayed financial data.

7. **Efficient Reporting and Aggregation Mechanisms**

   Optimize reporting and aggregation workflows to improve performance of financial summaries and analytics computations. Aggregation operations such as monthly spending totals, vendor analysis, and payment distribution must be efficiently computed using optimized database queries and structured data processing.

8. **Improved Data Reliability and Extraction Consistency**

   Enhance overall system reliability by reducing extraction ambiguity and improving structured data completeness. Template-based parsing ensures consistent extraction of critical financial fields for known vendor formats, reducing errors and improving accuracy of downstream analytics and reporting.

9. **Seamless Integration with Existing Analytics and Persistence Layers**

   Ensure that all Milestone 4 enhancements integrate seamlessly with existing database, analytics, and dashboard modules. The system must preserve compatibility with normalization, validation, storage, and visualization layers while improving performance and extraction accuracy.

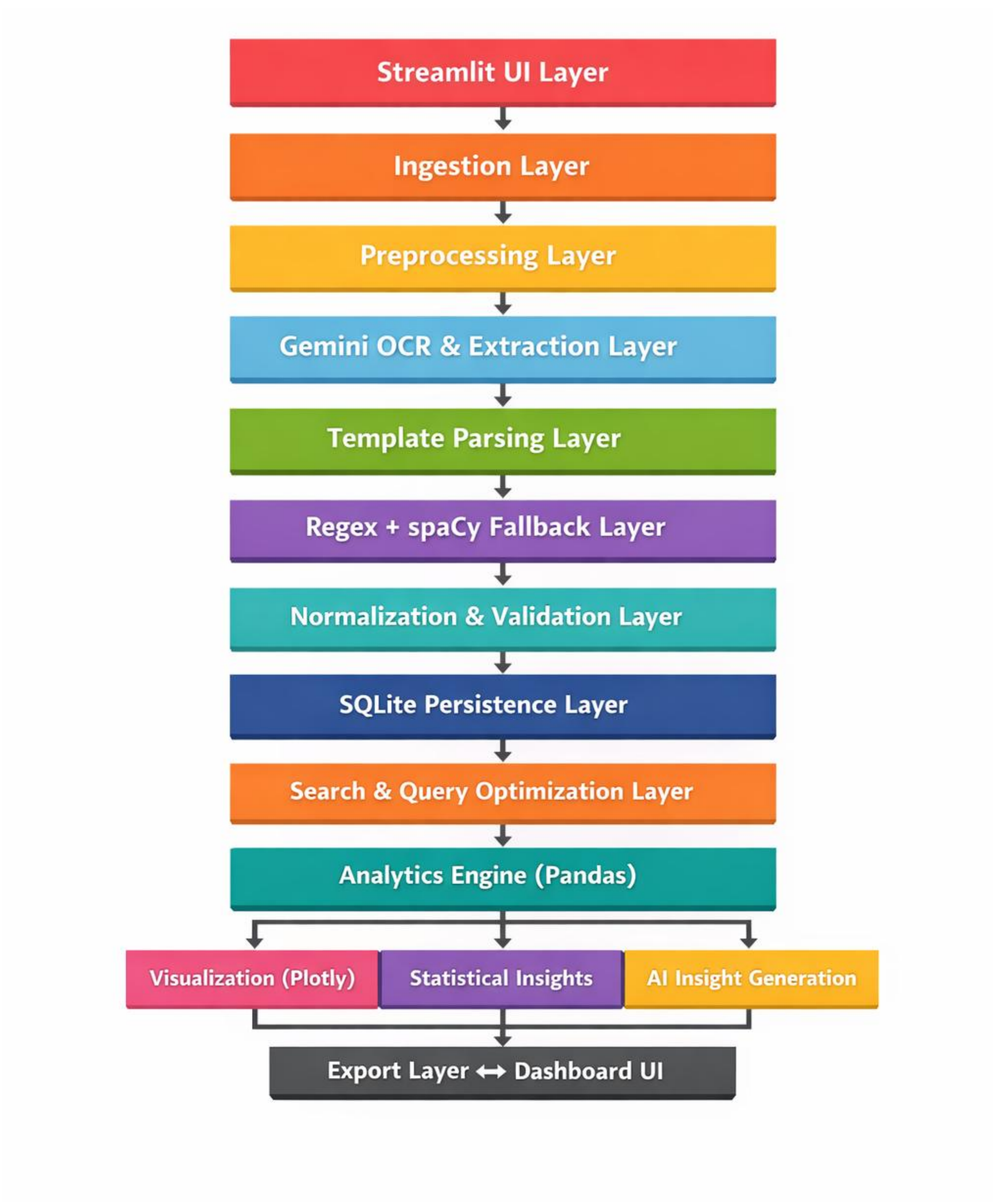
Outcome of Milestone 4

By the completion of Milestone 4, the system evolves into a highly accurate, efficient, and user-friendly financial document intelligence platform. Template-based parsing improves extraction precision for vendor-specific formats, while advanced search and filtering capabilities enable efficient exploration of stored financial data.

Database query optimization and reporting improvements ensure fast dashboard performance and scalable analytics, even as data volume grows. These enhancements significantly improve system accuracy, usability, and responsiveness while maintaining architectural stability and modularity.

This milestone completes the polishing and integration phase of development, ensuring reliable extraction, efficient querying, and high-performance financial analytics for real-world usage.

8. High-Level Architecture


Streamlit UI Layer
Ingestion Layer
Preprocessing Layer
Gemini OCR & Extraction Layer
Template Parsing Layer
Regex + spaCy Fallback Layer
Normalization & Validation Layer
SQLite Persistence Layer
Search & Query Optimization Layer
Analytics Engine (Pandas)
Visualization (Plotly)
Statistical Insights
AI Insight Generation
Export Layer ↔ Dashboard UI

9. Technology Stack

Layer	Technology	Purpose
User Interface (UI)	Streamlit	Multi-page interactive web application for document upload, dashboard visualization, analytics, AI insights, and search/filter functionality
Backend Language	Python 3.13+	Core programming language used for ingestion, extraction, template parsing, analytics, AI integration, and dashboard logic
OCR & AI Extraction	Google Gemini API (gemini-2.5-flash)	Performs OCR, structured receipt extraction, and AI-generated financial insights
Template Parsing Engine (Milestone 4)	Custom Python modules, JSON-based vendor templates, Python re	Improves extraction accuracy using vendor-specific deterministic parsing and structured extraction rules
Image Processing	OpenCV, Pillow (PIL)	Image preprocessing including grayscale conversion, binarization, noise removal, and contrast enhancement
PDF Handling	pdf2image, Poppler	Converts PDF documents into OCR-ready images
NLP & Regex Processing	spaCy (en_core_web_sm), Python re	Vendor name extraction using NER and deterministic fallback extraction using regex
Data Normalization & Validation	Custom Python modules	Standardizes extracted fields, enforces schema consistency, validates financial correctness
Currency Conversion	Custom Python module + Exchange Rate API / static rates	Converts multi-currency receipts into USD while preserving original currency metadata
Database & Persistence	SQLite	Relational database storing bills and line items with transactional integrity
Query Optimization & Search Layer (Milestone 4)	SQLite indexed queries, custom SQL filtering functions	Enables efficient search, filtering, and fast retrieval of financial records

Data Retrieval Layer	Custom database.py module	Provides structured APIs for querying bills, filtered records, and analytics data
Analytics Engine	Pandas	Performs statistical aggregation, KPI computation, and financial analytics
Visualization Engine	Plotly (plotly.graph_objects)	Builds interactive charts including line, bar, pie, histogram, and comparison charts
AI Insight Generation	Google Gemini API, hashlib, json	Generates intelligent financial summaries using structured analytics data
Dashboard & State Management	Streamlit Session State, Streamlit Cache	Maintains application state, caching, and optimized dashboard performance
Export & Reporting	Pandas, ReportLab, XLSXWriter, io.BytesIO	Enables export of financial data to CSV, Excel, and PDF formats
File Security & Hashing	hashlib (SHA-256)	Ensures file integrity and prevents duplicate processing
Date & Time Handling	datetime	Handles transaction date parsing, filtering, and normalization
Data Serialization	JSON	Structured data exchange between extraction, template parser, and AI services
Dependency Management	pip, virtualenv / venv	Python package and environment management
Version Control	Git, GitHub	Source code management, version tracking, and collaboration
Development Environment	VS Code / PyCharm	Code development, debugging, and project management
Operating System Support	Windows, Linux, macOS	Cross-platform compatibility

10.Modules Implemented

10.1 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.

10.2 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**.

10.3 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.

10.4 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.

10.5 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.

10.6 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.

10.7 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.

10.8 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:

10.9 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

10.10 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.

10.11 Dashboard UI Layer (dashboard_page.py)

Purpose:

Provides the main Streamlit dashboard interface and orchestrates analytics, charts, insights, and AI features.

Key Responsibilities:

• Load bills and line items from database
• Apply filters (date range, payment method)
• Compute KPIs
• Render charts and visualizations
• Display statistical insights
• Trigger AI insights generation
• Provide export functionality

Key Functions:

• page_dashboard() — Main dashboard entry point
• _cached_bills() — Cached loader for bill records
• _cached_items() — Cached loader for line items
• _render_ai_insights() — Render AI insights output

Dependencies:

• analytics.py
• charts.py
• insights.py
• ai_insights.py
• exports.py
• database.py

10.12 Analytics Layer (analytics.py)

Purpose:

Provides pure data processing and aggregation logic.

Key Responsibilities:

- Clean and normalize dataframe structure
- Compute KPIs and aggregated metrics
- Prepare vendor analytics
- Prepare payment analytics
- Prepare item analytics

Key Functions:

- prepare_bills_dataframe()
- calculate_kpis()
- monthly_spending()
- monthly_transaction_counts()
- top_vendors()
- payment_distribution()
- high_value_transactions()
- prepare_items_dataframe()

Dependencies:

- pandas

Design Principle:

Pure computation layer with no UI or visualization logic.

10.13 Visualization Layer (charts.py)

Purpose:
Builds interactive Plotly charts for dashboard visualization.

Key Responsibilities:

- Generate line charts
- Generate bar charts
- Generate pie charts
- Generate histogram charts
- Generate comparison charts

Key Functions:

- monthly_spending_line()

• vendor_bar_chart()

• payment_method_pie()

• transaction_histogram()

• top_items_bar()

• yoy_comparison()

Dependencies:

• plotly.graph_objects

Design Principle:

Chart rendering is isolated from data processing logic.

10.14 Statistical Insights Layer (insights.py)

Purpose:
Generates human-readable insights from statistical analytics.

Key Responsibilities:

• Interpret numerical analytics

• Generate meaningful textual insights

• Highlight trends and anomalies

Key Functions:

• monthly_spending_insight()

• vendor_insight()

• payment_insight()

• transaction_histogram_insight()

• top_items_insight()

Dependencies:

• pandas

Design Principle:

Transforms analytics into user-friendly explanations.

10.15 AI Insight Generation Layer (ai_insights.py)

Purpose:
Uses Google Gemini AI to generate intelligent financial analysis.

Key Responsibilities:

- Build structured summary from analytics
- Generate AI insights using Gemini API
- Cache results using summary hash
- Enforce structured AI output format

Key Functions:

- build_summary()
- summary_hash()
- generate_ai_insights()

Dependencies:

- google.genai
- pandas
- hashlib
- json

Design Principle:

AI provides higher-level interpretation beyond statistical insights.

10.16 Export Layer (exports.py)

Purpose:
Provides data export capabilities.

Key Responsibilities:

- Export data to CSV
- Export data to Excel
- Export data to PDF
- Export detailed bill and line-item data

Key Functions:

• export_csv()
• export_excel()
• export_pdf()
• export_detailed_csv()
• export_detailed_excel()
• export_detailed_pdf()

Dependencies:

• pandas
• reportlab
• xlsxwriter

Design Principle:

Supports reporting and external data sharing.

10.17 Template Parsing Layer (template_parser.py)

Purpose:
Provides vendor-specific template-based parsing to improve extraction accuracy by applying deterministic extraction rules to OCR text.

Key Responsibilities:
• Load vendor templates from JSON files and cache them for efficient access
• Match vendor names using alias normalization and template lookup
• Extract structured fields such as invoice number, date, subtotal, tax, and total using regex and label-based parsing
• Parse structured line items using template-defined patterns, markers, and capture groups
• Safely return partial extraction results without overwriting reliable AI-extracted data
• Improve extraction reliability for known vendor formats

Key Functions:
• find_template_for_vendor()
• parse_with_template()
• _load_templates()
• _normalize_vendor_key()
• _find_first()
• _find_amount_after_label()

• _slice_lines_by_markers()

• _parse_line_items()

Dependencies:

• json

• os

• re

• typing (Dict, List, Optional, Any)

• regex_patterns.py (AMOUNT_PATTERN)

Design Principle:

Provides deterministic, vendor-aware structured extraction using external JSON templates, improving accuracy while maintaining modularity, extensibility, and compatibility with the existing extraction pipeline.

11. Streamlit Application Design

11.1 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

11.2 Sidebar Navigation

The sidebar manages:

- **Gemini API key input**

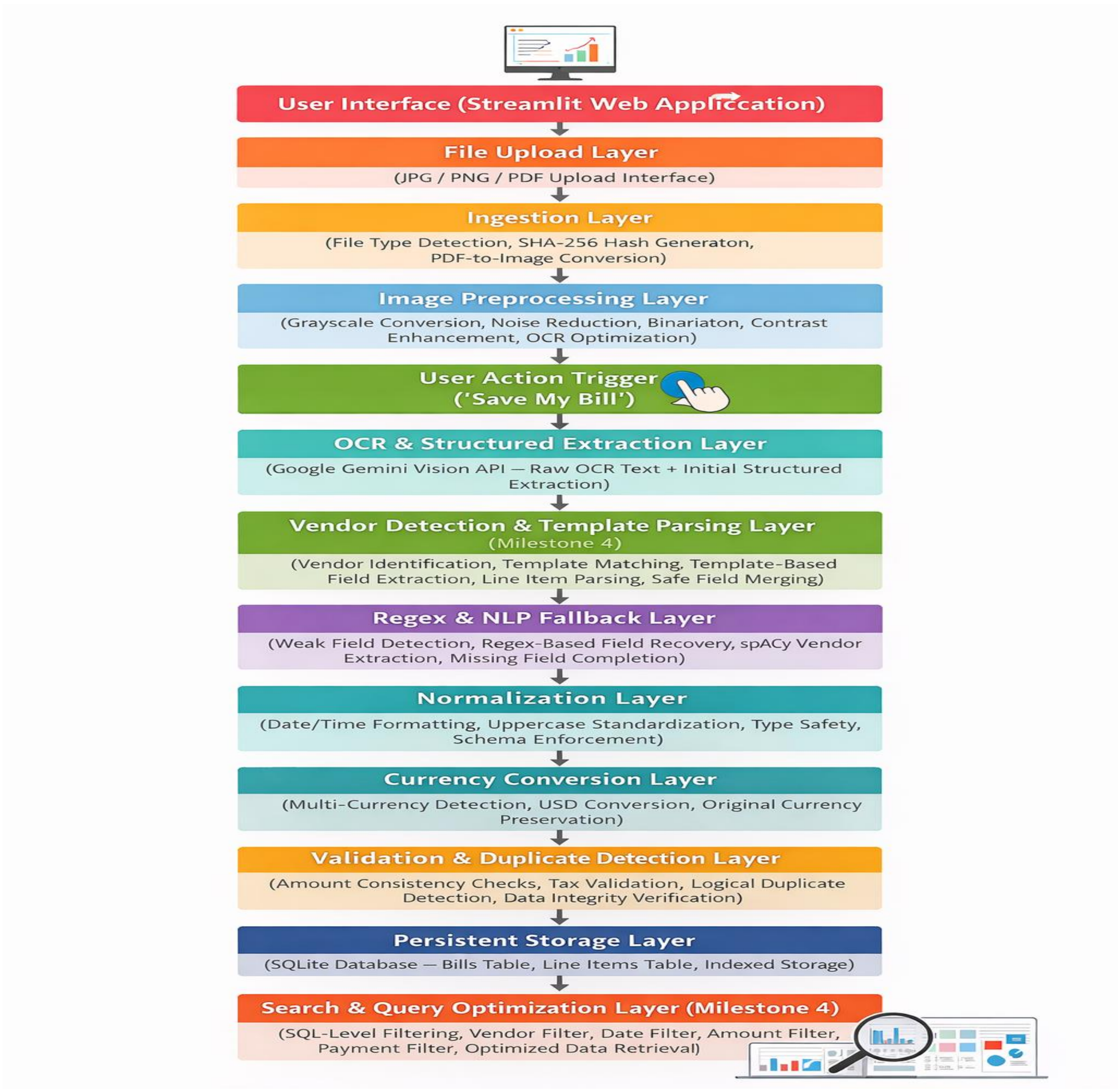
- **Page navigation**
- **Application metadata**

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

11.3 Upload & Process Workflow

Supported Scenarios:

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



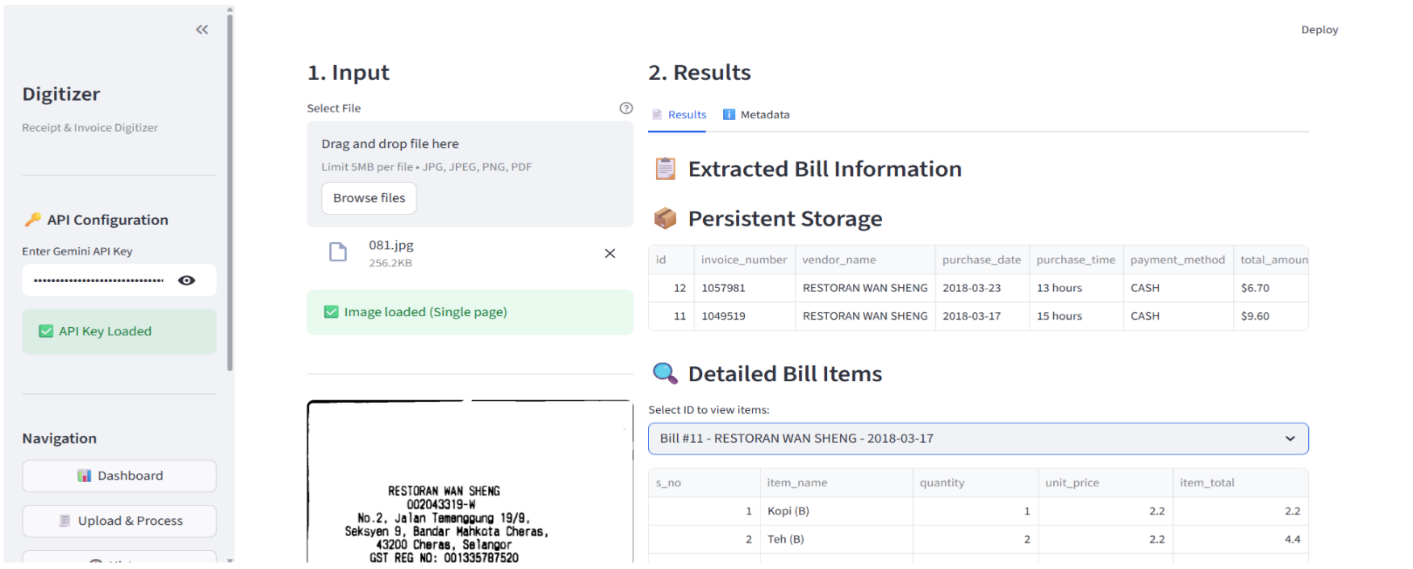
User Interface (Streamlit Web Appliccation)
File Upload Layer
(JPG / PNG / PDF Upload Interface)
Ingestion Layer
(File Type Detection, SHA-256 Hash Generaton, PDF-to-Image Conversion)
Image Preprocessing Layer
(Grayscale Conversion, Noise Reduction, Binariaton, Contrast Enhancement, OCR Optimization)
User Action Trigger ('Save My Bill')
OCR & Structured Extraction Layer
(Google Gemini Vision API — Raw OCR Text + Initial Structured Extraction)
Vendor Detection & Template Parsing Layer
(Milestone 4)
(Vendor Identification, Template Matching, Template-Based Field Extraction, Line Item Parsing, Safe Field Merging)
Regex & NLP Fallback Layer
(Weak Field Detection, Regex-Based Field Recovery, spACy Vendor Extraction, Missing Field Completion)
Normalization Layer
(Date/Time Formatting, Uppercase Standardization, Type Safety, Schema Enforcement)
Currency Conversion Layer
(Multi-Currency Detection, USD Conversion, Original Currency Preservation)
Validation & Duplicate Detection Layer
(Amount Consistency Checks, Tax Validation, Logical Duplicate Detection, Data Integrity Verification)
Persistent Storage Layer
(SQLite Database — Bills Table, Line Items Table, Indexed Storage)
Search & Query Optimization Layer (Milestone 4)
(SQL-Level Filtering, Vendor Filter, Date Filter, Amount Filter, Payment Filter, Optimized Data Retrieval)

11.4 Results Display

Results are presented using:

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

Tabbed layout improves readability and user experience.



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

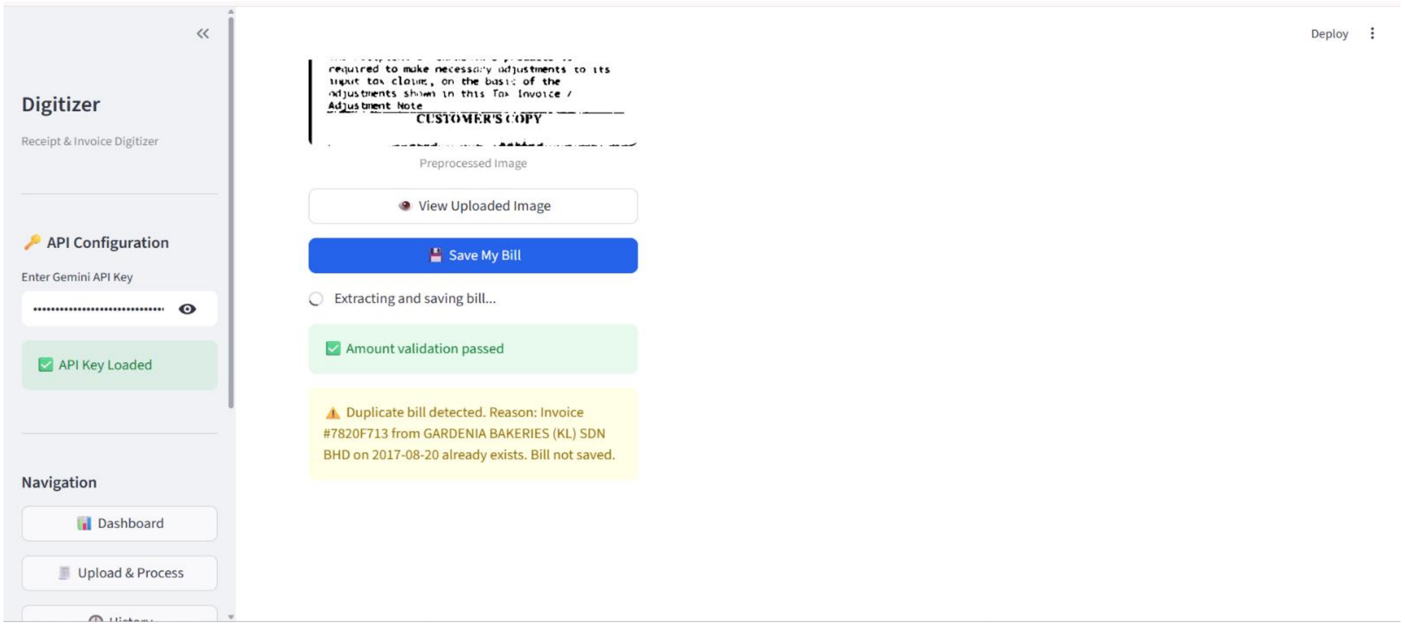
id	invoice_number	vendor_name	purchase_date	purchase_time	payment_method	total_amoun
12	1057981	RESTORAN WAN SHENG	2018-03-23	13 hours	CASH	$6.70
11	1049519	RESTORAN WAN SHENG	2018-03-17	15 hours	CASH	$9.60

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



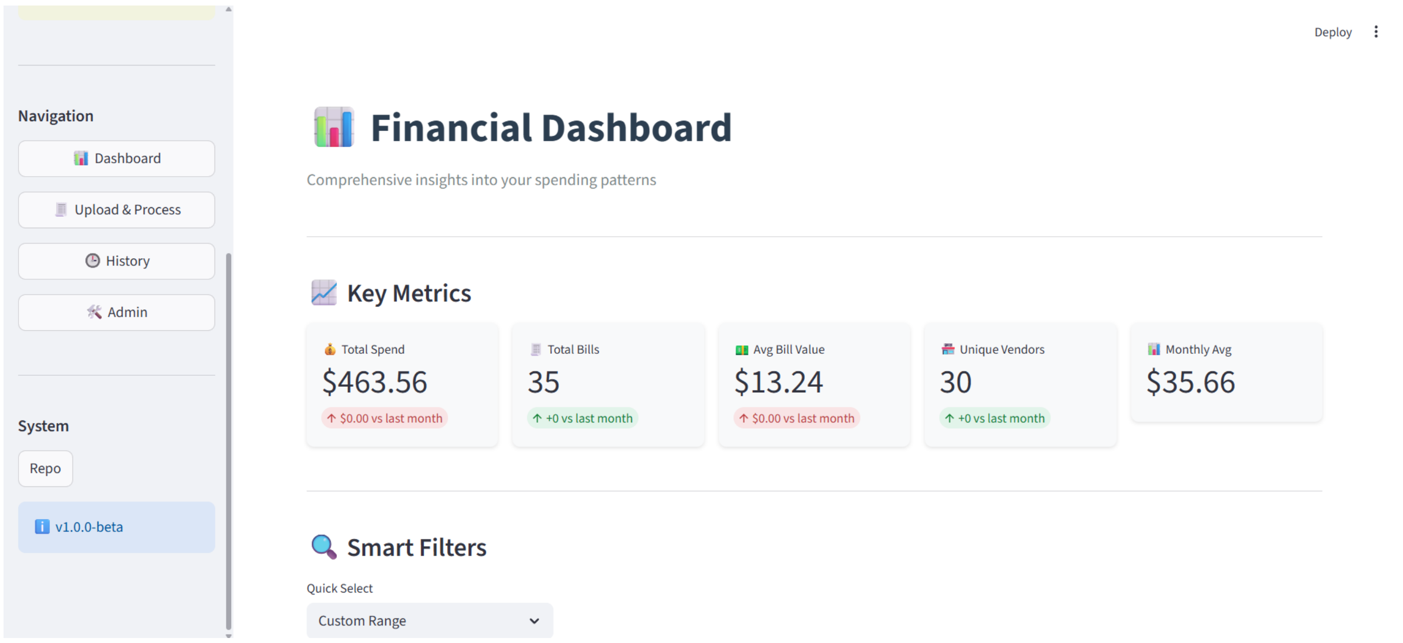
required to make necessary adjustments to its input tax claim, on the basis of the adjustments shown in this Tax Invoice / Adjustment Note

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.



Navigation

📊 Dashboard

Upload & Process

History

Admin

System

Repo

v1.0.0-beta

Deploy

📊 Financial Dashboard

Comprehensive insights into your spending patterns

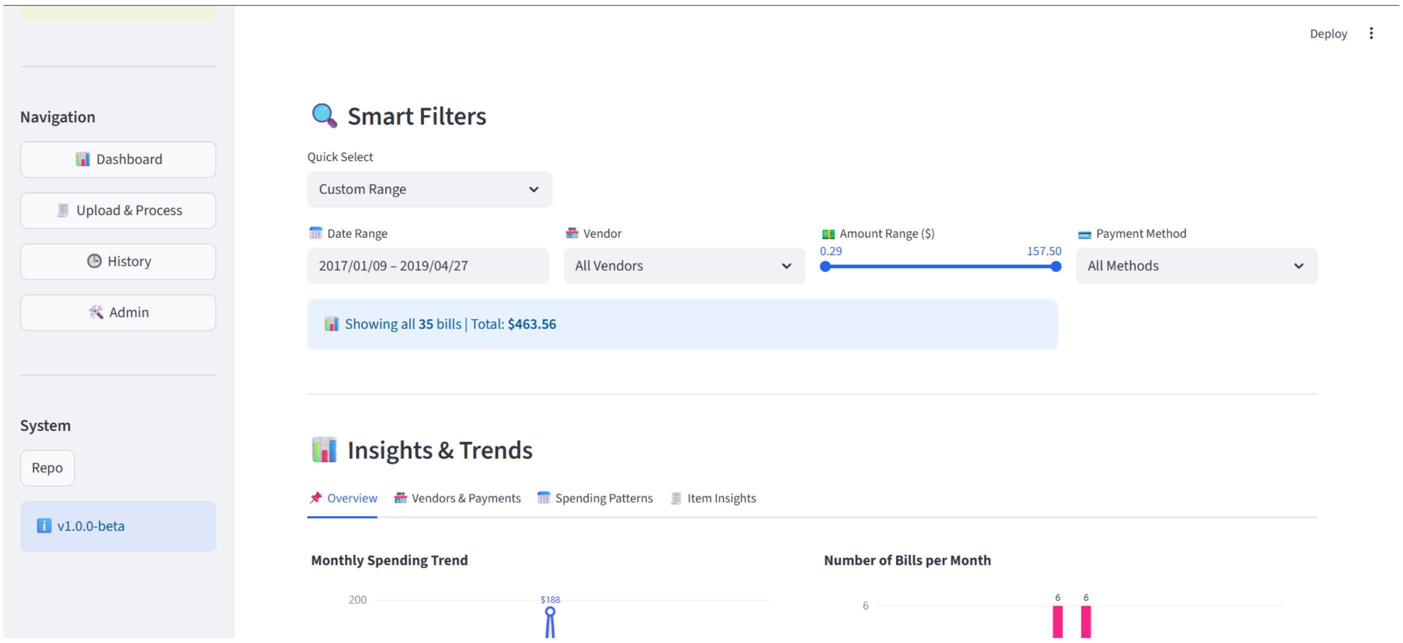
📈 Key Metrics

Total Spend	Total Bills	Avg Bill Value	Unique Vendors	Monthly Avg
$463.56	35	$13.24	30	$35.66
↑ $0.00 vs last month	↑ +0 vs last month	↑ $0.00 vs last month	↑ +0 vs last month	

🔍 Smart Filters

Quick Select

Custom Range



Navigation

Dashboard

Upload & Process

History

Admin

System

Repo

v1.0.0-beta

Deploy

🔍 Smart Filters

Quick Select

Custom Range

Date Range: 2017/01/09 – 2019/04/27

Vendor: All Vendors

Amount Range ($): 0.29 – 157.50

Payment Method: All Methods

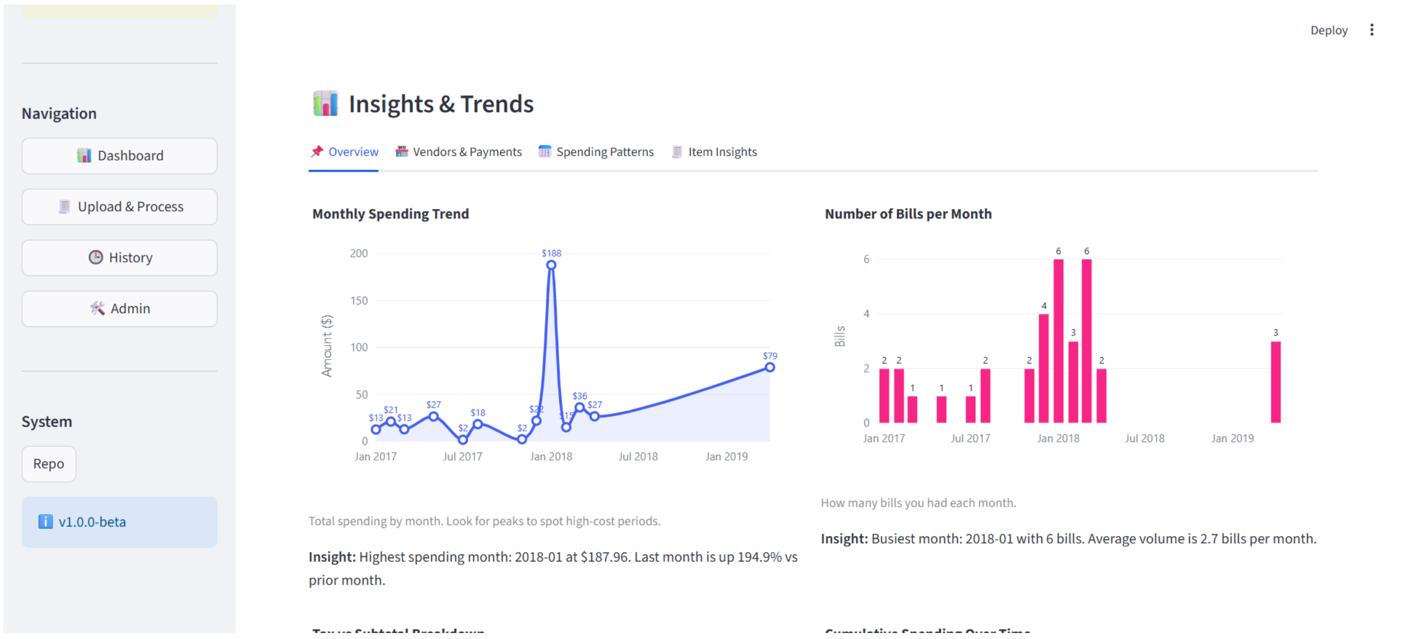
Showing all **35** bills | Total: **$463.56**

📊 Insights & Trends

📌 Overview | Vendors & Payments | Spending Patterns | Item Insights

Monthly Spending Trend

Number of Bills per Month



Navigation

Dashboard

Upload & Process

History

Admin

System

Repo

v1.0.0-beta

Deploy

📊 Insights & Trends

📌 Overview | Vendors & Payments | Spending Patterns | Item Insights

Monthly Spending Trend

Total spending by month. Look for peaks to spot high-cost periods.

Insight: Highest spending month: 2018-01 at $187.96. Last month is up 194.9% vs prior month.

Number of Bills per Month

How many bills you had each month.

Insight: Busiest month: 2018-01 with 6 bills. Average volume is 2.7 bills per month.

Tax vs Subtotal Breakdown

Cumulative Spending Over Time

12. Error Handling & Validation

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

12.1 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

12.2 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

12.3 OCR & AI Extraction Handling

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

12.4 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

12.5 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

12.6 Currency Conversion Validation

- Converts non-USD currencies safely

- Preserves original currency and exchange rate
- Skips conversion if unsupported without failing

12.7 Amount Validation

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

12.8 Duplicate Detection

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

12.9 Database Safety

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

12.10 User Feedback

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

12.11 Dashboard Data Retrieval and Analytics Validation

- Validates database query results before performing analytics computations
- Handles empty datasets gracefully without causing runtime failures
- Prevents division-by-zero and invalid aggregation errors
- Automatically handles missing, null, or incomplete values in analytics processing
- Ensures safe dataframe transformations and aggregation using Pandas

12.12 Visualization Layer Safety

- Validates required fields before generating charts
- Prevents chart rendering when insufficient or invalid data is present
- Displays empty chart placeholders instead of crashing the dashboard
- Ensures safe rendering of interactive Plotly visualizations
- Handles chart generation errors without affecting other dashboard components

12.13 AI Insight Generation Validation

- Validates summary data before sending requests to Gemini AI
- Prevents AI calls when insufficient analytics data is available
- Verifies API key presence before initiating AI insight generation
- Handles API failures, timeouts, or malformed responses gracefully
- Prevents duplicate or redundant AI calls using summary hash comparison
- Displays structured error messages without interrupting dashboard functionality

12.14 Export and Reporting Safety

- Validates data availability before generating export files
- Prevents exporting empty or invalid datasets
- Ensures safe conversion of data into CSV, Excel, and PDF formats
- Uses memory-safe file generation through in-memory streams
- Handles export failures without affecting dashboard operation

12.15 Session State and Cache Validation

- Ensures safe initialization and access of Streamlit session state variables
- Prevents invalid or stale session state data from affecting application behavior
- Automatically refreshes cached data when database changes occur
- Prevents redundant database queries through controlled caching
- Maintains consistency between dashboard display and stored data

12.16 Template Parsing Validation and Safety

- Validates vendor template existence before attempting template-based extraction
- Ensures templates are loaded safely and ignores malformed or missing template files
- Applies template extraction only when vendor match confidence is sufficient
- Prevents overwriting valid AI-extracted fields during template merging
- Safely handles regex failures and invalid template patterns without interrupting extraction

12.17 Template Data Integrity and Field Validation

- Ensures extracted template fields conform to expected data types and schema
- Validates numeric fields such as subtotal, tax, and total before merging
- Handles missing or incomplete template fields gracefully
- Prevents propagation of invalid template extraction results into normalization and storage
- Ensures line item extraction produces structured and consistent output

12.18 Search and Filter Query Validation

- Validates filter parameters before executing database queries
- Prevents invalid date ranges, null values, or malformed filter inputs
- Ensures SQL queries execute safely using controlled query construction
- Prevents empty or invalid query results from affecting dashboard stability
- Maintains consistent filtered results across dashboard analytics and reports

12.19 Database Query Optimization and Safety

- Ensures optimized SQL queries execute safely without affecting database integrity
- Prevents excessive memory usage by retrieving only required records
- Uses safe query execution to prevent database lock or corruption
- Handles database query failures gracefully without crashing the dashboard
- Ensures consistent and reliable data retrieval for analytics and reporting

12.20 Dashboard Filter and Analytics Consistency Validation

- Ensures filtered datasets remain consistent across charts, metrics, and insights
- Prevents analytics computation on empty or invalid datasets
- Safely handles missing or incomplete records during analytics processing
- Ensures cached data remains synchronized with database updates
- Prevents stale or inconsistent dashboard state after filter changes

13. Project Work Timeline

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**
Day 25	**23 Jan 2026**	**Milestone 3 initialization, dashboard module folder structure setup**
Day 26	**24 Jan 2026**	**Implementation of analytics.py for KPI computation and financial aggregation**
Day 27	**25 Jan 2026**	**Implementation of monthly spending, transaction trends, and vendor analytics functions**
Day 28	**26 Jan 2026**	**Implementation of charts.py for Plotly-based visualization (line, bar, pie charts)**
Day 29	**27 Jan 2026**	**Integration of charts into Streamlit dashboard interface**
Day 30	**28 Jan 2026**	**Implementation of dashboard_page.py main dashboard layout and UI structure**

Day 31	**29 Jan 2026**	**Implementation of dashboard filters (date range, payment method filtering)**
Day 32	**30 Jan 2026**	**KPI cards implementation (total spend, transactions, vendor count, averages)**
Day 33	**31 Jan 2026**	**Implementation of vendor analysis charts and payment distribution charts**
Day 34	**01 Feb 2026**	**Implementation of item-level analytics and item spend aggregation**
Day 35	**02 Feb 2026**	**Implementation of insights.py for automated statistical insight generation**
Day 36	**03 Feb 2026**	**Integration of automated insight text into dashboard UI**
Day 37	**04 Feb 2026**	**Implementation of ai_insights.py and Gemini integration for AI-generated financial summaries**
Day 38	**05 Feb 2026**	**AI insight prompt engineering, structured output enforcement, and testing**
Day 39	**06 Feb 2026**	**Implementation of caching mechanisms for database queries and analytics performance optimization**
Day 40	**07 Feb 2026**	**Implementation of export functionality (CSV, Excel export using Pandas and XLSXWriter)**
Day 41	**08 Feb 2026**	**Implementation of PDF export using ReportLab**
Day 42	**09 Feb 2026**	**Dashboard performance optimization, bug fixes, and error handling improvements**
Day 43	**10 Feb 2026**	**UI/UX refinement, dashboard layout improvements, and responsiveness enhancements**
Day 44	**11 Feb 2026**	**Milestone 3 documentation drafting, architecture updates, and module documentation**
Day 45	**12 Feb 2026**	**Final documentation completion, testing, validation, and Milestone 3 closure**
Day 46	**13 Feb 2026**	**Milestone 4 initialization, template parsing module design and template structure planning**
Day 47	**14 Feb 2026**	**Implementation of template_parser.py, vendor template loading, alias normalization, and field extraction logic**
Day 48	**15 Feb 2026**	**Integration of template parser into extraction pipeline, safe field merging, and template validation testing**
Day 49	**16 Feb 2026**	**Dashboard search/filter optimization, SQL query performance improvements, and Milestone 4 documentation updates**

14. Conclusion

The Receipt & Invoice Digitizer is a production-ready financial document intelligence system that enables accurate extraction, validation, storage, and analysis of receipt and invoice data. It combines AI-based OCR with deterministic extraction methods, spaCy-based entity recognition, and template-based parsing to achieve high-accuracy, vendor-specific data extraction.

Structured normalization, currency conversion, validation, and duplicate detection ensure data integrity before persistence in SQLite. The system provides an interactive analytics dashboard with optimized querying, dynamic visualizations, AI-generated insights, and export capabilities in CSV, Excel, and PDF formats.

The modular architecture ensures scalability, maintainability, and performance, enabling reliable transformation of unstructured financial documents into structured, searchable, and analytics-ready data.

15. References

[1] A. K. Kar et al., "Intelligent invoice processing: A machine learning approach," *Expert Systems with Applications*, vol. 98, pp. 317–329, May 2018.

[2] X. Zhou et al., "Financial document automation using OCR and NLP techniques," *IEEE Access*, vol. 8, pp. 156933–156947, 2020.

[3] M. Ahmad et al., "Automated reporting tools for business intelligence systems," *Journal of Business Analytics*, vol. 2, no. 1, pp. 45–58, 2019.

[4] Streamlit, "Streamlit documentation," 2024. [Online]. Available: https://docs.streamlit.io/

[5] Plotly, "Plotly Python documentation," 2024. [Online]. Available: https://plotly.com/python/

[6] W. McKinney, "Data structures for statistical computing in Python," in *Proc. 9th Python in Science Conference*, 2010, pp. 51–56.

[7] SQLite, "SQLite documentation," 2024. [Online]. Available: https://www.sqlite.org/docs.html

[8] Python Software Foundation, "Python documentation," 2024. [Online]. Available: https://docs.python.org/3/

[9] ReportLab, "ReportLab documentation," 2024. [Online]. Available: https://www.reportlab.com/documentation/