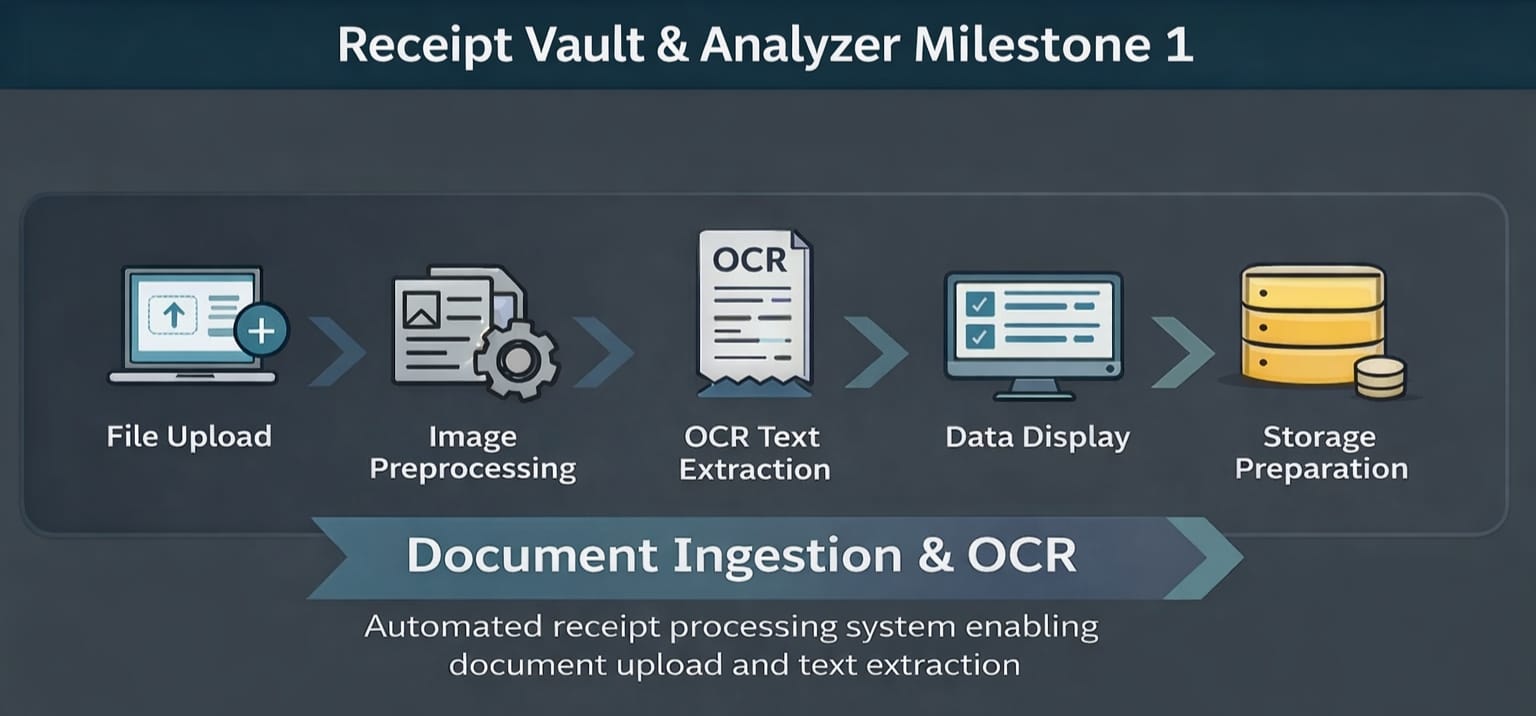
Title

# Receipt and Invoice Digitizer



1. Problem Statement

Managing paper-based receipts and invoices is inefficient and error-prone. Manual data entry consumes time, increases the risk of mistakes, and makes searching or analyzing expenses difficult. There is a need for an automated system that can scan receipts, extract text accurately, and convert physical documents into digital records for further processing and analysis.

2 .Project Overview

The Receipt and Invoice Digitizer is a web-based system designed to automate the process of converting paper receipts and invoices into digital data. The system allows users to upload receipt images or PDF documents and automatically extracts readable text using Optical Character Recognition (OCR). This reduces manual effort, minimizes errors, and enables easy storage and analysis of financial documents.

In Milestone 1, the project focuses on building the core foundation by implementing document ingestion, image preprocessing, and raw text extraction. This milestone establishes a reliable pipeline that prepares the system for advanced AI-based data extraction and analytics in future stages.

3. Architecture



Architecture Explanation – Milestone 1

The system architecture is divided into four main layers, each responsible for a specific task in the receipt digitization process.

1. User Interaction Layer

* The user interacts with the system through a web-based interface.
* Users upload receipt images or PDF files.
* (Optional) Users can enter a Gemini API key to enable AI-based features.

2. Backend Processing & Upload Layer

* The uploaded file first goes through file validation to check format and size.
* Valid files are sent to the OCR engine for text extraction.
* Image preprocessing is applied to improve OCR accuracy.
* OCR extracts raw text from the receipt.
* If Gemini AI is enabled, the text is further processed for:
  + Merchant name
  + Date
  + Total amount
  + Basic spending insights

3. Data Storage Layer

* Extracted and processed receipt data is stored in an SQLite database.
* Data is saved in a structured format for easy retrieval.
* This layer ensures data persistence and consistency.

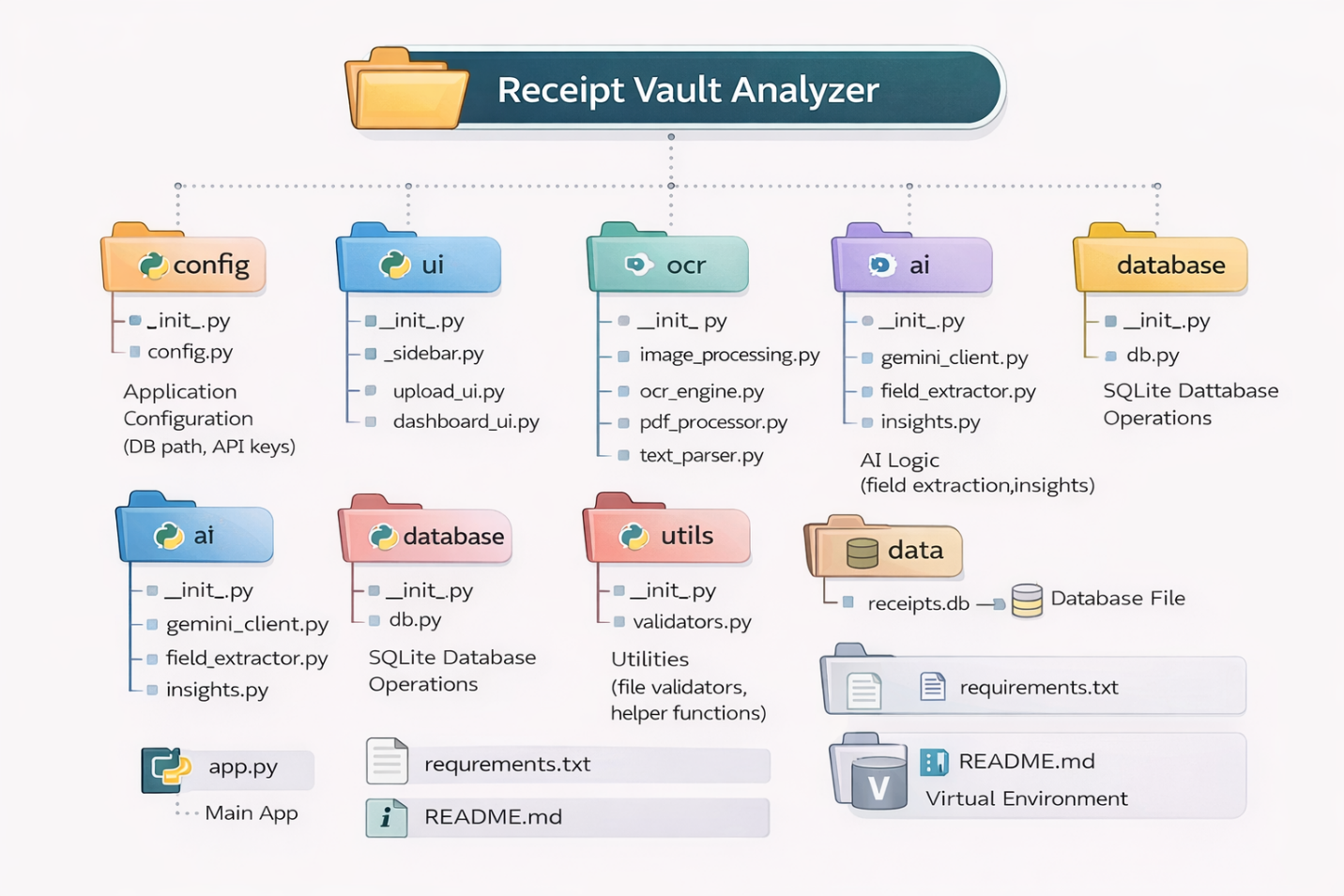
4. Analytics Dashboard Layer

* The dashboard fetches stored receipt data from the database.
* Users can view, search, and analyze receipt information.
* Displays visual insights such as:
  + Spending summaries
  + Receipt details
  + Basic analytics

Overall Data Flow (One Line)

User uploads receipt → Backend validates & extracts text → Data is stored → Dashboard displays insights.

Project Directory Structure



**4. File and Folder Description**

The project follows a modular and well-structured directory design, where each folder is responsible for a specific functionality. This improves readability, scalability, and maintainability of the system.

Root Directory

* app.py  
  Main entry point of the application.  
  It initializes the system, loads configurations, and connects UI components with backend logic.
* requirements.txt  
  Contains the list of Python libraries required to run the project.
* README.md  
  Provides an overview of the project, setup instructions, and usage details.
* venv/  
  Virtual environment containing project dependencies.

config/

* config.py  
  Stores application-level configurations such as database path and API keys.
* \_\_init\_\_.py  
  Marks the folder as a Python package.

ui/

* sidebar.py  
  Manages the sidebar user interface, including Gemini API key input and navigation.
* upload\_ui.py  
  Handles receipt upload, image display, OCR triggering, and extracted data visualization.
* dashboard\_ui.py  
  Displays analytics such as charts, tables, and receipt summaries.
* \_\_init\_\_.py  
  Initializes the UI module.

ocr/

* image\_processing.py  
  Performs image preprocessing such as grayscale conversion and noise reduction.
* ocr\_engine.py  
  Extracts raw text from images using OCR.
* pdf\_processor.py  
  Converts uploaded PDF files into images for OCR processing.
* text\_parser.py  
  Handles and cleans OCR-extracted raw text.
* \_\_init\_\_.py  
  Initializes the OCR module.

ai/

* gemini\_client.py  
  Handles interaction with the Google Gemini API.
* field\_extractor.py  
  Extracts structured fields such as merchant, date, and total using AI.
* insights.py  
  Generates basic insights from extracted receipt data.
* \_\_init\_\_.py  
  Initializes the AI module.

database/

* db.py  
  Manages SQLite database operations such as table creation and data insertion.
* \_\_init\_\_.py  
  Initializes the database module.

utils/

* validators.py  
  Validates uploaded files (file type, size, format).
* helpers.py  
  Contains helper functions used across the project.
* \_\_init\_\_.py  
  Initializes utility functions.

data/

* receipts.db  
  SQLite database file that stores structured receipt data.

**5. Error Handling**

The system incorporates effective error-handling mechanisms to ensure stability, reliability, and smooth user experience during receipt processing.

5.1 File Upload Errors

* Validates uploaded files to ensure only supported formats (PNG, JPG, JPEG, PDF) are accepted.
* Displays user-friendly error messages for:
  + Unsupported file formats
  + Empty or corrupted files
  + File size exceeding allowed limits

5.2 OCR Processing Errors

* Handles poor image quality or unreadable text gracefully.
* Prevents application crashes when OCR fails to extract text.
* Displays informative messages when no readable text is detected.

5.3 PDF Processing Errors

* Catches errors during PDF-to-image conversion.
* Handles missing or unsupported PDF dependencies.
* Ensures the system does not stop execution if a PDF page fails to load.

5.4 AI Integration Errors (Gemini API)

* Validates the presence of the Gemini API key before enabling AI features.
* Displays warnings if:
  + API key is missing
  + API request fails
  + AI service is unavailable
* Allows OCR-only processing when AI features are disabled.

5.5 Database Errors

* Handles database connection issues gracefully.
* Prevents duplicate or invalid data insertion.
* Ensures safe storage of extracted receipt data.

5.6 User Interface Error Handling

* Uses alerts, warnings, and info messages to guide users.
* Prevents abrupt application termination during runtime errors.
* Maintains session state to avoid data loss during reruns.

**6. Performance and Optimization**

The system is designed to deliver efficient performance while processing receipt documents and extracting text. Various optimization techniques are applied to ensure smooth execution and minimal delay.

6.1 Optimized Image Processing

* Image preprocessing is applied only when required.
* Grayscale conversion and noise reduction reduce OCR computation time.
* Avoids unnecessary image transformations to improve speed.

6.2 Efficient OCR Execution

* OCR is performed only on validated and preprocessed images.
* Supports single-image and multi-page PDF processing efficiently.
* Reduces redundant OCR calls by processing each file only once.

6.3 Controlled AI Usage

* AI-based extraction is executed only when a valid Gemini API key is provided.
* Prevents unnecessary API calls, reducing latency and cost.
* Allows OCR-only processing for faster execution when AI is disabled.

6.4 Database Optimization

* Uses lightweight SQLite for fast local data storage.
* Inserts data in a structured format to enable quick retrieval.
* Avoids repeated database queries during a single session.

6.5 UI Performance Optimization

* Uses Streamlit session state to prevent repeated reprocessing.
* Displays results only after successful completion of OCR.
* Maintains responsiveness during file uploads and processing.

6.6 Scalability Considerations

* Modular architecture allows easy optimization in future milestones.
* Can be extended to use cloud storage and distributed databases.
* Supports future performance improvements such as batch processing.

**7. Testing and Validation**

Testing and validation were performed to ensure that the system functions correctly, produces accurate results, and handles errors gracefully during receipt processing.

7.1 Functional Testing

* Verified successful upload of receipt images and PDF files.
* Tested support for multiple file formats (PNG, JPG, JPEG, PDF).
* Confirmed that only valid files are accepted by the system.
* Ensured proper navigation between upload and dashboard sections.

7.2 OCR Accuracy Testing

* Tested OCR extraction on different types of receipts.
* Verified text extraction from both clear and noisy images.
* Compared extracted text with original receipt content for accuracy.
* Validated OCR output visibility for user confirmation.

7.3 PDF Processing Validation

* Tested conversion of single-page and multi-page PDFs.
* Verified correct image extraction from PDF documents.
* Ensured OCR operates correctly on converted PDF images.

7.4 AI Feature Validation (Optional)

* Tested AI-based field extraction using a valid Gemini API key.
* Verified correct extraction of merchant name, date, and total amount.
* Ensured the system works correctly when AI features are disabled.

7.5 Database Validation

* Verified successful storage of extracted receipt data.
* Checked data consistency after insertion into SQLite database.
* Ensured no duplicate or corrupted records are stored.

7.6 Error Handling Validation

* Tested system behavior with invalid files and corrupted documents.
* Verified appropriate error messages are displayed.
* Ensured the application remains stable during failures.

7.7 User Interface Validation

* Confirmed correct display of original and processed images.
* Validated extracted text and tables are displayed properly.
* Ensured smooth user experience without application crashes.

**Future Enhancements**

The system can be extended beyond Milestone 1 to build a complete, intelligent receipt and invoice management solution. The following enhancements are planned across future milestones.

1. Advanced Field Extraction (Milestone 2)

* Apply regex and NLP techniques to extract key fields such as:
  + Vendor name
  + Date
  + Total amount
* Improve extraction accuracy for complex and noisy receipts.
* Validate extracted totals to detect mismatches or errors.
* Detect duplicate receipts based on vendor, date, and amount.
* Store fully structured receipt data in the database.

2. Intelligent Dashboard & Reporting (Milestone 3)

* Develop a comprehensive Streamlit dashboard for receipt review.
* Enable CSV and Excel export for accounting and reporting.
* Display monthly and category-wise expense summaries.
* Provide simple visual analytics for spending trends.
* Improve user experience for managing large volumes of receipts.

3. Search, Filtering & Optimization (Milestone 4)

* Add advanced search and filter options in the dashboard.
* Allow filtering by date range, vendor, and amount.
* Optimize database queries to improve performance.
* Enhance report generation speed and accuracy.
* Improve UI responsiveness and usability.

4. Accuracy & Performance Improvements

* Introduce template-based parsing for commonly used receipt formats.
* Improve OCR accuracy for low-quality images.
* Optimize validation logic to meet evaluation accuracy benchmarks.
* Ensure consistent performance across different receipt types.

5. Integration & Scalability

* Enable integration with accounting and ERP systems.
* Support cloud-based storage for scalability.
* Prepare the system for multi-user access.
* Extend the platform for enterprise-level usage.

**Milestone 2: Field Extraction & Validation (Weeks 3–4)**

**Implementation**

**1. Input Handling**

* Accept invoice images or PDF files from the user interface.
* Validate file type and size before processing.
* Convert PDF pages into images if required.

**2. OCR Processing**

* Apply OCR engine to extract raw text from invoice images.
* Preprocess images (resize, grayscale, noise removal) to improve OCR accuracy.
* Store extracted text temporarily for further processing.

**3. Text Parsing using Regex & NLP**

* Use **regular expressions (Regex)** to extract structured fields such as:
  + Invoice number
  + Date
  + Vendor name
  + Subtotal, tax, and total amount
* Apply **NLP techniques** to identify line items and context-based fields.
* Normalize extracted values (date formats, currency, numbers).

**4. Key Field Extraction**

* Extract mandatory invoice fields:
  + Vendor
  + Invoice date
  + Invoice ID
  + Line items
  + Subtotal, tax, and total
* Organize extracted data into a structured format (JSON / dictionary).

**5. Field Validation**

* Validate presence of all required fields.
* Verify subtotal, tax, and total using business rules.
* Check date format and numeric value consistency.
* Flag missing or incorrect fields.

**6. Duplicate Detection**

* Check database for existing invoices using:
  + Invoice number
  + Vendor name
  + Invoice date
  + Total amount
* Identify and mark duplicate invoices.
* Prevent duplicate records from being stored multiple times.

**7. Error Handling & Logging**

* Log validation errors and duplicate detection results.
* Maintain validation logs for debugging and audit purposes.
* Generate warning messages for failed validations.

**8. Database Storage**

* Store validated invoice data in the database.
* Save duplicate invoices with appropriate status flags.
* Index important fields for faster retrieval.

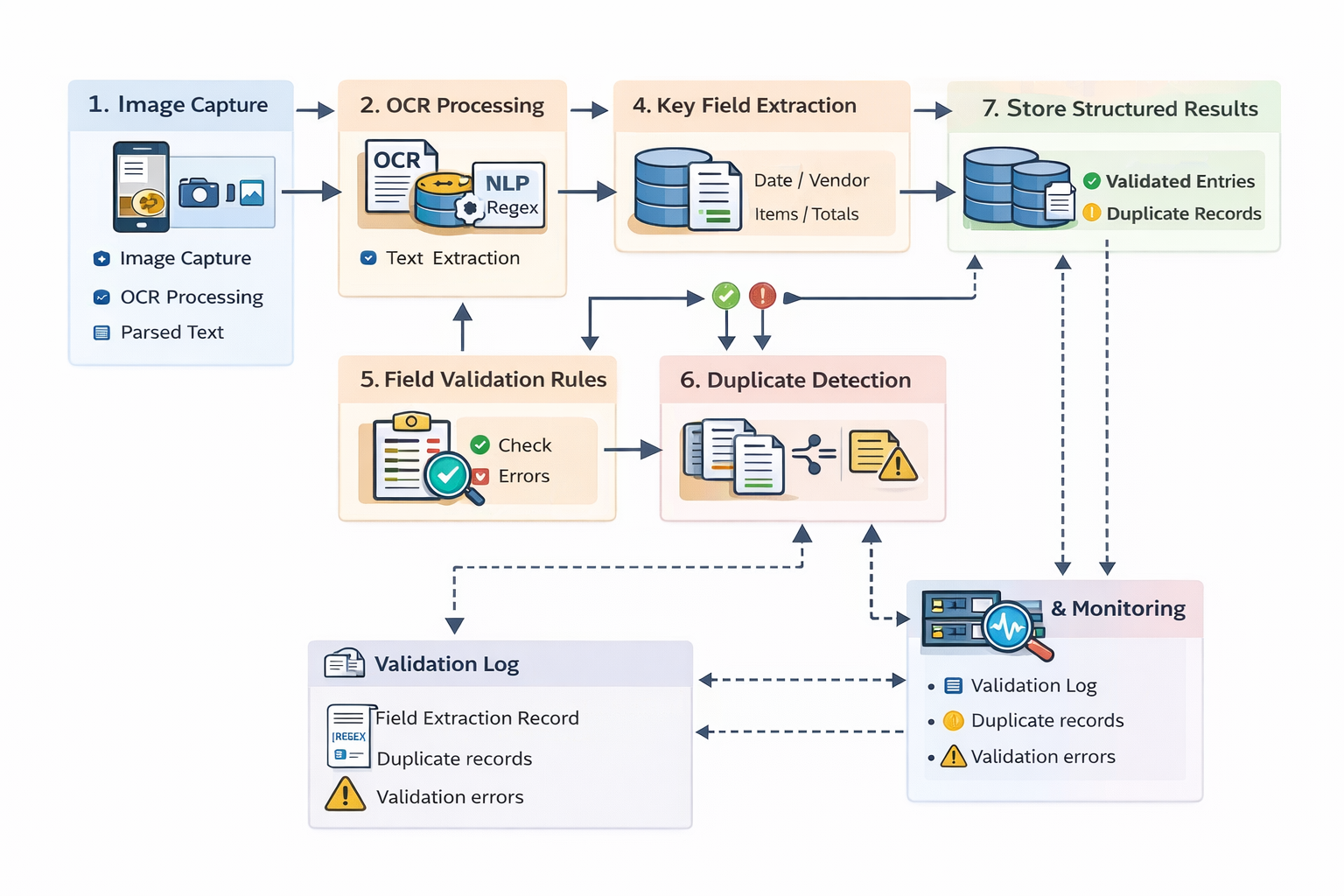
**9. Output Generation**

* Return structured invoice data to the frontend.
* Display validation status (success, warning, duplicate).
* Provide error messages when validation fails.

**10. Monitoring & Maintenance**

* Track OCR accuracy and validation success rate.
* Monitor duplicate detection efficiency.
* Maintain logs for continuous system improvement.

**Architecture : Milestone 2 – Field Extraction & Validation**

****

This architecture describes how invoice data is extracted, validated, and stored after OCR processing.

**1. Image Capture**

* The process starts when a user uploads or captures an invoice image.
* Supported inputs include scanned images or photos taken from a mobile device.
* The image is passed to the OCR module for text extraction.

**2. OCR Processing**

* The OCR engine converts the invoice image into raw text.
* Image preprocessing (such as noise removal and alignment) improves text accuracy.
* The output of this stage is parsed, machine-readable text.

**3. NLP & Regex Parsing**

* Natural Language Processing (NLP) and Regex rules are applied to the OCR text.
* These techniques identify meaningful invoice data from unstructured text.
* This step prepares the text for structured field extraction.

**4. Key Field Extraction**

* Important invoice fields are extracted, such as:
  + Invoice date
  + Vendor name
  + Line items
  + Subtotal, tax, and total amount
* The extracted information is organized into a structured format.

**5. Field Validation Rules**

* Business validation rules are applied to verify data correctness.
* Checks include:
  + Required field presence
  + Total = Subtotal + Tax
  + Valid date and numeric formats
* Errors are flagged if validation rules fail.

**6. Duplicate Detection**

* The system checks for duplicate invoices in the database.
* Comparison is done using:
  + Invoice number
  + Vendor name
  + Date
  + Total amount
* Duplicate invoices are identified and marked instead of being re-stored.

**7. Store Structured Results**

* Validated invoice data is saved in the database.
* Duplicate records are stored with a special duplicate flag.
* This ensures clean and structured data storage.

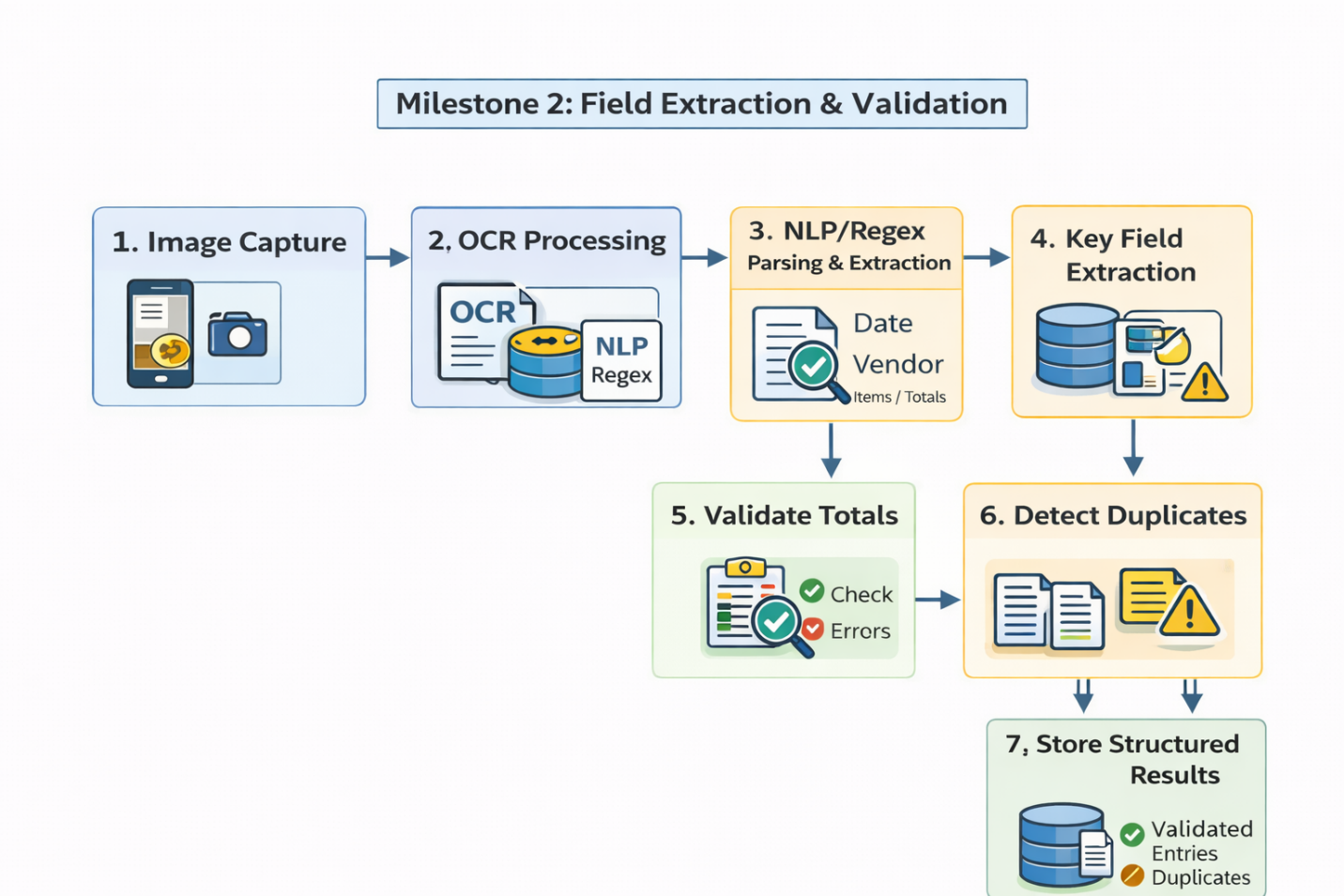
**8. Validation Log**

* All extraction and validation results are logged.
* Logs include:
  + Extracted field records
  + Validation errors
  + Duplicate detection information
* These logs help in debugging and auditing.

**9. Monitoring & Reporting**

* Monitoring services track validation performance and failures.
* Validation logs, duplicate records, and errors are continuously monitored.
* Reports help improve OCR accuracy and validation rules over time.

**Flowchart : Milestone 2 – Field Extraction & Validation**

****

This flowchart shows how invoice data is processed after capturing an image, focusing on extracting and validating important fields.

**1. Image Capture**

* The process starts by capturing or uploading an invoice image.
* The image can be taken using a mobile camera or uploaded as a scanned document.
* This image is passed to the OCR module.

**2. OCR Processing**

* Optical Character Recognition (OCR) converts the invoice image into text.
* The extracted text is unstructured and may contain noise.
* This text is forwarded for parsing and analysis.

**3. NLP / Regex Parsing & Extraction**

* NLP techniques and regular expressions are applied to the OCR text.
* This step identifies meaningful invoice information such as:
  + Date
  + Vendor name
  + Line items
  + Totals
* Parsed data becomes semi-structured.

**4. Key Field Extraction**

* Required invoice fields are extracted from the parsed text.
* Fields include invoice date, vendor, items, subtotal, tax, and total.
* Any missing or suspicious fields are flagged.

**5. Validate Totals**

* Business rules are applied to verify financial correctness.
* Checks include:
  + Subtotal + Tax = Total
  + Valid numeric values
* Errors are detected and marked if validation fails.

**6. Detect Duplicates**

* The system checks whether the invoice already exists in the database.
* Duplicate detection is based on invoice number, vendor, date, and total.
* Duplicate invoices are identified to avoid repeated storage.

**7. Store Structured Results**

* Successfully validated invoices are stored in the database.
* Duplicate records are saved with a duplicate status flag.
* Final output is clean, structured invoice data.

**Error Handling – Milestone 2**

Error handling ensures the system remains reliable when incorrect, missing, or duplicate data is encountered during invoice processing.

**1. Input-Level Error Handling**

* Validate uploaded file type (image/PDF only).
* Check file size limits before processing.
* Handle corrupted or unreadable files gracefully.
* Display user-friendly error messages for invalid inputs.

**2. OCR Errors**

* Detect low-confidence OCR outputs.
* Handle cases where text extraction fails.
* Log unreadable or partially extracted text.
* Retry OCR with preprocessing if accuracy is low.

**3. Parsing Errors (NLP / Regex)**

* Handle missing or unmatched regex patterns.
* Capture partially extracted fields instead of failing completely.
* Mark unrecognized text segments for review.
* Apply fallback rules when NLP confidence is low.

**4. Missing Field Errors**

* Check for mandatory fields (Invoice ID, Date, Vendor, Total).
* Flag invoices with missing required fields.
* Prevent storage of incomplete records.
* Generate validation error messages for missing fields.

**5. Validation Errors**

* Detect incorrect totals (Subtotal + Tax ≠ Total).
* Identify invalid numeric or date formats.
* Flag unrealistic values (negative totals, zero amounts).
* Mark invoices with validation failures for manual review.

**6. Duplicate Detection Errors**

* Handle partial matches safely to avoid false duplicates.
* Allow tolerance thresholds for minor amount differences.
* Mark duplicates instead of rejecting them completely.
* Prevent duplicate records from overwriting original data.

**7. Database Errors**

* Handle database connection failures.
* Use transactions to avoid partial data storage.
* Roll back changes if an insert/update fails.
* Log database exceptions with error details.

**8. Logging & Monitoring**

* Log all errors with timestamps and error types.
* Store OCR, validation, and duplicate errors separately.
* Maintain error severity levels (warning, critical).
* Enable monitoring dashboards for error tracking.

**9. User Feedback**

* Show clear validation status (Success / Warning / Error).
* Provide readable error messages instead of technical logs.
* Allow users to re-upload or correct invoice data.

**10. Recovery & Fallback Mechanisms**

* Retry OCR on failure with enhanced preprocessing.
* Use default values where appropriate.
* Allow manual correction of flagged invoices.
* Continue pipeline execution without system crash.

**Testing and Validation – Milestone 2**

Testing and validation ensure that invoice data is accurately extracted, correctly validated, and safely stored.

**1. Input Testing**

* Test supported file formats (JPG, PNG, PDF).
* Verify file size limits and corrupted file handling.
* Validate multiple image orientations and resolutions.
* Ensure system rejects unsupported or invalid files.

**2. OCR Testing**

* Test OCR accuracy on clear and noisy invoice images.
* Validate text extraction for different fonts and layouts.
* Measure OCR confidence scores.
* Test fallback OCR preprocessing for low-quality images.

**3. Regex & NLP Parsing Testing**

* Test regex patterns against multiple invoice formats.
* Validate NLP extraction for:
  + Vendor name
  + Date
  + Line items
  + Totals
* Test partial extraction scenarios.
* Ensure no system failure when patterns do not match.

**4. Key Field Extraction Validation**

* Verify extraction of mandatory fields.
* Test handling of missing or extra fields.
* Validate normalization of dates and currency formats.
* Compare extracted values with expected outputs.

**5. Business Rule Validation Testing**

* Test subtotal + tax = total rule.
* Validate tax percentage calculations.
* Test edge cases (zero tax, discounts, rounding).
* Ensure incorrect totals are properly flagged.

**6. Duplicate Detection Testing**

* Test duplicate invoices with same invoice number.
* Test partial duplicates (same vendor + date + amount).
* Validate tolerance limits for amount variations.
* Ensure duplicates are flagged, not reinserted.

**7. Database Validation**

* Verify only validated invoices are stored.
* Check duplicate flag storage accuracy.
* Test database indexing and retrieval performance.
* Validate rollback on failed insert operations.

## **Conclusion – Milestone 2: Field Extraction & Validation**

* Milestone 2 successfully implements the core logic for extracting and validating invoice data from unstructured inputs. By integrating OCR, NLP, and regex-based parsing, the system accurately identifies key invoice fields such as vendor details, dates, line items, and totals. Validation rules ensure financial correctness, while duplicate detection prevents redundant data storage.
* Robust error handling, testing, and logging mechanisms improve system reliability and maintain data integrity. The structured storage of validated results enables efficient retrieval and further processing in later milestones. Overall, this milestone forms a strong foundation for building an accurate, scalable, and reliable invoice digitization system.