InvoiceGuard: Automated Validation, Payment & Regional Insights

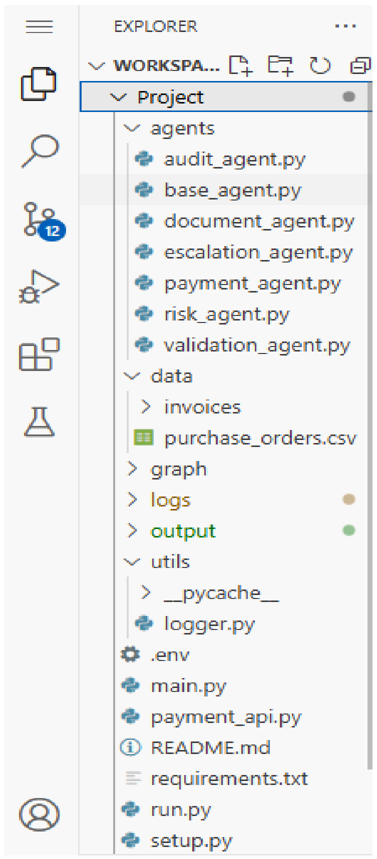
Problem Statement

Enterprises dealing with high invoice volumes often struggle with manual validation, payment tracking, and risk triaging. This leads to payment delays, human error, and missed regional performance trends. Your task is to build a smart automation platform that streamlines the entire invoice lifecycle — from unstructured PDF extraction to AI-powered decisioning and insightful visualization.

Objective

Design and implement a fully automated system that: - Extracts structured invoice data from unstructured PDFs using GenAI. - Validates invoices against purchase orders to detect mismatches (e.g., quantity or overbilling). - Automatically triggers payments for clean invoices due today. - Triages mismatched invoices using GenAI recommendations (Escalate, Hold, Approve). - Visualizes overdue unpaid invoices and region-wise sales/shipping performance. - Generates business-friendly justifications and region summaries using GenAI.

File Structure:



```
EXPLORER ...
∨ WORKSPA...
  ∨ Project
    ∨ agents
        audit_agent.py
        base_agent.py
        document_agent.py
        escalation_agent.py
        payment_agent.py
        risk_agent.py
        validation_agent.py
    ∨ data
      > invoices
        purchase_orders.csv
    > graph
    > logs
    > output
    ∨ utils
      > __pycache__
        logger.py
      .env
      main.py
      payment_api.py
      README.md
      requirements.txt
      run.py
      setup.py
```

Input Files

1) data/invoices/*.pdf: Raw scanned invoices in PDF format.
2) data/purchase_orders.csv: Ground truth reference for validation. Each row in this file corresponds to a previously recorded approved purchase and includes the following fields:

- invoice_number - Unique invoice reference ID, also used in invoice parsing
- order_id - Unique order identifier to be matched against extracted invoice data (Remember it might me same between user. A user will not have a same order_id)
- customer_name - Name of the customer the PO was issued to (Unique)
- item_name - Full item description (for context; not used in validation logic)
- quantity - Quantity approved in the purchase order
- rate - Unit price as per PO
- expected_amount - Computed amount (quantity × rate) to compare against invoice

3) payment_api.py (Do Not Modify) You are given a mock FastAPI-based payment engine defined in payment_api.py. This simulates a real-time payment gateway and responds with a transaction ID, timestamp, and status message.

4) requirements.txt - Necessary Installations.

Modules to be Implemented:

agents/base_agent.py

As part of the automation pipeline, you are required to implement a Base Agent class that provides common functionality and interface for all specialized agents with logging, error handling, and state management.

Class:

```
class BaseAgent(ABC):
    """
    Abstract base class for all invoice processing agents
    Provides common functionality like logging, error handling, and state
management
    """

    async def execute(self, state: InvoiceProcessingState) ->
InvoiceProcessingState:
        """
        Execute the agent's main functionality
        Must be implemented by all concrete agents
        """
        pass
```

Input: state: InvoiceProcessingState containing all processing information Output: Updated InvoiceProcessingState with processing results

agents/document_agent.py

To extract structured fields from unstructured invoice PDFs, you must implement a Document Agent that handles PDF extraction, text processing, and AI-powered invoice parsing.

Class:

```
class DocumentAgent(BaseAgent):
    """
    Agent responsible for document processing and invoice data extraction
    Uses multiple extraction methods and AI parsing for robust data
extraction
    """

    async def execute(self, state: InvoiceProcessingState) ->
InvoiceProcessingState:
        """
        Execute document processing workflow
        """
        pass
```

Input: state: InvoiceProcessingState containing PDF file information Expected Output: Updated state with extracted invoice data, including: - invoice_number, order_id, customer_name - item details with quantity, rate, amount - extraction confidence score

agents/validation_agent.py

As part of the validation pipeline, you are required to implement a Validation Agent that handles purchase order matching, discrepancy detection, and validation scoring.

Class:

```
class ValidationAgent(BaseAgent):
    """
    Agent responsible for validating invoice data against purchase orders
    Performs fuzzy matching, discrepancy detection, and validation scoring
    """

    async def execute(self, state: InvoiceProcessingState) -> 
InvoiceProcessingState:
        """
        Execute validation workflow
        """
        pass
```

Input: state: InvoiceProcessingState containing invoice data Expected Output: Validation results with status and discrepancy information

agents/risk_agent.py

For invoices that need risk assessment, your task is to implement a Risk Agent using GenAI model for fraud detection, compliance checking and risk scoring.

Class:

```
class RiskAgent(BaseAgent):
    """
    Agent responsible for risk assessment, fraud detection, and compliance 
checking
    Uses AI-powered analysis combined with rule-based risk factors
    """

    async def execute(self, state: InvoiceProcessingState) -> 
InvoiceProcessingState:
        """
        Execute risk assessment workflow
        """
        pass
```

Input: state: InvoiceProcessingState containing invoice and validation data Output: Risk assessment with risk level, fraud indicators and recommendation

agents/payment_agent.py

To handle payment processing, you must implement a Payment Agent that handles payment decisions, processing, and transaction management.

Class:

```
class PaymentAgent(BaseAgent):
    """
    Agent responsible for payment processing decisions and execution
    Integrates with payment systems and provides intelligent payment routing
    """

    async def execute(self, state: InvoiceProcessingState) -> 
InvoiceProcessingState:
        """
        Execute payment processing workflow
        """
        pass
```

Input: state: InvoiceProcessingState containing validated and risk-assessed data Output: Payment decision with status, transaction details and approval chain

agents/audit_agent.py

To prepare audit trails for compliance, you must implement an Audit Agent that handles compliance tracking, audit trail generation, and regulatory reporting.

Class:

```
class AuditAgent(BaseAgent):
    """
    Agent responsible for audit trail generation, compliance tracking, and 
reporting
    Ensures all processing steps are properly documented for regulatory 
compliance
    """

    async def execute(self, state: InvoiceProcessingState) -> 
InvoiceProcessingState:
        """
        Execute audit workflow
        """
        pass
```

Input: state: InvoiceProcessingState containing complete processing data Output: Audit records with compliance checks, summaries and reportable events

agents/escalation_agent.py

To handle cases that need human review, you must implement an Escalation Agent that handles human-in-the-loop workflows, escalation routing, and approval management.

Class:

```
class EscalationAgent(BaseAgent):
    """
    Agent responsible for escalation management and human-in-the-loop 
workflows
    Routes issues to appropriate human reviewers and manages approval 
processes
    """

    async def execute(self, state: InvoiceProcessingState) -> 
InvoiceProcessingState:
        """
        Execute escalation workflow
        """
        pass
```

Input: state: InvoiceProcessingState requiring escalation or human review Output: Escalation records with approver assignment and notification

main.py

This is the core controller script of the project, built using Streamlit. Expected to integrate all modular components (PDF extraction, validation, payment logic, GenAI reasoning, and visualization) through a LangGraph-based agent workflow and display the end-to-end functionality of the Invoice Automation System.

Responsibilities of main.py

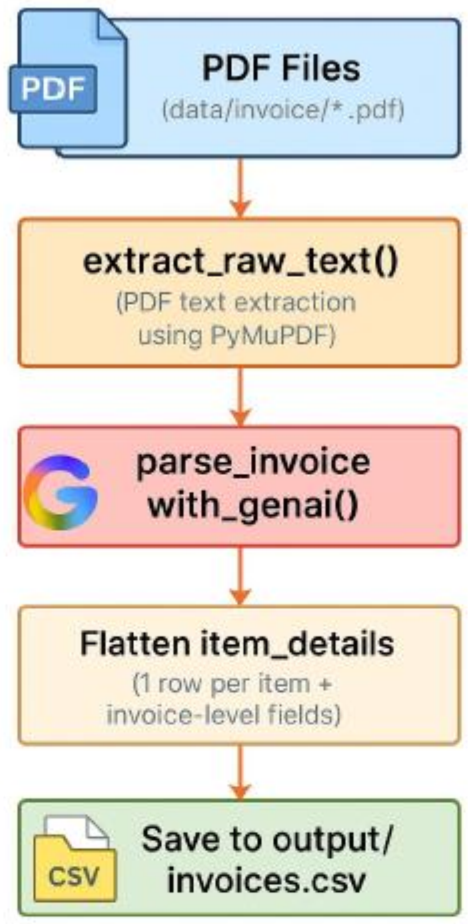
Must implement a modular, tabbed Streamlit dashboard with the following features: - Initialize LangGraph workflow with all AI agents - Process invoices through multi-agent system (Document → Validation → Risk → Payment → Audit → Escalation) - Display processing results and audit trails - Show agent performance metrics - Visualize analytics and insights

Streamlit Tabs to Build

1. **Tab 1: Overview Dashboard**
   - Show processing status distribution
   - Display processing timeline visualization
   - Show success/failure metrics
   - Display invoice processing results with audit trail for each invoice
2. **Tab 2: Invoice Details**
   - Show detailed invoice information

    - Display processing results
    - Show audit trail for each invoice
    - Detailed view of invoice data, validation results, and risk assessment
3. **Tab 3: Agent Performance**
    - Display agent execution metrics
    - Show success rates and duration
    - Performance analytics for all 6 AI agents
4. **Tab 4: Escalations**
    - Show issues requiring human attention
    - Display escalation reasons and details
    - Show approval workflow status
5. **Tab 5: Analytics**
    - Show risk vs amount analysis
    - Display processing efficiency metrics
    - Business intelligence insights
    - Regional performance visualization

Architecture:


PDF
PDF Files
(data/invoice/*.pdf)
extract_raw_text()
(PDF text extraction using PyMuPDF)
parse_invoice with_genai()
Flatten item_details
(1 row per item + invoice-level fields)
CSV
Save to output/ invoices.csv

Multi-Agent System Architecture:

The system implements a sophisticated LangGraph-based architecture with 6 specialized AI agents:

Document Agent: PDF text extraction using multiple methods (PyMuPDF, PDFPlumber), AI-powered invoice parsing with Gemini 2.0 Flash, confidence scoring and data validation

Validation Agent: Purchase order matching with fuzzy logic, discrepancy detection and analysis, three-way matching validation

Risk Agent: AI-powered fraud detection, compliance checking against business rules, risk scoring and recommendation generation

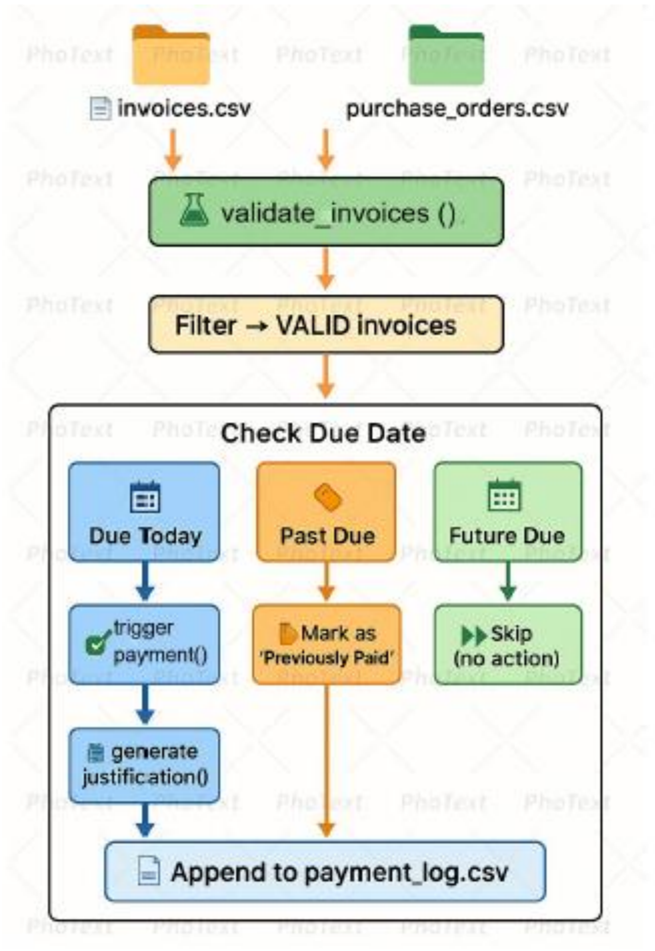
Payment Agent: Intelligent payment routing and method selection, integration with payment APIs, retry logic and error handling

Audit Agent: Comprehensive audit trail generation, compliance reporting (SOX, GDPR, etc.), regulatory documentation

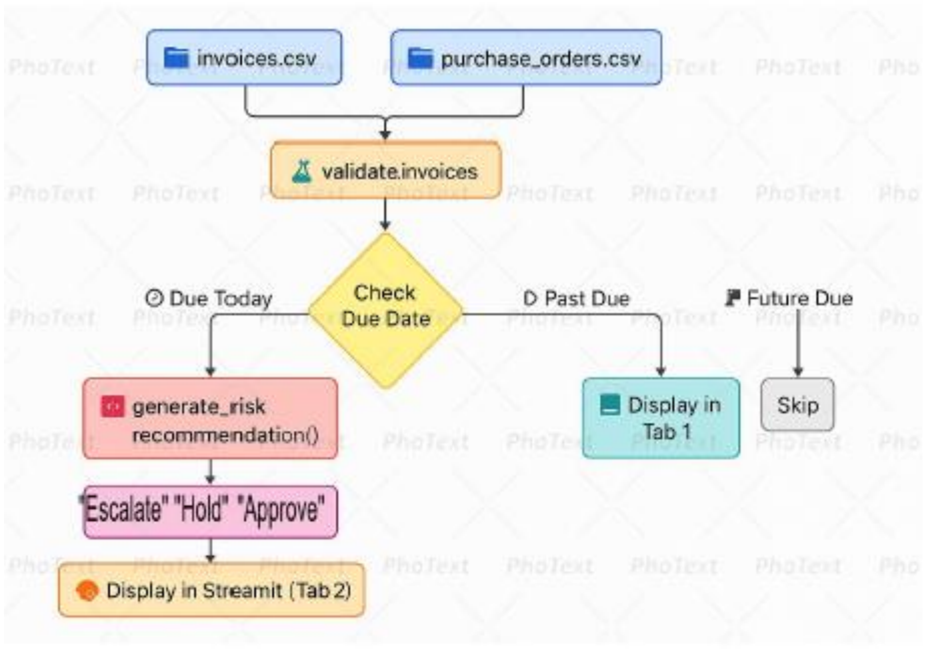
Escalation Agent: Human-in-the-loop workflow management, approval hierarchy routing, SLA monitoring and notifications

LangGraph Orchestration: Intelligent workflow routing and state management with conditional routing based on validation results and risk scores

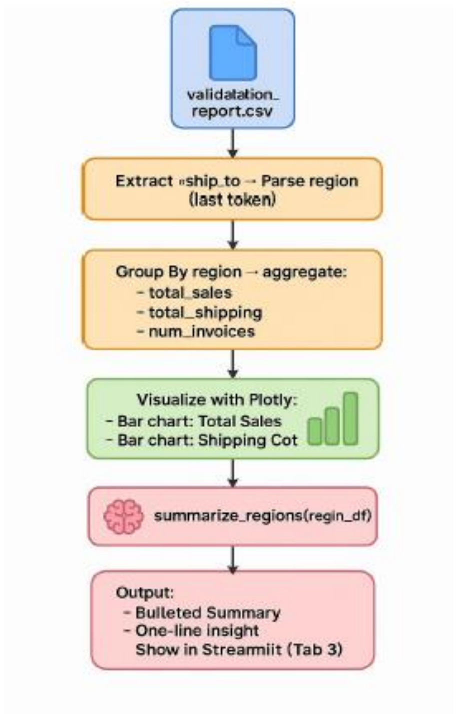
Valid Invoice Data Flow: PDF → Document Agent → Validation Agent → Risk Agent → Payment Agent → Audit Agent → Complete



Invalid Invoice Data Flow: PDF → Document Agent → Validation Agent → Risk Agent → Escalation Agent → Human Review → Approval



Analytics tracking flow: All agents contribute to comprehensive audit trail and metrics collection in a unified state management system


validatation_ report.csv
Extract ship_to → Parse region (last token)
Group By region → aggregate: - total_sales - total_shipping - num_invoices
Visualize with Plotly: - Bar chart: Total Sales - Bar chart: Shipping Cot
summarize_regions(regin_df)
Output: - Bulleted Summary - One-line insight Show in Streamiit (Tab 3)

Commands to Create a Google Gemini API Key

1. Open your web browser.
2. Launch any browser (e.g., Chrome, Firefox) on your computer.
3. Go to Google AI Studio.
4. In the address bar, type aistudio.google.com and press Enter.
5. Sign in to your Google account.
6. Click the “Sign In” button in the top-right corner.
7. Enter your Google email and password, then click “Next” to log in.
8. If you don’t have an account, click “Create Account” and follow the prompts to make one.
9. Navigate to the API Key section.
10. On the Google AI Studio homepage, look at the left sidebar.
11. Click on “Get API Key” (usually near the top-left corner).
12. Create a new API key.
13. In the API Key section, click the “Create API Key” button.
14. A pop-up will appear—select “Create API Key in new project” (or choose an existing project if you have one).
15. Click “Create” to generate the key.
16. Copy the generated API key.
17. Once the key is created, it will appear on the screen.
18. Click the “Copy” button next to the key (or highlight it and press Ctrl+C/Command+C).
19. Save the key in a secure place (e.g., a text file or password manager) because it won’t be shown again.

Implementation Explanation:

Before executing the main.py, enter the Gemini API key in the .env file.

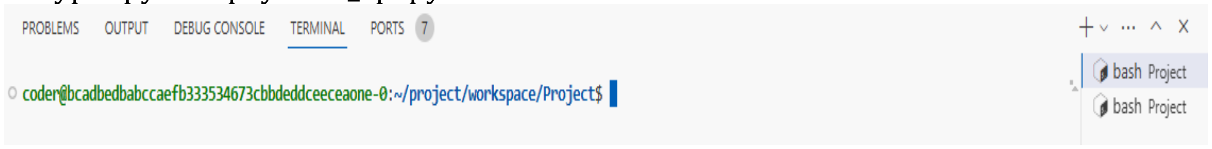
1. Open the main.py integrated terminal.

2. Check the path it in the Project directory, if not use cd command to navigate.

3. To install required packages, run pip install -r requirements.txt in terminal.

4. Use a fresh terminal for starting fastAPI payment_api.py.

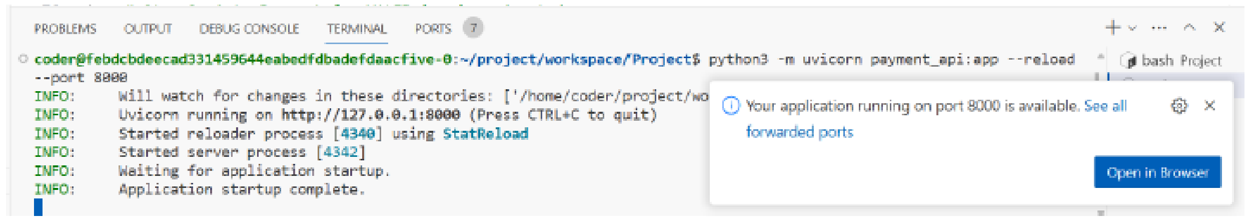
5. Type “python payment_api.py” in terminal to start the API.



```
coder@bcadbedbabccaefb333534673cbbdeddceeceaone-0:~/project/workspace/Project$
```

Type **"python3 -m uvicorn payment_api:app --reload --port 8000"** in terminal to start the API.

Use get as below in the terminal.

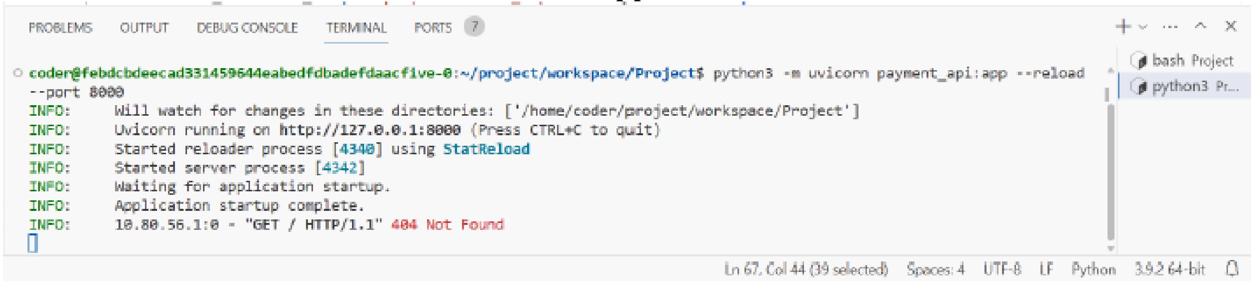


```
coder@febdcbdeecad331459644eabedfdbadefdaacfive-0:~/project/workspace/Project$ python3 -m uvicorn payment_api:app --reload
--port 8000
INFO:     Will watch for changes in these directories: ['/home/coder/project/wo
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [4340] using StatReload
INFO:     Started server process [4342]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Your application running on port 8000 is available. See all forwarded ports

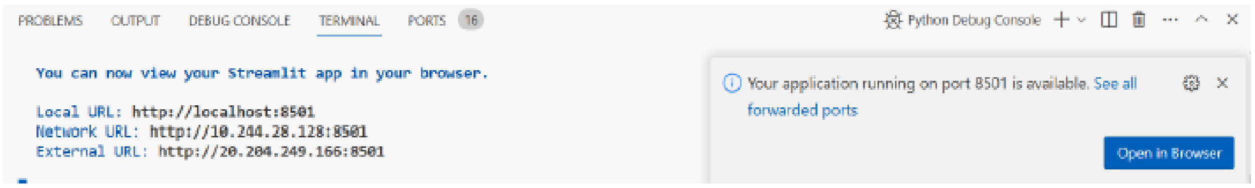
Please Select the open in brower just to start monitoring and a proxy web page will be opened. Just don't bother it, leave it as such or even you can close the proxy window. But if you see in terminal, the server is running and waiting for the response.

6. Use another terminal tab to run the main application.



```
coder@febdcbdeecad331459644eabedfdbadefdaacfive-0:~/project/workspace/Project$ python3 -m uvicorn payment_api:app --reload
--port 8000
INFO:     Will watch for changes in these directories: ['/home/coder/project/workspace/Project']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [4340] using StatReload
INFO:     Started server process [4342]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     10.80.56.1:0 - "GET / HTTP/1.1" 404 Not Found
```

7. Now open another terminal, use “streamlit run main.py” to execute the application, then you will get the pop-up window below, click the assigned port (Open in Browser) which will navigate to streamlit application window.



```
  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://10.244.28.128:8501
  External URL: http://20.204.249.166:8501
```

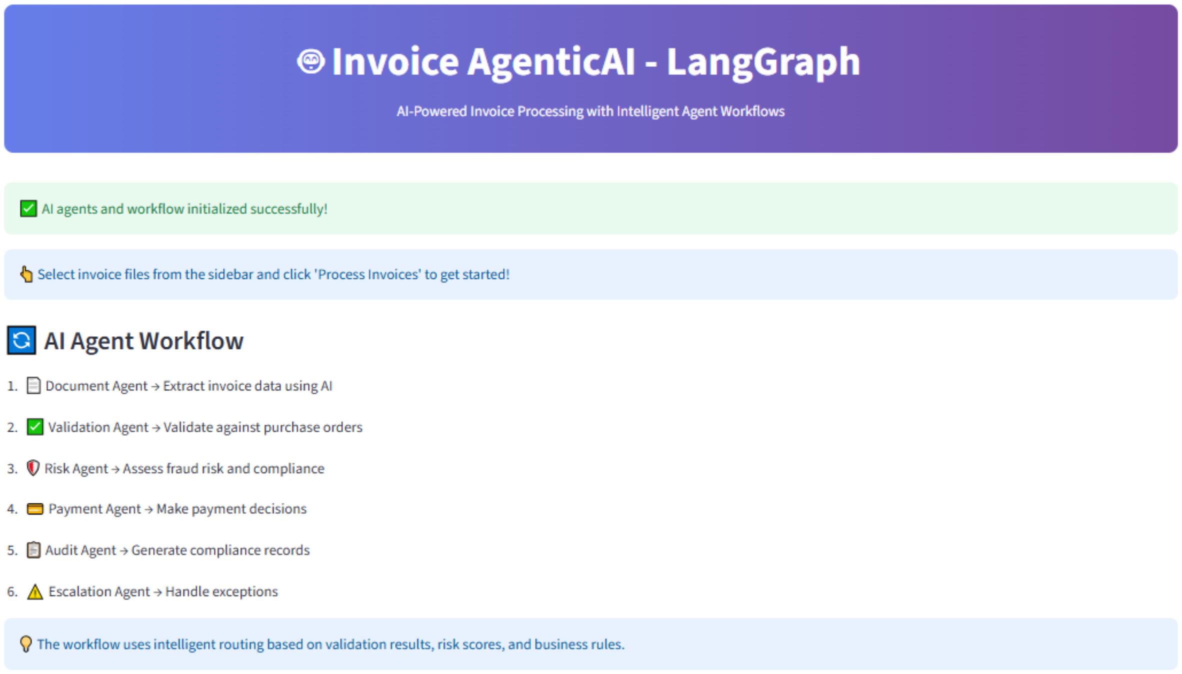
Your application running on port 8501 is available. See all forwarded ports

IMPORTANT: Before you are running testcases make sure your Gemini API free tier is not exhausted.

To check the testcases, you can use python3 -W ignore -m pytest tests.py -v (check the directory it should be Project directory)

Sample Output:

This below shared UI image will be the first page when we start the streamlit. The application processes PDFs through the multi-agent workflow automatically.


🤖 Invoice AgenticAI - LangGraph

AI-Powered Invoice Processing with Intelligent Agent Workflows

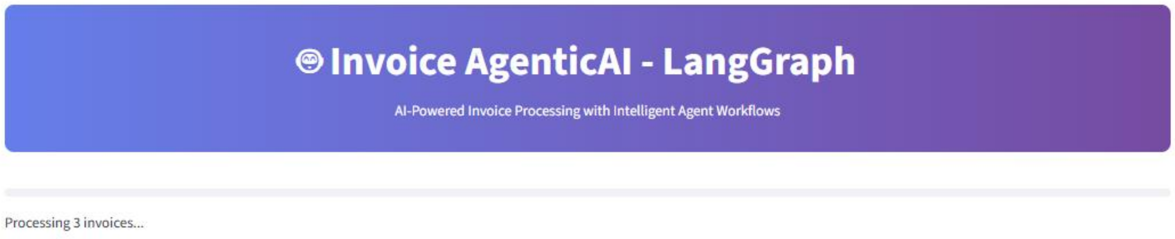
✅ AI agents and workflow initialized successfully!

👆 Select invoice files from the sidebar and click 'Process Invoices' to get started!

🔄 AI Agent Workflow

1. 📄 Document Agent → Extract invoice data using AI
2. ✅ Validation Agent → Validate against purchase orders
3. 🛡️ Risk Agent → Assess fraud risk and compliance
4. 💳 Payment Agent → Make payment decisions
5. 📋 Audit Agent → Generate compliance records
6. ⚠️ Escalation Agent → Handle exceptions

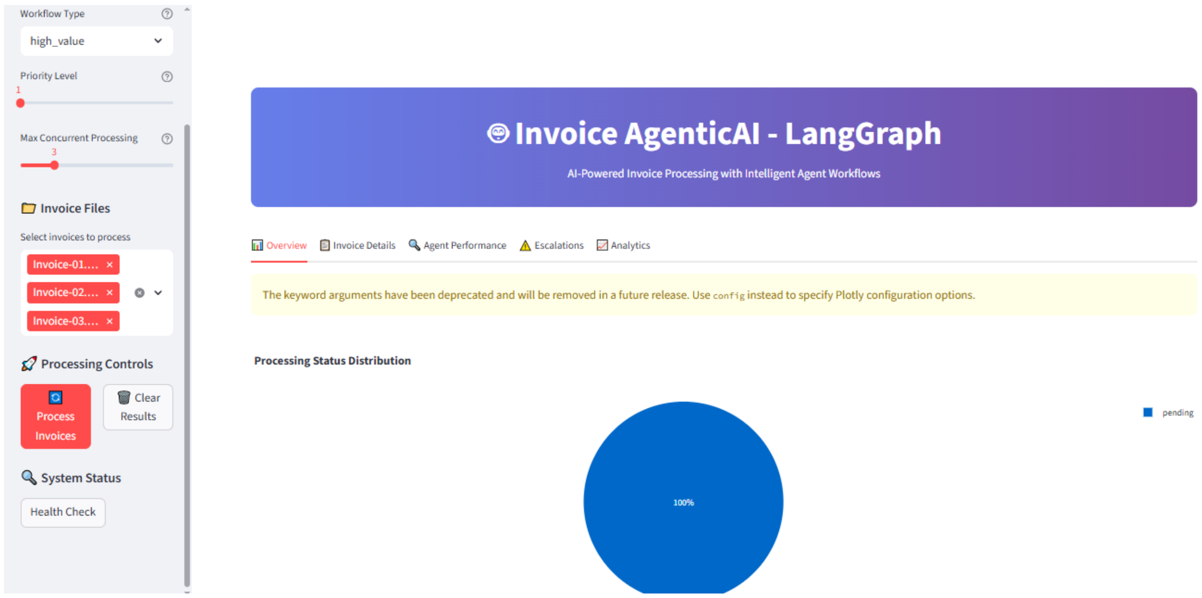
💡 The workflow uses intelligent routing based on validation results, risk scores, and business rules.


🤖 Invoice AgenticAI - LangGraph

AI-Powered Invoice Processing with Intelligent Agent Workflows

Processing 3 invoices...

Display the Summary in Overview Tab showing processing statistics and validation status



Workflow Type

high_value

Priority Level

1

Max Concurrent Processing

3

Invoice Files

Select invoices to process

Invoice-01.... ×

Invoice-02.... ×

Invoice-03.... ×

Processing Controls

Process Invoices

Clear Results

System Status

Health Check

Invoice AgenticAI - LangGraph

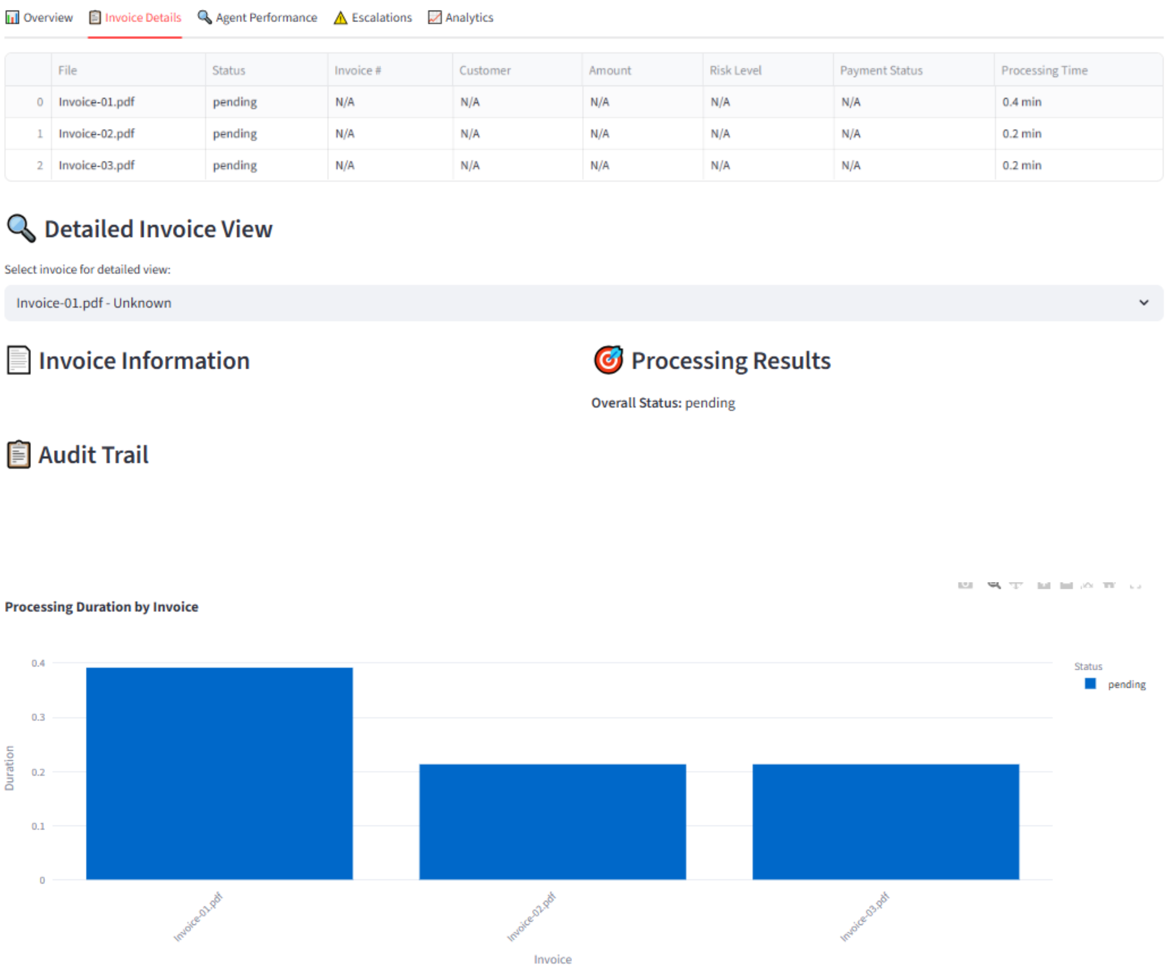
AI-Powered Invoice Processing with Intelligent Agent Workflows

Overview | Invoice Details | Agent Performance | Escalations | Analytics

The keyword arguments have been deprecated and will be removed in a future release. Use `config` instead to specify Plotly configuration options.

Processing Status Distribution

Display the Invoice Details Tab with processing results and audit trails



Overview | Invoice Details | Agent Performance | Escalations | Analytics

	File	Status	Invoice #	Customer	Amount	Risk Level	Payment Status	Processing Time
0	Invoice-01.pdf	pending	N/A	N/A	N/A	N/A	N/A	0.4 min
1	Invoice-02.pdf	pending	N/A	N/A	N/A	N/A	N/A	0.2 min
2	Invoice-03.pdf	pending	N/A	N/A	N/A	N/A	N/A	0.2 min

Detailed Invoice View

Select invoice for detailed view:

Invoice-01.pdf - Unknown

Invoice Information

Processing Results

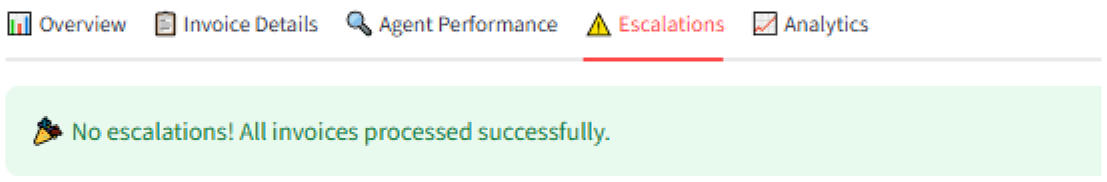
Overall Status: pending

Audit Trail

Processing Duration by Invoice

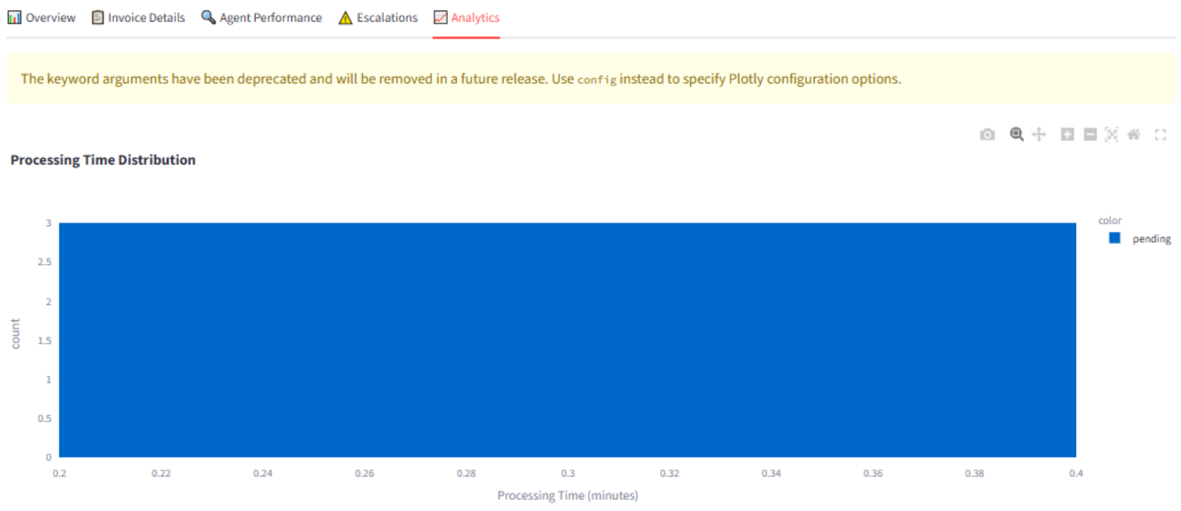
In Tab 3, “Agent Performance” displays metrics for all 6 AI agents in the system.

In Tab 4, “Escalations” shows issues requiring human attention.



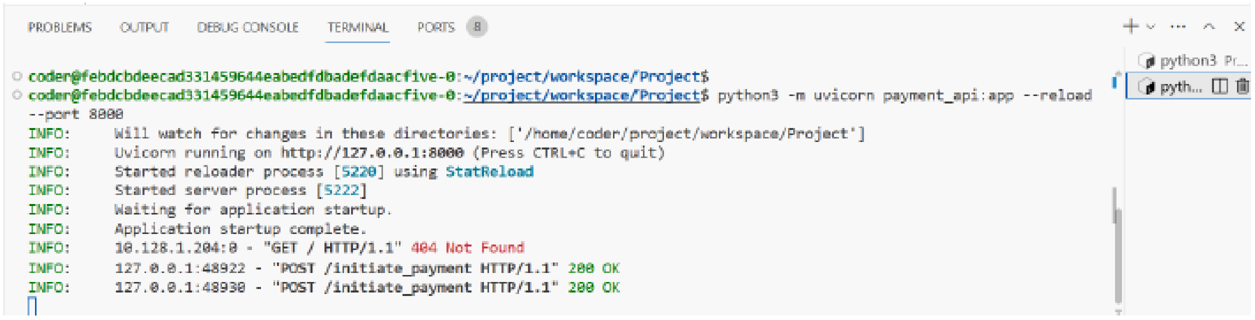
🎉 No escalations! All invoices processed successfully.

Tab 5, “Analytics” displays risk analysis and business intelligence insights.



The keyword arguments have been deprecated and will be removed in a future release. Use `config` instead to specify Plotly configuration options.

And you can verify the transaction in the terminal for fastAPI for payment_api.py as below.



```
coder@febdcbdeecad331459644eabedfdbadefdaacfive-0:~/project/workspace/Project$
coder@febdcbdeecad331459644eabedfdbadefdaacfive-0:~/project/workspace/Project$ python3 -m uvicorn payment_api:app --reload
--port 8000
INFO:     Will watch for changes in these directories: ['/home/coder/project/workspace/Project']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [5220] using StatReload
INFO:     Started server process [5222]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     10.128.1.204:0 - "GET / HTTP/1.1" 404 Not Found
INFO:     127.0.0.1:48922 - "POST /initiate_payment HTTP/1.1" 200 OK
INFO:     127.0.0.1:48930 - "POST /initiate_payment HTTP/1.1" 200 OK
```

Bonus Feature: AI Escalation Alert System with Email Notification (Optional)

Feature Summary:

Integrate an AI-driven escalation alert system that not only flags high-risk invoices but also notifies finance managers instantly via email with a detailed side-by-side comparison of discrepancies.

Description:

Enhance the invoice automation system with a bonus feature that activates when the GenAI triage model recommends “Escalate” for a mismatched invoice. Upon escalation: - The system sends a formal escalation email to the Finance Controller (use your mail-id for demo) - The email includes: - Key invoice details (Order ID, Customer, Amount, Due Date) - AI-generated recommendation and reason for escalation - A clear field-by-field comparison table (Quantity, Rate, Amount) showing mismatches between the invoice and its corresponding purchase order

This empowers finance teams to take timely action based on AI decisions

Sample Email Contents:

Subject: Escalation Alert: Invoice ES-2025-BE11335139-41340 Requires Review

Body: Order ID: ES-2025-BE11335139-41340 Customer: Bill Eplett Amount: $9466.50 Due Date: Aug 06 2025 AI Recommendation: Escalate Reason: Quantity Mismatch, Overbilling

Field Comparison: Invoice Value Vs PO Value

Please review this invoice and take appropriate action.