Documentation

June 6, 2025

Project Name:	Intelligent Invoice Reimbursement System
Version:	1.0
Date:	June 06, 2025
Author:	Sibi Krishnamoorthy

---

1 Executive Summary

This document outlines the project plan for the development of an **Intelligent Invoice Reimbursement System**. The system is designed to automate the process of analyzing employee expense invoices against company policy, storing the results efficiently, and providing an intuitive chatbot interface for querying this information.

The project leverages Large Language Models (LLMs) for intelligent document understanding, vector databases for high-speed similarity search, and a modern API framework to expose its functionality. The primary business drivers are to increase operational efficiency, ensure consistent policy enforcement, and provide stakeholders with rapid, on-demand access to reimbursement data.

The project will deliver two core API endpoints: one for processing and analyzing invoices, and a second for a Retrieval Augmented Generation (RAG) chatbot that allows users to query the processed data using natural language.

---

2 Project Goals & Objectives

The primary goal is to build a robust and intelligent proof-of-concept system that meets all functional and technical requirements of the assignment within the specified 48-hour timeframe.

Key Objectives: Automation: To automate the comparison of PDF invoices against a given PDF HR policy, determining the reimbursement status and providing clear justifications.

Efficient Data Management: To structure, embed, and store the analysis results (invoice content, status, reasons, metadata) in a vector database for scalable and fast retrieval.

Intelligent Retrieval: To develop a conversational AI (chatbot) that can understand user queries, perform semantic and metadata-based searches on the stored data, and generate accurate, context-aware responses.

Technical Demonstration: To showcase proficiency in Python, FastAPI, LLM integration (e.g., LangChain), vector databases (e.g., ChromaDB), and sound software architecture principles.

---

3 Project Scope

3.1 In-Scope

- **Part 1: Invoice Analysis Endpoint:**
  - A FastAPI `POST` endpoint that accepts a policy PDF, a ZIP file of invoice PDFs, and an employee name.
  - PDF parsing for both policy and invoices.
  - LLM-driven analysis to categorize each invoice as **Fully Reimbursed**, **Partially Reimbursed**, or **Declined**, with a detailed reason.
  - Generation of text embeddings for invoice content and analysis results.
  - Storing embeddings and associated metadata (invoice ID, employee name, date, status, reason) in a vector store.
  - Returning a JSON response confirming the success or failure of the processing task.
- **Part 2: RAG Chatbot Endpoint:**
  - A FastAPI `POST` endpoint that accepts a user's natural language query.
  - Maintains conversation history/context for follow-up questions.
  - An LLM agent equipped with a vector search tool.
  - The tool must perform both semantic similarity search and filtering based on metadata (employee name, status, date).
  - Generation of coherent, helpful responses in Markdown format.
- **Documentation & Deliverables:**
  - A comprehensive `README.md` file.
  - Well-commented, modular, and readable source code.
  - A short (under 2 minutes) video demonstration.

3.2 Out-of-Scope

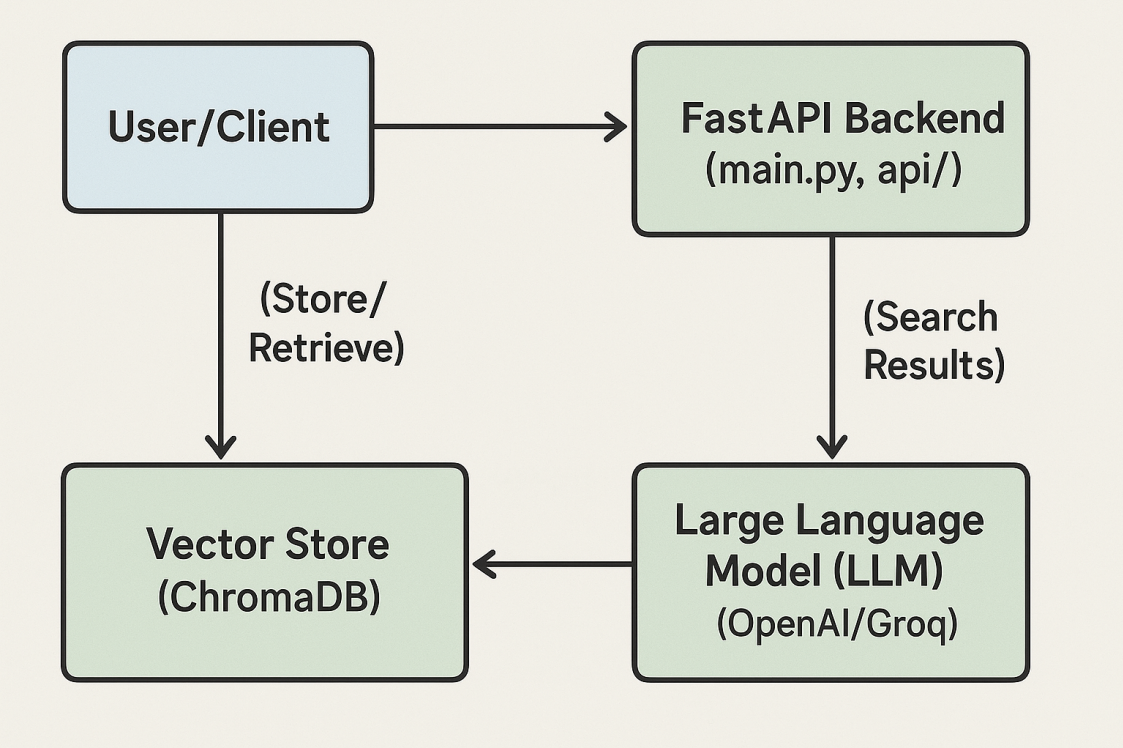
- A production-grade, multi-user authentication and authorization system.
- A graphical user interface (GUI), unless developed as an optional enhancement (e.g., with Streamlit/Gradio).
- Long-term database administration, backup, and scaling strategies.
- Processing of non-PDF or scanned, image-based invoice formats that require OCR.
- Deployment to a cloud production environment (development will be containerized via Docker for portability).

---

4 Technical Architecture & System Design

The system is designed with a modular, service-oriented architecture to ensure separation of concerns and maintainability.

4.1 System Components Diagram



4.2 Technology Stack

Component	Technology Choice	Justification
Programming Language	Python 3.10+	Industry standard for AI/ML, rich ecosystem of libraries.
API Framework	FastAPI	High performance, asynchronous support, automatic interactive documentation (Swagger UI).
LLM Orchestration	LangChain	Simplifies chaining LLM calls, managing prompts, and integrating tools like vector stores.
Large Language Model	OpenAI GPT-4 / Gemini	State-of-the-art reasoning and instruction-following capabilities for accurate analysis and chat.
Embedding Model	`all-MiniLM-L6-v2` (Sentence-Transformers)	Excellent performance for semantic search, runs locally, and is highly efficient.
Vector Database	ChromaDB	Lightweight, open-source, and easy to set up locally. Perfect for rapid prototyping.

Component	Technology Choice	Justification
Document Parsing	PyPDF2	Robust libraries for extracting text content from PDF files.
Containerization	Docker	Ensures a consistent development and runtime environment, simplifying setup.

4.3 Data Flow

A. Invoice Analysis Workflow:

1. **Request:** User sends a `POST` request to `/api/v1/analyze-invoice` with the policy PDF, invoice ZIP, and employee name.
2. **Unpack & Parse:** The system extracts and parses the text from the policy and each invoice PDF.
3. **Analysis:** For each invoice, the system sends the policy text and invoice text to the LLM using a structured prompt (see Appendix A).
4. **Response Generation:** The LLM returns a structured JSON object containing the `reimbursement_status` and `reason`.
5. **Embedding & Storage:**
   - The system generates embeddings for the invoice text.
   - It creates a metadata object containing the LLM's analysis, employee name, and current date.
   - The embeddings and metadata are stored as a document in ChromaDB.
6. **Confirmation:** The API returns a `200 OK` response with a success message.

B. RAG Chatbot Workflow:

1. **Request:** User sends a `POST` request to `/api/v1/chat` with a query (e.g., "Show me all declined invoices for John Doe from last week").
2. **Query Analysis:** The RAG agent's LLM analyzes the user's query to extract intent and entities (e.g., status: "Declined", name: "John Doe", date: "last week").
3. **Tool Execution:** The LLM invokes the `vector_search_tool` with the query text for semantic search and the extracted entities as metadata filters.
4. **Data Retrieval:** The tool queries ChromaDB, which performs a filtered similarity search and returns the most relevant invoice analysis documents.
5. **Response Synthesis:** The retrieved documents are passed back to the LLM as context. The LLM synthesizes this information into a human-readable, Markdown-formatted answer.
6. **Response:** The API returns the final answer to the user.

---

5 Appendix A: Prompt Engineering Strategy

1. Invoice Analysis Prompt:

```
system: You are an expert at analyzing expense invoices against company reimbursement policies
Your task is to analyze the given invoice against the provided policy and determine:
```

```
1. The reimbursement status (FULLY_REIMBURSED, PARTIALLY_REIMBURSED, or DECLINED)
2. The reimbursable amount
3. Detailed reasons for the decision
4. Any policy violations found

Policy:
{policy_text}

Invoice:
{invoice_text}

{format_instructions}

Analyze the invoice based *only* on the provided policy. For partial reimbursements
or declines, your reason must explicitly reference the part of the policy that
justifies the decision.
```

2. RAG Chatbot System Prompt:

```
You are an AI assistant specialized in helping users query and understand invoice
reimbursement analyses.
You have access to a database of invoice analyses and can provide detailed information
about them.

When responding:
1. Use markdown formatting for better readability
2. Be concise but informative
3. If you reference specific invoices, mention their IDs
4. If you're unsure about something, say so
5. Use the provided context to answer questions accurately

Previous conversation:
{chat_history}

Relevant invoice analyses:
{invoice_analyses}

User query: {query}
```