📄Invoice Reimbursement System — Documentation

🗒️Project Overview

The **Invoice Reimbursement System** is an intelligent document processing application that helps automate employee invoice reimbursements by leveraging LLMs and vector databases. It is designed in two main parts:

- **Invoice Analysis API**: Compares uploaded invoice PDFs against an HR policy PDF using an LLM and classifies them as *Fully Reimbursed*, *Partially Reimbursed*, or *Declined*. The analysis is stored with metadata and vector embeddings.
- **RAG Chatbot API**: A natural language interface to query processed invoices based on employee name, date, reimbursement status, and reasons using vector similarity and metadata filtering.

⚙️Installation Instructions

1. Clone the Project or Extract ZIP

```
git clone <repo_url>
cd invoice_rag
```

2. Set Up Virtual Environment

```
python -m venv venv
source venv/bin/activate     # On Windows: venv\Scripts\activate
```

3. Install Dependencies

```
pip install -r requirements.txt
```

4. Add Environment Variables

Create a `.env` file and add your API keys:

```
GROQ_API_KEY=your_groq_api_key
```

5. Run FastAPI Server

```
uvicorn app.main:app --reload
```

🧪Usage Guide

🕐1. Invoice Analysis Endpoint — `/analyze`

- **Method**: `POST`
- **Inputs**:
- `policy_pdf`: HR policy PDF file
- `invoices_zip`: ZIP containing invoice PDFs
- `employee_name`: String
- **Response Example**:

```
{
  "status": "success",
  "processed": 3,
  "skipped": 0,
  "errors": []
}
```

🕐2. RAG Chatbot Endpoint — `/chat`

- **Method**: `POST`
- **Inputs**:
- `query`: Natural language query (e.g., "List all declined invoices for Ramesh in Sep 2024")
- **Response Example**:

```
{
  "answer": "Invoice #397123 from Ramesh dated 17 Sep 2024 was Declined because airport charges are not reimbursable."
}
```

You can also interact via Swagger UI: `http://localhost:8000/docs`

🔍Technical Details

🔧Frameworks & Languages

- **Python** – Core programming language
- **FastAPI** – API development framework
- **Streamlit** – Optional UI for demo and testing

LLMs & Prompting

- **Groq API** with `llama3-8b-8192` model used for:
- Invoice analysis prompt
- Chatbot response generation

🔎Embeddings & Vector Store

- **Sentence-Transformers**: `all-MiniLM-L6-v2`
- **FAISS**: In-memory vector similarity search (cosine distance)
- Metadata filtering used: `employee_name`, `date`, `status`, `invoice_id`

📄PDF Parsing

- **PyMuPDF** (`fitz`) – For text extraction from PDFs
- *Optional:* `unstructured.io` for layout-aware parsing

📦Data Flow Summary

1. Upload PDFs → Extract & Analyze → Store in FAISS with metadata
2. Query via chatbot → Vector search + filters → LLM formats answer

---

Prompt Design

Invoice Analysis Prompt

```
You are an HR compliance assistant. Given a reimbursement policy and an invoice,
determine:
1. Reimbursement status (Fully, Partially, Declined)
2. Reason, based on policy terms.
```

Chatbot Query Prompt

```
Act as a helpful assistant. Use the retrieved invoice metadata and analysis to answer user queries. Format the answer in markdown. Always include invoice status, date, and reason.
```

🕑Challenges & Solutions

Challenge	Solution
Handling large and unstructured PDFs	Used `fitz` and optionally `unstructured.io`
Slow inference with public APIs	Used Groq's ultra-fast LLM (`llama3-8b-8192`)
Metadata filtering mismatch	Normalized and validated formats (e.g., date standardization)
LLM misinterpretation of large policy PDFs	Simplified policy format and used concise prompts

---

📰Code Comments

- All core files (e.g., `analyze.py`, `chat.py`, `vector_store.py`) are commented with:
- Function-level **docstrings**
- **Inline comments** for logic clarity
- Separation of **concerns** (parsing, analysis, storage)

Example:

```python
def analyze_invoice(invoice_text: str, policy_text: str) -> dict:
    """
    Uses LLM to analyze the invoice based on policy.
    Returns status and reason.
    """
    # Step 1: Construct prompt
    # Step 2: Call Groq API
    # Step 3: Extract and return structured response
```

🔗Code Quality

- **PEP8 Compliant**

• Modular structure:
• `api/analyze.py` – Invoice processing logic
• `api/chat.py` – RAG chatbot endpoint
• `utils/vector_store.py` – FAISS integration
• `utils/pdf_parser.py` – PDF text extractor
• Easy to extend for:
• Adding new vector DBs
• Switching LLM providers
• Integrating authentication or a frontend

📦 Tools & Technologies Used

Category	Tool
Language	Python
Backend Framework	FastAPI
UI (optional)	Streamlit
LLM	Groq (`llama3-8b-8192`)
Embedding Model	Sentence-Transformers (`all-MiniLM-L6-v2`)
Vector Store	FAISS
PDF Parsing	PyMuPDF (`fitz`), unstructured.io (optional)
Prompting	Custom prompts for analysis & chat
Env Management	python-dotenv
Server	Uvicorn