⚡ Operations Copilot: Technical Documentation

🚀 Project Overview

Operations Copilot is a next-generation AI-powered platform designed for electrical distribution companies. It integrates **Multimodal LLMs**, **Agentic workflows**, and **Retrieval-Augmented Generation (RAG)** to assist field inspectors and control room operators in real-time.

🏗 System Architecture

The system is built on **Django**, utilizing a modular architecture where AI capabilities are injected into core business workflows.

1. Knowledge Assistant (RAG Pipeline)

- **Engine**: Powered by **CrewAI** and **LangChain**.
- **Workflow**:
    1. User enters a technical query (e.g., "Standard transformer grounding safety").
    2. The `RAGTool` performs a semantic search against the PDF/DOCX knowledge base stored in PostgreSQL.
    3. A specialized **Knowledge Retrieval Agent** synthesizes the answer with exact source citations.
- **Key Tech**: `asgiref`, `langchain-openai`, `ChromaDB` (conceptual storage).

2. Field Inspection Analysis (Vision & Reasoning)

- **Engine**: **Vision LLM** (e.g., GPT-4o or LLaVA).
- **Workflow**:
    1. Field inspector uploads an image of equipment (Transformer, Switchgear, etc.).
    2. The **Vision LLM** identifies components and anomalies (rust, arcing, leakage).
    3. A **Multi-Agent Orchestrator** (Technical, Safety, and Risk Agents) processes the findings sequentially to produce a high-confidence maintenance recommendation.
- **Output**: Risk classification (Low/Med/High/Critical) and a prioritized action checklist.

3. Operations Dashboard

- **Real-time Heartbeat**: Provides a unified view of system health.
- **Dynamic Stats**: Syncs directly with the database to show total inspections, emergency counts, and AI search volumes.
- **Activity Feed**: A chronological stream of all system human/AI interactions.

🛠 Technology Stack

Layer	Technology
Backend	Django 5.x, Python 3.11+
API	Django REST Framework (DRF)
Orchestration	CrewAI, LangChain
AI Models	Groq (Llama), OpenAI (Vision), OpenAI (Embeddings)
Database	PostgreSQL
Frontend	Vanilla JavaScript, Modern CSS (Glassmorphism), Marked.js

---

📂 Directory Structure

```
├── core/                  # Main UI and Agent Orchestration
│   ├── crews/             # CrewAI agent definitions
│   ├── flows/             # State-driven AI workflows
│   ├── tests/             # (Removed for production)
│   ├── tools/             # Custom AI tools (RAG, Vision)
│   └── templates/         # Modern UI views
├── knowledge_base/        # RAG data models and PDF processing
├── field_operations/      # Equipment analysis and Agent reasoning
├── audit/                 # System-wide logging and integrity
├── project/               # Django project settings
└── manage.py              # Application entry point
```

Getting Started

1. Environment Configuration

Ensure your `.env` file contains the required API keys:

- `GROQ_API_KEY`
- `OPENAI_API_KEY`
- `DATABASE_URL`

2. Running the Server

Use the provided shortcut script:

```
./start_server.sh
```

Or use the standard Django command:

```
python manage.py runserver
```

3. Accessing the Platform

- **Dashboard**: `http://localhost:8000/`
- **Knowledge Base**: `http://localhost:8000/assistant/`
- **Admin Control**: `http://localhost:8000/admin/`

Security & Reliability

- **Thread Isolation**: The `RAGTool` uses Python's `threading` library to safely execute Django ORM calls outside of asynchronous event loops, preventing `SynchronousOnlyOperation` crashes.
- **CSRF Protection**: All sensitive actions (like Logout) are protected by Django's state-of-the-art security middleware.
- **Theme Persistence**: User preferences (Dark/Light mode) are cached locally for an instant, responsive UI experience.