AgenticAI - SAP Early Watch Analyzer

AI-Powered SAP System Health Analysis with LangGraph Workflow

A sophisticated Retrieval-Augmented Generation (RAG) system built with LangGraph for analyzing SAP Early Watch Alert (EWA) reports. This application leverages multi-agent workflows to process PDF documents, create embeddings, perform semantic search, and generate comprehensive system health summaries.

Features

Core Capabilities

- **Multi-Agent RAG Pipeline**: LangGraph-powered workflow with specialized agents
- **PDF Processing**: Advanced text extraction from SAP EWA reports
- **Vector Embeddings**: OpenAI embeddings with ChromaDB storage
- **Semantic Search**: Intelligent document retrieval and analysis
- **Email Integration**: Automated report delivery (Gmail/Outlook)
- **Interactive Dashboard**: Streamlit-based web interface

SAP-Specific Features

- **Universal SAP Support**: Works with S/4HANA, IBP, BusinessObjects, and more
- **Traffic Light Analysis**: Automated critical/warning/healthy status detection

- **SAP Recommendations**: Extraction of actionable SAP optimization suggestions
- **Performance Monitoring**: System health metrics and trends analysis
- **Configuration Validation**: Built-in checks for SAP system parameters

Architecture

LangGraph Workflow

The application uses a sophisticated multi-agent workflow built with LangGraph that processes documents through the following sequence:

PDF Processor → Embedding Creator → Vector Store Manager → Search Agent → Summary Agent → System Output Agent → Email Agent → Complete

Key Workflow Features:

- **Sequential Processing**: Each agent builds on the previous agent's output
- **Error Handling**: Failed agents trigger workflow recovery mechanisms
- **Conditional Routing**: Email agent only runs if email is enabled
- **State Management**: Centralized state tracking across all agents
- **Performance Monitoring**: Built-in timing and metrics collection

Agent Responsibilities

Agent	Function	Input	Output
PDF Processor	Extract text from uploaded PDFs	Raw PDF files	Structured text data
Embedding Creator	Generate vector embeddings	Text chunks	Vector embeddings
Vector Store Manager	Store/retrieve embeddings	Embeddings + metadata	ChromaDB storage
Search Agent	Semantic similarity search	User queries	Relevant documents
Summary Agent	Generate comprehensive summaries	Search results	Structured summaries
System Output Agent	Format final outputs	Summaries	Display-ready content
Email Agent	Send automated reports	Formatted content	Email delivery

Installation

Prerequisites

- Python 3.8 or higher
- OpenAI API key
- Git

Quick Start

1. **Clone the Repository**

bash

```bash
git clone https://github.com/srini118us/AgenticAI.git
cd AgenticAI
```

2. **Create Virtual Environment**

bash

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

3. **Install Dependencies**

bash

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

4. **Environment Configuration** Create a `.env` file in the root directory:

```env
# Required
OPENAI_API_KEY=your-openai-api-key-here

# Optional - Email Configuration
EMAIL_ENABLED=true
EMAIL_PROVIDER=gmail
GMAIL_EMAIL=your-email@gmail.com
GMAIL_APP_PASSWORD=your-app-password

# Optional - Advanced Settings
VECTOR_STORE_TYPE=chroma
EMBEDDING_MODEL=text-embedding-ada-002
LLM_MODEL=gpt-4-turbo-preview
CHUNK_SIZE=1000
CHUNK_OVERLAP=200
TOP_K=10
```

5. **Run the Application**

```bash
streamlit run app.py
```

The application will be available at `http://localhost:8501`

Project Structure

```
AgenticAI/
├── app.py              # MAIN APPLICATION (All-in-One)
│                    # ├── Complete LangGraph workflow
│                    # ├── All agent implementations
│                    # ├── Full Streamlit UI
│                    # ├── Configuration management
│                    # ├── Vector store integration
│                    # └── Email functionality
├── data/               # Data storage directory
│   └── chroma/         # ChromaDB vector store
├── requirements.txt      # Python dependencies
├── .env                # Environment variables
├── .env.example         # Environment template
├── README.md            # This file
│
├── Legacy Files (Original Modular Design - Not Used)
├── workflow.py          # Original workflow definition
├── agents.py            # Original agent implementations
├── config.py            # Original configuration
├── models.py            # Original data models
├── core_components.py    # Original shared utilities
├── workflow_utils.py     # Original workflow helpers
└── ui/                 # Original UI components
    └── components.py
```

Current Architecture:

- **Single File**: Everything consolidated in `app.py` for simplicity
- **Self-Contained**: No external dependencies between files

- **Easy Deployment**: Just run `streamlit run app.py`
- **Legacy Files**: Original modular files remain but aren't actively used

Configuration

Environment Variables

Variable	Description	Default	Required
`OPENAI_API_KEY`	OpenAI API key for embeddings/LLM	-	✅
`EMAIL_ENABLED`	Enable email functionality	`false`	❌
`EMAIL_PROVIDER`	Email provider (`gmail`/`outlook`)	`gmail`	❌
`VECTOR_STORE_TYPE`	Vector database type	`chroma`	❌
`EMBEDDING_MODEL`	OpenAI embedding model	`text-embedding-ada-002`	❌
`LLM_MODEL`	OpenAI language model	`gpt-4-turbo-preview`	❌
`CHUNK_SIZE`	Text chunk size for processing	`1000`	❌
`CHUNK_OVERLAP`	Overlap between text chunks	`200`	❌
`TOP_K`	Number of search results	`10`	❌

Gmail Setup (for Email Features)

1. Enable 2-Factor Authentication

2. Generate App Password: Google Account > Security > App passwords
3. Use App Password in `GMAIL_APP_PASSWORD` (not regular password)

Usage

Basic Workflow

1. **Upload Documents**: Drag and drop SAP EWA PDF files
2. **Process Files**: Click "Process Documents" to extract and embed content
3. **Search & Analyze**: Enter queries to search through your documents
4. **Review Results**: View summaries, recommendations, and system health status
5. **Email Reports**: Optionally send results via email

Search Examples

Query Type	Example
Critical Issues	"Show me all critical alerts and errors"
Performance	"Analyze performance bottlenecks and slow queries"
Recommendations	"What are the top SAP recommendations for optimization"
System Health	"Overall system health status and warnings"
Configuration	"Check system configuration and parameter settings"

Advanced Features

- **System Selection**: Filter analysis by specific SAP systems

- **Traffic Light Status**: Automatic categorization of issues (Critical, Warning, Healthy)
- **Export Results**: Download analysis results and workflow diagrams
- **Debug Mode**: Detailed logging and performance metrics

Key Components

LangGraph Workflow Engine

The core workflow is built using LangGraph, providing:

- **State Management**: Centralized workflow state tracking
- **Error Handling**: Robust error recovery and reporting
- **Conditional Routing**: Dynamic workflow paths based on data
- **Agent Coordination**: Seamless communication between agents

Intelligent PDF Processing

- **Multi-Library Support**: PyPDF2, pdfplumber, PyMuPDF for reliability
- **Text Cleaning**: Removal of PDF artifacts and encoding issues
- **Metadata Preservation**: Document tracking and debugging information
- **Fallback Mechanisms**: Multiple extraction methods for resilience

Vector Storage & Search

- **ChromaDB Integration**: High-performance vector database
- **Semantic Search**: Context-aware document retrieval
- **Embedding Caching**: Optimized performance for repeated queries

- **Similarity Scoring**: Relevance-ranked search results

Testing

Run Basic Tests

```bash
python -c "from workflow_utils import test_workflow_basic; test_workflow_basic()"
```

Validate Configuration

```bash
python -c "from workflow_utils import validate_workflow_config, get_default_config; vali
```

Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit your changes (`git commit -m 'Add some AmazingFeature'`)
4. Push to the branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request

Requirements

Core Dependencies

```txt
streamlit>=1.20.0
langchain>=0.1.0
langgraph>=0.1.0
openai>=1.0.0
chromadb>=0.4.0
PyPDF2>=3.0.0
pdfplumber>=0.9.0
PyMuPDF>=1.23.0
python-dotenv>=1.0.0
```

Optional Dependencies

```txt
# Email functionality
smtplib
ssl

# Advanced features
numpy>=1.24.0
pandas>=2.0.0
```

Troubleshooting

Common Issues

1. OpenAI API Errors

```bash
Error: Incorrect API key provided
Solution: Verify OPENAI_API_KEY in .env file
```

2. PDF Processing Failures

```bash
Error: Failed to extract text from PDF
Solution: Try different PDF or check file corruption
```

3. ChromaDB Issues

```bash
Error: ChromaDB connection failed
Solution: Clear ./data/chroma directory and restart
```

4. Email Configuration

```bash
Error: Gmail authentication failed
Solution: Use App Password, not regular password
```

Debug Mode

Enable debug mode in the application for detailed logging:

```python
# In Streamlit sidebar
debug_mode = True
```

License

Acknowledgments

- **LangGraph**: For the powerful workflow orchestration framework
- **LangChain**: For the comprehensive LLM toolkit
- **OpenAI**: For embeddings and language model APIs
- **Streamlit**: For the intuitive web application framework
- **ChromaDB**: For efficient vector storage and retrieval

Support

For questions, issues, or contributions:

- **GitHub Issues**: Create an issue
- **Documentation**: Check inline code comments and docstrings
- **Community**: Join discussions in GitHub Discussions

Built with ❤️ for SAP System Analysis

Making SAP Early Watch Alert analysis intelligent, automated, and actionable.