Getting Started with Smart Document Q&A System

---

1. What You're Building

An AI-powered document question-answering platform featuring:

- **Multi-Agent Intelligence** – 3 coordinated AI agents using LangGraph
- **Hybrid Search** – Keyword + semantic (vector) retrieval
- **Persistent Memory** – Multi-turn, context-aware conversations
- **Source Citations** – Answers grounded in documents
- **Transparent Reasoning** – Agent decisions are visible and traceable

This system is suitable for enterprise knowledge bases, healthcare documents, internal wikis, and AI portfolio demonstrations.

---

2. Prerequisites

Ensure the following are installed:

- Python **3.10+**
- Docker Desktop (running)
- OpenAI API key
- Basic familiarity with command line
- Estimated setup time: **~10 minutes**

---

3. Step-by-Step Setup

Step 1: Configure OpenAI API Key

Open the `.env` file and set your API key:

```
OPENAI_API_KEY=sk-proj-your-actual-key-here
```

Save the file.

Step 2: Run Automated Setup

From the project root, open PowerShell and run:

```
python setup.py
```

This script will:

- Verify Python version
- Validate environment variables
- Install dependencies
- Initialize the database
- Check Typesense availability

Sample Output:

```
Python version: OK
Environment: Configured
Dependencies: Installed
Database: Initialized
Typesense: running
```

Step 3: Start Typesense

Run the following command or double-click `start_typesense.bat`:

```
docker run -d -p 8108:8108 -v ${PWD}/typesense-data:/data
typesense/typesense:26.0 --data-dir /data --api-key=xyz --enable-cors
```

Verify:

```
curl http://localhost:8108/health
```

Expected response:

```
{"ok": true}
```

Step 4: Start the API Server

```
uvicorn app.main:app --reload --port 8000
```

Expected logs:

```
INFO: Uvicorn running on http://127.0.0.1:8000
INFO: Application startup complete.
```

Step 5: Run the Demo

Open a **new terminal** and run:

```
python demo.py
```

The demo will:

1. Upload sample documents
2. Create a conversation
3. Ask multiple question types
4. Display agent reasoning
5. Show answers with citations

---

4. What to Observe

Agent Reasoning Output

```
Agent Decisions:
• QueryAnalyzer → Semantic search
• SearchCoordinator → Hybrid weighting applied
• ResponseGenerator → Answer generated with citations
```

Hybrid Search Behavior

- Factual questions → Keyword-heavy
- Conceptual questions → Semantic-heavy
- Complex queries → Balanced hybrid

Memory Handling

- Previous questions influence new answers
- Context tokens are tracked and optimized
- Older context is summarized intelligently

---

5. Quick API Test

```
curl -X POST "http://localhost:8000/api/v1/ask" \
  -H "Content-Type: application/json" \
  -d '{"question": "What is Retrieval-Augmented Generation?"}'
```

Interactive Swagger UI: □ http://localhost:8000/docs

---

6. Upload Your Own Documents

```
curl -X POST "http://localhost:8000/api/v1/documents/upload" \
  -F "file=@path/to/your/document.txt"
```

Ask questions:

```
curl -X POST "http://localhost:8000/api/v1/ask" \
  -H "Content-Type: application/json" \
  -d '{"question": "Your question here", "use_context": true}'
```

---

7. System Overview (Conceptual)

AI Agents

1. **Query Analyzer** – Understands intent and selects search strategy
2. **Search Coordinator** – Executes hybrid search and ranks results
3. **Response Generator** – Produces grounded answers with citations

Memory System

- Conversation history stored persistently
- Context window optimized under token limits
- Search history logged for analysis

Search Engine

- Keyword (BM25)
- Semantic (vector embeddings)
- Hybrid (adaptive weighting)

---

8. Troubleshooting

Typesense Not Reachable

```
docker ps
```

Restart if needed:

```
docker start smart-qa-typesense
```

OpenAI API Errors

- Verify API key in `.env`
- Ensure sufficient credits
- Test connectivity manually

Port Conflicts

```
uvicorn app.main:app --port 8001
```

---

9. Project Structure

```
smart_document_search/
├── app/
│   ├── agents/
│   ├── memory/
│   ├── search/
│   └── main.py
├── sample_documents/
├── demo.py
├── setup.py
└── docs/
```

---

10. Success Checklist

After setup, you should observe:

- Documents indexed successfully
- Multiple questions answered
- Citations included
- Agent reasoning visible
- Response time under 4 seconds

---

11. Why This System Stands Out

- Multi-agent reasoning (not a basic RAG)
- Adaptive hybrid retrieval
- Transparent decision-making
- Production-ready structure
- Ideal for interviews, demos, and real deployments

---

12. Next Steps

- Review **ARCHITECTURE docs** for design decisions
- Explore API via Swagger UI
- Test with domain-specific documents
- Extend with authentication or caching
