Projekt-Blueprint: Agentic RAG Business Assistant

1. Projektidee

Wir bauen eine REST-basierte AI-Anwendung, die Dokumente (z. B. PDF-Berichte, interne Knowledge-Docs) verarbeiten und intelligente Antworten auf Fragen geben kann.

Das System kombiniert Retrieval-Augmented Generation (RAG) mit Agentic AI, wodurch ein LLM selbstständig Tools (z. B. Vektor-Suche oder externe APIs) nutzen kann, um Anfragen zu beantworten. Die REST-API wird mit FastAPI entwickelt, LangChain dient als Framework für RAG und Agenten, und PostgreSQL mit pgvector wird für die Vektor-Suche verwendet. Alles wird mittels Docker containerisiert.

2. Projektziele

1. Dokumentenverwaltung: Dokumente hochladen, zerlegen (Chunking) und als Embeddings speichern.
2. Frage-Antwort-System: Fragen zu Dokumenten mithilfe von RAG beantworten.
3. Agentic Layer: AI-Agent entscheidet zwischen mehreren Tools (z. B. RAG oder externe APIs).
4. REST-API Endpoints:
   - /health - Systemstatus
   - /ingest - Dokumente hochladen
   - /query - Frage stellen
   - /agent - Agentic-Antwort (mit Toolauswahl)
5. Testdaten: Nutzung von Beispiel-PDFs (Business-Reports oder technische Dokumentation).

3. Benötigte Komponenten

Software & Tools:

- Python 3.11+
- FastAPI (REST-API)
- LangChain (für RAG, Tools, Agenten)
- PostgreSQL + pgvector (Vektor-Datenbank)
- Docker & docker-compose (Containerisierung)
- OpenAI API (LLM & Embeddings)

Infrastruktur:

- 2 Container: API (FastAPI + LangChain) und DB (Postgres + pgvector)

- Lokale Entwicklungsumgebung (VSCode oder PyCharm)
- API-Testtool wie Postman oder cURL

Testdaten:
- Beispiel-PDFs (Business-Report oder technische Dokumentation)

4. Architekturüberblick

Das System folgt einem modularen Aufbau:

[User/Frontend] -> [FastAPI Backend] -> [LLM via OpenAI API]
-> [Postgres + pgvector]
-> [LangChain Agent Tools]

FastAPI dient als REST-Schnittstelle, LangChain übernimmt Agenten- und RAG-Logik, und Postgres mit pgvector speichert Embeddings und Dokumenten-Chunks.

5. Schritt-für-Schritt-Plan

Phase 1: Planung & Setup
- Projektstruktur festlegen (Ordner, Module, Dockerfiles).
- docker-compose mit FastAPI & Postgres aufsetzen.
- Testdaten (PDFs) auswählen.

Phase 2: Backend-Basis (FastAPI)
- FastAPI-Server einrichten.
- Endpunkte definieren: /health, /ingest, /query.

Phase 3: Vektor-Datenbank
- Postgres mit pgvector vorbereiten.
- Schema für Dokumente und Embeddings planen.
- Embeddings mit OpenAI generieren und speichern.

Phase 4: RAG-Implementierung
- Chunking-Logik planen (z. B. 1000 Zeichen pro Chunk).
- Retriever implementieren (Vektor-Suche nach Top-K Chunks).
- LLM-Anbindung (OpenAI API).

Phase 5: Agentic Layer

- Agent-Workflow definieren (Toolauswahl zwischen RAG und API).
- Tools definieren: vector_search, external_api.
- LangChain-Agent einbinden.

Phase 6: Testing & Monitoring

- Testfälle für Dokumente und Fragen.
- Optional: Integration von Langfuse (Monitoring).

Phase 7: Erweiterungen (optional)

- Frontend: Weboberfläche (Streamlit/React).
- Deployment: Cloud-Hosting (z. B. Render oder später Azure/AWS).

6. Endziel

Ein vollständig containerisiertes AI-System mit FastAPI, LangChain und pgvector, das Fragen zu Dokumenten beantwortet und agentische Fähigkeiten besitzt. Das Projekt ist modular erweiterbar und für Business-Anwendungen optimiert.