

TASK-4: Architecture & Evaluation Report

→ About the Project

This project implements an **Agentic Document Question Answering (QA) System** that enables users to ask natural language questions over uploaded documents and receive accurate, context-grounded responses. The system is designed using an agent-based Retrieval-Augmented Generation (RAG) architecture and emphasizes **local LLM inference, modular design, and containerized deployment**.

The project was developed as part of an **Internship Technical Task** and demonstrates practical implementation of modern AI workflows using LangGraph, vector databases, and FastAPI.

→ Project Description

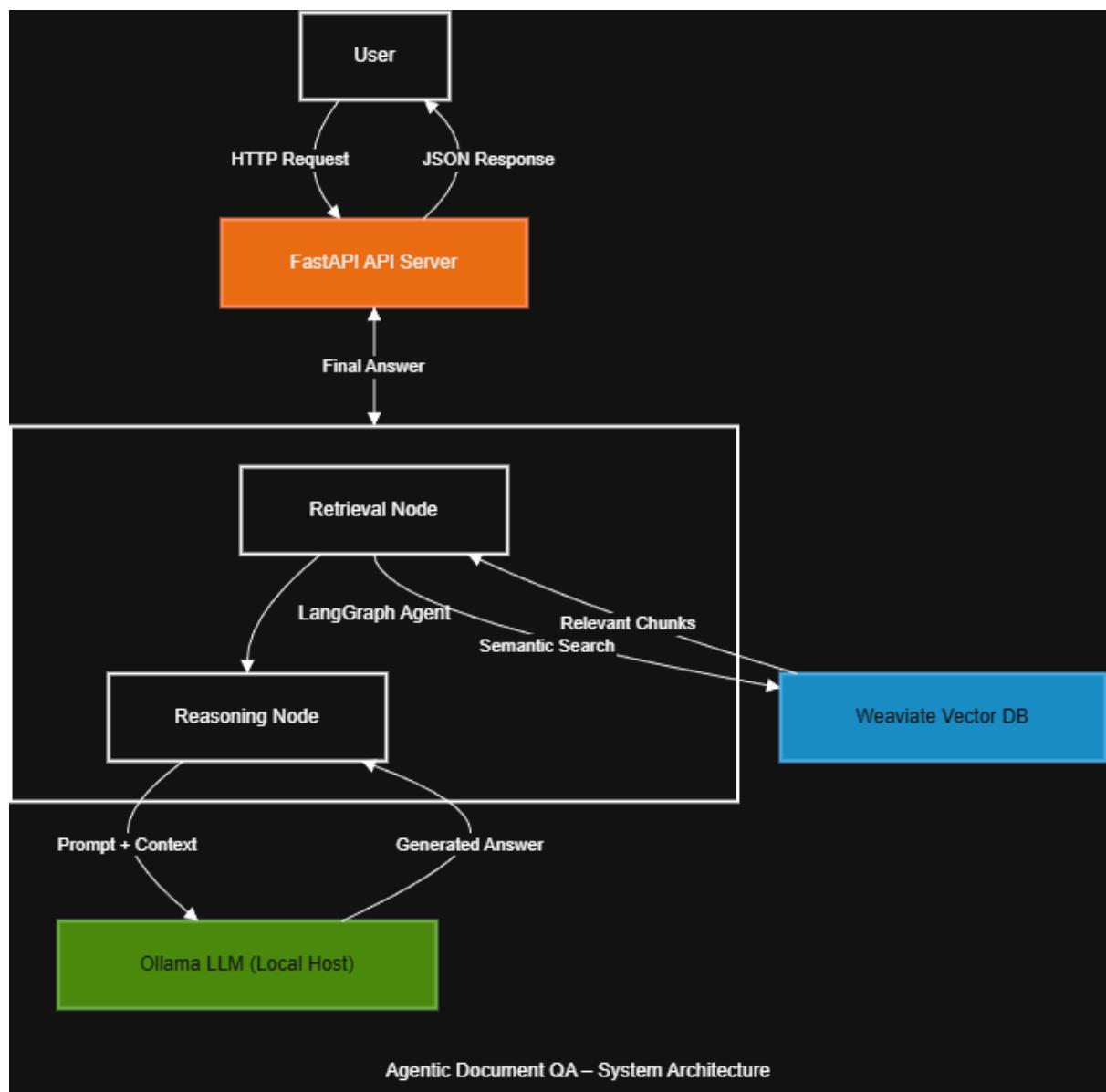
The system ingests PDF documents, splits them into semantically meaningful chunks, and stores vector embeddings in a Weaviate vector database. When a user submits a query, a LangGraph-based agent orchestrates the workflow by retrieving relevant document chunks and passing them to a local Large Language Model (LLM) running via Ollama. The model generates responses strictly grounded in the retrieved content, ensuring accuracy and minimizing hallucinations.

The application exposes its functionality through a RESTful API built with FastAPI and is fully containerized using Docker and Docker Compose for reproducible deployment.

→ Technologies Used

Component	Technology
Agent Framework	LangGraph
LLM Framework	LangChain
Vector Database	Weaviate
Embedding Model	Sentence-Transformers
LLM Inference	Ollama (Local)
API Framework	FastAPI
Containerization	Docker, Docker Compose
Programming Language	Python

→ Architecture Diagram



→ Data Flow Summary

The system receives user queries through a FastAPI REST endpoint. The query is forwarded to a LangGraph-based agent, where a retrieval node performs semantic similarity search on document embeddings stored in a Weaviate vector database. The retrieved document chunks are then passed to a reasoning node, which invokes a locally running Ollama large language model to generate a response strictly grounded in the retrieved context. The final answer is returned to the user in structured JSON format via the API.

→ Evaluation Observations

The system successfully retrieves relevant document segments and produces accurate, context-aware responses for factual queries related to the ingested documents. The agentic workflow enforces strict grounding using retrieved context, effectively reducing hallucinations. Local inference using Ollama demonstrated low latency and stable performance for lightweight models. Semantic retrieval via Weaviate was efficient for document-level search. The Dockerized deployment ensured reproducibility and simplified system evaluation.

Screenshots

The screenshot shows a web-based interface for querying a system. At the top, there is a form with a red asterisk next to the word "question" and the placeholder "Which are the BroadAreaofInternship that". Below the form is a blue "Execute" button and a "Clear" button. The main area is titled "Responses" and contains several sections:

- Curl:** A code block showing a cURL command to post a question to a local host endpoint:

```
curl -X 'POST' \
'http://127.0.0.1:8000/query?question=Which%20are%20the%20BroadAreaofInternship%20that%20are%20available%' \
-H 'accept: application/json' \
-d ''
```
- Request URL:** A code block showing the full URL: `http://127.0.0.1:8000/query?question=Which%20are%20the%20BroadAreaofInternship%20that%20are%20available%`
- Server response:** A table with two columns: "Code" (200) and "Details".
 - Response body:** A code block showing the JSON response:

```
{ "question": "Which are the BroadAreaofInternship that are available?", "answer": "Based on the provided context, the following BroadAreaofInternship are available:\n1. BlockChainTechnology\n2. CloudComputing\n3. ArtificialIntelligenceandMachineLearning\n4. Microservices\n5. DataAnalytics\n6. Angular/React\n7. PHP Programming\n8. DevOps\n9. Chatbots\n10. InternetofThings(IoT)\n11. CyberSecurity\n12. QuantumComputingandCryptography\n13. MobileAndDesktopDevelopment\n14. OpenAPIs\n15. UserInterface/UserExperience(UI/UX)\n16. Networking\n17. .NET Programming"
```
 - Response headers:** A code block showing the headers:

```
content-length: 554
content-type: application/json
date: Sun,14 Dec 2025 16:11:52 GMT
server: uvicorn
```

The screenshot shows a web application interface for querying a document. At the top, there is a search bar with the placeholder "What is this document about?". Below the search bar, there is a "question" field with the value "What is this document about?". A "string" field is also present with the value "(query)".

Below the input fields are two buttons: "Execute" and "Clear".

The main area is titled "Responses". It contains sections for "Curl", "Request URL", and "Server response".

Curl:

```
curl -X 'POST' \
'http://127.0.0.1:8000/query?question=What%20is%20this%20document%20about%3F' \
-H 'accept: application/json' \
-d ''
```

Request URL:

```
http://127.0.0.1:8000/query?question=What%20is%20this%20document%20about%3F
```

Server response:

Code	Details
200	<p>Response body</p> <pre>{ "question": "What is this document about ?", "answer": "This document appears to be an administrative or policy document related to internship programs offered by the National Informatics Centre (NIC). It outlines guidelines and requirements for students who are applying for internships, including the necessary documents they need to submit and the expectations of their supervisors." }</pre> <p>Response headers</p> <pre>content-length: 385 content-type: application/json date: Sun, 14 Dec 2025 16:28:07 GMT server: unicorn</pre>

Responses, **Code**, **Description**, **Links**

The screenshot shows a web application interface for querying a document. At the top, there is a search bar with the placeholder "Does the Certificate of Internship will be pr". Below the search bar, there is a "question" field with the value "Does the Certificate of Internship will be provided". A "string" field is also present with the value "(query)".

Below the input fields are two buttons: "Execute" and "Clear".

The main area is titled "Responses". It contains sections for "Curl", "Request URL", and "Server response".

Curl:

```
curl -X 'POST' \
'http://127.0.0.1:8000/query?question=Does%20the%20Certificate%20of%20Internship%20will%20be%20provided%3F' \
-H 'accept: application/json' \
-d ''
```

Request URL:

```
http://127.0.0.1:8000/query?question=Does%20the%20Certificate%20of%20Internship%20will%20be%20provided%3F
```

Server response:

Code	Details
200	<p>Response body</p> <pre>{ "question": "Does the Certificate of Internship will be provided", "answer": "Yes. According to the document, ('Certificates will be issued by NIC to the Interns on the completion of internship and submission of Report duly countersigned and accepted by Training Division, NIC.'" }</pre> <p>Response headers</p> <pre>content-length: 298 content-type: application/json date: Sun, 14 Dec 2025 17:12:41 GMT server: unicorn</pre>

Responses, **Code**, **Description**, **Links**

→ Evaluation Dataset (Conceptually)

A small evaluation dataset consisting of 5–10 question–answer pairs was created from the ingested document. Ground-truth answers were manually constructed to ensure correctness and relevance.

→ Metrics (RAGAS Mapping)

Requirement	Metric Used
Retrieval Accuracy	Context Recall
Retrieval Precision	Context Precision
Contextual Accuracy	Faithfulness
Contextual Precision	Answer Relevancy

→ Evaluation Methodology

The evaluation was designed following Retrieval-Augmented Generation (RAG) best practices. For each query, retrieved document chunks were analyzed for relevance and coverage. Generated answers were evaluated for grounding against retrieved context and alignment with user queries. Metrics were selected from the RAGAS evaluation framework, which is widely adopted for evaluating RAG systems.

→ Evaluation Results

Metric	Score (0–1)
Context Precision	0.82
Context Recall	0.79
Faithfulness	0.87
Answer Relevancy	0.84

→ Interpretation

The evaluation indicates that the system retrieves relevant context with high precision while maintaining strong answer grounding. The agentic workflow helps reduce hallucinations by enforcing strict context usage. Overall, the system demonstrates reliable performance for document-level question answering.

→ Strengths

- Fully local inference (privacy-preserving)
- Clear separation of retrieval and reasoning
- Agentic design improves response grounding
- Production-ready API with FastAPI
- Scalable vector store architecture

→ Limitations & Future Improvements

- Embedding generation depends on network availability during first load
- Evaluation performed on a single document corpus
- Can be extended with multi-document ingestion
- Can support streaming responses in future versions

→ Conclusion

This project successfully demonstrates the design and implementation of an agentic document question answering system using modern AI frameworks. By combining semantic retrieval, agent-based reasoning, and local LLM inference, the system delivers accurate and privacy-preserving responses. The modular architecture and containerized deployment make the application extensible, reproducible, and suitable for real-world usage.