# **Application Workflow: An Intelligent Document Q&A System**

## **1. Introduction**

This document outlines the complete operational flow of the automated Document Q&A system. The application is designed to intelligently interpret and answer questions based on a wide variety of documents (PDFs, PowerPoints, web pages, etc.). Its core feature is a sophisticated routing mechanism that analyzes both the provided document and the user's specific query to choose the most efficient and effective method for finding an answer.

## **2. High-Level Process Overview**

The application operates in four distinct stages, moving from general document analysis to specific question-answering and final reporting.

+---------------------------------+

| STAGE 1: DOCUMENT |

| PREPARATION & INGESTION |

+---------------------------------+

|

▼

+---------------------------------+

| STAGE 2: STRATEGY ROUTING |

| (Document-Level Analysis) |

+---------------------------------+

|

+--------+---------+

| |

▼ ▼

+--------------+ +---------------+

| PATH A: | | PATH B: |

| PARALLEL | | SEQUENTIAL |

| RAG | | AGENT |

+--------------+ +---------------+

| |

+--------+---------+

|

▼

+---------------------------------+

| STAGE 3: FINAL OUTPUT |

| & REPORTING (JSON) |

+---------------------------------+

## **3. Detailed Stage Breakdown**

### **Stage 1: Document Preparation & Ingestion**

This is the foundational, one-time setup that occurs for each new document. The goal is to make the document's content searchable and understandable for the AI.

* **Initialization:** The core MultiDocumentRAG system starts, establishing connections with Azure AI services for language models (LLMs) and the Qdrant vector database.
* **Document Download & Type Detection:** The system downloads the content from the provided URL and automatically identifies the file type (e.g., .pdf, .pptx, .html).
* **Comprehensive Text Extraction:**
  + The system uses a specialized parser for the detected file type to extract all readable text.
  + Crucially, for documents like PowerPoint (.pptx) or image-heavy PDFs, it performs **Optical Character Recognition (OCR)** on embedded images to capture text that would otherwise be missed.
* **Chunking & Vectorization:**
  + The complete extracted text is segmented into small, semantically related chunks.
  + Each chunk is converted into a numerical representation (a "vector embedding") that captures its meaning.
* **Indexing for Search:**
  + **Vector Index:** The text chunks and their corresponding vectors are stored in the Qdrant database. This allows for "semantic search," where the system can find chunks that are conceptually similar to a user's query, not just those with matching keywords.
  + **Keyword Index:** A parallel BM25 index is created. This is a traditional keyword-based index that excels at finding chunks with exact term matches.
* **Caching:** All processed data (chunks and indexes) are saved to a local directory. If the same document is processed again, the system skips the ingestion steps and loads directly from this cache, saving significant time.

### **Stage 2: Initial Strategy Routing (The Master Decision)**

After preparing the document, the system makes a single, high-level decision to determine the most efficient way to handle all subsequent questions for this document.

* **Heuristic Analysis:** The system analyzes the document based on two key factors:
  + **URL Type:** Does the URL point to a static file (e.g., document.pdf) or a dynamic endpoint (e.g., .../get-token)?
  + **Keyword Density:** It calculates the ratio of action-oriented keywords (like api and http) to the total amount of text.
* **Strategy Selection:**
  + **Simple Strategy (Parallel RAG):** Chosen if the document is a static file with a **low** keyword density. This indicates an informational document (report, article, manual) where URLs are likely just references.
  + **Complex Strategy (Sequential Agent):** Chosen if the document is a non-static URL **OR** if it has a **high** density of keywords. This indicates an instructional document where interacting with APIs or URLs is the primary purpose.

### **Stage 3: Question Answering Execution**

The application now proceeds down one of two paths based on the strategy decision.

#### **Path A: Parallel RAG (The "Fast Lane" for Simple Documents)**

* **Concurrent Processing:** All user questions are submitted for processing **simultaneously** using asyncio.gather for maximum speed.
* **Hybrid Search:** For each question, the system queries both the Vector Index (for semantic meaning) and the Keyword Index (for exact matches) to retrieve the most relevant text chunks from the document.
* **Answer Synthesis:** The retrieved chunks are compiled into a context, which is then passed to an LLM with a precise prompt: *"Answer the user's question based only on the provided text."* The LLM synthesizes a final, accurate answer.
* **Result Collection:** The system gathers all the answers from the parallel tasks.

#### **Path B: Sequential Agent (The "Deep Dive" for Complex Documents)**

For complex documents, each question is processed individually through the Orchestrator—a sophisticated workflow graph—to allow for multi-step reasoning and tool use.

For each question, the following workflow is executed:

1. **analyse\_document\_content (Per-Query Router):** The agent first analyzes the *intent* of the specific question.
   * A general query like *"What is this document about?"* is routed to the simple rag\_agent.
   * An action-oriented query like *"What is the flight number?"* is routed to the set\_goal\_agent.
2. **set\_goal\_agent (The Planner):** If the agentic path is chosen, this node uses an LLM to create a step-by-step JSON plan to solve the query.
   * *Example Plan:* {"steps": [{"description": "Call the API at https://..."}, {"description": "Extract the 'city' value from the result"}]}
3. **perform\_action (The Executor):** The agent executes one step of the plan at a time. It decides if the step requires calling a tool (like an API) or just retrieving information from the document text.
4. **call\_tool (The Tool User):** If a tool is needed, this node makes the actual API call (e.g., requests.get(...)). The data returned by the API is added to the agent's memory.
5. **Execution Loop:** The agent loops through the perform\_action -> call\_tool -> increment\_step nodes until all steps in its plan are completed.
6. **answer\_query (The Synthesizer):** Once the plan is fully executed, this final node reviews the entire history of actions and results to generate a comprehensive, final answer to the user's original question.

## **4. Stage 4: Final Output & Reporting**

This final stage consolidates the results from either execution path into a structured output.

* **Data Aggregation:** A list of all question-and-answer pairs is compiled.
* **JSON Report Generation:** A orchestrator\_session\_results.json file is created, containing:
  + The source DOC\_URL.
  + The complete list of questions and their generated answers.
  + A timing\_stats object with performance metrics, including document processing time, total querying time, and the average time per question.