Knowledge Base (FAQ / RAG) System – Full Architecture and Feature Breakdown

---

1. Feature Overview

The Knowledge Base (KB) system is designed to ingest, enrich, and retrieve organizational knowledge, enhancing an AI-driven chatbot's capabilities. It supports various input formats, manages periodic refresh and change detection, and integrates with Azure-native services for enterprise-level deployment.

Primary Use Cases

- **FAQ-style response generation**
- **Intelligent document answering (RAG)**
- **Periodic auto-refresh of hosted files and URLs**
- **Full lifecycle management through a React-based admin panel**

---

2. Data Flow Breakdown

1. **Source Submission:** Users can upload a file or provide a link via the React panel.
2. **Validation:** A Node.js BFF validates the input and stores metadata in Cosmos DB.
3. **Queueing:** The source is enqueued into Azure Service Bus.
4. **Ingestion:** A Python worker consumes the queue:
   - Cleans and chunks content
   - Generates embeddings via Azure OpenAI
   - Indexes vectors in Azure AI Search
   - Updates Cosmos DB metadata
5. **Refresh Scheduler:** A Python scheduler calculates future refresh intervals and enqueues jobs.
6. **Retrieval:** LangChain retriever and LangGraph agent process queries and return answers.

⚙️ *Typical end-to-end ingestion latency per document is approximately 5–15 seconds. Refresh-based re-indexing typically completes in under 10 seconds, depending on document size and Azure OpenAI throughput.*

---

3. Architecture Overview

Layer	Technology	Purpose
Frontend	React + MSAL + Entra ID	Admin panel, user auth, source management
Backend (BFF)	Node.js	API orchestration, token validation
Queue	Azure Service Bus	Asynchronous ingestion & refresh task queueing
Scheduler	APScheduler (Python)	Trigger refresh tasks per cron/interval
Worker	Python + LangChain	Content processing, embedding, refresh engine
Vector Store	Azure AI Search	Embedding storage and semantic retrieval
Metadata DB	Cosmos DB (Mongo API)	Tracks sources, chunks, and logs

---

4. Component Responsibilities

🟦 React Admin Panel

- Upload and manage sources
- Configure refresh and retention policies
- Filter and inspect chunks and logs
- Secure access via Microsoft Entra ID

🔁 Node.js BFF

- Serves all frontend-backed endpoints
- Authenticates via Entra tokens
- Manages queue publishing and audit logging

🐍 Python Worker & Scheduler

- Pulls messages from Azure Service Bus
- Manages ingestion, retries, and refresh logic
- Applies diffing and chunk versioning

- Emits traceable logs for observability

---

5. Supported Source Types

Type	Refreshable	Notes
File Upload	Optional	Supports .txt, .pdf, .xlsx, .docx
Link (HTML)	Yes	Single-page scrape; does not follow child links
Link (Files)	Yes	Treated like file upload with periodic re-check
Image / OCR	No	Out of scope

---

6. Refresh, Retry, and Deletion Policies

🔄 Refresh

- Triggered on a cron schedule or fixed interval
- Scheduled using APScheduler

🔁 Retry

- max_retries: Configurable per source
- retry_backoff_hours: Delay between retry attempts

🗑️ Deactivation & Deletion

- If a source fails max_retries, alerts are sent
- After action_grace_hours, the system will deactivate or delete content
- Chunks are deactivated if content changes (based on hash diffs)
- delete_after_days controls removal of deactivated chunks:
  - null: Retain indefinitely
  - Integer: Auto-delete X days after deactivation

This unified field replaces the need for a preserve flag.

---

7. API Endpoints (Expanded)

GET /chunks

Filter and sort chunks

Query Params:

- source_id, status, tags, version, sort_by, order

GET /sources

List sources with filtering

Query Params:

- type, tags, created_by, refresh_enabled, active, sort_by, order

POST /ingest

Submit a new source and configure behavior

Body:

```
{
  "source_type": "file",
  "source_url": "https://...",
  "refresh_strategy": { "type": "cron", "value": "0 9 * * MON" },
  "max_retries": 5,
  "action_grace_hours": 48,
  "delete_after_days": 30,
  "notification_emails": ["ops@example.com"],
  "tags": ["legal"],
  "refresh_enabled": true
}
```

POST /query

RAG query execution

Body:

```
{
"query": "How do I update my invoice?"
}
```

PATCH /sources/:id

Update metadata or refresh policies

DELETE /sources/:id

Remove or deactivate a source (mode: delete or deactivate)

GET /status/source

Return status of ingestion and refresh tasks including queue state, retry counts, logs, and scheduling details

Response Includes:

- queued_at, ingestion_started_at, ingestion_completed_at
- current_status: queued | processing | completed | failed

Query Parameters:

- source_id: ID of the source

POST /sources/test

Test the accessibility of a given URL and automatically detect the content type.

Purpose:

- Used before ingesting a link-based source (especially hosted documents)
- Determines whether the link points to a valid and accessible resource
- Classifies the content as text/html, application/pdf, application/vnd.ms-excel, etc.

Typical Use Cases:

- Validate user input before ingestion
- Classify unknown file types to route to appropriate parser
- Provide UI feedback (e.g., link is broken, or type not supported)

Request Body:

```
{
  "url": "https://example.com/resource.pdf"
}
```

Response:

```
{
  "status": "ok",
  "content_type": "application/pdf",
  "file_size": 134900,
  "headers": { "Last-Modified": "...", "ETag": "..." }
}
```

---

8. Cosmos DB Schema

sources

```
{
  "_id": "source_xyz",
  "type": "file",
  "source_url": "https://...",
  "refresh_enabled": true,
  "refresh_strategy": { "type": "cron", "value": "0 9 * * MON" },
  "max_retries": 3,
  "action_grace_hours": 24,
  "delete_after_days": 30,
  "notification_emails": ["ops@client.com"],
  "tags": ["legal", "faq"],
  "created_by": "user@client.com",
  "active": true,
  "status": {
    "last_success": "...",
    "retry_count": 0,
    "last_known_hash": "abc123"
  }
}
```

chunks

Note: The version field tracks internal chunk revisions based on hash diffs. This is maintained automatically and is not exposed or user-managed. Versioning is incremented per ingestion cycle when changes are detected. No rollback or user-facing history is currently supported.

```
{
  "_id": "chunk_123",
  "source_id": "source_xyz",
  "content": "...",
  "active": true,
  "content_hash": "abc123",
  "version": 2,
  "deactivated_at": null,
  "ingested_at": "..."
}
```

logs

```
{
  "_id": "log_001",
  "source_id": "source_xyz",
  "action": "refresh",
  "status": "fail",
```

```
  "triggered_by": "system",
  "timestamp": "...",
  "retry_count": 2,
  "message": "Timeout while fetching URL"
}
```

---

9. LangChain + LangGraph Integration

- **Retriever:** Filters only active chunks
- **Agent Flow:**
  - Nodes: faq, tools, END
  - Tools: search_docs defined with @tool
  - Prompting via dynamic template injection
- **Fallback Logic:**
  - If search_docs returns no results, the agent can optionally return a fallback response (e.g., default message, direct LLM completion).
  - Future plans include reranking (e.g., BGE reranker, GPT-4 re-ranking), hybrid retrieval, and metadata filtering.
- **Tracing:**
  - Uses ToolMessage, AIMessage, HumanMessage
  - Fully observable via LangSmith (tool calls, document hits, final output)

---

10. Security & Access Control

- **Auth:** Microsoft Entra ID via MSAL (React) and token auth (Node)
- **Logging:** All actions logged with user identity
- **Access Control:** Panel access limited to verified organization members

---

11. Glossary

- **RAG** – Retrieval-Augmented Generation
- **MSAL** – Microsoft Authentication Library
- **BFF** – Backend for Frontend
- **TTL** – Time-To-Live