ITNB AG

AI ENGINEERING INTERNSHIP TECHNICAL ASSESSMENT

Marc Arias

December 17, 2025

STAGE 2: Written Design Questions

Question 1: Role-Based Access Control (RBAC) Implementation

Based on the GroundX documentation and available API capabilities, how would you design a document-level access control (authentication/authorization) mechanism?

In my opinion, the key to have a professional RAG system is not just that "finds the correct answer", but that it ensures that the **user has the right to view it according to the company's rules**. In enterprise environments, documents often have different access levels depending on the user (e.g., by department or role).

In practice, in a RAG system that uses GroundX, the access control shouldn't be managed solely within the search system; it should be synchronized with the company's identity infrastructure (e.g. MS SharePoint).

How you would associate documents and users (e.g., using metadata, buckets, or tags)

It depends on the required granularity of the access control.

- For **broad organizational purposes** (e.g. organizing documents by departments), I would group by **buckets** to separate documents during ingestion (for example, HR or Financial buckets). But I would not use them as the main security boundary, I would consider them just for an organizational function.
- For more **specific access control**, I would say **metadata tags** are more appropriate. Then, the metadata tag would be attached to the ingested document, reflecting the permissions defined in the company's authorization system (e.g. MS SharePoint). In GroundX, it would look like:
  `{"roles": ["director", "executive"]}`

How you would enforce access restrictions at query time

In order to enforce access restrictions at query time I would rely on **metadata filters**, based on the user's identity information (e.g., provided by MS SharePoint). In GroundX, the metadata filter for a query would look like:`{"roles":"director"}`

Then, once the user is authenticated (e.g. MS SharePoint), before sending the query to GroundX the system would automatically build a metadata filter based on the user attributes. This way, GroundX would

only retrieve documents that the user is authorized to see, while unauthorized documents wouldn't even be considered during retrieval. I believe this could be actually really important because it reduces the risk of leaking information because of semantic similarity.

In summary, access control should never rely on the LLM. It must be enforced before retrieval by means of metadata and user attributes filtering.

What limitations or security considerations you foresee

A key risk would be **incorrect/outdated metadata**, which could lead to an unauthorized access if the permissions are not properly synchronized. This could be prevented by reliable ingestion pipelines and periodic updates (when permissions change).

Also another risk is **metadata leaking**. This is critical when metadata itself contains sensitive information. This would be solved by minimizing sensitive metadata and avoiding returning metadata to the users.

Another concern would be **side-channel inference**, where the user doesn't have permission to check an information but deduces its existence by observing indirect clues, such as the response time or specific error messages. This would be solved by normalizing responses so that the system always appears to behave the same.

Also **auditability** would be a key consideration, as we would log user/query/document and would help to provide a trail, for example, for internal security reviews.

Question 2: Scaling RAG for Large and Dynamic Knowledge Bases

How would you architect a scalable RAG solution in the context of an AI Concierge, focusing especially on document management and user experience?

Mechanisms you would use to handle large-scale, frequently changing document sets

In order to manage dynamic large-scale document sets, I would **avoid full re-indexing**. Instead, I would use a modular architecture based on **event-driven incremental ingestion pipeline**. This would reduce cost and latency (the new information would arrive to the AI Concierge almost immediately), leading to a better user experience. In that case, a potential solution would be to version the documents and split them into chunks, assigning a unique hash to both the document and each individual chunk to detect and update only partial changes.

On the other hand, to handle scalability, a good option would be to consider a **decoupled storage strategy**. This consists in separating the storage into three sections: original storage (raw documents), metadata storage (permissions, dates..) and vector database (embeddings). This separation would ensure that the system can scale each component independently (if only documents grow, we would only scale the original storage; if only searches grow, we would only scale the vector database). It would also maintain high performance (as components are not mixed, the metadata queries would be faster) and would also facilitate audits (using the metadata database to locate and retrieve the original document from the object database)

Whether you would empower users to manage documents themselves (and if so, how and why)

Yes, I would empower users to manage documents themselves, within defined permissions (**RBAC**), mainly because they are usually the closest to the knowledge and also know best when information needs to be added,

updated or removed. It would also **avoid IT bottlenecks** (imagine the situation that every document update has to go through a technical team, then the system would become slow to maintain, with outdated information).

To ensure a seamless experience, I would provide a **simple drag-and-drop interface** integrated with existing tools (e.g. MS365), where users can upload documents without knowing technical aspects. I would also organize the system into **departmental buckets** (HR, Financial...) so each team has its own dedicated storage space. From a technical perspective, the document set management would follow the **incremental ingestion pipeline** that I have discussed in the previous section: the system detects changes via hashing and updates only those chunks that have been modified.

What would you design to keep retrieval both efficient and up-to-date (algorithms, agents..)

From an algorithmic perspective, I would implement **Semantic Chunking**. Instead of cutting chunks each a fixed number of words, it would be a "structure-aware" chunking method, which could be split by headings, sections, etc. This would permit better retrieval of information and then higher-quality answers from the LLM.

Also **Hybrid Indexing Strategy**, which leverages several retrieval views (semantic similarity, keyword / exact match, filters..) to make sure the AI finds the meaning but also specific details of the query.

Since users often ask vague questions, **Query Rewriting** could be considered, basically using an LLM-based agent to rewrite the query before searching.

Regarding automations, **TTL (Time-To-Live) Metadata Algorithm** would keep the system up to date. If the document has an "expiry_date" in its metadata, an automation could daily scan the vector database and remove documents when required. For example, it would prevent AI Concierge from giving advice based on 2023 when 2025 is available.

To conclude, regarding agents, I would implement a **Retrieval Quality Monitoring Agent**, who would basically be an agent that improves UX by observing (low satisfaction signals), deciding (retrieval is likely weak) and acting (adjust retrieval weights / re-rank / re-chunk...).