**Role Based Access Control Chatbot with LangChain + ChromaDB**

This document demonstrates the development of a chatbot that integrates **LangChain** with **Chroma DB** to enable semantic search and retrieval on the vector database.

The system incorporates **Role-Based Access Control (RBAK)** using JWT authentication, ensuring that users can only access knowledge aligned to their roles (e.g., HR, Finance, Tech, Admin).

A **FastAPI** backend manages authentication and query filtering, while a simple **Streamlit UI** provides an interactive chat interface.

Documents are pre-processed and stored as vector embeddings, allowing for **semantic search** rather than simple keyword matching. By combining vector-based retrieval with role-based filtering, the system ensures that users receive **accurate, context-aware answers** while maintaining strict access control over sensitive organizational information.

**Objectives**

**Secure Chatbot Development**

* AI chatbot that can query knowledge base documents using semantic search powered by **LangChain** and **Chroma DB**.

**Role-Based Access Control (RBAK)**

* **Role-restricted** document retrieval so that each user can only access information permitted for their role (e.g., HR, Finance, Tech, Admin).

**Authentication & Authorization**

* **JWT tokens** for mock authentication and role verification.

**Simple User Interface**

* Lightweight **Streamlit UI** for login and chatbot interaction.

**Scope**

* **Document Ingestion & Indexing**
* Ingesting sample departmental documents (HR, Finance, Tech, Employee).
* Storing document embeddings in **ChromaDB** with role-based metadata.
* **Semantic Search & Query Answering**
* Uses **LangChain RetrievalQA** to answer user queries.
* Restricting retrieval to documents allowed for the user’s role.
* **Role-Based Access Control (RBAK)**
* Implementing mock authentication with username/password.
* Assigning JWT tokens carrying role information.
* Enforcing role restrictions during query answering.
* Only admin can access every data
* **Frontend (UI)**
* A basic **Streamlit app** for login and chat interaction.
* Shows restricted responses if the user doesn’t have access.

**Tech Stack**

**Framework**: Python (LangChain)

**Database**: Chroma DB

**LLM**: Gemini(gemini-1.5-flash)

**Auth & RBAK**: FastAPI with JWT tokens for authentication

**UI**: Streamlit

**System Architecture**

**Frontend (Streamlit)**

* Provides a simple, interactive web-based UI.
* Handles login (username, password).
* Displays chatbot conversation session.
* Sends authenticated queries to backend.

**Backend (FastAPI)**

* Exposes APIs for:
  + User authentication (login → JWT token).
  + Query handling (role-based access).
* Mock Users for user credentials & roles.
* Uses Gemini LLM to generate contextual answers.
* Applies role-based filtering before query processing.

**Database Layer**

* **Chroma DB (Vector Database)**
  + Stores embeddings of documents.
  + Used for **semantic search** to retrieve most relevant chunks.

**Model & AI Layer**

* **HuggingFace Embeddings**
  + Converts text documents into vector embeddings.
* **Gemini (Google Generative AI)**
  + Generates role-specific answers using context retrieved from Chroma DB.
  + Integrated through LangChain’s **ChatGoogleGenerativeAI** with API key authentication.

**Data Preparation & Ingestion**

* **Data Preparation**
* Gathered internal enterprise documents (HR policies, Finance reports, Technical setup instructions).
* Each document tagged with metadata (e.g., role: hr, role: finance, role: tech) to enforce role-based access.
* **Ingestion Pipeline**
  + HuggingFace model sentence-transformers/all-MiniLM-L6-v2 used to transform documents into dense vector embeddings.
  + Documents and embeddings persisted in **Chroma DB** collections.
  + Role-based metadata ensures only users with matching roles can retrieve relevant documents.

**Authentication & Role-Based Access**

* **User Authentication**
  + Users log in via **Streamlit** using username and password.
  + Credentials are verified against a mock Userbase data.
  + On successful login, a **JWT (JSON Web Token)** is issued.
  + Token contains user identity + role information, used in subsequent API requests.
* **Role-Based Access Control (RBAC)**
* Each ingested document is stored in Chroma DB with metadata like role: hr, role: finance, or role: tech.
* When a user submits a question, the system filters retrieved documents based on the role in their JWT token, ensuring access only to permitted content according to their roles.

**Query Processing Flow**

* **Retrieval Stage**
* User submits a **question** from Streamlit UI → sent to FastAPI backend with role-based token.
* **Chroma DB Retriever** fetches top-k relevant documents, **filtered by role metadata**.
* **Answer Generation Stage**
* Retrieved context is passed to **Gemini LLM (via API key)**.
* Gemini generates a **role-specific response**, combining user query + retrieved knowledge from the vector database.

**Frontend (Streamlit) Interface**

* **User Interaction**
* Provides a **login form** for role-based authentication (username, password, role).
* After login, users see a **chat interface** to ask queries.
* **Chat Interface**
* Users ask role-specific questions, and responses include both Gemini LLM answers and retrieved sources from Chroma DB.
* The responses include both the **answer** from Gemini LLM and the **retrieved source documents** from Chroma DB for transparency.

**Backend (FastAPI) Services**

* **Authentication Service**
* Validates **username, password, and role** from the user against stored credentials.
* Issues a **JWT token** upon successful login for secure session handling.
* **Query Service**
* Accepts user queries and role information.
* Uses **Chroma DB retriever** (filtered by role) + **Gemini LLM** to generate context-aware answers.

**Security & Role-Based Access Control (RBAC)**

* **Role-Based Access**
* Each document ingested into **Chroma DB** carries metadata specifying the allowed role (e.g., HR, Finance, Tech).
* During query execution, the retriever enforces **role-based filters**, ensuring users only access documents aligned to their role.
* **Authentication & Authorization**
* **JWT tokens** are used to authenticate users after login.
* The token includes the user’s role, which is verified before answering queries to enforce **authorization policies**.

**Testing & Evaluation**

* **Functional Testing**
* Verified that **users can only query documents allowed for their role** (e.g., HR cannot access Finance documents).
* Checked that **Gemini LLM responses always include retrieved sources** for context validation.
* Checked that the chatbot also answers some generic question regardless of the roles are concerned.
* **Performance Testing**
* Tested retrieval speed of Chroma DB with different dataset sizes to ensure scalability.
* Evaluated response latency from Gemini API under concurrent user sessions.

**Security Considerations**

* **Authentication & Authorization**
* Implemented **Role-Based Access Control (RBAC)** to restrict access to sensitive documents by user role.
* Enforced **JWT-based authentication** to validate user sessions before granting query access.
* **Data & API Security**
* Stored the **Gemini API Key securely in environment variables (.env)** to avoid leaks.
* Enabled **secure persistence in Chroma DB** ensuring document metadata and embeddings remain isolated per role.

**Deployment Plan**

* **Backend Deployment**
* **FastAPI app** deployed on a server (e.g., AWS EC2 / GCP VM / Docker container).
* **Chroma DB persistent storage** maintained on the same server or a mounted volume for durability.
* **Frontend Deployment**
* **Streamlit app** deployed alongside the backend, allowing users to authenticate and interact with the chatbot.
* Configured **reverse proxy (e.g., Nginx)** to route requests securely between Streamlit UI, FastAPI backend, and Gemini API.

**Results & Discussion**

* **Results**
* The system successfully **restricted access to documents** based on user roles (HR, Finance, Tech).
* Queries were **answered accurately** when relevant documents existed in Chroma DB, with sources returned for transparency.
* **Discussion**
* Integration with **Gemini API** improved natural language understanding and response quality.
* The approach demonstrates that combining **vector search (ChromaDB)** with **role-based filters** ensures both security and semantic accuracy.

**Conclusion & Future Work**

* **Conclusion**
* Built a **role-based enterprise chatbot** using **ChromaDB** for semantic search and **Gemini LLM** for generating answers.
* Successfully implemented **authentication, JWT-based sessions, and RBAC**, ensuring secure and context-aware responses.
* Streamlit frontend provides a **simple, interactive interface** for users to query documents according to their role.
* **Future Work**
* **Expand document ingestion** to handle larger datasets and multiple file formats.
* Implement **advanced RBAC policies** (e.g., hierarchical roles or temporary permissions).
* Add **logging and analytics** to track query patterns and system performance.
* Explore **fine-tuning LLMs** on domain-specific documents for even more accurate responses.

---END---