Company Internal Chatbot with Role-Based Access Control (RBAC)

Project Statement

The goal of this project is to build a secure internal chatbot system that processes natural language queries and retrieves department-specific company information using Retrieval-Augmented Generation (RAG). The system will authenticate users, assign roles (Finance, Marketing, HR, Engineering, C-Level, Employees), and provide role-based access to company documents stored in a vector database. Users will input queries and receive context-rich, sourced responses restricted by their role permissions. All documents for RAG are provided via GitHub repository: https://github.com/springboardmentor441p-coderr/Fintech-data

Outcomes

- Extract, preprocess, and index company documents (markdown and CSV) into a vector database with role-based metadata tags.
- Implement secure user authentication and role-based access control (RBAC) middleware.
- Build a RAG pipeline integrating semantic search with free LLMs (OpenAI GPT or open-source alternatives) to generate evidence-based responses.
- Enforce role-based data access: Finance users see finance docs, Marketing sees marketing docs, C-Level sees all. Develop a Streamlit web interface for user login, chat interaction, and role-specific information retrieval. Deploy a complete, fully documented system on GitHub using only free and open-source tools.

Milestones

Milestone	Weeks	Focus
1: Data Preparation & Vector DB	1–2	Parse documents, generate embeddings, index into vector store
2: Backend Auth & Search	3–4	User authentication, RBAC middleware, role-filtered search
3: RAG Pipeline & LLM	5–6	Integrate LLM, build RAG pipeline with source attribution
4: Frontend & Deployment	7–8	Streamlit UI, system integration, testing, documentation

Milestone 1: Data Preparation & Vector DB (Weeks 1–2)

Module 1: Environment Setup & Data Exploration (Week 1)

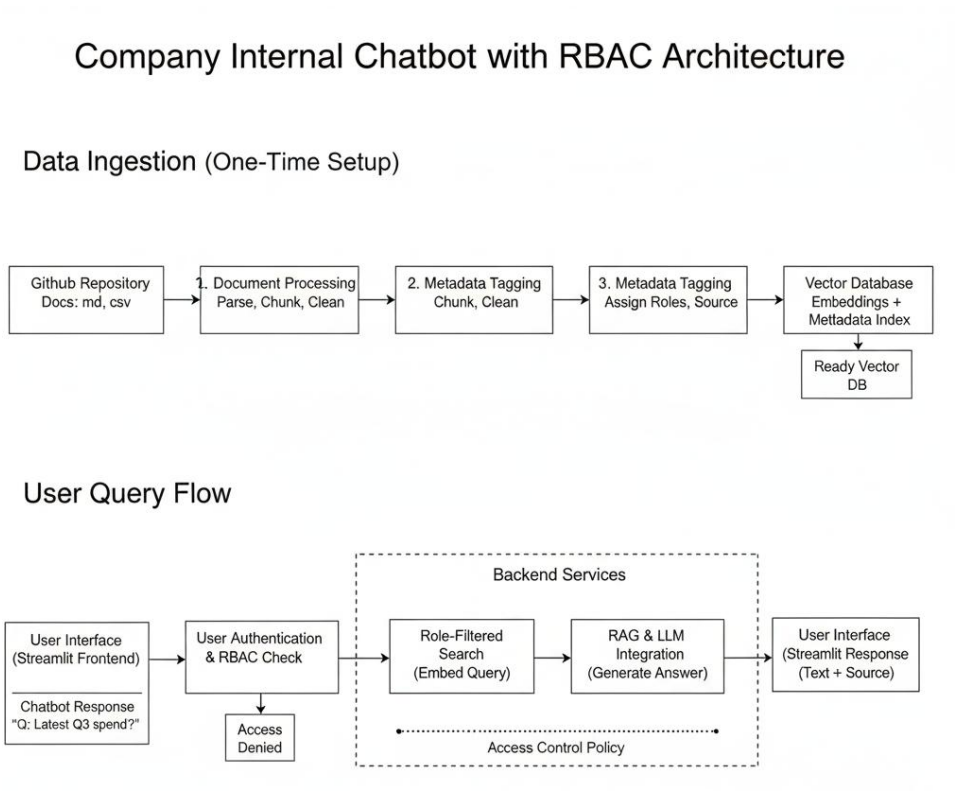
Objective: Configure dev environment, clone GitHub repo with RAG documents, explore company documents, map roles to documents.

Tasks:

- Set up Python virtual environment; install FastAPI, Streamlit, LangChain, sentence-transformers, pandas Clone GitHub repository containing RAG documents and starter code templates
- Explore all provided documents (markdown and CSV formats)
- Understand document content and structure
- Create role-to-document mapping: Finance → financial reports; Marketing → marketing reports; HR → employee data; Engineering → tech docs; C-Level → all docs; Employees → general handbook

Deliverables:

- Configured Python environment with dependencies
- Project folder structure initialized on local system
- Role-document mapping documentation
- Data exploration and content summary report



Module 2: Document Preprocessing & Metadata Tagging (Week 2)

Objective: Parse all documents from GitHub, clean text, chunk into segments, assign role-based metadata tags.

Tasks:

- Parse markdown and CSV documents from GitHub repository
- Extract titles, sections, and content
- Clean text: normalize whitespace, remove special characters, handle encoding issues
- Chunk documents into 300–512 token segments with sequential identifiers
- Assign role-based metadata: source document, department, accessible roles
- Create metadata mapping linking each chunk to role permissions

Deliverables:

- Preprocessing and data ingestion module
- Cleaned and formatted document chunks
- Role-based metadata mapping document
- Preprocessing validation and quality assurance report

Milestone 2: Backend Auth & Search (Weeks 3–4)

Module 3: Vector Database & Embedding Generation (Week 3)

Objective: Generate embeddings for all chunks from GitHub documents, index into vector DB, enable semantic search.

Tasks:

- Select and download embedding model (sentence-transformers/all-MiniLM-L6-v2)
- Generate vector embeddings for all document chunks
- Initialize vector database (Chroma or Qdrant - free tier)
- Index embeddings with comprehensive metadata
- Implement semantic search functionality

Deliverables:

- Embedding generation module
- Populated vector database with indexed documents
- Semantic search functionality and query interface
- Search quality and performance benchmarking report

Module 4: Role-Based Search & Query Processing (Week 4)

Objective: Implement RBAC filtering at search level; ensure role-based data access enforcement.

Tasks:

- Build RBAC filtering logic for document access based on user roles
- Implement role hierarchy: C-Level access > department staff access > general employee access Preprocess and normalize incoming queries
- Select most relevant document chunks for each query
- Test and validate role-based access: verify Finance users cannot access HR documents, etc.

Deliverables:

- Role-based access control filtering module
- Query processing and normalization utilities
- Role permission configuration and hierarchy definition
- Role-based access validation test suite and results

Milestone 3: RAG Pipeline & LLM (Weeks 5–6)

Module 5: User Authentication & RBAC Middleware (Week 5)

Objective: Implement FastAPI backend with secure authentication, JWT tokens, and RBAC enforcement.

Tasks:

- Initialize FastAPI backend application with middleware support
- Set up user data storage (SQLite database)
- Implement JWT-based authentication and token management
- Create authentication endpoints for user login and session management
- Build RBAC middleware to enforce role-based access on protected endpoints
- Implement access audit logging

Deliverables:

- FastAPI backend application
- User authentication and JWT implementation
- RBAC middleware and permission verification
- User database with sample accounts
- Authentication and authorization test cases

Module 6: RAG Pipeline & LLM Integration (Week 6)

Objective: Integrate free LLM, build RAG pipeline with retrieved documents, generate responses with source attribution.

Tasks:

- Select and integrate free LLM (OpenAI GPT free trial, HuggingFace, or open-source)
- Design system prompts and context templates
- Implement RAG pipeline: authenticate user → filter by role → retrieve relevant chunks → augment prompt → generate response
- Add source citation and document attribution to responses
- Implement confidence scoring based on retrieval relevance

Deliverables:

- LLM integration and API management module
- RAG pipeline implementation
- Prompt templates and augmentation logic
- Source attribution and citation system
- RAG functionality test cases with sample queries

Milestone 4: Frontend & Deployment (Weeks 7–8)

Module 7: Streamlit Frontend Development (Week 7)

Objective: Build interactive chat UI with login, conversation interface, and source document display.

Tasks:

- Design and implement Streamlit application interface
- Create user authentication and login interface
- Build chat message display and input components

- Display user role and department information
- Show source documents and citations for transparency
- Integrate frontend with backend API

Deliverables:

- Streamlit frontend application
- API client for backend communication
- Login and authentication interface
- Chat interaction components
- User guide documentation for each role type

Module 8: System Integration, Testing & Deployment (Week 8)

Objective: Complete end-to-end system testing, performance optimization, and deployment preparation.

Tasks:

- Test complete workflow for all user roles
- Verify role-based access enforcement and data isolation
- Test error handling and edge cases
- Measure and optimize performance metrics
- Write comprehensive system documentation
- Prepare deployment package and GitHub repository

Deliverables:

- Integration test suite with comprehensive test coverage
- System architecture and technical documentation
- API specification and endpoint reference
- User guide for each role and use case
- Deployment and setup guide
- Performance and security testing report
- Demo video showcasing system features
- Production-ready GitHub repository with complete documentation

Evaluation Criteria

Milestone	Metric	Target
1	Document parsing and metadata accuracy	100% of documents parsed; accurate role mapping
2	Role-based access and search quality	Zero unauthorized data access; retrieval latency < 500ms
3	Authentication and RAG functionality	Secure authentication; end-to-end response < 3s

4	Frontend usability and deployment readiness	Intuitive interface; complete documentation; working demo

Free Tech Stack

Component	Technology
Backend	FastAPI, Python 3.8+
Frontend	Streamlit
Vector Database	Chroma or Qdrant (free)
Embeddings	Sentence Transformers
LLM	OpenAI GPT (free trial) or HuggingFace/LLaMA
Database	SQLite
Authentication	PyJWT
Version Control	GitHub

Data Sources

All RAG documents are provided in GitHub repository with the following structure:

- Finance documents (quarterly reports, financial summaries)
- Marketing documents (campaign reports, market analysis)
- HR documents (employee data, handbook, policies)
- Engineering documents (technical architecture, processes)
- General documents (employee handbook, company policies)