Website Chatbot

GitHub Link:

https://github.com/sikri2408/Website_chatbot

Solution Approach:

The solution consists of several key components:

1. **Authentication Service:** The `auth_service.py` module handles API key and client ID authentication for the system. It defines an `AuthService` class that manages credential validation and provides a `get_api_key` dependency function for the FastAPI routes.

2. **Vector Store Manager:** The `index_service.py` module contains the `VectorStoreManager` class, which is responsible for indexing content from URLs in a vector store (using the Chroma library) and providing search capabilities.

3. **RAG Service:** The `rag_service.py` module defines the `RAGService` class, which is the main entry point for processing chat queries. It uses the `VectorStoreManager` to retrieve relevant context and then employs a custom Retrieval-Augmented Generation chain to generate responses.

4. **FastAPI:** The `main.py` module sets up the FastAPI application, including routes for indexing URLs, processing chat queries, and retrieving collection statistics. It utilizes the authentication and RAG service components.

5. **Streamlit UI:** The `streamlit_ui.py` module provides a web-based user interface built with Streamlit, allowing users to authenticate, submit URLs for indexing, and chat with the RAG system.

Challenges:

1. **Fine-Tuning Prompt**: Ensuring accurate citations, restricting hallucinations, and providing a default response when no relevant information is found in the context.

2. **Retrieval-Augmented Generation:** Tweaking indexing parameters like chunk size, chunk overlap, similarity score threshold, and chunk count for better quality responses.

3. **Authentication Integration**: Carefully integrating the authentication service with the API and UI components to ensure a seamless and secure authentication flow.

4. **User Interface Design**: Designing a user-friendly and intuitive Streamlit interface with features like persistent chat history, source viewing, and clear authentication handling.

5. **API Gateway Integration**: Exposing the Streamlit app endpoint through AWS API Gateway was challenging, leading to the decision to expose the public IP of the EC2 instance.

Potential Improvements:

1. Better Authentication Mechanism: Implementing more robust, standards-based authentication like OAuth 2.0 or JSON Web Tokens (JWT) for improved security, access control, and scalability.

2. Deploying Streamlit UI via API-Gateway: Deploying the Streamlit-based user interface through AWS API Gateway would allow for better scalability, monitoring, and integration with other AWS services.

3. Scalability and Performance: Evaluating the system's scalability and performance as the volume of indexed content grows, potentially considering techniques like sharding, distributed processing, or more efficient indexing approaches.

4. Error Handling and Logging: Implementing more robust error handling and logging mechanisms to provide better visibility into system failures and facilitate troubleshooting.

5. Additional Features: Exploring feature enhancements like support for multimedia content, multi-modal interaction, personalized recommendations, and integration with external knowledge sources.

6. Security and Privacy: Implementing role-based access controls and secure data storage for credentials, depending on the specific use case and privacy requirements.

Tools Used:

1. LangChain: A framework for building applications with large language models.

2. ChromaDB: A vector store for efficient retrieval of indexed content.

3. OpenAI: The language model used for generating responses.

4. Streamlit: A framework for building the interactive web-based user interface.

5. FastAPI: A web framework used for building the backend API.