**Product Help Bot**

**1. Overview**

The Product Help Bot system provides a chatbot interface for users to ask questions and receive responses. The system integrates a frontend application using Streamlit and a backend API using FastAPI. The backend leverages RAG (Retrieval-Augmented Generation) techniques for understanding user queries and generating appropriate responses.

**2. Architecture Components**

**a. Frontend: Streamlit Application**

• **Purpose**: Provides the user interface for interacting with the chatbot.

• **Components**:

• **Chat Interface**: Allows users to input questions and view responses.

• **Feedback Mechanism**: Collects user feedback on the responses provided.

• **Technology**: Streamlit for quick development of interactive web applications.

**b. Backend: FastAPI Application**

• **Purpose**: Handles user requests, processes queries, and returns responses.

• **Components**:

• **API Endpoints**:

• /get-response/: Receives user queries, processes them, and returns responses.

• /submit-feedback/: Collects user feedback on the responses.

• **Query Processing**:

• **Text Embeddings**: Utilizes SentenceTransformer (all-MiniLM-L6-v2) to convert user queries into embeddings.

• **Document Retrieval**: Uses pgvector in PostgreSQL to perform similarity searches on stored document embeddings.

• **Generative AI**: Utilizes “Claude Sonnet” via AWS Bedrock for generating responses based on the retrieved context.

• **LangChain Framework**: Orchestrates the retrieval and generative components, ensuring seamless integration between them.

• **Technology**: FastAPI for building and serving APIs, PostgreSQL with pgvector for vector storage and search, AWS Bedrock for deploying the Claude Sonnet model.

**c. Database: PostgreSQL with pgvector**

• **Purpose**: Stores document data and embeddings for similarity searches.

• **Components**:

• **Document Table**: Stores text data and corresponding vector embeddings.

• **Feedback Table** (optional): Stores user feedback on responses.

• **Technology**: PostgreSQL for relational data storage and pgvector extension for vector operations.

**3. Functional Features**

• **User Interaction**:

• Users can input questions through the Streamlit interface.

• Users receive AI-generated responses in real-time.

• Users can provide feedback on the responses.

• **Data Management**:

• Store and retrieve documents with associated embeddings in PostgreSQL.

• Perform efficient similarity searches using vector data.

• Store user feedback for future analysis and improvement.

• **Response Generation**:

• Process user queries using RAG technique.

• Retrieve relevant documents based on query embeddings.

• Use Claude Sonnet via AWS Bedrock to generate context-aware responses.

**4. Future Considerations**

• **Reading Images and Smart Document Storage**: Implement functionality for reading and processing images and storing various document types (text, images, PDFs) intelligently.

• **Supporting Different Document Types**: Extend the system to handle and process different document types beyond plain text, including PDFs, images, and other formats.

• **Semantic Search Threshold**: Implement a configurable threshold for semantic search similarity to improve result relevance.

• **Agents for Web Searches**: Integrate agents that can perform web searches to fetch additional information when the internal database lacks sufficient answers.