**Documentation Report: Dynamic Knowledge Base Chatbot**

**1. Project Overview**

The Dynamic Knowledge Base Chatbot is an intelligent system designed to provide informative, context-aware responses by leveraging a continuously expanding knowledge base. The system's core strength is its **dynamic nature**, allowing for both manual addition of knowledge and automated, periodic updates from configured web data sources.

The project provides two distinct implementations:

1. **app.py (Advanced Model):** Uses **Sentence-Transformers** and **FAISS** for highly accurate semantic search.
2. **app\_lightweight.py (Baseline Model):** Uses the simpler **TF-IDF** and **Cosine Similarity** for a resource-efficient alternative.

The user interface is a web application built with **Gradio**, featuring dedicated tabs for Chat, Adding Knowledge, Configuring Data Sources, and viewing Performance Statistics.

**2. Core Architecture and Components**

The system's logic is modularized across several key Python classes and files:

**2.1. Knowledge Management**

| Component | File | Core Technology | Function |
| --- | --- | --- | --- |
| **Dynamic KnowledgeBase** | app.py | **SentenceTransformer** (all-MiniLM-L6-v2), **FAISS** | Manages the document corpus, generates embeddings, and handles high-speed vector retrieval. |
| **Lightweight KnowledgeBase** | app\_lightweight.py | **TF-IDF**, **Numpy** | Manages documents and performs retrieval using term frequency-based vectors and cosine similarity. |

**2.2. Data Ingestion & Scheduling**

| Component | Files | Core Technology | Function |
| --- | --- | --- | --- |
| **Data Source Manager** | app.py, app\_lightweight.py | requests, BeautifulSoup, threading, time | Handles fetching data from external URLs, cleans HTML, chunks text, and schedules periodic updates. |

**2.3. User Interface**

| Component | File | Core Technology | Function |
| --- | --- | --- | --- |
| **Gradio Interface** | app.py, app\_lightweight.py | **Gradio** | Provides the interactive web UI for all user interactions (chat, knowledge addition, source configuration). |

**3. Data Flow and Knowledge Ingestion Pipeline**

New knowledge (from a manual text input or an external web URL) is processed through a multi-step pipeline before being indexed for search:

1. **Data Ingestion:** Fetches raw data from the configured URL using the requests library.
2. **Noise Removal:** Uses **BeautifulSoup** to parse the raw HTML and removes irrelevant tags (<script>, <style>, <nav>, <footer>) to isolate clean content.
3. **Text Chunking:** The cleaned text is split into smaller, coherent chunks (e.g., minimum character length > 50-100) to ensure high-quality retrieval.
4. **Embedding/Vectorization:**
   * **Advanced Model (app.py):** **Sentence-Transformers** convert chunks into **dense vector embeddings**.
   * **Baseline Model (app\_lightweight.py):** **TF-IDF Vectorizer** converts chunks into **sparse vectors**.
5. **Indexing:** The vectors are stored in an optimized index for quick retrieval.
   * **Advanced Model:** **FAISS** (Facebook AI Similarity Search) index is used.
   * **Baseline Model:** Vectors are stored in a NumPy array.
6. **Persistence:** The knowledge base state, including the FAISS index (knowledge\_base.index), documents, and metadata, is saved to disk to ensure data persistence across application restarts.

**4. Model Comparison: Retrieval Mechanism**

The project highlights two distinct retrieval strategies, demonstrating the trade-off between performance and resource consumption.

| Feature | **TF-IDF (Baseline Model - app\_lightweight.py)** | **Sentence-Transformers + FAISS (Advanced Model - app.py)** |
| --- | --- | --- |
| **Vector Type** | **Sparse Vectors** (based on word frequency) | **Dense Vectors** (semantic embeddings) |
| **Core Capability** | **Keyword-Based Matching** (Identifying important keywords) | **Semantic Similarity** (Understanding context and meaning) |
| **Retrieval Speed** | **Fast**, especially for smaller corpuses. | **Extremely Fast** due to FAISS optimization. |
| **Accuracy / Context** | **Lower**. Lacks true semantic understanding; fails when keywords are absent but context is similar. | **Superior**. Can match documents with a query even if they don't share exact words. |
| **Resource Needs** | **Low** (Simple statistical model). | **High** (Requires loading a deep learning model, MB). |

The main application (app.py) utilizes the advanced model for its superior ability to handle complex, natural language queries, operating on a **Retrieval Augmented Generation (RAG)** principle by providing up to relevant documents as context.

**5. Dependencies**

The required software packages are detailed in requirements.txt:

| Package | Version | Purpose |
| --- | --- | --- |
| gradio | 4.19.2 | Primary library for the web-based user interface. |
| sentence-transformers | 2.3.1 | Generates dense vector embeddings for semantic search. |
| faiss-cpu | 1.7.4 | High-performance library for similarity searching on vector sets. |
| numpy | 1.24.3 | Fundamental library for numerical operations, array storage, and vector manipulation. |
| requests | 2.31.0 | Used for making HTTP requests to fetch data from configured URLs. |
| beautifulsoup4 | 4.12.3 | Used for parsing HTML and extracting clean text. |
| torch | 2.1.2 | Underlying deep learning framework for Sentence-Transformers. |
| transformers | 4.36.2 | Provides access to the transformer model architecture. |
| schedule | 1.2.0 | Used for scheduling the periodic automated updates from data sources. |