**Jupiter FAQ Bot Architecture Documentation**

**Overview**

The Jupiter FAQ Bot is a Flask-based web application designed to provide instant answers to user queries about Jupiter's financial services. It leverages a combination of natural language processing (NLP), vector search, and large language models (LLMs) to deliver accurate and conversational responses. The system is built to handle FAQ data, preprocess it, and serve answers through a web interface and API endpoints.

**Architecture**

**1. Web Crawler**

* **Purpose**: Extracts FAQs from Jupiter's website to populate the FAQ dataset.
* **Details**:
  + Uses a breadth-first search (BFS) to crawl up to a configurable number of pages (max\_pages=150).
  + Avoids non-HTML content or restricted URLs (e.g., PDFs, admin pages).
  + Employs multiple extraction strategies: FAQ-specific divs, accordion-style FAQs, text patterns, and heading-based extraction.
  + Saves FAQs incrementally to faqs\_2.json to prevent data loss.
* **Implementation**: FAQCrawler class in crawler.py, using requests and BeautifulSoup.

**2. Data Preprocessing**

* **Purpose**: Cleans and categorizes FAQ data for efficient storage and retrieval.
* **Components**:
  + **Cleaning**: Removes HTML tags, extra whitespace, and special characters using BeautifulSoup and regex.
  + **Normalization**: Lowercases text, removes stopwords, and lemmatizes using NLTK.
  + **Categorization**: Assigns categories (e.g., KYC, Rewards, Payments) based on keyword matching with fuzzywuzzy.
  + **Deduplication**: Removes duplicate questions using fuzzy matching (fuzz.ratio > 90).
* **Implementation**: Handled in data.py, which processes raw FAQ data (faqs\_2.json) into processed\_faq.json and cleaned\_faq.json.

**3. Embedding Model**

* **Model**: sentence-transformers/all-MiniLM-L6-v2 (default) or ONNX-based ONNXMiniLM\_L6\_V2 (configurable via USE\_ONNX environment variable).
* **Purpose**: Converts FAQ questions and user queries into dense vector representations for similarity search.
* **Details**:
  + The all-MiniLM-L6-v2 model is a lightweight, high-performance sentence transformer optimized for semantic similarity tasks.
  + ONNX support allows for optimized inference on CPU, reducing latency in resource-constrained environments.
  + Embeddings are normalized to ensure consistent cosine similarity calculations.
* **Implementation**: Uses HuggingFaceEmbeddings from the LangChain library or ONNXMiniLM\_L6\_V2 from ChromaDB, depending on configuration.

**4. Vector Database**

* **Database**: ChromaDB
* **Purpose**: Stores FAQ question embeddings and their associated metadata (answers, categories, source URLs) for efficient similarity search.
* **Details**:
  + Persistent storage is enabled with a directory (chroma\_db) to cache embeddings and avoid recomputation.
  + FAQs are stored as documents with questions as the primary text and metadata containing answers, categories, and source URLs.
  + Similarity search uses cosine similarity to retrieve the most relevant FAQ for a user query.
  + Configurable collection name (faq\_collection) and persistence directory.
* **Implementation**: Uses LangChain's Chroma integration for vector storage and retrieval.

**5. Large Language Model (LLM)**

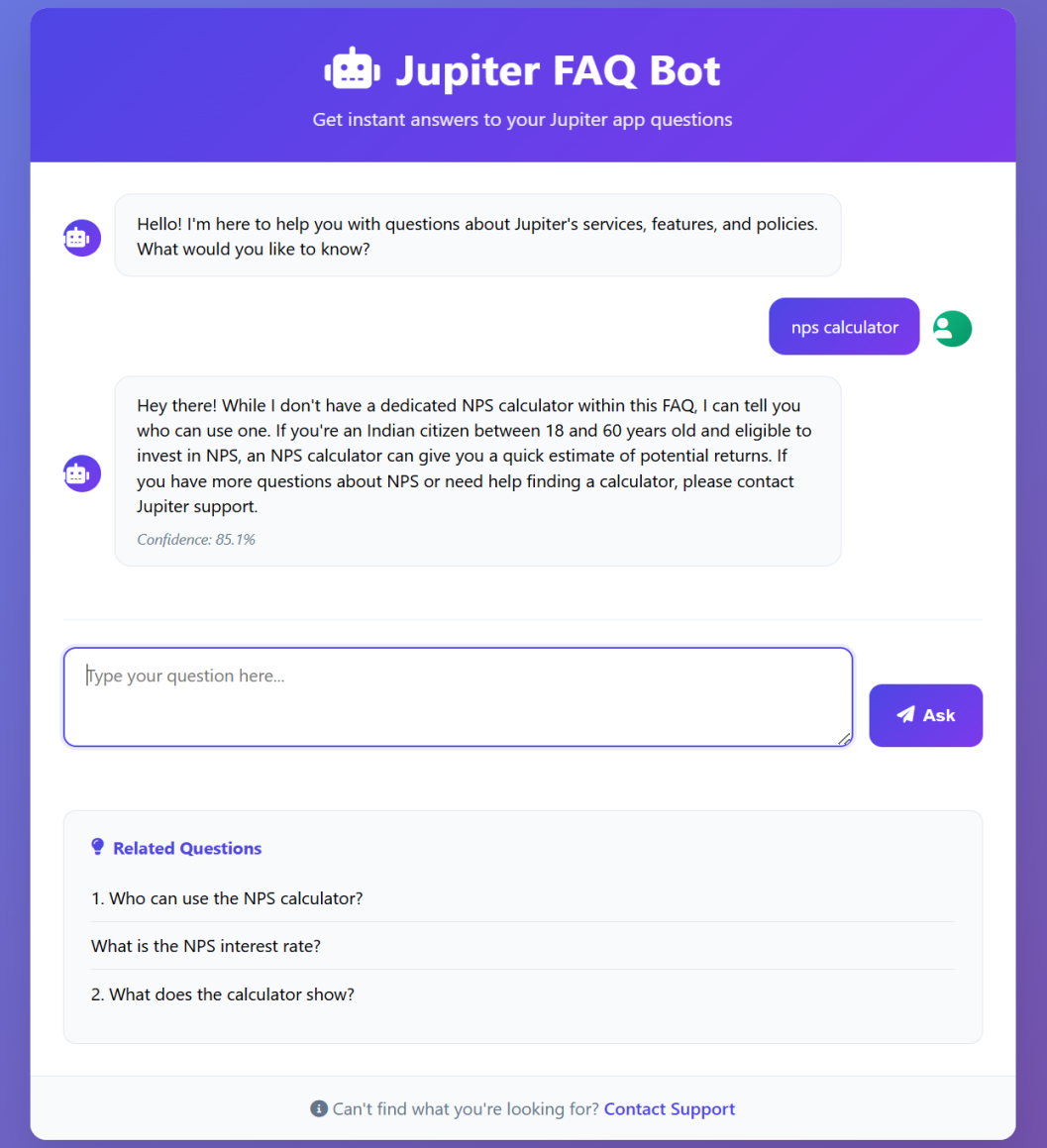
* **Model**: Google Gemini 1.5 Flash (gemini-1.5-flash)
* **Purpose**: Generates conversational responses by rephrasing retrieved FAQ answers.
* **Details**:
  + Configurable parameters: temperature (0.7), max tokens (500).
  + Requires a Google API key (GOOGLE\_API\_KEY) for authentication.
  + Uses a prompt template to ensure responses are friendly, concise, and contextually appropriate.
* **Implementation**: Integrated via LangChain's GoogleGenerativeAI module.

**6. Fallback Logic**

* **Purpose**: Handles cases where no relevant FAQ is found or the similarity score is below the threshold.
* **Details**:
  + If the similarity score is below the configured threshold (0.7 by default), the system returns a fallback response:  
    *"I don't have a specific answer for that question. Please check the Jupiter Help Centre or contact our support team for assistance."*
  + If the ChromaDB is unavailable or no FAQs are loaded, it returns:  
    *"FAQ service is currently unavailable. Please contact support."*
  + For empty or invalid queries, it returns:  
    *"Please enter a question."*
  + Errors during processing trigger a generic error response:  
    *"I'm experiencing technical difficulties. Please try again or contact support."*
* **Implementation**: Handled in the FAQBot.answer\_question method with appropriate error logging.

**7. Web Application**

* **Framework**: Flask
* **Endpoints**:
  + /: Renders the main FAQ interface (index.html).
  + /ask: Handles POST requests for user questions (supports both form and JSON inputs).
  + /api/ask: JSON-only API endpoint for programmatic access.
  + /evaluate: Testing endpoint that runs predefined queries and returns results.
  + /health: Health check endpoint to verify service status and FAQ count.
* **Error Handling**:
  + 404: Returns JSON error for invalid endpoints.
  + 500: Returns JSON error for internal server issues.
* **Frontend**: A responsive HTML/CSS/JavaScript interface with a chat-like UI, loading animations, and related question suggestions.



**Design Decisions**

1. **Embedding Model Choice**:
   * all-MiniLM-L6-v2 was chosen for its balance of performance and efficiency, suitable for FAQ similarity tasks.
   * ONNX support was added for environments requiring optimized inference on CPU.
2. **Vector Database Selection**:
   * ChromaDB was selected for its simplicity, persistence, and integration with LangChain, making it ideal for small to medium-sized FAQ datasets.
   * Persistent storage ensures embeddings are reused across sessions, reducing startup time.
3. **LLM Integration**:
   * Gemini 1.5 Flash was chosen for its cost-effectiveness and ability to generate conversational responses.
   * A prompt template ensures responses are user-friendly and aligned with Jupiter's branding.
4. **Fallback Logic**:
   * A similarity threshold (0.7) ensures only relevant answers are returned, with fallbacks for low-confidence matches.
   * Multiple fallback messages provide clear guidance for users when answers are unavailable.
5. **Preprocessing and Categorization**:
   * NLTK and fuzzy matching were used for robust text preprocessing and categorization, ensuring clean and organized FAQ data.
   * Deduplication prevents redundant FAQs, improving search efficiency.
6. **Web Crawler Design**:
   * BFS ensures comprehensive coverage of the website while respecting crawl limits.
   * Multiple extraction strategies handle diverse FAQ formats, increasing robustness.
   * Real-time saving of FAQs prevents data loss during long crawls.
7. **Flask Application**:
   * Flask was chosen for its lightweight nature and ease of integration with Python-based NLP tools.
   * Separate /ask and /api/ask endpoints support both web and programmatic access.
   * Health and evaluation endpoints facilitate monitoring and testing.

**How to Run**

**Prerequisites:**

* Python 3.8+
* Required packages (install via pip install -r requirements.txt):
  + flask, langchain, langchain-community, langchain-google-genai
  + chromadb, sentence-transformers, onnx (if using ONNX)
  + nltk, fuzzywuzzy, pandas, beautifulsoup4, requests
* Google API key (GOOGLE\_API\_KEY) set in a .env file
* FAQ data file (cleaned\_faq.json)

**Steps:**

1. **Clone the Repository**:
2. git clone <repository-url>
3. cd <repository-directory>
4. **Set Up Environment**:

Create a .env file:

* + GOOGLE\_API\_KEY=<your-google-api-key>
  + FAQ\_FILE\_PATH=cleaned\_faq.json
  + USE\_ONNX=false
  + SIMILARITY\_THRESHOLD=0.7

Install dependencies:

* + *pip install -r requirements.txt*

1. **Crawl FAQs (Optional)**:
   * Run the crawler to generate faqs\_2.json: *python crawler.py*
   * Preprocess the data: *python data.py*
2. **Run the Flask Application**:

*python app.py*

* + The application will be available at <http://localhost:5000>.

1. **Access the Application**:
   * Open http://localhost:5000 in a browser for the web interface.
   * Use the /api/ask endpoint for programmatic access (e.g., via curl or Postman):
   * curl -X POST -H "Content-Type: application/json" -d '{"question":"What are the fees for the Edge card?"}' <http://localhost:5000/api/ask>
2. **Verify Health**:
   * Check the service status:
   * curl <http://localhost:5000/health>

**Limitations:**

* + The all-MiniLM-L6-v2 model may struggle with highly nuanced or context-dependent questions due to its lightweight nature.
  + ChromaDB is optimized for small to medium datasets. Large FAQ datasets may require a more scalable solution (e.g., Pinecone or Weaviate).
  + Persistence relies on local storage, which may not be ideal for distributed deployments.