Zomato RAG Chatbot

For this project, I designed and implemented a complete Retrieval-Augmented Generation (RAG)-based chatbot using generative AI to improve user interaction with restaurant data, especially for a platform like Zomato. The idea was to allow users to ask natural language queries (like "What are the best vegetarian dishes under ₹300?") and get accurate, real-time responses drawn from a structured knowledge base. I ensured the chatbot could run efficiently on CPU-only systems, making it easy to deploy even on basic machines.

System Architecture and File-Level Breakdown

1. Web Scraping

To build the foundation of the knowledge base, I created three scrapers:

- `info_scraper.py`: This script gathers basic restaurant-level metadata such as the restaurant's name, address, contact number, and operating hours.

- `menu_scraper.py`: This is the most detailed scraper—it goes through each restaurant's menu to extract dish names, prices, descriptions, and tags like "veg" or "spicy."

- `review_scraper.py`: This collects user reviews which help us later during response generation by adding sentiment-based recommendations.

All scrapers use `BeautifulSoup` and rotate through random user-agent headers to prevent blocking. The scraped data is saved in structured JSON format under the `scraped_data/` directory. Each scraper is resilient to layout changes and missing fields, thanks to fallback CSS selectors and exception handling.

2. Data Enhancement and Knowledge Base Construction

Once I collected the raw data, I cleaned and structured it through:

- `enhance_menu_data.py`: This script performs data cleaning and feature engineering. I used keyword-based rules to tag dishes as "vegetarian," "gluten-free," etc., based on the menu description.

- `build_knowledge_base.py`: Here, I combined outputs from all scrapers into clean, normalized documents—each representing one restaurant. These documents are saved in the `knowledge_base/` directory. Each JSON document includes keys like `name`, `location`, `menu`, `features`, `hours`, `contact`, and `reviews`.

3. Embedding Creation and Semantic Indexing

This is a crucial step where I implemented semantic search:

- File: `create_embeddings.py`
- I loaded each restaurant document and passed it through the `sentence-transformers` library using the `all-MiniLM-L6-v2` model to generate 384-dimensional embeddings.
- These embeddings are stored in a FAISS index (`faiss_index.index`) which allows for fast and efficient similarity searches.
- Each embedding is stored along with its associated metadata like restaurant name and dish category, making it easy to filter later.

4. Chatbot with Retrieval-Augmented Generation (RAG)

The core chatbot logic is handled in:

- `rag_chatbot.py`

This file contains the full flow:

1. User submits a query.
2. The query is embedded using the same MiniLM model.
3. I use FAISS to retrieve the top-k most relevant restaurant documents.
4. The context and query are then passed to the `Flan-T5-base` model from HuggingFace.
5. The model generates a coherent, context-aware response.
6. I've also added basic session memory to allow for follow-up questions.

Fallbacks are in place in case no relevant documents are found, ensuring the user always receives a helpful message.

5. Streamlit-Based User Interface

To make the project interactive, I created a frontend using Streamlit in `app.py`. This provides:

- A simple interface for users to enter queries.
- Real-time streaming responses.

  All processing is done server-side, and users don't need to install anything beyond running the Streamlit app.

Dataset

- Raw scraped data is saved under: `Menu,Review folder and Restaurants.csv`
- Processed knowledge base documents are in: `knowledge_base`
- The FAISS semantic index is stored as: `faiss_index.index`

Challenges I Faced and How I Solved Them

- **Dynamic web pages**: I handled layout changes using fallback selectors and error checks in my scrapers.

- **Menu classification**: Since not all dishes clearly mention tags like “veg” or “gluten-free,” I built a keyword-based heuristic model for classification.

- **Speed and accuracy**: To ensure fast responses even on CPU, I used lightweight yet powerful models—MiniLM for embeddings and Flan-T5 for generation.

Future Work and Ideas

Here’s what I’m planning to add next:

- Use hybrid retrieval methods by combining TF-IDF and semantic embeddings for better recall.

- Make the FAISS index dynamically update when restaurant menus change.

- Personalize recommendations using user history or preferences.