RAG DM Bot — FastAPI + SQLite + FAISS + OpenRouter LLM

A **Retrieval-Augmented Generation (RAG)** system that answers **cosmetic and beauty product** queries using **FastAPI**, **SQLite**, **FAISS**, and a **free OpenRouter LLM (DeepSeek)**. The system leverages **semantic search** and **keyword search** to retrieve relevant product information from a database, then uses **LLM (DeepSeek)** to generate human-like responses in **Persian** (فارسی).

---

1. Project Overview

Introduction

This project is designed to provide a **conversational AI** that answers **beauty-related queries**. It uses a **RAG-based hybrid approach** to combine **semantic search** (via **FAISS**) with **keyword-based retrieval** (via **FTS5**) in **SQLite**. Once the relevant information is retrieved, an **OpenRouter LLM (DeepSeek)** model is used to generate responses.

Goal

The main goal of this project is to provide a chatbot capable of answering user queries related to **cosmetic and beauty products**, leveraging a robust retrieval system (FAISS + SQLite) and the power of **DeepSeek LLM** for generating human-like, contextually relevant responses.

Key Technologies

- **FastAPI**: Web framework used to build the backend API.
- **SQLite**: A lightweight relational database to store beauty product information.
- **FAISS**: A vector search library used for efficient semantic search.
- **OpenRouter LLM (DeepSeek)**: A pre-trained model used for generating human-like responses.
- **FTS5 (Full-Text Search)**: A search extension in SQLite for keyword-based retrieval.

---

2. Features

Retrieval

- **FAISS** for **semantic search** that allows the system to understand the meaning of the user's query and retrieve related products.
- **FTS5** in SQLite for **keyword-based retrieval** to match specific product names and descriptions.

LLM Integration

- The system integrates with **OpenRouter's DeepSeek LLM** to generate natural, contextual responses based on the retrieved data.

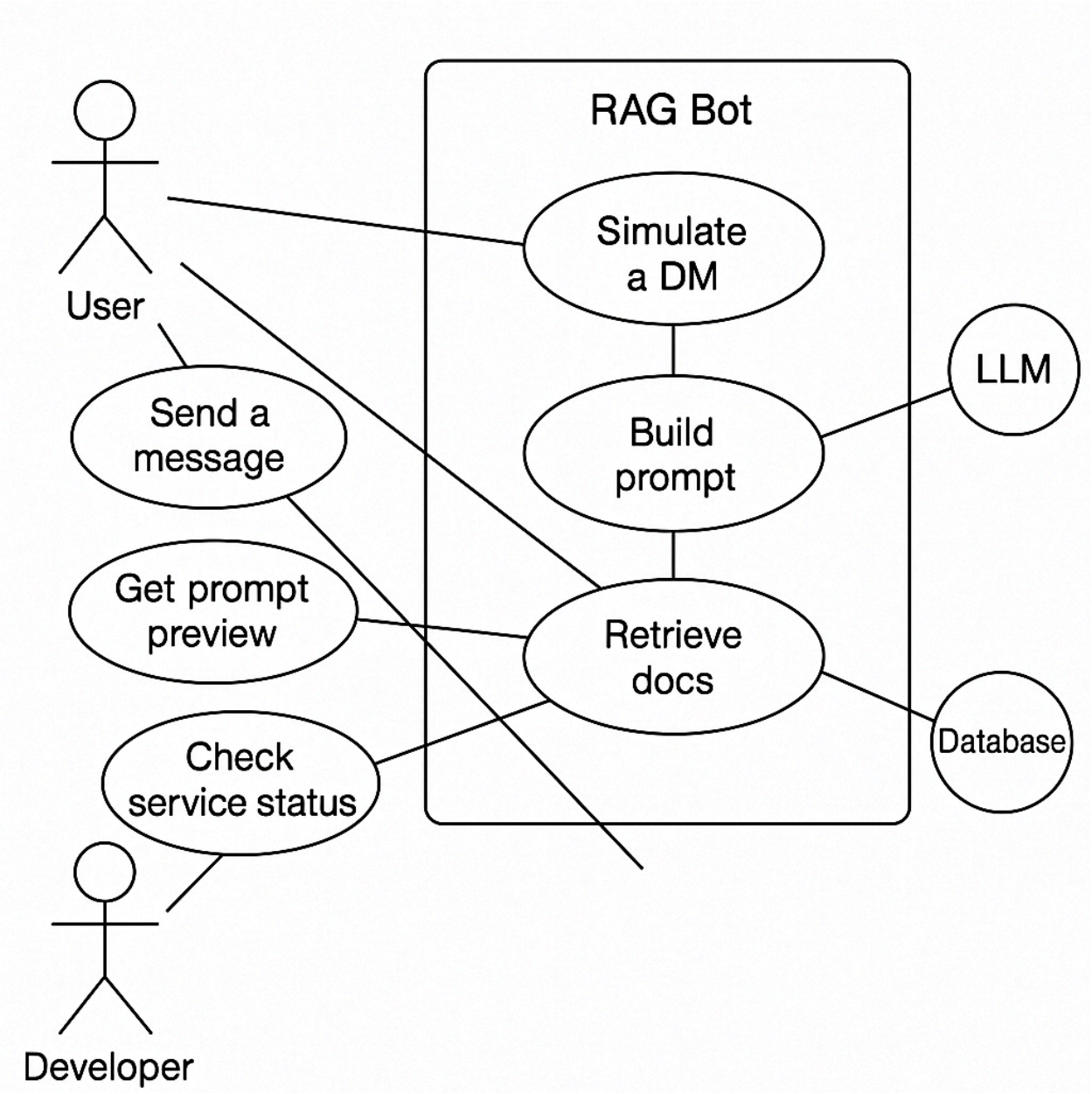
API Endpoints

- `/simulate_dm`: The primary endpoint where users send queries and receive responses.
- `/health`: An endpoint that checks the health of the system.
- `/metrics`: Provides system performance metrics, such as request counts, error counts, and latency.
- `/feedback`: Allows users to submit feedback on the responses they received.

Security

- **API Key Authentication**: Secure access to the API using an API key.
- **Rate-Limiting**: Prevents abuse of the API by limiting the number of requests a user can make.
- **Redaction**: Ensures that sensitive data (such as PII) is not sent to the LLM.

Use Case Diagram (UML):



Feedback and Metrics

- Users can provide feedback on responses, helping to improve the system over time.
- The system monitors performance using metrics like request count, error count, and latency.

---

3. System Architecture

Overview

The system is built with a modular architecture that separates the backend API, retrieval system, and LLM integration into distinct layers for better maintainability and scalability.

- **FastAPI Backend**: Handles incoming HTTP requests, processes the queries, and interacts with other layers.
- **SQLite Database**: Stores product data (names, descriptions, prices) in the `products` table, enabling efficient retrieval via FTS5 and FAISS.
- **FAISS Index**: A vector-based search system that allows for fast semantic search of product descriptions.
- **LLM Integration (DeepSeek)**: The LLM model generates natural language responses based on the context retrieved from the database.

Data Flow

1. **User Input**: The user sends a query to the `/simulate_dm` endpoint.
2. **Retrieval**: The query is processed, and the system retrieves relevant data from both FAISS (semantic) and FTS5 (keyword-based).
3. **LLM Generation**: The retrieved data is used to generate a prompt that is sent to the **DeepSeek LLM**, which returns a response.
4. **Response**: The generated response is returned to the user via the API.

---

4. Installation and Setup

Step 1: Clone the repository

git clone <repository_url>
cd <project_directory>

Step 2: Create a virtual environment and activate it

python3 -m venv .venvs
source .venvs/bin/activate

Step 3: Install required dependencies

pip install --upgrade pip
pip install -r requirements.txt

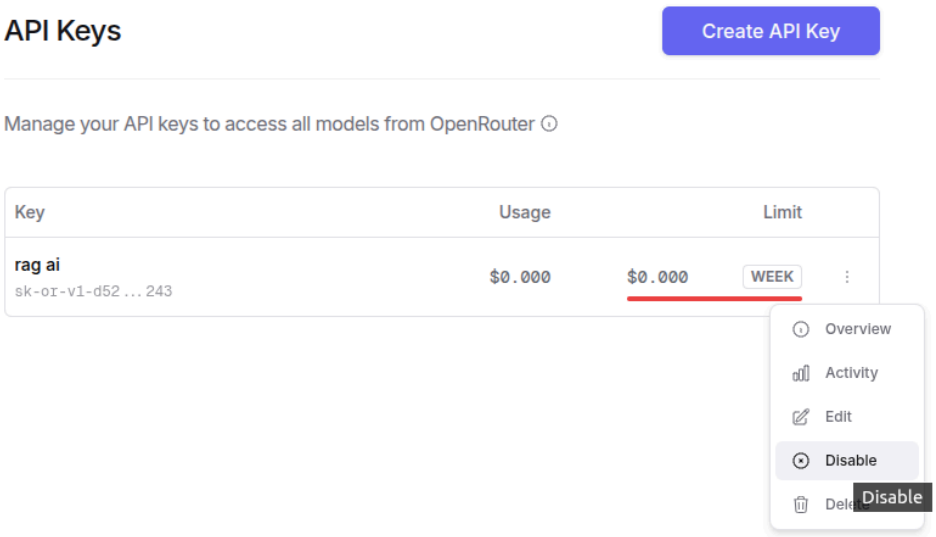
Step 4: Set up environment variables

Create a `.env` file and add the following variables:

```
# API keys and URLs
LLM_PROVIDER=openrouter
LLM_API_BASE=https://openrouter.ai/api/v1
LLM_API_KEY=<your_openrouter_api_key>
LLM_MODEL=deepseek/deepseek-chat-v3.1:free

# Database paths
DB_PATH=rag-instabot/db/app_data.sqlite
INDEX_PATH=rag-instabot/data/faiss_index

# Retrieval configuration
MIN_VECTOR_SCORE=0.30
MAX_CTX_ITEMS=4
MAX_DESC_CHARS=220

# Security & Rate-limiting
REQUIRE_API_KEY=true
API_KEY=<your_api_key>

RL_BUCKET_SIZE=60
RL_REFILL_PER_SEC=1.0
RL_IDENTITY_HEADER=X-API-Key
```

How to setup open router api key

- Proceed to Official OpenRouter web site: https://openrouter.ai/
- Make an account and login
- Navigate to key / APIkey section and generate a key (note: you will have one key and can access all models via that same key)
- Activate the key in your open router panel before run(note: some times will get disabled by default)



Step 5: Run the FastAPI app

uvicorn app.main:app --reload --port 8000

---

5. Environment Configuration

.env File

The `.env` file contains sensitive information like API keys and database paths. Ensure it is kept secure and never exposed.

API Key Authentication

The API key is required for accessing certain endpoints like `/simulate_dm`. Set it in your `.env` file and provide it in the request header.

---

6. API Endpoints

`/simulate_dm` (POST)

- **Description**: This is the main endpoint where users send queries to the chatbot.

Request Body:

```
{
 "sender_id": "u1",
 "message_id": "m1",
 "text": "کرم ضدآفتاب مناسب پوست چرب"
}
```

-

Response:

```
{
 "reply": "این کرم ضدآفتاب مناسب پوست چرب است. برند L'Oréal مناسب است."
}
```

-

`/health` (GET)

- **Description**: Health check endpoint to monitor the status of the system.

Response:

```
{
 "status": "ok",
 "env": "prod",
 "llm_provider": "openrouter",
 "llm_model": "deepseek/deepseek-chat-v3.1:free",
 "db_exists": true,
 "index_exists": true
```

}

-

`/metrics` (GET)

- **Description**: Provides performance metrics like request counts, fallback counts, and error counts.

Response:

```
 requests_total 10
fallback_total 0
errors_total 0
```

-

`/feedback` (POST)

- **Description**: Allows users to provide feedback on responses.

Request Body:

```
 {
  "message_id": "m1",
  "rating": "good",
  "note": "Helpful response"
}
```

-

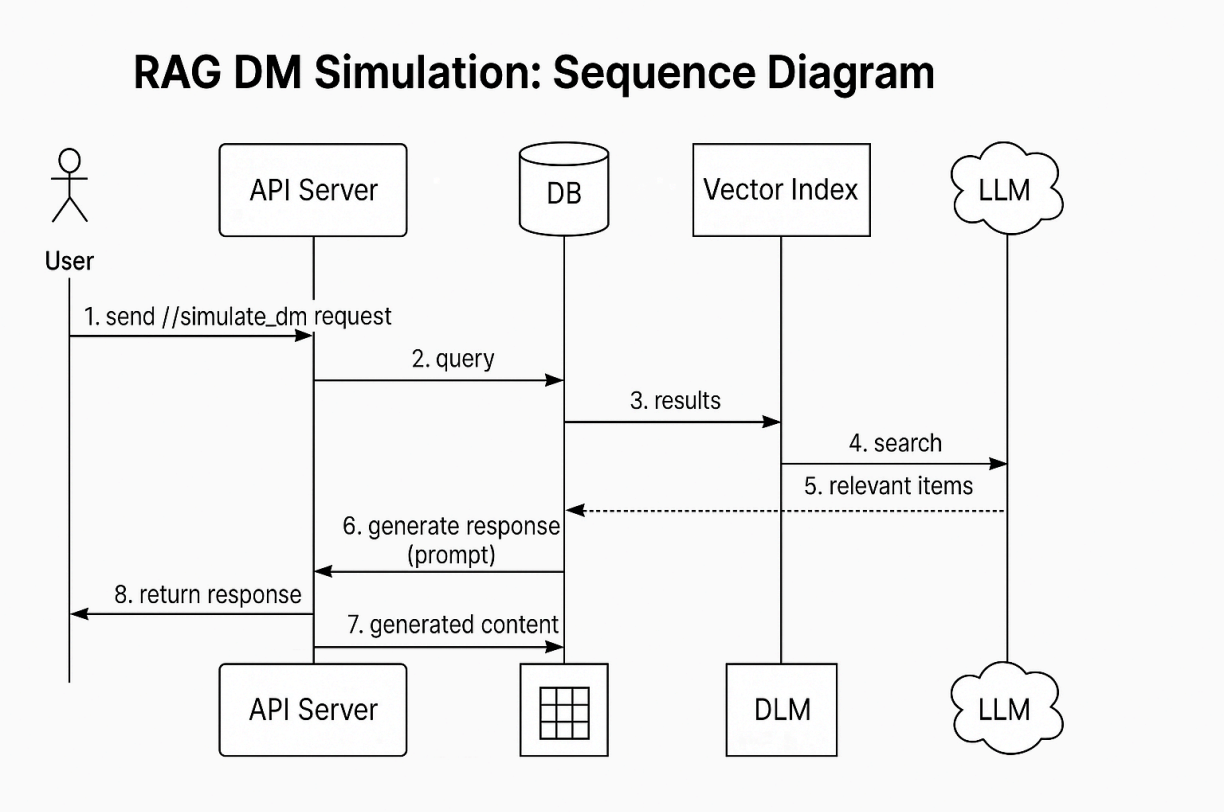
Response:

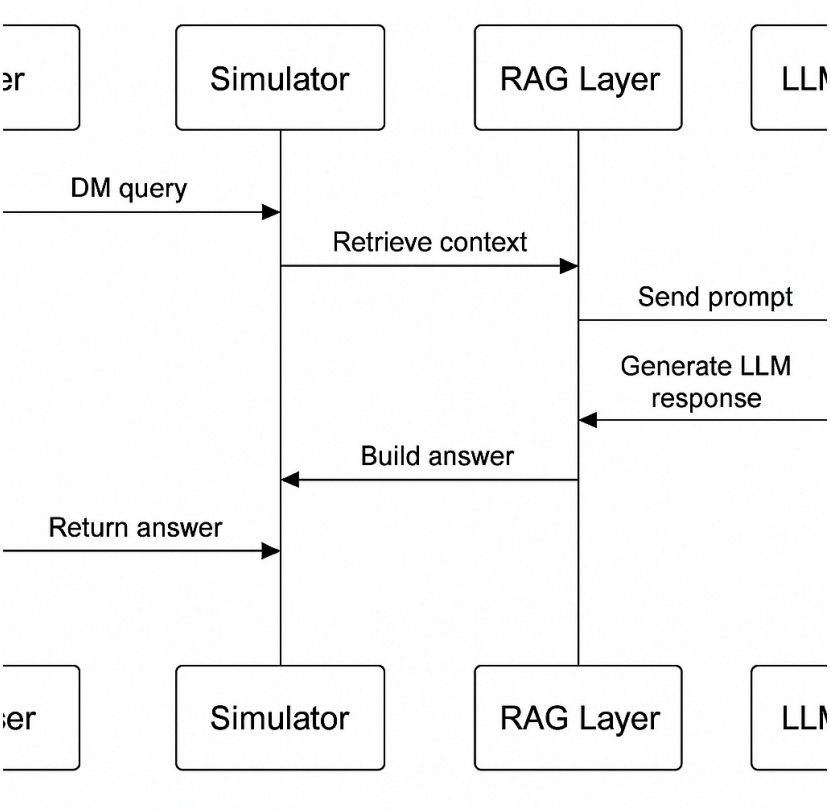
```
 {
  "ok": true
}
```

-

---

7. Data Flow





How to setup open router api key

1. **User Input**:
    - The user sends a message through the `/simulate_dm` endpoint.
    - The text is normalized and processed for retrieval.
2. **Retrieval**:
    - **FAISS** is used for **semantic retrieval** based on the query.
    - **FTS5** (Full-text search) is used for **keyword-based retrieval** from SQLite.
    - The results are merged to form the final context.
3. **LLM Prompt**:
    - A prompt is created using the retrieved context and user query.
    - The prompt is sent to the **OpenRouter LLM** (DeepSeek) for response generation.
4. **Response**:
    - The generated response from the LLM is returned to the user.

---

8. Testing

To run tests, ensure the following:

- **Test API Endpoints**: Use `curl` or Postman to send sample queries to `/simulate_dm`.
- **Validate Data Retrieval**: Use the `/debug/retrieve` and `/debug/prompt` endpoints to ensure correct data retrieval and prompt formatting.
- **Error Handling**: Test rate-limiting, fallback, and API key validation.

---

9. Security & Privacy

Security Measures

- **API Key Authentication**: Ensures only authorized requests are processed.
- **Redaction**: Ensuring that sensitive data such as PII is not sent to the LLM.
- **Rate limiting**: Protecting the service from abuse.
- **Logging**: Safe logging practices, ensuring no sensitive data is logged.

Privacy considerations

- **Minimal Data**: Only the necessary data (name, description, price) is sent to the LLM.
- **No Personal Identifiable Information (PII)**: PII is never included in the request to the LLM.

---

10. Future Improvements

1. **Advanced Semantic Retrieval**: Use better semantic models or even custom fine-tuned models.
2. **Offline Mode**: Implement a local LLM inference option for privacy-sensitive clients.
3. **Scaling**: Use a more powerful vector database for larger datasets (e.g., Weaviate, Qdrant).
4. **User Personalization**: Implement user profiles and personalized recommendations.
5. **Voice Integration**: Add speech-to-text (STT) and text-to-speech (TTS) for voice-based interactions.
