Ollama API Usage Documentation

This document describes how the Ollama LLM APIs are used in this project, their purpose, configuration, and example usages.

---

1. Where and How Ollama is Used

- **Ollama is used as the main LLM (Large Language Model) provider** for chat and RAG (Retrieval Augmented Generation) in the backend.
- The integration is done via the `langchain_ollama.ChatOllama` class, which wraps Ollama's HTTP API.
- The main usage is in `ChatService` (`chat_service.py`) and the LLM abstraction layer (`llm_models.py`).

---

2. Configuration and Endpoints

- **Base URL:**
  - Set by `OLLAMA_BASE_URL` (default: `http://localhost:11434`)
- **Model:**
  - Set by `OLLAMA_MODEL` (e.g., `llama3:8b-instruct-q8_0`)
- **Temperature:**
  - Set by `DEFAULT_TEMPERATURE` (default: `0.7`)
- **Max Tokens:**
  - Set by `MAX_TOKENS` (default: `2048`)
- **Timeouts:**
  - `timeout` and `request_timeout` (default: 120 seconds)

These are loaded from `app/core/config.py` and can be set via environment variables.

---

3. API Calls and Their Purpose

a. Chat/Completion

- **Purpose:** Generate a response to a user prompt or chat message.
- **How:**
  - The backend constructs a prompt (optionally with RAG context).
  - Calls `ChatOllama.ainvoke([HumanMessage(content=prompt)])` to get a response.
  - Used in `ChatService._generate_response` and `OllamaLLM.generate`.

b. Streaming

- **Purpose:** Stream LLM responses for real-time chat.
- **How:**
  - Calls `ChatOllama.astream([HumanMessage(content=prompt)])` to yield chunks.
  - Used in `OllamaLLM.generate_stream`.

c. Chat with History

- **Purpose:** Support multi-turn conversations with context.
- **How:**
  - Passes a list of messages (system, user, assistant) to `ChatOllama.ainvoke`.
  - Used in `OllamaLLM.chat`.

---

4. How to Configure/Change Models

- Change the following environment variables or `.env` file:
  - `OLLAMA_BASE_URL` — Ollama server URL
  - `OLLAMA_MODEL` — Model name/tag (e.g., `llama3:8b-instruct-q8_0`)
  - `DEFAULT_TEMPERATURE`, `MAX_TOKENS` — Generation parameters
- These are loaded in `app/core/config.py` and used throughout the backend.

---

5. Example Usage in Codebase

a. Initialization (ChatService)

```
from langchain_ollama import ChatOllama
self.llm = ChatOllama(
    base_url=settings.ollama_base_url,
    model=settings.ollama_model,
    temperature=settings.default_temperature,
    timeout=120,
    request_timeout=120,
    num_predict=2048,
)
```

b. Generating a Response

```
from langchain.schema import HumanMessage
response = await self.llm.ainvoke([HumanMessage(content=prompt)])
```

c. Streaming a Response

```
async for chunk in self.llm.astream([HumanMessage(content=prompt)]):
    yield chunk.content
```

d. Chat with History

```
from langchain.schema import HumanMessage, AIMessage, SystemMessage
messages = [SystemMessage(...), HumanMessage(...), AIMessage(...)]
response = await self.llm.ainvoke(messages)
```

---

6. Related Files

- `app/services/chat_service.py` — Main chat logic and LLM invocation
- `app/implementations/llm_models.py` — LLM abstraction and Ollama implementation
- `app/core/config.py` — Loads all model and API settings
- `.env` — Set environment variables for Ollama

---

7. Switching Between Ollama and OpenAI

- Set `LLM_PROVIDER=ollama` or `LLM_PROVIDER=openai` in your environment or `.env` file.
- The backend will auto-select the correct provider and configuration.

---

8. References

- Ollama Documentation
- LangChain Ollama Integration