Project Documentation

1. Introduction

Brief Overview of the Project

This project is a web service that provides a chat interface to a large language model (Ollama). It allows users to interact with the model through a simple API, supporting both text and image-based conversations.

Purpose and Goals

The primary goal of this project is to create a simple and extensible chatbot service that can be easily integrated into other applications. It aims to provide a seamless interface for developers to leverage the power of large language models without having to deal with the complexities of model interaction.

Target Users or Audience

The target audience for this project is primarily developers who want to build applications with chatbot functionality. It can also be used by anyone who wants to experiment with large language models.

Key Features and Business Value

- **Text and Image-based Chat:** Supports both text and image inputs for a richer conversational experience.
- **Chat History:** Maintains a history of the conversation, allowing the model to have context.
- **Simple API:** Provides a simple and easy-to-use RESTful API for interacting with the chatbot.
- **Extensible:** The project is designed to be easily extensible, allowing developers to add new features and functionality.

2. Technology Stack

- **Backend:** Python, FastAPI, Uvicorn
- **LLM:** Ollama
- **Libraries:**
    - `httpx`: For making asynchronous HTTP requests to the Ollama service.
    - `python-multipart`: For handling file uploads (images).
    - `streamlit`: For creating a simple frontend.

3. System Architecture

Overall Architecture

The system follows a simple client-server architecture. The backend is a FastAPI application that exposes a single API endpoint for chat. The frontend is a Streamlit application that provides a user interface for interacting with the chatbot.

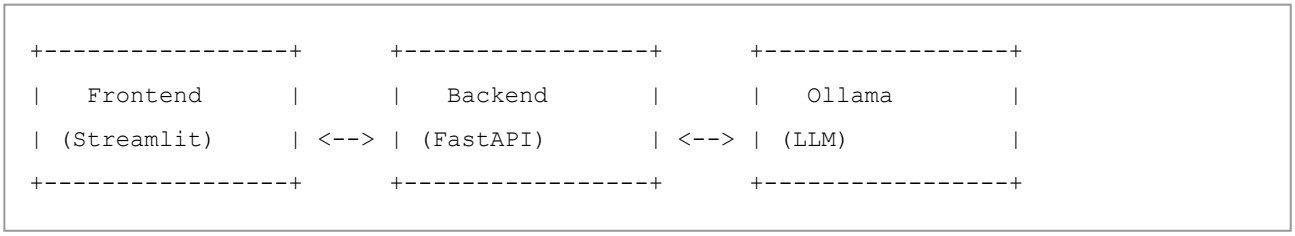
Backend

The backend is a FastAPI application that handles the chat logic. It receives user prompts and images, forwards them to the Ollama model, and returns the model's response.

Frontend

The frontend is a Streamlit application that provides a simple web-based interface for the chatbot. It allows users to enter text prompts and upload images.

ASCII Diagram



4. Functionalities in Detail

`chat_with_ollama` Endpoint

- **Description:** This is the main endpoint for interacting with the chatbot. It accepts a user's prompt and an optional image. It then forwards the request to the Ollama model and returns the model's response.
- **Inputs:**
  - `prompt` (string): The user's text query.
  - `image` (file, optional): An image file to be processed by the model.
  - `context` (string, optional): Additional context to be passed to the model.
- **Outputs:**
  - A JSON object containing the model's response.

Chat History

- **Description:** The service maintains a history of the conversation. The history is loaded before each request and updated with the new user prompt and model response.
- **Dependencies:** `get_chat_history` and `save_chat_history` functions. The implementation of these functions is not provided in the code snippet, but it is assumed that they handle the storage and retrieval of the chat history (e.g., from a file or a database).

5. API Overview

- **Endpoint:** `POST /chat`
- **Request:**
    - `Content-Type: multipart/form-data`
- **Form Data:**
    - `prompt`: string
    - `image`: file (optional)
    - `context`: string (optional)
- **Response:**
    - `Content-Type: application/json`
- **Example Response:**

```
{
    "response": "Hello! How can I help you today?"
}
```

6. Database Design

The project does not use a traditional database. The chat history is managed by the `get_chat_history` and `save_chat_history` functions. The exact implementation is not specified, but it could be a simple file-based storage (e.g., a JSON file).

7. Security

- **Authentication and Authorization:** There is no authentication or authorization implemented in the provided code. The API is open to the public.
- **Data Protection, Encryption, and Validation:**
    - Images are encoded in base64 before being sent to the Ollama service.
    - There is no explicit input validation beyond what FastAPI provides.

8. Deployment and Environment Setup

Local Setup

1. **Install Python and dependencies:**

```
pip install fastapi uvicorn python-multipart httpx streamlit
```

2. **Run the backend:**

```
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
```

3. **Run the frontend:**

```
streamlit run frontend/app.py
```

4. The `run.py` script is also provided to run both the backend and frontend in parallel.

Environment Variables

- `OLLAMA_MODEL`: The name of the Ollama model to be used (e.g., `llama2`).

9. Testing

There is no testing strategy or framework specified in the provided code. For a production-ready application, it is recommended to add unit tests for the API endpoints and services, and integration tests to verify the interaction between the backend and the Ollama service.

10. Future Enhancements

- **Database Integration:** Implement a proper database (e.g., PostgreSQL, MongoDB) for storing chat history to ensure data persistence and scalability.
- **User Authentication:** Add user authentication and authorization to control access to the API.
- **Input Validation:** Implement robust input validation to prevent security vulnerabilities and ensure data integrity.
- **Improved Frontend:** Enhance the frontend with more features, such as displaying chat history, user accounts, and a more polished user interface.
- **Error Handling:** Implement more specific error handling to provide better feedback to the user in case of failures.

11. Conclusion

This project provides a solid foundation for building a chatbot application. It demonstrates how to create a simple and effective API using FastAPI and how to interact with a large language model like Ollama. While it lacks some production-ready features, it serves as an excellent starting point for developers who want to explore the world of conversational AI.