**Occams Advisory Chatbot Documentation**

**1. Project Overview**

This project is a web-based chatbot built using **Node.js**, **LangChain**, and **Flask**. It is designed to answer questions based on the information available on the **Occams Advisory** website. The chatbot integrates a **frontend (HTML/JavaScript)** and a **backend (Node.js + Flask)**, which uses a **Phi-2 model** for generating responses, and **FAISS** for retrieving relevant information from the website content.

**Features:**

* **Frontend (UI)**: A simple interface for interacting with the chatbot.
* **Backend**: A server to handle requests and retrieve responses from a machine learning model (Phi-2).
* **Data Source**: Scraped content from the **Occams Advisory** website.

**2. Prerequisites**

Before you begin, ensure you have the following tools and dependencies installed on your local machine:

* **Node.js** (v14 or later): [Install Node.js](https://nodejs.org/en/)
* **npm** (Node Package Manager): Should be installed along with Node.js.
* **Python 3**: [Install Python](https://www.python.org/downloads/)
* **Python Libraries**:
  + flask
  + flask\_cors
  + transformers
  + sentence\_transformers
  + faiss-cpu
  + torch (ensure it is installed with the correct CUDA version if using a GPU)

You will also need:

* **Text scraping**: The fetch.py script scrapes data from the Occams Advisory website and saves it in a text file (occams\_contant.txt2).

**3. Setup and Installation**

**Step 1: Clone the Repository**

Start by cloning the project repository to your local machine:

git clone <repository\_url>

cd <repository\_directory>

**Step 2: Install Node.js Dependencies**

Navigate to the project directory and install the necessary dependencies for the frontend (Node.js):

npm install express cors axios

**Step 3: Install Python Dependencies**

Before running the Flask backend, you need to install the required Python libraries. Navigate to the directory where the backend.py script is located and run:

pip install flask flask\_cors transformers sentence\_transformers faiss-cpu torch

**Step 4: Scrape Occams Advisory Website**

To gather the content from the Occams Advisory website, run the fetch.py script, which scrapes the text from the site and saves it into a file (occams\_contant.txt2):

python fetch.py

**4. How the Code Works**

**4.1 Frontend (HTML, CSS, JavaScript)**

* **index.html**: This is the UI of the chatbot, built using HTML and styled with CSS. It includes a chatbot popup, a text input field, and an area to display messages.
* **app.js**: This JavaScript file handles the user interaction, including:
  + Displaying the chatbot window.
  + Sending user messages to the Node.js server (/chat endpoint).
  + Receiving and displaying the bot's response.

**4.2 Backend (Node.js + Flask)**

* **server.js**: This Node.js server listens on port 3000 and serves the frontend. It also handles POST requests to the /chat endpoint, forwarding messages to the Python backend (Flask) for processing.

It communicates with the **Flask server** using **Axios**. Once the Python backend provides a response, it sends it back to the frontend.

* **backend.py**: This Python script sets up a Flask server that handles user queries. It uses the **Phi-2 model** for generating responses based on the content of the Occams Advisory website:
  + **Content Extraction**: The occams\_contant.txt2 file contains the content scraped from the website.
  + **Embedding & Retrieval**: The script uses the **Sentence-Transformer** model to create embeddings of the content, which are stored in a **FAISS** index for efficient retrieval.
  + **Response Generation**: The model generates a response based on the query and the relevant context retrieved from the website content.

**4.3 Data Flow**

1. **User Input**: The user enters a message in the chatbot UI.
2. **Frontend**: The message is sent to the Node.js backend via an HTTP request.
3. **Node.js Server**: The backend forwards the message to the Python backend.
4. **Python Flask Server**: The message is processed using the Phi-2 model, and a response is generated using context from the scraped website content.
5. **Backend Response**: The Python server sends the response back to the Node.js server.
6. **Frontend**: The bot's response is displayed in the chatbot UI.

**5. Running the Application**

**Step 1: Start the Backend (Flask Server)**

First, navigate to the folder containing backend.py and run:

python backend.py

This will start the Python server on http://127.0.0.1:5000.

**Step 2: Start the Node.js Server**

Next, navigate to the folder containing server.js and run:

node server.js

This will start the Node.js server on http://localhost:3000.

**Step 3: Open the Frontend in a Browser**

Finally, open the index.html file in a web browser. The chatbot should appear, and you can start interacting with it.

**6. Testing the Application**

After setting up and running the application, you can test the chatbot by asking questions about Occams Advisory’s services. For example:

* "What services does Occams Advisory provide?"
* "How can I get in touch with Occams Advisory?"
* "Tell me about business solutions."

You should receive relevant answers based on the information scraped from the Occams Advisory website.

**7. Common Issues & Troubleshooting**

* **Flask Server Not Starting**:
  + Ensure all dependencies are installed by running pip install -r requirements.txt.
  + Check that you are running the server in the correct directory.
* **CORS Issues**:
  + Make sure the CORS policy is set correctly in the backend.py file (CORS is enabled in this project).
* **Missing Website Content**:
  + If the text file (occams\_contant.txt2) is not created, ensure the fetch.py script runs successfully and retrieves the correct data.

**8. Conclusion**

This chatbot allows users to interact with an AI assistant powered by the Phi-2 language model, providing answers strictly based on the content available on the **Occams Advisory** website. By following the instructions above, you should be able to set up, run, and interact with the chatbot successfully.

**Document written by**: *Abhishek Gupta*

*Date: 4th April, 2025*

Due to the limited availability of open-source resources, response times may be longer. However, with access to robust resources such as AWS, I can enhance efficiency, reduce response times, and improve the quality of the responses. Feel free to extend the project, improve the UI, or add additional features like logging or user authentication.