**LLaMA Chatbot Application**

**Overview:**

This chatbot application integrates a Flask-based backend, a dynamic HTML, CSS, JavaScript frontend, and uses LLaMA via Ollama for generating intelligent responses. It also implements web scraping, caching, and background scheduling to ensure up-to-date data.

**Features:**

1. **Web Scraping**
   * Extracts textual data (paragraphs, headers) from a given website.
   * Recursively scrapes linked pages up to a specified depth.
2. **Chatbot Integration**
   * Uses the LLaMA model via Ollama API to generate responses based on scraped data and user queries.
3. **Cache Management**
   * Stores scraped data in a global variable for faster access.
   * Automatically refreshes the cache every 24 hours using a background scheduler.
4. **API Endpoints**
   * /: Serves the home page.
   * /chat: Accepts user queries and returns chatbot responses.
5. **Error Handling**
   * Handles HTTP request errors during scraping.
   * Returns appropriate error messages when cache or chatbot response is unavailable.

**Prerequisites:**

1. Python 3.x
2. Required Python libraries:
   * Flask
   * flask\_cors
   * requests
   * beautifulsoup4
   * apscheduler
   * ollama
3. Ollama for LLaMA model access.

**Code SetUp**

**1. Global Setup:**

* Imported required libraries for Flask, web scraping, scheduling, and Ollama API integration.
* Declared a global variable cached\_data to store scraped content.

**from flask import Flask, request, jsonify, render\_template**

**from flask\_cors import CORS**

**from apscheduler.schedulers.background import BackgroundScheduler**

**import requests**

**from bs4 import BeautifulSoup**

**from urllib.parse import urljoin**

**import ollama**

**2. Flask App Configuration:**

* Initialized the Flask app and enabled CORS to allow cross-origin requests.

**app = Flask(\_name\_)**

**CORS(app)**

**3. Web Scraping Function:**

**scrape\_website(url, max\_depth=2)**

* Recursively scrapes text data from a given URL up to the specified depth.
* Uses BeautifulSoup to parse HTML content.

**visited = set()**

**data = []**

**def scrape\_page(page\_url, depth):**

**if page\_url in visited or depth > max\_depth:**

**return**

**visited.add(page\_url)**

**try:**

**response = requests.get(page\_url, timeout=10)**

**if response.status\_code == 200:**

**soup = BeautifulSoup(response.content, 'html.parser')**

**paragraphs = soup.find\_all(['p', 'h1', 'h2', 'h3'])**

**page\_content = ' '.join([para.get\_text() for para in paragraphs])**

**data.append(page\_content)**

**# Find and scrape links recursively**

**for link in soup.find\_all('a', href=True):**

**full\_url = urljoin(page\_url, link['href'])**

**if url in full\_url and full\_url not in visited:**

**scrape\_page(full\_url, depth + 1)**

**except requests.RequestException as e:**

**print(f"Error accessing {page\_url}: {e}")**

**scrape\_page(url, 0)**

**return ' '.join(data)**

1. **Preprocess scraped data:**

**preprocess\_scraped\_data()**

* This function filters scraped data by matching query keywords to sentences, extracting and returning relevant information or a default message if no matches are found.

**def preprocess\_scraped\_data(scraped\_data, query):**

**if not scraped\_data or not query:**

**return "No relevant data available."**

**keywords = query.lower().split()**

**relevant\_data = set()**

**for sentence in scraped\_data.split('. '):**

**if any(keyword in sentence.lower() for keyword in keywords):**

**relevant\_data.add(sentence.strip())**

**if relevant\_data:**

**return '. '.join(relevant\_data)**

**return "No matches found in the scraped data.**

1. **Cache Management**

**update\_cache()**

* Updates the cached\_data variable by scraping content from the target website.

**def update\_cache():**

**global cached\_data**

**website\_url = "https://manipaldigital.info/"**

**print("Updating cached data...")**

**cached\_data = scrape\_website(website\_url)**

**print("Cache updated.")**

1. **Chatbot Response Function**

**generate\_response(user\_query, scraped\_data)**

* Sends a user query and the scraped data to the LLaMA model via Ollama.
* Returns the generated response.

**def generate\_response(user\_query, scraped\_data):**

**messages = [**

**{"role": "user", "content": user\_query},**

**{"role": "system", "content": scraped\_data}**

**]**

**try:**

**response = ollama.chat(model="llama2", messages=messages)**

**print("DEBUG: Full response:", response)**

**return response.get("text", "Error: 'text' key not found in response.")**

**except Exception as e:**

**return f"Error generating response: {str(e)}"**

1. **API Routes**

* **Home Route (/)**
  + Renders the index.html template.

**@app.route('/')**

**def index():**

**return render\_template('index.html')**

* **Chat Route (/chat)**
  + Accepts a user query (via POST request).
  + Returns a response generated using the chatbot and cached data.

**@app.route('/chat', methods=['POST'])**

**def chat():**

**global cached\_data**

**user\_query = request.json.get('message')**

**if not cached\_data:**

**return jsonify({"response": {"message": {"content": "Error: Cached data is not available. Please try again later."}}})**

**response = generate\_response(user\_query, cached\_data)**

**return jsonify({"response": {"message": {"content": response}}})**

1. **Background Scheduler**

* Refreshes the cache every 24 hours.

**scheduler = BackgroundScheduler()**

**scheduler.add\_job(update\_cache, 'interval', hours=24)**

**scheduler.start()**

**Frontend Code**

**HTML**

* Provides the chatbot interface.

**<div class="chat-container" id="chat-container">**

**<header class="chat-header">**

**<div class="logo">**

**<h1>ChatBot</h1>**

**</div>**

**<button class="minimize-btn" id="minimize-btn">—</button>**

**</header>**

**<div class="chat-messages" id="chat-messages"></div>**

**<div class="predefined-buttons" id="predefined-buttons">**

**<button class="option-btn" data-query="Imaging">Imaging</button>**

**<button class="option-btn" data-query="Packaging">Packaging</button>**

**<button class="option-btn" data-query="CGI">CGI</button>**

**<button class="option-btn" data-query="AV">AV</button>**

**<button class="option-btn" data-query="Service">Service</button>**

**</div>**

**<form id="chat-form">**

**<input type="text" id="user-input" placeholder="Send a message..." autocomplete="off" required>**

**<button type="submit" class="send-btn">→</button>**

**</form>**

**</div>**

**<button id="chat-toggle" class="chat-toggle hidden">**

**<img src="/assets/images/bot1.webp" alt="Chat Icon">**

**</button>**

**JavaScript**

* Handles chat interactions and communication with the backend.

**const chatForm = document.getElementById("chat-form");**

**const userInput = document.getElementById("user-input");**

**chatForm.addEventListener("submit", async (event) => {**

**event.preventDefault();**

**const message = userInput.value.trim();**

**if (!message) return;**

**const response = await fetch("http://127.0.0.1:5000/chat", {**

**method: "POST",**

**headers: { "Content-Type": "application/json" },**

**body: JSON.stringify({ message }),**

**});**

**const data = await response.json();**

**});**

**Running the Application**

* Install dependencies using:

**pip install Flask flask-cors requests beautifulsoup4 apscheduler ollama**

* Start the Flask server: **python app.py.**
* Open index.html in a browser to interact with the chatbot.