Project Title: CODE-RUN AI

Table of Contents:

1. Introduction
2. Project Overview
3. Features
4. Application Components
5. System Architecture
6. Installation and Setup
7. Usage
8. User Management
9. Prompt Management

10. Stream Responses

11. Adjust The Markdown

1. Introduction: This document serves as the official documentation for the **ChatGPT-4 Question Answering Web Application**. It provides comprehensive information about the project, including its features, system architecture, installation instructions, usage guidelines, user and prompt management.

2. Project Overview: The ChatGPT-4 Question Answering Web Application is designed to provide users with the capability to ask questions and receive answers generated by **OpenAI's** **ChatGPT-4 model (gpt-4-0125-preview)**. The application utilizes a Flask backend for server-side processing and a **JavaScript, HTML, and CSS frontend** for the user interface.

3. Features:

* Question answering using ChatGPT-4.
* User authentication and management.
* Prompt customization for tailored responses.
* Password change functionality.
* Responsive and intuitive user interface.
* Secure data transmission and storage.

## 4.Application Components

The Flask application consists of the following components:

1. app.py: This Python script contains the main Flask application logic, including routes, database operations, file processing functions, and integration with **OpenAI's GPT-4 mode**l.
2. testing.html: This HTML file defines the user interface for the web application, and handling user interactions.
3. **user\_prompt.html :** This HTML file handle the prompt regarding task user can change the prompt using this page.
4. style.css: This CSS file contains styles for the user interface elements defined in **testing.html or user\_prompt.html**.
5. javascript.js: This JavaScript file contains client-side scripting for handling form submissions, displaying messages, and interacting with the backend server.

5. System Architecture: The system consists of two main components:

* Backend: Developed using Flask and Python, responsible for handling user requests, processing prompts, and interfacing with the ChatGPT-4 model.
* Frontend: Built with JavaScript, HTML, and CSS, providing the user interface for interacting with the application.

6. Installation and Setup:

* Install dependencies using pip .
* Configure environment variables for API keys and database connection.
* Run the Flask server and deploy the frontend files to a web server.

7. Usage:

* Access the web application through a supported web browser.
* Sign up or log in to your account.
* Ask questions using the provided input field and receive answers generated by ChatGPT-4.
* Optionally, customize prompts for more tailored responses.
* Change your password through the user settings page.

8. User Management:

* Users can log in using their credentials.
* User authentication is handled securely to prevent unauthorized access.
* Passwords are encrypted and stored securely in the database.

9. Prompt Management:

* Users can customize prompts to influence the responses generated by ChatGPT-4.
* Access to prompt management is available through the user **prompt page**.
* Changing prompts can lead to different types of answers for a given question.

10. Stream Responses:

**1. OpenAI Model Selection:**

* The code utilizes **OpenAI's GPT-4 model** for generating responses.
* The model identifier "**gpt-4-0125-preview**" is specified.

**2. Message Configuration:**

* The `**messages**` parameter contains a list of dictionaries representing the prompts for the GPT model.
* Each dictionary includes the role of the message ('**system**' or '**user**') and the content of the prompt.

**3. Generation Parameters:**

* **temperature**: Controls the randomness of the generated responses. Lower values lead to more deterministic responses, while higher values result in more diverse outputs.
* **frequency\_penalty**: Penalizes tokens based on their frequency in the training data. Higher values encourage the model to produce more unique responses.
* **n**: Specifies the number of completions to generate. In this case, it's set to 1, indicating a single response.

**4. Response Generation Loop:**

* The loop iterates over the completions generated by the **OpenAI model**.
* For each completion (referred to as a "**chunk**"), it extracts the content of the response.
* If the content is not None, it yields the encoded response content.
* If the content is None, the loop continues to the next iteration.

**5. Encoding and Yielding Responses:**

* The content of the response is encoded before yielding it.
* The encoded response is then yielded to the caller.

for chunk in client.chat.completions.create(  
 model="gpt-4-0125-preview", # Use the correct GPT-4 model identifier  
 messages=message,  
 temperature=temperature,  
 frequency\_penalty=frequency\_penalty,  
 n=1,  
 stop=None,  
 stream=True,  
):  
 content = chunk.choices[0].delta.content  
 print(content)  
 if content is not None:  
 yield content.encode()  
 else:  
 continue

This code snippet essentially orchestrates the process of generating responses from the GPT-4 model based on provided prompts and parameters, and it yields the responses one by one as they are generated.

11. Adjust The Markdown:

This JavaScript function, `**boldTextBetweenDoubleAsterisks**`, takes a string `**text**` as input and processes it to apply bold formatting to text enclosed between double asterisks **`\*\*`** or to lines starting with **`###`**.

Here's a breakdown of the function:

**1**. **Regular Expression Pattern**:

* var pattern = **/^(###.\*$)|\\*\\*(.\*?)\\*\\*/gm**;`
* This regular expression pattern matches:
* Lines starting with **`###`**.
* Text enclosed between pairs of **`\*\*`**.

**2. Replacement Logic:**

* The `**replace**()` method is used with a callback function to replace the matched patterns with bold text.
* If the match is a line starting with **`###`,** the `group1` parameter captures the matched text. The function removes **`###`** from the line and wraps the entire line with `<**b**>` tags to make it bold.
* If the match is text between **`\*\*`,** the `group2` parameter captures the text between the double asterisks. The function wraps this text with `<**b**>` tags to make it bold.

**3. Return Value:**

* The processed text with bold formatting applied is returned as the result.

**4. Usage:**

* You can call this function and pass a string of text as an argument. It will return the input text with bold formatting applied to the specified patterns.
* This function is useful for adding emphasis to specific parts of text, such as section headings **(`###`)** or inline emphasis **(`\*\*`)**.

function boldTextBetweenDoubleAsterisks(text) {  
 // Define the regular expression pattern to match lines starting with "###" or text between pairs of "\*\*"  
 var pattern = /^(###.\*$)|\\*\\*(.\*?)\\*\\*/gm;  
 // Use replace() with a callback function to replace the matches with bold text  
 var result = text.replace(pattern, function(match, group1, group2) {  
 if (group1) {  
 // If the line starts with "###", remove "###" and make the entire line bold  
 return '<b>' + group1.replace(/^###\s\*/, '') + '</b>';  
 } else {  
 // Otherwise, make the text between "\*\*" bold  
 return '<b>' + group2 + '</b>';  
 }  
 });  
 return result;  
}

NOTE:-

The Requirment is insatll using cmd “**pip install -r reqquirements.txt**”.

The Project will run using cmd “**Python app.py**”