Chat Interface Specification

Overview

This document outlines the specifications for a chat interface similar to Poe, including mechanisms for user login, interaction with a server for responses, and a token purchasing system.

Features

1. Chat Interface

A user-friendly chat interface allowing users to input prompts and receive responses.
Responses should be displayed in a conversational format.

2. Fetching Responses

Mechanism: Use an HTTP GET request to communicate with the server.
Request Format:

```
GET http://localhost:8888/ask
message body with a json:
{
    "address": 0x12345,
    "key": 0x12345,
    "prompt": "Hello, I am your father. How are You?"
}
```

Response:

```
{"response": "Fuck you. I am your grandfather actually."}
```

Parameters:
user_address: The wallet address of the user.
key: API key associated with the user, when they buy, they will get a new one.

prompt: User's input (the message to be sent to the llm).
response: The server will return a text response to be displayed in the chat interface.

3. User Login

Users must log in to access the chat functionality.
Storage: Store the user_address and key as **JavaScript variables**. (i.e. saved in the front end temporally)
Requirement: New users must purchase tokens to receive their key.
The address is the metamask wallet address.

4. Token Purchase

Users can specify the amount of tokens they wish to purchase.
Mechanism: User will buy the token by start a transaction using their wallet. Frontend need design a interface to generate this transaction by press a button and enter some number they would like to spend.

It's needed to get the new key from the back end. You can get it by this.

Response: The server will return a key by identifying the user_address .
Request Format:

```
GET http://localhost:8888/get_key
with message body:
{
    "user_address": 0x123456
}

Response:
{
    "key": 0x123456
}
```

Summary

This document provides a structured approach to implementing a chat interface, including user authentication, response handling, and token purchasing. Ensure all interactions with the server are secure and handle errors gracefully to enhance user experience.