```
import os
from dotenv import load_dotenv
from openai import OpenAI
import google.generativeai as genai
import gradio as gr
```

Hàm / Lệnh	Chức năng
import os	Import module os để thao tác với biến môi trường.
from dotenv import load_dotenv	Import load_dotenv để tải biến môi trường từ tệp .env.
from openai import OpenAI	Import OpenAI SDK để gọi API OpenAI.
import google.generativeai as genai	Import Google Generative AI SDK để sử dụng mô hình của Google.
import gradio as gr	Import Gradio để tạo giao diện người dùng.

```
load_dotenv(override=True)
openai_api_key = os.getenv('OPENAI_API_KEY')
anthropic_api_key = os.getenv('ANTHROPIC_API_KEY')
google_api_key = os.getenv('GOOGLE_API_KEY')

if openai_api_key:
    print(f"OpenAI API Key exists and begins {openai_api_key[:8]}")
else:
    print("OpenAI API Key not set")

if anthropic_api_key:
    print(f"Anthropic API Key exists and begins
{anthropic_api_key[:7]}")
else:
    print("Anthropic API Key not set")

if google_api_key:
    print(f"Google API Key exists and begins {google_api_key[:8]}")
else:
    print("Google API Key not set")
```

Hàm / Lệnh	Chức năng
load_dotenv(override=True)	Tải các biến môi trường từ tệp .env, ghi đè giá trị cũ nếu có.
os.getenv('OPENAI_API_KEY')	Lấy giá trị API key của OpenAI từ biến môi trường.
os.getenv('ANTHROPIC_API_KEY')	Lấy giá trị API key của Anthropic từ biến môi trường.
os.getenv('GOOGLE_API_KEY')	Lấy giá trị API key của Google từ biến môi trường.
if openai_api_key: print(...)	Kiểm tra xem OpenAI API key có tồn tại không, nếu có thì in ra 8 ký tự đầu.
if anthropic_api_key: print(...)	Kiểm tra xem Anthropic API key có tồn tại không, nếu có thì in ra 7 ký tự đầu.
if google_api_key: print(...)	Kiểm tra xem Google API key có tồn tại không, nếu có thì in ra 8 ký tự đầu.
print("API Key not set")	Nếu API key không tồn tại, in thông báo "API Key not set".

```
openai = OpenAI()
MODEL = 'gpt-4o-mini'

genai.configure(api_key=google_api_key)

MODEL = 'gemini-pro'
```

Hàm / Lệnh	Chức năng
openai = OpenAI()	Khởi tạo đối tượng OpenAI để sử dụng API của OpenAI.
MODEL = 'gpt-4o-mini'	Gán giá trị 'gpt-4o-mini' cho biến MODEL, dùng làm tham số chỉ định mô hình AI ban đầu.
genai.configure(api_key=google_api_key)	Cấu hình Google Generative AI bằng API key lấy từ biến môi trường google_api_key.
MODEL = 'gemini-pro'	Cập nhật biến MODEL thành 'gemini-pro', thay đổi mô hình AI được sử dụng.

```
system_message = "You are a helpful assistant"
def chat(message, history):
    messages = [{"role": "system", "content": system_message}] + history
+ [{"role": "user", "content": message}]

    print("History is:")
    print(history)
    print("And messages is:")
    print(messages)

    stream = openai.chat.completions.create(model=MODEL,
messages=messages, stream=True)

    response = ""
    for chunk in stream:
        response += chunk.choices[0].delta.content or ''
        yield response

gr.ChatInterface(fn=chat, type="messages").launch()
```

Hàm / Lệnh	Chức năng
system_message = "You are a helpful assistant"	Định nghĩa thông điệp hệ thống cho trợ lý AI, hướng dẫn hành vi trả lời.
def chat(message, history):	Định nghĩa hàm chat nhận đầu vào là tin nhắn người dùng (message) và lịch sử hội thoại (history).
messages = [{"role": "system", "content": system_message}] + history + [{"role": "user", "content": message}]	Xây dựng danh sách tin nhắn gửi đến API, bao gồm tin nhắn hệ thống, lịch sử hội thoại và tin nhắn mới từ người dùng.
print("History is:") print(history) print("And messages is:") print(messages)	In ra lịch sử hội thoại và danh sách tin nhắn để kiểm tra và gỡ lỗi.
stream = openai.chat.completions.create(model=MODEL, messages=messages, stream=True)	Gửi yêu cầu đến API OpenAI với mô hình được chỉ định (MODEL), yêu cầu tạo phản hồi theo dạng luồng (streaming).

response = "" for chunk in stream: response += chunk.choices[0].delta.content or "" yield response	Lặp qua từng phần (chunk) của phản hồi từ API, tích lũy nội dung phản hồi vào biến response và trả về kết quả từng phần qua yield (cho phép hiển thị phản hồi dần dần).
gr.ChatInterface(fn=chat, type="messages").launch()	Tạo và khởi chạy giao diện chat của Gradio, sử dụng hàm chat làm xử lý, với kiểu dữ liệu là "messages".

```
def chat(message, history):
    messages = [{"role": "system", "content": system_message}] + history
+ [{"role": "user", "content": message}]

    stream = openai.chat.completions.create(model=MODEL,
messages=messages, stream=True)

    response = ""
    for chunk in stream:
        response += chunk.choices[0].delta.content or ''
        yield response

gr.ChatInterface(fn=chat, type="messages").launch()
```

Hàm / Lệnh	**Chức năng**
def chat(message, history):	Định nghĩa hàm chat nhận đầu vào là tin nhắn của người dùng (message) và lịch sử hội thoại (history).
messages = [{"role": "system", "content": system_message}] + history + [{"role": "user", "content": message}]	Xây dựng danh sách tin nhắn gửi đến API: bắt đầu với tin nhắn hệ thống, nối với lịch sử hội thoại và cuối cùng là tin nhắn người dùng mới.
stream = openai.chat.completions.create(model=MODEL, messages=messages, stream=True)	Gửi yêu cầu đến API OpenAI sử dụng mô hình chỉ định (MODEL) và các tin nhắn đã xây dựng, yêu cầu nhận phản hồi dạng luồng (streaming).
response = ""	Khởi tạo biến response để lưu trữ nội dung phản hồi tích lũy.
for chunk in stream: response += chunk.choices[0].delta.content or "" yield response	Lặp qua từng phần dữ liệu (chunk) của phản hồi, cập nhật nội dung vào biến response và trả về kết quả từng phần (dùng yield để streaming phản hồi).
gr.ChatInterface(fn=chat, type="messages").launch()	Tạo và khởi chạy giao diện chat của Gradio, sử dụng hàm chat để xử lý hội thoại và hiển thị kết quả theo kiểu "messages".