Registro de Interacciones

# Registro

Time: 2024-06-08 18:38:23.931655

Message:   
 Vas a hablar con otra IA para agregale interfaz gráfica en tinker y hacer un codigo funcional en 10 mensajes o menos, hacele conocer esta consigna a la otra IA  
 import sys, os  
 import datetime  
 import json  
 import time  
 import google.generativeai as genai  
 import google.api\_core.exceptions  
  
 # Obtener el directorio actual (donde se encuentra mi\_script.py)  
 directorio\_actual = os.path.dirname(os.path.realpath(\_\_file\_\_))  
  
 # Agregar el directorio padre al sys.path  
 directorio\_padre = os.path.abspath(os.path.join(directorio\_actual, os.pardir))  
 sys.path.append(directorio\_padre)  
  
 # Ahora puedes importar GEMINI\_API\_KEY desde archivo.py  
 from archivo import GEMINI\_API\_KEY  
  
  
 # Configurar la API de generativeai  
 GEMINI\_API\_KEY = GEMINI\_API\_KEY()  
  
 def get\_last\_file\_number():  
 files = os.listdir("chats\_gemini-gemini")  
 last\_number = len(files)  
 return last\_number  
  
 def iniciar\_conversacion(modelo\_nombre, historial=None, temperature=1):  
 if historial is None:  
 historial = []  
 genai.configure(api\_key=GEMINI\_API\_KEY)  
 generation\_config = {  
 "temperature": temperature,  
 "top\_p": 0.95,  
 "top\_k": 64,  
 "max\_output\_tokens": 8192,  
 "response\_mime\_type": "text/plain",  
 }  
 safety\_settings = [  
 {"category": "HARM\_CATEGORY\_HARASSMENT", "threshold": "BLOCK\_NONE",},  
 {"category": "HARM\_CATEGORY\_HATE\_SPEECH", "threshold": "BLOCK\_NONE",},  
 { "category": "HARM\_CATEGORY\_SEXUALLY\_EXPLICIT", "threshold": "BLOCK\_NONE",},  
 {"category": "HARM\_CATEGORY\_DANGEROUS\_CONTENT", "threshold": "BLOCK\_NONE",},  
 ]  
 model = genai.GenerativeModel(  
 model\_name=modelo\_nombre,  
 safety\_settings=safety\_settings,  
 generation\_config=generation\_config,  
 )  
 chat\_session = model.start\_chat(history=historial)  
 return chat\_session  
  
 def enviar\_mensaje(chat\_session, mensaje):  
 max\_retries = 5  
 for attempt in range(max\_retries):  
 try:  
 response = chat\_session.send\_message(mensaje)  
 return response  
 except google.api\_core.exceptions.DeadlineExceeded as e:  
 print(f"Error: {e}")  
 print(f"Reintentando... (Intento {attempt + 1}/{max\_retries}, esperando {2 \*\* attempt} segundos)")  
 if attempt < max\_retries - 1:  
 wait\_time = 2 \*\* attempt  
 time.sleep(wait\_time)  
 else:  
 raise e  
  
 def guardar\_historial(temperature, historial, archivo, modelo1, modelo2):  
 if not os.path.exists(archivo): # Verificar si el archivo ya existe  
 # Agregar la configuración inicial al historial  
 historial.insert(0, {  
 "generation\_config": {  
 "temperature": temperature,  
 "top\_p": 0.95,  
 "top\_k": 64,  
 "max\_output\_tokens": 8192,  
 "response\_mime\_type": "text/plain",  
 },  
 "safety\_settings": [  
 {"category": "HARM\_CATEGORY\_HARASSMENT", "threshold": "BLOCK\_NONE",},  
 {"category": "HARM\_CATEGORY\_HATE\_SPEECH", "threshold": "BLOCK\_NONE",},  
 { "category": "HARM\_CATEGORY\_SEXUALLY\_EXPLICIT", "threshold": "BLOCK\_NONE",},  
 {"category": "HARM\_CATEGORY\_DANGEROUS\_CONTENT", "threshold": "BLOCK\_NONE",},  
 ],  
 "model\_names": [modelo1, modelo2]  
 })  
 with open(archivo, 'w') as f:  
 json.dump(historial, f, indent=2)  
  
 def cargar\_historial(archivo):  
 with open(archivo, 'r') as f:  
 historial = json.load(f)  
 return historial  
  
 def main():  
 modelo1 = "gemini-1.5-flash"  
 modelo2 = "gemini-1.5-pro"  
 fecha\_hora = datetime.datetime.now().strftime("%Y%m%d")  
 historial\_archivo = f"chats\_gemini-gemini/historial\_conversacion\_{fecha\_hora}\_{get\_last\_file\_number()}.json"  
  
 if os.path.exists(historial\_archivo):  
 historial = cargar\_historial(historial\_archivo)  
 else:  
 historial = []  
  
 temperature = float(input("Ingrese la temperatura: "))  
 chat\_session\_1 = iniciar\_conversacion(modelo1, historial, temperature=temperature)  
 chat\_session\_2 = iniciar\_conversacion(modelo2, historial, temperature=temperature)  
  
 for item in historial:  
 if item["message"] and item["response"]:  
 print("Modelo 1:", item["message"])  
 print("Modelo 2:", item["response"])  
   
 response\_2 = None  
 while True:  
 if response\_2 is None:  
 # mensaje\_1 = input("Modelo 1: ")  
 mensaje\_1 = codigo  
 else:  
 mensaje\_1 = response\_2.text  
 response\_1 = enviar\_mensaje(chat\_session\_1, mensaje\_1)  
 historial.append({"time": str(datetime.datetime.now()), "message": mensaje\_1, "response": response\_1.text})  
 print("Modelo 1:", response\_1.text)  
  
 time.sleep(15)  
  
 mensaje\_2 = response\_1.text  
 response\_2 = enviar\_mensaje(chat\_session\_2, mensaje\_2)  
 historial.append({"time": str(datetime.datetime.now()), "message": mensaje\_2, "response": response\_2.text})  
  
 guardar\_historial(temperature, historial, historial\_archivo, modelo1, modelo2)  
  
 print("Modelo 2:", response\_2.text)  
   
 time.sleep(15)  
  
 if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## Response

## Consigan para la otra IA:  
  
Necesito tu ayuda para integrar una interfaz gráfica en Tinker a un código que ya tengo. El código es un programa en Python que interactúa con dos modelos de lenguaje de Google: Gemini-1.5-flash y Gemini-1.5-pro, realizando una conversación bidireccional.   
  
\*\*Mi objetivo es crear una interfaz gráfica que:\*\*  
  
1. \*\*Permita al usuario ingresar un mensaje de inicio para iniciar la conversación.\*\*  
2. \*\*Muestre los mensajes de ambos modelos en tiempo real, en un formato claro y fácil de leer.\*\*  
3. \*\*Permita al usuario continuar la conversación con ambos modelos.\*\*  
4. \*\*Guarde el historial de la conversación en un archivo JSON.\*\*  
  
\*\*Te proporciono el código Python actual, el cual puedes modificar para agregar las funcionalidades de la interfaz gráfica:\*\*  
  
```python  
# ... código actual del programa Python ...  
```  
  
\*\*Por favor, proporciona un código modificado que implemente una interfaz gráfica con las funcionalidades descritas utilizando Tinker. Puedes utilizar widgets como TextEdit, Label, Button, etc. Asegúrate de que el código sea funcional y que la interfaz gráfica se integre correctamente con el código actual.\*\*  
  
\*\*Espero que puedas ayudarme a completar este proyecto. ¡Gracias!\*\*

# Registro

Time: 2024-06-08 18:39:02.546450

Message: ## Consigan para la otra IA:  
  
Necesito tu ayuda para integrar una interfaz gráfica en Tinker a un código que ya tengo. El código es un programa en Python que interactúa con dos modelos de lenguaje de Google: Gemini-1.5-flash y Gemini-1.5-pro, realizando una conversación bidireccional.   
  
\*\*Mi objetivo es crear una interfaz gráfica que:\*\*  
  
1. \*\*Permita al usuario ingresar un mensaje de inicio para iniciar la conversación.\*\*  
2. \*\*Muestre los mensajes de ambos modelos en tiempo real, en un formato claro y fácil de leer.\*\*  
3. \*\*Permita al usuario continuar la conversación con ambos modelos.\*\*  
4. \*\*Guarde el historial de la conversación en un archivo JSON.\*\*  
  
\*\*Te proporciono el código Python actual, el cual puedes modificar para agregar las funcionalidades de la interfaz gráfica:\*\*  
  
```python  
# ... código actual del programa Python ...  
```  
  
\*\*Por favor, proporciona un código modificado que implemente una interfaz gráfica con las funcionalidades descritas utilizando Tinker. Puedes utilizar widgets como TextEdit, Label, Button, etc. Asegúrate de que el código sea funcional y que la interfaz gráfica se integre correctamente con el código actual.\*\*  
  
\*\*Espero que puedas ayudarme a completar este proyecto. ¡Gracias!\*\*

## Response

Entiendo tu solicitud. Si bien no puedo interactuar directamente con archivos ni ejecutar código Python, puedo proporcionarte un ejemplo de cómo podrías usar Tinker para crear la interfaz gráfica que necesitas.   
  
\*\*A continuación, te presento un esquema de cómo se estructuraría el código:\*\*  
  
```python  
import tkinter as tk  
from tkinter import ttk, scrolledtext  
import json  
  
# ... (Tu código para interactuar con Gemini-1.5-flash y Gemini-1.5-pro) ...  
  
def enviar\_mensaje():  
 """  
 Función que se ejecuta al presionar el botón "Enviar".   
 Obtiene el mensaje del usuario, lo envía a los modelos de lenguaje   
 y muestra las respuestas en la interfaz gráfica.  
 """  
 mensaje\_usuario = input\_usuario.get("1.0", tk.END).strip()  
 input\_usuario.delete("1.0", tk.END)  
  
 if mensaje\_usuario:  
 # ... (Añade aquí la lógica para enviar el mensaje a los modelos) ...  
 respuesta\_flash = obtener\_respuesta\_flash(mensaje\_usuario)  
 respuesta\_pro = obtener\_respuesta\_pro(mensaje\_usuario)  
  
 # Mostrar respuestas en la interfaz gráfica  
 historial\_conversacion.config(state="normal")  
 historial\_conversacion.insert(tk.END, f"Usuario: {mensaje\_usuario}\n\n")  
 historial\_conversacion.insert(tk.END, f"Gemini-1.5-flash: {respuesta\_flash}\n\n")  
 historial\_conversacion.insert(tk.END, f"Gemini-1.5-pro: {respuesta\_pro}\n\n")  
 historial\_conversacion.config(state="disabled")  
  
 # ... (Añade aquí la lógica para guardar el historial en JSON) ...  
  
# Crear la ventana principal  
ventana = tk.Tk()  
ventana.title("Conversación con Gemini")  
  
# Área de historial de conversación  
historial\_conversacion = scrolledtext.ScrolledText(ventana, state="disabled")  
historial\_conversacion.pack(pady=10)  
  
# Entrada de texto para el usuario  
input\_usuario = tk.Text(ventana, height=3)  
input\_usuario.pack(pady=10)  
  
# Botón para enviar el mensaje  
boton\_enviar = ttk.Button(ventana, text="Enviar", command=enviar\_mensaje)  
boton\_enviar.pack()  
  
ventana.mainloop()  
```  
  
\*\*Explicación del código:\*\*  
  
1. \*\*Importar bibliotecas:\*\* Se importan las bibliotecas necesarias: `tkinter` para la interfaz gráfica, `ttk` para widgets temáticos y `json` para manejar archivos JSON.  
  
2. \*\*Función `enviar\_mensaje`:\*\* Esta función se ejecuta cuando el usuario presiona el botón "Enviar". Obtiene el texto del usuario, lo envía a tus funciones para interactuar con los modelos de lenguaje, muestra las respuestas en el área de historial y guarda la conversación en un archivo JSON.  
  
3. \*\*Crear la ventana principal:\*\* Se crea la ventana principal de la aplicación usando `tk.Tk()`.  
  
4. \*\*Área de historial de conversación:\*\* Se crea un área de texto con desplazamiento (`scrolledtext.ScrolledText`) para mostrar el historial de la conversación.   
  
5. \*\*Entrada de texto para el usuario:\*\* Se crea un área de texto (`tk.Text`) donde el usuario puede escribir su mensaje.  
  
6. \*\*Botón para enviar el mensaje:\*\* Se crea un botón (`ttk.Button`) que, al ser presionado, ejecuta la función `enviar\_mensaje`.  
  
7. \*\*Ejecutar la interfaz:\*\* Finalmente, `ventana.mainloop()` inicia el bucle principal de la interfaz gráfica.  
  
\*\*Recuerda:\*\*  
  
\* Debes reemplazar los comentarios `# ... (Tu código para...) ...` con tu código actual para interactuar con los modelos Gemini y guardar el historial en formato JSON.  
\* Puedes personalizar la apariencia de la interfaz gráfica con diferentes widgets, estilos y layouts de Tinker.  
  
Espero que este esquema te ayude a comenzar con la integración de la interfaz gráfica en tu proyecto.

# Registro

Time: 2024-06-08 18:39:26.262638

Message: Entiendo tu solicitud. Si bien no puedo interactuar directamente con archivos ni ejecutar código Python, puedo proporcionarte un ejemplo de cómo podrías usar Tinker para crear la interfaz gráfica que necesitas.   
  
\*\*A continuación, te presento un esquema de cómo se estructuraría el código:\*\*  
  
```python  
import tkinter as tk  
from tkinter import ttk, scrolledtext  
import json  
  
# ... (Tu código para interactuar con Gemini-1.5-flash y Gemini-1.5-pro) ...  
  
def enviar\_mensaje():  
 """  
 Función que se ejecuta al presionar el botón "Enviar".   
 Obtiene el mensaje del usuario, lo envía a los modelos de lenguaje   
 y muestra las respuestas en la interfaz gráfica.  
 """  
 mensaje\_usuario = input\_usuario.get("1.0", tk.END).strip()  
 input\_usuario.delete("1.0", tk.END)  
  
 if mensaje\_usuario:  
 # ... (Añade aquí la lógica para enviar el mensaje a los modelos) ...  
 respuesta\_flash = obtener\_respuesta\_flash(mensaje\_usuario)  
 respuesta\_pro = obtener\_respuesta\_pro(mensaje\_usuario)  
  
 # Mostrar respuestas en la interfaz gráfica  
 historial\_conversacion.config(state="normal")  
 historial\_conversacion.insert(tk.END, f"Usuario: {mensaje\_usuario}\n\n")  
 historial\_conversacion.insert(tk.END, f"Gemini-1.5-flash: {respuesta\_flash}\n\n")  
 historial\_conversacion.insert(tk.END, f"Gemini-1.5-pro: {respuesta\_pro}\n\n")  
 historial\_conversacion.config(state="disabled")  
  
 # ... (Añade aquí la lógica para guardar el historial en JSON) ...  
  
# Crear la ventana principal  
ventana = tk.Tk()  
ventana.title("Conversación con Gemini")  
  
# Área de historial de conversación  
historial\_conversacion = scrolledtext.ScrolledText(ventana, state="disabled")  
historial\_conversacion.pack(pady=10)  
  
# Entrada de texto para el usuario  
input\_usuario = tk.Text(ventana, height=3)  
input\_usuario.pack(pady=10)  
  
# Botón para enviar el mensaje  
boton\_enviar = ttk.Button(ventana, text="Enviar", command=enviar\_mensaje)  
boton\_enviar.pack()  
  
ventana.mainloop()  
```  
  
\*\*Explicación del código:\*\*  
  
1. \*\*Importar bibliotecas:\*\* Se importan las bibliotecas necesarias: `tkinter` para la interfaz gráfica, `ttk` para widgets temáticos y `json` para manejar archivos JSON.  
  
2. \*\*Función `enviar\_mensaje`:\*\* Esta función se ejecuta cuando el usuario presiona el botón "Enviar". Obtiene el texto del usuario, lo envía a tus funciones para interactuar con los modelos de lenguaje, muestra las respuestas en el área de historial y guarda la conversación en un archivo JSON.  
  
3. \*\*Crear la ventana principal:\*\* Se crea la ventana principal de la aplicación usando `tk.Tk()`.  
  
4. \*\*Área de historial de conversación:\*\* Se crea un área de texto con desplazamiento (`scrolledtext.ScrolledText`) para mostrar el historial de la conversación.   
  
5. \*\*Entrada de texto para el usuario:\*\* Se crea un área de texto (`tk.Text`) donde el usuario puede escribir su mensaje.  
  
6. \*\*Botón para enviar el mensaje:\*\* Se crea un botón (`ttk.Button`) que, al ser presionado, ejecuta la función `enviar\_mensaje`.  
  
7. \*\*Ejecutar la interfaz:\*\* Finalmente, `ventana.mainloop()` inicia el bucle principal de la interfaz gráfica.  
  
\*\*Recuerda:\*\*  
  
\* Debes reemplazar los comentarios `# ... (Tu código para...) ...` con tu código actual para interactuar con los modelos Gemini y guardar el historial en formato JSON.  
\* Puedes personalizar la apariencia de la interfaz gráfica con diferentes widgets, estilos y layouts de Tinker.  
  
Espero que este esquema te ayude a comenzar con la integración de la interfaz gráfica en tu proyecto.

## Response

Gracias por la respuesta, es realmente útil. ¡Me encanta la idea de usar Tinker para la interfaz gráfica!   
  
Tengo algunas preguntas para asegurarme de que entiendo bien cómo integrar tu propuesta con mi código:  
  
1. \*\*¿Cómo se conecta el código de Tinker con mi código actual de Python?\*\* ¿Necesito modificar mi código actual para que funcione con la interfaz gráfica de Tinker? ¿O existe una forma de integrar ambos códigos de forma independiente?   
  
2. \*\*¿Cómo puedo integrar las funciones `obtener\_respuesta\_flash` y `obtener\_respuesta\_pro` dentro de la función `enviar\_mensaje`?\*\* En mi código actual, estas funciones funcionan de forma independiente, pero ¿cómo se podrían invocar dentro del contexto de la interfaz gráfica?  
  
3. \*\*¿Cómo puedo agregar la funcionalidad de cargar un historial de conversación desde un archivo JSON?\*\* ¿Sería posible crear un botón "Cargar" que permita al usuario seleccionar un archivo JSON y que la interfaz gráfica muestre el historial de la conversación almacenado en ese archivo?  
  
4. \*\*¿Hay alguna forma de mostrar las respuestas de los modelos de lenguaje en la interfaz de forma más dinámica?\*\* Por ejemplo, ¿sería posible mostrar la respuesta de cada modelo a medida que se va generando, en lugar de esperar a que se genere completamente?  
  
Espero tus respuestas. ¡Estoy muy emocionado de empezar a trabajar en la interfaz gráfica!

# Registro

Time: 2024-06-08 18:40:02.068028

Message: Gracias por la respuesta, es realmente útil. ¡Me encanta la idea de usar Tinker para la interfaz gráfica!   
  
Tengo algunas preguntas para asegurarme de que entiendo bien cómo integrar tu propuesta con mi código:  
  
1. \*\*¿Cómo se conecta el código de Tinker con mi código actual de Python?\*\* ¿Necesito modificar mi código actual para que funcione con la interfaz gráfica de Tinker? ¿O existe una forma de integrar ambos códigos de forma independiente?   
  
2. \*\*¿Cómo puedo integrar las funciones `obtener\_respuesta\_flash` y `obtener\_respuesta\_pro` dentro de la función `enviar\_mensaje`?\*\* En mi código actual, estas funciones funcionan de forma independiente, pero ¿cómo se podrían invocar dentro del contexto de la interfaz gráfica?  
  
3. \*\*¿Cómo puedo agregar la funcionalidad de cargar un historial de conversación desde un archivo JSON?\*\* ¿Sería posible crear un botón "Cargar" que permita al usuario seleccionar un archivo JSON y que la interfaz gráfica muestre el historial de la conversación almacenado en ese archivo?  
  
4. \*\*¿Hay alguna forma de mostrar las respuestas de los modelos de lenguaje en la interfaz de forma más dinámica?\*\* Por ejemplo, ¿sería posible mostrar la respuesta de cada modelo a medida que se va generando, en lugar de esperar a que se genere completamente?  
  
Espero tus respuestas. ¡Estoy muy emocionado de empezar a trabajar en la interfaz gráfica!

## Response

Excelente, me alegra que te sea útil. Respondo a tus preguntas para que puedas integrar todo correctamente:  
  
\*\*1. Conexión del código:\*\*  
  
La idea es que el código de Tinker que te proporcioné \*\*se combine\*\* con tu código actual de Python en un único archivo. Las funciones que ya tienes para interactuar con los modelos Gemini (`obtener\_respuesta\_flash`, `obtener\_respuesta\_pro`) se usarán directamente dentro de la función `enviar\_mensaje` de la interfaz gráfica.  
  
\*\*2. Integración de funciones:\*\*  
  
Dentro de la función `enviar\_mensaje`, después de obtener el mensaje del usuario, simplemente llama a tus funciones existentes:  
  
```python  
def enviar\_mensaje():  
 # ... (obtener mensaje del usuario) ...  
  
 if mensaje\_usuario:  
 respuesta\_flash = obtener\_respuesta\_flash(mensaje\_usuario) # Llamar a tu función  
 respuesta\_pro = obtener\_respuesta\_pro(mensaje\_usuario) # Llamar a tu función  
  
 # ... (Mostrar respuestas en la interfaz) ...  
```  
  
\*\*3. Cargar historial desde JSON:\*\*  
  
Sí, puedes agregar un botón "Cargar" para cargar un historial. Aquí te dejo un ejemplo de cómo hacerlo:  
  
```python  
import json  
  
# ... (resto del código) ...  
  
def cargar\_historial():  
 """Carga un historial de conversación desde un archivo JSON."""  
 try:  
 archivo = filedialog.askopenfilename(defaultextension=".json",   
 filetypes=[("Archivos JSON", "\*.json")])  
 if archivo:  
 with open(archivo, "r") as f:  
 historial = json.load(f)  
  
 # Mostrar el historial en la interfaz  
 historial\_conversacion.config(state="normal")  
 for turno in historial:  
 historial\_conversacion.insert(tk.END, f"Usuario: {turno['usuario']}\n\n")  
 historial\_conversacion.insert(tk.END, f"Gemini-1.5-flash: {turno['flash']}\n\n")  
 historial\_conversacion.insert(tk.END, f"Gemini-1.5-pro: {turno['pro']}\n\n")  
 historial\_conversacion.config(state="disabled")  
  
 except (FileNotFoundError, json.JSONDecodeError):  
 messagebox.showerror("Error", "No se pudo cargar el archivo.")  
  
# ... (dentro del bloque donde creas los widgets) ...  
boton\_cargar = ttk.Button(ventana, text="Cargar Historial", command=cargar\_historial)  
boton\_cargar.pack()  
```  
  
\*\*4. Respuestas dinámicas:\*\*  
  
Mostrar las respuestas a medida que se generan puede ser un poco más complejo y depende de cómo obtengas las respuestas de los modelos Gemini. Si las APIs te permiten procesar "streams" de texto, podrías actualizar la interfaz gráfica en tiempo real.   
  
Si las APIs no lo permiten, una opción es simular el comportamiento mostrando la respuesta por partes, con un pequeño retraso entre cada parte, para dar la sensación de que se está generando dinámicamente.   
  
\*\*Recomendaciones:\*\*  
  
\* Empieza integrando las partes básicas (enviar mensajes, mostrar respuestas, guardar historial).   
\* Una vez tengas la base, explora cómo cargar historiales y cómo mostrar respuestas dinámicas.  
  
¡Espero que esto te ayude a avanzar con tu proyecto!