VIEWS.PY

A continuación se detallan los bloques lógicos más relevantes del controlador principal.

BLOQUE 1: CONSTRUCCIÓN DEL PERFIL DE USUARIO

Ubicación: Función kairos_main.

Este bloque es la esencia del proyecto, ya que ayuda a la personalización. Aquí se obtienen los datos de la base de datos y se transforman en lenguaje natural que se enviará a la api, esto permite que Gemini tenga en cuenta siempre los datos del usuario antes de responder.

CÓDIGO COMENTADO:

```
try: if metricas and config: # Traducimos los booleanos y números de la BD a lenguaje natural para la IA modo_txt = "ACTIVADO (Sé estricto)" if config.modo_competencia else "DESACTIVADO (Sé amable)"

    if config.modo_psicologico == 1: nivel_desc = "Nivel 1 (Técnico)"
  elif config.modo_psicologico == 2: nivel_desc = "Nivel 2 (Desafiante)"
  else: nivel_desc = "Nivel 3 (Empático)"

  # La IA recibirá esta "informacion" oculta antes de la pregunta del usuario
  contexto_usuario = f"""
  DATOS DEL ATLETA:
  - Nombre: {request.user.username}
  - Deporte: {metricas.deporte}
  - Nivel: {metricas.nivel}
  - Objetivo: {metricas.objetivo}
  - Peso/Altura: {metricas.peso}kg / {metricas.altura}m
  - MODO COMPETENCIA: {modo_txt}
  - MODO PSICOLÓGICO: {nivel_desc}
  """
else:
  contexto_usuario = "Usuario sin perfil configurado."


except Exception as ex_perfil: print(f"Error creando contexto usuario: {ex_perfil}")
```

BLOQUE 2: GESTIÓN DE MEMORIA CONVERSACIONAL (HISTORIAL)

Ubicación: Función kairos_main, antes de configurar el modelo.

Descripción: Gemini por defecto no tiene memoria, por lo que este bloque se encarga de recuperar las últimas interacciones de la base de datos, las convierte en estructura de tipo JSON que es el formato que requiere Google y las envía, de modo que la conversación sea fluida.

CÓDIGO COMENTADO:

```
historial_para_gemini = [] try: # Recuperamos los últimos 26 mensajes para no saturar el contexto (Token limit management) historial_db = Historial.objects.filter(usuario=request.user).order_by('-id')[:26]

# Invertimos el orden (reversed) para que la IA lea cronológicamente, de lo contrario pensará que el orden fue presente-pasado.
for h in reversed(historial_db):
    historial_para_gemini.append({"role": "user", "parts": [h.mensaje_usuario]})
    historial_para_gemini.append({"role": "model", "parts": [h.respuesta_ia]})


except: pass
```

BLOQUE 3: CONFIGURACIÓN DE SEGURIDAD (SAFETY SETTINGS) Ubicación: Función kairos_main, configuración del modelo.

Descripción: Dado que la aplicación trata temas deportivos (boxeo, artes marciales, fatiga muscular), es necesario ajustar los filtros de seguridad de la API para evitar falsos positivos de "violencia" o "daño físico".

La aplicación puede tratar temas deportivos (como artes marciales, heridas en entrenamiento, etc.) es necesario ajustar los filtros de seguridad de la API para evitar falsos positivos de "violencia" o "daño físico", lo que desactiva la API momentáneamente.

CÓDIGO COMENTADO:

```
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"}, ]
```

BLOQUE 4: INICIALIZACIÓN Y LLAMADA A LA API Ubicación: Función kairos_main.

Descripción: Aquí se instancia el cliente de Gemini con el prompt maestro (Contexto Base + Perfil Usuario) y se inicia la sesión de chat con el historial.

Nota: Se hizo uso del modelo gemini 2.5 flash por su rapidez, aunque podríamos cambiar de modelo si se requiere mayor análisis.

model = genai.GenerativeModel(model_name='models/gemini-2.5-flash', # Fusionamos el prompt maestrp (archivo txt) con los datos del usuario (BD) system_instruction=f"{contexto_base}\n\n{contexto_usuario}")

Se inicia el chat enviando el historial recuperado de la base de datos.

chat = model.start_chat(history=historial_para_gemini)

Se envía la solicitud del usuario

response = chat.send_message(mensaje_usuario, safety_settings=safety_settings)

BLOQUE 5: PERSISTENCIA DE DATOS Y MANEJO DE ERRORES Ubicación: Bloque try-except final y función intera_histo.

Descripción: Garantiza que la respuesta se guarde para futuras referencias y maneja posibles caídas de la API o bloqueos de seguridad.

try: respuesta_ia = response.text except Exception: # Fallback en caso de que la IA rechace responder por políticas de seguridad respuesta_ia = "Lo siento, mi sistema de seguridad se activó por error. Intenta reformular."

GUARDA LA INTERACCIÓN EN EL HISTORIAL.

intera_histo(request.user, mensaje_usuario, respuesta_ia)