# =========================================

# 🎨 Tema / Diseño

# =========================================

COL\_BG = "#FFF3DD" # Fondo general (igual al panel derecho)

COL\_BG\_ALT = "#F9E8C6" # Fondo alterno crema

COL\_ACCENT = "#F07C00" # Naranja principal

COL\_ACCENT\_DARK = "#D86E00" # Naranja oscuro para hover o encabezado

COL\_TEXT = "#4B2800" # Texto principal oscuro

COL\_MUTED = "#B9772A" # Texto secundario / etiquetas

COL\_BUBBLE\_USER = "#FFE7C2" # Burbuja del usuario (crema claro)

COL\_BUBBLE\_COACH = "#FFD6A0" # Burbuja del coach (naranja pálido)

FONT\_FAMILY = "Segoe UI"

RADIUS = 14

CHAT\_WIDTH = 520

CHAT\_HEIGHT = 580

# =========================================

# 🎙️ Audio / STT / TTS (robusto)

# =========================================

REC\_SECONDS = 6

REC\_SAMPLERATE = 16000

STT\_LANGUAGE = "es-MX" # reconocimiento en español (México)

TTS\_VOICE = "alloy" # voz neural OpenAI

TTS\_SPEED = 2.0 # ~2× más rápido (time-stretch)

TTS\_FILE = "tts.wav" # archivo temporal de salida TTS

AUDIO\_SELFTEST = True # beep corto al iniciar para asegurar salida

DEBUG\_AUDIO = False # prints de depuración en consola

# =========================================

# 🧠 Emoción → solo para badge/tono textual

# =========================================

TONO\_POR\_EMOCION = {

"triste": {"instruccion": "Responde con empatía, valida emociones y ofrece esperanza breve.", "emoji": "💙"},

"ansioso": {"instruccion": "Responde calmado y estructurado, con respiración y pasos cortos.", "emoji": "🫁"},

"estresado": {"instruccion": "Sereno y enfocado; sugiere micro-plan práctico (2–3 pasos).", "emoji": "🧘"},

"cansado": {"instruccion": "Suave y compasivo; sugiere descanso breve y paso pequeño.", "emoji": "😴"},

"enojado": {"instruccion": "Calma y contención; canaliza energía de forma útil.", "emoji": "⚡"},

"solo": {"instruccion": "Cálido; acompaña y refuerza conexión.", "emoji": "🫶"},

"agradecido": {"instruccion": "Afirmativo y luminoso; refuerza gratitud.", "emoji": "🌟"},

"amor": {"instruccion": "Afectuoso y amable; refuerza autocuidado y cuidado a otros.", "emoji": "❤️"},

"motivado": {"instruccion": "Energético y directo; impulsa mini-reto.", "emoji": "🔥"},

"feliz": {"instruccion": "Alegre y celebratorio; reconoce progreso.", "emoji": "🎉"},

}

EMOCIONES\_VALIDAS = list(TONO\_POR\_EMOCION.keys())

# =========================================

# 🧩 Utilidades UI

# =========================================

def draw\_gradient(canvas, x0, y0, x1, y1, c1, c2, steps=100):

r1, g1, b1 = canvas.winfo\_rgb(c1)

r2, g2, b2 = canvas.winfo\_rgb(c2)

r\_ratio = (r2 - r1) / steps

g\_ratio = (g2 - g1) / steps

b\_ratio = (b2 - b1) / steps

for i in range(steps):

nr = int(r1 + (r\_ratio \* i)); ng = int(g1 + (g\_ratio \* i)); nb = int(b1 + (b\_ratio \* i))

color = f"#{nr>>8:02x}{ng>>8:02x}{nb>>8:02x}"

y = y0 + (y1 - y0) \* i / steps

canvas.create\_rectangle(x0, y, x1, y + (y1 - y0) / steps + 1, outline="", fill=color)

def rounded\_rect(canvas, x, y, w, h, r, fill, outline=""):

canvas.create\_arc(x, y, x+2\*r, y+2\*r, start=90, extent=90, fill=fill, outline=outline)

canvas.create\_arc(x+w-2\*r, y, x+w, y+2\*r, start=0, extent=90, fill=fill, outline=outline)

canvas.create\_arc(x, y+h-2\*r, x+2\*r, y+h, start=180, extent=90, fill=fill, outline=outline)

canvas.create\_arc(x+w-2\*r, y+h-2\*r, x+w, y+h, start=270, extent=90, fill=fill, outline=outline)

canvas.create\_rectangle(x+r, y, x+w-r, y+h, fill=fill, outline=outline)

canvas.create\_rectangle(x, y+r, x+w, y+h-r, fill=fill, outline=outline)

def toast(root, text, kind="info", ms=2400):

bg = {"info": "#111827", "warn": "#422006", "err": "#3F1D1D"}.get(kind, "#111827")

fg = {"info": "#E5E7EB", "warn": "#FBBF24", "err": "#FCA5A5"}.get(kind, "#E5E7EB")

win = tk.Toplevel(root); win.overrideredirect(True); win.configure(bg=bg)

lbl = tk.Label(win, text=text, bg=bg, fg=fg, font=("Segoe UI", 10)); lbl.pack(ipadx=14, ipady=8)

win.lift(); win.update\_idletasks()

sw = root.winfo\_screenwidth(); sh = root.winfo\_screenheight()

ww = lbl.winfo\_reqwidth() + 28; wh = lbl.winfo\_reqheight() + 18

x = sw - ww - 24; y = sh - wh - 60

win.geometry(f"{ww}x{wh}+{x}+{y}")

win.after(ms, lambda: win.destroy())

# =========================================

# 🗨️ Bubbles

# =========================================

class Bubble(tk.Canvas):

def \_\_init\_\_(self, parent, text, side="coach", \*\*kwargs):

super().\_\_init\_\_(parent, bg=COL\_BG, bd=0, highlightthickness=0, \*\*kwargs)

self.text = text; self.side = side; self.width = CHAT\_WIDTH - 80; self.pad = 12

self.draw()

def draw(self):

self.delete("all")

text\_id = self.create\_text(self.pad + 6, self.pad + 4, text=self.text, font=("Segoe UI", 11),

fill=COL\_TEXT, width=self.width-2\*self.pad, anchor="nw")

bbox = self.bbox(text\_id); w = min(self.width, bbox[2] + self.pad + 6); h = bbox[3] + self.pad + 10

self.delete("all")

bubble\_color = COL\_BUBBLE\_COACH if self.side == "coach" else COL\_BUBBLE\_USER

rounded\_rect(self, 6, 8, w+18, h+22, RADIUS, fill="#0B0F1C")

rounded\_rect(self, 0, 0, w+12, h+16, RADIUS, fill=bubble\_color)

if self.side == "coach":

self.create\_polygon(12, h+14, 22, h+8, 22, h+18, fill=bubble\_color, outline=bubble\_color)

else:

self.create\_polygon(w+4, h+8, w+14, h+14, w+4, h+18, fill=bubble\_color, outline=bubble\_color)

self.create\_text(self.pad + 6, self.pad + 4, text=self.text, font=("Segoe UI", 11),

fill=COL\_TEXT, width=self.width-2\*self.pad, anchor="nw")

self.config(width=w+24, height=h+24)

# =========================================

# 🧭 Header

# =========================================

class Header(ttk.Frame):

def \_\_init\_\_(self, parent):

super().\_\_init\_\_(parent); self.configure(style="TFrame")

self.canvas = tk.Canvas(self, height=120, width=CHAT\_WIDTH, bd=0, highlightthickness=0, bg=COL\_BG)

self.canvas.pack(fill="x")

draw\_gradient(self.canvas, 0, 0, CHAT\_WIDTH, 120, "#0B1220", "#0E1B3B", 140)

self.canvas.create\_oval(18, 18, 82, 82, outline="", fill="#0F172A")

self.canvas.create\_text(50, 50, text="🩺", font=("Segoe UI", 32))

self.title\_id = self.canvas.create\_text(100, 34, text="Asistente de Bienestar", fill="#FFFFFF",

font=("Segoe UI", 18, "bold"), anchor="w")

self.sub\_id = self.canvas.create\_text(100, 64, text="¿Cómo estás hoy?", fill=COL\_MUTED,

font=("Segoe UI", 10), anchor="w")

self.badge = tk.Label(self, text="🙂 Neutral", bg="#0B1325", fg="#93C5FD",

font=("Segoe UI", 9), padx=10, pady=4)

self.badge.place(x=CHAT\_WIDTH-130, y=80)

def set\_status(self, text): self.canvas.itemconfigure(self.sub\_id, text=text)

def set\_emotion(self, emo\_tag): self.badge.configure(text=emo\_tag, bg="#0B1325", fg="#93C5FD")

# =========================================

# 💬 Área de chat

# =========================================

class ChatArea(ttk.Frame):

def \_\_init\_\_(self, parent):

super().\_\_init\_\_(parent); self.configure(style="TFrame")

self.canvas = tk.Canvas(self, width=CHAT\_WIDTH, height=CHAT\_HEIGHT, bg=COL\_BG, bd=0, highlightthickness=0)

self.scroll = ttk.Scrollbar(self, orient="vertical", command=self.canvas.yview)

self.inner = ttk.Frame(self); self.inner.configure(style="TFrame")

self.inner\_id = self.canvas.create\_window((0, 0), window=self.inner, anchor="nw")

self.canvas.configure(yscrollcommand=self.scroll.set)

self.canvas.pack(side="left", fill="both", expand=True)

self.scroll.pack(side="right", fill="y")

self.inner.bind("<Configure>", self.\_on\_configure)

self.canvas.bind("<Configure>", self.\_on\_canvas\_configure)

self.rows = 0

self.typing\_frame = ttk.Frame(self.inner)

self.typing\_label = tk.Label(self.typing\_frame, text="Escribiendo", fg=COL\_MUTED, bg=COL\_BG, font=("Segoe UI", 10))

self.dot\_count = 0; self.typing\_running = False

def \_on\_configure(self, \_): self.canvas.configure(scrollregion=self.canvas.bbox("all"))

def \_on\_canvas\_configure(self, event): self.canvas.itemconfig(self.inner\_id, width=event.width)

def add\_message(self, text, side="coach"):

row = ttk.Frame(self.inner); row.configure(style="TFrame")

ttk.Frame(self.inner, height=2, style="TFrame").grid(row=self.rows, column=0, sticky="ew"); self.rows += 1

bubble = Bubble(row, text=text, side=side)

bubble.pack(anchor="w" if side=="coach" else "e", padx=18, pady=4)

row.grid(row=self.rows, column=0, sticky="ew"); self.rows += 1

self.canvas.after(50, lambda: self.canvas.yview\_moveto(1.0))

def start\_typing(self):

if self.typing\_running: return

self.typing\_running = True

self.typing\_frame.grid(row=self.rows, column=0, sticky="w", padx=22, pady=(8, 0)); self.rows += 1

self.\_animate\_typing()

def \_animate\_typing(self):

if not self.typing\_running: return

self.dot\_count = (self.dot\_count + 1) % 4

self.typing\_label.configure(text="Escribiendo" + "." \* self.dot\_count)

if not self.typing\_label.winfo\_ismapped():

self.typing\_label.pack(side="left")

for \_ in range(3): tk.Label(self.typing\_frame, text="•", fg=COL\_MUTED, bg=COL\_BG).pack(side="left", padx=2)

self.after\_id = self.after(350, self.\_animate\_typing)

def stop\_typing(self):

self.typing\_running = False

try: self.after\_cancel(self.after\_id)

except: pass

try: self.typing\_frame.grid\_forget()

except: pass

# =========================================

# ⌨️ Barra de entrada

# =========================================

class InputBar(ttk.Frame):

def \_\_init\_\_(self, parent, on\_send, on\_mic):

super().\_\_init\_\_(parent); self.configure(style="TFrame")

self.on\_send = on\_send; self.on\_mic = on\_mic

wrap = tk.Frame(self, bg=COL\_BG, bd=0, highlightthickness=0); wrap.pack(fill="x", padx=12, pady=(8, 14))

box = tk.Canvas(wrap, height=56, bg=COL\_BG, bd=0, highlightthickness=0)

rounded\_rect(box, 2, 2, CHAT\_WIDTH-28, 54, 16, fill=COL\_BG\_ALT); box.pack(fill="x")

self.entry = tk.Entry(wrap, bg=COL\_BG\_ALT, fg=COL\_TEXT, insertbackground=COL\_TEXT, relief="flat", font=("Segoe UI", 11))

self.entry.place(x=24, y=18, width=CHAT\_WIDTH-28-24-120, height=22)

self.entry.bind("<Return>", lambda e: self.\_send())

self.mic\_btn = tk.Button(wrap, text="🎤", bg=COL\_BG\_ALT, fg=COL\_TEXT, relief="flat",

font=("Segoe UI", 12), activebackground=COL\_BG\_ALT, activeforeground=COL\_TEXT,

command=self.\_mic\_press, cursor="hand2")

self.mic\_btn.place(x=CHAT\_WIDTH-28-88, y=14, width=36, height=28)

self.send\_btn = tk.Button(wrap, text="➤", bg=COL\_ACCENT, fg="#0B1220", relief="flat",

font=("Segoe UI", 12, "bold"), activebackground=COL\_ACCENT\_DARK,

activeforeground="#0B1220", command=self.\_send, cursor="hand2")

self.send\_btn.place(x=CHAT\_WIDTH-28-44, y=12, width=40, height=32)

glow = tk.Canvas(wrap, width=70, height=70, bg=COL\_BG, bd=0, highlightthickness=0)

glow.place(x=CHAT\_WIDTH-28-58, y=0); glow.create\_oval(20, 10, 60, 50, fill="#193B2A", outline="")

def \_send(self):

text = self.entry.get().strip()

if not text: return

self.entry.delete(0, tk.END)

self.on\_send(text)

def \_mic\_press(self):

self.mic\_btn.configure(text="🔴")

self.after(200, lambda: self.mic\_btn.configure(text="🎤"))

self.on\_mic()

# =========================================

# 🔊 Utilidades de audio (robustas)

# =========================================

def \_debug(msg):

if DEBUG\_AUDIO: print(msg)

def audio\_output\_selftest():

"""Beep suave 0.2s para asegurarnos que hay salida de audio."""

if not AUDIO\_SELFTEST: return True

try:

fs = 24000

t = np.linspace(0, 0.20, int(fs\*0.20), False)

tone = 0.2 \* np.sin(2 \* math.pi \* 440 \* t) # 440 Hz

sd.play(tone.astype(np.float32), fs); sd.wait()

return True

except Exception as e:

\_debug(f"[SELFTEST] Falló salida audio: {e}")

return False

def play\_numpy\_audio(y: np.ndarray, sr: int):

"""Normaliza, asegura mono/float32 y reproduce bloqueante."""

if y is None or len(y) == 0:

raise RuntimeError("Buffer de audio vacío.")

if y.ndim > 1: y = y.mean(axis=1)

# normalizar a -1..1 con headroom

m = np.max(np.abs(y)) if np.max(np.abs(y)) > 0 else 1.0

y = (y / m) \* 0.9

y = y.astype(np.float32)

\_debug(f"[PLAY] sr={sr}, dur={len(y)/sr:.2f}s, peak={np.max(np.abs(y)):.3f}")

sd.play(y, sr); sd.wait()

def tts\_neural\_to\_wav(text: str, out\_path: str = TTS\_FILE):

"""Solicita TTS a OpenAI y guarda WAV. Devuelve ruta."""

if client is None:

raise RuntimeError("OPENAI\_API\_KEY no configurada.")

# Generar audio vía streaming (WAV)

with client.audio.speech.with\_streaming\_response.create(

model="gpt-4o-mini-tts",

voice=TTS\_VOICE,

input=text,

format="wav"

) as response:

response.stream\_to\_file(out\_path)

if not os.path.exists(out\_path) or os.path.getsize(out\_path) < 128:

raise RuntimeError("Archivo TTS no generado o vacío.")

return out\_path

def speak\_neural\_fast(text: str, speed: float = TTS\_SPEED):

"""

1) Pide TTS neural (WAV) a OpenAI.

2) Carga audio, valida y aplica time-stretch (~2×).

3) Reproduce con sounddevice.

"""

# Paso 1: generar wav

path = tts\_neural\_to\_wav(text, TTS\_FILE)

# Paso 2: leer y validar

y, sr = sf.read(path, dtype="float32") # conserva sr original

if y.ndim > 1: y = y.mean(axis=1) # mono

dur = len(y) / sr

\_debug(f"[TTS] archivo={path}, sr={sr}, dur={dur:.2f}s, samples={len(y)}")

if dur < 0.05:

raise RuntimeError("Duración de audio TTS demasiado corta.")

# Paso 2.5 (opcional): acelerar manteniendo tono

if speed and speed > 1.0:

try:

rate = float(speed)

if rate > 3.0: rate = 3.0

if rate < 1.1: rate = 1.1

y = librosa.effects.time\_stretch(y, rate=rate)

\_debug(f"[TTS] time-stretch aplicado: x{rate}")

except Exception as e:

\_debug(f"[TTS] fallo time-stretch: {e} — se reproduce a velocidad original.")

# Paso 3: reproducir

play\_numpy\_audio(y, sr)

# =========================================

# 🧠 App principal (misma lógica + bienestar)

# =========================================

class HeaderBar(Header): pass

class CoachApp(ttk.Frame):

def \_\_init\_\_(self, root):

super().\_\_init\_\_(root); self.configure(style="TFrame"); self.root = root

style = ttk.Style(self); style.theme\_use("clam")

style.configure("TFrame", background=COL\_BG)

style.configure("TLabel", background=COL\_BG, foreground=COL\_TEXT)

# UI

self.header = HeaderBar(self); self.header.pack(fill="x")

self.chat = ChatArea(self); self.chat.pack(fill="both", expand=True)

self.input = InputBar(self, on\_send=self.send\_text, on\_mic=self.press\_mic); self.input.pack(fill="x")

# Instrucción de sistema

self.historial = [

{"role": "system", "content":

("Eres un asistente de bienestar físico y emocional en español latino. Sé breve, claro y empático.\n"

"Si el usuario menciona dolor/molestia (espalda, cuello, hombro, rodilla, cadera, tobillo, etc.):\n"

"- Sugiere 1–3 ejercicios SUAVES, con pasos y repeticiones (2–3 series de 8–10 reps),\n"

"- Precauciones (mover sin dolor, no rebotes, respirar),\n"

"- Alternativa sentado si aplica,\n"

"- Si el dolor es fuerte/punzante/reciente por golpe o hay adormecimiento → sugiere consultar a un profesional.\n"

"Si no hay dolor, responde como coach motivacional (hidratación, pausas activas, respiración).\n"

"NO des diagnósticos médicos. Estilo cálido y directo.")

}

]

# Mensaje de entrada

self.chat.add\_message("Hola 👋 Soy tu asistente de bienestar. ¿Cómo estás hoy? ¿Alguna molestia que quieras mejorar?", side="coach")

# Autotest de salida de audio (beep)

if not audio\_output\_selftest():

toast(self.root, "No detecté salida de audio. Revisa altavoces/dispositivo.", "warn")

# ------ Interacciones ------

def send\_text(self, text):

self.chat.add\_message(text, side="user")

threading.Thread(target=self.pipeline\_response, args=(text,), daemon=True).start()

def press\_mic(self):

threading.Thread(target=self.capture\_and\_transcribe, daemon=True).start()

# ------ Mic → STT (6 s) ------

def capture\_and\_transcribe(self):

try:

self.header.set\_status("Escuchando 6 s…")

self.chat.add\_message("🎤 (grabando…)", side="coach")

data = sd.rec(int(REC\_SECONDS \* REC\_SAMPLERATE), samplerate=REC\_SAMPLERATE, channels=1, dtype="int16")

sd.wait()

wavio.write("input.wav", data, REC\_SAMPLERATE, sampwidth=2)

r = sr.Recognizer()

with sr.AudioFile("input.wav") as source:

audio = r.record(source)

texto = r.recognize\_google(audio, language=STT\_LANGUAGE)

self.chat.add\_message(texto, side="user")

threading.Thread(target=self.pipeline\_response, args=(texto,), daemon=True).start()

except Exception as e:

toast(self.root, f"No pude escuchar/entender. ({e})", "warn")

finally:

self.header.set\_status("¿Cómo estás hoy?")

# ------ Pipeline: emoción + intención → respuesta + voz neural rápida ------

def pipeline\_response(self, user\_text):

try:

if client is None:

self.chat.add\_message("(Configura OPENAI\_API\_KEY para habilitar IA y voz).", side="coach")

return

self.header.set\_status("Analizando…")

self.chat.start\_typing()

# 1) Emoción (UI/tono textual)

emocion = self.detect\_emotion(user\_text)

emo\_info = TONO\_POR\_EMOCION.get(emocion, {"instruccion": "Sé empático, positivo y breve.", "emoji": "🙂"})

self.header.set\_emotion(f"{emo\_info['emoji']} {emocion.capitalize()}")

# 2) Intención de bienestar (¿área/severidad?)

area, severidad = self.extract\_wellness\_intent(user\_text)

directrices = (f"Contexto bienestar: área='{area}', severidad='{severidad}'. "

"Si hay área ≠ 'ninguna', sugiere ejercicios suaves específicos para esa zona.")

msgs = self.historial + [

{"role": "system", "content": f"Adecua tu tono a la emoción detectada: {emocion}. {emo\_info['instruccion']}"},

{"role": "system", "content": directrices},

{"role": "user", "content": user\_text}

]

self.header.set\_status("Pensando…")

resp = client.chat.completions.create(model="gpt-3.5-turbo", messages=msgs, max\_tokens=260)

texto = resp.choices[0].message.content.strip()

self.historial.append({"role": "user", "content": user\_text})

self.historial.append({"role": "assistant", "content": texto})

self.chat.stop\_typing()

self.chat.add\_message(texto, side="coach")

# 3) Voz neural rápida (con checks)

self.header.set\_status("Hablando…")

try:

speak\_neural\_fast(texto, speed=TTS\_SPEED)

except Exception as e:

\_debug(f"[VOICE] Falla TTS/play: {e}")

toast(self.root, f"No pude reproducir la voz. {e}", "warn")

self.header.set\_status("¿Cómo estás hoy?")

except Exception as e:

self.chat.stop\_typing()

toast(self.root, f"Error: {e}", "err")

self.header.set\_status("¿Cómo estás hoy?")

# ------ Emoción (badge/tono textual) ------

def detect\_emotion(self, text):

try:

if client is None:

return "motivado"

cls = client.chat.completions.create(

model="gpt-3.5-turbo",

temperature=0,

messages=[

{"role": "system", "content":

"Clasifica la emoción principal del texto del usuario en UNA sola palabra de esta lista exacta: "

"feliz, triste, ansioso, enojado, estresado, cansado, motivado, solo, agradecido, amor. "

"Responde solo con la etiqueta, sin explicaciones."},

{"role": "user", "content": text}

],

max\_tokens=3

).choices[0].message.content.strip().lower()

return cls if cls in EMOCIONES\_VALIDAS else "motivado"

except:

return "motivado"

# ------ Intención de bienestar (área/severidad) ------

def extract\_wellness\_intent(self, text):

try:

if client is None:

return "ninguna", "desconocida"

res = client.chat.completions.create(

model="gpt-3.5-turbo",

temperature=0,

messages=[

{"role": "system", "content":

("Analiza el texto del usuario. "

"Si menciona dolor o molestia, devuelve solo dos palabras separadas por coma: "

"AREA,SEVERIDAD. "

"AREA ∈ {espalda, cuello, hombro, rodilla, cadera, tobillo, brazo, mano, general, ninguna}. "

"SEVERIDAD ∈ {leve, moderada, severa, desconocida}. "

"Si no hay molestia, responde: 'ninguna,desconocida'. Sin explicaciones.")},

{"role": "user", "content": text}

],

max\_tokens=10

).choices[0].message.content.strip().lower()

if "," in res:

area, sev = [p.strip() for p in res.split(",", 1)]

if area not in {"espalda","cuello","hombro","rodilla","cadera","tobillo","brazo","mano","general","ninguna"}:

area = "ninguna"

if sev not in {"leve","moderada","severa","desconocida"}:

sev = "desconocida"

return area, sev

return "ninguna", "desconocida"

except:

return "ninguna", "desconocida"

# =========================================

# 🚀 Standalone (opcional)

# =========================================

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

root.title("Asistente de Bienestar — Conversación (TTS Neural Rápida)")

root.configure(bg=COL\_BG)

W, H = CHAT\_WIDTH, CHAT\_HEIGHT + 200

sw = root.winfo\_screenwidth(); sh = root.winfo\_screenheight()

root.geometry(f"{W}x{H}+{(sw-W)//2}+{(sh-H)//2}")

root.minsize(460, 620)

app = CoachApp(root)

app.pack(fill="both", expand=True)

root.mainloop()