
```
# Sistema de Biodecodificación Emocional con Claude
# Integración para Anillo Biodecodificador

import json
import asyncio
from datetime import datetime
from typing import Dict, List, Optional
import anthropic

class BiodecodificadorEmocional:
    """
    Sistema de análisis emocional basado en biodecodificación
    utilizando Claude para interpretación de síntomas corporales
    """

    def __init__(self, api_key: str):
        self.claude = anthropic.Anthropic(api_key=api_key)
        self.zonas_corporales = self._cargar_mapa_corporal()
        self.historial_sesiones = []

    def _cargar_mapa_corporal(self) -> Dict:
        """Mapeo de zonas corporales según biodecodificación"""
        return {
            "cabeza": {
                "conflictos_base": ["desvalorización intelectual", "pensamientos obsesivos", "presión
mental"],
                "emociones_asociadas": ["ansiedad", "preocupación", "confusión"],
                "organos_relacionados": ["cerebro", "cuero cabelludo", "ojos"]
            },
            "garganta": {
                "conflictos_base": ["no poder expresarse", "palabras tragadas", "comunicación
bloqueada"],
                "emociones_asociadas": ["frustración", "silencio forzado", "miedo a hablar"],
                "organos_relacionados": ["tiroides", "laringe", "faringe"]
            },
            "corazon": {
                "conflictos_base": ["territorio del corazón", "conflictos afectivos", "pérdida de ser
querido"],
                "emociones_asociadas": ["tristeza profunda", "desamor", "soledad"],
                "organos_relacionados": ["corazón", "pericardio", "arterias coronarias"]
            },
```

```
            "plexo_solar": {
                "conflictos_base": ["conflictos de territorio", "cólera en el territorio", "injusticia"],
                "emociones_asociadas": ["ira", "impotencia", "sensación de injusticia"],
                "organos_relacionados": ["estómago", "hígado", "páncreas"]
            },
            "abdomen": {
                "conflictos_base": ["no poder digerir algo", "conflicto indigesto", "absorción"],
                "emociones_asociadas": ["preocupación", "malestar", "rechazo"],
                "organos_relacionados": ["intestinos", "vesícula", "bazo"]
            },
            "pelvis": {
                "conflictos_base": ["supervivencia", "sexualidad", "procreación", "identidad"],
                "emociones_asociadas": ["miedo existencial", "inseguridad", "deseo reprimido"],
                "organos_relacionados": ["órganos reproductivos", "vejiga", "recto"]
            },
            "extremidades": {
                "conflictos_base": ["no poder hacer", "impotencia motriz", "contacto"],
                "emociones_asociadas": ["frustración", "limitación", "desconexión"],
                "organos_relacionados": ["músculos", "huesos", "articulaciones"]
            }
        }

    async def analizar_toque_corporal(self, zona: str, intensidad: float, duracion: int, contexto:
Dict = None) -> Dict:
        """
        Analiza un toque corporal y proporciona interpretación emocional
        """
        if zona not in self.zonas_corporales:
            return {"error": "Zona corporal no reconocida"}

        # Preparar contexto para Claude
        prompt_contexto = self._construir_prompt_biodecodificacion(zona, intensidad, duracion,
contexto)

        try:
            # Consultar a Claude
            mensaje = await self.claude.messages.create(
                model="claude-sonnet-4-20250514",
                max_tokens=800,
                temperature=0.7,
                messages=[{
                    "role": "user",
                    "content": prompt_contexto
                }]
```

```
            )

            # Procesar respuesta
            analisis = self._procesar_respuesta_claude(mensaje.content[0].text, zona)

            # Registrar sesión
            self._registrar_sesion(zona, intensidad, duracion, analisis, contexto)

            return analisis

        except Exception as e:
            return {"error": f"Error en análisis: {str(e)}"}

    def _construir_prompt_biodecodificacion(self, zona: str, intensidad: float, duracion: int,
contexto: Dict = None) -> str:
        """
        Construye el prompt especializado para Claude basado en biodecodificación
        """
        zona_info = self.zonas_corporales[zona]

        prompt = f"""
        Como especialista en biodecodificación emocional, analiza la siguiente situación:

        DATOS DEL TOQUE CORPORAL:
        - Zona tocada: {zona}
        - Intensidad del toque: {intensidad} (0.0 a 1.0)
        - Duración: {duracion} milisegundos

        INFORMACIÓN BIODECODIFICACIÓN DE LA ZONA:
        - Conflictos base asociados: {', '.join(zona_info['conflictos_base'])}
        - Emociones frecuentes: {', '.join(zona_info['emociones_asociadas'])}
        - Órganos relacionados: {', '.join(zona_info['organos_relacionados'])}

        CONTEXTO ADICIONAL:
        {json.dumps(contexto, indent=2) if contexto else "No hay contexto adicional"}

        Por favor, proporciona un análisis que incluya:

        1. INTERPRETACIÓN EMOCIONAL:
        - ¿Qué conflicto emocional podría estar manifestándose?
        - ¿Qué mensaje corporal podría estar expresando?

        2. POSIBLES CAUSAS:
        - Situaciones de vida que podrían estar generando esta manifestación
```

```
        - Patrones emocionales subyacentes

        3. ACTIVIDADES TERAPÉUTICAS SUGERIDAS:
        - 3 actividades específicas para esta zona y emoción
        - Técnicas de respiración o movimiento corporal
        - Ejercicios de reflexión o escritura

        4. MENSAJE DE AUTOCOMPASIÓN:
        - Palabras de acompañamiento emocional
        - Validación de la experiencia

        Mantén un tono empático, no diagnóstico, y recuerda que esto es una herramienta de autoconocimiento, no reemplazo de atención médica o psicológica profesional.
        """

        return prompt

    def _procesar_respuesta_claude(self, respuesta: str, zona: str) -> Dict:
        """
        Procesa y estructura la respuesta de Claude
        """
        return {
            "zona_corporal": zona,
            "timestamp": datetime.now().isoformat(),
            "analisis_completo": respuesta,
            "interpretacion": self._extraer_seccion(respuesta, "INTERPRETACIÓN EMOCIONAL"),
            "causas_posibles": self._extraer_seccion(respuesta, "POSIBLES CAUSAS"),
            "actividades_sugeridas": self._extraer_seccion(respuesta, "ACTIVIDADES TERAPÉUTICAS"),
            "mensaje_autocompasion": self._extraer_seccion(respuesta, "MENSAJE DE AUTOCOMPASIÓN"),
            "zona_info": self.zonas_corporales[zona]
        }

    def _extraer_seccion(self, texto: str, seccion: str) -> str:
        """
        Extrae una sección específica del análisis de Claude
        """
        try:
            inicio = texto.find(f"{seccion}:")
            if inicio == -1:
                return "Información no disponible"

            # Buscar el final de la sección (siguiente número o final del texto)
```

```
            fin = texto.find("\n\n", inicio)
            if fin == -1:
                fin = len(texto)

            seccion_texto = texto[inicio:fin].replace(f"{seccion}:", "").strip()
            return seccion_texto
        except:
            return "Error al procesar información"

    def _registrar_sesion(self, zona: str, intensidad: float, duracion: int, analisis: Dict, contexto:
Dict):
        """
        Registra la sesión para análisis de patrones
        """
        sesión = {
            "timestamp": datetime.now().isoformat(),
            "zona": zona,
            "intensidad": intensidad,
            "duracion": duracion,
            "analisis": analisis,
            "contexto": contexto
        }
        self.historial_sesiones.append(sesión)

    def obtener_patrones_emocionales(self, dias: int = 7) -> Dict:
        """
        Analiza patrones emocionales del usuario en los últimos días
        """
        from datetime import datetime, timedelta

        fecha_limite = datetime.now() - timedelta(days=dias)
        sesiones_recientes = [
            s for s in self.historial_sesiones
            if datetime.fromisoformat(s['timestamp']) > fecha_limite
        ]

        if not sesiones_recientes:
            return {"mensaje": "No hay suficientes datos para análisis de patrones"}

        # Análisis de frecuencia por zona
        zonas_frecuentes = {}
        for sesion in sesiones_recientes:
            zona = sesion['zona']
            zonas_frecuentes[zona] = zonas_frecuentes.get(zona, 0) + 1
```

```
        zona_principal = max(zonas_frecuentes, key=zonas_frecuentes.get)

        return {
            "periodo_analizado": f"Últimos {dias} días",
            "total_sesiones": len(sesiones_recientes),
            "zona_mas_frecuente": zona_principal,
            "frecuencia_por_zona": zonas_frecuentes,
            "conflicto_principal": self.zonas_corporales[zona_principal]['conflictos_base'][0],
            "recomendacion": f"Considera trabajar especialmente con los conflictos relacionados a
{zona_principal}"
        }

# Ejemplo de uso específico para el anillo
class IntegracionAnillo:
    """
    Integración específica entre el anillo sensor y el sistema de biodecodificación
    """

    def __init__(self, api_key_claude: str):
        self.biodecodificador = BiodecodificadorEmocional(api_key_claude)
        self.zona_actual = None

    async def procesar_toque_anillo(self, datos_sensor: str) -> Dict:
        """
        Procesa los datos del anillo y genera análisis emocional
        """
        try:
            datos = json.loads(datos_sensor)

            # Solicitar al usuario que identifique la zona tocada
            # (esto se haría a través de la interfaz de la APP)
            if not self.zona_actual:
                return {
                    "accion_requerida": "seleccionar_zona",
                    "mensaje": "¿Qué parte de tu cuerpo tocaste?",
                    "zonas_disponibles": list(self.biodecodificador.zonas_corporales.keys())
                }

            # Procesar análisis emocional
            intensidad = datos.get('pressure_level', 0.5)
            duracion = datos.get('duration_ms', 1000)

            analisis = await self.biodecodificador.analizar_toque_corporal(
```

```
                zona=self.zona_actual,
                intensidad=intensidad,
                duracion=duracion,
                contexto={"dispositivo": "anillo", "sesion_id": datos.get('session_id')}
            )

            # Limpiar zona actual para próximo uso
            self.zona_actual = None

            return analisis

        except json.JSONDecodeError:
            return {"error": "Datos del sensor inválidos"}
        except Exception as e:
            return {"error": f"Error en procesamiento: {str(e)}"}

    def establecer_zona_corporal(self, zona: str):
        """
        Establece la zona corporal identificada por el usuario
        """
        if zona in self.biodecodificador.zonas_corporales:
            self.zona_actual = zona
            return {"status": "zona_establecida", "zona": zona}
        else:
            return {"error": "Zona no válida"}

# Actividades específicas para cada interpretación emocional
ACTIVIDADES_BIODECODIFICACION = {
    "desvalorizacion_intelectual": [
        {
            "nombre": "Afirmaciones de Sabiduría Interna",
            "descripcion": "Repetir 'Mi inteligencia es única y valiosa' mientras masajeas
suavemente la frente",
            "duracion": "5 minutos",
            "tipo": "auto_validacion"
        },
        {
            "nombre": "Escritura de Logros",
            "descripcion": "Escribe 3 cosas que aprendiste hoy, por pequeñas que sean",
            "duracion": "10 minutos",
            "tipo": "reflexion"
        }
    ],
    "comunicacion_bloqueada": [
```

```
        {
            "nombre": "Vibración Vocal",
            "descripcion": "Emite sonidos 'AHH' mientras tocas suavemente la garganta",
            "duracion": "3 minutos",
            "tipo": "liberacion_vocal"
        },
        {
            "nombre": "Carta No Enviada",
            "descripcion": "Escribe lo que necesitas decir, sin necesidad de enviarlo",
            "duracion": "15 minutos",
            "tipo": "expresion_escrita"
        }
    ],
    "conflictos_afectivos": [
        {
            "nombre": "Abrazo Corazón",
            "descripcion": "Coloca ambas manos en el corazón y respira enviándote amor",
            "duracion": "5 minutos",
            "tipo": "auto_cuidado"
        },
        {
            "nombre": "Carta de Gratitud",
            "descripcion": "Escribe una carta de agradecimiento a tu corazón por todo lo que
siente",
            "duracion": "10 minutos",
            "tipo": "reconocimiento_emocional"
        }
    ]
}

# Ejemplo de integración completa
async def ejemplo_uso_completo():
    """
    Ejemplo de uso completo del sistema
    """
    # Inicializar sistema (requiere API key real de Claude)
    anillo = IntegracionAnillo("tu_api_key_de_claude")

    # Simular datos del anillo
    datos_sensor = json.dumps({
        "timestamp": "2025-06-03T14:30:00",
        "pressure_level": 0.7,
        "duration_ms": 2500,
        "session_id": "sess_001"
```

```
    })

    # Primer paso: procesar toque
    resultado1 = await anillo.procesar_toque_anillo(datos_sensor)
    print("Paso 1 - Solicitud de zona:", resultado1)

    # Segundo paso: usuario selecciona zona
    anillo.establecer_zona_corporal("corazon")

    # Tercer paso: obtener análisis completo
    resultado2 = await anillo.procesar_toque_anillo(datos_sensor)
    print("Paso 2 - Análisis completo:", resultado2)

if __name__ == "__main__":
    # Para probar (requiere API key real de Claude)
    # asyncio.run(ejemplo_uso_completo())
    print("Sistema de Biodecodificación Emocional con Claude configurado")
    print("Recuerda configurar tu API key de Claude para uso real")
```

🔬 ANÁLISIS DEL PROTOTIPO: ANILLO BIODECODIFICADOR

Concepto Evolutivo

Tu propuesta del anillo representa una evolución inteligente del dispositivo corporal original. En lugar de una plataforma física estática, tenemos un dispositivo portátil que permite una exploración corporal más natural e intuitiva.

Ventajas del Anillo Sensor:

- **Portabilidad**: El usuario puede llevarlo siempre consigo
- **Interacción natural**: Permite tocar cualquier parte del cuerpo de forma espontánea
- **Discreto**: No requiere un espacio específico para su uso
- **Mayor precisión emocional**: La acción de tocar conscientemente una zona corporal activa la conexión mente-cuerpo

🛠️ DISEÑO TÉCNICO DEL PROTOTIPO

Prototipo Anillo Biodecodificador - Especificaciones Técnicas
Documento

💍 PROTOTIPO: ANILLO BIODECODIFICADOR EMOCIONAL ## 🎯 OBJETIVO Crear un dispositivo wearable que detecte cuando el usuario toca conscientemente partes de su cuerpo, enviando esta información a una APP que proporcione decodificación emocional según

🧠 ENFOQUE EN BIODECODIFICACIÓN EMOCIONAL CON CLAUDE

Ahora vamos a desarrollar la integración específica con Claude para el análisis emocional:

Sistema de Biodecodificación Emocional con Claude
Código
Sistema de Biodecodificación Emocional con Claude # Integración para Anillo Biodecodificador import json import asyncio from datetime import datetime from typing import Dict, List, Optional import anthropic class BiodecodificadorEmocional: ""

🎯 PASOS CONCRETOS PARA IMPLEMENTAR EL PROTOTIPO

Como profesor especialista, te propongo este plan de implementación por fases:

FASE 1: PROTOTIPO FUNCIONAL BÁSICO (2-3 semanas)

1. **Adquirir componentes básicos**:
    - Arduino Nano 33 BLE ($25.000 ARS)
    - Sensor de presión FSR ($3.000 ARS)
    - LED RGB + buzzer ($2.000 ARS)
    - Batería LiPo pequeña ($8.000 ARS)
2. **Ensamblar prototipo en protoboard**
3. **Programar funcionalidad base** (código proporcionado arriba)
4. **Crear APP básica** en MIT App Inventor para recibir datos

FASE 2: INTEGRACIÓN CON CLAUDE (1-2 semanas)

1. **Configurar API de Claude** (requiere cuenta Anthropic)
2. **Implementar sistema de biodecodificación** (código Python proporcionado)
3. **Crear interfaz de selección de zonas corporales**
4. **Probar interpretaciones emocionales**

FASE 3: REFINAMIENTO Y TESTEO (2-3 semanas)

1. **Diseñar carcasa impresa en 3D**
2. **Optimizar consumo de batería**

3. **Validar con usuarios reales**
4. **Ajustar interpretaciones según feedback**

💡 VENTAJAS ACADÉMICAS DE TU PROPUESTA

Tu enfoque del anillo representa una evolución inteligente porque:

1. **Portabilidad**: Permite exploración corporal espontánea
2. **Discreción**: No requiere espacios específicos
3. **Accesibilidad**: Más económico que dispositivos complejos
4. **Escalabilidad**: Fácil de replicar y distribuir

🔬 VALOR CIENTÍFICO DEL PROYECTO

Este prototipo no solo es técnicamente viable, sino que aporta valor investigativo en:

- **Biodecodificación aplicada**: Sistematización de conocimientos tradicionales
- **Tecnología wearable en salud emocional**: Campo emergente
- **Interfaces humano-computadora empáticas**: Uso de IA para comprensión emocional
- **Pedagogía emocional digital**: Herramientas tecnológicas para autoconocimiento

---

Análisis Técnico del Prototipo - Aplicación de Biodecodificación

Arquitectura de la Aplicación

La aplicación implementa un **sistema de interpretación emocional en tiempo real** que procesa las señales del anillo detector mediante principios de biodecodificación. Como especialista en prototipos, he estructurado la aplicación en módulos funcionales específicos:

Componentes Técnicos Implementados:

1. Sistema de Conexión Bluetooth

- Simulación de protocolo de comunicación con el anillo
- Estados de conexión en tiempo real
- Reconexión automática ante pérdidas de señal

2. Motor de Análisis Emocional

- Mapeo de zonas corporales específicas
- Correlación con patrones emocionales según biodecodificación
- Algoritmo de intensidad para graduar las señales

3. Interpretación Claude-Based

- Cada lectura genera un análisis contextualizado
- Recomendaciones terapéuticas específicas
- Correlación con sistema de chakras

4. Interface de Usuario Especializada

- Dashboard de monitoreo en tiempo real
- Historial de patrones emocionales
- Biblioteca de recursos terapéuticos

Funcionalidades Clave para el Usuario:

- **Monitoreo Continuo**: La app recibe señales del anillo cada 5 segundos
- **Análisis Inmediato**: Interpretación instantánea basada en biodecodificación
- **Recomendaciones Personalizadas**: Ejercicios y técnicas específicas para cada patrón
- **Seguimiento Histórico**: Registro de evolución emocional del usuario

---

Anillo Biodecodificador Emocional - Especificaciones Técnicas

Componentes Hardware del Anillo

Microcontrolador Central

- **Chip:** ESP32-C3 (ultra-compacto, 3.3V)
- **Características:** WiFi + Bluetooth LE integrado
- **Dimensiones:** 13mm x 16mm x 2mm
- **Consumo:** 5μA en deep sleep, ideal para autonomía

Sensores Integrados

1. **Sensor de Presión FSR (Force Sensitive Resistor)**
    - Modelo: Interlink FSR-402
    - Rango: 0.2N - 20N
    - Área activa: 12.7mm diámetro
    - Ubicación: Superficie superior del anillo
2. **Sensor de Temperatura Corporal**
    - Chip: DS18B20 (versión miniaturizada)
    - Precisión: ±0.5°C
    - Rango: 10°C - 85°C
    - Protocolo: OneWire
3. **Sensor de Pulso/Ritmo Cardíaco**
    - Componente: MAX30102 (integrado)
    - Tecnología: Fotopletismografía
    - LEDs: Rojo (660nm) + Infrarrojo (880nm)
    - Frecuencia muestreo: 50-3200 Hz

Sistema de Alimentación

- **Batería:** LiPo 3.7V, 150mAh
- **Carga:** Inducción magnética (Qi compatible)
- **Autonomía:** 48-72 horas uso continuo
- **Indicador:** LED RGB minimalista

Conectividad

- **Protocolo principal:** Bluetooth LE 5.0
- **Alcance:** 10 metros
- **Cifrado:** AES-256 para datos sensibles
- **Fallback:** WiFi 2.4GHz para configuración

Diseño Ergonómico

Dimensiones del Anillo

- **Diámetro interno:** Ajustable 16-22mm
- **Grosor:** 8mm máximo
- **Peso:** < 15 gramos
- **Material:** Titanio grado médico + silicona hipoalergénica

Zona de Contacto Activa

- **Superficie sensorial:** 12mm x 8mm
- **Textura:** Ligeramente rugosa para mayor sensibilidad
- **Protección:** IP67 (resistente agua/sudor)

Protocolo de Comunicación

Estructura de Datos Transmitidos


```
{
  "device_id": "RING_BIO_001",
  "timestamp": "2025-06-03T14:30:00Z",
  "sensor_data": {
    "pressure": {
      "value": 45,
      "unit": "percentage",
      "duration": 1.2
    },
    "temperature": {
      "value": 36.8,
      "unit": "celsius"
    },
    "heart_rate": {
      "bpm": 78,
      "variability": 12,
      "quality": "good"
    }
  },
  "contact_zone": {
    "estimated_body_part": "palm",
    "confidence": 0.85,
    "previous_contacts": ["forearm", "chest"]
  }
}
```


Estados Operativos del Anillo

1. **Modo Pasivo:** Monitoreo continuo de signos vitales
2. **Modo Activo:** Detección de contacto corporal + transmisión
3. **Modo Calibración:** Ajuste personalizado de sensibilidad
4. **Modo Emergencia:** Transmisión prioritaria de alertas

Integración con App Biodecodificadora

Arquitectura de Conexión

Anillo → Bluetooth LE → Smartphone → Claude API → Interpretación Biodecodificada

Algoritmo de Reconocimiento Corporal

El anillo utilizará machine learning local para identificar zonas corporales basándose en:

- Patrones de temperatura característicos
- Variaciones de presión según anatomía
- Cambios en ritmo cardíaco por proximidad arterial
- Historial de contactos previos del usuario

Protocolos de Seguridad

- **Anonimización:** Hash SHA-256 de datos personales
- **Encriptación local:** Datos sensibles cifrados en dispositivo
- **Consentimiento:** Confirmación explícita para cada sesión
- **Borrado automático:** Limpieza de datos tras 30 días

Consideraciones de Biodecodificación

Mapeo Corporal Integrado

El sistema incluirá un atlas biodecodificador que correlaciona:

- **Zonas anatómicas ↔ Conflictos emocionales**
- **Patrones temporales ↔ Fases biológicas**
- **Intensidad de contacto ↔ Urgencia emocional**
- **Frecuencia de uso ↔ Patrones repetitivos**

Algoritmo de Interpretación Claude

Prompt especializado que procesará:

"Zona contactada: [DATOS_SENSOR]
Contexto temporal: [PATRON_USO]
Signos vitales: [BIOMETRIA]
Historial emocional: [SESIONES_PREVIAS]

Proporciona interpretación biodecodificada considerando:
- Conflicto biológico subyacente
- Fase de resolución emocional
- Estrategias terapéuticas específicas

- Actividades de integración corporal"

Prototipo de Validación

Fase Alpha (Actual)

- Componentes básicos en protoboard
- Comunicación Bluetooth funcional
- App básica de recepción de datos

Fase Beta (Próxima)

- Anillo impreso 3D con sensores integrados
- App con interpretaciones Claude básicas
- Pruebas con 10 usuarios voluntarios

Fase Gamma (Objetivo)

- Anillo miniaturizado profesional
- IA de biodecodificación completamente entrenada
- Validación clínica con terapeutas especializados

Aplicación de Biodecodificación Emocional
Artefacto interactivo

```
import React, { useState, useEffect } from 'react';
import { Heart, Brain, Zap, Activity, AlertCircle, CheckCircle, Users, BookOpen } from
'lucide-react';

const BiodecodificacionApp = () => {
  const [connectionStatus, setConnectionStatus] = useState('disconnected');
  const [currentReading, setCurrentReading] = useState(null);
  const [emotionalAnalysis, setEmotionalAnalysis] = useState(null);
  const [historyData, setHistoryData] = useState([]);
  const [activeTab, setActiveTab] = useState('monitor');
```

```
// Simulación de datos del anillo
const bodyParts = {
  head: { name: 'Cabeza', emotion: 'Estrés Mental', color: 'bg-red-500' },
  throat: { name: 'Garganta', emotion: 'Expresión Bloqueada', color: 'bg-blue-500' },
  heart: { name: 'Corazón', emotion: 'Conflictos Afectivos', color: 'bg-pink-500' },
  stomach: { name: 'Estómago', emotion: 'Miedos/Ansiedad', color: 'bg-yellow-500' },
  liver: { name: 'Hígado', emotion: 'Ira Contenida', color: 'bg-green-500' },
  kidneys: { name: 'Riñones', emotion: 'Miedo Profundo', color: 'bg-purple-500' }
};

// Simulación de conexión con el anillo
useEffect(() => {
  const connectToRing = () => {
    setTimeout(() => {
      setConnectionStatus('connected');
    }, 2000);
  };

  connectToRing();

  // Simulación de lecturas periódicas
  const interval = setInterval(() => {
    if (connectionStatus === 'connected') {
      const parts = Object.keys(bodyParts);
      const randomPart = parts[Math.floor(Math.random() * parts.length)];
      const intensity = Math.floor(Math.random() * 100) + 1;

      const reading = {
        id: Date.now(),
        bodyPart: randomPart,
        intensity,
        timestamp: new Date().toLocaleTimeString(),
        emotion: bodyParts[randomPart].emotion
      };

      setCurrentReading(reading);
      analyzeEmotionalPattern(reading);
    }
  }, 5000);

  return () => clearInterval(interval);
}, [connectionStatus]);

const analyzeEmotionalPattern = (reading) => {
```

```
  const analyses = {
    head: {
      pattern: "Sobrecarga Mental",
      message: "El cerebro está procesando demasiada información. Necesitas pausas conscientes.",
      recommendation: "Practica meditación de 10 minutos. Reduce estímulos externos.",
      chakra: "Corona - Séptimo Chakra"
    },
    throat: {
      pattern: "Comunicación Bloqueada",
      message: "Hay palabras no dichas que necesitan expresarse. El miedo al juicio bloquea tu voz.",
      recommendation: "Ejercita la expresión creativa. Canta, escribe o habla contigo mismo.",
      chakra: "Garganta - Quinto Chakra"
    },
    heart: {
      pattern: "Heridas Emocionales",
      message: "El corazón guarda emociones no procesadas. Dolor o amor no correspondido.",
      recommendation: "Practica autocompasión. Permite que las emociones fluyan sin juzgarlas.",
      chakra: "Corazón - Cuarto Chakra"
    },
    stomach: {
      pattern: "Ansiedad Digestiva",
      message: "El estómago refleja miedos no resueltos y situaciones que 'no puedes digerir'.",
      recommendation: "Respira profundamente antes de comer. Identifica qué situaciones te generan ansiedad.",
      chakra: "Plexo Solar - Tercer Chakra"
    },
    liver: {
      pattern: "Ira Acumulada",
      message: "El hígado procesa la ira y el resentimiento. Hay emociones tóxicas acumuladas.",
      recommendation: "Expresa la ira de forma segura. Ejercicio físico intenso o golpear un cojín.",
      chakra: "Plexo Solar - Tercer Chakra"
    },
    kidneys: {
      pattern: "Miedo Ancestral",
      message: "Los riñones guardan miedos profundos, a veces heredados. Inseguridad existencial.",
      recommendation: "Trabaja la confianza básica. Conecta con la naturaleza y tu fuerza interior.",
      chakra: "Raíz - Primer Chakra"
```

```
    }
  };

  const analysis = analyses[reading.bodyPart];
  setEmotionalAnalysis({
    ...analysis,
    intensity: reading.intensity,
    bodyPart: bodyParts[reading.bodyPart].name
  });

  // Agregar al historial
  setHistoryData(prev => [reading, ...prev].slice(0, 10));
};

const ConnectionStatus = () => (
  <div className={`flex items-center gap-2 p-3 rounded-lg ${
    connectionStatus === 'connected' ? 'bg-green-100 text-green-800' : 'bg-gray-100
text-gray-600'
  }`}>
    <div className={`w-3 h-3 rounded-full ${
      connectionStatus === 'connected' ? 'bg-green-500 animate-pulse' : 'bg-gray-400'
    }`}></div>
    <span className="font-medium">
      {connectionStatus === 'connected' ? 'Anillo Conectado' : 'Conectando al Anillo...'}
    </span>
  </div>
);

const CurrentReading = () => {
  if (!currentReading) return null;

  return (
    <div className="bg-white rounded-xl shadow-lg p-6 border-l-4 border-blue-500">
      <div className="flex items-center justify-between mb-4">
        <h3 className="text-lg font-semibold text-gray-800">Lectura Actual</h3>
        <span className="text-sm text-gray-500">{currentReading.timestamp}</span>
      </div>

      <div className="grid grid-cols-2 gap-4">
        <div>
          <p className="text-sm text-gray-600">Zona Corporal</p>
          <p className="font-semibold
text-gray-800">{bodyParts[currentReading.bodyPart].name}</p>
        </div>
```

```
        <div>
          <p className="text-sm text-gray-600">Intensidad</p>
          <div className="flex items-center gap-2">
            <div className="w-20 h-2 bg-gray-200 rounded-full">
              <div
                className={`h-full rounded-full ${bodyParts[currentReading.bodyPart].color}`}
                style={{ width: `${currentReading.intensity}%` }}
              ></div>
            </div>
            <span className="text-sm font-medium">{currentReading.intensity}%</span>
          </div>
        </div>
      </div>
    </div>
  );
};

const EmotionalAnalysis = () => {
  if (!emotionalAnalysis) return null;

  return (
    <div className="bg-gradient-to-br from-purple-50 to-blue-50 rounded-xl p-6 shadow-lg">
      <div className="flex items-center gap-3 mb-4">
        <Brain className="w-6 h-6 text-purple-600" />
        <h3 className="text-lg font-semibold text-gray-800">Análisis Biodecodificación</h3>
      </div>

      <div className="space-y-4">
        <div className="bg-white rounded-lg p-4">
          <h4 className="font-medium text-purple-800 mb-2">Patrón Emocional Detectado</h4>
          <p className="text-gray-700">{emotionalAnalysis.pattern}</p>
        </div>

        <div className="bg-white rounded-lg p-4">
          <h4 className="font-medium text-blue-800 mb-2">Interpretación Claude</h4>
          <p className="text-gray-700">{emotionalAnalysis.message}</p>
        </div>

        <div className="bg-white rounded-lg p-4">
          <h4 className="font-medium text-green-800 mb-2">Recomendación Terapéutica</h4>
          <p className="text-gray-700">{emotionalAnalysis.recommendation}</p>
        </div>

        <div className="bg-white rounded-lg p-4">
```

```
          <h4 className="font-medium text-orange-800 mb-2">Centro Energético</h4>
          <p className="text-gray-700">{emotionalAnalysis.chakra}</p>
        </div>
      </div>
    </div>
  );
};

const HistoryView = () => (
  <div className="space-y-3">
    <h3 className="text-lg font-semibold text-gray-800 flex items-center gap-2">
      <Activity className="w-5 h-5" />
      Historial de Lecturas
    </h3>
    {historyData.length === 0 ? (
      <p className="text-gray-500 text-center py-8">No hay lecturas registradas aún</p>
    ) : (
      historyData.map((reading) => (
        <div key={reading.id} className="bg-white rounded-lg p-4 shadow-sm border">
          <div className="flex justify-between items-start">
            <div>
              <p className="font-medium text-gray-800">{bodyParts[reading.bodyPart].name}</p>
              <p className="text-sm text-gray-600">{reading.emotion}</p>
            </div>
            <div className="text-right">
              <p className="text-sm text-gray-500">{reading.timestamp}</p>
              <p className="text-sm font-medium">{reading.intensity}%</p>
            </div>
          </div>
        </div>
      ))
    )}
  </div>
);

const ResourcesView = () => (
  <div className="space-y-4">
    <h3 className="text-lg font-semibold text-gray-800 flex items-center gap-2">
      <BookOpen className="w-5 h-5" />
      Recursos de Biodecodificación
    </h3>

    <div className="grid gap-4">
      <div className="bg-white rounded-lg p-4 shadow-sm">
```

```
        <h4 className="font-medium text-blue-800 mb-2">Técnicas de Liberación
Emocional</h4>
        <ul className="text-sm text-gray-700 space-y-1">
          <li>• Respiración consciente para calmar el sistema nervioso</li>
          <li>• EFT (Tapping) para liberar bloqueos energéticos</li>
          <li>• Movimiento corporal para integrar emociones</li>
        </ul>
      </div>

      <div className="bg-white rounded-lg p-4 shadow-sm">
        <h4 className="font-medium text-green-800 mb-2">Correspondencias
Emocionales</h4>
        <ul className="text-sm text-gray-700 space-y-1">
          <li>• <strong>Cabeza:</strong> Pensamientos excesivos, preocupaciones</li>
          <li>• <strong>Garganta:</strong> Expresión bloqueada, secretos</li>
          <li>• <strong>Corazón:</strong> Amor, pérdidas, relaciones</li>
          <li>• <strong>Estómago:</strong> Miedos, situaciones indigestas</li>
        </ul>
      </div>

      <div className="bg-white rounded-lg p-4 shadow-sm">
        <h4 className="font-medium text-purple-800 mb-2">Ejercicios de Autoayuda</h4>
        <ul className="text-sm text-gray-700 space-y-1">
          <li>• Journaling emocional para identificar patrones</li>
          <li>• Meditación guiada específica por zona corporal</li>
          <li>• Visualización de sanación energética</li>
        </ul>
      </div>
    </div>
  </div>
);

 return (
  <div className="max-w-md mx-auto bg-gray-50 min-h-screen">
    {/* Header */}
    <div className="bg-gradient-to-r from-purple-600 to-blue-600 text-white p-6">
      <div className="flex items-center gap-3 mb-4">
        <Heart className="w-8 h-8" />
        <h1 className="text-xl font-bold">BioCodificación</h1>
      </div>
      <ConnectionStatus />
    </div>

    {/* Navigation */}
```

```
      <div className="flex bg-white shadow-sm">
        {[
          { id: 'monitor', label: 'Monitor', icon: Activity },
          { id: 'history', label: 'Historial', icon: Users },
          { id: 'resources', label: 'Recursos', icon: BookOpen }
        ].map(({ id, label, icon: Icon }) => (
          <button
            key={id}
            onClick={() => setActiveTab(id)}
            className={`flex-1 py-3 px-4 flex items-center justify-center gap-2 text-sm font-medium
border-b-2 transition-colors ${
              activeTab === id
                ? 'border-purple-500 text-purple-600'
                : 'border-transparent text-gray-500 hover:text-gray-700'
            }`}
          >
            <Icon className="w-4 h-4" />
            {label}
          </button>
        ))}
      </div>

      {/* Content */}
      <div className="p-4 space-y-6">
        {activeTab === 'monitor' && (
          <>
            <CurrentReading />
            <EmotionalAnalysis />
          </>
        )}

        {activeTab === 'history' && <HistoryView />}

        {activeTab === 'resources' && <ResourcesView />}
      </div>

      {/* Footer */}
      <div className="p-4 text-center text-xs text-gray-500">
        <p>Powered by Claude AI • Prototipo de Investigación</p>
        <p className="mt-1">La biodecodificación es complementaria, no reemplaza atención
médica</p>
      </div>
    </div>
  );
```

```
};

export default BiodecodificacionApp;
```

```
name: siento_y_comprendo
description: App para decodificación emocional con anillo sensor y Claude AI

publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: '>=2.17.0 <4.0.0'

dependencies:
  flutter:
    sdk: flutter

  # Comunicación Bluetooth
  flutter_bluetooth_serial: ^0.4.0

  # HTTP requests para Claude API
  http: ^1.1.0

  # Manejo de JSON
  json_annotation: ^4.8.1

  # Almacenamiento local
  shared_preferences: ^2.2.2

  # Base de datos local
  sqflite: ^2.3.0

  # Manejo de fechas
  intl: ^0.18.1

  # Iconos adicionales
  cupertino_icons: ^1.0.2

  # Permisos del sistema
  permission_handler: ^11.0.1
```

```
  # Notificaciones locales
  flutter_local_notifications: ^16.1.0

  # Gráficos y charts para el historial
  fl_chart: ^0.64.0

  # Animaciones
  lottie: ^2.7.0

dev_dependencies:
  flutter_test:
    sdk: flutter

  flutter_lints: ^3.0.0

  json_serializable: ^6.7.1
  build_runner: ^2.4.7

flutter:
  uses-material-design: true

  # Assets para la app
  assets:
    - assets/images/
    - assets/animations/
    - assets/sounds/

  # Fuentes personalizadas
  fonts:
    - family: Roboto
      fonts:
        - asset: fonts/Roboto-Regular.ttf
        - asset: fonts/Roboto-Bold.ttf
          weight: 700
```

```
import React, { useState, useEffect } from 'react';
import { Heart, Brain, User, Zap, Bluetooth, Play, Pause, BookOpen, Activity } from 'lucide-react';

const SientoYComprendo = () => {
  const [conectadoBluetooth, setConectadoBluetooth] = useState(false);
  const [zonaDetectada, setZonaDetectada] = useState(null);
```

```
const [interpretacion, setInterpretacion] = useState(null);
const [actividades, setActividades] = useState([]);
const [modo, setModo] = useState('niño');
const [sonidoActivo, setSonidoActivo] = useState(false);

// Simulación de conexión Bluetooth con anillo
const conectarAnillo = () => {
  setConectadoBluetooth(true);
  // Simulación de datos del anillo cada 3 segundos
  const intervalo = setInterval(() => {
    const zonas = ['cabeza', 'garganta', 'corazón', 'estómago', 'bajo_vientre', 'piernas'];
    const zonaAleatoria = zonas[Math.floor(Math.random() * zonas.length)];
    recibirSeñalAnillo(zonaAleatoria);
  }, 5000);

  return intervalo;
};

// Base de conocimiento biodecodificativa
const biodecodificacion = {
  cabeza: {
    conflicto: "Exceso de control mental, pensamientos repetitivos, desvalorización intelectual",
    emocion: "Preocupación, autocrítica, presión mental",
    organo: "Cerebro, sistema nervioso",
    actividades: ["Visualización creativa", "Pintura libre", "Respiración consciente"]
  },
  garganta: {
    conflicto: "Imposibilidad de expresar, tragar emociones, comunicación bloqueada",
    emocion: "Miedo a hablar, frustración no expresada",
    organo: "Tiroides, cuerdas vocales",
    actividades: ["Cantar libremente", "Gritar en cojín", "Escribir pensamientos"]
  },
  corazón: {
    conflicto: "Territorio emocional invadido, pérdida afectiva, desamor",
    emocion: "Tristeza profunda, abandono, soledad",
    organo: "Corazón, sistema circulatorio",
    actividades: ["Abrazo consciente", "Carta de amor propio", "Música sanadora"]
  },
  estómago: {
    conflicto: "No poder digerir una situación, contrariedad indigesta",
    emocion: "Ira contenida, impotencia, injusticia",
    organo: "Estómago, sistema digestivo",
    actividades: ["Masaje abdominal", "Expresión corporal", "Sonidos graves"]
  },
```

```
    bajo_vientre: {
      conflicto: "Miedo arcaico, supervivencia amenazada, pérdida de territorio",
      emocion: "Miedo profundo, inseguridad existencial",
      organo: "Intestinos, órganos reproductivos",
      actividades: ["Enraizamiento", "Movimiento pélvico", "Visualización seguridad"]
    },
    piernas: {
      conflicto: "Imposibilidad de avanzar, miedo al futuro, desvalorización motriz",
      emocion: "Estancamiento, inseguridad para actuar",
      organo: "Músculos, articulaciones, circulación",
      actividades: ["Caminar consciente", "Danza libre", "Ejercicios de equilibrio"]
    }
  };

  const recibirSeñalAnillo = (zona) => {
    setZonaDetectada(zona);
    const datos = biodecodificacion[zona];
    if (datos) {
      // Simulación de respuesta de Claude AI
      const interpretacionCompleta = {
        zona: zona.charAt(0).toUpperCase() + zona.slice(1).replace('_', ' '),
        conflicto: datos.conflicto,
        emocion: datos.emocion,
        organo: datos.organo,
        mensaje: generarMensajePersonalizado(zona, modo),
        actividades: datos.actividades
      };
      setInterpretacion(interpretacionCompleta);
      setActividades(datos.actividades);
    }
  };

  const generarMensajePersonalizado = (zona, modo) => {
    const mensajes = {
      niño: {
        cabeza: "Tu cabecita está muy ocupada pensando. ¿Qué tal si dejamos descansar esos
pensamientos con un juego?",
        garganta: "Parece que tienes algo importante que decir pero no encuentras las palabras.
¡Vamos a ayudar a tu voz!",
        corazón: "Tu corazoncito necesita amor y cuidado. Recordemos todo lo que te hace feliz.",
        estómago: "Tu pancita está sintiendo algo difícil de digerir. Vamos a ayudarla con cariño.",
        bajo_vientre: "Tu cuerpito está pidiendo sentirse seguro y protegido. Hagamos ejercicios
que te tranquilicen.",
```

```
      piernas: "Tus piernitas quieren moverte hacia adelante. ¡Vamos a encontrar tu fuerza interior!"
    },
    adulto: {
      cabeza: "Detectamos una sobrecarga mental. Es momento de liberar la presión cognitiva y reconectar con la intuición.",
      garganta: "Hay una expresión bloqueada en tu comunicación. Tu verdad necesita ser expresada auténticamente.",
      corazón: "El territorio emocional requiere atención. Es tiempo de sanar las heridas afectivas y recuperar el amor propio.",
      estómago: "Existe una situación que no logras procesar emocionalmente. Tu sistema digestivo refleja esta dificultad.",
      bajo_vientre: "Se activa un miedo primordial relacionado con la supervivencia. Necesitas reconectar con tu seguridad básica.",
      piernas: "Hay una resistencia al movimiento vital. Tu capacidad de avanzar está siendo cuestionada internamente."
    }
  };
  return mensajes[modo][zona] || "Zona detectada requiere atención emocional.";
 };

 const ejecutarActividad = (actividad) => {
  setSonidoActivo(true);
  setTimeout(() => setSonidoActivo(false), 3000);
  alert(`Iniciando actividad: ${actividad}\n\nSigue las instrucciones que aparecerán en pantalla para completar esta práctica de sanación emocional.`);
 };

 return (
  <div className="min-h-screen bg-gradient-to-br from-purple-100 via-blue-50 to-indigo-100 p-4">
   {/* Header */}
   <div className="max-w-md mx-auto bg-white rounded-2xl shadow-lg overflow-hidden">
    <div className="bg-gradient-to-r from-purple-600 to-indigo-600 p-6 text-white">
     <div className="flex items-center justify-between">
      <div className="flex items-center space-x-3">
       <Brain className="w-8 h-8" />
       <div>
        <h1 className="text-xl font-bold">Siento y Comprendo</h1>
        <p className="text-purple-200 text-sm">Biodecodificación Emocional</p>
       </div>
      </div>
      <div className="flex items-center space-x-2">
```

```
            <Bluetooth className={`w-6 h-6 ${conectadoBluetooth ? 'text-green-300' :
'text-gray-400'}`} />
            {sonidoActivo && <Activity className="w-6 h-6 text-yellow-300 animate-pulse" />}
          </div>
        </div>
      </div>

      {/* Configuración de Modo */}
      <div className="p-4 bg-gray-50 border-b">
        <div className="flex space-x-2">
          <button
            onClick={() => setModo('niño')}
            className={`flex-1 py-2 px-4 rounded-lg text-sm font-medium ${
              modo === 'niño' ? 'bg-pink-500 text-white' : 'bg-white text-gray-600'
            }`}
          >
            🧸 Modo Niño
          </button>
          <button
            onClick={() => setModo('adulto')}
            className={`flex-1 py-2 px-4 rounded-lg text-sm font-medium ${
              modo === 'adulto' ? 'bg-blue-500 text-white' : 'bg-white text-gray-600'
            }`}
          >
            👤 Modo Adulto
          </button>
        </div>
      </div>

      {/* Conexión con Anillo */}
      <div className="p-6">
        {!conectadoBluetooth ? (
          <div className="text-center">
            <div className="w-20 h-20 mx-auto mb-4 bg-gradient-to-r from-purple-400
to-pink-400 rounded-full flex items-center justify-center">
              <Zap className="w-10 h-10 text-white" />
            </div>
            <h3 className="text-lg font-semibold text-gray-800 mb-2">Conectar Anillo
Detector</h3>
            <p className="text-gray-600 text-sm mb-4">
              Activa el anillo biodecodificador para comenzar la exploración emocional
            </p>
            <button
              onClick={conectarAnillo}
```

```
            className="w-full bg-gradient-to-r from-purple-500 to-indigo-500 text-white py-3 px-6
rounded-xl font-medium hover:shadow-lg transition-all duration-300"
          >
            <Bluetooth className="w-5 h-5 inline mr-2" />
            Conectar Dispositivo
          </button>
        </div>
      ) : (
        <div className="space-y-6">
          {/* Estado de Conexión */}
          <div className="bg-green-50 border border-green-200 rounded-lg p-4">
            <div className="flex items-center">
              <div className="w-3 h-3 bg-green-500 rounded-full animate-pulse mr-3"></div>
              <span className="text-green-800 font-medium">Anillo conectado - Listo para
detectar</span>
            </div>
          </div>

          {/* Zona Detectada e Interpretación */}
          {zonaDetectada && interpretacion && (
            <div className="space-y-4">
              <div className="bg-gradient-to-r from-orange-400 to-red-400 text-white p-4
rounded-xl">
                <div className="flex items-center mb-2">
                  <Heart className="w-6 h-6 mr-2" />
                  <h3 className="text-lg font-bold">Zona: {interpretacion.zona}</h3>
                </div>
                <p className="text-orange-100 text-sm">
                  Órgano relacionado: {interpretacion.organo}
                </p>
              </div>

              {/* Interpretación Biodecodificativa */}
              <div className="bg-white border-2 border-purple-200 rounded-xl p-4">
                <h4 className="font-semibold text-purple-800 mb-2 flex items-center">
                  <BookOpen className="w-5 h-5 mr-2" />
                  Interpretación Emocional
                </h4>
                <div className="space-y-3 text-sm">
                  <div>
                    <span className="font-medium text-gray-700">Conflicto detectado:</span>
                    <p className="text-gray-600 mt-1">{interpretacion.conflicto}</p>
                  </div>
                  <div>
```

```
                  <span className="font-medium text-gray-700">Emoción asociada:</span>
                  <p className="text-gray-600 mt-1">{interpretacion.emocion}</p>
                </div>
                <div className="bg-indigo-50 p-3 rounded-lg border-l-4 border-indigo-400">
                  <p className="text-indigo-800 font-medium">{interpretacion.mensaje}</p>
                </div>
              </div>
            </div>

            {/* Actividades Sugeridas */}
            <div className="bg-gradient-to-r from-green-50 to-teal-50 p-4 rounded-xl border
border-green-200">
              <h4 className="font-semibold text-green-800 mb-3 flex items-center">
                <Play className="w-5 h-5 mr-2" />
                Actividades de Sanación
              </h4>
              <div className="space-y-2">
                {actividades.map((actividad, index) => (
                  <button
                    key={index}
                    onClick={() => ejecutarActividad(actividad)}
                    className="w-full text-left bg-white p-3 rounded-lg border border-green-200
hover:bg-green-50 transition-colors duration-200"
                  >
                    <span className="text-green-700 font-medium">🎯 {actividad}</span>
                  </button>
                ))}
              </div>
            </div>
          </div>
        )}

        {/* Instrucciones */}
        {!zonaDetectada && (
          <div className="text-center py-8">
            <div className="w-16 h-16 mx-auto mb-4 bg-blue-100 rounded-full flex
items-center justify-center animate-pulse">
              <User className="w-8 h-8 text-blue-500" />
            </div>
            <h3 className="text-lg font-medium text-gray-800 mb-2">
              Toca con el anillo la zona de tu cuerpo
            </h3>
            <p className="text-gray-600 text-sm">
              El anillo detectará automáticamente la región corporal y proporcionará
```

```
                una interpretación biodecodificativa personalizada
              </p>
            </div>
          )}
        </div>
      )}
    </div>

    {/* Footer */}
    <div className="bg-gray-50 p-4 text-center">
      <p className="text-xs text-gray-500">
        Desarrollado con Claude AI • Biodecodificación Emocional
      </p>
    </div>
  </div>
</div>
);
};

export default SientoYComprendo;
```

Arquitectura de la Aplicación con Claude AI

Basándome en el documento, implementaremos una aplicación que integre Claude AI en lugar de GPT para la interpretación biodecodificativa. Aquí está el prototipo funcional:

Siento y Comprendo - Prototipo App Biodecodificación
Artefacto interactivo

Especificaciones Técnicas del Prototipo

1. Arquitectura del Anillo Detector

Componentes del Anillo:

- **Microcontrolador:** ESP32-C3 (ultra compacto, Bluetooth integrado)
- **Sensor:** Acelerómetro/giroscopio MPU6050 para detectar orientación corporal
- **Batería:** LiPo 3.7V, 150mAh (autonomía 8-12 horas)

- **Carcasa:** Impresión 3D en TPU flexible, resistente al agua

2. Protocolo de Comunicación Claude AI

javascript

```javascript
// Estructura de datos enviados a Claude AI
const solicitudBiodecodificacion = {
  zona_detectada: "corazón",
  timestamp: "2025-06-03T10:30:00",
  modo_usuario: "adulto",
  contexto_adicional: "sesión matutina",
  historial_previo: ["estómago", "cabeza"]
}

// Respuesta estructurada de Claude
const respuestaClaudeAI = {
  interpretacion_biologica: "Conflicto de territorio emocional",
  emocion_asociada: "Pérdida afectiva, desamor",
  mensaje_personalizado: "Tu corazón refleja una necesidad de sanación...",
  actividades_sugeridas: ["Respiración cardíaca", "Visualización amorosa"],
  frecuencia_recomendada: "528 Hz - Frecuencia del amor"
}
```

3. Integración con Biodecodificación

La aplicación utiliza una base de datos biodecodificativa que mapea:

- **Zonas corporales → Conflictos emocionales específicos**
- **Síntomas físicos → Emociones bloqueadas**
- **Terapias sugeridas → Actividades de liberación emocional**

4. Ventajas del Prototipo con Claude AI

1. **Interpretación Contextual**: Claude AI comprende matices emocionales complejos
2. **Personalización Adaptativa**: Ajusta respuestas según edad y contexto
3. **Integración Multimodal**: Combina texto, audio y actividades interactivas
4. **Privacidad**: Procesamiento local de datos sensibles

5. Próximos Pasos de Implementación

1. **Prototipo Hardware**: Construcción del anillo con ESP32
2. **Calibración Sensorial**: Mapeo de zonas corporales por posición
3. **Integración Claude API**: Conexión con el modelo de IA
4. **Testing Pedagógico**: Pruebas con usuarios reales (niños/adultos)

5. **Refinamiento Biodecodificativo**: Ajuste de interpretaciones según feedback

---

Arquitectura del Hardware del Anillo Detector

Componentes Principales del Sistema

1. Unidad de Procesamiento Central (MCU)

- Microcontrolador ESP32-S3 con conectividad Wi-Fi y Bluetooth
- Factor de forma compacto (5mm x 5mm)
- Procesamiento de señales en tiempo real
- Bajo consumo energético para uso prolongado

2. Matriz de Sensores Biométricos

Sensores Primarios:

- **Sensor de conductancia galvánica de la piel (GSR)**: Detecta cambios en la resistencia eléctrica cutánea asociados con respuestas emocionales
- **Sensor de pulso óptico (PPG)**: Monitorea variabilidad del ritmo cardíaco, indicador clave del estado emocional
- **Sensor de temperatura corporal**: Termistor de precisión para detectar variaciones térmicas asociadas con estados emocionales
- **Acelerómetro/giroscopio de 6 ejes**: Detecta micro-movimientos y tensiones musculares involuntarias

Sensores Secundarios:

- **Sensor de presión capacitivo**: En el botón táctil para detección de tensión aplicada
- **Sensor de proximidad infrarrojo**: Confirma contacto con la piel

3. Sistema de Comunicación

- Módulo Bluetooth 5.0 BLE para conexión con smartphone
- Antena integrada en el diseño del anillo
- Protocolo de comunicación seguro con encriptación AES-256

4. Alimentación

- Batería de ión-litio flexible (25mAh)
- Circuito de carga inalámbrica por inducción

- Gestión inteligente de energía con modos de suspensión

Diseño Físico del Anillo

Estructura Principal:

- Carcasa de titanio biocompatible grado médico
- Diámetro interno ajustable: 16-22mm
- Grosor: 4mm máximo
- Peso: <8 gramos

Distribución de Componentes:

- Zona superior: MCU y batería
- Zona de contacto: Sensores GSR y temperatura
- Zona lateral: Botón táctil con sensor de presión
- Zona inferior: Sensores ópticos y de proximidad

Implementación de la Detección Bioemocional

Algoritmo de Procesamiento:

1. **Captura Multiparamétrica Continua**
    - Muestreo a 100Hz para GSR y temperatura
    - Muestreo a 25Hz para PPG
    - Muestreo a 50Hz para acelerómetro
2. **Procesamiento de Señales**
    - Filtrado digital de ruido
    - Normalización de datos según baseline individual
    - Análisis de patrones temporales
3. **Detección de Estados Emocionales**
    - Correlación de variabilidad del ritmo cardíaco con estados de estrés/relajación
    - Análisis de conductancia para detectar activación del sistema nervioso simpático
    - Identificación de patrones de micro-movimientos asociados con tensión emocional

Protocolo de Comunicación con la APP

Estructura de Datos Transmitidos:

```
{
  "timestamp": timestamp,
  "gsr_value": float,
  "heart_rate": int,
  "hrv_score": float,
```

```
  "temperature": float,
  "movement_pattern": array,
  "pressure_intensity": float,
  "emotional_state_raw": int,
  "confidence_level": float
}
```

Eventos de Activación:

- Detección automática de cambios significativos en biomarcadores
- Activación manual mediante botón táctil
- Modo de monitoreo continuo programable

Consideraciones de Biodecodificación

Mapeo Emocional-Corporal:

- Integración de tablas de correspondencia entre zonas corporales tocadas y posibles conflictos emocionales
- Algoritmos de aprendizaje para personalizar respuestas según el usuario
- Base de datos de patrones psicosomáticos según la Nueva Medicina Germánica

Calibración Individual:

- Período de adaptación de 72 horas para establecer baseline personal
- Ajuste de sensibilidad según características fisiológicas del usuario
- Aprendizaje de patrones emocionales específicos

Especificaciones Técnicas Finales

Dimensiones: 22mm × 4mm (diámetro × grosor) **Autonomía:** 5-7 días uso normal **Resistencia:** IP67 (resistente al agua y polvo) **Conectividad:** Bluetooth 5.0 BLE, alcance 10m **Precisión:** ±2% en mediciones biométricas **Tiempo de respuesta:** <200ms para detección de eventos

---

```
/*
 * ANILLO INTELIGENTE "SIENTO Y COMPRENDO"
 * Prototipo v1.0 - Detección corporal + Bluetooth
 * Hardware: ESP32-C3 + MPU6050 + TTP223
 */
```

```
#include <Wire.h>
#include <MPU6050.h>
#include <BluetoothSerial.h>
#include <ArduinoJson.h>

// Configuración Bluetooth
BluetoothSerial SerialBT;

// Sensor MPU6050
MPU6050 mpu;

// Pines
const int TOUCH_PIN = 2;     // Sensor táctil TTP223
const int LED_PIN = 8;       // LED de estado
const int BATTERY_PIN = A0;  // Monitoreo batería

// Variables de estado
bool isConnected = false;
bool touchActive = false;
unsigned long lastTouchTime = 0;
unsigned long touchDuration = 0;

// Datos de sensores
struct SensorData {
  float accelX, accelY, accelZ;
  float gyroX, gyroY, gyroZ;
  bool touchDetected;
  int batteryLevel;
  String zonaDetectada;
  unsigned long timestamp;
};

void setup() {
  Serial.begin(115200);

  // Configurar pines
  pinMode(TOUCH_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(BATTERY_PIN, INPUT);

  // Inicializar MPU6050
  Wire.begin();
  mpu.initialize();
```

```
  if (mpu.testConnection()) {
    Serial.println("MPU6050 conectado correctamente");
    digitalWrite(LED_PIN, HIGH);
    delay(500);
    digitalWrite(LED_PIN, LOW);
  } else {
    Serial.println("Error: MPU6050 no detectado");
    // Parpadeo de error
    for(int i = 0; i < 5; i++) {
      digitalWrite(LED_PIN, HIGH);
      delay(200);
      digitalWrite(LED_PIN, LOW);
      delay(200);
    }
  }

  // Inicializar Bluetooth
  SerialBT.begin("Anillo_SientoComprendo");
  Serial.println("Anillo listo para conexión Bluetooth");
  Serial.println("Nombre: Anillo_SientoComprendo");
}

void loop() {
  // Verificar conexión Bluetooth
  checkBluetoothConnection();

  // Leer sensores
  SensorData data = readSensors();

  // Detectar zona corporal
  data.zonaDetectada = detectZonaCorporal(data);

  // Procesar toque
  processTouchEvent(data);

  // Enviar datos si hay cambios significativos
  if (shouldSendData(data)) {
    sendDataToBluetooth(data);
  }

  // Gestión de batería
  checkBatteryLevel();

  delay(100); // Frecuencia de muestreo 10Hz
```

```
}

SensorData readSensors() {
  SensorData data;

  // Leer acelerómetro y giroscopio
  int16_t ax, ay, az, gx, gy, gz;
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  // Convertir a unidades g y grados/s
  data.accelX = ax / 16384.0;
  data.accelY = ay / 16384.0;
  data.accelZ = az / 16384.0;
  data.gyroX = gx / 131.0;
  data.gyroY = gy / 131.0;
  data.gyroZ = gz / 131.0;

  // Leer sensor táctil
  data.touchDetected = digitalRead(TOUCH_PIN);

  // Leer nivel de batería
  data.batteryLevel = map(analogRead(BATTERY_PIN), 0, 4095, 0, 100);

  data.timestamp = millis();

  return data;
}

String detectZonaCorporal(SensorData data) {
  String zona = "desconocida";

  if (!data.touchDetected) {
    return zona;
  }

  // Algoritmo de detección basado en orientación
  float accelMagnitude = sqrt(data.accelX*data.accelX +
                              data.accelY*data.accelY +
                              data.accelZ*data.accelZ);

  // Normalizar orientación
  float normY = data.accelY / accelMagnitude;
  float normX = data.accelX / accelMagnitude;
  float normZ = data.accelZ / accelMagnitude;
```

```
  // Detección por posición de la mano
  if (normY > 0.7) {
    zona = "cabeza";
  } else if (normY > 0.3 && normY <= 0.7) {
    if (abs(normX) < 0.3) {
      zona = "garganta";
    } else {
      zona = "cuello";
    }
  } else if (normY > -0.2 && normY <= 0.3) {
    if (normZ > 0.5) {
      zona = "corazon";
    } else {
      zona = "pecho";
    }
  } else if (normY > -0.6 && normY <= -0.2) {
    zona = "estomago";
  } else if (normY <= -0.6) {
    zona = "vientre";
  }

  return zona;
}

void processTouchEvent(SensorData data) {
  static bool lastTouchState = false;

  if (data.touchDetected && !lastTouchState) {
    // Inicio de toque
    touchActive = true;
    lastTouchTime = millis();
    digitalWrite(LED_PIN, HIGH);

    Serial.println("Toque iniciado en: " + data.zonaDetectada);

  } else if (!data.touchDetected && lastTouchState) {
    // Fin de toque
    touchActive = false;
    touchDuration = millis() - lastTouchTime;
    digitalWrite(LED_PIN, LOW);

    Serial.println("Toque finalizado. Duración: " + String(touchDuration) + "ms");
```

```
    // Enviar evento de toque completo
    sendTouchEvent(data.zonaDetectada, touchDuration);
  }

  lastTouchState = data.touchDetected;
}

bool shouldSendData(SensorData data) {
  static unsigned long lastSendTime = 0;
  static String lastZona = "";

  // Enviar cada 500ms o cuando cambie la zona
  if (millis() - lastSendTime > 500 || data.zonaDetectada != lastZona) {
    lastSendTime = millis();
    lastZona = data.zonaDetectada;
    return true;
  }

  return false;
}

void sendDataToBluetooth(SensorData data) {
  if (!isConnected) return;

  // Crear JSON con datos del sensor
  DynamicJsonDocument doc(1024);

  doc["timestamp"] = data.timestamp;
  doc["zona"] = data.zonaDetectada;
  doc["touch"] = data.touchDetected;
  doc["battery"] = data.batteryLevel;

  // Datos de orientación
  JsonObject orientation = doc.createNestedObject("orientation");
  orientation["x"] = data.accelX;
  orientation["y"] = data.accelY;
  orientation["z"] = data.accelZ;

  // Datos de movimiento
  JsonObject motion = doc.createNestedObject("motion");
  motion["gx"] = data.gyroX;
  motion["gy"] = data.gyroY;
  motion["gz"] = data.gyroZ;
```

```
  String jsonString;
  serializeJson(doc, jsonString);

  SerialBT.println(jsonString);
  Serial.println("Enviado: " + jsonString);
}

void sendTouchEvent(String zona, unsigned long duracion) {
  if (!isConnected) return;

  DynamicJsonDocument doc(512);
  doc["event"] = "touch_complete";
  doc["zona"] = zona;
  doc["duration"] = duracion;
  doc["timestamp"] = millis();

  String jsonString;
  serializeJson(doc, jsonString);

  SerialBT.println(jsonString);
  Serial.println("Evento enviado: " + jsonString);
}

void checkBluetoothConnection() {
  static bool lastConnectionState = false;

  isConnected = SerialBT.hasClient();

  if (isConnected != lastConnectionState) {
    if (isConnected) {
      Serial.println("Cliente Bluetooth conectado");
      // Parpadeo de confirmación
      for(int i = 0; i < 3; i++) {
        digitalWrite(LED_PIN, HIGH);
        delay(100);
        digitalWrite(LED_PIN, LOW);
        delay(100);
      }
    } else {
      Serial.println("Cliente Bluetooth desconectado");
    }
    lastConnectionState = isConnected;
  }
}
```

```
void checkBatteryLevel() {
  static unsigned long lastBatteryCheck = 0;

  if (millis() - lastBatteryCheck > 30000) { // Cada 30 segundos
    int batteryLevel = map(analogRead(BATTERY_PIN), 0, 4095, 0, 100);

    if (batteryLevel < 20) {
      Serial.println("Batería baja: " + String(batteryLevel) + "%");

      // Parpadeo de advertencia
      for(int i = 0; i < 10; i++) {
        digitalWrite(LED_PIN, HIGH);
        delay(50);
        digitalWrite(LED_PIN, LOW);
        delay(50);
      }
    }

    lastBatteryCheck = millis();
  }
}

// Función para calibración (llamar una vez al inicio)
void calibrateSensors() {
  Serial.println("Iniciando calibración...");
  Serial.println("Mantén el anillo quieto durante 5 segundos");

  delay(5000);

  // Aquí puedes agregar lógica de calibración específica
  // Por ejemplo, calcular offsets para el acelerómetro

  Serial.println("Calibración completada");
}
```

Consideraciones Críticas de Implementación

1. Desafíos de Miniaturización

La principal complejidad radica en integrar todos los componentes en un factor de forma tan reducido. El ESP32-S3 en su versión más compacta (QFN48) mide apenas 5×5mm, pero requiere componentes periféricos que ocupan espacio adicional. La solución propuesta utiliza un PCB flexible multicapa que se enrolla dentro del anillo, maximizando el aprovechamiento del espacio tridimensional.

2. Gestión Térmica Crítica

En un dispositivo tan compacto, la disipación de calor es fundamental. El diseño incorpora la carcasa de titanio como disipador pasivo, mientras que los algoritmos de gestión energética previenen el sobrecalentamiento mediante throttling dinámico del procesador.

3. Interferencias Electromagnéticas

La proximidad de múltiples sensores analógicos al transmisor Bluetooth requiere un diseño cuidadoso del plano de masa y filtrado activo. Cada sensor cuenta con su propio regulador de voltaje y filtros RC específicos.

4. Precisión en Biodecodificación

La correlación entre marcadores fisiológicos y estados emocionales debe calibrarse según los principios de la Nueva Medicina Germánica y la biodescodificación. El algoritmo integra:

- **Mapas de correspondencia órgano-emoción** según Hamer
- **Patrones de activación del sistema nervioso autónomo**
- **Correlaciones temporales** entre eventos emocionales y respuestas fisiológicas
- **Análisis de coherencia cardíaca** según HeartMath Institute

5. Validación Clínica

Para garantizar la efectividad del dispositivo en el contexto de biodecodificación, se requiere:

- Estudios piloto con terapeutas especializados
- Validación cruzada con técnicas tradicionales de diagnóstico emocional
- Correlación estadística entre lecturas del dispositivo y evaluaciones clínicas

---

Desarrollo Evolutivo del Prototipo: Anillo Biodecodificador Emocional

1. Arquitectura Ontológica de Datos Expandida

Partiendo de tu modelo ontológico inicial, propongo una estructura de datos jerárquica que capture la complejidad fenomenológica del proceso biodecodificador:

```
{
  "sesion_biodecodificacion": {
    "metadatos_temporales": {
      "timestamp": "ISO8601",
      "duracion_contacto": "segundos",
      "patron_ritmo_cardiaco": "array_bpm",
      "temperatura_cutanea": "celsius"
    },
    "localizacion_corporal": {
      "zona_anatomica": "string",
      "lateralidad": "derecha|izquierda|bilateral",
      "intensidad_presion": "0-100",
      "correlacion_meridiano": "medicina_tradicional_china"
    },
    "interpretacion_biodecodificada": {
      "conflicto_emocional_primario": "string",
      "fase_biologica": "simpaticotonia|vagotonia|crisis_epileptoide",
      "patron_generacional": "transgeneracional|proyecto_sentido",
      "simbolismo_organico": "string"
    }
  }
}
```

Diseño Técnico Detallado - Anillo Detector Bioemocional

1. DISEÑO DEL CIRCUITO IMPRESO (PCB)

Arquitectura del PCB Flexible

Configuración Multi-Capa:

- **Capa 1 (Superior)**: Componentes principales y MCU
- **Capa 2**: Plano de alimentación (+3.3V)
- **Capa 3**: Plano de masa (GND)
- **Capa 4 (Inferior)**: Sensores de contacto y antena

Especificaciones del Sustrato:

- Material: Poliimida flexible (Kapton) de 0.1mm
- Radio de curvatura mínimo: 3mm
- Temperatura operativa: -40°C a +85°C
- Conectores flexibles tipo ZIF para sensores modulares

Distribución Espacial Optimizada

Zona A (0°-90°): MCU ESP32-S3 + Cristal de 40MHz
Zona B (90°-180°): Batería Li-ion + Circuito de carga
Zona C (180°-270°): Sensores GSR + Temperatura
Zona D (270°-360°): PPG + Acelerómetro + Botón táctil

2. ALGORITMOS DE PROCESAMIENTO DE SEÑALES

Análisis de Conductancia Galvánica (GSR)

Procesamiento en Tiempo Real:

```
FUNCIÓN procesar_GSR(señal_cruda):
    // Filtro pasa-bajas Butterworth de 4º orden (fc = 5Hz)
    señal_filtrada = filtro_butterworth(señal_cruda, 5)

    // Separación de componentes tónica y fásica
    componente_tonica = filtro_mediana_movil(señal_filtrada, ventana=60s)
    componente_fasica = señal_filtrada - componente_tonica

    // Detección de picos de respuesta emocional
    picos = detectar_picos(componente_fasica, umbral=0.02μS)

    // Cálculo de métricas bioemocionales
    frecuencia_respuestas = contar_picos(picos, ventana=60s)
    amplitud_promedio = promedio(amplitudes_picos)
```

```
    RETORNAR {frecuencia_respuestas, amplitud_promedio, arousal_level}
FIN FUNCIÓN
```

Análisis de Variabilidad del Ritmo Cardíaco (HRV)

Métricas de Coherencia Emocional:

```
FUNCIÓN calcular_HRV(intervalos_RR):
    // Métricas temporales
    RMSSD = raiz_cuadrada(promedio(diferencias_consecutivas²))
    pNN50 = porcentaje_intervalos_>50ms

    // Análisis frecuencial (FFT)
    espectro = transformada_fourier(intervalos_RR)
    poder_LF = integrar_banda(espectro, 0.04-0.15Hz)  // Estrés
    poder_HF = integrar_banda(espectro, 0.15-0.4Hz)   // Relajación

    // Ratio de balance autonómico
    ratio_LF_HF = poder_LF / poder_HF

    // Índice de coherencia emocional (HeartMath Institute)
    coherencia = calcular_coherencia(intervalos_RR)

    RETORNAR {RMSSD, ratio_LF_HF, coherencia, estado_autonomico}
FIN FUNCIÓN
```

Fusión Sensorial Multimodal

Algoritmo de Integración Bioemocional:

```
FUNCIÓN detectar_estado_emocional(datos_sensores):
    // Normalización adaptativa
    gsr_norm = normalizar_adaptativo(datos_sensores.gsr, baseline_usuario)
    hrv_norm = normalizar_adaptativo(datos_sensores.hrv, baseline_usuario)
    temp_norm = normalizar_adaptativo(datos_sensores.temperatura, baseline_usuario)

    // Matriz de pesos adaptativos según biodecodificación
    pesos = {
        "activacion_simpatica": [0.6, 0.3, 0.1],  // GSR, HRV, Temp
        "coherencia_cardiaca": [0.2, 0.7, 0.1],
        "reactividad_termica": [0.1, 0.2, 0.7]
    }
```

```
    // Cálculo de vectores emocionales
    vector_arousal = suma_ponderada([gsr_norm, hrv_norm, temp_norm], pesos.activacion)
    vector_valencia = calcular_valencia(hrv_coherencia, micro_movimientos)

    // Mapeo a modelo circumplejo de emociones
    estado_emocional = mapear_circumplejo(vector_arousal, vector_valencia)

    RETORNAR {estado_emocional, confianza, marcadores_biodecodificacion}
FIN FUNCIÓN
```

3. SISTEMA DE ALIMENTACIÓN INTELIGENTE

Gestión Adaptativa de Energía

Estados de Operación:

Estado	Corriente	Sensores Activos	Duración
Activo	25mA	Todos	30s post-evento
Monitoreo	8mA	GSR + PPG	Continuo
Standby	2mA	Solo botón	>5min inactividad
Deep Sleep	50µA	Wake-up timer	>30min inactividad

Algoritmo de Predicción Energética:

```
FUNCIÓN gestionar_energia():
    patron_uso = analizar_historial_activacion(ultimas_24h)
    energia_disponible = leer_nivel_bateria()

    SI energia_disponible < 20%:
        reducir_frecuencia_muestreo(50%)
        desactivar_sensores_secundarios()

    SI patron_uso.periodo_inactivo_prolongado:
        programar_deep_sleep(duracion_estimada)

    optimizar_transmision_bluetooth(patron_uso.frecuencia_eventos)
FIN FUNCIÓN
```

4. PROTOCOLO DE COMUNICACIÓN AVANZADO

Estructura de Paquetes de Datos

Formato de Trama Bioemocional:

Header: [0xBE, 0xDC] (BioDecode)
Timestamp: 4 bytes (Unix timestamp)
Flags: 1 byte (estado sensores + calidad señal)
GSR_Raw: 2 bytes (0-4095, 12-bit ADC)
GSR_Processed: 2 bytes (conductancia en μS × 100)
HRV_Metrics: 8 bytes (RMSSD, LF/HF ratio, coherencia)
Temperature: 2 bytes (°C × 100)
Accel_Data: 6 bytes (X,Y,Z aceleración)
Pressure: 1 byte (intensidad botón 0-255)
Emotional_Vector: 4 bytes (arousal, valencia como float16)
Confidence: 1 byte (0-100%)
Checksum: 1 byte (XOR de todos los datos)

Sincronización y Buffering

Sistema de Cola Inteligente:

- Buffer circular de 512 muestras
- Compresión adaptativa según importancia del evento
- Retransmisión automática con backoff exponencial
- Priorización de eventos emocionales significativos

5. MAPEO BIODECODIFICACIÓN-HARDWARE

Correlaciones Psicosomáticas Detectables

Patrones Fisiológicos Específicos:

Conflicto Emocional	Marcador GSR	Patrón HRV	Respuesta Térmica
Miedo/Ansiedad	Picos >0.05μS	LF/HF >2.5	Descenso 0.2-0.5°C

Ira/Frustración	Meseta elevada	Baja coherencia	Aumento >0.3°C
Tristeza/Duelo	Variabilidad reducida	HF dominante	Estable/ligero desc.
Estrés Crónico	Nivel basal elevado	RMSSD <20ms	Fluctuaciones >±0.1°C

Algoritmo de Biodecodificación Integrada

```
FUNCIÓN interpretar_biodecodificacion(zona_tocada, datos_biometricos):
    // Mapeo según Nueva Medicina Germánica
    conflicto_potencial = tabla_biodecodificacion[zona_tocada]

    // Correlación con patrones fisiológicos
    patron_confirmatorio = buscar_patron(datos_biometricos, conflicto_potencial)

    // Cálculo de probabilidad
    probabilidad = calcular_correlacion(patron_confirmatorio)

    // Generación de insights
    insight = {
        "zona_afectada": zona_tocada,
        "conflicto_sugerido": conflicto_potencial,
        "marcadores_fisiologicos": patron_confirmatorio,
        "nivel_confianza": probabilidad,
        "recomendacion": generar_recomendacion(conflicto_potencial)
    }

    RETORNAR insight
FIN FUNCIÓN
```

6. CALIBRACIÓN Y PERSONALIZACIÓN

Proceso de Establecimiento de Baseline

Protocolo de 72 Horas:

1. **Horas 0-24**: Medición continua sin intervención
2. **Horas 24-48**: Calibración con eventos controlados
3. **Horas 48-72**: Refinamiento de algoritmos personalizados

Adaptación Continua del Sistema

```
FUNCIÓN adaptar_algoritmos_usuario():
    historial_respuestas = cargar_datos_ultimos_30_dias()

    // Análisis de patrones individuales
    baseline_personal = calcular_percentiles(historial_respuestas, [10, 50, 90])
    sensibilidad_usuario = analizar_variabilidad_respuestas()

    // Ajuste de umbrales de detección
    umbrales_personalizados = {
        "gsr_activacion": baseline_personal.gsr_p90 × 1.2,
        "hrv_estres": baseline_personal.hrv_p10 × 0.8,
        "temp_emocional": baseline_personal.temp_std × 2
    }

    // Actualización de modelo predictivo
    actualizar_red_neuronal(historial_respuestas, umbrales_personalizados)
FIN FUNCIÓN
```

---

3. Implementación Práctica del Prototipo Claude-Integrado

Procederé ahora a generar el código funcional que reemplaza la integración con ChatGPT por Claude, enfocándome específicamente en la biodecodificación emocional:

Código Arduino + Claude API - Anillo Biodecodificador
Código

```
// Anillo Biodecodificador Emocional - Integración con Claude API
// Desarrollado para ESP32-C3 con sensores integrados

#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Configuración de red y API
```

```
const char* ssid = "TU_WIFI_SSID";
const char* password = "TU_WIFI_PASSWORD";
const char* claude_api_key = "TU_CLAUDE_API_KEY";
const char* claude_endpoint = "https://api.anthropic.com/v1/messages";

// Pines del anillo
#define PRESSURE_SENSOR_PIN A0
#define TEMP_SENSOR_PIN 2
#define HEART_RATE_LED_PIN 3
#define HEART_RATE_SENSOR_PIN A1
#define STATUS_LED_PIN 4

// Configuración sensores
OneWire oneWire(TEMP_SENSOR_PIN);
DallasTemperature tempSensor(&oneWire);

// Variables globales para biodecodificación
struct BiodataReading {
  float pressure;
  float temperature;
  int heartRate;
  String contactZone;
  unsigned long contactDuration;
  String emotionalPattern;
};

BiodataReading currentReading;
bool isContactActive = false;
unsigned long contactStartTime = 0;

// Base de conocimiento biodecodificación simplificada
struct BiodecodingMap {
  String bodyZone;
  String primaryConflict;
  String biologicalPhase;
  String therapeuticAction;
};

BiodecodingMap biodecodingAtlas[] = {
  {"cabeza", "conflictos mentales, exceso de control", "simpaticotonia", "visualización_creativa"},
  {"cuello", "comunicación bloqueada, autoridad", "fase_reparacion", "expresión_vocal"},
  {"pecho", "conflictos afectivos, territorio", "crisis_epileptoide", "respiración_cardíaca"},
  {"abdomen", "conflictos digestivos, poder", "vagotonia", "masaje_abdominal"},
  {"espalda", "desvalorización, soporte", "simpaticotonia", "fortalecimiento_postural"},
```

```
  {"brazos", "capacidad de acción, abrazo", "fase_activa", "movimiento_expresivo"},
  {"piernas", "avance, dirección, libertad", "reparación", "caminar_consciente"}
};

void setup() {
  Serial.begin(115200);

  // Inicialización de pines
  pinMode(PRESSURE_SENSOR_PIN, INPUT);
  pinMode(HEART_RATE_SENSOR_PIN, INPUT);
  pinMode(STATUS_LED_PIN, OUTPUT);

  // Inicialización sensores
  tempSensor.begin();

  // Conexión WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi...");
    digitalWrite(STATUS_LED_PIN, !digitalRead(STATUS_LED_PIN));
  }

  Serial.println("Anillo Biodecodificador iniciado");
  digitalWrite(STATUS_LED_PIN, HIGH);
}

void loop() {
  // Lectura continua de sensores
  readBiosensors();

  // Detección de contacto corporal
  if (detectBodyContact()) {
    if (!isContactActive) {
      startContactSession();
    }
    updateContactReading();
  } else {
    if (isContactActive) {
      endContactSession();
    }
  }

  delay(100); // Frecuencia de muestreo 10Hz
```

```
}

void readBiosensors() {
  // Lectura sensor de presión
  int pressureRaw = analogRead(PRESSURE_SENSOR_PIN);
  currentReading.pressure = map(pressureRaw, 0, 4095, 0, 100);

  // Lectura temperatura corporal
  tempSensor.requestTemperatures();
  currentReading.temperature = tempSensor.getTempCByIndex(0);

  // Lectura ritmo cardíaco (simplificado)
  currentReading.heartRate = readHeartRate();
}

bool detectBodyContact() {
  return (currentReading.pressure > 15 &&
          currentReading.temperature > 30);
}

void startContactSession() {
  isContactActive = true;
  contactStartTime = millis();
  Serial.println("Iniciando sesión de contacto corporal");

  // Identificación de zona corporal basada en temperatura y presión
  currentReading.contactZone = identifyBodyZone();

  // LED indicador de sesión activa
  digitalWrite(STATUS_LED_PIN, LOW);
}

void updateContactReading() {
  currentReading.contactDuration = millis() - contactStartTime;

  // Análisis de patrones emocionales en tiempo real
  analyzeEmotionalPattern();
}

void endContactSession() {
  isContactActive = false;
  Serial.println("Finalizando sesión - Enviando a Claude para biodecodificación");

  // Envío de datos a Claude para interpretación
```

```
  sendToClaude();

  digitalWrite(STATUS_LED_PIN, HIGH);
}

String identifyBodyZone() {
  // Algoritmo simplificado de identificación basado en temperatura
  if (currentReading.temperature > 36.5) {
    if (currentReading.pressure > 50) return "pecho";
    else return "abdomen";
  } else if (currentReading.temperature > 35.5) {
    return "brazos";
  } else {
    return "piernas";
  }
}

void analyzeEmotionalPattern() {
  // Análisis de variabilidad cardíaca para detección emocional
  static int previousHR = 0;
  int hrVariability = abs(currentReading.heartRate - previousHR);

  if (hrVariability > 10) {
    currentReading.emotionalPattern = "activacion_emocional";
  } else if (currentReading.heartRate < 60) {
    currentReading.emotionalPattern = "estado_relajacion";
  } else {
    currentReading.emotionalPattern = "equilibrio_autonomico";
  }

  previousHR = currentReading.heartRate;
}

int readHeartRate() {
  // Simulación de lectura de pulso (implementar fotopletismografía)
  static unsigned long lastBeat = 0;
  static int currentBPM = 70;

  // Aquí iría el algoritmo real de detección de pulso
  // Por ahora simulamos variaciones naturales
  if (millis() - lastBeat > 800) {
    currentBPM = 70 + random(-10, 15);
    lastBeat = millis();
  }
```

```
  return currentBPM;
}

void sendToClaude() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin(claude_endpoint);

    // Headers para Claude API
    http.addHeader("Content-Type", "application/json");
    http.addHeader("x-api-key", claude_api_key);
    http.addHeader("anthropic-version", "2023-06-01");

    // Construcción del prompt especializado en biodecodificación
    String claudePrompt = buildBiodecodingPrompt();

    // JSON payload para Claude
    DynamicJsonDocument doc(2048);
    doc["model"] = "claude-3-sonnet-20240229";
    doc["max_tokens"] = 1000;

    JsonArray messages = doc.createNestedArray("messages");
    JsonObject message = messages.createNestedObject();
    message["role"] = "user";
    message["content"] = claudePrompt;

    String requestBody;
    serializeJson(doc, requestBody);

    // Envío de solicitud
    int httpResponseCode = http.POST(requestBody);

    if (httpResponseCode == 200) {
      String response = http.getString();
      processClaudeResponse(response);
    } else {
      Serial.println("Error en comunicación con Claude: " + String(httpResponseCode));
    }

    http.end();
  }
}
```

```
String buildBiodecodingPrompt() {
  String prompt = "Actúa como especialista en biodecodificación emocional. Analiza estos datos sensoriales del anillo biodecodificador:\n\n";

  prompt += "DATOS BIOMÉTRICOS:\n";
  prompt += "- Zona corporal contactada: " + currentReading.contactZone + "\n";
  prompt += "- Presión aplicada: " + String(currentReading.pressure) + "%\n";
  prompt += "- Temperatura de contacto: " + String(currentReading.temperature) + "°C\n";
  prompt += "- Ritmo cardíaco: " + String(currentReading.heartRate) + " BPM\n";
  prompt += "- Duración de contacto: " + String(currentReading.contactDuration/1000) + " segundos\n";
  prompt += "- Patrón emocional detectado: " + currentReading.emotionalPattern + "\n\n";

  // Contexto biodecodificador preconfigurado
  prompt += "CONTEXTO DE BIODECODIFICACIÓN:\n";
  for (int i = 0; i < 7; i++) {
    if (biodecodingAtlas[i].bodyZone == currentReading.contactZone) {
      prompt += "- Conflicto biológico asociado: " + biodecodingAtlas[i].primaryConflict + "\n";
      prompt += "- Fase biológica probable: " + biodecodingAtlas[i].biologicalPhase + "\n";
      break;
    }
  }

  prompt += "\nPROPORCIONA:\n";
  prompt += "1. Interpretación biodecodificada del síntoma/contacto\n";
  prompt += "2. Conflicto emocional subyacente probable\n";
  prompt += "3. Fase biológica de la manifestación\n";
  prompt += "4. Tres actividades terapéuticas específicas\n";
  prompt += "5. Mensaje de acompañamiento empático\n\n";
  prompt += "Responde en formato JSON estructurado para la aplicación móvil.";

  return prompt;
}

void processClaudeResponse(String response) {
  // Procesamiento de respuesta de Claude
  DynamicJsonDocument doc(2048);
  deserializeJson(doc, response);

  if (doc.containsKey("content")) {
    JsonArray content = doc["content"];
    if (content.size() > 0) {
      String claudeInterpretation = content[0]["text"];
```

```
      Serial.println("=== INTERPRETACIÓN BIODECODIFICADA ===");
      Serial.println(claudeInterpretation);

      // Aquí se enviaría a la app móvil vía Bluetooth
      sendToMobileApp(claudeInterpretation);
    }
  }
}

void sendToMobileApp(String interpretation) {
  // Transmisión Bluetooth hacia la aplicación móvil
  Serial.println("Enviando interpretación a app móvil...");

  // Construcción del paquete de datos para la app
  DynamicJsonDocument appData(1500);
  appData["timestamp"] = millis();
  appData["session_id"] = String(millis());
  appData["body_zone"] = currentReading.contactZone;
  appData["biometric_data"]["pressure"] = currentReading.pressure;
  appData["biometric_data"]["temperature"] = currentReading.temperature;
  appData["biometric_data"]["heart_rate"] = currentReading.heartRate;
  appData["contact_duration"] = currentReading.contactDuration;
  appData["claude_interpretation"] = interpretation;

  String jsonString;
  serializeJson(appData, jsonString);

  // Aquí se implementaría la transmisión Bluetooth LE
  // BluetoothSerial.println(jsonString);

  Serial.println("Datos enviados a aplicación móvil");
}
```

Integración de Hardware - Anillo Biodetector Emocional

Fase 3: Integración del Hardware del Anillo

3.1 Arquitectura del Sistema Integrado

Componentes Principales Seleccionados:

- **Microcontrolador**: ESP32-C3 (ultra bajo consumo, WiFi/Bluetooth integrado)
- **Sensores Biométricos**:
    - MAX30102 (frecuencia cardíaca y SpO2)
    - Sensor de temperatura corporal NTC 10kΩ
    - Acelerómetro/giroscopio MPU-6050 (detección de gestos)
- **Actuadores**: Motor de vibración lineal LRA (feedback háptico)
- **Alimentación**: Batería Li-Po 40mAh recargable
- **Conectividad**: Bluetooth Low Energy (BLE) 5.0

3.2 Diseño de la PCB del Anillo

Especificaciones Técnicas:

- **Dimensiones**: 20mm x 8mm x 3mm (PCB flexible)
- **Capas**: 4 capas para optimizar espacio y reducir interferencias
- **Material**: Poliimida flexible para adaptarse a la curvatura del anillo

Distribución de Componentes:

[Batería] ← [ESP32-C3] → [Sensores]
  ↓ ↓ ↓
[Carga] ← [Reguladores] → [Botón]

3.3 Esquema de Conexiones

Conexiones del ESP32-C3:

- GPIO0: Botón principal (entrada con pull-up)
- GPIO1-2: I2C (SDA/SCL) para sensores
- GPIO3: Control del motor de vibración
- GPIO4-5: LEDs de estado (opcional)
- GPIO6: Sensor de temperatura (ADC)

Protocolo de Comunicación:

- **I2C**: Comunicación con sensores (400kHz)
- **SPI**: Interface con memoria flash externa
- **UART**: Debug y programación
- **BLE**: Transmisión de datos a la APP

3.4 Gestión de Energía

Sistema de Alimentación:

- **Batería Principal**: Li-Po 40mAh, 3.7V
- **Regulador LDO**: 3.3V para lógica digital
- **Regulador Lineal**: 1.8V para sensores analógicos
- **Carga Inalámbrica**: Bobina de 10mm para carga por inducción

Optimización de Consumo:

```
// Modos de operación del ESP32-C3
typedef enum {
    ACTIVE_MODE,     // 80mA - Transmisión activa
    SENSING_MODE,    // 15mA - Solo sensores activos
    STANDBY_MODE,    // 2mA - Monitoreo básico
    DEEP_SLEEP_MODE  // 10µA - Solo RTC activo
} power_mode_t;
```

3.5 Algoritmos de Detección Biométrica

Procesamiento de Señales:

```
// Estructura de datos biométricos
typedef struct {
    uint16_t heart_rate;      // BPM
    uint16_t hrv;             // Variabilidad cardíaca (ms)
    float temperature;        // Temperatura corporal (°C)
    uint16_t stress_index;    // Índice de estrés calculado
    gesture_t gesture;        // Gesto detectado
    timestamp_t time;         // Marca temporal
} biometric_data_t;

// Filtro digital para señal cardíaca
float butterworth_filter(float input, float* history) {
    // Implementación de filtro Butterworth orden 2
    static const float a[] = {1.0, -1.142, 0.412};
    static const float b[] = {0.067, 0.134, 0.067};

    float output = b[0] * input + b[1] * history[0] + b[2] * history[1]
                 - a[1] * history[2] - a[2] * history[3];

    // Actualizar historial
    history[1] = history[0];
    history[0] = input;
    history[3] = history[2];
    history[2] = output;
```

```
    return output;
}
```

3.6 Protocolo de Comunicación BLE

Servicios BLE Definidos:

```
// UUID del servicio principal
#define BIOEMOTIONAL_SERVICE_UUID "12345678-1234-1234-1234-123456789abc"

// Características BLE
typedef struct {
    char uuid[37];
    uint8_t properties;
    char description[50];
} ble_characteristic_t;

ble_characteristic_t characteristics[] = {
    {"12345678-1234-1234-1234-123456789ab1", BLE_GATT_CHR_F_READ |
BLE_GATT_CHR_F_NOTIFY, "Datos biométricos"},
    {"12345678-1234-1234-1234-123456789ab2", BLE_GATT_CHR_F_WRITE,
"Configuración"},
    {"12345678-1234-1234-1234-123456789ab3", BLE_GATT_CHR_F_READ, "Estado batería"},
    {"12345678-1234-1234-1234-123456789ab4", BLE_GATT_CHR_F_WRITE, "Calibración"}
};
```

3.7 Integración con Biodecodificación Emocional

Mapeo de Patrones Biométricos:

```
// Correlación entre datos biométricos y emociones
typedef struct {
    float hr_baseline;        // Frecuencia cardíaca basal
    float hr_stress_threshold; // Umbral de estrés
    float temp_variation;     // Variación térmica
    emotion_pattern_t pattern; // Patrón emocional asociado
} emotion_biometric_map_t;

// Patrones de biodecodificación
emotion_biometric_map_t emotion_patterns[] = {
    {72.0, 90.0, 0.5, ANXIETY_PATTERN},      // Ansiedad
    {65.0, 85.0, -0.3, DEPRESSION_PATTERN}, // Depresión
    {80.0, 100.0, 1.0, ANGER_PATTERN},      // Ira
```

```
    {70.0, 80.0, 0.2, CALM_PATTERN}         // Calma
};
```

3.8 Proceso de Ensamblaje

Secuencia de Montaje:

1. **Soldadura de componentes SMD** en PCB flexible
2. **Pruebas eléctricas** de continuidad y funcionalidad
3. **Calibración de sensores** en ambiente controlado
4. **Integración en carcasa** del anillo
5. **Sellado hermético** para resistencia al agua
6. **Pruebas finales** de conectividad y autonomía

Herramientas Requeridas:

- Estación de soldadura con control de temperatura (260°C-300°C)
- Microscopio estereoscópico (20x-40x)
- Multímetro de precisión
- Osciloscopio para análisis de señales
- Cámara térmica para verificación de disipación

3.9 Validación y Calibración

Protocolo de Calibración:

```
void calibrate_sensors(void) {
    // Calibración del sensor cardíaco
    calibrate_heart_rate_sensor();

    // Calibración térmica
    calibrate_temperature_sensor();

    // Calibración del acelerómetro
    calibrate_motion_sensor();

    // Guardar valores en memoria no volátil
    save_calibration_data();
}

// Función de auto-calibración continua
void adaptive_calibration(biometric_data_t* data) {
    static float baseline_hr = 0;
    static uint32_t sample_count = 0;
```

```
    // Actualizar línea base de frecuencia cardíaca
    baseline_hr = (baseline_hr * sample_count + data->heart_rate) / (sample_count + 1);
    sample_count++;

    // Ajustar umbrales dinámicamente
    adjust_detection_thresholds(baseline_hr);
}
```

3.10 Consideraciones de Biocompatibilidad

Materiales Certificados:

- **Carcasa**: Titanio grado 2 (hipoalergénico)
- **Sensores**: Recubrimiento biocompatible
- **Sellado**: Silicona médica USP Clase VI
- **Certificaciones**: ISO 10993 para dispositivos médicos

3.11 Próximos Pasos

Fase 4: Desarrollo de la Aplicación Móvil

- Diseño de la interfaz de usuario
- Algoritmos de procesamiento de datos
- Sistema de notificaciones inteligentes
- Base de datos de patrones emocionales

Fase 5: Integración Sistema Completo

- Sincronización anillo-aplicación
- Pruebas de campo con usuarios
- Refinamiento de algoritmos de detección
- Validación clínica del sistema

Recursos Técnicos Adicionales

Bibliotecas de Software Recomendadas:

- **ESP-IDF**: Framework oficial para ESP32
- **FreeRTOS**: Sistema operativo en tiempo real
- **NimBLE**: Stack Bluetooth de bajo consumo
- **ESP-DSP**: Biblioteca de procesamiento digital de señales

Proveedores de Componentes:

- **Mouser Electronics**: Componentes electrónicos
- **JLCPCB**: Fabricación de PCB flexibles
- **Shenzhen Manufacturers**: Carcasas de titanio personalizadas

---

```
// ======================================================
// PROTOTIPO ANILLO BIODECODIFICADOR EMOCIONAL
// Sistema Integrado de Detección Corporal y Decodificación
// ======================================================

import 'dart:convert';
import 'dart:async';

// ======================================================
// 1. MODELO DE DATOS FUNDAMENTAL
// ======================================================

enum TipoSintoma {
  fisico,
  emocional,
  energetico,
  mixto
}

enum NivelIntensidad {
  leve,      // 1-3
  moderado,  // 4-6
  intenso,   // 7-8
  severo     // 9-10
}

enum ZonaCorporal {
  cabeza,
  cuello,
  torax,
  abdomen,
  brazos,
  manos,
  piernas,
  pies,
```

```
  espalda,
  corazon,
  garganta,
  estomago
}

// ====================================================
// 2. MODELO SESIÓN EMOCIONAL EXPANDIDO
// ====================================================

class SesionEmocional {
  final String id;
  final DateTime fechaInicio;
  DateTime? fechaFin;
  final String usuarioId;

  // Datos del anillo sensor
  final String anilloId;
  final Map<String, dynamic> datosSensor;

  // Detección corporal
  final ZonaCorporal zonaCorporalDetectada;
  final NivelIntensidad intensidadDetectada;
  final TipoSintoma tipoSintoma;

  // Biodecodificación emocional
  final String sintomaPrimario;
  final List<String> sintomasSecundarios;
  final String conflictoEmocional;
  final String patronBiologico;
  final String significadoSimbolico;

  // Contexto emocional
  final String estadoEmocionalPrevio;
  final List<String> emocionesPredominantes;
  final String situacionDesencadenante;

  // Análisis profundo
  final String arquetipoEmocional;
  final String etapaEvolutiva;
  final List<String> creenciasLimitantes;
  final String misionDelSintoma;

  // Recomendaciones terapéuticas
```

```
final List<String> floresTomadas;
final List<String> afirmacionesSugeridas;
final String ejercicioTerapeutico;
final String meditacionGuiada;

// Métricas y seguimiento
final double nivelConciencia;
final double gradoResistencia;
final double potencialSanacion;
final Map<String, double> indicadoresBiologicos;

// Notas del terapeuta/usuario
final String notasPersonales;
final List<String> insights;
final String planSeguimiento;

SesionEmocional({
  required this.id,
  required this.fechaInicio,
  this.fechaFin,
  required this.usuarioId,
  required this.anilloId,
  required this.datosSensor,
  required this.zonaCorporalDetectada,
  required this.intensidadDetectada,
  required this.tipoSintoma,
  required this.sintomaPrimario,
  required this.sintomasSecundarios,
  required this.conflictoEmocional,
  required this.patronBiologico,
  required this.significadoSimbolico,
  required this.estadoEmocionalPrevio,
  required this.emocionesPredominantes,
  required this.situacionDesencadenante,
  required this.arquetipoEmocional,
  required this.etapaEvolutiva,
  required this.creenciasLimitantes,
  required this.misionDelSintoma,
  required this.floresTomadas,
  required this.afirmacionesSugeridas,
  required this.ejercicioTerapeutico,
  required this.meditacionGuiada,
  required this.nivelConciencia,
  required this.gradoResistencia,
```

```
  required this.potencialSanacion,
  required this.indicadoresBiologicos,
  required this.notasPersonales,
  required this.insights,
  required this.planSeguimiento,
});

// Duración de la sesión
Duration get duracion {
  if (fechaFin != null) {
    return fechaFin!.difference(fechaInicio);
  }
  return DateTime.now().difference(fechaInicio);
}

// Serialización para almacenamiento
Map<String, dynamic> toJson() {
  return {
    'id': id,
    'fechaInicio': fechaInicio.toIso8601String(),
    'fechaFin': fechaFin?.toIso8601String(),
    'usuarioId': usuarioId,
    'anilloId': anilloId,
    'datosSensor': datosSensor,
    'zonaCorporalDetectada': zonaCorporalDetectada.name,
    'intensidadDetectada': intensidadDetectada.name,
    'tipoSintoma': tipoSintoma.name,
    'sintomaPrimario': sintomaPrimario,
    'sintomasSecundarios': sintomasSecundarios,
    'conflictoEmocional': conflictoEmocional,
    'patronBiologico': patronBiologico,
    'significadoSimbolico': significadoSimbolico,
    'estadoEmocionalPrevio': estadoEmocionalPrevio,
    'emocionesPredominantes': emocionesPredominantes,
    'situacionDesencadenante': situacionDesencadenante,
    'arquetipoEmocional': arquetipoEmocional,
    'etapaEvolutiva': etapaEvolutiva,
    'creenciasLimitantes': creenciasLimitantes,
    'misionDelSintoma': misionDelSintoma,
    'floresTomadas': floresTomadas,
    'afirmacionesSugeridas': afirmacionesSugeridas,
    'ejercicioTerapeutico': ejercicioTerapeutico,
    'meditacionGuiada': meditacionGuiada,
    'nivelConciencia': nivelConciencia,
```

```
    'gradoResistencia': gradoResistencia,
    'potencialSanacion': potencialSanacion,
    'indicadoresBiologicos': indicadoresBiologicos,
    'notasPersonales': notasPersonales,
    'insights': insights,
    'planSeguimiento': planSeguimiento,
  };
}

factory SesionEmocional.fromJson(Map<String, dynamic> json) {
  return SesionEmocional(
    id: json['id'],
    fechaInicio: DateTime.parse(json['fechaInicio']),
    fechaFin: json['fechaFin'] != null ? DateTime.parse(json['fechaFin']) : null,
    usuarioId: json['usuarioId'],
    anilloId: json['anilloId'],
    datosSensor: json['datosSensor'],
    zonaCorporalDetectada: ZonaCorporal.values.firstWhere(
      (e) => e.name == json['zonaCorporalDetectada']
    ),
    intensidadDetectada: NivelIntensidad.values.firstWhere(
      (e) => e.name == json['intensidadDetectada']
    ),
    tipoSintoma: TipoSintoma.values.firstWhere(
      (e) => e.name == json['tipoSintoma']
    ),
    sintomaPrimario: json['sintomaPrimario'],
    sintomasSecundarios: List<String>.from(json['sintomasSecundarios']),
    conflictoEmocional: json['conflictoEmocional'],
    patronBiologico: json['patronBiologico'],
    significadoSimbolico: json['significadoSimbolico'],
    estadoEmocionalPrevio: json['estadoEmocionalPrevio'],
    emocionesPredominantes: List<String>.from(json['emocionesPredominantes']),
    situacionDesencadenante: json['situacionDesencadenante'],
    arquetipoEmocional: json['arquetipoEmocional'],
    etapaEvolutiva: json['etapaEvolutiva'],
    creenciasLimitantes: List<String>.from(json['creenciasLimitantes']),
    misionDelSintoma: json['misionDelSintoma'],
    floresTomadas: List<String>.from(json['floresTomadas']),
    afirmacionesSugeridas: List<String>.from(json['afirmacionesSugeridas']),
    ejercicioTerapeutico: json['ejercicioTerapeutico'],
    meditacionGuiada: json['meditacionGuiada'],
    nivelConciencia: json['nivelConciencia'].toDouble(),
    gradoResistencia: json['gradoResistencia'].toDouble(),
```

```
      potencialSanacion: json['potencialSanacion'].toDouble(),
      indicadoresBiologicos: Map<String, double>.from(json['indicadoresBiologicos']),
      notasPersonales: json['notasPersonales'],
      insights: List<String>.from(json['insights']),
      planSeguimiento: json['planSeguimiento'],
    );
  }
}

// ========================================================
// 3. SERVICIO DE DETECCIÓN DEL ANILLO
// ========================================================

class AnilloSensorService {
  final StreamController<Map<String, dynamic>> _sensorController =
      StreamController<Map<String, dynamic>>.broadcast();

  Stream<Map<String, dynamic>> get sensorStream => _sensorController.stream;

  // Simulación de detección del anillo en diferentes zonas corporales
  Future<Map<String, dynamic>> detectarZonaCorporal() async {
    // Simular lectura de sensores del anillo
    await Future.delayed(Duration(seconds: 2));

    // Datos simulados del sensor
    Map<String, dynamic> datosSensor = {
      'timestamp': DateTime.now().toIso8601String(),
      'temperatura': 36.5 + (DateTime.now().millisecond / 1000),
      'pulso': 70 + (DateTime.now().millisecond / 100),
      'conductividad': 2.3 + (DateTime.now().millisecond / 10000),
      'presion': 120 + (DateTime.now().millisecond / 1000),
      'vibracion': DateTime.now().millisecond / 1000,
      'proximidad': determinarProximidadZona(),
    };

    _sensorController.add(datosSensor);
    return datosSensor;
  }

  ZonaCorporal determinarProximidadZona() {
    // Algoritmo simulado para determinar zona corporal
    List<ZonaCorporal> zonas = ZonaCorporal.values;
    return zonas[DateTime.now().millisecond % zonas.length];
  }
```

```
  void dispose() {
    _sensorController.close();
  }
}

// ========================================================
// 4. MOTOR DE BIODECODIFICACIÓN EMOCIONAL
// ========================================================

class BiodecodificacionEngine {

  // Mapeo de zonas corporales a conflictos emocionales
  static const Map<ZonaCorporal, List<String>> _mapaConflictos = {
    ZonaCorporal.cabeza: [
      'Conflictos de autoridad y poder',
      'Exceso de control mental',
      'Negación de la intuición'
    ],
    ZonaCorporal.cuello: [
      'Rigidez en la comunicación',
      'Dificultad para expresar emociones',
      'Conflictos entre razón y corazón'
    ],
    ZonaCorporal.torax: [
      'Conflictos del corazón emocional',
      'Dificultades respiratorias de la vida',
      'Opresión en el pecho vital'
    ],
    ZonaCorporal.abdomen: [
      'Conflictos digestivos emocionales',
      'Dificultad para procesar experiencias',
      'Miedo a lo desconocido'
    ],
    ZonaCorporal.brazos: [
      'Conflictos de acción y creatividad',
      'Dificultad para abrazar la vida',
      'Resistencia al cambio'
    ],
    ZonaCorporal.manos: [
      'Conflictos de manifestación',
      'Dificultad para dar y recibir',
      'Bloqueos creativos'
    ],
```

```
  ZonaCorporal.piernas: [
    'Conflictos de avance en la vida',
    'Miedo al futuro',
    'Falta de apoyo emocional'
  ],
  ZonaCorporal.pies: [
    'Conflictos de conexión con la tierra',
    'Falta de enraizamiento',
    'Dificultades en el camino vital'
  ],
};

// Arquetipos emocionales según la biodecodificación
static const List<String> _arquetiposEmocionales = [
  'El Guerrero Herido',
  'La Madre Sacrificada',
  'El Niño Abandonado',
  'El Juez Implacable',
  'El Sanador Herido',
  'El Rebelde Incomprendido',
  'El Perfeccionista Ansioso',
];

Future<Map<String, dynamic>> procesarBiodecodificacion({
  required ZonaCorporal zona,
  required NivelIntensidad intensidad,
  required Map<String, dynamic> datosSensor,
}) async {

  await Future.delayed(Duration(seconds: 1)); // Simular procesamiento

  String conflictoPrincipal = _determinarConflictoEmocional(zona, intensidad);
  String arquetipo = _determinarArquetipo(zona, intensidad);
  String patronBiologico = _determinarPatronBiologico(zona);
  String significadoSimbolico = _determinarSignificadoSimbolico(zona);
  String misionDelSintoma = _determinarMisionDelSintoma(zona, intensidad);

  return {
    'conflictoEmocional': conflictoPrincipal,
    'arquetipoEmocional': arquetipo,
    'patronBiologico': patronBiologico,
    'significadoSimbolico': significadoSimbolico,
    'misionDelSintoma': misionDelSintoma,
    'floresTomadas': _sugerirFloresBach(zona),
```

```
    'afirmacionesSugeridas': _generarAfirmaciones(zona),
    'ejercicioTerapeutico': _sugerirEjercicio(zona),
    'meditacionGuiada': _sugerirMeditacion(zona),
    'nivelConciencia': _calcularNivelConciencia(intensidad),
    'gradoResistencia': _calcularGradoResistencia(intensidad),
    'potencialSanacion': _calcularPotencialSanacion(zona, intensidad),
  };
}

String _determinarConflictoEmocional(ZonaCorporal zona, NivelIntensidad intensidad) {
  List<String> conflictos = _mapaConflictos[zona] ?? ['Conflicto emocional general'];
  int index = intensidad.index % conflictos.length;
  return conflictos[index];
}

String _determinarArquetipo(ZonaCorporal zona, NivelIntensidad intensidad) {
  int index = (zona.index + intensidad.index) % _arquetiposEmocionales.length;
  return _arquetiposEmocionales[index];
}

String _determinarPatronBiologico(ZonaCorporal zona) {
  switch (zona) {
    case ZonaCorporal.cabeza:
      return 'Patrón neurológico de hiperactividad mental';
    case ZonaCorporal.cuello:
      return 'Patrón de tensión comunicativa';
    case ZonaCorporal.torax:
      return 'Patrón cardio-respiratorio emocional';
    case ZonaCorporal.abdomen:
      return 'Patrón digestivo de procesamiento emocional';
    default:
      return 'Patrón biológico de adaptación';
  }
}

String _determinarSignificadoSimbolico(ZonaCorporal zona) {
  switch (zona) {
    case ZonaCorporal.cabeza:
      return 'Centro de comando y control de la vida';
    case ZonaCorporal.cuello:
      return 'Puente entre pensar y sentir';
    case ZonaCorporal.torax:
      return 'Templo del amor y la respiración vital';
    case ZonaCorporal.abdomen:
```

```
      return 'Laboratorio de transformación emocional';
    default:
      return 'Zona de manifestación vital';
  }
}

String _determinarMisionDelSintoma(ZonaCorporal zona, NivelIntensidad intensidad) {
  return 'Invitarte a reconocer y sanar el patrón emocional subyacente en ${zona.name}';
}

List<String> _sugerirFloresBach(ZonaCorporal zona) {
  switch (zona) {
    case ZonaCorporal.cabeza:
      return ['White Chestnut', 'Vervain', 'Impatiens'];
    case ZonaCorporal.cuello:
      return ['Agrimony', 'Mimulus', 'Larch'];
    case ZonaCorporal.torax:
      return ['Holly', 'Willow', 'Pine'];
    default:
      return ['Rescue Remedy', 'Centaury', 'Oak'];
  }
}

List<String> _generarAfirmaciones(ZonaCorporal zona) {
  switch (zona) {
    case ZonaCorporal.cabeza:
      return [
        'Mi mente está en calma y equilibrio',
        'Permito que la sabiduría fluya a través de mí',
        'Libero el control y confío en la vida'
      ];
    case ZonaCorporal.torax:
      return [
        'Mi corazón está abierto al amor',
        'Respiro profundamente la vida',
        'Libero todas las tensiones de mi pecho'
      ];
    default:
      return [
        'Sano y libero este patrón emocional',
        'Mi cuerpo es mi aliado en la sanación',
        'Permito que la energía fluya libremente'
      ];
  }
```

```
  }

  String _sugerirEjercicio(ZonaCorporal zona) {
    switch (zona) {
      case ZonaCorporal.cabeza:
        return 'Masaje circular en las sienes con aceite de lavanda';
      case ZonaCorporal.cuello:
        return 'Ejercicios de rotación suave del cuello';
      case ZonaCorporal.torax:
        return 'Respiración consciente con expansión del pecho';
      default:
        return 'Movimiento suave y consciente de la zona afectada';
    }
  }

  String _sugerirMeditacion(ZonaCorporal zona) {
    return 'Meditación de sanación enfocada en ${zona.name} con visualización de luz dorada';
  }

  double _calcularNivelConciencia(NivelIntensidad intensidad) {
    switch (intensidad) {
      case NivelIntensidad.leve:
        return 0.3;
      case NivelIntensidad.moderado:
        return 0.5;
      case NivelIntensidad.intenso:
        return 0.7;
      case NivelIntensidad.severo:
        return 0.9;
    }
  }

  double _calcularGradoResistencia(NivelIntensidad intensidad) {
    return 1.0 - _calcularNivelConciencia(intensidad);
  }

  double _calcularPotencialSanacion(ZonaCorporal zona, NivelIntensidad intensidad) {
    double baseConciencia = _calcularNivelConciencia(intensidad);
    // Algunas zonas tienen mayor potencial de sanación
    double factorZona = zona == ZonaCorporal.corazon ? 1.2 : 1.0;
    return (baseConciencia * factorZona).clamp(0.0, 1.0);
  }
}
```

```
// ==========================================================
// 5. CONTROLADOR PRINCIPAL DEL PROTOTIPO
// ==========================================================

class PrototipoAnilloBiodecodificador {
  final AnilloSensorService _anilloService = AnilloSensorService();
  final BiodecodificacionEngine _biodecodificador = BiodecodificacionEngine();

  final StreamController<SesionEmocional> _sesionController =
      StreamController<SesionEmocional>.broadcast();

  Stream<SesionEmocional> get sesionStream => _sesionController.stream;

  SesionEmocional? _sesionActual;

  Future<SesionEmocional> iniciarSesionBiodecodificacion(String usuarioId) async {
    print('🔮 Iniciando sesión de biodecodificación...');

    // 1. Detectar zona corporal con el anillo
    print('📱 Detectando zona corporal con anillo sensor...');
    Map<String, dynamic> datosSensor = await _anilloService.detectarZonaCorporal();
    ZonaCorporal zonaDetectada = _anilloService.determinarProximidadZona();

    // 2. Determinar intensidad basada en datos del sensor
    NivelIntensidad intensidad = _determinarIntensidad(datosSensor);

    // 3. Procesar biodecodificación
    print('🧬 Procesando biodecodificación emocional...');
    Map<String, dynamic> resultadoBiodecodificacion =
        await _biodecodificador.procesarBiodecodificacion(
      zona: zonaDetectada,
      intensidad: intensidad,
      datosSensor: datosSensor,
    );

    // 4. Crear sesión emocional completa
    _sesionActual = SesionEmocional(
      id: DateTime.now().millisecondsSinceEpoch.toString(),
      fechaInicio: DateTime.now(),
      usuarioId: usuarioId,
      anilloId: 'ANILLO_PROTO_001',
      datosSensor: datosSensor,
      zonaCorporalDetectada: zonaDetectada,
      intensidadDetectada: intensidad,
```

```
    tipoSintoma: TipoSintoma.emocional,
    sintomaPrimario: 'Tensión en ${zonaDetectada.name}',
    sintomasSecundarios: ['Malestar general', 'Inquietud emocional'],
    conflictoEmocional: resultadoBiodecodificacion['conflictoEmocional'],
    patronBiologico: resultadoBiodecodificacion['patronBiologico'],
    significadoSimbolico: resultadoBiodecodificacion['significadoSimbolico'],
    estadoEmocionalPrevio: 'Estado de tensión emocional',
    emocionesPredominantes: ['Ansiedad', 'Preocupación', 'Resistencia'],
    situacionDesencadenante: 'Situación de estrés detectada por el anillo',
    arquetipoEmocional: resultadoBiodecodificacion['arquetipoEmocional'],
    etapaEvolutiva: 'Fase de reconocimiento',
    creenciasLimitantes: ['No puedo relajarme', 'Debo controlarlo todo'],
    misionDelSintoma: resultadoBiodecodificacion['misionDelSintoma'],
    floresTomadas: resultadoBiodecodificacion['floresTomadas'],
    afirmacionesSugeridas: resultadoBiodecodificacion['afirmacionesSugeridas'],
    ejercicioTerapeutico: resultadoBiodecodificacion['ejercicioTerapeutico'],
    meditacionGuiada: resultadoBiodecodificacion['meditacionGuiada'],
    nivelConciencia: resultadoBiodecodificacion['nivelConciencia'],
    gradoResistencia: resultadoBiodecodificacion['gradoResistencia'],
    potencialSanacion: resultadoBiodecodificacion['potencialSanacion'],
    indicadoresBiologicos: {
      'variabilidadCardíaca': datosSensor['pulso'],
      'conductividad': datosSensor['conductividad'],
      'temperatura': datosSensor['temperatura'],
    },
    notasPersonales: '',
    insights: [],
    planSeguimiento: 'Seguimiento en 24 horas',
  );

  _sesionController.add(_sesionActual!);

  print('✅ Sesión de biodecodificación completada');
  return _sesionActual!;
}

NivelIntensidad _determinarIntensidad(Map<String, dynamic> datosSensor) {
  double pulso = datosSensor['pulso'];
  double conductividad = datosSensor['conductividad'];

  // Algoritmo simple para determinar intensidad
  if (pulso > 90 || conductividad > 3.0) {
    return NivelIntensidad.intenso;
  } else if (pulso > 80 || conductividad > 2.5) {
```

```
      return NivelIntensidad.moderado;
    } else {
      return NivelIntensidad.leve;
    }
  }

  Future<void> finalizarSesion() async {
    if (_sesionActual != null) {
      _sesionActual!.fechaFin = DateTime.now();
      print('🔚 Sesión finalizada. Duración: ${_sesionActual!.duracion}');
    }
  }

  void dispose() {
    _anilloService.dispose();
    _sesionController.close();
  }
}

// ========================================================
// 6. EJEMPLO DE USO DEL PROTOTIPO
// ========================================================

Future<void> ejemploDeUso() async {
  print('🚀 INICIANDO PROTOTIPO ANILLO BIODECODIFICADOR');
  print('=' * 50);

  PrototipoAnilloBiodecodificador prototipo = PrototipoAnilloBiodecodificador();

  // Escuchar las sesiones
  prototipo.sesionStream.listen((sesion) {
    print('\n📊 REPORTE DE SESIÓN EMOCIONAL:');
    print('ID: ${sesion.id}');
    print('Zona detectada: ${sesion.zonaCorporalDetectada.name}');
    print('Intensidad: ${sesion.intensidadDetectada.name}');
    print('Conflicto emocional: ${sesion.conflictoEmocional}');
    print('Arquetipo: ${sesion.arquetipoEmocional}');
    print('Misión del síntoma: ${sesion.misionDelSintoma}');
    print('Flores sugeridas: ${sesion.floresTomadas.join(", ")}');
    print('Nivel de conciencia: ${(sesion.nivelConciencia * 100).toInt()}%');
    print('Potencial de sanación: ${(sesion.potencialSanacion * 100).toInt()}%');
    print('\n💡 AFIRMACIONES SUGERIDAS:');
    sesion.afirmacionesSugeridas.forEach((afirmacion) {
      print('  • $afirmacion');
```

```
  });
 });

 // Simular uso del anillo
 await prototipo.iniciarSesionBiodecodificacion('USUARIO_001');

 // Simular tiempo de sesión
 await Future.delayed(Duration(seconds: 5));

 await prototipo.finalizarSesion();
 prototipo.dispose();

 print('\n🎯 PROTOTIPO COMPLETADO EXITOSAMENTE');
}
```

```
//
// ========================================================================
// PROTOTIPO: ANILLO DETECTOR DE SÍNTOMAS
// Sistema integrado de detección corporal y análisis emocional
//
// ========================================================================

import 'dart:async';
import 'dart:convert';
import 'dart:math';

//
// ========================================================================
// MODELOS DE DATOS FUNDAMENTALES
//
// ========================================================================

/// Modelo base para representar una sesión emocional completa
class SesionEmocional {
 final String id;
 final DateTime timestamp;
 final String usuarioId;
```

```
final List<LecturaAnillo> lecturas;
final EstadoEmocional estadoInicial;
final EstadoEmocional? estadoFinal;
final Map<String, dynamic> contextoAdicional;
final int duracionMinutos;
final bool sesionCompleta;

SesionEmocional({
  required this.id,
  required this.timestamp,
  required this.usuarioId,
  required this.lecturas,
  required this.estadoInicial,
  this.estadoFinal,
  this.contextoAdicional = const {},
  required this.duracionMinutos,
  this.sesionCompleta = false,
});

Map<String, dynamic> toJson() => {
  'id': id,
  'timestamp': timestamp.toIso8601String(),
  'usuarioId': usuarioId,
  'lecturas': lecturas.map((l) => l.toJson()).toList(),
  'estadoInicial': estadoInicial.toJson(),
  'estadoFinal': estadoFinal?.toJson(),
  'contextoAdicional': contextoAdicional,
  'duracionMinutos': duracionMinutos,
  'sesionCompleta': sesionCompleta,
};

factory SesionEmocional.fromJson(Map<String, dynamic> json) => SesionEmocional(
  id: json['id'],
  timestamp: DateTime.parse(json['timestamp']),
  usuarioId: json['usuarioId'],
  lecturas: (json['lecturas'] as List)
      .map((l) => LecturaAnillo.fromJson(l))
      .toList(),
  estadoInicial: EstadoEmocional.fromJson(json['estadoInicial']),
  estadoFinal: json['estadoFinal'] != null
      ? EstadoEmocional.fromJson(json['estadoFinal'])
      : null,
  contextoAdicional: json['contextoAdicional'] ?? {},
  duracionMinutos: json['duracionMinutos'],
```

```
    sesionCompleta: json['sesionCompleta'] ?? false,
  );
}

/// Representa una lectura individual del anillo detector
class LecturaAnillo {
  final String id;
  final DateTime timestamp;
  final TipoSensor sensor;
  final double valor;
  final String unidadMedida;
  final ZonaCorporal zonaDetectada;
  final int nivelConfianza; // 0-100
  final Map<String, dynamic> metadatos;

  LecturaAnillo({
    required this.id,
    required this.timestamp,
    required this.sensor,
    required this.valor,
    required this.unidadMedida,
    required this.zonaDetectada,
    required this.nivelConfianza,
    this.metadatos = const {},
  });

  Map<String, dynamic> toJson() => {
    'id': id,
    'timestamp': timestamp.toIso8601String(),
    'sensor': sensor.toString(),
    'valor': valor,
    'unidadMedida': unidadMedida,
    'zonaDetectada': zonaDetectada.toString(),
    'nivelConfianza': nivelConfianza,
    'metadatos': metadatos,
  };

  factory LecturaAnillo.fromJson(Map<String, dynamic> json) => LecturaAnillo(
    id: json['id'],
    timestamp: DateTime.parse(json['timestamp']),
    sensor: TipoSensor.values.firstWhere((e) => e.toString() == json['sensor']),
    valor: json['valor'].toDouble(),
    unidadMedida: json['unidadMedida'],
    zonaDetectada: ZonaCorporal.values.firstWhere((e) => e.toString() == json['zonaDetectada']),
```

```
    nivelConfianza: json['nivelConfianza'],
    metadatos: json['metadatos'] ?? {},
  );
}

/// Estado emocional interpretado a partir de las lecturas
class EstadoEmocional {
  final String id;
  final DateTime timestamp;
  final EmotionType emocionPrimaria;
  final List<EmotionType> emocionesSecundarias;
  final int intensidad; // 1-10
  final double valencia; // -1.0 a 1.0 (negativo a positivo)
  final double activacion; // 0.0 a 1.0 (calma a excitación)
  final Map<String, double> indicadoresFisiologicos;
  final String interpretacion;

  EstadoEmocional({
    required this.id,
    required this.timestamp,
    required this.emocionPrimaria,
    this.emocionesSecundarias = const [],
    required this.intensidad,
    required this.valencia,
    required this.activacion,
    this.indicadoresFisiologicos = const {},
    required this.interpretacion,
  });

  Map<String, dynamic> toJson() => {
    'id': id,
    'timestamp': timestamp.toIso8601String(),
    'emocionPrimaria': emocionPrimaria.toString(),
    'emocionesSecundarias': emocionesSecundarias.map((e) => e.toString()).toList(),
    'intensidad': intensidad,
    'valencia': valencia,
    'activacion': activacion,
    'indicadoresFisiologicos': indicadoresFisiologicos,
    'interpretacion': interpretacion,
  };

  factory EstadoEmocional.fromJson(Map<String, dynamic> json) => EstadoEmocional(
    id: json['id'],
    timestamp: DateTime.parse(json['timestamp']),
```

```
    emocionPrimaria: EmotionType.values.firstWhere((e) => e.toString() ==
json['emocionPrimaria']),
    emocionesSecundarias: (json['emocionesSecundarias'] as List)
        .map((e) => EmotionType.values.firstWhere((type) => type.toString() == e))
        .toList(),
    intensidad: json['intensidad'],
    valencia: json['valencia'].toDouble(),
    activacion: json['activacion'].toDouble(),
    indicadoresFisiologicos: Map<String, double>.from(json['indicadoresFisiologicos'] ?? {}),
    interpretacion: json['interpretacion'],
  );
}

//
==========================================================================
====
// ENUMERACIONES Y TIPOS
//
==========================================================================
====

enum TipoSensor {
  proximidad,
  presion,
  temperatura,
  pulso,
  conductanciaGalvanica,
  acelerometro,
  giroscopio,
  magnetometro
}

enum ZonaCorporal {
  frente,
  sienes,
  mejillas,
  barbilla,
  cuello,
  hombros,
  brazos,
  munecas,
  pecho,
  espalda,
  abdomen,
```

```
  cintura,
  caderas,
  muslos,
  rodillas,
  pantorrillas,
  tobillos,
  desconocida
}

enum EmotionType {
  alegria,
  tristeza,
  ira,
  miedo,
  sorpresa,
  disgusto,
  anticipacion,
  confianza,
  ansiedad,
  estres,
  relajacion,
  euforia,
  melancolia,
  frustracion,
  curiosidad,
  aburrimiento,
  neutral
}

//
================================================================================
// SIMULADOR DEL HARDWARE DEL ANILLO
//
================================================================================

class AnilloDetectorSimulador {
  static const String _version = "1.0.0";
  bool _conectado = false;
  Timer? _timerLecturas;
  final StreamController<LecturaAnillo> _streamController = StreamController.broadcast();
  final Random _random = Random();
```

```
Stream<LecturaAnillo> get streamLecturas => _streamController.stream;
bool get conectado => _conectado;
String get version => _version;

/// Simula la conexión Bluetooth con el anillo
Future<bool> conectarAnillo() async {
  print("🔵 Iniciando conexión Bluetooth con anillo detector...");
  await Future.delayed(Duration(seconds: 2));

  _conectado = _random.nextBool() ? true : _random.nextDouble() > 0.2;

  if (_conectado) {
    print("✅ Anillo conectado exitosamente");
    _iniciarLecturasContinuas();
    return true;
  } else {
    print("❌ Error de conexión con el anillo");
    return false;
  }
}

void _iniciarLecturasContinuas() {
  _timerLecturas = Timer.periodic(Duration(milliseconds: 500), (timer) {
    if (_conectado) {
      _generarLecturaSimulada();
    }
  });
}

void _generarLecturaSimulada() {
  final sensores = [
    TipoSensor.proximidad,
    TipoSensor.presion,
    TipoSensor.temperatura,
    TipoSensor.pulso,
    TipoSensor.conductanciaGalvanica
  ];

  final zonas = [
    ZonaCorporal.frente,
    ZonaCorporal.sienes,
    ZonaCorporal.cuello,
    ZonaCorporal.munecas,
    ZonaCorporal.pecho
```

```
];

final sensor = sensores[_random.nextInt(sensores.length)];
final zona = zonas[_random.nextInt(zonas.length)];

double valor;
String unidad;

switch (sensor) {
  case TipoSensor.proximidad:
    valor = _random.nextDouble() * 10; // 0-10 cm
    unidad = "cm";
    break;
  case TipoSensor.presion:
    valor = 80 + _random.nextDouble() * 40; // 80-120 mmHg
    unidad = "mmHg";
    break;
  case TipoSensor.temperatura:
    valor = 36.0 + _random.nextDouble() * 2; // 36-38°C
    unidad = "°C";
    break;
  case TipoSensor.pulso:
    valor = 60 + _random.nextDouble() * 40; // 60-100 bpm
    unidad = "bpm";
    break;
  default:
    valor = _random.nextDouble();
    unidad = "units";
}

final lectura = LecturaAnillo(
  id: DateTime.now().millisecondsSinceEpoch.toString(),
  timestamp: DateTime.now(),
  sensor: sensor,
  valor: valor,
  unidadMedida: unidad,
  zonaDetectada: zona,
  nivelConfianza: 70 + _random.nextInt(31), // 70-100%
  metadatos: {
    'bateria': 85 + _random.nextInt(15),
    'calidad_senal': _random.nextDouble(),
    'ruido_ambiental': _random.nextDouble() * 0.3,
  },
);
```

```
    _streamController.add(lectura);
  }

  void desconectar() {
    _conectado = false;
    _timerLecturas?.cancel();
    print("🔴 Anillo desconectado");
  }

  void dispose() {
    desconectar();
    _streamController.close();
  }
}

//
================================================================================
====
// MOTOR DE ANÁLISIS EMOCIONAL
//
================================================================================
====

class MotorAnalisisEmocional {
  final Map<String, List<LecturaAnillo>> _bufferLecturas = {};
  static const int _ventanaAnalisis = 10; // Últimas 10 lecturas para análisis

  /// Analiza las lecturas y genera un estado emocional
  EstadoEmocional analizarEstadoEmocional(List<LecturaAnillo> lecturas) {
    if (lecturas.isEmpty) {
      return _crearEstadoNeutral();
    }

    // Agregar lecturas al buffer
    for (var lectura in lecturas) {
      final key = lectura.sensor.toString();
      _bufferLecturas[key] ??= [];
      _bufferLecturas[key]!.add(lectura);

      // Mantener solo las últimas lecturas
      if (_bufferLecturas[key]!.length > _ventanaAnalisis) {
        _bufferLecturas[key]!.removeAt(0);
      }
```

```
    }

    return _procesarPatronesEmocionales();
  }

  EstadoEmocional _procesarPatronesEmocionales() {
    final indicadores = <String, double>{};

    // Análisis de pulso
    final lecturasPulso = _bufferLecturas[TipoSensor.pulso.toString()] ?? [];
    if (lecturasPulso.isNotEmpty) {
      final promedioPulso = lecturasPulso.map((l) => l.valor).reduce((a, b) => a + b) /
lecturasPulso.length;
      indicadores['pulso_promedio'] = promedioPulso;
      indicadores['variabilidad_pulso'] = _calcularVariabilidad(lecturasPulso.map((l) =>
l.valor).toList());
    }

    // Análisis de temperatura
    final lecturasTemp = _bufferLecturas[TipoSensor.temperatura.toString()] ?? [];
    if (lecturasTemp.isNotEmpty) {
      final promedioTemp = lecturasTemp.map((l) => l.valor).reduce((a, b) => a + b) /
lecturasTemp.length;
      indicadores['temperatura_promedio'] = promedioTemp;
    }

    // Análisis de conductancia galvánica (indicador de estrés)
    final lecturasGSR = _bufferLecturas[TipoSensor.conductanciaGalvanica.toString()] ?? [];
    if (lecturasGSR.isNotEmpty) {
      final promedioGSR = lecturasGSR.map((l) => l.valor).reduce((a, b) => a + b) /
lecturasGSR.length;
      indicadores['conductancia_promedio'] = promedioGSR;
    }

    return _interpretarIndicadores(indicadores);
  }

  double _calcularVariabilidad(List<double> valores) {
    if (valores.length < 2) return 0.0;

    final promedio = valores.reduce((a, b) => a + b) / valores.length;
    final varianza = valores.map((v) => pow(v - promedio, 2)).reduce((a, b) => a + b) /
valores.length;
    return sqrt(varianza);
```

```
}

EstadoEmocional _interpretarIndicadores(Map<String, double> indicadores) {
  EmotionType emocionPrimaria = EmotionType.neutral;
  double valencia = 0.0;
  double activacion = 0.5;
  int intensidad = 5;
  String interpretacion = "Estado emocional estable";

  final pulso = indicadores['pulso_promedio'] ?? 70.0;
  final variabilidadPulso = indicadores['variabilidad_pulso'] ?? 0.0;
  final temperatura = indicadores['temperatura_promedio'] ?? 36.5;
  final conductancia = indicadores['conductancia_promedio'] ?? 0.5;

  // Lógica de interpretación basada en patrones psicofisiológicos
  if (pulso > 90 && variabilidadPulso > 5 && conductancia > 0.7) {
    emocionPrimaria = EmotionType.ansiedad;
    valencia = -0.3;
    activacion = 0.8;
    intensidad = 7;
    interpretacion = "Detectados indicadores de ansiedad: pulso elevado y alta conductancia";
  } else if (pulso > 100 && temperatura > 37.0) {
    emocionPrimaria = EmotionType.estres;
    valencia = -0.5;
    activacion = 0.9;
    intensidad = 8;
    interpretacion = "Signos de estrés agudo: pulso muy elevado y temperatura alta";
  } else if (pulso > 85 && conductancia > 0.6) {
    emocionPrimaria = EmotionType.ira;
    valencia = -0.4;
    activacion = 0.7;
    intensidad = 6;
    interpretacion = "Patrón compatible con irritación o ira";
  } else if (pulso < 65 && variabilidadPulso < 2) {
    emocionPrimaria = EmotionType.relajacion;
    valencia = 0.4;
    activacion = 0.2;
    intensidad = 3;
    interpretacion = "Estado de relajación detectado";
  } else if (pulso > 80 && temperatura > 36.8 && conductancia < 0.4) {
    emocionPrimaria = EmotionType.alegria;
    valencia = 0.6;
    activacion = 0.6;
    intensidad = 6;
```

```
      interpretacion = "Indicadores positivos: posible estado de alegría";
    }

    return EstadoEmocional(
      id: DateTime.now().millisecondsSinceEpoch.toString(),
      timestamp: DateTime.now(),
      emocionPrimaria: emocionPrimaria,
      intensidad: intensidad,
      valencia: valencia,
      activacion: activacion,
      indicadoresFisiologicos: indicadores,
      interpretacion: interpretacion,
    );
  }

  EstadoEmocional _crearEstadoNeutral() {
    return EstadoEmocional(
      id: DateTime.now().millisecondsSinceEpoch.toString(),
      timestamp: DateTime.now(),
      emocionPrimaria: EmotionType.neutral,
      intensidad: 5,
      valencia: 0.0,
      activacion: 0.5,
      interpretacion: "Sin datos suficientes para análisis",
    );
  }
}

//
==========================================================================
====
// CONTROLADOR PRINCIPAL DEL SISTEMA
//
==========================================================================
====

class ControladorSistemaAnillo {
  final AnilloDetectorSimulador _anillo = AnilloDetectorSimulador();
  final MotorAnalisisEmocional _motorAnalisis = MotorAnalisisEmocional();

  SesionEmocional? _sesionActual;
  final List<LecturaAnillo> _lecturasBuffer = [];
  Timer? _timerAnalisis;
```

```
  final StreamController<EstadoEmocional> _streamEstados = StreamController.broadcast();
  final StreamController<SesionEmocional> _streamSesiones = StreamController.broadcast();

  Stream<EstadoEmocional> get streamEstadosEmocionales => _streamEstados.stream;
  Stream<SesionEmocional> get streamSesiones => _streamSesiones.stream;
  bool get anilloConectado => _anillo.conectado;

  /// Inicializa el sistema completo
  Future<bool> inicializarSistema() async {
    print("🚀 Inicializando Sistema de Anillo Detector...");

    final conectado = await _anillo.conectarAnillo();
    if (!conectado) {
      print("❌ No se pudo conectar al anillo");
      return false;
    }

    // Suscribirse al stream de lecturas del anillo
    _anillo.streamLecturas.listen(_procesarLecturaAnillo);

    // Iniciar análisis periódico
    _timerAnalisis = Timer.periodic(Duration(seconds: 3), (timer) {
      _analizarEstadoActual();
    });

    print("✅ Sistema inicializado correctamente");
    return true;
  }

  void _procesarLecturaAnillo(LecturaAnillo lectura) {
    _lecturasBuffer.add(lectura);

    // Mantener buffer manejable
    if (_lecturasBuffer.length > 50) {
      _lecturasBuffer.removeAt(0);
    }

    print("📊 Lectura: ${lectura.sensor.toString().split('.').last} = ${lectura.valor.toStringAsFixed(2)}
${lectura.unidadMedida} (${lectura.zonaDetectada.toString().split('.').last})");
  }

  void _analizarEstadoActual() {
    if (_lecturasBuffer.isEmpty) return;
```

```
    final estadoEmocional = _motorAnalisis.analizarEstadoEmocional(_lecturasBuffer);
    _streamEstados.add(estadoEmocional);

    // Actualizar sesión actual
    if (_sesionActual != null) {
      _sesionActual = SesionEmocional(
        id: _sesionActual!.id,
        timestamp: _sesionActual!.timestamp,
        usuarioId: _sesionActual!.usuarioId,
        lecturas: List.from(_sesionActual!.lecturas)..addAll(_lecturasBuffer),
        estadoInicial: _sesionActual!.estadoInicial,
        estadoFinal: estadoEmocional,
        contextoAdicional: _sesionActual!.contextoAdicional,
        duracionMinutos: DateTime.now().difference(_sesionActual!.timestamp).inMinutes,
        sesionCompleta: false,
      );
    }

    print("🧠 Estado: ${estadoEmocional.emocionPrimaria.toString().split('.').last}
(${estadoEmocional.intensidad}/10) - ${estadoEmocional.interpretacion}");
  }

  /// Inicia una nueva sesión de monitoreo
  void iniciarSesion(String usuarioId, {Map<String, dynamic>? contexto}) {
    if (_lecturasBuffer.isEmpty) {
      print("⚠️ No hay lecturas disponibles para iniciar sesión");
      return;
    }

    final estadoInicial = _motorAnalisis.analizarEstadoEmocional(_lecturasBuffer);

    _sesionActual = SesionEmocional(
      id: DateTime.now().millisecondsSinceEpoch.toString(),
      timestamp: DateTime.now(),
      usuarioId: usuarioId,
      lecturas: List.from(_lecturasBuffer),
      estadoInicial: estadoInicial,
      contextoAdicional: contexto ?? {},
      duracionMinutos: 0,
    );

    print("🎯 Nueva sesión iniciada para usuario: $usuarioId");
  }
```

```
/// Finaliza la sesión actual
SesionEmocional? finalizarSesion() {
  if (_sesionActual == null) {
    print("⚠️ No hay sesión activa para finalizar");
    return null;
  }

  final estadoFinal = _motorAnalisis.analizarEstadoEmocional(_lecturasBuffer);

  final sesionFinalizada = SesionEmocional(
    id: _sesionActual!.id,
    timestamp: _sesionActual!.timestamp,
    usuarioId: _sesionActual!.usuarioId,
    lecturas: _sesionActual!.lecturas,
    estadoInicial: _sesionActual!.estadoInicial,
    estadoFinal: estadoFinal,
    contextoAdicional: _sesionActual!.contextoAdicional,
    duracionMinutos: DateTime.now().difference(_sesionActual!.timestamp).inMinutes,
    sesionCompleta: true,
  );

  _streamSesiones.add(sesionFinalizada);
  print("✅ Sesión finalizada. Duración: ${sesionFinalizada.duracionMinutos} minutos");

  _sesionActual = null;
  return sesionFinalizada;
}

/// Obtiene estadísticas de la sesión actual
Map<String, dynamic> obtenerEstadisticasSesion() {
  if (_sesionActual == null) return {};

  final lecturas = _sesionActual!.lecturas;
  final duracion = DateTime.now().difference(_sesionActual!.timestamp).inMinutes;

  return {
    'duracion_minutos': duracion,
    'total_lecturas': lecturas.length,
    'sensores_activos': lecturas.map((l) => l.sensor).toSet().length,
    'zonas_detectadas': lecturas.map((l) => l.zonaDetectada).toSet().length,
    'confianza_promedio': lecturas.isNotEmpty
        ? lecturas.map((l) => l.nivelConfianza).reduce((a, b) => a + b) / lecturas.length
        : 0,
    'estado_inicial': _sesionActual!.estadoInicial.emocionPrimaria.toString().split('.').last,
```

```
      'estado_actual': _sesionActual!.estadoFinal?.emocionPrimaria.toString().split('.').last ??
'N/A',
    };
  }

  void dispose() {
    _timerAnalisis?.cancel();
    _anillo.dispose();
    _streamEstados.close();
    _streamSesiones.close();
  }
}

//
================================================================================
====
// EJEMPLO DE USO DEL SISTEMA
//
================================================================================
====

void main() async {

print("═══════════════════════════════════════════════════════════════════════
═══════");
  print("🎯 PROTOTIPO: SISTEMA DE ANILLO DETECTOR DE SÍNTOMAS");

print("═══════════════════════════════════════════════════════════════════════
═══════");

  final controlador = ControladorSistemaAnillo();

  // Suscribirse a los streams para monitoreo
  controlador.streamEstadosEmocionales.listen((estado) {
    print("🧠 ESTADO EMOCIONAL:");
    print("   Emoción: ${estado.emocionPrimaria.toString().split('.').last}");
    print("   Intensidad: ${estado.intensidad}/10");
    print("   Valencia: ${estado.valencia.toStringAsFixed(2)}");
    print("   Activación: ${estado.activacion.toStringAsFixed(2)}");
    print("   Interpretación: ${estado.interpretacion}");
    print("
──────────────────────────────────────────────────────────────────────────────");
  });
```

```
controlador.streamSesiones.listen((sesion) {
  print("📋 SESIÓN COMPLETADA:");
  print("   Usuario: ${sesion.usuarioId}");
  print("   Duración: ${sesion.duracionMinutos} minutos");
  print("   Lecturas: ${sesion.lecturas.length}");
  print("   Estado inicial: ${sesion.estadoInicial.emocionPrimaria.toString().split('.').last}");
  print("   Estado final: ${sesion.estadoFinal?.emocionPrimaria.toString().split('.').last ?? 'N/A'}");
  print("
═══════════════════════════════════════════════════════════════════════════════");
});

// Inicializar sistema
final inicializado = await controlador.inicializarSistema();
if (!inicializado) {
  print("❌ Error al inicializar el sistema");
  return;
}

// Simular uso del sistema
await Future.delayed(Duration(seconds: 2));
print("\n🎯 Iniciando sesión de demostración...");
controlador.iniciarSesion("usuario_demo_001", contexto: {
  'actividad': 'sesion_demostracion',
  'ambiente': 'laboratorio',
  'hora_dia': 'tarde',
});

// Dejar correr el sistema por un tiempo
await Future.delayed(Duration(seconds: 15));

print("\n📊 Estadísticas de sesión actual:");
final stats = controlador.obtenerEstadisticasSesion();
stats.forEach((key, value) {
  print("   $key: $value");
});

// Finalizar sesión
await Future.delayed(Duration(seconds: 5));
print("\n🏁 Finalizando sesión...");
final sesionFinalizada = controlador.finalizarSesion();

if (sesionFinalizada != null) {
  print("\n💾 Datos de la sesión para almacenamiento:");
  print(jsonEncode(sesionFinalizada.toJson()));
```

```
  }

  // Cleanup
  await Future.delayed(Duration(seconds: 2));
  controlador.dispose();
  print("\n✅ Sistema finalizado correctamente");
}
```