

Universidad Autónoma de Sinaloa

Facultad de Ingeniería Mochis



Inteligencia Artificial.

Agente Inteligente para Optimización de Salud Ocupacional en Entornos de Oficina.

Información de los Estudiantes:

- Nombres:
 - Amaral Espinoza Ernesto.
 - Amezcuá Solano Eric Eduardo.
 - Félix Sarmiento Juan Leonel.
 - Galaviz López Jesús Daniel.
 - Rojo Torres José Ángel.
- **Grupo:** 501
- **Semestre:** IX

Información de la Entrega:

- **Profesor:** M.I.A Rocío Jacqueline Becerra Urquidez
 - **Fecha de entrega:** 17/09/2025
-

Contenido

Introducción:.....	3
Descripción del proyecto.....	4
Especificación REAS.....	6
REAS – AGENTE DE SALUD OCUPACIONAL.....	6
Especificación del entorno de trabajo.....	7
Propiedades del entorno:	8
Definición de la estructura del agente.....	9
Esquema de la arquitectura	9
Especificación de los componentes del problema	10
Especificación de recursos.....	11
Hardware:	11
Software.....	16
.....	16
Recursos humanos	19
Desarrollo.....	20
Base de conocimiento.....	20
Hechos Estáticos.....	20
Hechos dinámicos.....	22
Alimentación desde Base de datos MySQL (Esquema de la base de datos).	23
Comunicación con los sensores (Conexión Python-arduino).....	28
Comunicación Python-Prolog (Integración con prolog).....	30
Control de sensores y actuadores (Firmware de Arduino).	34
Integración con la librería OpenCV.....	36
Integración con la librería pyttsx3.....	40
Api del backend en node JS.....	42
Dashboard en react.....	44

Introducción:

El presente documento detalla los puntos de la documentación del diseño e implementación de un agente inteligente personal para optimización de salud ocupacional en estaciones de trabajo individuales.

En orden de presentación se incluye la descripción detallada del proyecto y la problemática de salud ocupacional que se busca resolver en puestos de trabajo de escritorio.

La especificación REAS (Rendimiento, Entorno, Actuadores, Sensores) que define las características fundamentales del agente inteligente. Las propiedades del entorno de trabajo donde operará el sistema y la justificación de la arquitectura seleccionada. La definición de la estructura del agente con sus componentes perceptuales, cognitivos, reactivos y deliberativos.

La especificación detallada de los componentes del problema incluyendo sensores, actuadores y base de conocimiento en Prolog.

La integración con librerías de terceros para computer vision y reconocimiento de voz. Finalmente, se presenta la especificación completa de recursos humanos, de hardware, de software y presupuesto necesarios para la implementación del prototipo.

Descripción del proyecto.

1. Problemática

En la actualidad, los trabajadores de oficina pasan en promedio entre 8 y 10 horas diarias frente a una computadora, lo que ha generado un incremento significativo en problemas de salud ocupacional. Estudios recientes demuestran que el 70% de los trabajadores de escritorio experimentan fatiga visual, el 60% sufre dolores posturales, y más del 50% reporta niveles elevados de estrés laboral.

Los espacios de trabajo cerrados presentan desafíos adicionales relacionados con la calidad del aire. La acumulación de CO₂ en oficinas con ventilación deficiente puede alcanzar niveles superiores a 1000 ppm, lo que provoca somnolencia, dificultad de concentración y disminución del rendimiento cognitivo. Asimismo, los niveles de ruido ambiental por encima de 65 dB afectan negativamente la concentración y generan estrés auditivo crónico.

La mayoría de los trabajadores no son conscientes de estos riesgos hasta que se manifiestan como problemas crónicos de salud. Los síntomas iniciales como fatiga visual leve, tensión muscular o pérdida de concentración suelen ignorarse, permitiendo que evolucionen hacia condiciones más graves como síndrome de visión computacional, trastornos musculoesqueléticos y agotamiento laboral.

Adicionalmente, los trabajadores tienden a no tomar pausas regulares, permaneciendo sentados durante períodos prolongados que superan las dos horas sin descanso. Esta conducta sedentaria, combinada con malas posturas y ambientes laborales no optimizados, contribuye al deterioro progresivo de la salud ocupacional.

2. Justificación

La prevención de problemas de salud ocupacional no solo beneficia al trabajador individual, sino que también impacta positivamente en la productividad organizacional. Según la Organización Internacional del Trabajo (OIT), las empresas que implementan programas de salud ocupacional reducen el ausentismo laboral hasta en un 25% y aumentan la productividad hasta en un 15%.

Sin embargo, los sistemas tradicionales de monitoreo de salud ocupacional presentan limitaciones significativas:

- Falta de personalización: Los estándares generales no consideran las necesidades individuales de cada trabajador
- Intervención reactiva: Los problemas se detectan después de que ya han causado daño

- Costos elevados: Implementar sistemas de monitoreo continuo para cada estación de trabajo resulta económicamente inviable para muchas organizaciones
- Supervisión manual: Requiere personal especializado para evaluar condiciones ambientales y ergonómicas

Es necesario desarrollar una solución tecnológica que permita el monitoreo continuo, personalizado y automatizado de las condiciones de salud ocupacional a nivel individual, utilizando sensores de bajo costo e inteligencia artificial para detectar problemas de forma temprana y ejecutar acciones correctivas inmediatas.

La implementación de un agente inteligente basado en conocimiento experto codificado en Prolog permite combinar el razonamiento lógico sobre estándares de salud ocupacional con la capacidad de aprender y adaptarse a patrones individuales de cada trabajador. Esta aproximación híbrida ofrece ventajas sobre sistemas puramente reactivos o basados únicamente en machine learning.

Este proyecto desarrolla un agente inteligente personal diseñado para monitorear, evaluar y optimizar las condiciones de salud ocupacional en un puesto de trabajo individual mediante el uso de sensores de escritorio, actuadores USB e inteligencia artificial. El sistema está pensado para funcionar en un cubículo o estación de trabajo personal, monitoreando al trabajador durante su jornada laboral.

El agente opera conectado a una computadora personal, recolectando datos ambientales y biométricos del trabajador, procesándolos mediante una base de conocimiento implementada en Prolog, y ejecutando acciones correctivas automáticas para mantener condiciones óptimas de trabajo. La solución aborda problemas específicos del trabajo de escritorio como fatiga visual, mala postura, calidad del aire deficiente y falta de pausas regulares.

Especificación REAS.

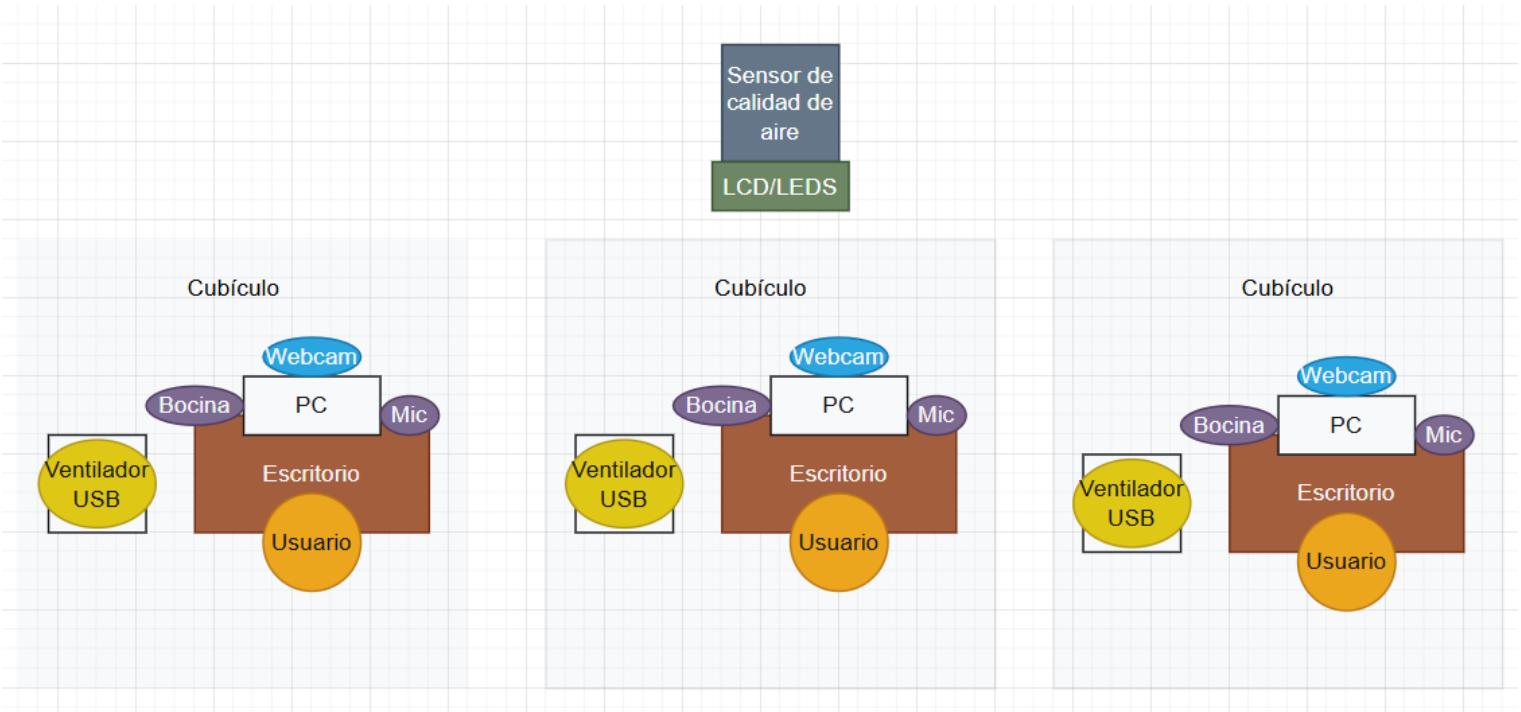
REAS – AGENTE DE SALUD OCUPACIONAL

Tipo de agente	Medidas de rendimiento	Entorno	Actuadores	Sensores
Agente de Salud Ocupacional para Oficinas	<ul style="list-style-type: none">Mantener calidad del aire óptimaDetectar fatiga/estrés en empleadosReducir niveles de ruidoTiempo de respuesta a emergencias menores a 30 segundosMejorar bienestar general del personal	<ul style="list-style-type: none">Cubículo individualEstación de trabajo personalEscritorio de computadoraEspacio de 2x2 metros aproximadamenteAmbiente de oficina para una persona <p>Trabajadores</p>	<ul style="list-style-type: none">Ventilador USB personal controlableDisplay LCD 16x2 con LEDs de estadoAltavoz de escritorio para alertas de vozRelé para control de dispositivos USB	<ul style="list-style-type: none">Sensor de calidad del aireCámara web Logitech C920 para monitoreo facialSensor acústico I2S para ruido ambienteMicrófono integrado en cámara

Especificación del entorno de trabajo.

Se llevará a cabo en un entorno real (Oficina piloto/espacio de trabajo personal). El cuál tendrá las siguientes características:

- Espacio físico: Escritorio personal de 1.5m x 0.8m
- Dispositivos: Arduino conectado vía USB a computadora personal, junto con los dispositivos sensores y actuadores.



Propiedades del entorno:

Observable:

- Parcialmente observable mediante sensores distribuidos
- Datos biométricos personales opcionales

Estocástico:

- Estocástico debido a factores externos impredecibles
- Comportamiento humano no determinístico
- Condiciones climáticas variables

Secuencial:

- Secuencial: decisiones actuales afectan estados futuros
- Memoria de patrones históricos para optimización

Dinámico:

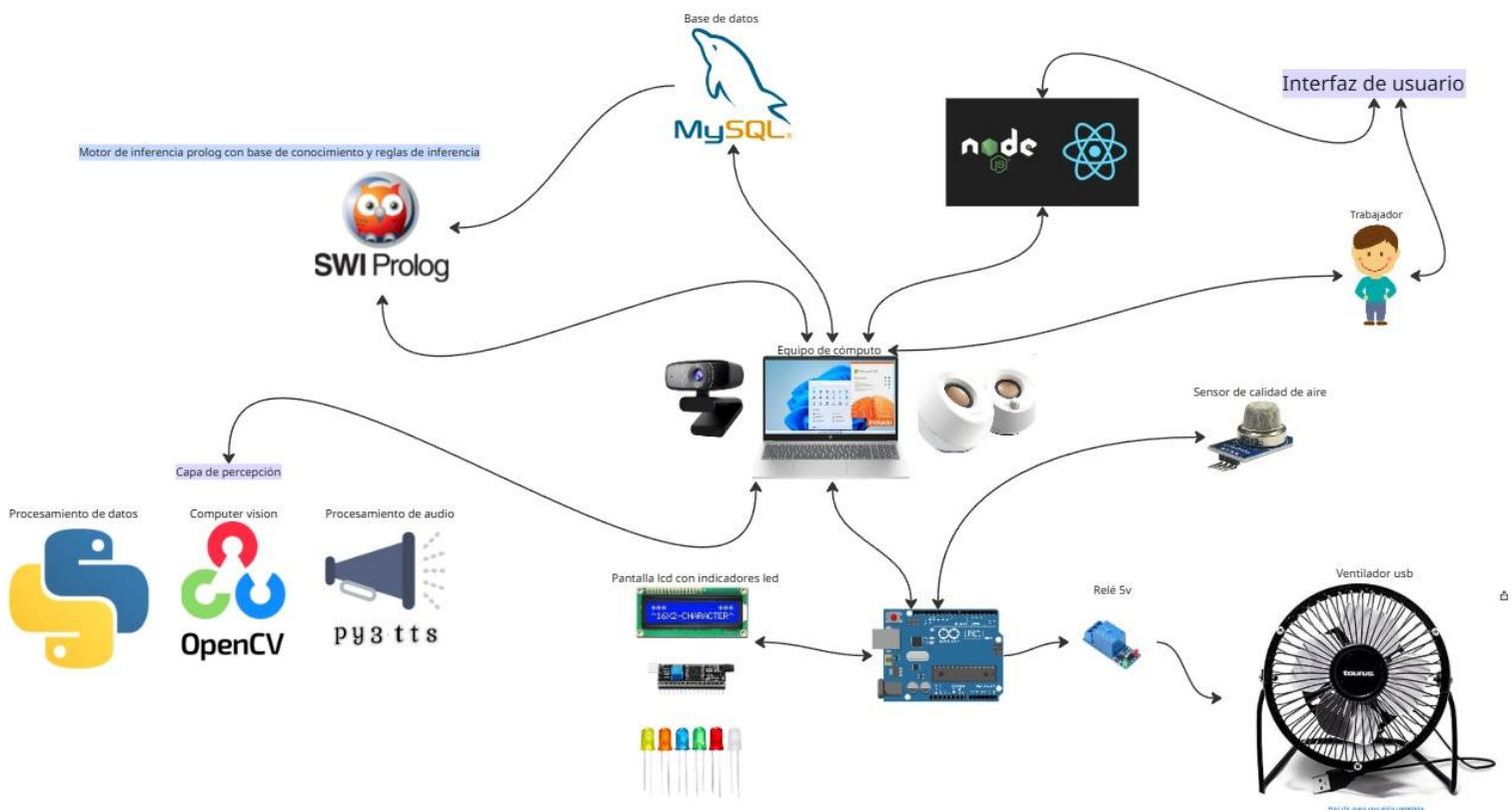
- Dinámico: condiciones cambian constantemente
- Requiere monitoreo y adaptación en tiempo real

Discreto y Continuo:

- Sensores producen datos continuos, decisiones discretas
- Muestreo cada 30 segundos para balance eficiencia/precisión

Definición de la estructura del agente.

Esquema de la arquitectura



Especificación de los componentes del problema

Estados: Todas las configuraciones posibles del ambiente laboral considerando niveles de calidad del aire (CO₂, VOCs), nivel de ruido ambiental, estado de fatiga del trabajador (bajo, medio, alto), postura del trabajador, y tiempo transcurrido de la sesión de trabajo actual y estado de los actuadores.

Estado inicial: Calidad del aire óptima (CO₂ < 800 ppm), ruido bajo (< 50 dB), fatiga nula, postura correcta, sesión de trabajo iniciando (0 minutos transcurridos y actuadores apagados).

Función sucesor: Genera estados legales mediante:

Cambios en lecturas de sensores ambientales (CO₂, ruido).

Detección de cambios en el estado de fatiga mediante visión artificial.

Activación/desactivación de actuadores (ventilador, alertas) según reglas de prolog.

Incremento del tiempo de sesión de trabajo.

Registro de pausas y ejercicios realizados.

Test objetivo: Comprueba si las condiciones ambientales están dentro de rangos óptimos (CO₂ < 1000 ppm, ruido < 65 dB), el nivel de fatiga es bajo o medio, y el trabajador ha tomado pausas regulares cada 50-60 minutos.

Costo del camino: Cada acción correctiva (activar ventilador, emitir alerta, sugerir pausa) tiene costo 1. El objetivo es mantener condiciones óptimas con el mínimo de intervenciones, priorizando la salud del trabajador.

Especificación de recursos.

Hardware:

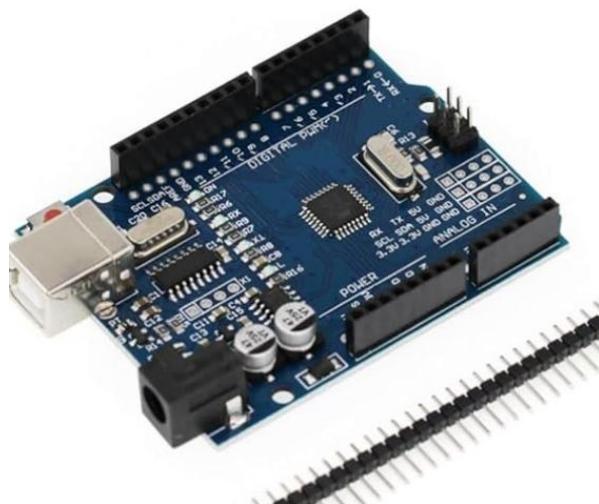


Sensor MQ135 DE Calidad DE Aire.

Cantidad: 1

Precio: \$135

Monitorear continuamente la concentración de gases nocivos en el área inmediata del usuario, específicamente CO₂, amoniaco, alcohol, humo y otros compuestos orgánicos volátiles que se acumulan en espacios cerrados durante jornadas laborales prolongadas.



Microcontrolador (Arduino)

Cantidad: 1

Precio: \$150

Microcontrolador central que actúa como intermediario entre todos los sensores, actuadores y la computadora principal. Gestiona la lectura de datos analógicos y digitales de los sensores, controla los actuadores mediante señales digitales, y establece comunicación serial con la computadora para envío de datos y recepción de comandos de control.



Equipo de cómputo

Cantidad: 1

Precio: Variable

Equipo de cómputo principal donde se ejecuta toda la lógica del agente inteligente, incluyendo el procesamiento de computer vision, motor de inferencia Prolog, base de datos MySQL, y aplicación web React. Actúa como cerebro central del sistema mientras el usuario realiza sus actividades laborales normales.



Webcam con micrófono

Cantidad: 1

Precio: \$250 (Se usará la del equipo de cómputo)

Realizar análisis visual no invasivo del estado físico y emocional del usuario mediante técnicas de computer vision, detectando indicadores tempranos de fatiga, estrés y mala postura que el propio trabajador no percibe conscientemente y monitorear niveles de ruido ambiental en el puesto de trabajo y capturar comandos de voz básicos del usuario para interacción natural con el sistema sin interrumpir el flujo de trabajo.

↳ Altavoces



Cantidad: 1

Precio \$300 (Se usarán los del equipo de cómputo)

Proporcionar retroalimentación auditiva del sistema mediante alertas de voz, confirmaciones de comandos, y guía de ejercicios de relajación o pausas activas, utilizando síntesis de voz para comunicación natural.

↳ Ventilador USB



Cantidad: 1

Precio: \$150

Mejorar la circulación de aire personal en el puesto de trabajo para reducir la concentración de CO₂ y otros gases, además de proporcionar sensación de frescura que combate la somnolencia y mejora el estado de alerta.

[Haz clic para una vista completa](#)



PANTALLA LCD 16x12

Cantidad: 1

Precio: \$78

Proporcionar información visual inmediata y no intrusiva sobre el estado actual del ambiente de trabajo y las recomendaciones del sistema, sin requerir que el usuario desvíe la atención de sus tareas principales.



Diodos LED

Cantidad: 3

Precio: \$10

Proporcionar información visual inmediata y no intrusiva sobre el estado actual del ambiente de trabajo y las recomendaciones del sistema, sin requerir que el usuario desvíe la atención de sus tareas principales.

Módulo relevador

Cantidad: 1

Precio: \$80



Actúa como interfaz de conmutación entre las señales digitales de bajo voltaje del Arduino (5V) y los actuadores de mayor potencia como el ventilador USB, permitiendo control automático de dispositivos eléctricos de forma segura y aislada.

Software



VS Code

Entorno de desarrollo integrado (IDE) principal utilizado para escribir, debuggear y mantener todo el código del proyecto. Sirve como plataforma central para desarrollo tanto del firmware de Arduino como de las aplicaciones Python y JavaScript del sistema.

Precio: \$0 (Código abierto)



SWI Prolog

Motor de inferencia lógica que implementa la base de conocimiento del agente. Procesa reglas de salud ocupacional, realiza razonamiento deductivo sobre condiciones del usuario, y genera recomendaciones basadas en conocimiento experto codificado.

Precio: \$0 (Código abierto)



MySQL

SGBD para alimentar la base de conocimiento

Precio: \$0 (Código abierto)

Sistema de gestión de base de datos relacional que almacena de forma persistente todos los datos históricos del usuario, lecturas de sensores, decisiones del agente, y métricas de efectividad. Proporciona integridad referencial y capacidades de consulta compleja para análisis de tendencias.



OpenCV

Librería de computer vision que procesa el video de la cámara web en tiempo real. Implementa algoritmos de detección facial, análisis de expresiones, y medición de indicadores de fatiga como frecuencia de parpadeo y postura de cabeza.

Precio :\$0 (Código abierto)



PYTTSX3

Motor de síntesis de voz que convierte texto en audio hablado usando voces sintéticas locales del sistema. Genera alertas auditivas, confirmaciones de comandos, y guías de ejercicios sin requerir conexión a internet.

Precio: \$0 (Código abierto)

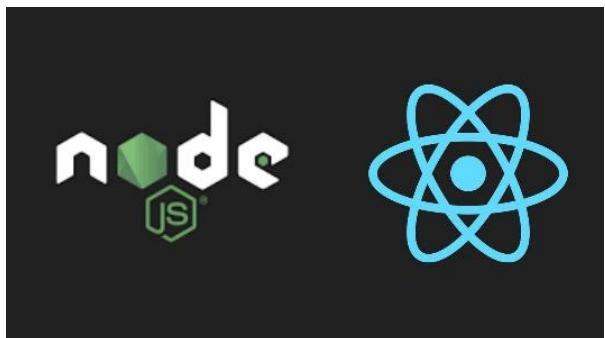


Python

Precio: \$0 (Código abierto)

Lenguaje de programación central que orquesta toda la lógica del agente. Gestiona la lectura de sensores, procesamiento de datos, comunicación con base de datos, control de actuadores, y coordinación entre todos los subsistemas.

Node js + React



Framework de desarrollo web utilizado para crear una interfaz gráfica opcional que permite al usuario visualizar en tiempo real el estado de todos los sensores, historial de recomendaciones, tendencias de salud ocupacional, y configuración personalizada del sistema.

Precio \$0 (Código abierto)

Recursos humanos

5 estudiantes del noveno semestre de la carrera de Ingeniería de Software que presentarán este proyecto para la materia de Inteligencia Artificial.

Desarrollo

Base de conocimiento.

Se guardará la información y configuración personalizada de los usuarios, lecturas de los sensores, registros de las sesiones de trabajo, recomendaciones hechas al usuario. Tendrá hechos estáticos como estándares de salud ocupacional correspondientes a temperatura, nivel de ruido, calidad del aire, etc. Indicadores de fatiga por análisis facial, tiempos recomendados de trabajo, ejercicios recomendados y umbrales personalizados.

Los hechos dinámicos corresponderán a los estados de un determinado momento en los niveles de calidad en el aire, el nivel de ruido, la duración y la hora de inicio de la sesión de trabajo actual y el nivel de fatiga del usuario.

Además de la posibilidad de considerar condiciones de salud específica del trabajador.

Hechos Estáticos.

1. Estándares de calidad del aire (4 hechos)

- estandar_co2(optimo, 400, 800).
- estandar_co2(aceptable, 801, 1000).
- estandar_co2(deficiente, 1001, 1500).
- estandar_co2(malo, 1501, 5000).

2. Estándares de nivel de ruido (5 hechos)

- estandar_ruido(silencioso, 0, 40).
- estandar_ruido(tranquilo, 41, 50).
- estandar_ruido(moderado, 51, 65).
- estandar_ruido(ruidoso, 66, 85).
- estandar_ruido(muy_ruidoso, 86, 120).

3. Estándares de temperatura (5 hechos)

- estandar_temperatura(frio, 0, 18).
- estandar_temperatura(fresco, 19, 21).
- estandar_temperatura(optimo, 22, 24).
- estandar_temperatura(calido, 25, 27).
- estandar_temperatura(caluroso, 28, 40).

4. Indicadores de fatiga visual (4 hechos)

- indicador_fatiga(parpadeo_frecuente, visual, moderado).
- indicador_fatiga(ojos_entrecerrados, visual, alto).
- indicador_fatiga(frotarse_ojos, visual, alto).
- indicador_fatiga(mirada_perdida, cognitiva, moderado).

5. Indicadores de fatiga postural (4 hechos)

- indicador_fatiga(cabeza_inclinada, postural, moderado).
- indicador_fatiga(hombros_caidos, postural, alto).
- indicador_fatiga(espalda_encorvada, postural, alto).
- indicador_fatiga(cuello_extendido, postural, moderado).

6. Tiempos recomendados minutos (4 hechos)

- tiempo_trabajo_continuo(50).
- tiempo_pausa_corta(5).
- tiempo_pausa_larga(15).
- intervalo_ejercicios(30).

7. Ejercicios recomendados (7 hechos)

- ejercicio(visual, '20-20-20: cada 20 min, mirar 20 seg a 20 pies', 1).
- ejercicio(postural, 'Estiramiento de cuello: 10 repeticiones', 2).
- ejercicio(postural, 'Rotacion de hombros: 15 repeticiones', 2).
- ejercicio(postural, 'Estiramiento de espalda: mantener 30 seg', 3).
- ejercicio(cognitiva, 'Respiracion profunda: 5 repeticiones', 2).
- ejercicio(general, 'Caminar 5 minutos', 5).
- ejercicio(general, 'Estiramiento completo de brazos', 1).

8. Umbrales personalizables (5 hechos)

- umbral_default(co2_critico, 1200).
- umbral_default(ruido_critico, 70).
- umbral_default(tiempo_max_sentado, 60).
- umbral_default(frecuencia_parpadeo_normal, 15).
- umbral_default(frecuencia_parpadeo_fatiga, 25).

9. Acciones correctivas (6 hechos)

- accion_correctiva(co2_alto, activar_ventilador, automatica).
- accion_correctiva(co2_alto, sugerir_ventilar, manual).
- accion_correctiva(ruido_alto, alerta_ruido, manual).
- accion_correctiva(fatiga_visual, ejercicio_visual, manual).

- accion_correctiva(fatiga_postural, ejercicio_postural, manual).
- accion_correctiva(tiempo_prolongado, sugerir_pausa, manual).

10. Prioridades de alertas (5 hechos)

- prioridad_alerta(co2_critico, 1).
- prioridad_alerta(fatiga_alta, 1).
- prioridad_alerta(ruido_excesivo, 2).
- prioridad_alerta(tiempo_trabajo_largo, 2).
- prioridad_alerta(postura_inadecuada, 3).

11. Mensajes de alerta (5+ hechos)

- mensaje_alerta(co2_alto, 'Nivel de CO2 elevado. Activando ventilador.').
- mensaje_alerta(ruido_alto, 'Nivel de ruido elevado. Considera usar audifonos.').
- mensaje_alerta(fatiga_detectada, 'Signos de fatiga detectados. Toma un descanso.').
- mensaje_alerta(pausa_recomendada, 'Llevas mucho tiempo trabajando. Pausa recomendada.').
- mensaje_alerta(ejercicio_sugerido, 'Es momento de hacer ejercicios de estiramiento.').

Hechos dinámicos.

Estos se actualizarán desde Python/MySQL

- :- dynamic(estado_actual/2).
- :- dynamic(sesion_trabajo/3).
- :- dynamic(lectura_sensor/3).
- :- dynamic(nivel_fatiga/2).
- :- dynamic(usuario_actual/1).
- :- dynamic(preferencia_usuario/3).

Alimentación desde Base de datos MySQL (Esquema de la base de datos).

```
CREATE DATABASE IF NOT EXISTS salud_ocupacional;
USE salud_ocupacional;

-- =====
-- TABLA DE USUARIOS
-- =====

CREATE TABLE usuarios (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE NOT NULL,
    edad INT,
    condiciones_salud TEXT,
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- =====
-- TABLA DE CONFIGURACIÓN DE USUARIO
-- =====

CREATE TABLE configuracion_usuario (
    id INT PRIMARY KEY AUTO_INCREMENT,
    usuario_id INT NOT NULL,
    umbral_co2_critico INT DEFAULT 1200,
    umbral_ruido_critico INT DEFAULT 70,
    tiempo_max_sentado INT DEFAULT 60,
    sensibilidad_alertas ENUM('baja', 'media', 'alta') DEFAULT 'media',
    volumen_alertas INT DEFAULT 70,
    frecuencia_recordatorios INT DEFAULT 60,
    FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE CASCADE
);
```

```
-- =====
-- TABLA DE DETECCIÓN DE FATIGA
-- =====

CREATE TABLE deteccion_fatiga (
    id INT PRIMARY KEY AUTO_INCREMENT,
    sesion_id INT NOT NULL,
    tipo_fatiga ENUM('visual', 'postural', 'cognitiva') NOT NULL,
    nivel_fatiga ENUM('bajo', 'moderado', 'alto') NOT NULL,
    indicador VARCHAR(100),
    frecuencia_parpadeo INT,
    postura_detectada VARCHAR(50),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sesion_id) REFERENCES sesiones_trabajo(id) ON DELETE CASCADE,
    INDEX idx_sesion_tipo (sesion_id, tipo_fatiga)
);

-- =====
-- TABLA DE ACCIONES DEL SISTEMA
-- =====

CREATE TABLE acciones_sistema (
    id INT PRIMARY KEY AUTO_INCREMENT,
    sesion_id INT NOT NULL,
    tipo_accion VARCHAR(50) NOT NULL,
    descripcion TEXT,
    actuador ENUM('ventilador', 'pantalla_lcd', 'altavoz', 'led') NOT NULL,
    estado_anterior VARCHAR(20),
    estado_nuevo VARCHAR(20),
    automatica BOOLEAN DEFAULT TRUE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sesion_id) REFERENCES sesiones_trabajo(id) ON DELETE CASCADE,
    INDEX idx_sesion_timestamp (sesion_id, timestamp)
);
```

```
-- =====
-- TABLA DE SESIONES DE TRABAJO
-- =====
CREATE TABLE sesiones_trabajo (
    id INT PRIMARY KEY AUTO_INCREMENT,
    usuario_id INT NOT NULL,
    fecha DATE NOT NULL,
    hora_inicio TIME NOT NULL,
    hora_fin TIME,
    minutos_totales INT,
    pausas_tomadas INT DEFAULT 0,
    estado ENUM('activa', 'pausada', 'finalizada') DEFAULT 'activa',
    FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE CASCADE
);

-- =====
-- TABLA DE LECTURAS DE SENSORES
-- =====
CREATE TABLE lecturas_sensores (
    id INT PRIMARY KEY AUTO_INCREMENT,
    sesion_id INT NOT NULL,
    tipo_sensor ENUM('co2', 'ruido', 'temperatura') NOT NULL,
    valor DECIMAL(10, 2) NOT NULL,
    unidad VARCHAR(10),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sesion_id) REFERENCES sesiones_trabajo(id) ON DELETE CASCADE,
    INDEX idx_sesion_sensor (sesion_id, tipo_sensor),
    INDEX idx_timestamp (timestamp)
);
```

```
-- =====
-- TABLA DE ALERTAS GENERADAS
-- =====

CREATE TABLE alertas_generadas (
    id INT PRIMARY KEY AUTO_INCREMENT,
    sesion_id INT NOT NULL,
    tipo_alerta VARCHAR(50) NOT NULL,
    prioridad ENUM('alta', 'media', 'baja') NOT NULL,
    mensaje TEXT NOT NULL,
    visualizada BOOLEAN DEFAULT FALSE,
    descartada BOOLEAN DEFAULT FALSE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sesion_id) REFERENCES sesiones_trabajo(id) ON DELETE CASCADE,
    INDEX idx_sesion_prioridad (sesion_id, prioridad)
);

-- =====
-- TABLA DE EJERCICIOS RECOMENDADOS
-- =====

CREATE TABLE ejercicios_recomendados (
    id INT PRIMARY KEY AUTO_INCREMENT,
    tipo_fatiga ENUM('visual', 'postural', 'cognitiva', 'general') NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    descripcion TEXT NOT NULL,
    duracion_minutos INT NOT NULL,
    dificultad ENUM('facil', 'moderado', 'dificil') DEFAULT 'facil',
    activo BOOLEAN DEFAULT TRUE
).
```

```
-- =====
-- TABLA DE HISTORIAL DE EJERCICIOS
-- =====
CREATE TABLE historial_ejercicios (
    id INT PRIMARY KEY AUTO_INCREMENT,
    sesion_id INT NOT NULL,
    ejercicio_id INT NOT NULL,
    completado BOOLEAN DEFAULT FALSE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sesion_id) REFERENCES sesiones_trabajo(id) ON DELETE CASCADE,
    FOREIGN KEY (ejercicio_id) REFERENCES ejercicios_recomendados(id) ON DELETE CASCADE
);

-- =====
-- TABLA DE ESTADÍSTICAS DIARIAS
-- =====
CREATE TABLE estadisticas_diarias (
    id INT PRIMARY KEY AUTO_INCREMENT,
    usuario_id INT NOT NULL,
    fecha DATE NOT NULL,
    total_minutos_trabajo INT DEFAULT 0,
    total_pausas INT DEFAULT 0,
    promedio_co2 DECIMAL(10, 2),
    promedio_ruido DECIMAL(10, 2),
    alertas_generadas INT DEFAULT 0,
    nivel_fatiga_promedio ENUM('bajo', 'moderado', 'alto'),
    FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE CASCADE,
    UNIQUE KEY unique_usuario_fecha (usuario_id, fecha)
);
```

Comunicación con los sensores (Conexión Python-arduino).

```
class AgenteInteligenteSaludOcupacional:
    def __init__(self, puerto_arduino='COM3', baudrate=9600):
        """
        Inicializa el agente inteligente
        """
        # Conexión a Arduino
        self.arduino = serial.Serial(puerto_arduino, baudrate, timeout=1)
        time.sleep(2) # Esperar inicialización de Arduino

        # Conexión a MySQL
        self.db = mysql.connector.connect(
            host="localhost",
            user="root",
            password="",
            database="salud_ocupacional"
        )
        self.cursor = self.db.cursor(dictionary=True)

        # Conexión a Prolog
        self.prolog = Prolog()
        self.prolog.consult("salud_ocupacional.pl")

        # Variables de estado
        self.sesion_activa = None
        self.usuario_id = 1 # Por defecto usuario demo
        self.ultima_leitura = {}
```

```
def leer_sensores_arduino(self):
    """
        Lee datos de los sensores desde Arduino
        Formato esperado: "CO2:450,RUIDO:45,TEMP:23"
    """
    if self.arduino.in_waiting > 0:
        linea = self.arduino.readline().decode('utf-8').strip()

        # Parsear datos
        datos = {}
        for par in linea.split(','):
            if ':' in par:
                sensor, valor = par.split(':')
                datos[sensor.lower()] = float(valor)

    return datos
return None
```

Comunicación Python-Prolog (Integración con prolog).

```
def actualizar_prolog_con_sensores(self):
    """
    Actualiza la base de conocimiento de Prolog con datos de sensores
    """
    timestamp = datetime.now().strftime('%H:%M:%S')

    for tipo_sensor, valor in self.ultima_lectura.items():
        # Eliminar lectura anterior
        query_retract = f"retractall(lectura_sensor({tipo_sensor}, _, _))"
        list(self.prolog.query(query_retract))

        # Insertar nueva lectura
        query_assert = f"assertz(lectura_sensor({tipo_sensor}, {valor}, '{timestamp}'))"
        list(self.prolog.query(query_assert))
```

```
def consultar_calidad_aire(self):
    """
    Consulta la calidad del aire según Prolog
    """
    resultado = list(self.prolog.query("calidad_aire(Nivel)"))
    if resultado:
        return resultado[0]['Nivel']
    return "desconocido"

def consultar_nivel_ruido(self):
    """
    Consulta el nivel de ruido según Prolog
    """
    resultado = list(self.prolog.query("nivel_ruido(Nivel)"))
    if resultado:
        return resultado[0]['Nivel']
    return "desconocido"

def verificar_condicion_critica_co2(self):
    """
    Verifica si hay condición crítica de CO2
    """
    resultado = list(self.prolog.query("condicion_critica_co2"))
    return len(resultado) > 0

def verificar_requiere_ventilacion(self):
    """
    Verifica si se requiere activar ventilación
    """
    resultado = list(self.prolog.query("requiere_ventilacion"))
    return len(resultado) > 0
```

```
def verificar_requiere_pausa(self):
    """
    Verifica si el usuario requiere tomar una pausa
    """
    resultado = list(self.prolog.query("requiere_pausa"))
    return len(resultado) > 0

def obtener_acciones_recomendadas(self):
    """
    Obtiene todas las acciones recomendadas por Prolog
    """
    resultado = list(self.prolog.query("acciones_recomendadas(Lista)"))
    if resultado:
        return resultado[0]['Lista']
    return []

def actualizar_estado_actuador(self, actuador, estado):
    """
    Actualiza el estado de un actuador en Prolog
    """
    query_retract = f'retractall(estado_actual({actuador}, _))'
    list(self.prolog.query(query_retract))

    query_assert = f'assertz(estado_actual({actuador}, {estado}))'
    list(self.prolog.query(query_assert))
```

```
def ejecutar_accion_ventilador(self, accion):
    """
    Ejecuta acción en el ventilador (ON/OFF)
    """
    if accion == "activar":
        comando = "VENTILADOR:ON"
        estado = "encendido"
    else:
        comando = "VENTILADOR:OFF"
        estado = "apagado"

    # Enviar comando a Arduino
    selfarduino.write(f"{comando}\n".encode())

    # Actualizar Prolog
    selfactualizar_estado_actuador('ventilador', estado)

    # Registrar en base de datos
    query = """
        INSERT INTO acciones_sistema
        (sesion_id, tipo_accion, descripcion, actuador, estado_nuevo, automatica)
        VALUES (%s, %s, %s, %s, %s, TRUE)
    """
    self.cursor.execute(query, (
        selfsesion_activaa,
        'control_ventilador',
        f'Ventilador {accion}do por condición crítica',
        'ventilador',
        estado
    ))
    selfdb.commit()
```

Control de sensores y actuadores (Firmware de Arduino).

```
int leerMQ135() {
    // Leer valor analógico del sensor
    int lectura = analogRead(PIN_MQ135);

    // Convertir a PPM (conversión simplificada)
    // En producción usar calibración real del sensor
    int ppm = map(lectura, 0, 1023, 400, 2000);

    return ppm;
}
```

```
int leerMicrofono() {
    // Tomar múltiples muestras para calcular promedio
    long suma = 0;
    int muestras = 50;

    for (int i = 0; i < muestras; i++) {
        int lectura = analogRead(PIN_MICROFONO);
        suma += lectura;
        delay(2);
    }

    int promedio = suma / muestras;

    // Convertir a decibeles (conversión simplificada)
    int db = map(promedio, 0, 1023, 30, 90);

    return db;
}
```

```
float leerTemperatura() {
    // Leer sensor de temperatura (LM35)
    int lectura = analogRead(PIN_TEMPERATURA);

    // Convertir a grados Celsius
    // LM35: 10mV por grado, 5V = 1023
    float temperatura = (lectura * 5.0 / 1023.0) * 100.0;

    return temperatura;
}
```

Integración con la librería OpenCV

```
class DetectorFatigaPostura:
    def __init__(self, db_config):
        """
        Inicializa el detector de fatiga y postura
        """

        # Configuración de la cámara
        self.cap = cv2.VideoCapture(0)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

        # Cargar clasificadores de OpenCV
        self.face_cascade = cv2.CascadeClassifier(
            cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
        )
        self.eye_cascade = cv2.CascadeClassifier(
            cv2.data.haarcascades + 'haarcascade_eye.xml'
        )

        # Conexión a base de datos
        self.db = mysql.connector.connect(**db_config)
        self.cursor = self.db.cursor(dictionary=True)

        # Variables de estado
        self.contador_parpadeos = 0
        self.ojos_cerrados_frames = 0
        self.umbral_parpadeo = 3 # Frames con ojos cerrados
        self.postura_actual = "desconocida"

        # Métricas de fatiga
        self.frecuencia_parpadeo_minuto = []
        self.tiempo_inicio_analisis = time.time()

        # Sesión activa
        self.sesion_id = None
```

```
def detectar_parpadeo(self, ojos):
    """
    Detecta parpadeo basado en la presencia de ojos
    """

    # Si se detectan menos de 2 ojos, están cerrados
    if len(ojos) < 2:
        self.ojos_cerrados_frames += 1
    else:
        # Si estaban cerrados y ahora abiertos, es un parpadeo
        if self.ojos_cerrados_frames >= self.umbral_parpadeo:
            self.contador_parpadeos += 1
        self.ojos_cerrados_frames = 0

    return len(ojos) < 2
```

```
def analizar_postura(self, rostro):
    """
    Analiza la postura basándose en la posición del rostro
    """
    if rostro is None:
        return "sin_deteccion"

    (x, y, w, h) = rostro

    # Calcular centro del rostro
    centro_y = y + h/2
    altura_frame = 480 # Altura del frame

    # Detectar postura según posición vertical
    if centro_y < altura_frame * 0.35:
        postura = "cabeza_alta" # Posible hiperextensión
    elif centro_y > altura_frame * 0.65:
        postura = "cabeza_baja" # Posible fatiga postural
    else:
        postura = "correcta"

    # Detectar inclinación por ratio ancho/alto
    ratio = w / h
    if ratio < 0.7:
        postura = "cabeza_inclinada"

    self.postura_actual = postura
    return postura
```

```
def detectar_rostro_ojos(self, frame):
    """
    Detecta rostro y ojos en el frame
    """

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detectar rostros
    rostros = self.face_cascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(100, 100)
    )

    if len(rostros) == 0:
        return None, None, frame

    # Tomar el primer rostro detectado
    (x, y, w, h) = rostros[0]
    cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

    # Región de interés para ojos
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = frame[y:y+h, x:x+w]

    # Detectar ojos
    ojos = self.eye_cascade.detectMultiScale(
        roi_gray,
        scaleFactor=1.1,
        minNeighbors=10,
        minSize=(20, 20)
    )

    # Dibujar ojos detectados
    for (ex, ey, ew, eh) in ojos:
        cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)

    return rostros[0], ojos, frame
```

Integración con la librería pyttsx3.

```
class AsistenteVoz:
    def __init__(self):
        """
        Inicializa el motor de síntesis de voz
        """
        # Inicializar motor de texto a voz
        self.engine = pyttsx3.init()

        # Configurar propiedades de voz
        self.configurar_voz()

        # Cola de mensajes para procesar
        self.cola_mensajes = Queue()

        # Thread para procesar mensajes
        self.procesando = False
        self.thread_voz = None

        print("✓ Asistente de voz inicializado")

    def configurar_voz(self, velocidad=150, volumen=0.9, voz_id=0):
        """
        Configura las propiedades de la voz sintetizada
        """
        # Velocidad de habla (palabras por minuto)
        self.engine.setProperty('rate', velocidad)

        # Volumen (0.0 a 1.0)
        self.engine.setProperty('volume', volumen)

        # Seleccionar voz (0 = masculina, 1 = femenina generalmente)
        voices = self.engine.getProperty('voices')
        if voz_id < len(voices):
            self.engine.setProperty('voice', voices[voz_id].id)
```

```

def alerta_co2_alto(self):
    """Alerta de CO2 elevado"""
    mensaje = "Atención. Nivel de dióxido de carbono elevado. Se ha activado el ventilador automáticamente."
    self.hablar(mensaje)

def alerta_ruido_alto(self):
    """Alerta de ruido elevado"""
    mensaje = "Nivel de ruido elevado detectado. Considera utilizar audífonos con cancelación de ruido."
    self.hablar(mensaje)

def recordatorio_pausa(self, minutos_trabajados):
    """Recordatorio de tomar pausa"""
    mensaje = f"Has trabajado {minutos_trabajados} minutos sin descanso. Es momento de tomar una pausa."
    self.hablar(mensaje)

def alerta_fatiga_detectada(self, tipo_fatiga):
    """Alerta de fatiga detectada"""
    if tipo_fatiga == "visual":
        mensaje = "Se han detectado signos de fatiga visual. Descansa la vista por unos momentos."
    elif tipo_fatiga == "postural":
        mensaje = "Tu postura no es correcta. Ajusta tu posición y realiza algunos estiramientos."
    else:
        mensaje = "Se han detectado signos de fatiga. Toma un descanso breve."
    self.hablar(mensaje)

```

```

def guiar_estiramiento_cuello(self):
    """
    Guía el estiramiento de cuello
    """

    pasos = [
        "Vamos a realizar estiramientos de cuello.",
        "Siéntate derecho con la espalda recta.",
        "Inclina lentamente tu cabeza hacia el hombro derecho.",
        "Mantén la posición. Cinco segundos.",
        "Regresa al centro.",
        "Ahora inclina hacia el hombro izquierdo.",
        "Mantén la posición. Cinco segundos.",
        "Regresa al centro.",
        "Inclina la cabeza hacia adelante, llevando el mentón al pecho.",
        "Cinco segundos.",
        "Regresa al centro.",
        "Ejercicio completado."
    ]

    for paso in pasos:
        self.hablar(paso, esperar=True)
        if "cinco segundos" in paso.lower():
            time.sleep(5)
        else:
            time.sleep(2)

```

Api del backend en node JS.

```
/**  
 * API Backend - Sistema de Salud Ocupacional  
 * Servidor REST con Node.js + Express  
 */  
  
const express = require('express');  
const mysql = require('mysql2/promise');  
const cors = require('cors');  
const { spawn } = require('child_process');  
  
const app = express();  
const PORT = 3001;  
  
// Middleware  
app.use(cors());  
app.use(express.json());  
  
// ======  
// CONFIGURACIÓN DE BASE DE DATOS  
// ======  
  
const dbConfig = {  
  host: 'localhost',  
  user: 'root',  
  password: 'tu_password',  
  database: 'salud_ocupacional'  
};  
  
let dbPool;  
  
async function inicializarDB() {  
  dbPool = mysql.createPool(dbConfig);  
  console.log('✓ Pool de conexiones MySQL creado');  
}  
}
```

```
app.get('/api/sensores/ultimas/:sesion_id', async (req, res) => {
  try {
    const { sesion_id } = req.params;

    const [lecturas] = await dbPool.query(`SELECT tipo_sensor, valor, unidad, timestamp FROM lecturas_sensores WHERE sesion_id = ? AND tipo_sensor IN ('co2', 'ruido', 'temperatura') ORDER BY tipo_sensor, timestamp DESC`, [sesion_id]);

    // Obtener la última lectura de cada tipo
    const ultimasLecturas = {};
    lecturas.forEach(lectura => {
      if (!ultimasLecturas[lectura.tipo_sensor]) {
        ultimasLecturas[lectura.tipo_sensor] = lectura;
      }
    });

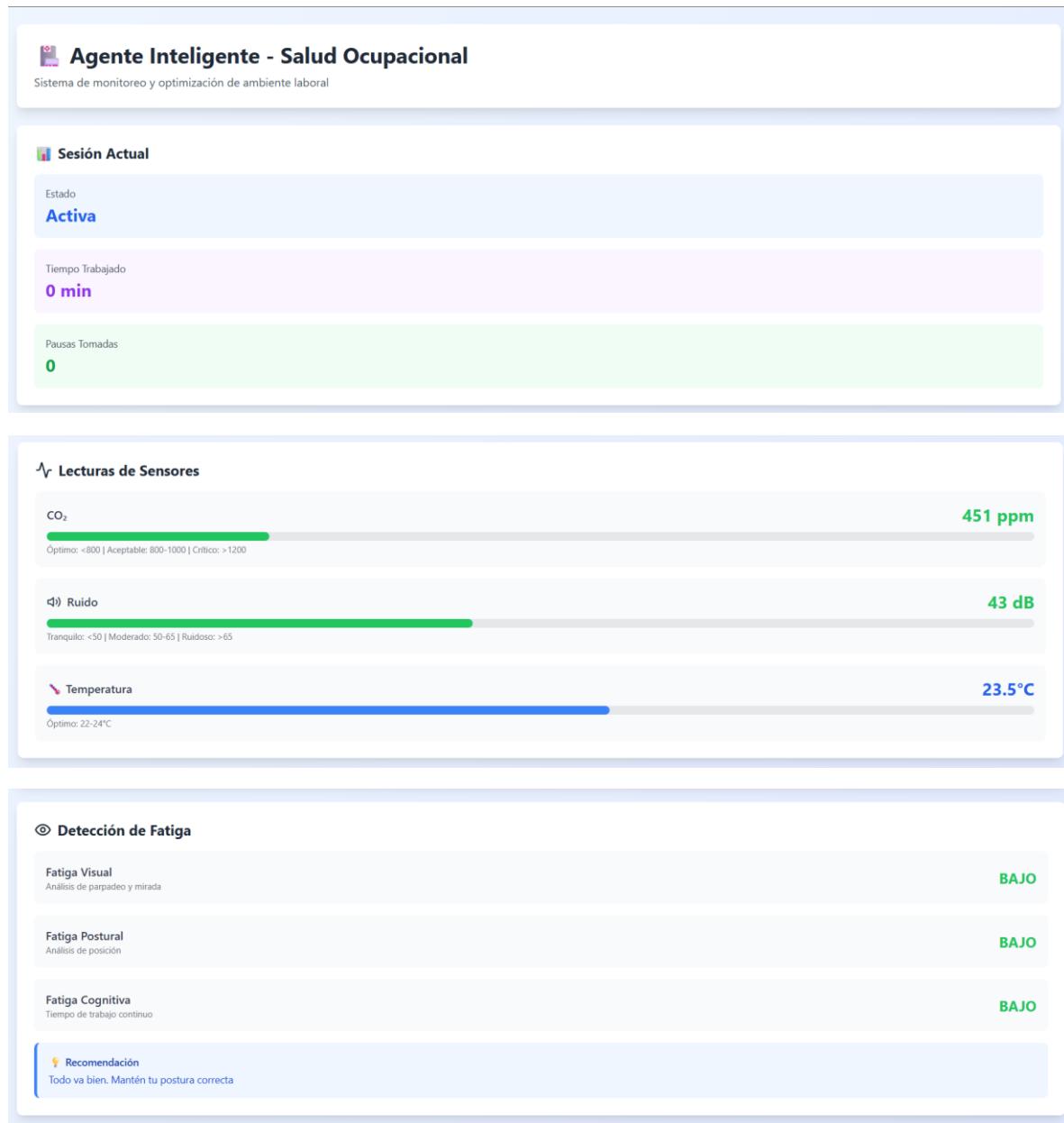
    res.json({ success: true, lecturas: ultimasLecturas });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});

// Obtener historial de sensor
app.get('/api/sensores/historial/:sesion_id/:tipo', async (req, res) => {
  try {
    const { sesion_id, tipo } = req.params;
    const { limite = 50 } = req.query;

    const [lecturas] = await dbPool.query(`SELECT valor, timestamp FROM lecturas_sensores WHERE sesion_id = ? AND tipo_sensor = ? ORDER BY timestamp DESC LIMIT ?`, [sesion_id, tipo, parseInt(limite)]);

    res.json({ success: true, historial: lecturas.reverse() });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});
```

Dashboard en react



① Alertas Activas

 No hay alertas activas. Condiciones de trabajo óptimas.

⇒ Estado de Actuadores



Ventilador
APAGADO



LED Verde
Estado OK



LED Amarillo
Precaución



LED Rojo
Alerta

