Diseño de Arquitectura del Sistema - Modelo basado en IA para detección de somnolencia

**Fecha:** 10/04/2025

**Versión:** 1.0

# 1. Introducción

Este documento describe de manera detallada la arquitectura del sistema para el modelo basado en IA, cuyo propósito es detectar la somnolencia al volante y generar una alerta temprana y precisa. Se define la estructura general del sistema, los componentes que lo conforman, sus interacciones y las tecnologías empleadas.

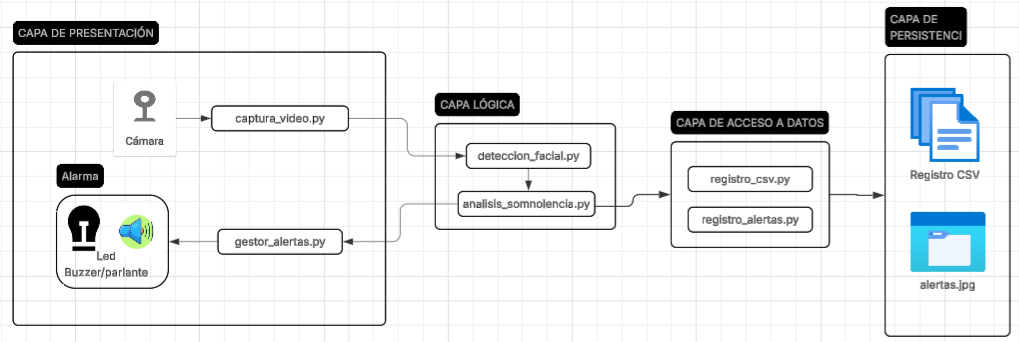
# 2. Visión General de la Arquitectura

## 2.1. Estilo Arquitectónico

El sistema adopta una arquitectura en capas monolítica, ya que no cuenta con capas que estén distribuidas y no utiliza servicios externos (todo se comunica dentro del mismo entorno). Se organiza en las capas principales:

* **Capa de Presentación:** Se encarga de la emisión de la alarma en caso que se identifique somnolencia.
* **Capa de Lógica de Negocio:** Procesamiento de IA, gestión de alertas y archivos CSV.
* **Capa de Acceso a Datos:** Gestiona los archivos que serán almacenados de manera local.
* **Capa de Persistencia de Datos:** Almacenamiento local de eventos de somnolencia, frames (datasets de entrenamiento), y el modelo .

## 2.2. Diagrama General de Arquitectura:



# 3. Descripción de Componentes Principales

## 3.1. Capa de presentación

La capa de presentación cuenta con una arquitectura monolítica, contiene componentes de HW que son la cámara y los dispositivos que conforman la alarma (led y buzzer) que serán simulados en una interfaz. Asimismo, se cuenta con los módulos encargados de la captura de video en tiempo real para pasar los datos a la capa lógica y el módulo encargado de gestionar la activación y desactivación de las alertas.

* Framework/Librería: Python (Tkinter para simulación de alarma en interfaz).
* Responsabilidades:
  + Capturar video en tiempo real de la cámara.
  + Alertas: Generación de alertas sonoras y notificaciones visuales (simulados en interfaz) para alertar al conductor si el sistema detecta signos de somnolencia.
  + Desactivar las alertas tras reacción del usuario.

### 3.1.1. Módulos lógicos

* captura\_video.py: Se encarga de capturar el video en tiempo real de la cámara para ser procesado.
* gestor\_alertas.py: Gestiona la creación y disparo de alertas, tanto sonoras como visuales (simuladas en interfaz).

## 3.2. Capa de Lógica de Negocio

La capa lógica presentará una arquitectura monolítica con los módulos encargados de la detección facial y el análisis de nivel de somnolencia según los parámetros establecidos (para activar alarma o no).

* Lenguaje y Frameworks: Python con OpenCV, TensorFlow/PyTorch.
* Responsabilidades:
  + **Detección facial**: Uso de OpenCV o MediaPipe para identificar los landmarks faciales y procesar la imagen.
  + **Modelo de IA**: Implementación de machine learning para analizar la somnolencia mediante métricas como PERCLOS, bostezos y la inclinación de la cabeza.

### 3.2.1. Módulos lógicos

* deteccion\_facial.py: Módulo encargado de la captura y procesamiento de la imagen para identificar el rostro del conductor y sus características faciales.
* analisis\_somnolencia.py: Cálculo de las métricas de somnolencia como PERCLOS (porcentaje de ojos cerrados), cantidad de bostezos por minuto y ángulo de inclinación de la cabeza.

## 3.3. Capa de Acceso a Datos

Esta capa también sigue una arquitectura monolítica y cuenta con los módulos encargados de la gestión de la capa de persistencia, es decir, el registro de los archivos CSV y frames de incidentes.

* Responsabilidades:
  + Gestión de registros: Genera archivos CSV para mantener registro del uso del modelo.
  + Gestión de alertas: Registra las frames con métricas tras la activación de la alarma.

### 3.3.1. Módulos lógicos

* registro\_csv.py: Registra los eventos y resultados del sistema en un archivo CSV para seguimiento y análisis posterior (por ejemplo, hora de la alerta, PERCLOS, etc.).
* registro\_alertas.py: Registra frames de incidentes de somnolencia para poder analizar el rendimiento del sistema.

## 3.4. Capa de Persistencia

La arquitectura de la capa de datos es una arquitectura monolítica con archivos planos, los registros de actividad serán almacenados en archivos CSV y los incidentes registrados en imágenes jpg también de manera local.

* Sistema Gestor: Archivos CSV. Archivos .jpg
* Esquema de Datos:
  + timestamp | PERCLOS | bostezos\_por\_minuto | inclinación\_cabeza | estado\_somnolencia.

# 4. Integraciones Externas (Opcionales)

# 5. Seguridad

Teniendo en cuenta que el desarrollo del sistema se realizará en un entorno controlado (sin acceso externo) y haciendo uso de datos públicos (datasets de entrenamiento libres). Las medidas de seguridad planteadas son las siguientes:

* **Acceso Local Restringido:** Los archivos (CSV, imágenes) solo pueden ser leídos/modificados por el sistema (la aplicación completa), no por usuarios u otros programas. Se configuran permisos de sistema operativo para evitar accesos no autorizados.
* **Validación de Datos:** Se verifica que los registros en CSV no contengan caracteres extraños o maliciosos (ej: scripts o comandos).

Ejemplo: Si el timestamp incluye un símbolo raro (\* o !), se rechaza.

* **Seguridad Física (Cámara):** La cámara solo transmite video en tiempo real al sistema. **No se guardan videos completos**, evitando riesgos de vigilancia no deseada.
* **Logging Seguro:** Los archivos de registro (registros.csv) guardan solo eventos técnicos (ej: "Alerta: PERCLOS alto a las 10:30"). **No** almacenan datos personales ni imágenes.

# 6. Escalabilidad y Despliegue

Posibles problemas que pueden surgir durante el desarrollo.

* **Simulación de interfaz**: Desplegado en Tkinter
* **Módulos**: Python
* **Almacenamiento:** Archivos planos
* La arquitectura monolítica por capas del sistema facilita su mantenimiento y comprensión en entornos locales.
* La separación lógica del sistema permite desacoplar progresivamente sus componentes, facilitando que funciones como la gestión de alertas o el análisis de somnolencia puedan convertirse en servicios independientes en el futuro.
* La persistencia con archivos planos puede migrar a una base de datos estructurada, permitiendo separar la capa de datos en una instancia dedicada.

# 7. Conclusiones

La arquitectura del sistema combina eficiencia, modularidad y potencial de escalabilidad, alineándose con los objetivos del proyecto. Su diseño monolítico por capas (presentación, lógica de negocio y persistencia) claramente diferenciadas. La modularidad de la capa lógica, dividida en componentes funcionales, facilita el mantenimiento y futuras actualizaciones. Además, aunque inicialmente el sistema se ejecutará de forma local, la separación lógica de sus bloques permite planificar una evolución progresiva hacia una arquitectura más escalable. Esto incluye la posibilidad de desacoplar funciones críticas como el análisis de somnolencia o la gestión de alertas, y migrar la capa de datos a una base de datos estructurada en servidores dedicados o en la nube.