

Sistema de Control de Acceso

Reconocimiento Facial y Detección de Gestos

Asignatura: LANAI

Máster Universitario de Informática Industrial y Robótica.

Autor: Álvaro Viña Pérez

Fecha: 15 de diciembre de 2025



Índice

1. Descripción General	3
1.1. Introducción	3
1.2. Objetivos del Proyecto	3
1.3. Características Principales	3
1.4. Tecnologías Utilizadas	4
1.5. Estructura del Proyecto	4
2. Desarrollo de la solución adoptada	5
2.1. Fase 0: Preparación del Entorno Virtual (venv)	5
2.2. Fase 1: Selección de Tecnologías y Arquitectura Base	6
2.3. Fase 2: Base de Datos — Diseño y Estructura	6
2.4. Fase 3: Implementación del Reconocimiento Facial	8
2.4.1. Arquitectura del Módulo	8
2.4.2. Funcionamiento del Reconocimiento Facial	8
2.4.3. Proceso de Registro de Rostros	8
2.5. Fase 4: Verificación por PIN y uso de bcrypt	9
2.6. Fase 5: Detección de Gestos	9
2.6.1. Arquitectura del Módulo	9
2.6.2. Gestos Implementados	9
2.6.3. Funcionamiento de la Detección de Gestos	10
2.6.4. Algoritmo de Detección	10
2.6.5. Integración con Autenticación	10
2.7. Fase 6: Desarrollo de la Interfaz Gráfica	10
2.7.1. Tecnología de la GUI	10
2.7.2. Estructura de la Ventana	11
2.7.3. Gestión del Ciclo de Video	11
2.7.4. Flujo de Verificación en la GUI	11
2.7.5. Concurrencia y Estado	11
3. Demostración del Flujo del Sistema	12
3.1. Paso 1: Solicitud y Validación de Gesto	12
3.2. Paso 2: Captura de Frame para Reconocimiento Facial	12
3.3. Paso 3: Verificación de PIN y Acceso	13
3.4. Paso 4: Registro de salida	13
3.5. Panel de Administración	14
3.5.1. Registro de nuevo usuario	14
4. Conclusiones	16
5. Anexo	18
5.1. Códigos principales	18
5.1.1. Código main	18
5.1.2. Código de la ventana principal	19
5.1.3. Código de la pantalla de administrador	36
5.1.4. Código para la base de datos	51
5.1.5. Código para el reconocimiento facial	55
5.1.6. Código para el reconocimiento de gestos	57

Índice de figuras

1.	Tabla de eventos vista mediante el software SQLite	7
2.	Base de datos en SQLite	7
3.	Verificación del gesto solicitado con superposición de landmarks y barra de progreso.	12
4.	Captura de cámara para la extracción del embedding facial y proceso de reconocimiento.	13
5.	Diálogo de solicitud de PIN para confirmar la identidad.	13
6.	Confirmación de acceso concedido y registro de salida del usuario.	14
7.	Panel de administración: gestión de usuarios y estado del sistema.	14
8.	Registro de nuevo usuario: solicitud de nombre.	15
9.	Registro de nuevo usuario: captura de datos faciales.	15
10.	Registro de nuevo usuario: solicitud de PIN y confirmación de alta.	16
11.	Registro de nuevo usuario: base de datos actualizada.	16

1. Descripción General

1.1. Introducción

Este proyecto desarrollado para la asignatura **LANAI** presenta un sistema avanzado de **control de acceso inteligente** basado en **reconocimiento facial y detección de gestos**. El sistema integra tecnologías de visión por computadora, aprendizaje automático y gestión de bases de datos para proporcionar un control de acceso seguro, robusto e intuitivo.

1.2. Objetivos del Proyecto

El proyecto tiene como objetivo principal desarrollar un sistema de control de acceso que sea:

- **Seguro:** Utiliza tecnologías de reconocimiento facial de última generación para garantizar autenticación confiable.
- **Intuitivo:** Interfaz gráfica amigable que permite a los administradores gestionar usuarios y permisos de forma sencilla.
- **Escalable:** Arquitectura modular que permite expandir funcionalidades e integrar nuevas características.
- **Eficiente:** Procesamiento optimizado de imágenes y análisis de datos en tiempo real.
- **Auditável:** Registro detallado de intentos de acceso con timestamps y datos de seguridad.

1.3. Características Principales

El sistema incluye las siguientes funcionalidades:

1. **Registro de Rostros:** Captura y almacenamiento de características faciales de usuarios autorizados.
2. **Autenticación Biométrica:** Verificación de identidad mediante análisis facial en tiempo real.
3. **Detección de Gestos:** Reconocimiento de gestos corporales para interacción adicional con el sistema.
4. **Gestión de Usuarios:** Panel administrativo para agregar/eliminar usuarios, asignar permisos y visualizar historial.
5. **Interfaz Gráfica Moderna:** Aplicación de escritorio basada en Tkinter con diseño intuitivo.
6. **Base de Datos Segura:** Almacenamiento de datos de usuarios con encriptación de credenciales.
7. **Logging y Auditoría:** Registro comprensivo de eventos de acceso y actividades del sistema.

1.4. Tecnologías Utilizadas

- **Lenguaje:** Python 3.11
- **GUI:** Tkinter para interfaz gráfica
- **Visión por Computadora:** OpenCV para procesamiento de imágenes y video
- **Reconocimiento Facial:** DeepFace con soporte para múltiples modelos
- **Detección de Gestos:** MediaPipe para análisis de pose y gestos corporales
- **Base de Datos:** SQLite para almacenamiento relacional
- **Cálculo Numérico:** NumPy para operaciones matemáticas

1.5. Estructura del Proyecto

El código está organizado en módulos especializados:

- **core/** - Módulos del sistema:
 - db_manager.py: Gestión de base de datos
 - face_recognition.py: Lógica de reconocimiento facial
 - gesture_detection.py: Detección de gestos
- **gui/** - Interfaz gráfica:
 - admin_window.py: Panel de administración
 - access_window.py: Ventana de acceso
- **dialogs/** - Diálogos especializados para registro de usuarios y captura de rostros
- **utils/** - Utilidades de autenticación y configuración

2. Desarrollo de la solución adoptada

En esta sección se describe el proceso de desarrollo del proyecto, incluyendo los desafíos encontrados y las soluciones implementadas.

2.1. Fase 0: Preparación del Entorno Virtual (venv)

Para aislar dependencias y asegurar reproducibilidad se utiliza un entorno virtual de Python (venv). Los pasos se realizan desde el terminal integrado de VS Code en la carpeta del proyecto (d:/Master/LANAI).

Creación y activación del entorno

```
# Comprobar versión de Python
py --version

# Crear el entorno (Python 3.11 recomendado)
py -3.11 -m venv .venv

# Activar (PowerShell)
.\venv\Scripts\Activate.ps1
# Si PowerShell bloquea scripts:
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

# Activar (CMD clásico)
.\venv\Scripts\activate.bat
```

Instalación de dependencias

```
# Actualizar pip
python -m pip install --upgrade pip

# Instalar desde requirements.txt (recomendado)
pip install -r requirements.txt

# Si no existe requirements.txt, instalar paquetes principales:
pip install opencv-python mediapipe deepface tensorflow pillow numpy bcrypt
```

Selección del intérprete en VS Code Desde la paleta de comandos: *Python: Select Interpreter* y escoger .venv\Scripts\python.exe.

Ejecución y desactivación

```
# Ejecutar la aplicación
python d:/Master/LANAI/main.py

# Desactivar el entorno cuando se termine
deactivate
```

Este entorno garantiza que la GUI (Tkinter), MediaPipe, DeepFace, OpenCV y el acceso a SQLite funcionen con versiones compatibles sin afectar otras instalaciones del sistema.

2.2. Fase 1: Selección de Tecnologías y Arquitectura Base

En la fase inicial del proyecto se evaluaron diferentes tecnologías para implementar el sistema de reconocimiento facial. La solución adoptada se basa en **OpenCV** y **DeepFace**, herramientas ampliamente utilizadas en visión por computadora que ofrecen un equilibrio entre precisión y eficiencia computacional.

Durante la investigación, se identificaron métodos avanzados de anti-spoofing y detección de ataques biométricos que podrían mejorar significativamente la robustez del sistema. Sin embargo, estas técnicas presentaban requisitos computacionales elevados que superaban las capacidades del hardware disponible para el desarrollo. Entre los métodos descartados se incluyen:

- **Análisis de Textura 3D:** Requiere procesamiento intensivo de múltiples capas de profundidad
- **Detección de Parpadeo:** Demanda análisis de fotogramas a alta velocidad
- **Análisis de Movimiento Facial:** Necesita procesamiento paralelo de secuencias de video
- **Verificación Multimodal Avanzada:** Combina múltiples biometría simultáneamente

Ante estas limitaciones, se decidió complementar el reconocimiento facial con un **sistema de detección de gestos** basado en **MediaPipe**. Este enfoque proporciona una capa adicional de autenticación mediante la verificación de gestos corporales específicos, mejorando significativamente la seguridad del sistema sin requerir recursos computacionales excesivos.

2.3. Fase 2: Base de Datos — Diseño y Estructura

Para el almacenamiento persistente se emplea **SQLite**, por su sencillez de despliegue y compatibilidad con Python sin necesidad de servidor externo. La base de datos se inicializa automáticamente al arrancar la aplicación si no existe el fichero, creando las tablas y claves foráneas necesarias. Los accesos se encapsulan en el módulo `core/db_manager.py`, que expone funciones para alta/baja de usuarios, almacenamiento de embeddings faciales y registro de eventos.

La estructura se compone de tres tablas principales:

- **users:** información básica del usuario.

```
id INTEGER PRIMARY KEY AUTOINCREMENT
name TEXT NOT NULL
pin TEXT NOT NULL
active INTEGER DEFAULT 1
created_at DATETIME DEFAULT CURRENT_TIMESTAMP
```

- **faces**: múltiples embeddings por usuario.

- **id** INTEGER PRIMARY KEY AUTOINCREMENT
- **user_id** INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE
- **encoding_json** TEXT NOT NULL
- **created_at** DATETIME DEFAULT CURRENT_TIMESTAMP

- **events**: auditoría de acciones y resultados.

- **id** INTEGER PRIMARY KEY AUTOINCREMENT
- **ts** DATETIME DEFAULT CURRENT_TIMESTAMP
- **device** TEXT
- **user_id** INTEGER REFERENCES users(id)
- **result** TEXT
- **note** TEXT

Estructura Hoja de datos Editar pragmas Ejecutar SQL						
Tabla: events						
	Id	ts	device	user_id	result	note
1	1	2025-12-13 13:36:46	demo-door-1	1	granted	Usuario registrado desde panel admin
2	2	2025-12-13 13:37:14	demo-door-1	1	Entrada Permitida	Acceso Permitido: Alvaro Viña ...
3	3	2025-12-13 13:37:17	demo-door-1	1	salida	Salida registrada: Alvaro Viña
4	4	2025-12-13 13:37:31	demo-door-1	1	Entrada Permitida	Acceso Permitido: Alvaro Viña ...
5	5	2025-12-13 13:37:34	demo-door-1	1	salida	Salida registrada: Alvaro Viña
6	6	2025-12-13 13:39:08	demo-door-1	1	Entrada Permitida	Acceso Permitido: Alvaro Viña ...
7	7	2025-12-13 13:39:33	demo-door-1	1	salida	Salida registrada: Alvaro Viña
8	8	2025-12-13 15:43:19	demo-door-1	2	granted	Usuario registrado desde panel admin
9	9	2025-12-13 15:43:43	demo-door-1	NULL	Entrada Denegada	Tiempo para gesto agotado
10	10	2025-12-13 15:44:09	demo-door-1	2	Entrada Permitida	Acceso Permitido: AlvaroPerez ...
11	11	2025-12-13 15:44:19	demo-door-1	2	salida	Salida registrada: AlvaroPerez

Figura 1: Tabla de eventos vista mediante el software SQLite

El **flujo de inicialización** crea las tablas si no existen y aplica **PRAGMA foreign_keys=ON** para garantizar integridad referencial. Durante el *registro de usuario*, se inserta la fila en **users** con el PIN **criptado con bcrypt**, y a continuación se añaden varios embeddings en **faces** vinculados por **user_id**. En *autenticación*, se carga un **snapshot** de usuarios activos y sus embeddings (**fetch_active_users_and_faces**) para el matching por similitud coseno. Todos los eventos críticos (acceso permitido/denegado, tiempo agotado, errores de cámara o rostro no detectado) se registran en **events** con su timestamp para trazabilidad.

Nombre	Tipo	Esquemas
Tables (4)		
events		CREATE TABLE events(id INTEGER PRIMARY KEY AUTOINCREMENT, ts DATETIME DEFAULT CURRENT_TIMESTAMP, device TEXT, user_id INTEGER, result TEXT, note TEXT)
faces		CREATE TABLE faces(id INTEGER PRIMARY KEY AUTOINCREMENT, user_id INTEGER NOT NULL, encoding_json TEXT NOT NULL, created_at DATETIME DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY(user_id) REFERENCES users(id))
sqlite_sequence		CREATE TABLE sqlite_sequence(name TEXT NOT NULL, seq INTEGER)
users		CREATE TABLE users(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, pin TEXT NOT NULL, active INTEGER DEFAULT 1, created_at DATETIME DEFAULT CURRENT_TIMESTAMP)
Indices (2)		CREATE INDEX idx_events_ts ON events(ts)
		CREATE INDEX idx_faces_user ON faces(user_id)
Views (0)		
Triggers (0)		

Figura 2: Base de datos en SQLite

2.4. Fase 3: Implementación del Reconocimiento Facial

2.4.1. Arquitectura del Módulo

El módulo de reconocimiento facial se implementó en `core/face_recognition.py` siguiendo un patrón modular que separa las responsabilidades de captura, procesamiento y verificación.

La clase `FaceRecognizer` encapsula la lógica principal:

- **Captura de Video:** Inicialización de cámara mediante OpenCV
- **Detección de Rostros:** Utilización de modelos pre-entrenados
- **Extracción de Características:** Generación de embeddings faciales mediante DeepFace
- **Verificación de Identidad:** Comparación de características con base de datos
- **Logging de Eventos:** Registro de intentos de autenticación

2.4.2. Funcionamiento del Reconocimiento Facial

El reconocimiento facial se basa en la extracción de un vector de características (embedding) a partir de un fotograma capturado con OpenCV. El frame se pasa directamente a *DeepFace*, que aplica el detector configurado y, si encuentra rostro, genera la representación numérica del mismo mediante el modelo elegido en `config.py`. Esta representación es un vector de alta dimensión que describe la identidad del rostro de forma robusta frente a variaciones moderadas de pose, iluminación y expresión.

Una vez obtenido el embedding del usuario en tiempo real, el sistema lo compara contra los embeddings almacenados por cada usuario activo. Para cada usuario existe un conjunto de ejemplos previos que reflejan distintas condiciones de captura. La comparación se realiza con la similitud coseno, que mide el ángulo entre vectores y favorece comparaciones en espacios normalizados, evitando que la magnitud distorsione el resultado. Para cada usuario se calcula el mejor caso (el máximo valor de similitud) y se selecciona el usuario cuyo valor global sea mayor. Este valor se contrasta en la interfaz con un umbral definido para decidir si la coincidencia es suficientemente fiable como para continuar con la verificación por PIN. Si no se detecta rostro, *DeepFace* lanza una excepción y la interfaz muestra el error correspondiente, manteniendo el flujo controlado y predecible.

2.4.3. Proceso de Registro de Rostros

Durante el registro, el sistema captura múltiples fotogramas del usuario desde diferentes ángulos para crear un modelo facial robusto. El proceso incluye:

1. Captura de 5 fotogramas con calidad mínima de detección
2. Validación de cada fotograma para evitar imágenes borrosas o parcialmente ocultas
3. Generación de embeddings faciales usando el modelo ArcFace de DeepFace
4. Almacenamiento de las características en la base de datos SQLite
5. Confirmación visual al usuario del registro exitoso

2.5. Fase 4: Verificación por PIN y uso de bcrypt

En esta fase se añade una segunda capa de seguridad tras el reconocimiento facial. La interfaz solicita al usuario su PIN mediante un diálogo modal. El PIN introducido nunca se almacena en claro: se compara contra el hash guardado en la base de datos utilizando *bcrypt*, un algoritmo de hashing adaptativo diseñado para credenciales.

El flujo es el siguiente: una vez identificada la mejor coincidencia facial por encima del umbral, se muestra el diálogo de PIN. El valor introducido se transforma a bytes y se verifica con `bcrypt.checkpw(pin_bytes, hash_bytes)`. Si la verificación es correcta, se concede el acceso y se registra el evento en la tabla `events`. En caso contrario, se deniega y se notifica el fallo.

Para el registro de nuevos usuarios, el PIN se cifra con *bcrypt* antes de persistirlo (`bcrypt.hashpw(pin, salt)`), incluyendo una sal aleatoria y un coste configurable que determina el número de rondas de cálculo. Este enfoque evita almacenar credenciales en texto plano y dificulta ataques por fuerza bruta al incrementar el coste computacional de cada intento.

La combinación de autenticación biométrica y verificación por PIN endurece el sistema frente a intentos de suplantación, manteniendo la experiencia de usuario simple y controlada desde la GUI.

2.6. Fase 5: Detección de Gestos

2.6.1. Arquitectura del Módulo

El módulo de detección de gestos se implementó en `core/gesture_detection.py` utilizando **MediaPipe Hands** (21 landmarks por mano). La clase **GestureDetector** encapsula:

- **Análisis de Manos:** Detección y seguimiento de manos con `mp.solutions.hands`
- **Clasificación de Gestos:** Reglas geométricas sobre coordenadas de landmarks (x, y)
- **Validación Temporal:** Requisito de frames consecutivos válidos para confirmar gesto

2.6.2. Gestos Implementados

El sistema reconoce:

1. **Pulgar Arriba:** Pulgar extendido y resto de dedos plegados
2. **Victoria (2 dedos):** Índice y medio extendidos, resto plegados
3. **OK (Círculo):** Pulgar e índice en contacto; al menos 3 dedos levantados
4. **Mano Abierta (5 dedos):** Todos los dedos extendidos
5. **Puño Cerrado:** Ningún dedo extendido

2.6.3. Funcionamiento de la Detección de Gestos

La detección de gestos utiliza *MediaPipe Hands* para identificar y seguir 21 puntos de referencia por mano en cada fotograma del vídeo. A partir de las coordenadas normalizadas de estos landmarks, el sistema aplica reglas geométricas sencillas que comparan posiciones relativas entre la punta de cada dedo y su articulación media, y emplea condiciones específicas para el pulgar, cuya orientación requiere comprobaciones en los ejes X e Y. Con estas reglas se clasifican gestos como pulgar arriba, victoria (dos dedos), OK (círculo pulgar–índice), mano abierta y puño cerrado.

La interfaz solicita un gesto aleatorio y superpone los landmarks junto con una barra de progreso que refleja la validación temporal. No basta con un único fotograma correcto: se exige coherencia a lo largo del tiempo. Por ello, el sistema requiere 30 fotogramas consecutivos válidos del gesto solicitado para confirmarlo. Cuando el fotograma no coincide, se penaliza el contador para evitar falsos positivos por ruido o detecciones inestables. Tras validar el gesto, se captura un fotograma para el reconocimiento facial y, si la similitud supera el umbral, se solicita el PIN y se completa el proceso de acceso.

2.6.4. Algoritmo de Detección

1. Captura continua de video con OpenCV
2. Procesamiento del fotograma con MediaPipe Hands
3. Extracción de coordenadas (x, y) de los 21 landmarks por mano
4. Conteo de dedos levantados comparando puntas con articulaciones medias
5. Reglas específicas por gesto (p. ej., proximidad pulgar–índice para “OK”)
6. Validación por **frames consecutivos**: se requiere **30** frames válidos

2.6.5. Integración con Autenticación

- Antes del reconocimiento facial, se solicita **un gesto aleatorio** del conjunto disponible
- El usuario dispone de **GESTURE_TIMEOUT** segundos (config.py) para completarlo
- El progreso se muestra con una **barra** y porcentaje; al alcanzar 30 frames válidos, el gesto se valida
- Si el gesto se valida, se continúa con reconocimiento facial y, posteriormente, verificación de PIN
- Fallos o timeout registran el evento y deniegan el acceso

2.7. Fase 6: Desarrollo de la Interfaz Gráfica

2.7.1. Tecnología de la GUI

La interfaz se desarrolló con **Tkinter**, integrando:

- **Canvas de Video:** Renderizado de frames de cámara (OpenCV → PIL → Tkinter).
- **Panel de Control:** Botón principal de verificación, estado del sistema y acceso a administración.
- **Indicadores:** Número de usuarios activos y mensajes de estado en tiempo real.

2.7.2. Estructura de la Ventana

La clase `VentanaAcceso` crea una ventana principal con:

- **Panel Izquierdo (Cámara):** Muestra la vista en vivo y, durante la verificación, superpone landmarks de MediaPipe Hands y barra de progreso del gesto.
- **Panel Derecho (Controles):** Botón “Verificar Acceso”, estado textual, separadores y acceso al panel de administración.
- **Diálogos:** Solicitud de PIN y ventana de `VentanaSalida` para registrar la salida.

2.7.3. Gestión del Ciclo de Video

- **Captura:** `cv2.VideoCapture` con ajustes de resolución (Windows: CAP_DSHOW).
- **Actualización:** Bucle con `root.after(30)` (33 FPS) que pinta el frame en el canvas.
- **Pausa/Reanudación:** Al abrir el panel de admin se libera la cámara y se muestra un mensaje; al cerrar, se reanuda.

2.7.4. Flujo de Verificación en la GUI

1. **Gestos:** Se solicita un gesto aleatorio; se valida con `GestureDetector` y 30 frames consecutivos correctos, mostrando una barra de progreso.
2. **Captura de Frame:** Se toma un frame espejado para el reconocimiento facial.
3. **Reconocimiento Facial:** Se obtiene el embedding (DeepFace) y se compara contra usuarios activos.
4. **PIN:** Para el mejor match sobre el umbral, se solicita PIN y se verifica con `bcrypt`.

2.7.5. Concurrencia y Estado

- **Hilo de Verificación:** La lógica completa corre en un hilo `daemon` para no bloquear la GUI.
- **Estados:** Se actualiza el texto y color del estado (*Cargando, Paso 1/4, Permitido, etc.*).
- **Manejo de Errores:** Mensajes `messagebox` y `log_event` ante fallos o tiempo agotado.

3. Demostración del Flujo del Sistema

En esta sección se muestran capturas del proceso completo: solicitud de gesto, validación, reconocimiento facial y verificación por PIN.

3.1. Paso 1: Solicitud y Validación de Gesto

El sistema inicia la verificación pidiendo al usuario un gesto aleatorio de la lista soportada (pulgar arriba, victoria, OK, mano abierta o puño). En pantalla se muestran los *landmarks* de la mano detectada y una barra de progreso que refleja la validación temporal: el gesto debe mantenerse correctamente durante 30 fotogramas consecutivos para considerarse válido. Si el gesto no coincide en un fotograma, el progreso se penaliza para evitar falsos positivos.



Figura 3: Verificación del gesto solicitado con superposición de landmarks y barra de progreso.

3.2. Paso 2: Captura de Frame para Reconocimiento Facial

Una vez validado el gesto, se captura un fotograma de la cámara (vista espejada) y se envía a DeepFace para extraer el *embedding* facial. Este vector se compara contra los embeddings almacenados por usuario para determinar la mejor coincidencia y su nivel de similitud.

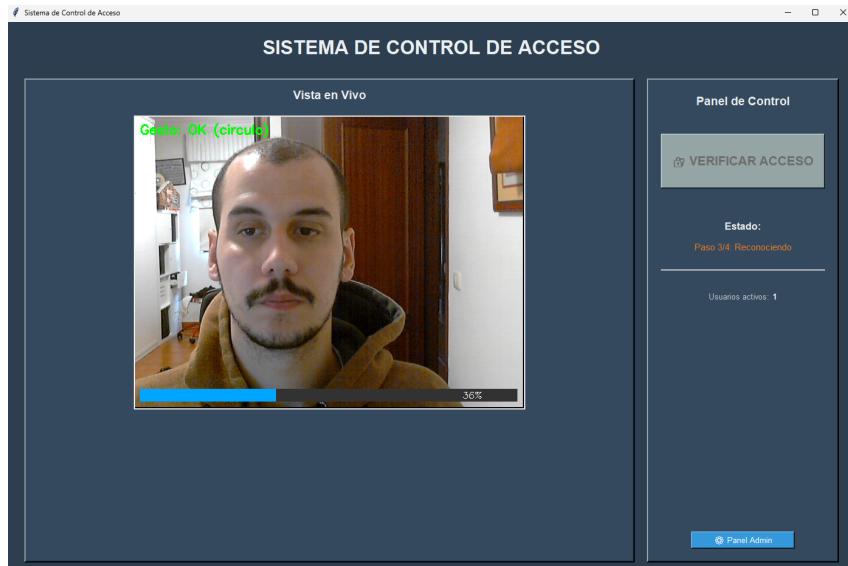


Figura 4: Captura de cámara para la extracción del embedding facial y proceso de reconocimiento.

3.3. Paso 3: Verificación de PIN y Acceso

Si la similitud supera el umbral definido, se solicita el PIN del usuario reconocido como segunda capa de seguridad. El PIN se verifica contra el hash almacenado en base de datos y, en caso de coincidencia, se concede el acceso y se muestra la ventana de registro de salida.

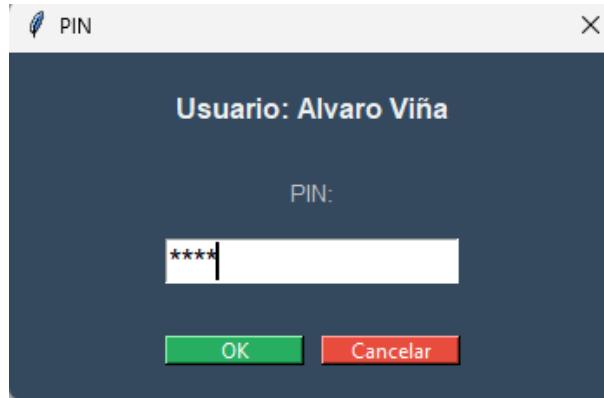


Figura 5: Diálogo de solicitud de PIN para confirmar la identidad.

3.4. Paso 4: Registro de salida

Tras conceder el acceso, el sistema muestra la ventana de “Registro de salida” con el nombre del usuario identificado. Desde esta interfaz se confirma la salida introduciendo de nuevo el PIN asociado, que se verifica contra el hash almacenado en la base de datos. Una vez validado, se registra el evento en el sistema de logging con timestamp y usuario, y se cierra la sesión mostrando un mensaje de confirmación. Este paso garantiza trazabilidad de entradas y salidas, y evita usos indebidos al requerir una confirmación explícita antes de finalizar.



Figura 6: Confirmación de acceso concedido y registro de salida del usuario.

3.5. Panel de Administración

El panel de administración ofrece funciones de *superusuario* para gestionar el sistema de forma segura. Desde esta interfaz es posible consultar la base de datos, registrar y eliminar usuarios, así como activar o desactivar cuentas según las necesidades operativas. Su diseño prioriza la claridad y la trazabilidad, integrando confirmaciones y mensajes de estado durante cada acción.

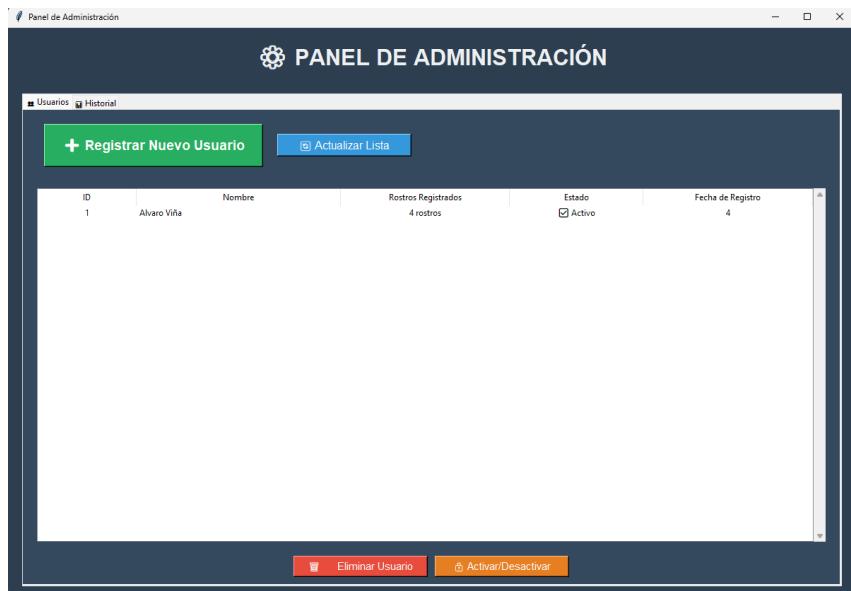


Figura 7: Panel de administración: gestión de usuarios y estado del sistema.

3.5.1. Registro de nuevo usuario

El alta de usuarios sólo puede ser realizada por un administrador autorizado. El proceso se compone de tres pasos guiados y verificados para garantizar la calidad de los datos registrados.

Paso 1: Nombre del usuario El sistema solicita el nombre del nuevo usuario mediante un diálogo sencillo. Este identificador se valida para evitar duplicidades y asegurar su consistencia con la base de datos.



Figura 8: Registro de nuevo usuario: solicitud de nombre.

Paso 2: Captura de datos faciales Se realizan cinco capturas del rostro con variaciones de pose y expresión (frontal, desviación lateral leve, sonrisa y ceño fruncido). A partir de estas imágenes, el sistema genera los *embeddings* faciales que servirán como referencia para la autenticación. Este conjunto de ejemplos mejora la robustez frente a cambios de iluminación y pose en el acceso.



Figura 9: Registro de nuevo usuario: captura de datos faciales.

Paso 3: Establecimiento de PIN Para añadir una segunda capa de seguridad, se solicita un PIN asociado al usuario. El PIN se almacena cifrado (*bcrypt*) en la base de datos, evitando guardar credenciales en claro y reforzando la protección de la información.

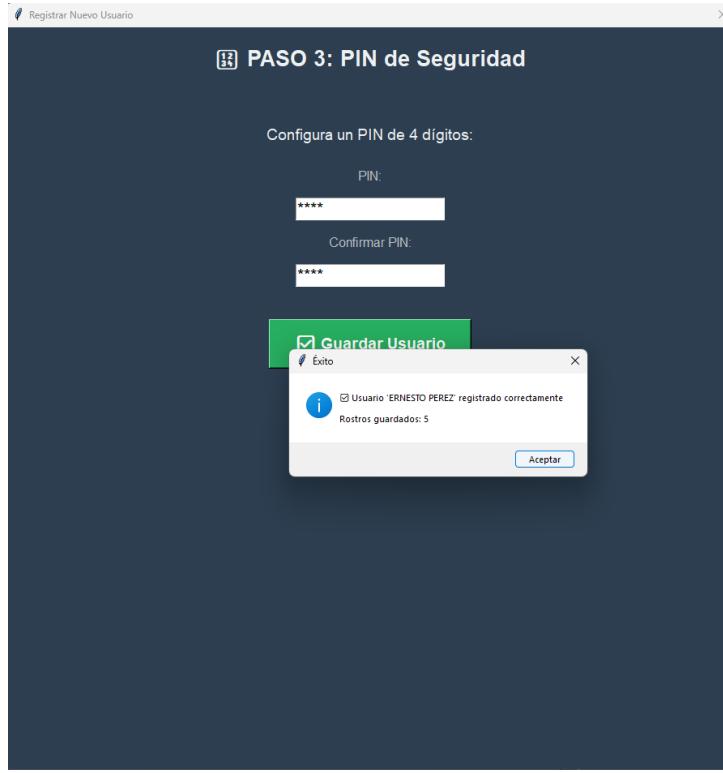


Figura 10: Registro de nuevo usuario: solicitud de PIN y confirmación de alta.

Actualización de la base de datos Tras completar el registro, los datos del usuario y sus *embeddings* se guardan en la base de datos. El panel confirma la operación y actualiza el listado de usuarios para reflejar el nuevo estado del sistema.

PANEL DE ADMINISTRACIÓN					
Eventos recientes:					
Fecha/Hora	Usuario	Resultado	Notas		
Fecha/Hora	Usuario	Resultado	Notas		
2025-12-13 13:10:20	Desconocido	Concedido	Usuario registrado desde panel admin		
2025-12-13 12:18:13	Desconocido	X Denegado	Salida registrada: Alvaro Vila		
2025-12-13 12:17:58	Desconocido	X Denegado	Acceso Permitido: Alvaro Vila score=0.724		
2025-12-13 12:16:59	Desconocido	X Denegado	Salida registrada: Alvaro Vila		
2025-12-13 12:16:14	Desconocido	X Denegado	Acceso Permitido: Alvaro Vila scores=0.752		
2025-12-13 12:14:26	Desconocido	X Denegado	No se detectó Rostro		
2025-12-13 12:14:25	Desconocido	X Denegado	Tentativa de acceso agendada		
2025-12-13 12:14:22	Desconocido	X Denegado	Salida registrada: Alvaro Vila		
2025-12-13 12:14:05	Desconocido	X Denegado	Acceso Permitido: Alvaro Vila score=0.775		
2025-12-13 12:13:01	Desconocido	X Denegado	Salida registrada: Alvaro Vila		
2025-12-13 12:37:36	Desconocido	X Denegado	score=0.769		
2025-12-13 12:37:18	Desconocido	X Denegado	unknown_scores=0.419		
2025-12-13 12:37:05	Desconocido	X Denegado	no_face		
2025-12-13 12:31:49	Desconocido	X Denegado	Salida registrada: Alvaro Vila		
2025-12-13 12:31:06	Desconocido	Concedido	score=0.779		
2025-12-13 12:23:18	Desconocido	Concedido	score=0.800		
2025-12-12 12:22:58	Desconocido	X Denegado	wrong_pin		
2025-12-04 14:31:48	Desconocido	Concedido	score=0.863		
2025-12-04 14:26:59	Desconocido	Concedido	Usuario registrado desde panel admin		
2025-12-04 14:18:33	Desconocido	Concedido	Usuario registrado desde panel admin		

Figura 11: Registro de nuevo usuario: base de datos actualizada.

4. Conclusiones

El sistema desarrollado integra reconocimiento facial, verificación por gestos y PIN cifrado, ofreciendo una autenticación multifactor sencilla de usar y con buena trazabilidad gracias al registro de eventos. La arquitectura modular y el uso de SQLite facilitan el

despliegue en entornos con recursos limitados, mientras que la GUI en Tkinter permite una operación clara y directa.

Ventajas observadas

El enfoque combinado biométrico+PIN reduce la probabilidad de suplantación y mejora la seguridad sin penalizar en exceso la experiencia de usuario. La validación temporal de gestos incrementa la robustez ante ruido en detección de manos. El uso de embeddings múltiples por usuario aumenta la tasa de acierto en condiciones variadas de iluminación y pose.

Desventajas y limitaciones

La ausencia de técnicas avanzadas de anti-spoofing deja abierto el riesgo frente a presentaciones con fotos o vídeos. La detección de pulgar puede verse afectada por orientación de la mano (diestra/zurda) y ángulo de cámara. El rendimiento depende del hardware disponible; en equipos modestos, DeepFace y MediaPipe pueden limitar la frecuencia de actualización.

Posibles mejoras técnicas

Software: incorporar anti-spoofing (detección de vida, parpadeo o textura), suavizado temporal de landmarks, calibración de umbral por usuario, almacenamiento binario de embeddings (en lugar de JSON) y tests automatizados para flujo de errores. **Hardware:** cámara con mejor óptica y sensor para baja iluminación, iluminación frontal controlada, GPU dedicada para acelerar inferencia y, opcionalmente, dispositivos de autenticación complementarios (lector NFC/QR) para escenarios híbridos.

Repositorio del proyecto

El código fuente completo y las instrucciones de instalación se encuentran en:

<https://github.com/AlvaroVP96/Proyecto-LANAI.git>

5. Anexo

5.1. Códigos principales

5.1.1. Código main

```
columns
1 # main.py
2 #
3 # Punto de entrada principal del sistema
4 #
5
6 import tkinter as tk
7 from gui.access_window import VentanaAcceso
8 from core import ensure_schema
9
10
11 def main():
12     """Función principal"""
13     # Asegurar que la BD existe con el esquema correcto
14     ensure_schema()
15
16     # Crear ventana principal
17     root = tk.Tk()
18     app = VentanaAcceso(root)
19
20     # Configurar cierre
21     root.protocol("WM_DELETE_WINDOW", app.cerrar)
22
23     # Iniciar loop
24     root.mainloop()
25
26
27 if __name__ == "__main__":
28     main()
```

Listing 1: Código main del programa

```
columns
1 # config.py
2 #
3 # Configuración global del sistema
4 #
5
6 # Base de datos
7 DB_PATH = "acceso.db"
8 DEVICE_NAME = "demo-door-1"
9
10 # Cámara
11 CAMERA_ID = 1
12 CAMERA_WIDTH = 640
13 CAMERA_HEIGHT = 480
```

```

14
15 # Reconocimiento facial
16 FACE_THRESHOLD = 0.70 # Umbral de similitud
17 FACE_MODEL = "ArcFace"
18 FACE_DETECTOR = "opencv"
19
20 # Gestos
21 GESTURE_TIMEOUT = 15 # segundos
22 GESTURE_FRAMES_REQUIRED = 30 # frames consecutivos
23
24 # Administrador
25 ADMIN_PIN_HASH = None # Se configurará en primera ejecución
26
27 # Colores GUI
28 COLOR_BG = "#2C3E50"
29 COLOR_PANEL = "#34495E"
30 COLOR_SUCCESS = "#27AE60"
31 COLOR_ERROR = "#E74C3C"
32 COLOR_WARNING = "#E67E22"
33 COLOR_INFO = "#3498DB"
34 COLOR_TEXT = "#ECF0F1"
35 COLOR_TEXT_SECONDARY = "#BDC3C7"

```

Listing 2: Código de configuracion de parámetros

5.1.2. Código de la ventana principal

```

columns
1 # gui/access_window.py
2 #
3 # Ventana principal de acceso
4 #
5
6 import tkinter as tk # Importa Tkinter base
7 from tkinter import ttk, messagebox # Importa widgets y cuadros de diálogo
8 import threading # Para ejecutar tareas en segundo plano
9 import cv2 # OpenCV para manejo de cámara y video
10 from PIL import Image, ImageTk # Para convertir imágenes a formato Tkinter
11 import random # Selección aleatoria de gestos
12 import time # Tiempos y esperas
13 import bcrypt # Verificación segura de PINs
14
15 from config import * # Configuración general (colores, tamaños,
16 thresholds)
16 # Importa funciones y clases esenciales desde el módulo 'core':
17 from core import (
18     fetch_active_users_and_faces, # Obtiene usuarios activos y sus embeddings
19     faciales
19     log_event, # Registra eventos (entradas/salidas, errores
19     , etc.)

```

```

20     get_embedding_deepface,           # Genera el embedding del rostro usando
21     DeepFace                      # DeepFace
22     best_match_per_user,          # Encuentra el mejor usuario que coincide con
23     el embedding
24     GestureDetector               # Clase para detectar y verificar gestos de
25     mano
26   )
27
28 import mediapipe as mp           # MediaPipe para detección de manos
29
30 class VentanaAcceso:
31     def __init__(self, root):
32         self.root = root           # Guarda la ventana
33         raíz
34         self.root.title("Sistema de Control de Acceso") # Título de ventana
35         self.root.geometry("1400x900") # Tamaño inicial
36         self.root.configure(bg=COLOR_BG) # Color de fondo
37
38         # Variables de estado
39         self.cap = None           # Capturador de cámara
40         self.verificando = False  # Flag de proceso de verificación en curso
41         self.detector = GestureDetector() # Instancia del detector de gestos
42         self.camara_activa = False # Flag para saber si la cámara está activa
43
44         self.frames_correctos = 0 # Contador de frames válidos del gesto
45         self.frames_necesarios = 30 # Frames consecutivos requeridos para validar gesto
46         self.gesto_actual = None # Identificador del gesto solicitado
47         self.gesto_objetivo = None # No usado (reservado)
48     )
49
50         self.usuario_verificando = None # No usado (reservado)
51
52
53         self.mp_hands = mp.solutions.hands # Referencia al módulo de manos
54         self.hands = self.mp_hands.Hands( # Inicializa el modelo de manos
55             static_image_mode=False, # Modo video (
56             seguimiento) # Máximo manos
57             max_num_hands=2, # Detectables
58             min_detection_confidence=0.5, # Detección
59             min_tracking_confidence=0.5 # Seguimiento
60

```

```

52     )
53     self.mp_drawing = mp.solutions.drawing_utils           # Utilidad para
dibujar landmarks
54
55     self.setup_ui()                                       # Construye la
interfaz
56     self.iniciar_video()                                 # Arranca la cámara
57
58 def setup_ui(self):
59     """Configura todos los elementos de la interfaz"""
60
61     # Título superior
62     titulo = tk.Label(
63         self.root,
64         text="SISTEMA DE CONTROL DE ACCESO",
65         font=("Arial", 24, "bold"),
66         bg=COLOR_BG,
67         fg=COLOR_TEXT
68     )
69     titulo.pack(pady=20)                                  # Añade el título con
espacio
70
71     # Frame principal que contendrá cámara y controles
72     frame_principal = tk.Frame(self.root, bg=COLOR_BG)
73     frame_principal.pack(expand=True, fill="both", padx=20, pady=10)
74
75     # Panel izquierdo: muestra la cámara
76     frame_camara = tk.Frame(frame_principal, bg=COLOR_PANEL, relief="raised",
bd=3)
77     frame_camara.pack(side="left", padx=10, fill="both", expand=True)
78
79     tk.Label(
80         frame_camara,
81         text="Vista en Vivo",
82         font=("Arial", 14, "bold"),
83         bg=COLOR_PANEL,
84         fg=COLOR_TEXT
85     ).pack(pady=10)                                     # Etiqueta del panel
de cámara
86
87     # Canvas donde se pinta el frame de video
88     self.canvas_video = tk.Canvas(
89         frame_camara,
90         width=CAMERA_WIDTH,
91         height=CAMERA_HEIGHT,
92         bg="#000000"
93     )
94     self.canvas_video.pack(pady=10, padx=10)
95
96     # Panel derecho: controles y estados
97     frame_controles = tk.Frame(frame_principal, bg=COLOR_PANEL, relief="
```

```

    raised", bd=3)
98     frame_controles.pack(side="right", padx=10, fill="both")

99
100    tk.Label(
101        frame_controles,
102        text="Panel de Control",
103        font=("Arial", 14, "bold"),
104        bg=COLOR_PANEL,
105        fg=COLOR_TEXT
106    ).pack(pady=20)                                # Título del panel
de control

107
108    # Botón principal para iniciar verificación
109    self.btn_verificar = tk.Button(
110        frame_controles,
111        text="VERIFICAR ACCESO",
112        font=("Arial", 16, "bold"),
113        bg=COLOR_SUCCESS,
114        fg="white",
115        activebackground="#229954",
116        command=self.iniciar_verificacion,          # Llama al proceso
de verificación
117        width=20,
118        height=3
119    )
120    self.btn_verificar.pack(pady=20)

121
122    # Estado actual del sistema
123    tk.Label(
124        frame_controles,
125        text="Estado:",
126        font=("Arial", 12, "bold"),
127        bg=COLOR_PANEL,
128        fg=COLOR_TEXT
129    ).pack(pady=(30, 5))

130
131    self.label_estado = tk.Label(
132        frame_controles,
133        text="Esperando...",
134        font=("Arial", 11),
135        bg=COLOR_PANEL,
136        fg=COLOR_WARNING,
137        wraplength=200
138    )
139    self.label_estado.pack(pady=5)

140
141    # Separador visual
142    ttk.Separator(frame_controles, orient="horizontal").pack(fill="x", pady
=20, padx=20)

143
144    # Bloque de información del sistema

```

```

145     info_frame = tk.Frame(frame_controles, bg=COLOR_PANEL)
146     info_frame.pack(pady=10, padx=20)
147
148     tk.Label(
149         info_frame,
150         text="Usuarios activos:",
151         font=("Arial", 10),
152         bg=COLOR_PANEL,
153         fg=COLOR_TEXT_SECONDARY
154     ).grid(row=0, column=0, sticky="w", pady=2)
155
156     self.label_usuarios = tk.Label(
157         info_frame,
158         text="0",
159         font=("Arial", 10, "bold"),
160         bg=COLOR_PANEL,
161         fg=COLOR_TEXT
162     )
163     self.label_usuarios.grid(row=0, column=1, sticky="e", pady=2)
164
165     # Botón para abrir el panel de administración
166     self.btn_admin = tk.Button(
167         frame_controles,
168         text="Panel Admin",
169         font=("Arial", 10),
170         bg=COLOR_INFO,
171         fg="white",
172         command=self.abrir_admin,                      # Abre panel admin
173         width=20
174     )
175     self.btn_admin.pack(side="bottom", pady=20)
176
177     # Carga número de usuarios activos
178     self.actualizar_info_sistema()
179
180 def iniciar_video(self):
181     """Inicia la captura de video"""
182     if not self.cap or not self.cap.isOpened():          # Si no
183         hay cámara activa
184         self.cap = cv2.VideoCapture(CAMERA_ID, cv2.CAP_DSHOW)      # Abre c
ámbara DirectShow (Windows)
185         self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH)    # Ajusta
ancho
186         self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT)   # Ajusta
alto
187         self.camara_activa = True                                # Marca
cámara como activa
188         self.actualizar_video()                                #
Empieza el loop de actualización
189

```

```

190     """Pausa la cámara sin liberarla"""
191     self.camara_activa = False
192     # Detiene el loop de actualización
193     if self.cap and self.cap.isOpened():
194         self.cap.release()
195         # Libera el dispositivo de cámara
196         self.cap = None
197         # Muestra mensaje de pausa en el canvas
198         self.canvas_video.delete("all")
199         # Limpia el canvas
200         self.canvas_video.create_text(
201             CAMERA_WIDTH // 2,
202             CAMERA_HEIGHT // 2,
203             text="CÁMARA PAUSADA\n\n(Panel de administración abierto)",
204             font=("Arial", 20, "bold"),
205             fill=COLOR_WARNING
206         )
207
208
209     def reanudar_camara(self):
210         """Reanuda la cámara"""
211         if not self.camara_activa:
212             # Solo si está pausada
213             self.iniciar_video()
214             # Reinicia cámara y loop
215
216     def actualizar_video(self):
217         """Actualiza el video en tiempo real"""
218         if not self.camara_activa:
219             # Si está pausada, no continúa
220             return
221
222         if self.cap and self.cap.isOpened():
223             ret, frame = self.cap.read()
224             # Lee un frame de la cámara
225             if ret:
226                 if self.verificando:
227                     # Si estás verificando gesto
228                     frame = self.procesar_frame_gestos(frame)
229                     # Procesa y dibuja overlay de gestos
230                 else:
231                     frame = cv2.flip(frame, 1)
232                     # Espejo para vista normal
233
234                     frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
235                     # Convierte a RGB para PIL
236                     img = Image.fromarray(frame_rgb)
237                     # Crea imagen PIL
238                     img_tk = ImageTk.PhotoImage(image=img)
239                     # Convierte a objeto Tkinter
240
241                     self.canvas_video.create_image(0, 0, anchor="nw", image=img_tk)

```

```

Pinta en canvas
228         self.canvas_video.image = img_tk                      #
Referencia para evitar GC
229
230     if self.camara_activa:                                     #
Reprograma el próximo frame
231         self.root.after(30, self.actualizar_video)           # ~33
FPS aprox.
232
233 def procesar_frame_gestos(self, frame):
234     """Procesa frame para gestos"""
235     frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)          #
Prepara frame para MediaPipe
236     results = self.hands.process(frame_rgb)                   #
Ejecuta detección de manos
237
238     gesto_correcto = False                                    # Flag
del estado del gesto actual
239
240     cv2.putText(frame, f"Gesto: {self.gesto_nombre}",        # Dibuja
nombre del gesto solicitado
241             (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
242
243     if results.multi_hand_landmarks:                          # Si hay
manos detectadas
244         for hand_landmarks in results.multi_hand_landmarks: # Itera
por cada mano
245             self.mp_drawing.draw_landmarks(                      # Dibuja
landmarks y conexiones
246                 frame, hand_landmarks, self.mp_hands.HAND_CONNECTIONS,
247                 self.mp_drawing.DrawingSpec(color=(0,255,0), thickness=2,
circle_radius=2),
248                 self.mp_drawing.DrawingSpec(color=(0,255,255), thickness=2)
249             )
250
251         if self.detector.verificar_gesto(self.gesto_actual,
hand_landmarks.landmark):
252             gesto_correcto = True                                # Marca
gesto correcto
253             self.frames_correctos += 1                         # Suma
frame válido
254
255         if not gesto_correcto:
256             self.frames_correctos = max(0, self.frames_correctos - 1) # Penaliza si no coincide
257
258         progreso = min(int((self.frames_correctos / self.frames_necesarios) *
100), 100) # % progreso
259
260         cv2.rectangle(frame, (10, 450), (630, 470), (50, 50, 50), -1) # Barra
de fondo

```

```

261     if progreso > 0:
262         color = (0, 255, 0) if gesto_correcto else (255, 165, 0)      # Verde
263         si va bien, naranja si no
264         cv2.rectangle(frame, (10, 450), (10 + int(progreso * 6.2), 470),
265         color, -1) # Barra progreso
266
267         cv2.putText(frame, f"{progreso}%", (540, 465),                  # Texto
268         del porcentaje
269             cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1
270
271     return frame
272
273 Devuelve frame con overlay
274
275 def verificacion_gesto_gui(self, timeout=GESTURE_TIMEOUT):
276     """Verificación de gesto"""
277     self.gesto_actual = random.choice(list(self.detector.gestos_disponibles.
278     keys())) # Elige gesto aleatorio
279     self.gesto_nombre = self.detector.gestos_disponibles[self.gesto_actual]
280         # Nombre legible
281     self.frames_correctos = 0
282         # Reinicia contador
283     self.modo_normal = False
284         # Activa modo verificación
285
286     start_time = time.time()
287         # Marca inicio
288
289     while True:
290         if time.time() - start_time > timeout:
291             # Si expira tiempo
292                 self.modo_normal = True
293                 self.gesto_actual = None
294                 raise TimeoutError("Tiempo agotado")
295             # Notifica timeout
296
297         if self.frames_correctos >= self.frames_necesarios:
298             # Si cumple frames
299                 time.sleep(0.5)
300             # Pequeña espera
301                 self.modo_normal = True
302                 self.gesto_actual = None
303                 return True
304             # Gesto validado
305
306             time.sleep(0.1)
307             # Evita busy-wait
308             self.root.update()
309             # Refresca GUI
310
311 def actualizar_info_sistema(self):
312     """Actualiza info del sistema"""

```

```

296     users, faces = fetch_active_users_and_faces()           # Obtiene
297     usuarios y embeddings
298     self.label_usuarios.config(text=str(len(users)))      # Muestra
299     cantidad de usuarios activos
300
301     def cambiar_estado(self, texto, color=COLOR_WARNING):
302         """Cambia estado"""
303         self.label_estado.config(text=texto, fg=color)        # Actualiza texto
304         y color del estado
305         self.root.update()                                  # Refresca GUI
306
307     def iniciar_verificacion(self):
308         """Inicia verificación"""
309         if self.verificando:                                # Evita múltiples
310             verificaciones simultáneas
311             messagebox.showwarning("Aviso", "Verificación en curso")
312             return
313
314         self.btn_verificar.config(state="disabled", bg="#95A5A6") # Deshabilita
315         botón mientras procesa
316         self.verificando = True                            # Marca estado
317         verificando
318
319         thread = threading.Thread(target=self.proceso_verificacion, daemon=True)
320         # Hilo en segundo plano
321         thread.start()                                    # Inicia hilo
322
323     def proceso_verificacion(self):
324         """Proceso de verificación completo"""
325         try:
326             self.cambiar_estado("Cargando...", COLOR_INFO)    # Estado:
327             cargando
328             users, faces = fetch_active_users_and_faces()      # Carga
329             usuarios y embeddings
330
331             if not users:                                     # Si no hay
332                 usuarios activos
333                 messagebox.showerror("Error", "No hay usuarios")
334                 return
335
336             # Paso 1: Gesto
337             self.cambiar_estado("Paso 1/4: Gesto", COLOR_WARNING) # Indica paso
338             try:
339                 if not self.verificacion_gesto_gui():            # Ejecuta
340                     verificación de gesto
341                     log_event(None, "Entrada Denegada", "Gesto fallido")
342                     messagebox.showerror("Denegado", "Gesto fallido")
343                     return
344             except TimeoutError:
345                 log_event(None, "Entrada Denegada", "Tiempo para gesto agotado")
346                 messagebox.showerror("Error", "Tiempo agotado")

```

```

336         return
337
338     # Paso 2: Captura de frame
339     self.cambiar_estado("Paso 2/4: Captura", COLOR_WARNING)
340     time.sleep(1)                                     # Pequeña
341     espera
342     ret, frame = self.cap.read()                      # Captura un
343     frame
344     if not ret:
345         messagebox.showerror("Error", "Captura fallida")
346         return
347     frame = cv2.flip(frame, 1)                         # Voltea para
348     vista natural
349
350     # Paso 3: Reconocimiento facial
351     self.cambiar_estado("Paso 3/4: Reconociendo", COLOR_WARNING)
352     try:
353         query_emb = get_embedding_deepface(frame)        # Obtiene
354         embedding del rostro
355     except:
356         log_event(None, "Entrada Denegada", "No se detecto Rostro")
357         messagebox.showerror("Error", "Sin rostro")
358         return
359
360         best_uid, best_score = best_match_per_user(query_emb, faces) # Busca
361         mejor coincidencia
362
363         if best_uid is None or best_score < FACE_THRESHOLD:      # Comprueba
364             umbral de similitud
365             log_event(None, "Entrada Denegada", f"No reconocido: {best_score
366             :.3f}")
367             messagebox.showerror("Denegado", f"Desconocido\nScore: {
368             best_score:.3f}")
369             return
370
371         # Paso 4: PIN del usuario reconocido
372         try:
373             user = users.get(best_uid)
374             print(best_uid)                                     # Datos del usuario
375             self.cambiar_estado(f"Usuario: {user['name']}", COLOR_INFO)
376         except Exception as e:
377             messagebox.showerror("NONE", str(e))
378
379         try:
380             pin = self.solicitar_pin(user['name'])  # Pide PIN
381         except Exception as e:
382             messagebox.showerror("NONE", str(e))
383
384         if not pin:
385             return                                         # Cancelado

```

```

379         if bcrypt.checkpw(pin.encode(), user["pin"].encode()): # Verifica
PIN contra hash
380             self.cambiar_estado("PERMITIDO", COLOR_SUCCESS)      # Estado
permítido
381             log_event(best_uid, "Entrada Permitida",
382                         f"Acceso Permitido: {user['name']} || score={best_score
383 :.3f}")
384             VentanaSalida(self.root, user['name'], best_uid)      # Abre
ventana de salida
385         else:
386             log_event(best_uid, "Entrada Denegada", "Pin Incorrecto")
387             messagebox.showerror("Denegado", "PIN incorrecto")
388
389     except Exception as e:
390         messagebox.showerror("Error", str(e))                      # Muestra
cualquier error inesperado
391     finally:
392         self.verificando = False                                # Resetea
flags
393         self.modo_normal = True
394         self.gesto_actual = None
395         self.btn_verificar.config(state="normal", bg=COLOR_SUCCESS) # Rehabilita botón
396         self.cambiar_estado("Esperando...", COLOR_WARNING)       # Estado por
defecto
397
398     def solicitar_pin(self, nombre):
399         """Diálogo PIN"""
400         dialog = tk.Toplevel(self.root)                          # Crea
ventana secundaria
401         dialog.title("PIN")
402         dialog.geometry("350x200")
403         dialog.configure(bg=COLOR_PANEL)
404         dialog.transient(self.root)                            # Se muestra
sobre la principal
405         dialog.grab_set()                                    # Bloquea
interacción con la raíz
406
407         dialog.geometry("+%d+%d" % (self.root.winfo_x() + 275, self.root.winfo_y()
408 () + 250)) # Posición
409
410         resultado = {"pin": None}                            # Contenedor
para resultado
411
412         tk.Label(dialog, text=f"Usuario: {nombre}", font=("Arial", 12, "bold"),
413                     bg=COLOR_PANEL, fg=COLOR_TEXT).pack(pady=20)      # Muestra
nombre
414
415         tk.Label(dialog, text="PIN:", font=("Arial", 10),
416                     bg=COLOR_PANEL, fg=COLOR_TEXT_SECONDARY).pack(pady=5) # Etiqueta
PIN

```

```

415     entry_pin = tk.Entry(dialog, show="*", font=("Arial", 14), width=15) #
416     Campo PIN
417     entry_pin.pack(pady=10)
418     entry_pin.focus()                                     # Foco para
419     escribir
420
421     def confirmar():
422         resultado["pin"] = entry_pin.get()                 # Guarda el
423         PIN
424         dialog.destroy()                                 # Cierra di
425     álogo
426
427     frame_btns = tk.Frame(dialog, bg=COLOR_PANEL)        #
428     Contenedor botones
429     frame_btns.pack(pady=20)
430
431     tk.Button(frame_btns, text="OK", command=confirmar,      # Botón
432     aceptar
433         bg=COLOR_SUCCESS, fg="white", width=10).pack(side="left", padx=5)
434     tk.Button(frame_btns, text="Cancelar", command=dialog.destroy, # Botón
435     cancelar
436         bg=COLOR_ERROR, fg="white", width=10).pack(side="left", padx=5)
437
438     entry_pin.bind("<Return>", lambda e: confirmar())       # Enter
439     confirma
440
441     dialog.wait_window()                                  # Espera
442     cierre
443     return resultado["pin"]                            # Devuelve
444     el PIN
445
446     def abrir_admin(self):
447         """Abre panel admin"""
448         self.abrir_panel_admin()                         # Delegado
449
450     def abrir_panel_admin(self):
451         """Abre el panel de administración"""
452         # PAUSAR CÁMARA ANTES DE ABRIR
453         self.pausar_camara()                           # Detiene c
454         ámara
455
456         from gui.admin_window import VentanaAdmin      # Importa
457         clase del panel
458         admin_window = VentanaAdmin(self.root)        # Instancia
459         panel admin
460
461         # Esperar a que se cierre el panel admin
462         if admin_window.window:
463             admin_window.window.wait_window()            # Espera

```

```

cierre de ventana

452
453     # REANUDAR CÁMARA AL CERRAR
454     self.reanudar_camara()                      # Reinicia
455     cámara
456
457     def cerrar(self):
458         """Cierra la aplicación"""
459         self.camara_activa = False                 # Detiene
460         loop
461         if self.cap:
462             self.cap.release()                      # Libera cá-
463             mara
464
465             # AGREGAR - Cerrar MediaPipe Hands
466             if hasattr(self, 'hands'):
467                 self.hands.close()                  # Cierra
468                 recursos de MediaPipe
469
470             self.root.destroy()                  # Cierra
471             ventana principal
472
473 class VentanaSalida:
474     """Ventana para registrar la salida del usuario"""
475
476     def __init__(self, parent, nombre_usuario, user_id):
477         self.parent = parent                      # Ventana
478         padre
479         self.nombre_usuario = nombre_usuario      # Nombre
480         del usuario
481         self.user_id = user_id                   # ID del
482         usuario
483
484         self.dialog = tk.Toplevel(parent)          # Crea
485         ventana de salida
486         self.dialog.title("Acceso Concedido")
487         self.dialog.geometry("500x350")
488         self.dialog.configure(bg=COLOR_BG)
489         self.dialog.transient(parent)              # Sobre la
490         principal
491         self.dialog.grab_set()                  # Bloquea
492         la principal
493
494         # Centrar ventana con respecto a la principal
495         self.dialog.geometry("+%d+%d" % (
496             parent.winfo_x() + 450,
497             parent.winfo_y() + 275
498         ))
499
500         self.setup_ui()                          # Construye
501         UI

```

```

490         self.dialog.wait_window()                                # Espera
491         cierre
492
493     def setup_ui(self):
494         """Configura la interfaz"""
495
496         # Título de bienvenida
497         tk.Label(
498             self.dialog,
499             text="ACCESO CONCEDIDO",
500             font=("Arial", 20, "bold"),
501             bg=COLOR_BG,
502             fg=COLOR_SUCCESS
503         ).pack(pady=20)
504
505         # Nombre del usuario
506         tk.Label(
507             self.dialog,
508             text=f"!Bienvenido, {self.nombre_usuario}!",
509             font=("Arial", 16),
510             bg=COLOR_BG,
511             fg=COLOR_TEXT
512         ).pack(pady=10)
513
514         # Separador
515         ttk.Separator(self.dialog, orient="horizontal").pack(fill="x", pady=20)
516
517         # Mensaje
518         tk.Label(
519             self.dialog,
520             text="Presiona el botón para registrar tu salida",
521             font=("Arial", 11),
522             bg=COLOR_BG,
523             fg=COLOR_TEXT_SECONDARY
524         ).pack(pady=10)
525
526         # Botón salir
527         tk.Button(
528             self.dialog,
529             text="Registrar Salida",
530             font=("Arial", 16, "bold"),
531             bg=COLOR_WARNING,
532             fg="white",
533             command=self.solicitar_salida,                                # Abre diá
534             logo de salida
535             width=20,
536             height=2
537         ).pack(pady=30)
538
539     def solicitar_salida(self):
540         """Abre diálogo para registrar la salida"""

```

```

539     dialog_salida = tk.Toplevel(self.dialog) # Ventana
secundaria
540     dialog_salida.title("Registrar Salida")
541     dialog_salida.geometry("400x280")
542     dialog_salida.configure(bg=COLOR_PANEL)
543     dialog_salida.transient(self.dialog) # Sobre la
de acceso concedido
544     dialog_salida.grab_set() # Bloquea
interacción
545
546     resultado = {"confirmado": False} # Contenedor del resultado
547
548     tk.Label(
549         dialog_salida,
550         text="Registrar Salida",
551         font=("Arial", 16, "bold"),
552         bg=COLOR_PANEL,
553         fg=COLOR_TEXT
554     ).pack(pady=20)
555
556     # Nombre
557     tk.Label(
558         dialog_salida,
559         text="Nombre:",
560         font=("Arial", 11),
561         bg=COLOR_PANEL,
562         fg=COLOR_TEXT
563     ).pack(pady=5)
564
565     entry_nombre = tk.Entry(
566         dialog_salida,
567         font=("Arial", 12),
568         width=25
569     )
570     entry_nombre.pack(pady=5)
571     entry_nombre.insert(0, self.nombre_usuario) # Pre-
llenado con el nombre
572
573     # PIN
574     tk.Label(
575         dialog_salida,
576         text="PIN:",
577         font=("Arial", 11),
578         bg=COLOR_PANEL,
579         fg=COLOR_TEXT
580     ).pack(pady=5)
581
582     entry_pin = tk.Entry(
583         dialog_salida,
584         font=("Arial", 12),

```

```

585         width=25,
586         show="*"
587     )
588     entry_pin.pack(pady=5)
589     entry_pin.focus()                                     # Foco en
el PIN

590
591     def confirmar_salida():
592         nombre = entry_nombre.get().strip()                 # Lee
nombre
593         pin = entry_pin.get().strip()                      # Lee PIN
594
595         if not nombre:
596             messagebox.showerror("Error", "El nombre es obligatorio", parent=
dialog_salida)
597             return
598
599         if not pin:
600             messagebox.showerror("Error", "El PIN es obligatorio", parent=
dialog_salida)
601             return
602
603         # Verificar PIN y nombre
604         try:
605             from core import fetch_active_users_and_faces           # Import
para obtener usuarios
606             users, _ = fetch_active_users_and_faces()
607
608             if self.user_id in users and nombre == users[self.user_id]["name"]:
609                 # Comprueba identidad
610                 user_pin_hash = users[self.user_id]["pin"]          # Hash del
PIN
611
612                 if bcrypt.checkpw(pin.encode(), user_pin_hash.encode()): #
Verifica PIN
613                     # PIN correcto - registrar salida
614                     log_event(
615                         self.user_id,
616                         "salida",
617                         f"Salida registrada: {nombre}"
618                     )
619
620                     messagebox.showinfo(
621                         "Éxito",
622                         f"Salida registrada correctamente.\n!Hasta luego, {nombre}!",
623                         parent=dialog_salida
624                     )
625
626                     resultado["confirmado"] = True                      # Marca
confirmación

```

```

626         dialog_salida.destroy()                                # Cierra
secundario
627         self.dialog.destroy()                                # Cierra
principal
628     else:
629         messagebox.showerror("Error", "PIN incorrecto", parent=
dialog_salida)
630         entry_pin.delete(0, tk.END)                         # Limpia
campo PIN
631         entry_pin.focus()
632     else:
633         messagebox.showerror("Error", "Usuario no encontrado", parent=
dialog_salida)
634
635     except Exception as e:
636         messagebox.showerror("Error", f"Error al verificar: {e}", parent=
dialog_salida)
637
# Botones de acción
638     frame_botones = tk.Frame(dialog_salida, bg=COLOR_PANEL)
639     frame_botones.pack(pady=20)
640
641     tk.Button(
642         frame_botones,
643         text="Confirmar",
644         font=("Arial", 11),
645         bg=COLOR_SUCCESS,
646         fg="white",
647         command=confirmar_salida,                                # Ejecuta
verificación y registro
648         width=12
649     ).pack(side="left", padx=5)
650
651
652     tk.Button(
653         frame_botones,
654         text="Cancelar",
655         font=("Arial", 11),
656         bg=COLOR_ERROR,
657         fg="white",
658         command=dialog_salida.destroy,                            # Cierra el
diálogo
659         width=12
660     ).pack(side="left", padx=5)
661
662     # Enter para confirmar
663     entry_pin.bind("<Return>", lambda e: confirmar_salida())    # Atajo de
teclado

```

Listing 3: Ventana de acceso GUI

5.1.3. Código de la pantalla de administrador

```
columns
1 # gui/admin_window.py
2 #
3 # Panel de administración simplificado
4 #
5 import tkinter as tk
6 from tkinter import ttk, messagebox, filedialog
7 from datetime import datetime
8 import cv2
9 from PIL import Image, ImageTk
10 import bcrypt
11 import time
12
13 from config import *
14 from core import (
15     ensure_schema,
16     get_all_users,
17     update_user_status,
18     delete_user,
19     get_recent_events,
20     insert_user,
21     insert_face,
22     get_embedding_deepface,
23     log_event
24 )
25 from utils.admin_auth import verificar_admin
26
27
28 class VentanaAdmin:
29     def __init__(self, parent):
30         self.parent = parent
31
32         # Verificar autenticación
33         if not verificar_admin(parent):
34             self.window = None
35             return
36
37         self.window = tk.Toplevel(parent)
38         self.window.title("Panel de Administración")
39         self.window.geometry("1200x800")
40         self.window.configure(bg=COLOR_BG)
41
42         # Variables
43         self.usuarios_data = []
44         self.cap_registro = None # <-- Cámara para registro
45         self.camara_registro_activa = False # <-- Estado de cámara de registro
46
47         self.setup_ui()
48         self.cargar_usuarios()
```

```

49         self.cargar_eventos()
50
51     self.window.protocol("WM_DELETE_WINDOW", self.cerrar)
52
53 def setup_ui(self):
54     """Configura la interfaz"""
55     # Título
56     tk.Label(
57         self.window,
58         text="PANEL DE ADMINISTRACIÓN",
59         font=("Arial", 24, "bold"),
60         bg=COLOR_BG,
61         fg=COLOR_TEXT
62     ).pack(pady=20)
63
64     # Tabs
65     self.notebook = ttk.Notebook(self.window)
66     self.notebook.pack(expand=True, fill="both", padx=20, pady=10)
67
68     # Tab 1: Usuarios
69     self.tab_usuarios = tk.Frame(self.notebook, bg=COLOR_PANEL)
70     self.notebook.add(self.tab_usuarios, text=" Usuarios")
71
72     # Tab 2: Historial
73     self.tab_historial = tk.Frame(self.notebook, bg=COLOR_PANEL)
74     self.notebook.add(self.tab_historial, text=" Historial")
75
76     self.setup_tab_usuarios()
77     self.setup_tab_historial()
78
79     # ===== TAB USUARIOS =====
80
81 def setup_tab_usuarios(self):
82     """Configura la pestaña de usuarios"""
83     # Frame superior - Botones
84     frame_botones = tk.Frame(self.tab_usuarios, bg=COLOR_PANEL)
85     frame_botones.pack(pady=20, fill="x", padx=20)
86
87     tk.Button(
88         frame_botones,
89         text=" Registrar Nuevo Usuario",
90         font=("Arial", 14, "bold"),
91         bg=COLOR_SUCCESS,
92         fg="white",
93         command=self.registrar_nuevo_usuario,
94         width=25,
95         height=2
96     ).pack(side="left", padx=10)
97
98     tk.Button(
99         frame_botones,

```

```

100     text="Actualizar Lista",
101     font=("Arial", 12),
102     bg=COLOR_INFO,
103     fg="white",
104     command=self.cargar_usuarios,
105     width=20
106 ).pack(side="left", padx=10)
107
108 # Frame tabla
109 frame_tabla = tk.Frame(self.tab_usuarios, bg=COLOR_PANEL)
110 frame_tabla.pack(expand=True, fill="both", padx=20, pady=10)
111
112 # Scrollbar
113 scrollbar = ttk.Scrollbar(frame_tabla)
114 scrollbar.pack(side="right", fill="y")
115
116 # Treeview
117 columns = ("ID", "Nombre", "Rostros", "Estado", "Fecha Registro")
118 self.tree_usuarios = ttk.Treeview(
119     frame_tabla,
120     columns=columns,
121     show="headings",
122     yscrollcommand=scrollbar.set,
123     height=15
124 )
125
126 # Configurar columnas
127 self.tree_usuarios.heading("ID", text="ID")
128 self.tree_usuarios.heading("Nombre", text="Nombre")
129 self.tree_usuarios.heading("Rostros", text="Rostros Registrados")
130 self.tree_usuarios.heading("Estado", text="Estado")
131 self.tree_usuarios.heading("Fecha Registro", text="Fecha de Registro")
132
133 self.tree_usuarios.column("ID", width=50, anchor="center")
134 self.tree_usuarios.column("Nombre", width=200)
135 self.tree_usuarios.column("Rostros", width=150, anchor="center")
136 self.tree_usuarios.column("Estado", width=100, anchor="center")
137 self.tree_usuarios.column("Fecha Registro", width=150, anchor="center")
138
139 self.tree_usuarios.pack(expand=True, fill="both")
140 scrollbar.config(command=self.tree_usuarios.yview)
141
142 # Botones de acción
143 frame_acciones = tk.Frame(self.tab_usuarios, bg=COLOR_PANEL)
144 frame_acciones.pack(pady=10)
145
146 tk.Button(
147     frame_acciones,
148     text="Eliminar Usuario",
149     font=("Arial", 11),
150     bg=COLOR_ERROR,

```

```

151     fg="white",
152     command=self.eliminar_usuario,
153     width=20
154 ).pack(side="left", padx=5)
155
156     tk.Button(
157         frame_acciones,
158         text="Activar/Desactivar",
159         font=("Arial", 11),
160         bg=COLOR_WARNING,
161         fg="white",
162         command=self.toggle_usuario,
163         width=20
164     ).pack(side="left", padx=5)
165
166 def cargar_usuarios(self):
167     """Carga la lista de usuarios"""
168     # Limpiar tabla
169     for item in self.tree_usuarios.get_children():
170         self.tree_usuarios.delete(item)
171
172     # Obtener usuarios
173     usuarios = get_all_users()
174     self.usuarios_data = usuarios
175
176     # Agregar a la tabla
177     for usuario in usuarios:
178         user_id, nombre, pin, active, created_at = usuario
179
180         # Contar rostros manualmente desde la BD
181         import sqlite3
182         from config import DB_PATH
183         conn = sqlite3.connect(DB_PATH)
184         c = conn.cursor()
185         c.execute("SELECT COUNT(*) FROM faces WHERE user_id = ?", (user_id,))
186         num_rostros = c.fetchone()[0]
187         conn.close()
188
189         estado = "Activo" if active else "Inactivo"
190         rostros_text = f"{num_rostros} rostros" if num_rostros else "Sin
191         rostros"
192
193         # CORREGIR MANEJO DE FECHA
194         if isinstance(created_at, str):
195             fecha = created_at[:10] # Ya es string
196         else:
197             # Si es timestamp o None
198             fecha = str(created_at) if created_at else "N/A"
199
200         self.tree_usuarios.insert(
201             "",

```

```

201         "end",
202         values=(user_id, nombre, rostros_text, estado, fecha)
203     )
204
205     def eliminar_usuario(self):
206         """Elimina un usuario seleccionado"""
207         seleccion = self.tree_usuarios.selection()
208         if not seleccion:
209             messagebox.showwarning("Advertencia", "Selecciona un usuario primero")
210         return
211
212         item = self.tree_usuarios.item(seleccion[0])
213         user_id = item['values'][0]
214         nombre = item['values'][1]
215
216         if messagebox.askyesno("Confirmar", f"?Eliminar usuario '{nombre}'?\n\nEsta acción no se puede deshacer."):
217             try:
218                 delete_user(user_id)
219                 messagebox.showinfo("Éxito", f"Usuario '{nombre}' eliminado
correctamente")
220                 self.cargar_usuarios()
221             except Exception as e:
222                 messagebox.showerror("Error", f"Error al eliminar usuario: {e}")
223
224     def toggle_usuario(self):
225         """Activa/desactiva un usuario"""
226         seleccion = self.tree_usuarios.selection()
227         if not seleccion:
228             messagebox.showwarning("Advertencia", "Selecciona un usuario primero")
229         return
230
231         item = self.tree_usuarios.item(seleccion[0])
232         user_id = item['values'][0]
233         nombre = item['values'][1]
234         estado_actual = "Activo" in item['values'][3]
235
236         nuevo_estado = 0 if estado_actual else 1
237         accion = "desactivar" if estado_actual else "activar"
238
239         if messagebox.askyesno("Confirmar", f"?{accion.capitalize()} usuario '{nombre}'?"):
240             try:
241                 update_user_status(user_id, nuevo_estado)
242                 messagebox.showinfo("Éxito", f"Usuario '{nombre}' {accion}do
correctamente")
243                 self.cargar_usuarios()
244             except Exception as e:
245                 messagebox.showerror("Error", f"Error al {accion} usuario: {e}")

```

```

246
247 # ===== REGISTRO NUEVO USUARIO =====
248
249 def registrar_nuevo_usuario(self):
250     """Inicia el proceso de registro de nuevo usuario"""
251     # Crear ventana de registro
252     self.dialog_registro = tk.Toplevel(self.window)
253     self.dialog_registro.title("Registrar Nuevo Usuario")
254     self.dialog_registro.geometry("900x700")
255     self.dialog_registro.configure(bg=COLOR_BG)
256     self.dialog_registro.transient(self.window)
257     self.dialog_registro.grab_set()
258
259     # Variables del proceso
260     self.nombre_usuario = None
261     self.capturas_rostro = []
262     self.embeddings_rostro = []
263     self.pin_usuario = None
264     self.cap_registro = None
265     self.paso_actual = 1 # 1: Nombre, 2: Captura rostros, 3: PIN, 4: Guardar
266
267     self.setup_dialogo_registro()
268
269     self.dialog_registro.protocol("WM_DELETE_WINDOW", self.cerrar_registro)
270
271 def setup_dialogo_registro(self):
272     """Configura el diálogo de registro"""
273     # Título
274     self.label_titulo_registro = tk.Label(
275         self.dialog_registro,
276         text="PASO 1: Nombre del Usuario",
277         font=("Arial", 20, "bold"),
278         bg=COLOR_BG,
279         fg=COLOR_TEXT
280     )
281     self.label_titulo_registro.pack(pady=20)
282
283     # Frame contenido (cambiará según el paso)
284     self.frame_contenido = tk.Frame(self.dialog_registro, bg=COLOR_BG)
285     self.frame_contenido.pack(expand=True, fill="both", padx=40, pady=20)
286
287     # Mostrar paso 1
288     self.mostrar_paso_nombre()
289
290 # ===== PASO 1: NOMBRE =====
291
292 def mostrar_paso_nombre(self):
293     """Muestra el formulario para ingresar nombre"""
294     # Limpiar frame
295     for widget in self.frame_contenido.winfo_children():
296         widget.destroy()

```

```

297     tk.Label(
298         self.frame_contenido,
299         text="Introduce el nombre completo del usuario:",
300         font=("Arial", 14),
301         bg=COLOR_BG,
302         fg=COLOR_TEXT
303     ).pack(pady=20)
304
305
306     self.entry_nombre = tk.Entry(
307         self.frame_contenido,
308         font=("Arial", 16),
309         width=30
310     )
311     self.entry_nombre.pack(pady=10)
312     self.entry_nombre.focus()
313
314     tk.Button(
315         self.frame_contenido,
316         text="Continuar",
317         font=("Arial", 14, "bold"),
318         bg=COLOR_SUCCESS,
319         fg="white",
320         command=self.validar_nombre,
321         width=20,
322         height=2
323     ).pack(pady=30)
324
325     self.entry_nombre.bind("<Return>", lambda e: self.validar_nombre())
326
327 def validar_nombre(self):
328     """Valida el nombre y pasa al siguiente paso"""
329     nombre = self.entry_nombre.get().strip()
330
331     if not nombre:
332         messagebox.showerror("Error", "El nombre es obligatorio", parent=self.dialog_registro)
333         return
334
335     if len(nombre) < 3:
336         messagebox.showerror("Error", "El nombre debe tener al menos 3 caracteres", parent=self.dialog_registro)
337         return
338
339     self.nombre_usuario = nombre
340     self.paso_actual = 2
341     self.label_titulo_registro.config(text="PASO 2: Captura de Rostros")
342     self.mostrar_paso_rostros()
343
344 # ===== PASO 2: CAPTURA ROSTROS =====
345

```

```

346     def actualizar_video_registro(self):
347         """Actualiza el video en tiempo real"""
348         if self.cap_registro and self.cap_registro.isOpened() and self.
349             camara_registro_activa:
350             ret, frame = self.cap_registro.read()
351             if ret:
352                 frame = cv2.flip(frame, 1)
353
354                 # Dibujar guía
355                 h, w = frame.shape[:2]
356                 cv2.rectangle(frame, (w//4, h//4), (3*w//4, 3*h//4), (0, 255, 0),
357                               2)
358                 cv2.putText(frame, "Centra tu rostro aqui",
359                             (w//4 + 10, h//4 - 10),
360                             cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
361
362                 frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
363                 img = Image.fromarray(frame_rgb)
364                 img_tk = ImageTk.PhotoImage(image=img)
365
366                 self.canvas_registro.create_image(0, 0, anchor="nw", image=img_tk)
367             )
368             self.canvas_registro.image = img_tk
369
370             if hasattr(self, 'dialog_registro') and self.dialog_registro.winfo_exists():
371                 self.dialog_registro.after(30, self.actualizar_video_registro)
372
373     def mostrar_paso_rostros(self):
374         """Muestra la interfaz de captura de rostros"""
375         # Limpiar frame
376         for widget in self.frame_contenido.winfo_children():
377             widget.destroy()
378
379         # Instrucciones
380         tk.Label(
381             self.frame_contenido,
382             text=f"Capturando rostros para: {self.nombre_usuario}",
383             font=("Arial", 14, "bold"),
384             bg=COLOR_BG,
385             fg=COLOR_TEXT
386         ).pack(pady=10)
387
388         tk.Label(
389             self.frame_contenido,
390             text="Captura 5 fotos desde diferentes ángulos",
391             font=("Arial", 12),
392             bg=COLOR_BG,
393             fg=COLOR_TEXT_SECONDARY
394         ).pack(pady=5)
395
396

```

```

393     # Canvas para video
394     self.canvas_registro = tk.Canvas(
395         self.frame_contenido,
396         width=640,
397         height=480,
398         bg="#000000"
399     )
400     self.canvas_registro.pack(pady=20)
401
402     # Progreso
403     self.label_progreso = tk.Label(
404         self.frame_contenido,
405         text="0 / 5 fotos capturadas",
406         font=("Arial", 14),
407         bg=COLOR_BG,
408         fg=COLOR_WARNING
409     )
410     self.label_progreso.pack(pady=10)
411
412     # Botón capturar
413     self.btn_capturar = tk.Button(
414         self.frame_contenido,
415         text="Capturar Foto",
416         font=("Arial", 14, "bold"),
417         bg=COLOR_SUCCESS,
418         fg="white",
419         command=self.capturar_rostro,
420         width=20,
421         height=2
422     )
423     self.btn_capturar.pack(pady=10)
424
425     # Iniciar cámara de registro
426     if not self.cap_registro or not self.cap_registro.isOpened():
427         self.cap_registro = cv2.VideoCapture(CAMERA_ID, cv2.CAP_DSHOW)
428         self.cap_registro.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
429         self.cap_registro.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
430
431         self.camara_registro_activa = True # <-- ACTIVAR CÁMARA
432         self.actualizar_video_registro()
433
434     def capturar_rostro(self):
435         """Captura una foto del rostro"""
436         if len(self.capturas_rostro) >= 5:
437             return
438
439         self.btn_capturar.config(state="disabled")
440
441         # Countdown
442         for i in range(3, 0, -1):
443             self.label_progreso.config(text=f"Capturando en {i}...")
```

```

444     self.dialog_registro.update()
445     time.sleep(1)
446
447     # Capturar
448     ret, frame = self.cap_registro.read()
449     if ret:
450         frame = cv2.flip(frame, 1)
451         self.capturas_rostro.append(frame.copy())
452
453         # Actualizar progreso
454         num = len(self.capturas_rostro)
455         self.label_progreso.config(text=f"{num} / 5 fotos capturadas")
456
457         # Flash
458         self.canvas_registro.config(bg="white")
459         self.dialog_registro.update()
460         time.sleep(0.1)
461         self.canvas_registro.config(bg="black")
462
463     if num >= 5:
464         self.label_progreso.config(fg=COLOR_SUCCESS)
465         self.btn_capturar.config(text="Completado", state="disabled")
466
467     # Botón continuar
468     tk.Button(
469         self.frame_contenido,
470         text="Verificar y Continuar",
471         font=("Arial", 12, "bold"),
472         bg=COLOR_INFO,
473         fg="white",
474         command=self.verificar_rostros,
475         width=20
476     ).pack(pady=10)
477 else:
478     self.btn_capturar.config(state="normal")
479
480 def verificar_rostros(self):
481     """Verifica que los rostros se detectaron correctamente"""
482     self.btn_capturar.config(state="disabled")
483     self.label_progreso.config(text="Verificando rostros...")
484     self.dialog_registro.update()
485
486     embeddings_ok = 0
487
488     for frame in self.capturas_rostro:
489         try:
490             embedding = get_embedding_deepface(frame)
491             self.embeddings_rostro.append(embedding)
492             embeddings_ok += 1
493         except Exception as e:
494             print(f"Error al procesar rostro: {e}")

```

```

495     if embeddings_ok >= 3: # Al menos 3 rostros válidos
496         messagebox.showinfo(
497             "Éxito",
498             f"{embeddings_ok} rostros verificados correctamente",
499             parent=self.dialog_registro
500         )
501         self.cerrar_camara_registro()
502         self.paso_actual = 3
503         self.label_titulo_registro.config(text="PASO 3: PIN de Seguridad")
504         self.mostrar_paso_pin()
505     else:
506         messagebox.showerror(
507             "Error",
508             f"Solo se detectaron {embeddings_ok} rostros válidos.\nIntenta de
509             nuevo con mejor iluminación.",
510             parent=self.dialog_registro
511         )
512         self.capturas_rostro = []
513         self.embeddings_rostro = []
514         self.mostrar_paso_rostros()
515
516 # ===== PASO 3: PIN =====
517
518 def mostrar_paso_pin(self):
519     """Muestra el formulario para ingresar PIN"""
520     # Limpiar frame
521     for widget in self.frame_contenido.winfo_children():
522         widget.destroy()
523
524     tk.Label(
525         self.frame_contenido,
526         text="Configura un PIN de 4 dígitos:",
527         font=("Arial", 14),
528         bg=COLOR_BG,
529         fg=COLOR_TEXT
530     ).pack(pady=20)
531
532     tk.Label(
533         self.frame_contenido,
534         text="PIN:",
535         font=("Arial", 12),
536         bg=COLOR_BG,
537         fg=COLOR_TEXT_SECONDARY
538     ).pack(pady=5)
539
540     self.entry_pin = tk.Entry(
541         self.frame_contenido,
542         font=("Arial", 16),
543         width=15,
544         show="*"

```

```

545     )
546     self.entry_pin.pack(pady=10)
547     self.entry_pin.focus()
548
549     tk.Label(
550         self.frame_contenido,
551         text="Confirmar PIN:",
552         font=("Arial", 12),
553         bg=COLOR_BG,
554         fg=COLOR_TEXT_SECONDARY
555     ).pack(pady=5)
556
557     self.entry_pin_confirm = tk.Entry(
558         self.frame_contenido,
559         font=("Arial", 16),
560         width=15,
561         show="*"
562     )
563     self.entry_pin_confirm.pack(pady=10)
564
565     tk.Button(
566         self.frame_contenido,
567         text="Guardar Usuario",
568         font=("Arial", 14, "bold"),
569         bg=COLOR_SUCCESS,
570         fg="white",
571         command=self.guardar_usuario,
572         width=20,
573         height=2
574     ).pack(pady=30)
575
576     self.entry_pin.bind("<Return>", lambda e: self.entry_pin_confirm.focus())
577     self.entry_pin_confirm.bind("<Return>", lambda e: self.guardar_usuario())
578
579 def guardar_usuario(self):
580     """Valida el PIN y guarda el usuario en la BD"""
581     pin = self.entry_pin.get().strip()
582     pin_confirm = self.entry_pin_confirm.get().strip()
583
584     # Validaciones
585     if not pin or not pin.isdigit() or len(pin) != 4:
586         messagebox.showerror("Error", "El PIN debe ser 4 dígitos numéricos",
parent=self.dialog_registro)
587         return
588
589     if pin != pin_confirm:
590         messagebox.showerror("Error", "Los PINs no coinciden", parent=self.
dialog_registro)
591         return
592
593     # Encriptar PIN

```

```

594     pin_hash = bcrypt.hashpw(pin.encode(), bcrypt.gensalt()).decode()
595
596     try:
597         # Guardar usuario
598         user_id = insert_user(self.nombre_usuario, pin_hash)
599
600         # Guardar rostros
601         for embedding in self.embeddings_rostro:
602             insert_face(user_id, embedding)
603
604         # Log
605         log_event(user_id, "granted", "Usuario registrado desde panel admin")
606
607         messagebox.showinfo(
608             "Éxito",
609             f"Usuario '{self.nombre_usuario}' registrado correctamente\n\n"
610             f"Rostros guardados: {len(self.embeddings_rostro)}",
611             parent=self.dialog_registro
612         )
613
614         self.cerrar_registro()
615         self.cargar_usuarios()
616
617     except Exception as e:
618         messagebox.showerror("Error", f"Error al guardar usuario:\n{e}",
parent=self.dialog_registro)
619
620     def cerrar_camara_registro(self):
621         """Cierra la cámara del registro"""
622         self.camara_registro_activa = False  # <-- DESACTIVAR PRIMERO
623         if self.cap_registro:
624             self.cap_registro.release()
625             self.cap_registro = None
626
627     def cerrar_registro(self):
628         """Cierra el diálogo de registro"""
629         self.cerrar_camara_registro()
630         if hasattr(self, 'dialog_registro') and self.dialog_registro.winfo_exists():
631             self.dialog_registro.destroy()
632
633     # ===== TAB HISTORIAL =====
634
635     def setup_tab_historial(self):
636         """Configura la pestaña de historial"""
637         # Frame superior
638         frame_top = tk.Frame(self.tab_historial, bg=COLOR_PANEL)
639         frame_top.pack(pady=20, fill="x", padx=20)
640
641         tk.Label(
642             frame_top,

```

```

643     text="Eventos recientes:",
644     font=("Arial", 14, "bold"),
645     bg=COLOR_PANEL,
646     fg=COLOR_TEXT
647 ).pack(side="left")

648

649     tk.Button(
650         frame_top,
651         text="Actualizar",
652         font=("Arial", 11),
653         bg=COLOR_INFO,
654         fg="white",
655         command=self.cargar_eventos,
656         width=15
657 ).pack(side="right", padx=5)

658

659     tk.Button(
660         frame_top,
661         text="Exportar CSV",
662         font=("Arial", 11),
663         bg=COLOR_SUCCESS,
664         fg="white",
665         command=self.exportar_csv,
666         width=15
667 ).pack(side="right", padx=5)

668

669 # Frame tabla
670 frame_tabla = tk.Frame(self.tab_historial, bg=COLOR_PANEL)
671 frame_tabla.pack(expand=True, fill="both", padx=20, pady=10)

672

673 # Scrollbar
674 scrollbar = ttk.Scrollbar(frame_tabla)
675 scrollbar.pack(side="right", fill="y")

676

677 # Treeview
678 columns = ("Fecha/Hora", "Usuario", "Resultado", "Notas")
679 self.tree_eventos = ttk.Treeview(
680     frame_tabla,
681     columns=columns,
682     show="headings",
683     yscrollcommand=scrollbar.set,
684     height=20
685 )
686
687     self.tree_eventos.heading("Fecha/Hora", text="Fecha/Hora")
688     self.tree_eventos.heading("Usuario", text="Usuario")
689     self.tree_eventos.heading("Resultado", text="Resultado")
690     self.tree_eventos.heading("Notas", text="Notas")
691
692     self.tree_eventos.column("Fecha/Hora", width=180)
693     self.tree_eventos.column("Usuario", width=200)

```

```

694     self.tree_eventos.column("Resultado", width=120, anchor="center")
695     self.tree_eventos.column("Notas", width=400)
696
697     self.tree_eventos.pack(expand=True, fill="both")
698     scrollbar.config(command=self.tree_eventos.yview)
699
700 def cargar_eventos(self):
701     """Carga el historial de eventos"""
702     # Limpiar tabla
703     for item in self.tree_eventos.get_children():
704         self.tree_eventos.delete(item)
705
706     # Obtener eventos
707     eventos = get_recent_events(100)
708
709     # Agregar a la tabla
710     for evento in eventos:
711         event_id, ts, device, user_id, result, note = evento
712
713         # Obtener nombre de usuario
714         usuario_nombre = "Desconocido"
715         for user in self.usuarios_data:
716             if user[0] == user_id:
717                 usuario_nombre = user[1]
718                 break
719
720         resultado = "Concedido" if result == "granted" else "Denegado"
721
722         self.tree_eventos.insert(
723             "",
724             "end",
725             values=(ts, usuario_nombre, resultado, note or ""))
726     )
727
728 def exportar_csv(self):
729     """Exporta el historial a CSV"""
730     archivo = filedialog.asksaveasfilename(
731         defaultextension=".csv",
732         filetypes=[("CSV files", "*.csv"), ("All files", "*.*")],
733         initialfile=f"historial_{datetime.now().strftime('%Y%m%d_%H%M%S')}.
734 csv"
734     )
735
736     if archivo:
737         try:
738             import csv
739             eventos = get_recent_events(1000)
740
741             with open(archivo, 'w', newline='', encoding='utf-8') as f:
742                 writer = csv.writer(f)
743                 writer.writerow(["Fecha/Hora", "Dispositivo", "Usuario ID", ""])

```

```

    Resultado", "Notas"])
744
745         for evento in eventos:
746             event_id, ts, device, user_id, result, note = evento
747             writer.writerow([ts, device, user_id, result, note or ""))
748     ]
749     messagebox.showinfo("Éxito", f"Historial exportado a:{archivo}")
750 except Exception as e:
751     messagebox.showerror("Error", f"Error al exportar: {e}")
752
753 def cerrar(self):
754     """Cierra la ventana de administración"""
755     # Asegurar que se cierra la cámara de registro si está abierta
756     self.cerrar_camara_registro()
757
758     # Cerrar diálogo de registro si está abierto
759     if hasattr(self, 'dialog_registro') and self.dialog_registro.winfo_exists():
760         self.dialog_registro.destroy()
761
762     # Cerrar ventana principal
763     if self.window:
764         self.window.destroy()

```

Listing 4: Ventana de administrador

5.1.4. Código para la base de datos

```

columns
1 # core/db_manager.py
2 #
3 # Gestión de base de datos
4 #
5
6 import sqlite3
7 import json
8 from collections import defaultdict
9 from config import DB_PATH, DEVICE_NAME
10
11
12 def ensure_schema():
13     """Crea tablas si no existen"""
14     conn = sqlite3.connect(DB_PATH)
15     c = conn.cursor()
16
17     c.execute("""
18         CREATE TABLE IF NOT EXISTS users(
19             id INTEGER PRIMARY KEY AUTOINCREMENT,
20             name TEXT NOT NULL,

```

```

21     pin TEXT NOT NULL,
22     active INTEGER DEFAULT 1,
23     created_at DATETIME DEFAULT CURRENT_TIMESTAMP
24 );
25 """
26
27 c.execute("""
28 CREATE TABLE IF NOT EXISTS faces(
29     id INTEGER PRIMARY KEY AUTOINCREMENT,
30     user_id INTEGER NOT NULL,
31     encoding_json TEXT NOT NULL,
32     created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
33     FOREIGN KEY(user_id) REFERENCES users(id) ON DELETE CASCADE
34 );
35 """
36
37 c.execute("""
38 CREATE TABLE IF NOT EXISTS events(
39     id INTEGER PRIMARY KEY AUTOINCREMENT,
40     ts DATETIME DEFAULT CURRENT_TIMESTAMP,
41     device TEXT,
42     user_id INTEGER,
43     result TEXT,
44     note TEXT
45 );
46 """
47
48 c.execute("CREATE INDEX IF NOT EXISTS idx_faces_user ON faces(user_id);")
49 c.execute("CREATE INDEX IF NOT EXISTS idx_events_ts ON events(ts);")
50
51 conn.commit()
52 conn.close()
53
54
55 def fetch_active_users_and_faces():
56 """
57 Devuelve:
58     users: dict user_id -> {"name": str, "pin": str}
59     faces: dict user_id -> [embedding_list, ...]
60 """
61 conn = sqlite3.connect(DB_PATH)
62 c = conn.cursor()
63
64 c.execute("SELECT id, name, pin FROM users WHERE active=1")
65 users_rows = c.fetchall()
66 users = {uid: {"name": name, "pin": pin} for uid, name, pin in users_rows}
67
68 c.execute("SELECT user_id, encoding_json FROM faces")
69 faces_rows = c.fetchall()
70 faces = defaultdict(list)
71 for user_id, enc_json in faces_rows:

```

```

72     try:
73         emb = json.loads(enc_json)
74         faces[user_id].append(emb)
75     except Exception:
76         continue
77
78     conn.close()
79     return users, faces
80
81
82 def get_all_users():
83     """Obtiene todos los usuarios (activos e inactivos)"""
84     conn = sqlite3.connect(DB_PATH)
85     c = conn.cursor()
86     c.execute("""
87         SELECT id, name, active, created_at,
88             (SELECT COUNT(*) FROM faces WHERE user_id = users.id) as
89             face_count
90             FROM users
91             ORDER BY created_at DESC
92         """)
93     users = c.fetchall()
94     conn.close()
95     return users
96
97 def insert_user(name: str, pinhash: str) -> int:
98     """Inserta un nuevo usuario y retorna su ID"""
99     conn = sqlite3.connect(DB_PATH)
100    c = conn.cursor()
101    c.execute("INSERT INTO users(name, pin) VALUES(?, ?)", (name, pinhash))
102    user_id = c.lastrowid
103    conn.commit()
104    conn.close()
105    return user_id
106
107
108 def insert_face(user_id: int, embedding) -> None:
109     """Inserta un embedding facial para un usuario"""
110     conn = sqlite3.connect(DB_PATH)
111     c = conn.cursor()
112     c.execute(
113         "INSERT INTO faces(user_id, encoding_json) VALUES(?, ?)",
114         (user_id, json.dumps(list(embedding)))
115     )
116     conn.commit()
117     conn.close()
118
119
120 def update_user_status(user_id: int, active: bool):
121     """Activa o desactiva un usuario"""

```

```

122     conn = sqlite3.connect(DB_PATH)
123     c = conn.cursor()
124     c.execute("UPDATE users SET active=? WHERE id=?", (1 if active else 0,
125     user_id))
126     conn.commit()
127     conn.close()

128
129 def delete_user(user_id: int):
130     """Elimina un usuario y todos sus rostros"""
131     conn = sqlite3.connect(DB_PATH)
132     c = conn.cursor()
133     c.execute("DELETE FROM users WHERE id=?", (user_id,))
134     conn.commit()
135     conn.close()

136
137
138 def log_event(user_id, result, note=""):
139     """Registra un evento de acceso"""
140     conn = sqlite3.connect(DB_PATH)
141     c = conn.cursor()
142     c.execute(
143         "INSERT INTO events(device, user_id, result, note) VALUES(?, ?, ?, ?)",
144         (DEVICE_NAME, user_id, result, note)
145     )
146     conn.commit()
147     conn.close()

148
149
150 def get_recent_events(limit=50):
151     """Obtiene los eventos más recientes"""
152     conn = sqlite3.connect(DB_PATH)
153     c = conn.cursor()
154     c.execute("""
155         SELECT e.id, e.ts, e.device, u.name, e.result, e.note
156         FROM events e
157         LEFT JOIN users u ON e.user_id = u.id
158         ORDER BY e.ts DESC
159         LIMIT ?
160     """, (limit,))
161     events = c.fetchall()
162     conn.close()
163     return events

164
165
166 def get_user_stats(user_id: int):
167     """Obtiene estadísticas de un usuario"""
168     conn = sqlite3.connect(DB_PATH)
169     c = conn.cursor()
170
171     # Total de accesos

```

```

172     c.execute("""
173         SELECT COUNT(*) FROM events
174         WHERE user_id = ? AND result = 'granted'
175     """, (user_id,))
176     total_accesos = c.fetchone()[0]
177
178     # Accesos denegados
179     c.execute("""
180         SELECT COUNT(*) FROM events
181         WHERE user_id = ? AND result = 'denied'
182     """, (user_id,))
183     accesos_denegados = c.fetchone()[0]
184
185     # Último acceso
186     c.execute("""
187         SELECT ts FROM events
188         WHERE user_id = ? AND result = 'granted'
189         ORDER BY ts DESC LIMIT 1
190     """, (user_id,))
191     row = c.fetchone()
192     ultimo_acceso = row[0] if row else "Nunca"
193
194     # Número de rostros registrados
195     c.execute("""
196         SELECT COUNT(*) FROM faces WHERE user_id = ?
197     """, (user_id,))
198     num_rostros = c.fetchone()[0]
199
200     conn.close()
201
202     return {
203         'total_accesos': total_accesos,
204         'accesos_denegados': accesos_denegados,
205         'ultimo_acceso': ultimo_acceso,
206         'num_rostros': num_rostros
207     }

```

Listing 5: Libreria de la base de datos

5.1.5. Código para el reconocimiento facial

```

columns
1 # core/face_recognition.py
2 #
3 # Reconocimiento facial con DeepFace
4 #
5
6 import os
7 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
8 os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
9

```

```

10 import warnings
11 warnings.filterwarnings('ignore', category=DeprecationWarning)
12 warnings.filterwarnings('ignore', category=FutureWarning)
13
14 import math
15 from deepface import DeepFace
16 from config import FACE_MODEL, FACE_DETECTOR
17
18
19 def get_embedding_deepface(frame_bgr):
20     """
21         Obtiene el embedding facial con DeepFace.
22
23     Args:
24         frame_bgr: Frame en formato BGR de OpenCV
25
26     Returns:
27         list: Embedding facial
28
29     Raises:
30         ValueError: Si no se detecta rostro
31     """
32     reps = DeepFace.represent(
33         img_path=frame_bgr,
34         model_name=FACE_MODEL,
35         detector_backend=FACE_DETECTOR,
36         enforce_detection=True
37     )
38     if not reps:
39         raise ValueError("No se detectó rostro en la imagen")
40     return reps[0]["embedding"]
41
42
43 def cosine_similarity(a, b):
44     """Similitud coseno entre dos embeddings"""
45     num = sum(x * y for x, y in zip(a, b))
46     den = math.sqrt(sum(x * x for x in a)) * math.sqrt(sum(y * y for y in b))
47     return (num / den) if den else 0.0
48
49
50 def best_match_per_user(query_emb, faces_by_user):
51     """
52         Encuentra el mejor match entre usuarios.
53
54     Returns:
55         tuple: (best_user_id, best_score)
56     """
57     best_user, best_score = None, 0.0
58     for uid, emb_list in faces_by_user.items():
59         if not emb_list:
60             continue

```

```

61     score = max(cosine_similarity(query_emb, e) for e in emb_list)
62     if score > best_score:
63         best_score = score
64         best_user = uid
65     return best_user, best_score

```

Listing 6: Reconocimiento facial

5.1.6. Código para el reconocimiento de gestos

```

columns
1 # core/gesture_detection.py
2 #
3 # Detección de gestos con MediaPipe
4 #
5
6 class GestureDetector:
7     """Detector de gestos de mano""" # Clase que agrupa la lógica de detección
8     # de gestos basados en landmarks
9
10    def __init__(self):
11        # Diccionario de gestos disponibles con sus descripciones legibles
12        self.gestos_disponibles = {
13            'pulgar_arriba': 'Pulgar arriba',
14            'victoria': 'Victoria (2 dedos)',
15            'ok': 'OK (circulo)',
16            'mano_abierta': 'Mano abierta (5 dedos)',
17            'punto': 'Puño cerrado'
18        }
19
20    def contar_dedos_levantados(self, landmarks):
21        """Cuenta dedos levantados""" # Devuelve cuántos dedos están extendidos
22        # según posiciones de landmarks
23        dedos = [] # Lista para marcar cada dedo como levantado (1) o no (0)
24
25        # Pulgar: se considera levantado si la punta (4) está a la izquierda de
26        # la articulación previa (3) en eje X
27        # Nota: esto asume una mano derecha y coordenadas normalizadas, puede
28        # requerir ajustes por mano/rotación
29        if landmarks[4].x < landmarks[3].x:
30            dedos.append(1) # Pulgar levantado
31        else:
32            dedos.append(0) # Pulgar no levantado
33
34        # Otros dedos: índice (8), medio (12), anular (16), meñique (20)
35        tip_ids = [8, 12, 16, 20] # IDs de las puntas de los dedos en MediaPipe
            for tip in tip_ids:
                # Un dedo está levantado si la punta (tip) está por encima de la
                # articulación media (tip - 2) en eje Y
                # En coordenadas de imagen, menor Y usualmente significa más arriba
                if landmarks[tip].y < landmarks[tip - 2].y:

```

```

36         dedos.append(1) # Dedo levantado
37     else:
38         dedos.append(0) # Dedo no levantado
39
40     return sum(dedos) # Retorna el total de dedos levantados
41
42 def detectar_pulgar_arriba(self, landmarks):
43     """Detecta pulgar arriba"""\ # Comprueba si el gesto corresponde a "
pulgar arriba"
44     # Pulgar con sus segmentos ordenados verticalmente: punta (4) arriba de
(3) y éste arriba de (2)
45     pulgar_arriba = landmarks[4].y < landmarks[3].y < landmarks[2].y
46     # Otros dedos hacia abajo: sus puntas por debajo de la articulación media
47     otros_abajo = all(landmarks[i].y > landmarks[i - 2].y for i in [8, 12,
16, 20])
48     return pulgar_arriba and otros_abajo # Verdadero si pulgar arriba y
resto abajo
49
50 def detectar_victoria(self, landmarks):
51     """Detecta victoria"""\ # Gesto de dos dedos levantados (índice y medio)
52     dedos = self.contar_dedos_levantados(landmarks) # Cuenta dedos
levantados
53     indice_arriba = landmarks[8].y < landmarks[6].y # Índice levantado
54     medio_arriba = landmarks[12].y < landmarks[10].y # Medio levantado
55     return dedos == 2 and indice_arriba and medio_arriba # Exactamente esos
dos levantados
56
57 def detectar_ok(self, landmarks):
58     """Detecta OK"""\ # Gesto de círculo entre pulgar e índice con otros
dedos preferiblemente abiertos
59     # Distancia Manhattan aproximada entre punta de pulgar (4) y punta de í
ndice (8)
60     dist = abs(landmarks[4].x - landmarks[8].x) + abs(landmarks[4].y -
landmarks[8].y)
61     # Considera OK si las puntas están muy cerca y hay al menos 3 dedos
levantados
62     return dist < 0.05 and self.contar_dedos_levantados(landmarks) >= 3
63
64 def detectar_mano_abierta(self, landmarks):
65     """Detecta mano abierta"""\ # Todos los dedos levantados
66     return self.contar_dedos_levantados(landmarks) == 5
67
68 def detectar_puno(self, landmarks):
69     """Detecta puño"""\ # Ningún dedo levantado
70     return self.contar_dedos_levantados(landmarks) == 0
71
72 def verificar_gesto(self, gesto_solicitado, landmarks):
73     """Verifica si el gesto coincide"""\ # Selecciona el detector según el
nombre y lo ejecuta
74     metodos = {
75         'pulgar_arriba': self.detectar_pulgar_arriba, # Mapa de nombre a

```

```
función detectora
76     'victoria': self.detectar_victoria,
77     'ok': self.detectar_ok,
78     'mano_abierta': self.detectar_mano_abierta,
79     'puno': self.detectar_puno
80 }
81 # Obtiene el método, o una función que devuelve False si no existe, y lo
llama con landmarks
82 return métodos.get(gesto_solicitado, lambda x: False)(landmarks)
```

Listing 7: Reconocimiento de gestos