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: 13 de diciembre de 2025

Universidad de La Laguna

Índice

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.
- **Auditable:** 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 solucion adoptada

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

2.1. 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.

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

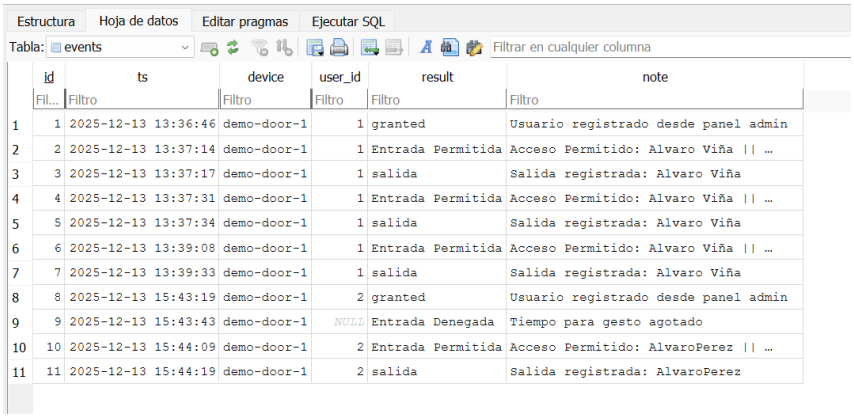
Para el almacenamiento persistente se emplea **SQLite**, por su sencillez de despliegue y compatibilidad con Python sin necesidad de servidor externo. La base de datos se inicializa automáticamente al arrancar la aplicación si no existe el fichero, creando las tablas y claves foráneas necesarias. Los 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



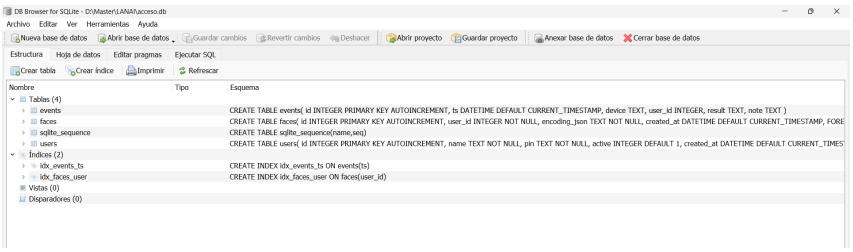
Estructura | Hoja de datos | Editar pragmas | Ejecutar SQL

Tabla: events — Filtrar en cualquier columna

	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 **encriptado 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.



DB Browser for SQLite - D:\Master\LANA\acceso.db

Archivo Editar Ver Herramientas Ayuda

Nueva base de datos | Abrir base de datos | Guardar cambios | Revertir cambios | Deshacer | Abrir proyecto | Guardar proyecto | Anexar base de datos | Cerrar base de datos

Estructura | Hoja de datos | Editar pragmas | Ejecutar SQL

Crear tabla | Crear índice | Imprimir | Refrescar

Nombre	Tipo	Esquema
Tablas (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, FORE
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
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_TIMES
Índices (2)		
idx_events_ts		CREATE INDEX idx_events_ts ON events(ts)
idx_faces_user		CREATE INDEX idx_faces_user ON faces(user_id)
Vistas (0)		
Disparadores (0)		

Figura 2: Base de datos en SQLite

2.3. Fase 3: Implementación del Reconocimiento Facial

2.3.1. 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

2.3.2. 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.

2.3.3. 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 al menos 10 fotogramas con calidad mínima de detección
2. Validación de cada fotograma para evitar imágenes borrosas o parcialmente ocluidas
3. Generación de embeddings faciales usando el modelo ResNet50 de DeepFace
4. Almacenamiento de las características en la base de datos SQLite
5. Confirmación visual al usuario del registro exitoso

2.3.4. Proceso de Autenticación

El proceso de autenticación en tiempo real implementa los siguientes pasos:

1. Captura continua del stream de video
2. Detección de rostros en cada fotograma utilizando MTCNN (Multi-task Cascaded Convolutional Networks)
3. Extracción de embedding facial del rostro detectado
4. Comparación con embeddings almacenados usando distancia euclidiana
5. Aplicación de threshold de similitud (configurado en 0.7 en el archivo config.py)
6. Generación de log con timestamp, usuario identificado y grado de confianza
7. Activación de mecanismo de acceso si la similitud supera el threshold

2.3.5. Optimizaciones Implementadas

Se aplicaron varias optimizaciones para mejorar el rendimiento:

- **Reducción de Resolución:** Procesamiento de fotogramas a 320x240 píxeles para detectar rostros, manteniendo precisión
- **Caché de Embeddings:** Almacenamiento en memoria de características faciales para reducir accesos a base de datos
- **Procesamiento Asincrónico:** Utilización de threads para evitar bloqueos en la interfaz gráfica
- **Detección Selectiva:** Análisis solo cuando se detecta movimiento significativo en la imagen
- **Modelos Ligeros:** Selección de DeepFace con modelo VGGFace2 como balance entre precisión y velocidad

2.3.6. Gestión de Errores y Casos Especiales

El módulo implementa manejo robusto de situaciones problemáticas:

- **Rostro No Detectado:** Reintentos automáticos durante 5 segundos
- **Múltiples Rostros:** Selección del rostro más prominente o rechazo si la ambigüedad es alta
- **Oclusión Parcial:** Evaluación de calidad antes de procesar
- **Problemas de Iluminación:** Ajuste automático de contraste y brillo
- **Fallo de Cámara:** Mensaje de error y logging de incidente

2.4. 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.

2.5. Fase 5: Detección de Gestos

2.5.1. 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

2.5.2. 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

2.5.3. 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.

2.5.4. 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

2.5.5. 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

2.6. Fase 6: Desarrollo de la Interfaz Gráfica

2.6.1. 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.

2.6.2. 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.

2.6.3. 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.

2.6.4. 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`.

2.6.5. 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.

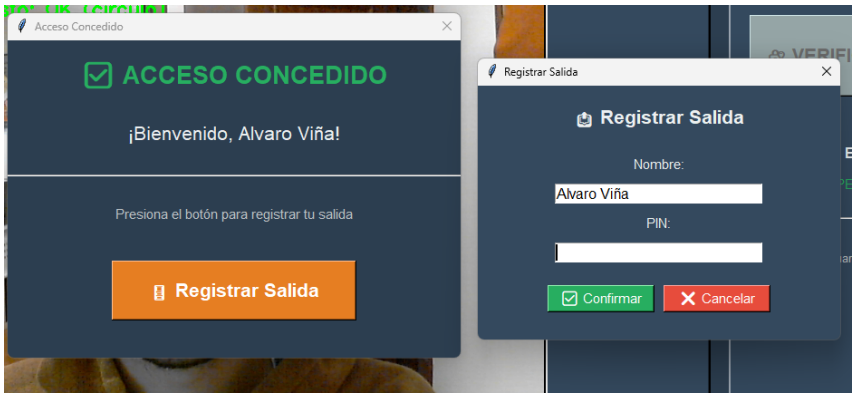


Figura 6: Confirmación de acceso concedido y registro de salida del usuario.

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.

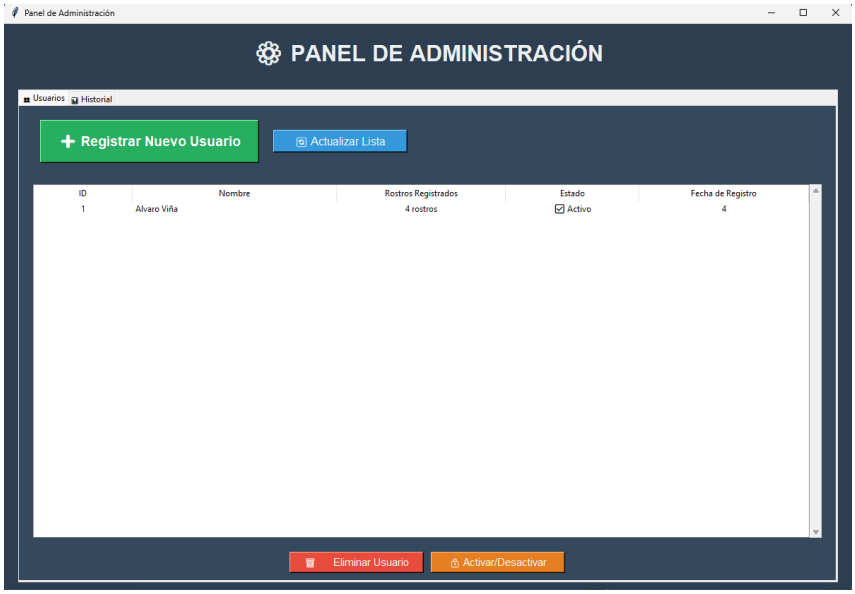


Figura 7: Panel de administración: gestión de usuarios y estado del sistema.

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.

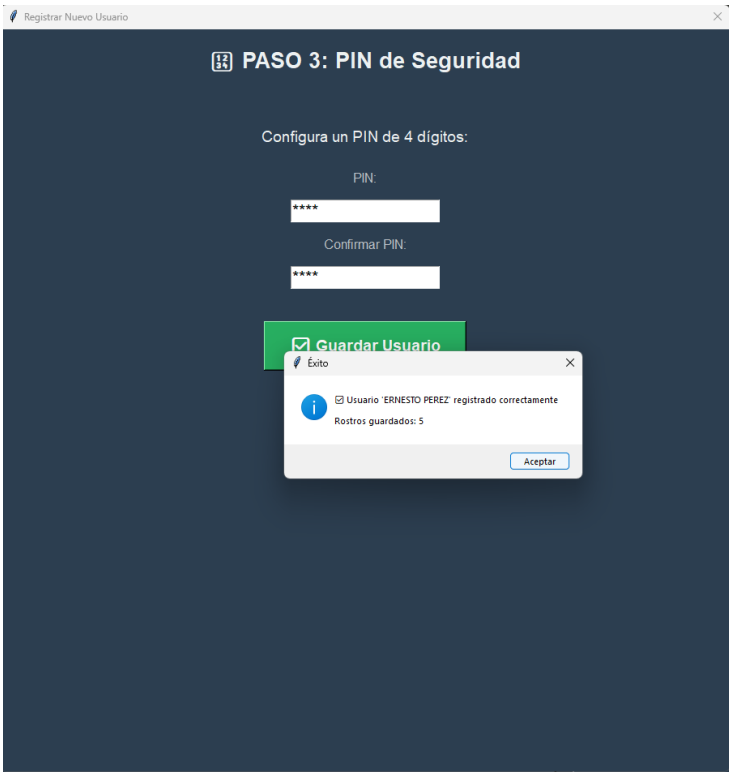


Figura 10: Registro de nuevo usuario: solicitud de PIN y confirmación de alta.

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.

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 umbrales 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`